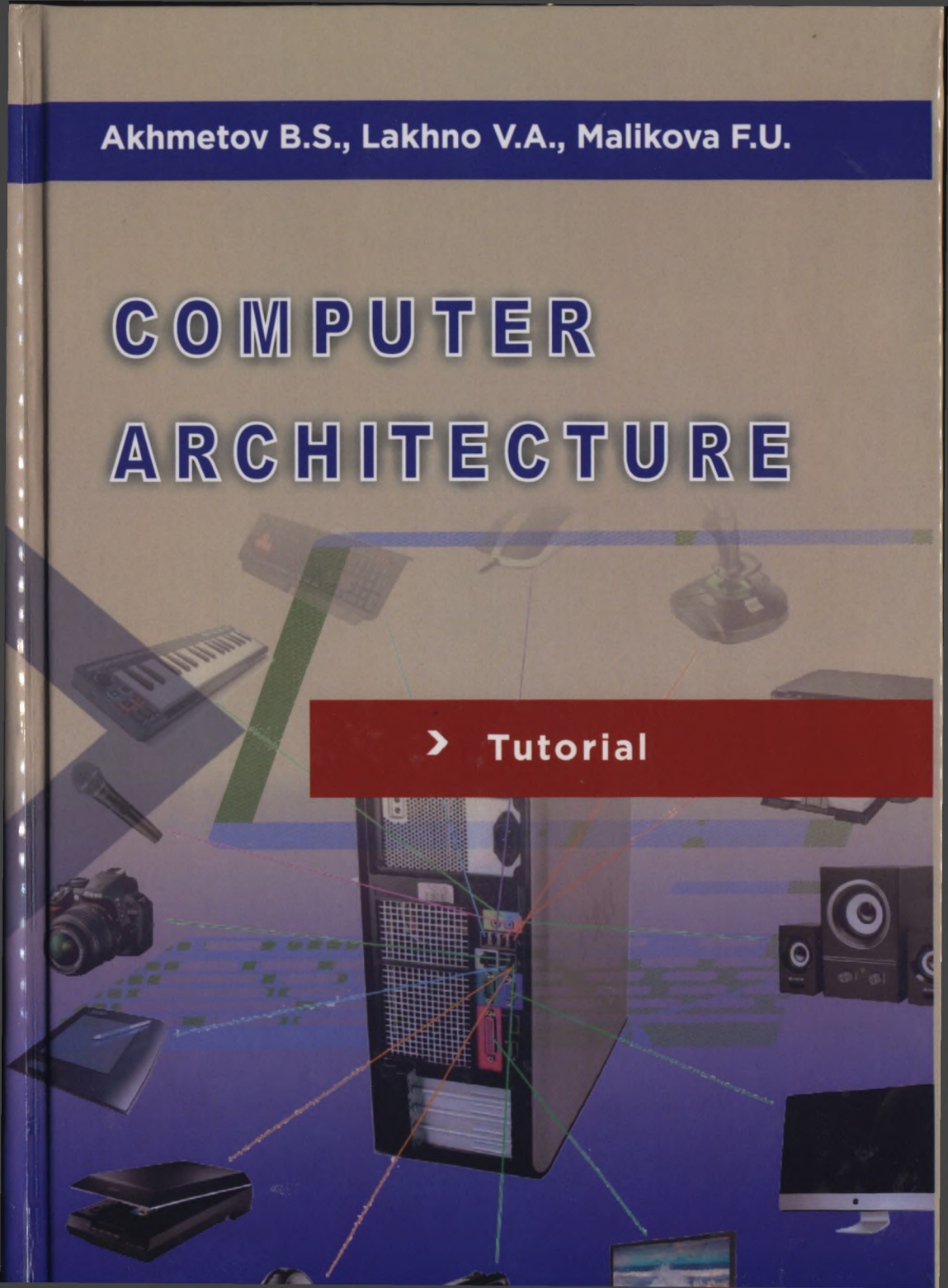


Akhmetov B.S., Lakhno V.A., Malikova F.U.

COMPUTER ARCHITECTURE

➤ Tutorial

The background of the cover is a collage of various computer hardware components. At the center is a vertical server tower. Radiating from it are several thin, colorful lines (red, green, blue, orange) that connect to other hardware items scattered around. These items include a keyboard, a mouse, a camera, a microphone, a scanner, a printer, and a set of speakers. The overall aesthetic is technical and interconnected, representing computer architecture.

MINISTRY OF EDUCATION AND SCIENCE OF THE REPUBLIC OF
KAZAKHSTAN

Almaty University of Power Engineering and Telecommunication

Akhetov B.S., Lakhno V.A., Malikova F.U.

COMPUTER ARCHITECTURE

Almaty 2019

UDC 004.031
LBC 32.973.202
A 27

Reviewers:

Utepbergenov I.T. - Doctor of technical sciences, professor;

Kartbaev T.S. – Dr. PhD;

Alibieva Zh.M. – Dr. PhD;

Akhmetov B.S.

A 27 Computer architecture. Tutorial/ B.S.Akhmetov, V.A.Lakhno, F.U.Malikova -
Almaty: AUPET, 2019. - 288 p. Fig. 109. Tab. 25. Bibliography - 23 titles.

ISBN 978-601-7307-74-5

The study of the discipline "Computer Architecture" is one of the important components of the professional training of modern specialists in the field of information technology. The rapid development of computers and other digital electronics leads to the saturation of almost all spheres of human activity.

In these conditions, for a specialist in computer or software engineering, knowledge of the basics of the computer hardware, its basic technical characteristics and functional capabilities is necessary.

This textbook on the discipline "Computer Architecture" is intended to help students in gaining knowledge about the functional components of modern computers, their purpose, types and characteristics, as well as forms of representing information in computers, the basics of machine mathematics and logic.

The training manual is intended for the preparation of bachelors and masters of the following educational programs of the field of information and communication technologies, as well as for all specialists who are interested in computer architecture.

ISBN 978-601-7307-74-5

Considered and approved at the Scientific Council of the AUPET
(protocol No.3, 17.09.2019)

Recommended by the Educational-methodical association of the Republican
educational-methodical council(protocol No.1, 23.10.2019)

UDC 004.031
LBC 32.973.202

© Akhmetov B.S., Lakhno V.A.,
Malikova F.U. 2019

© AUPET, 2019

© IE Balausa, 2019

CONTENT

INTRODUCTION.....	5
CHAPTER 1. THE CONCEPT OF COMPUTER ARCHITECTURE. CLASSIFICATIONS OF COMPUTER ARCHITECTURE.....	6
1.1. History of computer engineering	7
1.2. Computer architecture classification systems	18
1.3. The concept of information. Information measurement	22
CHAPTER 2. FUNCTIONAL NODES OF A COMPUTER.....	30
2.1. Arithmetic and logical device.....	31
2.2. Control device.....	34
2.3. Input-Output devices.....	37
CHAPTER 3. ARITHMETIC FOUNDATIONS OF A COMPUTER	41
3.1 Number systems and code concept.....	41
3.2. Arithmetic computer basics.....	46
3.2.1 Fixed point number representation form	47
3.2.2. Floating point number representation form	63
CHAPTER 4. LOGICAL FOUNDATIONS OF A COMPUTER	73
4.1. The logical basis for the construction and operation of computers	73
4.2. Electronic technology of computer logic elements	88
CHAPTER 5. MICROPROCESSOR SOFTWARE MODEL	101
CHAPTER 6. IA-32 PROCESSOR STRUCTURE.....	120
CHAPTER 7. AMD PROCESSOR ARCHITECTURE	138
CHAPTER 8. PROCESSOR OPERATING MODES	144
CHAPTER 9. MULTI-CORE PROCESSORS	158
CHAPTER 10. CO-PROCESSORS. METHODS FOR INFORMATION EXCHANGE BETWEEN CPU AND CO-PROCESSOR	170
CHAPTER 11. ORGANIZATION OF A MEMORY SUBSYSTEM IN A COMPUTER	179
11.1. Key parameters and characteristics of the storage devices	179
11.2. Storage devices classification	182
11.3. Cache memory	188
11.4. Random access memory (RAM).....	192
11.5. External computer memory	195
CHAPTER 12. RISC PROCESSORS	202
CHAPTER 13. MOTHERBOARDS AND CHIPSETS	208
CHAPTER 14. INTERRUPT AND EXCEPTION SYSTEM IN THE IA-32 ARCHITECTURE.....	215
14.1. Interrupt and exception system of the processor.....	215
14.2. Advanced Programmable Interrupt Controller (APIC).....	220
14.3. Interrupt Handling Based on 8259A Controller	221
CHAPTER 15. TYPES AND CHARACTERISTICS OF INTERFACES.....	225
15.1. System interfaces	225
15.2. Storage device interface	229

15.3. Interfaces SCSI, RS-232C, IEEE 1284, USB, FireWire.....	231
15.4. Wireless Interfaces.....	237
CHAPTER 16. INPUT-OUTPUT DEVICES.....	243
CHAPTER 17. NEW DIRECTIONS OF COMPUTER DEVELOPMENT.....	254
CHAPTER 18. BRIEF REVIEW ABOUT THE ASSEMBLER	265
18.1. The structure of the program in assembler	265
18.2. Examples of programming in assembler.....	268
18.2.1. Built-in assembler	268
18.2.2. Examples in MASM and FASM.....	277
REFERENCES	284

INTRODUCTION

On the modern computer market there is a wide variety of different types of computers. Therefore, it is possible to assume that the consumer has a question - how to assess the capabilities of a particular type (or model) of a computer and its distinguishing features from computers of other types (models). Consideration of only the structural diagram of a computer for this is not enough, since it fundamentally differs little from different machines: all computers have RAM, a processor, and external devices. There are various ways, means and resources using which the computer operates as a single mechanism. In order to put together all the concepts that characterize a computer in terms of its functional program-controlled properties, there is a special term - computer architecture. For the first time, the concept of computer architecture began to be mentioned with the advent of the 3rd generation machines for their comparative evaluation.

This tutorial, consisting of 2 parts, discusses the basic elements of the modern computers architecture, as well as the fundamentals of machine mathematics and logic.

The first part includes sections devoted to the mathematical and logical fundamentals of computer operation, as well as a description of the software model and microprocessors structure of the IA-32 and AMD architecture.

In the second part of the tutorial, the following topics are discussed: processor operating modes, multi-core processor architecture, organization of the input-output subsystem, the basics of assembly language programming, etc.

CHAPTER 1. THE CONCEPT OF COMPUTER ARCHITECTURE. CLASSIFICATIONS OF COMPUTER ARCHITECTURE

Computer architecture covers a wide range of problems associated with the construction of a complex of hardware and software means and many other factors. Among these components, the most important are: cost, sphere of application, functionality, ease of use.

The main components of the computer architecture can be represented in the form of the circuit shown in Fig. 1.1.

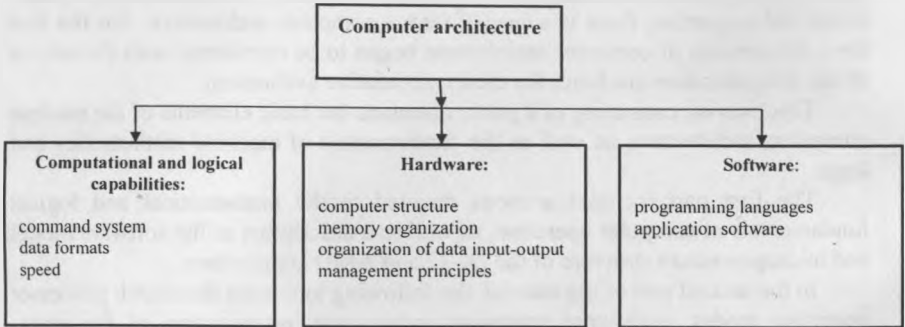


Fig. 1.1. Key components of computer architecture

The architecture of a computing tool should be distinguished from its structure. The structure of a computing tool determines its specific composition at a certain level of detailization (devices, blocks, nodes, etc.) and describes the connections within the tool in its entirety. Architecture, however, defines the rules for the interaction of the components of a computing tool, the description of which is carried out to the extent necessary for the formation of the rules for their interaction. It regulates not all communications, but the most important ones that should be known for the most competent use of this tool.

So the computer user does not care on which elements the electronic circuits are executed, how the commands are implemented, etc. Another thing is important: what features are provided to the user, how these features are related to the structural features of the computer, how the characteristics of the individual devices that make up the computer are related, and what effect they have on the general characteristics of the machine. In other words, the computer architecture really reflects a range of issues related to the general design and construction of computers and their software.

Therefore, during the study the subject, we will, whenever possible, consider all components of the computer, including the structure of the computer, memory organization, organization of data input-output, principles of computer control, command systems, and the software composition. Before starting to study directly computers, firstly, we will study when and why they appeared, and in order to

understand the basics of computers better, we will briefly dwell on what they work with and what they process, i.e. - information and methods for its presentation in computers.

1.1. History of computer engineering

The beginning of technology development is considered to be from Blaise Pascal, who in 1642 invented a device that mechanically performs the addition of numbers. His machine was designed to work with 6-8 bit numbers and could only add and subtract, and also had a better way than everything else before, to fix the result [1, 5, 10]. Pascal's ideas had a huge impact on many other inventions in the field of computer technology.

The next stage in the development of computer technology and information theory is associated with the outstanding German mathematician and philosopher Gottfried Wilhelm Leibniz, who in 1672 expressed the idea of mechanical multiplication without sequential addition. A year later, at the Paris Academy, he introduced a machine that made it possible to perform mechanically four arithmetic operations [1, 10].

In 1812, the English mathematician Charles Babbage began working on the so-called difference machine, which was supposed to calculate any functions, including trigonometric ones, as well as make tables. However, due to lack of funds, this machine was not finished, and its mechanical part was handed over to the Royal College Museum in London, where it is stored as an exhibit. However, this failure did not stop Babbage. In 1834, he embarked on a new project to create an analytical machine that was supposed to perform calculations without human intervention. According to experts [1, 5, 6, 10, etc.], Babbage's merit lies in the fact that he first proposed, and partially realized, the idea of software-controlled computing.

In 1818, Thomas K. designed a calculating machine, focusing on the manufacturability of the mechanism, and called it an arithmometer. Within three years, 16 arithmometers were manufactured in Thomas' workshops. Thus, Thomas laid the foundation for counting engineering. His arithmometers were produced for a hundred years, constantly improving and changing the name from time to time [1, 10].

Since the 19th century, arithmometers have been very widely used. They performed even very complex calculations, for example, calculations of ballistic tables for artillery firing. By 1914, in the Russian Empire there were more than 22 thousand arithmometers [1].

Previously, when mankind did not know about electric and magnetic phenomena or did not yet know how to use them, the most accessible, and, therefore, convenient, was the mechanical form of representing information in computing devices. In arithmometers, operations on numbers were performed with the help of wheels, which at adding a unit rotated 360° and with the help of a pin set in motion the next wheel according to seniority each time when the number 9 went to the number 0 (a dozen accumulated). However, mechanical devices are bulky,

expensive and inertial (with their help it is impossible to create universal and high-speed computers). Therefore, now in all computers, electrical signals (most often DC voltage) are used as the main form of information representation. Computers were invented long time ago. In those days, electronics was not even mentioned. The first computers were vacuum tubes and took up a lot of space. However, at that time the basic principles of computer operation were laid down, which are still valid. Their essence is as follows. Data is transmitted using some kind of a signal by the method "there is a signal or not" or, in other words, "on or off." Due to this, there was appeared a "bit". A bit is a unit of information that can take the value 0 or 1, that is, "on or off". Eight bits are combined into bytes, one byte is 8 bits. Why exactly 8? Yes, because the first computers were eight-bit and could only work with 8 bits at a time, for example, 010000111. All the first zeros can be deleted, so the number 010000111 can be written as - 10000111. This is the same as in the usual decimal number system, where each digit can take values from 0 to 9. Here, too, no one will write the number 6743 as 00006743.

In 1 byte, you can write any number from 0 to 255. The specified range of numbers is quite small. Therefore, there are often used larger gradations:

- two bytes = a word;
- two words = a double word.

In the first decades of the 20th century, designers paid attention to the possibility of using new elements in electromagnetic devices — electromagnetic relays. In 1941, a German engineer Konrad Zuse, created a computing device that runs on such relays.

Almost simultaneously, in 1943, American Howard Aiken, using Babbage's works based on 20th-century technology - electromechanical relays - created the legendary Harvard Mark -1 (Mark-1, and later Mark-2) at one of IBM's enterprises. Mark-1 had a length of 15 meters and a height of 2.5 meters, contained 800 thousand parts, had 60 registers for constants, 72 memory registers for addition, a central unit of multiplication and division, could calculate elementary transcendental functions. The machine worked with 23-digit decimal numbers and performed addition operations in 0.3 seconds, and multiplication in 3 seconds. However, Aiken made two mistakes: the first was that both of these machines were more electromechanical than electronic; the second is that Aiken did not adhere to the concept that programs should be stored in the computer's memory, like the data obtained [10].

At the same time, in England, the first relay computer began to work, which was used to decrypt messages transmitted by a German encoded transmitter. By the middle of the 20th century, the need for automation of computing (including for military purposes - ballistics, cryptography, etc.) became so great that several groups of researchers worked on the creation of machines like Mark-1 and Mark-2 in different countries.

Work on the creation of the first electronic computer was apparently begun in 1937 in the USA by professor John Atanasov, a Bulgarian by birth. This machine was specialized and intended for solving problems of mathematical physics. During the development, Atanasov created and patented the first electronic

devices, which were subsequently used quite widely in the first computers. The full project of Atanasov was not completed, but after three decades, as a result of the trial, the professors were recognized as the founder of electronic computer technology [1, 5, 10].

Beginning from 1943, a group of specialists led by Howard Aiken, J. Mauchly, and P. Eckert in the USA began to design a computer based on electron tubes, rather than electromagnetic relays [1, 10]. This machine was named ENIAC (Electronic Numeral Integrator And Computer) and it operates a thousand times faster than the Mark-1. ENIAC contained 18 thousand vacuum tubes, occupied an area of 9-15 meters, weighed 30 tons and consumed a capacity of 150 kilowatts. ENIAC also had a significant disadvantage - it was controlled using a patch panel, it had no memory, and in order to set up a program, it was necessary to connect the wires in the right way for several hours or even days. The worst of all the disadvantages was the terrifying unreliability of the computer, since about a dozen vacuum tubes could fail during the day of work.

In order to simplify the process of setting programs, Mauchly and Eckert began to design a new machine that could store the program in its memory. In 1945, the famous mathematician John von Neumann was involved in the work, who prepared a report on this machine. In this report, von Neumann clearly and simply formulated the general principles of functioning of universal computing devices, i.e. computers. This is the first operating machine, created on vacuum tubes, was officially put into operation on February 15, 1946. They tried to use this machine to solve some problems prepared by von Neumann and related to the atomic bomb project. Then it was transported to the Aberdeen Proving Ground, where it worked until 1955.

ENIAC became the first representative of the 1st generation of computers. Any classification is conditional, but most experts agreed that the generation should be distinguished on the basis of the elemental base on which the machines were created.

The report of von Neumann and his colleagues G. Goldstein and A. Berks (June 1946) formulated requirements for the computer structure. Let note the most important of them [1, 6, 10, 12, 16]:

- machines on electronic elements should work not in decimal, but in binary number system;
- the program, like the initial data, must be located in the machine's memory;
- the program, like numbers, should be written in binary code;
- difficulties in the physical implementation of the storage device, whose speed corresponds to the speed of logical circuits, require a hierarchical organization of memory (that is, allocation of operational, intermediate and long-term memory);
 - an arithmetic device (processor) is constructed on the basis of circuits that perform the addition operation; the creation of special devices for performing other arithmetic and other operations is impractical;
 - the machine uses the parallel principle of organization of the computational process (operations on numbers are performed simultaneously for all digits).

Almost all the von Neumann recommendations were subsequently used in the machines of the first three generations, their combination was called "von Neumann architecture". The first computer in which von Neumann principles were embodied was created in 1949 by the English researcher Maurice Wilkes. Since then, computers have become much more powerful, but the vast majority of them were made in accordance with the principles that John von Neumann set forth in his report in 1945.

New first-generation machines replaced each other pretty quickly. In 1951, there was launched the first Soviet electronic computer MESM, with an area of about 50 square meters. MESM had 2 types of memory: random access memory, in the form of 4 panels 3 meters high and 1 meter wide; and long-term memory in the form of a magnetic drum with a volume of 5000 numbers. In total, MESM had 6000 electronic tubes, and it was possible to work with them only after 1.5-2 hours after turning on the machine. Data input was carried out using magnetic tape, and output – by digital printing device, conjugate with memory. MESM could perform 50 mathematical operations per second, memorize 31 numbers and 63 commands in RAM (there were 12 different commands in total), and consumed power equal to 25 kilowatts.

In 1952, the American EDWAC machine appeared. It is also worth noting the English computer EDSAC (Electronic Delay Storage Automatic Calculator) created earlier in 1949, the first machine with a stored program. In 1952, Soviet designers put into operation the BESM, the fastest machine in Europe, and in the following year in the USSR there began operating "Strela" - the first high-class production machine in Europe. Among the creators of domestic machines, the names of S.A. Lebedeva, B.Ya. Bazilevsky, I.S. Brooke, B.I. Rameeva, V.A. Melnikova, M.A. Kartseva, A.N. Mamlina. In the 1950s, other computers appeared: Ural, M-2, M-3, BESM 2, and Minsk 1, which embodied increasingly progressive engineering solutions.

Projects and implementation of Mark-1, EDSAC and EDVAC machines in England and in the USA, MESM in the USSR laid the foundation for the development of work on the creation of vacuum tube technology computers - first-generation serial computers. The development of the first electronic serial machine UNIVAC (Universal Automatic Computer) was started around 1947 by Eckert and Mauchly. The first model of the machine (UNIVAC-1) was created for the US Census Bureau and put into operation in spring of 1951. The synchronous, sequential computer UNIVAC-1 was created on the basis of the ENIAC and EDVAC computers. It worked with a frequency of 2.25 MHz and contained about 5000 electronic tubes.

Compared with the USA, the USSR and England the development of electronic computers in Japan, in the Federal Republic of Germany and Italy was delayed. The first Japanese "Fujik" machine was put into operation in 1956, mass production of computers in Germany began only in 1958.

The capabilities of the machines of the first generation were quite modest. So, their performance according to current concepts was small: from 100 (Ural-1) to 20,000 operations per second (M-20 in 1959). These figures were determined

primarily by the inertia of vacuum tubes and by the imperfection of storage devices. The amount of RAM was extremely small - an average of 2,048 numbers (words), this was not enough even to accommodate complex programs, even data. Intermediate memory was organized on bulky and slow-moving magnetic drums of relatively small capacity (5,120 words on BESM-1). Printing devices, as well as data input units, worked slowly. If we dwell on input-output devices in more detail, then we can say that since the beginning of the first computers, there has been revealed a contradiction between the high speed of central devices and the low speed of external devices. In addition, the imperfection and inconvenience of these devices was also revealed. The first data carrier in computers, as you know, was a punch card. Then came punching paper tapes or just punched tapes. They came from telegraph technology after in the beginning of the 19th century the father and son from Chicago, Charles and Howard Krama, invented teletype.

First-generation computers quickly left the stage because they did not find wide commercial application due to unreliability, high cost, and programming difficulties.

Semiconductors became the elemental base of the second generation. Without a doubt, transistors can be considered one of the most impressive miracles of the XX century.

A patent for opening a transistor was issued in 1948 to Americans D. Bardin and U. Brattein, and eight years later they, together with theorist W. Shockley, became Nobel Prize winners. Speed switching of the first transistor elements turned out to be hundreds of times higher than of tube ones, as its reliability and economy too. For the first time, memory on ferrite cores and thin magnetic films began to be widely used; there were tested inductive elements - parametrons.

The first onboard computer for installation on an intercontinental rocket - "Atlas" - was put into operation in the United States in 1955. The machine used 20 thousand transistors and diodes, it consumed 4 kilowatts. In 1961, the "Burrows" ground-based "STRETCH" computers controlled the spaceflight of "Atlas" rockets, and IBM's machines controlled the flight of astronaut Gordon Cooper. Under the control of the computer, there took place flights of unmanned vehicles of the "Ranger" type to the Moon in 1964, as well as the "Mariner" vehicle to the Mars. Similar functions were performed by Soviet computers.

In 1956, IBM company developed floating magnetic air cushion heads. Their invention made it possible to create a new type of memory - disk storage devices, the significance of which was fully appreciated in the following decades of the development of computer technology. The first disk storage devices appeared in the IBM-305 and RAMAC machines. The last one had a package consisting of 50 magnetically coated metal disks that rotated at a speed of 12,000 rpm. On the surface of the disc there were 100 tracks for recording data, for 10,000 signs each [1].

The first serial universal transistor computers were released in 1958 simultaneously in the USA, Germany and Japan.

In the Soviet Union, the first lampless machines "Setun", "Razdan" and "Razdan 2" were created in 1959-1961. In the 60s, Soviet designers developed

about 30 models of transistor computers, most of which began to be mass-produced. The most powerful of them - "Minsk 32" performed 65 thousand operations per second. Entire families of machines appeared: Ural, Minsk, BESM [10].

The record holder among second-generation computers was BESM 6, which had a speed of about a million operations per second - one of the most productive in the world. The architecture and many technical solutions of this computer were so advanced and ahead of their time that it was successfully used almost until our time.

Especially for the automation of engineering calculations at the Institute of Cybernetics of the Academy of Sciences of the Ukrainian SSR under the leadership of Academician V.M. Glushkov there were developed computers MIR (1966) and MIR-2 (1969). An important feature of the MIR-2 machine was the use of a television screen for visual control of information and a light pen, with the help of which it was possible to correct data directly on the screen.

The construction of computer systems, which included about 100 thousand switching elements, would be simply impossible on the basis of lamp technology. The priority in the invention of integrated circuits, which became the elemental base of third-generation computers, belongs to the American scientists D. Kilby and R. Neuss, who made this discovery independently of each other. Mass production of integrated circuits began in 1962, and in 1964 the transition from discrete to integrated elements began to take place rapidly.

Despite the successes of integrated technology and the emergence of mini-computers, large machines continued to dominate in the 60s. Thus, the third generation of computers, originating inside the second generation, gradually grew out of it.

The first mass series of machines on integrated elements began to be produced in 1964 by IBM. This series, known as IBM-360, had a significant impact on the development of computer technology in the second half of the 60s. It combined a whole family of computers with a wide range of performance, moreover, compatible with each other. The last one meant that it became possible to connect machines into complexes, and also without any alterations to transfer programs written for one computer to any other in this series. Thus, there was identified for the first time a commercially viable requirement for standardizing computer hardware and software.

In the USSR, the first serial computer on integrated circuits was the "Nairi-3" machine, which appeared in 1970. In the second half of the 60s, the Soviet Union, together with the CMEA countries, began developing a family of universal machines similar to the IBM-360 system. In 1972, mass production began of the starting, least powerful model of the Unified System - the EC-1010 computer, and a year later - five other models. Their performance ranged from ten thousand (EC-1010) to two million (EC-1060) operations per second.

Within the framework of the third generation in the USA, a unique machine "ILLIAK-4" was created, in the structure of which in the initial version it was planned to use 256 data processing devices made on monolithic integrated circuits.

Later, the project was changed due to a rather high cost (more than \$ 16 million). The number of processors had to be reduced to 64, and also to switch to integrated circuits with a small degree of integration. A shortened version of the project was completed in 1972, the nominal speed of "ILLIAC-4" amounted to 200 million operations per second. For almost a year, this computer has been a champion in computing speed.

During the development of the third generation an extremely powerful computing industry arose, which began to produce computers for mass commercial use in large quantities. Computers are increasingly being incorporated into information systems or production management systems. They acted as an obvious lever of the modern industrial revolution.

The beginning of the 70s marks the transition to the fourth-generation computers - on ultra-large integrated circuits (ULIC). Another sign of a new generation of computers is a sharp change in architecture.

The equipment of the fourth generation gave rise to a qualitatively new computer element - a microprocessor. In 1971, they came up with the idea of limiting the capabilities of the processor by laying in it a small set of operations, the microprograms of which must be previously entered into the permanent memory. Estimates have shown that the use of a 16 kilobit read-only memory will eliminate 100,200 conventional integrated circuits. So the idea of a microprocessor arose, which can be implemented even on a single chip, and the program is written to its memory forever. At that time, in an ordinary microprocessor, the integration level corresponded to a density of about 500 transistors per square millimeter, and there was achieved very good reliability.

By the mid-70s, the situation in the computer market began to change abruptly and unexpectedly. Two concepts of computer development were clearly distinguished. Supercomputers became the embodiment of the first concept, and personal computers became the second one.

Beyond the fourth-generation large computers on ultra-large integrated circuits, American machines "Cray-1" and "Cray-2", as well as Soviet models "Elbrus-1" and "Elbrus-2" were especially distinguished. Their first samples appeared at about the same time - in 1976. All of them belong to the category of supercomputers, as they have extremely achievable characteristics for their time and a very high cost.

The fourth-generation machines made a departure from von Neumann architecture, which was the leading sign of the vast majority of all previous computers.

Multiprocessor computers, due to their enormous speed and architectural features, are used to solve a number of unique problems in hydrodynamics, aerodynamics, long-term weather forecasting, etc. Along with supercomputers, the fourth generation includes many types of mini-computers, which also rely on the elemental base of ultra-large integrated circuits.

Although personal computers belong to the fourth generation of computers, the possibility of their wide distribution, despite the achievements of ULIC technology, would remain very small.

In 1970, an important step was taken towards a personal computer - Marchian Edward Hoff, who worked at Intel, designed an integrated circuit that was similar in function to the central processor of a large computer. So there was the first Intel 4004 microprocessor, which was put on sale in 1971. This was a real breakthrough, because the Intel 4004 microprocessor less than 3 cm in size was more productive than the giant machines of the 1st generation. But the Intel 4004 capabilities were much more modest than that of the central processor of large computers of that time — it worked much slower and could only process 4 bits of information at a time (large computer processors processed 16 or 32 bits at a time), but its cost was in tens of thousands of times cheaper.

In 1972, the Intel 800 8-bit microprocessor appeared. The size of its registers corresponded to the standard unit of digital information - byte. The Intel 8008 processor was a simple development of the Intel 4004.

But in 1974, a much more interesting Intel 8080 microprocessor was created. From the very beginning of development, it was laid down as an 8-bit chip. He had a wider array of microcommands (the multitude of microcommands 8008 was expanded). In addition, it was the first microprocessor which could divide numbers. And until the end of the 70s, the Intel 8008 microprocessor became the standard for the microcomputer industry.

Several engineers of the company had ideas for improving the 8080. They left Intel to implement them. They organized Zilog Corporation, which presented the world with the Z80 microprocessor. In fact, the Z80 was a further development of the 8080 microprocessor. The number of its commands was simply increased, which made it possible to create and to use standard operating systems on personal computers.

Between 1970 and 1975, computer lovers' clubs sprang up throughout the United States. The most noteworthy was the Homebrew Computer Club, formed in March 1975 in Menlo Park, California. Its first members included Steve Jobs and Steve Wozniak, who later founded Apple Macintosh company.

Therefore, when the first microcomputer appeared, there was immediately a huge demand for it among thousands of fans. The first microcomputer was the "Altair-8800" created in 1974 by a small company in Albuquerque, New Mexico. This computer was sold for about \$500. And although its capabilities was very limited (RAM was only 256 bytes, there was no keyboard and screen), and there were also serious operational disadvantages, the Altair-8800 became a bestseller. Later, buyers themselves supplied this computer with additional devices: a monitor for outputting information, a keyboard, memory expansion units, etc. Soon, these devices began to be produced by other companies.

At the end of 1975, Paul Allen and Bill Gates (future founders of Microsoft) created the Basic language - interpreter for the "Altair" computer, which allowed users to simply communicate with the computer and easily write programs for it. It also contributed to the popularity of personal computers.

The success of the "Altair-8800" forced many companies also to engage in the production of personal computers. Personal computers began to be sold fully

equipped, with a keyboard and a monitor, the demand for them amounted to tens, and then hundreds of thousands of pieces per year.

In 1979, Intel released the new Intel 8086/8088 microprocessor. Then the first Intel 8087 co-processor appeared. The frequencies at which the Intel-8086/8088 microprocessor could work were 4.77, 8 and 10 MHz.

In the late 70s, the spread of personal computers even led to some decrease in demand for large computers and mini-computers. This became a matter of serious concern for IBM, and in 1979, IBM decided to try its hand at the personal computer market.

First of all, Intel 8088, the then latest 16-bit microprocessor, was chosen as the main microprocessor of the computer. Its use significantly increased the potential of the computer, since the new microprocessor made it possible to work with 1 MB of memory, and then all available computers were limited to 64 KB. Other components of various companies were used in the computer, and its software was entrusted to be developed by small Microsoft company. And thus, in 1981, the first version of the operating system for the IBM PC computer appeared - MS DOS 1.0.

In August 1981, a new computer called "IBM Personal Computer" was officially presented to the public and shortly thereafter, it gained great popularity among users. The IBM PC had 64 Kb of RAM, a tape recorder for loading / saving programs and data, a drive and an integrated version of the BASIC language.

After one or two years, the IBM PC took a leading position on the market, displacing 8-bit computers.

IBM did not make its computer a single integral device and did not protect it with patents. On the contrary, it assembled a computer from independently manufactured parts and did not keep the specifications of these parts and how to connect them in secret. On the contrary, the design principles of IBM PC were available to everybody. This approach, called the "*principle of open architecture*", ensured tremendous success for the IBM PC, although it deprived IBM of the opportunity to use the benefits of this success alone.

A new generation of microprocessors replaces the previous one every two years and becomes obsolete in 3-4 years. The microprocessor, along with other microelectronic devices, allows to create fairly economical information systems.

The reason for the popularity of the microprocessor is that with their appearance there is no longer any need for special information processing schemes, it is enough to program its function and enter the microprocessor into the read-only memory (ROM).

After a short period of time, the IBM PC model was improved. The new modification was called the "expanded" IBM PC/XT (*Personal Computer/ eXTended version*). In this modification, manufacturers abandoned the use of a tape recorder as a storage device, added a second floppy drive, as well as the ability to use a hard disk with a capacity of 10-30 MB. Nowadays, having a hard drive in an PC/XT is almost mandatory. The model was based on the use of the same microprocessor - Intel 8088.

In 1982, Intel launched the new Intel 80286 microprocessor, which had 134 thousand transistors and was developed using 1.5 micron technology (microns - micrometers or microns). He could work with 16 MB of RAM at frequencies: 8, 12 and 16 MHz. Its fundamental innovation - protected mode and virtual memory up to 1 GB in size - did not find mass application, the processor was mostly used as a very fast 8088.

In the same year, a new computer model was released under the name IBM PC/AT (*Personal Computer/Advanced Technolog* - "Advanced Technology PC"). Due to the use of the new microprocessor with the 80287 co-processor, the system performance increased more than in two times. It is equipped with floppy drives of a new type (with integrated volume of stored information), a hard disk of 40 MB or more. The bus of the PC motherboard is expanded to 16 bits.

The heat of competition forced IBM developers ultimately to abandon the principle of "open architecture". The new family of IBM PC models is called PS/2 ("Personal System/2"). It was absolutely incompatible with the first generation at the hardware level, but retains compatibility at the software level. The first models of the PS/2 family used the Intel 80286 microprocessor and actually copied the PC AT, but were based on a different architecture.

In 1985 there were appeared Intel 80386SX and Intel 80386DX. It discovered a class of 32-bit processors. The Intel 80386 microprocessor had 275 thousand transistors and was manufactured using 1.5 micron technology.

In 1987, Microsoft developed version 3.3 (3.30) of the MS DOS operating system, which became the standard for the next 3-4 years.

In 1989, Intel released a new microprocessor 80486SX/DX/DX2, which had 1.2 million transistors on a chip, and manufactured using 1 micron technology.

In June 1991, Microsoft released MS-DOS 5.0, which has its own characteristics: it has improved shell menu interfaces, a full-screen editor, utilities on disk and the ability to change tasks. The subsequent versions of MS-DOS 6.0, MS-DOS 6.21 and MS-DOS 6.22, in addition to the standard set of programs, include backup programs, an antivirus program, and other improvements in the operating system.

In 1992, the Intel 80486DX4 processor appeared with a frequency up to 100 MHz.

The following major development dates for Microsoft operating systems went along with the development of the personal computer hardware.

April 6, 1992 - Windows 3.1 OS released, which became the most popular in the USA (by the number of installations).

October 27, 1992 - Windows for Workgroups 3.1 released. It integrates features aimed at serving network users and workgroups, including email delivery, file and printer sharing, and scheduling. Version 3.1 was the forerunner of the small LAN boom, but failed commercially, receiving the offensive nickname "Windows for Warehouse".

November 8, 1993 - Windows for Workgroups 3.11 released.

In 1993, the first Pentium processors with a frequency of 60 and 66 MHz appeared - these were 32-bit processors with a 64-bit data bus. In 1995, processors

for 120 and 133 MHz appeared, made using 0.35 micron technology. 1996 is called the year of Pentium - 150, 166 and 200 MHz processors appeared, and Pentium became an ordinary processor for PCs of wide application.

On August 24, 1995, Windows 95 was released.

July 25, 1998 Microsoft launches Windows 98, the latest version of Windows based on the old kernel, which runs on the foundation of DOS. Windows 98 is integrated with Internet Explorer 4 and is compatible with many, from USB to ACPI power management specifications.

On October 6, 1998, Intel announced a 450 MHz Pentium® II Xeon™ processor for dual-processor (dual-channel) servers and workstations.

The last decade is characterized not only by the rapid growth of computer productivity, but also by the emergence of new approaches to increase their productivity. The main characteristics of computers of different generations are presented in table 1.1 [1, 5, 10, 16, 22].

Table 1.1

The main characteristics of computers of various generations

generation	1	2	3	4	5
Period of development, years	1946-1960	1955-1970	1965-1980	1980 – p.t.	from 1990 r.
Elementary base	Vacuum electronic lamps	Semiconductor diodes and transistors	Integrated circuits	Extra Large Integrated Circuits	Nowadays, several fundamentally different directions are being developed:
Architecture	Von Neumann architecture	Multiprogram mode	Local area networks of computers, computer systems for collective use	Multiprocessor systems, personal computers, global networks	1) an optical computer in which all components will be replaced by their optical counterparts (optical repeaters, fiber-optic communication lines, memory based on the principles of holography;
Spess, op/c	10 – 20 th. op/c	100-500 - th. op/c	≈ 1 mln. op/c	Tens and hundreds of millions of op/c	2) a molecular computer, the principle of which will be based on the ability of some molecules to be in different states;
Software	Machine languages	Operating Systems, Algorithmic Languages	OS, interactive systems, computer graphics systems	Application packages, data and knowledge bases, browsers	3) a quantum computer, consisting of subatomic sized components and working on the principles of quantum mechanics. The fundamental possibility of creating such computers is confirmed by both theoretical work and the existing components of memory and logic circuits.
External devices	Punch and punched card input devices	APD, teletypes, MTD, MDD	Video Terminals, HDD	HDD, modems, scanners, laser printers	
Application areas	Settlement Tasks	Engineering, scientific, economic problems	ACS, CADs, scientific and technical tasks	Tasks of management, communication, creation of workstation, word and multimedia processing.	
examples	ENIAC, UNIVAC (USA); BESM - 1,2, M-1, M-20 (USSR)	IBM 701/709 (USA) BESM -4, M-220, Minsk, BESM -6 (USSR)	IBM 360/370, PDP -11/20, Cray-1 (USA); EC 1050, 1066, Elbrus 1,2 (USSR)	Cray T3 E, SGI (USA), PC, servers, workstations of various manufacturers	

1.2. Computer architecture classification systems

In 1966, M. Flynn proposed an extremely convenient approach to the classification of computing system architectures [2, 5, 12, 16, 17, etc.]. It was based on the concept of a stream, which is understood as a sequence of elements, commands, or data processed by a processor. The corresponding classification system is based on the consideration of the number of instruction streams and data streams and describes four architectural classes [12, 13, 16, 18]:

SISD = Single Instruction Single Data;

MISD = Multiple Instruction Single Data;

SIMD = Single Instruction Multiple Data;

MIMD = Multiple Instruction Multiple Data.

Below are the decrypted notations of these architectural classes, respectively:

SISD – *a single instruction stream / single data stream*. The **SISD** class includes serial computer systems that have one central processor capable of processing only one stream of sequentially executed instructions. Nowadays, almost all high-performance systems have more than one central processor, however, each of them executes unrelated instruction streams, which makes such systems complexes of SISD-systems operating in different data spaces. In order to increase the speed of commands processing and the speed of arithmetic operations, there can be used the conveyor processing. In case of vector systems, a vector data stream should be considered as a stream of single indivisible vectors. Examples of computers with SISD architecture are the majority of Compaq, Hewlett-Packard, and Sun Microsystems workstations [16, 17];

MISD – *a multiple instruction stream / single data stream*. Theoretically, in this type of machine, many instructions should be executed on a single data stream. So far, not a single real machine falling into this class has been created;

SIMD – *a single instruction stream / multiple data stream*. These systems usually have a large number of processors, from 1024 to 16384, which can execute the same instruction regarding different data in a hard configuration. A single instruction is executed in parallel on many data elements. Examples of **SIMD** machines are CPP DAP, Gamma II, and Quadrics Apemille [16, 17, 20]. Another subclass of **SIMD** systems is vector computers. Vector computers manipulate arrays of similar data similar to how scalar machines process individual elements of such arrays. This is done through the use of specially designed vector central processing units. When data is processed by means of vector modules, the results can be output on one, two or three cycles of the frequency generator (cycle of the frequency generator is the main parameter of the system). Working in vector mode, vector processors process data almost in parallel, which makes them several times faster than at working in scalar mode. Examples of systems of this type are, for example, Hitachi S3600 computers [16, 18];

MIMD – a multiple instruction stream / multiple data stream. These machines simultaneously execute multiple instruction streams on different data streams. Unlike multiprocessor **SISD** machines, commands and data are related because they represent different parts of the same task. For example, **MIMD** systems can simultaneously perform many sub-tasks in order to reduce the execution time of the main task. A wide variety of systems falling into this class makes Flynn's classification not entirely adequate. Indeed, both the NEC four-processor SX-5 and the thousand-processor Cray T3E fall into this class. This forces us to use a different classification approach, which otherwise describes the classes of computer systems. The main idea of this approach may consist, for example, in the following. We assume that a multiple command stream can be processed in two ways: either by a single conveyor processing device operating in time-sharing mode for individual threads, or each stream is processed by its own device. The first feature is used in **MIMD** computers. They are usually called conveyor or vector, the second - in parallel computers. Vector computers are based on the concept of conveyor, i.e. explicitly segmenting the arithmetic device into separate parts, each of which performs its own subtask for a pair of operands. A parallel computer is based on the idea of using several processors working together to solve a single problem, and processors can be either scalar or vector.

For parallel computer systems, there is a classification.

I. Vector-Conveyor Computers (PVP).

They have a MIMD architecture (a lot of instructions on a lot of data).

Key features:

- conveyor functional devices;
- a set of vector instructions in the command system;
- command engagement (used as a means of speeding up calculations).

II. Massively parallel computers with distributed memory.

There are combined several serial microprocessors, each with its own local memory, through some communication environment.

Such architecture has many advantages: if you need high performance, you can add more processors; if finances are limited or the required computing power is known in advance, then it is easy to choose the optimal configuration, etc.

Each processor has access only to its local memory, and if the program needs to know the value of a variable located in the memory of another processor, then the message transfer mechanism is activated. This approach allows to create computers that include thousands of processors.

But this architecture has two significant disadvantages:

- there is required a high-speed communication equipment that provides a messaging environment;
- during the creation of programs, it is necessary to take into account the topology of the system and to distribute data between processors in a special way in order to minimize the number of transfers and the amount of transferred data.

The last circumstance prevents the widespread adoption of such architectures.

III. Parallel computers with shared memory (SMP).

All RAM is shared between several identical processors. This removes the problems of the previous class, but adds new ones - the amount of processors with access to shared memory cannot be large.

The main advantage of such computers is the relative ease of programming. In a situation where all processors have equally fast access to shared memory, the question which processor will perform which calculations will not be so crucial, and a significant part of the computational algorithms developed for serial computers can be accelerated using parallelizing and vectorizing translators. SMP computers are the most common parallel computers today. However, the total amount of processors in SMP systems, as a rule, does not exceed 16, and their further increase does not give a gain due to conflicts at accessing memory.

IV. Cluster architecture.

Cluster architecture is a combination of the previous three types. A computing node is formed from several processors (traditional or vector-conveyor) and a common memory for them. If the received computing power is not enough, then several nodes are combined with high-speed channels.

Let consider a few more common criteria for classifying computers.

Classification by purpose [2, 5, 6, 8, 13, 15]:

- large electronic computers or Main Frame;
- mini-computers;
- microcomputers;
- personal computers.

Classification by level of specialization [2, 5, 6, 8, 13, 15, 18]:

- universal;
- specialized.

Classification by the size:

- desktop;
- portable (notebook);
- pocket (palmtop).

Classification by compatibility:

- hardware compatibility (IBM PC platform and Apple Macintosh);
- compatibility at the operating system level;
- software compatibility;
- data level compatibility.

Therefore, choosing the above listed characteristics as the classification criteria, first of all, computing power and dimensions, we obtain the following scheme, see Fig. 1.2.

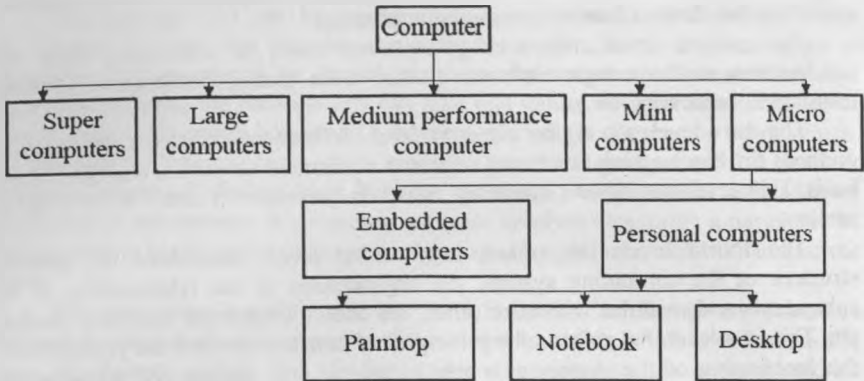


Fig. 1.2. Computer classification, based on their computing power and dimensions

It should be noted that any classification is conditional, since the development of computer science and technology is so rapid that, for example, today's microcomputers are not inferior in power to mini computers, to supercomputers of the recent past. In addition, the assignment of computers to a certain class is rather arbitrary through the fuzziness of the separation of groups, and as a result of the implementation of custom assembly of computers, where the nomenclature of nodes and specific models are adapted to customer requirements.

In the organization of a digital computing system, there can be distinguished nine levels of the hierarchy, shown on Fig. 1.3.

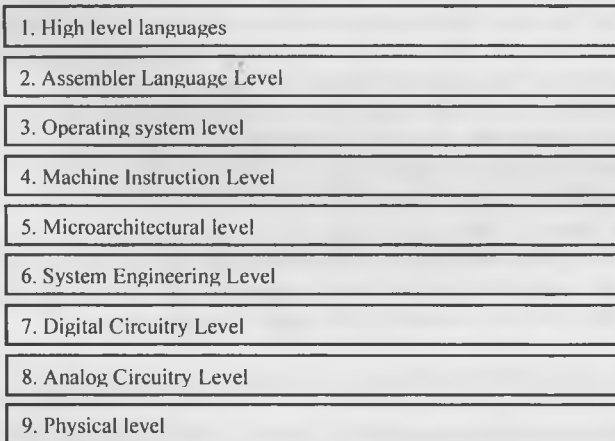


Fig. 1.3. Levels of hierarchy in the organization of a digital computing system

The first level, *physical*, combines the physical laws, phenomena and effects that underlie the creation and functioning of the hardware component of a

computing system. This is the level of integrated and functional microelectronics, providing the element base of computing equipment.

The second level, *the analog circuitry level*, is associated with the construction of basic logic elements (gates) from analog components (diodes, transistors, capacitors, etc.).

The third level, *the digital circuitry level*, defines the principles, models and methods for constructing functional units and equipment devices in a given logical basis. This level has its own hierarchy, which is presented in detail in the chapter section.

The fourth level, *the system engineering level*, determines the general structure of the computing system, the organization of the relationships of its subsystems and modules with each other, the choice of optimal operating modes, etc. This is a level that reflects the principles of construction and the principles of the functioning of the system as a whole, taking into account the influence of external factors, technical, economic and other indicators. At this level, to a first approximation, the functions are distributed between the hardware and software components of the architecture of the computing system.

The fifth level, *microarchitectural*, is associated with the organization of computer hardware management in the language of microcommands. This is the level of interpretation of machine instructions at which effective instruction execution technologies are implemented (hardware or firmware) using instruction prefetching, conveyor method, parallelization, caching, dynamic branch prediction, renaming registers and other techniques that contribute to increasing the efficiency of the computing process.

The sixth level, *the machine instructions level*, represents a set of commands (instructions) executed by hardware or an interpreter firmware. This is a connecting link between hardware and software, therefore, its organization should be rational both from the point of view of hardware developers and from the point of view of creators of translator programs from high-level languages.

The seventh level, *the operating system level*, differs from the previous one by the presence of additional commands, its memory organization, multiprogram mode and other extensions implemented by a special interpreter created on the basis of the sixth and possibly fifth levels and which is called *the operating system*.

The last three of the considered levels are initially planned as a tool environment for system software creation (translators, operating systems shells and other extensions supporting higher level languages). In contrast, the means of the eighth and ninth levels are oriented on the applied programmers.

1.3. The concept of information. Information measurement

Information is always presented as a message. The basic unit of a message is a symbol. Symbols are grouped together - words.

A message in the form of words or individual characters is always transmitted in the material-energy form (electrical signal, light, etc.).

There is distinguished continuous and discrete information.

The function $x(t)$ can be represented in a continuous and discrete form. In continuous form, this function can take any real values in a given range of changes of the argument t , i.e. the set of values of a continuous function is infinite. In a discrete form, the function $x(t)$ can take real values only for certain values of the argument. No matter how small a discrete interval t is chosen, the set of values of the discrete function for a given range of argument changes will be finite. A structural approach distinguishes between geometric, combinatorial and additive measures of information. A geometric measure involves measuring a parameter of the geometric model of an information message (length, area, volume) in discrete units.

The maximum possible amount of information in the given structures determines the information capacity of the model (system), which is defined as the sum of discrete values for all dimensions (coordinates). In a combinatorial measure, the amount of information is defined as the number of combinations (elements). The possible amount of information coincides with the number of possible combinations, permutations and placements of elements. Additive measure (Hartley measure). In accordance with this measure, the amount of information is measured in binary units - bits. There are introduced the concepts of the depth " q " of the number and the length " n " of the number. *The depth of the number "q"* - the number of characters (elements) adopted to represent the information (this is the basis of the number system). At any moment of time, one symbol is realized. *The length of the number "n"* - the number of positions necessary and sufficient to represent numbers of a given value. At a given " q " and " n ", the number of all possible displayed states is $N=q^n$. The value of N is not convenient for assessing information capacity. We introduce a logarithmic measure that allows to calculate the amount of information - bit [11]:

$$I(q) = \log N = n \log q.$$

Therefore, 1 bit of information corresponds to one elementary event, which may or may not occur. This allows to operate with a measure as with a number. The amount of information is equivalent to the amount of binary characters "0" or "1".

If there are several sources of information, the total amount of information is [9, 11]:

$$I(q, q, \dots, q) = I(q) + I(q) + \dots + I(q).$$

The transmission of information is a random process, because the content of the transmitted message is not known in advance. Interferences affecting the process of transmitting information are also of a random nature.

Therefore, the study of the laws of transmission and transformation of information is carried out by methods of probability theory, and information theory is sometimes called the statistical theory of message transmission.

At the same time, the theory of information is often and narrowly regarded as a theory of a measure of the amount of information and coding. The introduction of a numerical measure of the amount of information allows:

- to compare different messages according to their content;
- to evaluate the speed of information transfer in various systems;
- to compare these systems for effectiveness;
- to determine the maximum amount of information that can be transmitted in these specific conditions.

The need for information transfer arises only when the system may have a number of random states. It is assumed that the probabilities of the system in these states are known a priori.

About the system state we learn by message. A message about an event whose occurrence is known in advance does not contain information.

From two messages, one of which contains the results of two possible outcomes (throwing a coin), and the other results of six possible outcomes (throwing a dice), the second contains more information.

In the first case, the uncertainty of the outcome was relatively small, therefore, even before receiving the message that, for example, there is an "eagle" with a probability of $P=1/2$, it is possible to predict the occurrence of the outcome.

In the second case, $P=1/6$, the uncertainty was much greater, and, therefore, the receipt message to a greater extent contains an element of surprise, of novelty.

Therefore, the amount of information contained in a message can be quantified by the probability of occurrence of this message.

This evaluation criterion allows to establish an objective numerical measure of the amount of information contained in any possible messages, regardless of their specific meaning.

In accordance with the above mentioned, the requirements that must be met by the measure of the amount of information can be formulated as follows:

1. The amount of information contained in this message should not be determined by its specific meaning, but only by the degree of uncertainty removed upon receipt of this message.

2. The amount of information should be zero if the event of interest has only one outcome.

3. The principle of additivity should be observed, i.e. the amount of information contained in this message should be proportional to its length.

In case of equally probable events (messages), the logarithm of the inverse probability of each of them is taken as a numerical measure of the amount of information. If we use the binary logarithm and get $N=2$, then $I=\log_2 2=1$.

As a unit of information quantity, it is customary to consider such a quantity of information that removes uncertainty at choosing one of two equally probable outcomes. This unit is called a binary unit or bit. In the general case of receiving unequal messages, the uncertainty of the appearance of a particular (i -th) message is characterized by its probability P_i .

Information processed by the computer is represented by the corresponding symbols. Information is diverse - from numbers to music.

The semantic measures of information are considered:

- content;
- logical amount;
- expediency.

The content of the event i through the measure function $m(i)$ is the content of its negation.

The logical functions of truth $m(i)$ and false $m(i)=1-m(i)$ have formal similarities with the probability functions of the events $P(i)$ and $q(i)=1-P(i)$. The logical amount of information I is calculated by the formula:

$$I = \log(1/m(i)) = -\log m(i).$$

The difference between a statistical estimate and a logical one is that in the first case the probabilities of the implementation of certain events are taken into account, that brings us closer to information meaning assessment. The measure of the expediency of information is defined as the change in the probability of achieving a goal when additional information is obtained

$$I_{\text{усп}} = \log P_n - \log P_k = \log P_n / P_k.$$

where P_n, P_k - initial and final probabilities of achieving the goal.

Electronic computers, i.e. computers, are designed to process digital information.

A computing machine - a physical system designed to automate the process of algorithmic information processing. Thus, the concept of "**computing machine**" (CM) is most closely related to the concepts of "**information**" and "**algorithmic processing**" [2, 5, 6].

The object of transmission and conversion in computing systems (machines) is information. In this sense, a CM (system) can be called informational.

Information is presented in the form of drawings, text, sound and light signals, energy pulses, etc. and transmitted by signals of any physical nature along the lines of communication of the source with the receiver.

Algorithmic processing - the information processing in accordance with a previously developed algorithm.

An algorithm - a rule (rules) formulated in a certain language that defines actions whose sequential execution leads from the initial data to some result.

Therefore, the specificity of information processes consists not only in the transmission of information messages through a given physical environment, but also in the transformation, processing and storage of information.

Information defines many processes in a CM. In the most general form, the process of solving a problem on a CM goes through the following steps:

- 1 - input of information or initial data set;
- 2 - processing or conversion of the input information;
- 3 - determination of results and processed information output.

Modern computers can solve a wide variety of problems. In order to do this, you just need to use the program to "train" the computer the algorithm for solving a particular problem and to set initial data into it. The program is written in an algorithmic language (for example, Pascal, C ++ or BASIC), which is quite close to the natural language (especially English).

However, the computer does not understand not only natural language, but also algorithmic. In order to decrypt the text of a program written in Pascal or C ++, the machine must have a special program - a compiler (translator), which translates the text of the original program from C ++ or Pascal to the computer language. Therefore, a computing machine - a technical device in which information about the initial data of the problem, the rules for its solution (algorithm) and the results of the calculations must be set in the form of a change in any physical quantities:

- the magnetization of the material (such as, for example, to play a melody using a tape recorder or to save data on certification of students in the curriculum disciplines on a magnetic disk);
- screen illumination (display), etc.

Questions for self-control

1. What are the generations of computers?
2. By what criteria are computers classified?
3. What are the levels of hierarchy in the organization of a digital computing system?
4. What is the structural diagram of the computer?
5. Define the concept of information. What is an algorithm?

Test questions

1. What is the peculiarity of von Neumann architecture?
 - A) Separate areas of physical memory for storing data and commands (instructions).
 - B) A single area of physical memory for storing data and commands (instructions).
 - C) The CISC command system is used.
 - D) All answers are incorrect.

2. In what year was the first serial microprocessor released?
 - A) 1971
 - B) 1968
 - C) 1945
 - D) 1956

3. What modifications in the development of the IA-32 architecture appeared in the i486 microprocessor?
- A) 32-bit external data bus
 - B) SIMD floating point number processing
 - C) SIMD fixed point number processing
 - D) embedded floating point processor.
4. What was the very first Intel processor called?
- A) intel 1001
 - B) intel 2201
 - C) intel 8008
 - D) intel 4004
5. A simplified version of PII for cheap computers is:
- A) Pentium P55
 - B) AMD
 - C) Cyrix
 - D) Celeron
6. Indicate the missing class in the Flynn classification of parallel computing systems SISD, MIMD, MISD,
- A) NUMA
 - B) SIMD
 - C) AMP
 - D) SMP
7. What measure of information was proposed by Hartley?
- A) exponential
 - B) lognormal
 - C) polynomial
 - D) logarithmic
8. What is the name of the binary unit of information?
- A) byte
 - B) nat
 - C) child
 - D) bit
9. What is the name of a quantity that measures the amount of information?
- A) voltage
 - B) entropy
 - C) power
 - D) probability
10. What is the relationship between signs and words?

- A) syntax
- B) semantics
- C) sigmatics
- D) pragmatics

11. What is not included in the classification of parallel computer systems?

- A) Vector conveyor computers
- B) Massively Parallel Computers
- C) Parallel computers with shared memory
- D) Cluster architecture

12. In what century did the first devices capable of performing arithmetic operations appear?

- A) in the 16th century
- B) in the 17th century
- C) in the 18th century
- D) in the 19th century.

13. What was the name of the first mechanical device for performing four arithmetic operations?

- A) serobyán
- B) suan pan
- C) seven-point accounts
- D) an arithmometer

14. A mechanical device that allows to add numbers, invented by:

- A) P. Norton
- B) B. Pascal
- C) G. Leibniz
- D) D. Neumann

15. The idea of a mechanical machine with the idea of software control was combined by:

- A) C. Babbage (first half of the 19th century)
- B) J. Atanasov (30s of the XX century)
- C) C. Berry (XX century)
- D) S.A. Lebedev (1951)

16. Abacus is:

- A) musical instrument
- B) accounts
- C) a device for working on a given program
- D) the first mechanical machine

17. In what century did mechanical arithmometers appear?

- A) in the 14th century
- B) in the 16th century
- C) in the 17th century
- D) in the 19th century.

18. The first programmer in the world is:

- A) G. Leibniz
- B) A. Lovelace
- C) B. Pascal
- D) S. Lebedev

19. The first inventor of punch cards was:

- A) D. Nepper
- B) I. Shikard
- C) J. Jacquard
- D) B. Pascal

20. In what century did a radical change in the development of computer technology occur?

- A) in the 19th century.
- B) in the 20th century.
- C) in the 18th century
- D) in the 17th century

CHAPTER 2. FUNCTIONAL NODES OF A COMPUTER

The fig. 2.1 shows the functional structure of the computer.

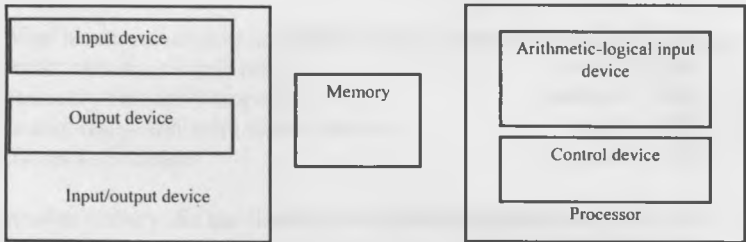


Fig. 2.1. Basic computer functional devices

Therefore, a computer consists of five main, functionally independent parts:

- input device;
- output device;
- memory;
- arithmetic-logical device;
- control device.

The input device receives encoded information from digital operators via digital communication lines, for example, via a keyboard, mouse or other network computers. The information received is either stored in the computer's memory for later use, or is immediately used by arithmetic and logic circuits to perform the necessary operations. The sequence of processing steps is determined by the program stored in memory. The results are sent back to the outside world through an output device. All these actions are coordinated by the control unit. On fig. 2.1. the connections between functional devices are not shown. This is explained by the fact that such relations can be implemented in different ways; we will dwell on this a little later. Arithmetic and logic circuits in conjunction with the main control circuits are called the processor, and all together taken equipment for input and output is often called an *input-output unit*.

Now let turn to the information processed by the computer. It is convenient to divide it into two main categories: commands and data, see chapter 1. Commands, or machine instructions, are explicitly given instructions that:

- manage the transfer of information inside the computer, as well as between the computer and its input-output devices;
- determine the arithmetic and logical operations to be performed.

The list of commands that perform a task is called a program. Typically, programs are stored in memory. The processor in turn extracts program instructions from memory and implements the operations they define. The computer is fully controlled by the stored program, except for the possibility of external intervention by the operator and input/output devices connected to the machine.

Data is numbers and encoded characters used as operands of commands. However, the term “data” is often used to mean any digital information. According to this definition, a program itself (i.e. a list of commands) can also be considered as data if it is being processed by another program. An example of processing one program by another is compiling a source program written in a high-level language into a list of machine instructions that make up a machine language program called an object program. The source program is input to the compiler, which translates it into a machine language program.

Information intended for computer processing must be encoded in order to have a format suitable for the computer. Most modern hardware is based on digital circuits, which have only two stable states, ON and OFF. As a result of encoding, any number, character or command is converted into a line of binary digits called bits, each of which has one of two possible values: 0 or 1.

2.1. Arithmetic and logical device

Let consider the functional units of the processor using the model 8086 as an example, see Fig. 2.2.

The Central Processing Unit (CPU) 8086 has four 16-bit general purpose registers *AX, BX, CX, DX*, four pointer registers *SI, DI, BP and SP*, four *segment registers CS, DS, ES, SS*, one 16-bit register of flags *FLAGS* and program pointer *IP*, see fig. 2.2. The fig. 2.3 shows a block diagram of a processor 80286.

The structure of ALD is presented on Fig. 2.4. The initial data (operands) by the commands of the control device (CD) are read from RAM into the registers of the first and second operands (communication line 1 - (CL1) on Fig. 2.4 *there are shown only numbers*).

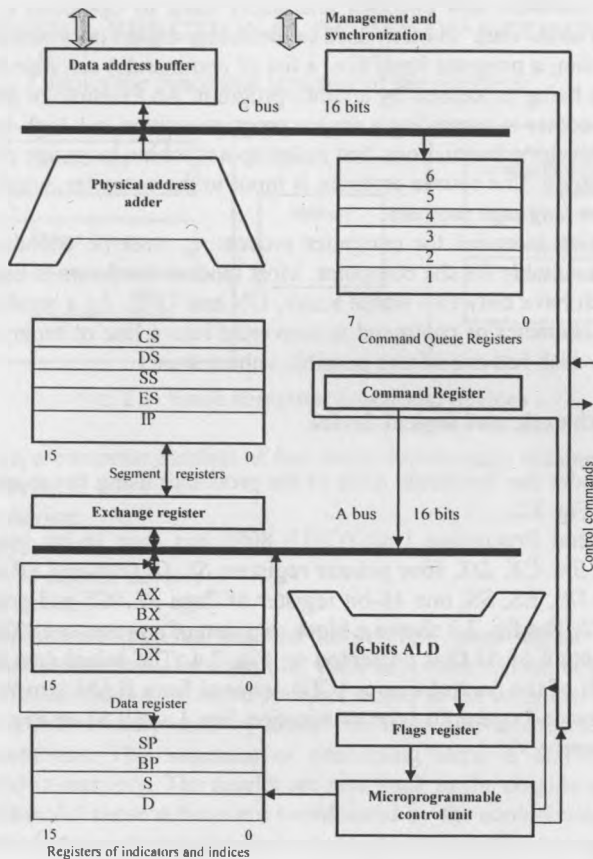


Fig. 2.2. The structure of the processor i8086

A command is sent from the CD to the ALD to perform one or another operation (CL 2), which is transmitted to them in the operational part (CL 3).

In accordance with this command, the operating part performs the desired action with data that is selected from the registers of the first and second operands (CL 6). The result is entered in the result register (CL 4), from where - in RAM (CL 5).

The structure of ALD registers, where the initial and resulting data are placed, as well as the size of the registers (the number of binary bits) form the concept of a bit grid.

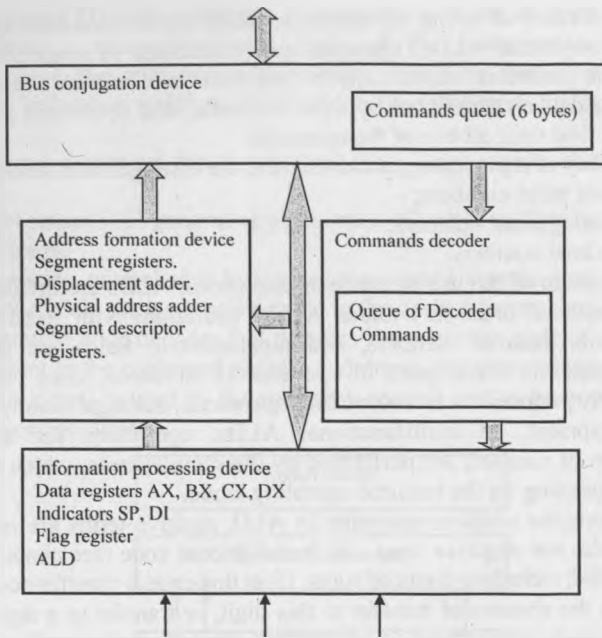


Fig. 2.3. The structure of the MP 80286

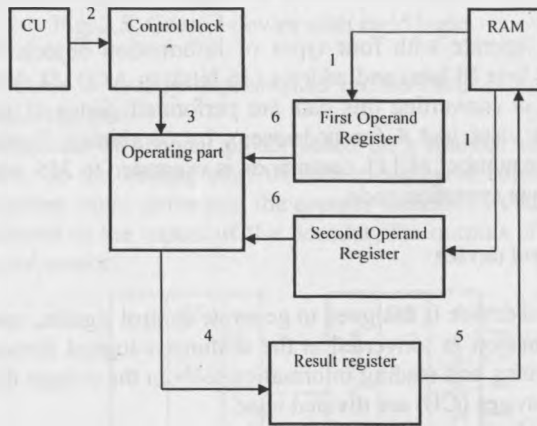


Fig. 2.4. Arithmetic-logical device diagram of the processor

By the method of acting on operands, ALDs are divided into sequential and parallel. In sequential ALDs, operands are represented in sequential code, and operations are performed sequentially in time over their individual bits. In parallel ALDs, operands are represented by a parallel code, and operations are performed in parallel in time over all bits of the operands.

By the way of representing numbers there are distinguished following ALDs:

- for fixed point numbers;
- for floating point numbers;
- for decimal numbers.

