Computer Organization

Lecturer : R. Setiawan Aji Nugroho

Office: Computer Science Department of UNIKA Soegijapranata

Telephone: (024) 5441555 ext. 138

Web Site: http://www.unika.ac.id/aji

E-mail: wawan@unika.ac.id

COMPUTER SYSTEMS ORGANIZATION

A digital computer consists of an interconnected system of processors, memories, and input/output devices. This chapter is an introduction to these three components and to the methods by which computers are interconnected. The purpose of the chapter is to provide an overview of how computers are organized, as background for the detailed examination of specific levels in the five succeeding chapters.

2.1. PROCESSORS

The organization of a simple computer is shown in Fig. 2-1. The central processing unit (CPU) is the "brain" of the computer. Its function is to execute programs stored in the main memory by fetching their instructions, examining them, and then executing them one after another. The CPU is composed of several distinct parts. The control unit is responsible for fetching instructions from main memory and determining their type. The arithmetic and logical unit performs operations such as addition and Boolean AND needed to carry out the instructions.
The CPU also contains a small, high-speed memory used to store temporary results and certain control information. This memory consists of a number of registers, each of which has a certain function. The most important register is the pro gram counter (PC), which points to the next instruction to be executed. The name "program counter" is somewhat misleading because it has nothing to do with *Counting* anything, but the term is universally used. Also important is the instruction register (IR), which holds the instruction currently being executed. Most computers

have other registers as well, some of them available to the level 2 and 3 programmers for storing intermediate results.

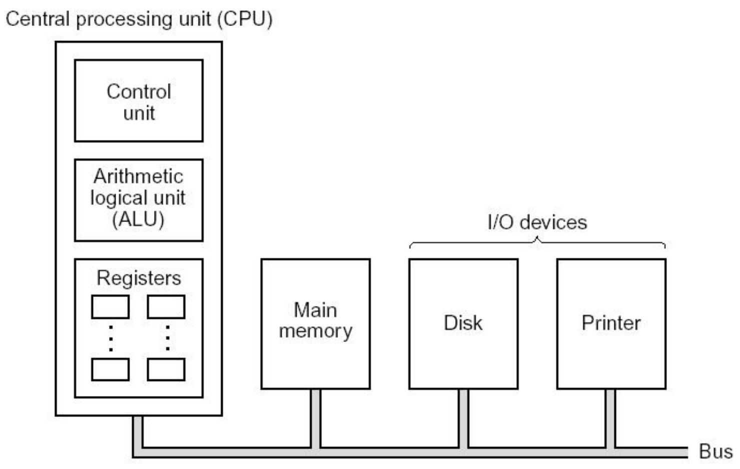


Fig. 2-1. The organization of a simple computer with one CPU and two I/O devices.

2.1.1. Instruction Execution

The CPU executes each instruction in a series of small steps:

1. Fetch the next instruction from memory into the instruction register.
2. Change the program counter so that it points to the following instruction.
3. Determine the type of instruction just fetched.
4. If the instruction uses data in memory, determine where they are.
5. Fetch the data, if any, into internal CPU registers.
6. Execute the instruction.
7. Store the results in the proper place.
8. Go to step 1 to begin executing the following instruction.

This description of how a CPU works closely resembles a program written in English. Figure 2-2 shows this informal program rewritten as a Pascal procedure. The very fact that it is possible to write a program that can imitate the function of a CPU shows that a program need not be executed by a "hardware" CPU consisting of a box full of electronics. Instead, a program can be carried out by having another program fetch, examine, and execute its instructions. A program (such as Fig. 2-2) that fetches, examines, and executes the instructions of another program is called an interpreter, as mentioned in Chap. 1.

This equivalence between hardware processors and interpreters has important implications for computer organization. After having specified the machine language, L, for a new computer, the design group can decide whether they want to build a hardware processor to execute programs in L directly or whether they want to write an interpreter instead. If they choose to write an interpreter, they must also provide some machine to run the interpreter.

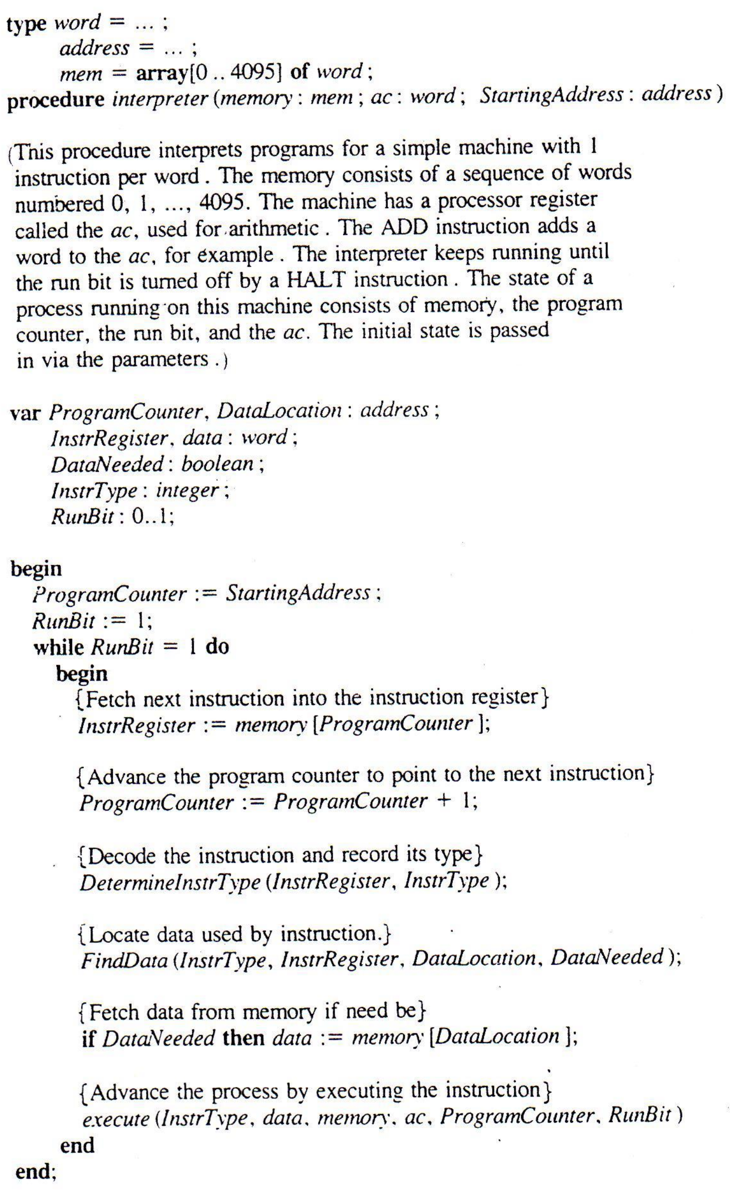
Because an interpreter breaks the instructions of its target machine into small steps, the machine on which the interpreter runs can often be much simpler, and therefore less expensive than a hardware processor for the target machine would be. For economic as well as other reasons, programs at the conventional machine level of most modern computers are carried out by an interpreter running on a totally different and much more primitive level 1 machine that we have called the microprogramming level.

