Digital Computer System

Daryl Hobbs

DIGITAL COMPUTER SYSTEM

DIGITAL COMPUTER SYSTEM

Daryl Hobbs

BIBLIOTEX
Digital Library

Digital Computer System
by Daryl Hobbs

Ebook ISBN: 9781984663948

Published by:
Bibliotex
Canada
Website: www.bibliotex.com

Contents

1

Introduction

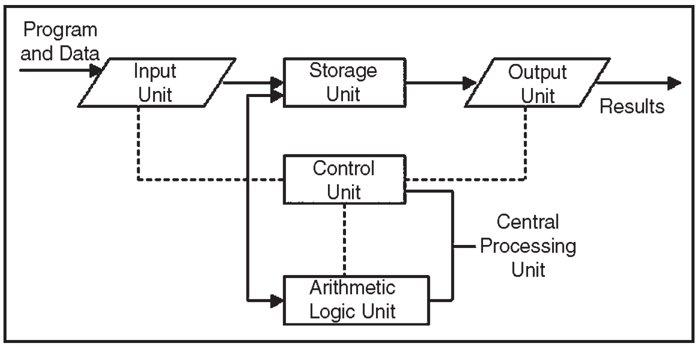
Digital computer systems consist of three distinct units. *These units are as follows*:

- Input unit
- Central Processing unit
- Output unit

These units are interconnected by electrical cables to permit communication between them. This allows the computer to function as a system.

Input Unit

A computer must receive both data and program statements to function properly and be able to solve problems. The method of feeding data and programs to a computer is accomplished by an input device. Computer input devices read data from a source, such as magnetic disks, and translate that data into electronic impulses for transfer into the CPU. Some typical input devices are a keyboard, a mouse, or a scanner.



Central Processing Unit

The brain of a computer system is the central processing unit (CPU). The CPU processes data transferred to it from one of the various input devices. It then transfers either an intermediate or final result of the CPU to one or more output devices.

A central control section and work areas are required to perform calculations or manipulate data. The CPU is the computing center of the system. It consists of a control section, an arithmetic-logic section as shown in figure, and an internal storage section (main memory). Each section within the CPU serves a specific function and has a particular relationship with the other sections within the

Cpu.Control Section

The control section directs the flow of traffic (operations) and data. It also maintains order within the computer. The flow of control is indicated by dotted arrows in figure.

The control section selects one program statement at a time from the program storage area, interprets the statement, and sends the appropriate electronic impulses to the arithmetic-logic and storage sections so they can carryout the instructions.

The control section does not perform actual processing operations on the data. The control section instructs the input device on when to start and stop transferring data to the input storage area. It also tells the output device when to start and stop receiving data from the output storage area.

Arithmetic-Logic Section

The arithmetic-logic section performs arithmetic operations, such as addition, subtraction, multiplication, and division. Through internal logic capability, it tests various conditions encountered during processing and takes action based on the result.As indicated by the solid arrows in figure, data flows between the arithmetic-logic section and the internal storage section during processing.

Specifically, data is transferred as needed from the storage section to the arithmetic-logic section, processed, and returned to internal storage. At no time does processing take placein the storage section.

Data maybe transferred back and forth between these two sections several times before processing is completed. The results are then transferred from internal storage to an output device, as indicated by the solid arrow in figure below.

The binary addition and multiplication tables are:

(

0 + 0 = 0

0 + 1 = 1

1 + 1 = 10

1 + 0 = 1

0 × 0 = 0

0 × 1 = 0

1 × 1 = 1

1 × 0 = 0

) (4)

Note that if carries are ignored,7 subtraction of two single-digit binary numbers yields the same bit as addition. Computers use high and low voltage values to express a bit, and an array of such voltages express numbers akin to positional notation. Logic circuits perform arithmetic operations.

Exercise: Add twenty-five and seven in base 2. Note the carries that might occur. Why is the result "nice"?

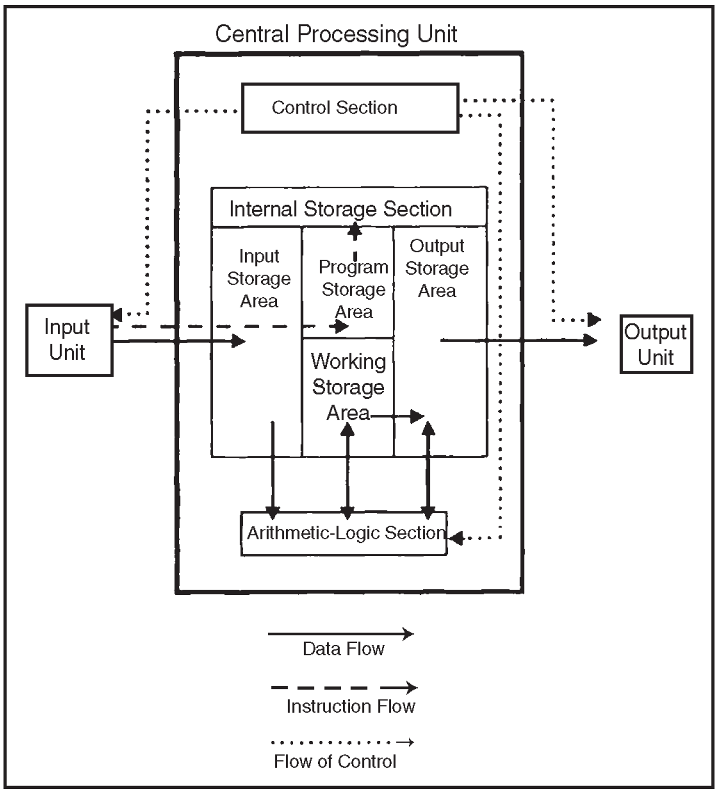
Solution: $25=11011_2$ and $7=111_2$. We find that $11001_2 + 111_2 = 100000_2=32$.

Also note that the logical operations of AND and OR are equivalent to binary addition (again if carries are ignored). The variables of logic indicate truth or falsehood. *A*Â"*B*, the AND of *A* and *B*, represents a statement that both *A* and *B* must be true for the statement to be true.

You use this kind of statement to tell search engines that you want to restrict hits to cases where both of the events *A* and *B* occur. *A*Ã"*B*, the OR of *A* and *B*, yields a value of truth if either is true.

Note that if we represent truth by a "1" and falsehood by a "0," *binary multiplication corresponds to AND and addition (ignoring carries) to OR*. The Irish mathematician George Boole discovered this equivalence in the mid-nineteenth century. It laid the foundation for what we now call Boolean algebra, which expresses as equations logical statements.

More importantly, any computer using base-2 representations and arithmetic can also easily evaluate logical statements. This fact makes an integer-based computational device much more powerful than might be apparent.



Internal Storage Section

The internal storage section is sometimes called primary storage, main storage, or main memory, because this section functions similar to our own human memory. The storage section serves four purposes; three relate to retention (holding) of data during processing.

First, as indicated by the solid arrow as shown in figure, data is transferred from an input device to the inputstorage area where it remains until the computers ready to process it. Second, a workingstorage area ("scratch pad" memory) within the storage section holds both the data being processed and the intermediate results of the arithmetic-logic operations. Third, the storage section retains the processing results in the output storage area. From there the processing results can be transferred to an output device. The fourth storage section, the program storage area, contains the program statements transferred from an input device to process the data. Please note that the four areas (input, working storage, output, and program storage) are NOT famed in size or location but are determined by individual program requirements.

Output Unit

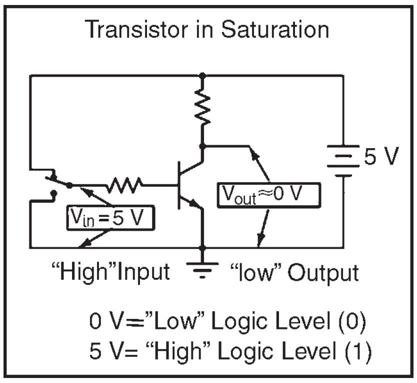
As program statements and data are received by the CPU from an input device, the results of the processed data are sent from the CPU to an OUTPUT DEVICE. These results are transferred from the output storage area onto an output medium, such as a floppy disk, hard drive, video display, printer, and so on.

DIGITAL SYSTEM

While the binary numeration system is an interesting mathematical abstraction, we haven't yet seen its practical application to electronics. What makes binary numeration so important to the application of digital electronics is the ease in which bits may be represented in physical terms. Because a binary bit can only have one of two different values, either 0 or 1, any physical medium capable of switching between two saturated states may be used to represent a bit. Consequently, any physical system capable of representing binary bits is able to represent numerical quantities, and potentially has the ability to manipulate those numbers. Electronic circuits are physical systems that lend themselves well to the representation of binary numbers. Transistors, when operated at their bias limits, may be in one of two different states: either cutoff.

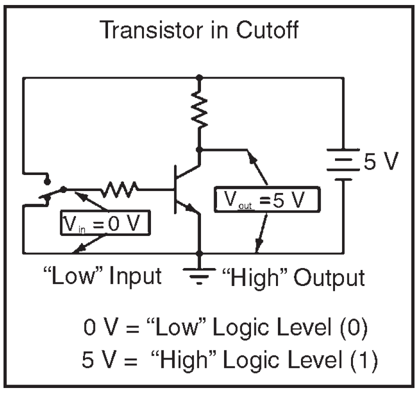
If a transistor circuit is designed to maximize the probability of falling into either one of these states (and not operating in the linear, or active, mode), it can serve as a physical representation of a binary bit. A voltage signal measured at the output of such a circuit may also serve as a representation of a single bit, a low voltage representing a binary "0" and a (relatively) high voltage representing a binary "1." Consider the following transistor circuit:

In this circuit, the transistor is in a state of saturation by virtue of the applied input voltage (5 volts) through the two-position switch. Because it's saturated, the transistor drops very little voltage between collector and emitter, resulting in an output voltage of (practically) 0 volts.



If we were using this circuit to represent binary bits, we would say that the input signal is a binary "1" and that the output signal is a binary "0." Any voltage close to full supply voltage (measured in reference to ground, of course) is considered a “1” and a lack of voltage is considered a “0.” Alternative terms for these voltage levels are high (same as a binary “1”) and low (same as a binary “0”).

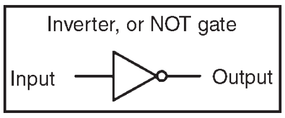
A general term for the representation of a binary bit by a circuit voltage is logic level. Moving the switch to the other position, we apply a binary “0” to the input and receive a binary “1” at the output:



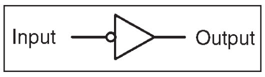
What we’ve created here with a single transistor is a circuit generally known as a logic gate, or simply gate. A gate is a special type of amplifier circuit designed to accept and generate voltage signals corresponding to binary 1’s and 0’s. As such, gates are not intended to be used for amplifying analog signals (voltage signals between 0 and full voltage). Used together, multiple gates may be applied to the task of binary number storage (memory circuits) or manipulation (computing circuits), each gate’s output representing one bit of a multi-bit binary number.

The gate shown here with the single transistor is known as an inverter, or NOT gate, because it outputs the exact opposite digital signal as what is input. For convenience, gate circuits are generally represented by their own symbols rather than by their constituent transistors and resistors.

The following is the symbol for an inverter:

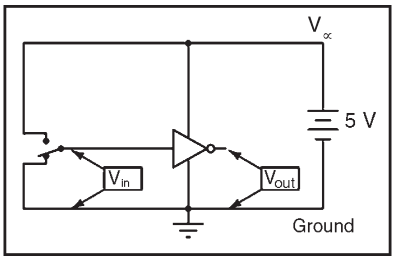


An alternative symbol for an inverter is shown here:

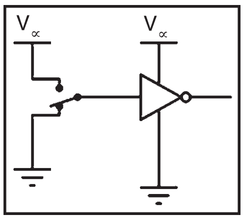


Notice the triangular shape of the gate symbol, much like that of an operational amplifier. As was stated before, gate circuits actually are amplifiers. The small circle, or "bubble" shown on either the input or output terminal is standard for representing the inversion function. As you might suspect, if we were to remove the bubble from the gate symbol, leaving only a triangle, the resulting symbol would no longer indicate inversion, but merely direct amplification. Such a symbol and such a gate actually do exist, and it is called a buffer, the subject of the next section. Like an operational amplifier symbol, input and output connections are shown as single wires, the implied reference point for each voltage signal being "ground." In digital gate circuits, ground is almost always the negative connection of a single voltage source (power supply).

Dual, or "split," power supplies are seldom used in gate circuitry. Because gate circuits are amplifiers, they require a source of power to operate. Like operational amplifiers, the power supply connections for digital gates are often omitted from the symbol for simplicity's sake. If we were to show all the necessary connections needed for operating this gate, the schematic would look something like this:



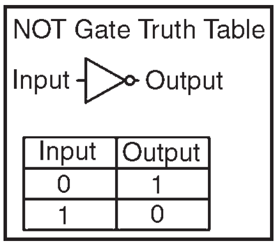
Power supply conductors are rarely shown in gate circuit schematics, even if the power supply connections at each gate are. Minimizing lines in our schematic, we get this:



"V_{cc}" stands for the constant voltage supplied to the collector of a bipolar junction transistor circuit, in reference to ground. Those points in a gate circuit marked by the label "V_{cc}" are all connected to the same point, and that point is the positive terminal of a DC voltage source, usually 5 volts. As we will see in other sections of this chapter, there are quite a few different types of logic gates, most of which have multiple input terminals for accepting more than one signal.

The output of any gate is dependent on the state of its input(s) and its logical function.

One common way to express the particular function of a gate circuit is called a truth table. Truth tables show all combinations of input conditions in terms of logic level states (either "high" or "low," "1" or "0," for each input terminal of the gate), along with the corresponding output logic level, either "high" or "low." For the inverter, or NOT, circuit just illustrated, the truth table is very simple indeed:



Input	Output
0	1
1	0

Truth tables for more complex gates are, of course, larger than the one shown for the NOT gate. A gate's truth table must have as many rows as there are possibilities for unique input combinations. For a single-input gate like the NOT gate, there are only two possibilities, 0 and 1. For a two input gate, there are four possibilities (00, 01, 10, and 11), and thus four rows to the corresponding truth table. For a three-input gate, there are eight possibilities (000, 001, 010, 011, 100, 101, 110, and 111), and thus a truth table with eight rows are needed. The mathematically inclined will realise that the number of truth table rows needed for a gate is equal to 2 raised to the power of the number of input terminals.

MEMORY MANAGEMENT

The memory management subsystem is one of the most important parts of the operating system. Since the early days of computing, there has been a need for more memory than exists physically in a system. Strategies have been developed to overcome this limitation and the most successful of these is virtual memory. Virtual memory makes the system appear to have more memory than it actually has by sharing it between competing processes as they need it.

Large Address Spaces

The operating system makes the system appear as if it has a larger amount of memory than it actually has. The virtual memory can be many times larger than the physical memory in the system,

Protection

Each process in the system has its own virtual address space. These virtual address spaces are completely separate from each other and so a process running one application cannot affect another. Also, the hardware virtual memory mechanisms allow areas of memory to be protected against writing. This protects code and data from being overwritten by rogue applications.

Memory Mapping

Memory mapping is used to map image and data files into a processes address space. In memory mapping, the contents of a file are linked directly into the virtual address space of a process.

Fair Physical Memory Allocation

The memory management subsystem allows each running process in the system a fair share of the physical memory of the system,

Shared Virtual Memory

Although virtual memory allows processes to have separate (virtual) address spaces, there are times when you need processes to share memory. For example there could be several processes in the system running the bash command shell.

Rather than have several copies of bash, one in each processes virtual address space, it is better to have only one copy in physical memory and all of the processes running bash share it. Dynamic libraries are another common example of executing code shared between several processes.

Abstract Model of Virtual Memory

Before considering the methods that Linux uses to support virtual memory it is useful to consider an abstract model that is not cluttered by too much detail. As the processor executes a programme it reads an instruction from memory and decodes it. In decoding the instruction it may need to fetch or store the contents of a location in memory. The processor then executes the instruction and moves onto the next instruction in the programme. In this way the processor is always accessing memory either to fetch instructions or to fetch and store data.

Demand Paging

As there is much less physical memory than virtual memory the operating system must be careful that it does not use the physical memory inefficiently. One way to save physical memory is to only load virtual pages that are currently being used by the executing programme.

For example, a database programme may be run to query a database. In this case not all of the database needs to be loaded into memory, just those data records that are being examined. If the database query is a search query then it does not make sense to load the code from the database programme that deals with adding new records. This technique of only loading virtual pages into memory as they are accessed is known as demand paging.

Swapping

If a process needs to bring a virtual page into physical memory and there are no free physical pages available, the operating system must make room for this page by discarding another page from physical memory. If the page to be discarded from physical memory came from an image or data file and has not been written to then the page does not need to be saved. Instead it can be discarded and if the process needs that page again it can be brought back into memory from the image or data file.

However, if the page has been modified, the operating system must preserve the contents of that page so that it can be accessed at a later time. This type of page is known as a *dirty* page and when it is removed from memory it is saved in a special sort of file called the swap file. Accesses to the swap file are very long relative to the speed of the processor and physical memory and the operating system must juggle the need to write pages to disk with the need to retain them in memory to be used again.

Shared Virtual Memory

Virtual memory makes it easy for several processes to share memory. All memory access are made via page tables and each process has its own separate page table.

For two processes sharing a physical page of memory, its physical page frame number must appear in a page table entry in both of their page tables shows two processes that each share physical page frame number 4. For process *X* this is virtual page frame number 4 whereas for process *Y* this is virtual page frame number 6. This illustrates an interesting point about sharing pages: the shared physical page does not have to exist at the same place in virtual memory for any or all of the processes sharing it.

Processors and Memory

There are two main components which have been part of nearly all computer systems ever designed and built. The first is called the processor (known also as the Central Processing Unit or CPU)and the second is called the memory. The processor is the part of a computer system which does the actual computing.

That is, the part which adds, subtracts, multiplies and divides. Most processors can also compare values and perform conditional actions as a result of such comparisons. Many processors have instructions which perform various types of conversions between different representations of data. The processor itself is divided into three components that carry out the various functions that the computer is capable of performing. The first component of the processor is the controller. The controller acts as a foreman that oversees the tasks of the processor. The controller looks at the next instruction to be executed and assigns the sub-tasks that must be accomplished to carry out that instruction to the other components of the processor.

Another component of the processor is the Arithmetic-Logic Unit (ALU). This unit is the part of the processor which performs the mathematical computations and logical tasks that we expect a computer to be able to do. Addition, subtraction, comparison, etc., are all carried out by the ALU.

The last component of the processor is a collection of one or more registers. Registers are special named memory cells in the processor where information is temporarily stored during various stages of a computation. The currently executing instruction, for example, resides in a register called the Instruction Register. Since modern processors can execute millions of instructions per second it is expected that information would not stay in a register for more than a few millionths of a second. A computer memory system is accessed (or read) by specifying the location (called an address) of the memory cell. The memory system then responds by producing a copy of the contents of that cell. The original value of the cell is not changed by this process. This is sometimes called a non-destructive read.

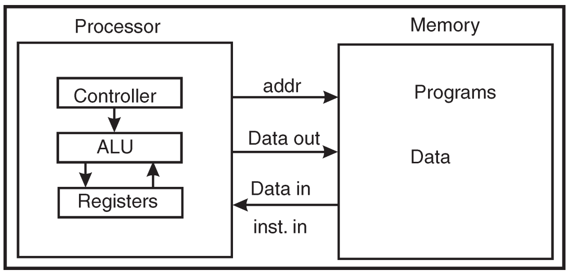
A computer memory system is changed (or written) by specifying the location of the memory cell together with the new value for that cell. The previous value stored in the cell is replaced by the new value.

This is sometimes called a destructive write. The values stored in a computer memory are simply numbers. Numbers are used to represent both data and instructions. In fact, one cannot distinguish instructions from data when examining the contents of a memory system.

It is up to the programmer or operating system to keep track of which memory cells hold data and which are programme instructions. Programs have been written which manipulate and produce programs. Such programs treat instructions as data.

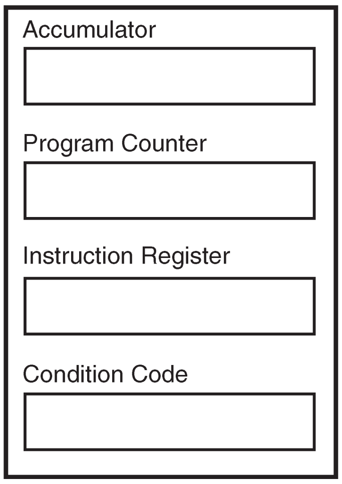
The processor and memory unit are wired together wiring connections called buses. A *bus* is a low resistance connection consisting of 1 or more wires. There are three such connections. The first, called the address bus, is used by the processor to tell the memory system the number or location of the memory cell the processor wishes to access. The second bus is used to send data out to the memory. The third bus is used to transmit data and instructions to the processor.

A typical computer might be organized as indicated in the following diagram:



The processor contains a few memory cells, called registers, which are used for efficient temporary storage:

Processor Registers



The Contents of a memory cell or a register is simply a number. For purposes of illustration, suppose that each memory cell is large enough to hold numbers up to 4 decimal digits in size. If a memory cell holds an an instruction, use the first two decimal digits represent the operation and the remaining digits to represent the adderess or location of the instruction operand.

Instruction Format

Operation	Address

Using this scheme, we could set up the following operation codes:

Operation	Code	Function
add	01	c(acc)=c(acc)+c(addr)
sub	02	c(acc)=c(acc)−c(addr)
load	03	c(acc)=c(addr)
store	04	c(addr)=c(acc)

These codes do not correspond to any known real computer, but rather, they are the operation codes for a hypothetical model computer which we will use to illustrate important aspects of computer organization.

COMPUTER SYSTEM

In describing computer system, a distinction is often made between computer architecture and computer organization. Computer architecture refers to those attributes of a system visible to a programmer, or put another way, those attributes that have a direct impact on the logical execution of a program. Computer organization refers to the operational units and their interconnection that realise the architecture specification.

Examples of architecture attributes include the instruction set, the number of bit to represent various data types (*e.g.*., numbers, and characters), I/O mechanisms, and technique for addressing memory. Examples of organization attributes include those hardware details transparent to the programmer, such as control signals, interfaces between the computer and peripherals, and the memory technology used. As an example, it is an architectural design issue whether a computer will have a multiply instruction. It is an organizational issue whether that instruction will be implemented by a special multiply unit or by a mechanism that makes repeated use of the add unit of the system.

The organization decision may be bases on the anticipated frequency of use of the multiply instruction, the relative speed of the two approaches, and the cost and physical size of a special multiply unit.

Historically, and still today, the distinction between architecture and organization has been an important one. Many computer manufacturers offer a family of computer model, all with the same architecture but with differences in organization. Consequently, the different models in the family have different price and performance characteristics. Furthermore, an architecture may survive many years, but its organization changes with changing technology.

DEVELOPMENT OF COMPUTERS

First Generation: Vacuum Tubes

ENIAC: The ENIAC (Electronic Numerical Integrator And Computer), designed by and constructed under the supervision of Jonh Mauchly and John Presper Eckert at the University of Pennsylvania, was the world's first general-purpose electronic digital computer.

The project was a response to U.S. wartime needs. Mauchly, a professor of electrical engineering at the University of Pennsylvania and Eckert, one of his graduate students, proposed to build a general-purpose computer using vacuum tubes. In 1943, this proposal was accepted by the Army, and work began on the ENIAC.

The resulting machine was enormous, weighting 30 tons, occupying 15,000 square feet of floor space, and containing more than 18,000 vacuum tubes. When operating, it consumed 140 kilowatts of power.

It was aloes substantially faster than any electronic-mechanical computer, being capable of 5000 additions per second. The ENIAC was decimal rather than a binary machine. That is, numbers were represented in decimal form and arithmetic was performed in the decimal system. Its memory consisted of 20 "accumulators", each capable of holding a 10-digit decimal number.

Each digit was represented by a ring of 10 vacuum tubes. At any time, only one vacuum tube was in the ON state, representing one of the 10 digits.

The major drawback of the ENIAC was that it had to be programmed manually by setting switches and plugging and unplugging cables.

The ENIAC was completed in 1946, too late to be used in the war effort. Instead, its first task was to perform a series of complex calculations that were used to help determine the feasibility of the H-bomb. The ENIAC continued to be used until 1955.

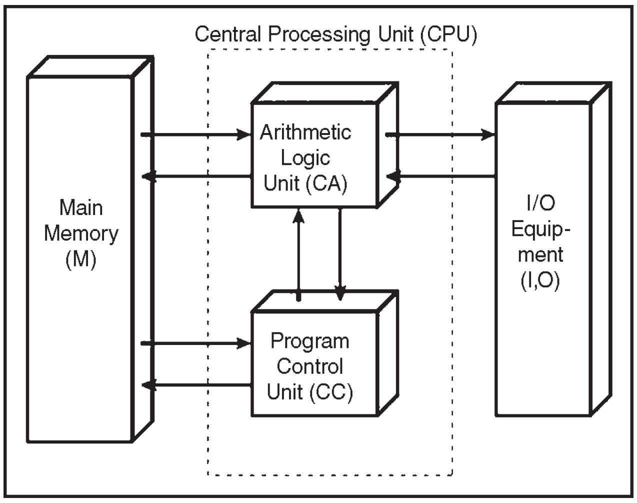
Von Neumann Machine

The programming process could be facilitated if the program could be represented in a form suitable for storing in memory alongside the data. Then, a computer could get its instructions by reading them from memory, and a program could be set of altered by setting the values of a portion of memory. This idea, known as the Stored-program concept, is usually attributed to the ENIAC designers, most notably the mathematician John von Neumann, who was a consultant on the ENIAC project.

The idea was also developed at about the same time by Turing. The first publication of the idea was in a 1945 proposal by von Neumann for a new computer, the EDVAC (Electronic Discrete Variable Computer).

