

A.M. Sergiyenko

Computer Architecture

Part 1. Central Processing Unit and Peripherals



Kyiv-2016

УДК 004.383 ББК 32.973-018 Рецензенти: А. М. Романкевич,
доктор технічних наук, професор (кафедра спеціалізованих
обчислювальних систем Національного технічного університету
України "КПІ"); В. П. Сімоненко, доктор технічних наук, професор
(кафедра обчислювальної техніки Національного технічного
університету України "КПІ")

Сергієнко А. М. Архітектура комп'ютерів. Частина 1. Центральний
процесорний елемент та периферійні пристрой: Конспект лекцій.
Англійською мовою - К.: НТУУ«КПІ», 2016. - 182 с. Розглянуті
теоретичні відомості про архітектури сучасних комп'ютерів. Для
студентів, аспірантів, викладачів вузів в галузі електроніки, вимірю-
вальної і обчислювальної техніки, зв'язку та захисту інформації. УДК
004.383 ББК 32.973-018

© А. М. Сергієнко, 2016

CONTENTS

FOREWORD	3
1. CENTRAL PROCESSING UNIT ARCHITECTURE	5
1.1 Introduction. Basic definitions and principles of computers	5
1.2 Element basis of computers	33
1.3. Von Neumann Computer Architecture	44
1.4. Instruction set of the computer	51
1.5. Data formats and operations	56
1.6. Addressing modes	69
1.7 Intel 8051 architecture	79
1.8 RISC and CISC processors	94
1.9. Interrupt system	103
2. PROCESSOR ENVIRONMENT ARCHITECTURE	120
2.1. Interface basics	120
2.2 Memory in computers	142
2.3. External storage devices	156
2.4 Computer console	169
2.5. Structure of a single processor computer	177
BIBLIOGRAPHY	180
ANNEX 1	182

FOREWORD

The computer architecture is the basic information about a computer, which is necessary both for its design and programming. The development of a new program is the creative process of implementation of a given algorithm on a computer with the specific architecture. The programmer creating a new program must clearly understand what processes and in what order are executed in the computer, which implements that program. Otherwise, the calculation process, which is planned by the programmer, may differ from the computational process, which is really running in a specific computer. As a result, the purpose of the program development may be unachievable.

Every year, more complicated computer architectures are in operation, increasing the degree of inherent parallelism, and the level of information security. At the same time, many programmers understand and percept the computer architecture as unchanged one for decades. On the one hand, this means that programmers can not achieve the most effective implementation of its programs in modern computers without the knowledge of new architectures. On the other hand, the particular organization of computing processes in these computers can lead to inefficient or even incorrect implementation of the programs, which are written for legacy architectures.

Also without architecture knowledge, it is impossible to develop, adjust, and analyse many complex and mission-critical applications that have special requirements for speed, reliability, information protection, energy consumption.

In the "Computer Architecture" discipline the students learn the theoretical information about the architectural elements of modern computers and communication between them, as well as gain the skills for

their use in the selection and programming of microprocessors. Therefore, this discipline is central to the education of specialists in the field of "Information Control Systems and Technologies", "Computer systems and networks".

1. CENTRAL PROCESSING UNIT ARCHITECTURE

1.1 Introduction. Basic definitions and principles of computers

To understand what is architecture, and why it is studied and investigated, we should recall the history of computers.

1.1.1 The history of computers and their architectures

The most significant impetus for the electronic computing machine invention has been a need for improvement of weapons during the Second World War. The problem of calculating the trajectory of shells and bombs was important for the preparation of firing and bombing tables. It was desirable to calculate such a data in real time. To speed up the solution of this problem in 1943 the University of Pennsylvania began to develop the first electronic computer ENIAC (Electronic Numerical Integrator And Computer) by the US Ministry of Defense order. It started its operation in 1946 and worked until 1955.

The Eniac computer had 18,000 electronic lamps and performed multiplication of decimal 10-bit numbers for 2.8 ms. It consisted of several integrators-accumulators, registers, constant and input-output data tables. It performed iterative algorithms for solving differential equations. The programming of such a computer consisted in connection of that registers, accumulators, etc. to the network via data lines in accordance with the algorithm graph.

At the first period, this network was done manually using the wires with connectors, and the switching field, which resembled the telephone exchange. But the algorithm changing process required several days. To

accelerate this process, the special patch panel with the network configuration was invented. Then the algorithm exchange was performed manually by replacing the "programmed" patch panel. So, Eniac was the programmable computer, but both the algorithm set, and data types were very limited.

In 1943 in England, ten copies of the Colossus computer have been developed and manufactured under the leadership of N. Wiener. They were used to decipher German radio messages. According to its principle of operation this computer was the symbolic correlator, which consisted of the 1,500 lamps. However, the structure of it also reflected the structure of the algorithm, and this algorithm was unchanged.

Even earlier — from 1939 to 1942 — D. Atanasof at the State College of Iowa developed the ABC electronic computer. It performed a single algorithm of the linear equation system solution. It has a parallel structure of thirty sequential Arithmetic Logic Units (ALU) with a parallel data storage based on the capacitor drum.

Thus, the first electronic computers were able to perform only one or more similar algorithms that have been encoded in their control devices or interconnect structure. Their ideology was borrowed from the mechanical or electro-mechanical calculators, which were distributed to the middle of the twentieth century. They were very costly. And the computer must be the versatile installation to be much more popular, and effective.

In 1936, Alan Turing has proposed the computational model, in which the arbitrary algorithm is coded as a table in the control unit, and as an initial state of the endless storage tape. This tape was the model of the computer memory. Thus the idea of a computer, that can perform any hypothetical algorithm, was born.

It is believed that the idea of implementing a universal programmable computer first appeared in the US in the Moore Electrical School of the

Pennsylvania University. There was a team of scientists led by von Neumann, which in 1946 drew up a prominent report. In this report, the programmable computer design principles were proposed. Among them were the ideas to store both data and program in a single computer memory, to perform the data processing using their binary representation and the apparatus of the Boolean algebra, to fetch the instructions using the instruction counter. Thus, the memory of a computer as the basis of its functionality and versatility was considered.

In those days, the computer structure was significantly dependent on the memory design. The register memory based on the vacuum tubes was fast but too expensive and unreliable. The memory based on the mercury acoustic delay lines was very popular. This memory was of the sequential type, and therefore, it determined the sequential, bit-wise processing of the data. The Williams cathode ray tube (CRT) recorded the data bits in the form of pixels on the tube screen and provided the random access to the data. But the volume of such a memory not exceeded thousands of bits.

T. Kilburn, and F. Williams were the first who in June 1948 have calculated the first program in the computer SSEM (Small-Scale Experimental Machine) at Manchester University. It was the world's first stored-program computer, which had successfully demonstrated the practicality of the stored-program approach and the Williams tube memory effectiveness. SSEM was redesigned into the Mark1 computer, which has runned the first program in April 1949.

The Cambridge professor M. Wilkes took over the idea of the Neumann's group and has implemented his first program in the computer EDSAC (Electronic Delay Storage Automatic Computer) in 1949. A year later, the computer EDVAC (Electronic Discrete Variable Automatic Computer) was in the operation in the Moore School. In fact, the Neumann group report was

compiled to develop this computer. Both these computers were built on the delay lines. Therefore, they operate synchronously and sequentially, that is, the words are processed bit by bit.

At the same period of time in Kiev, S. A. Lebedev has developed the computer MESM (Malaja Electronno-Stchetnaja Mashina), in which many new ideas have been incorporated. One of them is the parallel ALU data processing (such as was first used in the Whirlwind computer in 1951 in the USA). Second of them is the microprogramming principle (which was published in 1951 and implemented in 1957 by M. Wilks). MESM was running for the first time in 1950 and was put into operation in late 1951. Its successful use for complex calculations in the defense projects has contributed to the emergence, and expanding of the computer industry in the USSR.

The Univac-1 machine, which was produced from 1951 by the Univak affiliate of the Remington company, gave the big push for the commercial distribution of computers. It contained 5000 vacuum tubes and hundreds of delay lines for a thousand 12-digit decimal numbers. The program and data input-output was performed by switches, electric typewriter and a magnetic tape device. In 1953, this company was an early adopter of the random access memory (RAM), based on the magnetic rings. This memory invention improved the computer performance by 50 times.

The increased memory capacity and the random access to it made it possible to introduce the high-level language programming and algorithmic languages like Fortran. As early as 1952 the Remington company offered the use of a first interpreter for the program with the algebraic entry. A female scientist Grace Hopper has developed the first compiler for the Univac computer. She also participated in the development of the Cobol language. Thus, in the 50-s there were first programming languages, and some of them, such as Algol, Fortran, Cobol become standard languages in the 60-ies.

Despite the high cost of the of computers of the first generation (about one million US dollars), their production became commercially viable. Most computer implementations were found in banks and government offices. There they were used for the statistical data processing and the cash payment account. In these areas, the computer can replace hundreds of employees, many times accelerate the solution of problems and, therefore, its use makes a profit. Therefore, the main application of computers of the first generation was the accumulation and processing of large amounts of structured data that were stored on magnetic tape. In past 50-ies, there was a number of computer companies, which competed in speed and price of computers. Thus, in 1959, the DEC (Digital Equipment Corporation) company released the 18-bit computer PDP-1 with the RAM up to 32 Kwords, which cost only 120 thousand US dollars.

The RAM miniaturization and use of the semiconductor devices instead of lamps (second-generation computers) contributed to increased RAM capacity and performance. The concept of the operating system became possible when the RAM capacity achieved several tens of thousands of words in 60-ies. The operating system (OS) provides the users with new capabilities such as virtual memory and time scharing. The study of OS in the early 60-ies in Dartmund College, held by General Electric, has pushed to the processor time scharing. The time-sharing system made it possible for multiple users to manage a single computer at a time. This made it possible to significantly reduce the rent for the use of a computer.

The software in those days was the expensive and necessary complement to each computer. And this was the reason for the disappearance of software incompatible computers in the market. For example, in 1960, the IBM computer Stretch was the high technical achievement, in which the pipelined instruction processing and preliminary analysis of the instruction

queue were introduced. But this and other models of computers have lost the commercial success due to software incompatibilities and other difficulties of working with them.