The collection of all instructions available to the programmer at a level is called the instruction set of that level. The number of instructions in the instruction set varies from machine to machine and from level to level. For the conventional machine level, for example, the size of the instruction set is typically in the range 20 to 300. A large instruction set is not necessarily better than a small one. In fact, the opposite tends to be true. A large instruction set often means that the instructions are not very general. Compilers for high-level languages, such as Ada, FORTRAN, Pascal, and PL/1, generally perform better on machines with small, well-chosen instruction sets than on machines with large, unwieldy ones.

Be sure you realize that the instruction set and organization of the microprogramming level is, in fact, the instruction set and organization of the hardware (CPU). The instruction set and organization of the conventional machine level, in contrast, is determined by the microprogram, and not by the hardware.

Not all processors are general-purpose CPUs. Computers often contain one or more processors with a specific function that requires a limited, specially designed instruction set. Such special-purpose processors are widely used to perform input/output. As an example, consider a graphics terminal, a device with a screen similar to that of a television set, on which the computer can display drawings of electrical circuits, blueprints of buildings, maps, graphs of data, and many other kinds of useful information.

A graphics terminal normally contains a highly specialized processor and a memory. The graphics processor may have instructions to plot points in several intensities and colors; draw solid, dotted, and dashed lines: display characters; and produce geometric figures such as squares, circles, and triangles. Such an instruction set is obviously different from what a general-purpose CPU needs.



```
type word = ... ;
      address = ... ;
      mem = array[0 .. 4095] of word ;
procedure interpreter (memory : mem ; ac : word ;  StartingAddress : address )

(This procedure interprets programs for a simple machine with 1
 instruction per word . The memory consists of a sequence of words
 numbered 0, 1, ..., 4095. The machine has a processor register
 called the ac, used for arithmetic . The ADD instruction adds a
 word to the ac, for example . The interpreter keeps running until
 the run bit is turned off by a HALT instruction . The state of a
 process running on this machine consists of memory, the program
 counter, the run bit, and the ac. The initial state is passed
 in via the parameters .)

var ProgramCounter, DataLocation : address ;
     InstrRegister, data : word ;
     DataNeeded : boolean ;
     InstrType : integer ;
     RunBit : 0..1;

begin
   ProgramCounter := StartingAddress ;
   RunBit := 1;
   while RunBit = 1 do
      begin
        {Fetch next instruction into the instruction register}
        InstrRegister := memory [ProgramCounter ];

        {Advance the program counter to point to the next instruction}
        ProgramCounter := ProgramCounter + 1;

        {Decode the instruction and record its type}
        DetermineInstrType (InstrRegister, InstrType );

        {Locate data used by instruction.}
        FindData (InstrType, InstrRegister, DataLocation, DataNeeded );

        {Fetch data from memory if need be}
        if DataNeeded then data := memory [DataLocation ];

        {Advance the process by executing the instruction}
        execute (InstrType, data, memory, ac, ProgramCounter, RunBit )
      end
 end;
```

Fig. 2-2. An interpreter for a simple computer.

2.1.2. Parallel Instruction Execution

Although early computers had a single CPU that executed instructions one at a time, certain more advanced designs increase the effective computing speed by allowing several instructions to be executed at the same time. The simplest method is to have two or more independent CPUs sharing a common memory, as shown in Fig. 2-3(a). Usually, each CPU has its own program to run, with no interaction between the CPUs. A computer with multiple processors that share a common main memory is called a multiprocessor.

Some problems require carrying out the same computation on many sets of data. For example, a weather prediction program might read hourly temperature measurements taken from 1000 weather stations and then compute the daily average at each station by performing exactly the same computation on each set of 24 hourly readings. For each station's data, it would load the first value into a register, then add the second value, then the third, and so on. Finally, it would divide the sum by 24. Because the same program is used on each data set, a processor with one program counter and one instruction decoder but *n* arithmetic units and *n* register sets could carry out computations on *n* data sets simultaneously.

This configuration, which is shown in Fig. 2-3(b), is sometimes called a singleinstruction-stream multiple-data-stream processor, or an array _processor. An example of this type of organization is the ILLIAC IV, designed at the University of Illi nois and constructed by the Burroughs Corporation. The ILLIAC IV consists of four control units, each of which operates on 64 data sets simultaneously. The ILLIAC IV can perform four different computations at the same time, each computation being carried out on 64 sets of data, making a total of 256 calculations in parallel.

