Unit-I

Overview and Instructions

Eight ideas – Components of a computer system – Technology – Performance – Power wall – Uniprocessors to multiprocessors; Instructions – operations and operands – representing instructions – Logical operations – control operations – Addressing and addressing modes.

Computer

- A Computer is a machine which accepts input information in the digitized form, processes the input according to a set of stored instructions and produces the resulting output information.

Program and Data

- The set of stored instructions written using a computer to solve the task is called program and input and output information is called data. The internal storage where programs are stored is called Memory.

Characteristics of computer:

Speed:

- Computers perform various operations at a very high speed.

Accuracy:

- Computers are very accurate. Do not make mistakes in calculations.

Reliability:

- Computers give correct and consistent results always even if they are used in adverse conditions.
- Many times errors are caused by human interventions not by computer. Computer output is reliable, subject to the condition that the input data and the instructions (programs) are correct. Incorrect input data and unreliable programs give us wrong results.

Storage Capacity:

- The computer can store large amount of data and can be retrieved at any time in fractions of a second.

- This data can be stored in permanent storage devices like hard disk, CDs etc.

Versatility:

- Computers can do a variety of jobs based on the instructions given to them. They are used in each and every field, making the tasks easier.

Hardware:

- Hardware is the physical aspect of computers, telecommunications, and other device. Hardware implies permanence and invariability
- The components include keyboard, floppy drive, hard disk, monitor, CPU, printer, wires, transistors, circuits etc.

Software:

- It is a set of programs used to perform certain tasks. Program is set of instructions to carry out a particular task

Computer Organization

- It refers to the operational units and their interconnections that realize the architectural specifications.
- It describes the function of and design of the various units of digital computer that store and process information.

Computer Architecture

- It is concerned with the structure and behavior of the computer. It includes the information formats, the instruction set and techniques for addressing memory.

1.1 Eight Great Ideas in Computer Architecture:

1. Design for Moore's Law.
2. Use Abstraction to Simplify Design
3. Make the common case fast
4. Performance via parallelism
5. Performance via pipelining
6. Performance via prediction
7. Hierarchy of memories
8. Dependability via redundancy

1.1.1 Design for Moore's Law

- The Number of Transistors in an integrated circuit doubles approximately every two years. (Gordon Moore, one of the founders of Intel).
- As computer designs can take years, the resources available per chip can easily double or quadruple between the start and finish of the project.
- Computer architects must anticipate where the technology will be when the design finishes rather than where it starts.
- Moore's law graph to represent designing for rapid change.

1.1.2 Use Abstraction to Simplify Design

- In computer architecture, a computer system consists of five abstraction levels:
    1. hardware
    2. firmware
    3. assembler
    4. operating system and
    5. Processes.
- In computer science, an abstraction level is a generalization of a model or algorithm. The simplification provided by abstraction layer helps us to reuse easily.
- Both computer architects and programmers had to invent techniques to make them more useful, otherwise design time would grow affectedly as resources grew by Moore's Law.

- A major productivity technique for hardware and software is to use abstractions represent the design at different levels of representation; lower-level details are hidden to offer a simpler model at higher levels.

1.1.3 Make the common case fast

- Making the common case fast will tend to enhance performance better than optimizing the rare case.
- It implies that you know what the common case is, which is only possible with careful experimentation and measurement.
- We use a sports car as the icon for making the common case fast

1.1.4 Performance via parallelism

- Computer architects have offered designs that get more performance by performing operations in parallel.
- Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones which are then solved concurrently.

- Parallelism has been employed for many years, mainly in high performance computing. We use multiple jet engines of a plane as our icon for parallel performance.

1.1.5 Performance via pipelining

- Pipelining is a technique used in the design of computers to increase the instruction throughput (the number of instructions that can be executed in a unit of time).
- The basic instruction cycle is broken up into a series of pipeline stages.
- Rather than processing each instruction sequentially, each instruction is split up into a sequence of steps so that different steps can be executed concurrently and in parallel.
- Pipelining increases instruction throughput by performing multiple operations at the same time but does not reduce instruction latency (the time to complete a single instruction from start to finish) as it still must go through all steps.
- Our pipeline icon is a sequence of pipes, with each section representing one stage of the pipeline.

1.1.6 Performance via prediction

- To improve the flow and throughput in a instruction pipeline, Branch predictors play a critical role in achieving high effective performance in many modern pipelined microprocessor architectures.
- Without branch prediction, the processor would have to wait until the conditional jump instruction has passed the execute stage before the next instruction can enter the fetch stage in the pipeline.
- The branch predictor attempts to avoid this waste of time by trying to guess whether the conditional jump is most likely to be taken or not taken.

1.1.7 Hierarchy of memories

- Programmers want memory to be fast, large, and cheap, as memory speed often shapes performance, capacity limits the size of problems that can be solved, and the cost of memory today is often the majority of computer cost.
- Architects have found that they can address these conflicting demands with a hierarchy of memories, with the fastest, smallest, and most expensive memory per bit at the top of the hierarchy and the slowest, largest, and cheapest per bit at the bottom.
- Caches give the programmer the illusion that main memory is nearly as fast as the top of the hierarchy and nearly as big and cheap as the bottom of the hierarchy.

1.1.8 Dependability via redundancy

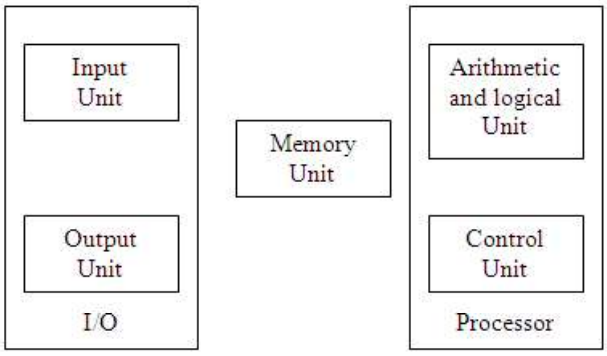
- Computers not only need to be fast; they need to be dependable. Since any physical device can fail, we make systems dependable by including redundant components that can take over when a failure occurs and to help detect failures.
- We use the tractor-trailer as our icon, since the dual tires on each side of its rear axles allow the truck to continue driving even when one tire fails

1.2 Components of Computer System:

Computer Hardware:

A computer consists of 5 main parts.

1. Input Unit,
2. Output Unit
3. Memory Unit,
4. Processing Unit,



1.2.1 Input Unit

- Computers accept coded information through input units, which read the data.

- Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Keyboard:

- Keyboard is a input device which helps in entering information into the computer. Keyboards are of two sizes 84 keys or 101/102 keys, but now keyboards with 104 keys or 108 keys are also available for Windows and Internet.

Mouse:

- Mouse is a pointing device. It is a very famous cursor-control device having a small palm size box with a round ball at its base which senses the movement of mouse and sends corresponding signals to CPU when the mouse buttons are pressed.

Joystick:

- Joystick is also a pointing device which is used to move cursor position on a monitor screen.
- It is mainly used in Computer Aided Designing(CAD) and playing computer games.

