# Chapter 1

# Computer System Engineering

This chapter covers basic knowledge of the subject. An overview and the perspective of computer system engineering are given. The components and the organisation of computer systems in many levels of abstractions are discussed. The relationship between architecture and computer languages is important and several issues have been addressed. One important aspect of modern computer systems, the performance issue, is discussed. Finally, a brief history of computer is narrated. Computer history itself is a very fascinating subject.

## 1.1 Introduction

A computer system consists of many parts. A part can be divided into subparts and forms a hierarchy. Computer system engineering concerns how to compose these parts to provide a system that has desired functions under various constraints. A computer system has a central processing unit (CPU), memory, input/output, interconnections. A CPU consists of an arithmetic logic unit (ALU), data path, and a control unit. The memory subsystem consists of hierarchical structure: cache memory (high speed memory), main memory, virtual memory. The input/output system consists of various peripherals such as a visual display unit, a keyboard, input devices, an interface to the network, various kinds of secondary storage, bulk memory, a hard disk etc. The interconnections link every parts together, the internal bus, the external bus, I/O channels, ports.

There are many possibilities of choosing and integrating various *components* of a system to satisfy a set of constraints stated in a requirement. A computer system engineer must make decision how to select and integrate various components such as processors, memory, input/output into a computer system. A computer system is driven by the advancement of technology. Various parts of a computer system can be either hardware or software. Hardware and software are interchangeable depending on technology.

## Computation

A computer system performs computation. What is computation? Computation can be defined as *symbols transformation*. It is a process that transforms input symbols to output symbols. *Symbol* is an abstraction. A symbol can represent something in the real world, or it can represent some mathematical object. The real world is connected to a computation by sensors and actuators. A sensor transforms real world events, such as temperature, into symbols that are fed into computation. An actuator transforms symbols from a computation to affect the real world. An actuator such as a motor has effect in the real world. It may change the state of the world. The relationship between computation and the real world can be shown as the figure 1.1.

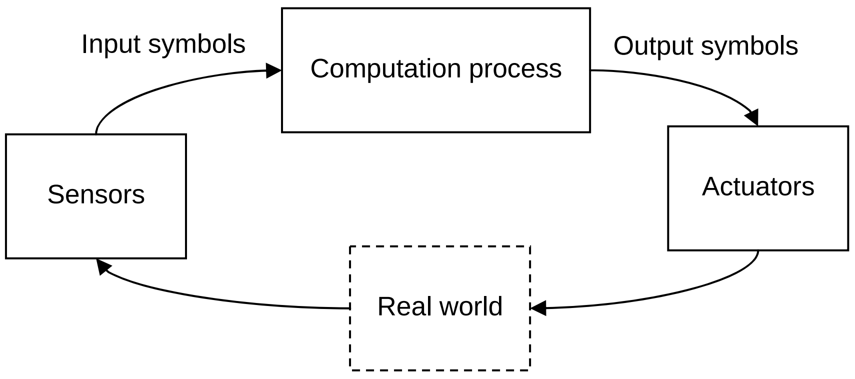


Figure 1.1 Relationship between computation and real world

Software is a specification of a computation. From this point of view, a software does not describe sensors, actuators nor the events in the real world. Hence, it is necessarily incomplete, i.e. it cannot describe the computation plus the real world connected to that computation completely.

## Hardware and Software

The most important property of computer systems is it *programmability*. This property differentiates a computer from all other artifacts. Software is the result of this property. Software as a specification of computation enables a computer to be multi-function, and even adaptive. An application software runs on a computer system. At the bottom level of a computer there are electronic circuits which are called *hardware*. The interface between a program and a hardware is the instruction set. An instruction set defines an abstraction of hardware. This abstraction allows a programmer to *program* a hardware to perform multiple functions.

## Components of a computer

There are many possibilities in realizing a programmable system. The most influencial concept is the *stored program* concept invented by John Von Neumann. In this model, a computer is composed of two parts: processor and memory. Memory stores both data and program. Furthurmore, memory can be accessed directly at any location. This is called *random access model*. Other possible realization of a programmable system includes data flow architecture, systolic architecture etc. We will restrict our study to the stored program concept.

A processor is connected to memory through two ports: address and data (Fig. 1.2). The *access* to memory by a processor is done by sending an address to a memory device then a value can be read or write through the data port. The size of value (measured in the number of bits) that can be accessed is the width of the data. This size defines the bit-size of a processor, such as 8-bit, 16-bit, 32-bit, 64-bit processor.

A processor contains an arithmetic-logic unit (ALU), registers, a program counter (PC), an instruction register (IR) and a countrol unit. An ALU performs arithmetic and logic functions: add, substract, multiply, divide, and, or, not and others. Registers are the fast memory used by a processor to store the intermediate results. A program counter keeps track where the current instruction is. It is changed by instructions that alter the flow of control of a program (if-then-else, loop, and function call in a high level language). An instruction register stores the current instruction fetched from memory. Its content (the instruction) signals the control unit to initiate the execution of that instruction. The control unit sends control signals to all parts in the processor to co-ordinate their activities. The control unit is a large finite state machine. It is the most complex part of a processor.