Therefore, IBM (International Business Machines) in 1961 committed itself to create a unified lineup of computers. Computers need to be really versatile — they are used both for commercial and scientific works, are coupled with a variety of peripheral devices via standard interfaces. In the development of new computers, the leading roles were played by Jim Amdahl and Gerrit Blau, who introduced the name of the computer architecture. The basic idea of the architecture concept was the fact that all members of a computer lineup have to be characterized by the same architectural features: the same instruction set, addressing modes, etc. Due to this, they can implement the same program with the equal results for the original data. To implement the same system of complex instructions in the variety of processors, the microprogramming was introduced in all the computer models.

Simultaneously, six models of computers with an IBM-360 architecture were designed, which were differed among themselves by the cost and speed. Up to 47 different peripheral devices in any configuration could be installed in the sold computers. The cache RAM was first introduced in the 195 model of the system in 1969. In 1970, the IBM-360 architecture was replaced by the IBM-370 architecture. The inner architecture of the IBM-370 processors clearly show the details of modern superscalar microprocessors. So, the IBM-360/370 architecture became the most popular architecture in 60-, and 70-ies.

The IBM-360 computer contained several cabinets and was installed in a separate room. The central processing unit (CPU) was housed in the main frame of the main cabinet of the computer. Therefore, a large computer was usually called (and still is called) as a mainframe. In our country, counterparts

computers with this architecture were known as the EC-1010,..., EC-1065 models. The next generations of large IBM computers were designed on the microprocessors, which instruction sets had inherited the IBM-360/370 instructions to meet the portability requirements of the software.

The computer history shows that successful architectures are not those architectures that incorporate advanced scientific ideas, but those that are supported by the best sales management and market conditions. For example, the company Burroughs had introduced in their computers many technical innovations, but they did not contribute to its success. So, in its computer B5000, the multiprogram operating system, and the virtual memory were used at the first time, which preceded their widespread adoption in the decade. But the computer performance in 1963 did not satisfy the users. In the most perfect machine of this brand B6500, sample 1966, the operating system provided the multiprogramming, parallel processing, time-sharing operation, and even the implementation of programs with different instruction sets. But these achievements did not save the company from the downfall.

Another example is a company Control Data Corporation (CDC). Seymour Cray, its founder, in 1963, has developed the CDC-6600 model consisting of multiple ALUs, and peripheral processors. He has found many signs, which were introduced later in the RISC-processors, such as, simple instructions; instructions working with the registers are separated from other instructions; register instructions have three address fields; instruction format is simple and has the equal field widths; pipelined instruction implementation; multiple hardware resources are redistributed between the instruction streams, as in the modern superscalar processors; register renaming technique.

As a result, the speed of the CDC computer was three times the speed of the Stretch computer, but it has not received the popularity. Ideas, tested in

it, have been introduced in the first Cray supercomputer Cray1, which entered service in 1976.

In 1965, the company DEC pushed the 12-bit PDP-8 computer, which gave the impetus for the development of minicomputers. It was the first computer, which was sold for less than 20 thousand USD. The presentation of the DEC PDP-11 architecture in 1970 was the most significant. A minicomputer architecture is characterized in that, that all of its processors and peripherals are connected by the asynchronous bidirectional common bus. This allows the computer to form a configuration in a modular fashion.

A few years later this architecture was repeated in the USSR in the CM-4, M-4030, Саратов computers, and later — in microcomputers Электроника-60, and ДВК.

The PDP-11 architecture was perfect one, so it has become very popular among the programmers. This architecture is repeated in minicomputers, and workstations of the VAX architecture after its upgrade. On the VAX architecture, and its successor Alpha-RISC the DEC company has broken its story because it could not compete with Intel architectures in the market, in 90-ies.

In 1968, the Nova architecture of the Data General company also was included in the minicomputer history. This model was most popular in the world among the minicomputers for its price of only eight thousand US dollars. This architecture is distinguished by the built-in read-only memory (ROM), which stored the software. Then this architectural feature was integrated in all microcomputers.

In the late 60-ies, in Kiev Institute of Cybernetics the minicomputer МИР-1 has been developed and implemented in the production. It is distinguished by the fact that his high-level language interpreter was implemented in the form of the microprogram firmware. This computer was

respected among engineers, researchers, thanks to user-friendly interface. His successor МИР-2 had a console with display and stylus, and was considered as an engineer personal computer (PC).

The quick growth of the calculator market in the late 60-ies was the direct cause of the microprocessor invention. It is believed that the appearance in 1971 of the I4004 chip, developed by the Intel engineers, was the beginning of a microprocessor era. The chief designer of this microprocessor, M. E. Hoff has created it after he has been inspired by the PDP-8 architecture. The emergence of the microprocessors became possible, clear and inevitable when the number of transistors on a chip became more than a few thousand.

The architectural features of the microprocessors were constantly expanded with increasing the number of transistors on a chip. Back in the mid-1960s, Gordon Moore, one of Intel's founders, analyzing different successful chip manufacturers, noticed that about every two years the number of transistors doubles in the chips. This law, named after the Moore, have adopted by the Intel economists and they demanded that the designers of new chips perform it, because it was the key to business success in the field of microelectronics. As a result, this law is strictly carried out for decades and is still in use.

Therefore, the microprocessor architectures were constantly growing with the grow of the number of transistors in a chip. Thus, the I8080 microprocessor had ca. 4500 transistors and provided a single program execution. The I8086 microprocessor with 29 thousand transistors had several programs running under the operating system. 275 thousand of transistors in the I80386 microprocessor provided already the multitasking and the virtual memory.

According to the Moore's law, the memory chips also have developed. The advent of the cheap dynamic memory chips in the early 1980s allowed to design small-sized memory with the capacity of up to 1 MB, and thus ensured the mass production of personal computers. After the RAM capacity reached the values 4-10 MB, it has become possible to run the operating system with a graphical interface, such as Windows-3. When this capacity reached approximately 100 MB, it was possible to process the multimedia data in computers.

New architectural features in the microprocessors repeated the same features in the mainframes of the 1960s - 1980s, while the number of transistors on a chip did not become greater than the total number of transistors in the mainframe chips. This limit is equal to about 1 mln. transistors and has been crossed in the early 90-ies.

At present, several billion of transistors are available for developers of new microprocessors. All the architectural ideas, which were embedded in the old computers, have been exhausted. Moreover, miniaturization is approaching the limit of using the silicon technology. For the constant growth of the computer speed the new architectural ideas are of demand.

Based on the review of the computer and its architecture history we can formulate the following conclusions.

- The advent of computers with the same architecture occurred due to the high programmer labor intensity and the need for software portability between different computers.
- Modern architectures, to a large extent are formed as a set of reinterpreted architectural features which were invented before.
- Most of the architectural features were invented at a time when the hardware resources were severely limited.

- The purpose of the computer architecture, like military tasks, scientific calculations, economic calculations, management, affects it.
- The development of computers and, consequently, their architecture is greatly influenced by the commercial benefits and militaristic needs.
- Miniaturization and new inventions allowed to increase the memory amount. When a certain memory limit of the computer was achieved then the architecture, programming techniques, and operating system were fundamentally changed.
- Not only the potential effectiveness, speed, size, etc. of a new architecture, but the commercial management, competition, and market conditions affect the computer architecture success.
- The high cost of hardware and time resources, a small amount of RAM of computers in the early 60-s led to introduce a system of time-sharing, multi-tasking, memory protection, virtual memory, cache memory, remote access to the processor.
- The architecture represents the main knowledge about the computer potential properties, which serve as a standard both for computer designer and for programmers.

1.1.2 Basic definitions concerning the architectures

To understand the structure of the existing computers, or successfully complete the development of new computers, to communicate with experts and to understand the relevant technical literature, it is necessary to know the number of definitions, axioms and principles of computer science. Some of them are listed below.

State of the computer is a state St_i of all its memory cells in discrete time moment i of its operation. For example, the microprocessor state is determined by the content of its instruction, data registers, program counter,

flag triggers, interface buffers, as well as all the RAM cells. When we deal with the hand made model on the sheet of paper, then its empty squares can serve as the memory cells, in which the writing is performed by a pen.

The different ***constructive objects*** can be stored in the memory cells. They represent individual bits, characters, words, strings, numbers, as well as more complex objects, such as records, lists, files, arrays, and the like. These constructive objects specify the state of the computer according to the semantic of the computational process implemented in.

The computational process is a succession of states St_i of a computer, beginning by the initial state St_0 , and ending by the resulting state St_E . Moreover, the resulting state can be inaccessible, for example, in a digital signal processor, or in the control processor. In the initial state, the defined memory cells store the initial data, and in the final state the cells store the calculation results. During the computing process, the constructive objects are read from the memory, converted, calculated, forwarded, and stored by means of certain operations, instructions, which are specific to the computer.

The algorithm is the computational process of calculating a specific function F in the computational model, which is described by the strict mathematical concepts.

This definition was firstly formulated by E. L. Post, and A.Turing. In the Turing's definition, the Turing machine is taken. This machine is constructed as the endless tape and the control unit. The writing-reading head can read and write data bits on the tape. The control unit controls the movement, and operation of the head according to the algorithm. The state of this model is determined by the state of the control unit, the position of the head, and the state of the tape, and is called the Turing machine configuration.

The concept of the algorithm exists as an intuitive concept, rather than as a strictly limited definition. This concept implies firstly, that there must be

a specific subject or a processor that is able to read, recognize these objects and correctly perform operations with them according to the algorithm.

Second, certainly the algorithm is designed to efficiently compute some useful function F in order to obtain the correct result for a particular kind of input data. Moreover, the reaching of that result must be guaranteed.

Third, for ease of understanding and implementation of the algorithm, the minor details of the computational process (which usually do not affect the computational model) are not taken into account. A set of different computational models is usually used in a practice. They are distinguished both in the field of application and in the abstraction level. In both situations, the model must be as simple as possible to arrange the computational process in useful manner.

The concept of the constructive objects serves not to consider the minor details of the algorithm as well. The constructive objects usually form a set of abstraction levels. For example, if we consider the array processing then we may not consider the separate data of that arrays, and moreover, we could not take into account the processing of separate bits of that data.

So, the algorithm concept is inseparably connected with the meanings of the computational process and computational model, which deal with the constructive objects (see Fig. 1.1). When we say about some computational process, then we consider that it takes place in a given computational model. When we have the proper computational model, then we may arrange some computational process on it. And in both these situations, we deal with the algorithm, which meaning remains not exact.