Light Pen:

- Light pen is a pointing device which is similar to a pen. It is used to select a displayed menu item or draw pictures on the monitor screen.
- When the tip of a light pen is moved over the monitor screen and pen button is pressed, its photocell sensing element detects the screen location and sends the corresponding signal to the CPU.

Track Ball:

- Track ball is an input device that is mostly used in notebook or laptop computer, instead of a mouse. It is used by graphical designers.

Scanner:

- Scanner is used when some information is available on a paper and it is to be transferred to the hard disc of the computer.

- Scanner captures images from the source which are then converted into the digital form that can be stored on the disc.

Magnetic Ink Card Reader (MICR):

- The bank's code number and cheque number are printed on the cheques with a special type of ink that contains particles of magnetic material that are machine readable. This reading process is called Magnetic Ink Character Recognition (MICR).

Optical Character Reader(OCR):

- OCR is an input device used to read a printed text. OCR scans text optically character by character, converts them into a machine readable code and stores the text on the system memory.

Bar Code Readers:

- Bar Code Reader is a device used for reading bar coded data Bar Code Reader scans a bar code image, converts it into an alphanumeric value which is then fed to the computer to which bar code reader is connected.

Optical Mark Reader (OMR):

- OMR is a special type of optical scanner used to recognize the type of mark made by pen or pencil.
- It is specially used for checking the answer sheets of examinations having multiple choice questions.

1.2.2 Output Unit

- Its function is to send the processed results to the outside world.
    1. Monitors

2. Graphic Plotter
3. Printer

Monitors:

- Monitors are also called as Visual Display Unit (VDU).
- It forms images from tiny dots, called pixels that are arranged in a rectangular form.
- The sharpness of the image depends upon the number of pixels.
- There are two kinds of viewing screen used for monitors.
    1. Cathode-Ray Tube (CRT)
    2. Flat- Panel Display

Printers:

- Printer is an output device, which is used to print information on paper. There are two types of printers:
    1. Impact Printers
    2. Non-Impact Printers

Non-impact Printers:

- Non-impact printers print the characters without using ribbon. These printers print a complete page at a time so they are also called as Page Printers. These printers are of two types
    1. Laser Printers
    2. Inkjet Printers

Laser Printers:

- They use laser lights to produce the dots needed to form the characters to be printed on a page.

Inkjet Printers:

- They print characters by spraying small drops of ink onto paper.
- Some models of Inkjet printers can produce multiple copies of printing also.

1.2.3 Memory Unit

- A memory is just like a human brain. It is used to store data and instructions.
- Computer memory is the storage space in computer where data is to be processed and instructions required for processing are stored. Memory is primarily of three types
    1. Cache Memory
    2. Primary Memory/Main Memory
    3. Secondary Memory

Cache Memory:

- Cache memory is a very high speed semiconductor memory which can speed up CPU. It acts as a buffer between the CPU and main memory.
- It is used to hold data and program which are most frequently used by CPU.

Primary Memory (Main Memory):

- Primary memory holds only those data and instructions on which computer is currently working.
- It has limited capacity and data is lost when power is switched off. It is divided into two subcategories RAM and ROM.

Secondary Memory:

- This type of memory is also known as external memory or non-volatile. It is slower than main memory.
- These are used for storing data/Information permanently. For example: disk, CD-ROM, DVD etc.

RAM:

- Random Access Memory is the internal memory of the CPU for storing data, program and program result. It is read/write memory which stores data until the machine is working.

- As soon as the machine is switched off, data is erased.RAM is volatile, i.e. data stored in it is lost when we switch off the computer or if there is a power failure.

ROM:

- ROM stands for Read Only Memory. The memory from which we can only read but cannot write on it. This type of memory is non-volatile.
- The information is stored permanently in such memories during manufacture.
- A ROM, stores such instructions that are required to start a computer. This operation is referred to as bootstrap.

Motherboard:

- The motherboard serves as a single platform to connect all of the parts of a computer together.
- A motherboard connects CPU, memory, hard drives, optical drives, video card, sound card, and other ports and expansion cards directly or via cables.
- It can be considered as the backbone of a computer. Hardware represents the physical and tangible components of a computer i.e. the components that can be seen and touched.

Examples of Hardware are:

1. **Input devices --** keyboard, mouse etc.
2. **Output devices --** printer, monitor etc.
3. **Secondary storage devices --** Hard disk, CD, DVD etc.
4. **Internal components --** CPU, motherboard, RAM etc.

1.2.4 Central Processing Unit (CPU)

- It performs computing and manipulating functions. It controls other hardware devices. It has 3 elements: the Arithmetic/Logic Unit (ALU), Control Unit, and Registers.

Arithmetic and Logic Unit:

- Most computer operations are executed in ALU.
- It can perform arithmetic operations like addition, Subtraction, multiplication, division and also the logical operations like AND, OR, NOT operations.

- Consider an example, Suppose 2 numbers located in memory are to be added. They are brought into the processor and the actual addition is carried out by the ALU.
- The sum may then be stored in the memory or retained in the processor for immediate use.
- When operands are brought into the processor, they are stored in high-speed storage elements called registers. Each can store 1 word of data.
- Access times of registers are faster than cache unit in memory.

Control Unit:

- The operations of input unit, output unit, ALU are coordinated by the control unit.
- The control unit sends control signals to other units and senses their states.
- Data transfers between the processor and the memory are also controlled by the control unit through timing signals.

Registers:

- High-speed storage areas used to temporarily hold small units of program instructions and data that are being transferred from the primary storage to the CPU for processing.

Computer Software:

- Software is a set of programs, which is designed to perform a well-defined function. A program is sequences of instructions written to solve a particular problem.There are two types of software
    1. System Software
    2. Application Software

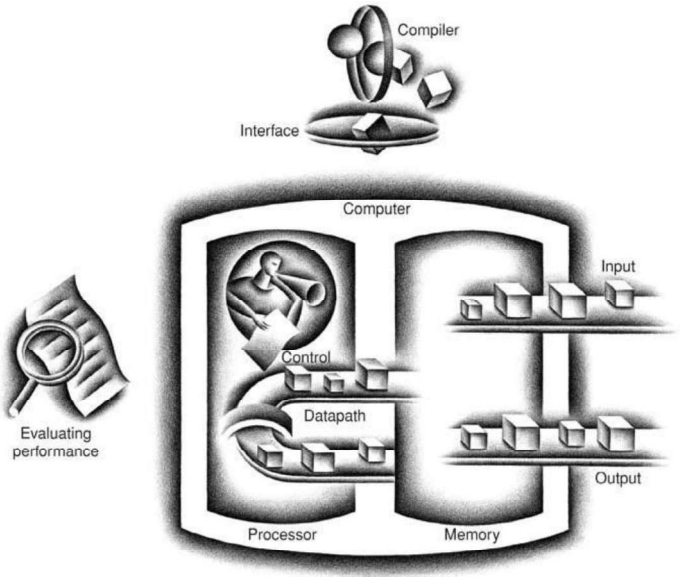
System Software:

- The system software is collection of programs designed to operate, control, and extend the processing capabilities of the computer itself.
- These software products include programs written in low-level languages which interact with the hardware at a very basic level.

- Some examples of system software are Operating System, Compilers, Interpreter, and Assemblers etc.

