3.
Den lagdelte maskine

Indhold:

3.1.a

Structured computer organization,
Second edition,
Prentice Hall
Andrew S. Tannenbaum
1984

Uddrag af kapitel 1.

A digital computer is a machine that can solve problems for people by carrying out instructions given to it. A sequence of instructions describing how to perform a certain task is called a **program**. The electronic circuits of each computer can recognize and directly execute a limited set of simple instructions into which all its programs must be converted before they can be executed. These basic instructions are rarely much more complicated than:

Add 2 numbers.

Check a number to see if it is zero.

Move a piece of data from one part of the computer's memory to another.

Together, a computer's primitive instructions form a language in which it is possible for people to communicate with the computer. Such a language is called a **machine language**.

The people designing a new computer must decide what instructions to include in its machine language. Usually they try to make the primitive instructions as simple as possible, consistent with the computer's intended use and performance requirements, in order to reduce the complexity and cost of the electronics needed. Because most machine languages are so simple, it is difficult and tedious for people to use them.

This problem can be attacked in two principal ways: both involve designing a new set of instructions that is more convenient for people to use than the set of built-in machine instructions. Taken together, these new instructions also form a language, which we will call L2, just as the built-in machine instructions form a language, which we will call L1. The two approaches differ in the way programs written in L2 are executed by the computer, which, after all, can only execute programs written in its machine language, L1.

One method of executing a program written in L2 is first to replace each instruction in it by an equivalent sequence of instructions in L1. The resulting program consists entirely of L1 instructions. The computer then executes the new L1 program instead of the old L2 program. This technique is called **translation**.

The other technique is to write a program in L1 that takes programs in L2 as input data and carries them out by examining each instruction in turn and executing the equivalent sequence of L1 instructions directly. This technique does not require first generating a new program in L1. It is called **interpretation** and the program that carries it out is called an **interpreter**.

Translation and interpretation are similar. In both methods instructions in L2 are ultimately carried out by executing equivalent sequences of instructions in L1. The difference is that, in translation, the entire L2 program is first converted to an L1 program, the L2 program is thrown away, and then the new L1 program is executed. In interpretation, after each L2 instruction is examined and decoded, it is carried out immediately. No translated program is generated. Both methods are widely used.

Rather than thinking in terms of translation or interpretation, it is often more convenient to imagine the existence of a hypothetical computer or **virtual machine** whose machine language is L2. If such a machine could be constructed cheaply enough, there would be no need for having L1 or a machine that executed programs in L1 at all. People could simply write their programs in L2 and have the computer execute them directly. Even though the virtual machine whose language is L2 is too expensive to construct out of electronic circuits, people can still write programs for it. These programs can either be interpreted or translated by a program written in L1 that itself can be directly executed by the existing computer. In other words, people can write programs for virtual machines, just as though they really existed.

To make translation or interpretation practical, the languages L1 and L2 must not be "too" different. This constraint often means that L2, although better than L1, will still be far from ideal for most applications. That L2 should be far from ideal is perhaps discouraging in light of the original purpose for creating it—namely, to relieve the programmer of the burden of having to express algorithms in a language more suited to machines than people. However, the situation is far from hopeless.

The obvious approach is to invent still another set of instructions that is more people-oriented and less machine-oriented than those of L2. This third set also forms a language, which we will call L3. People can write programs in L3 just as though a virtual machine with L3 as its machine language really existed. Such programs can either be translated to L2 or executed by an interpreter written in L2.

The invention of a whole series of languages, each one more convenient than its predecessors, can go on indefinitely until a suitable one is finally achieved. Each language uses its predecessor as a basis, so we may view a computer using this technique as a series of **layers** or **levels**, one on top of another, as shown in Fig. 1-1. The bottom-most language or level is the simplest and the highest language or level is the most sophisticated.