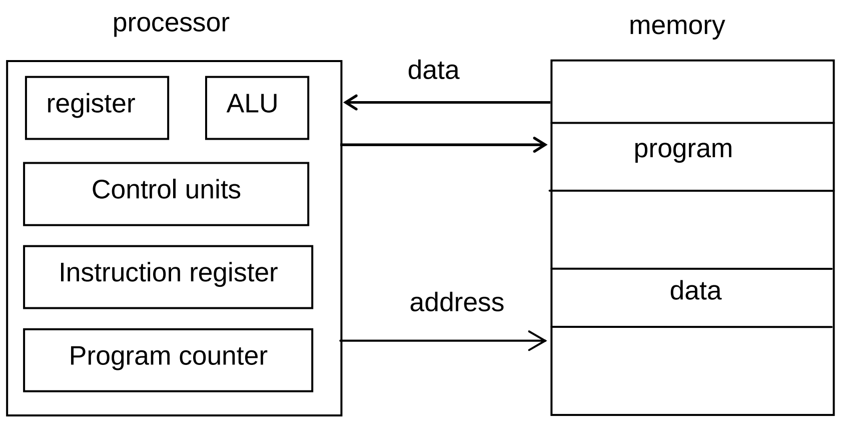


Figure 1.2 Components of a computer

## 1.2 Computer system structure

A computer system can be seen as many level of descriptions, from the applications to the lowest level of electronic circuits. A computer system consists of many parts of which can be regarded as *layers* (Fig. 1.3). These layers are described at different *level of abstraction*. There are many ways to define the level of abstractions. For example, a computer system at the bottom level consists of the actual hardware devices: a central processing unit, a memory, input/output devices and interconnections. These hardware devices can be described at the level of: functional units, finite state machines, logic gates down to the electronic circuits. On top of hardware of the system, an operating system gives services to application programs. The interface between programs and hardware is the instruction set description. A computer system can also be viewed as having two aspects: physical and logical. The *physical* system is composed of the actual physical components. The *logical* system describes the design and the organization.

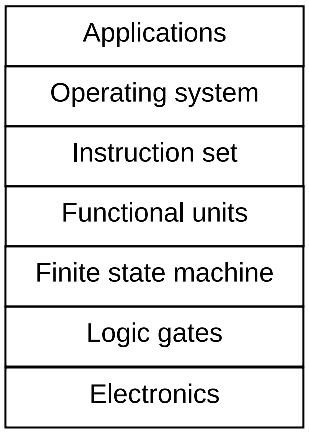


Figure 1.3  The level of description of computer systems

*Application level* is what a user typically sees a computer system, running his/her application programs. An application is usually written in a computer language which used many system functions provided by the operating system. An *operating system* is abstraction layers that separate a user program from the underlying system-dependent hardware and peripherals.

The level of traditional computer architecture begins at the *instruction set*. An instruction set is what a programmer at the lowest level sees of a processor (programming in an assembly language). In the past, instruction set design is at the very heart of computer design. The concept of the family of computer was promoted by IBM around 1970. They proposed the concept of one instruction set with different level of performance for many models. This concept is possible because of the research effort of IBM in using “microprogram” as the method to implement a control unit. However as the present day processor designs converge, their instruction sets become more similar than different. The effort of the designer had turned to other important issues in computer design.

*Finite state machine* description is a mathematical description of the *behaviour* of a system. It is becoming an important tool for verification of the correct behaviour of the hardware during designing of a processor. As a processor becomes more and more complex, a mathematical tool is required in order to guarantee the correct working behaviour since an exhaustive testing is impossible and partial testing is expensive (but still indispensable). Presently it is estimated that more than half of the cost in developing a processor is spent on verifying that the design works according to its specification.

The lower level of *logic gates* and *electronics* describe the logical and actual circuits of a computer system and belongs to the realm of an electrical engineer.

This level of abstraction enables separate layers to be designed and implemented independently. It also provides a high degree of tolerance to changes. A change in one layer has limited effect on other layers. This degree of *decoupling* is important as a computer system is highly changeable and technology-dependent. The changes are very frequent; a new microelectronic fabrication process leads to a higher speed device, a new version of operating system provides more functionality, new applications are created. Without separation into layers all these changes will interact in a complex and uncontrollable way. The level of abstraction is a key concept in designing and implementing a complex system.

## 1.3 Computer hardware

The basic elements are logic gates. A complete set of gates composed of: AND, OR, NOT gates. (This is not the only basis, there are several others). NAND gate (NOR gate) is complete because it can performed the same function as AND, OR, NOT gates. Logic gates are used to build larger *functional units* which are the building blocks of a computer. There are two types of logic gates, one with memory and one without.

A *combinational logic* has no memory; its output is the function of its input only. To create memory, the output is fed back to the input. The resulting circuit is called a sequential logic.

A *sequential logic* is the logic gate with memory. The basic element is called flip-flop. There are many types of flip-flop such as RS, JK, T and D-type flip-flop. A sequential logic has "states". The output depends on both inputs and states. There are two types, synchronous and asynchronous. A *synchronous logic* has a common clock. It is a rule of thumb for design engineers to choose a synchronous logic because it is much simpler to design and to debug. One draw back of synchronous design is that the maximum speed of the clock is determined by the slowest part of the circuit. Therefore it is a worst-case design. An *asynchronous logic* has no central clock, hence it can be much faster than synchronous design when the clock rate is very high and clock skew becomes a problem. The output of one stage is used to drive the next stage. It is difficult to arrange the timing for the circuit to operate properly as the delay of each element affects the timing of the whole circuit. There are large variation of delay when fabricating each logic element. This fact often makes asynchronous design impractical or very expensive.