The CDC 6600 and some of the CDC Cybers use yet another form.of parallel instruction execution. These machines have separate arithmetic units for addition, multiplication, division, and other operations, for a total of 10 separate units (includ ing two of some units). Each one has a single control unit that fetches and decodes instructions. As soon as the type of an instruction is known, it is sent to the appropriate unit (e.g., multiplication), and the fetching and decoding of the next instruction begins, concurrently with the execution of the first instruction. A well-tuned program may have up to 10 instructions being executed simultaneously. This scheme only makes sense if the time needed to fetch and decode an instruction is small compared to the time needed to carry it out, which is the case on the CDC machines. Figure 2-3(c) illustrates this concept.

A related processor organization is the pipeline machine, shown in Fig. 2-3(d). In this design, each of the steps in executing an instruction (e.g., fetch instruction into the instruction register, determine instruction type, locate the data) has a separate unit within the CPU to perform it. When the machine starts, the initial instruction is fetched by the first unit. Then the second unit begins decoding the initial instruction while the first unit is busy fetching the second instruction. A little later the third unit is busy examining the initial instruction to see if it needs data from memory, while the second unit is decoding the second instruction and the first unit is busy fetching the next instruction. The IBM 360/195, CDC STAR, and University of Manchester MU5 use sophisticated forms of pipelining.

A pipeline machine can be compared to an automobile assembly line: at each position along the (dis)assembly line the same function is performed on the stream of instructions that come by. The ones near the end of the pipeline are nearly completed; and the ones near the beginning of the pipeline are barely started.

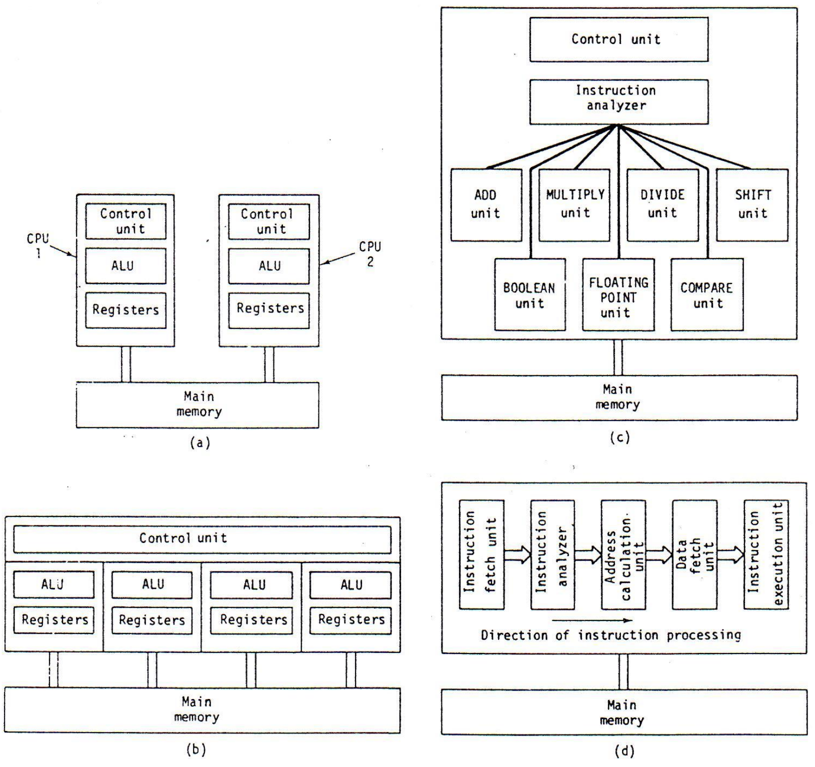


Fig. 2-3. (a) Multiple CPUs. (b) Array processor. (c) Multiple functional units. (d) Pipelining.

2.1.3. Processor Classification

Judging by the amount of attention microprocessors and microcomputers get in both the popular press and in technical journals, one might assume that everybody knew what they were talking about. Nothing could be further from the truth. If you ask *n* different computer scientists to define "microprocessor" or "microcomputer" you are likely to get *n* different answers. To see how this has come about, we need a little ancient history.

Until about 1960, all computers were big: they occupied hundreds of square meters, gave off enough heat to warm whole buildings, cost a small fortune, and had staffs of dozens of programmers, operators, and technicians to keep them contented With minor modifications, this model still holds for most computer centers. These behemoths are called mainframes.

Around 1961, DEC (Digital Equipment Corporation) introduced the PDP-1, which occupied less than 10 square meters, could barely heat its own room, cost a mere 5120,000, and could be managed by two or three people. This was the first mini computer. By 1965 minicomputers could be had for $20,000 and were selling like hotcakes. At this price a department of a company or university could buy its own computer, and thousands did.

In 1972 Intel introduced the first CPU contained on a single chip, the 4004. By current standards it was a primitive machine but it was an instan: success and within a few years had given birth to an entire industry. Because the term "minicomputer" was already in use for something much larger, these single-chip CPUs were called microprocessors. A computer system using a microprocessor as CPU was obviously called a microcomputer.

At first, mainframes, minicomputers, and microcomputers had clear technical differences. For example, mainframes had word lengths of at least 32 bits, minicom - puters had 16-bit word lengths, and microcomputers had 8-bit word lengths. Also, mainframes could handle large memories, minis could handle. medium-sized memories, and micros could have only small memories.

As a result of enormous improvements in integrated circuit technology, these differences no-longer exist. Some micros have 32-bit word lengths and can address as much memory as a mainframe. It is therefore meaningless to try to make a distinction based on technical characteristics.

What is still (more-or-less) meaningful is a distinction made on typical usage. Mainframe is a big machine operated by a computer center for the benefit of a large community of often unrelated users. When used for time sharing, a mainframe is nor mally equipped with many megabytes of main memory, thousands of megabytes of disk memory, and can support 100 or so interactive users.