In 1946, von Neumann and his colleagues began the design of a new stored-program computer, referred to as the IAS computer, at the Princeton Institute for Advanced Studies.

The IAS computer, although not completed until 1952, is the prototype of all subsequent general-purpose computers. Figure shows the general structure of the IAS computer.


Central Processing Unit (CPU)
Main Memory (M)
Arithmetic Logic Unit (CA)
Program Control Unit (CC)
I/O Equip-ment (I,O)

It consists of:

- A main memory, which stores both data and instructions.
- An arithmetic-logical unit (ALU) capable of operating on binary data.

- A control unit, which interprets the instructions in memory and causes them to be executed.
- Input and output (I/O) equipment operated by the control unit.

Commercial Computers

The 1950s saw the birth of the computer industry with two companies, Sperry and IBM, dominating the marketplace. In 1947, Eckert and Mauchly formed the Eckert-Maunchly computer Corporation to manufacture computers commercially. Their first successful machine was the UNIVAC I (Universal Automatic Computer), which was commissioned by the Bureau of the Census for the 1950 calculations. The Eckert-Maunchly Computer Corporation became part of the UNIVAC division of Sperry-Rand Corporation, which went on to build a series of successor machines.

The UNIVAC II, which had greater memory capacity and higher performance than the UNIVAC I, was delivered in the late 1950s and illustrates several trends that have remained characteristic of the computer industry. First, advances in technology allow companies to continue to build larger, more powerful computers. Second, each company tries to make its new machines upward compatible with the older machines. This means that the programs written for the older machines can be executed on the new machine. This strategy is adopted in the hopes of retaining the customer base; that is, when a customer decides to buy a newer machine, he is likely to get it from the same company to avoid losing the investment in programs.

The UNIVAC division also began development of the 1100 series of computers, which was to be its bread and butler.

This series illustrates a distinction that existed at one time. The first model, the UNIVAC 1103, and its successors for many years were primarily intended for scientific applications, involving long and complex calculations. Other companies concentrated on business applications, which involved processing large amounts of text data. This split has largely disappeared but it was evident for a number of years.

IBM, which was then the major manufacturer of punched-card processing equipment, delivered its first electronic stored-program computer, the 701, in 1953. The 70l was intended primarily for scientific applications. In 1955, IBM introduced the companion 702 product, which had a number of hardware features that suited it to business applications. These were the first of a long series of 700/7000 computers that established IBM as the overwhelmingly dominant com puter manufacturer.

Second Generation: Transistors

The first major change in the electronic computer came with the replacement of the vacuum tube by the transistor. The transistor is smaller, cheaper, and dissipates less heal than a vacuum tube but can be used in the same way as a vacuum tube to con struct computers.

Unlike the vacuum tube, which requires wires, metal plates, a glass capsule, and a vacuum, the transistor is a solid-state device, made from silicon. The transistor was invented at Bell Labs in 1947 and by the 1950s had launched an electronic revolution. It was not until the late 1950s, however, that fully transisto rized computers were commercially available. IBM again was not the first company to deliver the new technology. NCR and. more successfully.

RCA were the front-run ners with some small transistor machines. IBM followed shortly with the 7000 series.

The use of the transistor defines the second generation of computers. It has become widely accepted to classify computers into generations based on the fundamental hard ware technology employed. Each new generation is characterized by greater processing performance, larger memory capacity, and smaller size than the previous one.

Third Generation: Integrated Circuits

A single, self-contained transistor is called a discrete component. Throughout the 1950s and early 1960s, electronic equipment was composed largely of discrete com ponents—transistors, resistors, capacitors, and so on.

Discrete components were manufactured separately, packaged in their own containers, and soldered or wired together onto circuit boards, which were then installed in computers, oscilloscopes, and other electronic equipment. Whenever an electronic device called for a transistor, a little lube of metal containing a pinhead-sized piece of silicon had to be soldered to a circuit hoard. The entire manufacturing process, from transistor to circuit board, was expensive and cumbersome.

These facts of life were beginning to create problems in the computer industry. Early second-generation computers contained about 10,000 transistors.

This figure grew to the hundreds of thousands, making the manufacture of newer, more power ful machines increasingly difficult. In 1958 came the achievement that revolutionized electronics and started the era of microelectronics: the invention of the integrated circuit. It is the integrated circuit that defines the third generation of computers. Perhaps the two most important members of the third generation are the IBM System/360 and the DEC PDP-8.

Later Generations

Beyond the third generation there is less general agreement on defining generations of computers. There have been a fourth and a fifth genera tion, based on advances in integrated circuit technology. With the introduction of large-scale integration (LSI), more than 1000,000 components can be placed on a single integrated circuit chip. Very-large-scale integration (VLSI) achieved more than 1000,000,000 components per chip, and current VLSI chips can contain more than 1000.000 components.

COMPUTERS CLASSIFICATION AND DATA PROCESSING

Computers are classified according to their data processing speed, amount of data that they can hold and price. Generally, a computer with high processing speed and large internal storage is called a big computer. Due to rapidly improving technology, we are always confused among the categories of computers.

Depending upon their speed and memory size, computers are classified into following four main groups:

- Supercomputer.
- Mainframe computer.
- Mini computer.
- Microcomputer.

Supercomputer

Supercomputer is the most powerful and fastest, and also very expensive. It was developed in 1980s. It is used to process large amount of data and to solve the complicated scientific problems. It can perform more than one trillions calculations per second. It has large number of processors connected parallel.

So parallel processing is done in this computer. In a single supercomputer thousands of users can be connected at the same time and the supercomputer handles the work of each user separately.

Supercomputer are mainly used for:

- Weather forecasting.
- Nuclear energy research.
- Aircraft design.
- Automotive design.
- Online banking.
- To control industrial units.

The supercomputers are used in large organizations, research laboratories, aerospace centers, large industrial units etc. Nuclear scientists use supercomputers to create and analyse models of nuclear fission and fusions, predicting the actions and reactions of millions of atoms as they interact. The examples of supercomputers are CRAY-1, CRAY-2, Control Data CYBER 205 and ETA A-10 etc.

Mainframe Computers

Mainframe computers are also large-scale computers but supercomputers are larger than mainframe. These are also very expensive. The mainframe computer specially requires a very large clean room with air-conditioner.

This makes it very expensive to buy and operate. It can support a large number of various equipments. It also has multiple processors. Large mainframe systems can handle the input and output requirements of several thousand of users. For example, IBM, S/390 mainframe can support 50,000 users simultaneously. The users often access then mainframe with terminals or personal computers. Tere are basically two types of terminals used with mainframe systems.

Dumb Terminal

Dumb terminal does not have its own CPU and storage devices. This type of terminal uses the CPU and storage devices of mainframe system. Typically, a dumb terminal consists of monitor and a keyboard (or mouse).

Intelligent Terminal

Intelligent terminal has its own processor and can perform some processing operations. Usually, this type of terminal does not have its own storage. Typically, personal computrers are used as intelligent terminals. A personal computer as an intelligent terminal gives facility to access data and other services from mainframe system. It also enables to store and process data locally. The mainframe computers are specially used as servers on the World Wide Web. The mainframe computers are used in large organizations such as Banks, Airlines and Universities etc. where many people (users) need frequent access to the same data, which is usually organized into one or more huge databases. IBM is the major manufacturer of mainframe computers. The examples of mainframes are IBM S/390, Control Data CYBER 176 and Amdahl 580 etc.

Minicomputers

These are smaller in size, have lower processing speed and also have lower cost than mainframe. These computers are known as minicomputers because of their small size as compared to other computers at that time. The capabilities of a minicomputer are between mainframe and personal computer. These computers are also known as midrange computers.

The minicomputers are used in business, education and many other government departments. Although some minicomputers are designed for a single user but most are designed to handle multiple terminals. Minicomputers are commonly used as servers in network environment and hundreds of personal computers can be connected to the network with a minicomputer acting as server like mainframes, minicomputers are used as web servers. Single user minicomputers are used for sophisticated design tasks.

The first minicomputer was introduced in the mid-1960s by Digital Equipment Corporation (DEC). After this IBM Corporation (AS/400 computers) Data General Corporation and Prime Computer also designed the mini computers.

Microcomputer

The microcomputers are also known as personal computers or simply PCs. Microprocessor is used in this type of computer. These are very small in size and cost. The IBM's first microcomputer was designed in 1981 and was named as IBM-PC. After this many computer hardware companies copied the design of IBM-PC. The term “PC-compatible” refers any personal computer based on the original IBM personal computer design. The most popular types of personal computers are the PC and the Apple. PC and PC-compatible computers have processors with different architectures than processors in Apple computers. These two types of computers also use different operating systems. PC and PC-compatible computers use the Windows operating system while Apple computers use the Macintosh operating system (MacOS). The majority of microcomputers sold today are part of IBM-compatible. However the Apple computer is neither an IBM nor a compatible. It is another family of computers made by Apple computer. Personal computers are available in two models.

These are:

- Desktop PCs
- Tower PCs

A desktop personal computer is most popular model of personal computer. The system unit of the desktop personal computer can lie flat on the desk or table. In desktop personal computer, the monitor is usually placed on the system unit. Another model of the personal computer is known as tower personal computer. The system unit of the tower PC is vertically placed on the desk of table. Usually the system unit of the tower model is placed on the floor to make desk space free and user can place other devices such as printer, scanner etc. on the desktop. Today computer tables are available which are specially designed for this purpose. The tower models are mostly used at homes and offices.

Microcomputer are further divided into following categories:

- Laptop computer
- Workstation
- Network computer
- Handheld computer

Laptop Computer

Laptop computer is also known as notebook computer. It is small size (85-by-11 inch notebook computer and can fit inside a briefcase. The laptop computer is operated on a special battery and it does not have to be plugged in like desktop computer. The laptop computer is portable and fully functional microcomputer.

It is mostly used during journey. It can be used on your lap in an airplane. It is because it is referred to as laptop computer. The memory and storage capacity of laptop computer is almost equivalent to the PC or desktop computer. It also has the hard dist, floppy disk drive, Zip disk drive, CD-ROM drive, CD-writer etc. it has built-in keyboard and built-in trackball as pointing device.

Laptop computer is also available with the same processing speed as the most powerful personal computer. It means that laptop computer has same features as personal computer. Laptop computers are more expensive than desktop computers. Normally these computers are frequently used in business travellers.

Workstations

Workstations are special single user computers having the same features as personal computer but have the processing speed equivalent to minicomputer or mainframe computer. A workstation computer can be fitted on a desktop. Scientists, engineers, architects and graphic designers mostly use these computers.

Workstation computers are expensive and powerful computers. These have advanced processors, more RAM and storage capacity than personal computers. These are usually used as single-user applications but these are used as servers on computer network and web servers as well.

Network Computers

Network computers are also version of personal computers having less processing power, memory and storage. These are specially designed as terminals for network environment. Some types of network computers have no storage. The network computers are designed for network, Internet or Intranet for data entry or to access data on the network. The network computers depend upon the network's server for data storage and to use software. These computers also use the network's server to perform some processing tasks. In the mid-1990s the concept of network computers became popular among some PC manufacturers. As a result several variations of the network computers quickly became available.

In business, variations of the network computer are Windows terminals, NetPCs and diskless workstations. Some network computers are designed to access only the Internet or to an Intranet. These devices are sometimes called Internet PCs, Internet boxes etc. In home some network computers do not include monitor. These are connected to home television, which serves as the output devices. A popular example of a home-based network computer is Web TV, which enables the user to connect a television to the Internet.

The Web TV has a special set-top box used to connect to the Internet and also provides a set of simple controls which enable the user to navigate the Internet, send and receive e-mails and to perform other tasks on the network while watching television. Network computers are cheaper to purchase and to maintain than personal computers.

Handheld Computer

In the mid 1990s, many new types of small personal computing devices have been introduced and these are referred to as handheld computers. These computers are also referred to as Palmtop Computers. The handheld computers sometimes called Mini-Notebook Computers. The type of computer is named as handheld computer because it can fit in one hand while you can operate it with the other hand.

Because of its reduced size, the screen of handheld computer is quite small. Similarly it also has small keyboard. The handheld computers are preferred by business traveller. Some handheld computers have a specialized keyboard. These computers are used by mobile employees, such as meter readers and parcel delivery people, whose jobs require them to move from place to place.

The examples of handheld computers are:

- Personal digital assistance
- Cellular telephones
- H/PC pro devices

Personal Digital Assistance (PDAs)

The PDA is one of the more popular lightweight mobile devices in use today. A PDA provides special functions such as taking notes, organizing telephone numbers and addresses. Most PDAs also offer a variety of other application software such as word processing, spreadsheet and games etc. Some PDAs include electronic books that enable users to read a book on the PDA's screen. Many PDAs are web-based and users can send/receive e-mails and access the Internet. Similarly, some PDAs also provide telephone capabilities.

The primary input device of a PDA is the stylus. A stylus is an electronic pen and looks like a small ballpoint pen. This input device is used to write notes and store in the PDA by touching the screen. Some PDAs also support voice input.

Cellular Phones

A cellular phone is a web-based telephone having features of analog and digital devices. It is also referred to as Smart Phone. In addition to basic phone capabilities, a cellular phone also provides the functions to receive and send e-mails & faxes and to access the Internet.

H/PC Pro Devices

H/PC Pro dive is new development in handheld technology. These systems are larger than PDAs but they are not quite as large as typical notebook PCs. These devices have features between PDAs and notebook PCs. The H/PC Pro device includes a full-size keyboard but it does not include disk. These systems also have RAM with very low storage capacity and slow speed of processor.

2

Structure of Digital Systems

Engineers use many methods to minimize logic functions, in order to reduce the circuit's complexity. When the complexity is less, the circuit also has fewer errors and less electronics, and is therefore less expensive. The most widely used simplification is a minimization algorithm like the Espresso heuristic logic minimizer within a CAD system, although historically, binary decision diagrams, an automatedQuine–McCluskey algorithm, truth tables, Karnaugh maps, and Boolean algebra have been used.

Representations are crucial to an engineer's design of digital circuits. Some analysis methods only work with particular representations. The classical way to represent a digital circuit is with an equivalent set of logic gates. Another way, often with the least electronics, is to construct an equivalent system of electronic switches (usually transistors). One of the easiest ways is to simply have a memory containing a truth table. The inputs are fed into the address of the memory, and the data outputs of the memory become the outputs.

For automated analysis, these representations have digital file formats that can be processed by computer programmes. Most digital engineers are very careful to select computer programmes ("tools") with compatible file formats. To choose representations, engineers consider types of digital systems. Most digital systems divide into "combinational systems" and "sequential systems." A combinational system always presents the same output when given the same inputs. It is basically a representation of a set of logic functions. A sequential system is a combinational system with some of the outputs fed back as inputs. This makes the digital machine perform a "sequence" of operations. The simplest sequential system is probably a flip flop, a mechanism that represents a binary digit or "bit". Sequential systems are often designed as state machines. In this way, engineers can design a system's gross behaviour, and even test it in a simulation, without considering all the details of the logic functions. Sequential systems divide into two further subcategories. "Synchronous" sequential systems change state all at once, when a "clock" signal changes state. "Asynchronous" sequential systemspropagate changes whenever inputs change. Synchronous sequential systems are made of well-characterised asynchronous circuits such as flip-flops, that change only when the clock changes, and which have carefully designed timing margins.

The usual way to implement a synchronous sequential state machine is to divide it into a piece of combinational logic and a set of flip flops called a “state register.” Each time a clock signal ticks, the state register captures the feedback generated from the previous state of the combinational logic, and feeds it back as an unchanging input to the combinational part of the state machine. The fastest rate of the clock is set by the most time-consuming logic calculation in the combinational logic. The state register is just a representation of a binary number. If the states in the state machine are numbered (easy to arrange), the logic function is some combinational logic that produces the number of the next state.

In comparison, asynchronous systems are very hard to design because all possible states, in all possible timings must be considered. The usual method is to construct a table of the minimum and maximum time that each such state can exist, and then adjust the circuit to minimize the number of such states, and force the circuit to periodically wait for all of its parts to enter a compatible state (this is called “self-resynchronisation”). Without such careful design, it is easy to accidentally produce asynchronous logic that is “unstable”, that is, real electronics will have unpredictable results because of the cumulative delays caused by small variations in the values of the electronic components. Certain circuits (such as the synchroniser flip-flops, switchdebouncers, arbiters, and the like which allow external unsynchronised signals to enter synchronous logic circuits) are inherently asynchronous in their design and must be analysed as such.

As of 2005, almost all digital machines are synchronous designs because it is much easier to create and verify a synchronous design—the software currently used to simulate digital machines does not yet handle asynchronous designs. However, asynchronous logic is thought to be superior, if it can be made to work, because its speed is not constrained by an arbitrary clock; instead, it runs at the maximum speed of its logic gates. Building an asynchronous circuit using faster parts makes the circuit faster. Many digital systems are data flow machines. These are usually designed using synchronous register transfer logic, using hardware description languages such as VHDL or Verilog. In register transfer logic, binary numbers are stored in groups of flip flops called registers. The outputs of each register are a bundle of wires called a "bus" that carries that number to other calculations. A calculation is simply a piece of combinational logic. Each calculation also has an output bus, and these may be connected to the inputs of several registers. Sometimes a register will have a multiplexer on its input, so that it can store a number from any one of several buses. Alternatively, the outputs of several items may be connected to a bus through buffers that can turn off the output of all of the devices except one. A sequential state machine controls when each register accepts new data from its input. In the 1980s, some researchers discovered that almost all synchronous register-transfer machines could be converted to asynchronous designs by using first-in-first-out synchronisation logic. In this scheme, the digital machine is characterised as a set of data flows. In each step of the flow, an asynchronous "synchronisation circuit" determines when the outputs of that step are valid, and presents a signal that says, "grab the data" to the stages that use that stage's inputs. It turns out that just a few relatively simple synchronisation circuits are needed. The most general-purpose register-transfer logic machine is a computer. This is basically an automatic binary abacus. The control unit of a computer is usually designed as a microprogram run by a microsequencer. A microprogram is much like a player-piano roll. Each table entry or "word" of the microprogram commands the state of every bit that controls the computer. The sequencer then counts, and the count addresses the memory or combinational logic machine that contains the microprogram. The bits from the microprogram control the arithmetic logic unit, memory and other parts of the computer, including the microsequencer itself. In this way, the complex task of designing the controls of a computer is reduced to a simpler task of programming a collection of much simpler logic machines.

Computer architecture is a specialised engineering activity that tries to arrange the registers, calculation logic, buses and other parts of the computer in the best way for some purpose. Computer architects have applied large amounts of ingenuity to computer design to reduce the cost and increase the speed and immunity to programming errors of computers. An increasingly common goal is to reduce the power used in a battery-powered computer system, such as a cell-phone. Many computer architects serve an extended apprenticeship as microprogrammers. "Specialised computers" are usually a conventional computer with a special-purpose microprogram.

Automated design tools

To save costly engineering effort, much of the effort of designing large logic machines has been automated. The computer programmes are called “electronic design automation tools” or just “EDA.” Simple truth table-style descriptions of logic are often optimised with EDA that automatically produces reduced systems of logic gates or smaller lookup tables that still produce the desired outputs. The most common example of this kind of software is the Espresso heuristic logic minimizer. Most practical algorithms for optimising large logic systems use algebraic manipulations or binary decision diagrams, and there are promising experiments with genetic algorithms and annealing optimisations.

To automate costly engineering processes, some EDA can take state tables that describe state machines and automatically produce a truth table or a function table for the combinational logic of a state machine. The state table is a piece of text that lists each state, together with the conditions controlling the transitions between them and the belonging output signals. It is common for the function tables of such computer-generated state-machines to be optimised with logic-minimization software such as Minilog. Often, real logic systems are designed as a series of sub-projects, which are combined using a “tool flow.”

The tool flow is usually a “script,” a simplified computer language that can invoke the software design tools in the right order. Tool flows for large logic systems such as microprocessors can be thousands of commands long, and combine the work of hundreds of engineers. Writing and debugging tool flows is an established engineering specialty in companies that produce digital designs. The tool flow usually terminates in a detailed computer file or set of files that describe how to physically construct the logic. Often it consists of instructions to draw the transistors and wires on an integrated circuit or a printed circuit board.

Parts of tool flows are "debugged" by verifying the outputs of simulated logic against expected inputs. The test tools take computer files with sets of inputs and outputs, and highlight discrepancies between the simulated behaviour and the expected behaviour. Once the input data is believed correct, the design itself must still be verified for correctness. Some tool flows verify designs by first producing a design, and then scanning the design to produce compatible input data for the tool flow. If the scanned data matches the input data, then the tool flow has probably not introduced errors. The functional verification data are usually called "test vectors." The functional test vectors may be preserved and used in the factory to test that newly constructed logic works correctly.

However, functional test patterns don't discover common fabrication faults. Production tests are often designed by software tools called "test pattern generators". These generate test vectors by examining the structure of the logic and systematically generating tests for particular faults. This way the fault coverage can closely approach 100 per cent, provided the design is properly made testable. Once a design exists, and is verified and testable, it often needs to be processed to be manufacturable as well. Modern integrated circuits have features smaller than the wavelength of the light used to expose the photoresist. Manufacturability software adds interference patterns to the exposure masks to eliminate open-circuits, and enhance the masks' contrast.

Design for testability

There are several reasons for testing a logic circuit. When the circuit is first developed, it is necessary to verify that the design circuit meets the required functional and timing specifications. When multiple copies of a correctly designed circuit are being manufactured, it is essential to test each copy to ensure that the manufacturing process has not introduced any flaws. A large logic machine (say, with more than a hundred logical variables) can have an astronomical number of possible states. Obviously, in the factory, testing every state is impractical if testing each state takes a microsecond, and there are more states than the number of microseconds since the universe began. Unfortunately, this ridiculous-sounding case is typical. Fortunately, large logic machines are almost always designed as assemblies of smaller logic machines. To save time, the smaller sub-machines are isolated by permanently-installed "design for test" circuitry, and are tested independently. One common test scheme known as "scan design" moves test bits serially (one after another) from external test equipment through one or more serial shift registers known as "scan chains". Serial scans have only one or two wires to carry the data, and minimize the physical size and expense of the infrequently-used test logic. After all the test data bits are in place, the design is reconfigured to be in "normal mode" and one or more clock pulses are applied, to test for faults (*e.g.*, stuck-at low or stuck-at high) and capture the test result into flip-flops and/or latches in the scan shift register(s). Finally, the result of the test is shifted out to the block boundary and compared against the predicted "good machine" result. In a board-test environment, serial to parallel testing has been formalised with a standard called "JTAG" (named after the "Joint Test Action Group" that proposed it). Another common testing scheme provides a test mode that forces some part of the logic machine to enter a "test cycle." The test cycle usually exercises large independent parts of the machine.

Trade-offs

Several numbers determine the practicality of a system of digital logic: cost, reliability, fanout and speed. Engineers explored numerous electronic devices to get an ideal combination of these traits.

Cost

The cost of a logic gate is crucial. In the 1930s, the earliest digital logic systems were constructed from telephone relays because these were inexpensive and relatively reliable. After that, engineers always used the cheapest available electronic switches that could still fulfill the requirements. The earliest integrated circuits were a happy accident. They were constructed not to save money, but to save weight, and permit the Apollo Guidance Computer to control an inertial guidance system for a spacecraft. The first integrated circuit logic gates cost nearly $50 (in 1960 dollars, when an engineer earned $10,000/year). To everyone's surprise, by the time the circuits were mass-produced, they had become the least-expensive method of constructing digital logic. Improvements in this technology have driven all subsequent improvements in cost.

With the rise of integrated circuits, reducing the absolute number of chips used represented another way to save costs.

The goal of a designer is not just to make the simplest circuit, but to keep the component count down. Sometimes this results in slightly more complicated designs with respect to the underlying digital logic but nevertheless reduces the number of components, board size, and even power consumption. For example, in some logic families, NAND gates are the simplest digital gate to build. All other logical operations can be implemented by NAND gates. If a circuit already required a single NAND gate, and a single chip normally carried four NAND gates, then the remaining gates could be used to implement other logical operations like logical and. This could eliminate the need for a separate chip containing those different types of gates.

Reliability

The "reliability" of a logic gate describes its mean time between failure (MTBF). Digital machines often have millions of logic gates. Also, most digital machines are "optimised" to reduce their cost. The result is that often, the failure of a single logic gate will cause a digital machine to stop working. Digital machines first became useful when the MTBF for a switch got above a few hundred hours. Even so, many of these machines had complex, well-rehearsed repair procedures, and would be non-functional for hours because a tube burned-out, or a moth got stuck in a relay. Modern transistorised integrated circuit logic gates have MTBFs greater than 82 billion hours (8.2×10^{10}) hours, and need them because they have so many logic gates.

Fanout

Fanout describes how many logic inputs can be controlled by a single logic output without exceeding the current ratings of the gate.[6] The minimum practical fanout is about five. Modern electronic logic using CMOS transistors for switches have fanouts near fifty, and can sometimes go much higher.

Speed

The “switching speed” describes how many times per second an inverter (an electronic representation of a “logical not” function) can change from true to false and back. Faster logic can accomplish more operations in less time. Digital logic first became useful when switching speeds got above fifty hertz, because that was faster than a team of humans operating mechanical calculators. Modern electronic digital logic routinely switches at five gigahertz (5×10^{9} hertz), and some laboratory systems switch at more than a terahertz (1×10^{12} hertz).