An example of asynchronous design illustrates the point above. The super computer ILLIAC from the university of Illinois at Urbana-Champaign has asynchronous design to achieve high clock rate [BEL71]. Each connecting wire has to be trimmed manually to properly adjusted the delay time of each module. In the era of VLSI, most design is synchronous because it is much easier to get the design to work properly. Presently due to the advancement of asynchronous design methodology and the promise of very high speed result (and low power consumption) the asynchronous design is coming back. It is an active area of research. There are many standard textbooks on digital logic design which students can explore the subject in much more details such as the one by Katz [KAT93].

In order for a computer to execute a program, many functional units are necessary. Functional units are the building blocks of processors. These building blocks plus the control unit constitute the basic structure of a processor. Basic units to perform arithmetic functions are: adder, multiplier, shifter etc. A functional unit may be built on smaller units, for example, in an adder, a Half adder is built out of basic gates and two Half adders combined into a Full adder. The length of operand affects the speed of adder circuits. The delay comes from the need to propagate the carry bits. *Carry-look-ahead* logic, invented by Charles Babbage [LEE95] who was considered the father of modern computer, is used to speed up the propagation of the carry bits.

## Instruction execution cycle

Instructions reside in memory. This is why this architecture is called *stored program*. Instructions can be accessed from a processor similar to any piece of data in memory. A sequence of instructions is a program. A processor starts running a program by reading instructions from memory and executing them one instruction at a time.

The cycle starts by a processor sending the address of the current instruction to memory via the address bus. The current instruction is read from the memory via the data bus and is stored in the instruction register (IR). IR causes the control unit to co-ordinate activities in the processor to execute that instruction. The processor then starts to read the next instruction (the program counter is increment to point to the next instruction) and executing it and so on.

The result of executing of an instruction can effect many parts: registers, data in the memory, or the program counter. When an instruction changes the program counter, it causes the program to change the flow, either the program entering the loop or selecting the next statement depending on the flags affected by previous instructions.

## Hardware level

A processor consists of a data path and a control unit. A data path contains all the necessary computing elements to carry out a computation task. The control unit sends control signals to harmonise the data flow in the data path so that the desired computation occurs. To give an analogy of a processor to an orchestra, the data path is the musician, the control unit is the conductor.

Components in a data path consist of: logic, register, multiplexer and bus.

- *Logic* is a combinational function, $out = f(in_1 \ldots in_n)$ where $f$ is a Boolean function {not, and, or}. For example

  $$out = \overline{x_1} + x_2 \cdot x_3 + x_1 \cdot \overline{x_2}$$

  Where $\overline{\ }$ denotes not, + denotes or, · denotes and functions. A Boolean expression can be represented as a truth table. An enumeration of all cases of input values to output values. Logic minimisation is a process to realise a desired function with minimum number of logic elements (such as gates). Logic minimisation is an NP-hard problem.

- *Registers* are storage elements, out(t+1) = in(t), with the control signal "load" (the change can be either on the positive or the negative edge of the clock depends on the model). The width of a register defines the number of bits that can be stored.

- *Muliplexors* have *n* inputs (of width *m*) and select one input to be the output, called *n:1* multiplexer. The control signal to determine the output is called the select signal.

- *Bus* consists of wires and buffers. Wires carry data (signal). Buffer controls the traffic of data from any element to a bus. A bus can be shared to reduce the number of wire within a circuit. A bus can broadcast data to many receivers limited by the *fan-out* electrical characteristic of the bus, the ability to drive other circuits.

With these four elements: logic, register, multiplexor and bus, a processor can be built.

## Data path

The simplest data path consists of a *loop* from registers to functional units (logic) and back (Fig. 1.4).

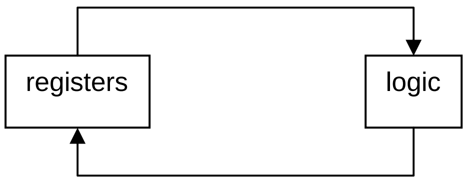


Figure 1.4 A simple data path

For example, suppose there are two registers, named A and B, and an adder. This data path can perform

$$A = A + B$$

with the following control,

1. read two registers into two inputs of the adder.
2. the adder outputs the result of adding its two inputs.
3. the result is written back to a register.

There can be multiple function units working in parallel. The result is more work done in one cycle round the loop. There are complexities involving in doing many tasks concurrently such as competing for the same resource.

## 1.4 How a processor performs computation

Suppose we want to calculate value of a polynomial

$$f(x) = a\,x + b\,x^2$$

The functional units required to do this computation are multipliers and adders. The desired computation can be performed by directly connecting an appropriate number of functional units together (Fig 1.5).