By the nature of the use of elements and nodes, ALDs are divided into block and multifunctional ones. In a block ALDs, operations with fixed and floating point numbers, decimal numbers, and alphanumeric fields are performed in separate blocks, while the speed of operation is increased, since the blocks can simultaneously perform the corresponding operations, but significantly increase the cost of equipment. In multifunctional ALDs, operations for all forms of representation of numbers are performed by the same circuits, which are switched as needed depending on the required operating mode.

Performing the addition operation in ALD, positive terms are represented in the direct code, and negative ones - in the additional code (see chapter 3). Binary codes are added, including digits of signs. If, in this case, a transfer occurs from the sign digits in the absence of transfer to this digit, or transfer to a sign digit in the absence of transfer from the digit of the sign, then there is overflow of the digit grid, respectively, for negative and positive sums. If there are no transfers from the sign digit to the sign digit of the sum, or there are both transfers, then there is no overflow and at zero in the sign digit the sum is positive and presented in the direct code, and at 1 in the sign digit the sum is negative and presented in the additional code.

ALD can operate with four types of information objects: Boolean (1 bit), digital (4 bits), byte (8 bits) and address (16 bits). In ALD, 51 different operations of transferring or converting this data are performed. Since 11 addressing modes are used (7 for data and 4 for addresses), by combining "operation/addressing mode" the basic number of 111 commands is expanded to 255 out of 256 possible with a single-byte operation code.

2.2. Control device

The control device is designed to generate control signals, under the influence of which information is converted in the arithmetic-logical device, as well as the operation of writing and reading information to/from the storage device.

Control Devices (CD) are divided into:

- CD with rigid or circuit logic;
- CD with programmable logic (firmware CD).

In the control devices of the first type, for each command specified by the operation code, there are constructed a set of combinational circuits that generate the necessary control signals in the necessary cycles.

In firmware CD, each command is assigned by a set of words stored in a special memory - microcommands. Each of the microcommands contains information about the microoperations which should be performed in a given cycle, and an indication which word should be selected from the memory in the next cycle.

The control device of the circuit type (Fig. 2.5) consists of the following nodes:

- a signal sensor (SS) generating a sequence of pulses uniformly distributed in time over its buses;
- an operation control unit that generates control signals, that is, switching signals from the SS in the appropriate cycle to the desired control bus;
- an operation code decoder that decodes the operation code of the command currently present in the command register and drives one bus corresponding to this operation; this signal is used by the operation control unit to generate the desired sequence of control signals.

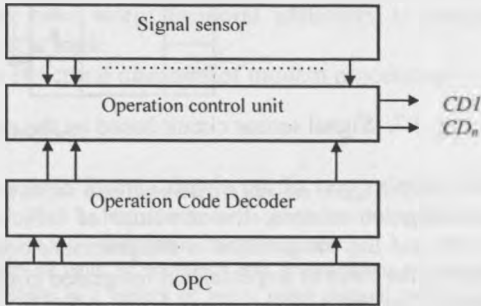


Fig. 2.5. Control device with rigid logic

The signal sensor is usually implemented on the basis of a counter with a decoder or on a shift register.

The implementation of a signal sensor based on a counter with a decoder is shown on Fig. 2.6. On the trailing edge of each clock cycle entering the control device from the system pulse generator, the counter increases its state; the counter outputs are connected to the inputs of the decoder, the outputs of which are the outputs of the signal sensor.

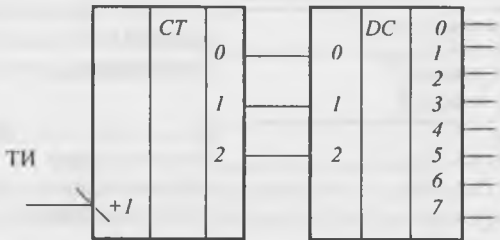


Fig. 2.6. Signal sensor circuit based on counter with decoder

The implementation of the signal sensor on the shift register requires only its "looping", that is, the connection of the output of the last digit to the input through which the information is recorded during the shift, and the initial installation (Fig. 2.7). In the initial state, the register contains 1 only in the zero digit. The inputs of the parallel register loading for its initial installation and the control input of the register corresponding to this operation are not shown in the diagram.

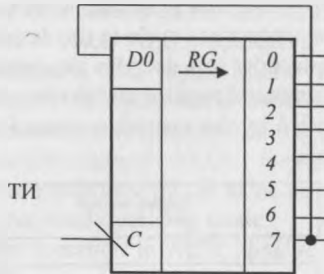


Fig. 2.7. Signal sensor circuit based on the shift register

The most complex part of the circuit control device is the operation control unit. It is an irregular scheme, the structure of which is determined by the command system and the composition of the processor equipment. Such a CD can be implemented in the form of a specialized integrated circuit.

The firmware control device is shown on Fig. 2.8

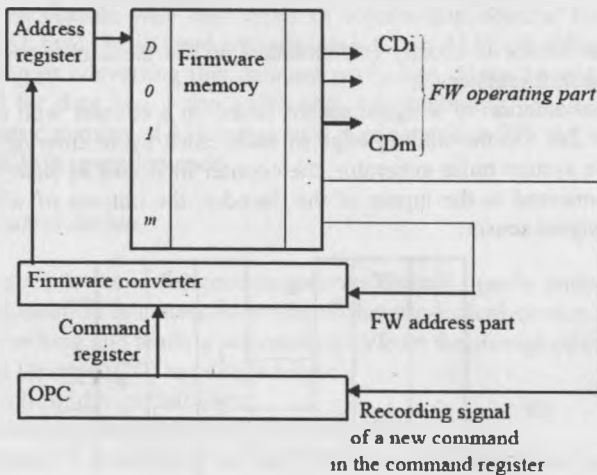


Fig. 2.8. Functional diagram of the firmware control device

The microcommand address converter converts the operation code of the command currently present in the command register to the initial address of the firmware that implements this operation, and also determines the address of the next microcommand of the executed firmware by the value of the address part of the current microcommand. *CD*, - control signals generated by the control device.

An analysis of the structure and principles of operation of the circuit and microprogram control devices shows that the control units of the first type have a complex irregular structure, which requires special development for each command system and should be almost completely reworked with any modifications to the command system. At the same time, it has a rather high speed determined by the speed of the used elemental basis.

The control device, implemented by the firmware principle, can be easily configured for possible changes in the operating part of the computer. At the same time, the setup largely comes down to replacing the firmware memory. However, control units of this type have worse temporal indicators in comparison with control devices based on rigid logic.

We will dwell on the structural diagrams of modern processors in chapter 6.

2.3. Input-output devices

The development and the use of data input-output devices is a practical area that is closely related to computer technology. With its historical roots, it goes even deeper than computers, and the best minds of the computer era were engaged in its development.

Thanks to the many years of work of engineers and programmers, it became possible to enter information into the machine in a variety of ways: using manual switches, typing on an alphanumeric or numeric keypad, drawing with a pen on an electronic tablet, speaking into a microphone or touching the display screen with your fingertip. Some output devices, such as displays, are best suited for presenting temporary, i.e., rapidly changing data. They are preferred, for displaying text while typing or editing. When the document is ready, printing devices are already used. In recent years, peripheral input-output devices have become very diverse, often they are even more expensive and take up more space than the computer itself. I/O devices are discussed in more detail in Chapter 16.

Questions for self-control

1. What are the functional elements of a computer?
2. What is the purpose of ALD?
3. Draw the structure of MP 80286.
4. Functions of the control device?

Test questions

1. What function does the arithmetic logical device perform?
A) Processes information in accordance with the instruction received by the processor
B) Organizes the interaction of all processor nodes

- C) Synchronizes the processor with external signals
 - D) Does not read commands from memory
2. What function does the control logic perform?
- A) Organizes the interaction of all processor nodes
 - B) Reads commands from memory
 - C) Handles an interrupt request to the processor
 - D) Does not read commands from memory
3. The structural diagram of a computer in the general case includes:
- A) processor, RAM, ROM, input devices, output devices
 - B) ALD, control devices, printer, display
 - C) microprocessor, RAM, keyboard, display
 - D) system unit, display, RAM
4. The computer logic does not include:
- A) Arithmetic - logical device
 - B) Control device
 - C) Addressable memory
 - D) External devices
5. Which of the following devices relates to the central devices of the computer:
- A) monitor
 - B) RAM
 - C) keyboard
 - D) drive
6. Which device does not apply to input devices:
- A) mouse
 - B) keyboard
 - C) scanner
 - D) printer
7. Which device does not apply to external storage media:
- A) floppy disk
 - B) cd
 - C) hard drive ("Winchester")
 - D) RAM
8. Device controllers are needed
- A) for storing the running program
 - B) to organize the operation of peripheral devices at the program level
 - C) to organize the interaction of peripheral devices with each other
 - D) for connecting devices to the trunk at the physical level

9. From which device (unit) that is a part of the computer does the processor select the next command for execution?

- A) external storage devices
- B) read only memory
- C) RAM
- D) keyboard

10. Classical architecture is called

- A) John von Neumann architecture
- B) Charles Babbage's architecture
- C) the architecture of Blaise Pascal
- D) Bill Gates architecture

11. The functions of the ALD is to perform

- A) data movement
- B) graphic computing
- C) arithmetic operations
- D) decoding processor instructions

12. In order to transfer data between the functional nodes of the computer there is (are) used...

- A) measures
- B) buses
- C) adder
- D) cache

13. The control device (CD) is a part of ...

- A) microprocessor
- B) system bus
- C) main computer memory
- D) a cycle generator

14. Arithmetic logical device (ALD) is an integral part of ...

- A) system bus
- B) main computer memory
- C) microprocessor
- D) a cycle generator

15. What types of information objects ALD can not operate?

- A) boolean
- B) digital
- C) byte
- D) analog

16. How many types of information objects ALD can not operate?

A) 2 B) 4 C) 6 D) 3

17. How many different data transfer or data conversion operations are performed in ALD?

A) 48 B) 69 C) 51 D) 37

18. How many addressing modes are used in ALD?

A) 11 B) 6 C) 15 D) 7

CHAPTER 3. ARITHMETIC FOUNDATIONS OF A COMPUTER

3.1. Number systems and code concept

As it was already noted, during the process of information processing, digital computers operate with numbers that are represented in some number system.

The number system - a set of techniques and rules for recording numbers with digital signs. Recording a number in some number system is often called *a number code*.

Elements (signs) of the alphabet, which are used to write numbers in a certain number system, are usually called *numbers*. Each digit of a given number is uniquely associated with its quantitative (numerical) equivalent.

There are distinguished positional and non-positional number systems.

A non-positional number system is a system for which the meaning of a symbol, i.e. figure, does not depend on its position in the number. Such systems include, in particular, the Roman system (though with some reservations). This is probably the most famous system, after the Arabian. We use it quite often in everyday life. These are chapter numbers in books, century indications, numbers on watch dials, etc. This numbering arose in ancient Rome. Here, for example, the symbol *V* always means five, regardless of where it appears in the number record. It was used for an additive alphabetical number system. The digits of the number were recorded, starting with large values and ending with smaller ones, from left to right. If a digit with a lower value was written before a digit with a higher value, then it was subtracted, for example, *IV*. Or *CCXXXVII* = 237, but *XXXIX* = 39.

There are other modern non-positional systems. The disadvantage of non-positional number systems is the unlimited number of different digits needed to represent any number.

A positional number system - a system in which the value of each digit depends on its numerical equivalent and on its place (position) in the number, i.e. one and the same sign (digit) can take on different meanings.

The most famous positional number system is the decimal number system. For example, in the decimal number 254, the first digit on the right means 4 units, the neighboring one - 5 dozen, and the left - 2 hundreds.

Due to the fact that digital automatons mainly use positional number systems, we will consider only such systems.

The binary number system is the youngest of the existing ones. This system has a number of qualities that make it very beneficial for use in digital machines (computers). The official birth of binary arithmetics is associated with the name of G.V. Leibniz, who published an article in 1703 in which he examined the rules for performing arithmetic operations on binary numbers. For a binary system, only two signs are needed - 0, 1. In a binary system, arithmetic operations are especially simple.

Any positional number system is characterized by a base.

The base or basis q of a natural positional number system is the amount of signs or symbols used to represent a number in a given system.

Therefore, countless positional systems are possible, as any number can be taken as the basis, forming a new number system.

When we record a certain number in a positional number system, we place the corresponding figure of the number on the individual necessary positions, which are usually called the digits of a number in this positional number system. The amount of digits in a number record is called the *bit depth* of the number and matches its *length*.

In a positional number system the equality is true:

$$A_q = a_{n-1}q^{n-1} + a_{n-2}q^{n-2} + \dots + a_kq^k + a_0q^0 + a_{-1}q^{-1} + \dots + a_{-m}q^{-m}, \quad (3.1)$$

where A_q - an arbitrary number written in the number system with the base q ; a_i - row coefficients, i.e. number system digits; n , m - the number of integer and fractional digits, respectively.

The maximum integer that can be represented in m digits, $A_{max} = q^n - 1$. The smallest significant non-0 number that can be written in m digits of the fractional part $A_{min} = q^{-m}$.

For example, the number $A=123,45$ denotes an abbreviated expression of $A=1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}$.

It is easy to verify that in any number system using Arabic numerals, the base of the system is represented as the number 10.

For example, according to the (3.1)

$$1964,52_{10} = 1 \cdot 10^3 + 9 \cdot 10^2 + 6 \cdot 10^1 + 4 \cdot 10^0 + 5 \cdot 10^{-1} + 2 \cdot 10^{-2},$$

$$124=537_8 = 1 \cdot 8^2 + 2 \cdot 8^1 + 4 \cdot 8^0 + 5 \cdot 8^{-1} + 3 \cdot 8^{-2} + 7 \cdot 8^{-3},$$

$$1001,1101_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4}.$$

An index assigned to a number indicates the number system in which the number is represented.

The base of the number system shows how many different values within the i -th digit each digit can take the a_i figure of the number A . The bit numbers in the positional number system are counted in the integer part to the left of the decimal point, and in the fractional part, to the right of the decimal point. Moreover, the numbering of digits begins with 0. The value of the base of the positional number system determines its name: for the decimal system it will be 10, for the octal - 8, for binary - 2, etc. As already noted, the term "**number code**" is usually used instead of the system name. For example, the concept of binary code means the number represented in the binary system, the concept of decimal code in the decimal system, etc.

To record a number in the decimal system, 10 different digits are used from 0 to 9.

$$A_{10} = (0,1,2,3,4,5,6,7,8,9).$$

In hexadecimal – 16

$A_{16} = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)$,

where $A=10, B=11, C=12, D=13, E=14, F=15$.

In octal – 8

$A_8 = (0, 1, 2, 3, 4, 5, 6, 7)$.

In binary – 2

$A_2 = (0, 1)$.

According to (1.1), each digit in a binary number system to the left of the decimal point is represented by a two in the corresponding positive degree, and to the right of the decimal point - two in the negative degree. For example

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
16	8	4	2	1,0	0,5	0,25	0,125	0,0625

Examples of representing numbers in various number systems:

$$\begin{aligned}10_{10} &= 1010_2 = 12_8 = A_{16} \\16_{10} &= 10000_2 = 20_8 = 10_{16} \\255_{10} &= 11111111_2 = 377_8 = FF_{16}.\end{aligned}$$

In order to process information in a computer, a binary number system is usually used. This is due, in particular, to the fact that for the placement of numbers (operands) in computers, registers and memory cells are used, consisting of triggers or elements with a trigger characteristic, which, as you know, have two stable states. One of these states is assigned 1, and the other - 0. The number of triggers, i.e. binary bits in a register or memory cell determines the word length characteristic for a given computer, and the combination of these binary bits is called a **bit grid**. The digit number of such a grid reserved for the image of an integer in the binary number system coincides with the corresponding exponent of two.

Therefore, the *length of a number* is the number of positions (or digits) in the record number. Different numeral systems are characterized by different lengths of the bit grid needed to record the same number. For example, the decimal number 999 (999_{10}), respectively, in the hexadecimal, octal and binary number system will have the following form $3E7_{16}$, 1747_8 , 1111100111_2 , i.e. the smaller the base of the number system, the greater the length of the number.

In any digital machines, the length of the bit grid of the selected number system is fixed, which fundamentally limits the accuracy and range of numbers.

The range of numbers representation in a given number system is the interval of the numerical axis, concluded between the maximum and minimum numbers, the value of which depends on the length of the bit grid.

During the exchange of data between a computer and external devices, it becomes necessary to exchange symbolic and alphabetic characters. These characters in the computer are also associated with some code in the binary system. For the representation of numbers and letters in the binary system, ASCII code is

currently the most common. In order to represent any character in this code, 8 binary digits are allocated, i.e. one byte. Examples of ASCII code are given in table 3.1.

Table 3.1

Examples of ASCII code

Symbols	Decimal code	Binary code	Octal code	Hexadecimal code
0	48	0110000	060	30
1	49	0110001	61	31
2	50	0110010	62	32
A	65	1000001	101	41
B	66	1000010	102	42
F	70	1000110	106	46
:	58	0111010	72	3F
(40	0101000	50	28

Choosing a number system for a computer, it is necessary to take into account that, firstly, the base of the number system determines the amount of stable states that a functional element must have, selected to display the digits of a number; secondly, the length of the number substantially depends on the base of the number system; thirdly, the number system should provide simple algorithms for performing arithmetic and logical operations.

The widespread use of the binary number system is due to the following reasons:

- the variety and simplicity of the technical implementation of elements with two stable states;
- good distinguishability of the two states, reducing the possibility of distortion of signals and failures;
- simplicity of arithmetic operations;
- cost-effectiveness of the equipment.

The profitability of equipment using a binary number system is illustrated by the following example. We are designing a device in which one element is required to display each digit of a number. Then, to represent any digit in the number system with the base q , q different elements are required. To display in the same number system any number containing n_q digits, it is necessary $L_q = q \cdot n_q$ elements.

For example, to represent any number from 000 to 999_{10} in the decimal system, $n_{10} = 3$ digits are required, each of which contains 10 digits; with $L_{10} = 30$ elements. To display the same numbers in the binary system, it requires $n_2 = 10$, each of which contains two digits, i.e. $L_2 = 20$ elements.

A similar comparison of decimal and binary number systems shows that decimal is 1.5 times less economical than binary.

The most convenient conditions for the implementation of binary digits, as physical processes having two stable states are much more than processes with the number of clearly distinguishable states more than two. In addition, in processes

with two stable states, the difference between these states is qualitative, not quantitative, which ensures reliable implementation of binary digits.

Therefore, the simplicity of arithmetic and logical operations, the minimum of equipment used to represent numbers, and the most convenient conditions for the implementation of only two stable states determined the use of binary number systems in almost all existing and designed digital computers.

In the binary system, arithmetic operations are especially simple. In the binary system, there is no "addition table" that would need to be remembered, since the transfer to the high order starts with $1+1=10$. Adding large numbers, you only need to add them in columns or by digits, as in the decimal system, you should remember only that as soon as the sum in the column reaches number 2, it will be transferred to the next column (to the left) as a unit of the highest order. Subtraction is performed in the same way as in the decimal system, without thinking about that, if necessary, you will need to occupy 2, and not 10, from the left column.

However, for this ease you have to pay a large number of signs multiplying even small numbers.

Table 3.2 lists the rules for adding, subtracting and multiplying binary numbers.

Table 3.2

Rules for Adding, Subtracting and Multiplying Binary Numbers

Adding	Subtracting	Multiplying
$0 + 0 = 0$	$0 - 0 = 0$	$0 \times 0 = 0$
$1 + 0 = 1$	$1 - 0 = 1$	$1 \times 0 = 0$
$0 + 1 = 1$	$1 - 1 = 0$	$0 \times 1 = 0$
$1 + 1 = 10$	$10 - 1 = 1$	$1 \times 1 = 1$

In general case, the procedures for adding and subtracting two numbers in any positional number system begin with the least significant digits, see Chapter 3.

The sum code of each i -th digit c_i is obtained by adding $a_i + b_i + l$, where the unit corresponds to the transfer from the lowest $(i-1)$ -digit to the i -th digit if the sum code in the lower digit is greater than or equal to the base of the number system.

The difference code of each i -th digit is obtained by subtracting $a_i - b_i - l$, where the unit corresponds to the "loan", if it was, in the lower digits of the value equal to the base of the number system.

Therefore, the rules and methods of addition and subtraction in any positional number system in principle remain the same as in the decimal system.

Let consider the rules of arithmetics with numbers represented in decimal, binary, octal and hexadecimal codes using a simple example, see table. 3.3.

Table 3.3

Addition rules

Terms	Decimal code	Binary code	Octal code	Hexadecimal code
<i>Перенос</i>		1 111	11	1
1-e	166	10100110	246	A5
2-e	47	00101111	57	2F
<i>Result</i>	213	11010101	325	D5

Subtraction is also performed bitwise, starting with the least significant bit. Subtracting units in a given digit from zero, it is necessary to take a unit from a neighboring senior digit, which is equal to two units of this digit.

The summation of binary numbers in computers is carried out using binary adders, and the subtraction - by binary subtractors. But, as will be shown later, the subtraction can also be organized using the addition procedure, i.e. using binary adders, if the deductible is presented in an additional or reverse code and thereby eliminate the need for binary subtractors.

The multiplication of binary numbers is carried out by the formation of intermediate products and their subsequent summation.

In addition to *arithmetic* operations, computers also implement *logical* operations, see section 4.

In computers, another operation is performed on binary numbers - a *shift* of a number along the bit grid to the left or right. In the case of a shift to the left, the binary number is actually multiplied by 2, and when it shifted to the right - division by 2, where - the number of bits by which the binary number is shifted. For example: $000011_2 = 3_{10}$ we shift to the left by 2 digits, we get $001100_2 = 12_{10}$, i.e. $3 \times 4(2^2) = 12_{10}$, and now $001000_2 = 8_{10}$ we shift 2 bits to the right, we get $000010_2 = 2_{10}$, i.e. $8:4(2^2) = 2_{10}$.

In computers, a *cyclic shift* is often used, during which the bit grid allocated for the operand appears to be closed in a ring. Then, shifted to the left, the contents of the highest order fall into the least significant bit of the operand, and shifted to the right - the opposite.

As it was already noted, any information processing in a computer is usually carried out in a binary number system. At the same time, exchanging information between a computer and a user, there are used decimal, binary-decimal, octal or hexadecimal systems for better visualization of data representation. Each digit in octal and hexadecimal code is equivalent to three and four binary digits, respectively. Therefore, the representation of numbers in these number systems is more compact and visual.

3.2. Arithmetic computer basics

A form of representing numbers in digital machines - a set of rules that allow to establish a mutual correspondence between the record of a number and its quantitative equivalent.

A machine (automatic) image of a number - a representation of a number in the bit grid of a digital automaton. The symbol of the machine image of a number, for example, A , will be represented as $[A]$.

Due to the limited length of machine words, the set of numbers that can be represented in a machine is finite. Comparison of various forms of representation of numbers in computers is usually based on an assessment of the range and accuracy of the numbers representation.

In everyday practice, the most common is the form of representing numbers in the form of a sequence of numbers, separated by a comma into integer and fractional parts. Numbers represented in this form are called natural comma numbers or natural numbers. In a natural form, a number is written in a natural kind, for example, 2360 is an integer, 0.0065 is a correct fraction, 64.89760 is an incorrect fraction.

Presenting numbers in this form, an indication of the position of its comma in the bit grid allocated to represent the number in the machine is required for each number, which requires additional hardware costs of a sufficiently large volume. Therefore, two other forms of representation have its widespread in computers: fixed and floating point.

3.2.1. Fixed point number representation form

There is no need to indicate the position of the comma if the place of the comma in the bit grid of the machine is fixed in advance. This form of representation of numbers is called fixed point (dot) representation, see fig. 3.1.

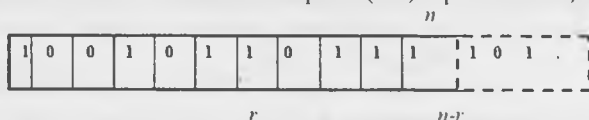


Fig. 3.1. Fixed point representation

In the form of a fixed point representation, all numbers are represented as a sequence of digits with a constant position for all numbers, a comma separating the integer part from the fractional. For example, in the decimal system, there are five digits in the integer part of the number and five digits in the fractional part. The numbers written in such a bit grid will have the following form: +00567,22100; +00000,00044; -23501,20450.

Let suppose that in a bit grid you need to place a binary number containing integer and fractional parts. If r cells of an n -bit grid are allocated to accommodate the integer part, then (without taking into account the sign) $n-r$ free cells will remain to accommodate the fractional part. If the number of bits in the fractional part of the placed number exceeds $n-r$, then some of the lower bits will be outside the bit grid. Therefore, any binary number less than $0.0 \dots 1$ ($n-r$ zeros) is perceived as zero and is called *machine zero*.

Since numbers are positive and negative, the format (bit grid) of the machine image is divided into the *sign part* and *the field of the number*. The field of the number contains the image of the number itself, which we will arbitrarily call the *mantissa of the number*. To encode the sign of the number, there is used the oldest bit of the bit grid reserved for the image of the binary number, and the remaining bits are allocated to the mantissa of the number. The position of the comma in the bit grid is strictly fixed, usually either to the right of the smallest bit of the mantissa, or to the left of the oldest one. In the first case, the number is represented as an integer, in the second - as a regular fraction. On computers in fixed point format there are represented integers.

The sign part contains information about the sign of the number. It is accepted that the sign of the positive number "+" is represented by the symbol 0, and the sign of the negative number "-" is represented by the symbol 1.

For example, in binary code, using a 6-bit grid, the number 7 in a fixed point form can be represented as -0.00111_2 . The digit to the left of the point is the sign of the number, and five digits to the right of the point is the mantissa of the number in direct code. Here, it is assumed that the comma is fixed to the right of the least significant bit, and the point in the image of the number in this case simply separates the sign bit from the mantissa of the number.

You can use another form of representation of a number in machine form - $[0]00111_2$. Sign digit is indicated by square brackets.

The number of bits in the bit grid reserved for the image of the mantissa of a number determines the range and accuracy of the representation of a fixed point number. The absolute binary value maximum is represented by units in all digits, except for the signed one, i.e. for an integer [2, 8, 9]:

$$|A|_{max} = (2^{(n-1)} - 1), \quad (3.2)$$

where n - full length of the bit grid.

In the case of a 16-bit grid $|A| = (2^{(16-1)} - 1) = 32767_{10}$, i.e. the range of representation of integers in this case will be from $+32767_{10}$ to -32767_{10} .

For the case when the comma is fixed to the right of the lowest digit of the mantissa, i.e. for integers, numbers whose modulus is greater than $(2^{(n-1)} - 1)$ and less than 1 are not represented in fixed point form. Numbers, in absolute value less than the unit of the least significant digit of the bit grid, are called in this case machine zero. Negative zero is not permitted.

In some cases, when it is possible to operate with only modules of numbers, the entire bit grid, including the most significant bit, is reserved for representing the number, which allows to expand the range of the image of numbers.

In computers, in order to simplify the performance of arithmetic operations, special binary codes are used to represent negative numbers: reverse and additional. Using these codes the determination of the sign of the operation result in algebraic addition simplifies. The operation of subtraction (or algebraic

addition) is reduced to the arithmetic addition of operands, the development of signs of bit grid overflow is facilitated. As a result, computer devices performing arithmetic operations are simplified.

One of the methods of subtraction is to replace the sign of the subtracted number with the opposite and to add it to the decreasing number $A-B = A + (-B)$. Thus, the operation of arithmetic subtraction is replaced by the operation of algebraic addition, which can be performed using binary adders. For machine representation of negative numbers, forward, backward, and reverse codes are used.

Therefore, the operation of arithmetic subtraction is replaced by the operation of algebraic addition, which can be performed using binary adders.

For machine representation of negative numbers, there are used *direct, additional, and reverse codes*.

Supplementing the notation of the number in binary code with a sign digit, which is zero for positive numbers and one for negative numbers, we obtain the notation of the sign and modulus of the number in *direct code*. For example, $+6=0.110$, $-6=1.110$.

The *inverse binary code* of a negative number is obtained from the direct code equal to its modulus of a positive number by inverting the values of all its bits. For example, the reverse code of 6 is 1.001.

An *additional code* of a negative number is obtained from its inverse code by adding one to the least significant bit. For example, an additional code of 6 is 1.010.

In table 3.4. there are given decimal digits with a sign and their equivalent representations in direct, reverse and additional codes. The leading digit is used to represent the sign, and the remaining three digits specify the absolute value of the number.

In the general case, the following expressions are true::

Direct number code $A-[A]_{np}$. Let $A=a_1, a_2, \dots, a_m$;

If $A > 0$, then $[A]_{np} = 0, a_1, a_2, \dots, a_m$;

If $A < 0$, then $[A]_{np} = 1, a_1, a_2, \dots, a_m$;

If $A = 0$, then there is an ambiguity: $[0]_{np} = 0, 0$ or $= 1, 0 \dots$

Summarizing the results, we obtain:

$$[A]_{np} = \begin{cases} A, & \text{if } A \geq 0, \\ 1-A, & \text{if } A < 0. \end{cases}$$

Table 3.4

Examples of representing signed digits in four-digit binary codes

Decimal digit with a sign	Code			Decimal digit with a sign	Code		
	Direct	Reverse	Additional		Direct	Reverse	Additional
-7	0.111	0.111	0.111	-0	1.000	1.111	0.000
+6	0.110	0.110	0.110	-1	1.001	1.110	1.111
+5	0.101	0.101	0.101	-2	1.010	1.101	1.110
+4	0.100	0.100	0.100	-3	1.011	1.100	1.101
+3	0.011	0.011	0.011	-4	1.100	1.011	1.100
+2	0.010	0.010	0.010	-5	1.101	1.010	1.011
+1	0.001	0.001	0.001	-6	1.110	1.001	1.010
+0	0.000	0.000	0.000	-7	1.111	1.000	1.001

Reverse code of the number $A-[A]_{o6p}$. The designation \bar{a} means the value, opposite to a (inversion to a), i.e. if $a=1$, then $\bar{a}=0$, and vice versa:

If $A > 0$, then $[A]_{o6p} = [A]_{np} = 0, a_1, a_2, \dots, a_m$;

If $A < 0$, then $[A]_{o6p} = 1, \bar{a}_1, \bar{a}_2, \dots, \bar{a}_m$;

If $A = 0$, then there is an ambiguity: $[0]_{o6p} = 0, 0..0$ or $= 1, 11 \dots 1$.

Summarizing the results, we obtain:

$$[A]_{o6p} = \begin{cases} A, & \text{if } A \geq 0, \\ 10 - 1 \cdot 10^{-n} + A, & \text{if } A < 0. \end{cases}$$

Additional code of the number $A-[A]_{don}$:

If $A > 0$, then $[A]_{don} = [A]_{np} = 0, a_1, a_2, \dots, a_m$;

If $A < 0$, then $[A]_{don} = 1, \bar{a}_1, \bar{a}_2, \dots, \bar{a}_m + 0, 000 \dots 1$;

Summarizing the results, we obtain:

$$[A]_{don} = \begin{cases} A, & \text{if } A \geq 0, \\ 10 + A, & \text{if } A < 0. \end{cases}$$

Let consider that we have only two digits to represent numbers in decimal. Then the maximum number that can be represented will be 99, and the weight of the third non-existent high order will be 10^2 , i.e. 100. In this case, for the number 20, the additional number will be 80, which complements 20 to 100 ($100 - 20 = 80$). Let consider a similar example for numbers represented by 4-bit binary code.

Find the additional number for $0010_2 = 2_{10}$. It is necessary to subtract $[0] 0010$ from $[1] 0000$, then we get the number $[0] 1110$, which is additional code 2. The digit shown in square brackets does not actually exist. But since we have a 4-bit grid, it is basically impossible to perform such a subtraction, and even more we try to get rid of the subtraction. Therefore, an additional number code is obtained by the method described previously, i.e. firstly we get the inverse code of the number, and then add 1 to it. After doing all this with our number (2), it's easy to make sure that you get a similar answer.

Let consider a few examples of algebraic addition in reverse code. Obviously, in the absence of overflow, four cases of a combination of signs and term modules are possible.

Case 1. $A > 0, B > 0, A + B < 1$. This case corresponds to the usual addition of direct codes of numbers: $[A > 0]_i + [B > 0]_i = A + B$.

Case 2. $A > 0, B < 0, A + B > 0$. $[A > 0]_i + [B < 0]_i = A + 2 + B - 2^{-n}$. We call this result preliminary. The true value of the result in the case under consideration (the sum is positive) will be $A + B$. Therefore, the preliminary result needs to be corrected by subtracting 2 and adding -2^{-n} .

Case 3. $A > 0, B < 0, A + B < 0$. $[A > 0]_i + [B < 0]_i = A + 2 + B - 2^{-n}$. This result is correct, since the case of a negative sum is considered.

Case 4. $A < 0, B < 0, |A + B| < 1$. $[A < 0]_i + [B < 0]_i = 2 + A - 2^{-n} + 2 + B - 2^{-n}$. Here, the preliminary result needs to be corrected by subtracting 2 and adding 2^{-n} , as in case 2, since the true value of the negative amount presented in the reverse code, $A + B + 2 - 2^{-n}$.

In cases 2 and 4, the same correction is required: $-2 + 2^{-n}$, and only in these two cases occur the transfer from the sign digit. Indeed, in case 4 both sign digits are equal to 1, and in case 2, the sign of the result is 0, which for different signs of the terms can be obtained only when a transfer from the first digit to zero (sign) appears, and therefore there will be a transfer from the sign digit. The weight of the sign digit corresponds to 2^0 , and the weight of the transfer from it is -2^1 . Therefore, ignoring the transfer from the sign discharge, we *subtract from the result 2*, which corresponds to the first term of the correction expression. To account for the second term, we should add 1 to the lowest order of the amount, the weight of which is 2^{-n} .

In cases 1 and 3, there does not occur a transfer from the sign digit and the correction of the result is not required.

Therefore, in order to perform the algebraic addition of binary numbers represented in the reverse code, it is enough, without analyzing the ratio of signs and modules, to add the numbers, including sign digits, according to the rules of binary arithmetic, and the transfer occurring in the sign bit should be added to the low order result, thereby correcting the preliminary amount. The resulting code is the algebraic sum of terms represented in the reverse code.

Example 1.

Add two numbers in reverse code.

$$A = +0,1101 \qquad [A]_d = 0,1101 \qquad [A]_i = 0,1101$$

$$\begin{array}{rcl}
B = - 0,0011 & [B]_d = 1,0011 & [B]_i = \underline{1,1100} \\
& & 1 \leftarrow 0,1001 \\
& & \underline{\qquad\qquad\qquad 1} \\
C = 0,1010 & \leq [C]_d = 0,1010 & \leq [C]_i = 0,1010
\end{array}$$

The above mentioned example corresponds to the above described case 2; the transfer occurring in the sign bit is cyclically transferred to the least significant bit of the preliminary result.

Example 2.

Add two numbers in reverse code (case 3).

$$\begin{array}{rcl}
A = - 0,1101 & [A]_d = 1,1101 & [A]_i = 1,0010 \\
B = + 0,0011 & [B]_d = 0,0011 & [B]_i = \underline{0,0011} \\
C = - 0,1010 & \leq [C]_d = 1,1010 & \leq [C]_i = 1,0101
\end{array}$$

Example 3.

Add two numbers in reverse code (case 4).

$$\begin{array}{rcl}
A = - 0,0101 & [A]_d = 1,0101 & [A]_i = 1,1010 \\
B = -0,0110 & [B]_d = 1,0110 & [B]_i = \underline{1,1001} \\
& & 1 \leftarrow 1,0011 \\
& & \underline{\qquad\qquad\qquad 1} \\
C = -0,1011 & \leq [C]_d = 1,1011 & \leq [C]_i = 1,0100
\end{array}$$

Example 4.

Add two numbers in reverse code (same modules but different signs).

$$\begin{array}{rcl}
A = -0,0101 & [A]_d = 1,0101 & [A]_i = 1,1010 \\
B = +0,0101 & [B]_d = 0,0101 & [B]_i = \underline{0,0101} \\
C = -0,0000 & \leq [C]_d = 1,0000 & \leq [C]_i = 1,1111
\end{array}$$

Thus, a zero in the reverse code is “positive” and “negative” and adding to the number of “negative” zero, as well as “positive”, results in the value of the first term.

Example 5.

Add two numbers in reverse code: 3 + (-0).

$$\begin{array}{rcl}
A = + 0,0011 & [A]_d = 0,0011 & [A]_i = 0,0011 \\
B = -0,0000 & [B]_d = 1,0000 & [B]_i = \underline{1,1111} \\
& & 1 \leftarrow 0,0010 \\
& & \underline{\qquad\qquad\qquad 1} \\
C = + 0,0011 & \leq [C]_d = 0,0011 & \leq [C]_i = 0,0011
\end{array}$$

Now let consider the cases when $|A+B| \geq 1$, which corresponds to bit grid overflow. Obviously, given that $|A| < 1$ and $|B| < 1$, overflow is possible only by adding numbers with the same signs. Let look at some examples.

Example 6.

Add two numbers in reverse code: $13/16 + 5/16 = 18/16$.

$$\begin{array}{lll}
 A = +0,1101 & [A]_d = 0,1101 & [A]_i = 0,1101 \\
 B = +0,0101 & [B]_d = 0,0101 & [B]_i = \underline{0,0101} \\
 C = -0,1101 \leq & [C]_d = 1,1101 & \leq [C]_i = 1,0010
 \end{array}$$

Example 7.

Add two numbers in reverse code: $(-11/16) + (-8/16) = (-19/16)$.

$$\begin{array}{lll}
 A = -0,1011 & [A]_d = 1,1011 & [A]_i = 1,0100 \\
 B = -0,1000 & [B]_d = 1,1000 & [B]_i = \underline{1,0111} \\
 & & \quad \quad \quad \mathbf{1} \leftarrow 0,1011 \\
 & & \quad \quad \quad \underline{\quad \quad \quad 1} \\
 C = +0,1100 \leq & [C]_d = 0,1100 & \leq [C]_i = 0,1100
 \end{array}$$

Thus, the sign of overflow in the reverse code can be considered the sign of the result, opposite to the same signs of the terms

$$OV = \bar{a}_0 \bar{b}_0 c_0 \vee a_0 b_0 \bar{c}_0.$$

Example 8.

Add two numbers in reverse code: $(A > B, B > 0, |A+B|=1)$.

$$\begin{array}{lll}
 A = +0,0111 & [A]_d = 0,0111 & [A]_i = 0,0111 \\
 B = +0,1001 & [B]_d = 0,1001 & [B]_i = \underline{0,1001} \\
 C = +0,1111 \leq & [C]_d = 0,1111 & \leq [C]_i = 1,0000
 \end{array}$$

Example 9.

Add two numbers in reverse code $(A < 0, B < 0, |A+B|=1)$.

$$\begin{array}{lll}
 A = -0,0111 & [A]_d = 1,0111 & [A]_i = 1,1000 \\
 B = -0,1001 & [B]_d = 1,1001 & [B]_i = \underline{1,0110} \\
 & & \quad \quad \quad \mathbf{1} \leftarrow 0,1110 \\
 & & \quad \quad \quad \underline{\quad \quad \quad 1} \\
 C = +0,1111 \leq & [C]_d = 0,1111 & \leq [C]_i = 0,1111
 \end{array}$$

So, using reverse code in algebraic addition/subtraction operations allows to:

- use only the action of arithmetic addition of binary codes;
- receive the true value of the result sign, performing the same actions on the signed digits of the operands as on the number digits;
- detect overflow of the bit grid.

Another advantage of using reverse code can be considered in the simplicity of the mutual conversion of direct and reverse codes.

However, the use of the inverse code has one significant disadvantage - the correction of the preliminary sum requires the addition of a unit to its least significant digit and can cause (in some cases) the spread over the whole number,

which, in turn, leads to a doubling of the summation time. In order to overcome this disadvantage, you can use *additional code* instead of reverse code.

Let consider a few examples of algebraic addition in additional code.

Case 1.

$A > 0, B > 0, A + B < 1$. This case corresponds to the usual addition of direct codes of numbers: $[A > 0]_c + [B > 0]_c = A + B$.

Case 2.

$A > 0, B < 0, A + B > 0$. $[A > 0]_c + [B < 0]_c = A + 2 + B$.

The true value of the result in this case (the sum is positive) will be $A + B$. Therefore, the preliminary result needs to be corrected by subtracting of 2.

Case 3.

$A > 0, B < 0, A + B < 0$. $[A > 0]_c + [B < 0]_c = A + 2 + B$. This result is correct, since the case of a negative sum is considered.

Case 4.

$A < 0, B < 0, |A + B| < 1$. $[A < 0]_c + [B < 0]_c = 2 + A + 2 + B$. Here the preliminary result needs to be corrected, since the true value of the negative amount presented in the reverse code $A + B + 2$.

In order to convert the numbers of a negative binary code to an additional code, you must first convert it to the reverse one, and then, after inverting, add one to the highest digit, see table. 3.5.

Table 3.5
Examples of converting binary numbers to additional code

Number	Direct code	Reverse code	Additional code
-0,0111	$[A]_d = 1,0111$	$[A]_r = 1,1000$	$[A]_c = 1,1001$
-0,1000	$[A]_d = 1,1000$	$[A]_r = 1,0111$	$[A]_c = 1,1000$
-0,0101	$[A]_d = 1,0101$	$[A]_r = 1,1010$	$[A]_c = 1,1011$

Example 1.

Add two numbers in additional code: $(+13/16) + +(-3/16) = (+10/16)$.

$$\begin{array}{lll}
 A = +0,1101 & [A]_d = 0,1101 & [A]_c = 0,1101 \\
 B = -0,001 & [B]_d = 1,0011 & [B]_c = \underline{1,1101} \\
 C = +0,1010 & \Leftarrow [C]_d = 0,1010 & \Leftarrow [C]_c = 1,1010
 \end{array}$$

Example 2.

Add two numbers in additional code (case 3).

$$\begin{array}{lll}
 A = -0,1101 & [A]_d = 1,1101 & [A]_c = 1,0011 \\
 B = +0,0011 & [B]_d = 0,0011 & [B]_c = \underline{0,0011}
 \end{array}$$

$$C = -0,1010 \Leftarrow [C]_d = 1,1010 \Leftarrow [C]_c = 1,0110$$

Example 3.

Add two numbers in additional code (case 4).

$$\begin{array}{lll} A = -0,0101 & [A]_d = 1,0101 & [A]_c = 1,1011 \\ B = -0,0110 & [B]_d = 1,0110 & [B]_c = \underline{1,1010} \\ C = -0,1011 \Leftarrow & [C]_d = 1,1011 \Leftarrow & [C]_c = 11,0101 \end{array}$$

Example 4.

Add two numbers in additional code (same modules but different signs).

$$\begin{array}{lll} A = -0,0101 & [A]_d = 1,0101 & [A]_c = 1,1011 \\ B = +0,0101 & [B]_d = 0,0101 & [B]_c = \underline{0,0101} \\ C = +0,0000 \Leftarrow & [C]_d = 0,0000 \Leftarrow & [C]_c = 10,0000 \end{array}$$

Obviously, for additional code, as well as for the reverse, the expression is true

$$OV = \bar{a}_0 \bar{b}_0 c_0 \vee a_0 b_0 \bar{c}_0.$$

Example 5.

Add two numbers in additional code of the number for the case $A < 0, B < 0, |A+B|=1$. $(-11/16) + (-5/16) = (-16/16)$.

$$\begin{array}{lll} A = -0,1011 & [A]_d = 1,1011 & [A]_c = 1,0101 \\ B = -0,0101 & [B]_d = 1,0101 & [B]_c = \underline{1,1011} \\ C = -0,0000 \Leftarrow & [C]_d = 1,0000 \Leftarrow & [C]_c = 10,0000 \end{array}$$

No overflow by expression $OV = \bar{a}_0 \bar{b}_0 c_0 \vee a_0 b_0 \bar{c}_0$ was detected. However, the result of the operation is "negative zero", which cannot be used in additional code. I.e. for the case $A < 0, B < 0, |A+B|=1$, the result code is **100.0** as a sign of result overflow.

The value of the overflow sign in the additional code can be obtained in accordance with the expression

$$OV = \bar{a}_0 \bar{b}_0 c_0 \vee a_0 b_0 \bar{c}_0 \vee c_0 \bar{c}_1 \bar{c}_2 \dots \bar{c}_n.$$

Summary. The use of an additional code, in comparison with the reverse, has one significant advantage - the correction of the result can be conducted simply by discarding the transfer from the sign digit and does not require additional time. The disadvantages of using an additional code include:

- a more complex procedure for the inverse transformation of a $PC \leftarrow \rightarrow DC$, which requires additional time;
- problems with overflow detection.

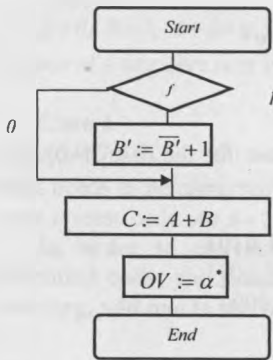
In order to eliminate these problems, the data in the CA memory is often stored in additional code.

The fig. 3.2-3.4 show the algorithms for performing subtraction, addition operations for numbers represented in the additional, direct, and reverse codes, respectively.

Nowadays, computers typically use additional code to represent negative numbers in fixed point format.

Binary numbers multiplication is usually done in direct code. The sign of the product is determined by the sign digits of the multiplicand and the multiplier in accordance with the following rule. **If the sign of the operands is the same, then the sign of the product is positive. Otherwise, the sign of the product is negative.**

The sign of the product of two numbers does not affect the algorithm of the operation of multiplying the modules of these numbers.



where

OV - overflow sign value;

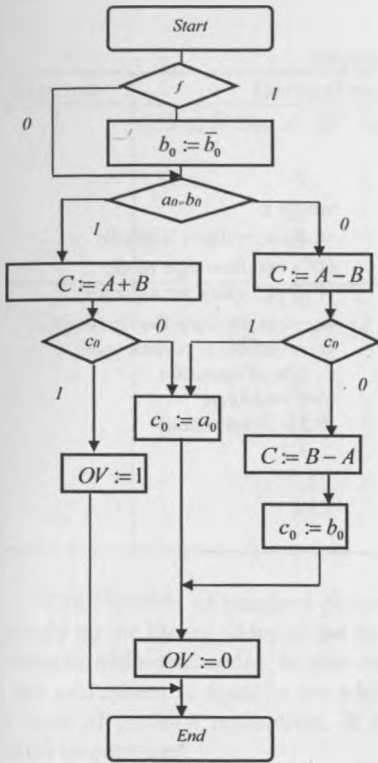
B' - number module;

c_i - transfer from the sign digit;

α^* - overflow in additional code;

f - type of operation ($f=0$ - addition, $f=1$ - subtraction)

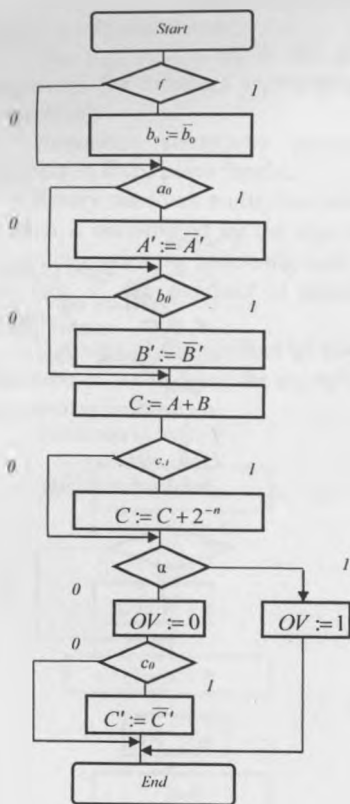
Fig. 3.2. Addition/Subtraction Algorithm in Additional Code



where
 a_n, b_0, c_o - digits (signed);
 OV - overflow sign value;
 A', B', C' - number modules;
 c_i - transfer from the sign digit;
 α - overflow in reverse code;
 f - type of operation
 $(f=0$ - addition,
 $f=1$ - subtraction)

$$A = a_0 a_1 a_2 \dots a_n, B = b_0 b_1 b_2 \dots b_n, C = A + B = c_0 c_1 c_2 \dots c_n,$$

Fig. 3.3. Addition/Subtraction Algorithm in Direct Code



where
 a_o, b_o, c_o – digits (signed);
 OV - overflow sign value;
 A', B', C' - number modules;
 c_1 – transfer from the sign digit;
 α - overflow in reverse code;
 $f=0$ – addition,
 $f=1$ – subtraction)

Fig. 3.4. Addition/Subtraction Algorithm in Reverse Code

Let consider one of the possible variants of the multiplication algorithm when the operands are presented in direct code. Before performing the multiplication procedure itself, according to the usual arithmetic rules of multiplication, the sign of the product is determined and stored. Then both operands are represented in direct code, and the procedure of multiplication by one of the two previously described methods is performed with the obligatory control of the overflow of the bit grid.

Table 3.6 shows examples of the multiplication operation for two fixed point numbers.

Table 3.6

Multiplication Examples

Example	Decimal numbers	Binary numbers
1	$A=5_{10}, B=3_{10}, A \cdot B = 5_{10} \cdot 3_{10} = 15_{10}$ $\begin{array}{r} 5 \\ x \ 3 \\ \hline 15 \end{array}$	$A=0.0101, B=0.0011$ $\begin{array}{r} 0.0101 \\ x \ 0.0011 \\ \hline 0101 \\ + \ 0101 \\ \hline 0.1111 = 15_{10} \end{array}$
2	$A=13_{10}, B=11_{10}$ $A \cdot B = 13_{10} \cdot 11_{10} = 143_{10}$ $\begin{array}{r} 13 \\ x \ 11 \\ \hline 13 \\ 13 \\ \hline 143 \end{array}$	$A=1101, B=1011$ $\begin{array}{r} 1101 \\ x \ 1011 \\ \hline 1101 \\ 1101 \\ 0000 \\ +1101 \\ \hline 10001111 \end{array}$

Multiplication of numbers presented in a fixed point form can be organized not only on the binary adder of the direct code, but also on the binary adders of the reverse or additional codes. In this case, the product of additional or inverse codes of the multipliers is equal to the additional or inverse code, respectively, only in the case of positive multipliers. If the multipliers are negative, then the result should be corrected.

Division of binary numbers is much the same as division of decimal numbers. The division algorithm consists in the fact that the divisor at each step is subtracted from the dividend as many times (starting with the highest digits) as possible in order to obtain the smallest positive remainder. Then a digit equal to the number of divisors contained in the dividend at this step is written into the next digit of the quotient. Therefore, dividing, the subtraction operation is repeated until the reduction is less than the subtraction. The number of these repetitions shows how many times the subtracted fits into the minuend. Table 3.7 shows examples of the division operation for two fixed point numbers.

Let explain example 2 shown in table 3.7. Binary, as well as decimal division, begins with the analysis of the dividend ($A=11001100$) and the divisor ($B=1100$). Since the divisor fits in 1100, the unit is recorded in the senior digit of the quotient. The divisor is multiplied by 1 and subtracted from 1100. The difference is 0. The remainder is combined with the value of the next digit of the dividend equal to 1. Since the divisor ($B=1100$) fits 1 for 0 times, we write 0 in the next highest digit of the quotient field, and the number 1 is combined with the next digit of the dividend, etc. until the dividend is exhausted.

Table 3.7

Division Examples

Example	Dividend, divisor, result
1	$A=45_{10}=1001101_2$, $B=5_{10}=101_2$, $A/B=45_{10}/5_{10}=9_{10}$ $\begin{array}{r} 1001101 \\ / \quad 101 \\ \underline{\quad 01} \\ \quad 010 \\ \underline{\quad 0101} \\ \quad 1001 \end{array}$
2	$A=204_{10}=11001100_{(2)}$, $B=12_{10}=1100_{(2)}$, $A/B=204_{10}/12_{10}=17_{10}$ $\begin{array}{r} 11001100 \quad \quad 1100 \\ \underline{1100} \quad \quad 10001 \\ \text{remainder } 00001 \\ \quad \underline{- 0} \\ \quad \quad 11 \\ \quad \underline{- 0} \\ \quad \quad 110 \\ \quad \underline{- 0} \\ \quad \quad 1100 \\ \quad \underline{- 1100} \\ \quad \quad 0000 \end{array}$

For example, let divide $A=35_{10}=0.100011_2$ on $B=5_{10}=101_2$ ($S_A=1.011_A$). In register C, as in the previous example, the quotient is formed.

0.100011	- dividend
<u>+ 1.011000</u>	- first subtraction of the divisor
1.111011	- $C=0$ restore the remainder to the dividend
<u>+ 0.101000</u>	
0.10001	- shift left the remainder
1.00011	
<u>+ 1.01100</u>	
0.01111	- $C=01$, shift left the remainder
0.1111	
<u>+ 1.0110</u>	
0.0101	- $C=011$, shift the remainder
0.101	
<u>+ 1.011</u>	
0.000	obtain the result - $C=0111=7_{10}$.

Of course, the computer cannot speculate on how many times the divisor fits in one or another number, so the whole division process is reduced to the subtraction and shift operations, see table. 3.8. This method of division is called *division with recovery of the remainder* [11].

We will demonstrate this method with the same example, but first we will present the divisor $B = (1100)$ in an additional code, which will allow to limit ourselves with addition in all cases when it is necessary to add or subtract: $B = 1100_{\text{up}} = 1.0100_{\text{d}}$. The quotient is formed in some register C , whose unfilled digits will be denoted by X .

Table 3.8

Example of division operation with remainder recovery

Step	Operation	Comments
1	$\begin{array}{r} 0.11001100 \\ +1.01000000 \\ \hline 0.00001100 \end{array}$ first remainder	We begin to subtract the divisor from the dividend. If the remainder turns out to be positive, then we write 1 in the digit of the quotient, otherwise - 0. The first (senior) bit of the quotient is 1, because the remainder was positive: $C = 1XXXX$.
2	$\begin{array}{r} 0.00011000 \\ +1.01000000 \\ \hline 1.01011000 \end{array}$ second remainder	We shift the first remainder by one digit to the left and subtract the divisor from it. The remainder is negative, so in the next digit of the quotient we write 0, $C = 10XXX$.
3	$\begin{array}{r} 1.01011000 \\ +0.11000000 \\ \hline 0.00011000 \end{array}$ shifted first remainder	The divisor bits are returned back to the first remainder, i.e. add the divisor (in direct code) and the second remainder.
4	$\begin{array}{r} 0.00110000 \\ +1.01000000 \\ \hline 1.01110000 \end{array}$ third remainder	We shift the shifted first remainder by one bit to the left and subtract the divisor from it. The third remainder is negative, therefore, the next (third) digit of the quotient is 0, $C = 100XX$.
5	$\begin{array}{r} 1.01110000 \\ +0.11000000 \\ \hline 0.00110000 \end{array}$ double shifted first remainder	Return the divisor to the third remainder.
6	$\begin{array}{r} 0.01100000 \\ 1.01000000 \\ \hline 1.10100000 \end{array}$ fourth remainder	We shift the double shifted first remainder by one digit to the left and subtract the divisor. The fourth remainder is negative again, so $C = 1000X$.
7	$\begin{array}{r} 0.11000000 \\ +1.01000000 \\ \hline 0.00000000 \end{array}$ fifth remainder	Add the divisor to the fourth remainder, shift the result one digit to the left, and then subtract the divisor again. The remainder is positive, which means $C = 10001 = 17_{(10)}$.

The division of numbers represented in a fixed point form can also be done on binary adders of the reverse and additional code. Before performing the procedure for dividing numbers in a fixed point form, the quotient is determined and stored. Further, both operands are represented in direct code, and the divisor is also in the additional one in order to replace the subtraction of the divisor by addition, and the division procedure itself is performed according to the method described above with the obligatory control of overflow of the bit grid. If the quotient is negative, then the answer, if necessary, is presented in an additional code.

Let consider another example. Let divide $A=506_{10}=0.11111010$ на $B=23_{10}=0.10111$.

After each subtraction, we check the sign of the result. With a positive sign, we write 1 in the lowest digit of the quotient and proceed to the next subtraction step. With a negative sign, we write 0 in the lowest digit of the quotient and restore the remainder by adding a divisor to it. After that we carry out the next subtraction. Each subsequent subtraction is performed after the divisor is shifted by one digit to the right or the dividend - by one digit to the left. After processing all digits of the dividend, the last subtraction result represents the remainder of the division. In the considered examples the remainder is zero. If the number is not completely divisible, then the subtraction is performed until the required number of digits of the quotient is obtained after the decimal point.

0.11111010	- dividend
+ 1.01001	- first subtraction of the divisor
10.010001	1- the result is positive
+ 1.01001	- second subtraction of the divisor
1.11010	0 - the result is negative
+ 0.10111	- adding a divisor
10.100010	- recovered remainder
+ 1.01001	- third subtraction of the divisor
10.010111	1 - the result is positive
+ 1.01001	- fourth subtraction of the divisor
10.000000	1- the remainder is zero.

The considered division algorithm with the remainder recover (adding to the negative remainder of the dividend) provides for a different sequence of operations for positive and negative remainder. Let have a look at one of our examples, another method used in digital automata for the operation of dividing binary numbers with a fixed point - *the division method without restoring the remainder*. As already noted, the basis of the division is the operation of subtraction in order to obtain the remainder, the sign of which determines the number of quotients. In this case, the remainder sign determines not only the next digit of the quotient, but also the nature of the following procedure: add the divisor to the shifted remainder, if this remainder is less than 0, and the subtraction of the divisor from the shifted remainder if the remainder is greater than or equal to 0. This division method is called *division without remainder recovery*.

For example, let divide $A=506_{10}=0.11111010$ on $B=23_{10}=0.10111$.

0.11111010	- dividend is positive
+ 1.01001	- first subtraction of the divisor
10.010001	1 - the result is positive
+ 1.01001	- second subtraction of the divisor
1.110100	0 - the result is negative
+ 0.10111	- adding a divisor

$$\begin{array}{r} 100.010111 \\ + \quad 1.01001 \\ \hline 10.000000 \end{array}$$

- 1 - the result is positive
- third subtraction of the divisor
- 1 - the remainder is zero.

3.2.2. Floating point number representation form

An exponential form, or a floating point form, is used to expand the range and to reduce the relative error in the representation of numbers in computing devices.

In general, a floating point number is represented as:

$$A = \pm m \cdot q^{\pm p}, \quad (3.3)$$

where m – mantissa of a number;

q – number system base;

p – the order of the number, which for simplicity in the examples will sometimes be depicted as $\pm p$ (do not confuse with the transfer to senior digit).

Then it is obvious that p – an exponent of the order, which is usually called simply the order of the number, because basically always $q = 2$. Therefore, the previous expression can be written as follows:

$$A = \pm m_A \cdot \pm p_A, \quad (3.4)$$

Taking into account that in computers usually $q = 2$.

So, for example, the number 1964 in a floating point form in decimal notation can be written as follows: $1964 \cdot 10^0$; $0,1964 \cdot 10^4$; $19,64 \cdot 10^2$; $196400 \cdot 10^{-2}$ etc.

The ambiguity of the record lies in the fact that the mantissa can take an infinite number of different values, less than one, with the corresponding order values (i.e., the comma in the mantissa can float).

It is customary to represent a floating point number in the so-called *normalized* form for the most accurate representation of the number. If the inequality is true

$$q^{-1} \leq |m| < 1,$$

and in case of the binary number system:

$$0.5 \leq |m| < 1,$$

then it is believed that the number is presented in normalized form. For example, $0,1964 \cdot 10^4$ is a normalized form of the number 1964 in floating point form in decimal notation. Similar considerations apply to binary numbers. For example, the normalized form for a binary number $1110110,011_2$ has the following form $0,1110110011 \cdot 10^{11}$.

Thus, the mantissa has a regular fraction in the normalized binary number in the floating point form, and the mantissa always has a 1 in the highest digit. The operation of reducing the number to normalized form is called *normalization*.

Normalization of numbers in the computer is performed either automatically or by a special program.

Since the number system for a given digital automaton (computer) remains constant, then presenting a number in a floating point format, there is no need to indicate its base, it is only necessary to present an exponent of the order of the number.

In order to represent a binary number in floating point form in a bit grid, two groups of bits should be distinguished. The first group (r -digits) is designed to accommodate the mantissa, the second ($n-r$ -digits) is used to place the order code (excluding the sign digits of the mantissa and the order), see Fig. 3.5.



Fig. 3.5. Floating point representation

Let assume that binary numbers are placed in a bit grid in a normalized form. This means that the numerical values of the mantissa are always greater than or equal to 2^{-1} , but do not exceed 1. The "weight" of the least significant bit of the mantissa is 2^{-r} , and the "weight" of the highest one is 2^{-1} . The maximum representable number will be at the maximum values of the mantissa and the order: $A = (1 - 2^{-r}) \cdot 2^{(2^r - 1)}$, and the minimum representable number at the minimum value of the mantissa and the maximum negative modulus: $A = 2^{-1} \cdot 2^{(2^r - 1)}$.

Usually, in the floating point format, instead of the p index, the so-called characteristic ("shifted order") is used γ :

$$\gamma = \pm p + l,$$

where l - excess (shift), the value of which is selected so that when the value of the indicator changes from a certain minimum value $-|p_{\max}|$ to the maximum $+|p_{\max}|$, the characteristic of r varied from 0 to r_{\max} . Therefore, the characteristic does not change its sign.

Let have a look on some examples of representing numbers in floating point form. First, we note that the exponent of two in the digits of a bit grid of length n allocated for representing integers varies from 0 to $n-1$, and in the case of correct fractional numbers, from -1 to $-n$.

If 4 digits are allocated to represent the order index, then $A = 2^3 = 8_{10} = 1000_2$. For this case, the table 3.9 shows the values of the order index and the mantissa for some numbers presented in the form of a floating point.

Table 3.9

**Indicators of the order, characteristics and mantissa of numbers
represented in floating point form**

Number A_{10}	Number order p_{10}	Mantissa m_2
0	0	0,0
1	1	0,1
2	2	0,1
3	2	0,11
0,5	0	0,1
0,25	-1	0,1
0,75	0	0,11
0,375	-1	0,11

The length of the bit grid allocated for the characteristic determines the range of representation of numbers in floating point format.

The main advantage of representing numbers in a floating point form is the large range of machine numbers and the high accuracy of their representation.

The range is determined by the length of the bit grid allocated for the characteristic, and the accuracy is determined by the length of the bit grid allocated for the mantissa.

Fixed point numbers most often have a word format (2 bytes) or a half-word (1 byte). Floating point numbers - double format (4 bytes) and an extended word (8 bytes) (PC math coprocessors can work with 10-byte words). A sequence of several bits or bytes is often called a *data field*. Variable-length fields can be any size from 0 to 255 bytes, but necessarily equal to an integer number of bytes.

As an example, let consider the notation of the decimal number 193_{10} (11000001_2) in the bit grid of a PC, for a fixed point number in the format of a word with a sign, Fig. 3.6 and for a floating point number in double word format, fig. 3.7.

Binary-coded decimal numbers can be represented on a PC with variable-length fields in the so-called packed (Fig. 3.8) and unpacked formats (Fig. 3.9). In a packed format, 4 binary digits (half byte) are allocated for each decimal digit, while the sign of the number is encoded in the rightmost half byte of the number (1100 - the "+" sign and 1101 - the "-" sign).

Digit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Number	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1
	sign	The absolute value of a number														

Fig. 3.6. Fixed point number in a signed word format

Digit	31	30	29	28	27	26	25	24	23	22	20	21	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Number	1	0	0	0	1	0	0	0	1	1	0	0	0	0	0	1	0															0	0
		Order																Mantissa															

Рис. 3.7. Floating point number in a double word format

On fig. 3.8 and 3.9 Dg - a digit, a sign - a sign of a number. The packed format is used in the PC performing operations of addition and subtraction of binary decimal numbers. In the unpacked format, an integer byte is allocated for each decimal digit, while the highest half bytes (zone) of each byte, except the lowest, are filled in with the code 0011 on the PC, and decimal digits are normally encoded in the lower (left) half bytes. The highest half bytes (zone) of the least significant byte is used to encode the sign of the number, see Fig. 3.8.



Fig. 3.8. Packed format field structure

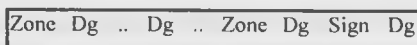


Fig. 3.9. Unpacked format field structure

For example, the decimal number $-193_{10} = -000110010011_{2-10}$ on the PC will be presented, respectively, in packed and unpacked formats, as shown on Fig. 3.10 and 3.11.

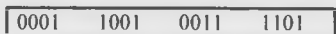


Fig. 3.10. Packed format field structure

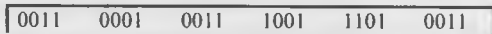


Fig. 3.11. Unpacked format field structure

Next, we consider examples of arithmetic operations of addition (subtraction) with numbers presented in floating point format.

For a simpler perception of the material, we give examples of addition in the decimal system.

$A=0,315291 \cdot 10^{-2}$, $B=0,114082 \cdot 10^{+2}$. The mantissa of numbers are normalized (i.e., the highest digit of the mantissa modulus is not 0, for example, for direct codes $0,2364 \cdot 10^4 \approx 0,0024 \cdot 10^6$).

Before adding the mantissa, you need to convert the numbers so that they have the same orders.

1st method - **Decrease a larger order to a smaller one.**

2nd method - **Increase a smaller order to a larger one.**

$$\begin{array}{r|l}
 A= & 0,315291 \cdot 10^{-2} \\
 B= 1140 & 0,820000 \cdot 10^{-2} \\
 \hline
 C= & 1,135291 \cdot 10^{-2}
 \end{array}$$

a)

$$\begin{array}{r|l}
 A= 0,000031 & 5291 \cdot 10^{-2} \\
 B= 0,114082 & \cdot 10^{-2} \\
 \hline
 C= 0,114114 & \cdot 10^{-2}
 \end{array}$$

b)

In the first case (a), the senior digits of the shifted mantissa go beyond the bit grid, and the addition result is incorrect. In the second case (b), the lower digits of the mantissa are lost during the shift, which does not affect the accuracy of the result. **Therefore, aligning the orders, you should always increase the smaller order to a larger one with a corresponding decrease of the mantissa.**

To align the orders, we should determine the difference in the orders of the terms and shift the mantissa of the number with the lower order to the right by the value of this difference. If the difference of orders exceeds the bit depth of the mantissa field, then the value of the term with a lower order can be taken as 0, and the summation result will be equal to the term with a higher order.

After aligning the orders, the mantissa should be added and the order of any of the terms should be determined as the order of the result. If overflow occurs at adding the mantissa, the result can be corrected by shifting the mantissa sum by one digit to the right and adding one to the order of the result.

It should be remembered that if as a result of this addition there will be an overflow of the bit grid of orders, the result will be incorrect - a **positive overflow** $OV:=1$.

Positive overflow example ($OV:=1$).

$$\begin{array}{l}
 A=0,96502 \cdot 10^{+2}, \\
 B=0,73005 \cdot 10^{+1}.
 \end{array}$$

$$\begin{array}{l}
 A= 0,96503 \cdot 10^{+2} \\
 B= 0,07300 \cdot 10^{+2} \\
 \hline
 \end{array}$$

$$\begin{array}{l}
 C= 1,03803 \cdot 10^{+2} \quad - \text{mantissa overflow!} \\
 C= 1,0380 \cdot 10^{+2} \quad - \text{correct result.}
 \end{array}$$

As a result of the algebraic addition of the mantissa, the result may not be normalized. To normalize the result, it is necessary to shift the mantissa result to the left until a digit other than 0 appears in the most significant digit, accompanying each shift by a decrease of 1 order of the result.

$$\begin{array}{l}
 A= 0,24512 \cdot 10^{-8} \\
 B= -0,24392 \cdot 10^{-2} \\
 \hline
 \end{array}$$

$$C= 0,00120 \cdot 10^{-8} = 0,12000 \cdot 10^{-10}$$

During the process of decreasing the order during normalization, it may turn out that the order modulus has exceeded the maximum value placed in the order field. This case is called **negative overflow**. It can be avoided by leaving the result unnormalized, but it is generally accepted that storing an abnormalized result in

memory is unacceptable. Therefore, in the event of a negative overflow, the result assumes the value "machine zero".