Logic Families

Design started with relays. Relay logic was relatively inexpensive and reliable, but slow. Occasionally a mechanical failure would occur. Fanouts were typically about ten, limited by the resistance of the coils and arcing on the contacts from high voltages. Later, vacuum tubes were used. These were very fast, but generated heat, and were unreliable because the filaments would burn out. Fanouts were typically five to seven, limited by the heating from the tubes’ current. In the 1950s, special “computer tubes” were developed with filaments that omitted volatile elements like silicon. These ran for hundreds of thousands of hours. The first semiconductor logic family was resistor-transistor logic. This was a thousand times more reliable than tubes, ran cooler, and used less power, but had a very low fan-in of three. Diode-transistor logic improved the fanout up to about seven, and reduced the power. Some DTL designs used two power-supplies with alternating layers of NPN and PNP transistors to increase the fanout. Transistor transistor logic (TTL) was a great improvement over these. In early devices, fanout improved to ten, and later variations reliably achieved twenty. TTL was also fast, with some variations achieving switching times as low as twenty nanoseconds. TTL is still used in some designs. Emitter coupled logic is very fast but uses a lot of power. It was extensively used for high-performance computers made up of many medium-scale components (such as the Illiac IV). By far, the most common digital integrated circuits built today use CMOS logic, which is fast, offers high circuit density and low-power per gate. This is used even in large, fast computers, such as the IBM System z.

Digital Integrated Circuits

Digital integrated circuits, sometimes called an IC, chip or microchip, are the building blocks for nearly every type of modern electronics and electrical devices, whether for home or office, work or pleasure.

Origin

Digital integrated circuit development was necessary to resolve the fundamental issues of vacuum tubes, the leading technology of the first half of the 20th century. Vacuum tubes are bulky, fragile, power hungry, hot and failed frequently.

History

On Sept. 12, 1958, Jack Kilby successfully demonstrated the first IC with a single transistor and all its components on a single piece of germanium semiconductor material. In the first 50 years, ICs have advanced from single transistors to complex microprocessors with multiple CPU cores. On Dec. 3, 2009, the Helsinki University of Technology announced that a team of researchers had succeeded in building the first single atom transistor, the theoretical minimum component size. Technological alternatives such as quantum computing evolved as semiconductor technology approached theoretical limits.

Features

Digital integrated circuits operate based on binary mathematics using electrical signals to represent “ones” and “zeros.” Digital integrated circuits can perform any number of operations, from a simple switch to a complex microprocessor.

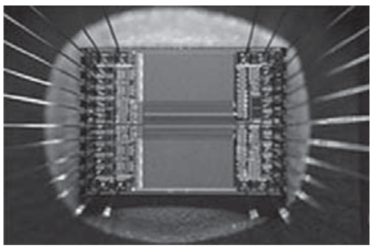
History of Integrated Circuits

The experiments performed on semi conductors revealed that they can perform the variety of vacuum tube functions. The incorporation of bulk of puny transistors on a small chip was a major leap in manually assembling electronic components on a really small circuit. Mass production and efficiency of Integrated Circuits greatly replaced the discrete transistors from many devices.

The two main features of integrated circuits like low cost and high performance gave it edge over the discrete transistors. The low cost is associated to the bulk production and printing using photolithography. The material used in constructing an IC is less than that used in discrete circuits.

The high performance can be contributed to the quick switching and low power consumption.

The idea of IC was developed by the scientists working for Royal Radar. The idea propagated further when Jack Kilby and Robert Noyce working separately brought a remarkable invention. The chips made by both of them were meant to perform same functions whereas the material used was not the same. Noyce made a silicone chip and the Kilby manufactured a germanium chip. The development in this field was not new it was in 1949 when the first idea related to integrated chip came to view. From then onwards the development and growth is continued.



Applications of Integrated Circuts

Integrated circuits today have become ubiquitous. It seems that it was this invention which has led to the growth in technology. This remarkable invention has helped companies and manufacturers of electronic devices to design and introduce new and advance products. It can be said that it was due to the micro chips that we can use portable digital devices today.

The integrated chip is an important component of laptops, palm tops, MP3s, playstaions, mobile phones, net books and an unlimited array of electronic devices. The existence of all modern communication and electronic devices is based today on the use of integrated circuits. This chip is small but really efficient and swift when it comes to the performance. It is due to this desirable feature that it is part of almost all cell phones and we carry them anywhere in the world without being worrying about performance decline.

Taxonomy of Integrated Circuits

The integrated circuits can be divided into three broad categories like digital ICs, analog ICs and mixed signal ICs. Digital integrated circuits contain many flip flops, multiplexers and logic gates in a single chip comprising only few millimeters. This small size has provided its edge over all other technologies. Therefore, the manufactures of digital devices now prefer it over discrete transistors. These small chips consume less power and they are efficient. The manufacturing of these circuits help in reducing manufacturing cost due to the wide scale integration.

The working of the digital microchips is guided by the binary mathematical process in the form of 1 and 0 signals. On the contrary the analog micro circuits work by continuous flow of signals. The various examples of analog circuits include power management circuits, sensors and operational amplifiers.

These integrated circuits are used for performing functions like active filtering, demodulation, amplification and mixing. The expertly designed analog circuits remove the need for manufacturing an entirely new circuit from abrasion. This provides the facility to the mobile phone manufactures to buy and install the IC instead of preparing one. The digital and analog circuits can be combined in on one chip to enhance the performance.

Design issues in digital circuits

Digital circuits are made from analog components. The design must assure that the analog nature of the components doesn't dominate the desired digital behaviour. Digital systems must manage noise and timing margins, parasitic inductances and capacitances, and filter power connections. Bad designs have intermittent problems such as "glitches", vanishingly-fast pulses that may trigger some logic but not others, "runt pulses" that do not reach valid "threshold" voltages, or unexpected ("undecoded") combinations of logic states. Additionally, where clocked digital systems interface to analog systems or systems that are driven from a different clock, the digital system can be subject to metastability where a change to the input violates the set-up time for a digital input latch. This situation will self-resolve, but will take a random time, and while it persists can result in invalid signals being propagated within the digital system for a short time.

Since digital circuits are made from analog components, digital circuits calculate more slowly than low-precision analog circuits that use a similar amount of space and power. However, the digital circuit will calculate more repeatably, because of its high noise immunity. On the other hand, in the high-precision domain (for example, where 14 or more bits of precision are needed), analog circuits require much more power and area than digital equivalents.

Construction

A digital circuit is often constructed from small electronic circuits called logic gates that can be used to create combinational logic. Each logic gate represents a function of boolean logic. A logic gate is an arrangement of electrically controlled switches, better known as transistors. Each logic symbol is represented by a different shape. The actual set of shapes was introduced in 1984 under IEEE\ANSI standard 91-1984. "The logic symbol given under this standard are being increasingly used now and have even started appearing in the literature published by manufacturers of digital integrated circuits." The output of a logic gate is an electrical flow or voltage, that can, in turn, control more logic gates. Logic gates often use the fewest number of transistors in order to reduce their size, power consumption and cost, and increase their reliability. Integrated circuits are the least expensive way to make logic gates in large volumes. Integrated circuits are usually designed by engineers using electronic design automation software.

Another form of digital circuit is constructed from lookup tables, (many sold as "programmable logic devices", though other kinds of PLDs exist). Lookup tables can perform the same functions as machines based on logic gates, but can be easily reprogrammed without changing the wiring. This means that a designer can often repair design errors without changing the arrangement of wires. Therefore, in small volume products, programmable logic devices are often the preferred solution. They are usually designed by engineers using electronic design automation software. When the volumes are medium to large, and the logic can be slow, or involves complex algorithms or sequences, often a small microcontroller is programmed to make an embedded system. These are usually programmed by software engineers.

When only one digital circuit is needed, and its design is totally customised, as for a factory production line controller, the conventional solution is a programmable logic controller, or PLC. These are usually programmed by electricians, using ladder logic.

3

Advantages of Digital Signals

The usual advantages of digital circuits when compared to analog circuits are:

- *Noise Margin (resistance to noise/robustness):* Digital circuits are less affected by noise. If the noise is below a certain level (the noise margin), a digital circuit behaves as if there was no noise at all. The stream of bits can be reconstructed into a perfect replica of the original source. However, if the noise exceeds this level, the digital circuit cannot give correct results.
- *Error Correction and Detection:* Digital signals can be regenerated to achieve lossless data transmission, within certain limits. Analog signal transmission and processing, by contrast, always introduces noise.
- *Easily Programmable:* Digital systems interface well with computers and are easy to control with software. It is often possible to add new features to a digital system

without changing hardware, and to do this remotely, just by uploading new software. Design errors or bugs can be worked-around with a software upgrade, after the product is in customer hands. A digital system is often preferred because of (re-)programmability and ease of upgrading without requiring hardware changes.

- Cheap Electronic Circuits: More digital circuitry can be fabricated per square millimeter of integrated-circuit material. Information storage can be much easier in digital systems than in analog ones. In particular, the great noise-immunity of digital systems makes it possible to store data and retrieve it later without degradation. In an analog system, aging and wear and tear will degrade the information in storage, but in a digital system, as long as the wear and tear is below a certain level, the information can be recovered perfectly. Theoretically, there is no data-loss when copying digital data. This is a great advantage over analog systems, which faithfully reproduce every bit of noise that makes its way into the signal.

Disadvantages

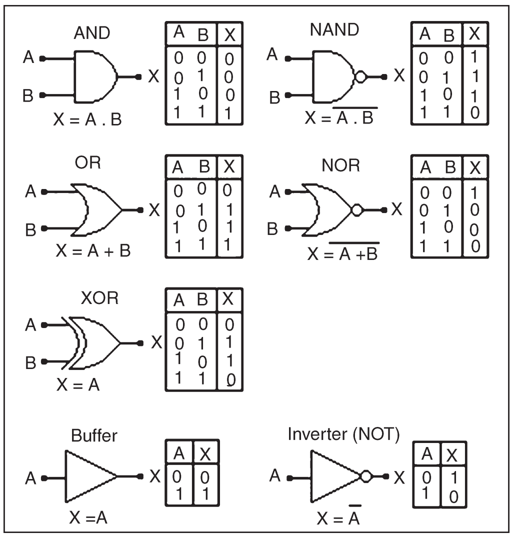
The world in which we live is analog, and signals from this world such as light, temperature, sound, electrical conductivity, electric and magnetic fields, and phenomena such as the flow of time, are for most practical purposes continuous and thus analog quantities rather than discrete digital ones. For a digital system to do useful things in the real world, translation from the continuous realm to the discrete digital realm must occur, resulting in quantisation errors. This problem can usually be mitigated by designing the system to store enough digital data to represent the signal to the desired degree of fidelity. The Nyquist-Shannon sampling theorem provides an important guideline as to how much digital data is needed to accurately portray a given analog signal.

Digital systems can be fragile, in that if a single piece of digital data is lost or misinterpreted, the meaning of large blocks of related data can completely change. This problem can be diminished by designing the digital system for robustness. For example, a parity bit or other error-detecting or error-correcting code can be inserted into the signal path so that minor data corruptions can be detected and possibly corrected. Digital circuits use more energy than analog circuits to accomplish the same calculations and signal processing tasks, thus producing more heat as well. In portable or battery-powered systems this can be a major limiting factor. Digital circuits are made from analog components, and care has to be taken to all noise and timing margins, to parasitic inductances and capacitances, to proper filtering of power and ground connections, to electromagnetic coupling amongst data lines. Inattention to these can cause problems such as “glitches”, pulses do not reach valid switching (threshold) voltages, or unexpected (“undecoded”) combinations of logic states. A corollary of the fact that digital circuits are made from analog components is the fact that digital circuits are slower to perform calculations than analog circuits that occupy a similar amount of physical space and consume the same amount of power. However, the digital circuit will perform the calculation with much better repeatability, due to the high noise immunity of digital circuitry.

Gates, Decoders, Multiplexers

Gates

Logic gates (or simply gates) are the fundamental building blocks of digital circuitry. As their name implies, they function by "opening" or "closing" to admit or reject the flow of digital information. Gates implement electronically simple logical opera-tions on boolean (Bool's algebra) variables, *i.e.*, variables that can have only one of two states (0/1, low/ high, false/true). From an electrical point of view and for the TTL (transistor-transistor-logic) family of digital electronics, any voltage in the range 0-0,7 V and in the range 2,5-5 V, represent logic states 0 and 1, respectively. In the following figure the accepted electronic symbols for different gates are shown, along with their corresponding "truth tables" and their symbolic logical expressions. All variables (X, A, B, ...) are booleans.

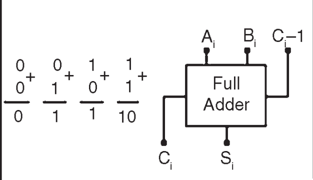

AND
A B X: 0 0 0; 0 1 0; 1 0 0; 1 1 1
X = A . B
NAND
A B X: 0 0 1; 0 1 1; 1 0 1; 1 1 0
$X = \overline{A . B}$
OR
A B X: 0 0 0; 0 1 1; 1 0 1; 1 1 1
X = A + B
NOR
A B X: 0 0 1; 0 1 0; 1 0 0; 1 1 0
$X = \overline{A + B}$
XOR
A B X: 0 0 0; 0 1 1; 1 0 1; 1 1 0
X = A
Buffer
A X: 0 0; 1 1
X =A
Inverter (NOT)
A X: 0 1; 1 0
$X = \overline{A}$

The most typical logical operations are implemented by AND and OR gates. The logical ex-pres-sion for the AND operation is "if *A* is true AND *B* is true then *X* is true", and for the OR operation is "if A is true OR B is true then *X* is true". The inverted logic AND and OR gates are com-mon-ly known as NAND (Not AND) and NOR (Not OR) gates. *A* XOR (Exclusive-OR) gate im-plements the logical expression "if *A* is different than B then *X* is true", hence sometimes this ga-te is called "inequality comparator".

The buffer and the inverter are not gates but their use is closely associated with them. A buffer doesn't change the logic state of its input. It is only occasionally used for increasing the fan-out, *i.e.*, the capability of the output of one gate to drive a number of other gates. The inverter is much more important and it is used for inverting a logic state, *i.e.*, for performing the logical operation of negation (NOT). The logical expressions for a buffer and an inverter are "*X* is *A*" and "*X* is NOT *A*", respectively. AND, OR, NAND and NOR gates can have more than 2 inputs. In this case their truth tables are extended to all inputs combinations and their corresponding expressions as well. For example, the logical expression for a 4-input AND is "if *A* is true AND B is true AND *C* is true AND *D* is true then *X* is true". The corresponding expression for a 3-input NOR gate is "if *A* is true OR *B* is true OR *C* is true then *X* is false"

Decoders

Decoders are circuits with two or more inputs and one or more outputs, resulting by combining various types of gates. Their basic function is to accept a binary word (code) as an input and crea-te a different binary word as an output. A typical decoder is the so-called full adder (3 inputs-2 outputs) implementing the addition of two one-digit numbers (A_i, B_i) taking into consideration the status of any previous carry (C_{i-1}), resulting into the sum (S_i), and generating a new carry (C_i). The addition of two 1-digits numbers and the correspon-ding truth table of full adder are shown below:

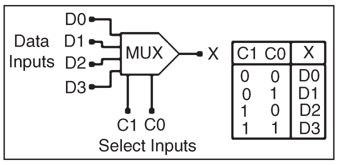


A_i	B_i	C_i-1	S_i	C_i
0	0	0	0	0
0		0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0		1	0	1
1	0	1	0	1
1	1	1	1	1

N full adders can be cascaded to form a unit for the addition of two *N*-digits binary numbers. Decoders with any type of truth table can be constructed by using simple or complicated combi-nations of gates. Implementation of Bool's algebra rules generally simplifies the overall design. Simple and useful decoders are the so-called "2-to-4" and "3-to-8" decoders.

Multiplexers

Generally, multiplexers are circuits behaving like a controlled rotary switch, *i.e.*, any one of a number of inputs may be selected as output. In digital electronics, a multiplexer is a combination of logic gates resulting into circuits with two or more inputs (data inputs) and one output. The selection of the channel to be read into the output is controlled by supplying a specific digital word to a different set of inputs (select inputs). A typical 4 input channels (D3-D0) digital multi-plexer, and its corresponding truth table is shown below:



C1	C0	X
0	0	D0
0	1	D1
1	0	D2
1	1	D3

The active input channel is selected by supplying the appropriate code to select inputs (*C1*, *C0*).

Applet

With this easy to use applet you can train yourself with some simple simulated circuits of digital gates.

You can select one out of five (sets of) circuits by clicking on the corresponding radiobutton:

- Circuit "Gates 1" contains all 2-input gates including a buffer and an inverter.
- Circuit "Gates 2" contains some typical examples of gates with more than 2 inputs.
- Circuit "Full adder" contains a combination of gates implementing the function of a full adder. In the same circuit a 4-bit adder is implemented by cascading 4 full adder circuits.
- Circuit "Decoders" contains a 2-to-4 and a 3-to-8 decoder.
- Circuit "Multiplexer" contains a 4-input line multiplexer.

With all circuits you can change the logic state of any input by clicking on the corresponding small buttons, of which, each one is acting as logic state generator. Observe how the logic state of the out-put(s) and of some "test points" of their internal circuitry is affected by these changes and verify the validity of the truth tables of individual gates. It is of interest to note how a 2-input AND, OR, NAND or NOR gate can control the "flow" of digital data supplied to one of their inputs, by applying different logic states to the other input (control input). Observe also how a XOR gate can act as a buffer or an inverter by applying 0 or 1 to one of its inputs.

Transformers Electrical Energy

A transformer is a device that transfers electrical energy from one circuit to another through inductively coupled conductors — the transformer's coils or "windings". Except for air-core transformers, the conductors are commonly wound around a single iron-rich core, or around separate but magnetically-coupled cores. A varying current in the first or "primary" winding creates a varying magnetic field in the core (or cores) of the transformer. This varying magnetic field induces a varying electromotive force (EMF) or "voltage" in the "secondary" winding. This effect is called mutual induction.

If a load is connected to the secondary, an electric current will flow in the secondary winding and electrical energy will flow from the primary circuit through the transformer to the load. In an ideal transformer, the induced voltage in the secondary winding (V_S) is in proportion to the primary voltage (V_P), and is given by the ratio of the number of turns in the secondary to the number of turns in the prima

$$\frac{V_S}{V_P} = \frac{N_S}{N_P}$$

By appropriate selection of the ratio of turns, a transformer thus allows an alternating current (AC) voltage to be "stepped up" by making N_S greater than N_P, or "stepped down" by making N_S less than N_P. Transformers come in a range of sizes from a thumbnail-sized coupling transformer hidden inside a stage microphone to huge units weighing hundreds of tons used to interconnect portions of national power grids.

All operate with the same basic principles, although the range of designs is wide. While new technologies have eliminated the need for transformers in some electronic circuits, transformers are still found in nearly all electronic devices designed for household ("mains") voltage. Transformers are essential for high voltage power transmission, which makes long distance transmission economically practical.

First Steps: Experiments with Induction Coils

What would become the "transformer principle" was revealed in 1831 by Michael Faraday in his demonstration of electromagnetic induction, but without recognition of its future role in manipulating EMF. The first "induction coils" to see wide use were invented by Rev. Nicholas Callan of Maynooth College, Ireland in 1836, one of the first researchers to realize that the more turns the secondary winding has in relation to the primary winding, the larger the increase in EMF. Induction coils evolved from scientists' and inventors' efforts to get higher voltages from batteries.

Rather than alternating current (AC), their action relied upon a vibrating "make-and-break" mechanism that regularly interrupted the flow of direct current (DC) from the batteries. Between the 1830s and the 1870s, efforts to build better induction coils, mostly by trial and error, slowly revealed the basic principles of transformers. Efficient, practical designs did not appear until the 1880s, but within a decade the "transformer" would be instrumental in the "War of Currents", and in seeing AC distribution systems triumph over their

DC counterparts, a position in which they have remained dominant ever since.

In 1876, Russian engineer Pavel Yablochkov invented a lighting system based on a set of induction coils where the primary windings were connected to a source of alternating current and the secondary windings could be connected to several “electric candles” (arc lamps) of his own design. The coils used in the system behaved as primitive transformers. The patent claimed the system could “provide separate supply to several lighting fixtures with different luminous intensities from a single source of electric power”.

In 1878, the engineers of the Ganz Company in Hungary assigned part of its extensive engineering works to the manufacture of electric lighting apparatus for Austria-Hungary, and by 1883 made over fifty installations. It offered an entire system consisting of both arc and incandescent lamps, generators, and other accessories. Lucien Gaulard and John Dixon Gibbs first exhibited a device with an open iron core called a “secondary generator” in London in 1882, then sold the idea to the Westinghouse company in the United States. They also exhibited the invention in Turin, Italy in 1884, where it was adopted for an electric lighting system.

Induction coils with open magnetic circuits are inefficient for transfer of power to loads. Various methods of adjusting the cores or bypassing magnetic flux around part of a coil were developed, since until about 1880 the paradigm for AC power transmission from a high voltage supply to a low voltage load was a series circuit.

In practice, several coils with a ratio near 1:1 were connected with their primaries in series to allow use of a high voltage for transmission while presenting a low voltage to the lamps. The inherent flaw in this method was that turning off a single lamp affected all the others on the circuit, and many adjustable coil designs were introduced in an effort to accommodate this problematic characteristic of the series circuit.

Between 1884 and 1885, Hungarian engineers Zipernowsky, Bláthy and Déri from the Ganz company in Budapest created the efficient "ZBD" closed-core model, which were based on the design by Gaulard and Gibbs. (Gaulard and Gibbs designed just an open core model). They discovered that all former (coreless or open-core) devices were incapable of regulating voltage, and were therefore impracticable.

Their joint patent described a transformer with no poles and comprised two versions of it, the "closed-core transformer" and the "shell-core transformer. In the closed-core transformer the iron core is a closed ring around which the two coils are arranged uniformly.

In the shell type transformer, the copper induction cables are passed through the core. In both designs, the magnetic flux linking the primary and secondary coils travels (almost entirely) in the iron core, with no intentional path through air.

The core consists of iron cables or plates. Based on this invention, it became possible to provide economical and cheap lighting for industry and households." Zipernowsky, Bláthy and Déri discovered the mathematical formula of transformers: $V_s/V_p = N_s/N_p$. With this formula, transformers became calculable and proportionable.

Their patent application made the first use of the word "transformer", a word that had been coined by Ottó Bláthy. George Westinghouse had bought both Gaulard and Gibbs' and the "ZBD" patents in 1885. He entrusted William Stanley with the building of a ZBD-type transformer for commercial use. Stanley built the core from interlocking E-shaped iron plates. This design was first used commercially in 1886.

Early Developments and Applications

Russian engineer Mikhail Dolivo-Dobrovolsky developed the first three-phase transformer in 1889. In 1891 Nikola Tesla invented the Tesla coil, an air-cored, dual-tuned resonant transformer for generating very high voltages at high frequency. Audio frequency transformers (at the time called repeating coils) were used by the earliest experimenters in the development of the telephone.

Basic Principles

The transformer is based on two principles: firstly, that an electric current can produce a magnetic field (electromagnetism) and secondly that a changing magnetic field within a coil of wire induces a voltage across the ends of the coil (electromagnetic induction). Changing the current in the primary coil changes the magnitude of the applied magnetic field. The changing magnetic flux extends to the secondary coil where a voltage is induced across its ends.

A simplified transformer design is shown to the left. A current passing through the primary coil creates a magnetic field. The primary and secondary coils are wrapped around a core of very high magnetic permeability, such as iron; this ensures that most of the magnetic field lines produced by the primary current are within the iron and pass through the secondary coil as well as the primary coil.

Induction law

The voltage induced across the secondary coil may be calculated from Faraday's law of induction, which states that:

$$V_S = N_S \frac{d\Phi}{dt}$$

where V_S is the instantaneous voltage, N_S is the number of turns in the secondary coil and Ö equals the magnetic flux through one turn of the coil. If the turns of the coil are oriented perpendicular to the magnetic field lines, the flux is the product of the magnetic field strength *B* and the area *A* through which it cuts.

The area is constant, being equal to the cross-sectional area of the transformer core, whereas the magnetic field varies with time according to the excitation of the primary. Since the same magnetic flux passes through both the primary and secondary coils in an ideal transformer, the instantaneous voltage across the primary winding equals

$$V_P = N_P \frac{d\Phi}{dt}$$

Taking the ratio of the two equations for V_S and V_P gives the basic equation for stepping up or stepping down the voltage.

Ideal Power Equation

If the secondary coil is attached to a load that allows current to flow, electrical power is transmitted from the primary circuit to the secondary circuit. Ideally, the transformer is perfectly efficient; all the incoming energy is transformed from the primary circuit to the magnetic field and into the secondary circuit. If this condition is met, the incoming electric power must equal the outgoing power.

$$P_{\text{incoming}} = I_P V_P = P_{\text{outgoing}} = I_S V_S$$

giving the ideal transformer equation

If the voltage is increased (stepped up) ($V_S > V_P$), then the current is decreased (stepped down) ($I_S < I_P$) by the same factor. Transformers are efficient so this formula is a reasonable approximation.

The impedance in one circuit is transformed by the *square* of the turns ratio. For example, if an impedance Z_S is attached across the terminals of the secondary coil, it appears to the primary circuit to have an impedance of. This relationship is reciprocal, so that the impedance Z_P of the primary circuit appears to the secondary to be.

Detailed Operation

The simplified description above neglects several practical factors, in particular the primary current required to establish a magnetic field in the core, and the contribution to the field due to current in the secondary circuit.

Models of an ideal transformer typically assume a core of negligible reluctance with two windings of zero resistance. When a voltage is applied to the primary winding, a small current flows, driving flux around the magnetic circuit of the core. The current required to create the flux is termed the *magnetizing current*; since the ideal core has been assumed to have near-zero reluctance, the magnetizing current is negligible, although still required to create the magnetic field.

The changing magnetic field induces an electromotive force (EMF) across each winding. Since the ideal windings have no impedance, they have no associated voltage drop, and so the voltages V_P and V_S measured at the terminals of the transformer, are equal to the corresponding EMFs. The primary EMF, acting as it does in opposition to the primary voltage, is sometimes termed the “back EMF”. This is due to Lenz’s law which states that the induction of EMF would always be such that it will oppose development of any such change in magnetic field.

Practical Considerations

Leakage Flux

The ideal transformer model assumes that all flux generated by the primary winding links all the turns of every winding, including itself. In practice, some flux traverses paths that take it outside the windings. Such flux is termed *leakage flux*, and results in leakage inductance in series with the mutually coupled transformer windings.