Application Software:

- Application software may consist of a single program, such as a Microsoft's notepad for writing and editing simple text.
- It may also consist of a collection of programs, often called a software package, which work together to accomplish a task, such as a spreadsheet package. Examples of Application software is following:
    1. Income Tax Software
    2. Railways Reservation Software
    3. Microsoft Word



Relationship between Hardware and Software:

1. Hardware and software are mutually dependent on each other. Both of them must work together to make a computer produce a useful output.
2. Software cannot be utilized without supporting hardware.
3. Hardware without set of programs to operate upon cannot be utilized and is useless.
4. To get a particular job done on the computer, relevant software should be loaded into the hardware
5. Hardware is a one-time expense.

6. Software development is very expensive and is a continuing expense.
7. Different software applications can be loaded on hardware to run different jobs.
8. Software acts as an interface between the user and the hardware.
9. If hardware is the 'heart' of a computer system, then software is its 'soul'. Both are complimentary to each other.

1.3 Technologies for Building Processors and Memories:

- To improve the performance of computer, Processors and memory are more important.
- Because the computer designers have to know the latest technology to design a better computer.
- Technology shapes what computers will be able to do and how quickly they will evolve.

Year	Technology used in computer	Relative performance / Unit cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit	900
1995	Very large scale integrated circuit	2400000
2005	Ultra large scale integrated circuit	6,200,000,000

Figure: Relative performances per unit cost of technologies used in computers over time

Transistor:

- A transistor is simply an on/off switch controlled by electricity

Vacuum tube:

- An electronic component, predecessor of the transistor, that consists of a hollow glass tube about 5 to 10 cm long from which as much air has been removed as possible and which uses an electron beam to transfer data.

Very large scale integrated (VLSI):

- A device that containing hundreds of thousands to millions of transistors.

Chip:

- Chip is the basic element for manufacturing integrated circuits.
- The manufacture of a chip 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:
    1. Excellent conductors of electricity (using either microscopic copper or aluminum wire)
    2. Excellent insulators from electricity (like plastic sheathing or glass)
    3. Areas that can conduct or insulate under special conditions (as a switch)

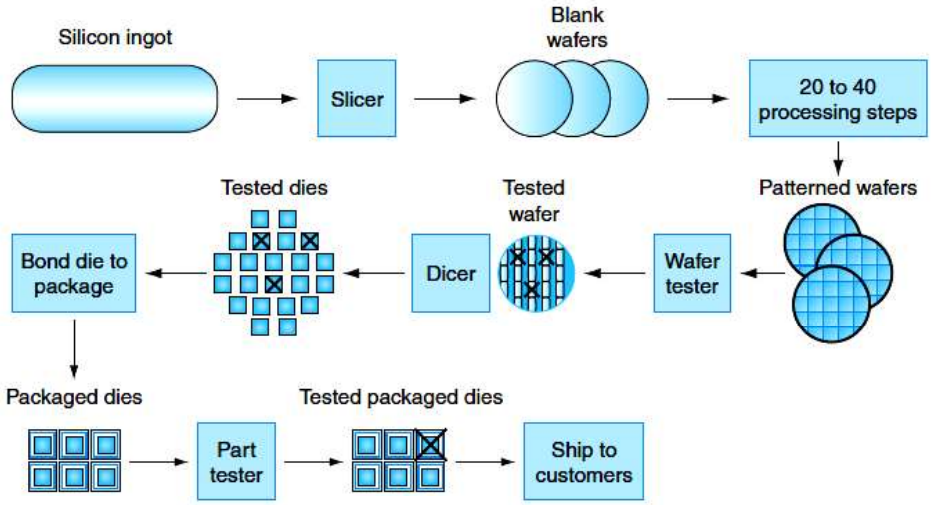


Figure: The chip manufacturing process

- A VLSI circuit is just billions of combinations of conductors, insulators, and switches manufactured in a single, small package.
- The manufacturing process for integrated circuits is critical to the cost of the chips and hence important to computer designers.
- The process starts with a silicon crystal ingot, which looks like a giant sausage.

- Silicon crystal ingot is a rod composed of a silicon crystal that is between 6 and 12 inches in diameter and about 12 to 24 inches long.
- An ingot is finely sliced into wafers no more than 0.1 inches thick.
- Wafer is a slice from a silicon ingot no more than 0.1 inch thick, used to create chips.
- These wafers then go through a series of processing steps to create the transistors, conductors, and insulators.
- Today's integrated circuits contain only one layer of transistors and two to eight levels of metal conductor, separated by layers of insulators.
- Defect is a microscopic flow in a wafer or in patterning steps that can result in the failure of the die containing that defect.
- Die is the individual rectangular sections that are cut from a wafer, more informally known as chips.
- Dicing enables us to discard only those dies that were unlucky enough to contain the flows, rather than the whole wafer. This concept is known as yield of a process
- Yield process is defined as the percentage of good dies from the total number of dies on the wafer.
- The cost of an integrated circuit rises quickly as the die size increases.
- To reduce the cost we can use smaller sizes transistors and wires. This improves the yield and the die count per wafer.

Bonding:

- Once we have found good dies, they are connected to the input/output pins of a package, using a process called bonding. These packaged parts are tested a final time, since mistakes can occur in packaging.
- If there is no mistake it can be shipped to customers.
- The cost of an integrated circuit can be expressed in three simple equations:

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

$$\text{Dies per wafer} \approx \frac{\text{Wafer area}}{\text{Die area}}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

- The first equation is straightforward to derive.
- The second is an approximation. It does not subtract the area near the border of the round wafer that cannot accommodate the rectangular dies.
- The final equation is based on empirical observations of yields at integrated circuit factories, with the exponent related to the number of critical processing steps.
- Hence, depending on the defect rate and the size of the die and wafer, costs are generally not linear in die area.

1.4 Performance

- Assessing the performance of computers is not an easy task. Performance depends on modern software systems and the wide range of performance improvement techniques in hardware side.
- To choose better computer, performance is an important attribute. For selecting a computer it is necessary to know how t to measure performance and limitations of performance measurements.
- Computer user and designer must know what the metrics for measuring performance are

Response time:

- It is also called execution time. The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, and CPU execution time and so on.

Throughput and Response Time:

Throughput:

- It is also called bandwidth. It is another measure of performance. It is defined as the number of tasks completed per unit time.

Example:1

Do the following changes to a computer system increase throughput, decrease response time, or both?

1. Replacing the processor in a computer with a faster version
2. Adding additional processors to a system that uses multiple processors for separate task.

Answer:

- Decreasing response time almost always improves throughput.
- Hence, in case 1, both response time and throughput are improved.
- In case 2, no one task gets work done faster, so only throughput increases. The system might force requests to queue up. In this case, increasing the throughput could also improve response time.
- It will reduce the waiting time in the queue.
- Thus, in many real computer systems, changing either execution time or throughput often affects the other.
- To maximize performance, we want to minimize response time or execution time for some task. Thus, we can relate performance and execution time for a computer X:

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$

- Now consider two computers X and Y, if the performance of X is greater than the performance of Y, we have

$$\text{Performance}_X > \text{Performance}_Y$$

$$\frac{1}{\text{Execution time}_X} > \frac{1}{\text{Execution time}_Y}$$

$$\text{Execution time}_Y > \text{Execution time}_X$$

- The execution time on Y is longer than that on X, if X is faster than Y.