Therefore, the definition above is merely an explanation of the algorithm with the goal to represent the architecture concept below. Thus, the algorithm can exist apart from the computational model - a model of the rules list — or be part of the model as a control unit in a Turing machine, or to be

the very model, for example, a finite state machine model or the dataflow graph model. For example, if the executor of the algorithm is a man or a programmed processor, then the algorithm is traditionally referred to us as a list of instructions that should be followed consistently.

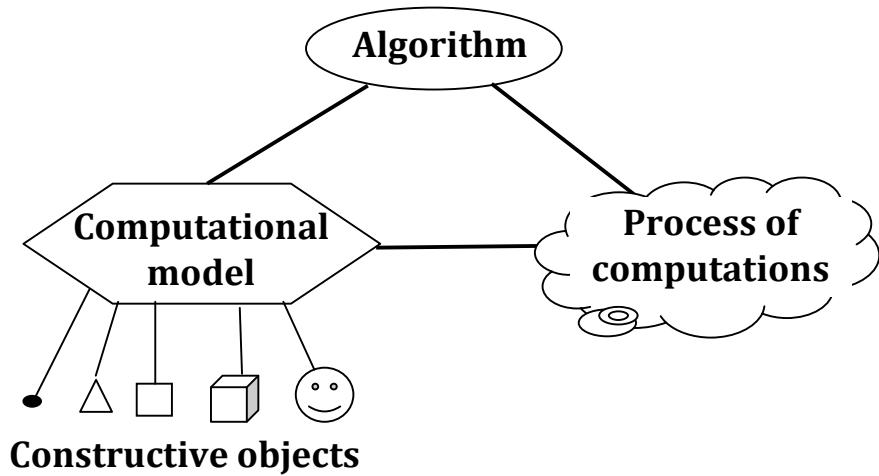


Fig.1.1. Algorithm concept

So, we can give the following definition of the architecture and other definitions connected with.

The architecture is a model of a real computer with a level of details, which is sufficient for its development or programming, and provides the implementation of the corresponding set of algorithms.

The program is an algorithm that is specified on the computer architecture by the means of the algorithmic language or the machine codes depending on the level of the architecture details, or the abstraction level. So, a program, which is written for a particular architecture, should be computed for the same input data with the same intermediate and final results on different computers that have the same architecture.

The architecture from the developer's point of view is a computer model with a level of details that is sufficient for its development and

production. The architecture, as a rule, includes information about the instruction set, address space, address management, system interrupts, memory protection, interfaces, peripherals, that is, all information on the detailed terms of reference for the computer development. Description of this architecture does not include information on the element basis, its speed, computer dimensions, power consumption, security settings, etc., as they do not directly influence the computing process.

The architecture from the programmer's point of view is a computer model with a level of detail which is sufficient for a successful programming certain computational tasks at a particular algorithmic language.

For example, a programmer in Pascal language takes into account the model of computer memory that is addressed by a 32-bit bus, with the ALU, which handles integers or floating point numbers, with the memory at an arbitrary amount of magnetic hard disk, a keyboard and a display, the access to which is provided by the procedures that he/she can find in the library of Pascal procedures.

Each line of the algorithm, described in the assembly language, means a specific machine instruction, performing basic operations with data or controlling the selection of the next instruction. Therefore, the assembly language programmer needs to know the computer architecture perfectly in details up to the separate register and its bit, the interrupt code, the peripheral device address, and the like.

Some of the architectures for the high-level language is defined as an ***interface*** between the language and the system software, which directly implements the compiled program. The operating system or the virtual machine model usually serves as such an interface.

The architectural platform is a common computer architecture, which guarantee to be consistent over the next few years. The architectural

platform can be used in the new computing tools and computers, ensuring the software compatibility, the use of prefabricated components in computers, connection of existing peripherals and devices manufactured by other companies. The most famous architectural platform is i80x86. Its synonym is 32-bit Intel Architecture (IA-32).

The architectural paradigm is a set of common principles and approaches for designing computer architectures. For example, the computer ENIAC, was made using the special processor paradigm, which uses the mapping the algorithm graph into the processor structure with configurable connections. That is, the graph node is associated with an operating unit, i.e. adder or multiplier, and its edge is mapped into the link between the operating units. Most of the computers mentioned above are built on the von Neumann machine paradigm, which will be discussed below.

1.1.3 Principles of computer design

To explain the choice of a particular computer architecture and instruction set, let us consider the principles of their design, which were invented and perfected over the years of the computer history.

The hierarchy principle means that the complex computing function $F = (F_1, F_2, \dots, F_i, \dots)$, that is the composition of functions F_i , is implemented in the calculator $S = (S_1, S_2, \dots, S_i, \dots)$, where each function F_i is performed by the certain unit S_i . In turn, functions F_i are performed by modules of the system at a lower level of the hierarchy.

The principle of hierarchy allows us to understand, design and implement complex technical systems, which are computers. The overview, synthesis (design), or analysis of the system is usually directed upwards (***down-up concept***), i.e. from the elements to complex units or downward

(up-down concept), when the system is first treated at a high level and gradually descends to the level of elements.

This principle is the basis of the design of any complex technical system. To deal with some complex object it must be represented as simple as possible. According to this principle, we can see the different abstraction layers in hardware, data structures, and software. So the computer is built as a set of building blocks, which in turn are implemented on the base of modules, such as ALU, register, counter, and the like. Those, in turn, are performed on the basis of certain logical elements. Similarly, the hierarchical dataset consists of individual words, which, respectively, are sets of bits. The complex program consists of several software modules, which contain routines that are written as a sequence of operators, and in the compiled program each statement is substituted by one or more machine instructions.

The concept of operational and control automata claims that the processor S can be effectively realized, developed and implemented as the composition of operational and control automata. The first one performs a specific set of operations with the data, and the second one controls the execution of these operations according to a given algorithm. For example, a data processing unit, which is discussed below, is composed of an arithmetic logic unit (ALU), data registers, representing an operating automaton, and the control automaton, which supplies control signals to the registers and ALU according to algorithms of the processor instruction performing.

According to the **principle of program control**, the operations of the function F calculation are coded by instructions that are decoded by the control automaton and executed in the operation automaton. The process of calculating the function F is described by the algorithm that is represented as a list of instructions, in which their order is encoded as well. I.e., such a list

represents a program. To change the order of instructions dynamically the control, or branch instructions are used.

Microgram program control principle. It is derived from the previous principle. According to it, the implementation of the complex operation, such as a processor instruction, is decomposed into micro-operations, which are carried out in the serial-parallel order by separate units of the operational automaton. Micro-operations, which are performed simultaneously, are encoded in a single microinstruction, in which also the order of microinstruction execution is encoded. A set of microinstructions is named as a ***firmware***, and is stored in the read-only memory (ROM).

The principle of storing programs and data in a single memory (von Neumann principle) is the program control principle, according to which the instructions are stored in memory and processed in the same way as the data are, the next instruction starts only after the end of the previous instruction, the sequence of instructions is given by the natural order of their location in memory except for the branch instructions. Since the memory usually has a linear addressing, sequential instructions are fetched from it by an instruction counter.

Thanks to this principle, it is relatively easy to define the computational process in computers and carry them to build programs. In our time, the computing process can be arranged by many other methods using the automating programming, designing architectures, which are working on different principles. But this principle remains the most common principle of both programming and designing of most architectures.

But the von Neuman machines have a lot of disadvantages, which limit their use in our time. In the computer, which is running on this principle, the instructions and data types are distinguished implicitly. It is believed that the word, which is fetched to the instruction register, is an instruction but not a

datum, an arithmetic instruction refers correctly to the data of some particular type, and the branch instruction takes a branch to a specific instruction of the program. But it is possible that the arithmetic instruction would fetch the data of improper type, or even not the data at all. When the branch instruction occasionally makes the jump to the data array, the computational process is failed. Such situations occur in most cases of malfunctioning of the computers.

It is also contrary to the requirements of programming languages, according to which the operations refer to the constructive objects (numbers, strings, arrays) explicitly, and the mismatch of operation types and objects is not allowed.

According to this principle, it is impossible to perform simultaneously multiple instructions in the processor. Therefore, due to the strict observance of this principle, it could not increase the computer performance by the parallel implementation of operations, which hampers the development of computer technology. In modern architecture, this principle is violated in hardware to improve performance. Because of this, the reliability of the software, which has developed in accordance with the von Neumann principle, can decline.

The synchronous control principle means that all the computer memory elements, including registers, change their state St_i simultaneously under the influence of a common clock impulse generator, and time is measured at discrete moments. Then the period of clock impulses is determined by the maximum delay of the signal which is propagated from one clocked register to other one.

Every computer, which was designed in the past three decades, can be considered as a set of logical circuits and a set of registers that are clocked by a common clock signal. The process of information processing consists in that

signals, outputted from the registers at the beginning of each cycle, are propagated through logic circuits with inherent delay and converted therein according to logical functions, and are stored in the registers at the end of the clock cycle. This information processing order in the physical layer of design is clear and is supported by all modern computer-aided design (**CAD**) tools for the digital circuits.

At a higher level — the level of parallel processes — due to this principle, the parallel computing processes terminate or exchange the results simultaneously under a single source of control instructions, which is the manager of the operating system. The synchronous control principle manifests itself at the level of procedures as well. For example, the consistent implementation of program streams is planned by quanta of time, or in the case of the multiprocessor system programming.

An asynchronous control principle sometimes is used, when the memory element changes its state only by the special signal of the data availability. This signal is generated by a logic circuitry which processes the data. Therefore, the computer state can be changed faster, and the computer can have higher performance. But it is more difficult to design such a computer, and its speed can be lost when it is connected to the devices controlled synchronously or performing algorithms with the feedback of data. This principle is often used in telecommunication systems, for example, in the computer network protocols.

Due to *the principle of balanced flexibility and specialization*, the optimal computer architecture lays on the boundary of universal and specialized architectures. It is known from the computer design practice, that the general purpose processor is capable performing a large set of algorithms, but its speed, power consumption, size, and cost are two — three orders of magnitude worse than these parameters in the application specific computer

configured to perform a single algorithm. The mainframe computers, which are designed to perform a certain set of sophisticated algorithms for a small period of time, i.e., supercomputers have the largest parameters of price, performance, size, power consumption.

In addition, a plurality of computer algorithms for a new architecture is defined by the number of programs that can be created for it. This number is limited as well as the program design time, or the number of the programmers in the development team is limited.