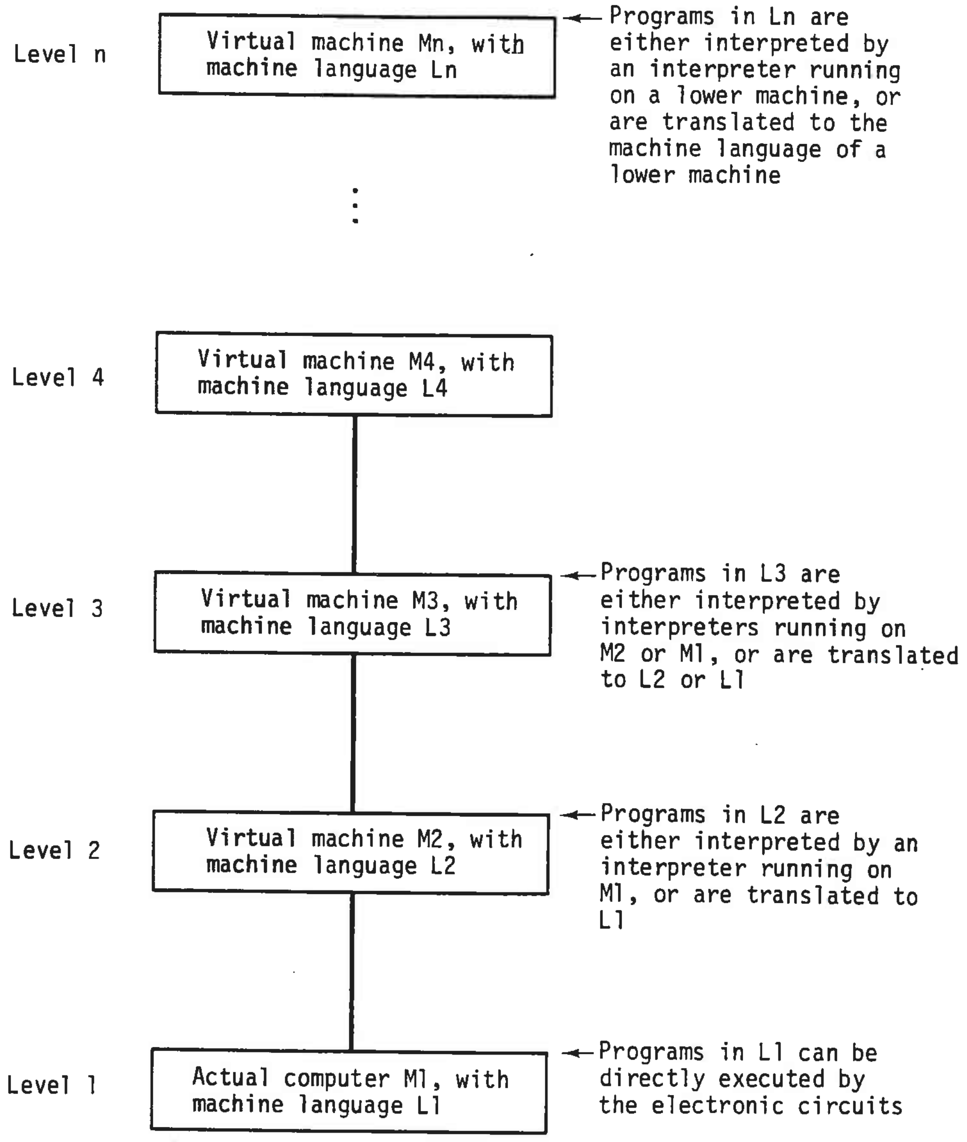

Level n
Virtual machine Mn, with machine language Ln
Programs in Ln are either interpreted by an interpreter running on a lower machine, or are translated to the machine language of a lower machine
Level 4
Virtual machine M4, with machine language L4
Level 3
Virtual machine M3, with machine language L3
Programs in L3 are either interpreted by interpreters running on M2 or M1, or are translated to L2 or L1
Level 2
Virtual machine M2, with machine language L2
Programs in L2 are either interpreted by an interpreter running on M1, or are translated to L1
Level 1
Actual computer M1, with machine language L1
Programs in L1 can be directly executed by the electronic circuits

Fig. 1-1. A multilevel machine.