A minicomputer is a smaller machine typically owned and operated by a single department or project. As a time-sharing machine, it might have at most a few megabytes of main memory, a few hundred megabytes of disk memory, and the ability to support a few dozen terminals. With the arrival of the "superminis" (e.g., DEC VAX 11/780) in the late 1970s, the distinction between a mini and a mainframe was blurred even more, because the superminis can be configured to have the computing power of small mainframes.

A microcomputer, in contrast, is typically not thought of as a computer at all, but is embedded inside a piece of equipment such as a car or television. In this configuration it has little memory and no disks. The confusion arises because it is possible to equip the larger microcomputers with a megabyte of memory and several disks, in effect using them as minicomputers or even small mainframes.

The conclusion of this discussion is that no conceptual difference exists between mainframes, minicomputers, and microcomputers. They are simply rough names for various overlapping parts of a continuous spectrum of processor power. Furthermore, future developments in technology are likely to make these names even less meaningful than they are already.

2.1. MEMORY

The memory is that part of the computer where programs and data are stored. Some computer scientists use the term store or storage rather than memory. Without a memory from which the processors can read and write information, there would be no stored-program digital computers as we know them.

2.2.1. Bits

The basic unit of memory is the binary digit, called a bit. A bit may contain a 0 or a l. It is the simplest possible unit. (A device capable of storing only zeros could hardly form the basis of a memory system.)

People often say that computers use binary arithmetic because it is "efficient." What they mean (although they rarely realize it) is that digital information can be stored by distinguishing between different values of some continuous physical quan tity, such as voltage or current. The more values that must be distinguished, the less separation between adjacent values, and the less reliable the memory. The binary number system requires only two values to be distinguished. Consequently, it is the most reliable method for encoding digital information.

Some computers are advertised as having decimal rather than binary arithmetic. This trick is accomplished by using 4 bits to store one decimal digit. Four bits provide 16 combinations, used for the 10 digits 0 through 9, with six combinations not used. The number 1944 is shown below encoded in decimal and in pure binary, using lfi bits in each example:

decimal: 0001 1001 0100 0100   binary: 00000 111100 11000

Sixteen bits in the decimal format can store the numbers from 0 to 9999, giving only 10,000 combinations, whereas a 16-bit pure binary number can store 65536 different combinations. For this reason, people say that binary is more efficient.

However, consider what would happen if some brilliant young electrical engineer invented a highly reliable electronic device that could directly store the digits 0 to 9 by dividing the region from 0 to 10 volts into 10 parts. Four of these devices could store any decimal number from 0 to 9999. Four such devices would provide 10,000 combinations. They could also be used to store binary numbers, by only using 0 and 1, in which case, four of them could only store 16 combinations. With such devices, the decimal system is obviously more efficient.

2.2.2. Memory Addresses

Memories consist of a number of cells (or locations) each of which can store a piece of information. Each cell has a number, called its address, by which programs can refer to it. If a memory has n cells, they w111 have addresses 0 to n - 1. All cells in a memory contain the same number of bits. If a cell consists of k bits, it can hold any one of 2^r different bit combinations. Figure 2-4 shows three different organizations for a 96-bit memory. Note that adjacent cells have consecutive addresses (by definition).

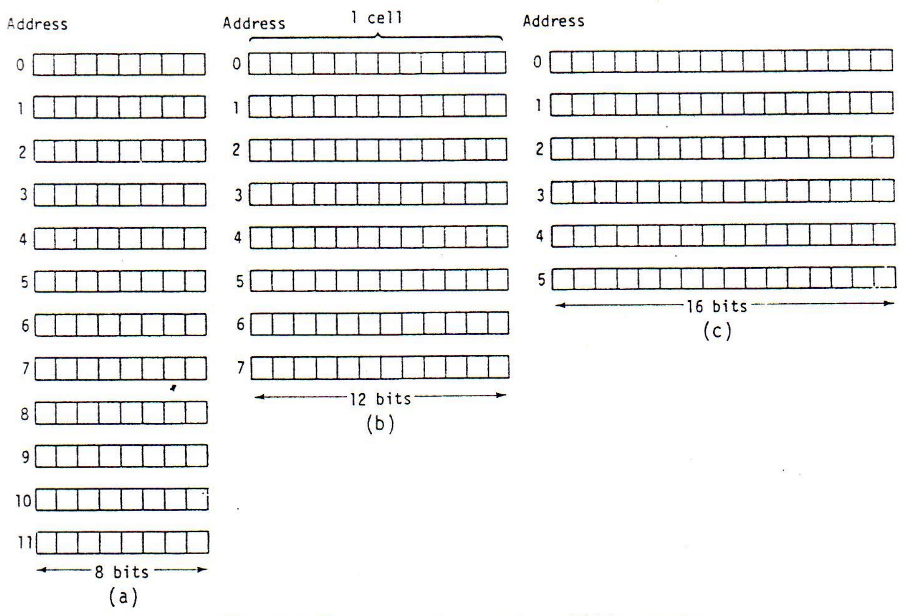


Fig. 2-4. Three ways of organizing a 96-bit memory.

Computers that use the binary number system (including octal and hexadecimal notation for binary numbers) also express memory addresses as binary numbers. If an address has m bits, the maximum number of cells directly addressable is 2'. For example, an address used to reference the memory of Fig. 2-4(a) would need at least 4 bits in order to express all the numbers from 0 to 11. A 3-bit address would be sufficient for Fig. 2-4(b) and (c), however. The number of bits in the address is related to the maximum number of directly addressable cells in the memory and is independent of the number of bits per cell. A memory with $2^{1`}$ cells of 8 bits each and a memory with 2^{1z} cells of 60 bits each would each need 12-bit addresses.

The number of bits per cell for some computers that have been sold commercially follows.