Therefore, designers of new computational tools are always looking for a compromise between the flexibility and cost of a new computer in its production and use, including the software, trying to use the old, well-behaved architectures. For example, mobile communication devices perform more and more algorithms, but this set of algorithms is limited to, at least, opportunities of these devices to save the energy, their battery charge, and weight. This also corresponds to the following principle.

According to ***the principle of effective multi-functionality***, increasing the functionality of the computer should not lead to a proportional increase in its cost and complexity. In this regard, the von Neumann architecture is perfect, as it ensures the execution of arbitrary algorithm with unchanged hardware costs.

The principle of parallel processing of information. Due to this principle, the independent steps of calculating a function F and management are shared between several operating and control automata (or processors), in order to achieve the high performance and (or) high reliability of computers.

The most apparent and therefore, more common are the sequential algorithms which are described, for example, by sequential programs. Operators of such an algorithm are executed sequentially on a single resource,

such as an ALU in the Neumann processor. But to accelerate the algorithm, using the same element basic, it is possible only having involved the principle of parallel processing.

According to ***the principle of pipelined computing***, the calculation of the function F is divided into successive stages F_1, F_2, \dots, F_K , each of which is calculated in a separate computing resource S_i of the pipelined processor. The operand X_j is sequentially processed while passing through resources S_i , called ***pipeline stages***. Due to the fact, that the flow of the data X_j through the pipeline is a continuous one, the overall throughput is increased up to K times, and up to K operands are considered to be processed in parallel.

The variability principle means the ability to change a set of computer components and connections between them. This principle is manifested in the properties of persistency and scalability of computers, the possibility of adapting the architecture to the specific nature of the task. It ensures the implementation of the multifunctionality principle.

The homogeneity principle is that the structure of the computer or parts of it includes the same elements and the repetitive connections. This reduces the cost to design, manufacture and operation of the computer. The computer memory, consisting of thousands or millions of identical cells, is based on this principle. All multi-processor computers consist of multiple identical processing elements. Moreover, due to the principle of homogeneity, the task scheduling in the multiprocessors is significantly easier than in the heterogeneous computers, consisting of unequal processors.

The principle of unification is a universal principle of creation of technical systems. This principle insists that an object that must interact with other objects, has to be created according to the standard requirements. The object is connected seamlessly unified with other objects even when they were manufactured in different locations and at any time. The idea of a

computer architecture is the result of this principle so that every program runs correctly on any computer of the same architecture.

The main factor in the unification of the computer technology is the interface. **An interface** as a standard means of connection in a computer is used both in busses and in programs. The interface between a man and hardware and software system is of particular importance because it helps to effectively create synergies between the various operators and the computer using the same software.

1.1.4 Architecture hierarchy

Due to the principle of the hierarchy, two or more architecture levels are considered in the modern computers. The computers with as many as five levels exist, as shown in Fig. 1.2.

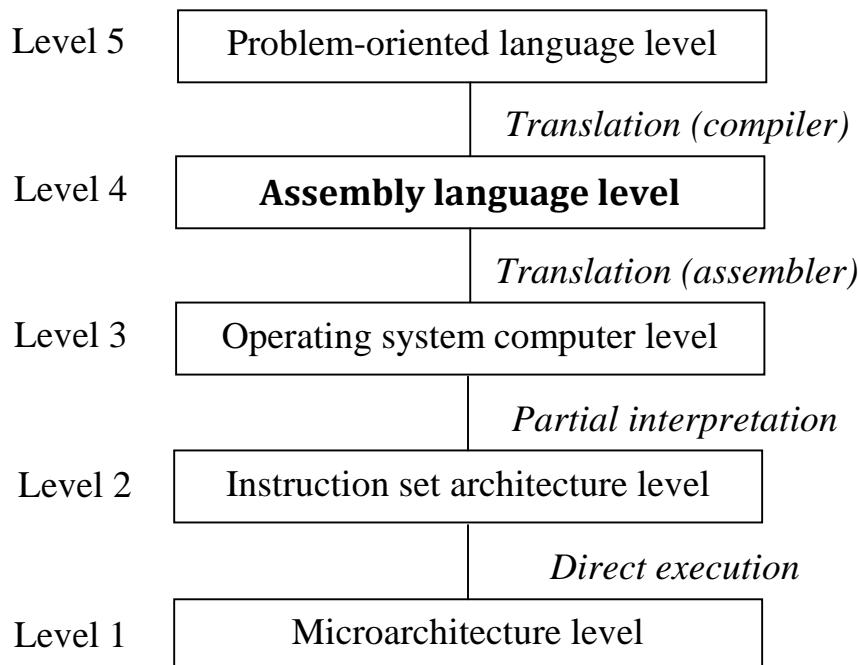


Fig.1.2. Hierarchy levels of the architecture

The ***microarchitecture level*** is considered as the bottom architecture level 1. It describes the details of the hardware architecture implementation. They are how the arithmetic and logic instructions are implemented in ALU, the inner structure of the registered RAM, how the busses connect the units of the processor, and which control signals control the data writing into registers and data output to the common buses, and so on.

Each processor series has its own microarchitecture. Therefore, the microarchitecture can be valued as a kind of the architecture rather relatively, because it depends on the hardware element basis.

On some processors, the operation in the microarchitecture is controlled directly by hardware. On other processors, it is controlled by a microprogram. So, the ***microprogram*** is an interpreter for the processor instructions at the level 2 architecture. It fetches, examines, and executes instructions one by one, using ALU and registered RAM to do so. For example, for an ADD instruction, the instruction would be fetched, its operands located and brought into registers, the sum computed by the ALU, and finally the result routed back to the place it belongs. On a processor with hardwired control, similar steps would take place, but without an explicit stored microprogram to control the interpretation of the level 2 instructions.

The level 2 architecture is called as the ***Instruction Set Architecture*** level (**ISA** level). Every computer manufacturer publishes a manual for each of the computers it sells, entitled “Machine Language Reference Manual,” or something similar. These manuals are really about the ISA level, not the underlying levels. Moreover, the level 1 is usually a secret information. When such a manual describes the processor’s instruction set, it is in fact describing the instructions carried out interpretively by the microprogram or hardware execution circuits.

The next level is usually a hybrid level. Most of the instructions in its language belong also to the ISA level. In addition, there is a set of new instructions, a different memory organization including the memory protection, the ability to run two or more programs concurrently, and various other features.

The new facilities added at level 3 are carried out by an interpreter running at level 2, which, historically, has been called an operating system.

The Operating System (OS) is a set of programs that allows the user to organize effectively the performing a variety of computing tasks in the computer, and take the computer system in working condition. The main tasks of OS, running at level 3, are loading the user program and data in the memory, starting its operation, providing the parallel implementation of several programs in the time-division mode, triggering and working off the exceptions. The special instructions and hardware, which are not accessible directly from the user programs, support these tasks.

Those level 3 instructions that are identical to level 2's are executed directly by the microprogram (or hardwired control), not by the operating system. In other words, some of the level 3 instructions are interpreted by the operating system and some are interpreted directly by the microprogram (or hardwired control). This is what we mean the level 3 by the "hybrid" level.

There is a fundamental break between levels 3 and 4. The lowest three levels are not designed for use by the usual programmer. Instead, they are intended primarily for running the interpreters and translators needed to support the higher levels. These interpreters and translators are written by people called the system programmers who specialize in designing and implementing new virtual machines. Levels 4 and above are intended for the application programmer with a problem to solve.

Another change occurring at level 4 is the method, by which the higher levels are supported. Levels 2 and 3 are always interpreted. Levels 4, 5, and above are usually supported by translation. Yet another difference between levels 1, 2, and 3, on the one hand, and levels 4, 5, and higher, on the other, is the nature of the language provided. The machine languages of levels 1, 2, and 3 are numeric codes. Starting at level 4, the languages contain words and abbreviations meaningful to people.

Level 4, the ***assembly language level***, is really a symbolic form for one of the underlying languages. This level provides a method for programmers to write programs for levels 2, and 3 in a form that is not as unpleasant as the virtual machine languages themselves. Programs in assembly language are first translated to level 2, or 3 language and then interpreted by the appropriate virtual or actual machine. The program that performs the translation is called an ***assembler***.

The architecture level 5 usually consists of languages designed to be used by applications programmers with problems to solve. Such languages are often called ***high-level languages***. A few of the better-known ones are C, C++, C#, Java, Pascal, Python, and PHP. Programs written in these languages are generally translated to level 3 or level 4 by translators known as compilers.

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

The set of data types, operations, and features of each level is called its architecture. The architecture deals with those aspects that are visible to the user of that level of abstraction. Features that the programmer sees, such as

how much memory is available, are part of the architecture. Implementation aspects, such as what kind of technology is used to implement the memory, are not part of the architecture. Therefore, the levels 1, 2, 3 we can mean as the architectures from the designer's point of view, and the levels 4, 5 - as the architecture from the programmer's point of view. In common practice, however, computer architecture and computer organization mean essentially the same thing.

1.1.5 Problems

- 1) For which purpose the OS, and the multiprogramming mode invented were?
- 2) Why the IBM company rent their computers but not sold them in 50-s?
- 3) Estimate the dimensions, speed, and power consumption of a hypotetic mainframe in early 60-s, which implements the architecture of the Pentium microprocessor, containing 3.1 mln. transistors, and operating with the clock frequency of 60 MHz. Note, that in 60-s the main computer module was the printed circuit board of dimensions ca. 150x200x15 containing ca. 40 transistors, operating at 1 MHz with the switching current approx. 10 mA, power voltage 12 V.
- 4) Estimate the volume of the tape storage shelves for 1 GB of data in the mainframe data centre. Note, that the tape of the width 16 mm, and thickness of 20 μm stores 30 bytes per mm.
- 5) RAM on the ferrite cores has the volume of 1 MB. A worker in 60-s can weave 50 cores per minute making the storage, and had a fee 300 USD per month. Estimate the cost of the RAM.
- 6) Each line of the Basic program has its own number. The Basic interpreter was the first product of the Microsoft company. Explain, why.

7) The algorithm is given by a formula $y = a \cdot x^2 + b \cdot x + c$. Explain, what are the constructive objects, computational model, and computational process.

8) The algorithm of the finite state machine is given by the state graph. Explain, what are the constructive objects, computational model, and computational process.

9) The algorithm of the finite state machine is given by the network of this machine. Explain, what are the constructive objects, computational model, and computational process.

10) The algorithm of calculating the sum values is given by the Excel table. Explain, what are the constructive objects, computational model, and computational process.