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = n$$

- If X is n times faster than Y, then the execution time on Y is n times longer than it is on X:

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

Relative Performance:

Example:2

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?

Answer:

We know that A is n times faster than B if

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus the performance ratio is

$$\frac{15}{10} = 1.5$$

and A is therefore 1.5 times faster than B.

In the above example, we could also say that computer B is 1.5 times *slower than* computer A, since

$$\frac{\text{Performance}_A}{\text{Performance}_B} = 1.5$$

means that

$$\frac{\text{Performance}_A}{1.5} = \text{Performance}_B$$

- Performance and execution time are reciprocals; increasing performance requires decreasing execution time.

1.4.1 Measuring Performance

- Time is the measure of computer performance: the computer that performs the same amount of work in the least time is the fastest.
- Program execution time is measured in seconds per program. However, time can be defined in different ways, depending on what we count.
- The most straightforward definition of time is called wall clock time, response time, or elapsed time.
- Processor may work on several programs simultaneously so the system try to optimize throughput rather than minimize the elapsed time for one program.

- Hence, we often want to distinguish between the elapsed time and the time that the processor is working on the program. CPU execution time will recognizes this distinction.
- CPU execution time also called CPU time. The actual time the CPU spends computing for a specific task.

Types of CPU time:

CPU time can be classified into two types:

1. User CPU time
2. System CPU time

User CPU time:

- The CPU time spent in a program itself.

System CPU time:

- The CPU time spent in the operating system performing tasks on behalf of the program. Differentiating between system and user CPU time is difficult to do accurately.
- To increase the system performance the computer designers must know how fast the hardware can perform basic functions.
- All computers are constructed using a clock that determines when events take place in the hardware.

Clock cycle:

- Also called tick, clock tick, clock period, clock, cycle. Clock cycle is the time for one clock period, usually of the processor clock, which runs at a constant rate.

Clock period:

- The length of each clock cycle.

1.4.2 CPU Performance and Its Factors

- Users and designers have different metrics to measure the performance. CPU performance is one of the metric for measuring the performance.

- To know the CPU performance we must find the CPU execution time.

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

- Alternatively, because clock rate and clock cycle time are inverses,

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

- This formula makes it clear that the hardware designer can improve performance by reducing the number of clock cycles required for a program or the length of the clock cycle.

Example:3

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

Solution:

Let's first find the number of clock cycles required for the program on A:

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{Clock rate}_A}$$

$$10 \text{ seconds} = \frac{\text{CPU clock cycles}_A}{2 \times 10^9 \frac{\text{cycles}}{\text{second}}}$$

$$\text{CPU clock cycles}_A = 10 \text{ seconds} \times 2 \times 10^9 \frac{\text{cycles}}{\text{second}} = 20 \times 10^9 \text{ cycles}$$

CPU time for B can be found using this equation:

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B}$$

$$6 \text{ seconds} = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{\text{Clock rate}_B}$$

$$\text{Clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = \frac{0.2 \times 20 \times 10^9 \text{ cycles}}{\text{second}} = \frac{4 \times 10^9 \text{ cycles}}{\text{second}} = 4 \text{ GHz}$$

- To run the program in 6 seconds, B must have twice the clock rate of A.

1.4.3 Instruction Performance

- The performance equations e did not include any reference to the number of instructions needed for the program.
- The compiler clearly generated instructions to execute, and the computer had to execute the instructions to run the program.
- The execution time must depend on the number of instructions in a program, it equals the number of instructions executed multiplied by the average time per instruction.
- Therefore, the number of clock cycles required for a program can be written as

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \frac{\text{Average clock cycles}}{\text{per instruction}}$$

Clock cycles per instruction (CPI):

- Average number of clock cycles per instruction for a program or program fragment.

Using the Performance Equation:

Example:4

Suppose we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

Answer:

We know that each computer executes the same number of instructions for the program; let's call this number I.

First, find the number of processor clock cycles for each computer:

$$\text{CPU clock cycles}_A = I \times 2.0$$

$$\text{CPU clock cycles}_B = I \times 1.2$$

Now we can compute the CPU time for each computer:

$$\text{CPU time}_A = \text{CPU clock cycles}_A \times \text{Clock cycle time}$$

$$= I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps}$$

Likewise, for B:

$$\text{CPU time}_B = I \times 1.2 \times 500 \text{ ps} = 600 \times I \text{ ps}$$

Clearly, computer A is faster. The amount faster is given by the ratio of the execution times:

$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

We can conclude that computer A is 1.2 times as fast as computer B for this program.

1.4.4 The Classic CPU Performance Equation

- We can now write this basic performance equation in terms of instruction count (the number of instructions executed by the program), CPI, and clock cycle time: instruction count.
- The number of instructions executed by the program.

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

or, since the clock rate is the inverse of clock cycle time:

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

- These formulas are particularly useful because they separate the three key factors that affect performance.
- We can use these formulas to compare two different implementations or to evaluate a design alternative if we know its impact on these three parameters.

Understanding Program Performance:

- The performance of a program depends on the algorithm, the language, the compiler, the architecture, and the actual hardware.

- The following table summarizes how these components affect the factors in the CPU performance equation.

Hardware or software component	Affects what?	How?
Algorithm	Instruction count, possibly CPI	The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more floating-point operations, it will tend to have a higher CPI.
Programming language	Instruction count, CPI	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.
Compiler	Instruction count, CPI	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 complex ways.
Instruction set architecture	Instruction count, clock rate, CPI	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.

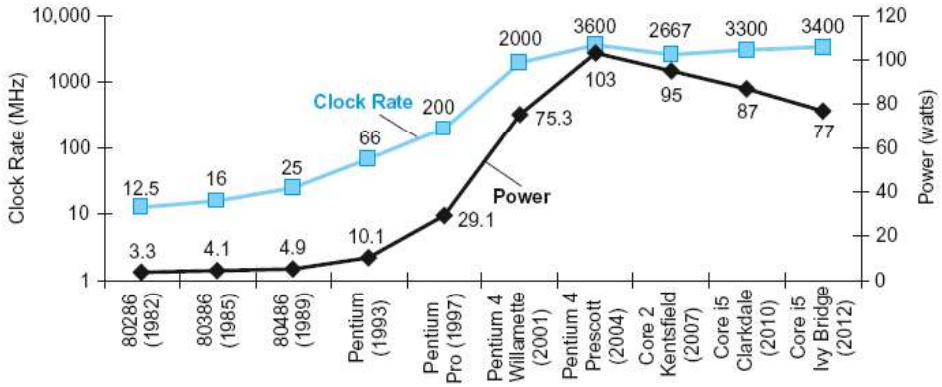
1.5 Power Wall: Power & Energy in Integrated circuits

Power is the biggest challenge for IC for two reasons:

1. First, power must be brought in and distributed around the chip which includes hundreds of pins and multiple interconnection layers just for power and ground.
2. Second, power is dissipated as heat and must be removed. Server chip can burn more than 100 watts and cooling the chip .The system is a major expense in warehouse scale computers.