Burroughs B 1700:	1 bit per cell

IBM 370:	8 bits per cell
DEC PDP-8:	12 bits per cell
DEC IBM 1130:	16 bits per cell
DEC PDP-15:	18 bits per cell
XDS 940:	24 bits per cell
Electrologica X8:	27 bits per cell
XDS Sigma 9:	32 bits per cell
Honeywell 6180:	36 bits per cell
CDC 3600:	48 bits per cell
CDC Cyber	60 bits per cell

On some machines the term word is used in place of cell. On other machines, especially machines with 8 bits per cell, the term byte is used instead of cell, and the term "word" is reserved for 2 bytes, or 4 bytes, depending on the machine. This situation, unfortunately, can lead to much confusion.

The significance of a cell is that it is the smallest addressable unit. Each cell has a unique address; consecutive cells have addresses differing by 1. The significance of a word is that, at level 2, registers that can hold one word are usually available for temporary storage. Furthermore, there are instructions to fetch and store an entire word in one operation. Following popular usage, we will use the term "word" rather than "cell" to indicate the basic information unit of memory, except where the distinction is significant. Keep in mind that the relation between words and cells depends on the machine in question.

The CPU communicates with the main memory using two registers--the memory address register (MAR) and the memory buffer register (MBR). When the CPU needs to read a word from memory, either an instruction or data, it loads the address of the word it wants into the MAR and sends a read signal to the memory. The memory begins operating; after one memory cycle it puts the desired word in the MBR, where the CPU can take it out and use it. To write a word into memorv, the CPU puts the address to be stored into the MAR and the word to be stored in the 1VIBR. Then it signals the memory to begin a store operation. Except on a few computers, the MAR and MBR are not directly accessible to level 2 programs, although they are always accessible to level 1 programs.

2.2.3. Metabits

Although main memory is used to store both programs and data, the processor generally has no way of telling whether a particular cell contains an instruction or data. A common programming error is for a program to jump into the data accidentally and try to execute them. Inadvertent attempts to execute data (or to multiply one machine instruction by another) could be caught immediately by the processor if each cell (or word) had an extra bit associated with it, containing a 0 for instructions and a 1 for data, as shown in Fig. 2-5.

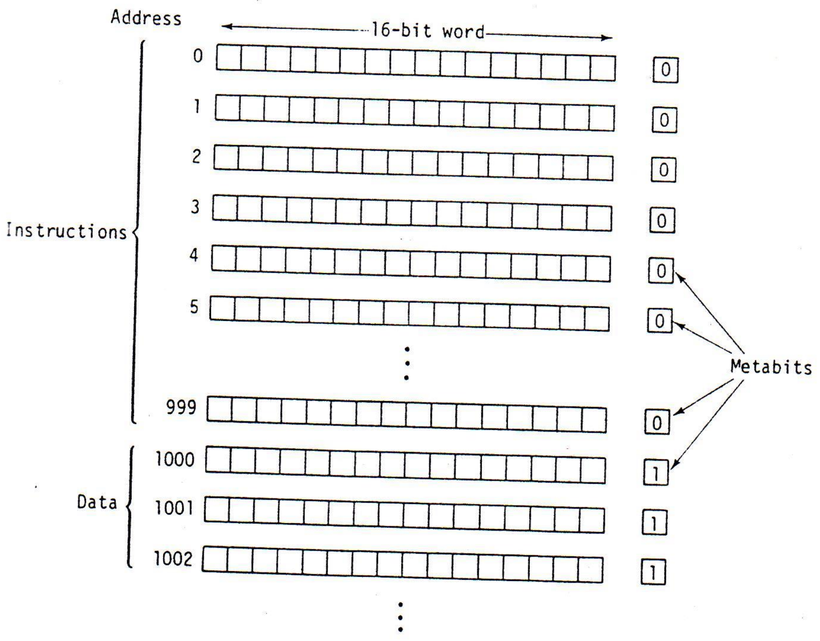


Fig. 2-5. A memory consisting of words with 16 data bits and 1 metabit. The metabit is 0 for instructions and 1 for data.

An extra bit associated with each cell (or word) used to indicate its contents a called a metabit or tag bit. The following method shows how the metabits of Fig. 2-5 could be used to detect some programming errors. When the processor fetches a cell (or word) from memory, either data or instruction, the metabit associated with that memory word would be loaded into a special metabit register within the processor. If the metabit register contains a 1 after an instruction fetch or a 0 after a data fetch, the processor would stop execution and print an error message.

Metabits can be used for other purposes as well. For example, if each cell (or word) had several metabits instead of just one, the metabits could not only distinguish between instructions and data but for the latter could also specify the type of data. Characters, integers, floating-point numbers (see Appendix B), decimal numbers, and other data types could each be uniquely indicated by the metabits. Doing so would eliminate the need for different instructions for integer addition, floating-point addition, and decimal addition. There could be a single addition instruction and the processor would examine the metabits to see which kind was needed. A further extension would provide metabits (a) to indicate integer arrays, floating-point arrays, and decimal arrays as well as other data types, and (b) to allow the addition instruction to operate on whole one-, two-, and three-dimensional arrays as well as individual numbers.

2.2.4. Secondary Memory

Because every word in main memory is directly accessible in a very short time, main memory is relatively expensive. Consequently, most computers have slower, cheaper, and usually much larger secondary memories as well. Secondary memories are used to hold large sets of data.

Magnetic Tapes

Historically, magnetic tape was the first kind of secondary memory. A computer tape drive is analogous to a home tape recorder: a 2400-ft-long tape is wound from the feed reel past a recording head to the take-up reel. By varying the current in the recording head, the computer cam write information on the tape in the form of little magnetized spots.

Figure 2-6 shows how information is organized on a magnetic tape. On a computer with 8-bit bytes, each frame contains 1 byte, plus an extra, redundant bit, called a parity bit, to improve reliability. Typical recording density is 1600 frames (bytes) per inch (noted as 1600 bpi), which means that the distance between frames is less than 1/1000 of an inch. Other common densities are 800 bpi and 6250 bpi. After a tape drive has finished writing a record, it leaves a gap on the tape while slowing down. If the program writes short physical records on the tape, most of the space will be wasted in the gaps. By writing physical records that are much longer than the gap, the tape utilization can be kept high.