Stages of the algebraic addition of floating point numbers:

1. Alignment orders.

2. Algebraic addition of mantissas as fixed point numbers.

3. Normalization of the result.

The fig. 3.12 shows the algorithm for adding (subtracting) floating point numbers.

During the operation of multiplying two numbers represented in the form of a floating point, their mantissas are multiplied as fixed point numbers, and the orders are added.

In both cases, an overflow check is required. But, because the characteristics of numbers really add up, then the multiplication is performed according to the following formula:

$$A_1 \cdot A_2 = m_1 \cdot 2^{p_1} \cdot m_2 \cdot 2^{p_2} = (m_1 \cdot m_2) \cdot 2^{p_1 + p_2 - l}, \quad (3.5)$$

where $p_1 + p_2 - l$ - characteristic of the result.

The sign of the answer is determined in the usual way. If the response is not normalized, then there are performed normalization and rounding of the response.

The presence of denormalization to the left is determined by the same methods as overflow of the bit grid. Denormalization to the right is characterized by the same value of the mantissa digits on both sides of the comma.

An example of finding the product of two numbers represented in exponential form $A=(+0,10101 \cdot 10^{100})_2$, $B=(-0,10001 \cdot 10^{011})_2$. The mantissa of the product is in denormalized form - 0.0101100101 has a negative sign, the order of the result is 111 After normalization, the order value should be reduced by one. Finally we get $-0,101100101 \cdot 10^{110}_2$, i.e. the result is presented in normal form.

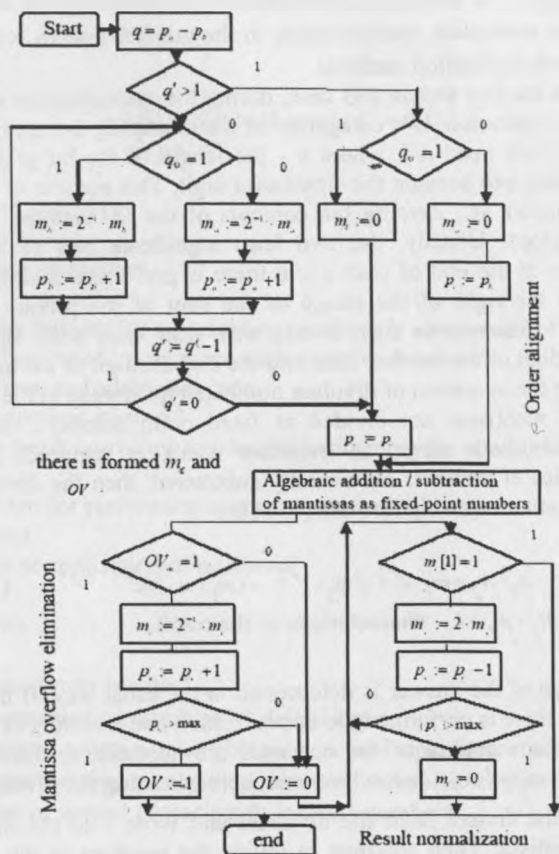


Fig. 3.12. Floating point numbers addition algorithm

By the time of execution, the multiplication operation refers to long operations. The time spent on multiplying of two numbers in the direct code can be estimated by the following formula (for the case of sequential analysis of the bits of the multiplier):

$$t_{y_{mul}} = \sum_{i=1}^n (t_{c_{db}} + P_i t_{cn}),$$

where $t_{c_{db}}$ – one digit shift time;

t_{cn} – summation time on the adder;

P_i – probability of occurrence of a unit in the multiplier digits;

n – amount of the multiplier digits.

There are several methods to accelerate the multiplication procedure: analysis of two digits of the multiplier at the same time, analysis of an arbitrary number of digits of the multiplier, multiplication in the number system with the base $q = 2^k$ and matrix multiplication methods.

Due to the fact that in this case, during the multiplication operation, on each cycle of the operation, two categories of the multiplier are analyzed at once, then such cycles will need $n/2$, where n - the length of the bit grid of the multiplier without taking into account the significant digit. This number of cycles is recorded in some counter SC . Zeroing the contents of the SC counter, the multiplication procedure stops. Usually, the two least significant bits of the multiplier are analyzed, so at the end of each cycle there is performed a simultaneous shift by two bits to the right of the image of the sum of the private products and the multiplier. Moreover, in such a way that with each such shift the next least significant digit of the number falls into the highest digit of the multiplier mantissa.

During the operation of dividing numbers represented in the form of a floating point, their mantissas are divided as fixed point numbers, and the orders are subtracted. In both cases, an overflow check is required. But, because the characteristics of numbers are actually subtracted, then the division is performed according to the following formula [7, 11]:

$$A_1/A_2 = m_1 \cdot 2^{p_1} / m_2 \cdot 2^{p_2} = (m_1/m_2) \cdot 2^{p_1-p_2+l}, \quad (3.6)$$

where $p_1 - p_2 + l$ - characteristic of the result.

The sign of the answer is determined in the usual way. If the response is not normalized, there is performed the normalization and rounding of the response.

Since the mantissa of the operands are normalized, there are cases when $|m_1| > |m_2|$, $|m_1| < |m_2|$. In the first case, before starting the division, it is necessary to subtract the divisor from the dividend and write 1 to the integer part of the quotient mantissa. Then continue to divide the numbers in the usual way. After receiving a quotient, it is obvious that it will not be normalized. Therefore, it is necessary to normalize the quotient, i.e. in this case, move it 1 digit to the right, and add 1 to the quotient order.

It should be noted that implementing algorithms of mathematical operations in the floating point format, each time a procedure involving the characteristics of the operands or result is performed, overflow and order disappearance are controlled.

Let consider the example. $A=10_{10}=0.1010$, $p_1=p_a=4$, $B=2=0.1$, $p_2=p_b=2$, $|m_1| > |m_2|$.

At first subtraction of m_2 from m_1 I write to the integer part of the quotient:

$$\begin{array}{r} 0.1010 \\ + 1.1000 \\ \hline \end{array}$$

$0.0010C=1.XX$, further, we will divide by the division method without the remainder recovery

$$\begin{array}{r} + 1.1000 \\ \hline 1.1100 \end{array} \quad C = 1.0$$

$$\begin{array}{r} 1.1000 \\ + 0.1000 \\ \hline 0.0000 \end{array} \quad C = 1.01. \text{ Shift to the right } C = 0.101, \\ p = p + 1 = 3.$$

The result $C = 5_{10} = 0.101$.

Questions for self-control

1. What are the formats for representing numbers in computers?
2. Features of arithmetic operations with negative numbers?
3. Direct, reverse and additional code.
4. What is a mantissa number?
5. Why is the result correction conducted performing the operations in D-codes?
6. The algorithm for performing operations with numbers presented in floating point format.
7. What is the normalization of numbers?

Test questions

1. The number system is called ...
 - A) a way of representing numbers by digital signs
 - B) a way of representing numbers by numeric and alphabetic characters
 - C) a way of representing numbers by means of alphabetic characters
 - D) a collection of special characters to indicate numbers.
2. The binary number system is used in computer technology, because ...
 - A) binary code saves computer memory
 - B) they are the simplest in terms of circuit implementation
 - C) two state electronic components consume less power
 - D) they allow most to easily implement arithmetic operations.
3. About the number 11101,10101 we can say that it is given in ...
 - A) binary number system
 - B) octal number system
 - C) decimal notation
 - D) all answers are correct
4. How many bytes of RAM is needed to accommodate the word "mathematics-computer science"?

A) 10 B) 24 C) 21 D) 22

5. The smallest addressable unit of memory is
 A) byte B) bit C) word D) kilobyte
6. Convert the number $(31F5,156C)_{16}$ to the 8th number system:
 A) 30765, 05354 B) 30765, 05355 C) 3076.05354 D) 765, 05355
7. Calculate in the binary number system 11011×101 :
 A) 100010110 B) 11000010 C) 10000111 D) 11100010
8. Convert the number $(3F, DC)_{16}$ to binary:
 A) 111111, 1101111 B) 111111, 110111 C) 11111, 110111
 D) 111111, 110 110
9. Find the extra code for the number $(-0, 3A)_{16}$:
 A) F, C5 B) F, D6 C) 1, C6 D) 1, D6
10. Find the binary number by its reverse code 1,10111
 A) 1,10111 B) $-0,01001$ C) 0,001,000 D) $-0,11011$
11. Return codes are used to ...
 A) to multiply and divide numbers
 B) to replace the subtraction operation by addition
 C) analysis of the overflow of the bit grid
 D) for the division of numbers
12. $+0.000$ and -0.0000 zero has a single representation in ...
 A) additional code
 B) direct code
 C) reverse code
 D) answers A and C are correct
13. What is the modified additional code of the number -0.1110 equal to?
 A) 11.0010 B) 11.0001 C) 1.0010 D) 1.0001
14. What is the simple reverse code of the number 0.1110?
 A) 0.1110 B) 1.1110 C) 1.0001 D) 0.0001
15. What is the number equal to its additional code is 0.1110?
 A) 0.1110 B) 1.0010 C) 1.0001 D) 0.0001
16. What is the number equal to its additional code is -0.1110 ?
 A) 1.0010 B) 1.0001 C) 0.0010 D) 0.0001

CHAPTER 4. LOGICAL FOUNDATIONS OF A COMPUTER

4.1. The logical basis for the construction and operation of computers

All digital computers are created on elements that perform certain logical operations. Some elements provide processing of binary symbols representing information, others - switching channels through which information is transmitted, others - control, etc. The electrical signals acting on the inputs and outputs of these elements have, as a rule, two different levels, therefore, they can be represented by binary symbols, for example **1** and **0**. Let agree to designate the occurrence of an event, for example, a high voltage at a certain point in the circuit of a digital automaton, by a symbol **1**. This symbol is called *a log unit* (not the same as binary 1). The absence of an event is indicated by the symbol 0 and is called *a logical zero*.

For further perception of the material, let say a few words about the physical forms of information in computers. The most common methods of physical presentation of information are impulse and potential:

- impulse or its absence;
- high or low potential;
- high potential or its lack.

With an impulse display method, the unit code is identified by the presence of an electrical impulse, respectively, the zero code is identified by its absence. The impulse is characterized by amplitude and duration. The impulse duration should be less than the time cycle of the computer. The shape and amplitude of the signal are not taken into account. It follows from the foregoing that for the analysis and synthesis of circuits in digital automata, for example computers, the apparatus of the logic algebra can be used, which also operates with two concepts - *true* or *false*.

Therefore, each signal at the input or output of a binary element is associated with a *logical variable* that can take one of two values: the state of the logical unit (the event is true) and the state of the logical zero (the event is false).

The theory of the logical foundations of digital automata is extremely saturated with rather specific terms and concepts. Therefore, in this chapter and in all subsequent ones many definitions of these concepts will be given.

The function $f(x_1, x_2, \dots, x_n)$ is called *logical (switching)*, or Boolean, if it, like its arguments x_i , can take only two values: 0 or 1.

The logic algebra is a state algebra, not a number algebra, therefore this algebra is also called a propositional algebra.

A statement is a statement that you can definitely say about whether it is true or false. If the statement is true, then they say that its value of truth is equal to unity. If the statement is false, then its truth value is zero. Statements at the same time true and false does not happen.

Statements can be simple and complex. Simple separate statements - logical variables, they are usually denoted by the letters of the Latin alphabet. For example, if a simple statement x is true, then $x = 1$, if false, then $x = 0$.

Statements with different contents are indicated by different letters and are considered different. Two statements are called *equivalent* if their truths are the same. Equivalence of statements is indicated by an equal sign or identity. For example, the notation $x = y$ means that the statements x and y are either true or false at the same time.

Signs combining logical variables into complex statements, i.e. in logical functions, are signs of logical actions, more precisely *logical connectives*, and not mathematical actions

The set of values of the arguments of a logical function is called a *set (or point)* and can be denoted, in particular, as x_1, x_2, \dots, x_n where x_i is zero or one ($i = 1, 2, \dots, n$). Obviously, the set of argument values is actually a binary number. Each set of argument values is assigned by a number equal to a binary number that corresponds to the value of this set. For example, for four arguments 0, 0, 0, 0 - a zero set; 0, 0, 0, 1 - the first set; 0, 0, 1, 0 - the second set; 1, 0, 1, 0 - the tenth set, etc.

Thus, a logical function (a function of the logic algebra) is a function $y=f(x_1, x_2, \dots, x_n)$, which takes the value 0 or 1 on the set of logical variables x_1, x_2, \dots, x_n . Each logical function of a given set of arguments is also customarily assigned by a number: 0, 1, 2,

Any Boolean function can be set using a table in which all possible sets of values of binary variables are associated with the corresponding function values. Such a table is called a truth table, because it determines the truth or falsity of a complex statement depending on the truth or falsity of the component statements. For functions of one variable, there can be only four different Boolean functions F_1, F_2, F_3 and F_4 , presented in Table 4.1.

Table 4.1

The truth table for functions of one variable

x	F_1	F_2	F_3	F_4
0	0	0	1	1
1	0	1	0	1

It follows from the table that the functions F_1 and F_4 are independent of the argument and are respectively constants 0 and 1, and the function F_2 repeats the value of the argument, i.e. $F_2 = x$. The function F_3 is called the negation or inversion of the variable x .

No matter how complicated the logical connection between the logical function and its arguments is, this connection can always be represented in the form of simple logical operations.

These operations include:

- denial (operation "NOT");
- logical addition (operation "OR").
- logical multiplication (operation "AND").

Let consider these operations in more detail.

Negation is a logical connection between the input logical variable x and the output logical variable y , in which y is true only when x is false, and, conversely, y is false only when x is true. This functional dependence is presented in table 4.2. Similar tables representing the correspondence of all possible combinations of binary argument values to the values of a logical function are called truth tables.

Table 4.2

Operation "NOT"	
x	y
0	1
1	0

With the help of logical and mathematical symbolism, the logical function "NOT" of the variable y is written as $y = \bar{x}$ (read - "y is not x"). If, for example, x is a statement about the presence of a signal at a point in the CA circuit, then y corresponds to a statement - there is no signal.

The function $f(x)$, which takes a value opposite to the value of x , is a logical *negation* (*inversion*), or the function "NOT" can be denoted by one of the following methods:

$$f(x) = \bar{x} = \neg x$$

Let pay attention to the following functions:

The *logical addition* of several variables is a function that is false only when all the variables are false at the same time. The truth table of the logical addition operation is given below, see table. 4.3.

Table 4.3

Operation "OR"		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Logical addition is also called - *disjunction* and is denoted as follows:

$$y(x_1, x_2) = x_1 + x_2 = x_1 \vee x_2 = x_1 ! x_2$$

For example, the expression $y = x_1 \vee x_2$ reads as follows: "y is x_1 or x_2 ".

Logical multiplication of several variables is a function that is true only when all multiplied variables are true at the same time. The truth table of the logical multiplication operation is given below, see table. 4.4.

Table 4.4

Operation "AND"

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Logical multiplication is also called a *conjunction* and is denoted as follows:

$$y(x_1, x_2) = x_1 \cdot x_2 = x_1 \wedge x_2 = x_1 \& x_2.$$

For example, the expression $y = x_1 x_2$ reads as follows: « y is x_1 and x_2 ».

The "NOT-AND" function (*Schaeffer's NAND stroke*) is a function that is false when all variables are true. The symbol for this function is:

$$y(x_1, x_2) = x_1/x_2.$$

It reads as follows: "the function y is false, i.e. equals 0 when both arguments x_1 and x_2 are simultaneously true, i.e. equal to unity, and the function is true, i.e. equal to unity when either both arguments are false at the same time, or at least one of them is false".

The "NOT-OR" function (*Pierce arrow* or **NOR**) is a function that is true only when all variables are false. The symbol for this function is:

$$y(x_1, x_2) = x_1 \downarrow x_2 = x_1 \circ x_2.$$

It reads as follows: "the function y is false, i.e. is 0 when at least one of its arguments x_1 and x_2 is true, or both are true at the same time, i.e. equal to unity, and the function is true, i.e. equal to unity when both arguments are simultaneously false".

The "IF-THEN" *implication* is a function that is false if and only if x_1 is true and x_2 is false. The argument x_1 is called the *premise*, and x_2 is called the *consequence*. Its symbol

$$y(x_1, x_2) = x_1 \rightarrow x_2.$$

The exclusive "OR" function (**XOR**) is the function $y(x_1, x_2)$, which is denoted by a sign \vee . This operation implements *an ambiguity* function, i.e. in fact, **the summation on the 2 module** procedure is implemented, which is denoted by a sign \oplus :

$$y(x_1, x_2) = x_1 \vee x_2 = x_1 \oplus x_2.$$

From all the above definitions, it is clear that in the logic algebra all signs of actions: \wedge or $\&$, \vee or $+$, \rightarrow , ∇ , etc., unlike ordinary algebra, are signs of *logical connectives*, i.e. *logical actions*, not signs of arithmetic operations.

Table 4.5 shows examples of all elementary logical functions of two variables x_1 and x_2 .

Table 4.5

The list of elementary logical functions of two variables

# of the function		Value			Example
		x_1	x_2	y	
f_0	constant zero	0	0	0	$f0$
f_1	conjunction	0	0	0	$x_1 \wedge x_2$
f_2	forbiddance x_2	0	0	1	$x_1 \wedge x_2$
f_3	variable x_1	0	0	1	$x_1 \wedge x_2 \vee x_1 \wedge x_2 = x_1$
f_4	forbiddance x_1	0	1	0	$x_1 \wedge x_2$
f_5	variable x_2	0	1	0	$x_1 \wedge x_2 \vee x_1 \wedge x_2 = x_2$
f_6	addition on module 2	0	1	1	$x_1 \oplus x_2$
f_7	disjunction	0	1	1	$x_1 \vee x_2$
f_8	Pierce function	1	0	0	$x_1 \downarrow x_2$
f_9	equivalence	1	0	0	$x_1 \equiv x_2$
f_{10}	inversion x_2	1	0	1	$\bar{x}_1 \wedge x_2 \vee x_1 \wedge \bar{x}_2 = x_2$
f_{11}	implication	1	0	1	$x_2 \rightarrow x_1$
f_{12}	inversion x_1	1	1	0	$x_1 \wedge x_2 \vee x_1 \wedge x_2 = x_1$
f_{13}	implication	1	0	0	$x_1 \rightarrow x_2$
f_{14}	Schaeffer function	1	1	0	$x_1 \uparrow x_2$
f_{15}	constant unit	1	1	1	$f1$

Let introduce three more specific concepts of the logic algebra.

Two functions are considered equivalent to each other if they take the same values on all possible sets of their arguments.

The logical variable x_i is true if the value of the logical function $f(x_1, x_2, \dots, x_n)$ changes with x_i . Otherwise, this variable for the given function is fictitious, i.e. is not her argument.

The need to introduce these last two concepts arose for the following reason. Analyzing some unknown logical function (*logical circuit*), for which it is necessary to form an analytical expression, not all logical variables connected for this analysis can be arguments of this function, which is revealed as a result of the analysis. In the future it will be shown that any logical operations on logical variables can be reduced to a certain set of elementary logical functions, for example, such as "AND", "OR", "NOT-OR".

Elementary logical operations on computers are also performed on binary numbers, *bitwise*.

Elements that implement the simplest logical functions are schematically represented in the form of rectangles, on the field of which there is displayed a

symbol representing the function, performed by this element. The fig. 4.1 and the table 4.6 shows the designations of elements that implement the logical functions “AND”, “OR”, “NOT”. Input variables are usually depicted on the left, and weekend variables - on the left. It is believed that the information transfer occurs from left to right.

Table 4.6

Logical multiplication «AND»	Logical addition «OR»	Negation «NOT»
01011010 <u>11110000</u> 01010000	0101010 <u>1111000</u> 1111010	<u>01011010</u> 10100101

In the transition from logic functions to logic circuits, it is usually assumed that logical unit (1) corresponds to an impulse signal of standard amplitude, for example, of a high level, and logical zero. (0) - low level and usually fixed duration. Moreover, all input signals must arrive at each element at the same time.

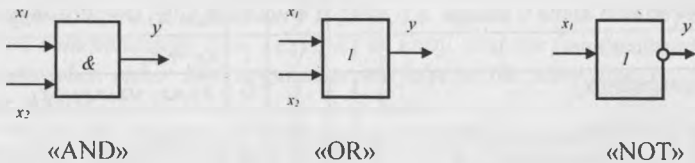


Fig. 4.1. Designations of elements that implement the logical functions “AND”, “OR”, “NOT”

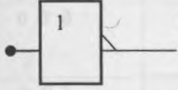
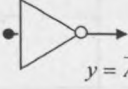
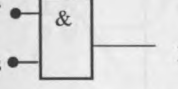
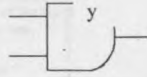
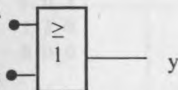
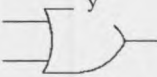
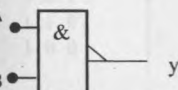
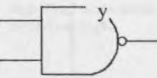
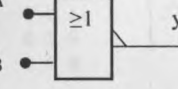
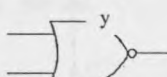
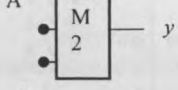
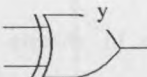


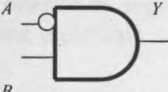

Any logical function can be implemented using the appropriate combination of elementary logical functions, any logical circuit can be formed using the corresponding combination of logical elements.

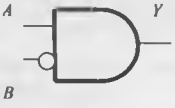


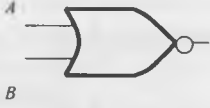
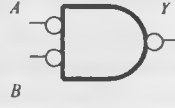



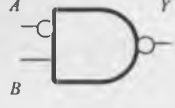
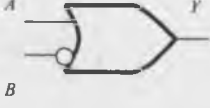


Note that each real logic element has some delay time for changing the output signal relative to the input. However, in combinational circuits, at their formally description, the delay time of logic elements is neglected.

As it was already noted, simple (elementary) logic functions are implemented in hardware using the corresponding combination of electronic logic elements - *gates*. We can assume that these elementary logical functions are logical operators of the mentioned electronic elements, i.e. circuits. Each such scheme is indicated by a certain graphic symbol, which can be oriented or non-oriented. We give examples of the graphic designation of these schemes, see table. 4.7.

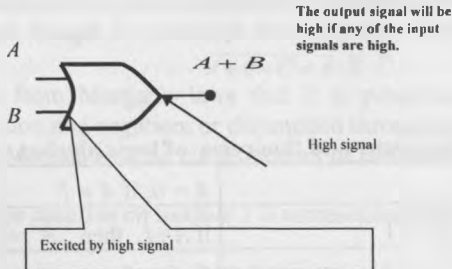
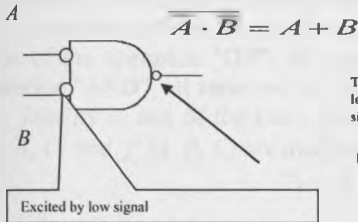
Table 4.7

Examples of graphic representation of logic circuit elements

91-1984 IEEE/ANSI	Traditional image of logic elements	Logical functions
		«NOT» (NOT Gate, or Inverter) $y = \bar{A}$
		«AND» (AND Gate) $y = A \cdot B = A \wedge B =$ $= A \& B$
		«OR» (OR Gate) $y = A + B = A \vee B$
		«AND-NOT» (NAND Gate) $y = \overline{A \cdot B} = \overline{A \wedge B}$
		«OR-NOT» (NOR Gate) $y = \overline{A + B} = \overline{A \vee B}$
		Addition on module 2 (XOR Gate) $y = A \oplus B =$ $= \bar{A}B + A\bar{B}$
«AND»	«OR»	A B Y
		1 1 1 1 0 0 0 1 0 0 0 0
		1 1 0 1 0 0 0 1 1 0 0 0

91-1984 IEEE/ANSI	Traditional image of logic elements	Logical functions
		<pre> 1 1 0 1 0 1 0 1 0 0 0 0 </pre>
		<pre> 1 1 0 1 0 0 0 1 0 0 0 1 </pre>
		<pre> 1 1 1 1 0 1 0 1 1 0 0 0 </pre>
		<pre> 1 1 1 1 0 0 0 1 1 0 0 1 </pre>
		<pre> 1 1 1 1 0 1 0 1 0 0 0 0 </pre>
		<pre> 1 1 0 1 0 1 0 1 1 0 0 1 </pre>

INTERPRETATION OF THE LOGICAL ELEMENTS DESIGNATION
Interpretation of two designation variants of the logical element "OR"



Analyzing or synthesizing logical circuits of computers, the following circumstance should be taken into account.

In order to carry out logical functions in practice, there are used various so-called logical (digital) semiconductor circuits - gates, the output signals of which are uniquely determined by combinations of signal levels at the inputs of these circuits. Moreover, both the input and output signals of these gates can be impulse or potential and have two fixed values: high (H) or low (L) level. When a logical "1" corresponds to a high level, then a logical "0" corresponds to a low level.

If the logical "1" corresponds to the presence of a signal (high or low level), then in this case it is said that the logic circuits operate *in positive logic*. If the logical "1" corresponds to the absence of a signal, then it is considered that the circuits operate *in negative logic*. There is also *mixed logic*, that is, when in the considered electronic node some gates operate in positive logic and others - in negative.

There are four basic laws in the logic algebra: *commutative* (commutativity properties); *combinational* (associativity properties); *distribution* (distributational properties), inversion (de Morgan rule).

Some laws of ordinary algebra can be applied to the logic algebra.

Commutative law:

for multiplication $(A \cdot B) = (B \cdot A)$;

for addition $(A + B) = (B + A)$.

Combinational law:

for multiplication $A \cdot (B \cdot C) = (A \cdot B) \cdot C$;

for addition $A + (B + C) = (A + B) + C$.

Distribution law:

$A \cdot (B + C) = A \cdot B + A \cdot C$;

$(A + B) \cdot C = A \cdot C + B \cdot C$.

The logic algebra has a number of specific axioms and theorems, the main of which are necessary for the analysis and synthesis of logical circuits, they are given in table 4.8.

Table 4.8

Specific axioms and theorems of logic algebra

1)	$A = 1, \text{ if } A \neq 0$	$A = 0, \text{ if } A \neq 1$
2)	if $A = 0$, then $\bar{A} = 1$	if $A = 1$, then $\bar{A} = 0$
3)	$0 + 0 = 0$	$0 \cdot 0 = 0$
4)	$0 + 1 = 1$	$1 \cdot 0 = 0$
5)	$1 + 1 = 1$	$1 \cdot 1 = 1$
6)	$\bar{0} = 1$	$\bar{1} = 0$
7)	$A \vee 0 = A$	$A \cdot 1 = A$;
8)	$A \vee 1 = 1$	$A \cdot 0 = 0$;
9)	$A \vee A = A$	$A \cdot A = A$;
10)	$A \vee \bar{A} = 1$	$\bar{A} \cdot A = 0$
11)	$\overline{A + B} = \bar{A} \cdot \bar{B}$	$\overline{A \cdot B} = \bar{A} + \bar{B}$
de Morgan rule		
12)	$\overline{A \wedge B} = \bar{A} \vee \bar{B}$ и $\overline{A \vee B} = \bar{A} \wedge \bar{B}$	
The rule of double negation		
13)	$\overline{(\bar{A})} = A$	

Bonding law		
14)	$A \cdot B \vee A \cdot \bar{B} = A$	$(A \vee B) \cdot (A \vee \bar{B}) = A$

Axioms and theorems written in the second column (i.e., left) are called *dual* to the axioms and theorems written in the third column (i.e., right).

Duality is defined as the change of all signs of the operation "AND" by the signs of the operation "OR", all signs of the operation "OR" by the signs of the operation "AND", all zeros per unit and all units to zeros.

Duality is one of the main properties of the logic algebra and means that if $f(A, B, C)$ and $f'(A, B, C)$ are dual functions, then

$$f(\overline{A, B, C}) = f'(\overline{A, B, C}).$$

The laws of de Morgan are one of the illustrations of the properties of duality and, as already noted, can be formulated as:

$$\overline{A \cdot B \cdot C} = \bar{A} + \bar{B} + \bar{C};$$

$$\overline{A + B + C} = \bar{A} \cdot \bar{B} \cdot \bar{C}.$$

It follows from Morgan's laws that it is possible to express conjunction through disjunction and negation, or disjunction through conjunction and negation. The laws of de Morgan and the consequences of them are valid for any number of variables.

The addition function on module 2 is represented as follows:

$$A \otimes B = \bar{A} \cdot B + A \cdot \bar{B}.$$

The following axioms are valid for this function:

$$A \otimes A = 0; \quad A \otimes A \otimes A = A; \quad A \otimes \bar{A} = 1; \quad A \otimes 1 = \bar{A}; \quad A \otimes 0 = A.$$

Based on the considered axioms and the properties of elementary logical functions, for example, it is possible to derive the rules for representing the functions "AND", "OR", "NOT" through the addition function on module 2 and vice versa:

$$A + B = A \oplus B \oplus AB;$$

$$A \cdot B = (A \oplus B) \oplus (A + B).$$

The functions "AND", "OR", "NOT" through the Scheffer function are expressed as follows:

$$\overline{A \cdot B} = A | B;$$

$$A + B = \overline{A \cdot B} = \overline{A} | \overline{B}.$$

The Pierce function can be described by the following expressions:

$$A \downarrow B = \overline{A + B} = \bar{A} \cdot \bar{B}.$$

The following axioms are valid for this function:

$$A \downarrow A = \bar{A}; \quad A \downarrow 0 = \bar{A}; \quad A \downarrow \bar{A} = 0; \quad A \downarrow 1 = 0.$$

The functions "AND", "OR", "NOT" are expressed through the Pierce function as follows:

$$A \cdot B = (A \downarrow A) \downarrow (B \downarrow B); \quad A + B = (A \downarrow B) \downarrow (A \downarrow B); \quad \bar{A} = A \downarrow A.$$

It should be noted that logical expressions containing disjunction and conjunction operations can be transformed (open brackets, remove the common factor, rearrange terms, etc.) according to the rules of algebra, formally considering a disjunction as an addition operation, and a conjunction as a multiplication operation. It should be remembered that in the logic algebra, in contrast to ordinary algebra, the + sign or the \vee sign means the logical connective "OR", and the multiplication sign " \cdot " or the signs \wedge , and $\&$, mean the logical connective "AND" [9, fourteen]. A *Boolean expression* is a formula consisting of logical constants and logical variables connected by signs of logical operations.

Since each logical function is implemented using a specific set of devices, therefore, the fewer elements an expression contains, the simpler the circuit that implements its corresponding function. Thus, the study of methods for minimizing logical functions is of considerable interest.

There are distinguished analytical and tabular methods of minimization.

In principle, any logical function can be simplified directly using axioms and theorems of logic, but such transformations require certain calculations and checking for errors. Therefore, it is more advisable to use special algorithmic minimization methods that allow simplifying the function more simply, quickly and accurately. Such methods include, for example, *the Quine method, the Carnot method of maps*, etc. These methods are most suitable for ordinary technical practice [9, 14].

The analytical method of logical functions minimizing consists in sequentially applying the laws and rules of the logic algebra to a certain formula. However, this method does not lend itself to a clear algorithmization. The actions used in the implementation of this method are determined by the type of the initial transformed expression, as well as by the skills of the performer. The lack of process algorithmization of the minimization significantly increases the probability of errors and the possibility of obtaining an incompletely minimized logical function.

The direct conversion method is most suitable for simple formulas when the sequence of transformations is obvious.

Let consider an example of applying the direct transformation method to the example of minimizing the function of three variables given by its *PDFNF*:

$$y = f(A, B, C) = (\bar{A} \cdot \bar{B} \cdot C) \vee (A \cdot \bar{B} \cdot \bar{C}) \vee (A \cdot \bar{B} \cdot C) \vee (A \cdot B \cdot \bar{C}) \vee (A \cdot B \cdot C).$$

Let combine in pairs the first and third, second and third, as well as the fourth and fifth elementary products:

$$y = (\bar{A} \cdot \bar{B} \cdot C \vee A \cdot \bar{B} \cdot C) \vee (A \cdot \bar{B} \cdot \bar{C} \vee A \cdot \bar{B} \cdot C) \vee (A \cdot B \cdot \bar{C} \vee A \cdot B \cdot C).$$

It should be noted that one and the same elementary product can be used several times after being combined. In this example, the elementary product is used twice. After putting out the brackets we get:

$$y = \bar{B} \cdot C \cdot (A \vee \bar{A}) \vee A \cdot \bar{B} \cdot (\bar{C} \vee C) \vee A \cdot B \cdot (C \vee \bar{C}) = \bar{B} \cdot C \vee A \cdot \bar{B} \vee A \cdot B.$$

The resulting logical function is simpler than the original PDNF, however, it is not minimal. Combining the second and third elementary works, after putting out of the bracket A we finally get:

$$y = \bar{B} \cdot C \vee A \cdot (\bar{B} \vee B) = \bar{B} \cdot C \vee A.$$

It is easy to see that in each pair the combined elementary products differ in only one variable, which is included in the first work with negation, and in the second - without negation. Such elementary products are called neighboring. Operation (i.e.) is applicable to neighboring products, as a result of which the number of summarized products is reduced and the number of variables is reduced by one.

Let consider another example. You must select the smallest value from the three variables A, B, C .

The first version of the analysis gives the following results:
 if $A < B$ and $A < C$ then least = A ,
 if $A > B$ and $B < C$ then the least = B ,
 if $A > C$ and $B > C$ then the least = C .

The analysis yielded a suboptimal result. Reduce it by outlining some comparisons:

if $A < B$ then
 if $A < C$ then least = A ,
 otherwise smallest = C .
 otherwise
 if $B < C$ then least = B ,
 otherwise smallest = C .

In the optimized version of the example, there are only three comparisons instead of six. Moreover, if in the first example, six comparisons are always performed to achieve the result, then in the second example, only two comparisons are always performed.

The desire for algorithmization of the search for elementary products has led to the development of tabular methods to minimize logical functions. One of them is a method based on the use of Carnot cards.

Carnot maps (a variation of them are *Veitch maps*, which are constructed as sweeps of a cube on a plane), are a graphical *representation of truth tables*.

Therefore, they are created either according to the truth table of the analyzed function, or according to its *PDNF*.

There should be noted that each row of the truth table, for which the function is equal to one, corresponds to the *minterm* of the function presented in the *PDNF*. The row for which the function is equal to zero is the *maksterm* of the function represented in *PDNF*.

The Carnot map is a rectangle divided into squares, the number of which is equal to the total number of sets of n variables for a given function, i.e. it is equal to 2^n . So, for a function of four variable squares there will be 16, for five variables - 32, etc. Each square corresponds to a specific set or term, and the sets are arranged so that *adjacent sets or terms, both horizontally and vertically, differ only in the value of one variable: in one square it is with inversion, and in the other, neighboring one - without.*

The function in the *PDNF* is plotted on the map, marking, for example, with a sign 1 squares corresponding to those sets on which the function is equal to one, i.e. in *PDNF* functions there is a corresponding *minterm*. The remaining squares are marked with 0. Sometimes a set number is placed in the corner of the square. This method is used if the function is specified numerically, but it is inconvenient for the minimization procedure.

Let list the sequence of actions performed to minimize functions by the Carnot method:

- 1) the initial function which should be minimized should be presented in the *DNF*. Then it must be submitted to the *PDNF*. Or, the truth table of the minimized function is compiled. As already noted, there is a one-to-one correspondence between the rows of the truth table and the cells on the Carnot map. When the Carnot map is compiled using the *PDNF* of the minimized function, it is obvious that each variable without negation is replaced by its value 1, and with negation - 0;

- 2) then the Carnot map is constructed according to the principle described above. Imagine a coordinate system in which, for example, for a function of two variables puts the values of one argument on the horizontal axis, and the other on the vertical. At the intersection of the corresponding coordinates, we get a cell where the value of the function (0 or 1) corresponding to this set is written. If the function is presented in *PDNF*, then 1 is written in the cell corresponding to the existing *minterm*, and 0 is written in the cell of the non-existent *minterm*;

- 3) after this, adjacent cells in which the units are written, are grouped as follows: the even number of neighboring cells with units is combined necessarily, both vertically and horizontally. Moreover, each cell with 1 can fall simultaneously into two groups obtained as a result of combining units, both vertically and horizontally;

- 4) each group is assigned by a new *minterm* for the image of the original function in the form of minimal *DNF*;

- 5) the image of each new *minterm* is formed according to the following algorithm:

- a) the variable, which in each cell of the formed group has a value of only 0, is depicted by its inversion;
- b) the variable, which in each cell of the group has a value of only 1, is depicted without inversion;
- c) the variable that changes its value within an educated group is not depicted, i.e. discarded.

Therefore, the Carnot map can be considered as a graphical representation of the totality of all (existing and non-existing) *minterm* functions in the PDNF of a given number of logical variables.

Let return to tables 6.8 and 6.9. The Carnot map for the function of three variables has $2^n=2^3=8$ cells, and for four variables $2^4=16$ cells highlighted in the table.

The map is marked with a coordinate system corresponding to the value of the input variables. For example, the top line of the map for the function of three variables corresponds to the zero value of the variable A , and the bottom - to its unit value. Each column of this map is characterized by the values of two variables: B and C .

The combination of numbers with which each column is marked shows for which values of the variables B and C the function located in the cells of this column is calculated. So for the case of a Carnot map of four variables function, the function located in the cells of the column with coordinates 01 is calculated for the values of the variables $C = 0$ and $D = 1$. The function located in the cell at the intersection of this column and the row with coordinates 11 is determined by a set of input variables $A = 1, B = 1, C = 0, D = 1$.

If the function is equal to unity on the indicated set of variables, then its PDNF necessarily contains an elementary product that takes a unit value on this set. Thus, the cells of the Carnot map representing the function contain as many units as there are elementary products in its PDNF, and each unit corresponds to one of the elementary products.

The coordinates of the rows and columns in the Carnot map do not follow in the natural order of binary codes increasing, but in the order 00, 01, 11, 10. Changing the order of the sets is done so that the neighboring sets are neighboring in the geometric sense.

Let consider the minimization process using the example of a function defined by the following logical equation:

$$f(A,B,C,D) = B \cdot C \cdot D + \bar{A} \cdot B \cdot D + \bar{B} \cdot C \cdot D + A \cdot \bar{B} \cdot \bar{C} + A \cdot \bar{C} \cdot D + \bar{B} \cdot \bar{C} \cdot D + \bar{A} \cdot B \cdot C + \bar{A} \cdot \bar{B} \cdot \bar{C}.$$

Let present this function in PDNF:

$$\begin{aligned}
 f &= B \cdot C \cdot D \cdot (A + \bar{A}) + \bar{A} \cdot B \cdot D \cdot (C + \bar{C}) + B \cdot C \cdot D \cdot (A + \bar{A}) + \\
 &+ A \cdot \bar{B} \cdot \bar{C} \cdot (D + \bar{D}) + A \cdot \bar{C} \cdot D \cdot (B + \bar{B}) + \\
 &+ \bar{B} \cdot \bar{C} \cdot D \cdot (A + \bar{A}) + \bar{A} \cdot B \cdot C \cdot (D + \bar{D}) + \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot (D + \bar{D}) = \\
 &A \cdot B \cdot C \cdot D + \bar{A} \cdot B \cdot C \cdot D + \bar{A} \cdot B \cdot \bar{C} \cdot D + \\
 &+ A \cdot \bar{B} \cdot C \cdot D + \bar{A} \cdot \bar{B} \cdot C \cdot D + \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D + A \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} + A \cdot B \cdot \bar{C} \cdot D + \\
 &\bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} + \bar{A} \cdot B \cdot C \cdot \bar{D} + \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D.
 \end{aligned}$$

The Carnot map corresponding to the function under consideration is shown below, see Fig. 4.2.

Minterm functions form four groups in a map - $\bar{A} \cdot \bar{B} \cdot \bar{C}$, $A \cdot \bar{B} \cdot \bar{C}$, D , $(\bar{A} \cdot B \cdot C)$.

	$\bar{A} \cdot \bar{B} = 00$	$\bar{A} \cdot B = 01$	$A \cdot B = 11$	$A \cdot \bar{B} = 10$	
$\bar{C} \cdot \bar{D} = 00$	1	0	0	1	$(\bar{A} \cdot \bar{B} \cdot \bar{C})$
$\bar{C} \cdot D = 01$	1	1	1	1	
$C \cdot D = 11$	1	1	1	1	(D)
$C \cdot \bar{D} = 10$	0	1	0	0	
		$(\bar{A} \cdot \bar{B} \cdot C)$	$(\bar{A} \cdot B \cdot C)$		

Fig. 4.2. The Carnot map

Therefore, we obtain

$$f = \bar{A} \cdot \bar{B} \cdot \bar{C} + A \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot B \cdot C + D = \bar{B} \cdot \bar{C} \cdot (A + \bar{A}) + \bar{A} \cdot B \cdot C + D = \bar{B} \cdot \bar{C} + \bar{A} \cdot B \cdot C + D$$

But we must keep in mind that in the general case, a function can have several minimal forms.

4.2. Electronic technology of computer logic elements

Electronic technologies and elements on the basis of which computers were created have changed many times. The first generation of computers was based on electron tubes, the second on discrete semiconductor devices (diodes and triodes — transistors), and the next generations on integrated semiconductor circuits.

The electronic semiconductor elements were changed according to the type of used elements, the type of connections between transistors. In particular, the following element systems were used:

- resistor - diode;
- resistor - transistor;
- ferrite transistor;

- diode-transistor;
- transistor-transistor.

The most widely used in modern integrated circuits transistor-transistor systems of elements (TTL - transistor-transistor logic), in which transistors with fixed voltages on their electrodes play the role of resistors and diodes. This system ensures complete uniformity of the microcircuit structure — they contain only transistors, which facilitates the technology of their manufacture [8, 12, 14, 15].

The architecture used in computer transistors has also changed:

Second-generation machines used bipolar germanium and silicon *pnp* and *npn* transistors;

In integrated circuits, unipolar field-effect MOS were used (MOS-metal-oxide-semiconductor, or MOS).

Field effect transistors have three electrodes, see fig. 4.3.

- shutter (analog of the bipolar transistors base);
- source (analog of the emitter);
- stock (collector analog).

The shutter is electrically isolated from other electrodes by a silicon oxide film and controls the current flow between the source and drain not by diffusion of electrons (as in *npn* transistors) or holes (as in *pnp* transistors), but by the electrostatic field created by it. Therefore, MOS transistors are called field-effect transistors.

Unipolar transistors have a higher speed than bipolar transistors, since the mechanism of their operation is not associated with slow diffusion processes. The elements of the transistor are placed on a flat silicon substrate, see Fig. 4.3.

The architecture of logical element systems has also changed. Field effect transistors have several varieties:

- *n*MOS;
- *p*MOS;
- MOS with additional symmetry (CMOS - transistors - complementary metal-oxide-semiconductor structure, CMOS).

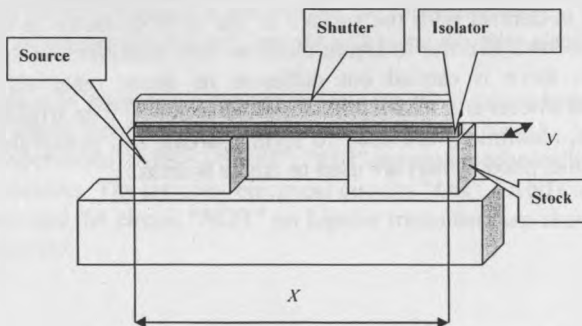


Fig. 4.3. Field-effect transistor structure

Initially, field-effect transistors were called MDC transistors (metal – dielectric – conductor), but since silicon oxide was used as the dielectric, they were renamed as MOS transistors. But, probably, in the near future it will be necessary to return to their original name, because another more effective dielectric, having a lower dielectric constant than oxide, and thereby creating lower values of stray capacitances between the electrodes, begins to be used as an insulator [14].

The n MOS and p MOS transistors are called series-connected with respect to the power source, and parallel-connected with respect to the output signal. Since the shutter of n MOS or p MOS transistors are connected in parallel, always one of these transistors turns on and the other turns off, and the power consumption and output resistance of the CMOS circuit will be small (a small current will flow only in transient transistor modes). The shutter of the transistor is electrically isolated from the source and drain, the control is carried out by an electrostatic field, so the input resistance of the field-effect transistors is very large.

This circumstance creates the convenience of connecting CMOS circuits to each other and ensures the stability of their work. CMOS circuits have lower power consumption than bipolar transistors and other types of field-effect transistors, can be more densely packed; integrated circuits created on their basis can be used on a more miniature scale of microtechnologies.

Nowadays, CMOS transistors are also used in systems of random access memory and flash memory. In memory modules, a capacitor is used to store one bit of information. The value of the charge of this capacity determines the stored bit: the presence of charge is “0”, the absence of charge is “1”.

In order to ensure non-volatility in the CMOS flash transistors, another so-called *floating shutter* is placed under the shutter, see Fig. 4.4. The floating shutter has metallization (a film of gallium arsenide, chromium, nickel, tungsten, etc.) to create a potential Schottky barrier at the interface between the metal and the semiconductor, which allows storing the capacitor charge for a long time.

Integrated circuits with MOS transistors are manufactured using planar technology. A protective layer of a dielectric (usually silicon dioxide) is applied to the surface of a wafer made of semiconductor (silicon), in which micro-windows are opened by photolithography methods. A metal film is deposited on top of the dielectric layer in contact with the surface of the semiconductor in the windows. Through the windows in order to create electron-hole transitions of the desired (n - or p -) polarity, there is carried out diffusion of donor materials or electron acceptors. Since silicon is a tetravalent chemical element, then trivalent materials (boron, gallium, aluminum) are used to form p -areas, and pentavalent materials (antimony, arsenic, phosphorus) are used to create n -areas.

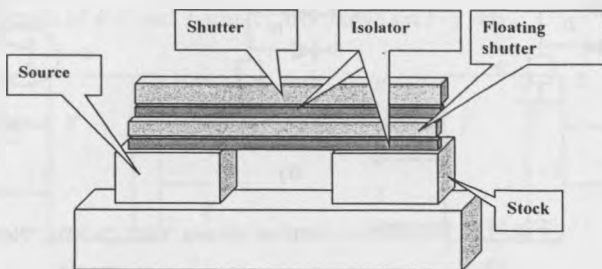


Fig. 4.4. Flash memory element structure

A promising is the technology developed at the University of Buffalo using "self-organizing" chemicals - materials with microscopic structures ("quantum dots") in the manufacture of semiconductor devices [14, 20]. According to the researchers [14, 16–20], in these substances, even at room temperature, a reaction occurs spontaneously, leading to the creation of regular microscopic structures with cells with a diameter of $0.04 \mu\text{m}$.

The parameters of transistors depend on the scale of the manufacturing process (scale of technology), which is constantly being improved. Now there are used technologies of $0.09\text{-}0.045$ microns.

Reducing the size of transistors increases the density of their placement, reduces stray inductance and capacitance of the electrodes, and allows to increase the operating frequency of the chip. But at the same time, the miniaturization of transistors (in some cases the thickness of the insulating layers in the transistor is comparable with the size of atoms) leads to an increase in spurious currents leakage, which, in turn, increases energy consumption and reduces the stability of the circuit. Reducing the supply voltage of the circuit reduces the heating of the circuits only partially, and the power of currents leakage can reach hundreds of watts.

Reducing currents leakage is achieved in the following ways:

- the use of copper conductors (instead of having a higher electrical resistivity of aluminum);
- application of stressed silicon technology (as the distance between the atoms of the crystal lattice increases, the electrical resistivity decreases).

Logical operations "AND", "NOT", "OR" are quite technically performed on any element systems. The simplest electrical circuits "OR", "AND" on the resistor-diode elements and the circuit "NOT" on bipolar transistors are shown on Fig. 4.5, a, b, c, respectively.

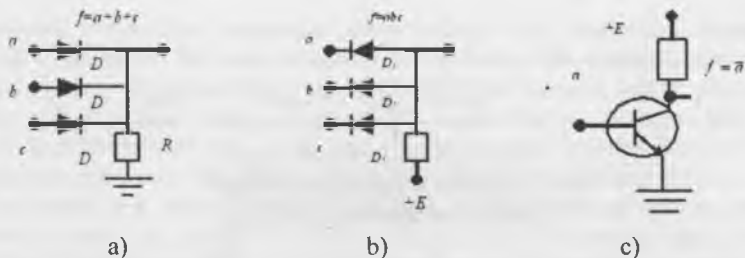


Fig. 4.5. The simplest electrical circuits "OR", "AND", "NOT"

Explanations for the "OR" circuit: a positive impulse at the output occurs when a positive impulse appears at any (a , b , c) input, since the internal resistance of the diode in the forward direction is small (much less than R).

Explanations for the "AND" circuit: a positive impulse at the output occurs only when there are positive impulses at all three (a , b , c) outputs. In the absence of at least one input impulse, the corresponding diode will be open and close the voltage supply $+E$ through the internal resistances of the diode and the input signal source (they are much less than R) to ground.

Explanations for the "NOT" circuit: when a positive impulse is applied to the input (base) of the npn transistor, the triode will open and the output voltage (collector) will drop from high to almost zero.

The implementation of "OR", "AND", "NOT" based on unique operators is used in the logical synthesis of computational schemes, because for the basic operators, there are developed the procedures of formalized logical synthesis in the most detailed way constructively.

Among the many elementary circuits in the computer, the most widely used trigger circuit is a static memory and logical element.

On triggers there are created systems of static memory, registers, counters, frequency dividers and many other computer circuits.

A trigger is an element that can be in one of two stable states, conditionally referred to as the states "0" and "1". The trigger has two outputs:

- output "0" (sometimes referred to as \bar{q} -output);
- output "1" (sometimes referred to as q -output).

If the trigger is in the "0" state, then its output has a "high" voltage (of the order of several volts or less), the output has a "low" (zero) voltage, if the trigger is in the "1" state, then the voltages are distributed vice versa.

Triggers can have separate inputs:

- R (*Reset*) - input of the "0" setting;
- S (*Set*) - input of setting "1".

Each input sets the trigger to the corresponding state, such triggers are called R - S - triggers. Triggers can have a counting input T (relaxer), the next impulse "1" input will change the state of the trigger. Such triggers are called T -triggers.

A trigger set to a state saves it until an impulse, applied to one of the inputs, changes this state.

Logic circuits of R - S and T -trigger are shown on Fig. 4.6.

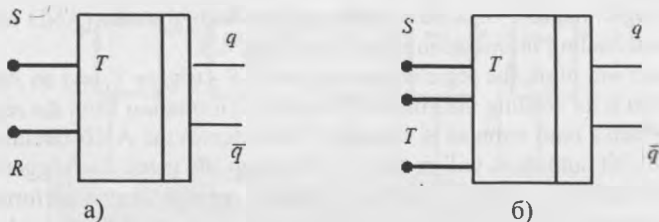


Fig. 4.6. Logic circuits R - S (a) and T -trigger (b)

The basic electric CMOS circuit of the R - S -trigger, made by transistor-transistor technology, is shown on Fig. 4.7. In order to set the trigger to the state "1", it is necessary to apply an impulse or high voltage to the point (S, q).

This signal will go to the shutters of transistors T_2 and T_4 . A transistor T_2 with a channel of n type opens, and a transistor T_4 with a channel of p type closes. The voltage at the drain of the transistor T_2 (point) will become low. Low voltage will pass to the shutter of transistors T_1 and T_3 . The transistor T_1 with the channel n type closes, and the transistor T_3 with the channel p type opens. A high voltage will appear on the drain of the transistor T_3 , which will pass to the shutters of transistors T_1 and T_4 and will maintain the trigger state "1" until an impulse or high voltage arrives at the input (R, \bar{q}).

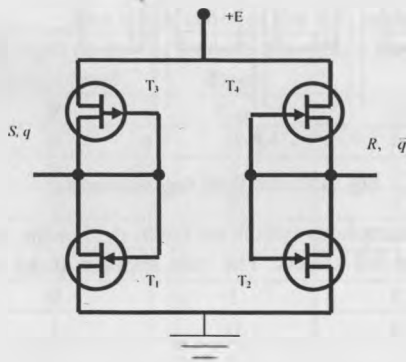


Fig. 4.7. Electric CMOS circuit of R - S -trigger made by transistor-transistor technology

Information from the trigger is read from its outputs by interrogating them through **AND** circuits. If the output, which is connected to one input of the **AND**

circuit, has a high voltage, then the polling signal will pass and will carry information about the state of the trigger.

Triggers are used at organizing memory registers and counters. At the same time, triggers with separate inputs are usually used in registers, and in counters – counter inputs.

The logic diagram of a three-digit register with shutters (AND circuits) for entering and reading information is shown on Fig. 4.8.

In each i -th digit, the register contains an R - S trigger T_i and an AND circuit connected to it for reading information. Reading information from the register is as follows. When a read impulse is applied, it interrogates the AND circuits of all the triggers, the bit outputs a_i will receive “1” through the gates, the triggers of which were in the state “1”. Writing information to the register can be performed in two modes - single and push-pull. In single-cycle mode, “1” is applied to the corresponding input of each trigger. In push-pull mode, all the inputs of the R triggers are connected to a single installation wire “0”, through which all the triggers are reset to zero, and then the corresponding impulse is applied to the inputs S of those triggers that need to be set to “1”.

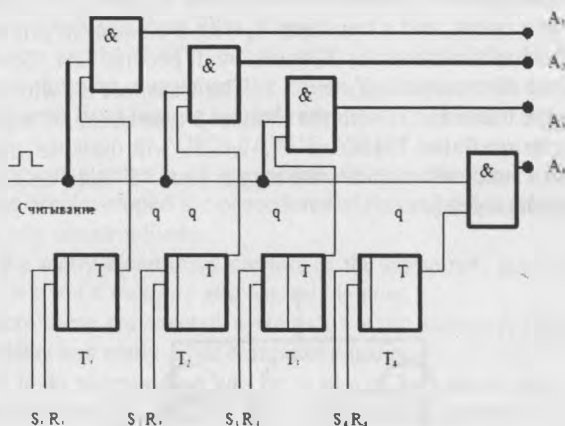


Fig. 4.8. Logical register circuit

Let consider an example in which we try to design the logic circuit of a five-bit adder using separate full adders. The sum appears at the outputs S_4, S_3, S_2, S_1, S_0 .

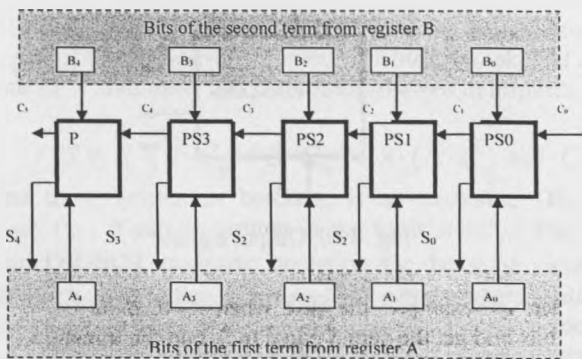


Fig. 4.9. Block diagram of a parallel five-bit adder using separate full adders

The device is called a parallel adder, because all the bits of the added numbers are fed to the inputs of the circuit **at the same time**. This means that in each category the addition is performed in one cycle. This differs from the usual addition on paper, when we add the bits (numbers) in turn, i.e. parallel addition is faster.

Below there is a truth table of a device having three inputs A, B, C_{in} , as well as two outputs: S и C_{out} . In total, eight sets of states are possible for three inputs, and for each particular case the truth table and the Fig. 4.10 shows the possible output signals.

Table 4.9

The truth table of the full adder

Bit input of the first term	Bit input of the second term	Transfer bit input	Sum bit output	Transfer bit output
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

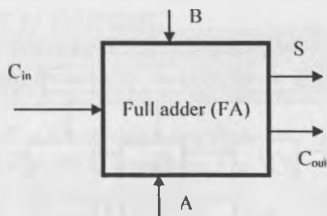


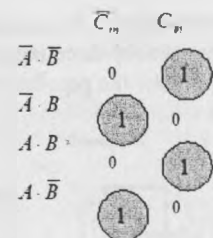
Fig. 4.10. Output signals

Let consider, for example, the case when $A=0, B=0, C_{in}=1$. The full adder must add these bits and get the sum S equal to 0, and the transfer C_{out} equal to 1.

The circuit has two outputs, so you can design the logic for each output individually. We start with exit S . The truth table shows that four situations are possible when $S = 1$. You can write an expression for S using the disjunctive form:

$$S = \bar{A} \cdot \bar{B} \cdot C_{in} + \bar{A} \cdot B \cdot \bar{C}_{in} + A \cdot \bar{B} \cdot \bar{C}_{in} + A \cdot B \cdot C_{in}.$$

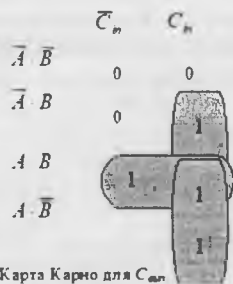
The expression for the signals at the S and C_{out} outputs can be minimized using Carnot cards, see Fig. 4.11.



Карта Карно для S

$$S = \bar{A} \bar{B} C_{in} + \bar{A} B \bar{C}_{in} + A \bar{B} \bar{C}_{in} + A B C_{in}$$

a)



Карта Карно для C_{out}

$$C_{out} = B C_{in} + A \cdot C_{in} + A \cdot B$$

б)

Fig. 4.11. Carnot cards for full adder output signals

The fig. 4.11 a) the Carnot card is shown for the exit S . This card does not have adjacent unit cells, therefore it is impossible to group them to get quartets or even pairs, i.e. the expression for output S cannot be minimized. The Carnot map for exit C_{out} is shown on Figure 4.11 b).

Similarly, this problem can be solved analytically. The circuit has two outputs, so you can design the logic for each output individually. We start with exit S . The truth table shows that four situations are possible when $S = 1$. You can write an expression for S using the disjunctive form:

$$S = \bar{A} \cdot \bar{B} \cdot C_{in} + \bar{A} \cdot B \cdot \bar{C}_{in} + A \cdot \bar{B} \cdot \bar{C}_{in} + A \cdot B \cdot C_{in}.$$

Let try to simplify this expression, i.e. take out the common multipliers. Since not a single pair of terms has two common variables, we take out the brackets the first two terms of \bar{A} multiplier and A from the last two multipliers. As a result, we get:

$$S = \bar{A} \cdot (\bar{B} \cdot C_{in} + B \cdot \bar{C}_{in}) + A \cdot (\bar{B} \cdot \bar{C}_{in} + B \cdot C_{in}).$$

The first term, written in brackets, is an exclusive OR expression for variables B and C_{in} . It can be written in the form $B \otimes C_{in}$. The second term in brackets is an OR-NOT exclusive operation for the same variables. It can be written as $B \otimes \bar{C}_{in}$. Then the expression for S will take the following form:

$$S = \bar{A} \cdot (B \otimes C_{in}) + A \cdot (\overline{B \otimes C_{in}}).$$

Replace $B \otimes C_{in}$ on X . Then $S = \bar{A} \cdot X + A \cdot \bar{X} = A \otimes X$, and it will be just an exclusive OR operation. Let carry out the reverse replacement $S = A \otimes [B \otimes C_{in}]$. Now consider the output signals C_{out} , nplisted in the truth table. Write the expression for the signal C_{out} in disjunctive form:

$$C_{out} = \bar{A} \cdot B \cdot C_{in} + A \cdot \bar{B} \cdot C_{in} + A \cdot B \cdot \bar{C}_{in} + A \cdot B \cdot C_{in}.$$

Simplify this expression by transferring the multipliers out of the brackets

$$C_{out} = B \cdot C_{in} \cdot (\bar{A} + A) + A \cdot C_{in} \cdot (\bar{B} + B) + A \cdot B \cdot (\bar{C}_{in} + C_{in}) = B \cdot C_{in} + A \cdot C_{in} + A \cdot B.$$

Further minimization of the resulting expression is impossible.

The circuit obtained as a result will look as follows, see Fig. 4.12.

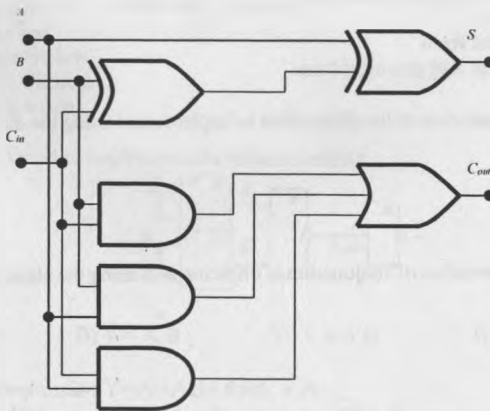


Fig. 4.12. Logical circuit of the full adder

Questions for self-control

1. Draw the main logical elements of the computer?
2. Why is it necessary to know the logic algebra?
3. Give the concept of the basis of a logical circuit.
4. What technologies are used to create logical elements of computers?
5. What are the basic laws of the logic algebra?
6. What methods of minimizing logical functions do you know?
7. Basic logical operations.
8. Truth tables of logical operations.
9. Logical elements.
10. Elementary functions of the logic algebra.
11. Axioms of the logic algebra.
12. The basic laws of the logic algebra.
13. The concept of the functional completeness of the system of elements.
14. Ways to set logical functions.

Test questions

1. The mathematical apparatus used in the design of computing devices is called ...
A) logic algebra
B) binary algebra
C) disjunction
D) propositional algebra
2. Which of the following operations do not apply to Boolean algebra?
A) conversion
B) addition on module 2
C) disjunction
D) conjunction
3. Logic elements can be set as...
A) truth tables and analytical form
B) truth tables
C) in analytical form
D) in analytical and graphical form
4. The logical operation of the disjunction is implemented using the element ...
A) OR
B) AND
C) NOT
D) Schaeffer
5. The logical operation of conjunction is implemented using the element ...
A) OR
B) and
C) NOT
D) Pierce
6. The Schaeffer element implements a logical function ...
A) AND-NOT
B) OR-NOT

- C) AND-OR-NOT
- D) AND

7. Which of the following elements form a functionally complete system of elements?
- A) Scheffer element, Pierce arrow
 - B) OR element
 - C) pierce arrow
 - D) Schaeffer element

8. Logical variables, combined by the signs of logical operations, make up ...
- A) logical expressions
 - B) logical functions
 - C) the perfect normal disjunctive form
 - D) logical statement

9. Distribution law:
- A) $X1 + X2 \cdot X3 = (X1 + X2) \cdot (X1 + X3)$
 - B) $X1 + (X2 + X3) = (X1 + X2) + X$
 - C) $X1 + X1 \cdot X2 = X1$
 - D) $X1 \cdot (X1 + X2) = X1$

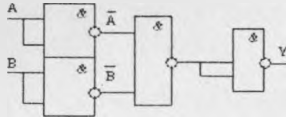
10. Determine the value of the logical function $Y = A + A B + A B C$.
- A) 1 B) 0 C) A D) B

11. Determine the value that the function $Y = A + A (B + C)$ takes for given values of the variables $A = 0$ and $B = 1$.
- A) 1 B) 0 C) A D) B

12. The formula $A B = A + B$ defines the rule ...
- A) de Morgan
 - B) absorption
 - C) denial
 - D) repetition

13. The equality $A (B + C) = A B + A C$ is called ...
- A) distribution law
 - B) by commutative law
 - C) de Morgan's theorem
 - D) the absorption rule

14. What logical function implements the scheme below?



- A) $Y = A + B$
- B) $Y = A B$
- C) $Y = \bar{A} B$
- D) $Y = A \bar{B}$

15. Simplify the expression $Y = A + AB + ABC \cdot A$
- A) 0
 - B) 1
 - C) $A + B$
 - D) $A B$

16. Equality $(\text{NOT } A) \text{ AND } B = 1$ (here AND is the logical AND, NOT is the negation) is satisfied with the values ...

- A) $A = 0, B = 1$
- B) $A = 1, B = 1$
- C) $A = 0, B = 0$
- D) $A = 1, B = 0$

17. Equality $(\text{NOT } B) \text{ AND } A = 1$ (here AND is the logical AND, NOT is the negation) is satisfied with the values ...

- A) $A = 0, B = 0$
- B) $A = 1, B = 0$
- C) $A = 0, B = 1$
- D) $A = 1, B = 1$

18. Equality $(\text{NOT } A) \text{ XOR } B = C$ (here XOR is the exclusive OR, NOT is the negation) is satisfied with the values ...

- A) $A = 0, B = 1, C = 0$
- B) $A = 1, B = 0, C = 1$
- C) $A = 0, B = 0, C = 0$
- D) $A = 1, B = 1, C = 0$

19. Equality $(A \text{ OR } B) \text{ AND } B = C$ (here OR is a logical OR, AND is a logical AND) if the values are ...

- A) $A = 1, B = 1, C = 0$
- B) $A = 0, B = 0, C = 1$
- C) $A = 0, B = 1, C = 1$
- D) $A = 1, B = 0, C = 1$

20. Equality $(A \text{ AND } C) \text{ AND } (B \text{ OR } C) = 1$ (here OR is a logical OR, AND is a logical AND) is satisfied with the values ...

- A) $A = 0, B = 0, C = 1$
- B) $A = 1, B = 1, C = 0$
- C) $A = 1, B = 0, C = 1$
- D) $A = 0, B = 0, C = 1$

CHAPTER 5. MICROPROCESSOR SOFTWARE MODEL

The software model of the processor in the IA-32 architecture (Intel Architecture) of Intel processors is the following set of resources (see Fig. 5.1):

- addressable memory space up to 2^{32} bytes (4 GB), for Reptim II and higher - up to 2^{36} bytes (64 GB);
- registers for storing general-purpose data;
- segment registers;
- status and control registers;
- registers of a floating point computing device (coprocessor);
- a set of registers of integer MMX extension mapped to coprocessor registers (first appeared in the Pentium MMX processor architecture);
- a set of MMX expansion registers with floating point (first appeared in the architecture of the Pentium III processor);
- software stack - a special information structure for which special machine instructions are provided.

In addition, the resources supported by the IA-32 architecture include I/O ports and performance monitoring counters.

The software models of earlier processors (i486, the first Pentium up to Pentium MMX inclusive) are distinguished by a smaller address space of RAM (maximum 2^{32} addresses, since the width of their address bus is 32 bits) and the absence of some register groups, see Fig. 5.1.

On fig. 5.1 for each group of registers in brackets it is shown starting with what model it appeared in the software model of Intel processors. If there is no such designation, then this means that this group of registers was present in the i386 and i486 processors.

As for the earlier i8086 / 88 processors, in fact they are also fully represented in the diagram, but they make up only a small part of it. The program model of these processors includes 8- and 16-bit general-purpose registers, segment registers, FLAGS, IP registers, and memory address space up to 1 MB in size.

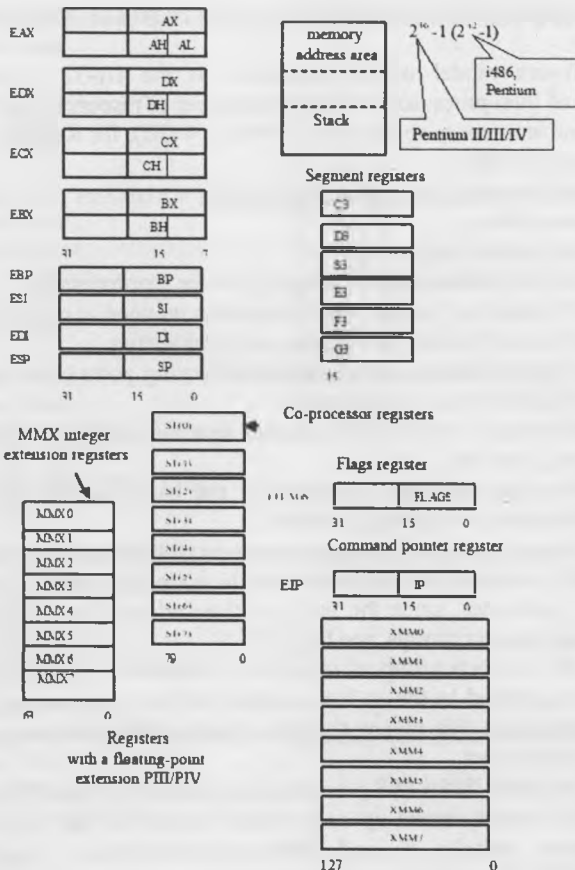


Fig. 5.1. Software model of the processor IA-32

Register set

Registers are the elements of high-speed memory located inside the processor in the immediate vicinity of its executive core. Access to them is much faster than to RAM cells.