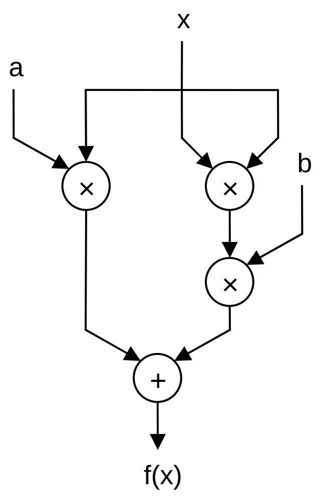


Figure 1.5  A computation graph to evaluate a polynomial

The solution of this computation problem becomes a graph whose nodes are functional units and arcs are connections of data through these units. The computation is performed by the flow of data. In this model every units can be active concurrently. “Programming” in this model becomes specifying the computation graph.

Another way to compute *f*(*x*) is by sequentialise the operations. The required functional units are memory and a general processing unit (Fig. 1.6). A memory stored all the necessary values: input *x*, constant *a*, *b*, temporary places to keep intermediate values *t1*, *t2*, and the final result *f*(*x*). The memory can be read and written to. Two values can be read from memory at once and the data is fed to a general processing unit, so called Arithmetic Logic Unit (ALU).

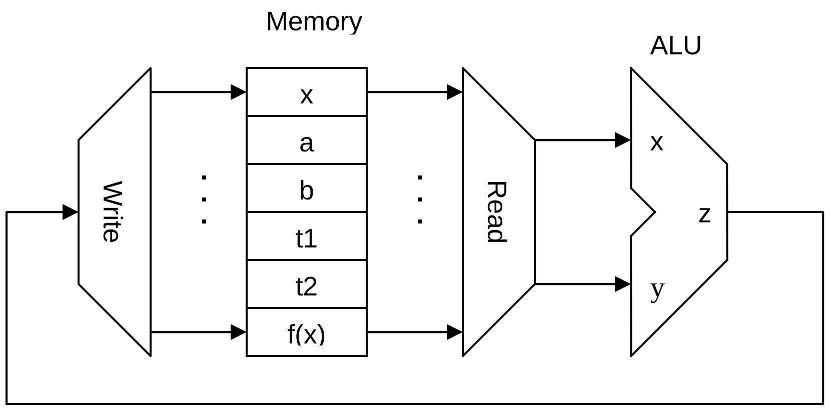


Figure 1.6 A sequential model of computation

The processing unit can perform multiplication and addition. It has internal storage to store two input values and one output value. In general, ALU can do a number of computations. Assume its inputs are *x*, *y*, output *z*, ALU performs *z* = f(*x*, *y*) where *f* = { *add*, *sub*, *mul*, *increment*, ...}. The output of the processing unit (z) is connected to the write port of the memory. Now the desired computation can be performed by executing these steps:

```
read(x,a)
alu(mul)
write(t1)
read(x,x)
alu(mul)
write(t2)
read(t2,b)
alu(mul)
write(t2)
read(t1,t2)
alu(add)
write(result)
```

Sequential approach to computation enables functional units to be reused as the computation is performed step-by-step. The intermediate values can be saved in the memory and they can be used in the later steps. The general processing unit can perform a number of different functions such as add, subtract, so that only one unit is sufficient for most kinds of computation. The trade-off is the speed as the computation becomes sequential there is no opportunity for concurrent operations as in the graph model. Sequential machines are highly flexible, use less resource to implement a computation but are slower than the graph machines. However both graph model and sequential model are similar in the sense that the computation is carried out by directing the flow of data through functional units.

The step-by-step instructions of computation in sequential machines become "program". Burks, Goldstein and Von Neumann [BUR46] are the first to propose that programs can reside in the same memory as data. This gives rise to a class of architecture called *Stored program computer* (Fig 1.7).

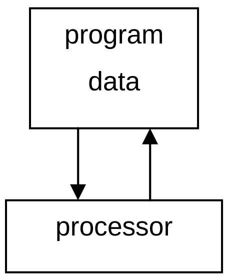


Figure 1.7 Von Neumann (or Princeton) architecture

This is the most popular organisation even today. Storing programs and data in the same memory enables a processor to manipulate programs easily. The main disadvantage is the limit of memory bandwidth, which affects the speed of running an application. As the need for more complex applications which required large amount of computation increases, having only one connection between a processor and a memory becomes bottleneck. This phenomenon is called *Von Neumann bottleneck*.

Other organisation is possible such as storing programs and data in separate memories (Fig. 1.8).

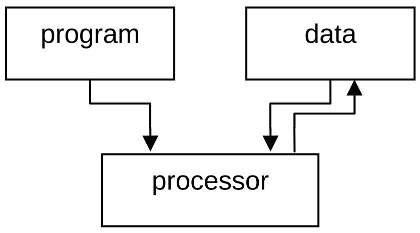


Figure 1.8 Harvard architecture

This organisation is called *Harvard architecture* and is extensively used in the high-speed processor for the purpose for signal processing. This class of processors is called Digital Signal Processor (DSP). DSP has many applications. It is used in modems, in sound synthesizer, in graphic generators etc.

## 1.5 Computer languages and architecture

Programming techniques influence the design of computers since the early days of assembly language programming. Most computers today are implemented as sequential machines. They are suitable to be programmed in a class of high level programming language, procedural languages. The examples of procedural languages are C, Pascal, C++, Java. In these languages, the computation is viewed as step-by-step manipulation of values of variables stored in memory.