Magnetic tapes are sequential-access devices. If the tape is positioned at the beginning, to read physical record n, it is first necessary to read physical records 1 through n - 1, one at a time. If the information desired is near the end of the tape, the program will have to read almost the entire tape, which may take several minutes. Forcing a CPU that can execute millions of instructions per second to wait 200 sec while a tape is advanced is wasteful. Tapes are most appropriate when the data must be accessed sequentially.

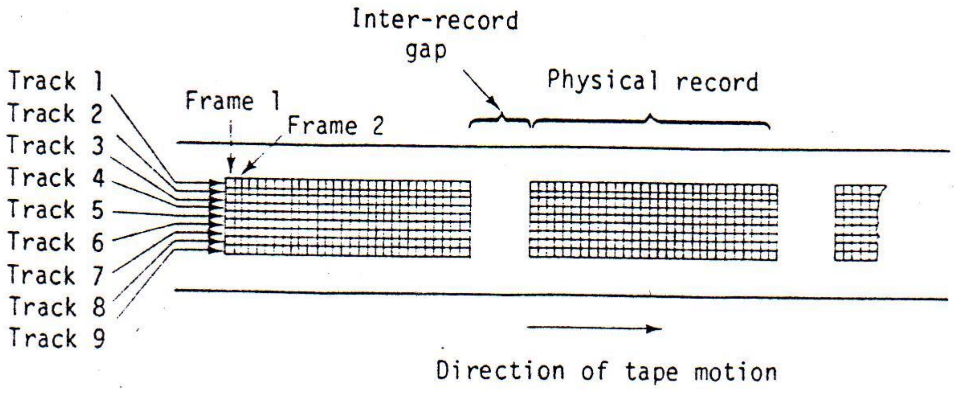


Fig. 2-6. Information on a magnetic tape is recorded as a sequence of rectangular bit matrices.

Magnetic Disks

A disk is a piece of metal, about the size and shape of an LP phonograph record, to which a magnetizable coating has been applied at the factory, generally on both sides. Information is recorded on a number of concentric circles, called tracks (see rig. 2-7). Disks typically have a few hundred tracks per surface. Each drive has a movable head that can be moved closer to, or farther from, the center. The head is wide enough to read or write information from exactly one track. A disk drive often nas several disks stacked vertically about an inch apart. In such a configuration the arm will have one head next to each surface, all of which move in and out together. The radial position of the heads (distance from the spindle) is called the cylinder address. A disk drive with n platters will have *2n* surfaces, hence *2n* tracks per cylinder.

Tracks are divided into sectors, normally between 10 and 100 sectors per track. A sector consists of a certain number of machine words, typically *32* to *256*. On some disks the number of sectors per track can be set by the program.

Each disk drive has a small special-purpose computer associated with it, called the disk controller. The controller helps to transfer information between main memory and the disk. To specify a transfer, the program must provide the following information: the cylinder and suface, which together specify a unique track, the sec:or number where the information starts, the number of words to be transmitted, the main memory address where the information comes from or goes to, and whether information is to be read from the disk or written onto it. Disk transfers always start .a the beginning of a sector, never in the middle. If a multisector transfer crosses a track boundary within a cylinder, no time is lost, but if it crosses a cylinder boundary, at least) one rotation time is lost on account of the seek.

If the head happens to be positioned over the wrong cylinder, it must first be moved. This motion is called a seek. A seek typically takes 5 msec between adjacent tracks and 50 msec to go from the innermost cylinder to the outermost cylinder.

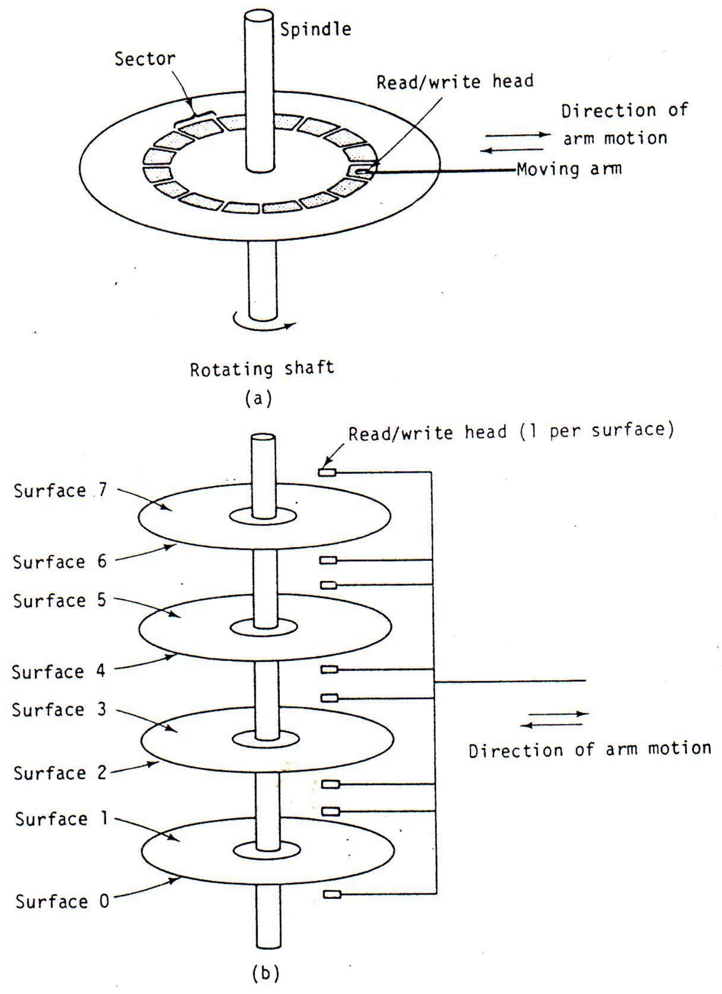


Fig. 2-7. (a) A disk with one platter. (b) A disk with four platters.