- The increase in clock rate and power of eight generations of Intel microprocessors over 25 years.

- Both clock rate and power increased rapidly for decades, and then flattened off recently.
- Power provides a limit to what we can cool and in the PC Era the critical resource is energy.
- Battery life can trump performance in the personal mobile device.
- The architects of warehouse scale computers try to reduce the costs of powering and cooling 1,00,000 servers as the cost are high at this scale.
- The time can be measured in two ways one is seconds and another is MIPS.
- Measuring time in seconds is a safer measure for program performance than the rate like MIPS.
- The energy metric Joule is a better measure than a power rate like watts.



- The dominant technology for integrated circuits is called CMOS (complementary metal oxide semiconductor).
- For CMOS, the primary source of energy consumption is so-called dynamic energy that is, energy that is consumed when transistors switch states from 0 to 1 and vice versa.
- The dynamic energy depends on the capacitive loading of each transistor and the voltage applied:

Energy α Capacitive load x Voltage^2

- This equation is the energy of a pulse during the logic transition of 0→ 1→0 or 1→0→1. The energy of a single transition is then

Energy α 1/2 x Capacitive load x Voltage^2

- The power required per transistor is just the product of energy of a transition and the frequency of transitions:

Power α 1/2 x Capacitive load x Voltage^2 x Frequency switched

- Frequency switched is a function of the clock rate.
- The capacitive load per transistor is a function of both the number of transistors connected to an output (called the fan out) and the technology, which determines the capacitance of both wires and transistors.
- Even today about 40% of the power consumption is due to leakage.
- If transistors started leaking more then the whole process could become waste.
- To avoid the power problem, designers have already attached large devices to increase cooling, and they turn off parts of the chip that are not used in a given clock cycle.
- Many techniques are used to cool chips but it raises their power consumption .So computer designers slammed into a power wall, they needed a new way forward.

What is the maximum power a processor ever requires?

- If it attempts to draws more Power than a Power supply can provide, by drawing more current, the voltage will eventually drop which can cause the device to malfunction.
- Modern processors can vary widely in power consumption with high peak currents. Hence, they provide voltage indexing methods that allow the processor to slow down and regulate voltage within a wider margin. Obviously, doing so decreases the performance.

What is the sustained Power consumption?

- This Metric is called Thermal design power (TDP), since it determines the cooling requirement. Power supply is usually designed to match or exceed TDP.

- Failure to provide adequate cooling will allow the temperature to exceed the maximum value resulting in device failure.

Modern processors provide two features to manage heat.

1. Reduce clock rate, thereby reducing the power.
2. Thermal overload trip is activated to power down the chip.

Which metric is the right one for comparing processors: energy or power?

- Power is energy per unit time.

1 watt = 1 joule per second.

- Energy is a better metric because it is tied to a specific task and the time required for that task.
- Power consumption will be a useful measure if the workload is fixed.

1.6 Uniprocessors to Multiprocessors

- Increasing the clock speed of Uniprocessor has reached saturation and cannot be increased beyond a certain limit because of power consumption and heat dissipation issues.
- As the physical size of chip decreased, while the number of transistors/chip increased, clock speed increased, which boosted the heat dissipation across the chip to a dangerous level. Cooling & heat sink requirement issues were there.
- There were limitations in the use of silicon surface area.
- There were limitations in reducing the size of individual gates further.
- To gain Performance within a single core, many techniques like pipelining, super pipelined, superscalar architectures are used .
- Most of the early dual core processors were running at lower clock speeds, the rational behind is that a dual core processor with each running at 1GHz should be equivalent to a single core processor running at 2 GHz.
- The Problem is that this does not work in practice when the applications are not written to take advantage of the multiple processors. Until the software is

written this way, unthreaded applications will run faster on a single processor than on a dual core CPU.

- In Multi-core processors, the benefit is more on throughput than on response time.
- In the past, programmers could rely on innovations in the hardware, Architecture and compilers to double performance of their programs every 18 months without having to change a line of code.
- Today, for programmers to get significant improvement in response time, they need to rewrite their programs to take advantage of multiple processors and also they have to improve performance of their code as the number of core increases.
- Ability to write Parallel programs.
- Care must be taken to reduce Communication and Synchronization overhead. Challenges in Scheduling, load balancing have to be addressed.

1. 7 Instructions

- The words of a computer's language are called **instructions,** and its vocabulary is called an **instruction set**.
- The instruction set will describe the functions of architecture so computer designer must know about the instruction set.
- The instruction set comes from MIPS technologies and 3 popular instruction sets are as follows:
    1. ARM V7 is similar to MIPS .More than 9 billion chips with ARM processor were manufactured in 2011,making it the most popular instruction set in the world.
    2. Intel X86 powers both the PC and the cloud of post PC era.
    3. ARM V8 extends the addressing size of ARM V7 from 32 to 64 bits.

- The instructions and data of man y types can be stored in memory as numbers; it is called the **stored program concept.**

1. Operations of the Computer Hardware
2. Operands of the Computer Hardware
3. Representing Instructions in the Computer
4. Logical Operations

1.7.1 Operations of the Computer Hardware

- Every computer must be able to perform arithmetic operations. The MIPS assembly language notation for performing addition operation is

add a, b, c

- It instructs a computer to add the two variables **b** and **c** and to put their sum in **a**. Each MIPS arithmetic instruction performs only one operation and must always have exactly three variables.
- For example, suppose we want to place the sum of variables **b, c, d,** and **e** into variable **a**. The following sequence of instructions adds the four variables:

```
add  a, b, c        // The sum of b and c is placed in a.
add  a, a, d        // The sum of b, c, and d is now in a.
add  a, a, e        // The sum of b, c, d, and e is now in a.
```

- Thus, it takes three instructions to take the sum of four variables.

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; data in registers
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	Used to add constants
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Data from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Data from register to memory

- Hardware for a variable number of operands is more complicated than hardware for a fixed number.
- Hardware technology has three design principles that will give detailed information about fixed and variable number of operands to perform any operation.
- The three principles of hardware design are:

Design Principle 1: Simplicity favors regularity.

Design Principle 2: Smaller is faster.

Design Principle 3: Make the common case fast.

Design Principle 4: Good design demands good compromises.

Design Principle 1: Simplicity favors regularity.

To understand the concept of first design principle, let us consider the following examples.

Example: 1

Compiling Two C Assignment Statements into MIPS: This segment of a C program contains the five variables a, b, c, d, and e.

a = b + c;

d = a – e;

The translation from C to MIPS assembly language instructions is performed by the compiler. Show the MIPS code produced by a compiler.

Answer:

A MIPS instruction operates on two source operands and places the result in one destination operand. Hence, the two simple statements above compile directly into these two MIPS assembly language instructions:

```
add  a, b, c
sub  d, a, e
```

Example: 2

A somewhat complex statement contains the five variables f, g, h, i, and j:

f = (g + h) – (i + j)

What might a C compiler produce?

Answer:

The compiler must break this statement into several assembly instructions since only one operation is performed per MIPS instruction.