The first group consists of user registers, which include:

- general purpose registers *EAX/AX/AH/AL*, *EBX/BX/BH/BL*, *EDX/DX/DH/DL*, *ECX/CX/CH/CL*, *EBP/BP*, *ESI/SI*, *EDI/DI*, *ESP/SP* are for storing data and addresses. Programmers use them to implement their algorithms;
- segment registers *CS*, *DS*, *SS*, *ES*, *FS*, *GS* are used to store values, the interpretation of which depends on the processor operating mode. In real mode, the segment registers contain the address of the paragraph at the beginning of the

segment in memory. In protected mode, segment registers store the entry index into one of the system descriptor tables - *GDT* or *LDT* (they will be discussed in the next chapter);

- co-processor registers *ST(0)*, *ST(1)*, *ST(2)*, *ST(3)*, *ST(4)*, *ST(5)*, *ST(6)*, *ST(7)* are designed to write programs that operate with data from floating point (dot);
- *MMX* integer registers - extensions *MMX0*, *MMX1*, *MMX2*, *MMX3*, *MMX4*, *MMX5*, *MMX6*, *MMX7*;
- *XMM* registers - extensions with floating point *XMM0*, *XMM1*, *XMM2*, *XMM3*, *XMM4*, *XMM5*, *XMM6*, *XMM7*;
- status and control registers (*EFLAGS/FLAGS* flag register and *EIP/IP* instruction pointer register) contain information about the state of the processor, the executable program and allow changing this state.

The second group includes system registers, that is, designed to provide various operating modes, service functions, as well as registers specific to a particular processor model. System registers supported by IA-32 include:

- control registers *CR0-CR4* - determine the processor operating mode and characteristics of the current executable task;
- memory control registers *GDTR*, *IDTR*, *LDTR* and *TR* are used in the protected processor mode to localize the control structures of this mode;
- *DR0-DR7* debug registers are designed to control and manage various aspects of debugging;
- *MTRR* memory area type registers are used for hardware-based caching control in order to assign corresponding properties to memory area;
- machine-dependent *MSR* registers are used to control the processor, monitor its performance, as well as receive error information.

In the designations of many registers of the program model there is an oblique dividing line, which means that these registers can be used in the program as separate objects. This is done to ensure the functionality of programs written for the previous 16-bit processor models from Intel, starting with i8086. The i486 and Pentium processors include mostly 32-bit registers. Their number, with the exception of segment registers, is the same as that of the i8086, but the capacity is greater, which is reflected in the notation - they have the prefix E (Extended).

General purpose registers

General purpose registers are used for storage:

- of instructions operands of an integer device;
- of addresses and address components.

There are no special restrictions on the storage of operands, but under certain conditions some of the registers have a rigid functional purpose, fixed at the level of the logic of machine instructions. In this regard, among all general-purpose registers, attention should be paid to the *ESP* register. It stores a pointer to the top of the program stack.

Let list the registers belonging to the group of general purpose registers and physically located in the processor inside the arithmetic-logical device (ALD registers):

- the accumulator register *EAX/AX/AH/AL* is used to store intermediate data, in some commands its use is mandatory;
- the base register *EBX/BX/BH/BL* is used to store the base address of some object in memory;
- the count register *ECX/CX/CH/CL* is used in commands that perform some repeated actions;
- the data register *EDX/DX/DH/DL*, as well as the register *EAX/AX/AH/AL*, allows storing intermediate data.

The following two registers are designed to support so-called chain operations, that is, operations that sequentially process chains of elements, each of which can be 32, 16 or 8 bits long:

- the source index register *ESI/SI* in chain operations contains the current address of the element in the source chain;
- the destination index register *EDI/DI* in chain operations contains the current address in the destination chain.

There are special commands for working with the stack in the processor command system, and in the processor software model there are special registers for this:

- the stack pointer register *ESP/SP* contains a pointer to the top of the stack in the current stack segment;
- the base pointer register *EBP/BP* of the stack frame is designed to organize random access to data inside the stack

Segment registers

Intel processors support segmented program organization in hardware. This means that any program consists of at least three segments: code, data and stack. Logically, machine instructions in the IA-32 architecture are constructed so that when fetching each instruction, information from well-defined segment registers is implicitly used to access program data or the stack. Depending on the operating mode of the processor, their contents determine the memory addresses from which the corresponding segments begin. The IA-32 programming model has six segment registers *CS, SS, DS, ES, GS, FS*, used to access four types of segments.

Code segment. Contains program commands. To access this segment, use the code segment register *CS*. It contains a value that is interpreted differently by the processor, depending on the current operating mode. In real mode (see the chapter) - this is the address of the first paragraph of the segment, in protected - the index of the element in the descriptor table (global *GDT* or local *LDT*).

Data segment. Stores data processed by the program. To access this segment, use the data segment register *DS*, in which the address of the data segment of the current program is placed.

Stack segment. Represents a region of memory called a stack. The processor organizes the work with the stack according to the following principle: the last element recorded in this area is selected first. Access to the stack area is done through the stack segment register *SS*, which contains the address of the stack segment.

Additional data segment. Implicitly, the execution algorithms for most machine instructions assume that the data they process is located in the data segment whose address is in the data segment register *DS*. If the program does not have enough one data segment, then it can simultaneously use three more additional data segments, the addresses of which should be contained in the registers of the additional data segment (extension data segment registers) *ES*, *GS*, *FS*. There should be paid attention to the *ES* register. It is used to provide chain commands to the processor by addressing the source operand.

Status and Control Registers

Two registers are included in the Intel processor (see. Fig. 5.1), constantly containing information about the status of both the processor itself and the program whose commands it is currently processing:

- instruction pointer register *EIP / IP*;
- flag register *EFLAGS / FLAGS*.

The *EIP / IP* instruction pointer register is 32/16 bit wide and contains the displacement of the next instruction to be executed relative to the beginning of the code segment. This register is not directly accessible to the programmer, that is, it cannot be specified as an operand of commands. But indirect access is possible to it, so loading and changing its contents is carried out as a result of the work of various program flow control commands, which include commands of conditional and unconditional jumps, call of procedures and return from procedures. Interruptions also modify the contents of the *EIP / IP* register.

EFLAGS / FLAGS flag register width (FLAG register) is 32/16 bits. The individual bits of this register have a functional purpose and are called flags. The smallest part of the *EFLAGS / FLAGS* register is completely similar to the *FLAGS* register of the i8086 processor. The fig. 5.2 shows the contents of the *EFLAGS* register.

ФЛАГИ СОСТОЯНИЯ

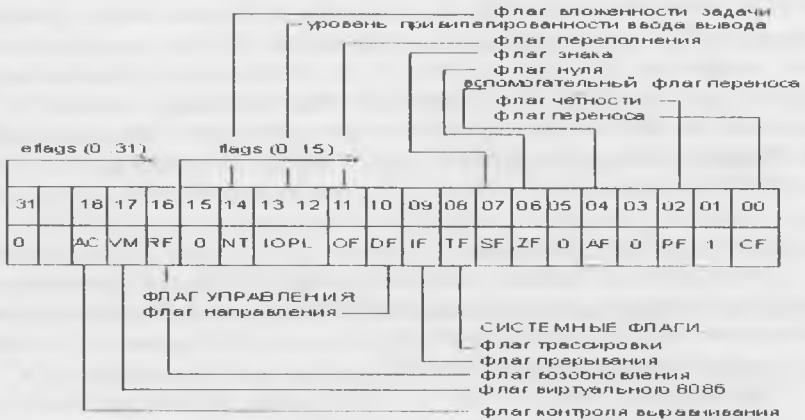


Fig. 5.2 EFLAGS register contents

Based on the features of use, the flags of the EFLAGS / FLAGS register are divided into three groups:

1. **State flags** - reflect the features of the result of the execution of arithmetic or logical operations and can be changed consciously:

- carry flag *CF* - if is 1, then the arithmetic operation carried out the transfer from the high bit of the result (or the transfer is initiated by the shift command) or the high bit of the result. The oldest is the 7th, 15th or 31st bit, depending on the dimension of the operand; if *CF* is 0, there was no transfer. This flag shows the overflow condition for unsigned arithmetic;

- parity flag *PF* - if it is 1, then the lower 8 bits (this flag is valid only for the lower 8 bits of the operand of any size) of the result contains an even number of units; if it is 0 - among the 8 least significant bits of the result, the number of units is odd;

- auxiliary carry flag *AF* - applies only to teams working with BCD numbers. *AF* - fixes the fact of a loan from the lower tetrad of the result: if *AF* is 1 - as a result of the addition operation, transfer was made from a digit 3 to the senior digit or during the subtracting - a loan to a digit 3 of the lower tetrad from the value in the senior tetrad; if it is 0, there was no result in transfers and loans to the third digit (from the third digit) of the younger tetrad;

- zero flag *ZF* - when equal to 1 - the result is zero; if equal to 0, the result is nonzero;

- sign flag *SF* - reflects the state of the highest bit of the result (bits 7, 15, or 31 for 8-, 16-, or 32-bit operands, respectively): if it is 1, the most significant bit of the result is 1; if equal to 0 - the leading bit of the result is also equal to 0;

- overflow flag *OF* — used to record the fact of the loss of a significant bit during sign arithmetic operations: if it is 1, the operation transfers to the most significant sign bit of the result or a make loan from the most significant sign bit of

the result (bits 7, 15 or 31 for 8-, 16- or 32-bit operands, respectively); a value of zero indicates no transfer or loan.

2. Directory flag *DF* - the tenth bit of the EFLAGS register.

The *DF* flag is used by chain commands and determines the direction of elementwise data processing by these commands: $DF = 0$ - from the beginning of the chain to its end, that is, from lower addresses to senior ones; $DF = 1$ - from the end of the chain to its beginning, that is, from the highest addresses to the lowest. The state of the *DF* flag can be changed with the *CLD* (clear the *DF* flag) and *STD* (set the *DF* flag) commands.

3. System flags and IOPL field - control input/output, masked interrupts, debugging, switching between tasks and the mode of the 8086 virtual processor:

- trace flag *TF* - designed to organize step-by-step operation of the processor: if it is 1, the processor generates an interrupt with number 1 after each machine command (it can be used at debugging programs); if it is equal to 0 - normal operation;

- interrupt enable flag *IF* - designed to enable or disable (mask) hardware interrupts (interrupts at the *INTR* input); when it is 1 - hardware interrupts are enabled; with equality to 0 - hardware interrupts are prohibited;

- input / output privilege level *IOPL* - used in the protected processor mode to control access to I/O commands depending on the priority assigned to the task;

- nested task flag *NT* — used in the protected processor mode to remember the fact that one task is connected to another in a call chain. The work of a chain of related tasks is organized using the *CALL*, *IRET* commands, *TR* task register, and *TSS* segment;

- resume flag *RF* - used when processing interrupts from debug registers;

- flag of the mode of the virtual processor 8086 (virtual 8086 mode) *VM* - a sign of the current mode of the processor: if it is 1, the processor operates in the mode of virtual processor 8086; if it is 0 - the processor works in real or protected mode;

- alignment check flag *AC* - designed to enable alignment control accessing memory. Used in conjunction with the *AM* bit in the *CR0* system register. IA-32 processors (starting with i80486) allow placing commands and data starting from any address. Setting these bits (with privilege level $CPL = 3$), as a rule, will lead to an exception trying to access data and commands at addresses multiple of 2 or 4;

- virtual interrupt flag *VIF* (Pentium only) - under certain conditions, one of which is processor operation in *V* mode, is an analog of the *IF* flag. The *VIF* flag is used in conjunction with the *VIP* flag (with $CR4.VME = 1$);

- virtual interrupt pending flag *VIP* (Pentium only) - set at 1 to indicate a delayed interrupt;

- identification flag *ID* - used to indicate the fact that the processor supports the *CPUID* instruction. If the program can set or clear this flag, then this means that this processor model supports the *CPUID* instruction.

CPU system registers

These registers perform specific functions in the system. Their use is strictly regulated. They provide the protected mode. They can also be seen as part of the processor architecture, which is intentionally left visible in order to allow a qualified system programmer to perform the most low-level operations.

System registers can be divided into three groups:

- four control registers;
- four registers of system addresses;
- eight debug registers.

The following changes have been made to the system registers of Pentium processors:

- the previously reserved CR4 control register is activated;
- added a group of *MSR* registers (*MSR* - *Model Specific Register*), the purpose and capabilities of which are subordinate to the architecture of a particular processor model.

Control registers

The group of control registers includes five registers: *CR0*, *CR1*, *CR2*, *CR3*, *CR4*. They are intended for general system control. Control registers are available only to programs with privilege level 0. From their total number, only four registers are available; *CR1* is excluded, whose functions have not yet been determined (it is reserved for future use).

The *CR1* register contains system flags that control the processor operating modes and reflect its status globally, regardless of specific tasks being performed. Register *CR1* is used in the page model of memory organization. Purpose of system flags:

PE (protect enable), bit 0 - enable protected operation mode. The state of this flag indicates in which of the two modes - real ($PE = 0$) or protected ($PE = 1$) - the processor is operating at a given time.

MP (math present), bit 1 - the presence of a coprocessor. Always 1.

TS (task switched), bit 3 - task switching. The processor automatically sets this bit when switching to another task.

AM (alignment mask), bit 18 is the alignment mask. This bit enables ($AM = 1$) or disables ($AM = 0$) alignment control.

CD (cache disable), bit 30 - cache blocking. Using this bit, you can disable ($CD = 1$) or enable ($CD = 0$) the use of the internal cache (first level).

PG (paging), bit 31 - enable ($PG = 1$) or disable ($PG = 0$) pagination.

The *CR2* register is used in the paging organization of random access memory to register a situation when the current command has addressed to an address belonging to a memory page that is not currently in memory. In such a situation, an exception with the number 14 occurs in the processor, and the linear 32-bit address of the command that caused the exception is written to register *CR2*. Having this information, the exception handler 14 determines the desired page, loads it into memory, and resumes the normal operation of the program.

Register *CR3* is also relevant for paging memory.

This is the so-called first level page directory register. It contains the 20-bit physical base address of the page directory of the current task. This directory contains 1024 32-bit descriptors, each of which stores the address of the page table of the second level. In turn, each of the second-level page tables contains 1,024 32-bit descriptors addressing page frames in memory. The size of the page frame is 4 KB.

The *CR4* register is a collection of features, mostly permissive, that characterize certain architectural elements that first appeared in various models of Pentium processors. Examples of such properties include the following: support for 36-bit addressing, the use of delayed interrupts in the i8086 virtual processor mode, support for 4 MB pages, etc. By setting these or those bits in the *CR4* register, you can enable or disable support for these properties.

System Address Registers

These registers are also called memory control registers. They are designed to protect programs and data in a multitask processor mode. Working in protected processor mode, the address space (AS) is divided into:

- global AS - a common solution to all problems;
- local AS - separate for each task.

It is the division that explains the following system registers in the MP architecture:

- the global descriptor table register is 48 bits wide and contains the 32-bit base address of the global descriptor table GDTR and the 16-bit limit size value, which is the size in bytes of the GDT table;
- the local descriptors table register (has a size of 16 bits) LDTR contains a descriptor selector for the local descriptor table;
- the Interrupt Descriptop Table Register is 48 bits wide and has a 32-bit base address for the interrupt descriptors table (IDT) and a 16-bit limit value, that is, size in bytes of the IDT table;
- The 16-bit task register TR, like the LDTR register, contains a selector, that is, a pointer to a descriptor in the GDT table. This descriptor describes the current task state segment (TSS). The latter is created for each task in the system, has a strictly regulated structure and stores the context (current state) of the task. The main purpose of TSS segments is to remember the current state of the task at the time of switching to another task.

Debug Registers

The following group of registers designed for hardware control of program execution. Hardware debugging tools first appeared in the i486. The processor contains eight debug registers, but in reality only six are used from them.

The registers *DR0*, *DR1*, *DR2*, *DR3* have a capacity of 32 bits and are designed to set the linear addresses of the four debug points. The following mechanism applies: any address generated by the current program is compared with the addresses in the *DR0-DR1* registers, and if they match, a debug exception is generated with number 1.

The *DR6* register is called the debug state register. The bits of this register are set in accordance with the reasons that caused the last exception occurrence with the number 1. We list these bits and their purpose.

B0 - if set at 1, then the last exception (interruption) occurred as a result of reaching the control point defined in the *DR0* register;

B1 - similar to *B0*, but for the control point in the *DR1* register;

B2 - similar to *B0*, but for the control point in the *DR2* register;

B3 - similar to *B0*, but for the control point in the *DR3* register;

BD (bit 13) - serves to protect debug registers;

BS (bit 14) - set at 1 if exception 1 was caused by the status of the flag *TF* = 1 in the *EFLAGS* register;

BT (bit 15) - set at 1 if exception 1 was caused by switching to a task with the trap bit set (*T* = 1) in *TSS*.

All other bits in this register are filled with zeros. The exception handler 1 from the contents of *DR6* must determine the reason for the exception and perform the necessary actions.

The *DR7* register is called the debug control register. In it, for each of the four registers of control points of debugging, there are allocated fields with the help of which we can clarify the following conditions under which an interrupt should be generated:

- control point registration location - only in the current task or in any task. The corresponding signs are occupied by the least eight bits of the *DA7* register: two bits for each control point (actually, an interruption point) defined by the registers *DR0*, *DR1*, *DR2*, *DR3*. The first bit of each pair is the so-called local resolution, its setting indicates that the control point is valid if it is within the address space of the current task. The second bit in each pair determines the global resolution, that is, this control point acts within the address spaces of all tasks running in the system;

- type of access by which interruption is initiated: only at a command selection, writing, or writing/reading data. The bits that determine the nature of the interrupt are concentrated in the upper part of this register.

Most of the system registers are software accessible. The above part of the software model of the processor is not complete. It is enough to discuss integer processor instructions, see chapter 18.

Microprocessor command format

The Intel-8086 microprocessor has a dual-address instruction system [1,10]. Its feature is the lack of instructions using both operands from RAM. The only exceptions are commands for sending and comparing byte chains or words, which will not be considered in this chapter of the manual.

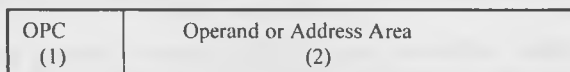
The instruction system is tightly connected with a specific processor type, since it is determined by the hardware structure of the instruction decryption unit, and usually does not have portability to other types of processors (although there may be bottom-up compatibility within a series of processors, as, for example, in the i80x86 series).

From a physical point of view, the command code is no different from ordinary data in binary code located in the memory of the calculator. A specific binary code is perceived and processed by the processor as an instruction in case when it enters the processor in the phase of reading the instruction code.

From a logical point of view, in the binary code of a command there are groups of bits - fields - with different functional purposes (Fig. 5.3).

OPC - operation code - a binary code that unambiguously instructs the processor to perform specific actions (transfer, addition, etc.), and determines the form of specifying operand addresses;

Figure 5.3 shows two main fields in the command format:



1 - operation code field (КОП) of a size of 1 or 2 bytes;

2 - address part (АР) field of the command of a size of 1 to 4 bytes.

Fig. 5.3 Typical command format structure

AP - address part - a binary number, which can be the address (s) of the operands, the value of the operand, the address of the next command (transition address, control transfer). Therefore, the format of a command is a combination of its characteristics such as the number, size and purpose of the fields.

Therefore, the command is divided into two areas: the area of the operation code (OPC) and the area of addresses. An address area can consist of several parts - these are the so-called multicast commands. From the point of view of the programmer, the most convenient are multi-address ones.

The address area consists of three fields: the first two are the addresses of the operands, and the third will contain the address of the result of the action on the operands.

In two-address commands, the address area consists of two fields: the address field (AF) of the first operand and the AF of the second operand. The address of the result of the action on the operands will be recorded in the first field, see Fig. 5.4.

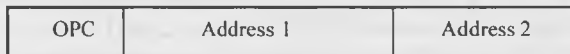


Fig. 5.4. Two-Address Command Diagram

In uniaddress commands, the address area consists of one single field, which contains the address of the operand, and the address of the second operand and the result coincides with the adder. There are also addressless commands that are used at working with stacks. The most commonly used two-, single- and non-address commands.

The addressing task is to indicate the current memory location to which the processor is accessing.

Memory addressing methods are of particular importance at programming in a low-level language (assembly language), see chapter 18.

With direct addressing, no memory access is required to select the operand and memory cell for storing it. This helps to reduce the execution time of the program and the amount of memory it takes. Direct addressing is convenient for storing various kinds of constants.

With direct addressing, the address is indicated directly in the form of some value, all cells are located on one page. The advantage of this method is that it is the simplest one, and the disadvantage is that the width of the general-purpose registers of the processor must be no less than the width of the bus address of the processor.

At a relative addressing, the executive address is defined as the sum of the address code of the command and the base address, usually stored in a special register - the base register.

Relative addressing allows for a shorter address code for the command to provide access to any memory location. For this, the number of bits in the base register is chosen so that it is possible to address any memory cell, and the address code of the command is used to represent only a relatively short "displacement". The displacement determines the position of the operand relative to the beginning of the array specified by the base address.

With a shortened addressing, the address field of the control word contains only the least significant bits of the addressable cell.

Register addressing is a special case of shortened. It is used when intermediate results are stored in one of the working registers of the central processor. Since there are much fewer registers than memory cells, a small address field may be enough for addressing.

In the case of indirect addressing, the address code of the instruction in this case indicates the address of the memory cell in which the address of the operand or instruction is located. Indirect addressing is widely used in small and microcomputers with a short machine word to overcome the limitations of the short command format (register and indirect addressing are shared), see Fig. 5.5. At indirect addressing, the number of cells with addresses of other cells can be several (chain), see Fig. 5.6.

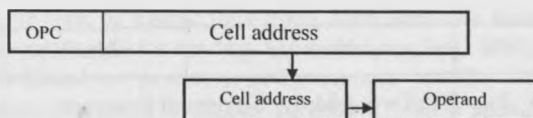


Fig. 5.5. Indirect Addressing Scheme

Since register indirect addressing requires preliminary loading of the register with an indirect address from RAM, which is associated with time loss, this type of

addressing is especially effective when processing a data array if there is a mechanism for automatically incrementing or decreasing the contents of the register each time it is accessed. Such a mechanism is called auto-incrementing and auto-decrementing addressing, respectively. In this case, it is enough to load the address of the first processed element of the array into the register once, and then with each access to the register the address of the next element of the array will be formed in it.

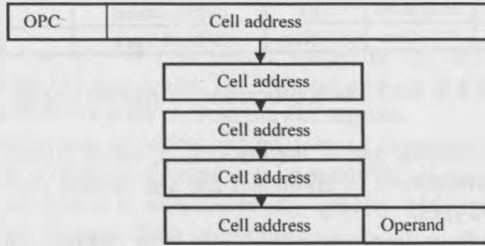


Fig. 5.6. Indirect addressing chain scheme

In auto-incrementing addressing, firstly the contents of the register are used as the address of the operand, and then it receives an increment equal to the number of bytes in the array element. In case of auto-decrementing addressing, firstly the contents of the register specified in the command are reduced by the number of bytes in the array element, and then it is used as the operand address.

Direct addressing assumes that the operand occupies one of the fields of the command and, therefore, is selected from RAM simultaneously with it. Depending on the formats of the data processed by the processor, the direct operand may have a length of 8 or 16 bits.

The addressing mechanisms of operands located in register memory and in random access memory differ significantly. Only direct register addressing is allowed to register memory. At the same time, the number of the register containing the operand is indicated in the command. The 16-bit operand can be in the registers *AX, BX, CX, DX, DI, SI, SP, BP*, and the 8-bit operand can be in the registers *AL, AH, BL, BH, CL, CH, DL, DH*.

Addressing of RAM has its own characteristics associated with its partitioning into segments and the use of a segment group of registers to indicate the starting address of the segment. The 16-bit address obtained in the operand address generation unit based on the specified addressing mode is called an *effective address (EA)*. Sometimes an effective address is referred to as **EA**. A 20-bit address, which is obtained by adding the effective address and the value of the corresponding segment register increased by 16 times, is called the *physical address (FA)*.

It is the physical address that is transferred from the microprocessor through 20 address lines that are part of the system bus to the RAM and is used when

accessing its cell at the physical level. Obtaining an effective address, all the basic addressing modes discussed above, as well as some combinations, can be used.

Direct addressing assumes that an effective address is part of a command. Since **EA** consists of 16 bits, the corresponding field of the command must have the same length.

In general, the IA-32 microprocessor instruction may contain the following fields, see Fig. 5.7.

prefix	OPC	Mod R/M	SIB	displacement	direct operand
0/1 bites	1/2 bites	0/1 bites	0/1 bites	0/1/2/4 bites	0/1/2/4 bites

Fig. 5.7. IA-32microprocessor instruction circuit

The prefix - an optional part of the instruction, which allows to change some features of its implementation. A command can use several prefixes of different types at once. Prefix Types:

- command prefixes (repetition prefixes) *REP*, *REPE* / *REPZ*, *REPNE* / *REPNZ*;

- bus lock prefix *LOCK*; size prefixes;

- segment replacement prefixes.

OPC - operation code.

The "**Mod R/M**" byte defines the addressing mode, as well as sometimes an additional opcode. The need for the "**Mod R/M**" byte depends on the type of instruction.

The *SIB (Scale-Index-Base)* byte determines the addressing method accessing memory in 32-bit mode. The need for a **SIB** byte depends on the addressing mode specified by the "**Mod R/M**" field.

In addition, the instruction may comprise an immediate operand and/or a displacement of the operand in the data segment.

The instruction size is limited to 15 bytes. A larger instruction may result from incorrect use of a large number of prefixes. In IA-32, then there is generated an exception # 13.

If the microprocessor instruction requires operands, then they can be specified in the following ways: directly in the instruction code (source operand only); in one of the registers; through the input/output port; in memory.

For compatibility with 16-bit processors, the IA-32 architecture uses the same codes for instructions that operate on both 16-bit and 32-bit operands. The new architecture also provides new capabilities specifying the address for the operand in memory. How the processor will read the operand or its address depends on the effective size of the operand and the effective size of the address for this instruction. These values are determined based on the operating mode, bit **D** of the descriptor of the used segment and the presence of certain prefixes in the instruction.

Direct addressing mode involves the inclusion of the source operand in the instruction code. The operand can be 8-bit or 16-bit if the value of the effective operand size is 16. The operand can be 8-bit or 32-bit if the value of the effective size of the operand is 32. Typically, direct operands are used in arithmetic instructions.

Register addressing mode will determine the source operand or receiver operand in one of the registers of the processor or coprocessor.

In some cases, (for example, in the *DIV* and *MUL* instructions, see chapter 18, pairs of 32-bit registers (for example, *EDX: EAX*) can be used, forming a 64-bit operand.

Addressing via the I/O port means receiving the operand or storing the operand through the I/O port space. The I/O port address is either directly included in the instruction code, or is taken from the *DX* register.

A very common way of addressing an operand is memory addressing. Thus, a source operand or a receiver operand can be specified. It should be noted that the processor does not allow to simultaneously specify both operands via memory (with the exception of some chain instructions).

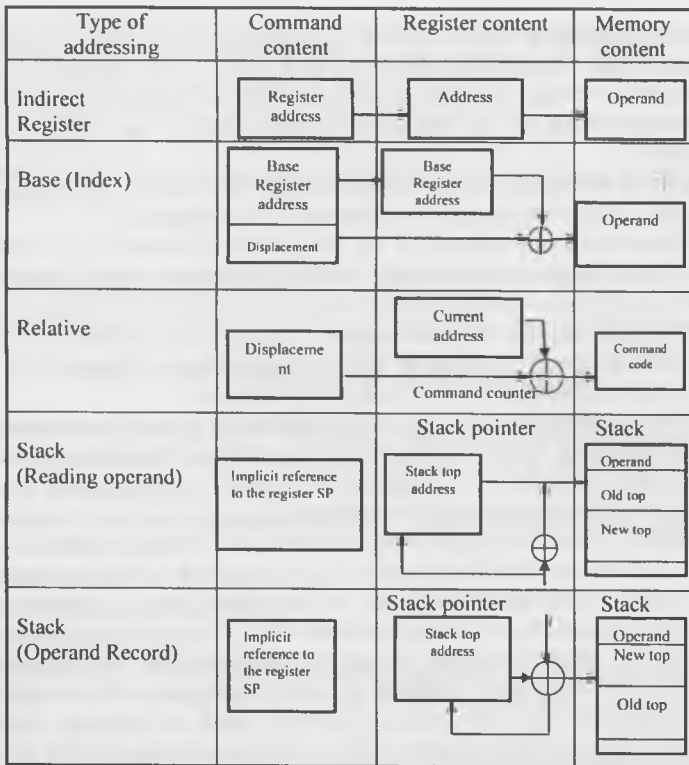
In order to obtain an operand from memory, the processor needs to know the segment selector and the displacement in the segment. In some commands, the selector can be specified directly in the instruction code. In other cases, the processor may explicitly or implicitly use the value of one of the segment registers. Implicit use of segment registers means that, depending on the destination of the operand, the processor uses a specific segment register to access memory: *CS* - to select instructions; *SS* - for working with the stack or accessing the memory through the *ESP* or *EBP* registers; *ES* - to obtain the address of the operand receiver in chain commands; *DS* - for all other memory accesses. Explicit use of segment registers is possible if a segment change prefix is included in the instruction code. Specifying a segment change prefix is not permissible for all commands: you cannot change a segment for stack operations commands (*SS* is always used); for chained instructions, you can only change the segment of the source operand (the destination operand is always addressed via *ES*).

Table 5.1 shows the main addressing methods in MP.

Table 5.1

Basic Addressing Methods Implementation

Type of addressing	Command content	Register content	Memory content
Direct	Address		Operand
Straight	Operand		
Register	Register address	Operand	



Questions for self-control

1. What is a software processor model?
2. List the register groups.
3. Assignment of debug registers?
4. The microprocessor command format.
5. The purpose of segment registers?
6. Microprocessor instruction formats.
7. Types of addressing?
8. What is a command prefix?
9. Define the mechanism for addressing operands

Test questions

1. The command system, types of processed data, addressing modes and principles of microprocessor operation are
 - A) Macroarchitecture

- B) Microarchitecture
- C) Miniarchitecture
- D) Monoarchitecture

2. With what help does the microprocessor coordinate the operation of a devices of the digital system?

- A) using the data bus
- B) using the address bus
- C) using the control bus
- D) using read-only memory (ROM)

3. What is the structural element of the format of any command?

- A) Register
- B) Cell Address
- C) operand
- D) Operation Code (CPC)

4. is a procedure or scheme for converting information about an operand to its executive address

- A) Memory coding mode
- B) memory addressing mode
- C) memory format mode
- D) memory maintenance mode

5. One of the ways to exchange memory to external devices is:

- A) Direct memory access mode
- B) Memory interrupt generation mode
- C) Program memory management mode
- D) memory maintenance mode

6. The command distribution: according to their functional purpose, data transfer, data processing, transfer of control and

- A) without address
- B) one address
- C) optional
- D) two address

7. The processor has 7 general purpose registers. How many bits in the command field are needed to address them.

- A) 7
- B) 4
- C) 3
- D) 8

8. The processor has 14 general-purpose registers. How many bits in the command field are needed to address them.
- A) 7
 - B) 4
 - C) 3
 - D) 8
9. General purpose registers are used to store:
- A) arithmetic operands
 - B) address components
 - C) memory pointers
 - D) logical operations
10. What resolution is Ultra HD quality?
- A) 4096x3072
 - B) 4096x1440
 - C) 3840x2160
 - D) 3920x1080
11. In the universal 32-bit microprocessor, the following groups of registers are distinguished:
- A) main functional registers;
 - B) floating point processor registers;
 - C) system registers;
 - D) debug and test registers.
12. The structure of the registers of this group includes:
- A) general purpose registers;
 - B) instruction pointer register;
 - C) flag register;
 - D) segment registers.
13. What addressing method has the most compact code?
- A) Register
 - B) direct
 - C) indirect
 - D) relative register
14. The microprocessor software model consists of
- A) 32 registers
 - B) a set of special programs
 - C) there is no correct answer
 - D) a set of control programs
15. The processor registers are designed to

- A) there is no correct answer
- B) permanent storage of information
- C) signal transcoding
- D) temporary storage of information

16. A functional computer unit for recording, storing, reading an n-bit word of information is called:

- A) by register
- B) trigger
- C) encoder
- D) counter

17. What is the main way of addressing in MP is not?

- A) straight
- B) indirect
- C) reverse
- D) index

18. What registers are designed to protect programs and data in a multitask processor mode?

- A) system address registers
- B) general purpose registers
- C) stack registers
- D) flag registers

19. What registers are designed for hardware control of program execution?

- A) system address registers
- B) general purpose registers
- C) debug registers
- D) flag registers

20. What registers are physically located in the processor inside the ALD?

- A) system address registers
- B) general purpose registers
- C) debug registers
- D) flag registers

CHAPTER 6. IA-32 PROCESSOR STRUCTURE

The main device of any computer is the central processor. Earlier (see Chapter 5), it was noted that each processor is characterized by a set of instructions that it is able to execute. For example, Intel's Pentium processor cannot process programs (and thus instructions) written for Sun's SPARC processor, and SPARC cannot run programs written for Pentium.

The enlarged structural diagram of a typical processor can be represented in the form of three main blocks: a control unit (CU), an operating unit (OU) and an interface unit (IU). The control unit performs the functions of sampling, decoding and calculating the addresses of operands, and also generates sequences of microcommands that implement processor instructions. It contains a control device, interrupt, synchronization. The operating unit is used for data processing. It combines the arithmetic logic device (ALU), general purpose registers (GPR) and special registers. ALU performs arithmetic (addition, subtraction, etc.) and logical (logical AND, OR, etc.) operations. Registers are a kind of memory CU, designed to store intermediate results and some control commands, information about the state of the processor. Information from them is read and written very quickly, since they are inside the processor. Registers can be from a few dozen to several hundred pieces, depending on the type of processor. A large number of registers are characterized by RISC - processors, and a small - CISC - processors. IU allows to connect memory and peripherals to the processor, and also acts as a channel for direct access to memory. The processor interface contains information buses for data (DB), addresses (AB), and control (CB). It should be noted that such a distribution of the processor hardware blocks between the functional parts is very arbitrary and is given as an example.

The processor executes each command in several steps:

- calls the next command from memory and transfers it to the register of commands;
- changes the position of the command counter, which should now point to the next command;
- determines the type of command called;
- if the command uses data from memory, determines the location of the data;
- transfers data to the processor register;
- executes a command;
- proceeds to step 1 to begin the execution of the next command.

This sequence of steps (sampling - decoding - execution) is the basis of work for all processors.

A simplified block diagram of a typical processor is shown in the following figure.

The basic structure of processors (Π) IA-32 can be considered on the example of Intel-386 (Fig. 6.1). In the Intel-386 structure, there are six main units that work in parallel: an interface unit with a trunk, an instruction prefetch unit, an instruction decoding unit, an executive unit, a segment control unit, and a page translation unit.

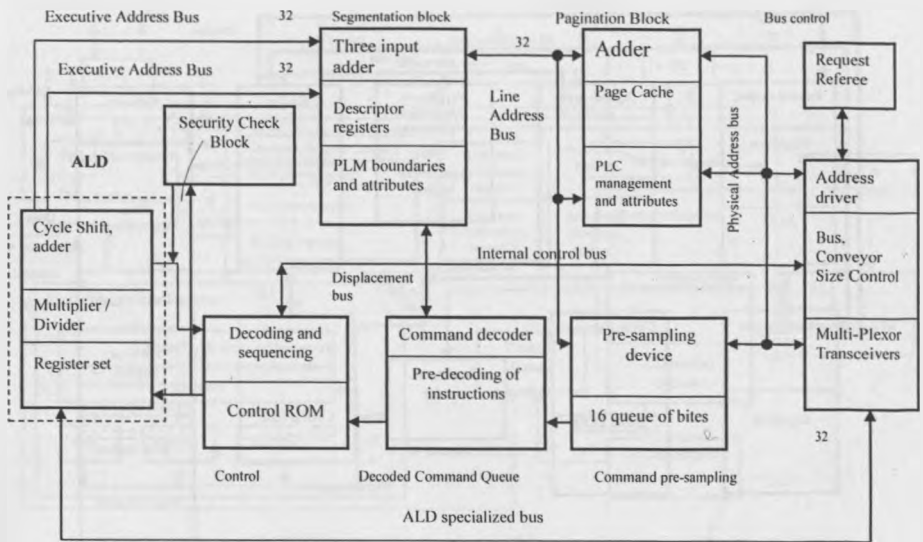


Fig. 6.1. Block diagram of the processor 80386

The interface block with the trunk contains the address driver, address size and conveyor control circuits, a multiplexer, transceivers, etc. This provides an interface between the processor and its environment. It receives internal requests for sampling commands from the sample unit and for exchanging data with the execution unit and sets the priority of these requests. At the same time, it generates or processes signals to execute the current trunk cycle. These include address, data, and control signals for accessing external memory and I/O devices. Using the request arbiter scheme, the unit controls the interface with external line master and coprocessors.

In order to pre-receive commands or data before actually using them, there is a function of anticipating viewing a program, which in Intel-386 is executed by a block of commands pre-sampling. When an interface unit with a trunk does not take a trunk loop to execute a command, the command sampling block uses it to sequentially retrieve command bytes from the memory. These instructions are stored in a 16-byte instruction queue awaiting processing by the instruction decoding unit.

In the Intel-486 processor (Fig. 6.2), this block was supplemented by a parity (equalization) control scheme and a packet control scheme. Based on the last one, a special mode of operation of the trunk was implemented - the packaging mode. In this mode, when transmitting 4 words on the trunk, only the address of the first is set, which can significantly reduce the time for exchanging data with RAM or an external cache.

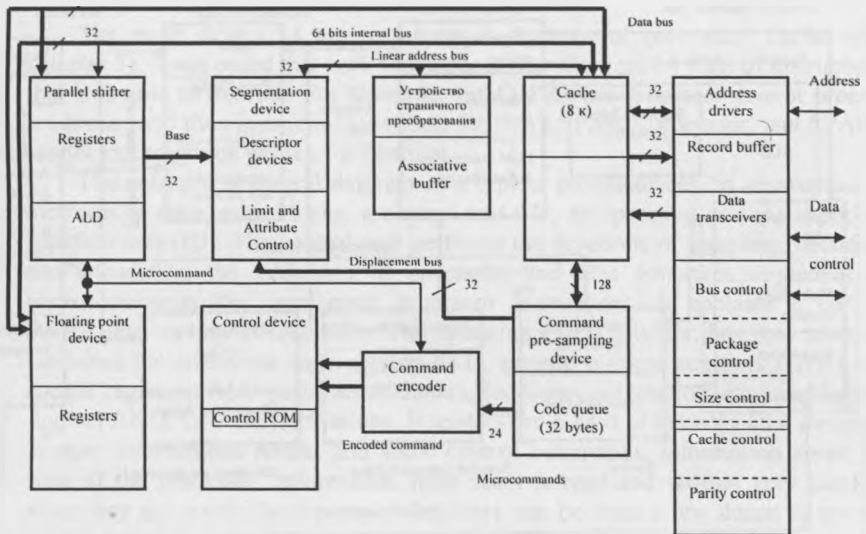


Fig. 6.2. Block diagram of the processor 8048

New Pentium processor microarchitecture, see fig. 6.3, and later based on the idea of superscalar processing. Superscalarism means the presence of more than one conveyor for processing commands (in contrast to the scalar - single-conveyor architecture).

Conveyorization allows several indoor units of the MP to work simultaneously, see table. 6.1, combining command decryption, ALD operations, calculating the effective address and bus cycles of several commands. MP 80286 included 4 conveyor devices:

- *BU* - bus unit (read from memory and input/output ports);
- *IU* - instruction unit (command decryption);
- *EU* - executive unit (execution of commands);
- *AU* - address unit (calculates all addresses, forms a physical address).

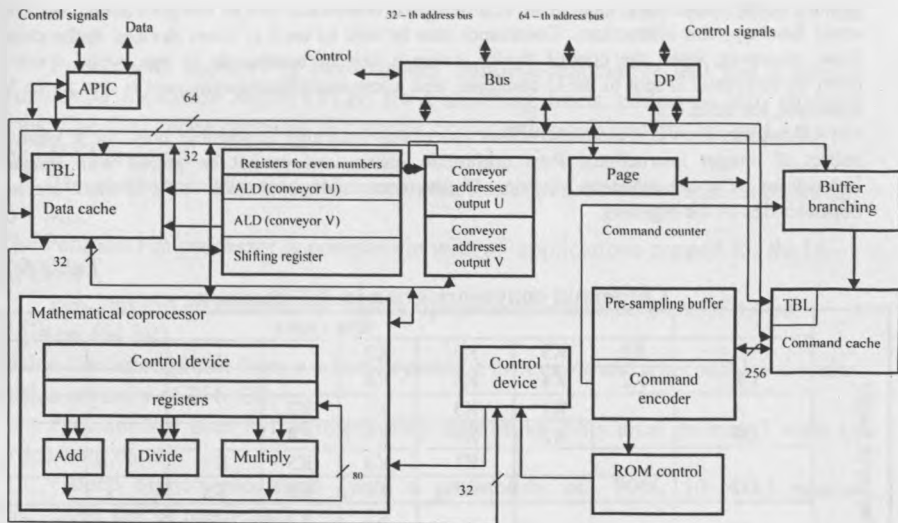


Fig. 6.3. Block diagram of the Pentium processor

Table 6.1

Command conveyORIZATION in MP 80286

		Bus cycles				
microoperations	B command sampling U	N+1	N+2			
	N-1	I command encoding U	N+1	N+2		
	N-2		A operand address formation U	N+1	N+2	
		N-2		B operand sampling U		N+2
			N-2	N-1	E command execution U	N+1
				N-2	N-1	B result record U

The idea of **conveyorization** was developed in the following models of this family. The Intel-486 MP has a five-stage **conveyorization** for processing commands:

- *PF (Prefetch)* - prefetch commands;
- *D1 (Instruction Decode)* - decoding a command;
- *D2 (Address Generate)* - address generation;
- *EX (Execute)* - command execution in ALD and access to cache memory;
- *WB (Write Back)* - write back.

In Pentium MP, commands are distributed across two independent execution conveyors (U and V). The U conveyor can execute any IA-32 family instruction, including integer and

floating-point instructions. Conveyor V is designed to execute simple integer instructions and some floating-point instructions. Commands can be sent to each of these devices at the same time, moreover, when the control device issues a pair of commands in one cycle, a more complex command is sent to the U conveyor, and a less complicated command is sent to the V conveyor, see table. 6.2.

However, such pairwise instruction processing (pairing) is possible only for a limited subset of integer instructions. Real arithmetic instructions cannot be paired with integer instructions. The simultaneous issuance of two commands is possible only if there are no dependencies on the registers.

Table 6.2

Command conveyorization in MP Pentium

Conveyor stages	Bus cycles								
		K1	K3	K5	K7				
PF		K1	K3	K5	K7				
		K2	K4	K6	K8				
D1			K1	K3	K5	K7			
			K2	K4	K6	K8			
D2				K1	K3	K5	K7		
				K2	K4	K6	K8		
EX					K1	K3	K5	K7	
					K2	K4	K6	K8	
WB						K1	K3	K5	K7
						K2	K4	K6	K8

The Pentium processor includes all the features of the Intel-486 processor and has a number of new significant features, such as:

- **superscalar architecture**, including two conveyor and allowing to execute more than one command in one processor cycle;
- prediction of branches in the program, which is implemented by special logic circuits that determine the control transfer point in the program and provide preliminary preparation for the execution of certain fragments of the program;
- conveyor device for floating point data processing (FPU);
- separate cache memory for commands and data with a capacity of 8 KB each;
- support for the MESI write-back protocol (Modified / Exclusive / Shared / Invalid) for the data cache;
- 64-bit DB and 32-bit DB;
- conveyorization of the machine cycle;
- control on the parity of the address and data;
- internal parity control;
- System management mode.

The Pentium processor includes the full set of instructions for the Intel-486 processor and contains a number of new instructions that expand its functionality.

To predict branching in a program, the Pentium processor contains two instruction preselection buffers, one of which provides preselection of instructions on a linear section, and the other serves to preselect instructions in accordance with the BTB (*Branch Target Buffer*) algorithm for operating the target branching

buffer. This almost always allows to prefetch the necessary execution of the command.

Each cache memory is two-channel multiple-associative and has a special *translation lookaside buffer* (TLB) for translating linear addresses into physical ones. The data cache provides Writeback or Writethrough line by line and supports the MESI protocol.

The memory management device in the Pentium processor supports pages up to 4 MB.

The Pentium Pro processor is compatible with all applications created for the IA-32 family.

Pentium Pro architecture is mainly optimized for intensive use of 32-bit registers (64 bit).

Inside the microcircuit there are two crystals: a processor and a second-level cache with a capacity of 256 KB.

Pentium Pro uses dynamic program execution. This term defines 3 ways of processing data:

- deep branch prediction (with a probability of > 90%, 10 = 15 nearest transitions can be predicted);
- analysis of the data flow (look at the program 20-30 steps ahead and determine the dependence of commands on data or resources);
- leading execution of instructions (P6 CPU can execute instructions in a manner different from their following in the program).

Many algorithms for working with multimedia data allow the simplest elements of parallelization, when one operation can be performed in parallel on several numbers. This approach is called *SIMD - single-instruction multiple-data*. This technology was first implemented in the P55 generation (Pentium MMX microprocessor).

MMX (Multi-Media eXtension) is a SIMD extension for stream processing of integer data, implemented on the basis of an FPU (using FPU registers). The MMX instruction set operates on 64-bit MM0-MM7 registers, which are aliases for the lower 64-bit part of FPU registers R0-R7), so that simultaneous execution of MMX instructions and real arithmetics is impossible. MMX instructions operate on 64-bit data types: packed bytes (8 x 8 bits); packed words (4 x 16 bits); packed double words (2 x 32 bits); four word (1 x 64 bit).

Therefore, a single MMX instruction can perform an arithmetic or logical operation on "packets" of integers packed in MMX registers. For example, a *PADDQB* instruction adds 8 bytes of one "packet" with the corresponding eight bytes of another packet, actually adding eight pairs of numbers with one instruction.

The first P6 processor (Pentium Pro) was developed before the Pentium MMX, so it does not have this feature, however, in subsequent P6 models, this technology has become fixed.

The Pentium II processor combines the best features of Intel processors: Pentium Pro processor performance and MMX technology capabilities. This combination provides a significant increase in the performance of Pentium II

processors compared to previous processors IA-32 architecture. The processor contains separate internal blocks of the commands cache and data of 16 KB and 512 KB of total non-blocking cache memory of the second level.

For the first time, the high-performance architecture of a double independent bus (system bus and cache bus) was implemented, providing increased throughput and performance, as well as scalability when using future technologies.

The development of the **SIMD** idea for real numbers was the *SSE (Streamed SIMD Extensions)* technology, first introduced in Pentium III processors. The SSE unit complements MMX technology with eight 128-bit XMM0-XMM7 registers and a 32-bit MXCSR control and status register. The XMM0-XMM7 registers are independent, i.e., unlike the MM0-MM7 registers, they are not mapped to any other processor registers. SSE instructions operate on a 128-bit single-precision packed data type (4 x 32 bits) containing 4 real numbers in the IEEE-754 single precision format. SSE instructions can perform operations on "packets" of real numbers, that is, one instruction performs an operation on a package of four pairs of real numbers.

Pentium 4 introduced SSE2 technology, complementing SSE with new data types and new instructions. SSE2 instructions also operate with 128-bit XMM0-XMM7 registers, but five new data types are added: packed double precision (2 x 64 bits IEEE-754 double precision); packed bytes (16 x 8 bits); packed words (8 x 16 bits); packed double words (4 x 32 bits); packed quad words (2 x 64 bits).

IA-32's performance improvement was achieved not only by optimizing the instruction conveyor and adding execution units, but, for example, by introducing cache memory into the processor core. In the IA-32 family, the created 8KB L1 cache was first implemented in Intel-486 processors. On Pentium processors, the cache size has been doubled. The first representatives of P6 (Pentium Pro) also contained L2 cache size of 256 or 512 KB. However, such a solution at that time was too expensive and disadvantageous, therefore, the Pentium II introduced the technology Dual Independent Bus (DIB) - *a double independent bus*. The processor was in the form of a cartridge with a printed edge connector, which displays the system bus. On a cartridge with a size of 14x6.2x1.6 cm, there were installed a processor core chip (CPU Core), several chips that implement a secondary cache, and auxiliary discrete elements (resistors and capacitors). Removing the secondary cache from the processor chip allowed the use of third-party microcircuits specializing in the release of ultra-fast memory for cache memory and tag memory. The size of the secondary cache was determined by the capacity and number of installed memory chips. Separate buses were used to access the cache and to access external memory. The same architectural solution was used in the first Pentium III models. Since 1999 (Pentium III Coppermine), the L2 cache has again been returned inside the processor crystals.

The Pentium 4 processor is a 32-bit representative of the IA-32 family, which belong to the new seventh generation (according to Intel classification) by its microarchitecture. From a software point of view, it is an IA-32 processor with the next expansion of the command system - SSE2. Pentium 4 repeats the Pentium III processor in a set of software-accessible registers. From an external, hardware

point of view, this is a processor with a new type of system bus, in which, in addition to increasing the frequency, the principles of double (2x) and fourfold (4x) synchronization are already familiar, as well as a number of measures have been taken to ensure operability at previously unimaginable frequencies. The processor microarchitecture, called NetBurst, see fig. 6.4, is designed taking into account the high frequencies of both the core (> 1.4 GHz) and the system bus (400 MHz). The name of the microarchitecture indicates the network orientation of the processor: its power is necessary for resource-intensive multimedia Internet applications.

Pentium 4 processor is single-chip. In addition to the actual computing core, it contains two levels of cache memory. The secondary cache, common to instructions and data, has a size of 256 KB and a bus width of 256 bits (32 bytes), as in the latest Pentium III processors. The secondary cache bus runs at the core frequency, which ensures its throughput is $32 \times 1.4 = 44.8$ GB/s at a frequency of 1.4 GHz. The secondary cache has ECC control to detect and to correct errors. The primary data cache has the same high throughput (44.8 GB / s), but its size has been halved (8 KB versus 16 in Pentium III). In the usual sense, there is no primary instruction cache; it was replaced by the trace cache. It stores sequences of micro-operations in which instructions are decoded. It can accommodate up to 12K micro-instructions.

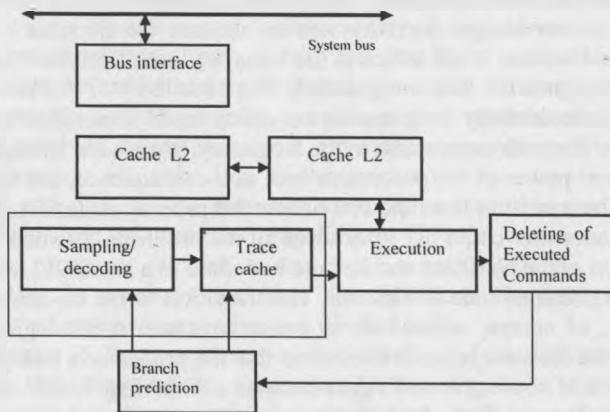


Fig. 6.4. NetBurst microarchitecture

The processor system bus interface is designed only for single-processor configurations; there is also no possibility of excessive functional control (FRC). The interface is much like the P6 bus, the protocol is also focused on the simultaneous execution of several transactions. In the Pentium 4 processor, the frequency of the bus is 400 MHz with "quad pumping" - the frequency of the system bus is 100 MHz, but the frequency of addresses and data transmission is higher. New information on lines with common synchronization can be transmitted on each cycle at a frequency of 100 MHz. For 2 and 4-fold transmission,

synchronization from the data source is used. Information is transmitted via the address bus in the 2-fold transmission mode; the strobes are two signals $ADSTB0\#$ and $ADSTB1\#$. The address is transmitted during the decline of these strobes, and along the edge - information about the type of transaction. Thus, in each bus cycle (for 10 ns), both the address and the type of transaction are transmitted (for P6, this required 2 cycles, which took 15-30 ns). Information is transmitted over the data bus at a quadruple frequency, for which a pair of gate signals $DSTBp [0: 3] \#$ and $DSTBn [0: 3] \#$ with a period of 5 ns (frequency 200 MHz) are used. The strobes are shifted relative to each other by 2.5 ns (half their small beat), synchronization by their decline gives a quadruple transmission frequency.

The capacity of the data bus, as in the previous two generations of processors, is 64 bits (8 bytes), which in the 4-fold transmission mode gives a maximum throughput of $100 \times 4 \times 4 = 3.2$ GB / s. For Pentium III processors, the bus provided $133 \times 8 = 1.06$ GB / s. The address bus has a digit of 36 bits, and this allows to address the same 64 GB of memory, of which only the first 4 GB are cached.

Actuators of MP (D) operate at double frequency, which makes it possible to execute most integer instructions in half a beat. Compared to previous generations of the IA-32, the Pentium 4 contains the longest command conveyor, consisting of 20 stages and is called the *hyper conveyor*.

Throughout the development of the Intel Pentium 4 family of processors, the main way to increase productivity was to increase the frequency. Actually, the NetBurst architecture itself, which is the basis for Intel Pentium 4 processors, was originally designed for frequency scaling. The peculiarity of this microarchitecture was an unprecedentedly long conveyor, which made it possible to increase the frequencies. But with an increase in the frequency, there were increased accordingly the consumed power of the processors and, as a consequence, the heat generation. And even the transition from the 130-nanometer process manufacturing processors to the 90-nanometer could not fully solve all the problems. It would seem that the development of the NetBurst architecture has come to a standstill, having stumbled upon the problem of heat dissipation. The transition to the 65-nanometer process technology, of course, will allow to create a certain technological reserve for increasing the frequency, but it is obvious that the processor's heat generation and impossibility of cooling it will again become a stumbling block. As a result, the frequency will most likely be increased, possibly even up to 5 GHz, but for the sake of a performance increase of only 20-30%, it is not advisable to spend billions of dollars to develop a new manufacturing process [20].

Of course, to say that increasing the frequency is the only way to increase the performance of processors with the NetBurst architecture would not be entirely correct. With each new version of the processor core, that is, with the transition to a new manufacturing process, small changes were also made to the microarchitecture of the core. So, the length of the conveyor gradually increased due to the addition of *Drive* gear ratios, which contributed to the possibility of a further increase in the frequency; L2 cache size increased, there were improved individual processor units. In addition, at the time, the NetBurst architecture was supplemented by *Hyper-Threading (HT technology)*. This technology increases

processor performance under certain workloads by providing “*useful work*” to *execution units* that would otherwise be inactive, for example, in cases of cache miss.

HT technology is a cross between medium processing implemented in multiprocessor systems and concurrency at the instruction level implemented in uniprocessor systems. From the point of view of the operating system and the running application, there are two processors in the system, which makes it possible to distribute the load between them. Using the parallelism principle implemented in HT technology, it is possible to process instructions in parallel mode (all instructions are divided into two parallel threads).

In terms of design, a processor with HT technology support consists of two logical processors, each of which has its own registers and an interrupt controller (*Architecture State, AS*). Thus, two parallel tasks run with their own independent registers and interrupts, but at the same time use the same processor resources to perform their tasks. After activation, each of the logical processors can independently from the other processor perform its task, process interrupts, or block. Therefore, the new technology differs from the real dual-processor configuration only in that both logical processors use the same executing resources, the same cache shared between the two threads and the same system bus. The idea of HT technology is closely related to the micro-architecture of the Pentium 4 NetBurst processor and is, in a sense, its logical continuation. Intel NetBurst microarchitecture allows to get the maximum performance gain at executing a single instruction stream, performing one task. However, even in case of special optimization of the program, not all executive modules of the processor turn out to be involved during each cycle. On average, executing code typical for the IA-32 instruction set, only 35% of the processor’s executive resources are actually used, and 65% of the processor’s executive resources are idle, which means inefficient use of processor capabilities [19]. It would be logical to implement the processor in such a way that in each cycle to maximize its capabilities. This is precisely the idea that HT technology implements by connecting unused processor resources to a parallel task.

The increase in frequency cannot be infinite and is determined by the manufacturing technology of the processor. At the same time, the increase in productivity is not directly proportional to the increase in the frequency, that is, a saturation trend is observed when a further increase in the frequency becomes unprofitable. The number of instructions executed during one cycle depends on the processor microarchitecture: on the number of execution units, on the length of the conveyor and its filling efficiency, on the prefetch block, on the optimization of the program code for the given processor microarchitecture. Therefore, a comparison of the performance of processors based on their frequency is possible only within the same architecture (with the same value of the number of operations per second).

The impossibility of effectively scaling the processor speed led to a gradual transition to a dual-core, and later to a multi-core architecture, see chapter 9.

The transition to a dual-core processor architecture actually allows to increase their performance, but with one caveat. This requires the use of applications that could be well parallelized, that is, would be initially oriented towards multiprocessing.

One solution to this problem is related to the implementation of the concept of "*parallelism at the level of threads (threads)*" - *TLP (Thread Level Parallelism)*. If program codes are not able to load all or even most functional devices with work, then you can allow the processor to perform more than one task so that additional threads load idle devices. Here it is easy to see an analogy with a multitask operating system: so that the processor does not stand idle when the task is in a wait state (for example, I/O completion), the operating system switches the processor to another task. Moreover, some scheduling mechanisms in the operating system (for example, quantization) have analogues in a multi-threading architecture (*MTA*). Obviously, an architecture that supports thread-level concurrency (*TLP*) must ensure that threads do not use the same resources at the same time, which requires additional hardware (for example, duplicate register files). Using the basic type of parallelism at the thread level in the microprocessor, you must have at least two hardware extensions for the threads. These are general purpose registers, a command counter, a process status word, etc. At any given time, only one thread (thread) is running. It is executed before a certain situation arises (for example, executing a register load command when there is no data in the cache). In this case, the processor switches to another thread. Since memory operations may require up to hundreds of processor cycles if they do not get into the cache memory, its downtime due to data waiting could be very significant. Modern processors that have the ability to speculative extraordinary execution of instructions can continue to execute other instructions in a similar situation, but in practice the number of independent instructions is quickly exhausted and the processor stops. Architecture with the simultaneous execution of threads - *SMT (Simultaneous Multi-Threading)* allows the simultaneous execution of multiple threads. In this case, at each new cycle, a command of any thread can be sent to an execution unit for execution. Compared with superscalar processors that support extraordinary speculative instruction execution and which use the register renaming mechanism, *SMT* requires, in particular, the following hardware:

- several command counters (one per stream) with the ability to select any of them at each cycle;
- tools associating commands with the flow to which they belong (necessary, in particular, for the operation of transition prediction mechanisms and register renaming);
- several stacks of return addresses (one per thread) for predicting return addresses from subprograms;
- special additional memory in the processor (per each thread) for the procedure of deleting out-of-order commands from the buffer.

One of the main features of *SMT* in many modern processors is the renaming of registers, when logical (architectural) registers are mapped to physical, with

which real work is being done. The technique of renaming registers can obviously be applied in order to avoid direct duplication of register files, as hardware flow.

In modern applications at any given time, as a rule, not one, but several tasks or several threads, also called threads, are executed. As an example, let consider the operation of two threads. When both threads are executed simultaneously, the processor will constantly switch between threads; in one processor cycle, only instructions from any one of the threads are executed. At each processor cycle, there are used not all processor execution units, so it is possible to partially combine the execution of instructions of individual threads on each processor cycle. For example, performing two arithmetic operations with integers of the first thread can be combined with loading data from the memory of the second thread and performing all three operations in one processor cycle. Similarly, on the second processor cycle, the operation of saving the results of the first stream can be combined with two operations of the second stream, etc. Actually, in such parallel execution of two streams lies the main idea of *HT* technology. Hyper-Threading is virtual multiprocessing, since the processor is actually one, and the operating system sees two processors. The classic "single core" processor has added another logical control unit *AS* (*Architectural State*) technology of IA-32.

AS monitors the status of registers (general purpose, control, interrupts - *APIC*, service) using a single physical core (branch prediction blocks, *ALDs*, *FPU*s, *SIMD blocks*, etc.) *AS1* is one logical processor (*LP1*), *AS2* - second logical processor (*LP2*). Each logical processor has its own Advanced Programmable Interrupt Controller (*APIC*) and a set of registers. For the correct use of shared registers by two logical processors, there is a special table - *RAT* (*Register Alias Table*), according to which you can establish a correspondence between general purpose registers of the general processor core. The table of *RAT* registers use for each logical processor is different. The result is a scheme in which two independent threads of program code can freely execute on the same core.

A processor with Hyper-Threading technology has two main operating modes:

- Single-Task (*ST*);
- Multi-Task (*MT*).

In *ST* mode, only one logical processor is active, which completely uses available resources, the other logical processor is stopped by the *HALT* command. When the second program stream appears, the idle logical processor is activated (by means of interruption), and the physical processor is put into operation with two threads, see Fig. 6.5.

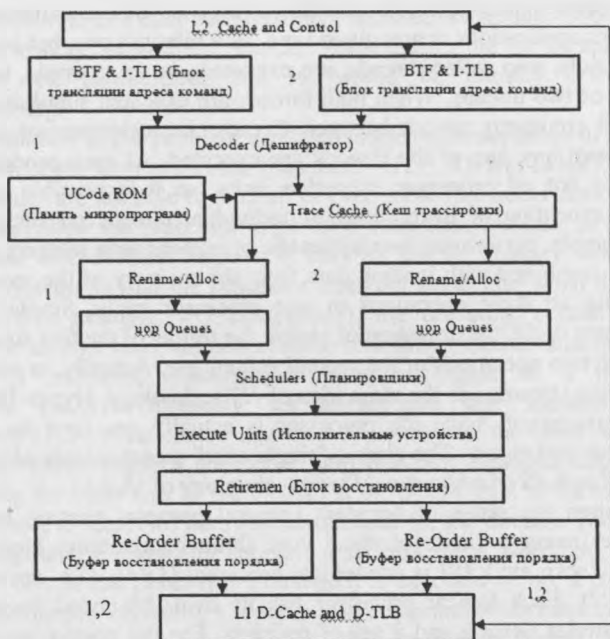


Fig. 6.5. Resource sharing by two logical processors

When two threads operate, two queues of instructions are supported (*Next Instruction Pointers*). Most of the instructions are taken from the *Trace Cache (TC)*, where they are stored in decoded form, and two active logical processors get access to the trace cache alternately, every cycle. At the same time, when only one logical processor is active, it gains exclusive access to the trace cache without interleaving on cycles. Similarly, access to the firmware memory (*Microcode ROM*) occurs. The instruction translation blocks (*ITLB - Instruction Translation Look-aside Buffer*) operate in the absence of the necessary instructions in the instruction cache and deliver the commands, each for its own stream. The instruction decoding unit (*IA-32 Instruction Decode*) is shared, and when decoding of instructions for both streams is required, it serves them one at a time (every cycle). The blocks of the queue of decoded commands (*Uop Queue*) and the commands block distribution by executive devices (*Allocator*) are divided in two, allocating half the elements for each logical processor. Five *Schedulers* process the queues of decoded instructions, regardless of whether they belong to any thread, and send instructions for execution to the necessary executive devices (*Execution Units*), depending on the willingness to execute commands and the availability of executive devices. Cache memory of all levels is completely shared between two logical processors, but to ensure the integrity of the data records in the data buffer (*DTLB - Data Translation Look-aside Buffer*) they are provided with descriptors in

the form of an identifier (ID) for each logical processor. The initial part of the Pentium 4 conveyor is responsible for supplying microoperations (decoded x86 instructions) to the executive part of the conveyor. Here, basically, is the place where blocks duplicate are found for each of the two logical processors. The trace cache (Trace Cache) contains already decoded instructions. Most of the commands in real processor operation are decoded in advance and are located in the trace cache. Trace Cache is not duplicated for each of the logical processors, but is shared between them. However, each logical processor has its own trace block, referencing the following instruction to execute. The instructions from the Trace Cache are selected in turn and become in the so-called sampling queue, also individual for both logical processors.

Then, the sorted microoperations come to the stage of determining the sequence of execution (*Scheduling*), where the sorting of the order of the instructions when they arrive at the executive devices is performed. Operations for scheduler blocks come on a rolling basis. If necessary, schedulers switch from queues of one logical processor to queues of another. At this stage, by the way, there is a final mixing of micro-operations coming from logical processors, for the possibility of their simultaneous execution. Since the machine registers of the physical processor at this point turn out to be rigidly tied to the registers of both logical processors, the execution of instructions really becomes possible without analyzing the ownership of the commands. After the execution stage, at which the processor does not distinguish logical processors, there follows a *Retirement* block, where the original order of instructions and their affiliation to each of the logical processors are restored. In this case, the *Re-Order Buffer* is divided in half between logical processors. Although the cache memory of the first and second levels is shared between logical processors, the *Data Translation Lookaside Buffer (DTLB)*, which compares the data addresses and their physical addresses, although it is shared between the processors, is added to the identifier of the processor to which each belongs from buffer lines. Therefore, the Hyper-Threading technology really allows to load the executive devices of the processor much stronger due to the simultaneous execution of two threads. However, it should be understood that the effect of this technique may not always be positive. Firstly, if the executed threads are similar in type to the executed instructions, there may not be a win at all, since one of the threads will completely occupy all the resources needed by the other thread. Downtime of other executive devices of the processor will not disappear from this. Secondly, a situation is possible when one of the threads can simply take up the resources needed by the other thread and expect, for example, data arrival. The operating system at the same time, being confident that there are two CPUs, will not take any action, in fact, the functioning of the processor will simply be blocked.

Pentium family processors have a number of architectural and structural features compared to previous Intel microprocessor models. The most characteristic of them are:

- *Harvard architecture with separation of instruction and data flows* by introducing separate internal cache blocks for storing instructions and data, as well as buses for their transmission;
- *superscalar architecture*, providing simultaneous execution of several commands in parallel actuators;
- *dynamic execution of commands* that implements a change in the sequence of commands, the use of an extended register file (renaming registers) and efficient branch prediction;
- *a double independent bus* containing a separate bus for accessing the L2 cache (runs at the processor frequency) and a system bus for accessing the memory and external devices (running at the system board frequency).

The main characteristics of the Pentium family processors are as follows:

- 32-bit internal structure;
- the use of the system bus with 36 address bits and 64 data bits;
- separate internal cache of the first level for commands and data with a capacity of 16 KB;
- support for a common cache of commands and data of the second level with a capacity of up to 2 MB;
- conveyor execution of commands;
- prediction of the direction of software branching with high accuracy;
- accelerated floating point operations;
- priority control at accessing memory;
- support for the implementation of multiprocessor systems;
- the availability of internal tools for self-testing, debugging and performance monitoring.

Questions for self-control

1. What are the main blocks and their functional purpose in the structure of IA-32?
2. What's new in the architecture of the Pentium III processor compared to the Pentium MMX?
3. What features does Net Burst architecture have?
4. What is the difference between Pentium IV CPU cache commands and all previous ones?
5. What registers make up the IA-32 software model?
6. What is the difference between the Intel-386 and the Intel-486?

Test questions

1. Features of 32-bit IA-32 processors:
 - A) 86+ - the opportunity is available, starting with 80286 processors;
 - B) 386+ - the opportunity is available, starting with Intel386 processors;
 - C) 486+ - the opportunity is available, starting with Intel486 processors;

D) P6 + - the feature is available on Pentium Pro, Pentium II, Pentium III and newer processors.