There are other paradigms of programming. Backus, the father of FORTRAN, gave a lecture in the occasion of his reception of Turing award, titled “Can computers be liberated from Von Neumann bottleneck?” [BAC78]. This lecture advocated a different programming paradigm called Functional Programming. In functional paradigm, programming is viewed as the activity of composing functions. The computation of a function has an important property of *referencial transparency*. This means the result of computing a function depends only on its arguments and does not change by where the function resides. This property is contrasted to procedural programming which computes by *side effect*, i.e. manipulation of variables depends on states. Functional programming helps to promote the correctness of programs. As this paradigm of programming views computation as composing functions, it maps nicely to the graph model of computation. Many proposals being put forward to build machines that are suitable for this class of programming languages, for example a graph reduction machine [KOO90].

Different programming paradigms lead to different architectures. Logic programming paradigm (Prolog programming language and others) requires architecture capable of inferring facts and rules and ability to backtrack efficiently, for example [WAR83]. A LISP machine has special instructions to manipulate the type-tag bits [STE88]. Japanese proposed and built various types of these machines in the period of their research on Fifth generation computer. Presently, object-orientated programming paradigm is becoming the dominate paradigm. The object-oriented programming languages (Java, C++, Smalltalk etc.) will benefit from machines whose architecture are suitable to implement them.

## 1.6 Performance

This section discusses the performance issue. How performance of a computer system is defined and measured. There are many standard references used to interpret performance figures. Performance can be used in a relative sense, it is the measurement of one system compares to another system.

The first commercial electronic computer appeared around 1950. In the first 25 years the performance improvement came mostly from technology and better computer architecture. Later, the improvement mostly came from the advent of microelectronics. The speed of components increased 18-35% per year. Technology progresses from vacuum tubes to transistors to integrated circuits. The birth of microprocessor around 1970 [FAG96] has great impact on performance of computers. The growth of performance has been highest for microprocessors. Since 1980 the performance double every two years. For example, around 1980 the first IBM PC appeared. Its CPU was an Intel 8088, a 16-bit CPU with 8 MHz clock. It had 16K bytes of memory, one floppy disk and no hard disk. The later model offered 5M bytes hard disk (so called IBM XT). Today, a PC is equipped with 32/64-bit CPU with 3 GHz clock, 1G bytes of memory and 100 G bytes disk. Its performance is around 10,000 times of the first PC.

Performance is measured by running *mixed jobs*. Therefore it is not an absolute figure. It depends on the kind of jobs that are used to measure the performance. One phenomenon that occurs in the computer technology is that the performance of a processor has been double every 18 months. This observation is proposed by Moore [SCH97], who is the pioneer (among a number of other engineers) of integrated circuit fabrication. He was with Fairchild, one of the earliest IC manufacturer. That observation is known as *Moore's law*. The main reason that makes this law possible is the rapid advance of the IC manufacturing technique, the shrinking of the physical dimension of the electronic circuits. For the last 30 years semiconductor technology has been roughly quadrupling every three years. This gives an exponential base of about 1.59 instead of the base 2 proposed in Moore's original paper. A more accurate formula for Moore's law is:

$$N_{\text{device on chip}} = 1.59^{(\text{year} - 1959)}$$

We define performance as:

Performance = how fast a processor complete its job.

Performance is measured by its execution time of a suite of programs called *benchmark programs*. The execution time depends on three factors.

execution time = number of instruction used × cycle per instruction × cycle time

These factors depend on various designs:

- *number of instruction* depends on instruction set design
- *cycle per instruction* depends on micro architecture
- *cycle time* depends on technology

The performance can also be measured by response time and throughput. The response time is the time between the starting of a user job and the time when the computer replies. Under multiple jobs, a better measurement is the throughput. Throughput measures how many jobs can be completed in a unit time. The response time is called *latency* of a system. The throughput is also called the *bandwidth* of a system.

Performance = how fast a computer can run
performance = response time ( latency)
performance = throughput (bandwidth)

## Relative performance

To compare the performance of two machines, it is natural to state "X is n% faster than Y". The ratio of the execution time is used to state how much one machine is faster than the other machine. The performance is the inverse of the execution time. The following relationships can be derived.

**X is n% faster than Y**

$$\text{execution time Y / execution time X} = 1 + n/100$$
$$\text{performance} = 1/\text{ execution time} \quad \text{(or } 1/t\text{)}$$
$$\text{execution time Y / execution time X = performance X / performance Y}$$
$$n = \text{(performance X – performance Y) / performance Y}$$

## Amdalh's law

The performance improvement can be measured in term of "speedup". With the advent of speed enhancement design such as pipeline and parallelism, Amdalh's law [AMD67] states how much performance improvement can be achieved for a given task using the enhancement. The speedup is defined as follows.

$$\text{speedup} = P_e / P$$
$$\text{speedup} = T / T_e$$

Where $P_e$ is performance with enhancement use, P is performance without enhancement use, $T_e$ is execution time with enhancement use, T is execution time without enhancement use.

If enhancement is used only partially, the speedup will be severely limited. Let f be the fraction that enhancement is used.

$$\text{execution time new} = \text{execution time old } ( (1 - f) + f / \text{speedup})$$

$$\text{speedup overall} = 1 / ((1 - f) + f / \text{speedup} )$$

Therefore the limit depends on how much the enhancement has been used. In achieving speedup by parallelization, Amdalh's law predicts that speedup will be limited by the sequential part of the program. Let see some numerical example.