The first MIPS instruction calculates the sum of g and h. We must place the result somewhere, so the compiler creates a temporary variable called t0.

```
add t0,g,h                  //temporary variable t0 contains g + h
```

Second operation is subtract, for that we need to calculate the sum of i and j before we can subtract.

Thus, the second instruction places the sum i and j in another temporary variable created by the compiler, called t1

add t1,i,j // temporary variable t1 contains i + j

Finally, the subtract instruction subtracts the second sum from the first and places the difference in the variable f, completing the compiled code:

sub f,t0,t1 // f gets t0 – t1, which is (g + h)–(i + j)

1.7.2 Operands of the Computer Hardware

- Unlike programs in high-level languages, the operands of arithmetic instructions are restricted; they must be from a limited number of special locations built directly in hardware called **registers**.
- Registers are the bricks of computer construction. Registers are primitives used in hardware design that are also visible to the programmer when the computer is completed.
- MIPS architecture has 32 bit registers and group of 32 bits are called word .
- The reason for the limit of 32 registers is based on second design principles of hardware technology:

Design Principle 2: Smaller is faster.

- Registers are the fast place to hold data in a computer.A very large number of registers may increase the clock cycle time because it takes electronic signals longer to travel it.
- MIPS convention use two-character names following a dollar sign to represent a register. We need temporary registers to compile the program into MIPS instructions.

Example:3

Compiling a C assignment using registers. It is the compiler's job to associate program variables with registers. Take, for instance, the assignment statement from our earlier example:

f = (g + h) – (i + j);

The variables f, g, h, i, and j are assigned to the registers $s0, $s1, $s2, $s3, and $s4, respectively. What is the compiled MIPS code?

Answer:

```
add $t0,$s1,$s2          //register $t0 contains g + h
add $t1,$s3,$s4          // register $t1 contains i + j
sub $s0,$t0,$t1          //f gets $t0 – $t1, which is (g + h)–(i + j
```

1.7.2.1 Memory Operands:

- Programming languages have simple variables that contain single data elements , but they also have more complex data structures like arrays and structures.
- These complex data structures can contain more data elements than there are registers in a computer. Now computer cannot represent and access such large structures.
- To access the large structures element memory operands are used.
- The processor can keep only a small amount of data in registers, but computer memory contains millions of data elements.
- Hence, data structures (arrays and structures) are kept in memory. In MIPS instructions arithmetic operations occur only on registers.

Data transfer instructions:

- Instructions that transfer data between memory and registers are called **data transfer instructions**.
- To access a word in memory, the instruction must supply the memory **address**. Memory is just a large, single-dimensional array, with the address acting as the index to that array, starting at 0.

Example:4

- The address of the third data element is 2, and the value of Memory[2] is 10.

- The data transfer instruction that copies data from memory to a register is traditionally called *load*.
- The format of the load instruction is the name of the operation followed by the register to be loaded, then a constant and register used to access memory.
- The sum of the constant portion of the instruction and the contents of the second register forms the memory address.
- In MIPS architecture the actual name for memory address is lw, standing for load word.

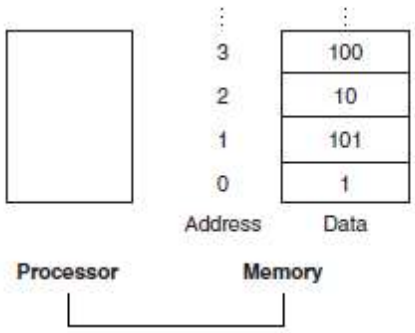


Figure : Memory addresses and contents of memory at those locations.

- The compiler can then place the proper starting address into the data transfer instructions.
- Many programs use 8-bit *bytes* .Therefore, the address of a word matches the address of one of the 4 bytes within the word. Hence, addresses of sequential words differ by 4.

Alignment restriction:

- In MIPS, words must start at addresses that are multiples of 4. This requirement is called an alignment restriction**.** A requirement that data be aligned in memory on natural boundaries.

Big Endian and Little Endian:

- Computers divide 8 bit bytes into two parts: Address of the left most byte is called “big end” and the right most is called “little end”.

Store instruction:

- The instruction complementary to load is traditionally called *store;* it copies data from a register to memory.
- The format of a store is similar to that of a load: the name of the operation, followed by the register to be stored then the offset array element, and finally the base register.
- The MIPS address is specified by the part of the address in the constant and part of the address in the register. The actual MIPS name for store is sw, standing for store word.

Example:5

Compiling an Assignment When an Operand is in Memory. Let's assume that A is an array of 100 words and that the compiler has associated the variables g and h with the registers $s1 and $s2 as before. Let's also assume that the starting address, or base address, of the array is in $s3. Compile this C assignment statement:

g = h + A[8];

Answer:

There is a single operation in this assignment statement, one of the operands is in memory, so we must first transfer A[8] to a register. The address of this array element is the sum of the base of the array A, found in register $s3, plus the number to select element 8. The data should be placed in a temporary register for use in the next instruction.

The first compiled instruction is

```
lw $t0,8($s3)          // Temporary reg $t0 gets A[8]
```

The following instruction can operate on the value in $t0 (which equals A[8]) since it is in a register. The instruction must add h (contained in $s2) to A[8] ($t0) and put the sum in the register corresponding to g (associated with $s1):

```
add $s1,$s2,$t0        // g = h + A[8]
```

The constant in a data transfer instruction is called the offset, and the register added to form the address is called the base register.

Example:6

Compiling Using Load and Store: Assume variable h is associated with register $s2 and the base address of the array A is in $s3. What is the MIPS assembly code for the C assignment statement below?

A[12] = h + A[8];

Answer:

There is a single operation in the C statement, now two of the operands are in memory, so we need even more MIPS instructions. The first two instructions are the same as the prior example, except this time we use the proper offset for byte addressing in the load word instruction to select A[8], and the add instruction places the sum in $t0:

```
lw $t0,32($s3)          // Temporary reg $t0 gets A[8]
add $t0,$s2,$t0         //Temporary reg $t0 gets h + A[8]
```

The final instruction stores the sum into A[12], using 48 as the offset and register $s3 as the base register.

```
sw $t0,48($s3)          // Stores h + A[8] back into A[12]
```

Spilling registers:

- Many programs have more variables than computers have registers. Consequently, the compiler tries to keep the most frequently used variables in registers and places the rest in memory, using loads and stores to move variables between registers and memory.
- The process of putting less commonly used variables (or those needed later) into memory is called **spilling** registers.

1.7.2 .2 Constant or Immediate Operands

- Many times a program will use a constant in an operation for example, incrementing an index to point to the next element of an array.
- The MIPS arithmetic instructions have a constant as an operand.
- A MIPS arithmetic instruction can read two registers, operate on them, and write the result.
- The add instruction with one constant operand is called *add immediate* or addi. To add 4 to register $s3, we just write

```
addi $s3,$s3,4          // $s3 = $s3 + 4
```

Design Principle 3: Make the common case fast.

- Constant operands occur frequently by including constants inside arithmetic instructions; they are much faster than if constants were loaded from memory.