1.1. LANGUAGES, LEVELS, AND VIRTUAL MACHINES

There is an important relation between a language and a virtual machine. Each machine has some machine language, consisting of all the instructions that the machine can execute. In effect, a machine defines a language. Similarly, a language defines a machine—namely, the machine that can execute all programs written in the language. Of course, the machine defined by a certain language may be enormously complicated and expensive to construct directly out of electronic circuits but we can imagine it nevertheless. A machine with Ada*, Pascal, or COBOL as its machine language would be a complex beast indeed but it is certainly conceivable, and perhaps in a few years such a machine will be considered trivial to build.

A computer with *n* levels can be regarded as *n* different virtual machines, each with a different machine language. We will use the terms "level" and "virtual machine" interchangeably. Only programs written in language L1 can be directly carried out by the electronic circuits, without the need for intervening translation or interpretation. Programs written in L2, L3, ..., L*n* must either be interpreted by an interpreter running on a lower level or translated to another language corresponding to a lower level.

A person whose job it is to write programs for the level *n* virtual machine need not be aware of the underlying interpreters and translators. The machine structure ensures that these programs will somehow be executed. It is of little interest whether they are carried out step by step by an interpreter which, in turn, is also carried out by another interpreter, or whether they are carried out directly by the electronics. The same result appears in both cases: the programs are executed.

Most programmers using an *n*-level machine are only interested in the top level, the one least resembling the machine language at the very bottom. However, people interested in understanding how a computer really works must study all the levels. People interested in designing new computers or designing new levels (i.e., new virtual machines) must also be familiar with levels other than the top one. The concepts and techniques of constructing machines as a series of levels and the details of some important levels themselves form the main subject of this book. The title *Structured Computer Organization* comes from the fact that viewing a computer as a hierarchy of levels provides a good structure or framework for understanding how computers are organized. Furthermore, designing a computer system as a series of levels helps to ensure that the resulting product will be well structured.

1.2. CONTEMPORARY MULTILEVEL MACHINES

Most modern computers consist of two or more levels. Six-level machines are not at all unusual, as shown in Fig. 1-2. Level 0, at the bottom, is the machine's true hardware. Its circuits carry out the machine language programs of level 1. For the sake of completeness, we should mention the existence of yet another level below our level 0. This level, not shown in Fig. 1-2, because it falls within the realm of electrical engineering (and is thus outside the scope of this book) is called the **device level**. At this level, the designer sees individual transistors, which are the lowest-level

*Ada is a trademark of the U.S. Department of Defense.

primitives for computer designers. (Of course, one can also ask how transistors work inside but that gets into solid-state physics.)