Leakage results in energy being alternately stored in and discharged from the magnetic fields with each cycle of the power supply. It is not directly a power loss, but results in inferior voltage regulation, causing the secondary voltage to fail to be directly proportional to the primary, particularly under heavy load. Transformers are therefore normally designed to have very low leakage inductance. However, in some applications, leakage can be a desirable property, and long magnetic paths, air gaps, or magnetic bypass shunts may be deliberately introduced to a transformer’s design to limit the short-circuit current it will supply.

Leaky transformers may be used to supply loads that exhibit negative resistance, such as electric arcs, mercury vapour lamps, and neon signs; or for safely handling loads that become periodically short-circuited such as electric arc welders. Air gaps are also used to keep a transformer from saturating, especially audio-frequency transformers in circuits that have a direct current flowing through the windings.

Effect of Frequency

The time-derivative term in Faraday's Law shows that the flux in the core is the integral of the applied voltage. Hypothetically an ideal transformer would work with direct-current excitation, with the core flux increasing linearly with time. In practice, the flux would rise to the point where magnetic saturation of the core occurs, causing a huge increase in the magnetizing current and overheating the transformer. All practical transformers must therefore operate with alternating (or pulsed) current.

Transformer Universal EMF Equation

The EMF of a transformer at a given flux density increases with frequency. By operating at higher frequencies, transformers can be physically more compact because a given core is able to transfer more power without reaching saturation, and fewer turns are needed to achieve the same impedance. However properties such as core loss and conductor skin effect also increase with frequency. Aircraft and military equipment employ 400 Hz power supplies which reduce core and winding weight.

Operation of a transformer at its designed voltage but at a higher frequency than intended will lead to reduced magnetizing current; at lower frequency, the magnetizing current will increase. Operation of a transformer at other than its design frequency may require assessment of voltages, losses, and cooling to establish if safe operation is practical.

For example, transformers may need to be equipped with "volts per hertz" over-excitation relays to protect the transformer from overvoltage at higher than rated frequency. Knowledge of natural frequencies of transformer windings is of importance for the determination of the transient response of the windings to impulse and switching surge voltages.

Energy Losses

An ideal transformer would have no energy losses, and would be 100% efficient. In practical transformers energy is dissipated in the windings, core, and surrounding structures. Larger transformers are generally more efficient, and those rated for electricity distribution usually perform better than 98%.

Experimental transformers using superconducting windings achieve efficiencies of 99.85%, While the increase in efficiency is small, when applied to large heavily-loaded transformers the annual savings in energy losses are significant. A small transformer, such as a plug-in "wall-wart" or power adapter type used for low-power consumer electronics, may be no more than 85% efficient, with considerable loss even when not supplying any load. Though individual power loss is small, the aggregate losses from the very large number of such devices are coming under increased scrutiny.

The losses vary with load current, and may be expressed as “no-load” or “full-load” loss. Winding resistance dominates load losses, whereas hysteresis and eddy currents losses contribute to over 99% of the no-load loss. The no-load loss can be significant, meaning that even an idle transformer constitutes a drain on an electrical supply, which encourages development of low-loss transformers.

Transformer losses are divided into losses in the windings, termed copper loss, and those in the magnetic circuit, termed iron loss. Losses in the transformer arise from:

Winding Resistance

Current flowing through the windings causes resistive heating of the conductors. At higher frequencies, skin effect and proximity effect create additional winding resistance and losses.

Hysteresis Losses

Each time the magnetic field is reversed, a small amount of energy is lost due to hysteresis within the core. For a given core material, the loss is proportional to the frequency, and is a function of the peak flux density to which it is subjected.

Eddy Currents

Ferromagnetic materials are also good conductors, and a solid core made from such a material also constitutes a single short-circuited turn throughout its entire length. Eddy currents therefore circulate within the core in a plane normal to the flux, and are responsible for resistive heating of the core material. The eddy current loss is a complex function of the square of supply frequency and inverse square of the material thickness.

Magnetostriction

Magnetic flux in a ferromagnetic material, such as the core, causes it to physically expand and contract slightly with each cycle of the magnetic field, an effect known as magnetostriction. This produces the buzzing sound commonly associated with transformers, and in turn causes losses due to frictional heating in susceptible cores.

Mechanical Losses

In addition to magnetostriction, the alternating magnetic field causes fluctuating electromagnetic forces between the primary and secondary windings. These incite vibrations within nearby metalwork, adding to the buzzing noise, and consuming a small amount of power.

Stray Losses

Leakage inductance is by itself lossless, since energy supplied to its magnetic fields is returned to the supply with the next half-cycle. However, any leakage flux that intercepts nearby conductive materials such as the transformer's support structure will give rise to eddy currents and be converted to heat.

Equivalent Circuit

The physical limitations of the practical transformer may be brought together as an equivalent circuit model built around an ideal lossless transformer. Power loss in the windings is current-dependent and is represented as in-series resistances R_P and R_S. Flux leakage results in a fraction of the applied voltage dropped without contributing to the mutual coupling, and thus can be modeled as reactances of each leakage inductance X_P and X_S in series with the perfectly-coupled region.

Iron losses are caused mostly by hysteresis and eddy current effects in the core, and are proportional to the square of the core flux for operation at a given frequency. Since the core flux is proportional to the applied voltage, the iron loss can be represented by a resistance R_C in parallel with the ideal transformer.

A core with finite permeability requires a magnetizing current I_M to maintain the mutual flux in the core. The magnetizing current is in phase with the flux; saturation effects cause the relationship between the two to be non-linear, but for simplicity this effect tends to be ignored in most circuit equivalents.

With a sinusoidal supply, the core flux lags the induced EMF by 90° and this effect can be modeled as a magnetizing reactance (reactance of an effective inductance) X_M in parallel with the core loss component. R_C and X_M are sometimes together termed the *magnetizing branch* of the model. If the secondary winding is made open-circuit, the current I_0 taken by the magnetizing branch represents the transformer's no-load current.

The secondary impedance R_S and X_S is frequently moved (or "referred") to the primary side after multiplying the components by the impedance scaling factor. The resulting model is sometimes termed the "exact equivalent circuit", though it retains a number of approximations, such as an assumption of linearity.

Analysis may be simplified by moving the magnetizing branch to the left of the primary impedance, an implicit assumption that the magnetizing current is low, and then summing primary and referred secondary impedances, resulting in so-called equivalent impedance. The parameters of equivalent circuit of a transformer can be calculated from the results of two transformer tests: open-circuit test and short-circuit test.

Types

A wide variety of transformer designs are used for different applications, though they share several common features. Important common transformer types include:

Autotransformer

An autotransformer has only a single winding with two end terminals, plus a third at an intermediate tap point. The primary voltage is applied across two of the terminals, and the secondary voltage taken from one of these and the third terminal. The primary and secondary circuits therefore have a number of windings turns in common. Since the volts-per-turn is the same in both windings, each develops a voltage in proportion to its number of turns. An adjustable autotransformer is made by exposing part of the winding coils and making the secondary connection through a sliding brush, giving a variable turns ratio.

Polyphase Transformers

For three-phase supplies, a bank of three individual single-phase transformers can be used, or all three phases can be incorporated as a single three-phase transformer. In this case, the magnetic circuits are connected together, the core thus containing a three-phase flow of flux.

A number of winding configurations are possible, giving rise to different attributes and phase shifts. One particular polyphase configuration is the zigzag transformer, used for grounding and in the suppression of harmonic currents.

Leakage Transformers

A leakage transformer, also called a stray-field transformer, has a significantly higher leakage inductance than other transformers, sometimes increased by a magnetic bypass or shunt in its core between primary and secondary, which is sometimes adjustable with a set screw.

This provides a transformer with an inherent current limitation due to the loose coupling between its primary and the secondary windings. The output and input currents are low enough to prevent thermal overload under all load conditions – even if the secondary is shorted.

Leakage transformers are used for arc welding and high voltage discharge lamps (neon lamps and cold cathode fluorescent lamps, which are series-connected up to 7.5 kV AC). It acts then both as a voltage transformer and as a magnetic ballast. Other applications are short-circuit-proof extra-low voltage transformers for toys or doorbell installations.

Resonant Transformers

A resonant transformer is a kind of the leakage transformer. It uses the leakage inductance of its secondary windings in combination with external capacitors, to create one or more resonant circuits. Resonant transformers such as the Tesla coil can generate very high voltages, and are able to provide much higher current than electrostatic high-voltage generation machines such as the Van de Graaff generator. One of the applications of the resonant transformer is for the CCFL inverter. Another application of the resonant transformer is to couple between stages of a superheterodyne receiver, where the selectivity of the receiver is provided by tuned transformers in the intermediate-frequency amplifiers.

Audio Transformers

Audio transformers are those specifically designed for use in audio circuits. They can be used to block radio frequency interference or the DC component of an audio signal, to split or combine audio signals, or to provide impedance matching between high and low impedance circuits, such as between a high impedance tube (valve) amplifier output and a low impedance loudspeaker, or between a high impedance instrument output and the low impedance input of a mixing console.

Such transformers were originally designed to connect different telephone systems to one another while keeping their respective power supplies isolated, and are still commonly used to interconnect professional audio systems or system components.

Being magnetic devices, audio transformers are susceptible to external magnetic fields such as those generated by AC current-carrying conductors. “Hum” is a term commonly used to describe unwanted signals originating from the “mains” power supply (typically 50 or 60 Hz). Audio transformers used for low-level signals, such as those from microphones, often included shielding to protect against extraneous magnetically-coupled signals.

Instrument Transformers

Instrument transformers are used for measuring voltge, current, power and energy in electrical systems, and for protection and control. Where a voltage or current is too large to be conveniently measured by an instrument, it can be scaled down to a standardized low value. Instrument transformers isolate measurement and control circuitry from the high currents or voltages present on the circuits being measured or controlled.

A current transformer is a transformer designed to provide a current in its secondary coil proportional to the current flowing in its primary coil. Voltage transformers (VTs), also referred to as "potential transformers" (PTs), are used in high-voltage circuits. They are designed to present a negligible load to the supply being measured, to allow protective relay equipment to be operated at lower voltages, and to have a precise winding ratio for accurate metering.

Classification

Transformers can be classified in different ways:

- *By power capacity*: from a fraction of a volt-ampere (VA) to over a thousand MVA;
- *By frequency range*: power-, audio-, or radio frequency;
- *By voltage class*: from a few volts to hundreds of kilovolts;
- *By cooling type*: air cooled, oil filled, fan cooled, or water cooled;
- *By application*: such as power supply, impedance matching, output voltage and current stabilizer, or circuit isolation;

- *By end purpose*: distribution, rectifier, arc furnace, amplifier output;
- *By winding turns ratio*: step-up, step-down, isolating (equal or near-equal ratio), variable.

Laminated Steel Cores

Transformers for use at power or audio frequencies typically have cores made of high permeability silicon steel. The steel has a permeability many times that of free space, and the core thus serves to greatly reduce the magnetizing current, and confine the flux to a path which closely couples the windings.

Early transformer developers soon realized that cores constructed from solid iron resulted in prohibitive eddy-current losses, and their designs mitigated this effect with cores consisting of bundles of insulated iron wires. Later designs constructed the core by stacking layers of thin steel laminations, a principle that has remained in use. Each lamination is insulated from its neighbors by a thin non-conducting layer of insulation. The universal transformer equation indicates a minimum cross-sectional area for the core to avoid saturation.

The effect of laminations is to confine eddy currents to highly elliptical paths that enclose little flux, and so reduce their magnitude. Thinner laminations reduce losses, but are more laborious and expensive to construct. Thin laminations are generally used on high frequency transformers, with some types of very thin steel laminations able to operate up to 10 kHz.

One common design of laminated core is made from interleaved stacks of E-shaped steel sheets capped with Ishaped pieces, leading to its name of "E-I transformer". Such a design tends to exhibit more losses, but is very economical to manufacture. The cut-core or C-core type is made by winding a steel strip around a rectangular form and then bonding the layers together. It is then cut in two, forming two C shapes, and the core assembled by binding the two C halves together with a steel strap. They have the advantage that the flux is always oriented parallel to the metal grains, reducing reluctance.

A steel core's remanence means that it retains a static magnetic field when power is removed. When power is then reapplied, the residual field will cause a high inrush current until the effect of the remaining magnetism is reduced, usually after a few cycles of the applied alternating current. Overcurrent protection devices such as fuses must be selected to allow this harmless inrush to pass. On transformers connected to long, overhead power transmission lines, induced currents due to geomagnetic disturbances during solar storms can cause saturation of the core and operation of transformer protection devices.

Distribution transformers can achieve low no-load losses by using cores made with low-loss high-permeability silicon steel or amorphous (non-crystalline) metal alloy. The higher initial cost of the core material is offset over the life of the transformer by its lower losses at light load.

Solid Cores

Powdered iron cores are used in circuits (such as switch-mode power supplies) that operate above main frequencies and up to a few tens of kilohertz. These materials combine high magnetic permeability with high bulk electrical resistivity. For frequencies extending beyond the VHF band, cores made from non-conductive magnetic ceramic materials called ferrites are common. Some radio-frequency transformers also have movable cores (sometimes called 'slugs') which allow adjustment of the coupling coefficient (and bandwidth) of tuned radio-frequency circuits.

Toroidal Cores

Toroidal transformers are built around a ring-shaped core, which, depending on operating frequency, is made from a long strip of silicon steel or permalloy wound into a coil, powdered iron, or ferrite. A strip construction ensures that the grain boundaries are optimally aligned, improving the transformer's efficiency by reducing the core's reluctance.

The closed ring shape eliminates air gaps inherent in the construction of an E-I core. The cross-section of the ring is usually square or rectangular, but more expensive cores with circular cross-sections are also available. The primary and secondary coils are often wound concentrically to cover the entire surface of the core. This minimizes the length of wire needed, and also provides screening to minimize the core's magnetic field from generating electromagnetic interference.

Toroidal transformers are more efficient than the cheaper laminated E-I types for a similar power level. Other advantages compared to E-I types, include smaller size (about half), lower weight (about half), less mechanical hum (making them superior in audio amplifiers), lower exterior magnetic field (about one tenth), low off-load losses (making them more efficient in standby circuits), single-bolt mounting, and greater choice of shapes.

The main disadvantages are higher cost and limited power capacity. Ferrite toroidal cores are used at higher frequencies, typically between a few tens of kilohertz to a megahertz, to reduce losses, physical size, and weight of switch-mode power supplies.

A drawback of toroidal transformer construction is the higher cost of windings. As a consequence, toroidal transformers are uncommon above ratings of a few kVA. Small distribution transformers may achieve some of the benefits of a toroidal core by splitting it and forcing it open, then inserting a bobbin containing primary and secondary windings.

Air Cores

A physical core is not an absolute requisite and a functioning transformer can be produced simply by placing the windings in close proximity to each other, an arrangement termed an “air-core” transformer. The air which comprises the magnetic circuit is essentially lossless, and so an air-core transformer eliminates loss due to hysteresis in the core material.

The leakage inductance is inevitably high, resulting in very poor regulation, and so such designs are unsuitable for use in power distribution. They have however very high bandwidth, and are frequently employed in radio-frequency applications, for which a satisfactory coupling coefficient is maintained by carefully overlapping the primary and secondary windings.

Windings

The conducting material used for the windings depends upon the application, but in all cases the individual turns must be electrically insulated from each other to ensure that the current travels throughout every turn. For small power and signal transformers, in which currents are low and the potential difference between adjacent turns is small, the coils are often wound from enameled magnet wire, such as Formvar wire. Larger power transformers operating at high voltages may be wound with copper rectangular strip conductors insulated by oil-impregnated paper and blocks of pressboard. High-frequency transformers operating in the tens to hundreds of kilohertz often have windings made of braided Litz wire to minimize the skin-effect and proximity effect losses. Large power transformers use multiple-stranded conductors as well, since even at low power frequencies non-uniform distribution of current would otherwise exist in high-current windings.

Each strand is individually insulated, and the strands are arranged so that at certain points in the winding, or throughout the whole winding, each portion occupies different relative positions in the complete conductor. The transposition equalizes the current flowing in each strand of the conductor, and reduces eddy current losses in the winding itself. The stranded conductor is also more flexible than a solid conductor of similar size, aiding manufacture.

For signal transformers, the windings may be arranged in a way to minimize leakage inductance and stray capacitance to improve high-frequency response. This can be done by splitting up each coil into sections, and those sections placed in layers between the sections of the other winding. This is known as a stacked type or interleaved winding.

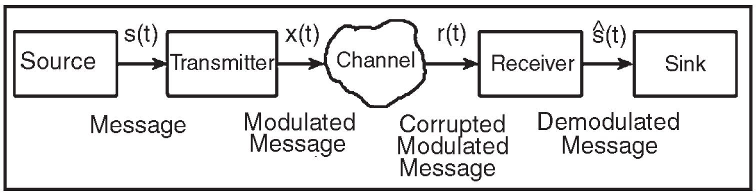
Both the primary and secondary windings on power transformers may have external connections, called taps, to intermediate points on the winding to allow selection of the voltage ratio. The taps may be connected to an automatic on-load tap changer for voltage regulation of distribution circuits. Audio-frequency transformers, used for the distribution of audio to public address loudspeakers, have taps to allow adjustment of impedance to each speaker.

A centre-tapped transformer is often used in the output stage of an audio power amplifier in a push-pull circuit. Modulation transformers in AM transmitters are very similar.

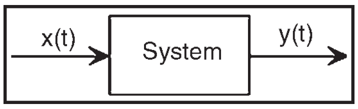
Certain transformers have the windings protected by epoxy resin. By impregnating the transformer with epoxy under a vacuum, one can replace air spaces within the windings with epoxy, thus sealing the windings and helping to prevent the possible formation of corona and absorption of dirt or water. This produces transformers more suited to damp or dirty environments, but at increased manufacturing cost.

Digital signals

introduction



Definition of a system



The fundamental model of communications is portrayed in Figure 1. In this fundamental model, each message-bearing signal, exemplified by s(t), is analog and is a function of time. A system operates on zero, one, or several signals to produce more signals or to simply absorb them (Figure above). In electrical engineering, we represent a system as a box, receiving input signals (usually coming from the left) and producing from them new output signals.

This graphical representation is known as a block diagram. We denote input signals by lines having arrows pointing into the box, output signals by arrows pointing away. As typified by the communications model, how information flows, how it is corrupted and manipulated, and how it is ultimately received is summarized by interconnecting block diagrams: The outputs of one or more systems serve as the inputs to others.

In the communications model, the source produces a signal that will be absorbed by the sink. Examples of time-domain signals produced by a source are music, speech, and characters typed on a keyboard. Signals can also be functions of two variables—an image is a signal that depends on two spatial variables—or more—television pictures (video signals) are functions of two spatial variables and time.

Thus, information sources produce signals. In physical systems, each signal corresponds to an electrical voltage or current. To be able to design systems, we must understand electrical science and technology. However, we first need to understand the big picture to appreciate the context in which the electrical engineer works.

In communication systems, messages—signals produced by sources—must be recast for transmission. The block diagram has the message s(t) passing through a block labeled transmitter that produces the signal x(t). In the case of a radio transmitter, it accepts an input audio signal and produces a signal that physically is an electromagnetic wave radiated by an antenna and propagating as Maxwell's equations predict.

In the case of a computer network, typed characters are encapsulated in packets, attached with a destination address, and launched into the Internet. From the communication systems "big picture" perspective, the same block diagram applies although the systems can be very different.

In any case, the transmitter should not operate in such a way that the message s(t) cannot be recovered from x(t). In the mathematical sense, the inverse system must exist, else the communication system cannot be considered reliable. (It is ridiculous to transmit a signal in such a way that no one can recover the original. However, clever systems exist that transmit signals so that only the "in crowd" can recover them. Such crytographic systems underlie secret communications.)

Transmitted signals next pass through the next stage, the evil channel. Nothing good happens to a signal in a channel: It can become corrupted by noise, distorted, and attenuated among many possibilities. The channel cannot be escaped (the real world is cruel), and transmitter design and receiver design focus on how best to jointly fend off the channel's effects on signals. The channel is another system in our block diagram, and produces r(t), the signal received by the receiver. If the channel were benign (good luck finding such a channel in the real world), the receiver would serve as the inverse system to the transmitter, and yield the message with no distortion.

However, because of the channel, the receiver must do its best to produce a received message s(t) that resembles s(t) as much as possible. Shannon showed in his 1948 paper that reliable—for the moment, take this word to mean error-free—digital communication was possible over arbitrarily noisy channels. It is this result that modern communications systems exploit, and why many communications systems are going"digital. Finally, the received message is passed to the information sink that somehow makes use of the message. In the communications model, the source is a system having no input but producing an output; a sink has an input and no output.

Understanding signal generation and how systems work amounts to understanding signals, the nature of the information they represent, how information is transformed between analog and digital forms, and how information can be processed by systems operating on information-bearing signals.

This understanding demands two different fields of knowledge. One is electrical science: How are signals represented and manipulated electrically? The second is signal science: What is the structure of signals, no matter what their source, what is their information content, and what capabilities does this structure force upon communication systems?

Logic signal voltage levels

Logic gate circuits are designed to input and output only two types of signals: “high” (1) and “low” (0), as represented by a variable voltage: full power supply voltage for a “high” state and zero voltage for a “low” state. In a perfect world, all logic circuit signals would exist at these extreme voltage limits, and never deviate from them (*i.e.*, less than full voltage for a “high,” or more than zero voltage for a “low”). However, in reality, logic signal voltage levels rarely attain these perfect limits due to stray voltage drops in the transistor circuitry, and so we must understand the signal level limitations of gate circuits as they try to interpret signal voltages lying somewhere between full supply voltage and zero.

TTL gates operate on a nominal power supply voltage of 5 volts, +/- 0.25 volts. Ideally, a TTL “high” signal would be 5.00 volts exactly, and a TTL “low” signal 0.00 volts exactly. However, real TTL gate circuits cannot output such perfect voltage levels, and are designed to accept “high” and “low” signals deviating substantially from these ideal values. “Acceptable” input signal voltages range from 0 volts to 0.8 volts for a “low” logic state, and 2 volts to 5 volts for a “high” logic state. “Acceptable” output signal voltages (voltage levels guaranteed by the gate manufacturer over a specified range of load conditions) range from 0 volts to 0.5 volts for a “low” logic state, and 2.7 volts to 5 volts for a “high” logic state:

If a voltage signal ranging between 0.8 volts and 2 volts were to be sent into the input of a TTL gate, there would be no certain response from the gate. Such a signal would be considered uncertain, and no logic gate manufacturer would guarantee how their gate circuit would interpret such a signal. As you can see, the tolerable ranges for output signal levels are narrower than for input signal levels, to ensure that any TTL gate outputting a digital signal into the input of another TTL gate will transmit voltages acceptable to the receiving gate. The difference between the tolerable output and input ranges is called the noise margin of the gate. For TTL gates, the low-level noise margin is the difference between 0.8 volts and 0.5 volts (0.3 volts), while the high-level noise margin is the difference between 2.7 volts and 2 volts (0.7 volts). Simply put, the noise margin is the peak amount of spurious or "noise" voltage that may be superimposed on a weak gate output voltage signal before the receiving gate might interpret it wrongly:

CMOS gate circuits have input and output signal specifications that are quite different from TTL. For a CMOS gate operating at a power supply voltage of 5 volts, the acceptable input signal voltages range from 0 volts to 1.5 volts for a "low" logic state, and 3.5 volts to 5 volts for a "high" logic state. "Acceptable" output signal voltages (voltage levels guaranteed by the gate manufacturer over a specified range of load conditions) range from 0 volts to 0.05 volts for a "low" logic state, and 4.95 volts to 5 volts for a "high" logic state: It should be obvious from these figures that CMOS gate circuits have far greater noise margins than TTL: 1.45 volts for CMOS low-level and high-level margins, versus a maximum of 0.7 volts for TTL. In other words, CMOS circuits can tolerate over twice the amount of superimposed "noise" voltage on their input lines before signal interpretation errors will result.

CMOS noise margins widen even further with higher operating voltages. Unlike TTL, which is restricted to a power supply voltage of 5 volts, CMOS may be powered by voltages as high as 15 volts (some CMOS circuits as high as 18 volts). Shown here are the acceptable "high" and "low" states, for both input and output, of CMOS integrated circuits operating at 10 volts and 15 volts, respectively:

The margins for acceptable "high" and "low" signals may be greater than what is shown in the previous illustrations. What is shown represents "worst-case" input signal performance, based on manufacturer's specifications. In practice, it may be found that a gate circuit will tolerate "high" signals of considerably less voltage and "low" signals of considerably greater voltage than those specified here.

Conversely, the extremely small output margins shown — guaranteeing output states for "high" and "low" signals to within 0.05 volts of the power supply "rails" — are optimistic. Such "solid" output voltage levels will be true only for conditions of minimum loading. If the gate is sourcing or sinking substantial current to a load, the output voltage will not be able to maintain these optimum levels, due to internal channel resistance of the gate's final output MOSFETs.

Within the "uncertain" range for any gate input, there will be some point of demarcation dividing the gate's actual "low" input signal range from its actual "high" input signal range. That is, somewhere between the lowest "high" signal voltage level and the highest "low" signal voltage level guaranteed by the gate manufacturer, there is a threshold voltage at which the gate will actually switch its interpretation of a signal from "low" or "high" or visa-versa. For most gate circuits, this unspecified voltage is a single point: In the presence of AC "noise" voltage superimposed on the DC input signal, a single threshold point at which the gate alters its interpretation of logic level will result in an erratic output:

If this scenario looks familiar to you, it's because you remember a similar problem with (analog) voltage comparator op-amp circuits. With a single threshold point at which an input causes the output to switch between "high" and "low" states, the presence of significant noise will cause erratic changes in the output:

The solution to this problem is a bit of positive feedback introduced into the amplifier circuit. With an op-amp, this is done by connecting the output back around to the non-inverting (+) input through a resistor.