2. Which microprocessor was the first that include all the main blocks characterizing the architecture of the A-32?

- A) i 8086
- B) Pentium
- C) i 286
- D) i 486

3. The software model of the processor in the IA-32 architecture includes the following resources:

- A) integer extension register set (MMX)
- B) a set of floating point extension registers (XMM = SSE + SSE2)
- C) software stack
- D) I / O ports

4. What blocks of MPs with IA-32 architecture are used for page address translation?

- A) FPU pager
- B) embedded floating point processor
- C) TLB memory management unit
- D) page conversion unit MMU

5. What units are included in the MMU in MP with IA-32 architecture?

What blocks are included in the microprocessor i486?

- A) block counters
- B) cache
- C) program memory
- D) memory management unit

6. What buses does the processor interface not contain?

- A) I / O bus
- B) data bus information
- C) address buses
- D) control buses

7. What is meant by superscalarity?

- A) the presence of more than one pipeline for commands processing
- B) special scheme for commands processing
- C) the presence of one conveyor for commands processing
- D) fast command processing

8. How many conveyor devices were included in the MP 80286?

- A) 2
- B) 5
- C) 4
- D) 3

9. How many command prefetch buffers does the Pentium processor contain for predicting branching in a program?

- A) 2
- B) 5
- C) 4
- D) 3

10. The memory management device in the Pentium processor supports pages up to ...

- A) 4 MB
- B) 2 MB
- C) 10 MB
- D) 3 MB

11. What method of data processing did not define the term dynamic execution of a program?

- A) data flow analysis
- B) command flow analysis
- C) deep branch prediction
- D) leading order execution

12. What was first implemented in the Pentium II processor?

- A) dual independent bus architecture
- B) superscalarity
- C) cache
- D) elements of parallelization

13. Which processor contains the longest instruction pipeline, consisting of 20 stages and called a hyper conveyor?

- A) Pentium
- B) Pentium III
- C) Pentium Pro
- D) Pentium 4

14. What hardware is not needed for SMT?

- A) several command counters
- B) means associating commands with the thread to which they belong
- C) multiple return address stacks
- D) special registers for addressing commands

15. What does AS do?

- A) monitors the status of the registers
- B) monitors stacks
- C) monitors the condition of the buses
- D) monitors the state of the kernel

16. What architectural or structural feature in comparison with previous models of Intel microprocessors is not available in Pentium family processors?

- A) superscalar architecture
- B) dynamic execution of commands
- C) Harvard architecture with the separation of the flow of commands and data
- D) triple independent bus

17. How many basic operating modes are provided for a processor with technology?

- A) 3
- B) 1
- C) 2
- D) 4

18. How many crystals does the Pentium 4 processor have?

- A) 1
- B) 2
- C) 3
- D) 4

19. By the number of large integrated circuits (LIC) in a microprocessor complex, microprocessors are distinguished on:

- A) single-channel, multi-channel and multi-channel sectional
- B) single-address, multi-address, and multi-address sectional
- C) single-chip, multi-chip and multi-chip sectional
- D) single-bit, multi-bit and multi-bit sectional

20. Which of the devices that make up universal microprocessors are absent, as a rule, in single-chip microcontrollers?

- A) floating point processor
- B) flag register
- C) general register block
- D) internal cache

CHAPTER 7. AMD PROCESSOR ARCHITECTURE

The American company AMD (*Advanced Micro Devices, Inc.*) is currently the second largest manufacturer of x86 and x64-compatible processors.

The core of processors with AMD architecture uses different principles of its operation compared to Intel. In fact, Athlon is a RISC processor - in real time, the stream of CISC commands is converted to unified RISC commands, which are called MacroOps (mOP) according to AMD terminology.

K8 is a classic Harvard architecture processor, see fig. 7.1. K8 architecture is used in many modern AMD server, desktop and mobile processors (Opteron, Sempron, Athlon 64 and Athlon 64 X2). The effective length of the conveyor (time in cycles from the beginning of the execution of the instruction to the moment when the results of the execution will be written into RAM) varies from 10-12 stages (for integer, logical calculations and accesses to RAM) to 17 stages (floating-point calculations). The number of simultaneously executed instructions per cycle in the established mode is up to three; frequencies of commercially available processors - from 1.6 to 2.8 GHz.

The volume of L1 D-cache caches (for data) and L1 I-cache caches (for code) is fixed at 64 Kbytes. In K8, there is a common second-level cache ranging in size from 128 to 1024 KB. Cache of the third and lower levels is not provided.

The execution of instructions on the K8 conveyor begins with the instruction fetch unit. In one cycle, the block selects 16 bytes of data from the cache and extracts from them one to three x86 instructions - how many fit in the selected data. Since the average length of an x86 instruction is 5-6 bytes, as a rule, a block manages to select three instructions per cycle. To facilitate the decoding process, the instructions stored in L1 caches are tagged (i.e., the cache lines store information about how x86 instructions are distributed within this line). Along the way, using the branch prediction block in the same cycle, the address of the block from which sampling will begin in the next cycle is determined. Tagging is performed when data is fetched from the L2 cache to the L1 I-cache cache. On the second cycle of the conveyor, the selected one or three x86 instructions are distributed among the three instruction decoding blocks. The most complex instructions requiring decoding using the processor microcode are sent to the VectorPath decoder. More simple - in DirectPath decoders or in dual DirectPath Double. From this moment, the processor switches to work with internal micro-instructions (mOP).

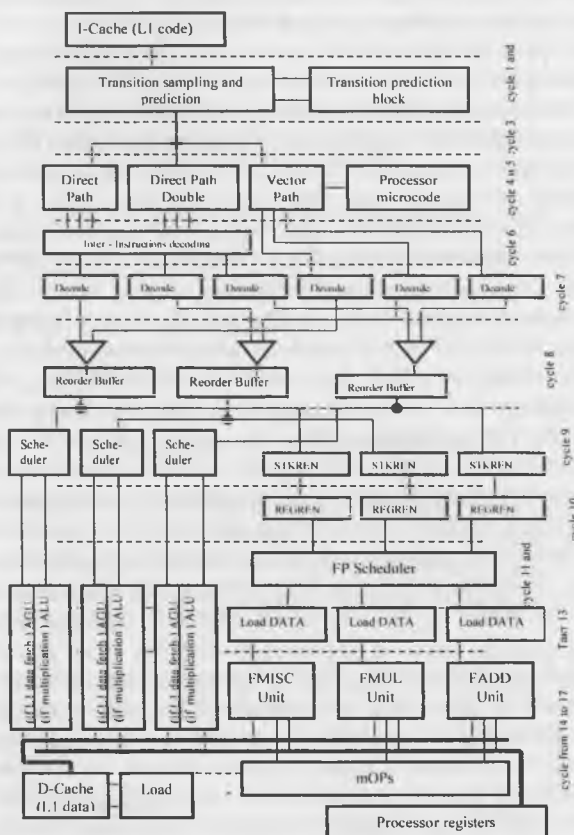


Fig. 7.1. The execution scheme of instructions in AMD K8

The whole further conveyor is created on the fact that work with mOP occurs in triples of instructions (AMD calls them lines, line). From a logical point of view, the K8 conveyor is constructed in such a way that it processes the lines, and not x86 - instructions or individual micro-operations. Moreover, in one line there can be less than three microoperations. In this case, the "lack" in the three is filled with special empty operations (*null-mOP*). At the same time, everything is elementary with "complex" vector instructions — the VectorPath decoder replaces the lines that are sewn in the processor microcode. But decoding "simple" instructions turns into a complex process of converting an x86 instruction into one (DirectPath) or two (DirectPath Double) mOPS, which are then packed into a single line by a special packer. Generated lines from VectorPath- and DirectPath-decoders, one cycle, go to a special device - Instructions Control Unit (ICU), where the lines prepared for execution are accumulated in a special queue (24 lines).

From a queue of 24 lines, three mOPs in each, ICU select one or three mOPs in the most convenient sequence and sends them to either ALU or FPU, depending on the type of micro-operation. In the case of ALD, microoperations immediately fall into the scheduler's queue (six elements of three mOPs each), which prepares the resources necessary for executing the microoperation, waits for their readiness, and only then sends the mOP together with all the necessary data for execution.

Moreover, executing one mOP, in fact, two actions can be performed at once - simple arithmetic calculations that often occur when accessing RAM (they are handled by the Address Generation Unit, AGU), and "complex" ones that require ALD intervention - the corresponding "two" microinstructions (ROP) is embedded in the mOP at the decoding stage. Preparation of data in the scheduler takes (ideally) one cycle, execution - from one (the vast majority of instructions) to three (when accessing RAM) and even five (64-bit multiplication) cycles.

In the FPU, things are a little more complicated. For starters, mOPs that have left ICU go through two stages in preparing their operands. Then they are accumulated in the FPU scheduler (twelve elements of three mOPs each), which is waiting until the data for these mOPs is ready and the actuators are released, and scatters the accumulated mOPs across three actuators. But unlike the integer part of the conveyor (which contains three identical ALDs and AGUs), FPU actuators are "specialized" - each one performs only its own specific set of actions on floating point numbers. Runtime: two cycles for renaming and displaying registers, one cycle (ideally) for planning and waiting for operands, four cycles for the actual execution.

The forecasting mechanism in AMD processors is more advanced than that of Intel's counterparts, in particular, not only the BTB table is used, but also the Global History Bimodal Counters table of transitions. And if BTB allows to predict transitions based on information about previous outcomes of this particular team, then GHBC allows to take into account at the stage of predicting the outcomes a certain number of conditional transitions completed before the current one.

Prediction of the return address is also used, a special buffer is used, in which the next EIP is entered after CALL fetch. At sampling the ret processor retrieves the return address from its buffer, and not from the stack (attempts to overwrite it are tracked).

The integer device in AMD processors is as follows, see fig. 7.2

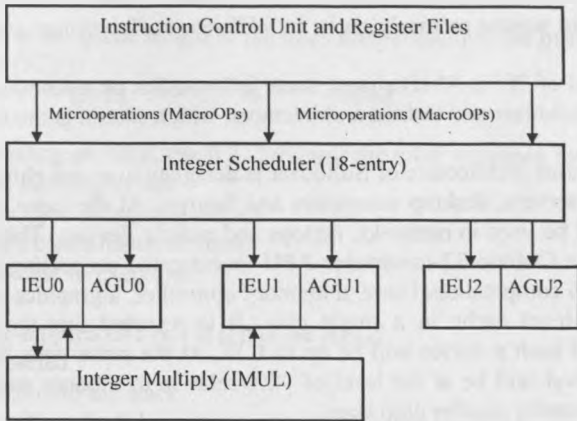


Fig. 7.2. Integer block in AMD processors

One of the reasons why AMD processors often outperform Intel counterparts in FPU tests is that they have an "advanced" block of floating-point operations.

This is more than just a block for FPU operations, in fact, it is a separate processor. There are used a separate register renaming unit, its own scheduler and its own executive devices.

Intel, although using different function blocks, FPU registers have the same status as IEU registers and are served by the same RAT (RAT - Register Alias Table) and ROB (ROB - ReOrder Buffer).

Here we have full isolation from the integer device, see fig. 7.3.

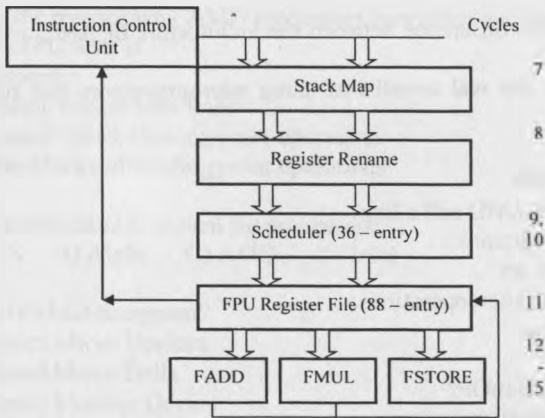


Fig. 7.3. Floating point block in AMD processors

After performing microoperations, LSU (Load / Store Unit) is turned on. Result buses are wound precisely on it. In addition, buffer strings can be used as operands.

At the end of 2010, AMD shared some information on upcoming processors based on the Bulldozer and Bobcat architectures, which should get to the market in 2011.

The processor architecture of Bulldozer is designed to create chips oriented to use as part of servers, desktop computers and laptops. At the same time, Bobcat processors will be used in netbooks, nettops and mobile devices. The first Bobcat chip will be the Ontario 32-nanometer APU (accelerated processing unit), it will receive one x86 computational core, a memory controller, a graphics core and 512 KB of second-level cache in a single chip. It is reported that the core power consumption of such a device will be up to 1 W. At the same time, the processor performance level will be at the level of 90% relative to modern mass solutions, but with significantly smaller chip sizes.

The Bulldozer architecture will debut in the server segment within the Interlagos and Valencia Opteron platforms. The first processors will also be manufactured according to 32-nanometer process standards. They will receive a modular structure, with each module including: one Float Point Scheduler, two Integer Scheduler units with their own first level cache for each block, and a common second level cache. At the same time, an increase in the number of used cores by 33% and an increase in performance by 50% compared with the processors of the previous generation are announced.

Questions for self-control

1. What are the main blocks and their functional purpose in the structure of AMD processors.
2. What is the difference between the architecture of AMD processors from Intel?
3. What are the real benefits of using microprocessors that support x86-64 technology?

Test questions

1. What does AMD call a line?
 - A) three instructions
 - B) command set
 - C) individual microoperations
 - D) instructions
2. What is null-mOP?
 - A) special empty operations
 - B) empty data
 - C) empty line
 - D) all answers are correct

3. What is the queue length of the lines accumulated in the Instructions Control Unit (ICU)?

- A) 24
- B) 12
- C) 36
- D) 40

4. Depending on what, the ICU forwards the mOP sequence for execution?

- A) from standing in line
- B) on the type of microoperation
- C) from a combination of lines
- D) from ease of execution

5. What happens to a line if it hits the ALD?

- A) performed
- B) pushes onto the stack
- C) enters the scheduler queue
- D) no correct answers

6. Which block performs operations on floating point numbers?

- A) FPU
- B) ALU
- C) AGU
- D) ROP

7. What is the forecasting mechanism in AMD processors in comparison with analogs from Intel?

- A) advanced
- B) at the same level
- C) lower level
- D) missing

8. One of the reasons why AMD processors very often outperform Intel counterparts in FPU tests is

- A) more speed
- B) processing occurs with lines
- C) "advanced" block floating point operations
- D) has two blocks of floating point operations

9. RISC architecture is typical for processors:

- A) CYRIX
- B) Alpha
- C) AMD
- D) Intel

10. How is AMD decrypted?

- A) Advanced Micro Devices
- B) Advanced Micro Dells
- C) Advanced Monitor Device
- D) All Micro Device

CHAPTER 8. PROCESSOR OPERATING MODES

For the first time, various modes of operation of Intel x86 processors began to be discussed with the advent of the processor 80286. This was the first representative of this processor family in which multitasking and a secure architecture were implemented [3, 12, 13, 15]. In order to ensure compatibility with previous representatives of this family (8086, 8088, 80186), two operating modes were implemented in the 80286 processor: 8086 emulation mode (real address mode) and protected mode, which uses all the processor capabilities. In subsequent generations of processors of this family, the protected mode becomes the main mode of operation. For processors, starting with 80386, sometimes they talk about another mode of operation - virtual mode (virtual mode 8086). In this manual, it will be considered as a special state of the task of the protected regime.

In new generations of Intel processors, another mode of operation has appeared - the System Management Mode. It was first implemented in the 386SL and 486SL processors. Starting with the extended 486x processor models, this mode has become an indispensable element of the architecture of x86-compatible processors. With its help, energy-saving functions are implemented transparently even for the operating system at the BIOS level.

The features of the above MP operating modes are discussed below.

Real Mode

After initialization (system reset), the MP is in real mode. In real mode, MP works like a very fast 8086 with the ability to use 32-bit extensions. The addressing mechanism, memory sizes, and interrupt handling (with their sequential limitations) of the MP 8086 completely coincide with similar functions of other IA-32 MPs in real mode.

Access to the memory in real mode is carried out by the “*segment: displacement*” design, which describes the logical address. The value of the segment, as well as the displacement, lies in the range from 0 to 0FFFFh.

Since it is possible to address only within one segment, the maximum segment size is 64 kilobytes. The physical address that is set on the address bus of the processor is calculated by the formula:

$$\text{linear address} = \text{segment} * 16 + \text{displacement}$$

Unlike 8086, other members of the IA-32 family throw exceptions in certain situations, for example, when a segment limit is exceeded, which for all segments in real mode is 0FFFFh.

There are two fixed areas in memory that are reserved in real addressing mode: the system initialization area and the interrupt table area.

Cells 00000h through 003FFh are reserved for interrupt vectors. Each of the 256 possible interrupts has a reserved 4-byte hop address. Cells FFFFFFF0h through FFFFFFFFh are reserved for system initialization.

System Management Mode

In new generations of Intel MP, another mode of operation has appeared - the system management mode. It was first implemented in MP 80386SL and i486SL. Starting with the extended Intel-486 models, this mode has become a mandatory element of the IA-32 architecture. With its help, energy-saving functions are implemented transparently even for the operating system at the BIOS level.

System management mode is designed to perform some actions with the possibility of their complete isolation from application software and even from the operating system. The MP switches to this mode only in hardware: at a low level on the SMI # pin or on command from the APIC bus (Pentium+). No software method is provided for entering this mode. The MP returns from the system control mode to that mode during operation in which the signal SMI # was received. Return occurs at the command of RSM. This command only works in system control mode and is not recognized in other modes, throwing exception #6 (invalid operation code).

When the MP is in SMM mode, it sets the signal SMI_{ACT} #. This signal can serve to enable a dedicated area of physical memory (System Management RAM), so that SMRAM can only be made available for this mode. Upon entering SMM mode, the MP saves its context in SMRAM (the coprocessor context is not saved) and transfers control to a procedure called the System Management Interrupt handler. The state of the MP at this moment is precisely defined: the *EFLAGS* register is reset (except for the reserved bits), the segment registers contain the selector *0000*, the segment bases are set to *00000000*, the limits are *0FFFFFFFh*.

It should be noted that in the SMM mode it is not possible to work with interrupts and special cases: IRQ and SMI # interrupts are masked, step traps, and stop points are disabled, the NMI interrupt processing is delayed until SMM mode is exited. If you need to work with interrupts or special cases, then you need to initialize the IDT and enable interrupts by setting the IF flag in the *EFLAGS* register. NMI interrupts will be unlocked automatically after the first *IRET* command.

When returning from SMM (according to RSM instructions), the MP restores its context from SMRAM. The processor can programmatically make changes to the MP context image, then the MP will go into the wrong state in which the SMI occurred.

These features of the system management mode allow to use it to implement a computer energy management system or security and access control functions.

Protected Mode

Protected mode - the main mode of MP operation. Key features of protected mode: virtual address space, protection and multitasking. The MP can be put into protected mode by setting bit 0 (Protect Enable) in the *CR0* register. The MP can return to the real address mode by a RESET signal or by resetting the PE bit (not available in Intel-286).

This mode has a complex structure compared to the real one. The logical address is represented by the construction "*selector:displacement*". The selector is

in the range from 0 to 0FFFFh (in fact, there are 4 times less selectors). The displacement, in contrast to the real mode, is 32-bit, which allows addressing segments of 4 gigabytes. The logical address is converted to linear according to the following scheme:

$$\text{linear address} = \text{segment base} + \text{displacement}$$

The linear address is subsequently set to the address bus if paging mode is not enabled. Otherwise, the linear address is converted to a physical one, and only after that it is set on the address bus. In addition, protected mode allows to organize virtual memory up to 64 terabytes in size and depending only on the size of the hard disk (for example, the same swap file in Windows implements virtual memory). In protected mode, almost all modern operating systems function.

To fully switch to protected mode with minimal settings, you need to perform the following steps:

1. check whether it is possible to switch to protected mode;
2. initialize descriptor tables;
3. prohibit interruptions (both masked and not masked);
4. open the line A20;
5. load memory management registers;
6. set the zero bit (hereinafter referred to as PE) of the CR0 register;
7. transition to the 32-bit code segment by overriding the CS register.

Disabling interrupts prevents us from rebooting. Interruptions are divided into either maskable or non-maskable. To disable masked interrupts, you must clear the *IF* flag of the *EFLAGS* register with the *cli* command, while interrupts are resolved with the *sti* command. Non-maskable interrupts are prohibited slightly differently. There are two ways to do this: programming the registers of the interrupt controller (this method will be discussed a bit later) or changing the seventh bit of *70h* port: if the bit is set, then interrupts are disabled, if the bit is reset, interrupts can be performed.

Line A20 is one of 32 address lines. When starting up the computer, the A20 line is closed. This leads to the generation of 20-bit addresses (that is, the entire address space is equal to $(2^{20})=1$ megabyte). This was introduced for compatibility with the 8086 processor: thus, trying to write to the linear address 12345678h, we actually write to the address 00045678h, which can lead to a completely unexpected result. Therefore, for the full functioning of the 32-bit application, the A20 line must be open. This is done by setting bit 1 of port 92h, closing the A20 line - reset this bit.

Example.

```
use16
org 100h
start:          ; We are in real mode
cli             ; Prevent maskable interrupts
               ; Prevent Non-Maskable Interrupts (NMI)
```

```

in  al, 70h
or  al, 80h
out 70h, al
                                ; Open line A20

in  al, 92h
or  al, 2
out 92h, al

                                ; Switch to protected mode
mov  eax, cr0
or  al, 1
mov  cr0, eax
                                ; Now we are in protected mode
                                ; Small double cycle

mov  cx, 20
cycle:
mov  ax, cx
mov  cx, 0ffffh
loop $
mov  cx, ax
loop cycle
                                ; Switch to real mode

mov  eax, cr0
and  al, 0feh
mov  cr0, eax
                                ; Close line A20

in  al, 92h
and  al, 0fdh
out 92h, al
                                ; Allow Non-Maskable Interrupts (NMI)

in  al, 70h
and  al, 7fh
out 70h, al

                                ; Allow maskable interrupts
sti ;*
                                ; We are in real mode again
ret
                                ; complete the program

```

In protected mode, the program operates with addresses that may refer to physically absent memory cells, therefore this address space is called virtual. The size of the virtual address space of the program can exceed the capacity of physical memory and reach 64 TB.

Conversion of a logical address into a physical one takes place in two stages. First, the segment control unit translates the address in accordance with the segmented memory model, obtaining a 32-bit linear address. The paging unit then paginates, converting the 32-bit linear address to a 32-bit or 36-bit (P6) physical address. The MP does not provide for mechanisms to prohibit segmentation; on the

other hand, page translation is an optional mechanism, and may or may not be used depending on the features of the operating system.

In a segmented addressing model for a program, memory is represented by a group of independent address blocks called segments. To address a memory byte, the program must use a logical address consisting of a segment selector and an offset. The segment selector selects a specific segment, and the offset indicates a specific byte in the address space of the selected segment. The segment selector can be either directly in the command code, or in one of the segment registers. The displacement can either be directly in the command code, or calculated based on the values of general purpose registers.

Each segment is associated with a special structure that stores information about it: a descriptor. A descriptor is an 8-byte unit of descriptive information that is recognized by the memory management device in protected mode, stored in a descriptor table. The segment descriptor contains the base address of the described segment, the segment limit, and access rights to the segment. In protected mode, segments can start from any linear address (which is called the base address of the segment) and have any limit up to 4 GB.

Descriptor tables are variable-length memory arrays containing 8-byte elements: descriptors. A descriptor table can have a length of 8 bytes to 64 KB and each table can have up to 8192 descriptors. There are two required descriptor tables, see chapter 5 - the **global descriptor table (GDT)** and the **interrupt descriptor table (IDT)**, as well as many (up to 8191) optional **local descriptor tables (LDT)**, of which only one is available to the processor at any given time. The location of the descriptor tables is determined by the registers of the processor **GDTR, IDTR, LDTR**.

GDT contains descriptors available to all tasks in the system. GDT can contain any type of descriptor: both segment descriptors and system descriptors (except interrupt gateways and traps). The first GDT element (with a zero index) is not used. It corresponds to a null selector denoting an "empty" pointer.

LDTs provide a way to isolate program segments and executable task data from other tasks. LDT is associated with a specific task and may contain only segment descriptors, call gateways, and task gateways.

A segment cannot be accessed by the task if its handle does not exist in the current LDT table or in the GDT table. The use of two descriptor tables allows, on the one hand, to isolate and to protect the segments of the executable task, and on the other hand, allows to share global data and code between different tasks.

An IDT can only contain task gateways, interrupt gateways, or trap gateways. To calculate the linear address of the MP performs the following steps, see Fig. 8.1.

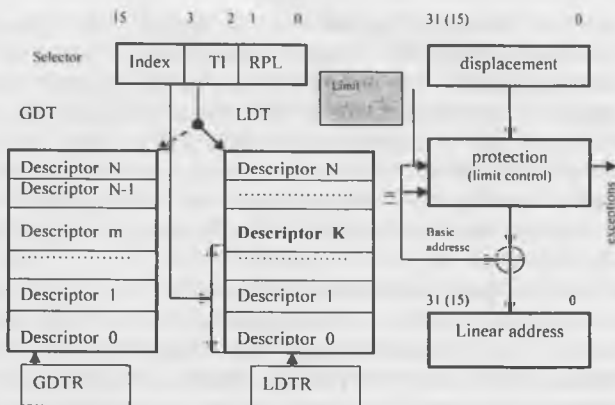


Fig. 8.1. Logical to linear address transition scheme

MP uses a segment selector to find a segment descriptor. The selector contains the descriptor index in the descriptor table (Index), the TI bit that determines which descriptor table is being accessed (LDT or GDT), as well as the requested segment access rights (RPL). If the selector is stored in the segment register, then access to descriptor tables occurs only when the selector is loaded into the segment register, since each segment register stores the corresponding descriptor in a software-inaccessible ("shadow") cache register.

The MP analyzes the segment descriptor, controlling access permissions (the segment is accessible from the current privilege level) and the segment limit (the displacement does not exceed the limit);

MP adds an displacement to the base address of the segment and receives a linear address.

If paging is disabled, the generated linear address is considered physical and is exposed on the processor bus to perform a read or write memory cycle.

The segmentation mechanism provides excellent protection, but it is not very convenient for implementing virtual memory (swap). There is a presence bit in the segment descriptor, according to which the processor determines whether this segment is in physical memory or on an external storage device (on the hard drive). In the latter case, exception # 11 is generated, the handler of which can load the segment into memory. The disadvantage is that different segments can have different lengths. This can be avoided if the swap mechanism is implemented based on *pagination*. A feature of this transformation is that the processor in this case operates with blocks of physical memory of equal length (4 Kbytes) - pages. Pages are not directly related to the logical structure of the program. In addition, in the P6 subfamily MP, *page translation* provides 36-bit physical memory addressing (64 GB). Page conversion is effective only in protected mode and is enabled by setting 1 bit PG in register CR0.

Two types of structures are involved in *paging*: table directories (*Page Directory*) and page tables (*Page Table*). These structures consist of 1024 32-bit elements. Elements contain the highest 20 bits of the physical address of the addressed objects. *Page Table Entry* (PTE) elements address pages, and table directory elements (*Page Directory Entry - PDE*) address page tables. The upper 20 bits of the physical address of the table catalog are stored in the CR3 register (*Page Directory Base Register - PDBR*) (this is the only register of the MP that contains the physical memory address). All structures are aligned on the page border, see fig. 8.2.

In the process of page translation of addresses, the resulting linear address is divided into three parts. The top ten bits (Directory) of the linear address are the index of the item from the table catalog. This element determines the physical address of the page table. Bits 21-12 (Table) of the linear address select an element from this page table. The selected item determines the physical address of the page. The lower 12 bits (displacement) of the linear address determine the displacement from the beginning of the page.

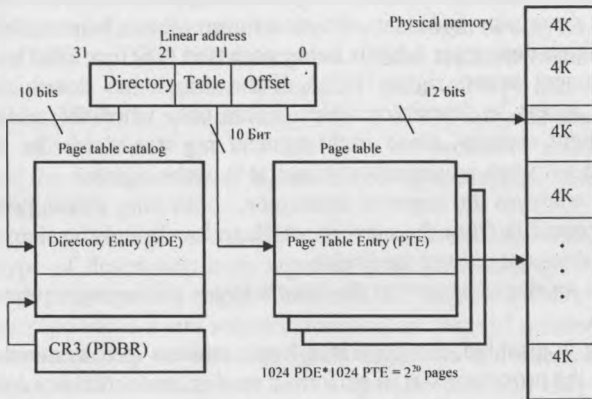


Fig. 8.2. The standard 2-level diagram of a broadcast paging

Pages start at 4 KB of memory, so the lower 12 bits of the page address are always zero. In the table catalog, items store the physical addresses of the page tables. In the page table, elements store the physical addresses of the pages themselves.

In Pentium MP, Intel has implemented a new feature - *Page Size Extension (PSE)*. PSE allows the use of 4 MB pages and a single-level paging mechanism. In the P6 subfamily, the address bus has been expanded to 36 bits. Accordingly, processors Pentium Pro, Pentium II, Pentium III and later are capable of addressing up to 64 GB of physical memory. This feature is called the Physical Address Extension (PAE) and is only available when using page translation. Built-in task switching provides multitasking in protected mode.

In the P6 subfamily, the address bus has been expanded to 36 bits. Accordingly, processors Pentium Pro, Pentium II, Pentium III and later are capable of addressing up to 64 GB of physical memory. This feature is called the Physical Address Extension (PAE) and is only available when using page translation.

Internal task of switching provides *multitasking* in protected mode.

A **task** is a "unit" of tasks for the processor that the processor can execute, pause and dispatch on it. As a task, an application program, an operating system service, an operating system kernel, an interrupt or exception handler, etc. can be executed. In protected mode, the IA-32 architecture provides a mechanism for saving the state of a task and switching from one task to another. All processor instructions are executed in the context of a particular task. Even the simplest systems must define at least one task. More complex systems can use task management tools to support multi-tasking applications.

The task environment consists of the contents of the MP registers and the entire code with data in the memory space. MP can quickly switch from one runtime to another, simulating the parallel operation of several tasks. For some tasks, memory management can be emulated, like MP 8086. This state of the task is called Virtual 8086 mode (Virtual 8086 Mode). The presence of a task in this state is signaled by the VM bit in the flag register. At the same time, the tasks of the virtual MP 8086 are isolated and protected, both from each other and from ordinary tasks of the protected mode.

The task consists of two components: the address space of the task and the *task state segment (TSS)*.

The task address space includes the segments of code, data, and stack available to it. If the privilege mechanism is used, then each task should be provided with stack segments for all *privilege levels* used.

The task state segment stores the state of the registers (context) of the processor:

- state of segment registers (segment selectors that form the task address space);
- state of general purpose registers;
- flag register state (EFLAGS);
- next command pointer (EIP);
- value of the CR3 register (PDBR);
- LDTR register value.

In multi-tasking systems, TSS provides a mechanism for linking (nesting) tasks.

Each task is identified by a selector for its corresponding TSS. This selector is loaded into the *Task Register (TR)* when switching to the task. The base address, limit, and TSS attributes are loaded into the shadow part of the register. The operating system can provide for each task its own linear address space (its own set of pages for page conversion), then when switching the task, the *CR3 register (PDBR)* is also stored, which stores the address of the table catalog for page conversion.

In protected mode, the processor provides certain protection mechanisms based on segmentation and based on paging. Security mechanisms allow you to restrict access to specific segments or pages using privilege levels (4 for segments and 2 for pages). For example, critical code and operating system data may be located at a more privileged level than application programs. This will limit and control the access of application programs to the functions and data of the operating system.

The protection mechanism ensures that any reference to memory cells matches certain conditions. All checks are performed before the start of the memory access cycle. Violation of any condition leads to the generation of an exception. Checks are performed in parallel with the formation of the address and therefore do not impair processor performance. All links must pass the following checks:

- limit control;
- type control;
- control of privilege level;
- alignment control;
- limitation of address space;
- restriction of entry points to procedures (for gateways);
- limitation of a set of commands (privileged instructions).

In protected mode, there is no way to disable the protection mechanism. Even if you assign all segments and tasks a zero (highest) privilege level, checks for limit and type control will still be performed. At the same time, the page-level security mechanism can be suppressed by assigning all pages the superuser privilege level and providing read and write access.

The control of the limits and types of segments ensures the integrity of code and data segments. The program does not have the right to access virtual memory that goes beyond the limits of a particular segment. The program does not have the right to refer to the data segment as a code, and vice versa.

The MP protection architecture provides 4 hierarchical privilege levels, which allows to restrict the task access to individual segments depending on its current privileges. Privileges are a property (usually set during system design) that determines which computer operations are allowed at any given time and which memory accesses are legal. Privileges are used to ensure security in a computer system. Privileges are implemented by assigning a value from 0 to 3 to key objects that are recognized by the processor. A value of 0 represents the highest privilege, while a value of 3 represents the least privilege.

Four privilege levels can be interpreted as rings of protection, see fig. 8.3 [12, 13, 15]. The center (level 0) is intended for segments containing the most critical programs (usually the kernel of the operating system). Outer rings are for segments with less critical programs or data. Using all four privilege levels is not necessary. Existing systems designed with fewer levels can simply ignore other acceptable levels. UNIX and Windows, for example, use only two privilege levels: 0 (for the system core) and 3 (for everything else), and OS / 2 uses levels 0 (for the system core), 2 (for I / O procedures) and 3 (for application programs).

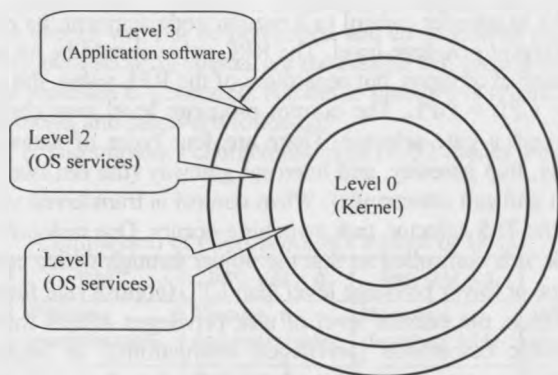


Fig. 8.3. Interpretation of privilege levels in the form of rings of protection

The microprocessor privilege level control mechanism operates with the following values:

CPL - Current Privilege Level: the privilege level at which the task is currently running. The CPL value is stored in the RPL field of the code segment selector, which is placed in the CS register. Typically, this value corresponds to the privilege level of the descriptor of the executable code segment. The privilege level changes when control is transferred to a code segment with a different DPL value (excluding subordinate code segments).

DPL - Descriptor Privilege Level: The least privileged level at which a task can access the segment or gateway associated with this descriptor. The DPL level is determined by bits 46 and 45 of the descriptor.

RPL - the Requested Privilege Level is used to temporarily lower your privilege level when accessing memory. RPL is entered in the lower bits of the selector.

The privilege level control mechanism usually compares the descriptor privilege level (DPL) with the maximum of the two numbers CPL and RPL. The least privileged of the current privilege level and the requested one is considered the effective privilege level: $EPL = \max(CPL, RPL)$.

Privileges for accessing data are controlled when the selector is loaded into the segment register *DS*, *ES*, *FS*, *GS* (or when accessing the memory if the selector is contained in the instruction code). A program can access a data segment that is at the same or lower privilege level (taking into account RPL), i.e. access to data is allowed if $\max(CPL, RPL) \leq DPL$; otherwise, a breach of general protection is generated.

Privilege control accessing the stack is performed when the selector is loaded into the SS register. A program must use a stack segment that is at the same privilege level, i.e. $CPL = RPL = DPL$

In order to transfer control to a regular code segment, its privilege level must match the current privilege level. The RPL value should be no more than CPL, so as not to cause exceptions, but regardless of the RPL value, the privilege level will not change: $CPL = DPL$. The current privilege level may change at transferring control through a gate selector. There are four types of gateways: call gateway, task gateway, trap gateway, and interrupt gateway (the last two types are not used in transition and call commands). When control is transferred via the task gateway selector or the TSS selector, task switching occurs. One task can transfer control to another task, it is controlled so that the object through which control is transferred is at the same or lower privilege level than CPL (control rule for data segments).

In addition, the current level of task privileges affects the ability to execute certain specific commands (privileged instructions). In addition to privileged instructions, there are instructions whose execution depends on the IOPL field in the flag register (*I/O privilege level*): IN, INS, OUT, OUTS, CLI, STI. The processor protection mechanism allows to follow these instructions only if the task has sufficient privileges, i.e. $CPL \leq IOPL$.

The paging features first introduced in the Intel 386 provide additional page-level security mechanisms. This is especially convenient when using a solid memory model, when both the operating system and application programs work in a single space of logical addresses, because page-level security can provide privilege separation between operating system pages and application programs. Page-level protection provides two types of control: address space limitation (supervisor pages and user pages) and access type restriction (read-only access and read-write access). If the checks fail, a page violation is generated (exception # 14).

Limiting address space at the page level is provided by two levels of privileges: supervisor mode (level 0) and user mode (level 1). Supervisor mode corresponds to privileges at the segment level $CPL = 0$, $CPL = 1$ and $CPL = 2$. In supervisor mode, all pages are available. User mode corresponds to privileges at the segment level $CPL = 3$. In user mode, only user pages are available. The address space of the supervisor includes pages for which the corresponding element of the page table or table catalog contains the bit $US = 0$. The user's address space contains pages for which both the corresponding element of the page table and the corresponding element of the table catalog contain the bit $US = 1$.

In Intel-386 MP, the type of access restriction applies only to the user's address space. The program in user mode has the right to change only those pages for which both the corresponding element of the page table and the corresponding element of the table catalog contain the bit $RW = 1$. If for any page the element of the page table or the element of the table catalog contains the bit $RW = 0$, then the page is read-only. In supervisor mode, all pages are available for reading and writing.

When both types of protection are used both at the page level and at the segment level, the processor first performs segment protection checks, and only if successful, page protection checks. I.e. if access to the memory is rejected by protection at the segment level, a violation of general protection will be generated, and page protection checks will not be performed and an additional page violation will not occur. If segment security checks are successful, but page protection rules are violated, a page violation is generated. Access to memory is granted only when all the rules for segments and pages are followed.

In the table. 8.1. The summary characteristics of IA-32 modes are presented.

Table 8.1
Comparison of microprocessor modes of IA-32

Characteristics	RM	PM, VM = 0	PM, VM = 1	SMM
Linear address formation	Without a descriptor table	Through a descriptor table	Without a descriptor table	Without a descriptor table
Segment limits	64 kb	Defined by a descriptor	64 kb	4 Gb
Default address/data size	16 bits	Defined by a descriptor	16 bits	16 bits
Maximum amount of available memory (virtual)	~1 Mb	~64 Tb	~1 Mb	~4 Gb
Protection	no	yes	yes	no
Page Broadcast	no	yes	yes	no
Multitasking	no	yes	yes	no
Interrupt handling	Vector table	Descriptor table	Descriptor table	no

Questions for self-control

1. In what modes can IA-32 work?
2. How is the physical address formed during segmented addressing?
3. How is the physical address formed by paging?
4. What is *multitasking*? By what means is it supported?
5. What privilege-based rules are used to protect code segments, stacks, and data?

Test questions

1. Which processor family was the first to support full 32-bit protected mode?
 - A) 80486
 - B) 80386
 - C) 80286
 - D) 8086/8088

2. What processor mode does not exist?
 - A) real mode
 - B) protected mode
 - C) system control mode
 - D) remote mode

3. When is the microprocessor in real mode?
 - A) after initialization (system reset)
 - B) constantly
 - C) upon reboot
 - D) after user action

4. Why are cells from FFFFFFF0H to FFFFFFFFH reserved?
 - A) for interrupt vectors
 - B) for microcommands
 - C) to initialize the system
 - D) for flags

5. Using which mode at the BIOS level are energy-saving functions implemented?
 - A) real mode
 - B) protected mode
 - C) system control mode
 - D) all modes

6. How can I switch to system management mode?
 - A) programmatically
 - B) after initialization (system reset)
 - C) hardware
 - D) upon reboot

7. How is the interrupt handling in the system control mode?
 - A) an error is issued
 - B) as in other modes
 - C) processed first
 - D) is delayed until exiting

8. What mode is the main mode of operation of the microprocessor?
 - A) real mode
 - B) protected mode
 - C) system control mode
 - D) user set

9. What does not belong to the key features of the protected mode?

- A) virtual address space
- B) protection
- C) multitasking
- D) multitasking

10. ... is an 8-byte unit of descriptive information recognized by the memory management device in protected mode

- A) descriptor
- B) segment
- C) address
- D) displacement

11. What does the Global Descriptor Table (GDT) not contain?

- A) type descriptors
- B) segment descriptors
- C) system descriptors
- D) interrupt and trap gateways

12. What does the Interrupt Descriptor Table (IDT) not contain?

- A) segment descriptors
- B) call gateways
- C) task gateways
- D) segment descriptors

13. ... is the "unit" of tasks for the processor that the processor can execute, pause and dispatch on it.

- A) task
- B) micro command
- C) interruption
- D) exception

14. How to disable the protection mechanism in protected mode?

- A) impossible to disconnect
- B) special team
- C) overload
- D) using the flag

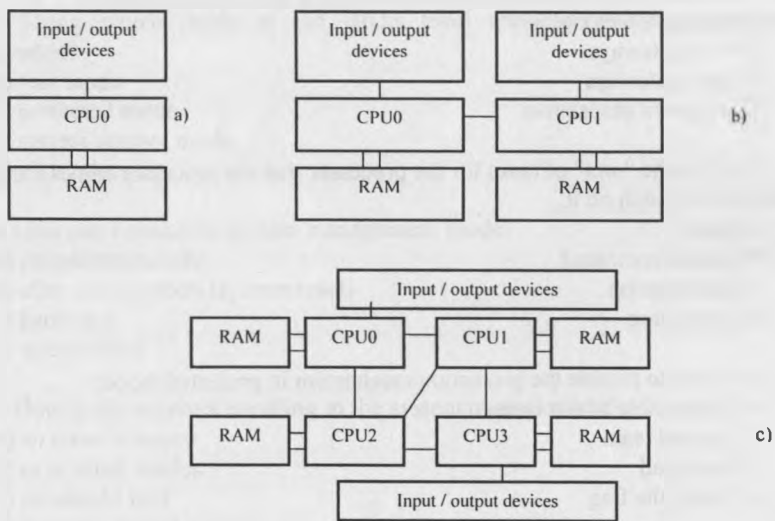
15. How many privilege levels does the MP protection architecture provide?

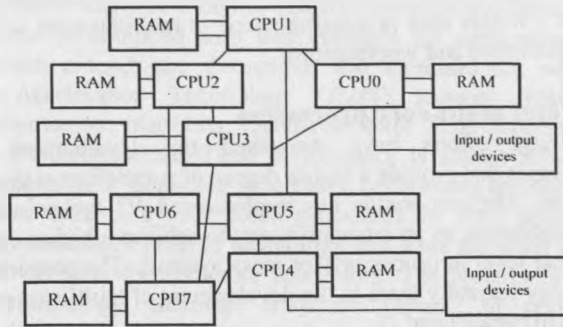
- A) 4
- B) 3
- C) 0
- D) 2

CHAPTER 9. MULTI-CORE PROCESSORS

The development of more sophisticated processor architectures containing a larger number of functional actuators, in order to increase the number of commands simultaneously executed in one cycle, is a traditional way to increase productivity alternative to increasing the frequency. But such developments are very complex and expensive. Development complexity increases exponentially with increasing of logic complexity. We can say that the idea of multi-core microprocessors creation is a development of the idea of clusters, but in this case the entire processor core is duplicated [12, 13, 15]. Another predecessor of the multi-core approach can be considered Intel technology - *HyperThreading*, in which there was a slight duplication of equipment and the use of two threads of instructions that use a common core. A multi-core processor has two or more "execution cores (CPU 0, CPU 1, ..., CPU n)", see fig. 9.1.

The operating system considers each of the executive cores as a discrete processor with all the necessary computing resources. Therefore, the multi-core processor architecture, with the support of the corresponding software, implements the completely parallel execution of several program threads.





a) – single processor system (1P); b) - 2P; c) - 4P; d) - 8P

Fig. 9.1. Multicore processor circuits

By 2006, all leading microprocessor developers created dual-core processors. The transition to the multi-core processors has become the main direction of increasing the performance of computing systems. In this regard, knowledge of the basics of the functioning of computing systems on multi-core processors is relevant.

The first appeared dual-core *RISC*-processors of Sun Microsystems (UltraSPARC IV), IBM (Power4, Power5) and HP (PA-8800 and PA-8900). AMD and Intel announced the release of dual-core processors with x86 architecture almost simultaneously [20].

In order to implement the process of parallel tasks execution, it is more efficient to integrate two cores or more in one microprocessor. Such a multi-core configuration on a single chip provides a higher exchange speed between the cores than the use of external buses, switches, etc. in multiprocessor systems. Multi-core technology and the use of 65 nm technology have allowed significant energy savings and increased productivity by 1 W of power consumption. Together with multi-core processors, there were introduced new technologies into the architecture of new platforms, such as independence of the respective software components (*Intel Virtualization Technology - VM*), acceleration of the data exchange mechanism (*Intel I / O Acceleration Technology - I / O AT*), remote controllability (*Intel Active Management Technology - iAMT*). A set of new technologies is aimed at improving the efficiency of the computing platform as a whole.

Intel has been working on the concept of concurrency and hardware multithreading for a long time. By 1994, Intel Pentium processor implemented parallelism at the command level - an architectural feature in which there are extracted the instructions of one code flow, executed in parallel, and then combined in the same order. In 1994, the corporation also implemented dual-processor processing (two full-fledged processors, inserted into two connectors on the system board) by creating a hardware multi-threaded environment for servers and workstations. In 1995, the Pentium Pro processor was introduced, which

supported the efficient integration of four processors on one motherboard, which ensured a higher data processing speed in multithreaded applications oriented on server platforms and workstations.

Intel multi-core processors

These efforts have stimulated the development of single-processor technologies that provide a higher degree of parallelism at the thread level for mass platforms. The corporation has implemented *HT* technology for Pentium 4 and Xeon processors, as an innovative way to achieve a higher degree of parallelism at the thread level in processors for mass systems. The corporation realized that *HT* technology naturally leads to the development of multi-core processors with higher degrees of parallelism.

Since 2005, dual-core Pentium D processors were released. Dual-core Pentium D processors contain two independent cores on a single silicon platten. Each core has its own cache of the second level L2 with the capacity of 1 MB.

The processor cores are based on the Pentium 4 NetBurst architecture. The cores are integrated by a common processor bus operating at 800 MHz. Pentium D processor cores do not support Hyper-Threading technology. For dual-core processors, it presents only in the Pentium Extreme Edition, which, due to this, is visible in the system as a quad-core.

The release of Pentium D processors was established according to the 90-nanometer process technology, while 230 million transistors were placed on the chip. The total size of the cache of the second level is 2 MB. Cache memory is divided in half between two cores in such a way that each of them operates with its own cache memory of the second level (L2) with a capacity of 1 megabyte. But the operation of dual-core processors at the same frequencies, as single cores, is impossible. This is primarily due to restrictions on the heat and power processors consumption, since the combination of two cores on a single crystal leads to a significant increase in these characteristics. Therefore, dual-core processors have a much lower cycle speed than single-core ones.

The dual-core Pentium Extreme Edition processor was released with a frequency of 3.2 GHz, a system bus frequency of 800 MHz and 2 MB of cache of the second level (1 MB for each core). Each core supports *HT* technology, therefore four processors are visible in the system.

In order to realize fully the performance potential growth provided by several cores, there is needed a method that provides the processor with a sufficient amount of data. According to experts [17, 18, 20], the existing Intel system bus architecture is able to satisfy the requirements of a maximum of four cores. In this architecture, the system bus connects the central processor to the general memory. The memory controller, which is a part of the corresponding chipset, controls the data transfer from the memory to the central processor and vice versa.

On October 20, 2006, Intel released its first quad-core processor for multiprocessor server systems, code-named Tigerton. A specialized high-speed intercomponent connection that connects each processor directly to the chipset provides more than a double increase of system performance and throughput. The

chipset supports FB-DIMM memory modules (DIMM modules equipped with a high-speed bus) and can be equipped with four channels for connecting to the memory modules, which extends the throughput and increases the supported memory size. I/O Acceleration Technology (*IOAT*) support firstly was implemented on multiprocessor platforms. Quad-core Intel Xeon processors are based on Intel Core microarchitecture. Their performance is almost 50% higher than of the modern generation of dual-core processors, maintaining the same level of energy consumption. Dual-core Intel Xeon processors of 7000 series were manufactured using 65-nanometer process technology and were designed in order to be installed on servers with four or more processors. The processors of this series support *HT* technology, therefore they are capable to perform 4 computational flows simultaneously. In addition, they are equipped with a 16 MB cache of the third level used by both cores. The processors support *Intel Virtualization Technology* and *Intel Cache Safe Technology*, which helps to minimize downtime in critical situations. Intel Xeon 7100 Series Dual-Core Processors are designed in order to be installed on servers with four or more processors. This family includes low-power processors (95W). The 7100 series processors are almost two times faster than the processors of the previous generation, and almost three times faster according to the feature “performance per watt”.

In September 2006, there was demonstrated a prototype of Intel processor with 80 cores and an integrated static memory (SRAM) with a total capacity of 20 MB (256 KB per core). Data exchange between the processor cores is carried out at a speed of more than a terabyte per second. Current samples of 80-core processors operate at a frequency of 3.1 GHz. It is expected that commercial availability of such processors will set in five years. At the moment, we can't say that we have a full-featured version of the processors of the future, rather it is one of the first implementations of the concept developed within the framework of the Intel Tera-Scale initiative, the aim of which is the availability of processors with teraflop performance by 2012 to the mass buyer.

AMD Multi-Core Solutions

AMD corporation has developed *Pacifica* virtualization technology and *Presidio* security technology for serial multi-core processors [20]. *Pacifica* technology allows to run several independent operating systems and applications on one computer, and *Presidio* technology increases the security of working with data using a special protected area in the processor. Multi-core chips designed by AMD engineers are characterized by lower power consumption, since they are made on a single substrate, unlike Intel crystals, in which several individual cores are mechanically connected. AMD uses the *Direct Connect* architecture in order to combine the computing cores, and the *HyperTransport* bus is used to communicate with the system logic set. The third version of this bus is three times exceeds the throughput of *HyperTransport 2.0*. Cumulative maximum throughput of

HyperTransport 2.0 is 22.4 GB/s. In 2004, AMD demonstrated a dual-core 64-bit AMD Opteron processor (with support of the x86 command system), see fig. 9.2.

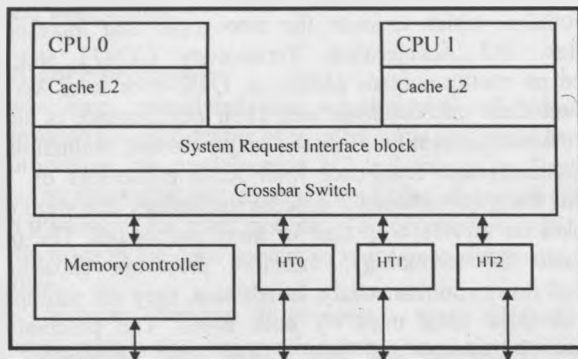


Fig. 9.2. Opteron Dual Core Processor block diagram

This has improved system-wide performance and data processing efficiency. Direct connection architecture - *Direct Connect* allows to connect several processors, a memory controller, and I/O modules directly. It neutralizes the disadvantages of modern system architectures and eliminates bottlenecks in data exchange. It is believed that this architecture reduces the delays at memory request, and since the I/O means are directly connected to the central processor, the balance between the processor performance and the throughput of the I/O subsystem improves. In turn, all processors are connected directly to each other, that provides an almost linear increase in performance for multiprocessor systems. With the increasing amount of processors, the throughput memory ability grows linearly. The dual-core AMD 64 processors support compatibility with AMD 64 applications for x86 platform, making their distribution easier. It should be noted that the dual-core implementation in AMD processors is somewhat different from the Intel implementation. Intel's approach is to place simply two cores on one chip. With such a dual-core organization, the processor does not have any special mechanisms for inter-core interaction. As in conventional dual-processor systems, the cores communicate via the system bus. Accordingly, the system bus is shared between the processor cores and at working with memory that leads to an increase in delays at memory request of two cores simultaneously. AMD processors have duplicated some resources. Although each of the Athlon 64X2 cores has its own set of executive devices and dedicated second-level cache memory, the memory controller and the *Hyper-Transport* bus controller common for both cores. The interaction of each core with shared resources is carried out through a special *Crossbar switch* and the *System Request Queue*. At the same level, there is organized the interaction of the cores with each other, due to which the issues of cache memory coherence are solved without additional load on the system bus and memory bus (Fig. 9.3).

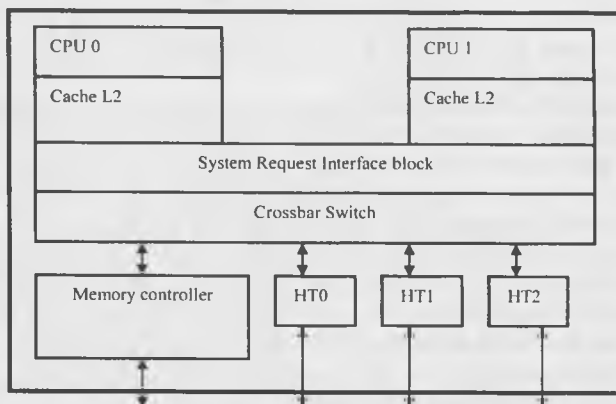


Fig. 9.3. Athlon 64X2 Dual Core Processor block diagram

The Athlon 64X2 (Toledo) processors, containing two cores with a second level cache memory of 1 MB per core, consist of approximately 233.2 million transistors and have an area of about 199 square millimeters. The crystal and the complexity of the dual-core processor turn out to be approximately twice as large as the crystal of the corresponding single-core processor.

IBM multi-core processors

In 2001, the corporation developed its first universal dual-core Power 4 processor for IBM eServer servers.

One case contained two 64-bit microprocessors. The architecture of the Power 4 crystal was distinguished by several modern solutions:

- superscalar structure,
- extraordinary execution of commands,
- large cache memory on a chip,
- specialized port for general memory,
- high-speed connections for combining microprocessors into systems with distributed shared memory architecture.

Each Power 4 processor had two conveyor blocks for working with 64-bit floating-point operands, which choose for execution five commands each, and two blocks for working with memory. The processors contained a separate cache of commands and data of the first level with a capacity of 64 KB each. In addition, there was a shared cache of the second level on the chip (with a capacity of 1.4 MB) and an external cache of the third level (with a capacity of 32 MB). The presence of L2 cache shared by two crystal processors, as well as by external processors of other crystals via high-speed 128-bit trunks operating at a frequency of more than 500 MHz (which provides a throughput of more than 10 GB/s) has become one of the distinguishing features of Power 4.

The architecture of the next representative of this line - the Power 5 processor, is created on the principles applied in Power 4. Two processor cores on

the same chip have a separate L1 cache for data and commands and a common L2 cache. The cache memory of the second level is organized in the form of three separate blocks, each of which has its own controller. The cores can access any of the three controllers independently.

A number of important innovations begins with the fact that although the L3 cache is located outside the chip, it is directly connected to the L2 cache, that reduces the delays during the work with the cache memory and improves scalability. A Power 5 based system can include up to 64 processor configurations. In Power 5 microprocessor for the first time IBM implemented *Micro-Partitioning* technology, which made it possible to present each physical processor as several (up to 10) logical ones. *Micro-Partitioning* also provides a single console in order to control all types of systems and a large set of system services for workloads control and for resources reallocation, which makes it possible to do more work. In Power 5, each processor core can simultaneously process two flows of commands, i.e., it works as two logical processors, and the crystal itself - as four logical processors. The commands of both flows are extracted from the same first-level command cache and are loaded together into execution units. Theoretically, both flows of commands after reading from the command cache should pass through the conveyor and use the resources of the physical processor without conflicts between the flows.

Power 5 executes 50% more commands than Power 4. The ultra-dense layout technology, the feature of which is the use of shared memory and high-throughput internode connections, enables high-speed connections between eight processors of Power 5. Four crystals of Power 5 with four crystals of L3 cache memory are packaged in a Multichip Module (MCM). Since Power 5 was designed to work with the next-generation data storage systems, then there is supported the addressing of the data storage with a capacity up to 96 Pentabytes. There should also be noted that Power 5 is compatible with Power 4 at the software level. The next step for IBM was Power 5+, which is the so-called "server-on-a-chip", see fig. 9.4.

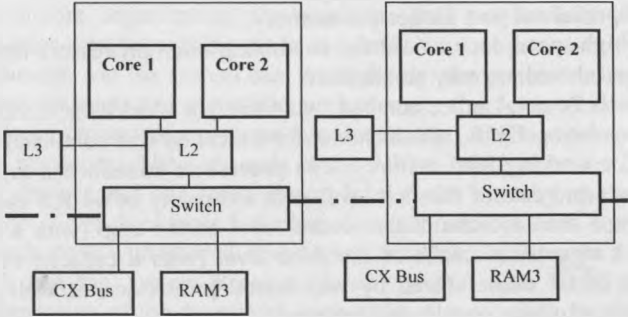


Fig. 9.4. Power 5+ processor circuit

It contains two processors that support SMT technology, a high-performance system switch, an internal cache memory up to 72 MB, and an input-output interface. The peak throughput of the processor-memory bus is 42.6 GB/s. Servers equipped with Power 5+ microprocessors are designed taking into account the requirements for computing systems of small and medium-sized businesses; they take into account the limited resources typical for many such companies.

Power technology was also the basis for Power PC processors. The latest product in this series, the Power PC 970MP, is the first dual-core 64-bit version of Power PC created on the basis of Power 4 architecture using 90nm technology. Dynamic switching is possible between 64- and 32-bit configurations. Efficiency is supported by the SMP optimization mechanism. Each core in the PowerPC 970MP has its own L2 cache, as well as an independent thermal diode and power bus. Compared to the previous modifications, it has an increased processor bus throughput and a size of L2 cache. The separate L2 cache allows to disable or to put into a sleep mode one of the cores. Up to eight commands can be executed per a cycle.

Based on the Power architecture, there was also created a Cell processor element, which is a "supercomputer-on-a-chip", a part of a tripartite partnership agreement between IBM, Sony and Toshiba Corporation. Its architecture includes eight complementary computing elements SPE (*Synergistic Processor Element*) and a core based on Power basis. All these components are interconnected by a high-speed EIB bus. Specialized SPE microcomputers are designed to operate at frequencies above 4 GHz, support mass processing of floating point data and for multiple operating systems support simultaneously. A single Cell computing element has a theoretical capacity of 250 GFLOPS (billions of floating point operations per second). In addition, the processor is optimized for performing broadband media applications (for example, games, programs, video) and has internal energy management technology. Not only game consoles, but also servers began to be produced on the basis of the Cell element, since it copes well with serious computing tasks.

Sun Microsystems Multi-Core SPARC Processors

In early 2004, Sun Microsystems released the dual-core UltraSparc IV processor for Sun Fire V servers and introduced the next-generation UltraSPARC IV+ processor (Panther). It is based on CMT (Chip Multithreading) technology - multithreading on a chip, and marks the next step in implementing the Throughput Computing strategy ("high throughput "). UltraSPARC IV+ was developed on the basis of 90-nm process technology of Texas Instruments. Compared to the UltraSPARC IV processor, it can double the application performance by the increase of cache and buffers, by an improved branch prediction mechanism, advanced memory prefetch capabilities, and new computing capabilities. In addition, UltraSPARC IV+ uses a three-level cache hierarchy, including a 2-megabyte cache integrated on a chip and a 32-megabyte external third-level cache. Like the UltraSPARC IV, the new processor has two cores integrated on a single chip. In addition to the new features for increasing productivity, the UltraSPARC

IV+ processor has a significantly higher cycle speed (initially 1.8 GHz), which provides the highest throughput compared to other UltraSPARC processor models. Compared to the UltraSPARC IV, the performance of each flow was doubled. Like the UltraSPARC IV processor, the UltraSPARC IV+ provides binary compatibility with previous generations of SPARC architecture processors. Due to this, users retain development tools and application software. In addition, it is easy for users to upgrade existing systems in order to increase their performance and reliability - processors can be installed in existing systems of the Sun Fire family and work with the UltraSPARC IV and UltraSPARC III processors already installed in the system.

In 2005, Sun Microsystems introduced the new UltraSPARC T1 multi-core processor, formerly code-named Niagara. The Niagara project is a major part of Sun's work aimed at keeping the SPARC processor family up to date with the widespread popularity of the x86 architecture from Intel and AMD and the with the growing power of IBM POWER processors.

The UltraSPARC T1 processor consists of eight cores, each of which is capable of processing four flows in parallel, so that each chip can work with 32 parallel flows, which is extremely important for types of applications such as databases and Internet searches. Initially, there was achieved the low power consumption in UltraSPARCT1 with the low frequency, which is only 1.2 GHz. At the same time, the processor architecture has undergone changes: almost all of the server's cache memory, system memory and input/output elements are intergated directly into the processor itself. In essence, UltraSPARC T1 is one of the first in the world full-fledged servers on a single chip. In addition, in order to reduce power consumption and heat dissipation, this architecture greatly simplifies the design of the servers themselves and allows to achieve the highest level of performance while maintaining a compact size.

Main technical characteristics of UltraSPARC T1:

- cache memory of the second level on a chip;
- encryption support for RSA public key;
- 48-bit virtual and 40-bit physical addressing;
- four page sizes support: 8 KB, 64 KB, 4 MB and 256 MB;
- Hypervisor technology support.

Versions of processors with 4, 6 or 8 cores:

- primary command cache with a capacity of 16 Kbytes per core;
- primary data cache of 8 Kbytes per core;
- a single cache memory of the second level of 3 MB;
- up to eight cores, 4 threads in each core;
- four 144-bit memory controllers DDR2 533 SDRAM ECC;
- up to four RAM modules per controller (up to 16 DIMMs);
- two-channel mode of random access memory;
- JBUS interface with a peak effective throughput of 3.1 GB / s, 128-bit bus, frequency - from 150 to 200 MHz;
- 0.09 micron process, CMOS;
- frequency - 1.0 or 1.2 GHz;

- power consumption: 72 W, peak - 79 W.

Therefore, with the occurrence of multi-core processors, it became possible to work in multi-tasking environments with the simultaneous execution of several active and background applications. This, in turn, increased efficiency and reduced power consumption while running multiple applications on one PC.

Questions for self-control

1. What is the novelty of the principle of multi-core processor?
2. What companies are developing multi-core microprocessor systems?
3. What are the main characteristics of multicore processor systems from leading manufacturers?

Test questions

1. The data processing technology in the processor, which ensures more efficient operation of the processor due to data manipulation, rather than simple execution of the list of commands, is:

- A) Hyper-Threading Technology
- B) speculative execution
- C) dynamic performance
- D) 3DNow technology!

2. Differences of the multi-core approach from the multiple-core approach are:

- A) core complexity
- B) the amount of cores located on the chip
- C) one approach relates to SIMD, another - to MIMD systems
- D) energy management approach

3. One of the main problems with the introduction of multi-core processors is associated with

- A) energy management approach
- B) with programming complexity
- C) increasing power consumption
- D) the complexity of boards circuit

4. How does the operating system view each of the executive kernels?

- A) discrete processor with all the necessary computing resources
- B) as a cluster
- C) physical processor
- D) as the user asks

5. What new technologies are not being implemented together with multi-core processors in the architecture of new platforms?

- A) the independence of the respective software components
- B) speeding up the data exchange mechanism
- C) remote controllability
- D) virtual data exchange

6. What does Pacifica technology do?

- A) run on a single computer several independent operating systems and applications
- B) manages multiple cores using one operating system
- C) run multiple applications in parallel
- D) run one application on multiple cores

7. What does Presidio technology do?

- A) improves data security
- B) improves data processing speed
- C) increases power consumption
- D) reduces the data processing speed

8. Architecture of direct connections - *Direct Connect* does not allow to connect

- A) multiple processors directly
- B) memory controller directly
- C) I / O modules directly
- D) multiple processors in parallel

9. Which of the solutions is not included in the architecture of the Power 4 crystal?

- A) superscalar structure
- B) alternate execution of commands
- C) extraordinary execution of commands
- D) dedicated port for general memory

10. How many more instructions does Power 5 execute in comparison to Power 4?

- A) 25% B) 30% C) 50% D) 40%

11. What is included in the Multichip Module (*MCM*)?

- A) Four Power 5 crystals with four L3 cache crystals
- B) two Power 5 crystals with two L3 cache crystals
- C) Four Power 5 crystals with two L3 cache crystals
- D) Two Power 5 crystals with four L3 cache crystals

12. What is included in the Cell architecture ("supercomputer-on-a-chip")?

- A) eight complementary computing elements SPE (Synergistic Processor Element) and a core based on Power basis

- B) four complementary computing elements SPE (Synergistic Processor Element) and a core based on Power basis
- C) six complementary computing elements SPE (Synergistic Processor Element) and a core based on Power basis
- D) sixteen complementary computing elements SPE (Synergistic Processor Element) and a core based on Power basis

13. Indicate among the presented systems the systems with mass parallelism.

- A) workstation
- B) multiprocessor complex
- C) Grid system
- D) cluster

14. How many cores does the UltraSPARC T1 processor consist of?

- A) 8 B) 4 C) 6 D) 12

15. How was the low power consumption achieved in the UltraSPARC T1?

- A) by a low cycle speed
- B) by a high cycle speed
- C) by the microcommands processing method
- D) by the compactness

CHAPTER 10. CO-PROCESSORS. METHODS FOR INFORMATION EXCHANGE BETWEEN CPU AND CO-PROCESSOR

A co-processor is a specialized processor that extends the capabilities of the central processor of a computer system, but is designed as a separate functional module. Physically, the co-processor can be a separate microcircuit or it can be integrated into the central processor (as it is organized in case of the mathematical co-processor in PC processors starting with Intel 486DX).

There are distinguished general-purpose mathematical co-processors, usually accelerating floating-point calculations, input-output co-processors (for example, Intel 8089), unloading the central processor from controlling I/O operations or expanding the standard address space of the processor, co-processors for some narrowly specialized calculations.

The co-processors can be a part of a logic set developed by one particular company (for example, Intel produced the 8087 and 8089 co-processors in a complex with the 8086 processor) or can be produced by a third-party manufacturer.

Co-processors perform such complex operations as dividing long operands, calculating trigonometric functions, extracting the square root and finding the logarithm, 10-100 times faster than the general processor with higher accuracy. Addition, subtraction and multiplication operations are performed by the general processor and are not transferred to the co-processor.

The co-processor command system is different from the processor command system. The executable program itself must determine the presence of a co-processor and then use the instructions written for it; otherwise, the co-processor only consumes current and does nothing. Most modern programs, designed for the use of co-processors, detect its presence and use the provided opportunities. The most effective co-processors are used in programs with complex mathematical calculations: in spreadsheets, databases, statistical programs and computer-aided design systems. At the same time, working with text editors, the co-processor is completely not used.

Despite the fact that almost all processors, starting from the 486th, are equipped with a built-in co-processor, their speed can vary. Historically, Intel co-processors are faster than AMD and Cyrix co-processors, but recently the situation has begun to change.

Typically, there are two ways of information exchange between the CPU and the co-processor:

- direct connection of input and output ports (the CPU has a special interface for interaction with the co-processor);
- with the exchange through the memory (the information exchange between the CPU and the co-processor occurs due to the access of the co-processor to the RAM through the system bus).

One of the most common types of co-processors is the *mathematical co-processor*. The mathematical co-processor is designed to perform quickly floating-point arithmetic operations, to provide frequently used real constants ($\log_2 10$, $\log_2 e$, $\ln 2$, ...), to calculate trigonometric and other transcendental functions (tg , $arctg$, \log , ...).