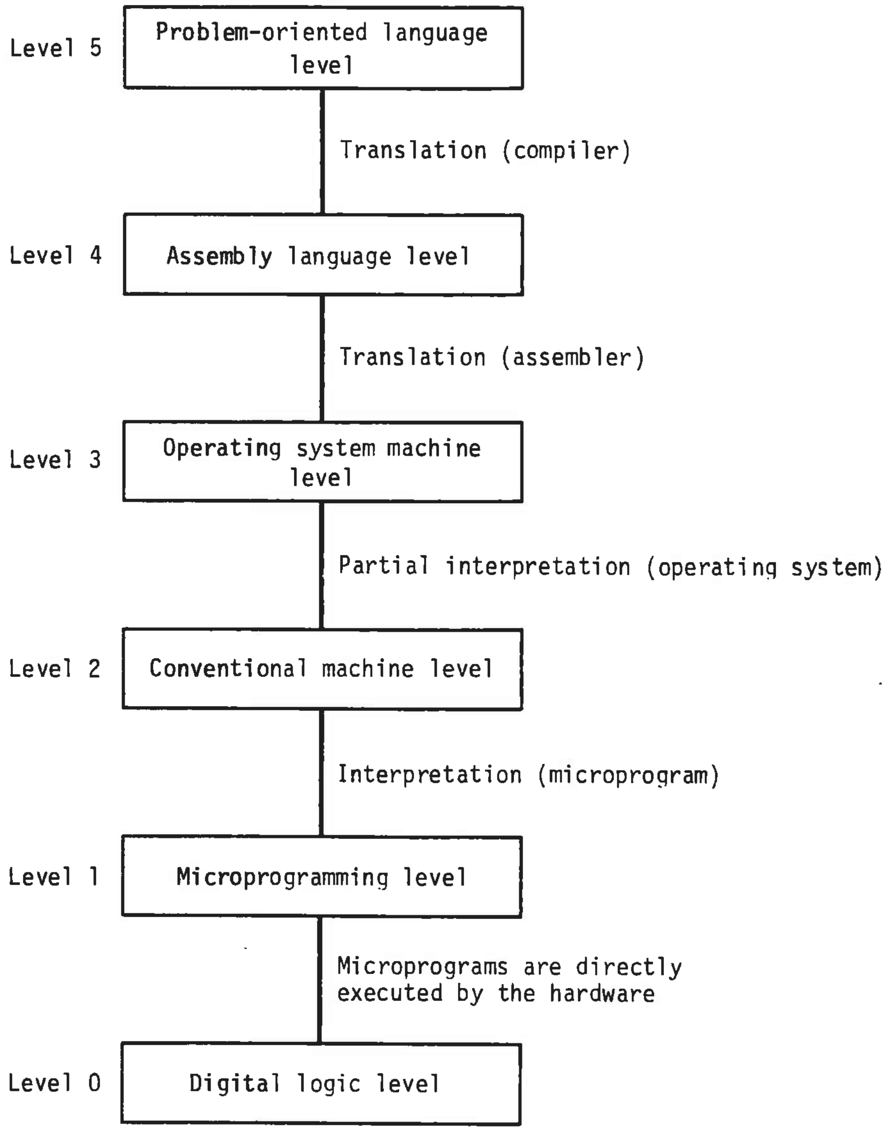

Level 5
Problem-oriented language level
Translation (compiler)
Level 4
Assembly language level
Translation (assembler)
Level 3
Operating system machine level
Partial interpretation (operating system)
Level 2
Conventional machine level
Interpretation (microprogram)
Level 1
Microprogramming level
Microprograms are directly executed by the hardware
Level 0
Digital logic level

Fig. 1-2. Six levels present on most modern computers. The method by which each level is supported is indicated below it, along with the name of the supporting program in parentheses.

At the lowest level that we will study, the **digital logic level**, the interesting objects are called **gates**. These gates are digital, unlike transistors, which are analog. Each gate has one or more digital inputs (signals representing 0 or 1) and computes as output some simple function of these inputs, such as AND or OR. Each gate is built up of at most a handful of transistors. We will examine the digital logic level in detail in Chap. 3. Although knowledge of the device level is something of a specialty, with the advent of microprocessors and microcomputers, more and more people are coming in contact with the digital logic level. For this reason we have included the latter in our model and devoted an entire chapter of the book to it.

The next level up is level 1, which is the true machine language level. In contrast to level 0, where there is no concept of a program as a sequence of instructions to be carried out, in level 1 there is definitely a program, called a **microprogram**, whose job it is to interpret the instructions of level 2. We will call level 1 the **microprogramming level**. Although it is true that no two computers have identical microprogramming levels, enough similarities exist to allow us to abstract out the essential features of the level and discuss it as though it were well defined. For example, few machines have more than 20 instructions at this level and most of these instructions involve moving data from one part of the machine to another, or making some simple tests.

Each level 1 machine has one or more microprograms that can run on it. Each microprogram implicitly defines a level 2 language (and a virtual machine, whose machine language is that language). These level 2 machines also have much in common. Even level 2 machines from different manufacturers have more similarities than differences. In this book we will call this level the **conventional machine level**, for lack of a generally agreed-upon name.

Every computer manufacturer publishes a manual for each of the computers it sells, entitled "Machine Language Reference Manual" or "Principles of Operation of the Western Wombat Model 100X Computer" or something similar. These manuals are really about the level 2 virtual machine, not the level 1 actual machine. When they describe the machine's instruction set, they are in fact describing the instructions carried out interpretively by the microprogram, not the hardware instructions themselves. If a computer manufacturer provided two interpreters for one of its machines, interpreting two different level 2 machine languages, it would need to provide two "machine language" reference manuals, one for each interpreter.

It should be mentioned that some computers, particularly older ones, do not have a microprogramming level. On these machines the conventional machine level instructions are carried out directly by the electronic circuits (level 0), without any level 1 intervening interpreter. As a result, level 1 and not level 2 is the conventional machine level. Nevertheless, we will continue to call the conventional machine level "level 2," despite these exceptions.