In a gate circuit, this entails redesigning the internal gate circuitry, establishing the feedback inside the gate package rather than through external connections. A gate so designed is called a Schmitt trigger. Schmitt triggers interpret varying input voltages according to two threshold voltages: a positive-going threshold (V_{T+}), and a negative-going threshold (V_{T-}):

Schmitt trigger gates are distinguished in schematic diagrams by the small "hysteresis" symbol drawn within them, reminiscent of the B-H curve for a ferromagnetic material. Hysteresis engendered by positive feedback within the gate circuitry adds an additional level of noise immunity to the gate's performance.

Schmitt trigger gates are frequently used in applications where noise is expected on the input signal line(s), and/or where an erratic output would be very detrimental to system performance. The differing voltage level requirements of TTL and CMOS technology present problems when the two types of gates are used in the same system.

Although operating CMOS gates on the same 5.00 volt power supply voltage required by the TTL gates is no problem, TTL output voltage levels will not be compatible with CMOS input voltage requirements. Take for instance a TTL NAND gate outputting a signal into the input of a CMOS inverter gate. Both gates are powered by the same 5.00 volt supply (V_{cc}). If the TTL gate outputs a "low" signal (guaranteed to be between 0 volts and 0.5 volts), it will be properly interpreted by the CMOS gate's input as a "low" (expecting a voltage between 0 volts and 1.5 volts):

However, if the TTL gate outputs a "high" signal (guaranteed to be between 5 volts and 2.7 volts), it might not be properly interpreted by the CMOS gate's input as a "high" (expecting a voltage between 5 volts and 3.5 volts):

Given this mismatch, it is entirely possible for the TTL gate to output a valid "high" signal (valid, that is, according to the standards for TTL) that lies within the "uncertain" range for the CMOS input, and may be (falsely) interpreted as a "low" by the receiving gate. An easy "fix" for this problem is to augment the TTL gate's "high" signal voltage level by means of a pullup resistor:

Something more than this, though, is required to interface a TTL output with a CMOS input, if the receiving CMOS gate is powered by a greater power supply voltage:

There will be no problem with the CMOS gate interpreting the TTL gate's "low" output, of course, but a "high" signal from the TTL gate is another matter entirely. The guaranteed output voltage range of 2.7 volts to 5 volts from the TTL gate output is nowhere near the CMOS gate's acceptable range of 7 volts to 10 volts for a "high" signal.

If we use an open-collector TTL gate instead of a totem-pole output gate, though, a pullup resistor to the 10 volt V_{dd} supply rail will raise the TTL gate's "high" output voltage to the full power supply voltage supplying the CMOS gate. Since an open-collector gate can only sink current, not source current, the "high" state voltage level is entirely determined by the power supply to which the pullup resistor is attached, thus neatly solving the mismatch problem:

Due to the excellent output voltage characteristics of CMOS gates, there is typically no problem connecting a CMOS output to a TTL input. The only significant issue is the current loading presented by the TTL inputs, since the CMOS output must sink current for each of the TTL inputs while in the "low" state.

When the CMOS gate in question is powered by a voltage source in excess of 5 volts (V_{cc}), though, a problem will result. The "high" output state of the CMOS gate, being greater than 5 volts, will exceed the TTL gate's acceptable input limits for a "high" signal. A solution to this problem is to create an "open-collector" inverter circuit using a discrete NPN transistor, and use it to interface the two gates together:

The "R_{pullup}" resistor is optional, since TTL inputs automatically assume a "high" state when left floating, which is what will happen when the CMOS gate output is "low" and the transistor cuts off. Of course, one very important consequence of implementing this solution is the logical inversion created by the transistor: when the CMOS gate outputs a "low" signal, the TTL gate sees a "high" input; and when the CMOS gate outputs a “high” signal, the transistor saturates and the TTL gate sees a “low” input. So long as this inversion is accounted for in the logical scheme of the system, all will be well.

Concept of information

The concept of information is central to communication. There is no precise definition of the word “information”. So, instead of information, we deal with “message”. Message is defined as the physical manifestation of information as produced by the source

Analog

An analog message is a physical quantity that varies with time usually in a smooth and continuous fashion.

Examples of analog messages are:

- The acoustic pressure produced when you speak
- The angular position of an aircraft gyro
- The light intensity at some point in a television image.

Digital

A digital message is an ordered sequence of symbols selected from a finite set of discrete elements.

Examples of digital messages are:

- Letters printed on this page
- Listing of hourly temperature readings
- The keys you press at a computer terminal.

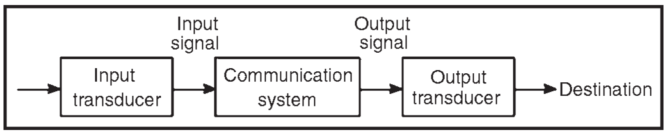
Whether the message is analog or digital, it needs to be converted into an electrical signal. (There are only few messages, which are inherently electrical.)

Input and Output

The input transducer converts the message to an electrical signal say a voltage or current. Another transducer at the destination converts the output signal to the desired message form.

Illustration

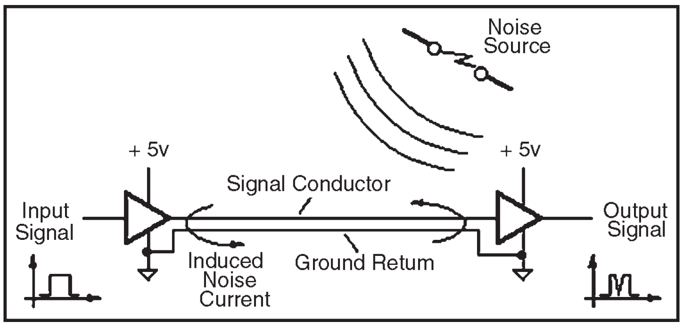
The transducers in a voice communication system could be microphone at the input and a loudspeaker at the output.



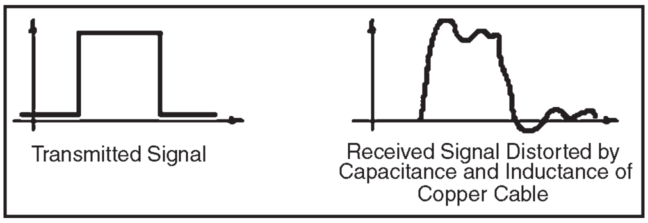
Communication system with input and output transducers

Step,Ramp, and Impulse Functions

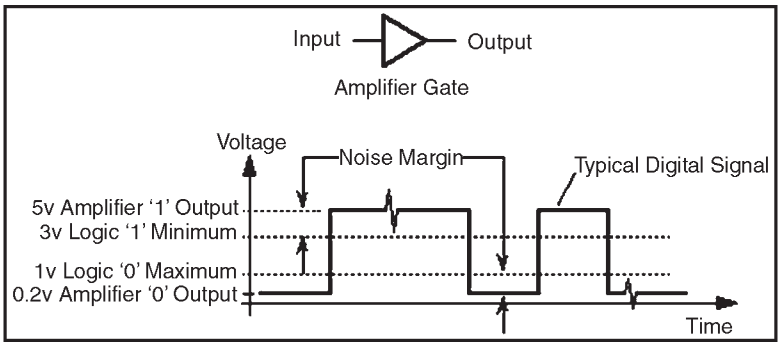
Because of the very high switching rate and relatively low signal strength found on data, address, and other buses within a computer, direct extension of the buses beyond the confines of the main circuit board or plug-in boards would pose serious problems. First, long runs of electrical conductors, either on printed circuit boards or through cables, act like receiving antennas for electrical noise radiated by motors, switches, and electronic circuits:



Such noise becomes progressively worse as the length increases, and may eventually impose an unacceptable error rate on the bus signals. Just a single bit error in transferring an instruction code from memory to a microprocessor chip may cause an invalid instruction to be introduced into the instruction stream, in turn causing the computer to totally cease operation. A second problem involves the distortion of electrical signals as they pass through metallic conductors. Signals that start at the source as clean, rectangular pulses may be received as rounded pulses with ringing at the rising and falling edges:


Transmitted Signal
Received Signal Distorted by Capacitance and Inductance of Copper Cable

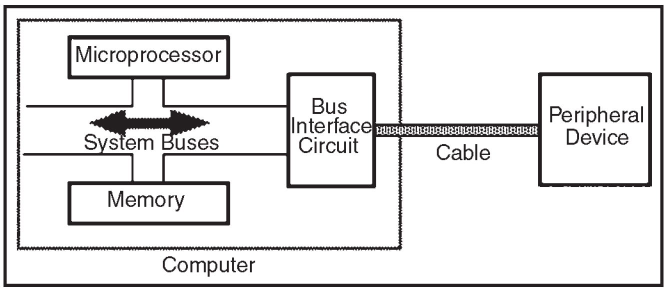
These effects are properties of transmission through metallic conductors, and become more pronounced as the conductor length increases. To compensate for distortion, signal power must be increased or the transmission rate decreased. Special amplifier circuits are designed for transmitting direct (unmodulated) digital signals through cables. For the relatively short distances between components on a printed circuit board or along a computer backplane, the amplifiers are in simple IC chips that operate from standard +5v power. The normal output voltage from the amplifier for logic '1' is slightly higher than the minimum needed to pass the logic '1' threshold. Correspondingly for logic '0', it is slightly lower. The difference between the actual output voltage and the threshold value is referred to as the noise margin, and represents the amount of noise voltage that can be added to the signal without creating an error:



Transmission over Medium Distances

Computer peripherals such as a printer or scanner generally include mechanisms that cannot be situated within the computer itself. Our first thought might be just to extend the computer's internal buses with a cable of sufficient length to reach the peripheral. Doing so, however, would expose all bus transactions to external noise and distortion even though only a very small percentage of these transactions concern the distant peripheral to which the bus is connected.

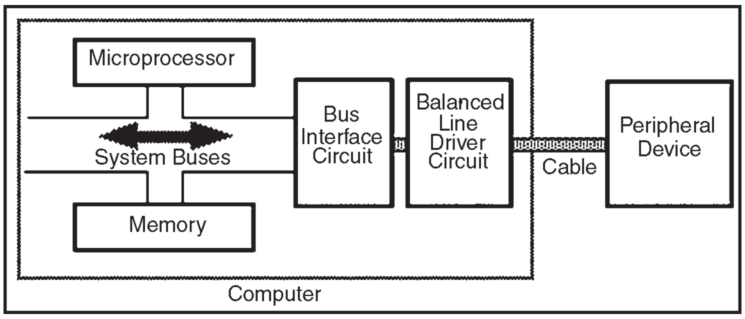
If a peripheral can be located within 20 feet of the computer, however, relatively simple electronics may be added to make data transfer through a cable efficient and reliable. To accomplish this, a bus interface circuit is installed in the computer:



It consists of a holding register for peripheral data, timing and formatting circuitry for external data transmission, and signal amplifiers to boost the signal sufficiently for transmission through a cable. When communication with the peripheral is necessary, data is first deposited in the holding register by the microprocessor. This data will then be reformatted, sent with error-detecting codes, and transmitted at a relatively slow rate by digital hardware in the bus interface circuit. In addition, the signal power is greatly boosted before transmission through the cable. These steps ensure that the data will not be corrupted by noise or distortion during its passage through the cable. In addition, because only data destined for the peripheral is sent, the party-line transactions taking place on the computer's buses are not unnecessarily exposed to noise. Data sent in this manner may be transmitted in byte-serial format if the cable has eight parallel channels (at least 10 conductors for half-duplex operation), or in bit-serial format if only a single channel is available.

Transmission over Long Distances

When relatively long distances are involved in reaching a peripheral device, driver circuits must be inserted after the bus interface unit to compensate for the electrical effects of long cables:

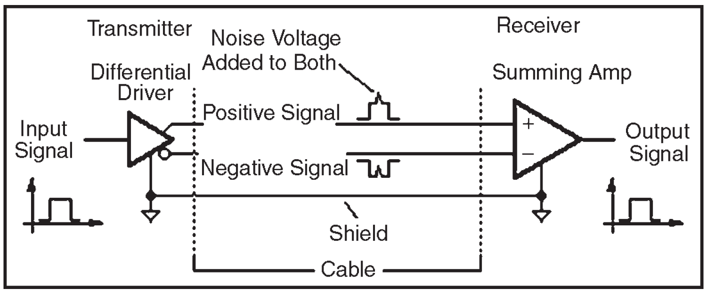


This is the only change needed if a single peripheral is used. However, if many peripherals are connected, or if other computer stations are to be linked, a local area network (LAN) is required, and it becomes necessary to drastically change both the electrical drivers and the protocol to send messages through the cable. Because multiconductor cable is expensive, bit-serial transmission is almost always used when the distance exceeds 20 feet.

In either a simple extension cable or a LAN, a balanced electrical system is used for transmitting digital data through the channel. This type of system involves at least two wires per channel, neither of which is a ground. Note that a common ground return cannot be shared by multiple channels in the same cable as would be possible in an unbalanced system.

The basic idea behind a balanced circuit is that a digital signal is sent on two wires simultaneously, one wire expressing a positive voltage image of the signal and the other a negative voltage image. When both wires reach the destination, the signals are subtracted by a summing amplifier, producing a signal swing of twice the value found on either incoming line. If the cable is exposed to radiated electrical noise, a small voltage of the same polarity is added to both wires in the cable. When the signals are subtracted by the summing amplifier, the noise cancels and the signal emerges from the cable without noise:

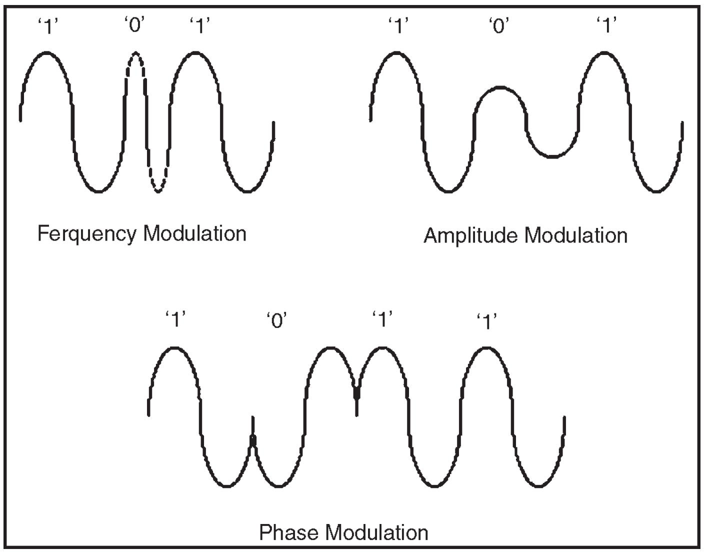
A great deal of technology has been developed for LAN systems to minimize the amount of cable required and maximize the throughput. The costs of a LAN have been concentrated in the electrical interface card that would be installed in PCs or peripherals to drive the cable, and in the communications software, not in the cable itself (whose cost has been minimized). Thus, the cost and complexity of a LAN are not particularly affected by the distance between stations.



Transmission over Distances (greater than 4000 feet)

Data communications through the telephone network can reach any point in the world. The volume of overseas fax transmissions is increasing constantly, and computer networks that link thousands of businesses, governments, and universities are pervasive. Transmissions over such distances are not generally accomplished with a direct-wire digital link, but rather with digitally-modulated analog carrier signals. This technique makes it possible to use existing analog telephone voice channels for digital data, although at considerably reduced data rates compared to a direct digital link.

Transmission of data from your personal computer to a timesharing service over phone lines requires that data signals be converted to audible tones by a modem. An audio sine wave carrier is used, and, depending on the baud rate and protocol, will encode data by varying the frequency, phase, or amplitude of the carrier. The receiver's modem accepts the modulated sine wave and extracts the digital data from it. Several modulation techniques typically used in encoding digital data for analog transmission are shown below:



Similar techniques may be used in digital storage devices such as hard disk drives to encode data for storage using an analog medium.

4

System Software

System software is computer software designed to operate the computer hardware and to provide a platform for running application software. The most basic types of system software are:

- The computer BIOS and device firmware, which provide basic functionality to operate and control the hardware connected to or built into the computer.
- The operating system (prominent examples being Microsoft Windows, Mac OS X and Linux), which allows the parts of a computer to work together by performing tasks like transferring data between memory and disks or rendering output onto a display device. It also provides a platform to run high-level system software and application software.
- Utility software, which helps to analyze, configure, optimize and maintain the computer.

In some publications, the term *system software* is also used to designate software development tools (like a compiler, linker or debugger). Computer purchasers seldom buy a computer primarily because of its system software (But purchasers of devices like mobile phones because of there system software, as is the case with the iPhone, as the system software of such devices is difficult for the end-user to modify). Rather, system software serves as a useful (even necessary) level of infrastructure code, generally built-in or pre-installed. In contrast to system software, software that allows users to do things like create text documents, play games, listen to music, or surf the web is called application software.

Types of System Software Programmes

System software helps use the operating system and computer system. It includes diagnostic tools, compilers, servers, windowing systems, utilities, language translator, data communication programmes, database systems and more. The purpose of system software is to insulate the applications programmer as much as possible from the complexity and specific details of the particular computer being used, especially memory and other hardware features, and such accessory devices as communications, printers, readers, displays, keyboards, etc. Specific kinds of system software include:

- Loaders
- Linkers
- Utility software
- Desktop environment / Graphical user interface

- Shells
- BIOS
- Hypervisors
- Boot loaders
- Database Management Systems(SQL, NoSQL)

If system software is stored on non-volatile memory such as integrated circuits, it is usually termed firmware.

ENTERPRIZE SOFTWARE

Enterprize software, also known as enterprize application software (EAS), is software used in organizations, such as in a business or government, as opposed to software chosen by individuals (for example, retail software). Enterprize software is an integral part of a (Computer Based) Information System. Services provided by enterprize software are typically business-oriented tools such as online shopping and online payment processing, interactive product catalogue, automated billing systems, security, content management, IT service management, customer relationship management, resource planning, business intelligence, HR management, manufacturing, application integration, and forms automation.

Definitions

While there is no single, widely accepted list of enterprize software characteristics, this section is intended to summarize definitions from multiple sources. Enterprize software describes a collection of computer programmes with common business applications, tools for modeling how the entire organization works, and development tools for building applications unique to the organization. The software is intended to solve an enterprize-wide problem (rather than a departmental problem) and often written using an Enterprize Software Architecture. Enterprize level software aims to improve the enterprize's productivity and efficiency by providing business logic support functionality. Capterra broadly defines enterprize software in the following manner:

- Targets any type of organization — corporations, partnerships, sole proprietorships, nonprofits, government agencies — but does not directly target consumers.
- Targets any industry.
- Targets both large and small organizations — from Fortune 500 to "mom and pop" businesses.
- Includes function-specific (Accounting, HR, Supply Chain, etc.) and industry-specific (Manufacturing, Retail, Healthcare, etc.) solutions.

Due to the cost of building or buying what is often non-free proprietary software, only large enterprizes attempt to implement such enterprize software that models the entire business enterprize and is the core IT system of governing the enterprize and the core of communication within the enterprize. As business enterprizes have similar departments and systems in common, enterprize software is often available as a suite of programmes that have attached enterprize development tools to customize the programmes to the specific enterprize. Generally, these tools are complex enterprize programming tools that require specialist capabilities. Thus, one often sees job listings for a programmer who is required to have specific knowledge of a particular set of enterprize tools, such as "must be an SAP developer". Characteristics of enterprize software are performance, scalability, and robustness. Enterprize software typically has interfaces to other enterprize software (for example LDAP to directory services) and is centrally managed (a single admin page for example).

Enterprize Application Software

Enterprize application software is application software that performs business functions such as order processing, procurement, production scheduling, customer information management, and accounting. It is typically hosted on servers and provides simultaneous services to a large number of users, typically over a computer network. This is in contrast to a single-user application that is executed on a user's personal computer and serves only one user at a time.

Types

- Enterprize software can be designed and implemented by an information technology (IT) group within a company.
- It may also be purchased from an independent enterprize software developer, that often installs and maintains the software for their customers. Installation, customization, and maintenance can also be outsourced to an IT consulting company.
- Another model is based on a concept called on-demand software, or Software as a Service (SaaS). The on-demand model of enterprize software is made possible through the widespread distribution of

broadband access to the Internet. Software as a Service vendors maintain enterprize software on servers within their own company data center and then provide access to the software to their enterprize customers via the Internet.

Enterprize software is often categorized by the business function that it automates - such as accounting software or sales force automation software. Similarly for industries - for example, there are enterprize systems devised for the health care industry, or for manufacturing enterprizes.

Developers

Major organizations in the enterprize software field include SAP, IBM, BMC Software, HP Software Division, Redwood Software, UC4 Software, JBoss (Red Hat), Microsoft, Adobe Systems, Oracle Corporation, Inquest Technologies, Computer Associates, and ASG Software Solutions but there are thousands of competing vendors.

Criticism

The word *enterprize* can have various connotations. Sometimes the term is used merely as a synonym for *organization*, whether it be very large (e.g., a corporation with thousands of employees), very small (a sole proprietorship), or an intermediate size. Often the term is used only to refer to very large organizations, although it has become a corporate-speak buzzword and may be heard in other uses. Some enterprize software vendors using the latter definition develop highly complex products that are often overkill for smaller organizations, and the application of these can be a very frustrating task. Thus, sometimes "enterprize" might be used sarcastically to mean overly complex software. The adjective "enterprizey" is sometimes used to make this sarcasm explicit. In this usage, the term "enterprizey" is intended to go beyond the concern of "overkill for smaller organizations" to imply the software is overly complex even for large organizations and simpler solutions are available.

ACCOUNTING SOFTWARE

Accounting software is application software that records and processes accounting transactions within functional modules such as accounts payable, accounts receivable, payroll, and trial balance. It functions as an accounting information system. It may be developed in-house by the company or organization using it, may be purchased from a third party, or may be a combination of a third-party application software package with local modifications. It varies greatly in its complexity and cost. The market has been undergoing considerable consolidation since the mid 1990s, with many suppliers ceasing to trade or being bought by larger groups.

Modules

Accounting software is typically composed of various modules, different sections dealing with particular areas of accounting. Among the most common are:

Core Modules

- Accounts receivable—where the company enters money received

- Accounts payable—where the company enters its bills and pays money it owes
- General ledger—the company's "books"
- Billing—where the company produces invoices to clients/customers
- Stock/Inventory—where the company keeps control of its inventory
- Purchase Order—where the company orders inventory
- Sales Order—where the company records customer orders for the supply of inventory
- Cash Book—where the company records collection and payment

Non Core Modules

- Debt Collection—where the company tracks attempts to collect overdue bills (sometimes part of accounts receivable)
- Electronic payment processing
- Expense—where employee business-related expenses are entered
- Inquiries—where the company looks up information on screen without any edits or additions
- Payroll—where the company tracks salary, wages, and related taxes
- Reports—where the company prints out data
- Timesheet—where professionals (such as attorneys and consultants) record time worked so that it can be billed to clients
- Purchase Requisition—where requests for purchase orders are made, approved and tracked

(Different vendors will use different names for these modules)

Implementations

In many cases, implementation (i.e. the installation and configuration of the system at the client) can be a bigger consideration than the actual software chosen when it comes down to the total cost of ownership for the business. Most midmarket and larger applications are sold exclusively through resellers, developers and consultants. Those organizations generally pass on a license fee to the software vendor and then charge the client for installation, customization and support services. Clients can normally count on paying roughly 50-200% of the price of the software in implementation and consulting fees. Other organizations sell to, consult with and support clients directly, eliminating the reseller.

Categories

Personal Accounting

Mainly for home users that use accounts payable type accounting transactions, managing budgets and simple account reconciliation at the inexpensive end of the market.

Low End

At the low end of the business markets, inexpensive applications software allows most general business accounting functions to be performed. Suppliers frequently serve a single national market, while larger suppliers offer separate solutions in each national market. Many of the low end products are characterized by being “single-entry” products, as opposed to double-entry systems seen in many businesses. Some products have considerable functionality but are not considered GAAP or IFRS/FASB compliant. Some low-end systems do not have adequate security nor audit trails.

Mid Market

The mid-market covers a wide range of business software that may be capable of serving the needs of multiple national accountancy standards and allow accounting in multiple currencies. In addition to general accounting functions, the software may include integrated or add-on management information systems, and may be oriented towards one or more markets, for example with integrated or add-on project accounting modules. Software applications in this market typically include the following features:

- Industry-standard robust databases
- Industry-standard reporting tools
- Tools for configuring or extending the application (eg an SDK, access to programme code.

High End

The most complex and expensive business accounting software is frequently part of an extensive suite of software often known as Enterprize resource planning or *ERP* software. These applications typically have a very long implementation period, often greater than six months. In many cases, these applications are simply a set of functions which require significant integration, configuration and customization to even begin to resemble an accounting system. The advantage of a high-end solution is that these systems are designed to support individual company specific processes, as they are highly customizable and can be tailored to exact business requirements. This usually comes at a significant cost in terms of money and implementation time.

Vertical Market

Some business accounting software is designed for specific business types. It will include features that are specific to that industry. The choice of whether to purchase an industry-specific application or a general-purpose application is often very difficult. Concerns over a custom-built application or one designed for a specific industry include:

- Smaller development team
- Increased risk of vendor business failing
- Reduced availability of support

This can be weighed up against:

- Less requirement for customization
- Reduced implementation costs
- Reduced end-user training time and costs

Some important types of vertical accounting software are:

- Banking
- Construction
- Medical
- Nonprofit
- Point of Sale (Retail)
- Daycare accounting (a.k.a. Child care management software)

Hybrid Solutions

As technology improves, software vendors have been able to offer increasingly advanced software at lower prices. This software is suitable for companies at multiple stages of growth. Many of the features of Mid Market and High End software (including advanced customization and extremely scalable databases) are required even by small businesses as they open multiple locations or grow in size. Additionally, with more and more companies expanding overseas or allowing workers to home office, many smaller clients have a need to connect multiple locations. Their options are to employ software-as-a-service or another application that offers them similar accessibility from multiple locations over the internet. Bob Frankston has noted that his VisiCalc wasn't an early accounting programme and that software that "overly tuned for such function (Javelin, Lotus Improv, etc.) completely failed."