The arithmetic co-processor contains eight 80-bit numerical registers for storing intermediate results of calculations, a control register, a status register, a tag register, a command pointer register, and an operand pointer register.

Numerical registers are indicated as - ST0 - ST7 (80 bits). They are shown on fig. 10.1.

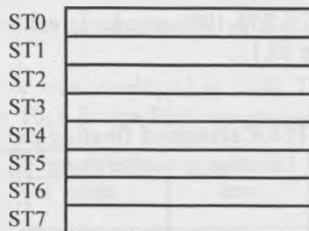


Fig. 10.1. Numerical registers of the arithmetic co-processor

Numerical registers are used as a stack. The state register in the ST field contains the number of the numerical register, which is the top of the stack. Executing the commands, numerical registers can act as an operand. In this case, the number of the register indicated in the command is added to the contents of the ST field of the status register and, thus, there is determined the used register. Most of the commands after execution increase the ST field of the status register, recording the results of their work to the stack of numerical registers.

Registers can be used as an array, but in this case it is necessary to take care of the constancy of the ST field of the status register, since otherwise the numbers of the numerical registers will change.

The tag register, see fig. 10.2 is divided into eight two-bit fields, which are denoted by TAG0 ... TAG7. Each field refers to its numerical register.



Fig. 10.2. Tag Register Format

The tag register fields classify the contents of "their" numerical register, for example, the field 00 - register contains a valid non-zero number, 10 - register contains an invalid number - non-number, infinity, uncertainty.

If all registers of the co-processor are empty, and then one valid non-zero value was recorded into the stack of numerical registers, the contents of the tag register will be 3FFFh.

Most modern *mathematical co-processors* use the standard IEEE 754-1985 “IEEE Standard for Binary Floating-Point Arithmetics” to represent real numbers. The high digit of the binary representation of a real number always encodes the sign of the number. The rest part is divided into two parts: the exponent and the mantissa:

$$R = m * p^n,$$

where m - mantissa;
p - number system base;
n - order.

According to the IEEE 754-1985 standard, real numbers are represented in three basic forms (see table 10.1).

Table 10.1

IEEE standard floating point data

Type	The size, bit	The range of number change		Decimal numbers accuracy	Machine ϵ
		max	min		
Single	32	$3.4 * 10^{-38}$	$3.4 * 10^{38}$	6	$1,192 * 10^{-7}$
Double	64	$1.7 * 10^{-308}$	$1.7 * 10^{308}$	15	$2,221 * 10^{-16}$
Long double	80	$3.4 * 10^{-4932}$	$3.4 * 10^{4932}$	19	$1,084 * 10^{-19}$

Here it should be noted that the characteristics of a floating point number of a double accuracy will depend on the arithmetic used on a particular computer.

Example 1

Let write the number 15.375 in binary form:

$$15.375 = 1111.011_2 = 1.111011 * 2^{11}_2$$

Then, according to the IEEE standard, the number will be represented as:

Single

$$15,375 = 0\ 1000.0001.0\ 111.0110.0000.0000.0000.0000_2 = 41760000_{16}$$

Long double

$$15,375 = 0\ 1000.0000.00010.1110.1100.0000.0000. \dots 0000_2 = 402EC00000000000_{16}.$$

Example 2

Let present an example of encoding of a real number 178.625:

$$178,625_{10} = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 10110010,101_2$$

In order to represent this number in accordance with IEEE-754, it needs to be normalized (expressed in an exponential form):

$$1,78625E_{10}2 = 1,0110010101E_2111.$$

Features of floating arithmetic can significantly affect the results of calculations, up to the fact that the error can make it impossible to get any result at all, therefore, knowledge of the details of the floating points arithmetic implementation is necessary for programmers.

Modern IA-32 architecture processors contain a block of operations with real numbers (Floating Point Unit - FPU). The functioning of this block is largely based on the architecture of the first representative of the mathematical co-processors family - Intel - 8087.

The 8087 co-processor was developed in 1980. The structure of the 8087 co-processor is shown on Fig. 10.3. In the 8087 co-processor circuit, two subsystems can be distinguished:

- bus interface device;
- floating point device.

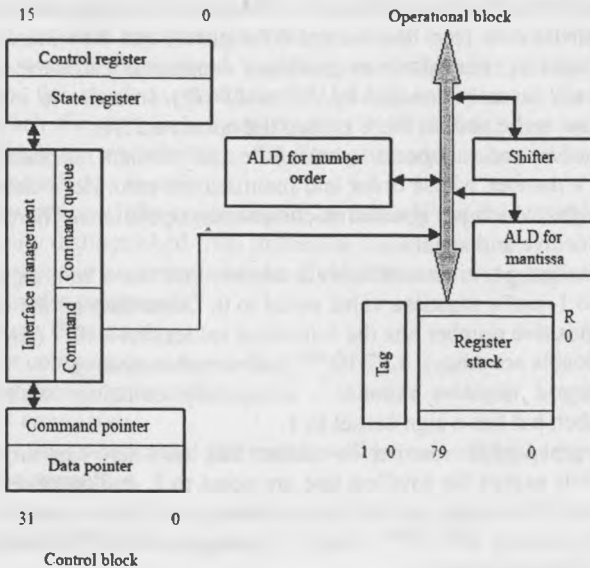


Fig. 10.3. 808 co-processor structure

A further development of the co-processors was the model - 80287, created in 1985. Unlike the 8087, the 80287 co-processor did not have access to the address bus, therefore, all the memory requests are performed by the CPU.

In the 80387 co-processor, the changes affected the floating-point device: the error processing scheme was changed, and a larger set of transcendental functions was also implemented.

In the fifth generation of Intel processors, the integration of FPUs into superscalar architecture has improved the efficiency of operations with real numbers. The FPU can perform one floating point operation in each cycle or receive and simultaneously execute 2 floating point commands, one of which must be an exchange command.

In general, the basic programming model of all of the above co-processors and the FPU for IA-32 are similar - a register stack (eight 80-bit registers $R0-R7$), a tag word, a control register, a status register, a command pointer, and a data pointer.

The registers $R0-R7$ are intended for data storage in the co-processor. These registers are organized in a stack and the access to them is conducted relative to the top of the stack - ST . The register number corresponding to the top of the stack is stored in the status register (TOS field). Like the CPU, the co-processor stack grows to registers with lower addresses. , that perform storing and extraction from the stack, transfer data from the current ST register, and then increment the TOS field in the status register. Many co-processor commands allow the requests to the top of the stack (usually denoted by ST or $ST(0)$). In order to indicate the i -th register relative to the vertex, there is used the notation $ST(i)$.

Let consider various special cases of the real numbers representation.

Zero - a number whose order and mantissa are zero. Zero can have positive or negative signs which are ignored in comparison operations. Therefore, there are two zeros - positive and negative.

The smallest positive number is a number that has a zero sign bit, an order value equal to 1, and a mantissa value equal to 0. Depending on the representation, the smallest positive number has the following values: $1.17 \cdot 10^{-38}$ (single accuracy), $2.23 \cdot 10^{-308}$ (double accuracy), $3.37 \cdot 10^{-4932}$ (advanced accuracy).

The largest negative number - completely coincides with the smallest positive number, but has a sign bit set to 1.

The largest positive number - a number that has a zero sign bit, an order field in which all bits except the smallest one are equal to 1, and contains 1 in all digits of the mantissa. Depending on the representation, the largest positive number has the following values: $3.37 \cdot 10^{38}$ (single accuracy), $1.67 \cdot 10^{308}$ (double accuracy), $1.2 \cdot 10^{4932}$ (advanced accuracy).

The smallest negative number - completely coincides with the largest positive number - has a sign bit set to 1.

Positive and negative infinity - this number contains all 1 in the order field and all 0 in the mantissa field. Depending on the state of the sign bit, there can be positive and negative infinities. Infinity can be obtained, for example, as a result of dividing a finite number by zero.

Non-number - contains all 1 in the order field and any value in the mantissa field. Non-number can occur from improper operations in masked special cases.

Uncertainty - contains all 1 in the order field, and in the mantissa field - the number 1000.0 (for single and double accuracy) or 11000.0 (for advanced accuracy, since the high bit of the mantissa is stored in this format).

The arithmetic co-processor, along with real numbers, can process integers. It has commands that convert integers to real numbers and vice versa.

There are four possible integer formats:

- integer;
- short integer;
- a long integer;
- packed decimal number.

An integer occupies two bytes. Its format is completely correspond to the format used by the central processor. An extra code is used to represent negative numbers. A short integer and a long integer have similar formats, but occupy 4 and 8 bytes, respectively.

A packed decimal number occupies 10 bytes. This number contains 18 decimal digits, two in each byte. The sign of the packed decimal number is located in the high bit of the leftmost byte. The remaining bits of the high byte must be equal to 0.

There are co-processor commands that convert numbers to packed decimal numbers from internal representation in advanced real format. If a program attempts to convert denormalized numbers, non-numbers, infinity, etc. into a packed format, the result is uncertainty. Uncertainty in a packed format is a number in which the two high bytes contain 1 in all bits. The contents of the remaining eight bytes are arbitrary. An attempt to use such a packed number in operations is recorded as an error.

All mnemonics of the co-processor commands begin with the letter *F*, so they can be easily distinguished from processor commands.

All co-processor commands can be divided into several groups:

- data transfer commands;
- arithmetic commands;
- number comparison commands;
- transcendental commands;
- control commands.

A part of the co-processor commands will be considered in the final chapter dedicated to the assembler.

Questions for self-control

1. What is a co-processor?
2. What are the main ways of information exchange between the processor and the co-processor.
3. List the functions of the mathematical co-processor.
4. Describe the IEEE-754 floating-point number formats.

5. What are the main differences between the structure of the co-processor 8087 from 80287 and 80387.

Test questions

1. What co-processors do not exist?

- A) mathematical
- B) input-output
- C) performing highly specialized calculations
- D) central

2. What operations does the co-processor not perform?

- A) addition, subtraction, multiplication
- B) finding the logarithm
- C) the calculation of trigonometric functions
- D) square root extraction

3. Working with which programs the co-processor is not used?

- A) in spreadsheets
- B) with text editors
- C) in databases
- D) in computer-aided design systems

4. Which co-processor contains numerical registers?

- A) arithmetic
- B) input-output
- C) performing highly specialized calculations
- D) all

5. In what basic forms according to the IEEE 754-1985 standard are real numbers not represented?

- A) Single
- B) double
- C) long double
- D) Short double

6. What are the R0-R7 registers in the co-processor intended for?

- A) for data storage
- B) for storing addresses
- C) for data transfer
- D) for any purpose

7. What is the smallest positive number?

- A) positive zero
- B) zero

- C) it is a number that has a zero sign bit, an order value of 1, and a mantissa value of zero
- D) it is a number that has a zero sign bit, an order value of 0, and a mantissa value of zero
8. What is positive and negative infinity?
- A) the largest positive number
- B) the largest negative number
- C) a number containing all units in the order field and all zeros in the mantissa field
- D) a number containing all units in the order field and all units in the mantissa field
9. That's a non-number?
- A) zero
- B) all 0 in the order field and all 1 in the mantissa field
- C) all 1 in the order field and all 1 in the mantissa field
- D) all 1 in the order field and any value in the mantissa field
10. What format for integers does not exist?
- A) advanced integer
- B) short integer
- C) packed decimal
- D) long integer
11. All mnemonics of the co-processor commands begin with a letter
- A) F B) E C) S D) D
12. What is uncertainty in a packed format?
- A) entropy
- B) a number in which the two high bytes contain 1 in all bits
- C) zero
- D) a number in which the two high bytes contain 0 in all bits
13. In which case does the uncertainty fail?
- A) if the program makes an attempt to convert denormalized numbers into a packed format
- B) if the program makes an attempt to convert non-numbers into a packed number
- C) if the program makes an attempt to convert an infinity into a packed format
- D) if the program makes an attempt to convert an integer into a packed format
14. What influence have the features of floating arithmetic?

- A) calculation speed
- B) number of operations
- C) calculation results
- D) all answers are correct

15. What groups of co-processor commands do not exist?

- A) data transfer commands
- B) arithmetic commands
- C) duplicate commands
- D) control commands

CHAPTER 11. ORGANIZATION OF A MEMORY SUBSYSTEM IN A COMPUTER

Computer memory is a set of devices used for memorizing, storing and issuing information. The individual devices included in this collection are called storage devices (SD) of one type or another [2, 5, 12, 16, 17].

The term "storage device" is usually used when we speak about the principle of creation of some kind of memory device (for example, a semiconductor SD, a hard disk drive SD, etc.). The term "memory" is used when we want to emphasize the logical function performed by the memory device or the location in the computer equipment (for example, random access memory - RAM, external memory, etc.).

Storage devices play an important role in the overall structure of computers. According to some estimates [18, 19, 20], computer performance in different classes of tasks by 40-50% is determined by the characteristics of the SD of various types included in its composition. The main parameters that characterize storage devices include capacity and speed.

11.1. Key parameters and characteristics of the storage devices

A number of indicators are used to evaluate the properties of storage devices and the quality of the functions they perform. Let consider the main parameters and characteristics of SD.

One of the main characteristics of SD is *memory capacity* - the largest amount of information that can simultaneously be stored in SD. The basic unit of measurement of memory capacity is *a bit*, representing one digit of a binary number (logical 0 or 1). Adding 2^3 , 2^{10} , 2^{20} multipliers to the main unit of measurement allows to get such units of measurement as a byte (1 byte = 8 bits), kilobyte (1 KB = 1024 bytes), megabyte (1 MB = 1024 KB = 1048576 bytes). In some cases, a (machine) word containing 8, 16 32 bits is used as a unit for memory capacity measurement. The bit is stored by the memory element, and the word by the memory cell. A memory cell is a set of memory elements, simultaneous access to which is only possible.

In order to detail the memory structure, there is used an indicator of the storage device organization, for evaluation of which there is used the product of the number of stored words by their capacity, which gives the memory capacity of the SD. For example, two SD with the organization 32×32 and 64×16 have the same memory capacity of 1024 bits.

The speed (performance) of SD can be characterized by the cycle time of recording or reading (sampling) of information and by the time of accessing the memory, which includes both cycles. *The recording cycle time* is estimated by the time interval after the recording signal occurrence before the input word is entered the memory cell, *the reading cycle time* is estimated by the interval between the moments of the reading signal occurrence and the word at the SD output. In cases

when accessing the first word of a transmitted word packet requires more time than the subsequent words, there are introduced two parameters: *the access time at the first access* and *the transmission rate for subsequent words of the packet*. The given parameters are operational, or measured.

In addition to them, there are set a number of mode parameters in the form of time relationships between various control signals, which are necessary for the normal functioning of SD. The main control information signals should include:

A - the address, or address code. The address code is the element number of the memory cell in which the bit, byte or the word of information is stored. The number of digits of the address code is expressed by an integer grade of two. For example, a 12-digit code $A_{11}A_{10}A_9\dots A_0$ allows to access any of $2^{12}=4048$ of the SD memory elements;

\overline{CS} (*Chip Select*) or \overline{CE} (*Chip Enable*) - *chip selection* signals, or microcircuits, activating the work of this microcircuit at a zero logic level;

$\overline{W/R}$ (*Write/Read*) - a signal that controls the operating mode of the memory: $\overline{W/R} = 0$ - recording; $\overline{W/R} = 1$ - reading;

DI, DO- (*Data Input, Data Output*) - input and output *m*-digit data of SD transmitted via combined or separate buses. The number of digits *m* is determined by the SD organization.

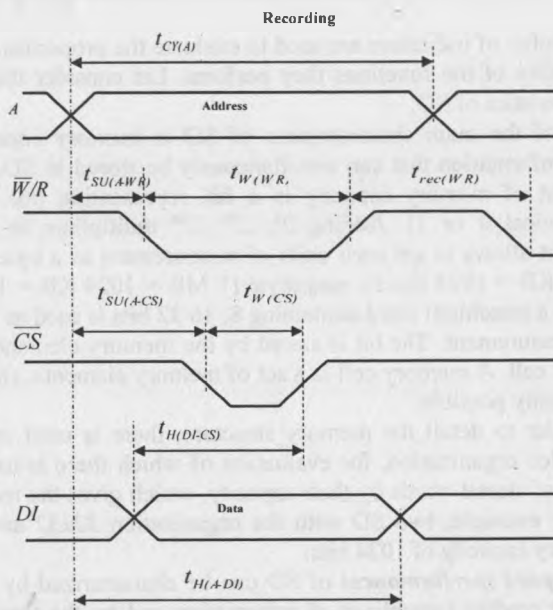


Fig. 11.1. Timing diagram of a static RAM operation in a recording mode

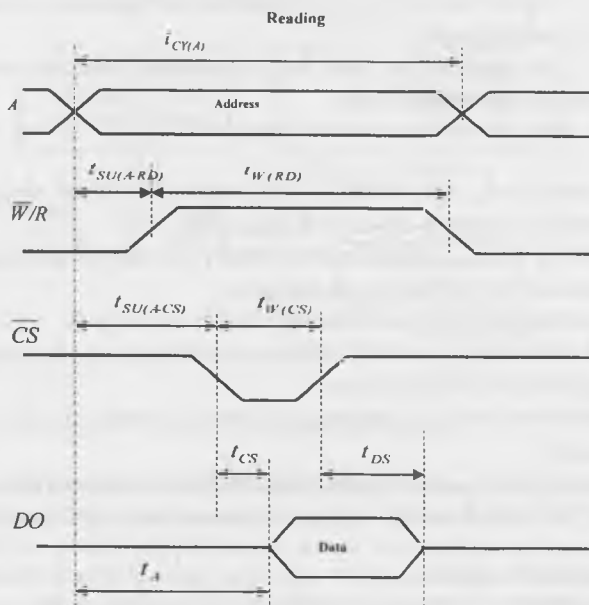


Fig. 11.2. Timing diagram of a static RAM operation in a reading mode

Let consider the sequence of control signals supply to the SD inputs. First of all, the address code of the required memory element (cell) is supplied in order to block the memory for accessing the cells of other operations. Then there is supplied the signal $\overline{CS}=0$, which activates the operation of the selected chip, and the signal $\overline{W/R}$ of recording/ reading.

Taken on Fig. 11.1 and 11.2 symbols of time intervals:

$t_{CY(A)}$ - addressing cycle time in the recording and reading modes;

$t_{SU(A,W/R)}$, $t_{SU(A,RD)}$ - the installation time of the recording and reading signals relative to the address;

$t_{SU(A,CS)}$ - the installation time of the chip selection signals relative to the address in the recording and reading modes;

$t_{W(WR)}$, $t_{W(RD)}$ - the duration of the recording and reading signals, the duration of the chip selection signals in the recording and reading modes;

$t_{W(CS)}$ - the duration of the chip selection signals in the recording and reading modes;

$t_{H(A,DO)}$ - retention time between the beginning of the receipt of the address and the end of the receipt of data;

$t_{H(D,CS)}$ - the retention time between the beginning of the data receipt and the end of the chip selection signal;

t_A - address sampling time;

t_{CS} - time of data occurrence regarding the beginning of the chip selection signal in the reading mode;

t_{DS} - the delay in the data output transition from the active state to the disabled state in the reading mode.

The time relationships between the control signals are set by the following values:

- cycle time t_{ST} , which is the interval between the beginnings (or terminations) of signals at any control input of the SD;
- settling time t_{SU} , which is estimated by the interval between the beginning of the signals of two different control signals;
- operation duration t_W of the specified control signal;
- retention time t_H , which is estimated by the interval between the beginning of one signal and the end of another;
- the storage time t_V , which is the interval between the ends of one and the other signals;
- access time t_A , which is estimated by the time interval from the occurrence of one of the control signals to the occurrence of the information signal at the output;
- and nearby others

The indicated values can be set for any SD control signals.

11.2. Storage Classification

Storage devices can be classified by a number of parameters and features. The fig. 11.3 presents a classification by type of request and organization of access to memory cells.

According to the type of access, the memory devices are divided into devices that allow both reading and recording information, and read-only memory devices (ROM), intended only for reading the data recorded in them (*ROM - read only memory*). The first type of SD is used in the process of processor operation to store executable programs, source data, intermediate and final results. In ROM, there are stored, as a rule, system programs that are necessary to start the computer, as well as constants. In some computers designed, for example, to work in control systems using the same immutable algorithms, all software can be stored in ROM.

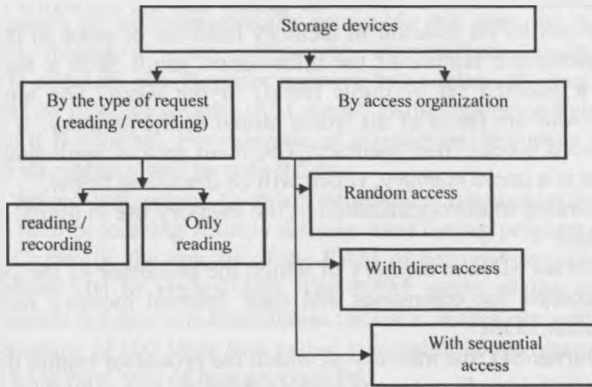


Fig. 11.3. Storage Classification

In SD with a random access (*RAM - random access memory*), the access time does not depend on the location of the memory (for example, RAM).

In SD with a direct (cyclic) access due to the continuous rotation of the storage medium (for example, a magnetic disk - MD), the ability to access a certain section of the medium is cyclically repeated. The access time here depends on the relative position of this section and the reading/recording heads and is largely determined by the speed of rotation of the medium.

In SD with a sequential access, sections of the storage medium are sequentially scanned until the desired section occupies a certain desired position opposite to the reading/recording heads (for example, magnetic tapes - MT).

According to the principle of action, there are distinguished:

- *semiconductor* SD made on integrated circuits;
- *magnetic* SD, among which there are devices with moving storage devices (drives on flexible and hard magnetic disks);
- *optical (laser)* SD, in which digital data is presented in the form of a binary relief of the reflective surface of the disk. Data is recorded and read by the optical scanning engine, which includes a laser and a mirror system.

According to the method of storing information, RAM is divided into two subgroups:

- *static RAM* (SRAM), in which the memory elements state during the storage of information remains unchanged;
- *addressable* SD, in which access to the memory cell is provided using the address code, which is essentially the number of the cell. These include operational memory (RAM), or SD with random sampling, and permanent SD, providing direct access to any memory cell with a given address both during recording and reading. In addressable SD all cells are equally accessible;
- SD with a *sequential access* that sequentially view all memory cells. This group of SD includes video memory, a FIFO buffer (*First-In First-Out*) and a stack, or LIFO (*Last-In First-Out*);

- *associative SD*, in which the search and the extraction of information is carried out not by its location in memory (address or place in the queue), but by some characteristic feature of the information itself. Such a feature may be, for example, a selected set of digits (field) of the word. The input word field is compared with the fields of all words stored in the memory. If the fields match, then the word is read from memory. The main area of application for associative data access is a *cache memory*, which will be discussed below.

According to the organization of the memory use in digital systems there are distinguished:

- *internal SD*, the memory of which the processor of the computing system directly accesses for commands and data. Internal memory includes RAM and semiconductor ROM

- *external SD*, the memory of which the processor cannot directly access for commands and data. In order to use the information stored in the external SD, it is previously transferred to the internal SD.

- *dynamic RAM (DRAM)*, in which the state of the memory elements (usually semiconductor capacitors) does not remain unchanged and requires a periodic regeneration process.

Obviously, an ideal storage device must have infinitely large capacity and have an infinitely short circulation time. In practice, these parameters are in conflict with each other: within the framework of one type of SD, an improvement in one of them leads to a deterioration in the value of the other. In addition, it should be noted the economic feasibility of the creation of a storage device with certain characteristics at a given level of development of a particular technology. Therefore, at present, computer storage devices are created on a hierarchical principle (Fig. 11.4).

The hierarchical memory structure allows cost-effective combination of storing large amounts of information with quick access to information during its processing.

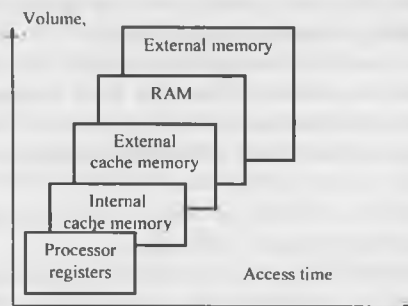


Fig. 11.4. The hierarchical organization of memory in computers

At the lower level of the hierarchy there is a register memory (RM) - a set of registers (see chapter 5) that are directly included in the microprocessor (CPU)

content. CPU registers are programmatically accessible and store the information most often used during program execution: intermediate results, address components, cycle counters, etc. Register memory has a relatively small capacity (up to several tens of machine words). RM works at the processor frequency, so the access time to it is minimal. For example, at a processor frequency of 2 GHz, the access time to its registers will be only 0.5 ns.

RAM - a device that serves to store information (programs, initial data, intermediate and final processing results) directly used during program execution in the processor. Currently, the volume of the RAM of personal computers ranges from a few hundred MB to several GB. The RAM works at the system bus frequency and requires 6-8 bus synchronization cycles to access. So, with a system bus operating frequency of 100 MHz (the period is equal to 10 ns), the access time to the RAM will be several tens of nanoseconds (ns).

In order to fill the gap between the RM and the RAM by the volume and time of access, there is currently used cache memory, which is organized as a faster (and therefore more expensive) static random access memory with a special mechanism for recording and reading information and is designed to store information, the most often used at running the program. As a rule, a part of the cache memory is located directly on the microprocessor chip (internal cache), and a part is located outside it (external cache memory). Cache is not software accessible. In order to access it, there are used the hardware of the processor and computer.

External memory is organized, as a rule, on magnetic and optical disks, magnetic tapes. The storage capacity reaches hundreds of gigabytes with an access time less than 1 μ s. Due to its low speed and high capacity, magnetic tapes are currently used mainly only as data backup devices, access to which is rare, and maybe never. The access time for them can reach several tens of seconds.

The processor registers make up its context and store data used by the processor commands executed at a particular moment. Access to the processor registers occurs, as a rule, according to their mnemonic designations in processor commands.

Regardless of the principles of construction in any SD, four blocks can be distinguished, see Fig. 11.5:

- words search block SED;
- storage device M;
- block of bit circuits BFD;
- control unit CO.

The SED word search block is the part of the SD used to search for the cell in which the searched word is stored or in which the given word should be recorded.

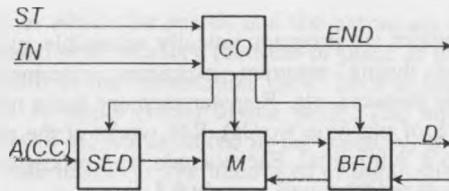


Fig. 11.5. SD general structure

The storage unit is the part of the SD in which the fixing and storage of words takes place and from which, if necessary, the previously recorded word can be read. The storage unit is divided into separate cells. Each cell can store a part of a word, a word or several words.

The block of bit circuits BFD is the part of the SD through which, at recording, the word D enters the storage unit, and at reading passes from the storage unit to the output of the SD. Some SDs have blocks that perform the functions of both address and bit circuits. Such blocks are called address-bit circuits.

The CO control unit is the part of the SD that, according to the start signal of the ST operation and according to the IN operation code (reading or recording), generates a sequence of control signals necessary to perform the specified operation. The completion of the operation is fixed by the signal of the end of the operation END at a special CO output.

By the methods of SED creation, the SD are divided into two types: with M-search (search for a place) and with B-search (search for time). In SDs with M-search, a cell is searched by determining at a given address (feature of a word for associative SD) the place (M) in a storage unit that is occupied by a cell with a given address (word feature). In SDs with B-search, a cell is searched by determining at a given address (feature) of a moment of time (B) when reading elements can read (record) a word from the desired cell.

In some types of SDs, information reading is accompanied by its destruction. Separate types of SD allow to save information when the power is turned off. Therefore, there are memory devices with destructive and non-destructive information reading, as well as SD that save and do not save information when the power is turned off.

The *cache* is used to match the speed of the CPU and general memory, see Fig. 11.6. In recording mode, the *cache* remembers copies of data words transmitted by the processor to the operational (general) memory. In reading mode, a *cache* is accessed to compare the tag address from the processor with the tags stored in the *cache* words. If it is discovered that one of the tags is equal to the tag address, then the *cache* contains a copy of the data of the addressed general memory cell. In this case, the *cache* generates a signal $Hit=1$ (hit) and provides data to the processor. Otherwise, there is generated the signal $Hit=0$, at which the processor reads from the operational memory and recording of the read data to the *cache* memory. At $Hit=0$, data in the *cache* can also come from the processor.

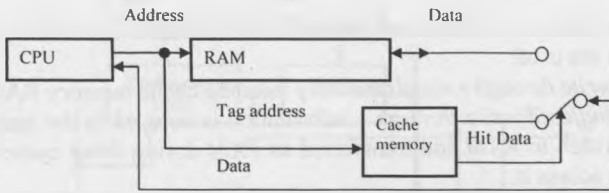


Fig. 11.6. Scheme for explaining the principle of operation of the cache-memory

Computing systems use a multi-level *cache*: cache of the I level (L1), cache of the II level (L2), etc. Desktop systems typically use a two-level cache, while server systems use a three-level cache. The cache stores commands or data that are likely to be sent to the processor for processing in the near future. The cache is transparent to software, so the cache is usually not software accessible.

An exclusive cache is the cache in which stored data of the first level does not have to be duplicated in the cache of the lower levels. Inclusive cache - when any information stored in higher-level caches is duplicated in the cache.

The RAM stores, as a rule, functionally complete program modules (the kernel of the operating system, executable programs and their libraries, drivers of the used devices, etc.) and their data directly involved in the work of the programs. RAM is also used to save the results of calculations or other data processing before sending them to an external SD, data output device or communication interfaces.

Each memory cell has a unique address. Organizational methods for allocating memory give programmers the ability to effectively use the entire computer system. Such methods include a solid ("flat" - *flat model*) memory model and a segmented memory model (*segmented model*).

Organizational methods of memory distribution make it possible to organize a computing system in which the working address space of the program exceeds the size of the actual memory in the system, while the lack of RAM is filled up with external slower or cheaper memory (hard drive, flash memory, etc.) This concept is called **virtual memory**. In this case, the linear address space can be mapped onto the space of physical addresses either directly (the linear address is the physical address), or using the page translation mechanism. In the second case, the linear address space is divided into pages of the same size that make up virtual memory. Page translation provides the mapping of the required pages of virtual memory into the physical address space.

In addition to implementing a virtual memory system, external SDs are used for long-term storage of programs and data in the form of files.

11.3. Cache memory

In order to match the contents of the cache and *RAM*, three recording methods are used:

- *write through* - simultaneously with the cache memory *RAM* is updated.
- *buffered write through* - information is delayed in the cache buffer before being written to *RAM* and transferred to *RAM* during those cycles when the CPU does not access it.
- *write back* — the change bit in the tag field is used, and the line is overwritten into the *RAM* only if the change bit is equal to 1.

As a rule, all recording methods, except *write through*, allow to postpone and to group recording operations into *RAM* in order to increase performance.

Two types of data blocks are distinguished in the cache structure:

- data display memory (actually the data itself, duplicated from the *RAM*);
- tag memory (features indicating the location of cached data in the *RAM*).

The data display memory space in the cache is divided into lines - blocks of a fixed length (for example, 32, 64 or 128 bytes), see. Fig. 11.7.

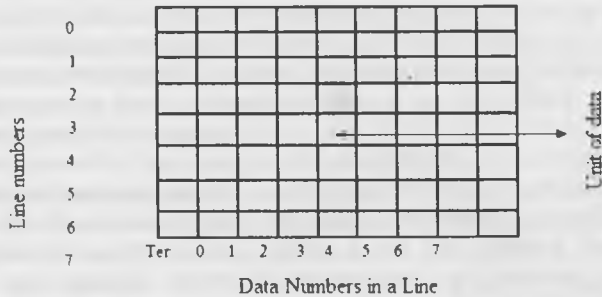


Fig. 11.7. Representation of cache memory as a set of lines

Each cache line may contain a continuous aligned block of bytes from *RAM*. Which block of *RAM* is mapped to this cache line is determined by the line tag and the mapping algorithm. According to the algorithms for *RAM* mapping into the cache, three types of cache memory are distinguished:

- fully associative cache;
- direct mapping cache;
- multiple associative cache.

For a *fully associative cache* (or *FASM*), it is typical that the cache controller can place any block of *RAM* in any line of the cache, see fig. 11.8 and 11.9.

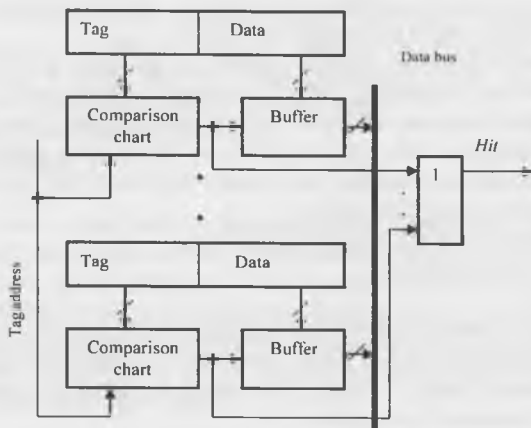


Fig. 11.8. Random Load Cache Scheme

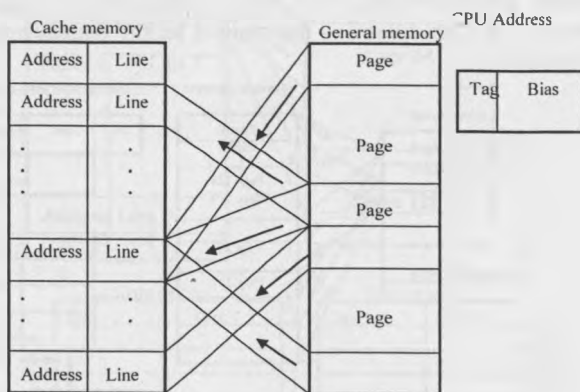


Fig. 11.9. Random Load Cache Organization

The physical address is divided into two parts:

- displacement in the block (cache line);
- block number.

When a block is placed in the cache, the block number is stored in the tag of the corresponding line. When the CPU accesses the cache for the necessary block, a cache miss will be detected only after comparing the tags of all the lines with the block number.

The main advantage of this scheme is the good utilization of the RAM, as there are no restrictions which block can be mapped to a particular cache line. The disadvantages include the complex hardware implementation of this method,

which requires a large number of comparators, which leads to an increase in access time to the cache and to an increase in its cost.

An alternative way of RAM mapping in the cache is a direct mapping cache (or a single-entry associative cache). Organizing a cache with direct allocation for one cache line, there are allocated several pages of main memory, see. Fig. 11.10 a). Usually the page size is 2^n . In this case, the memory address (block number) uniquely determines the cache line in which this block will be placed. The physical address is divided into three parts:

- displacement in the block (cache line);
- cache line number;
- tag.

One or another block will always be placed in a strictly defined cache line, if necessary, replacing another block stored there. When the CPU accesses the cache for the necessary block, it is enough to check the tag of only one line to determine a successful access or cache miss.

The operation of the cache in reading mode is shown on Fig. 11.10 b). According to the line address there is conducted the reading process. The address field of the read line is compared with the tag address came from the processor. If the tag address matches the tag of the line, a signal $Hit = 1$ is generated to output the data word. The output word is determined by the displacement and is output using the multiplexer.

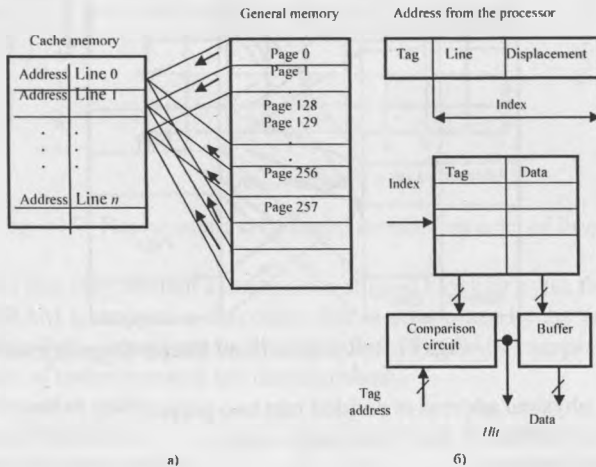


Fig. 11.10. Organization of cache memory with direct placement

The advantages of this scheme are the simplicity and low cost of implementation. The disadvantages include the low efficiency of such a cache due to the likely frequent reloading of lines. A compromise between the first and second circuits is a multiple associative cache or a partially associative cache (type-associated cache), see Fig. 11.11. With this method of organization of the

cache memory the lines are combined into groups, which may include 2,4,8, ... - lines. In accordance with the amount of lines in such groups, there are distinguished 2-input, 4-input, and etc. type-associative cache. Accessing the memory, the physical address is divided into three parts:

- displacement in the block (cache line);
- group (set) number;
- tag.

A memory block whose address corresponds to a particular group can be placed on any line of this group, and the corresponding value is placed in the line tag. Therefore, within the framework of the selected group, the principle of associativity is respected. However, this or that block can only fall into a strictly defined group, which is interconnected with the principle of direct mapping cache organization. For the identification of the cache miss the processor will need to check the tags of only one group of lines.

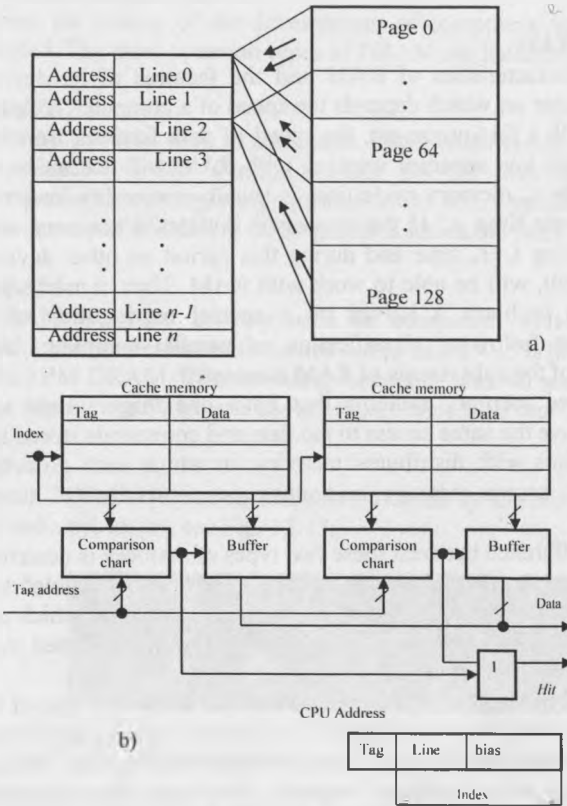


Fig. 11.11. Organization of type-associative cache memory

This scheme has the advantages of a fully associative cache (good memory utilization, high speed), and a direct access cache (simplicity and cheapness). That is why multiple associative cache is the most widespread.

Modern processors have a conveyor architecture, in which conveyor blocks work in parallel. Thus, the selection of the command and access to the command data is carried out at different stages of the conveyor, and the use of a separate cache allows to perform these operations in parallel.

The command cache can be implemented only for reading, therefore, it does not require the implementation of any recording back algorithms, which makes this cache easier, cheaper and faster.

Thus, all the latest IA-32 models, starting with Pentium, use the Harvard architecture to organize the first cache level.

A criterion for the effective operation of the cache can be considered in a decrease in the average access time to memory compared to a system without a cache.

11.4. RAM

The characteristics of RAM and the features of its device are the most important factor on which depends the speed of a computer. This is due to the fact that even with a fast processor, the speed of data fetching from memory may be slow. And this low speed of working with RAM will determine the speed of the computer. The t_m memory cycle time is usually noticeably longer than the central processor cycle time t_c . If the processor initiates a memory access, it will be occupied during t_c+t_m time and during this period no other device, including the processor itself, will be able to work with RAM. Thus, a memory access problem occurs. This problem is solved by a special organization of random access memory. The following classification of parallel computers according to the architecture of the subsystems of RAM is adopted:

- shared memory systems that have one large virtual memory and all processors have the same access to the data and commands stored in this memory;
- systems with distributed memory, in which each processor has its own local random access memory, and other processors do not have access to this memory.

The difference between these two types of memory is described in the virtual memory structure, that is, in the memory - how it is "visible" to the processor. Physically, memory is usually divided into parts, access to which can be organized independently. The difference between shared and distributed memory is in the way the address is interpreted.

It is often important for a programmer to know the type of the RAM of the computer on which he works. Indeed, the memory architecture determines the way of interaction between different parts of a parallel program. During the execution on a computer with distributed memory, the matrix multiplication program, for example, should create copies of the multiplied matrices on each processor, which is technically accomplished by sending a message containing the necessary data to

these processors. In case of a system with shared memory, it is enough only once to set the appropriate data structure, and then place it in the RAM.

In most modern computers, the random access memory is dynamic memory modules that contain semiconductor LSI SD and are organized according to the principle of random access devices

Massive RAM is created on dynamic memory modules, and static type memory is used to create cache memory in a microprocessor.

Dynamic memory type - DRAM is a set of memory cells that consist of capacitors and transistors located inside the semiconductor memory chips.

Modern RAM technologies mainly use two circuit solutions to improve DRAM performance:

- inclusion in the dynamic memory chips of a certain amount of static memory;
- synchronous operation of memory and CPU, i.e. using an internal conveyor architecture and alternating addresses.

Throughout the history of the development of computers, various types of RAM were created. The most common types of DRAM are listed below.

FPM DRAM

FPM DRAM - widely used in computers based on Intel-386 and Intel-486. With the occurrence of MP Pentium, EDO DRAM was supplanted. Its effectiveness is due to the conveyor organization of the MP. Typical access time at a system bus frequency of 66 MHz is 60 ns (35 ns is inside the line).

EDO DRAM

FPM DRAM has been actively used on computers with Intel Pentium processors and higher. Its performance turned out to be 10 - 15% higher compared to memory like FPM DRAM. The operating frequency was 40 and 50 MHz, the full access time was 60 and 50 ns, and the operation cycle time was 25 and 20 ns.

SDRAM

A feature of SDRAM (Synchronous DRAM) is the synchronous operation of memory chips and a processor, see Fig. 12.12.



Fig. 12.12. SDRAM module in a 72-pin SO-DIMM package

SDRAM has replaced FPM DRAM. The features of this type of memory were the use of a cycle generator to synchronize all signals and the use of conveyor information processing. At the same time, time delays during waiting cycles are

reduced, and data search is accelerated. This synchronization allows the memory controller accurately to know the data availability time. Thus, the access speed is increased due to the fact that data is available during each cycle of the timer. SDRAM technology allows the use of multiple memory banks that operate simultaneously, in addition to addressing with entire blocks. SDRAM chips have programmable parameters and their own sets of commands. The length of the batch of the reading-recording cycle can be programmed (1, 2, 4, 8, 256 elements). The cycle can be interrupted by a special command without losing data. Conveyor organization allows to initiate the next reading cycle before the end of the previous one.

The operating frequencies of this type of memory could be 66, 100 or 133 MHz, the full access time was 40 and 30 ns, and the operation cycle time was 10 and 7.5 ns.

Virtual Channel Memory (VCM) technology was used with this type of memory. VCM uses a virtual channel architecture that allows more flexible and efficient data transfer using register channels on a chip. This architecture is integrated into SDRAM.

SDRAM II (DDR)

Synchronous DRAM II, or DDR (Double Data Rate), became the development of SDRAM. DDR is based on the same principles as SDRAM, but included some enhancements that increased performance, see fig. 11.13. DDR made it possible to read data on the rising and falling edges of the cycle signal, performing two accesses during one request of the standard SDRAM, which increases the access speed by half compared to SDRAM (at the same frequency). DDR SDRAM operates at frequencies of 100, 133, 166 and 200 MHz, its full access time is 30 and 22.5 ns, and the duty cycle is 5, 3.75, 3 and 2.5 ns.

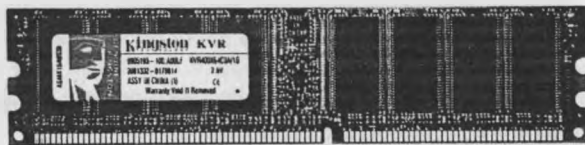


Fig. 11.13. Kingston DDR-SDRAM 1GB Module with 184 pins

RDRAM

The RDRAM memory type was developed by Rambus. The memory operating frequencies are 400, 600 and 800 MHz, the full access time is up to 30 ns, the operation cycle time is up to 2.5 ns.

The RDRAM technology is based on a multifunctional protocol for data exchange between microcircuits, which allows data transmission over a simplified bus operating at a high frequency. RDRAM is a system-integrated technology.

Rambus, first used in graphic workstations in 1995, uses the unique RSL technology (Rambus Signal Logic - Rambus Signal Logic), which allows the use

of data transfer frequencies up to 600 MHz on conventional systems and motherboards.

DDR2 SDRAM

DDR2 SDRAM, see fig. 11.14, began to be issued since 2004. DDR2 can work with a bus frequency of 200, 266, 333, 377, 400, 533, 575 and 600 MHz. In this case, the effective data transfer frequency will be 400, 533, 667, 675, 800, 1066, 1150 and 1200 MHz, respectively. Full access time - 25, 11.25, 9, 7.5 ns or less. Operation cycle time - from 5 to 1.67 ns.



Fig. 11.14. 1GB DDR2 module in 204-pin SO-DIMM package

DDR3 SDRAM

DDR3 is based on DDR2 SDRAM technologies with double increased frequency of data transfer via the memory bus. DDR3 memory modules are characterized by 40% lower power consumption compared to their predecessors (DDR2). Reducing the supply voltage is achieved through the use of 90-nm (first, then 65-, 50-, 40-nm) process technology for the production of microcircuits and the use of dual-gate transistors Dual-gate (which helps to reduce current leakage).

The frequency throughput lies in the range from 800 to 2400 MHz (frequency record - more than 3000 MHz), which provides greater throughput compared to all predecessors.

DDR3 memory microcircuits are made exclusively in BGA-type cases, see fig. 11.15.



Fig. 11.15. 4GB DDR3 Module

11.5. External computer memory

External (long-term) memory is a place of long-term data storage not currently used in the computer's RAM. External memory is energy independent. External storage media, in addition, provide data transport in cases where computers are not connected to a network (local or global).

In order to work with external memory, you must have a storage unit (a device that provides recording and (or) reading information) and a storage device - storage medium.

The main types of storage units:

- storage unit on flexible magnetic disks (HMD);
- hard disk drives (HDD);
- tape drives (NML);
- optical drives (CD, DVD, Blue-Ray, etc.);
- other

Storage devices are usually divided into types and categories in connection with their operating principles, operational and technical, physical, software, and other characteristics. So, for example, according to the operating principles, the following types of devices are distinguished: electronic, magnetic, optical and mixed - magneto-optical. Each type of device is based on its own technology for storing / reproducing / recording digital information.

The principle of operation of magnetic external storage devices is based on methods for storing information using the magnetic properties of materials. As a rule, magnetic storage devices consist of actual devices for reading/recording information and a magnetic medium onto which recording is directly carried out and from which information is read. Most often there are distinguished: disk and tape devices. The general technology of magnetic storage devices consists in magnetizing, by an alternating magnetic field, portions of a medium and reading out information encoded as areas of variable magnetization. Disk media, as a rule, are magnetized along concentric fields - paths located along the entire plane of the rotating medium. Magnetization is achieved by creating an alternating magnetic field using reading/recording heads. The heads are two or more magnetic controlled circuits with cores, the windings of which are supplied with alternating voltage. A change in the magnitude of the voltage causes a change in the direction of the lines of magnetic induction of the magnetic field and, when the medium is magnetized, means a change in the value of the information bit from 1 to 0 or from 0 to 1.

Disk devices are divided into flexible (Floppy Disk) and hard (Hard Disk) drives.

As a part of our tutorial, we only consider the features of HDD.

A hard disk (HDD - Hard Disk Drive or Winchester) is arranged as follows, see fig. 11.16, on a spindle connected to an electric motor, there is a block of several disks, above the surface of which there are heads for reading/recording information.

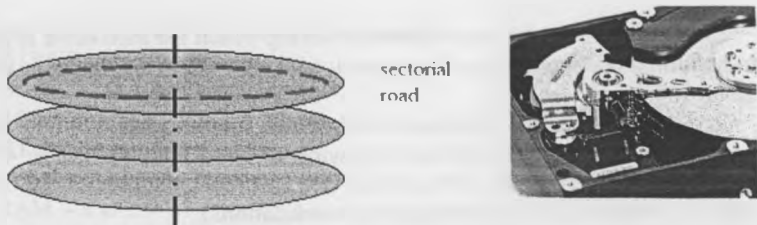


Fig. 11.16. Hard disk layout

During operation, the heads are located above the surface of the disks in the air flow that is created by the rotation of the disks.

The main property of disk magnetic devices is the recording of information on a medium onto concentric closed tracks using physical and logical digital coding of information. Flat disk media rotates during reading/recording, which ensures the maintenance of the entire concentric track, reading and recording is carried out using magnetic reading/recording heads, which are positioned along the radius of the medium from one track to another.

For the operating system, disk data is organized into tracks and sectors. The tracks (40 or 80) are narrow concentric rings on the disk. Each track is divided into parts called sectors. During the reading or recording, the device always reads or records an integer number of sectors, regardless of the amount of information requested. The sector size on a diskette is 512 bytes. A cylinder is the total number of tracks from which information can be read without moving the heads. Since the floppy disk has only two sides, and the floppy disk drive has only two heads, in the floppy disk there are two tracks per cylinder. A hard disk can have many disk plates, each of which has two (or more) heads, so many tracks correspond to one cylinder. A cluster (or data location cell) is the smallest disk area that the operating system uses recording a file. Typically, a cluster is one or more sectors.

Hard disk drives combine media and a reading/recording device in one case, as well as, often, an interface part called a hard disk controller.

The following are the main physical and logical characteristics of the HDD:

1. Type of interface - a set of communication lines, signals sent along these lines, hardware that supports the line, as well as exchange rules. Commercially available internal hard drives can use ATA (IDE and PATA), SATA, eSATA, SCSI, SAS, FireWire, SDIO, and Fiber Channel interfaces.

2. Capacity - the amount of data that can be stored by the drive. The capacity of modern hard drives (with a form factor of 3.5 inches) at the end of 2010 reaches 3 terabytes.

3. Physical size (form-factor). Most drives for personal computers and servers have a width of either 3.5 or 2.5 inches - the size of the standard mounts for them, respectively, in desktop computers and laptops. Also used formats are 1.8 inches, 1.3 inches, 1 inch and 0.85 inches.

4. Random access time - the time during which the hard drive is guaranteed to perform a reading or recording operation on any part of the magnetic disk - from 2.5 to 16 ms.

5. Spindle speed - the number of spindle rotations per a minute. Currently, hard drives are available with the following standard rotation speeds: 4200, 5400 and 7200 (laptops), 5400, 7200 and 10 000 (personal computers), 10 000 and 15 000 rpm (servers and high-performance workstations).

6. The amount of I/O operations per second. For modern disks, this is about 50 operations / sec for random access to the drive and about 100 operations / sec for sequential access.

7. Energy consumption (important for mobile devices).

8. The noise level. Silent drives are considered devices with a noise level of about 26 dB or less.

9. Data transfer rate (with sequential access). It is in the following ranges:

- internal disk area: from 44.2 to 74.5 Mb / s;
- external disk area: from 60.0 to 111.4 Mb / s.

10. The amount of buffer. A buffer is an intermediate memory designed to smooth out differences in reading / recording and transfer speeds over an interface. In modern drives, it usually varies from 8 to 64 MB.

Questions for self-control

1. What is the fastest computer memory?
2. What methods exist to reconcile the contents of the cache memory and general memory?
3. List the types of cache.
4. What circuitry solutions are used to improve DRAM performance?
5. In which chips of dynamic memory is the inclusion of a certain amount of static memory used?
6. Which dynamic memory chips use the internal conveyor architecture?

Test questions

1. The acronym DRAM stands for ...
 - A) Dynamic Random Access Memory
 - B) Dynamic Ramdon Access Memory
 - C) Denamics Rodman Acces Memory
 - D) Dynamic Random Access Memory

2. In dynamic memory, cells are created on the basis of ...
 - A) triggers
 - B) two stable state circuits
 - C) circuits with two unstable states
 - D) capacitor-based circuits

3. The volume of one memory module may be ...

- A) 512 GB
- B) 64 Mb
- C) 128 GB
- D) 312 Mb

4. RAM – it is ...?

- A) random access memory
- B) open storage device
- C) online reprogrammable memory
- D) online recording device

5. In static memory, elements (cells) are created on various options

- A) circuits with capacitors
- B) trigger circuits
- C) two steady state circuits
- D) single vibrator circuits

6. External memory serves:

- A) for storing operational, often changing information in the process of solving the problem
- B) for long-term information storage regardless whether the computer is working or not
- C) for information storage inside a computer
- D) for information processing at a given time

7. External memory devices are:

- A) floppy and optical drives
- B) plotters
- C) random access memory
- D) hard disk drives

8. When you turn on the computer, you heard a squeak that does not stop. You have determined that the sound comes from the video card. Question: what can make a sound?

- A) resistor
- B) the MP of the video card itself
- C) capacitor
- D) throttle

9. What is an elementary memory cell in DRAM?

- A) capacitor and transistor
- B) trigger
- C) single vibrator

D) streamer

10. What provides the process of memory regeneration?

- A) information update
- B) information recording.
- C) information deletion
- D) information rewriting

11. In order to store information for a long time, it should be recorded

- A) on hard drive
- B) in processor registers
- C) in RAM
- D) in memory

12. The main characteristics of drives and media:

- A) physical capacity
- B) the reliability of information storage
- C) manufacturer
- D) weight

13. External memory devices are ...?

- A) HDD and floppy disk drives (HDD and FDD)
- B) CD-ROM
- C) streamer
- D) plotter

14. What are the correct characteristics of external memory:

- A) energy independent, slow, can store a large amount of information
- B) energy dependent, fast, small in volume
- C) slow, energy dependent
- D) the reliability of information storage

15. According to the principle of action, what kind of memory does not happen?

- A) semiconductor
- B) magnetic
- C) optical
- D) permanent

16. In order to fill the gap between the RM and the OM in terms of volume and time of circulation there is used

- A) cache memory
- B) static memory
- C) external memory
- D) no correct answers

17. What memory is called long-term?

- A) operational
- B) constant
- C) external
- D) cache

18. What block is not in any memory?

- A) word search block
- B) words recording block
- C) bit circuits block
- D) control unit

19. For the operating system, data on disks is organized

- A) in tracks and sectors
- B) in rings and fields
- C) in blocks and rows
- D) in tables

20. According to the algorithms for mapping RAM into the cache, the following cache type does not exist:

- A) fully associative cache
- B) direct mapping cache
- C) multiple associative cache
- D) indirect mapping cache

CHAPTER 12. RISC PROCESSORS

In the 70s of the XX century, there was put forward the idea to create a microprocessor that operates with a minimum set of commands.

The main features of the RISC architecture with similar features of the CISC architecture are displayed below (Table 12.1).

Table 12.1

Key Features of RISC and CISC Architecture

CISC architecture	RISC architecture
Multibyte commands	Single Byte Commands
Low number of registers	A large number of registers
Difficult commands	Simple commands
One or less command per processor cycle	Multiple commands per processor cycle
Traditionally, one actuator	Multiple Actuators

One of the important advantages of the RISC architecture is the high speed of arithmetic calculations. RISC processors were the first ones that reach the level of the most common IEEE 754 standard, which establishes a 32-bit format for representing fixed-point numbers and a 64-bit "full accuracy" format for floating-point numbers. The high speed of arithmetic operations, combined with high accuracy of calculations, give RISC processors an advantage in speed compared to CISC processors.

Another feature of RISC-processors is a set of tools that provide non-stop operation of arithmetic devices:

- dynamic branch prediction mechanism;
- a large number of operational registers;
- multi-level internal cache memory.

Organization of the register structure is the main advantage and the main problem of the RISC. Almost any implementation of the RISC architecture uses triple processing operations in which the result and two operands are independently addressed – $R1: = R2, R3$. This allows to select the operands $R2$ and $R3$ from the addressable operational registers without a significant amount of time and to write the result of the operation into the register - $R1$. In addition, triple operations give the compiler more flexibility than typical double-case register-memory operations of the CISC architecture. In combination with high-speed arithmetic, RISC register-to-register operations become a very powerful way to increase processor performance.

However, a reliance on registers is the Achilles heel of the RISC architecture. The problem is that in the process of performing the task, the RISC system is repeatedly forced to update the contents of the processor registers, and in a minimum time, in order not to cause long downtime of the arithmetic device. For CISC-systems, this problem does not exist, since register modification can be conducted during the processing "memory-memory" format commands.

There are two approaches to solve the problem of register modification in the RISC architecture:

- 1) Hardware proposed in the RISC-1 and RISC-2 projects;
- 2) Software developed by experts of IBM and Stanford University.

The fundamental difference between them is that the hardware solution is based on the desire to reduce the time of procedure request by installing additional processor equipment, while the software solution is based on the capabilities of the compiler and is more economical in terms of processor hardware.

Key features of RISC processors [16]:

- reduced set of commands (from 80 to 150 commands);
- most commands are executed in 1 cycle;
- a large number of general registers;
- availability of rigid multi-stage conveyors;
- commands have a simple format and few addressing methods are used;
- the availability of a capacious separate cache;
- the use of optimizing compilers that analyze the initial code and partially change the order of the commands.

The largest developers of RISC processors are Sun Microsystems (SPARC architecture - Ultra SPARC), IBM (Power multi-chip processors, single-chip PowerPC - PowerPC 620), Digital Equipment (Alpha - Alpha 21064, 21164, 21264), Mips Technologies (Rxx00 - R family 10000), as well as Hewlett-Packard (PA-RISC architecture - PA-8000).

All third-generation RISC processors:

- are 64-bit and superscalar (at least 4 commands per cycle are launched);
- have internal conveyor units for floating point arithmetic;
- have a multi-level cache. Most RISC processors cache pre-decrypted commands;
- are manufactured according to CMOS technology with 4 layers of metallization.

For data processing, there are used a dynamic branch prediction algorithm and a register reassignment method, which allows the implementation of extraordinary execution of commands.

Improvement of the performance of RISC processors is achieved by increasing the frequency and by complicating the crystal circuit. Representatives of the first direction are Alpha processors of the DEC company, the most complex are Hewlett-Packard processors.

The reduction of the set of machine commands in the RISC architecture made it possible to place a large number of general purpose registers on the chip of the computing core. An increase of the amount of general-purpose registers made it possible to minimize the access to slow RAM, leaving only the operations of data reading from RAM to register and data recording from a register to RAM to work with RAM; all other machine commands use general-purpose registers as operands.

The main hardware blocks of the RISC MP architecture:

1. The instruction loading unit includes the following components: an instruction retrieval unit from the instruction memory, an instruction register where the instruction is placed after it is retrieved, and an instruction decoding unit. This step is called the instruction fetch step.

2. General purpose registers together with register control units form the second stage of the conveyor, which is responsible for reading operands of instructions. The operands can be stored in the instruction itself or in one of the general registers. This step is called the operand selection step.

3. An arithmetic-logical device and, if implemented in this architecture, a battery, together with control logic, which, based on the contents of the instruction register, determines the type of the performed microoperation. In addition to the instruction register, the data source can be a command counter, performing microoperations of a conditional or unconditional transition. This stage is called the executive stage of the conveyor.

4. A set of general-purpose registers, recording logic, and sometimes RAM, forms the stage of data storage. At this stage, the result of instructions executing is recorded to general purpose registers or to main memory.

The RISC processor uses at least 32 registers, often more than 100, while classic MPs use usually 8-16 general registers. As a result, the processor is 20% - 30% less has access to RAM that also increased the speed of data processing. In addition, the presence of a large amount of registers simplifies the work of the compiler in the distribution of registers for variables. The topology of the processor, made in the form of a single integrated circuit, has been simplified, the time for its development has been reduced, it has become cheaper.

The classic representatives of RISC architecture are the Alpha 21064 and 21264 processors, see fig. 12.1.

In RISC processors, the processing of a machine commands is divided into several stages, each stage is serviced by separate hardware and data transfer from one stage to the next is organized.

At the same time, the productivity increases due to the fact that several commands are executed simultaneously at different stages of the conveyor.

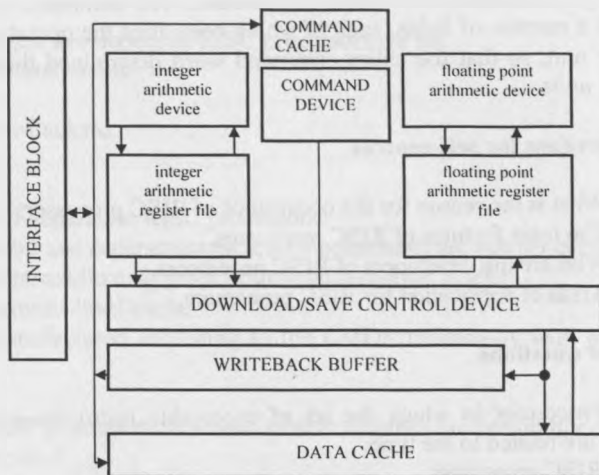


Fig. 12.1. Alpha 21064 processor structure

A typical command execution can be divided into the following steps:

1. command fetching - by the address specified by the counter commands;
2. decoding of the command;
3. execution of the operation, if the access to the memory is needed - calculation of the physical address;
4. memory access;
5. storing the result of WB.

Since the set of executable commands in processors with RISC architecture is reduced to a minimum, it is necessary to combine commands in order to implement more complex operations. Moreover, all commands have a fixed-length format (for example, 12, 14, or 16 bits), the command is fetched from memory and executed in one synchronization cycle. The command system of the RISC processor implies the possibility of equal use of all processor registers. This provides additional flexibility in a number of operations.

It should be noted that in the early 1990s, RISC architectures provide greater performance than CISC due to the superscalar and VLIW approaches, as well as due to the possibility of increasing the frequency and simplifying the crystal with freeing up the area under cache.

VLIW - a set of commands that implements horizontal microcode. Several (4-8) simple commands are packaged by the compiler into a long word. Such a word corresponds to a set of functional devices. **VLIW**-architecture is usually considered as a static superscalar architecture, since the parallelization of the code is performed at the compilation stage, and not dynamically at runtime.

Existing implementations of the **VLIW** approach, for example, **VLIW Multiflow**, are not widely used, except for - AP-120B/FPS-164/FPS-264 from Floating Point Systems company. These processors in the 80s were used at

conducting scientific and technical calculations. The command in these systems contained a number of fields, each of which controlled the operation of a separate processor unit, so that the entire command word determined the behavior of all processor units.

Questions for self-control

1. What is the reason for the occurrence of RISC processors?
2. The main features of RISC processors.
3. Who are the developers of RISC processors.
4. Areas of application for RISC processors.

Test questions

1. Processors in which the set of executable instructions is reduced to a minimum are related to the type:
 - A) RISC processors
 - B) Processors with Harvard architecture
 - C) CISC processors
 - D) Princeton architecture processors
2. The main features of RISC architecture
 - A) Single Byte Commands
 - B) A small number of registers
 - C) Simple commands
 - D) Multiple commands per processor cycle
3. What feature does not apply to the set of tools of RISC processors that provide non-stop operation of arithmetic devices:
 - A) dynamic branch prediction mechanism
 - B) a large number of operational registers
 - C) multi-level internal cache
 - D) a small number of operational registers
4. What is the Achilles heel of RISC architecture?
 - A) single byte commands
 - B) reliance on registers
 - C) addressing
 - D) dynamic branch prediction mechanism
5. The main features of RISC processors [16]:
 - A) a reduced set of commands (from 80 to 150 commands);
 - B) most of the commands are executed in 1 cycle;
 - C) a large amount of general purpose registers;
 - D) the presence of rigid multi-stage conveyors;

6. The largest developers of RISC processors are not

- A) Sun Microsystems
- B) IBM
- C) Hewlett-Packard
- D) Intel

7. All third-generation RISC processors:

- A) are 64-bit and superscalar (at least 4 commands are run per cycle)
- B) have internal floating point arithmetic conveyor blocks
- C) have a multi-level cache
- D) are manufactured according to the CMOS technology with 4 layers of metallization

8. The RISC processor command system assumes the possibility of ... using all processor registers

- A) equitably
- B) alternately
- C) parallel
- D) privileged

9. In RISC processors, machine command processing is divided into

- A) several steps
- B) several parts
- C) procedures
- D) no correct answers

10. All commands have a fixed-length format; command fetching from the memory and executing it is done in synchronization cycle.

- A) one
- B) four
- C) two
- D) three

CHAPTER 13. MOTHERBOARDS AND CHIPSETS

The *motherboard* is a complex multilayer printed circuit board on which the main components of the computer are installed (CPU, RAM modules and RAM controller, boot ROM, controllers of the basic input-output interfaces).

The motherboard, as a rule, contains connectors (slots) for connecting additional controllers, for which USB, PCI and PCI-Express buses are usually used.

Also, a chipset is installed on the motherboard - a special set of microcircuits. For example, in a PC, the chipset located on the motherboard acts as a connecting component that ensures the joint functioning of the memory subsystems, the central processor, input-output devices, etc.

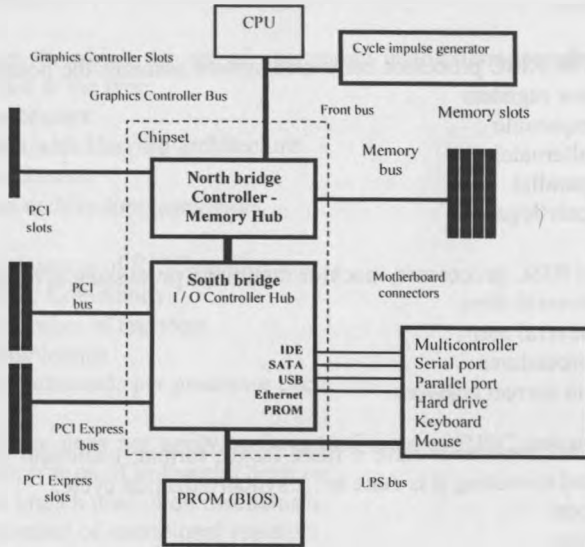


Fig. 13.1. Motherboard components

As a rule, the PC motherboard chipset consists of two main microcircuits, see fig. 13.1:

- 1) Memory Controller Hub (MCH) or the north bridge. MCH allows the CPU to communicate with memory. It connects to the CPU by a high-speed bus, see chapter 14. In modern CPUs, a memory controller can be integrated directly into the CPU. Some chipsets may integrate a graphics processor in the MCH;

- 2) I/O Controller Hub (ICH) or a south bridge. ICH provides the interface between the CPU and the hard drive, PCI cards, low-speed PCI Express interfaces, IDE, SATA, USB, etc.

Motherboards are characterized by a form factor (FF). FF - the standard that determines the size of the motherboard for the computer, the place of its location in

the case; the location of bus interfaces, input / output ports, the socket of the central processor (if any) and slots for RAM on it, as well as the type of connector for power supply connection.

The form factor has a recommendatory character. However, the vast majority of manufacturers prefer to comply with the specification. The main form factors of motherboards are listed below.

Deprecated: Baby-AT (1990); AT (1984); LPX (1987).

Modern: ATX (1995); microATX (1997); Flex-ATX (1999); NLX (1997); WTX (1999), CEB (2005).

Introduced: Mini-ITX and Nano-ITX (2003); Pico-ITX (2007); BTX (2004), MicroBTX and PicoBTX (2007).

There are motherboards that do not match any of the existing form factors. This is usually due to either the fact that the computer is highly specialized, or the desire of the motherboard manufacturer independently to manufacture peripheral devices for it, or the inability to use standard components (for example, Apple Computer, Commodore, Silicon Graphics, Hewlett Packard).

The fig. 13.2 shows the main components of the motherboard.