**Example**: A computer has an enhancement with 10 times speedup. That enhancement is used only 50% of the time. What is the overall speedup?

$$\text{speedup overall} = 1/ ((1 - 0.5) + 0.5/10 ) = 1.82$$

Please note that Amdalh's law applies only with the problem of fixed size. When the problem size much larger than the machine, Amdalh's law does not applied. This is why the massively parallel machine is still possible.

## 1.7 Brief history of computer

The history of computer is full of interesting episodes. We will to start off with asking the question "Who made the first computer?" To find out the answer we need to clarify some definition. What kind of machine is considered to be a computer?

In mechanical era, the computing machine is really a mechanical calculator. In 1890, Charles Babbage designed and attempted to build Analytical Engine, which contained many ideas that are used in modern computers such as Arithmetic Logic Unit. However, it was never finished as the British government finally stopped funding for the construction of Babbage's Analytical Engine.

The MARK 1 (also known as the IBM automatic sequence controlled calculator) developed in 1944 at Harvard University by Howard Aiken with the assistance of Grace Hopper. It was used, by the US Navy, for gunnery and ballistic calculations, and kept in operation until 1959. The computer was controlled by pre-punched paper tape and could carry out addition, subtraction, multiplication, division and reference to previous results. Numbers were stored and counted mechanically using 3000 decimal storage wheels. It was electro-mechanical computer and was slow requiring 3-5 seconds for a multiplication operation. This machine is a *configurable calculator,* in an essence it is an implementation of Babbage's machine with newer technology.

When does a machine become a computer? We will define a modern computer as *a general purpose programmable machine*. The "programmability" is considered an essential characteristic of a computer. Alan Turing was the genius who proved that the general purpose computer was possible and simple in 1937 in his seminal paper "On computable numbers" [TUR37]. To have this programmability a computer must have the *stored program*.

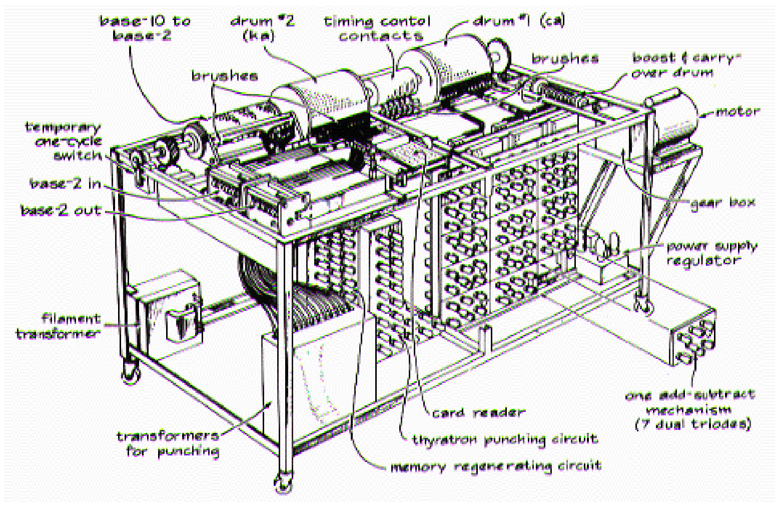


Figure 1.9  The ABC diagram [IOW99]

The ABC (Atanasoff Berry Computer) was built in 1937-1942 at Iowa State University by John V. Atanasoff  and Clifford Berry [BUR88] [MOL88].  It introduced the ideas of binary arithmetic, regenerative memory, and logic circuits.  This machine was essentially a powerful configurable calculator. Mauchly spent many days with Atanasoff in 1940 studying this machine. This was the first computer to use electronic valves (tubes) to perform arithmetic. Atanasoff stopped developing this with the advent of war, and never returned to it.  This machine doesn’t have the “stored program” ability.

In 1943 Flowers in Bletchley Park built the first Colossus machine, a programmable computer specially designed to crack the German Enigma military cypher machines.  It is not a *general purpose* and has no *stored program*.  In 1944 Zuse in Germany started work on a truly general purpose programmable computer of modern type, known as the Z4. The end of the war interrupted development.  Zuse’s earlier machines (Z1-Z3) were elegant and sophisticated in design, for example using the much more economical binary representation of numbers, but were basically modernised Babbage machines.

A group of scientists and engineers at the University of Pennsylvania, Moore School of Electrical Engineering  built ENIAC (Electronic Numerical Integrator and Computer) in 1946 [BUR81]. It was programmed by a plug board, which wired up the different calculation units in the right configuration, to evaluate a particular polynomial. Eckert and Mauchly, the designers, at this time patented a digital computing device, and are often claimed to be the inventors of the first computer. It was later proven in a 1973 US court battle between Honeywell and Sperry Rand that while spending five days at Atanastoff’s lab, Mauchly observed the ABC and read its 35-page manual. Later it was proven that Mauchly had used this information in constructing the ENIAC. Therefore, John Vincent Atanasoff is now (by some US historians) heralded as the inventor of the first electronic computer.