1.7.3 Representing Instructions in the Computer

- Instructions are kept in the computer as a series of high and low electronic signals and may be represented as numbers.
- Each piece of an instruction can be considered as an individual number, and placing these numbers side by side forms the instruction.

Instruction format:

- A form of representation of an instruction composed of fields of binary numbers.

Machine language:

- Binary representation used for communication within a computer system.

Registers:

- Registers are referred to by almost all instructions, there must be a convention to map register names into numbers.
- In MIPS assembly language, registers $s0 to $s7 map onto registers 16 to 23, and registers $t0 to $t7 map onto registers 8 to 15.
- Hence, $s0 means register 16, $s1 means register 17, $s2 means register 18, . . . , $t0 means register 8, $t1 means register 9.

Example:1

Translating a MIPS Assembly Instruction into a Machine Instruction. Let us consider the following MIPS Instruction

add $t0,$s1,$s2

Translate first as a combination of decimal numbers and then of binary numbers.

Answer:

The decimal representation is

0	17	18	8	0	32

- Each of these segments of an instruction is called a *field*.
- The first and last fields (0 and 32) in combination tell the MIPS computer that this instruction performs addition.
- The second field gives the number of the register that is the first source operand of the addition operation (17 = $s1), and the third field gives the other source operand for the addition (18 = $s2).
- The fourth field contains the number of the register that is to receive the sum (8 = $t0).
- The fifth field is unused in this instruction, so it is set to 0. Thus, this instruction adds register $s1 to register $s2 and places the sum in register $t0.
- This instruction can also be represented as fields of binary numbers as opposed to decimal:

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Hexadecimal:

- Computer can use of binary numbers to read and write data. So we can use higher base than binary that can be easily converted into binary.
- All computer data sizes are multiples of 4, in that hexadecimal numbers are popular.
- Since base 16 is a power of 2. So, we can convert by replacing each group of four binary digits by a single hexadecimal digit, and vice versa.

Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}

Hexadecimal	Binary	Hexadecimal	Binary
8_{hex}	1000_{two}	c_{hex}	1100_{two}
9_{hex}	1001_{two}	d_{hex}	1101_{two}
a_{hex}	1010_{two}	e_{hex}	1110_{two}
b_{hex}	1011_{two}	f_{hex}	1111_{two}

- To avoid confusion between various numbers, we can use subscript values for example
    1. decimal numbers with ten
    2. binary numbers with two
    3. Hexadecimal numbers with hex.
- If there is no subscript, the default is base 10.

Example:2

Convert the hexadecimal into binary: eca8 6420$_{hex}$

e	C	a	8	6	4	2	0
1110	1100	1010	1000	0110	0100	0010	0000

Answer: 1110 1100 1010 1000 0110 0100 0010 0000$_{two}$

MIPS Fields:

- A MIPS field has two kinds of format such as:
    1. R-type or R-format (for register)
    2. I-type or I-format (for immediate)

R-format:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

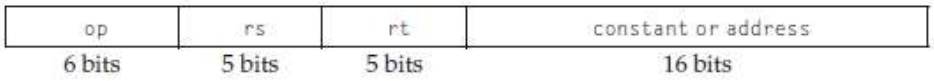
- Here is the meaning of each name of the fields in MIPS instructions:
    - *op:* Basic operation of the instruction, traditionally called the opcode.opcode denotes the operation and format of an instruction
    - rs: The first register source operand.

- rt: The second register source operand.
- rd: The register destination operand. It gets the result of the operation.
- shamt: Shift amount.
- funct: Function. This field selects the specific variant of the operation in the op field and is sometimes called the function code.

Design Principle 4: **Good design demands good compromises**.

I-format:

- It is used by the immediate and data transfer instructions.



- This constant is used to select elements from arrays or data structures.

1.8 Logical Operations

- The first computers perform their operations on full words. After that it is capable of performing operation on fields of bits within a word or even on individual bits.
- Using single bit of information some instructions can be executed in computer such instructions are called logical operations.
- Logical operation is an instruction in which the quality being operated on bit and the results of the operation can have two values (0 and 1).It include AND, OR, NAND, XOR and NOR.

Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit NOT	~	~	nor

1.8.1 Logical Shift Operation

- Shift Operation moves all the bits in a word to the left or right, filling the emptied bits with 0s.
- Based on the direction of shifting it can be classified into two types:
  1. Shift left
  2. Shift Right.

For example, if register $s0 contained

0000 0000 0000 00000 000 0000 0000 0000 1001two= 9_{ten}

and the instruction to shift left by 4 was executed, the new value would look like this:

0000 0000 0000 0000 0000 0000 0000 1001 0000two= 144_{ten}

- The actual name of the two MIPS shift instructions are called shift left logical (sll) and shift right logical (srl).

Shift left logical:

- It moves all the bits in a word to the left side and empty position is filled with 0s.
- The following instruction performs the operation above, assuming that the result should go in register $t2:

sll $t2,$s0,4 //reg $t2 = reg $s0 << 4 bits

- We delayed explaining the shamt field in the R- format.
- It stands for shift amount and is used in shift instructions. Hence, the machine language version of the instruction above is

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0

The encoding of sll is 0 in both the op and funct fields, rd contains $t2, rt contains $s0, and shamt contains 4. The rs field is unused, and thus is set to 0.

- Shift left logical provides a bonus benefit. Shifting left by i bits gives the same result as multiplying by 2^i . For example, the above sll shifts by 4, which gives the same result as multiplying by 24 or 16.

- The first bit pattern above represents 9, and 9 ¥ 16 = 144, the value of the second bit pattern.

1.8.2 Logical AND operation

- A logical AND operation is a bit by bit operation that places a 1 in the result only if both bits of the operands are 1.

For example, if register $t2 still contains

$0000\ 0000\ 0000\ 0000\ 0000\ 1101\ 0000\ 0000_{two}$

and register $t1 contains

$0000\ 0000\ 0000\ 0000\ 0011\ 1100\ 0000\ 0000_{two}$

then, after executing the MIPS instruction and

$t0,$t1,$t2 // reg $t0 = reg $t1 & reg $t2 the value of register $t0 would be

$0000\ 0000\ 0000\ 0000\ 0000\ 1100\ 0000\ 0000_{two}$

1.8.3 Logical OR operation

- A logical OR operation is a bit by bit operation that places a 1 in the result if either operand bit is a 1.

$t0,$t1,$t2 // reg $t0 = reg $t1 | reg $t2 is this value in register $t0:

$0000\ 0000\ 0000\ 0000\ 0011\ 1101\ 0000\ 0000_{two}$

1.8.4 Logical NOT operation

- A logical NOT operation is a logical bit by bit operation with one operand and places a 1 in the result if one operand bit is a 0, and 0 in the result if one operand bit is a 1 .

For example, A NOR 0 = NOT (A OR 0) = NOT (A).

If the register $t1 is unchanged from the preceding example and register $t3 has the value 0, the result of the MIPS instruction nor $t0,$t1,$t3 # reg $t0 = ~ (reg $t1 | reg $t3) is this value in register $t0:

$1111\ 1111\ 1111\ 1111\ 1100\ 0011\ 1111\ 1111_{two}$

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; overflow detected
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow detected
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $$s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory

Figure above shows the relationship between the C and Java operators and the MIPS instructions. Constants are useful in AND and OR logical operations as well as in arithmetic operations, so MIPS also provides the instructions and immediate (andi) and or immediate (ori). Constants are rare for NOR, since its main use is to invert the bits of a single operand; thus, the hardware has no immediate version.