The third level is usually a hybrid level. Most of the instructions in its language are also in the level 2 language. (There is no reason why an instruction appearing at one level cannot be present at other levels as well.) In addition, there is a set of new instructions, a different memory organization, the ability to run two or more programs in parallel, and various other features. More variation exists between level 3 machines than between either level 1 machines or level 2 machines.

The new facilities added at level 3 are carried out by an interpreter running at level 2, which, historically, has been called an **operating system**. Those level 3 instructions identical to level 2's are carried out directly by the microprogram, not by the operating system. In other words, some of the level 3 instructions are interpreted by the operating system and some of the level 3 instructions are interpreted directly by the microprogram. This is what we mean by "hybrid." We will call this level the **operating system machine level**.

There is a fundamental break between levels 3 and 4. The lowest three levels are not designed for direct use by the average garden-variety programmer. They are intended primarily for running the interpreters and translators needed to support the higher levels. These interpreters and translators are written by people called **systems programmers** who specialize in designing and implementing new virtual machines. Levels 4 and above are intended for the applications programmer with a problem to solve.

Another change occurring at level 4 is the method by which the higher levels are supported. Levels 2 and 3 are always interpreted. Levels 4, 5, and above are usually, although not always, supported by translation.

Yet another difference between levels 1, 2, and 3, on the one hand, and levels 4, 5, and higher, on the other, is the nature of the language provided. The machine languages of levels 1, 2, and 3 are numeric. Programs in them consist of long series of numbers, which are fine for machines but bad for people. Starting at level 4, the languages contain words and abbreviations meaningful to people.

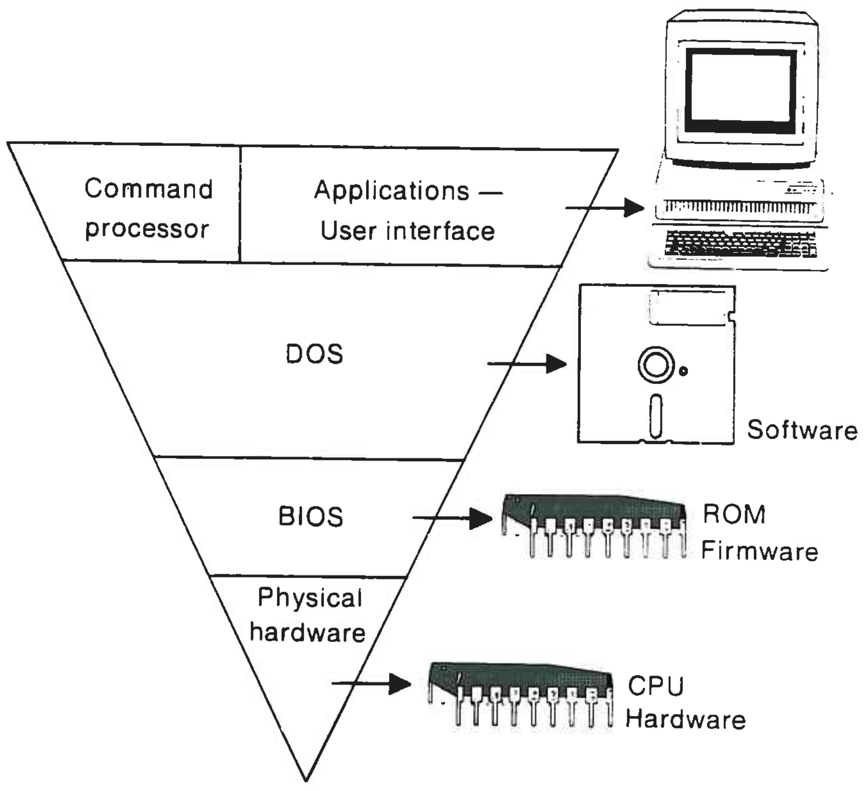
Level 4, the assembly language level, is really a symbolic form for one of the underlying languages. This level provides a method for people to write programs for levels 1, 2, and 3 in a form that is not as unpleasant as the virtual machine languages themselves. Programs in assembly language are first translated to level 1, 2, or 3 language and then interpreted by the appropriate virtual or actual machine. The program that performs the translation is called an **assembler**. Assembly language once was important but it is becoming less important as time goes on.