5

Computer Software

Computer software, or just software, is a collection of computer programmes and related data that provide the instructions telling a computer what to do and how to do it. We can also say software refers to one or more computer programmes and data held in the storage of the computer for some purposes. In other words software is a set of programmes, procedures, algorithms and its documentation. Programme software performs the function of the programme it implements, either by directly providing instructions to the computer hardware or by serving as input to another piece of software. The term was coined to contrast to the old term hardware (meaning physical devices). In contrast to hardware, software is intangible, meaning it "cannot be touched". Software is also sometimes used in a more narrow sense, meaning application software only. Sometimes the term includes data that has not traditionally been associated with computers, such as film, tapes, and records. Examples of computer software include:

- Application software includes end-user applications of computers such as word processors or video games, and ERP software for groups of users.
- Middleware controls and co-ordinates distributed systems.
- Programming languages define the syntax and semantics of computer programmes. For example, many mature banking applications were written in the COBOL language, originally invented in 1959. Newer applications are often written in more modern programming languages.
- System software includes operating systems, which govern computing resources. Today large applications running on remote machines such as Websites are considered to be system software, because the end-user interface is generally through a graphical user interface, such as a web browser.
- Testware is software for testing hardware or a software package.
- Firmware is low-level software often stored on electrically programmable memory devices. Firmware is given its name because it is treated like hardware and run ("executed") by other software programmes.
- Shrinkware is the older name given to consumer-purchased software, because it was often sold in retail stores in a shrink-wrapped box.
- Device drivers control parts of computers such as disk drives, printers, CD drives, or computer monitors.

- Programming tools help conduct computing tasks in any category listed above. For programmers, these could be tools for debugging or reverse engineering older legacy systems in order to check source code compatibility.

History

The first theory about software was proposed by Alan Turing in his 1935 essay *Computable numbers with an application to the Entscheidungsproblem (Decision problem).* The term "software" was first used in print by John W. Tukey in 1958. Colloquially, the term is often used to mean application software. In computer science and software engineering, software is all information processed by computer system, programmes and data. The academic fields studying software are computer science and software engineering. The history of computer software is most often traced back to the first software bug in 1946. As more and more programmes enter the realm of firmware, and the hardware itself becomes smaller, cheaper and faster as predicted by Moore's law, elements of computing first considered to be software, join the ranks of hardware. Most hardware companies today have more software programmers on the payroll than hardware designers, since software tools have automated many tasks of Printed circuit board engineers. Just like the Auto industry, the Software industry has grown from a few visionaries operating out of their garage with prototypes. Steve Jobs and Bill Gates were the Henry Ford and Louis Chevrolet of their times, who capitalized on ideas already commonly known before they started in the business. In the case of Software development, this moment is generally agreed to be the publication in the 1980s of the specifications for the IBM Personal Computer published by IBM employee Philip Don Estridge. Today his move would be seen as a type of crowd-sourcing.

Until that time, software was *bundled* with the hardware by Original equipment manufacturers (OEMs) such as Data General, Digital Equipment and IBM. When a customer bought a minicomputer, at that time the smallest computer on the market, the computer did not come with Pre-installed software, but needed to be installed by engineers employed by the OEM. Computer hardware companies not only bundled their software, they also placed demands on the location of the hardware in a refrigerated space called a computer room. Most companies had their software on the books for 0 dollars, unable to claim it as an asset (this is similar to financing of popular music in those days). When Data General introduced the Data General Nova, a company called Digidyne wanted to use its RDOS operating system on its own hardware clone. Data General refused to license their software (which was hard to do, since it was on the books as a free asset), and claimed their "bundling rights".

The Supreme Court set a precedent called Digidyne v. Data General in 1985. The Supreme Court let a 9th circuit decision stand, and Data General was eventually forced into licensing the Operating System software because it was ruled that restricting the license to only DG hardware was an illegal *tying arrangement*. Soon after, IBM 'published' its DOS source for free, and Microsoft was born. Unable to sustain the loss from lawyer's fees, Data General ended up being taken over by EMC Corporation. The Supreme Court decision made it possible to value software, and also purchase Software patents. The move by IBM was almost a protest at the time. Few in the industry believed that anyone would profit from it other than IBM (through free publicity). Microsoft and Apple were able to thus cash in on 'soft' products. It is hard to imagine today that people once felt that software was worthless without a machine. There are many successful companies today that sell only software products, though there are still many common software licensing problems due to the complexity of designs and poor documentation, leading to patent trolls. With open software specifications and the possibility of software licensing, new opportunities arose for software tools that then became the de facto standard, such as DOS for operating systems, but also various proprietary word processing and spreadsheet programmes. In a similar growth pattern, proprietary development methods became standard Software development methodology.

Overview

Software includes all the various forms and roles that digitally stored *data* may have and play in a computer (or similar system), regardless of whether the data is used as *code* for a CPU, or other interpreter, or whether it represents other kinds of information. Software thus encompasses a wide array of products that may be developed using different techniques such as ordinary programming languages, scripting languages, microcode, or an FPGA configuration. The types of software include web pages developed in languages and frameworks like HTML, PHP, Perl, JSP, ASP.NET, XML, and desktop applications like OpenOffice.org, Microsoft Word developed in languages like C, C++, Java, C#, or Smalltalk. Application software usually runs on an underlying software operating systems such as Linux or Microsoft Windows. Software (or firmware) is also used in video games and for the configurable parts of the logic systems of automobiles, televisions, and other consumer electronics. Computer software is so called to distinguish it from computer hardware, which encompasses the physical interconnections and devices required to store and execute (or run) the software. At the lowest level, executable code consists of machine language instructions specific to an individual processor. A machine language consists of groups of binary values signifying processor instructions that change the state of the computer from its preceding state.

Programmes are an ordered sequence of instructions for changing the state of the computer in a particular sequence. It is usually written in high-level programming languages that are easier and more efficient for humans to use (closer to natural language) than machine language. High-level languages are compiled or interpreted into machine language object code. Software may also be written in an assembly language, essentially, a mnemonic representation of a machine language using a natural language alphabet. Assembly language must be assembled into object code via an assembler.

Types of Software

Practical computer systems divide software systems into three major classes: system software, programming software and application software, although the distinction is arbitrary, and often blurred.

System Software

System software provides the basic functions for computer usage and helps run the computer hardware and system. It includes a combination of the following:

- Device drivers
- Operating systems
- Servers
- Utilities
- Window systems

System software is responsible for managing a variety of independent hardware components, so that they can work together harmoniously. Its purpose is to unburden the application software programmer from the often complex details of the particular computer being used, including such accessories as communications devices, printers, device readers, displays and keyboards, and also to partition the computer's resources such as memory and processor time in a safe and stable manner.

Programming Software

Programming software usually provides tools to assist a programmer in writing computer programmes, and software using different programming languages in a more convenient way. The tools include:

- Compilers
- Debuggers
- Interpreters

- Linkers
- Text editors

An Integrated development environment (IDE) is a single application that attempts to manage all these functions..

Application Software

Application software is developed to aid in any task that benefits from computation.

It is a broad category, and encompasses software of many kinds, including the internet browser being used to display this page.

This category includes:

- Business software
- Databases
- Decision making software
- Educational software
- Image editing
- Industrial automation
- Mathematical software
- Medical software
- Molecular modeling software
- Quantum chemistry and solid state physics software
- Simulation software
- Spreadsheets
- Telecommunications (i.e., the Internet and everything that flows on it)
- Video games
- Word processing

Software Topics

Architecture

Users often see things differently than programmers. People who use modern general purpose computers (as opposed to embedded systems, analog computers and supercomputers) usually see three layers of software performing a variety of tasks: platform, application, and user software.

- Platform software: Platform includes the firmware, device drivers, an operating system, and typically a graphical user interface which, in total, allow a user to interact with the computer and its peripherals (associated equipment). Platform software often comes bundled with the computer. On a PC you will usually have the ability to change the platform software.
- Application software: Application software or Applications are what most people think of when they think of software. Typical examples include office suites and video games. Application software is often purchased separately from computer hardware. Sometimes applications are bundled with the computer, but that does not change the fact that they run as independent applications. Applications are usually independent programmes from the operating system, though they are often tailored for specific platforms. Most users think of compilers, databases, and other "system software" as applications.
- User-written software: End-user development tailors systems to meet users' specific needs. User software

include spreadsheet templates and word processor templates. Even email filters are a kind of user software. Users create this software themselves and often overlook how important it is. Depending on how competently the user-written software has been integrated into default application packages, many users may not be aware of the distinction between the original packages, and what has been added by co-workers.

Documentation

Most software has software documentation so that the end user can understand the programme, what it does, and how to use it. Without clear documentation, software can be hard to use—especially if it is very specialized and relatively complex like Photoshop or AutoCAD. Developer documentation may also exist, either with the code as comments and/or as separate files, detailing how the programmes works and can be modified.

Library

An executable is almost always not sufficiently complete for direct execution. Software libraries include collections of functions and functionality that may be embedded in other applications. Operating systems include many standard Software libraries, and applications are often distributed with their own libraries.

Standard

Since software can be designed using many different programming languages and in many different operating systems and operating environments, software standard is needed so that different software can understand and exchange information between each other. For instance, an email sent from a Microsoft Outlook should be readable from Yahoo! Mail and vice versa.

Execution

Computer software has to be "loaded" into the computer's storage (such as the hard drive or memory). Once the software has loaded, the computer is able to *execute* the software. This involves passing instructions from the application software, through the system software, to the hardware which ultimately receives the instruction as machine code. Each instruction causes the computer to carry out an operation – moving data, carrying out a computation, or altering the control flow of instructions. Data movement is typically from one place in memory to another. Sometimes it involves moving data between memory and registers which enable high-speed data access in the CPU. Moving data, especially large amounts of it, can be costly. So, this is sometimes avoided by using "pointers" to data instead. Computations include simple operations such as incrementing the value of a variable data element. More complex computations may involve many operations and data elements together.

Quality and Reliability

Software quality is very important, especially for commercial and system software like Microsoft Office, Microsoft Windows and Linux. If software is faulty (buggy), it can delete a person's work, crash the computer and do other unexpected things. Faults and errors are called "bugs."

Many bugs are discovered and eliminated (debugged) through software testing. However, software testing rarely – if ever – eliminates every bug; some programmers say that "every programme has at least one more bug" (Lubarsky's Law). All major software companies, such as Microsoft, Novell and Sun Microsystems, have their own software testing departments with the specific goal of just testing. Software can be tested through unit testing, regression testing and other methods, which are done manually, or most commonly, automatically, since the amount of code to be tested can be quite large. For instance, NASA has extremely rigorous software testing procedures for many operating systems and communication functions. Many NASA based operations interact and identify each other through command programmes called software. This enables many people who work at NASA to check and evaluate functional systems overall. Programmes containing command software enable hardware engineering and system operations to function much easier together.

License

The software's license gives the user the right to use the software in the licensed environment. Some software comes with the license when purchased off the shelf, or an OEM license when bundled with hardware. Other software comes with a free software license, granting the recipient the rights to modify and redistribute the software. Software can also be in the form of freeware or shareware.

Patents

Software can be patented in some but not all countries;

however, software patents can be controversial in the software industry with many people holding different views about it. The controversy over software patents is about specific algorithms or techniques that the software contains, which may not be duplicated by others and considered intellectual property and copyright infringement depending on the severity.

Design and Implementation

Design and implementation of software varies depending on the complexity of the software. For instance, design and creation of Microsoft Word software will take much more time than designing and developing Microsoft Notepad because of the difference in functionalities in each one. Software is usually designed and created (coded/written/programmed) in integrated development environments (IDE) like Eclipse, Emacs and Microsoft Visual Studio that can simplify the process and compile the programme. As noted in different section, software is usually created on top of existing software and the application programming interface (API) that the underlying software provides like GTK+, JavaBeans or Swing. Libraries (APIs) are categorized for different purposes. For instance, JavaBeans library is used for designing enterprize applications, Windows Forms library is used for designing graphical user interface (GUI) applications like Microsoft Word, and Windows Communication Foundation is used for designing web services.

Underlying computer programming concepts like quicksort, hashtable, array, and binary tree can be useful to creating software. When a programme is designed, it relies on the API. For instance, if a user is designing a Microsoft Windows desktop application, he/she might use the .NET Windows Forms library to design the desktop application and call its APIs like *Form1.Close()* and *Form1.Show()* to close or open the application and write the additional operations him/herself that it need to have. Without these APIs, the programmer needs to write these APIs him/herself. Companies like Sun Microsystems, Novell, and Microsoft provide their own APIs so that many applications are written using their software libraries that usually have numerous APIs in them.

Computer software has special economic characteristics that make its design, creation, and distribution different from most other economic goods. A person who creates software is called a programmer, software engineer, software developer, or code monkey, terms that all have a similar meaning.

Industry and Organizations

A great variety of software companies and programmers in the world comprise a software industry . Software can be quite a profitable industry: Bill Gates, the founder of Microsoft was the richest person in the world in 2009 largely by selling the Microsoft Windows and Microsoft Office software products. The same goes for Larry Ellison, largely through his Oracle database software. Through time the software industry has become increasingly specialized. Non-profit software organizations include the Free Software Foundation, GNU Project and Mozilla Foundation. Software standard organizations like the W3C, IETF develop software standards so that most software can interoperate through standards such as XML, HTML, HTTP or FTP. Other well-known large software companies include Novell, SAP, Symantec, Adobe Systems, and Corel, while small companies often provide innovation.

APPLICATION SOFTWARE

Application software, also known as an application or an “app”, is computer software designed to help the user to perform singular or multiple related specific tasks. Examples include enterprize software, accounting software, office suites, graphics software and media players. Many application programmes deal principally with documents. Application software is contrasted with system software and middleware, which manage and integrate a computer’s capabilities, but typically do not directly apply them in the performance of tasks that benefit the user. A simple, if imperfect, analogy in the world of hardware would be the relationship of an electric light bulb (an application) to an electric power generation plant (a system). The power station merely generates electricity, not itself of any real use until harnessed to an application like the electric light that performs a service that benefits the user. Application software applies the power of a particular computing platform or system software to a particular purpose. Some apps such as Microsoft Office are available in versions for several different platforms; others have narrower requirements.

Terminology

In information technology, an application is a computer programme designed to help people perform an activity. An application thus differs from an operating system (which runs a computer), a utility (which performs maintenance or general-purpose chores), and a programming language (with which computer programmes are created). Depending on the activity for which it was designed, an application can manipulate text, numbers, graphics, or a combination of these elements. Some application packages offer considerable computing power by focusing on a single task, such as word processing; others, called integrated software, offer somewhat less power but include several applications. User-written software tailors systems to meet the user's specific needs. User-written software include spreadsheet templates, word processor macros, scientific simulations, graphics and animation scripts. Even email filters are a kind of user software. Users create this software themselves and often overlook how important it is.

The delineation between system software such as operating systems and application software is not exact, however, and is occasionally the object of controversy. For example, one of the key questions in the United States v. Microsoft antitrust trial was whether Microsoft's Internet Explorer web browser was part of its Windows operating system or a separable piece of application software. As another example, the GNU/Linux naming controversy is, in part, due to disagreement about the relationship between the Linux kernel and the operating systems built over this kernel. In some types of embedded systems, the application software and the operating system software may be indistinguishable to the user, as in the case of software used to control a VCR, DVD player or microwave oven. The above definitions may exclude some applications that may exist on some computers in large organizations. For an alternative definition of an app: *see Application Portfolio Management.*

Application Software Classification

Application software falls into two general categories; horizontal applications and vertical applications. Horizontal Application are the most popular and its widely spread in departments or companies. Vertical Applications are designed for a particular type of business or for specific division in a company. There are many types of application software:

- An *application suite* consists of multiple applications bundled together. They usually have related functions, features and user interfaces, and may be able to interact with each other, e.g. open each other's files. Business applications often come in suites, e.g. Microsoft Office, OpenOffice.org and iWork, which bundle together a word processor, a spreadsheet, etc.; but suites exist for other purposes, e.g. graphics or music.
- *Enterprize software* addresses the needs of organization processes and data flow, often in a large distributed environment. (Examples include financial systems, customer relationship management (CRM) systems and supply-chain management software). Note that Departmental Software is a sub-type of Enterprize Software with a focus on smaller organizations or groups within a large organization.

(Examples include Travel Expense Management and IT Helpdesk)

- *Enterprize infrastructure software* provides common capabilities needed to support enterprize software systems. (Examples include databases, email servers, and systems for managing networks and security.)
- *Information worker software* addresses the needs of individuals to create and manage information, often for individual projects within a department, in contrast to enterprize management. Examples include time management, resource management, documentation tools, analytical, and collaborative. Word processors, spreadsheets, email and blog clients, personal information system, and individual media editors may aid in multiple information worker tasks.
- *Content access software* is software used primarily to access content without editing, but may include software that allows for content editing. Such software addresses the needs of individuals and groups to consume digital entertainment and published digital content. (Examples include Media Players, Web Browsers, Help browsers and Games)
- *Educational software* is related to content access software, but has the content and/or features adapted for use in by educators or students. For example, it may deliver evaluations (tests), track progress through material, or include collaborative capabilities.
- *Simulation software* are computer software for simulation of physical or abstract systems for either research, training or entertainment purposes.

- *Media development software* addresses the needs of individuals who generate print and electronic media for others to consume, most often in a commercial or educational setting. This includes Graphic Art software, Desktop Publishing software, Multimedia Development software, HTML editors, Digital Animation editors, Digital Audio and Video composition, and many others.
- *Mobile applications* run on hand-held devices such as mobile phones, personal digital assistants and enterprize digital assistants : see mobile application development.
- *Product engineering software* is used in developing hardware and software products. This includes computer aided design (CAD), computer aided engineering (CAE), computer language editing and compiling tools, Integrated Development Environments, and Application Programmer Interfaces.
- A command-driven interface is one in which you type in commands to make the computer do something. You have to know the commands and what they do and they have to be typed correctly. DOS and Unix are examples of command-driven interfaces.
- A graphical user interface (GUI) is one in which you select command choices from various menus, buttons and icons using a mouse. It is a user-friendly interface. The Windows and Mac OS are both graphical user interfaces.

Applications can also be classified by computing platform.

6

Computer-Based Serials System

Basically, designing a computer-based serials system follows the prescribed pattern: the establishment of requirements, the provision of forms and the determination of the manual and machine procedures that will satisfy the requirements. In fact, the *primary* requirement of the serials system is to provide users with an accurate and current file of the library's serials holdings including all the necessary bibliographic information. In the instance we shall discuss, the file will be a union catalogue containing the data required on all serials in both the general library and the branch libraries.

The data in this file should be available not only in the general library but also in branches and to other possible users. *Second,* the system should be able to provide the means whereby the user as well as the library staff may request and receive within a reasonable time special catalogues or reports of the serials holdings by such categories as call number, language, and subject; and it should also provide special listings by publishers, vendors, types of serials, status (such as active, inactive), and by all condition of receipt (gift, purchase, or exchange). *Third,* the system should be able to guarantee that all issues not received by the library will be claimed promptly and on a set schedule and also that the renewal of all serial subscriptions will be controlled by the system. *Fourth*, the system should be able to predict the time of receipt of a journal issue with accuracy thereby establishing a positive control over the timely receipt of all issues. *Fifth,* the system must be able to maintain and control all the accounting records necessary for its own internal use and be able to report to the comptroller the information required for payment of invoices. *Sixth,* the system should be able to provide the binding department with necessary information and records for those journals that are to be bound. It also should be able to provide such information and records on a scheduled basis in order to maintain an even workload for the binding staff. *Seventh,* the serials system should have the ability to handle the many changes that occur in the various serials titles.

SERIAL RECORD LOAD SYSTEM

The next phase of the serials automation system consists of designing the programme required initially to load and format the serials records. Serial Record Load System, illustrates the generalized system. Requirements of the system are : (i) to produce a report for verification of each serials record loaded to date and arranged in Dewey call-number sequence; (ii) to produce a report for verification of each serials record loaded in the current run; and (iii) to produce a report of any record or records containing erroneous information (errors, duplicated records).

Inputs to the system are : (i) a paper tape containing new serials records; and (ii) A paper tape containing corrections, additions, and deletions to be made in existing computer records. The first requirement of the programme is to produce a report listing each error found in the editing procedure. Based on the editing report the data processing operator will correct the errors (as described previously) and will re-enter the corrections into the computer in a later run of the translate and edit programme. The second requirement is to assemble the input data into proper format as developed for the master serials record. Master Serials Record, shows the layout of the computer record which consists of five sections. The first section of the record is fixed in length and contains the majority of the data characterizes as "basic identification and control data."

The second section is fixed in length and contains the accounting data including a five-year historical record of the payment of each serial subscription. (A five-year average of the increase in serials cost should permit forecasting of serials budgets with a high degree of reliability.) The third section of the master serials record, variable in length, contains bibliographical information on each title. This includes the official entry, all subject entries, selected added entries, title changes, and all notes. Each of these items is variable in length and each field, except the official entry field, may contain more than one entry. For example, there might be two or three subject entries for a particular title.

The fourth section, variable in length, is the claiming and receipt section. This portion of the record is generated internally by the check-in programme at the time it forecasts the receipt of each issue of a title. The fifth section, variable in length, contains the binding information, the greater portion of which is created by the check-in programme at the time the updated information is received by the computer.

Thus, the inputs to the programme are (i) paper tape of new serials records; and (ii) paper tape of correction records. Output of the programme are : (i) serials transactions file on magnetic tape containing the new titles that have been loaded during a given run plus corrections to previous titles that were loaded at an earlier date and (ii) the editing report indicating what errors the computer has found during the edit process. Functions of the programme are : (i) to read in the data from the paper-tape reader (since the codes that are produced by the paper-tape typewriter are not the same as the code structure used in the computer, the characters on this tape must be translated into the appropriate machine language); and (ii) to edit each field of the serials record to make sure that it conforms to the requirements of length and content of the field and that the information as presented is valid. For example, under the area LIB (library) it is possible that the operator type a code that is not a valid departmental code— or in typing the MAIN ENTRY an invalid character was used or for an active serial the FREQ (frequency) code was omitted. Thus each item on the form, where it is possible, will be edited for validity and completeness. The requirement of the programme is to sort the new serial record file into Dewey call-number sequence. Input to the programme is the serials transactions file produced in the translate and edit programme and containing new and corrected serials records. Output of the programme is a sorted serials transactions file containing new and corrected serials records in Dewey call-number sequence. Function of the programme is to sort the records contained in the magnetic-tape files into a desired order or sequence.

Requirements of the programme are : (i) to generate a new master serials record file; (ii) to make corrections to existing records; and (iii) to eliminate duplicate records. Inputs to the programme are : (i) sorted serials transactions file of new and corrected serials record; and (ii) a sorted master file of existing serials records.

Outputs of the programme are:

(i) a magnetic tape file containing all records, existing and new,

(ii) a printed listing of new and corrected titles loaded in this run, and

(iii) a printed listing of erroneous records.

Functions of the programme are: (i) to merge old and new records on the two input tapes into one output master file; and (ii) to correct any record in the old tape file prior to generating the new tape file.

CATALOGUE REPORT GENERATOR SYSTEM

The catalogue report generator system allows flexibility to the library staff and to the user by obtaining information from the system in the sequence and in the format wanted. This system is a medium by which users can exploit the informational potentialities of the serials control system. Generalized Systems Chart of the Catalogue Report Generator System, outlines the programmes and their interrelationships required in the preparation of specified printed catalogues and reports. Requirements of the system are : (i) to provide the user with an accurate and current catalogue of the serials holdings containing the necessary bibliographic information; Serials Catalogue) and (ii) to provide the means whereby the user as well as staff members may request and receive special catalogues or reports of the serials holdings within a reasonable time.

Serials Listing by Subject; Serials Listing by Call No; Serials Listing Departmental Libraries- Arch). Inputs to the system are: (i) the master magnetic tape file of serial records and (ii) the data control punched cards for selection of data and desired sequence and format of the report. Outputs of the system are printed catalogues or reports. When printout of these catalogues or reports is required, it is only necessary for the serials librarian to indicate this by the appropriate code in a control card. The generator system based on this code will automatically select from the master record the data required for the printing of a given list.

The requirement of the programme is selection of desired information from the master file by means of the data control cards. Inputs of the programme are : (i) a the master file of serials records and (b) data control card for each item of information that is to appear in a given report. The output of the programme is a magnetic-tape file containing information for each of the requested reports, or catalogues, or both. Functions of the programme are : (i) stripping from the master serials record file the data required for the printing of catalogues or special reports as specified by the data control cards; and (ii) formatting such information and transferring the resulting record to an output work tape.

The requirement of this programme is to sort each of the report work tapes into desired sequence specified by the sort control card. Inputs to the programme are : (i) the report work tapes created by the catalogue report generator programme; and (ii) the sort control cards that define the order in which records are to appear. The output of the programme is a file of the records to be printed as a report or catalogue.

The requirement of the programme is to print in desired format any standard catalogues or special reports. Inputs to the programme are : (i) the sorted report file; and (ii) the print control card that defines the format in which the information is to appear.

OPERATING SYSTEMS

So far we have described in some detail the systems for compiling the original input to the serials system for setting up the master record file and for correcting this file and for setting up the reporting system required to produce the necessary reports or catalogues.

The primary purpose of the serials operating systems is to provide efficient and accurate control of all records and the transactions.

Systems charts of the monthly and weekly operating systems, outline the programmes and their interrelationships that are necessary to fulfil the following requirements:

1. Predict the time of receipt of the journal thereby establishing a positive control over the timely receipt of all issues.
2. Guarantee that all issues not received by the library will be claimed promptly and on a set schedule and that the renewal of all serials will be controlled by the system.
3. Maintain and control the accounting records necessary for the serials system internally as well as report to the comptroller the information required for payment of invoices.
4. Handle the many changes that occur in the various serials titles.
5. Provide the binding department with the necessary information and records for those titles that are to be bound.