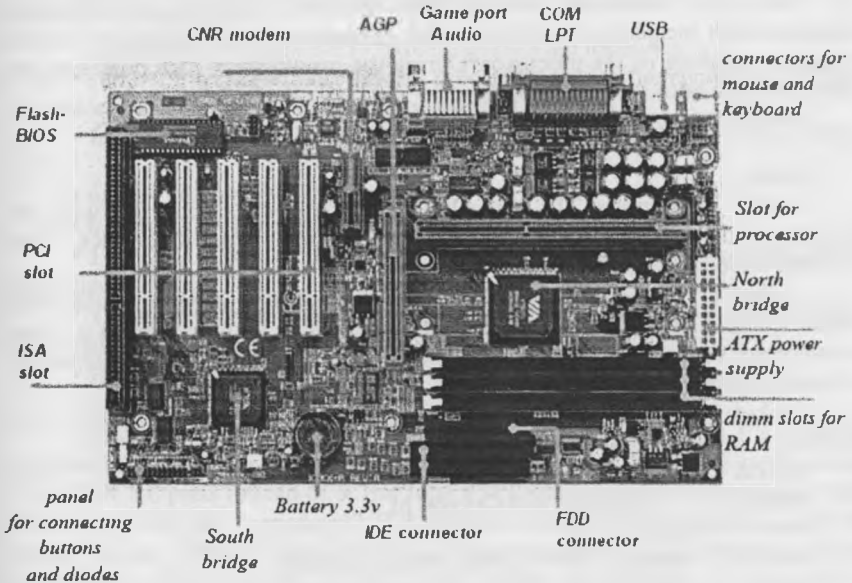


Fig. 13.2. The main components of the motherboard

As mentioned above, the main load of the chipset is to provide the central processor with data and commands, as well as to control the periphery - video

cards, sound system, RAM, disk drives and various input/output ports. In recent developments, the controllers of external devices began to be included in the composition of chipsets for integrated cards. Externally, the chipset chips look like the largest ones after the processor, according to the amount of pins - from several tens to two hundred. The name of the set usually comes from the marking of the main chip, for example, i810, i810E, i440BX, VIA Apollo pro 133A, SiS630, UMC491, i82C437VX, etc. In this case, there is used only the chip code inside the series: for example, the full name of SiS471 is SiS85C471. Recent developments use their own names; in some cases, this is a company name (INTEL, VIA, Viper) The type of the set mainly determines the functionality of the board: types of supported processors, structure and size of the cache, possible combinations of types and volumes of memory modules, support for power saving modes, the ability to configure parameters programmatically, etc. Several models of motherboards can be produced on the same set, from the simplest to the most complex with integrated port, disk, video, etc. controllers.

CPU socket - socket or slot connector designed to install a central processor. Using a connector instead of directly soldering the processor on the motherboard makes it easy to replace the processor to upgrade or repair the computer. The connector can be designed to install the actual processor or CPU-card. Each slot allows the installation of only a certain type of processor or CPU card.

The type of connectors constructively represents a plastic connector with a clamping latch located on the side of the connector case, designed to prevent spontaneous falling of the processor. During the installation of the processor, the latch should be raised as far as possible.

Below there are the main socket types for Intel and AMD processors.

Intel

Socket 7 - standard ZIF (Zero Input Force) - with a connector with 296 contacts, used by all P5 class processors - Intel Pentium, AMD K5 and K6, Cyrix 6x86 and 6x86MX and Centaur Technology IDT-C6.

Socket 8 - non-standard ZIF has 387 contacts and is incompatible with Socket 7, and is designed to install a Pentium Pro class P6 processor on it. Since the processor core and cache were combined on the same chip, its shape turned out to be rectangular rather than square like in Socket 7.

Socket 370 - non-standard ZIF is incompatible with either Socket 7 or Socket 8; it is designed to install the cheaper P6 Celeron prototype on it.

Socket 423, 478 - Pentium 4 and Celeron.

Socket 479 - Pentium M and Celeron M.

PAC418 - Itanium

PAC611 - Itanium 2, HP PA-RISC 8800 and 8900.

Socket B (LGA 1366) - Core i7 with integrated three-channel memory controller.

Socket H (LGA 1156) - Core i7/Core i5/Core i3 with integrated dual-channel memory controller.

Socket J (LGA 771) - Intel Xeon series 50xx, 51xx, 53xx, 54xx.

Socket M - Core Solo, Core Duo and Core 2 Duo.

Socket N - Dual-Core Xeon LV.

Socket P is a replacement for Socket 479.

AMD

Super Socket 7 - AMD K6-2, AMD K6-2 +, AMD K6-III, Rise mP6, Cyrix MII/6x86MX.

Socket 754 is a lower level Athlon 64 and Sempron with support for only single-channel DDR memory mode.

Socket 939 - Athlon 64 and Athlon 64 FX with dual channel support for DDR memory.

Socket 940 - Opteron.

Socket AM2 - with support for DDR2 memory, but not compatible with Socket 940.

Socket AM2+ - a replacement for Socket AM2, direct and backward compatibility with AM2 socket for all planned motherboards and processors.

Socket AM3 - with support for DDR3 memory. Replacement for Socket AM2 +.

Let consider the chipset device using the Intel P55 Express (2004) and Intel P67/H67 2010 examples.

Key features of the Intel P55 Express chipset are following:

- support for Core i7 and Core i5 processors during the connection to these processors via the DMI bus (with a throughput of ~ 2 GB/s);
- up to 8 PCIEx1 ports (PCI-E 2.0, but with a data transfer rate of PCI-E 1.1);
- up to 4 PCI slots;
- 6 Serial ATA II ports for 6 SATA300 devices (SATA-II);
- 14 USB 2.0 devices (on two EHCI host controllers) with the possibility of individual shutdown;
- Gigabit Ethernet MAC controller (connecting the i82577 / 82578 PHY controller via any free port of the PCIEx1 chipset);
- High Definition Audio (7.1);
- other things.

Compared with the previous generation of chipsets (ICH10R), the Intel P55 Express has mainly experienced only quantitative changes: 14 USB ports instead of 12, 8 PCI-E ports instead of 6. But Intel P55 Express has one more system-wide innovation: now the “peripheral” PCI Express controller also complies with the second version of the standard. However, it corresponds to it only in terms of new technologies of this version, but the speed is forcibly set at PCI-E 1.1 level - 250 + 250 MB/s (up to 250 MB/s in each of the two directions at the same time). Therefore, controllers with the first implementations of the expected soon USB 3.0 and Serial ATA III will have to be connected to ports operating at the same speed.

The new Intel P67/H67 chipsets (Fig. 13.3) have retained the previous level of functionality. But architecturally, the P6 is the only bridge connected to the processor via the DMI bus (now only the new version), as well as through a special

FDI interface in case of an "integrated" chipset, the main task of which is to ensure the operation of the bulk of peripheral devices on the motherboard.

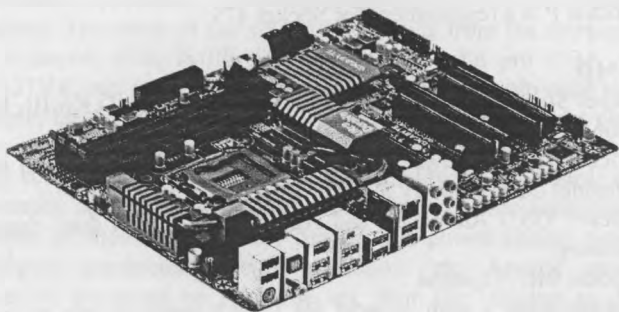


Fig. 13.3. Gigabyte P67A-UD5 motherboard based on Intel P67 chipset

The basic characteristics of the Intel P67/H67 chipset are following:

- support for all new processors based on the Sandy Bridge core during the connection to these processors via the DMI 2.0 bus (with a throughput of ≈ 4 GB/s);
- up to 8 PCIEx1 ports (PCI-E 2.0);
- 2 Serial ATA III ports for 2 SATA600 devices and 4 Serial ATA II ports for 4 SATA300 devices;
- 14 USB 2.0 devices (on two EHCI host controllers) with the possibility of individual shutdown;
- Gigabit Ethernet MAC controller and a special interface (LCI/GLCI) for connecting a PHY-controller (i82579 for implementing Gigabit Ethernet, i82562 for implementing Fast Ethernet);
- High Definition Audio (7.1);
- other.

Questions for self-control

1. The purpose of the motherboard?
2. The purpose of the south and north bridge.
3. What is a slot and form factor?
4. The main characteristics of the chipsets.
5. What main types of connectors for Intel and AMD processors do you know?

Test questions

1. What is a complex multilayer print circuit board on which the main components of a computer are installed?

- A) motherboard
- B) RAM
- C) processor
- D) Winchester

2. ... located on the motherboard serves as a connecting component that ensures the joint functioning of the memory subsystems, the central processor, input-output devices, etc.

- A) slot
- B) chipset
- C) bus
- D) connector

3. What are the characteristics of motherboards?

- A) memory capacity
- B) the amount of slots
- C) form factor
- D) amount of buses

4. What is the north bridge?

- A) I/O controller hub
- B) PROM (BIOS)
- C) cycle impulse generator
- D) memory hub controller

5. What is the south bridge?

- A) I/O controller hub
- B) PROM (BIOS)
- C) cycle impulse generator
- D) memory hub controller

6. Which motherboards do not correspond to the form factors?

- A) Apple Computer, Commodore, Silicon Graphics, Hewlett Packard
- B) Mini-ITX, Nano-ITX, Pico-ITX, BTX, MicroBTX, PicoBTX
- C) ATX, microATX, Flex-ATX, NLX, WTX, CEB
- D) Baby-AT, AT, LPX

7. What changes have occurred in the Intel P55 Express compared to the previous generation of chipsets (ICH10R)?

- A) only quantitative port changes
- B) only year of manufacture
- C) "peripheral" PCI Express controller complies with the second version of the standard
- D) speed of work

8. What provides the connection of the CPU to the nodes?

- A) north bridge
- B) form factor
- C) south bridge
- D) RAM

9. What is a socket?

- A) CPU brand
- B) Type of motherboard
- C) Video card
- D) CPU connector

10. What socket is currently in use?

- A) Socket LGA1156
- B) Socket LGA1366
- C) Socket LGA775
- D) Socket LGA771

CHAPTER 14. INTERRUPT AND EXCEPTION SYSTEM IN THE IA-32 ARCHITECTURE

14.1. CPU interrupt and exception system.

Interrupts and exceptions - events that indicate the occurrence of certain conditions in the system or in the task currently being performed that require processor intervention. The occurrence of such events forces the processor to interrupt the execution of the current task and to transfer the control to a special procedure or task called an interrupt handler or an exception handler. Various synchronous and asynchronous events in an IA-32 CPU-based system can be classified as follows [5, 13, 20], see fig. 14.1.

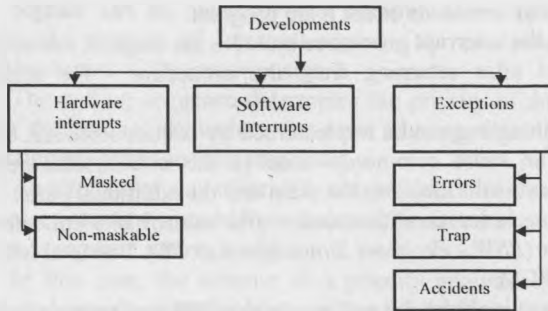


Figure 14.1. Classification of events in the IA-32 CPU-based system.

The interrupt system appeared in second-generation computer processors, which were used mainly as software devices for controlling various objects.

The main reasons for the occurrence of the interrupt system are:

- the desire to reduce computer downtime during an emergency in the processor (attempts to divide by zero, the use of a nonexistent command, device failure, etc.),
- the desire to load the processor with useful work, at a time when it expects a signal from a control object, i.e. the desire to realize the background work of the computer.

Despite the fact that the interrupt system occurred to solve two problems of a different nature, modern computers use a single interrupt mechanism. (In the history of the development of specialized computer architectures, there were examples of the use of two separate interrupt systems.)

In modern computers, despite the unified interrupt mechanism, there are distinguished features of the interrupt processing procedure in emergency situations in the processor and when interrupt signals come from external devices.

With the transition to multi-program modes of operation, the interrupt system has become an indispensable component of all computers.

The interrupt system is an effective way to implement the control functions of the operating system in order to support the specified computer operating modes such as a hardware-software complex.

The interrupt procedure consists in switching to the interrupt processing program with the possibility of returning to the main program.

In this statement, the interrupt procedure coincides with the "request for a procedure with return" procedure. The execution of each command (except NOT) changes the aftereffect, recording the result of the execution to the memory, GPR, or to the status register.

Request commands for a procedure with return are arranged by the programmer taking into account this aftereffect. But interrupt is an event, in general case, random in relation to the program. The interrupt processing program can change the context of the program, which can lead to a violation of the correct execution of subsequent commands of the main program.

For this reason, the interrupt procedure includes the stage of maintaining the program context, and after returning from the procedure - the stage of its restoration.

The context maintaining can be implemented by circuit, software, or circuit-software methods. The most commonly used is the circuit-software context maintaining. The context is divided into the main and the additional parts.

The main part includes condition codes and control bits collected in the program status register (*PSW - Program Status Word or PS*). For Intel MP - this is the flag register (*EFLAGS*).

The additional part includes the contents of the GPR and memory cells.

The main part is saved in hardware, usually on the stack. The contents of the GPR are saved and restored by the interrupt processing program, and only the GPRs used by the interrupt processing program are saved and restored.

Since the address spaces of the interrupt processing program and the main program are always separated, the storage of memory cells in the interrupt procedure is not provided.

Trap - this is the reaction of the system to the occurrence of abnormal situations in the processor: an attempt to divide by zero, identifying a nonexistent address, etc. Special programmable interrupt commands also belong to traps.

Interrupt - this is the reaction of the system to the request of an external device (interrupt signal) for performing a certain control procedure by the processor.

Therefore, traps are detected by fixation schemes for special situations (regular or non-regular in the processor) requiring control by the operating system. Moreover, with each scheme for fixing special situations, certain interrupt processing programs are associated.

Interrupt signals are signals from external, relative to the processor, devices at the moment of time that require control actions from the operating system.

The main issues for implementing the interrupt system are:

- receiving interrupt signals and traps. And then, highlighting the priority signal;

- determination of the moment of the interrupt procedure completion;
- selection of the interrupt procedure (the models of context maintaining and switching to the interrupt processing program);
- selection of the return procedure from the interrupt procedure.

Interrupt signals (requests) are received to determine the specific interrupt processing program. In most computers, the interrupt system provides 256 programs for handling situations that caused the interrupt request. Each interrupt processing program is associated with a specific interrupt source. Determination of the source of an interrupt signal is the definition of an interrupt processing program.

A single-processor system cannot execute interrupt processing program simultaneously on multiple requests. Therefore, there is used the priority allocation of the received interrupt signals. There can be some variants.

Signals can be received on a common wire. In this variant, in order to identify the sources of interrupt signals, the processor sequentially polls all possible sources of the interrupt signal.

The polling sequence determines the priority of the device interrupt signal. During the polling, all devices that send interrupt signals, determine the interrupt processing programs in their responses.

An alternative way of priority determination of the interrupt signal source is to receive interrupt signals from external devices via individual wires and to fix them on a separate interrupt register.

In this case, the scheme of a priority analysis of the received signals is introduced into the interrupt system. The fig. 14.2 presents a simplified scheme of the interrupt system mechanism. In this scheme, only two extreme (according to the implementation of the allocation of signals) classes of the interrupt system are considered.

This scheme only sets the hard priority of interrupt signals. But, the importance of priority handling interrupt causes can change. In order to do this, the mask register is introduced into the interrupt system for those interrupts which priorities is desirable to change.

The interrupt signal, in general, is not an immediate signal. Usually, an interrupt signal (from external devices) or a trap (from operation control circuits) is simply fixed in the corresponding category of the interrupt register.

The transition to the situation processing program always has a certain delay (reaction time of the interrupt system).

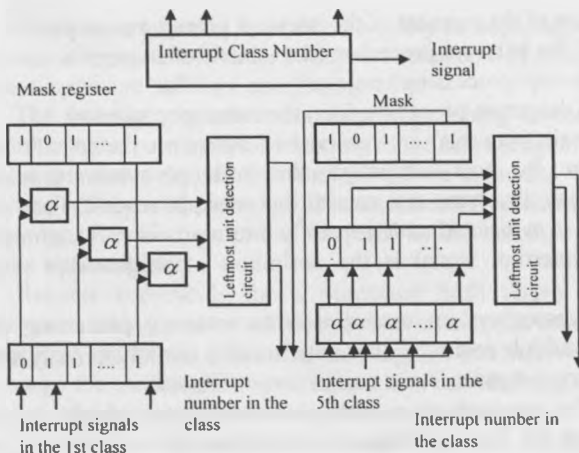


Figure 14.2. A simplified diagram of the system interrupt mechanism

There are several possible options for choosing the beginning of the interrupt procedure start.

1. Programs consist of sequences of autonomous sections (procedures) with a minimum set of transmitted information. This is usually the contents of memory or GPR. These are points with a minimum of context. The programmer can mark these points with special commands, for use as interrupt commands. But this solution significantly increases the average response time of the interrupt system.

2. The average reaction time of the interrupt system is significantly (at times) reduced if the interrupt procedure is performed immediately after the end of the current command.

3. There is still the possibility of reducing the reaction time of the interrupt system if the interrupt procedure is started immediately after the end of not the command, but of the current part of the microoperation, during which the interrupt request appeared. But at the same time, it is necessary to additionally save the result of the current microoperation.

In the vast majority of computer architectures, the interrupt procedure is performed at the end of the current command.

The picture 14.3 shows a simplified diagram of the processor cycle of the command execution with fixing a trap (emergency situation) and checking the interrupt signal.

Execution of commands in the processor is cyclical. We begin the review with the command selection procedure. This is the beginning of the cycle. The result of this stage is the selection and analysis of the command fields, operand addresses and of the result. The next steps are the steps of operands fetching. Subsequent steps decrypt the operation code of the command and transmit the

command to the execution blocks. The last step is to save the result and to record the result feature in the status register.

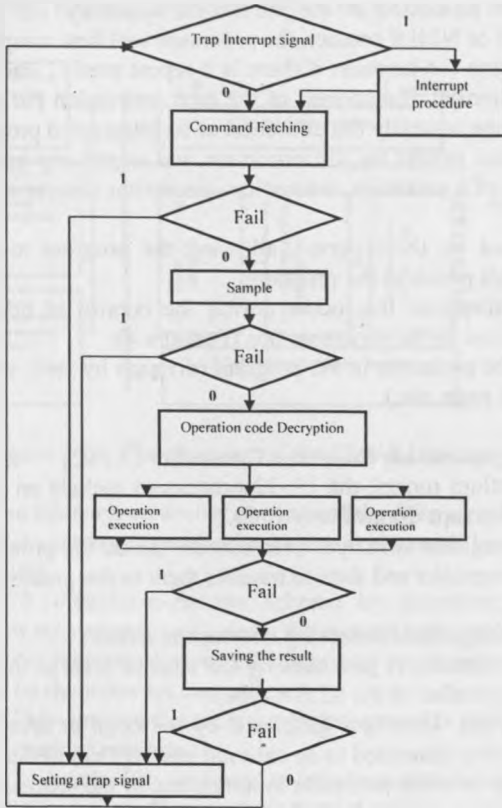


Figure 14.3. Processor cycle diagram of the command execution with fixing a trap

External interrupts are generated by a hardware signal from peripheral device when it requires maintenance. The processor determines the need for processing an external interrupt by the presence of a signal at one of the *INTR#* or *NMI#* contacts. When a signal occurs on the *INTR#* line, an external interrupt controller (for example, 8259A) should provide the processor with a vector (number) of the interrupt.

Interrupts that are generated when a signal occurs at the *INTR#* input are called masked hardware interrupts. The *IF* bit in the flag register allows to block (*mask*) the processing of such interrupts.

Interrupts generated by the *NMI#* signal are called non-maskable hardware interrupts. Non-maskable interrupts are not blocked by the *IF* flag. While the non-maskable interrupt handler is executing, the processor blocks the receipt of non-

maskable interrupts until the IRET instruction is executed in order to prevent the simultaneous processing of several non-maskable interrupts.

Interrupts are always processed at the instruction boundary, i.e. when a signal occurs on the INTR # or NMI # contact, the processor will first complete the currently executing instruction (or iteration if there is a repeat prefix), and then it will start processing the interrupt. The address of the next instruction put into the handler stack allows to resume correctly the execution of an interrupted program.

Exceptions are internal events for the processor and signal any erroneous conditions at the execution of a particular instruction. Exception sources are three types of events:

- exceptions generated by the program, allowing the program to control certain conditions at specified points in the program;
- machine control exceptions that occur during the control of operations inside the chip and transactions on the processor bus (Pentium 4);
- errors detected by the processor in the program (division by zero, violation of protection rules, lack of a page, etc.)

14.2. Advanced Programmable Interrupt Controller (APIC)

Starting with the Pentium model, the IA-32 processors include an internal Advanced Programmable Interrupt Controller (APIC).

APIC is designed to register interrupts from sources inside the processor or from an external interrupt controller and then to transfer them to the processor core for subsequent processing.

The internal APIC distinguishes following interrupt sources.

1. From local devices. Interrupts generated by the edge or level of the signal (for example, an interrupt controller of the 8259A type).
2. From external devices. Interrupts generated by the edge or level of the signal that comes from a device connected to an external interrupt controller.
3. Interprocessor (IPI). In multi-processor systems, one of the processors can interrupt the other with a message on the APIC bus (or on the system bus in Pentium 4).
4. From the APIC timer. The internal APIC contains a timer that can be programmed to generate an interrupt when a specified count is reached.
5. From a performance monitor timer.
6. From the temperature sensor. Many processors have an integrated temperature control unit that can be programmed to generate interrupts.
7. Internal APIC errors. The internal APIC can generate interrupts when internal error conditions occur (for example, trying to access a non-existent APIC register).

Sources 1, 4, 5, 6, 7 are considered local interrupt sources and are served by a special set of APIC registers called the *local vector table (LVT)*.

Sources 2 and 3 are processed by the APIC via a message mechanism transmitted either through a dedicated three-wire APIC bus or through a system bus (for example, PCI).

14.3. Interrupt Handling Based on 8259A Controller

The 8259A interrupt controller is a device that implements up to eight levels of interrupt requests, with the ability to mask and change the interrupt service order.

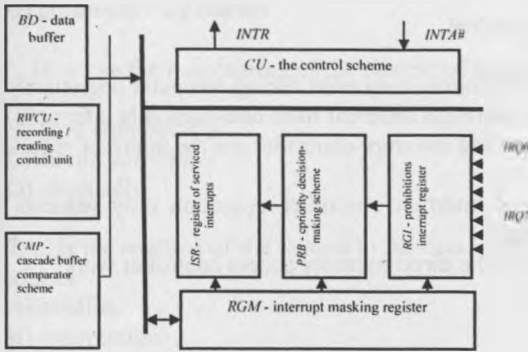


Figure 14.4. The structure of the 8259A interrupt controller

The interrupt controller consists of the following blocks, look at pic.14.4:

RGI - interrupt prohibition register; stores all levels to which IRQx requests are received.

PRB - decision-making scheme by priorities; the circuit identifies the priority of the requests and selects the request with the highest priority.

ISR - register of serviced interrupts; stores interrupt request levels that are serviced by the interrupt controller.

RGM - interrupt masking register; provides the prohibition of one or more lines of interrupt requests.

BD - data buffer; designed to interface with the system data base.

RVCU - recording/reading control unit; receives control signals from the microprocessor and sets the mode of operation of the interrupt controller.

CMP - scheme of the cascade buffer comparator; used to include multiple controllers in a system.

CU - control circuit; It generates interrupt signals and forms a three-byte CALL command for output on the data bus.

One 8259A controller can handle interrupts from 8 sources. Cascade controllers switching is used to service more devices.

Since more than one interrupt request can be received at any given time, the interrupt controller has a priority scheme. In general mode - full execution mode - as long as the digit in the **ISR** register corresponding to the requested interrupt is set, all subsequent requests with the same or lower priority are ignored, only requests with a higher priority are confirmed.

In cyclic mode, there is used a circular order of priority. The last served request is assigned the lowest priority, the next in a circle - the highest, which guarantees the maintenance of other devices until the next service of this device.

The controller allows masking of individual interrupt requests, which allows devices with lower priority to be able to generate interrupts. Special masking mode allows interrupts of all levels except the levels currently being serviced.

Questions for self-control

1. What exceptional situations may arise during computer operation?
2. How are masked interrupts different from non-maskable interrupts?
3. In which register of the interrupt controller are the interrupt request levels being maintained?
4. In what modes of interrupt controller operation only requests with a higher priority are confirmed?
5. In what modes does the direct memory access controller work?

Test questions

1. ... are events that indicate the occurrence of certain conditions in the system or in the current task that require processor intervention
 - A) interrupts and exceptions
 - B) errors
 - C) conflicts
 - D) viruses
2. What types of events do not exist in the IA-32 CPU-based system?
 - A) hardware interrupts
 - B) software interrupts
 - C) exceptions
 - D) conflicts
3. When did the interrupt system occur?
 - A) in first-generation computer processors
 - B) in second-generation computer processors
 - C) in third-generation computer processors
 - D) in fourth-generation computer processors
4. What does not belong to the main reasons for the occurrence of the interrupt system?
 - A) the desire to reduce computer downtime during an emergency in the processor
 - B) the desire to load the processor with useful work, while it is waiting for a signal from the control object
 - C) the desire to implement the background work of computers
 - D) the desire to fix errors faster

5. In this connection, the interrupt system has become an indispensable component of all computers

- A) using multi-core processors
- B) with increased processor speed
- C) with the transition to multi-program modes
- D) all answers are correct

6. How can the maintaining of the context of the program be implemented?

- A) circuit
- B) programmatically
- C) circuit software
- D) physically

7. ... is the reaction of the system to the occurrence of abnormal situations in the processor

- A) conflict
- B) interruption
- C) trap
- D) error

8. The main issues of the implementation of the interrupt system do not include

- A) receiving interrupt signals and traps
- B) interrupt stop
- C) determining the time of the interrupt processing program execution
- D) choice of interrupt procedure

9. Temporary switching of the microprocessor to the execution of another program with subsequent return to the interrupted program is called

- A) interrupt
- C) interrupt request
- C) a malfunction in the operating system
- D) there is no right answer

10. Interrupt may be caused by

- A) a non-standard situation in the microprocessor
- C) pressing a key on the keyboard
- C) the receipt of signals from external devices
- D) all answers are correct

11. Sources of exceptions are not

- A) external devices
- B) program-generated exceptions
- C) machine control exceptions
- D) errors detected by the processor in the program

12. What is APIC?

- A) interrupt controller
- B) special program
- C) microprocessor
- D) external device

13. What interrupt sources are not considered local?

- A) from local devices
- B) interprocessor
- C) from the performance monitor timer
- D) from the temperature sensor

14. What are the interruptions that are generated when a signal is received at the INTR# input

- A) maskable hardware interrupts
- B) software interrupts
- C) non-maskable hardware interrupts
- D) exceptions

15. Interrupts generated by the NMI# signal are called

- A) maskable hardware interrupts
- B) software interrupts
- C) non-maskable hardware interrupts
- D) exceptions

CHAPTER 15. TYPES AND CHARACTERISTICS OF INTERFACES

15.1. System interfaces

An interface is hardware and software (connection elements and additional control circuits, their physical, electrical and logical parameters), designed to interface systems or parts of a system (programs or devices). Interface refers to the following functions:

- issuing and receiving information;
- data transfer control;
- coordination of the source and receiver of information.

System interfaces, like others, also have different specifications, respectively their characteristics will be different.

SATA - (Serial ATA) - serial transfer interface, designed to work with data storage devices. The interface connector has seven pins. At the moment, the interface has three system standards and one for connecting external devices.

- **SATA 1** - the first version of the standard, its throughput is 1.2 Gb/s. The bus operates at a frequency of 1.5 GHz.

- **SATA 2** - became the second version of this standard, it works twice as fast as its predecessor, its throughput is 2.4 Gb/s and 3 GHz bus frequency.

- **SATA 3** - the newest standard, in theory it is capable of transmitting information at speeds of 6Gb/s, but in practice only 4.8 Gbit/s is achieved. However, developers want to name this specification of the interface as - SATA 6Gb/s.

IDE - (Integrated Drive Electronics) - a parallel interface designed to connect information storage devices to the motherboard. The IDE interface was released in 1986, its developer was not surprisingly the company WD (Western Digital). The maximum supported disk capacity for the IDE was 137 GB.

ATA - (Advanced Technology Attachment) - almost the same IDE, only in a more updated version. There are 2 versions of the interface, with a 40-wire and with 80-wire loop. In the first case, the maximum data transfer rate is 66 MB/s, in the second - from 66.7 MB/s.

PCI - (Peripheral component interconnect) - an interface, or rather, even a bus for input and output between the motherboard and peripheral devices. The first version of PCI occurred in 1992, its developer was the processor giant - Intel. The first version worked at a frequency of 33 MHz, and the throughput was 80 MB/s. The bus can be either 32 or 64 bit, for example, Macintosh computers often use the 64 bit bus. The maximum throughput of the PCI interface today is 4096 MB/s.

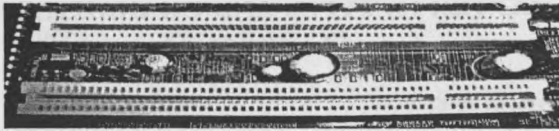


Figure 15.1. 32-bit PCI slots on the motherboard

On one PCI bus there can be no more than four devices (slots). A PCI bus bridge (PCI bridge) is the hardware used to connect a PCI bus to other buses. Host Bridge - the main bridge - is used to connect PCI to the system bus (processor or processor bus). Peer-to-Peer Bridge – is used to connect two PCI buses. Two or more PCI buses are used in powerful server platforms - additional PCI buses increase the number of connected devices.

The PCI bus interprets all exchanges as packet: each frame begins with an address phase, followed by one or more data phases. The amount of data phases in a packet is uncertain, but limited by a timer that determines the maximum time at which the device can use the bus. Each device has its own timer, the value for which is set at bus devices configuration.

Each exchange involves two devices - the initiator of the exchange and the target device. Arbitration of requests for the use of the bus is carried out by a special functional unit, which is a part of the chipset of the motherboard. To coordinate the performance of the devices participating in the exchange, two readiness signals *IRDY#* and *TRDY#* are provided.

The bus has versions with a power supply of 5 V, 3.3 V. There is also a universal version (with switching lines + V I/O from 5 V to 3.3 V). The keys are the missing rows of pins 12, 13 and 50, 51. For the 5 V-slot, the key is located at the pins 50, 51; for 3 V - 12, 13; for universal - two keys: 12, 13 and 50, 51. The keys do not allow to install the card in the slot with the wrong voltage. The 32-bit slot ends with A62/B62 contacts, the 64-bit slot ends with A94/B94.

Unlike other bus adapters, the components of PCI cards are located on the left surface of the boards. For this reason, the extreme PCI slot usually shares the adapter footprint with a neighboring ISA slot (Shared slot).

The PCI bus was until recently the second (after ISA) in terms of popularity of the application. In modern systems, ISA buses are abandoned, and the PCI bus goes to the main positions. Some companies produce prototype cards for this bus, but, of course, it is much more difficult to complement them with a peripheral adapter or a device of their own design than an ISA card. It requires more complex protocols and higher frequencies (8 MHz for the ISA bus versus 33 or 66 MHz for the PCI bus). The PCI bus also has poor noise immunity, so it is still relatively rarely used to create measurement systems and industrial computers.

Some system (motherboard) boards have a small connector called Media Bus. It is located behind the PCI bus connector of one of the slots. The signals from a regular ISA bus are output to this connector, and it is designed so that an inexpensive sound card chipset designed for the ISA bus can be placed on a

graphics adapter with a PCI bus. This connector, and even such combined audio-video cards, are not widely used.

AGP - (Accelerated Graphics Port) - interface for connecting a video card to the motherboard, see fig. 15.2. AGP specifications occurred in 1997, then Intel released the first version of the description, which includes two speeds: 1x and 2x. In the second version (2.0), AGP 4x appeared, and in 3.0 - 8x.

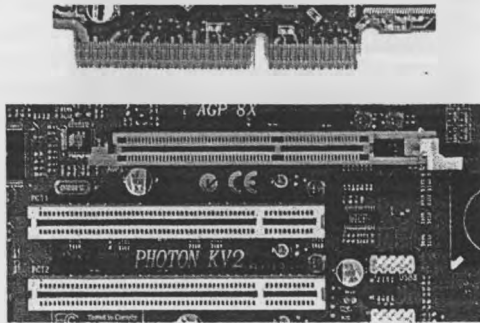


Figure 15.2. AGP Interface

Let consider all the variants in more detail:

AGP 1x is a 32-bit channel operating at a frequency of 66 MHz, with a throughput of 266 MB/s, which is two times higher than the PCI band (133 MB/s, 33 MHz and 32-bit).

AGP 2x is a 32-bit channel operating with doubled throughput of 533 MB/s at the same frequency of 66 MHz due to data transmission on two fronts, similar to DDR memory (only for the direction to the video card).

AGP 4x is the same 32-bit channel operating at 66 MHz, but as a result of further tricks the quadruple “effective” frequency of 266 MHz was reached, with a maximum throughput of more than 1 GB/s.

AGP 8x - additional changes in this modification made it possible to obtain a throughput up to 2.1 GB/s.

PCI Express (PCIe) - an interface that follows the aforementioned AGP. By the principle of operation of the software model, PCIe is similar to PCI. In this case, serial data transmission is used. Intel was engaged in the development; devices with this interface were sold in late 2002. In this case, the amount of modifications is greater than of AGP, there are 16x and 32x. However, in everyday life, only the 16x modification is used almost everywhere. The PCI Express slots in width and shape are not very different from PCI and are located in the same places on the motherboard, see fig. 15.3.

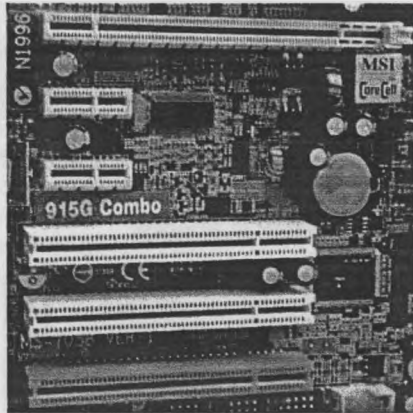


Figure 15.3. PCIe slots on the motherboard

PCI Express is created on the principles of simplex technology, which means that the signals go simultaneously, in opposite directions and along separate pairs of wires - a total of two pairs, called a line. The standard declares the throughput of a simplex line at around 2.5 Gbit/s in one direction or, respectively, 5 Gbit/s in both directions. However, these values are scalable.

The indicated throughput values are ideal - that is, in real life, unfortunately, they are not achieved.

When the line connecting two points (sender and addressee) is initialized, special initial sequences are sent. In these sequences, connection parameters can be encoded. During a communication session, such troubles can arise as a mismatch between the operation of the cycle generators of the transmitting and receiving sides. In order to correct the situation, special corrective sequences are transmitted periodically in the data flow. The degree of the need for the introduction of corrective sequences is determined on the basis of the difference between the cycle indicators of the generators.

In any case, additional coding sequences, as well as the need for introducing redundancy for the purpose of synchronization, negatively affect the performance, since they inevitably entail temporary losses. This way is not suitable for real-time communication, so the developers of such critical latency systems will have to invent their own exchange protocols.

We can say that at this stage the sequential transmission ends. The only PCI Express line connection, two pairs of wires, is not enough to provide high throughput. Therefore, the lines are usually arranged in a row - they can be 32, 16, 12, 8, 4 and 2. As a result, the entire sequence of data that needs to be transmitted is distributed to all available lines with a fan - transmission is parallel, but not synchronous. If there are 12 lines, then the first byte of the data block is transmitted on the first line, the second on the second, and so on, and the thirteenth byte again on the first. Theoretically, a bus with 32 lines is capable to have a

throughput of 20 Gb/s, from which we subtract 20% - 16 Gb/s, or 8 Gb/s in each direction.

A higher level of the PCI Express stack is responsible for the correct data transmission. Having received a data packet for transmission from the highest level of the hierarchy, the Data Link algorithm attaches the sequence number and its checksum to the last one. In addition, Data Link is also responsible for informing the rest of the stack levels about the state of the communication channel. The third important Data Link function is power management.

The most intelligent PCI Express layer is the Transaction Layer, which uses four different address spaces. This is memory (the address can be either 32- or 64-bit), messages, configuration, and I/O.

In addition to traditional energy-saving requirements, the PCI Express standard also has exclusive power management mechanisms - these are ASPM (*Active State Power Management*). ASPM has a fairly large autonomy and is able to put the device into optimal mode without instructions from above (from the software side). The PCI Express standard considers a device inactive if there was no data exchange with it for a period of 7 microseconds. As soon as there is a need for exchange, the device returns to working condition.

15.2. Drive Interfaces

The drive interfaces connect the drive itself with a controller connected to any system bus or data transfer interface. Nowadays, mainly parallel interfaces are used to connect disk and other drives. Omitting the long-obsolete ST506/412 interfaces (Shugart, Seagate Technology) and ESDI (Enhanced Small Computer Interface), we consider the two most common interface families: IDE (ATA-1, ATAPI, EIDE, ATA-3) and SCSI (Fast/Wide, Ultra 2, Fiber Channel and SSA).

ATA-1 IDE is historically the first IDE interface for hard drives with support for 2 devices on the bus.

The first implementation of the IDE interface was the ATA-1 version, which supported only two devices on the bus, and these devices had to be hard disks. In accordance with the specification, two devices can be connected to one IDE connector using their connection in the form of a daisy chain according to the "master" - "slave" scheme. The operation mode of the device (master or slave) is set on these devices using a mechanical switch. Therefore, primitive addressing of devices is implemented on the IDE bus.

The IDE interface supported PIO Mode 1 and 2, as well as DMA mode. The maximum data transfer speed on the bus in PIO Mode 2 was 8.3 MB/s.

The specification of the EIDE interface, having undergone changes aimed at improving the performance and reliability of data transmission, is called ATA-3 (or Fast ATA-3). Firstly, termination is introduced into the transmission line. Secondly, the support of the SMART (Self-Monitoring Analysis and Reporting Technology) hard drive failure prevention technology has been provided. The next step in the development of the IDE interface was the Quantum UltraATA specification, which is currently supported by the vast majority of new models of

drives and motherboards. It uses the new UltraDMA data transfer protocol (synonyms: UDMA, UltraDMA/33, UDMA/33). Due to the transmission of information along the edge and the cutoff of the cycle signal, the transmission speed in UltraATA was 33.3 MB/s, which is two times higher than the achieved in Multi Word DMA Mode 2. Due to the maintenance of the previous frequency, there is ensured backward compatibility with existing IDE standards. The UltraDMA/33 protocol actively uses Bus Mastering DMA to transfer data, which reduces processor load. In order to verify data integrity, there are used CRC (Cyclical Redundancy Code) control codes. The length of the interface cable should still be less than 45 cm (in practice it is recommended to use the shortest possible cable, but not shorter than 15 cm between the individual devices).

As a further development of UltraDMA, Quantum, with support from Intel and Western Digital, has proposed the UltraDMA/66 specification with a peak performance of 66 MB/s. Nowadays, full support for manufacturers has received the new ATA/100 protocol at speeds up to 100 Mb/s. The great interest has the occurrence of drives with the SerialATA interface. The main advantages of this interface are the combination of high data transfer speeds (up to 187.5 Mb/s in the first models) using convenient thin cables.

The SATA/150 standard provides the throughput up to 1.5 Gb/s (excluding 8B/10B encoding). The SATA/300 standard provides the throughput up to 3 Gb/s (excluding 8B/10B encoding). Each device operates on a separate cable. The standard provides for hot swapping of devices and the function of the command queue. SATA devices use two connectors: 7-pin - for connecting a data bus and 15-pin - for connecting power, see fig. 15.4 a) and b). Data is transmitted in duplex mode over two pairs of wires (one pair for reception, the other for transmission) using differential signal coding.

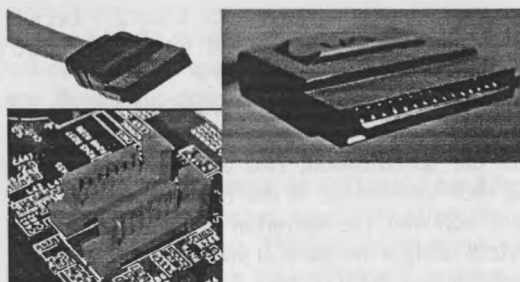


Figure 15.4. 7- and 15-pin Serial ATA data connector

In addition to the listed interfaces, universal peripheral interfaces are used for connecting drives, which will be discussed in the next section.

15.3. Interfaces SCSI, RS-232C, IEEE 1284, USB, FireWire

The *SCSI* interface was developed in the late 1970s and was proposed by Shugart Associates. The first standard for this interface was adopted in 1986.

SCSI defines only the logical and physical layer. Devices connected to the SCSI bus can play two roles: Initiator (master) and Target (slave), and at the same device can be either a master or a slave. Up to eight devices can be connected to the bus. Each device on the trunk has its own address (SCSI ID) in the range from 0 to 7. One of these devices is the SCSI host adapter. He is usually assigned as SCSI ID = 7. The host adapter is designed to communicate with the processor. The host adapter, as a rule, has connectors for connecting both internal and external SCSI devices, see the fig. 15.5.

The SCSI standard defines two signal transmission methods - common mode and differential. In the first case, the signals on the lines have TTL levels, while the cable length is limited with 6 m. SCSI bus versions with differential signal transmission ("current loop") make it possible to increase the bus length by 25 m.



Figure 15.5. SCSI-Terminator Connectors

The SCSI-2 standard was proposed in 1989 and existed in two versions - Fast SCSI and Wide SCSI. Fast SCSI is characterized by the double throughput (up to 10 MB/s). Wide SCSI in addition has doubled the bus width (16 bits), which allows to achieve transfer speeds of up to 20 MB/s. In this case, the maximum cable length was limited with three meters.

Also in this standard, a 32-bit version of Wide SCSI was provided, which allowed the use of two sixteen-bit cables on one bus, but this version was not widely used.

The SCSI-3 (or Ultra SCSI) standard appeared in 1992. The bus throughput was 20 MB/s for an eight-bit bus and 40 MB/s for a sixteen-bit one. The maximum cable length has remained equal to three meters.

Between 1997 and 2010, the following Ultra SCSI variants occurred:

Ultra-2 SCSI - 1997

Ultra-3 SCSI - 1999

Ultra-320 SCSI - 2001

Ultra-640 SCSI - 2003

In order to ensure the quality of the signals on the SCSI bus, the bus lines must be matched on both sides using a set of terminating resistors, or terminators.

Terminators must be installed on the host adapter and on the last trunk device. Usually one of three matching methods is used:

- passive matching with resistors;
- FPT (Force Perfect Termination) - improved matching with the exception of overloads using restrictive diodes;
- active matching with voltage regulators.

Data exchange between devices on the *SCSI* bus occurs in accordance with the high-level protocol based on the standard list of commands - *CCS* (Common Command Set). This universal set of commands provides access to data by addressing logical rather than physical blocks. With the introduction of *CSS* commands into the specification that support *CD-ROM* drives, communication devices, scanners, etc. (*SCSI-2* standard), the work with almost any block device has become feasible.

RS-232C interface

The *RS-232C* interface is designed to connect equipment transmitting or receiving data. The data transmission equipment (*DTE*) can be a computer, printer, plotter and other peripheral equipment. The *DRE* is usually a modem. The final connection goal is to connect two *DTE* devices.

The *RS-232C* serial interface standard was published in 1969 by the Electronic Industry Association (*EIA*). Initially, this interface was used to connect computers and terminals to a communication system via modems, as well as to connect directly terminals to machines. Now the *RS-232C* interface is being actively replaced by the *USB* interface.

Typically, *PCs* include two *RS-232C* interfaces, which are designated as *COM1* and *COM2*. It is possible to install additional equipment that ensures the functioning of four, eight and sixteen *RS-232C* interfaces as a part of a *PC*. In order to connect devices, there is used a 9-pin (*DB9*) or 25-pin (*DB25*) connector, see fig.15.6.

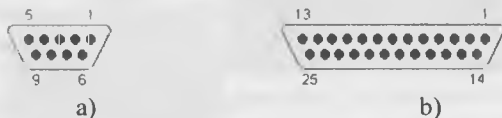


Figure 15.6. 9 (*DB9*) and 25 (*DB25*) - pin connector for connection of devices to *RS-232C*

The basic principles of information exchange on the *RS-232C* interface are following:

Data exchange is provided along two circuits, each of which is the transmitting side for one side and the receiving side for the other.

At the initial state, a binary unit is transmitted along each of these circuits, i.e. stop package. The transmission of the stop package can be performed as long as desired.

The transmission of each data packet is preceded by the transmission of the starting package, i.e. transmitting binary zero for a time equal to the transmission time of one data bit.

After transmission of the start package, a sequential transfer of all bits of data is ensured, starting with the least significant bit. The number of bits can be 5, 6, 7 or 8.

After the transmission of the last bit of data, it is possible to transmit a control bit, which complements the sum according to the module 2 of the transmitted bits to parity or oddness. On some systems, the transmission of the control bit is not performed.

After the transmission of the control bit or the last bit, if the formation of the control bit is not provided, the transmission of the stop package is ensured. The minimum sending duration may be equal to the transmission duration of one, one and a half or two data bits.

Data exchange according to the principles described above requires preliminary coordination of the receiver and transmitter in terms of speed (bit duration) (300-115200 bit/s), the amount of digits used in a symbol (5, 6, 7 or 8), the rules for the formation of a check digit (parity control, by oddness or lack of a check digit), the duration of the transmission of the stop package (1 bit, 1.5 bit or 2 bit).

The RS-232C specification for the electrical characteristics of signals determines that a high voltage level from +3V to +12V (for transmission - up to +15V) is considered as logical "0", and a low voltage level from 3V to 12V (for transmission - up to 15V) as logical "1". Signal range - 3V: +3V provides protection against interference and stability of transmitted data.

IEEE 1284 Interface

The standard parallel port interface got its original name from Centronics, an American printer manufacturer.

Parallel Port Centronics is a port used since 1981 in IBM personal computers to connect printing devices. Initially, this port was designed only for simplex (unidirectional) data transfer, since it was assumed that the port should be used only for working with the printer, see fig. 15.7.

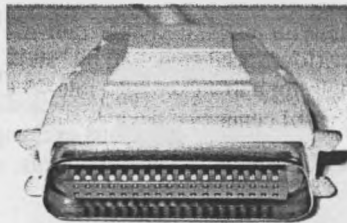


Figure 15.7. 36-pin cable connector for connecting an external device (IEEE 1284-B)

Subsequently, different companies have developed duplex interface extensions (byte mode, EPP, ECP). Then the international standard IEEE 1284 was adopted, describing both the basic Centronics interface and all its extensions.

The IEEE 1284 standard defines the parallel interface in three modes: Standard Parallel Port (*SPP*), Enhanced Parallel Port (*EPP*), and Extended Capabilities Port (*ECP*). Each of these modes provides two-way data transfer between the computer and the peripheral device.

SPP (Standard Parallel Port) mode is used for compatibility with older printers that do not support IEEE 1284. Three programmatically accessible registers correspond to SPP mode:

- BASE + 0 port - SPP Data - data register,
- BASE + 1 port - SPP Status - status register,
- BASE + 2 port - SPP Control - control register.

For an LPT1 device, the base address (BASE) in the I/O port space is typically 378h.

Nowadays, the IEEE-1284 standard is not being developed and is actively being superseded by the USB interface. Also, an alternative to a parallel interface is the Ethernet network interface.

USB interface

The USB peripheral bus specification was developed by computer and telecommunications industry leaders such as Compaq, DEC, IBM, Intel, Microsoft, NEC and Northern Telecom to connect computer peripherals outside the PC case with automatic auto-configuration (Plug & Play). The first version of the standard appeared in 1996.

The main goal of the standard, set before its developers, is to create a real opportunity for users to work in Plug & Play mode with peripheral devices. In addition, it is desirable to supply power to low-power devices from the bus itself. Bus speed should be sufficient for the vast majority of peripherals. Along the way, the historical problem of lack of resources on the internal buses of an IBM PC compatible computer is being solved - the USB controller takes only one interrupt regardless of the number of devices connected to the bus.

USB capabilities follow from its specifications:

- High speed exchange (full-speed signaling bit rate) - 12 Mb/s;
- Maximum cable length for a high exchange rate - 5 m;
- Low speed exchange (low-speed signaling bit rate) - 1.5 Mb/s;
- Maximum cable length for a low exchange rate - 3 m;
- The maximum number of connected devices (including multipliers) - 127;
- It is possible to connect devices with various exchange speeds;
- No need for the user to install additional elements, such as terminators for SCSI;
- Supply voltage for peripheral devices - 5 V;
- The maximum current consumption per device is 500 mA.

This interface is especially convenient for connecting frequently connected/disconnected devices. USB connector structure, see fig. 15.8, is designed for multiple articulation/partition.

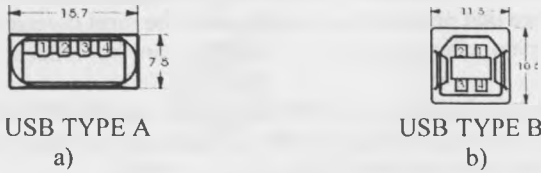


Figure 15.8. USB connectors

The bus allows to connect up to 127 physical devices to a PC. Each physical device can, in turn, consist of several logical ones.

The *USB 1.1* interface declares two modes:

- low-speed subchannel (throughput - 1.5 Mbps), designed for devices such as mice and keyboards;
- high-performance channel, providing a maximum throughput of 12 Mbps, which can be used to connect external drives or devices for processing and transmitting audio and video information.

The *USB 2.0* specification was released in April 2000. USB 2.0 differs from USB 1.1 by the introduction of Hi-speed mode.

For USB 2.0 devices, three modes of operation are regulated:

Low-speed, 10-1500 Kbps (used for interactive devices: keyboard, mouse, joystick);

Full-speed, 0.5-12 Mbps (audio, video devices);

Hi-speed, 25-480 Mbps (video devices, storage devices).

In the *USB 3.0* specification, the updated standard connectors and cables are physically and functionally compatible with USB 2.0. The USB 2.0 cable contains four lines (see. Fig. 15.8) - a pair for receiving/transmitting data, plus and zero power. In addition to these, USB 3.0 adds four more communication lines (two twisted pairs), making the cable much thicker. The new contacts in the USB 3.0 connectors are located separately from the old ones on the other contact row, see fig. 15.9.

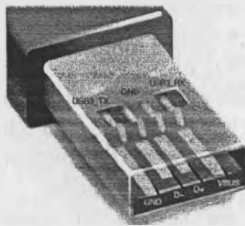


Figure 15.9. USB 3.0 Connectors

The USB 3.0 specification increases the maximum data transfer rate up to 4.8 Gb/s.

IEEE 1394 Interface - FireWire

FireWire is a universal, convenient in implementation and use of a high-speed interface that provides the connection of the most diverse devices.

The main characteristics of the bus can be reduced to the following indicators:

- data transfer rates up to 400 Mb/s according to IEEE-1394a standard and 800 Mb/s according to IEEE-1394b standard, agreed upon by the 1394 Trade Association at the end of May 2001;

- 16-bit address allows to address up to 64K nodes on the bus;
- maximum theoretical bus length 224 meters;
- "hot" connection/disconnection without data loss;
- automatic configuration similar to Plug & Play;
- arbitrary bus topology - by analogy with local networks, both a star and a common bus can be used (only in the form of a chain, unlike a network on a coaxial cable);
- lack of terminators;
- the ability of exchange with guaranteed throughput.

The maximum distance between two devices in the chain according to IEEE-1394a is 4.5 m, according to IEEE-1394b - 100 m.

Data transmission is carried out on a thin and flexible cable (up to 4.5 meters long) at a speed of up to 400 Mbps (i.e. 50 MB / s).

The IEEE-1394 topology allows both a tree-like and a chained architecture, as well as a combination of both. Therefore, it is easy to create any variants for connecting various devices to the bus. The standard provides for the architectural separation of the bus into 2 main blocks - the cable part and the controller (controllers). Since there can be several controllers, this part is also called the backplane (backplane - literally background, cross-board, etc.).

The node address on the "tree" is 16-bit, which allows to address up to 64K nodes. Up to 16 end devices can be connected to each node. On the backplane, up to 63 nodes can be connected to one bus bridge. Since 10 digits are allocated for the identifier of the bus (bridge) number, the total number of nodes is 64K.

Each node usually provides for the connection of 3 devices, although the standard itself allows the connection of up to 27 devices. Devices can be connected via standard cables up to 4.5 meters long.

Physical addresses (ID) of devices are assigned when power is supplied to the bus controller and devices connected to it, after a general reset of the bus, as well as when the device is hot-connected to the bus. Addresses are assigned in the order of device discovery and / or connection. No installation of jumpers or switches on the devices themselves is required.

The standard cable for IEEE-1394 consists of 2 twisted-pair bus signal transmission pairs, two power wires and all this is enclosed in a shielded shell.

Power wires are designed for current up to one and a half amperes and voltage from 8 to 40 volts. The fig. 15.10 shows one of the IEEE-1394 cable options.

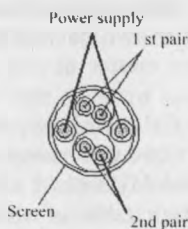


Figure 15.10. IEEE-1394 cable

Unlike USB, where the use of various types of connectors is regulated by the type of device, in FireWire everything is somewhat different. Here, the connectors are divided according to whether the device needs bus power or not. In case when there is no need for power supply, there is used a 4-pin connector (as a rule, this is used in video cameras). If the device may require power from the bus, then a 6-pin connector is used. Most computer devices are designed specifically for it.

15.4. Wireless interfaces

IrDA Infrared Interface

The first version of the IrDA standard (*Infra-Red Data Association*) was adopted in 1994 by the Infrared Data Transfer Association. The IrDA interface allows to connect to peripheral equipment without a cable using infrared radiation with a wavelength of 850-900 nm (nominally - 880 nm). The IrDA port allows to establish communications at a short distance of up to 1 meter in point-to-point mode.

The infrared interface device is divided into two main blocks: a converter (receiver-detector and diode modules with control electronics) and a codec. Blocks exchange data via an electrical interface, in which they are transmitted in the same form via an optical connection, except that here they are packaged in frames of simple format - data is transmitted in 10bit characters, with 8bit data, one start bit at the beginning and one stop bit at the end of the data.

The IrDA port itself is based on the architecture of the PC COM port, which uses the UART (*Universal Asynchronous Receiver Transmitter*) and operates at a data rate of 2400-115,200 bps.

Communication in IrDA is half duplex, as the transmitted infrared beam inevitably illuminates the adjacent PIN diode amplifier of the receiver. The air gap between the devices allows to receive infrared energy from only one source at a time.

The device circulation scheme is a data exchange protocol where there are phases of *requests* and *responses*. Information is exchanged only with the primary

device, which always acts as the initiator of the connection, however any of the devices supporting the necessary functions can play its role. Each device has a 32-bit address, randomly generated when a connection is established. At the start, the master assigns a 7-bit connection address to each frame within the connection. For possible but undesirable cases when two devices have the same address, such a mechanism is provided when the master device instructs all slave devices to change their addresses.

Link Management Protocol (IrLMP) is required, but some features may be optional. IrLMP protocol contains two components: LM-IAS (*Link Management Information Access Service*) and LM-MUX (*Link Management MultipleXer*).

Each IrDA device contains a table of services and protocols currently available. This information may be requested from other devices. The LM-IAS manages the information base so that stations can request which services are provided. This information is stored in the form of objects, each of which has a set of attributes.

The LM-MUX performs channel multiplexing on top of a single connection established by the IrLAP protocol. For this purpose, a plurality of LSAP (*Link Access Point*) access points are defined each with a unique identifier (selector). Thus, each of the LSAP connections defines logically different information flows. LM-MUX operates in two modes: multiplexing and exclusive. The first mode allows to share one physical connection to several tasks. In this case, flow control should be provided by top-level protocols (for example, TinyTP) or directly by the application. The second mode gives all the resources to one single application. IrLMP also provides three access options:

- with the establishment of a preliminary connection,
- without establishing a preliminary connection;
- a mode of collecting information about the capabilities, services and applications of a remote device.

In the last decade, IR ports on PCs have given the way to Bluetooth and Wi-Fi interfaces.

However, it is possible that the infrared port will soon be able to survive the renaissance. In 2008, Japanese developers were able to dramatically improve the parameters of the interface, primarily the exchange rate, while retaining one of its important advantages - complete immunity to electromagnetic interference. The new technology is distinguished by the use of semiconductor lasers instead of conventional LEDs. This allowed increasing the exchange rate to 1 Gbit/s, which is 250 times more than the theoretical limit of IrDA ports, equal to 4 Mbit / s, and more than 60 times higher than the capabilities of VFIR. In addition to increased speed, the novelty features improved noise immunity. The new IrDA interface can find application in PCs and mobile devices, especially phones.

Bluetooth interface

In early 1998, Ericsson, IBM, Intel, Toshiba and Nokia - the largest companies in the computer and telecommunications market - teamed up to develop jointly wireless technology for mobile devices. On May 20, 1998, an official

presentation of a special working group (*SIG - Special Interest Group*) took place, designed to ensure the unhindered implementation of the technology called Bluetooth. Now the group includes more than 1,400 companies (including 3COM/Palm, Axis Communication, Motorola, Compaq, Dell, Qualcomm, Lucent Technologies, UK Limited, Xircom, etc.) taking part in the work on the free open Bluetooth specification.

Devices using the Bluetooth standard operate in the 2.45 GHz ISM band (*Industrial, Scientific, Medical* - industrial, scientific and medical range) and are capable of transmitting data at speeds up to 720 Kb/s over a distance of 10 meters and the transmission of 3 voice channels. Such indicators are achieved using a transmit power of 1 mW and the involved frequency switching mechanism that prevents interference. If the receiving device determines that the distance to the transmitting device is less than 10 m, it automatically changes the transmit power to the level required for a given arrangement of devices. The device switches to energy saving mode when the amount of transmitted data becomes small or the transmission stops.

The technology uses FHSS - frequency hopping (1600 hops/s) with spread spectrum. During operation, the transmitter switches from one operating frequency to another according to a pseudo-random algorithm. For full-duplex transmission, time division duplex (TDD) is used. Isochronous and asynchronous data transfer is supported and easy integration with TCP/IP is provided. Time slots are deployed for synchronous packets, each of which is transmitted on its own radio frequency.

The power consumption of Bluetooth devices should be within 0.1 watts. Each device has a unique 48-bit network address that is compatible with the IEEE 802 LAN standard format.

The 2.45 GHz band is not licensed and can be freely used by everyone. It is only managed by the *Federal Communications Commission (FCC)*, limiting the part of the range that each device can use. Bluetooth devices are capable of connecting to each other, forming piconets, each of which can include up to 256 devices. In this case, one of the devices is the master (Master), seven more - slave (Slave), the rest are in standby mode. Piconets can overlap, and access to the resources of slaves can be arranged. Overlapping piconets can form a distributed network over which data can migrate.

In contrast to the infrared technology IrDA (Infrared Direct Access), which works on the principle of "point-to-point" in the line of sight, Bluetooth technology was developed to work both on the principle of "point-to-point" and as a multipoint radio channel controlled by a multilevel protocol similar to the GSM mobile communication protocol.

Bluetooth has become a competitor to technologies such as IEEE 802.11, HomeRF and IrDA, although the latter is not designed to create local area networks, but it is the most common wireless technology for connecting computers and peripherals.

The Bluetooth 3.0 specification was adopted on April 21, 2009. It supports theoretical data transfer rates of up to 24 Mbps. Its main feature is the addition of AMP (*Asymmetric Multiprocessor Processing*), an addition to 802.11 as a high-

speed message. Modules with support for the Bluetooth 3.0 specification combine two radio systems: the first provides data transfer at 3 Mbps (standard for Bluetooth 2.0) and has low power consumption; the second is compatible with the 802.11 standard and provides the ability to transfer data at speeds up to 24 Mbps (comparable to the speed of Wi-Fi networks). The choice of radio system for data transfer depends on the size of the transferred file. Small files are transmitted over a slow channel, and large files over a high-speed one. Bluetooth 3.0 uses the more general 802.11 standard (no suffix), i.e. it is not compatible with Wi-Fi specifications such as 802.11b/g or 802.11n.

In 2010, the Bluetooth Special Interest Group organization approved the new Bluetooth 4.0 interface, and now technology manufacturers can use it in their devices.

The following changes have occurred over the Bluetooth 3.0 standard. Firstly, the data transfer rate has been increased - now it reaches 24 Mbps. Secondly, the range of the Bluetooth module has significantly increased - it exceeds 100 meters. Third is the reduction in power consumption of the Bluetooth module. This parameter is important for portable devices.

Questions for self-control

1. What is an interface?
2. What are the main functions of computer interfaces.
3. Computer system interfaces and their features.
4. List the main features of the AGP interface.
5. What is the difference between PCI and PCI Express.
6. What expansion buses are currently used in PC architecture?
7. What are the features of the standard serial interface RS-232C.
8. List the features of the USB 3.0 interface.
9. What are the protocols on the IrDA stack and their purpose.
10. Describe the topology of the USB and FireWire interfaces.
11. Compare the technical specifications of the USB and FireWire interfaces.

Test questions

1. The name of the RAM connection interface (RAM)
A) DIMM B) DDR C) PCI D) SATA
2. Drive connection interface
A) HDMI B) AGP C) PCI D) SATA
3. Monitor connection interface
A) HDMI B) SSD C) HDD D) FX
4. LAN is the interface to connect
A) Internet

- B) LAN
- C) supply
- D) external devices

5. Parallel drive connection interface

- A) SSD B) IDE C) D-SUB D) HDD

6 Various bus interfaces are interconnected:

- A) conductors
- B) bridges
- C) channels
- D) nodes

7. In order to pair the central nodes of the computer with its external devices there are used:

- A) controllers
- C) slots
- C) bridges
- D) interfaces

8. Which of the standards of the internal interfaces is designed for the needs of the video system:

- A) ISA B) AGP C) LPC D) EISA

9. To connect the various bus interfaces to each other there are used:

- A) channels
- C) conductors
- C) bridges
- D) nodes

10. Interfaces are:

- A) central and peripheral
- C) internal and external
- C) active and passive
- D) parallel and serial

11. In which of the standards of internal interfaces, the fundamental principle was the use of bridges:

- A) ISA B) AGP C) LPC D) PCI

12. What function are not meant by pairing?

- A) the issuance and reception of information
- B) data transfer control
- C) the coordination of the source and receiver of information
- D) storage of information

13. What interface, or rather even the bus, serves for input and output between the motherboard and peripheral devices?

- A) SATA B) IDE C) ATA D) PCI

14. How does the PCI bus interpret all the exchanges?

- A) blocks
- B) packages
- C) words
- D) files

15. What interface allows users to work in Plug & Play mode with peripheral devices?

- A) SCSI B) IEEE 1284 C) USB D) FireWire

16. How many physical devices can a USB connect to a PC?

- A) up to 127
- B) up to 64
- C) up to 32
- D) up to 16

17. What does not belong to the main characteristics of the IEEE 1394 - FireWire bus?

- A) theoretical limit length of the tire 224 meters
- B) hot plug / unplug without losing data
- C) automatic configuration similar to Plug & Play
- D) the presence of terminators

18. What architecture does the IEEE-1394 topology not allow?

- A) tree
- B) chain
- C) tabular
- D) tree-chain

19. What are wireless interfaces?

- A) IrDA
- B) IEEE 1394 - FireWire
- C) SCSI
- D) Firewire

20. What is the wireless connection of mobile devices?

- A) IrDA
- B) IEEE 1394 - FireWire
- C) bluetooth
- D) Firewire

CHAPTER 16. INPUT-OUTPUT DEVICES

Since a large amount of specialized literature is devoted to input / output devices [3, 5, 8, 12, 17, 20], we will briefly dwell on some design aspects of the most common of them.

We can distinguish the following main functional classes of input-output devices (I/O):

1. I/O devices designed to communicate with the user, these primarily include keyboards, scanners, as well as manipulators - mice, trackballs and joysticks, monitors, printers, plotters, etc.;
2. Communication devices with the control object (ADC, DAC, sensors, digital controllers, relays, etc.);
3. Means of data transmission over long distances (telecommunications - modems, network adapters, etc.).

The main device for inputting information into a computer is a **keyboard**, which is a set of mechanical sensors that sense pressure on the keys and close in one way or another a certain electrical circuit. Currently, two types of keyboards are common: with mechanical or with membrane switches. In the first case, the sensor is a traditional mechanism with contacts made of a special alloy. In the second case, the switch consists of two membranes - the upper (active) and lower (passive). The membranes are separated by a third gasket (also a membrane).


As a rule, inside the body of any keyboard, except for key sensors, there are electronic decryption schemes and a microcontroller. Information is exchanged between the keyboard and the system board via a special serial interface with 11-bit blocks. The main principle of the keyboard is to scan the key switches. Closing and opening any of these switches corresponds to a unique digital code - a scan code. The keyboard is connected to the system board using the electrically identical connectors 5 DIN5 or 6 mini-DIN (PS / 2), see table. 16.1 or USB bus.



The first computer **mouse** was introduced on December 9, 1968 by Douglas Engelbart at the Fall Joint computer conference. The distribution of mice was due to the growing popularity of software systems with a graphical user interface.

The first mouse, when moving, rotated two wheels that were connected with the axes of variable resistors. The cursor movement of such a mouse was caused by a change in the resistance of the variable resistors.

Table 16.1.

Wiring of AT connectors (5 DIN) and PS / 2

Connector Numbering	Connector Assignment
	<p>5 DIN</p> <ol style="list-style-type: none"> 1. Frequency 2. Data 3. N / A (not used) 4. GND ("ground") 5. Power + 5V

PS/2	6 mini-DIN	PS/2 и 6 mini-DIN
		<ol style="list-style-type: none"> 1. Data 2. N / A (not used) 3. GND (“ground”) 4. Power + 5V 5. Frequency 6. N / A (not used)

Today, the most common mouse design is the fully optical design. With the help of an LED and a system of lenses focusing its light, a surface area is highlighted under the mouse. The light reflected from this surface, in turn, is collected by another lens and enters the receiving sensor of the image processing chip. This chip takes high-frequency images of the surface under the mouse and processes them. Based on the analysis of a series of consecutive images, which are a square matrix of pixels of different brightness, the integrated DSP processor (*Digital signal processor*) calculates the resulting indicators indicating the direction of mouse movement along the X and Y axes, and transfers the results of its work to the peripheral interface. The main characteristics that ensure the reliability of optical mice are determined by the technical parameters of the sensors used.

The first mice were connected to the PC via a special adapter board (the so-called mice with a bus interface - *bus mouse*). Then, the method of connecting a mouse via the RS-232C serial interface was widely used. In 1987, IBM released a series of personal computers PS / 2, which presented a dedicated serial interface for connecting a mouse with a 6 mini-DIN connector, see table. 16.1. In 2002, the Microsoft PC 2002 specification suggested dropping these ports in favor of a universal USB interface.

A trackball is an “inverted” optical-mechanical mouse - it’s not the device’s body itself that is set in motion, but only its ball. This can significantly increase the accuracy of cursor control and, in addition, save space, so trackballs are often used in laptops.

A touchpad (touchpad or trackpad) is an input device used in laptops that serves to move the cursor depending on the movements of the user’s finger. Used as a replacement for a computer mouse. Touch panels vary in size, but usually their area does not exceed 50 cm². The operation of the touch panel is based on measuring the capacitance of a finger or measuring capacitance between sensors. Capacitive sensors are located along the vertical and horizontal axes of the panel, which allows you to determine the position of the finger with the desired accuracy. The advantages of touch panels are:

- there is no need for a flat surface, as for a mouse;
- the location of the touch panel is usually fixed relative to the keyboard;
- to move the cursor to the full screen, just a little finger movement is enough;
- working with them does not require much getting used to, as, for example, in the case of a trackball.