11) The algorithm is given by the program on the assembly language. Explain, what are the constructive objects, computational model, and computational process.

12) The algorithm is given as the text on the HTML language. Explain, what are the constructive objects, computational model, and computational process.

13) Write the algorithm of the multiplication of two digit decimal numbers by the school method. Explain, what are the constructive objects, computational model, and computational process.

14) Describe the architecture of the decimal handheld calculator from the programmer's point of view.

15) Describe the architecture things of the processor implementing the HTML language (constructive objects, structure, programming language, etc.).

16) Write the program for computation (a sequence of the pushed buttons) of the 4-th order polynomial for the architecture of the handheld calculator (modeled by the Calc program).

- 17) Name 6 most popular architectural platforms.
- 18) Draw the graph of the algorithm calculating the function $z = a \cdot x^2 + b \cdot x + c + d \cdot y^2 + e \cdot y + f \cdot x \cdot y$. How many steps has to perform the von Neuman machine, and the computer, which calculates this function fully parallel? Calculate the speed-up factor.
- 19) Draw the pipeline structure, which calculates the formula in the task 7. Calculate the speed-up factor comparing to the sequential machine, and considering that the data flow through the pipeline is unlimited.
- 20) Propose the instruction set of the architecture, which performs the task 18.

1.2 Element basis of computers

As shown above, the computer architecture has evolved interrelated to the development of their components. The revolutionary impetus to the informatics and computerization of the society have been the emergence and widespread of the technology of the development and manufacturing of large scale integral circuits (ICs). Although, the computer architecture, in principle, has no information on the element base, it is largely determined by the characteristics of ICs, which it uses. Therefore, hereinafter we do a brief look at element basis of modern computers.

1.2.1 Element basis of integrated circuits

ICs are both the most important products of the electronic industry and the basis for all items of computer equipment. The transistor is an elementary component of the IC. The inverter, the simplest element of IC, consists of two **complementary** transistors with the structure of the **metal-oxide-semiconductor (CMOS)** as shown in the functional diagram in Fig. 1.3.

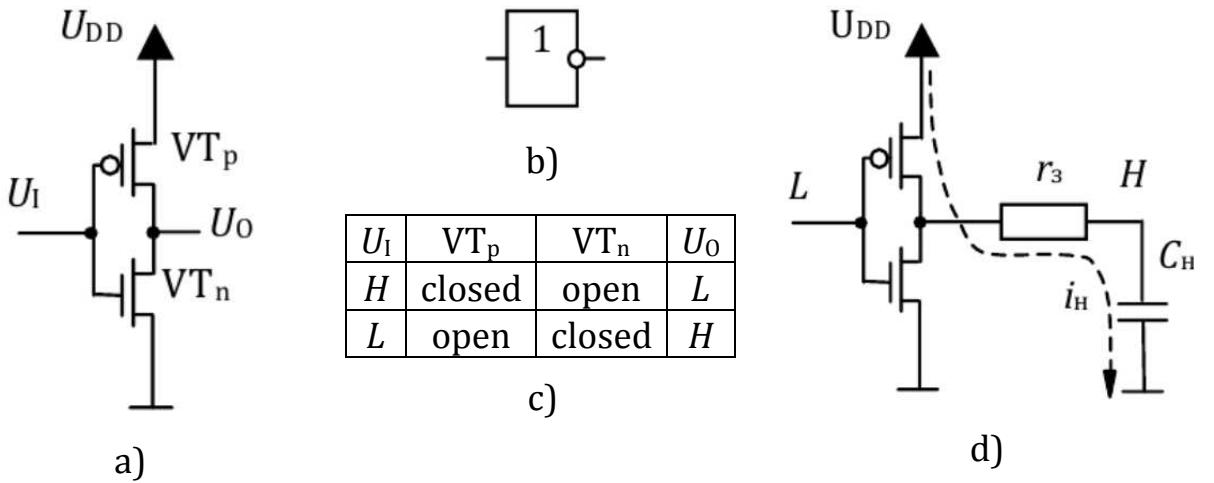


Fig.1.3. Functional diagram of the inverter (a), its symbol (b), its truth table (c), invertor connected to the loading (d)

The p-channel transistor VT_p in the inverter is always in the opposite state, that is in the complementary state with respect to the n-channel transistor VT_n, as shown in the truth table in Fig.1.3, c. In this table, the symbol H (high) is a high signal level, i.e. logic 1, a L (low) is a low level, or logic 0. The CMOS inverter has three components of power consumption: static, dynamic consumption and consumption in the load.

In the static mode, either the top or the bottom transistor of the inverter is closed. Therefore, the CMOS inverter in this mode consumes almost nothing. At the time of the inverter switching, both transistors in a very short period of time can become open and then there is little impetus to the dynamic current consumption.

But most of all the inverter power dissipation is due to the load current, which charges or discharges the load capacitance C_H and flows through the resistance r_w of the connection and resistance of the open transistor. The energy E_s , which is considered to be consumed by the inverter when it is switched, is equal to the energy dissipated in the resistor r_w and open transistor, and is the charging energy of the capacitance C_H , that is, it can be estimated as

$$E_s = \frac{C_h U_{DD}^2}{2}, \quad (1.1)$$

where U_{DD} is the supply voltage, which is approximately equal to the difference of the levels L and H, C_h is the equivalent load capacity, which is equal to the total capacity of the CMOS transistor gates in all circuits, which are connected to the inverter output, plus the communication line capacity.

As the number of switchings per second of the inverter circuits and similar ICs is proportional to the clock frequency f_c , the power consumption of the computer is also proportional to this frequency. Also, according to (1.1), this consumption is proportional to capacitance, which depends not only on the complexity of the circuit but also on the technology of its manufacturing. The value C_h is inversely proportional to the ***design norm*** of IC, which is equal to the minimum width of the constructive elements of IC, for example, the width of the wire, or the transistor gate. Furthermore, the decrease of C_h is proportional to its charge shortening and hence, to the inverter speed and in general, to the IP clock frequency. Therefore, with each new generation of IP technology the energy, produced by the chips, is significantly reduced and their performance is increased.

According to the formula (1.1), it is worth to reduce the power voltage U_{DD} to effectively minimize the power consumption. But when it is decreasing below the limit of 0.9 V, then one of the CMOS transistors of the inverter is not enough closed and proceeds substantial constant current.

A two input logic element (LE), called the ***logical gate*** is formed of four transistors as in Figure 1.4. If both inputs a and b are set high, the transistors VT_{n1} and VT_{n2} are opened, so that the output is set at a low level, while the transistors VT_{p1} and VT_{p2} are closed. In another combination of the input

signals, at least one of transistors VT_{p1} , VT_{p2} is opened and one of VT_{n1} , VT_{n2} is closed, and the output is a high signal level.

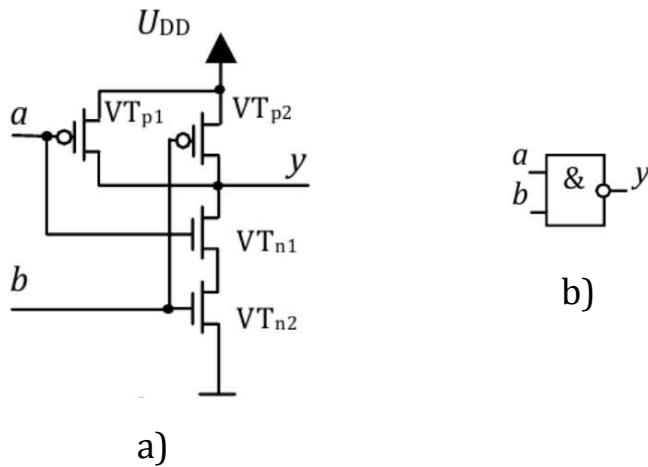


Fig.1.4. Functional network of the AND-NOT gate (a), its symbol (b), its truth table (c)

The basic element of computer memory is a flip-flop, which stores a single bit of information. There are triggers such as latch and synchronous trigger. The **latch** is a trigger, which has two main modes: transparent mode, in which the trigger output repeats the information at its input, and storing mode when it stores a bit in the internal bistable circuit with the state $Q = 0$, or $Q = 1$. The latch mode is changed by the level of the input data or clock signal. Since this trigger has the transparent mode, in which the output signal is changed regardless the clock signal, it is often called as the **asynchronous trigger**.

The clock signal is a control signal, which periodically goes from 0 to 1 and then back to 0. The clock signal is usually referred to as C or CLK or CLOCK. Using the clock signal to control the operation of the trigger allows the developer to set a predetermined moment of time, at which data can be stored in the triggers of the device.

Synchronized D-latch in the IC is created by the network as in Figure 1.5. Its inverters D1 and D2 form a bistable circuit via the feedback. If the

clock signal $C = 1$ opens the transistor VT_{n3} , the input signal D or opens either transistor VT_{n1} or transistor VT_{n2} so that the bistable circuit goes to a state corresponding to the signal D . In this case, a trigger goes to the transparency mode, and arbitrary changes in the signal D are transmitted to the output. After closing the transistor VT_{n3} when $C = 0$, the trigger goes into the storage mode and the bit, stored at the last moment, when $C = 1$, is outputted unchanged.