In fact, Outputs of the operating systems are : (i) an updated master file of all serials records; (ii) punched control cards anticipating receipt of issues; (iii) reports for controlling, claiming, renewals, and binding; and (iv) statistical and accounting reports. The operated systems work within three time frames: monthly, weekly, and daily. We shall now review functions that are performed by both the computer and the clerical.

OPERATING FUNCTIONS

This section describes the function of the staff of the serials system and the data processing operator in maintaining records.

1. *Check-in:* When an issue is received, the serials clerk removes from the punched-card file the corresponding check-in card for the particular volume and issue. In the case of an irregular publication it is necessary to record on the check-in card (if not there already) the volume and issue number, adding the month and year of publication. The clerk marks the call number on the issue and checks for changes such as title and frequency in the journal. If variations are not found, the check-in card and is forwarded to the data processing operator.

   If changes have occurred, they are recorded on the serials control record worksheet, indicating the call number and the changes in the appropriate fields. The check-in card and the worksheet are forwarded to the data processing operator who, using the auxiliary punched-card reader attached to the paper-tape typewriter, enters the check-in card data into the paper tape.

   The call number, issue, month, and year are punched into the paper tape as input to the weekly run of the check-in programme. If corrections or changes to a given title are made, the data processing operator holds these until the end of the day or week when the correction tape for input to the translate and edit programme is produced.

2. *Ordering of New Serial Titles* : On receipt of the first issue of a new title the serials librarian prepares a

serials control record worksheet that is forwarded to the data processing operator for entry of the new title into the system.

3. *Renewal of Subscriptions* : The renewal control cards and the renewal reports are received from the computer. The renewal punched card is filed. The renewal report is verified and two copies forwarded to the supplier or the publisher for the renewal of the indicated titles. When the invoice, the invoice is received, the renewal control card for each title on the invoice, the invoice number, and the amount of the renewal are posted to the renewal control card.

   The renewal control report is checked for each title and the renewal control card is then forwarded to the data processing operator. The operator duplicates the prepunched information on the renewal control card and key punches the new information (invoice date and number and renewal cost) and simultaneously produces a punched paper tape with this information. This is accomplished by using a key punch that is cabled to the paper-tape typewriter.

4. *Binding Control* : On receipt of the binding control list and the control cards the binding department uses the control list as its worksheet for the coming month. The journals are taken from the shelves and checked for completeness. Binding orders are then prepared for the completed volumes. It is the responsibility of the binding clerk to submit correction reports for any issues or indexes found missing.

5. *Claiming of Serial Issues*: The weekly run of the check-in programme prints out the claim notices. These notices are reviewed prior to mailing because notice may have been received from the supplier or publisher that item will be delayed. In such event the computer record is updated. If the claim is routine, the notice is forwarded to the supplier. On receipt of the claimed issue the check-in card for that issue is removed from the file and forwarded to the data processing operator.

The monthly run of the check-in programme is the key to the successful fulfilment of the operating system. It is the function of this programme to forecast the transactions that are to occur within the serials system for the coming month and to produce the reports and control cards that are needed to handle successfully these transactions. The monthly run of the check-in programme has specific functions that must be performed.

1. A forecasting must be made of the issues to be received for each of the active during the coming month. Utilizing the FREQ (frequency) data in conjunction with the months of publication data (JFM....field) the programme will determine whether an issue is to be received or claimed; the check-in programme will determine, when possible, the volume and issue numbers to received in the coming period. A control card will be punched for each issue to be received. Simultaneously the programme will post in the waiting receipt and claims section of the master record the control data for each particular issue. The check-in cards will be filed in the serials department awaiting receipt of the issue.

2. A listing must be produced of all titles whose subscriptions are to be renewed during the second month following the monthly run. (For example, during the June run the computer will forecast those titles to be renewed during August of that year.) This programme determines the renewal date by utilizing the date of last renewal found in the DATE SUBS START (date subscription start field and the period of the last subscription from the SUBS LGT (subscription length) field, both found in the accounting data section of the master record. The renewal report is produced by vendor or by publisher showing the titles that should be renewed in the period. The report contains the call number, the title, the last renewal date, the subscription length, and the estimated price of renewal. At this point, programme also produces a control card for each item being renewed. The control card contains the call number and short title. The programme also posts to the accounting data control section the renewal cost incurred.
3. A binding report is necessary for those titles with sufficient consecutive issues to warrant binding into a physical volume. A control card is produced for each volume to be bound and an indication is made in the binding control section of the master record that an order has been issued for binding.
4. Accounting reports are required for the financial control of the system. These reports are based on encumbrances incurred by each fund and department account. In order to obtain a net balance for each fund

and department account at the end of each month the encumbrances are adjusted to actual cost on receipt of invoices.

5. The programme must generate monthly statistics of the number of new titles added and their cost, the number of renewals and the estimated cost of renewals, the number of issues received, the number of bound volumes added to the serials collection, and other statistical information.

For all titles frequency of receipt is irregular the check-in programme will, on the receipt of an issue, produce a check-in card for the next arrival. This card contains only the call number and the short title of the issue and, if possible, the volume and the issue number but not the month or year. Remember, at end of each week all input paper tapes for the serials system are forwarded to the computer centre for entry into the weekly run of the check-in programme. These tapes are processed through the translate and edit programme. The output of this translate and edit programme. The output of this sort programme, arranged by call number and type of transaction, is the input to the weekly run of the check-in programme.

The function of the check-in programme is to post all transactions that have occurred to the serials master record file. The primary output of the weekly check-in programme is the claim notices. During the weekly run the programme audits each record, comparing the date of a forecasted arrival in the WAITING RECEIPT sector of the record to the CLAIM CTL (claim control) date. If the issue is overdue, a claim is issued. If a subsequent published issue is received prior to the receipt of an earlier published issue, a claim is issued is received, the WAITING RECEIPT sector of the record is cleared and the issue is posted to the binding control section of the record. The output of the weekly run is the updated serials master control file that is always the source of all printed reports. A critical operation within the serials system is the check-in and claiming of journal issues. There is an alternative to the punched card, paper-tape check-in and claiming operations in the computer-based serials system.

The direct use of the punched from the check-in file as input to the weekly run of the check-in programme could be a more efficient way to update the master serials record file. If the punched cards produced by the computer for the check-in operation contained the required information for check-in, the intermediate step of paper tape might well be eliminated.

If this were done, no translate programme would be necessary because punched cards do not require this. This phase of the weekly programme would only require a card-to-tape utility programme. Another alternative could be that instead of punched cards, a monthly printout listing of expected journal issues could be used. After a journal issue is received, it could be crossed off the printed list, marked, and sent to the shelves. At the end of the month the printout would be sent to the claiming subsystem for processing. Those titles not crossed off would require a claims notice. The initial notice, both to the vendor and the computer record, would be produced on the paper-tape typewriter.

Subsequent claim notices would be issued by the claims programme under computer control. Updating the computer record with information about the issues that have been received would be accomplished by the computer not being notified. If no notice were received, programming would provide for the computer record to be updated automatically on the assumption that the issue had been received.

7

Computer Systems Architecture

The discipline that defines the conceptual structure and functional behaviour of a computer system. It is analogous to the architecture of a building, determining the overall organization, the attributes of the component parts, and how these parts are combined. It is related to, but different from, computer implementation. Architecture consists of those characteristics which affect the design and development of software programs, whereas implementation focuses on those characteristics which determine the relative cost and performance of the system. The architect's main goal has long been to produce a computer that is as fast as possible, within a given set of cost constraints. Over the years, other goals have been added, such as making it easier to run multiple programs concurrently or improving the performance of programs written in higher-level languages.

A computer system consists of four major components: storage, processor, peripherals, and input/output (communication). The storage system is used to keep data and programs; the processor is the unit that controls the operation of the system and carries out various computations; the peripheral devices are used to communicate with the outside world; and the input/output system allows the previous components to communicate with one another.

STORAGE

The storage or memory of a computer system holds the data that the computer will process and the instructions that indicate what processing is to be done. In a digital computer, these are stored in a form known as binary, which means that each datum or instruction is represented by a series of bits. Bits are conceptually combined into larger units called bytes (usually 8 bits each) and words (usually 8 to 64 bits each). A computer will generally have several different kinds of storage devices, each organized to hold one or more words of data. These types include registers, main memory, and secondary or auxiliary storage.

Registers are the fastest and most costly storage units in a computer. Normally contained within the processing unit, registers hold data that are involved with the computation currently being performed.

Main memory holds the data to be processed and the instructions that specify what processing is to be done. A major goal of the computer architect is to increase the effective speed and size of a memory system without incurring a large cost penalty. Two prevalent techniques for increasing effective speed are interleaving and caching, while virtual memory is a popular way to increase the effective size. Interleaving involves the use of two or more independent memory systems, combined in a way that makes them appear to be a single, faster system. With caching, a small, fast memory system contains the most frequently used words from a slower, larger main memory.

Virtual memory is a technique whereby the programmer is given the illusion of a very large main memory, when in fact it has only a modest size. This is achieved by placing the contents of the large, "virtual" memory on a large but slow auxiliary storage device, and bringing portions of it into main memory, as required by the programs, in a way that is transparent to the programmer.

Auxiliary memory (sometimes called secondary storage) is the slowest, lowest-cost, and highest-capacity computer storage area. Programs and data are kept in auxiliary memory when not in immediate use, so that auxiliary memory is essentially a long-term storage medium. There are two basic types of secondary storage: sequential and direct-access. Sequential-access secondary storage devices, of which magnetic tape is the most common, permit data to be accessed in a linear sequence. A direct-access device is one whose data may be accessed in any order. Disks and drums are the most commonly encountered devices of this type.

Memory mapping is one of the most important aspects of modern computer memory designs. In order to understand its function, the concept of an address space must be considered. When a program resides in a computer's main memory, there is a set of memory cells assigned to the program and its data. This is known as the program's logical address space. The computer's physical address space is the set of memory cells actually contained in the main memory. Memory mapping is simply the method by which the computer translates between the computer's logical and physical address spaces. The most straightforward mapping scheme involves use of a bias register. Assignment of a different bias value to each program in memory enables the programs to coexist without interference.

Another strategy for mapping is known as paging. This technique involves dividing both logical and physical address spaces into equal-sized blocks called pages. Mapping is achieved by means of a page map, which can be thought of as a series of bias registers.

PROCESSING

A computer's processor (processing unit) consists of a control unit, which directs the operation of the system, and an arithmetic and logic unit, which performs computational operations. The design of a processing unit involves selection of a register set, communication paths between these registers, and a means of directing and controlling how these operate. Normally, a processor is directed by a program, which consists of a series of instructions that are kept in main memory.

Although the process of decoding and executing instructions is often carried out by logic circuitry, the complexity of instruction sets can lead to very large and cumbersome circuits for this purpose. To alleviate this problem, a technique known as microprogramming was developed. With microprogramming, each instruction is actually a macrocommand that is carried out by a microprogram, written in a microinstruction language. The microinstructions are very simple, directing data to flow between registers, memories, and arithmetic units.

It should be noted that microprogramming has nothing to do with microprocessors. A microprocessor is a processor implemented through a single, highly integrated circuit.

PERIPHERALS AND COMMUNICATION

A typical computer system includes a variety of peripheral devices such as printers, keyboards, and displays. These devices translate electronic signals into mechanical motion or light (or vice versa) so as to communicate with people.

There are two common approaches for connecting peripherals and secondary storage devices to the rest of the computer: The channel and the bus. A channel is essentially a wire or group of wires between a peripheral device and a memory device. A multiplexed channel allows several devices to be connected to the same wire. A bus is a form of multiplexed channel that can be shared by a large number of devices. The overhead of sharing many devices means that the bus has lower peak performance than a channel; but for a system with many peripherals, the bus is more economical than a large number of channels.

A computer controls the flow of data across buses or channels by means of special instructions and other mechanisms. The simplest scheme is known as program-controlled input/output (I/O). Direct memory access I/O is a technique by which the computer signals the device to transmit a block of data, and the data are transmitted directly to memory, without the processor needing to wait.

Interrupts are a form of signal by which a peripheral device notifies a processor that it has completed transmitting data. This is very helpful in a direct memory access scheme, for the processor cannot always predict in advance how long it will take to transmit a block of data. Architects often design elaborate interrupt schemes to simplify the situation where several peripherals are active simultaneously.

DIGITAL COMPUTER

A device that processes numerical information; more generally, any device that manipulates symbolic information according to specified computational procedures. The term digital computer—or simply, computer—embraces calculators, computer workstations, control computers (controllers) for applications such as domestic appliances and industrial processes, data-processing systems, microcomputers, microcontrollers, multiprocessors, parallel computers, personal computers, network servers, and supercomputers.

A digital computer is an electronic computing machine that uses the binary digits (bits) 0 and 1 to represent all forms of information internally in digital form. Every computer has a set of instructions that define the basic functions it can perform. Sequences of these instructions constitute machine-language programs that can be stored in the computer and used to tailor it to an essentially unlimited number of specialized applications. Calculators are small computers specialized for mathematical computations. General-purpose computers range from pocket-sized personal digital assistants (notepad computers), to medium-sized desktop computers (personal computers and workstations), to large, powerful computers that are shared by many users via a computer network. The vast majority of digital computers now in use are inexpensive, special-purpose microcontrollers that are embedded, often invisibly, in such devices as toys, consumer electronic equipment, and automobiles.

The main data-processing elements of a computer reside in a small number of electronic integrated circuits (ICs) that form a microprocessor or central processing unit (CPU). Electronic technology allows a basic instruction such as "add two numbers" to be executed many millions of times per second. Other electronic devices are used for program and data storage (memory circuits) and for communication with external devices and human users (input-output circuits). Nonelectronic (magnetic, optical, and mechanical) devices also appear in computers. They are used to construct input-output devices such as keyboards, monitors (video screens), secondary memories, printers, sensors, and mechanical actuators.

Information is stored and processed by computers in fixed-sized units called words. Common word sizes are 8, 16, 32, and 64 bits. Four-bit words can be used to encode the first 16 integers. By increasing the word size, the number of different items that can be represented and their precision can be made as large as desired. A common word size in personal computers is 32 bits, which allows 2^{32} = 4,294,967,296 distinct numbers to be represented.

Computer words can represent many different forms of information, not just numbers. For example, 8-bit words called characters or bytes are used to encode text symbols (the 10 decimal digits, the 52 upper-and lowercase letters of the English alphabet, and punctuation marks). A widely used code of this type is ASCII (American Standard Code for Information Interchange). Visual information can be reduced to black and white dots (pixels) corresponding to 0's and 1's. Audio information can be digitized by mapping a small element of sound into a binary word; for example, a compact disk (CD) uses several million 16-bit words to store an audio recording. Logical quantities encountered in reasoning or decision making can be captured by associating 1 with true and 0 with false. Hence, most forms of information are readily reduced to a common, numberlike binary format suitable for processing by computer.

LOGIC COMPONENTS

The operation of a digital computer can be viewed at various levels of abstraction, which are characterized by components of different complexity. These levels range from the low, transistor level seen by an electronic circuit designer to the high, system level seen by a computer user. A useful intermediate level is the logic level, where the basic components process individual bits. By using other basic components called gates, logic circuits can be constructed to perform many useful operations.

SYSTEM ORGANIZATION

An accumulator is a digital system that constitutes a simple processor capable of executing a few instructions. By introducing more data-processing circuits and registers, as well as control circuits for a larger set of instructions, a practical, general-purpose processor can be constructed. Such a processor forms the "brain" of every computer, and is referred to as its central processing unit. A CPU implemented on a single integrated-circuit chip is called a microprocessor.

A typical computer program is too large to store in the CPU, so another component called the main memory is used to store a program's instructions and associated data while they are being executed (Fig. 1). Main memory consists of high-speed integrated circuits designed to allow storage and retrieval of information one word at a time. All words in main memory can be accessed with equal ease; hence this is also called a random-access memory (RAM).

A computer program is processed by loading it into main memory and then transferring its instructions and data one word (or a few words) at a time to the CPU for processing. Hence, there is a continual flow of instructions and data words between the CPU and its main memory. As millions of words must be transferred per second, a high-speed communication link is needed between the CPU and main memory. The system bus fills this role.

A computer has input-output (I/O) control circuits and buses to connect it to external input-output devices (also called peripherals). Typical input-output devices are a keyboard, which is an input device, and a printer, which is an output device. Because most computers need more storage space than main memory can supply, they also employ secondary memory units which form part of the computer's input-output subsystem. Common secondary memory devices are hard disk drives, flexible (floppy) disk drives, and magnetic tape units. Compared to main memory, secondary memories employ storage media (magnetic disks and tapes) that have higher capacity and lower cost. However, secondary memories are also significantly slower than main memory.

No explicit instructions are needed for input-output operations if input-output devices share with main memory the available memory addresses. This is known as memory-mapped input-output, and allows load and store instructions to be used to transfer data between the CPU and input-output devices. In general, a computer's instruction set should include a selection of instructions of the following three types: (1) Data-transfer instructions that move data unchanged between the CPU, main memory, and input-output devices. (2) Data-processing instructions that perform numerical operations such as add, subtract, multiply, and divide, as well as nonnumerical (logical) operations, such as NOT, AND, EXCLUSIVE-OR, and SHIFT. (3) Program-control instructions that can change the order in which instructions are executed, for example branch, branch-on-zero, call procedure, and return from procedure.

The instruction unit (I unit) of a CPU, also called the program control unit, is responsible for fetching instructions from main memory, using the program counter as the instruction address register. The opcode of a newly fetched instruction *I* is placed in the instruction register. The opcode is then decoded to determine the sequence of actions required to execute *I*. These may include the loading or storing of data assigned to main memory, in which case the I unit computes all needed addresses and issues all needed control signals to the CPU and the system bus. Data are processed in the CPU's execution unit (E unit), also called the datapath, which contains a set of registers used for temporary storage of data operands, and an arithmetic logic unit (ALU), which contains the main data-processing circuits.

COMPUTER ARCHITECTURE TOPICS

SUB-DEFINITIONS

Some practitioners of computer architecture at companies such as Intel and AMD use more fine distinctions:

- Macroarchitecture-architectural layers that are more abstract than microarchitecture, e.g. ISA
- ISA (Instruction Set Architecture)-as defined above
- Assembly ISA-a smart assembler may convert an abstract assembly language common to a group of machines into slightly different machine language for different implementations
- Programmer Visible Macroarchitecture-higher level language tools such as compilers may define a consistent interface or contract to programmers using them, abstracting differences between underlying ISA, UISA, and microarchitectures. E.g. the C, C++, or

Java standards define different Programmer Visible Macroarchitecture-although in practice the C microarchitecture for a particular computer includes.

- UISA (Microcode Instruction Set Architecture)-a family of machines with different hardware level microarchitectures may share a common microcode architecture, and hence a UISA.
- Pin Architecture-the set of functions that a microprocessor is expected to provide, from the point of view of a hardware platform. E.g. the x86 A20M, FERR/IGNNE or FLUSH pins, and the messages that the processor is expected to emit after completing a cache invalidation so that external caches can be invalidated. Pin architecture functions are more flexible than ISA functions-external hardware can adapt to changing encodings, or changing from a pin to a message-but the functions are expected to be provided in successive implementations even if the manner of encoding them changes.

DESIGN GOALS

The exact form of a computer system depends on the constraints and goals for which it was optimized. Computer architectures usually trade off standards, cost, memory capacity, latency and throughput. Sometimes other considerations, such as features, size, weight, reliability, expandability and power consumption are factors as well.

The most common scheme carefully chooses the bottleneck that most reduces the computer's speed. Ideally, the cost is allocated proportionally to assure that the data rate is nearly the same for all parts of the computer, with the most costly part being the slowest. This is how skillful commercial integrators optimize personal computers.

PERFORMANCE

Computer performance is often described in terms of clock speed (usually in MHz or GHz). This refers to the cycles per second of the main clock of the CPU. However, this metric is somewhat misleading, as a machine with a higher clock rate may not necessarily have higher performance. As a result manufacturers have moved away from clock speed as a measure of performance.

Computer performance can also be measured with the amount of cache a processor has. If the speed, MHz or GHz, were to be a car then the cache is like the gas tank. No matter how fast the car goes, it will still need to get gas. The higher the speed, and the greater the cache, the faster a processor runs.

Modern CPUs can execute multiple instructions per clock cycle, which dramatically speeds up a program. Other factors influence speed, such as the mix of functional units, bus speeds, available memory, and the type and order of instructions in the programs being run.

There are two main types of speed, latency and throughput. Latency is the time between the start of a process and its completion. Throughput is the amount of work done per unit time. Interrupt latency is the guaranteed maximum response time of the system to an electronic event (*e.g.* when the disk drive finishes moving some data). Performance is affected by a very wide range of design choices — for example, pipelining a processor usually makes latency worse (slower) but makes throughput better. Computers that control machinery usually need low interrupt latencies. These computers operate in a real-time environment and fail if an operation is not completed in a specified amount of time. For example, computer-controlled anti-lock brakes must begin braking almost immediately after they have been instructed to brake.

The performance of a computer can be measured using other metrics, depending upon its application domain. A system may be CPU bound (as in numerical calculation), I/O bound (as in a webserving application) or memory bound (as in video editing). Power consumption has become important in servers and portable devices like laptops.

Benchmarking tries to take all these factors into account by measuring the time a computer takes to run through a series of test programs. Although benchmarking shows strengths, it may not help one to choose a computer. Often the measured machines split on different measures. For example, one system might handle scientific applications quickly, while another might play popular video games more smoothly. Furthermore, designers have been known to add special features to their products, whether in hardware or software, which permit a specific benchmark to execute quickly but which do not offer similar advantages to other, more general tasks.

POWER CONSUMPTION

Power consumption is another design criterion that factors in the design of modern computers. Power efficiency can often be traded for performance or cost benefits. With the increasing power density of modern circuits as the number of transistors per chip scales (Moore's Law), power efficiency has increased in importance. Recent processor designs such as the Intel Core 2 put more emphasis on increasing power efficiency. Also, in the world of embedded computing, power efficiency has long been and remains the primary design goal next to performance.

DIGITAL LOGIC CIRCUIT

DIGITAL ELECTRONICS

Digital electronics are systems that represent signals as discrete levels, rather than as a continuous range. In most cases the number of states is two, and these states are represented by two voltage levels: one near to zero volts and one at a higher level depending on the supply voltage in use. These two levels are often represented as "Low" and "High."

The fundamental advantage of digital techniques stem from the fact it is easier to get an electronic device to switch into one of a number of known states than to accurately reproduce a continuous range of values.

Digital electronics are usually made from large assemblies of logic gates, simple electronic representations of Boolean logic functions.

ADVANTAGES

One advantage of digital circuits when compared to analog circuits is that signals represented digitally can be transmitted without degradation due to noise. For example, a continuous audio signal, transmitted as a sequence of 1s and 0s, can be reconstructed without error provided the noise picked up in transmission is not enough to prevent identification of the 1s and 0s. An hour of music can be stored on a compact disc as about 6 billion binary digits.

In a digital system, a more precise representation of a signal can be obtained by using more binary digits to represent it. While this requires more digital circuits to process the signals, each digit is handled by the same kind of hardware. In an analog system, additional resolution requires fundamental improvements in the linearity and noise charactersitics of each step of the signal chain.

Computer-controlled digital systems can be controlled by software, allowing new functions to be added without changing hardware. Often this can be done outside of the factory by updating the product's software. So, the product's design errors can be corrected after the product is in a customer's hands.

Information storage can be easier in digital systems than in analog ones. The noise-immunity of digital systems permits data to be stored and retrieved without degradation. In an analog system, noise from aging and wear degrade the information stored. In a digital system, as long as the total noise is below a certain level, the information can be recovered perfectly.

DISADVANTAGES

In some cases, digital circuits use more energy than analog circuits to accomplish the same tasks, thus producing more heat. In portable or battery-powered systems this can limit use of digital systems.

For example, battery-powered cellular telephones often use a low-power analog front-end to amplify and tune in the radio signals from the base station. However, a base station has grid power and can use power-hungry, but very flexible software radios. Such base stations can be easily reprogrammed to process the signals used in new cellular standards.

Digital circuits are sometimes more expensive, especially in small quantities.

The sensed world is analog, and signals from this world are analog quantities. For example, light, temperature, sound, electrical conductivity, electric and magnetic fields are analog. Most useful digital systems must translate from continuous analog signals to discrete digital signals. This causes quantization errors.

Quantization error can be reduced if the system stores enough digital data to represent the signal to the desired degree of fidelity. The Nyquist-Shannon sampling theorem provides an important guideline as to how much digital data is needed to accurately portray a given analog signal.

In some systems, if a single piece of digital data is lost or misinterpreted, the meaning of large blocks of related data can completely change. Because of the cliff effect, it can be difficult for users to tell if a particular system is right on the edge of failure, or if it can tolerate much more noise before failing.

Digital fragility can be reduced by designing a digital system for robustness. For example, a parity bit or other error management method can be inserted into the signal path. These schemes help the system detect errors, and then either correct the errors, or at least ask for a new copy of the data. In a state-machine, the state transition logic can be designed to catch unused states and trigger a reset sequence or other error recovery routine.

Embedded software designs that employ Immunity Aware Programming, such as the practice of filling unused program memory with interrupt instructions that point to an error recovery routine. This helps guard against failures that corrupt the microcontroller's instruction pointer which could otherwise cause random code to be executed.

Digital memory and transmission systems can use techniques such as error detection and correction to use additional data to correct any errors in transmission and storage.

On the other hand, some techniques used in digital systems make those systems more vulnerable to single-bit errors. These techniques are acceptable when the underlying bits are reliable enough that such errors are highly unlikely.

A single-bit error in audio data stored directly as linear pulse code modulation (such as on a CD-ROM) causes, at worst, a single click. Instead, many people use audio compression to save storage space and download time, even though a single-bit error may corrupt the entire song.