In 1945 John Von Neumann published the EDVAC report, a review of the design of the ENIAC, and a proposal for the design of EDVAC. This is widely regarded as the origin of the idea of the modern computer, containing the crucial idea of the *stored program*. A processor fetches instructions from memory. It also reads and writes data to and from memory. This is called *Von Neumann architecture* where data and instruction co-resides in a memory.  This idea came from the proposal of an electronic computer by US Army Ordnance in 1946. Surprisingly, Von Neumann himself is not the first author of that proposal [BUR46]. However, Von Neumann name is honored because of his contribution to the development of this type of computer which has now becomes ubiquitous.  The implementation of this design was completed in 1952.

In 1946 The National Physical Laboratory appointed Turing, who had been developing ideas of implementing his Turing Machine concept of general purpose computation in electronic form, to a rival British project intended to outclass EDVAC, known as the ACE.  ACE design was at the time the most advanced and most detailed computer design in existence.  Its construction was completed in 1950 and named the Pilot ACE.

On 21st June 1948 the first stored program ran on the Small-Scale Experimental Machine (SSEM), nicknamed “Baby”, the precursor of the Manchester Mk 1 [LAV80].  So Manchester machine was the first to work.

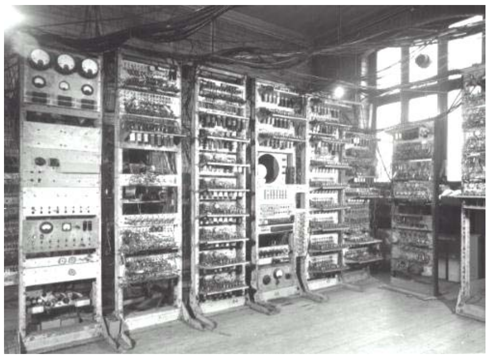
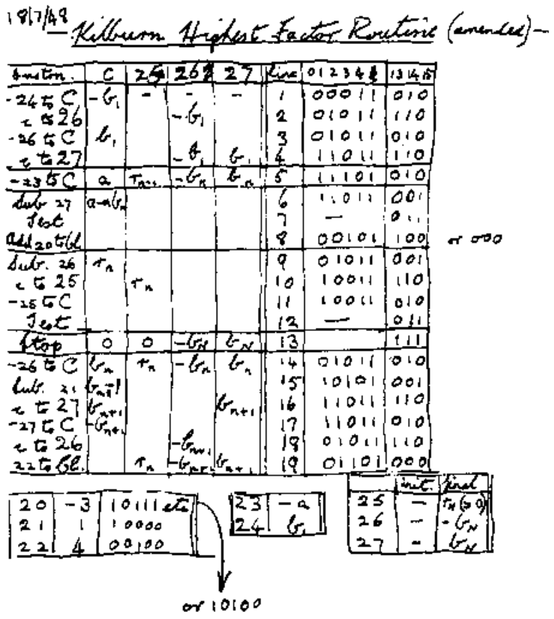


Figure 1.10  SSEM Baby from Manchester University archive [MAN]

The first program was written by Tom Kilburn. It was a program to find the highest proper factor of any number *a*.  This was done by trying every integer *b* from *a – 1* downward until one was found that divided exactly into *a*. The necessary divisions were done not by long division but by repeated subtraction of *b* (because the "Baby" only had a hardware subtractor).



18/7/48 — Kilburn Highest Factor Routine (amended) —

| Instn. | C | 25 | 26 | 27 | Line | 0 1 2 3 4 | 13 14 15 |
|---|---|---|---|---|---|---|---|
| -24 to C | $-b_1$ | - | - | - | 1 | 00011 | 010 |
| c to 26 | | | $-b_1$ | | 2 | 01011 | 110 |
| -26 to C | $b_1$ | | | | 3 | 01011 | 010 |
| c to 27 | | | $-b_1$ | $b_1$ | 4 | 11011 | 110 |
| -23 to C | $a$ | $r_{n-1}$ | $-b_n$ | $b_n$ | 5 | 11101 | 010 |
| Sub 27 | $a-nb_n$ | | | | 6 | 11011 | 001 |
| Test | | | | | 7 | — | 011 |
| Add 20 to Cl | | | | | 8 | 00101 | 100 |
| Sub. 26 | $r_n$ | | | | 9 | 01011 | 001 |
| c to 25 | | $r_n$ | | | 10 | 10011 | 110 |
| -25 to C | | | | | 11 | 10011 | 010 |
| Test | | | | | 12 | — | 011 |
| Stop | 0 | 0 | $-b_N$ | $b_N$ | 13 | | 111 |
| -26 to C | $b_n$ | $r_n$ | $-b_n$ | $b_n$ | 14 | 01011 | 010 |
| Sub. 21 | $b_n-1$ | | | | 15 | 10101 | 001 |
| c to 27 | $b_{n+1}$ | | | $b_{n+1}$ | 16 | 11011 | 110 |
| -27 to C | $-b_{n+1}$ | | | | 17 | 11011 | 010 |
| c to 26 | | | $-b_{n+1}$ | | 18 | 01011 | 110 |
| 22 to Cl. | | $r_n$ | $-b_{n+1}$ | $b_{n+1}$ | 19 | 01101 | 000 |

or 000

| | | |
|---|---|---|
| 20 | -3 | 10111 etc. |
| 21 | 1 | 10000 |
| 22 | 4 | 00100 |

or 10100

| | |
|---|---|
| 23 | $-a$ |
| 24 | $b_1$ |

| | init. | final |
|---|---|---|
| 25 | — | $r_N (=0)$ |
| 26 | — | $-b_N$ |
| 27 | — | $b_N$ |

Figure 1.11  The first program [MAN98]

Trying the program on $2^{18}$; here around 130,000 numbers were tested, which took about 2.1 million instructions and involved 3.5 million store accesses. The correct answer was obtained in a 52-minute run.