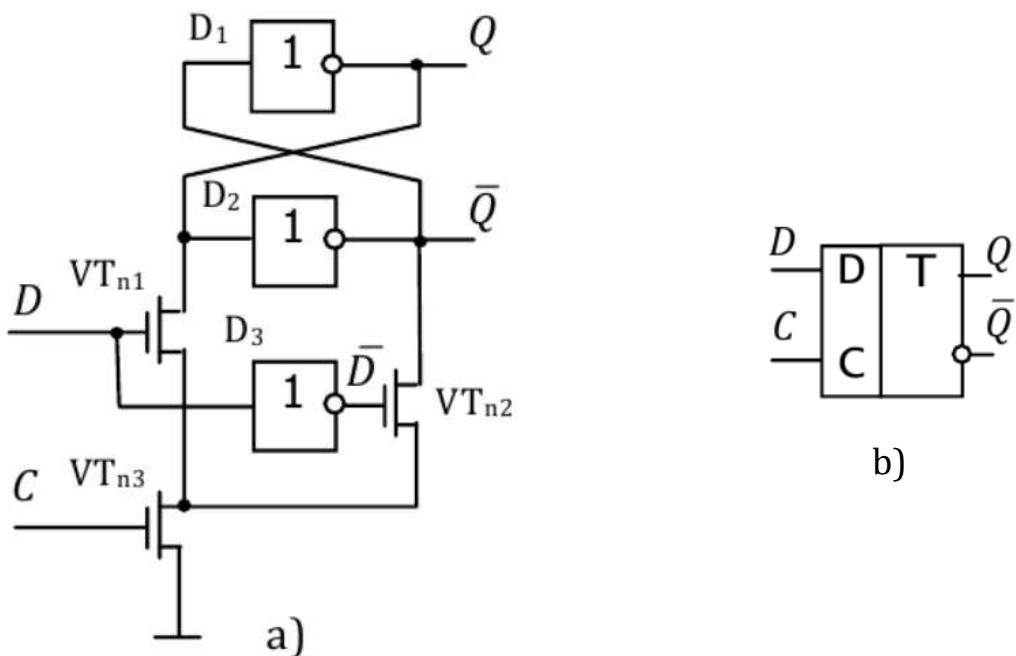


Fig.1.5. Functional network of the D-latch (a), its symbol (b)

The networks, where the latches are used, may have unpredictable behavior. So, the signal D can be distributed through a chain of series-connected latches, when they are in the transparent mode, or in the network the oscillations can be excited when the latches are connected through the feedback. Therefore, to prevent such phenomena, typically only synchronous triggers are used in all modern computers.

In the simplest network of the synchronous trigger, or ***flip-flop***, two simultaneous synchronized latches are connected in series, as shown in Figure 1.6.

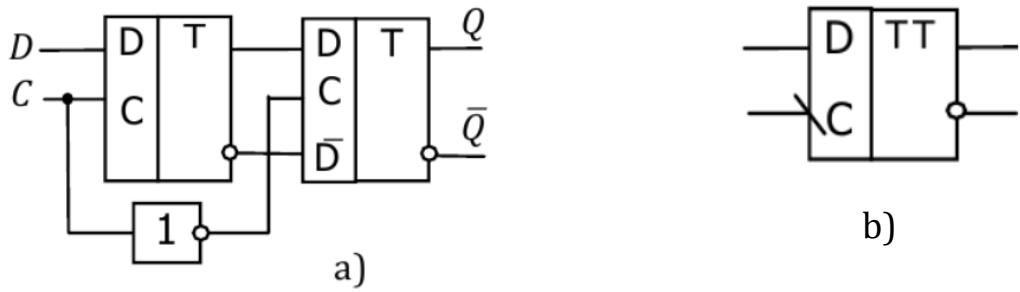


Fig.1.6. Functional network of the synchronous D-flip-flop (a), and its symbol (b)

The first latch acts as a master stage and is responsible for receiving the data. The second latch is a driven stage and is intended for the storage of the data, received from the master stage. In order not to propagate the data simultaneously through two stages in the transparency mode, the first and the second latch are synchronized by the opposite phases of the clock signal C. Consequently, the data from the master latch is rewritten to the slave latch (and becomes visible at the output Q) when the clock signal goes from $C = 1$ to $C = 0$. Because of such properties, this network is considered to be sensitive to the rising clock edge.

Any computer network can be represented as a set of combinational networks and a set of registers, built from synchronous triggers. These sets are linked by the respective communication system, such as in the Figure 1.7.

With the single-cycle synchronization, all the computer registers are implemented as synchronous ones and are clocked by a single clock signal. The clock signal transmission network in the modern chips from its source to all triggers is designed with the utmost care. Due to this, it provides minimal delay skew of the clock edge arrival to each trigger. Therefore, the minimum period $T_{C\min}$ of the clock signal of a chip is estimated as the maximum delay between the output of an arbitrary flip-flop and the input of another one. That

is, the signal between these triggers passes through the critical path in the combinational network (Figure 1.7). Accordingly, the maximum clock frequency of the processor, to a certain extent, determines the computer speed and is defined as $f_{C\max} = 1 / T_{C\min}$.

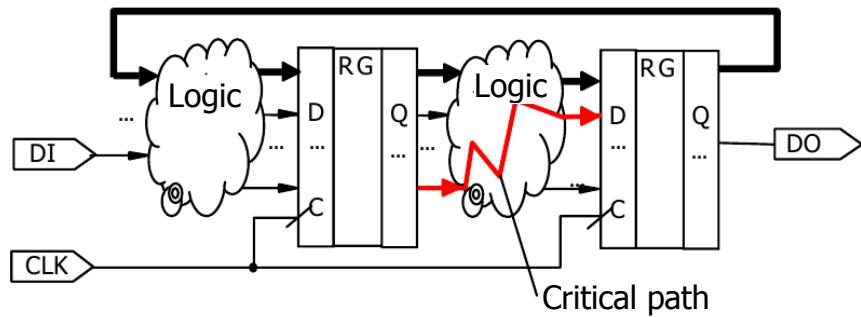


Fig.1.7. Functional network of the arbitrary computer

With the constant decrease of the design rules, the IC technology has moved into a period when the gate delay becomes less than the delay in the connections between gates. In addition, the interconnect wire thinning and the corresponded increase of their resistance, as well as the increasing of the frequency $f_{C\max}$ of up to several gigahertz, lead to substantial signal fading in interconnects, as well as a reduction of the propagation velocity (Fig. 1.3, d). This has led to the fact that on the one hand, to maintain the desired shape of the logic signals it is necessary to insert additional buffers and triggers, on the other hand, the growth of the clock frequency $f_{C\max}$ stopped at the limit values of 5 - 7 GHz.

1.2.2 Integral circuit classification

The complexity of digital ICs is generally expressed in the number of equivalent two-input logic gates. According to this number all ICs are divided into Ics of the small scale of integration, ***Large-Scale Integration (LSI)*** and

Very Large-Scale Integration (VLSI). VLSI circuit consist of more than hundred thousands of gates.

VLSI circuits are divided into microprocessors, microcontrollers, memory ICs, **ASICs** (*Application Specific Integral Circuits*), and **ASSPs** (*Application Specific Standard Products*). Modern VLSI circuit can contain more than ten million gates. In addition, the memory chips have a volume of up to billions of bits, wherein to store a single bit, one to ten transistors are needed.

The microprocessor is the main operating unit of the computer. Its functionality is not defined, not only during their development, production but also during its use. It depends on the user programs and OS (see. Table 1.1).

The microcontroller, as the instruction controlled processor, has the same properties as the microprocessor. But its functionality is usually constant and determines the functionality of the product in which it is embedded. This means that the device performs a single program, which usually remains unchanged throughout the time of use of the product.

The Digital Signal Processor (DSP) belongs to a separate subset of microcontrollers. Its architecture is adapted to the signal processing problems.

The **ASIC** chip has this name because of its functionality, which is put into it during the design and production. Therefore, ASIC implements a one, but a complex function. Examples of ASIC, or custom VLSI are the modem circuit, the drive controller, Ics of the computer chipset, encryption IC. Since the cost of VLSI design is constantly increasing, their development and production can be profitable when the production of the series reaches millions of chips.

The chip of **ASSP** can have the synonym of the multifunctional VLSI. The functionality of ASSP is less than the functionality of a microcontroller, but it is enough to set it up in a variety of different applications. From the perspective of the designer, the ASSP is the multi-purpose ASIC, which has the ability to be adapted to the application. Examples of ASSP are IC of dedicated memory, the microcontroller with a specific set of peripherals, such as the MP3-player, the decoder of the MPEG image.

Table 1.1. Integral circuit functionality

IC type	Design stage	Manufacturing stage	Introducing and use stage
ASIC	known	known	known
ASSP	unknown	known	known
Microcontroller	unknown	unknown	known
Microprocessor	unknown	unknown	unknown

The complex programmable logic device (CPLD) and the field programmable gate array (FPGA) form a special subset of ASSP. FPGA is an array of 2-6-input logic gates, flip-flops and a set of metal strips, which are interconnected by a large array of programmable bridges. These bridges are formed by MOS transistors, which are controlled by special programming triggers. The routes of the element connections are programmed by the change of the electric field in the gates of the MOS bridges, which mean the origin of the FPGA name. The logic gates are implemented on the base of small ROM. Therefore, any boolean function can be programmed in it. Before the FPGA use, the serial programming stream, called the configuration, is automatically loaded into the FPGA from external ROM. This process is called as the FPGA configuring.

Modern FPGAs include memory blocks, multipliers, fast interfaces, processor cores and other specific units. The logical volume of modern FPGAs

reaches two to four tens of millions of equivalent gates. Sometimes FPGA is part of another ASSP or even heterogeneous multicore system.

Designing complex systems on CPLD and FPGA is cost-effectively and quickly executed. So now the FPGA is often viewed as an alternative to ASIC, especially when a planned series of devices is not more than a hundred thousand units. As a result, each year the number of new projects in FPGA and CPLD is growing, and the number of ASIC projects decreases.

The memory ICs are divided into *the Random Access Memory*, or **RAM**, and *the Read-Only Memory (ROM)*.

A system that includes a processor, an application specific processor, RAM, ROM, peripheral devices, etc., which are implemented in a single IC, is usually called as the **System On the Chip (SOC)**.

1.2.3 Problems

1) Draw the network of the 3-input NAND gate. How many MOS transistors are used? Why to make the 8-input NAND gate by the similar scheme is not possible?

2) Draw the network of the 3-input NOR gate. How many MOS transistors are used? Why to make the 8-input NOR gate by the similar schema is not possible?

3) Draw the network of the 2-input XOR gate. How many MOS transistors are used?

4) Draw the D-latch network based on the NOR gates. Why the inverters are used in the latches (see Fig. 1.5) but not the NAND, or NOR gates, as in the computers of 70-ies?

5) To measure the gate delay in IC the circular generators are used, which consists of a set of gates, connected to a closed chain. And the odd

number of inverting gates are put in a chain. Explain the principle of such measuring.

6) Let the circular generator, described in the problem 5, consists of 15 inverters. Estimate the inverter delay if the generator frequency is 600 MHz.

7) The design norms are decreased from 45 nm to 22 nm. Estimate, how much the resistance of the 1 mm wire was increased. How much the resistance of the scaled wire bar was increased?

8) To speed up, the processor was overclocked from 2800 MHz to 3200 MHz, and its voltage 1.2 V was increased to 1.32 V. Calculate, how much its speed and power consumption were increased.

9) To save the energy, the processor voltage was decreased from 1.1 V to 0.9 V, but the clock frequency had reduced from 2200 MHz to 1900 MHz. Calculate, how much the processor's speed and power consumption were decreased.