Once the head is positioned properly, the controller must wait until the first sector has rotated under the head before beginning the transfer. The time wasted waiting for the right sector varies from 0, if the program is lucky, to the complete rotation time if it just missed. This waiting time is **called the rotational latency.** Most disks rotate at 3600 rotations/min, giving a maximum latency of 16.67 msec. The total access time is the seek time plus the rotational latency. The information is transferred at a rate of one track per rotation period.

Some disk drives are constructed so that the disks themselves can be removed and brought to another computer, or stored, just like a tape. On larger machines, removable disk packs (storage modules) typically have capacities in the range 40 megabytes to several hundred megabytes. On microcomputers, however, the most common removable disks are the floppy disks, so called because they are physically flexible. A floppy disk looks like a 45-rpm record encased in a square jacket and can hold anywhere from a few tenths of a megabyte to several megabytes.

Some microcomputers also use hard (i.e., nonflexible) Winchester disks, with capacities in the tens of megabytes. Larger winchester disks, with capacities up to about 1000 megabytes or more are also available for minis and mainframes. Winchester disks are characterized by having hermetically sealed disk packs, with each pack having a head assembly inside it. This arrangement is more expensive than the open storage module packs, because each pack has a built-in head assembly, but it is also better protected against dust, hence more reliable. On the least expensive winchester drives, the packs are not removable; on the more expensive ones they sometimes are.

Magnetic Drums

A variation on' the disk is the drum, a cylinder on which information can be recorded magnetically (see Fig. 2-8). Along the length of the drum are many fixed read/write heads. Each head can read or write one track. The tracks are again divided into sectors. Because the heads do not move, there is no seek time. Furthermore, several heads may be reading or writing in parallel. Drums have a smaller capacity than disks but access is much faster because there is no seek time. Some drums have two or more sets of heads, spaced uniformly around the circumference of the drum. With two sets of heads, a given sector will always appear under one set of heads or the other within at most one-half of the rotation period. Fixed-head disks are logically similar to drums in that the heads do not move but have the physical appearance of a disk. Do not confuse fixed-head disks with "fixed disks," which are merely disks whose recording medium cannot be removed from the drive. Fixed disks are often combined with removable ones, the fixed disk for normal use and the removable one for making backups. Drums are never combined with anything else in one device.

Optical Memories

In recent years, optical (as opposed to magnetic) memories have become available. They have much higher recording densities than conventional magnetic media. For example, a strip of ordinary 35-mm black-and-white film 3 ft long can hold more information than a 2400-ft magnetic tape.

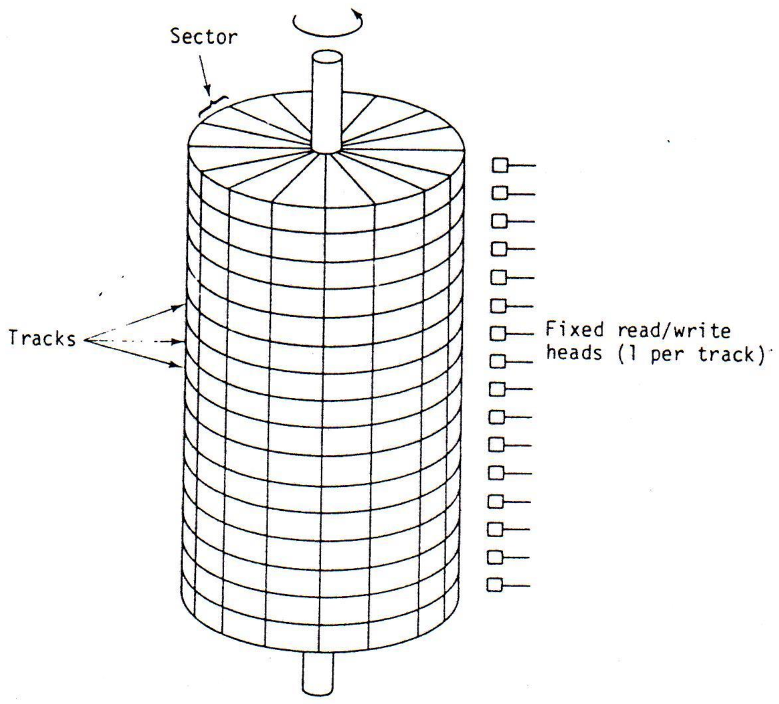


Fig. 2-8. A drum.

An especially interesting optical memory is the video disk. Although these disks were originally developed for recording television programs, they can be put to more esthetic use as computer storage devices. The disks are inherently digital, with the information recorded as a sequence of pits burned into the surface by an electron beam or laser. Television has a bandwidth of 6 MHz, so a 1-hour disk has a capacity of about 20 gigabits, depending on the recording details. Not only are video disks huge but both the drives and the disks themselves are dirt cheap compared to magnetic disks of comparable performance. One characteristic, however, that limits their application is that once a video disk has been written, it cannot be erased as a magnetic disk can be.

2.3. INPUT/OUTPUT

Before a computer can get to work solving a problem, it must be given the program, plus the data if there are any. After it has found the solution, the computer must communicate this solution to the human beings who posed the problem in the first place. The topic of getting information into and out of computers is called input/output, or usually just I/O.

Not all input comes from people and not all output is intended for people. A computer-operated solar telescope may get its input data directly from instruments observing the sun. A computer controlling an automated chemical plant may direct its output to machines throughout the plant that regulate the chemicals being produced.

2.3.1. I/0 Devices

Hundreds of kinds of I/O devices are available today and the number is growing rapidly. A few of the more common ones are listed below.