The disadvantage of touch panels is the low resolution, which makes it difficult to work in graphic editors.

A *joystick* is an analog coordinate information input device, usually performed in the form of two rheostatic sensors with + 5V power supply. The joystick handle is connected to two variable resistors that change their resistance when moving it. One resistor determines the movement along the X coordinate, the other along the Y coordinate. The joystick is usually connected to the game port adapter located on the *Multi I/O Card*.

A *scanner* is a device that allows to enter images presented in the form of text, drawings, slides, photographs or other graphic information into a computer. Scanners can be classified by the following criteria:

1. By the degree of transparency of the input image from the original:
 - opaque originals (photographs, drawings, pages of books and magazines), while the image is taken in reflected light;
 - transparent originals (slides, negatives, films), while processing the light transmitted through the original.
2. According to the kinematic mechanism of the scanner:
 - handheld scanners in which the task of smooth and uniform movement of the scanning head in the corresponding image is assigned to the user;
 - flatbed scanners - the scanning head moves relative to the paper using a stepper motor;
 - roll scanners - individual sheets of documents are pulled through the device so that the scanning head remains in place;
 - projection scanners - the input document is placed on the scanning surface with the image facing up, while the scanning unit is also located on top, and only the scanning device moves.
3. By type of input image:
 - black and white (dashed or halftone);
 - colored.

In black and white scanners, the image is illuminated with white light, obtained, as a rule, from a fluorescent lamp. The reflected light through a reducing lens enters the photosensitive element (CCD scale or CCD matrix). Each line of the scanned image corresponds to certain voltage values at the CCD, see fig.16.1.

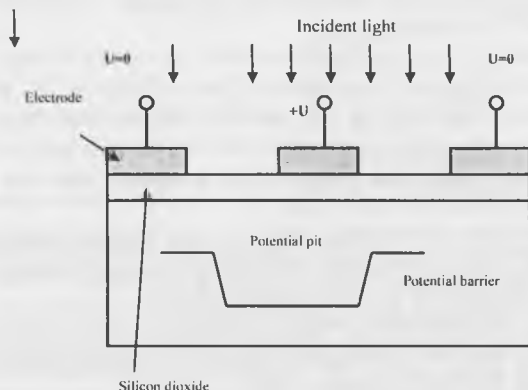


Figure 16.1. Principal device of a CCD matrix

On the semiconductor surface there is a thin (0.1-0.15 microns) layer of dielectric (usually oxide), on which are strips of conductive electrodes (made of metal or polycrystalline silicon). These electrodes form a linear or matrix regular system, and the distances between the electrodes are so small that the effects of the mutual influence of neighboring electrodes are significant. The principle of operation of the CCD is based on the occurrence, storage, and directional transfer of charge packets in potential wells formed in the surface layer of a semiconductor when external electric voltages are applied to the electrodes [20].

These voltage values are converted to digital form via the ADC (for grayscale scanners) or through a comparator (for two-level "bar" scanners).

There are several technologies for color images scanning. For example, the scanned image is alternately illuminated in red, green, and blue, so that the page is scanned in three passes. In a number of scanners, for example, Epson and Sharp, a color change occurs for each line, which avoids problems with the "alignment" of pixels with different passes.

In the design of Hewlett Packard (HP) and Ricoh scanners, the scanned image is illuminated with a white light source, and the reflected light through a reducing lens enters the three-band CCD line through a system of special filters that separate the light into three components: red, blue, green.

To communicate with a computer, scanners usually use one of the universal peripheral interfaces: SCSI, IEEE 1284 or USB.

To unify the application software of scanners in 1992, a number of companies (Kodak, HP, Logitech, etc.) developed the TWAIN specification [20].

A monitor (display) - a device for visualization of text or graphic information without its long-term fixation.

By the type of displayed information, monitors are divided into alphanumeric (currently not used) and graphic. By the method of image formation, graphic displays are divided into vector (not used in PC) and raster. In a vector display, an image is constructed from elementary segments of vectors (in the case

of CRT, the electron beam continuously “draws” the image contour, collecting it from these vectors). In raster displays, the image is obtained using a matrix of points (in the case of CRT, electron beams travel along the lines of the screen, highlighting the required points with their color).

The most widespread are monitors based on cathode ray tubes (CRT) and based on liquid crystals (LC).

The principle of operation of CRT monitors is that the electron beam emitted by the electrode (electron gun), falling on a screen coated with a phosphor, causes it to glow. There are additional electrodes in the path of the electron beam: a deflecting system (determines the direction of the beam) and a modulator (adjusts the brightness of the resulting image). In the case of a color monitor, there are three electron guns with separate control circuits, and a phosphor of three primary colors is applied to the screen surface: R (red) - red, G (green) - green, B (blue) - blue. In order for each gun to hit only the phosphor of its color, a shadow mask is used. The electron beam periodically scans the entire screen, forming nearby scan lines. As the beam moves along the lines, the video signal supplied to the modulator changes the brightness of certain pixels, forming a visible image. In the scanning cycle, the beam moves along a zigzag path from the upper left corner of the screen to the lower right. A direct horizontal beam path is carried out by a horizontal (horizontal) scan signal, and vertically - by a vertical (vertical) scan signal.

Obviously, the most important parameters for a monitor are: frame rate, horizontal frequency, and video bandwidth. The frequency of the frame scan largely determines the stability of the image (no flicker). Modern *monitors* support frame scans in the range of 60-160 Hz. The horizontal frequency is determined by the product of the vertical frequency by the number of displayed lines in one frame, taking into account the reverse stroke (vertical resolution), a typical value is 30-64 kHz (reflects the number of lines that the monitor can play in one second). The video throughput is determined by the product of horizontal resolution, taking into account the reverse by the horizontal frequency (reflects the number of dots in a line that the monitor can play in one second). The important factors determining the clarity of the image also include the dimensions of the phosphor points, or rather, the distance between them (*dot pitch*), a typical value is 0.25-0.28 mm.

The operation of LCD monitors is based on the property of some substances to exhibit anisotropy in a fluid (“liquid”) state. The first LCD monitor was demonstrated by the American company RCA in 1966. For the manufacture of LCD monitors use the so-called nematic crystals - optically uniaxial liquid crystals having a long-range orientation order and free to move. In the absence of an electric field, the molecules of this substance form twisted spirals (usually 90°). As a result of this orientation of the molecules, the plane of polarization of the transmitted light rotates. If a voltage is applied to the transparent electrodes, the spiral of molecules straightens (they are oriented along the field), while the rotation of the plane of polarization of transmitted light does not occur. Using a suitably oriented film polarizer, it is possible to ensure that in the first case, the LCD element transmits transmitted light, and in the second - no.

Using Twisted-Nematic (*TN*) elements, the contrast is 3: 1 (the illuminated dot is three times lighter than the dark one). The molecules of the Super-Twisted-Nematic (*STN*) element are twisted through an angle of 180 to 270 degrees. The contrast when using STN elements is 10: 1 and higher, but some color shift appears in them. To eliminate color errors in the currently used Triple STN elements (*TSTN*), also called FSTN - Film STN, a special polymer film is provided between the glass and the polarizer - the third layer (hence Triple).

The LCD screen has either a backlight (*backlight* or *backlit*) or a side (*sidelight* or *sidelit*). Each dot of the image on the LCD is a corresponding TSTN element, and the entire screen is a matrix of these elements.

Two methods can be used to address LCD elements: direct (passive) and indirect (active). With direct addressing of the elements, each selectable image point is activated by applying voltage to the corresponding conductor-electrode for the row (common for the whole row) and to the conductor-electrode for the column (common for the entire column). Matrices with passive control ("passive matrices") have insufficient image contrast, because an electric field arises not only at the intersection of the address conductors, but also along the entire current propagation path. This problem is solved by using the so-called active matrices, when each point in the image is controlled by its own independent electronic switch (usually TFT).

Using active matrices, such parameters as short response time (typical value - 1-8 μ s) and a large viewing angle (75°-120°) are of great importance.

During the connection of monitors to the video card, two types of connectors are used mainly: a DB-15 connector with an analog video signal and optionally with a digital DDC interface and a DVI (*Digital Visual Interface*) connector, which allows to transfer both analog video and digital.

A printer usually means a data output device that converts information into a readable form on paper. Typically, printers are classified according to the following criteria:

1. By printing method:

- sequential - a printed document is formed as "symbol by symbol";
- lowercase - at printing, the device forms the entire line at once;
- page - the image of the entire page is applied immediately to the paper.

2. According to the used printing technology:

- shock (mechanical shock is used to transfer the coloring matter);
- non-shocked.

Currently, matrix printers are considered to be impact printers. The printhead of 9, 18 or 24 needles, is driven by electromagnets. The head is attached to the carriage and moves together with it along the guides parallel to the paper along the printed line. Some of the matrix needles are set in motion, and they "hit" the ink ribbon located between the head and the paper, thus forming a printable character. The disadvantage of these printers is their low printing speed and high noise level. Advantage - non-stringent requirements for paper quality.

As non-shocked printers there can be considered inkjet printers. Their head moves in a horizontal plane above the paper. The print head contains nozzles

through which ink is supplied. For different models, the number of nozzles can vary from 12 to 64. Various inkjet printer technologies differ in the way that ink droplets are ejected from the nozzle. Cannon and HP printers use *bubble-jet* technology (or *thermal ink jet*). Each nozzle has a heating element (thin film resistor). With sudden heating, an ink vapor bubble forms, which pushes the next portion of ink from the nozzle. Epson printers use *piezo ink jet* technology. The discharge of the droplet from the nozzle is controlled by the diaphragm from the piezoelectric element. Under the influence of an electric field, the piezoelectric element is deformed and pushes a drop out of the nozzle. The undoubted advantage over matrix printers is the low noise level during operation, and the disadvantage is the rather high requirements for paper quality. In general, it should be noted that consumables for this technology are the most expensive compared to printers of other printing technologies.

Another popular non-shocked technology is electrographic printing technology, which is used in so-called laser printers. A semiconductor laser beam forms an electronic image on a photodetector drum. The drum is previously informed of a certain static charge. Therefore, the areas of the drum illuminated and not illuminated by a laser have different charges. Powder toner particles adhere to the charged areas. When the paper comes in contact with the drum, an imprint remains on it, which is fixed by heating the toner particles to the melting temperature. Laser printers have high print speed and high resolution. The disadvantage is the high price of printers and the need to use quality paper.

Special languages are used to control the printer. For dot matrix and inkjet printers, the ESC/P language is most widely used. For laser and some inkjet printers, the main control languages are HP PCL and Adobe PostScript.

PCL (*Printer Command Language*) is a printer control language developed by HP. PCL is currently only used by Microsoft and HP.

Postscript was by Adobe Systems in the early 80s. Postscript uses a model for displaying text (or pictures) on a blank page. When the page is ready, it is printed out and the "drawing" of the image of the next page begins. This is a compilation method. Each Postscript document is usually a program that prints (or displays on a monitor screen) successive pages.

In order to connect *printers* there are used RS-232C, IEEE 1284 or USB.

In conclusion, we note that recently, developers of various companies are increasingly offering alternative designs of input / output devices. Here is just one example. MTU specialists have proposed a device called Mouseless. The essence of the approach is to use traditional management technology for laptop owners, but without the participation of the mouse. The laser system mounted at the end of the laptop computer monitors the movement of the hand, which seems to lie on a classic mouse, and records the movement, perceiving the knocking of fingers on the table as clicks. The information received from the camera is processed by specially developed software that transforms the movement of the hand into the movement of the cursor on the screen.

However, Intel experts believe that over the next ten years, a person will be able to completely abandon the use of the keyboard and mouse, using only the

power of thought to control. At the moment, research is being conducted in the field of recording wave activity in order to determine the technology of its transformation for interfacing with control chips. Microcircuits theoretically installed on the surface of the brain will be able to perceive thoughts, interpret them and transmit them in the form of commands for computer systems.

Questions for self-control

1. Give a classification of peripheral devices.
2. Formulate the basic principles of operation of input devices.
3. List the classes of scanners.
4. What are the types of displays, the physical principles of image formation.
5. Give a comparative assessment of the various printing technologies.
6. What new trends in the development of input / output devices do you know?

Test questions

1. The main element of the keyboard:
 - A) key
 - B) number block
 - C) register
 - D) scan code
2. What is the name of a special program that provides the keyboard operation:
 - A) utility
 - B) driver
 - C) compiler
 - D) procedure
3. Locate and correct the error in the circuit of the principle of the keyboard:
 - A) Key-> Keyboard controller -> Chip UPI-> Interrupt output-> Keyboard buffer-> Interrupt keyboard-> Video buffer-> Monitor
 - B) Key-> Keyboard controller -> Chip UPI-> Keyboard interruption -> Keyboard buffer -> Interruption of output-> Video buffer-> Monitor
 - C) Key-> Keyboard controller -> Keyboard buffer -> Chip UPI -> Keyboard interruption-> Video buffer-> Monitor
 - D) Key-> Keyboard controller -> Keyboard buffer -> Keyboard interruption -> Chip UPI -> Video buffer-> Monitor
4. Scan code is:
 - A) a signal characterized by the number 0 or 1
 - C) a single-byte number assigned to each key

- C) a table of codes of characters and numbers of the keyboard
D) a special controller on the motherboard
5. According to the design of the keyboard there are distinguished:
A) keyboard with plastic pins
B) a keyboard with a click;
C) keyboard with microswitches;
D) touch keyboards;
E) multimedia keyboards.
6. According to the principle of action of the mice are divided into:
A) mechanical and optical
C) optical-mechanical and optical
C) infrared and optical
D) laser and mechanical
7. According to the principle of connection the mice are divided into:
A) wired and wireless
C) infrared and radio mice
C) optical and infrared
D) laser and mechanical
8. By the method of connection to a PC, mice are:
A) connectable to the COM port
C) connectable to the PS/2 port
C) connectable to the USB port
D) all answers are correct
9. At what distance from the receiver does the radio mouse operate:
A) 50-70 cm
C) up to 1.5 m
C) up to 3 m
D) unlimited
10. Which of the manipulators is used for digitizers:
A) mouse
C) trackball
C) cursor
D) touchpad
11. What type of joysticks are there:
A) analog
C) mechanical
C) digital
D) optical

12. What handling devices are used in laptops:
- A) mouse
 - B) trackball
 - C) joystick
 - D) touchpad
13. What parameter determines the quality of the mouse:
- A) the size of the button;
 - B) permission;
 - C) constructive performance;
 - D) the amount of buttons.
14. What are the typically performed dimensions of the Touch Pad:
- A) any
 - B) up to 10 cm²
 - C) up to 15 cm²
 - D) up to 20 cm²
15. Depending on the method of moving the photosensitive element and the image carrier, all scanners are divided into:
- A) roller and drum
 - B) desktop and manual
 - C) matrix and inkjet
 - D) color and black and white
16. What types of scanners are used to enter graphics and text on A4 or A3 format:
- A) tablet
 - B) roller
 - C) drum
 - D) projection
17. What types of scanners are connected to a PC without adapters?
- A) manual;
 - B) tablet;
 - C) roller.
 - D) projection
18. Multifunction cameras can not be used as:
- A) printer
 - B) copy machine
 - C) fax
 - D) modem

19. What hardware interface do the scanners support:

A) SCSI B) LPT C) USB D) PS / 2

20. By the method of printing printers are

A) consecutive

B) lowercase

C) paginated

D) bilateral

CHAPTER 17. NEW DIRECTIONS OF COMPUTER DEVELOPMENT

At present, there are a lot of promising directions related to the development of computer architecture. We will dwell only on the most intensively developing areas of research - quantum computers, nanotechnology, neurocomputers and photonics.

Quantum computers

The idea of creating a quantum computer (QC) belongs to the American physicist Richard Feynman. In 1958, modeling quantum processes on a computer, he realized that for solving quantum problems with many partial derivatives, the memory capacity of a classical computer is completely insufficient. Solving a problem with 1000 electronic spins, there should be enough cells in the memory to store 2^{1000} variables. And a gigabyte is only 2^{30} . A real breakthrough in the development of the QC idea occurred in 1995, when the American mathematician Shor shifted the algorithm for calculating prime factors of large numbers for a quantum computer. Shor showed that if a classical computer needs 2^{1000} operations to find factors of a number of 1000 binary digits, then a quantum computer will need only 1000^3 operations.

There are many theoretical models of a quantum computer. The problem, rather, is to find reasonable ways to create a real device. There are several approaches to implementing the idea of such a device. The first option is a high-resolution pulsed nuclear magnetic resonance (NMR) spectrometer. The spins of the core that make up the atoms, which in turn form the molecule studied in the NMR spectrometer, are Q-bits, units of measurement of quantum information. Each core has its own resonance frequency in a given magnetic field. When exposed to a pulse at the resonant frequency of one of the core, it begins to evolve, while the remaining cores are "silent". In order to force the second atom to evolve, you need to take a different frequency and give an impulse to it. In other words, the calculation process is controlled by pulses of an alternating magnetic field. For example, 1000^3 (i.e., a billion) operations in the Shor algorithm for a 1000-bit number is a billion actions on individual spins and their pairs. Moreover, there is a direct connection between the spins in the molecule, and therefore it is an ideal blank for a quantum computer, and the spectrometer itself is simply a ready-made "processor" for this computer. However, at present, it is possible to work with systems with a total number of spins no more than five to seven, while to solve full-scale problems they need about 1000.

Another approach is based on the use of so-called ion traps, or ions "suspended" in a vacuum. For the invention of ion traps, the scientist of the University of Bonn, Paul was once awarded the Nobel Prize [18,19,20]. These ion traps were able to "stretch" and to obtain a one-dimensional ionic crystal, which is held in the axial and radial directions by external fields. Each crystal ion takes two energy levels - this is one Q-bit; these ions are interconnected through vibrations

inside a one-dimensional crystal, which has a set of resonant frequencies. Experiments in this direction were actively conducted at the University of Innsbruck (Austria) and the Los Alamos Laboratory (USA) [21, 22]. Today, a chain of 30 ions has been obtained.

And the third approach is a solid-state quantum computer. For example, the Australian physicist Kane suggested that a quantum computer would be based precisely on the silicon that traditional microelectronics currently operates on. In the right places at distances of about 100 angstroms, phosphorus atoms are located - a common impurity in silicon. If two phosphorus atoms are located at such a distance, then the clouds of external electrons intersect a little, that is necessary for their interaction, and the atoms can exchange states. One atom controls the electrons of other 50-angstrom electrodes are made above these atoms, and with the help of voltage on this electrode the resonance frequency of the spin of the core of the phosphorus atom is changed. The design is similar to a field effect transistor - as the same shutters, only instead of the current - the state of the atom. A quantum computer will probably have a quantum communication channel based on an effect called "quantum teleportation". The principle of quantum teleportation is based on the effect of entangling the quantum states of two particles, which was analyzed back in 1935 by Einstein-Podolsky-Rosen. Entangled states arise during the interaction of two quantum particles and their subsequent separation; in this case, they find themselves in a certain "entangled" state, in which the state of the first particle is strictly correlated with the state of the second.

Now tens of millions of dollars are invested in quantum computer research. Of course, this can not even be compared with the money that goes to the development of traditional computers and even to research on nanotechnology. Nowadays, small teams in the laboratories of giants such as IBM and Intel are working on quantum computing. Many experiments are carried out in large centers, especially in Los Alamos (USA), at universities around the world: in Innsbruck (Austria), Bonn (Germany).

In 2009, the Canadian company D-Wave demonstrated the first working quantum computer Orion [21, 22]. Thus, real quantum computing became possible tens of years earlier than previously planned.

Orion is capable of simultaneously performing 64 thousand operations. The main part of any quantum computer is the quantum register, which is the aggregate of a certain number of qubits - quantum units of information. A qubit, unlike a regular bit, which can take the value 0 or 1, can simultaneously be in different quantum states, representing a superposition of 0 and 1.

Before entering information into the computer, all the qubits of the register must be brought into the basic states (the so-called initialization operation). Then each qubit is subjected to selective action (by pulses of an external electromagnetic field or in another way), and the entire register goes into a superposition of basic states.

Orion has a register of only 16 qubits. The information is then processed by a quantum processor that performs a sequence of quantum logic operations. The result of the conversion of information at the output of the computer is a new

superposition of states that can be further converted to a form suitable for further use.

The processor is only informed that it is a new type of analog processor with a scalable architecture and that it is based on quantum-mechanical principles.

D-Wave Systems said that a quantum computer will not be a competitor to the current one; rather, it is designed to solve problems with a huge amount of initial information and a large number of variables. Such tasks are characteristic of cryptography systems and secure data transfer, biology and medicine, modeling of quantum systems, and optimization of various processes.

Nanotechnology

Nanotechnology is a technology that operates on the order of a nanometer. These are technologies for manipulating individual atoms and molecules, as a result of which there are created structures of complex specifications. The word "nano" (in the ancient Greek language "a dwarf") means the billionth part of a unit of measure and is synonymous with an infinitesimal quantity, hundreds of times smaller than the wavelength of visible light and comparable to the size of atoms.

Nowadays, the work in the field of nanotechnology is carried out in four main areas:

- molecular electronics;
- biochemical and organic solutions;
- quasi-mechanical solutions based on nanotubes;
- quantum computers.

In 2007 (data from the pre-crisis period [21, 22]), the United States accounted for approximately 32% of all global investments in nanotechnology (European Union - approximately 16%, Japan - 20%). Research in this area is also actively being conducted in the countries of the former USSR, Australia, Canada, China, South Korea, Israel, Singapore and Taiwan. If in 2000 the total cost of the countries of the world for such studies amounted to approximately \$ 800 million, in 2010 they increased by 3.5. According to experts [22], for nanotechnology in order to become a reality, it is necessary to spend at least 1 trillion dollars annually.

Recently, the number of publications on new advances in nanotechnology has increased dramatically. The latest news can be found, for example, at <http://www.nanonewsnet.com/>.

The most significant practical results have been achieved in the field of molecular electronics. It is logically close to traditional semiconductor electronics. Using the methods of molecular electronics from hydrocarbon compounds, it is possible to obtain analogs of diodes and transistors, and, therefore, the main Boolean modules "AND", "OR" and "NOT", from which you can then create circuits of any complexity. This approach allows to maintain the continuity of architectural solutions.

In 1999, employees of the company HP and the University of California at Los Angeles were able to get a functioning molecular valve [22]. Its thickness is only one molecule. Initially, it can either just open or just close.

In 2001, a valve was demonstrated at Yale University that can take both of two positions, which allows arbitrarily to record 0 or 1. It is currently, according to publications [21, 22], work is underway to combine the valves into registers.

According to analysts, the miniaturization limit for traditional silicon electronics will come in 7-10 years.

Using an organic molecule and chemical internal processes, it was possible to reduce the size of the transistor to 1-2 nanometers. Creating such transistors, there was used the technique of "self-assembly", when the molecules themselves actually attach to each other using electrodes made of gold. This allowed to reduce the channel size to 1-2 nm, and the used technique is relatively inexpensive and allows to increase the density of transistors per unit area.

Philips scientists (2002) have developed a nanotransistor using the superconductivity effect. The transistor design consists of indium arsenide and aluminum superconducting contacts, and the charge is transferred not by electrons, but by Cooper pairs. Cooper pair - paired electrons with oppositely directed spins. In new elements, the current in the channel between the drain and the source is regulated by the gate voltage.

Scientists obtained indium arsenide-semiconductors with sizes from 10 to 100 nm using a complex evaporation process.

Scientists from the University of Pittsburgh (USA) proposed their nanotransistor design. The research team used dielectric materials such as lanthanum aluminate and strontium titanate. Taken together, these materials gained the ability to conduct positive electric charges.

As a result of scientific research at the University of California, a technology was developed for the formation of nanoconductors from various materials, with a clear, at the atomic level, separation of layers. This distinction is critical to the creation of high-performance transistors.

Heterogeneous structures — including, for example, silicon and germanium, are often used to form semiconductor components. But until now, it has not been possible to obtain nanoconductors with clear boundaries between the layers of these materials diffusing with each other, thereby violating the optimal conditions for using them as a transistor. In addition, the new technology, in contrast to the traditional one, involves vertical rather than horizontal placement of the layers forming the transistor. This feature can help to reduce the space occupied by each logic shutter, and provide the ability for further increase of the number of transistors.

Active research in the field of nanotransistor design is being carried out by scientists in South Korea. Six carbon atoms - just as many of them are involved in their proposed nanoelement, which has all the properties of a conventional transistor. Such a nanotransistor is a radically new solution, since its use will increase productivity by several times, significantly reducing energy consumption, according to Korean developers.

Serial production of nanotransistors has been delayed for several years, since only 15% of the transistor parameters obtained and tested by Korean scientists today met the required performance characteristics and the invention needs some

refinement. The second obstacle is the lack of technology which can integrate nanotransistors into existing microcircuits.

Nanotubes were first discovered in the laboratories of NEC (*Nippon Electric Corporation*) Japan. A nanotube is a cylindrical structure with a thickness of the order of 10 atoms, see fig. 17.1, which, depending on size and shape, may have conductive or semiconductor properties. For example, if the tube is straight, it is a conductor, and if it is twisted or bent, it is a semiconductor. The unusual electrical properties of nanotubes will make them one of the main materials of nanoelectronics.

In 2007, NEC announced the successful development of *carbon nanotube* (CNT) transistors made using inkjet technology. The properties of CNTs can be set by changing the length and density of nanotubes.

Two American universities in San Diego and Clemson succeeded in making the transistor entirely from carbon nanotubes branched in the shape of the letter "Y", see fig. 17.2. The nanotransistor is a few hundred microns in size, which is about 100 times smaller than the components used in today's microprocessors.

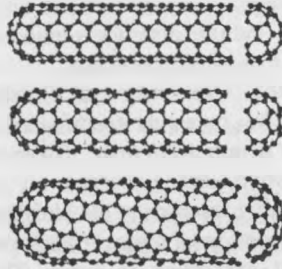


Figure 17.1. Scheme of carbon nanotubes

Most recently, scientists at IBM Research succeeded in producing a carbon monomolecular structure in the form of a nanotube, which fully implements one of the three main logical elements - the logical NOT element. At the same time, another feature of the created element is noted: the output signal is higher than the input signal by about one and a half times.

On nanotubes, there have already been created real display designs. For example, Japanese ISE Corporation manufactures stadium displays based on field emission panels.

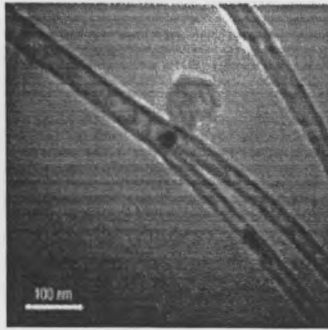


Figure 17.2. Carbon Nanotube Nanotransistor (Image from New Scientist)

Motorola has demonstrated the current prototype of the new color display, which uses many nanotubes. The prototype display has a size of 4.7 inches diagonally and gives an optical resolution of 128x96 pixels. It should become an element of a 42-inch high-definition television screen with a resolution of 1280 × 720 pixels.

Photonics

Photonics is a technology of radiation, transmission, registration of light using fiber optics and optoelectronics.

Silicon photonics is a combination of optical technology with traditional silicon, widely used for the production of a variety of microcircuits. A key feature of such a hybrid technology is the conversion of electrical signals into light and vice versa, that is, the conversion of electrons into photons and vice versa.

Intel Corporation was the first to create a prototype of a silicon-based hybrid optical data transmission system with a bandwidth of up to 50 Gbps.

This is a significant achievement in the field of silicon photonics, directly related to the prospect of creating hybrid computer microcircuits and entire systems using optical conductors. Currently, chips and other components of computer technology and other electronics are connected to each other via metal tracks on printed circuit boards or simply using wires. In this case, any metal, including copper, has some resistance, as a result of which, with an increase in the length of the conductor, the signal level decreases. Due to these losses, designers are forced to place electronic elements as close to each other as possible. In addition, the data exchange speeds over copper wires have reached the physical limit - it is extremely difficult to achieve the transmission of a signal of sufficient strength at any considerable distances at speeds of 10 Gb/s and higher.

An experimental optical data transmission system based on silicon lasers created at Intel laboratories will allow replacing traditional conductors with ultra-thin and light optical fibers that can transmit huge amounts of information over long distances without significant losses.

The prototype of the Silicon Photonics Link system with a throughput of 50 Gb/s includes a silicon transmitter and receiver. The transmitter consists of four hybrid silicon lasers and optical modulators that convert data into light beams, capable of transmitting data at speeds up to 12.5 Gb/s each. These four beams are connected into one using a multiplexer and transmitted over a single fiber optic cable. Then a ray of light enters the receiver, a demultiplexer divides it into four channels, and photodetectors convert the light fluxes back into an electrical signal.

The throughput of the silicon-optical networks will allow transmitting a three-dimensional video signal to a screen of the size of the entire wall with such a high resolution that it will create the full impression of the presence of actors from the film or interlocutors in the room by teleconference.

Neurocomputers

The term "neurocomputer" is used to denote the entire spectrum of work within the framework of the approach of artificial intelligence systems creation based on modeling elements, structures, interactions and functions of various nervous systems. Since currently research in this area is carried out mainly at the level of neural network models, the understanding of the term "neurocomputers" is narrowed down, putting an equal sign between it and neural networks.

Depending on the method of implementing models of neural networks, 4 levels of neurocomputers are distinguished.

Level 0. **Theoretical.** Works in which, in one form or another (mathematical, algorithmic, verbal, etc.), a description of neural network models is presented.

Level 1. **Software.** Models of neural networks, software implemented on ordinary serial computers.

Level 2. **Software and hardware.** Co-processors to accelerate the simulation of neural networks.

Level 3. **Hardware.** Physically implemented models of neural networks.

The specificity of neural network operations, as well as the superparallelism of the structure and functioning of neural network models, extremely slows their implementation on ordinary serial computers. The need to carry out a large amount of research work and the rapid functioning of the emerging application systems led to the emergence of specialized computing devices for the efficient simulation of neural networks - neurocomputers in the narrow sense of the word. Such an interpretation, corresponding to levels 2 and 3 according to the given classification, is widespread.

An elementary creation element of a neural network (NN) is a neuron that carries out a weighted summation of the signals arriving at its input x_1, x_2, \dots, x_n , see fig. 17.3. The result of this summation forms an intermediate output signal, which is converted by the activation function into the output signal of the neuron - Y . By analogy with electronic systems, the activation function can be considered a nonlinear amplifying characteristic of an artificial neuron having a large gain for weak signals (with incident gain for large excitations). The gain is calculated as the ratio of the output signal of the neuron to the small increment of the weighted sum

of the input signals that caused it. In addition, to ensure an increase in the processing power of multilayer NN, in comparison with single-layer ones, it is necessary that the activation function between the layers be nonlinear, i.e. as shown in considering the associativity of the matrix multiplication operation, any multilayer neural network without non-linear activation functions can be reduced to an equivalent single-layer neural network, which are very limited in their computational capabilities. But along with this, the presence of non-linearities at the output of a neuron cannot serve as a determining criterion; neural networks are well known and successfully work without non-linear transformations to the output, which are called neural networks on delay lines.

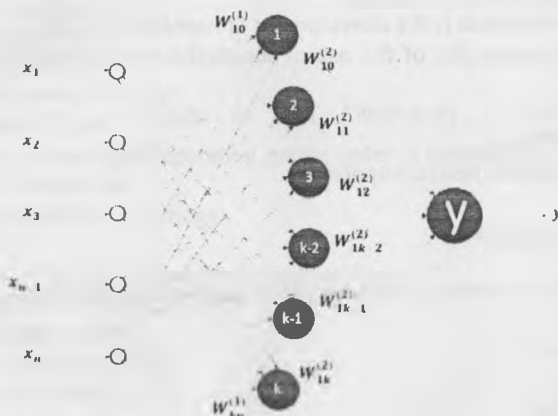


Figure 17.3. General view of the neuron

The set of output signals of the neurons of the network y_1, y_2, \dots, y_N is called the output activity vector, or the neural network activity pattern. It is convenient to represent the weights of the connections of the network neurons in the form of a matrix W , where ω_{ij} is the weight of the connection between the i - and j -th neurons. In the process of functioning (evolution of state) of the network, the input vector is converted into the output one, i.e. some processing of information that can be interpreted, for example, as a function of hetero- or auto-associative memory. The specific type of information transformation performed by the network is determined not only by the characteristics of neural-like elements, but also by the features of its architecture, i.e. one or another topology of interneuronal connections, the choice of certain subsets of neural-like elements for input and output of information or lack of competition, the direction and methods of controlling and synchronizing information flows between neurons, etc.

Among the main advantages of NNs in [22, 23], there are noted the invariance of NN synthesis methods to the dimension of the feature space and NN sizes, the adequacy of modern promising technologies, and fault tolerance in the sense of a monotonous rather than catastrophic change in the quality of the solution to the problem depending on the number of failed elements.

As noted, a neurocomputer is a computing system with MSIMD architecture, i.e. with parallel flows of the same commands and multiple data flows.

The issues of creation and using of NN are beyond the scope of this tutorial, we can recommend an interested reader to refer to the literature [21-24].

Questions for self-control

1. Formulate the main trends in the development of technologies used in microprocessors.
2. What ideas were the basis of quantum computers?
3. What is nanotechnology?
4. In what areas is the development of nanotechnology?
5. Give examples of the use of nanotechnology in the design of computer nodes.
6. What is photonics? Tell us about the use of photonics in telecommunications.
7. Define the neurocomputer.

Test questions

1. What is a unit of measurement of quantum information?
A) bit
B) byte
C) dit
D) Q-bit
2. Who owns the idea of creating a quantum computer?
A) Richard Feynman
B) Shore
C) Paul
D) to all three
3. What approach of creating a quantum computer does not exist?
A) high-resolution pulsed nuclear magnetic resonance (NMR) spectrometer
B) the use of so-called ion traps, or "suspended" in a vacuum of ions
C) solid state quantum computer
D) a quantum computer on neurons
4. When and by whom was the first working quantum computer demonstrated?
A) 2009 Canadian company D-Wave
B) 2007 IBM
C) 2010 Intel
D) not yet demonstrated

5. What is Orion?
- A) quantum computer company
 - B) the first quantum computer
 - C) unit of measurement of quantum information
 - D) quantum computer processor
6. In which areas of nanotechnology are there no researches?
- A) molecular electronics
 - B) biochemical and organic solutions
 - C) quantum computers
 - D) neural networks
7. What is nanotechnology?
- A) new technologies
 - B) these are technologies operating on the order of nanometer
 - C) costly technologies
 - D) new information technology
8. Who and when developed the first nanotransistor?
- A) Scientists at D-Wave (2005)
 - B) Intel scientists (2007)
 - C) Philips scientists (2002)
 - D) Philips scientists (2012)
9. What is the construction of the first nanotransistor made of?
- A) Indium arsenide and aluminum superconducting contacts
 - B) lanthanum aluminate
 - C) strontium titanate
 - D) germanium
10. What is the logical function of the carbon monomolecular structure in the form of a nanotube made by IBM Research?
- A) and
 - B) OR
 - C) NOT
 - D) AND NOT
11. ... is a technology of radiation, transmission, registration of light using fiber optics and optoelectronics
- A) photonics
 - B) mechatronics
 - C) Phototronics
 - D) optomatronics

12. What is used to produce a variety of microcircuits?
- A) silicon
 - B) aluminum
 - C) germanium
 - D) strontium
13. Depending on the method of implementing models of neural networks, the following level of neurocomputers does not exist
- A) theoretical
 - B) practical
 - C) software
 - D) hardware
14. What is the purpose of the activation function in neural networks?
- A) to convert the intermediate output signal to the output signal of a neuron
 - B) to activate the operation of neural networks
 - C) to start training
 - D) all answers are correct
15. The main advantages of neural networks does not apply
- A) the invariance of the methods of synthesis of NS to the dimension of the space of attributes and the size of the NS
 - B) the adequacy of modern advanced technologies
 - C) fault tolerance
 - D) noise immunity

CHAPTER 18. BRIEF REVIEW ABOUT THE ASSEMBLER

Today in the software market there are a huge number of products written in high-level languages. Against this background, programming in a low-level language - assembler - may seem somewhat outdated and irrational. However, it should be noted that assembler is actually the language of the processor, which means that you cannot work without it as long as processors exist. The main advantages of assembler programming language are maximum speed and minimum size of the resulting programs. In addition, the assembler allows the programmer to perform actions that either cannot be implemented at all in other languages, and, in particular, in high-level languages, or which will take too much computer time if expensive means of a high-level language are involved.

The disadvantages are often due to the tendency of the modern software market to prefer quantity to quality. As its size increases, the assembler program loses its visibility. This is due to the fact that assembler programs should pay a lot of attention to details. Language requires planning each step of the computer. In the case of small programs, this makes them optimal in terms of the efficiency of using hardware. However, in case of creating complex applications, a huge amount of details can interfere with optimizing the program as a whole. To program in assembler, you need to know very well the structure of the computer and the operation of hardware devices. From the foregoing, we can conclude that in assembler language can be written any application, any program, but for writing complex applications it is better to use high-level languages such as C ++, C # or Pascal, which will allow to focus on the task itself, and you don't need to take into account and to evaluate the features of the device and the microprocessor.

Since in the framework of this tutorial, we are not aiming to consider in detail all the features of assembler programming language (this issue is discussed in more detail in the books of V.I. Yurov, A.S. Krupnik, V.Yu. Pirogov and others), this section describes the basics of working in assembler with simple examples that are easy to reproduce with any computer on hand.

18.1. Assembler Program Structure

An assembler program is a collection of memory blocks called memory segments. A program may consist of one or more of these block segments. Each segment contains a set of language sentences, each of which occupies a separate line of program code.

Assembler sentences come in four types:

1. commands or instructions, which are symbolic analogues of machine instructions. In the process of translation, assembler instructions are converted into the corresponding commands of the microprocessor instruction system;
2. Macros - sentences of a program text formatted in a certain way, replaced during translation by other sentences;
3. directives, which are an instruction to the assembler translator to perform certain actions. Directives have no analogues in machine representation;

4. comment lines containing any characters. Comments are ignored by the translator.

Valid characters at writing program text are:

1. all latin letters: **A-Z, a-z**. In this case, upper and lower case letters are considered equivalent;

2. numbers from **0 to 9**;

3. signs **?, @, \$, _, &**;

4. delimiters **, . || () <> {} + / *% ! ' " ? \ = # ^**.

Assembler sentences are formed from *tokens*, which are syntactically inseparable sequences of valid language characters that make sense for the translator.

Tokens are:

- *identifiers* - sequences of valid characters used to designate program objects such as operation codes, variable names, and label names. An identifier may consist of one or more characters. As characters, you can use the letters of the Latin alphabet, numbers and some special characters - **_**, **?**, **\$**, **@**. The identifier cannot begin with a digit character. The length of the identifier can be up to 255 characters, although the translator only accepts the first 32, and ignores the rest. You can adjust the length of possible identifiers using the **mv** command line option. In addition, it is possible to instruct the translator to distinguish between uppercase and lowercase letters or ignore their difference (which is done by default). To do this, use the command line options **/ mu**, **/ ml**, **/ mx**;

- *character strings* - sequences of characters enclosed in single or double quotes;

- *integers in one of the following number systems: binary, decimal, hexadecimal.*

Almost every sentence contains a description of the object over which or by which some action is performed. These objects are called operands. Operands are objects (for example, registers or memory cells) that are acted upon by instructions or directives, or they are objects that define or specify the effect of instructions or directives.

The operands can be combined with arithmetic, logical, bitwise and attributive operators to calculate a certain value or determine the memory cell that this command or directive will affect.

You can perform the following classification of operands:

- permanent or direct operands;

- address operands;

- movable operands;

- address counter;

- register operand;

- base and index operands;

- structural operands.

For each processor architecture and for each OS or OS family, there is its own Assembler. There are also so-called "cross-assemblers" that allow machines on one architecture (or in the environment of one OS) to assemble programs for

another target architecture or another OS, and get executable code in a format suitable for execution on the target architecture or in the target environment OS

The most famous assemblers for the DOS operating system were Borland Turbo Assembler (TASM) and Microsoft Macro Assembler (MASM). Also, at one time, the simple assembler A86 was popular.

Initially, they supported only 16-bit instructions (before the appearance of the Intel 80386 processor). Later versions of TASM and MASM support 32-bit commands, as well as all commands entered in more modern processors, and command systems specific to a particular architecture (such as, for example, MMX, SSE, 3DNow! Etc.).

Convenience of programming, modest system requirements and high speed of translation provided TASM with leadership throughout the entire existence of MS-DOS. Currently Borland (Codegear) has stopped distributing its assembler.

With the advent of the Microsoft Windows operating system, a TASM extension called TASM32 appeared, allowing to create programs to run on Windows.

Microsoft supports its product called Microsoft Macro Assembler. In addition, Stephen Hutchesson created a MASM programming package called "MASM32".

A lot of books have been devoted to MASM (MASM32), which simplifies the learning process, and on the network you can find many source codes for assembler programs and libraries, freeing the programmer from the need to reinvent the wheel. MASM is also the output language for many disassemblers (Sourcer, IDA Pro). All this makes MASM the number one translator in programming for Windows / Intel.

Flat assembler (FASM) - assembler with extremely simplified syntax. FASM is a freeware multi-pass assembler written by Tomasz Gryshar. In addition to the basic instruction set of the processor and coprocessor, FASM supports instruction sets MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4a, AVX and 3DNow!, as well as EM64T and AMD64 (including AMD SVM and Intel SMX). Compiling a program in fasm consists of 2 stages: preprocessing and assembly.

At the preprocessor stage, all macros, symbolic constants are expanded, preprocessor directives are processed.

Unlike the assembly stage, preprocessing is performed only 1 time.

The only significant difference from the format adopted in other assemblers (MASM, TASM in MASM compatibility mode) is that the value of the memory cell is always written as [label_name], but simply label_name means the address (that is, the serial number) of the cell. This allows to do without the **offset** keyword. Also, in fasm, when overriding the size of the operand, instead of byte ptr, it simply says byte, instead of word ptr, it is word, etc. It is not allowed to use several square brackets in one operand, so instead of [bx][si] you need to write [bx+si]. These syntax changes have led to more unified and easier to read code.

Due to its modular architecture, adapting fasm to code generation for other platforms is very easy. The latest version introduced support for AMD64 / EM64T,

fasm has been tested many times to create the correct object files under Win64, so it is one of the few open development tools for the Win64 platform.

18.2. Assembly Programming Examples

18.2.1. Inline assembler

In this section, we will look at examples of using assembly language inserts in programs written in high-level languages.

The built-in assembler allows to include assembler operators in your C # or C ++ programs. Inline assembler instructions are compiled and assembled with your program, and you do not need to write separate modules.

To include assembler instructions in C ++ code, use the `asm` keyword and the following format:

```
asm operation_operand operands;
```

where "operation_code" is a valid processor instruction 80x86;

"operands" contain operands (operand) valid for the specified operation (constants, variables, and labels). The end of the `asm` operator is a character; or a new line.

After a semicolon, a new `asm` statement can be placed on the same line, but the statement cannot continue on the next line. To include several `asm` statements, you can enclose them in braces (the first bracket must be on the same line as `asm`):

```
asm {  
  pop ax; pop ds  
  iret  
}
```

The assembler part of the statement is copied directly to the output and included in the assembly language statements that Borland C ++ or VC ++ (VC #) generates for C ++ (C #) instructions. All C ++ identifiers (C #) are replaced with the corresponding assembler equivalents. Each `asm` statement is considered a C ++ (C #) statement.

Example 1. The operation of addition.

The addition commands are quite simple. The `INC` command increments, increasing the contents of the operand by one, for example `INC EAX` or `INC DWORD PTR [EBX]`. The `INC` command sets the flags **SF**, **ZF**, **AF**, **PF** depending on the result of addition.

The `ADD` command allows to add two operands. The result is placed in the first operand (receiver). For example, executing the `ADD EBX, 10` command, the number 10 is added to the contents of the `EBX` register. After the command is executed, the result will be in the `EBX` register. The first operand can be a register

or a variable. The second operand (source) is a register, variable, or number. It is not possible to perform the addition operation simultaneously on two variables.

The *XADD* command is similar to the *ADD* command, but with some difference: it first transfers the operand - receiver to the operand - source, and then puts the result of addition into the receiver operand. The first operand can be a register and a variable, the second - only a register. The action on flags is similar to the *ADD* command.

The *ADC* instruction adds two operands like the *ADD* instruction. Using this command, you can add numbers when the result exceeds 32 bits or initially the length of the operands exceeds 32 bits. Suppose that the first number is in a pair of registers *EDX: EAX*, and the second number is in a pair of *ECX: EBX*. Then the addition of these numbers can be represented as follows:

```
ADD EAX, EBX
```

```
ADC EDX, ECX
```

The result of the addition will be contained in a pair of registers *EDX: EAX*.

For the first example, we will examine in detail the progress of its implementation in C ++ **Builder 6 (2009) and Visual Studio 2008**.

We use Visual Studio 2008 as the compiler. In the New Project window, select the *Win32 Console Application* (see fig. 18.1). Enter the name of the project, for example, Ki. Click OK. In the Win32 Application Wizard window, click Next (see fig. 18.2).

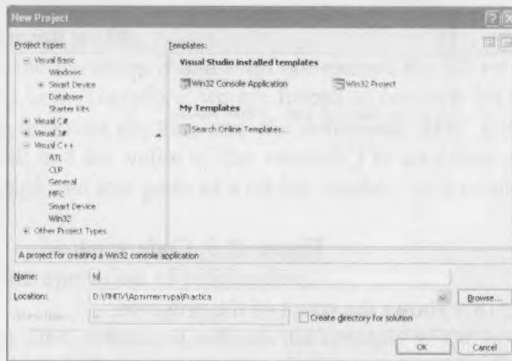


Figure 18.1. New Project



Figure 18.2. Win32 Application Wizard

In the codes window (see fig. 18.3) we will introduce the following example codes.

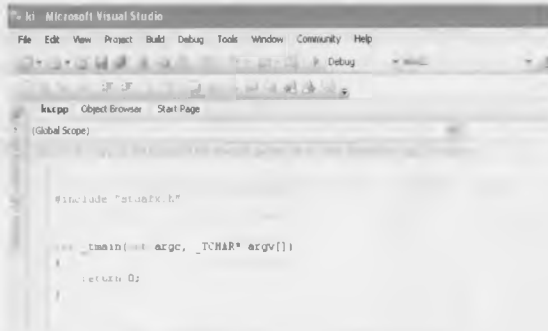


Figure 18.3. Code window

The fig. 18.4 shows the result of the program.

```
#include "stdafx.h"
#include <windows.h>
#include <conio.h>
#include <stdio.h>
int a,b,c;
DWORD d,e,f;
void main ()
{
    a=100; b=-200;
    f=0;
    d=0xffffffff;
    e=0x10;
```

```

asm{
// Addition of positive and negative numbers
    MOV EAX, a
    ADD EAX, b
    MOV c, EAX
// Addition of large numbers
    MOV EAX, e
    ADD d, EAX
    ADC f, 0
}
printf("%d %x %x", c, f, d);
getch ();
}

```

```

cmd: D:\ИПГУ\АрхитектураПрактика\KI\debug\KI.exe
-100 1 f

```

Figure 18.4. Program result

At the beginning of the assembler code, two signed numbers are added. Moreover, $b < 0$ ($b = -200$). The result is placed in a variable c and displayed as a signed number - the result is 100.

The values of the variables d and e do not exceed the 32-bit boundary, but the result does not fit in it. Therefore, we are forced to consider bit transfer over a 32-bit boundary. In order to do this, use the command $ADC\ f, 0$. The second operand is equal to 0, and the value of the variable f is also first equal to 0. We print separately the high and low parts of a 64-bit number. As a result, we see $1f$ on the screen.

Example 2. The operation of subtraction.

The decrement DEC command reduces the contents of the operand (register or variable) by 1. It acts in the same way as the increment on the flags, **SF**, **ZF**, **AF**, **PF**.

The SUB instruction subtracts the right operand from the left operand and affects the flags **CF**, **SF**, **ZF**, **AF**, **PF**. The left operand can be a register or a variable, the right operand can be a register, or a numerical constant. It makes no difference to the command whether the numbers are symbolic or not.

The SBB command is similar to SUB , but additionally subtracts the **CF** flag from the receiver. It is used to work with numbers whose length exceeds 32 bits.

Let the first number be contained in a pair of registers EDX : EAX , and the second in a pair of ECX : EBX . Then the subtraction of these numbers can be described by a couple of commands:


```
SUB EAX, EBX
SBB EDX, ECX
```

The result of the subtraction will be contained in the *EDX: EAX* pair.
Subtraction operation listing (C ++ Build 2009).

```
#include <vcl.h>
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
#pragma hdrstop
#pragma argsused
int a,b,c;
__int64 i,j,k;
void main ()
{
    a=100; b=-200;
    i=0x1fffffff;
    j=0x1fffffff;
    __asm {
//subtraction of 32-bit numbers
MOV EAX, a
SUB EAX, b
MOV c, EAX
//subtraction of 64-bit numbers
MOV EAX, DWORD PTR i
MOV EDX, DWORD PTR i+4
MOV EBX, DWORD PTR j
MOV ECX, DWORD PTR j+4
SUB EAX,EBX
SBB EDX,ECX
MOV DWORD PTR k,EAX
MOV DWORD PTR k+4,EDX
    }
printf("%d %l64x",c,k);
getch();
}
```

The fig. 18.5 shows the result of the program.



Figure 18.5. Program result

Performing the subtraction for 32-bit numbers, we act similarly to the addition operations.

For 64-bit numbers, we use variables that are of type `__int64`. They occupy 8 bytes, and the compiler supports actions with them.

For assembly operations, you must use the pairs of registers *EDX: EAX* and *ECX: EBX*. Thus, in order to get the result of subtraction, we use a pair of commands - *SUB / SBB*. The result is placed in a variable of type `__int64`. To load a 64-bit number, you must use two *MOV* commands, loading the lower and then the highest part of the number first.

Example 3. The operations of multiplication and division.

Unlike addition and subtraction operations, multiplication is sensitive to the number sign. There are two multiplication commands:

MUL - to multiply unsigned numbers;

IMUL - to multiply signed numbers.

We will analyze them in more detail.

MUL command. The only operand of this command can be a register or a variable. The size of this operand (source) is important here. If the operand is single-byte, then it is multiplied by *AL*, respectively, the result will be placed in the *AX* register, depending on whether it exceeds one byte or not. If the result does not exceed one byte, then the **OF** and **CF** flags will be zero, otherwise 1. If the operand is double-byte, then it is multiplied by *AX*, and the result will be placed in a pair of *DX: AX* registers. Accordingly, if the result fits entirely to the *AX*, that is, the contents of *DX* are 0, then the **CF** and **OF** flags will be equal to zero. If the source operand is 4 bytes long, then it is multiplied by *EAX*, and the result should be placed in a pair of *EDX: EAX* registers. If the contents of *EDX* after performing the multiplication turn out to be zero, then the **OF** and **CF** flags will also have a zero value.

The *IMUL* command has three different formats.

The first format is completely similar to the format of the *MUL* command. Second format:

IMUL operand1, operand2

operand1 must be a register, *operand2* may be a number, a register, or a variable. As a result of the multiplication of *operand1*operand2*, the result is

placed in *operand1*. However, we can get a number that does not fit in the receiver. In this case, the flags **CF** and **OF** will be equal to 1 (0 otherwise).

Third format:

IMUL operand1, operand2, operand3

In this case, operand (register or variable) is multiplied by operand3, and the result is placed in operand1. If overflow occurs during multiplication, the CF and OF flags are set.

Listing a Multiplication Operation (C ++ Build 2009)

```
#include <vcl.h>
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
#pragma hdrstop
#pragma argsused
DWORD a;
__int64 b;
int c,e;
void main()
{
a=100000;
c= -1000;
__asm {

// multiplication of unsigned numbers
MOV EAX, 100000
MUL DWORD PTR a;
MOV DWORD PTR b,EAX
MOV DWORD PTR b+4, EDX
// multiplication of signed numbers
IMUL EAX, c, 1000
MOV e, EAX
}
printf("%l64d %d" ,b,e);
getch();
}
```

The fig. 18.6 shows the result of the program.

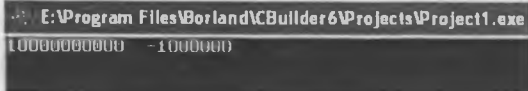


Figure 18.6. Result program

At multiplying unsigned numbers, we use two 32-bit numbers. The result of the operation should be placed in a pair of registers *EDX:EAX*. Then we place the values from this pair of registers in a 64-bit variable.

Performing the multiplication of signed numbers, we use the command format with three operands. The contents of the variable *c* are multiplied by *1000*. The result is placed in the register *EAX*.

Unsigned numbers are divided using the *DIV* command. A command has only one operand - a divider. The divider can be a register or memory cell. Depending on the size of the divider, a dividend is also chosen.

The following options are possible:

1. The divider is 1 byte in size. In this case, the dividend is placed in the *AX* register. The result of the division (private) will be in the *AL* register, while in the register *AH* there will be the remainder of the division.

2. The divider is 2 bytes in size. The result of the operation will be placed in the register *AX*. The remainder of the division is in the *DX* register.

4. The divider is 4 bytes in size. The dividend will be in a pair of registers *EDX:EAX*. The result of the operation will be placed in the *EAX* register. The remainder of the division is in the *EDX* register.

The sign division command *IDIV* is completely similar to the *div* command. It is significant that for the division commands the values of the flags of arithmetic operations are not defined. As a result of the division, either overflow or division by 0 may occur. In this case, an exception occurs - a special procedure will be called, which should handle the situation that has arisen. Such processing should be provided by the operating system.

Listing division operation (C ++ Build 2009).

```
#include <vcl.h>
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
#pragma hdrstop
#pragma argsused
DWORD a,b,c;
void main()
{
a=100000;
__asm{
```

```

// unsigned division
MOV EAX, a
MOV EDX, 0
MOV EBX, 20
DIV EBX
MOV b, EAX; // quotient
MOV c, EDX; //remainder
}
printf("%d %d ",b,c);
  getch();
}

```

The fig. 18.7 shows the result of the program.



Figure 18.7. Result program

Example 3. Methods for addressing memory (Visual Studio 2008).

```

#include "stdafx.h"
#include <windows.h>
#include <conio.h>
#include <stdio.h>
BYTE ar[6] = {1,12,128,50,200,10};
BYTE a,b,c,d;
WORD e;
DWORD f;
void main()
{
  __asm
  {
    MOV AL,BYTE PTR ar
    MOV a,al
    LEA EBX,ar
    MOV AL,BYTE PTR [EBX]
    MOV b,AL
    MOV AL,BYTE PTR [EBX]+3
    MOV c,AL
    MOV EDX,1
    MOV AL,BYTE PTR [EBX][EDX]+2
    MOV d,AL

```

```

MOV AX,WORD PTR [EBX]+2
MOV e,AX
MOV EAX,DWORD PTR [EBX]+2
MOV f,EAX
}
printf("%u %u %u %u %u %u",a,b,c,d,e,f);
getch ();
}

```

The fig. 18.8 shows the result of program execution.

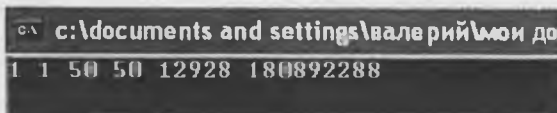


Figure 18.8. Result program

This example is created mainly on indirect addressing. Indirect addressing uses one of the common name registers. In this case, the *EBX* register is used, but any other register can be used. The format of the command is such that you can change the address of the memory cell without changing the contents of the register itself: $[EBX] + 2$. In this case, the resulting address is obtained by adding the address stored in the *EBX* register and the number 2. In order to change the address of the request, you can use another register: $[EBX] [EDX] + 2$. In this case, the resulting address is obtained by adding the contents of the *EBX* register, the *EDX* register, and the number 2. The expression $[EBX] [EDX] + 2$ can also be replaced by $[EBX + EDX + 2]$.

The following should be noted:

- that address space and memory that we cover is the property of our program only. There is no place for other programs. But we cannot go beyond the area allotted to us, and we will turn to the code of another program or memory area occupied by the kernel of the operating system;
- the real address is formulated not only from the 32-bit value that we use, but also using segment registers.

18.2.2. Examples in MASM and FASM

Example 1. Comparison of MASM and FASM

This simple example allows to see clearly some of the differences when using the two most popular assemblers for Windows.

MASM32

```
.386
.model flat, stdcall
.option casemap .none ; case
sensitive
include
\masm32\include\windows.inc
include \masm32\include\user32.inc
include
\masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
.code
.data
MsgBoxCaption db "Win32
Программа",0
MsgBoxText db "Привет
пользователь пк!",0
.code
start:
invoke MessageBox, NULL, ADDR
MsgBoxText, \
ADDR MsgBoxCaption, MB_OK

invoke ExitProcess, NULL
end start
```

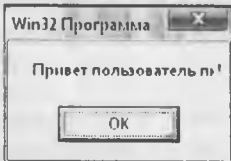


Fig. 18.9. The result of the MASM32 program execution

Let's analyze the listings.

MASM32

.386 indicates that there are used processor instructions no higher than 386. *include user32.inc*, *include kernel32.inc*, *include WINDOWS.INC* - a list of plug-ins that are in the libraries *user32.dll*, *kernel32.dll*.

The *.data* section contains data (which, if necessary, can be extracted).

MsgBoxCaption is a label (displacement). The number, 0 indicates the end of the line.

MsgBoxText is similar to *MsgBoxCaption*.

FASM

format PE GUI 4.0

```
include
'c:\fasmw\INCLUDE\WIN32AX.INC'

.data
MsgBoxCaption db "Win32
Программа",0
MsgBoxText db "Привет
пользователь пк!",0
.code
start:
invoke MessageBox, NULL,
MsgBoxText, MsgBoxCaption, MB_OK
invoke ExitProcess, NULL
.end start
```

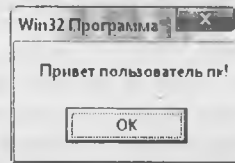


Fig. 18.10. The result of the FASM program execution

Section code (*.code*) - a set of processor instructions arranged in a certain way in order to get something useful.

Start: this is the label (offset) that tells the compiler where to start the program.

Invoke command, invented by Stephen Hutchesson (Australia) to simplify recording under Windows. *ExitProcess* has one *NULL* argument or just 0, with a *MessageBox* it is more complicated - 4 arguments.

ADDR MsgBoxText - transmission of the label address *MsgBoxText*. The *ADDR* command is valid only in the context of the *INVOKE* directive. You cannot use it to assign a label address to a register or variable. In this example, you can use *OFFSET* instead of *ADDR*.

MB_OK window type with one **OK** button.

FASM

format PE GUI 4.0 - the line describes the format of the output file. It is not necessary to write if you create a standard windows exe file, if you need a dll, or not a standard exe file, you need to specify additional parameters, for example: *format PE GUI 4.0 dll*

include '..\fasm\INCLUDE\WIN32AX.INC' - a library.

In FASM *ADDR* is optional.

Example. 2. Arithmetic operations in FASM

```
format PE GUI 4.0
entry start ; Program entry point
INCLUDE 'E:\FASM\INCLUDE\win32ax.inc'
INCLUDE 'E:\FASM\INCLUDE\encoding\win1251.inc'
INCLUDE 'E:\FASM\INCLUDE\api\user32.inc'
section '.data' data readable writable
formats db " %d ",0
result db 256 dup(?) ; Converting the number to a line,
; save the result here
section '.code' code readable executable
start:
; Addition of numbers 2 and 2
mov eax,2 ; Move number 2 to eax
mov edx,2 ; Move number 2 to edx
add eax,edx ; Add the contents of eax and edx (2 + 2)
; Outputs the result in eax.
invoke wsprintf,result,formats,eax ; Convert the number (result) to
; a line in order
; to display it on the screen.
invoke MessageBox,0,result,"Addition",MB_OK ; Outputs the result
; on the screen.
```



```

mov eax,10      ; Subtract 5 from 10
mov edx,5      ; Move number 10 to eax
sub eax,edx    ; Move number 5 to edx
               ; Subtract from the eax contents edx (10-5).
               ; Outputs the result in eax.
invoke wsprintf,result,formats,eax ; Converting the number (result)
               ; to a line in order
               ; to display it on the screen.
               ; Outputs the result on the screen.

```

```
invoke MessageBox,0,result,"Вычитание",MB_OK
```

```

mov ax,2      ; Multiplication of 2 by 3
imul ax,3    ; Move 3 to ax
             ; Multiply the content of ax by 3 (2*3).
             ; The result is located in eax.
invoke wsprintf,result,formats,eax ; Converting the number (result)
               ; to a line in order
               ; to display it on the screen.

```

```
; Outputs the result on the screen.
```

```

invoke MessageBox,0,result,"Умножение",MB_OK
invoke ExitProcess,0 ;Exit the program
section '.idata' import data readable
library kernel32,'KERNEL32.DLL',user32,'USER32.DLL'
INCLUDE 'E:\FASM\INCLUDE\api\kernel32.inc'

```

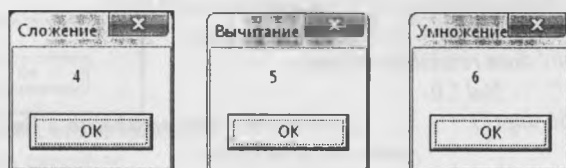


Figure 18.11. The result of the program

The *INCLUDE* directive, inserts the text from another file into the specified location. Open the *WIN32AX.INC* file in the **INCLUDE** folder using notepad or in FASM itself and make sure that we automatically connect the text from *win32a.inc*, *macro / if.inc*, and a common set of Windows function libraries to our program. With the help of plug-in files, we organize a semblance of a high-level language: in order to avoid the routine of describing each function manually, we connect whole libraries of descriptions of standard Windows functions.

Next, we have designated the data section - *.data*. In this section, we declare variables. The *db* command means *define byte*. The next section is the executable program code - *.code*. At the beginning of the section is the label *start*. It means that it is from this place that our program will begin to be executed. The *invoke* command (macroinstruction) calls the MessageBox API built into Windows.

Questions for self-control

1. List the basic software in assembler language.
2. What assembler operators are used to perform arithmetic operations?

Test questions

1. CPC <operand>, <shift_control> is the command format
 - A) shear
 - B) control transfer
 - C) data transfer
 - D) organization of cycles
2. The result of the linker is a file with the extension
 - A) exe B) obj C) lst D) crf
3. Launching the debugger for assembler is done by the command line
 - A) td.exe executable_name
 - B) tlink.exe / v object_module_name
 - C) tasm.exe / zi object_module_name
 - D) there is no correct answer
4. In order to organize cycles the following command is used
 - A) there is no correct answer
 - B) data transfer
 - C) logical
 - D) microprocessor control
5. In order to increment the value of the command counter in the loop commands, use the command
 - A) inc
 - B) dec
 - C) adc
 - D) there is no correct answer
6. Performing operations of adding binary numbers with a sign, it is necessary to analyze the state of the flags
 - A) carry (cf) and overflow (of)
 - B) overflow (of) and sign (sf)

- C) carry (cf) and sign (sf)
- D) there is no correct answer

7. In the mul and imul commands, use the immediate value as the operand

- A) it is impossible
- B) it is possible
- C) it is possible if it does not exceed 128
- D) there is no correct answer

8. A cyclic left shift through the flag is performed by the command

- A) rcl B) rcr C) rol D) ror

9. A simple cyclic shift to the right is performed by the command

- A) ror B) rcl C) rcr D) rol

10. In order to convert data according to the rules of formal logic, use the commands

- A) and, or, xor, not
- B) shl, shr, sal, sar
- C) and, or, inc, not
- D) there is no correct answer

11. Unconditional jump is performed by the command

- A) jmp
- B) jcc
- C) jcxz
- D) there is no correct answer

12. The address of the command from which the interrupted program will continue to be executed is contained in a pair of registers

- A) cs: ip
- B) ss: ip
- C) ds: ip
- D) there is no correct answer

13. What does not belong to the main advantages of assembler language programming

- A) maximum performance
- B) minimum size
- C) perform actions that cannot be implemented at all in high-level languages
- D) GUI

14. What does not apply to assembler statements?

- A) commands or instructions
- B) drawings

- C) macros
- D) directives

15. What is not a token?

- A) identifiers
- B) character strings
- C) integers in one of the following number systems: binary, decimal, hexadecimal
- D) real numbers

REFERENCES

1. Apokin I.A. The development of computers / Apokin I.A., Maistrov L.E. - M.: Nauka, 1990.- 262 p.
2. Broydo V. L. Architecture of computers and systems / Broydo V. L., Ilyina O. P.: Textbook for high schools. 2nd edition. - St. Petersburg: Peter, 2009.- 720 p.
3. Gilmore C. Introduction to microprocessor technology: [Translation] / Charles Gilmore. - Tbilisi: Tbil Publishing House. University, 1989 - 334 p.
4. Drozdov EA Fundamentals of the construction and functioning of computing systems / Drozdov EA, Pyatibratov AP - M.: Energy, 1973. - 368 p.
5. Zhmakin A.P. Computer Architecture / Zhmakin A.P. - BHV-Petersburg, 2010. - 352 p.
6. Kagan B.M. Electronic Computing Machines and Systems / Kagan B.M. - M.: Energoatomizdat. 2001.- 212 p.
7. Kartsev M.A. Arithmetic of digital machines / Kartsev M.A. - M.: Nauka, 1969- 134 p.
8. Kuzin A.V. Computer Architecture and Computing Systems / Kuzin A.V., Peskova S.A. - Forum, 2011. - 350 p.
9. Lakhno V.A. Applied Theory of Digital Automata / Lakhno V.A., Mogilny G.A., Petrov A.S. - Lugansk: VNU, 2009. - 248 p.
10. Malinovsky B.N. History of computer technology in persons / Malinovsky B.N. - K. : KIT firm, PTOO A.S.K., 1995. - 384 p.
11. Savelyev A.Ya. Arithmetic and logical foundations of digital automata / A. Saveliev. - M.: Higher school. 1980- 312 p.
12. Smirnov A.D. Architecture of Computing Machines / Smirnov A.D. - M.: Nauka, 1990- 320 p.
13. Tanenbaum E. Computer architecture / Tanenbaum E., Austin T. - St. Petersburg: Peter, 2018- 816 p.
14. Tocchi R. Digital systems. Theory and Practice, 8th edition / Tochi R., Widmer J., Neil S. - M.: Williams, 2004- 1024 p.
15. Chu Ya. Organization of computers and microprogramming / Chu Ya. - M.: Mir. 1989 - 592 p.
16. Evans, James S. and Richard H. Eckhouse, Alpha RISC Architecture for Programmers. Upper Saddle River, N.J.: Prentice Hall PTR, 1999.
17. Gerrit A. Blaauw, Frederick P. Brooks, Jr. Computer Architecture: Concepts and Evolution Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA. 1997.
18. Training on the Internet. Free distance learning in computer science, telecommunications, the basics of electronic business / [electronic resource]. - 2011. - <http://www.lessons-tva.info>
19. Computer Architecture / [electronic resource]. - 2011. http://www.psut.edu.jo/sites/qaralleh/CA/ca_doc/Computer%20Architecture_review.pdf
20. Quantum computers / [electronic resource]. - 2011. <http://www.wikipedia.org>

21. Prospects for the development of computer technology / [electronic resource]. - 2011. - <http://pcterra.org/pers.html>
22. Neural networks / [electronic resource]. - 2011. - <http://www.neuroproject.ru/neuro.php>
23. Neural networks / [electronic resource]. - 2011. - <http://www.statsoft.ru/home/textbook/modules/stneunet.html>
24. Akhmetov B.S., Gorbachenko V.I. Neural networks. Textbook. - Almaty: KazNITU named after K.I. Satpayev, 2016- 256 p.

Tutorial

Akhmetov B.S., Lakhno V.A., Malikova F.U.

COMPUTER ARCHITECTURE

Tutorial