By April 1949 the Manchester Mark 1 had been finished and was generally available for scientific computation in the University. With the integration of a high speed magnetic drum by the autumn; this was the first machine with a fast electronic and magnetic two-level store (i.e. the capability for virtual memory).

In 1951 the UNIVAC 1 commercial computer was produced in US, based on the EDVAC design, and made by Eckert and Mauchly, who by this time had sold their UNIVAC Company to Remington Rand. It employed decimal arithmetic.

We will stop our trip to the history of computer here. To find out more, there is a wonderful journal devoted to all aspects of history of computing, "Annals of the History of Computing", IEEE Computer Society.

## 1.8 Summary

We have outlined the whole spectrum of a computer system. A computer system can be understood as layers of abstraction. Each layer has well defined characteristic. A computer system engineering requires understanding each component and how many components interact in a computer system. The most important character of a computer system is its programmability. We have focused on computation and its relation to the hardware level, the data path. Computation can be realised as parallel or sequential in a data path. There is interchangeability between hardware and software. This fact gives rise to many choices in the design of a computer system.

## References

[AMD67] Amdahl, G., "Validity of the single processor approach to achieving large scale computing capabilities", AFIPS Conf. Proc., April 1967, pp. 483-485.

[BAC78] Backus, J. "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs", Communications of the ACM, August 1978, 20(8):613-641.

[BEL71] Bell, C., and Newell, A. Computer structure: Readings and examples. McGraw-Hill, 1971.

[BUR46] Burks, A. W., Goldstein, H. H. and von Neumann, “Preliminary discussion of the logical design of an electronic computing instrument”, US Army Ordnance Department Report 1946.

[BUR81] Burks, A., and Burks, A., The ENIAC: First General Purpose Electronic Computer, The University of Michigan Press, Ann Arbor, Michigan, 1981.

[BUR88] Burks, A., and Burks, A., The First Electronic Computer: The Atanasoff Story, the University of Michigan Press, Ann Arbor, Michigan, 1988.

[FAG96] Faggin, F., Hoff, M., Mazor, S., and Shima, M., “The history of 4004”, IEEE Micro, December, 1996, pp.10-20.

[GOL47] Goldstein, H., von Neumann, J., and Burks, A., “Report on the mathematical and logical aspects of an electronic computing instrument”, Institute of advanced study, 1947.

[IOW99] Iowa State University, Department of computer science, http:// www.cs.iastate.edu/jva/jva-archive.shtml

[KAT93] Katz, R., Contemporary Logic Design, Addison-Wesley, 1993.

[KOO90] Koopman, P., An Architecture for Combinator Graph Reduction, Academic Press, 1990.

[LAV80] Lavington, S., Early British Computers, Manchester University Press, 1980.

[LEE95] Lee, J., Computer Pioneers, IEEE CS Press, Los Alamitos, California, 1995.

[MAN98] Manchester university, computer science department, MARK1, http:// www.computer50.org/mark1/firstprog.html

[MAN] The university of Manchester celebrates the birth of the modern computer, http://www.computer50.org/mark1/

[MOL88] Mollenhoff, C., Atanasoff: Forgotten Father of the Computer, ISU Press, 1988.

[SCH97] Schaller, R., “Moore’s Law: Past, Present and Future”, IEEE Spectrum, June, 1997.

[STE88] Steenkiste, P., Hennessy, J., “Lisp on a reduced-instruction-set computer: characterization and optimization”, Computer, vol.21, no. 7, July 1988, pp.34-45.

[STN80] Stern, N., “Who invented the first electronic digital computer?”, Annals of the History of Computing, 2:4 (October), 375-376.

[TUR37] Turing, A., “On Computable Numbers, with an application to the Entscheidungsproblem”, Proc. Lond. Math. Soc. (2) 42 pp 230-265 (1936-7); correction ibid. 43, pp 544-546 (1937).

[WAR83] Warren, D., “An abstract Prolog instruction set, Technical report 309, SRI, 1983.

## Exercises

1.1 Write two realisations of the computation of summation of *1..n*, one is data flow paradigm, the other one is in a conventional data path.

1.2 A conventional way of thinking about program is that a program processes input to output. A new way of thinking about program is that an event occurs then program responds to it. One can form “if..then” rules into a program to reflect this new thinking. Write a program to sum *1..n* using the “if..then” rules.

1.3 Suppose we do not have programs. How can we built a circuit to solve Tower of Hanoi problem? (It requires a recursive program to solve it).

1.4 The performance factors: the number of instruction executed and the cycle per instruction are interrelated. Can it be possible that we design a computer system to succeed in reducing both factors at the same time? Please give examples from existing computer systems.

1.5 The question “who built the first electronic computer?” was a topic of debate in the last decade. There was the case of Mauchly and Eckert versus Atanasoff. In the end Atanasoff is credited. Look up the detail of the case in the internet. Describe what happened.