Level 5 consists of languages designed to be used by applications programmers with problems to solve. Such languages are called by many names, including **high-level languages** and **problem-oriented languages**. Literally hundreds of different ones exist. A few of the better known ones are Ada, ALGOL 68, APL, BASIC, C, COBOL, FORTRAN, LISP, Pascal, and PL/1. Programs written in these languages are generally translated to level 3 or level 4 by translators known as **compilers**, although occasionally they are interpreted instead.

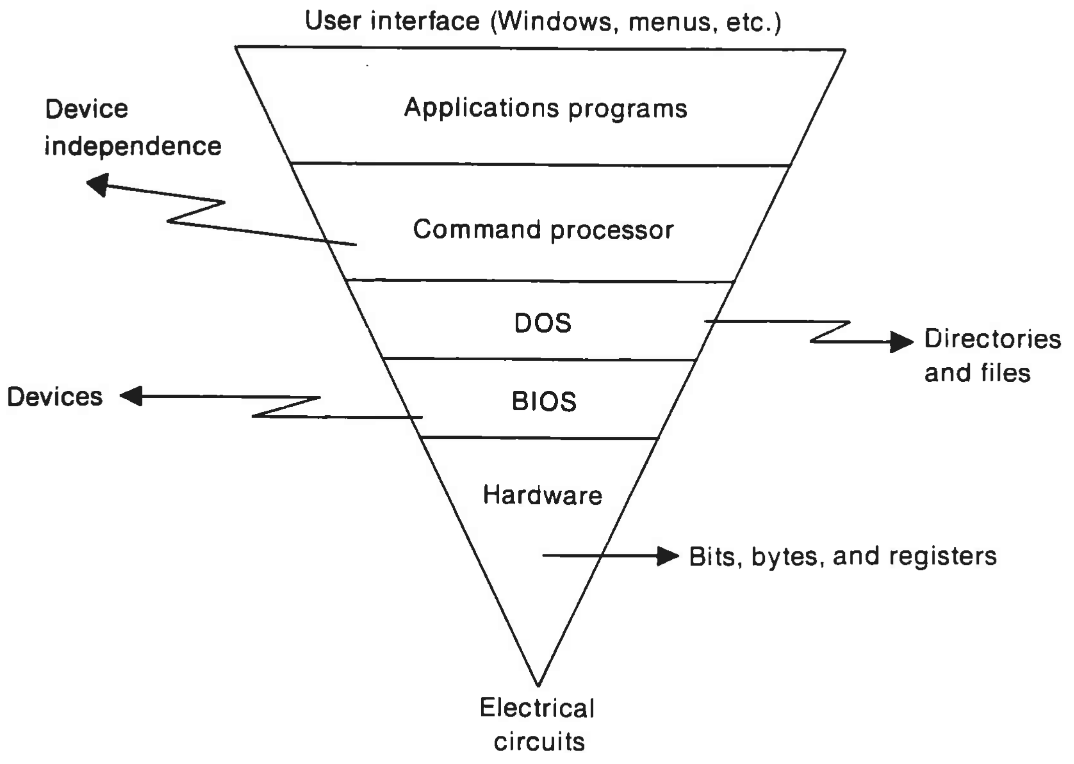
Levels 6 and above consist of collections of programs designed to create machines specifically tailored to certain applications. They contain large amounts of information about that application. It is possible to imagine virtual machines intended for applications in administration, education, computer design, and so on. These levels are an area of current research.

In summary, the key thing to remember is that computers are designed as a series of levels, each one built on its predecessor. Each level represents a distinct abstraction, with different objects and operations present. By designing and analyzing computers in this fashion, we are temporarily able to suppress irrelevant details and thus reduce a complex subject to something easier to understand.

3.1.b
The virtual machine hierarchy.



System layering.


User interface (Windows, menus, etc.)
Applications programs
Device independence
Command processor
DOS
Directories and files
Devices
BIOS
Hardware
Bits, bytes, and registers
Electrical circuits