Cathode ray tube (CRT) terminals

Modified electric typewriters (teletypewriters) Card readers

Card punches

High-speed line printers

Optical readers that read printed matter, such as books

Mark sensors that can read cards marked with a special pencil

Pen and ink plotters that draw graphs

Paper tape readers

Paper tape punches

Magnetic ink readers that read bank checks

Experimental equipment such as cyclotrons or telescopes

Rats and other laboratory animals used in psychology experiments

2.3.2. I/O Processors

Some I/O devices can transmit a large amount of data in a short time. If the CPU had to process every character separately, much CPU time would be wasted. To avoid tying up the CPU for long periods of time on I/O, most medium-size and large computers have one or more specialized, low-cost I/O processors. Because the I/O is performed by these special processors, the CPU is available to spend most of its time on more difficult computations. The I/O processors can run in parallel with the CPU. In other words, while the CPU is busy computing, the I/O processors can be doing I/O.

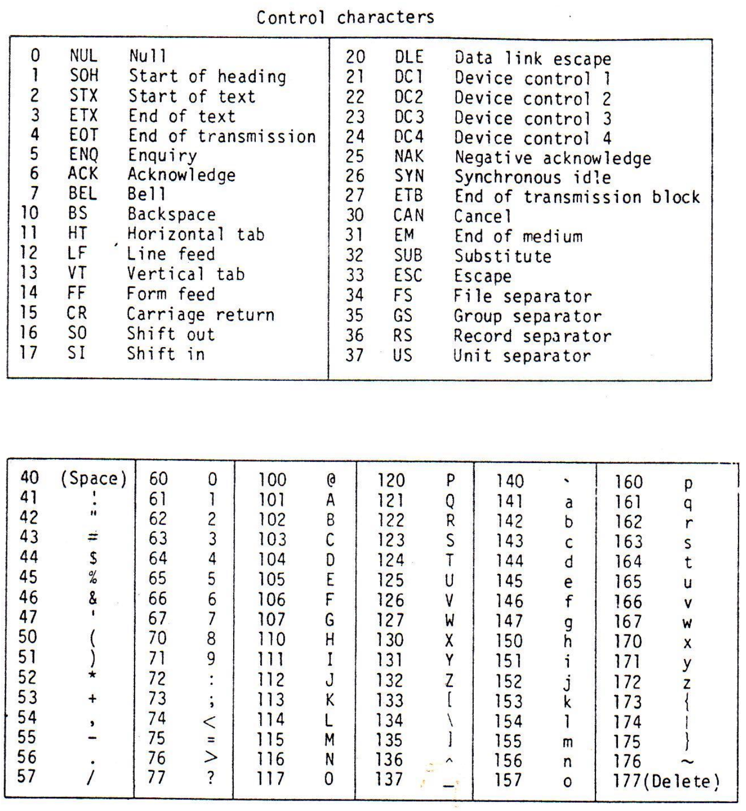
2.3.3. Character Codes

Each computer has a set of characters that it can input and output. As a bare minimum, this set includes the 26 capital letters, the 10 digits 0 through 9, and a few punctuation marks, such as space, period, minus sign, comma, and carriage return.

More sophisticated character sets include both capital letters and small letters, the 10 digits, a wide assortment of punctuation marks, a collection of special characters useful in mathematics and business, and sometimes even Greek letters.

In order to transfer these characters into the computer, each one is assigned a number: for example, a = 1, b = 2, ..., z = 26. + = 27, - = 28. The mapping of characters onto integers is called a character code. Present-day computers use either a 6-, 7-, 8- or 9-bit code. A 6-bit code allows only $2^6 = 64$ characters-namely, 26 letters, 10 digits, and 28 other characters, mostly punctuation marks and mathematical symbols.

For many applications 64 characters are not enough, in which case a 7- or 8-bit character code must be used. A 7-bit character code allows up to 128 characters. One widely used 7-bit code is called ASCII (American Standard Code for Information Interchange). Figure 2-9 shows the ASCII code. Codes 1 to 37 (octal) are control characters and do not print. The most widely used 8-bit character code is the IBM EBCDIC code.



Control characters

0	NUL	Null	20	DLE	Data link escape
1	SOH	Start of heading	21	DC1	Device control 1
2	STX	Start of text	22	DC2	Device control 2
3	ETX	End of text	23	DC3	Device control 3
4	EOT	End of transmission	24	DC4	Device control 4
5	ENQ	Enquiry	25	NAK	Negative acknowledge
6	ACK	Acknowledge	26	SYN	Synchronous idle
7	BEL	Bell	27	ETB	End of transmission block
10	BS	Backspace	30	CAN	Cancel
11	HT	Horizontal tab	31	EM	End of medium
12	LF	Line feed	32	SUB	Substitute
13	VT	Vertical tab	33	ESC	Escape
14	FF	Form feed	34	FS	File separator
15	CR	Carriage return	35	GS	Group separator
16	SO	Shift out	36	RS	Record separator
17	SI	Shift in	37	US	Unit separator

40	(Space)	60	0	100	@	120	P	140	`	160	p
41	!	61	1	101	A	121	Q	141	a	161	q
42	"	62	2	102	B	122	R	142	b	162	r
43	#	63	3	103	C	123	S	143	c	163	s
44	$	64	4	104	D	124	T	144	d	164	t
45	%	65	5	105	E	125	U	145	e	165	u
46	&	66	6	106	F	126	V	146	f	166	v
47	'	67	7	107	G	127	W	147	g	167	w
50	(	70	8	110	H	130	X	150	h	170	x
51	)	71	9	111	I	131	Y	151	i	171	y
52	*	72	:	112	J	132	Z	152	j	172	z
53	+	73	;	113	K	133	[	153	k	173	{
54	,	74	<	114	L	134	\	154	l	174	|
55	-	75	=	115	M	135	]	155	m	175	}
56	.	76	>	116	N	136	^	156	n	176	~
57	/	77	?	117	O	137	_	157	o	177(Delete)	

Fig. 2-9. The ASCII character code (octal).