10) How to feed all the triggers of IC to provide the minimum clock skew? Draw the clock signal network for such IC with 30 triggers, when the fanout (load capacity) of the clock buffer is 4 inputs.

11) In the modern IC the signal is faded dramatically at the distance of ca. 0.5 mm. How to provide the signal integrity in the chip of large dimensions?

12) Explain, why FPGA has usually in 3 — 10 times lower clock frequency, than ASIC has.

13) The multiplexers in ICs are usually built on the base of the MOS transistors used as switches. Draw the network of such 2-input multiplexor.

14) Sometimes the trigger cell in IC is built on the base of the distributed capacitance of some bus. Draw the network of such a trigger cell. Explain its properties.

1.3. Von Neumann Computer Architecture

1.3.1 Von Neuman processor structure

A typical software-controlled computer, which operates on the principle of von Neumann, consists of a control unit, datapath, and RAM. The **control unit** provides the fetching and decoding the instructions, and generating the appropriate control signals. In turn, the datapath consists of registered RAM and an **arithmetic and logic unit** (ALU) (see Figure 1.8.). Often the control unit with the datapath is referred to as the processor core or the **central processing unit (CPU)**.

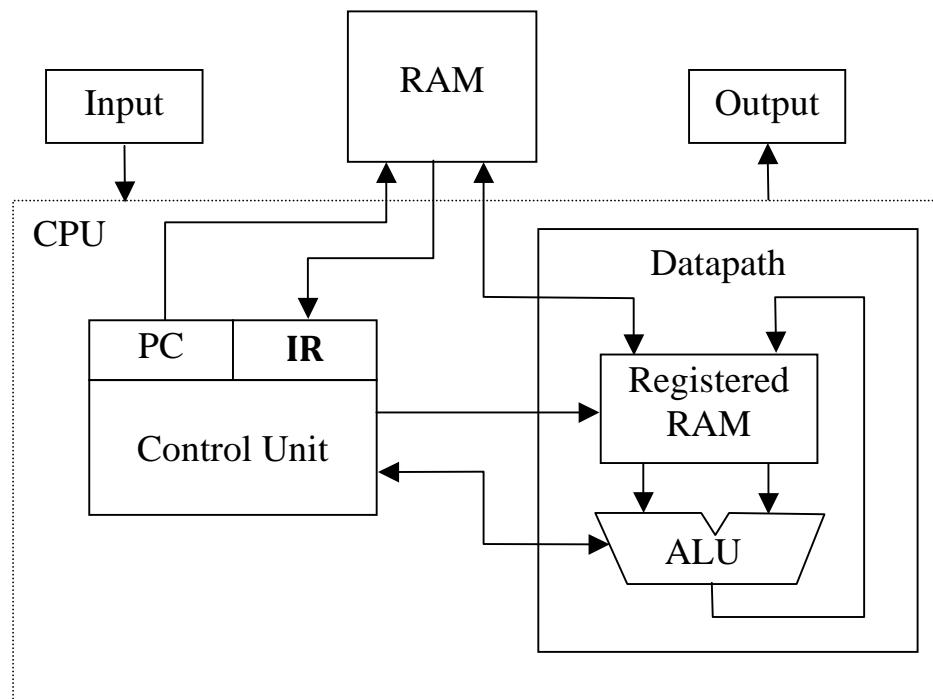


Fig.1.8. Von Neuman processor structure

The **datapath** implements in its ALU the operations, which are coded in the instructions. The **registered RAM** is a set of registers, storing the data, which take part in calculations of an instruction. The **ALU** executes all arithmetic and logic operations, and shift operations, which are coded in the

instructions. The initial data and program are loaded through the Input unit, and the results of computations are outputted through the Output unit. Input and Output units are usually formed as a console, which consists of the keyboard and display. The data output can be performed to the tape recorder, printer, etc. as well.

1.3.2 Instruction execution

During the computations, the von Neumann computer repeats the four-step procedure, executing each instruction.

At ***the first step***, which is called the instruction fetch, the control unit sends to RAM the instruction address from its ***Program Counter*** (PC), and a signal that the next instruction is of demand. PC is named as the ***Instruction Pointer*** (IP) in I80x86 architecture and sometimes called the ***Instruction Address Register*** (IAR). RAM responds that it sends the read instruction to the control unit, which is stored in the ***Instruction Register*** (IR).

At ***the second step*** — the step of decoding — the control unit decides, what kind of instruction is in IR and what action needs to be done to accomplish it. The resulting information is sent to the datapath as the control signals and to RAM as the data addresses.

The third step is the step of the instruction operation execution. The datapath reads the necessary data from the registered RAM or from RAM, processes them in ALU and sends the operation flags (zero result flag etc.) to the control unit.

The fourth step is the step of storing the results in RAM. Besides, the address of the next instruction is determined and is written into PC register. When the instruction flow is straight, then the next address is derived by the addition of the length of the previous instruction to PC. When the instruction is the branch (jump) instruction then the branch code from the IR register

influences the address in PC. For example, when the condition branch instruction is performed, and the condition is true (for instance, the zero flag is 1) then the displacement code from IR is added to PC.

Each instruction is executed with the same sequence of steps. The processor core repeats these four steps during the execution of the program. The only difference is that each instruction controls the data fetching and processing differently.

1.3.3 Von Neuman architecture improvements

The genuine von Neuman computer model has a set of limitations because it was invented without taking into account the programming experience achieved later. The first limitation is the absence of the facilities for the automatic address modification. Thus, for example, in the array access routine, the data read instruction must be modified by other instructions to index through the array. The resulting ***self-modifying code*** is very prone to the programming error.

The architecture does not provide the partitioning the instructions and data because no base addressing was involved. Next, the implementation of the input and output units were not clear. The PC register is the separate register without the architectural access, which makes impossible to do the subroutines. Therefore, just after the expansion of the von Neuman architecture, it was modified by different scientists.

The first incorporation of the index addressing was in the Mark I computer, built in Manchester. The IBM 704, announced in 1954, had already three index registers.

The introducing the standard interfaces provided the attachment to the CPU of a large set of I/O devices in any combination.

But the most prominent improvement was the introduce of the subroutines. This concept was proposed by M. Wilkes, D. Wheeler, and S. Gill. The ***subroutine*** or ***subprogram*** is a sequence of instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed. The subprograms may be defined within programs, or separately in the libraries that can be used by multiple programs.

A subroutine is usually coded so that it can be started (***called***) several times and from several places, including from other subroutines, and then branch back (***return***) to the next instruction after the call instruction, once the subroutine's task is done. A subroutine call not only branches but saves the contents of the PC somewhere. A return retrieves the saved contents of the PC and places it back in the PC, resuming sequential execution by the instruction following the subroutine call.

The advantages of programming with the subroutines include:

- decomposing a complex programming task into simpler steps;
- reducing the duplicate code within a program;
- enabling the reuse of codes across multiple programs;
- dividing a large programming problem among various programmers.

1.3.4 Functions implemented in the processor

All the functions described in the paragraph 1.3.3, are performed in the data processing unit and the control unit. These functions are categorized according to their purpose as the following.

The instruction fetch functions control the flow of instructions in the processor. They provide the next instruction fetch (calculating its address)

depending on condition flags. These flags are the conditions of the branch (jump) instructions, procedure call, etc.

The ***Control Unit*** performs a ***decoding function*** to determine, which function should be calculated on a specific instruction. The ***function of the datapath control*** provides a corresponding sequence of control signals for performing arithmetic and logic operations in ALU, as well as the data transfers in the Datapath.

The ***data addressing functions*** supply the necessary addresses of data in accordance with a given addressing mode. Some of the features serves as a memory management, including support for virtual memory, and management of cache RAM.

The ***process control functions*** provide simultaneous execution several computing processes in a computer, such as, the independent processes in the multiprogramming mode. The interruption is such a function as well.

The ***interruption*** is a function, which stops the normal flow of the instruction execution. There are two situations where it is executed. The first is called a ***trap***, and occurs when the processor determines, that there is some error (overflow, access to the protected area, etc.) during the instruction execution. The second one, called the ***interrupt*** itself, occurs when an external device or process sends a signal, that an event requires a special treatment.

1.3.5 Architecture classification

Von Neumann architectures can be classified into a number of features. These include:

- Instruction set;
- Data formats;
- Program and data memory spaces;
- Timing diagram of the instruction execution;

- CPU structure;
- CPU interfaces;
- Interrupt system;
- Memory protection mechanism;
- A set of peripherals;
- Software (firmware, system software, applications) and others.

1.3.6 Problems

- 1) Why are both the data and instructions stored in the same RAM in the von Neuman architecture?
- 2) For which purposes is the registered RAM used? Is it required for the von Neuman architecture?
- 3) Why is the bus, connecting the RAM and CPU in the von Neuman architecture, often named as the “bottleneck” of this architecture? Propose three methods to expand this “bottleneck”.
- 4) For which purposes is the PC register used? Why is this register named as a counter? Which numbers are added to its content during the address modifications?
- 5) How is the content of the PC register exchanged during the jump instruction implementation?
- 6) Why can the next instruction in the von Neuman processor start only after the previous instruction finishing?
- 7) Propose the computer architecture, in which the PC register is not a counter.
- 8) For which purposes is the interrupt mechanism used in the architecture? Is it possible to do without the interrupt system, and why?
- 9) Why is a kind of interrupts called as a trap? What traps this feature?

- 10) Why is the instruction set the main distinguishing feature of the computer architecture?
- 11) Which of the instruction executing steps is the longest and why?
- 12) For which purposes does ALU generate the flags?
- 13) What role plays the microprogramming in the von Neuman architecture?
- 14) Explain the situation, which occurs, when PC points not to the instruction but to the data in the RAM.
- 15) Why is CPU usually separated from the RAM in the processor structure?
- 16) Take 10 examples of the input and output units.
- 17) Why is the PC bit width usually less than the IR bit width?
- 18) Can the von Neuman architecture do without the jump type instructions?
- 19) What is the self-modified program, and why is it bad?
- 20) What is the similarity and difference between the subroutine call and interrupt?
- 21) What has to be stored and then renewed during the subroutine call and return, and why?
- 22) Why the subroutine concept plays the prominent role in the computer architectures?
- 23) Why is it important, that the PC content is able to be exchanged with the data in the RAM?