ANALOG ISSUES IN DIGITAL CIRCUITS

Digital circuits are made from analog components. The design must assure that the analog nature of the components doesn't dominate the desired digital behaviour. Digital systems must manage noise and timing margins, parasitic inductances and capacitances, and filter power connections.

Bad designs have intermittent problems such as "glitches", vanishingly-fast pulses that may trigger some logic but not others, "runt pulses" that do not reach valid "threshold" voltages, or unexpected ("undecoded") combinations of logic states.

Since digital circuits are made from analog components, digital circuits calculate more slowly than low-precision analog circuits that use a similar amount of space and power. However, the digital circuit will calculate more repeatably, because of its high noise immunity. On the other hand, in the high-precision domain (for example, where 14 or more bits of precision are needed), analog circuits require much more power and area than digital equivalents.

CONSTRUCTION

A digital circuit is often constructed from small electronic circuits called logic gates. Each logic gate represents a function of boolean logic. A logic gate is an arrangement of electrically controlled switches.

The output of a logic gate is an electrical flow or voltage, that can, in turn, control more logic gates. Logic gates often use the fewest number of transistors in order to reduce their size, power consumption and cost, and increase their reliability.

Integrated circuits are the least expensive way to make logic gates in large volumes. Integrated circuits are usually designed by engineers using electronic design automation software.

Another form of digital circuit is constructed from lookup tables, (many sold as “programmable logic devices”, though other kinds of PLDs exist). Lookup tables can perform the same functions as machines based on logic gates, but can be easily reprogrammed without changing the wiring. This means that a designer can often repair design errors without changing the arrangement of wires. Therefore, in small volume products, programmable logic devices are often the preferred solution. They are usually designed by engineers using electronic design automation software.

When the volumes are medium to large, and the logic can be slow, or involves complex algorithms or sequences, often a small microcontroller is programmed to make an embedded system. These are usually programmed by software engineers.

When only one digital circuit is needed, and its design is totally customized, as for a factory production line controller, the conventional solution is a programmable logic controller, or PLC. These are usually programmed by electricians, using ladder logic.

STRUCTURE OF DIGITAL SYSTEMS

Engineers use many methods to minimize logic functions, in order to reduce the circuit’s complexity. When the complexity is less, the circuit also has fewer errors and less electronics, and is therefore less expensive.

The most widely used simplification is a minimization algorithm like the Espresso heuristic logic minimizer within a CAD system, although historically, binary decision diagrams, an automated Quine–McCluskey algorithm, truth tables, Karnaugh Maps, and Boolean algebra have been used.

Representations are crucial to an engineer's design of digital circuits. Some analysis methods only work with particular representations.

The classical way to represent a digital circuit is with an equivalent set of logic gates. Another way, often with the least electronics, is to construct an equivalent system of electronic switches (usually transistors). One of the easiest ways is to simply have a memory containing a truth table. The inputs are fed into the address of the memory, and the data outputs of the memory become the outputs.

For automated analysis, these representations have digital file formats that can be processed by computer programs. Most digital engineers are very careful to select computer programs ("tools") with compatible file formats.

To choose representations, engineers consider types of digital systems. Most digital systems divide into "combinational systems" and "sequential systems." A combinational system always presents the same output when given the same inputs. It is basically a representation of a set of logic functions, as already discussed.

A sequential system is a combinational system with some of the outputs fed back as inputs. This makes the digital machine perform a "sequence" of operations. The simplest sequential system is probably a flip flop, a mechanism that represents a binary digit or "bit".

Sequential systems are often designed as state machines. In this way, engineers can design a system's gross behaviour, and even test it in a simulation, without considering all the details of the logic functions.

Sequential systems divide into two further subcategories. "Synchronous" sequential systems change state all at once, when a "clock" signal changes state. "Asynchronous" sequential systems propagate changes whenever inputs change. Synchronous sequential systems are made of well-characterized asynchronous circuits such as flip-flops, that change only when the clock changes, and which have carefully designed timing margins.

The usual way to implement a synchronous sequential state machine is divide it into a piece of combinational logic and a set of flip flops called a "state register." Each time a clock signal ticks, the state register captures the feedback generated from the previous state of the combinational logic, and feeds it back as an unchanging input to the combinational part of the state machine. The fastest rate of the clock is set by the most time-consuming logic calculation in the combinational logic.

The state register is just a representation of a binary number. If the states in the state machine are numbered (easy to arrange), the logic function is some combinational logic that produces the number of the next state.

In comparison, asynchronous systems are very hard to design because all possible states, in all possible timings must be considered. The usual method is to construct a table of the minimum and maximum time that each such state can exist, and then adjust the circuit to minimize the number of such states, and force the circuit to periodically wait for all of its parts to enter a compatible state. (This is called "self-resynchronization.") Without such careful design, it is easy to accidentally produce asynchronous logic that is "unstable", that is, real electronics will have unpredictable results because of the cumulative delays caused by small variations in the values of the electronic components. Certain circuits (such as the synchronizer flip-flops, switch debouncers, and the like which allow external unsynchronized signals to enter synchronous logic circuits) are inherently asynchronous in their design and must be analyzed as such.

As of 2005, almost all digital machines are synchronous designs because it is much easier to create and verify a synchronous design—the software currently used to simulate digital machines does not yet handle asynchronous designs. However, asynchronous logic is thought to be superior, if it can be made to work, because its speed is not constrained by an arbitrary clock; instead, it simply runs at the maximum speed permitted by the propagation rates of the logic gates from which it is constructed. Building an asynchronous circuit using faster parts implicitly makes the circuit "go" faster.

More generally, many digital systems are data flow machines. These are usually designed using synchronous register transfer logic, using hardware description languages such as VHDL or Verilog.

In register transfer logic, binary numbers are stored in groups of flip flops called registers. The outputs of each register are a bundle of wires called a “bus” that carries that number to other calculations. A calculation is simply a piece of combinational logic. Each calculation also has an output bus, and these may be connected to the inputs of several registers. Sometimes a register will have a multiplexer on its input, so that it can store a number from any one of several buses. Alternatively, the outputs of several items may be connected to a bus through buffers that can turn off the output of all of the devices except one. A sequential state machine controls when each register accepts new data from its input.

In the 1980s, some researchers discovered that almost all synchronous register-transfer machines could be converted to asynchronous designs by using first-in-first-out synchronization logic. In this scheme, the digital machine is characterized as a set of data flows. In each step of the flow, an asynchronous “synchronization circuit” determines when the outputs of that step are valid, and presents a signal that says, “grab the data” to the stages that use that stage’s inputs. It turns out that just a few relatively simple synchronization circuits are needed.

The most general-purpose register-transfer logic machine is a computer. This is basically an automatic binary abacus. The control unit of a computer is usually designed as a microprogram run by a microsequencer. A microprogram is much like a player-piano roll. Each table entry or “word” of the microprogram commands the state of every bit that controls the computer. The sequencer then counts, and the count addresses the memory or combinational logic machine that contains the microprogram. The bits from the microprogram control the arithmetic logic unit, memory and other parts of the computer, including the microsequencer itself.

In this way, the complex task of designing the controls of a computer is reduced to a simpler task of programming a relatively independent collection of much simpler logic machines.

Computer architecture is a specialized engineering activity that tries to arrange the registers, calculation logic, buses and other parts of the computer in the best way for some purpose. Computer architects have applied large amounts of ingenuity to computer design to reduce the cost and increase the speed and immunity to programming errors of computers. An increasingly common goal is to reduce the power used in a battery-powered computer system, such as a cell-phone. Many computer architects serve an extended apprenticeship as microprogrammers.

"Specialized computers" are usually a conventional computer with a special-purpose microprogram.

AUTOMATED DESIGN TOOLS

To save costly engineering effort, much of the effort of designing large logic machines has been automated. The computer programs are called "electronic design automation tools" or just "EDA." Simple truth table-style descriptions of logic are often optimized with EDA that automatically produces reduced systems of logic gates or smaller lookup tables that still produce the desired outputs. The most common example of this kind of software is the Espresso heuristic logic minimizer.

Most practical algorithms for optimizing large logic systems use algebraic manipulations or binary decision diagrams, and there are promising experiments with genetic algorithms and annealing optimizations.

To automate costly engineering processes, some EDA can take state tables that describe state machines and automatically produce a truth table or a function table for the combinatorial part of a state machine. The state table is a piece of text that lists each state, together with the conditions controlling the transitions between them and the belonging output signals.

It is common for the function tables of such computer-generated state-machines to be optimized with logic-minimization software such as Minilog.

Often, real logic systems are designed as a series of sub-projects, which are combined using a "tool flow." The tool flow is usually a "script," a simplified computer language that can invoke the software design tools in the right order.

Tool flows for large logic systems such as microprocessors can be thousands of commands long, and combine the work of hundreds of engineers.

Writing and debugging tool flows is an established engineering speciality in companies that produce digital designs. The tool flow usually terminates in a detailed computer file or set of files that describe how to physically construct the logic. Often it consists of instructions to draw the transistors and wires on an integrated circuit or a printed circuit board.

Parts of tool flows are “debugged” by verifying the outputs of simulated logic against expected inputs. The test tools take computer files with sets of inputs and outputs, and highlight discrepancies between the simulated behaviour and the expected behaviour.

Once the input data is believed correct, the design itself must still be verified for correctness. Some tool flows verify designs by first producing a design, and then scanning the design to produce compatible input data for the tool flow. If the scanned data matches the input data, then the tool flow has probably not introduced errors.

The functional verification data are usually called “test vectors.” The functional test vectors may be preserved and used in the factory to test that newly constructed logic works correctly. However, functional test patterns don’t discover common fabrication faults. Production tests are often designed by software tools called “test pattern generators.” These generate test vectors by examining the structure of the logic and systematically generating tests for particular faults. This way the fault coverage can closely approach 100%, provided the design is properly made testable.

Once a design exists, and is verified and testable, it often needs to be processed to be manufacturable as well. Modern integrated circuits have features smaller than the wavelength of the light used to expose the photoresist. Manufacturability software adds interference patterns to the exposure masks to eliminate open-circuits, and enhance the masks’ resolution and contrast.

DESIGN FOR TESTABILITY

A large logic machine (say,. with more than a hundred logical variables) can have an astronomical number of possible states. Obviously, in the factory, testing every state is impractical if testing each state takes a microsecond, and there are more states than the number of microseconds since the universe began. Unfortunately, this ridiculous-sounding case is typical.

Fortunately, large logic machines are almost always designed as assemblies of smaller logic machines. To save time, the smaller sub-machines are isolated by permanently-installed "design for test" circuitry, and are tested independently.

One common test scheme known as "scan design" moves test bits serially (one after another) from external test equipment through one or more serial shift registers known as "scan chains". Serial scans have only one or two wires to carry the data, and minimize the physical size and expense of the infrequently-used test logic.

After all the test data bits are in place, the design is reconfigured to be in "normal mode" and one or more clock pulses are applied, to test for faults (e.g. stuck-at low or stuck-at high) and capture the test result into flip-flops and/or latches in the scan shift register(s). Finally, the result of the test is shifted out to the block boundary and compared against the predicted "good machine" result.

In a board-test environment, serial to parallel testing has been formalized with a standard called "JTAG" (named after the "Joint Test Action Group" that proposed it).

Another common testing scheme provides a test mode that forces some part of the logic machine to enter a "test cycle." The test cycle usually exercises large independent parts of the machine.

TRADE-OFFS

Several numbers determine the practicality of a system of digital logic. Engineers explored numerous electronic devices to get an ideal combination of fanout, speed, low cost and reliability.

The cost of a logic gate is crucial. In the 1930s, the earliest digital logic systems were constructed from telephone relays because these were inexpensive and relatively reliable. After that, engineers always used the cheapest available electronic switches that could still fulfil the requirements.

The earliest integrated circuits were a happy accident. They were constructed not to save money, but to save weight, and permit the Apollo Guidance Computer to control an inertial guidance system for a spacecraft. The first integrated circuit logic gates cost nearly $50 (in 1960 dollars, when an engineer earned $10,000/year). To everyone's surprise, by the time the circuits were mass-produced, they had become the least-expensive method of constructing digital logic. Improvements in this technology have driven all subsequent improvements in cost.

With the rise of integrated circuits, reducing the absolute number of chips used represented another way to save costs. The goal of a designer is not just to make the simplest circuit, but to keep the component count down. Sometimes this results in slightly more complicated designs with respect to the underlying digital logic but nevertheless reduces the number of components, board size, and even power consumption.

For example, in some logic families, NAND gates are the simplest digital gate to build. All other logical operations can be implemented by NAND gates. If a circuit already required a single NAND gate, and a single chip normally carried four NAND gates, then the remaining gates could be used to implement other logical operations like logical and. This could eliminate the need for a separate chip containing those different types of gates.

The "reliability" of a logic gate describes its mean time between failure (MTBF). Digital machines often have millions of logic gates. Also, most digital machines are "optimized" to reduce their cost. The result is that often, the failure of a single logic gate will cause a digital machine to stop working.

Digital machines first became useful when the MTBF for a switch got above a few hundred hours. Even so, many of these machines had complex, well-rehearsed repair procedures, and would be nonfunctional for hours because a tube burned-out, or a moth got stuck in a relay. Modern transistorized integrated circuit logic gates have MTBFs of nearly a trillion (1×10^{12}) hours, and need them because they have so many logic gates.

Fanout describes how many logic inputs can be controlled by a single logic output. The minimum practical fanout is about five. Modern electronic logic using CMOS transistors for switches have fanouts near fifty, and can sometimes go much higher. The "switching speed" describes how many times per second an inverter (an electronic representation of a "logical not" function) can change from true to false and back. Faster logic can accomplish more operations in less time. Digital logic first became useful when switching speeds got above fifty hertz, because that was faster than a team of humans operating mechanical calculators. Modern electronic digital logic routinely switches at five gigahertz (5×10^9 hertz), and some laboratory systems switch at more than a terahertz (1×10^{12} hertz).

LOGIC FAMILIES

Design started with relays. Relay logic was relatively inexpensive and reliable, but slow. Occasionally a mechanical failure would occur. Fanouts were typically about ten, limited by the resistance of the coils and arcing on the contacts from high voltages.

Later, vacuum tubes were used. These were very fast, but generated heat, and were unreliable because the filaments would burn out. Fanouts were typically five to seven, limited by the heating from the tubes' current. In the 1950s, special "computer tubes" were developed with filaments that omitted volatile elements like silicon. These ran for hundreds of thousands of hours.

The first semiconductor logic family was Resistor-transistor logic. This was a thousand times more reliable than tubes, ran cooler, and used less power, but had a very low fan-in of three. Diode-transistor logic improved the fanout up to about seven, and reduced the power. Some DTL designs used two power-supplies with alternating layers of NPN and PNP transistors to increase the fanout. Transistor logic (TTL) was a great improvement over these. In early devices, fanout improved to ten, and later variations reliably achieved twenty. TTL was also fast, with some variations achieving switching times as low as twenty nanoseconds. TTL is still used in some designs.

Another contender was emitter coupled logic. This is very fast but uses a lot of power. It's now used mostly in radio-frequency circuits.

Modern integrated circuits mostly use variations of CMOS, which is acceptably fast, very small and uses very little power. Fanouts of forty or more are possible, with some speed penalty.

NON-ELECTRONIC LOGIC

It is possible to construct non-electronic digital mechanisms. In principle, any technology capable of representing discrete states and representing logic operations could be used to build mechanical logic. MIT students Erlyne Gee, Edward Hardebeck, Danny Hillis (co-author of The Connection Machine), Margaret Minsky and brothers Barry and Brian Silverman, built two working computers from Tinker toys, string, a brick, and a sharpened pencil. The Tinkertoy computer is supposed to be in the Houston Museum of Natural Science.

Hydraulic, pneumatic and mechanical versions of logic gates exist and are used in situations where electricity cannot be used. The first two types are considered under the heading of fluidics. One application of fluidic logic is in military hardware that is likely to be exposed to a nuclear electromagnetic pulse (nuclear EMP, or NEMP) that would destroy electrical circuits.

Mechanical logic is frequently used in inexpensive controllers, such as those in washing machines. Famously, the first computer design, by Charles Babbage, was designed to use mechanical logic. Mechanical logic might also be used in very small computers that could be built by nanotechnology.

Another example is that if two particular enzymes are required to prevent the construction of a particular protein, this is the equivalent of a biological “NAND” gate.

RECENT DEVELOPMENTS

The discovery of superconductivity has enabled the development of Rapid Single Flux Quantum (RSFQ) circuit technology, which uses Josephson junctions instead of transistors. Most recently, attempts are being made to construct purely optical computing systems capable of processing digital information using nonlinear optical elements.

BOOLEAN ALGEBRA (LOGIC)

Boolean algebra (or Boolean logic) is a logical calculus of truth values, developed by George Boole in the 1990s. It resembles the algebra of real numbers, but with the numeric operations of multiplication *xy*, addition *x* + *y*, and negation “*x* replaced by the respective logical operations of conjunction *x*’”*y*, disjunction *x*(“*y*, and complement ¬*x*. The Boolean operations are these and all other operations that can be built from these, such as *x*’”(*y*(“*z*). These turn out to coincide with the set of all operations on the set {0, 1} that take only finitely many arguments; there are 2^{2n} such operations when there are n arguments.

The laws of Boolean algebra can be defined axiomatically as certain equations called axioms together with their logical consequences called theorems, or semantically as those equations that are true for every possible assignment of 0 or 1 to their variables. The axiomatic approach is sound and complete in the sense that it proves respectively neither more nor fewer laws than the semantic approach.

VALUES

Boolean algebra is the algebra of two values. These are usually taken to be 0 and 1, as we shall do here, although F and T, false and true, etc. are also in common use. For the purpose of understanding Boolean algebra any Boolean domain of two values will do.

Regardless of nomenclature, the values are customarily thought of as essentially logical in character and are therefore referred to as truth values, in contrast to the natural numbers or the reals which are considered numerical values. On the other hand the algebra of the integers modulo 2, while ostensibly just as numeric as the integers themselves, was shown to constitute exactly Boolean algebra, originally by I.I. Zhegalkin in 1927 and rediscovered independently in the west by Marshall Stone in 1936. So in fact there is some ambiguity in the true nature of Boolean algebra: it can be viewed as either logical or numeric in character.

More generally Boolean algebra is the algebra of values from any Boolean algebra as a model of the laws of Boolean algebra. For example the bit vectors of a given length, as with say 32-bit computer words, can be combined with Boolean operations in the same way as individual bits, thereby forming a 2^{32}-element Boolean algebra under those operations. Any such combination applies the same Boolean operation to all bits simultaneously.

This passage from the Boolean algebra of 0 and 1 to these more general Boolean algebras is the Boolean counterpart of the passage from the algebra of the ring of integers to the algebra of commutative rings in general. The two-element Boolean algebra is the prototypical Boolean algebra in the same sense as the ring of integers is the prototypical commutative ring. Boolean logic as the subject matter of this article is independent of the choice of Boolean algebra (the same equations hold of every nontrivial Boolean algebra); hence, there is no need here to consider any Boolean algebra other than the two-element one. The article on Boolean algebra (structure) treats Boolean algebras themselves.

INTERNAL ORGANIZATION OF A CPU

PERFORMANCE MEASURES

A simple indicator of a CPU's performance is the frequency f of its central timing signal (clock), measured in millions of clock signals issued per second or megahertz (MHz). The clock frequency depends on the integrated-circuit technology used; frequencies of several hundred megahertz are achievable with current technology. Each clock signal triggers execution of a basic instruction such as a fixed-point addition; hence, the time required to execute such an instruction (the clock cycle time) is $1/f$ microseconds.

Complex instructions like multiplication or operations on floating-point numbers require several clock cycles to complete their execution. Another measure of CPU performance is the (average) instruction execution rate, measured in millions of instructions per second (MIPS).

Instruction execution time is strongly affected by the time to move instructions or data between the CPU and main memory. The time required by the CPU to access a word in main memory is typically about five times longer than the CPU's clock cycle time. This disparity in speed has existed since the earliest computers despite efforts to develop memory circuits that would be fast enough to keep up with the fastest CPUs. Maximum performance requires the CPU to be supplied with a steady flow of instructions that need to be executed. This flow is disrupted by branch instructions, which account for 20% or more of the instructions in a typical program.

To deal with the foregoing issues, various performance-enhancing features have been incorporated into the design of computers. The communication bottleneck between the CPU and main memory is reduced by means of a cache, which is a special memory unit inserted between the two units. The cache is smaller than main memory but can be accessed more rapidly, and is often placed on the same integrated-circuit chip as the CPU. Its effect is to reduce the average time required by the CPU to send information to or receive information from the memory subsystem. Special logic circuits support the complex flow of information among main memory, the cache, and the registers of the CPU. However, the cache is largely invisible to the programs being executed. The instruction execution rate can be increased by executing several instructions concurrently. One approach is to employ several E units that are tailored to different instruction types. Examples are an integer unit designed to execute fixed-point instructions and a floating-point unit designed for floating-point instructions. The CPU can then execute a fixed-point instruction and a floating-point instruction at the same time. Processors that execute several instructions in parallel in this way are called superscalar.

Another speedup technique called pipelining allows several instructions to be processed simultaneously in special circuits called pipelines. Execution of an instruction is broken into several consecutive steps, each of which can be assigned to a separate stage of the pipeline. This makes it possible for an *n*-stage E unit to overlap the execution of up to *n* different instructions. A pipeline processing circuit resembles an assembly line on which many products are in various stages of manufacture at the same time. The ability of a CPU to execute several instructions at the same time by using multiple or pipelined E units is highly dependent on the availability of instructions of the right type at the right time in the program being executed. A useful measure of the performance of a CPU that employs internal parallelism is the average number of clock cycles per instruction (CPI) needed to execute a representative set of programs.

CISCS AND RISCS

A software implementation of a complex operation like multiply is slower than the corresponding hardware implementation. Consequently, as advances in IC technology lowered the cost of hardware circuits, instruction sets tended to increase in size and complexity. By the mid-1980s, many microprocessors had instructions of several hundred different types, characterized by diverse formats, memory addressing modes, and execution times. The heterogeneous instruction sets of these complex instruction set computers (CISCs) have some disadvantages. Complex instructions require more processing circuits, which tend to make CISCs large and expensive. Moreover, the decoding and execution of complex instruction can slow down the processing of simple instructions.

To address the defects of CISCs, a new class of fast computers referred to as reduced instruction set computers (RISCs) was introduced. RISCs are characterized by fast, efficient—but not necessarily small—instruction sets. The following features are common to most RISCs: (1) All instructions are of fixed length and have just a few opcode formats and addressing modes. (2) The only instructions that address memory are load and store instructions; all other instructions require their operands to be placed in CPU registers. (3) The fetching and processing of most instructions is overlapped in pipelined fashion. In computer engineering, computer architecture is the conceptual design and fundamental operational structure of a computer system.

It is a blueprint and functional description of requirements and design implementations for the various parts of a computer, focusing largely on the way by which the central processing unit (CPU) performs internally and accesses addresses in memory.

It may also be defined as the science and art of selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals.

COMPUTER ARCHITECTURE COMPRISES AT LEAST THREE MAIN SUBCATEGORIES

- *Instruction set architecture*, or ISA, is the abstract image of a computing system that is seen by a machine language (or assembly language) programmer, including the instruction set, word size, memory address modes, processor registers, and address and data formats.
- *Microarchitecture*, also known as *Computer organization* is a lower level, more concrete and detailed, description of the system that involves how the constituent parts of the system are interconnected and how they interoperate in order to implement the ISA. The size of a computer's cache for instance, is an organizational issue that generally has nothing to do with the ISA.
- *System Design* which includes all of the other hardware components within a computing system such as:
  1. system interconnects such as computer buses and switches
  2. memory controllers and hierarchies
  3. CPU off-load mechanisms such as direct memory access
  4. issues like multi-processing.

Once both ISA and microarchitecture have been specified, the actual device needs to be designed into hardware.

This design process is called *implementation*. Implementation is usually not considered architectural definition, but rather hardware design engineering.

IMPLEMENTATION CAN BE FURTHER BROKEN DOWN INTO THREE (NOT FULLY DISTINCT) PIECES

- Logic Implementation-design of blocks defined in the microarchitecture at (primarily) the register-transfer and gate levels.
- Circuit Implementation-transistor-level design of basic elements (gates, multiplexers, latches etc) as well as of some larger blocks (ALUs, caches etc) that may be implemented at this level, or even (partly) at the physical level, for performance reasons.
- Physical Implementation-physical circuits are drawn out, the different circuit components are placed in a chip floor-plan or on a board and the wires connecting them are routed.

For CPUs, the entire implementation process is often called CPU design.

More specific usages of the term include more general wider-scale hardware architectures, such as cluster computing and Non-Uniform Memory Access (NUMA) architectures.

HISTORY

The term “architecture” in computer literature can be traced to the work of Lyle R. Johnson and Frederick P. Brooks, Jr., members in 1959 of the Machine Organization department in IBM’s main research centre. Johnson had the opportunity to write a proprietary research communication about Stretch, an IBM-developed supercomputer for Los Alamos Scientific Laboratory. In attempting to characterize his chosen level of detail for discussing the luxuriously embellished computer, he noted that his description of formats, instruction types, hardware parameters, and speed enhancements was at the level of "system architecture" – a term that seemed more useful than "machine organization." Subsequently, Brooks, one of the Stretch designers by writing, "Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints."

Brooks went on to play a major role in the development of the IBM System/360 line of computers, where "architecture" gained currency as a noun with the definition "what the user needs to know." Later the computer world would employ the term in many less-explicit ways.

The first mention of the term *architecture* in the referred computer literature is in a 1964 article describing the IBM System/360. The article defines architecture as the set of "attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behaviour, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation."

In the definition, the *programmer* perspective of the computer's functional behaviour is key. The conceptual structure part of an architecture description makes the functional behaviour comprehensible, and extrapolatable to a range of Use cases. Only later on did ‘internals’ such as “the way by which the CPU performs internally and accesses addresses in memory,” mentioned above, slip into the definition of computer architecture.