1.9 Control Operations

- Based on the input data and the values created during computation, different instructions execute.
- Decision making is commonly represented in programming languages to indicate that based on the condition what operation has to be executed by the computer.
- Using the *if* statement, sometimes combined with *go to* statements and labels. MIPS assembly language includes two decision-making instructions, similar to an *if* statement with a *go to*.
- The first instruction is

beq register1, register2, L1

- This instruction means go to the statement labeled L1 if the value in register1 equals the value in register2. The mnemonic beq stands for *branch if equal*.

- The second instruction is

bne register1, register2, L1

- It means go to the statement labeled L1 if the value in register1 does *not* equal the value in register2.
- The mnemonic bne stands for *branch if not equal*. These two instructions are traditionally called conditional branches.

1.9.1 Conditional branch

- An instruction that requires the comparison of two values and that allows for a subsequent transfer of control to a new address in the program based on the outcome of the comparison.

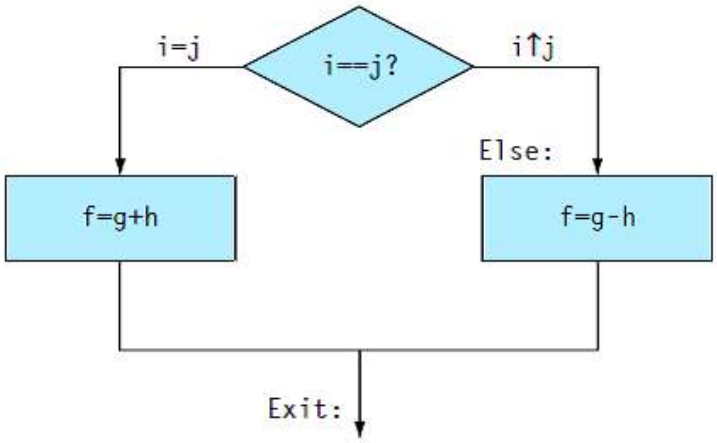
Compiling if-then-else into Conditional Branches:

Example:1

- In the following code segment, f, g, h, i, and j are variables. If the five variables f through j correspond to the five registers $s0 through $s4, what is the compiled MIPS code for this C if statement?

if (i == j) f = g + h; else f = g – h;

Answer:



- Above flowchart shows what the MIPS code should do.
- The first expression compares for equality, so we would want beq.
- Second instruction use bne,the code will be more efficient to test opposite condition.

- Here the opposite condition to branch over the code that performs the subsequent then part of the if

` **bne $s3,$s4,Else** // go to Else if i ≠ j

- The next assignment statement performs a single operation, and if all the operands are allocated to registers, it is just one instruction:

add $s0,$s1,$s2 // f = g + h (skipped if i ≠ j)

- To distinguish between conditional and unconditional branches, the MIPS name for this type of instruction is jump, abbreviated as j .

j Exit // go to Exit

- The label Exit that is after this instruction, showing the end of the if-then-else compiled code:

Else:sub $s0,$s1,$s2 // f = g – h (skipped if i = j)

Exit:

1.9.2 Loops

Compiling a while Loop in C

Example:2

Here is a traditional loop in C:

```
while (save[i] == k)
i += 1;
```

Assume that i and k correspond to registers $s3 and $s5 and the base of the array save is in $s6. What is the MIPS assembly code corresponding to this C segment?

Answer:

- The first step is to load save[i] into a temporary register. Before we can load save[i] into a temporary register, we need to have its address.
- Before we can add i to the base of array save to form the address, we must multiply the index i by 4 due to the byte addressing problem.
- We need to add the label Loop to it so that we can branch back to that instruction at the end of the loop:

Loop: sll $t1,$s3,2 // Temp reg $t1 = 4 * i

- To get the address of save[i], we need to add $t1 and the base of save in $s6:

add $t1,$t1,$s6 // $t1 = address of save[i]

- Now we can use that address to load save[i] into a temporary register:

  lw $t0,0($t1) // Temp reg $t0 = save[i]

- The next instruction performs the loop test, exiting if save[i] ≠ k:

  bne $t0,$s5, Exit // go to Exit if save[i] ≠ k

- The next instruction adds 1 to i:

  add $s3,$s3,1 // i = i + 1

- The end of the loop branches back to the while test at the top of the loop. We just add the Exit label after it, and we're done:

  j Loop // go to Loop

  Exit:

Basic block:

- A sequence of instructions without branches (except possibly at the end) and without branch targets or branch labels (except possibly at the beginning)

1.9.3 Comparison Instructions

- MIPS compilers use the slt, slti, beq, bne, and the fixed value of to create all relative conditions: equal, not equal, less than, less than or equal, greater than, greater than or equal.

1.9.5 Case/Switch Statement

- Most programming languages have a case or switch statement that allows the programmer to select one of many alternatives depending on a single value.
- Switch statement can be to implemented in two ways:
  - Using chain of if-then-else statements
  - Using jump address table
- It is also called jump table. It is a table of addresses of alternative instruction sequences, called a jump address table, and the program needs only to index into the table and then jump to the appropriate sequence.
- The jump table is then just an array of words containing addresses that correspond to labels in the code.

- Computers like MIPS include a jump register instruction (jr), meaning an unconditional jump to the address specified in a register.
- The program loads the appropriate entry from the jump table into a register,and then it jumps to the proper address using a jump register

1.10 Addressing Modes

The MIPS and ARM Addressing Modes:

- Addressing Mode is one of several addressing rule (regimes) surrounded by their varied use of operands and/or addresses.
- Multiple forms of addressing are generically called addressing modes. MIPS have the following addressing modes are the following:

1.10.1 Immediate addressing

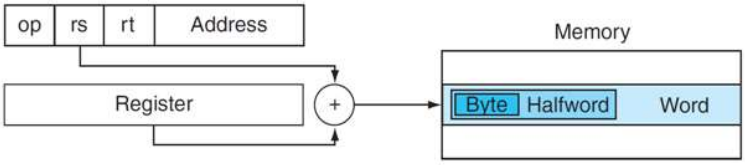
- Where the operand is a constant within the instruction itself

1.8.2 Register addressing

- Where the operand is in register

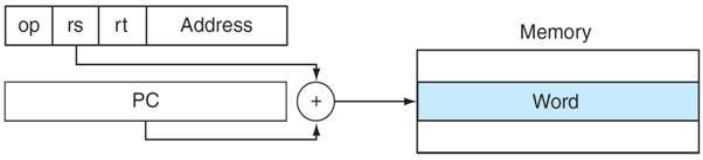
1.8.3 Base or displacement addressing

- Where the operand is at the memory location whose address is the sum of a register and a constant in the instruction



1.8.4 PC-relative addressing

- Where the address is the sum of the PC and a constant in the instruction



1.8.5 Pseudo direct addressing

- Where the jump address is the 26 bits of the instruction concatenated with the upper bits of the PC