1.4. Instruction set of the computer

1.4.1 Instruction structure

The processor instruction word can be divided into the operation code (opcode) field and address part, as in Fig. 1.9.

Opcode	Address part		
	Field 1	Field 2	Field 3

Fig. 1.9. Instruction word structure

1.4.2 Instruction coding

In the opcode field the instruction function is coded, which is performed by CPU. Obviously, the bit width of this field determines the maximum number of different instructions in the instruction set. This bit width may be constant or variable, and is selected for optimizing both the instruction length and the number of instructions.

The simplest approach for coding is the use of the constant opcode bit width. Such approach is usually used for the RISC instruction coding, which is discussed later. But when the instruction set is designed then it becomes obvious that some instructions are used in the programs much frequently than others. Therefore, it is useful to make both their Opcode and whole instruction bit width much shorter than for the rest of instructions.

For example, consider the 4-bit opcode for the instruction coding. Then 16 variants of such opcode are possible. Let 15 of them code the most useful instructions. Then the last code means, that it codes a subset of the rest instructions. The additional bits to the opcode are coding these instructions.

Another example. Codes 0000, ..., 1101 encode the 14 most useful instructions, and the codes 1110 and 1111 have the 4- and 8-bit suffixes, that represent sixteen codes from 11100000 to 11101111 and 256 codes from 111100000000 to 111111111111.

The instruction set of the Intel processors, which is built according to this principle, has the Opcode length of 5 to 19 bits. Because of this, as well as the fact that the instruction can perform complex actions, such an instruction set is called as a complex instruction set. Such an instruction set is used in the CISC-processors (***Complex Instruction Set Computer* — CISC**).

On the other hand, the choice of the instruction set of the low complexity instructions is common to the RISC processors (***Reduced Instruction Set Computer* — RISC**). In such an instruction set, both the opcode and the whole instruction should be of the constant length and a set of instruction formats should be small. The regularity of these instructions simplifies the instruction decoding and parallelization of their implementation.

1.4.3 Instruction address fields

In the address part of the instruction, the information about the addresses of operands and results, as well as the location of the next instruction are placed. There are 4, 3, 2, 1, and 0-address instructions. The 0-address instruction assumes that the address is given indirectly. For example, the data is stored in a predetermined register, such as an accumulator. Then this register address is encoded in the opcode. The address can be implicitly specified by a stack pointer, or it is calculated using the instruction counter contents as well. The 0-address instruction of another type contains a so-called immediate operand, that is a constant. This means that it does not contain the address at all.

The 1-address instruction assumes that one operand or result is stored in the specified address, and the other one is obtained as well as in the 0-address instruction. The 4-address instruction has four address fields. Then in addition to three operand addresses, it can contain the address of the next instruction.

The number of address fields in the instruction depends on the CPU address space. It is optimized during the design of the instruction set. On the one hand, one instruction with many address fields replaces a series of short instructions, providing high performance, and short programs. For example, a jump instruction was used in the computers of first generations, which provided for the branch to three addresses, depending on whether the result is greater, equal or less than zero.

On the other hand, the address space of the modern computers requires the address word length of 4 and even 8 bytes. Therefore, the instructions with large address fields occupy a significant amount of memory and have the increased fetching time. As a result, the choice of the instruction format is a complex optimization problem.

One of the ideas of creating the RISC processor was the idea to divide the instructions into two categories. The first kind instruction works only with the datapath, fetching the data from the registered RAM using 3 or 4 register address fields. These fields length is usually equal to 4 or 5 bits and they address the 16 or 32 registers respectively. The second kind instruction provides the data exchange between registers and main RAM. It has the address fields of the data register and the register, which stores the address of the data in RAM. It implements the indirect addressing.

The address field can represent either a direct address code, or code that plays a significant role in calculating such an address, for example, the

indirect address. More information about the address fields is presented below when considering the types of addressing modes in computers.

The address field does not necessarily indicate the direct address of the operand. In some instructions the address field contains the information about a set of RAM cells, or the range of addresses. For example, the LDM instruction in the ARM architecture loads up to 16 registers from the main RAM. Each of 16 registers in is indicated in it by the position of a one in the 16-bit address field.

1.4.4 Instruction set selection

The processor architecture design is usually started by the instruction set selection. A good instruction set provides both the high processor performance and minimized program length. However the processor performance is prevalent.

The effectiveness of the instruction set is verified by simulation of the processor with this set and comparing the simulation results with the parameters of other processors (the results of the benchmark execution, the program length after compilation, etc.). Usually the standard benchmark is used, which is a set of programs written in C, and representing the common problems in different fields, for example, text processor, equation solver, data base handler, etc.

1.4.5 Problems

- 1) Propose the instruction set of a simple processor, which consists of only null address instructions.
- 2) Propose the instruction set of a simple processor without the program counter.

3) Propose the instruction set of a simple processor, which data memory is a stack.

4) Consider an instruction set of 60 instructions, 10 of them are used very often, 20 are used less frequently and 30 are rarely. Propose their opcode set.

5) The conditions are the same as in the problem 4, but the instructions are divided in the groups of 7, 14 and 37 instructions, respectively.

6) The RISC processor has 32 general purpose registers and the instruction set with the 32-bit instructions of two types. The instructions of the first type are the one address instructions with the 28 bit address field. The instructions of the second type are the four address instructions addressing the registered RAM. How many instructions can contain this instruction set?

7) The conditions are the same as in the problem 6, but there are 128 registers, and the three address instructions.

8) The instruction set has 254 instructions, which are coded by a 8 bit opcode field. The instructions have the variable length from 1 to 6 bytes. The designer has the solution to expand the instruction set by adding the prefix byte. How many instructions can contain the new instruction set?

9) The conditions are the same as in the problem 8, but the instruction set has 250 instructions.

10) The instructions have not the opcode at all. How is arranged the programming of such a processor?

1.5. Data formats and operations

The data are the constructive objects of the computer architecture. They are always given in binary form in a certain format. The question of how to represent the data is complicated by the fact that the computers from different manufacturers often have the same data in different formats. In such computers, different word lengths and special character sets are used. Accordingly, in old computers, different methods of storing integers and floating point numbers are used. This complicated both the software and data compatibility between different machines. But the most of the microprocessor architectures obey the same data representation rules and standard data formats.

1.5.1 Character representation

For years, the same approach was used: a certain number of bits to represent a character was chosen, and a table of compliance of code and characters (letters, numbers, symbols, special characters) was built. For example, the first generation computers have used 6-bit characters to specify capital letters, numbers, operation signs. Now, to code the same symbols as well as the small letters the 7- or 8-bit codes are defined. The 8-bit codes represent both Latin and Cyrillic characters. In the world, the character code table is standardized.

The most common standard of 7-bit codes is **American Standard Code for Information Interchange**, in shorts — **ASCII**. ASCII encodes only 95 carefully selected printable characters, among them 94 glyphs and one space. The rest of the codes represent the “invisible” control characters. For example, the code 0 means Null, and is usually used to represent the empty objects or to construct the strings. The code 13 means the carriage return (CR).

The characters are numbered in the alphabetic order. This fact is usually used to substitute the alphabetic sorting to the sorting of natural numbers. The decimal digits from 0 to 9 are coded by integers from 48 to 57, or by hexadecimal numbers from 30h to 39h. Therefore, to select the decimal digit, the character code is usually ANDed with the mask 0fh.

The extended ASCII (EASCII or high ASCII) refers to 8-bit encodings that include the standard 7-bit ASCII characters, plus up to 128 additional characters. 8-bit codes may not be adapted to display any alphabet. Therefore, in different countries the, so-called, national ASCII coding tables are used, which are fully supported by letters and distinguished figures in the national alphabet. That is, the first half of the table origin from the ASCII codes, and the second half is coding of the national alphabet. So, the Ukrainian extended ASCII code page has the number 866. Microsoft was introduced the code page 1252 in its Windows operation system.

In order to support such alphabets as Japanese, Chinese, as well as for the international coding of the table, the letters that look the same on all operating systems and Internet browsers, the much larger code table is needed. For these purposes, the 16-bit and even 32-bit **Unicode** encoding is distributed now.

Some architectures support the characters so that they have the special instructions that distinguish the character codings. For example, in the Intel 80x86 architecture, there are instructions that provide arithmetic operations with bytes that represent the decimal digits. Eg. the instructions aaa, aas, aam, aad perform decimal correction after addition, subtraction, multiplication, and division of numbers provided by the ASCII-codes .

1.5.2 String representation

The character strings are represented by a sequence of bytes, one or two bytes per character. The microprocessor instruction set typically supports the string processing, such as moving, comparison, pattern matching, filling by a symbol, input-output of the strings. The maximum length of the string, which can be represented and processed by an instruction, depends on the specific architecture of the processor and can reach 4 billion symbols.

The string moving is usually implemented byte-by-byte. It is necessary to program the string moving carefully, when the source and destination address ranges overlap, as this can destroy the string, overwriting the bytes from one part of the string to another one. The strings have an arbitrary length so that the programming language must provide a way to determine the length of the string. For example, in the C language, the null byte signifies the end of the string.

1.5.3 Representation of integers and operations with them

In the first computers, as well as in the application specific processors, the basic representation of numbers was the binary fraction with the point, which stands between the sign bit and the most significant bit of the number. With such a representation, it is convenient to perform all arithmetic operations, rounding the results. But in that computers, the addresses are represented by integers, which should enable other operations of multiplication and division than fractional numbers. It is inconvenient to handle two sets of operations for the numbers with the related representation. So in modern computers, the fractional numbers and respective operations with them are not used directly.

Now the integers are represented in the computers only as two's complement binary codes. Although there was a time when such numbers

were represented by the decimal or even ternary codes. There have also been attempts to encode integers by the Fibonacci numbers or residue codes.

For the positive numbers, the binary code representation is obvious. Each bit represents a value of 2 to the power of the bit number. According to the convention of the computer scientists, the highest bit in the number writing is on the left. Thus, the number 130 is represented by a byte $1000\ 0010_2 = 82h$ or halfword $0000\ 0000\ 1000\ 0010_2 = 0082h$, where h is the suffix indicating a **hexadecimal number**.

There are several ways to represent negative numbers, but in all modern microprocessors the negative integers are represented as **two's complement numbers**. In this representation, two equal in magnitude and opposite in sign numbers give the number of 2^k as a result of their addition, where k is the number bit width. For example,

$$A = 0101_2 = 5, \text{ and } -A = 1011_2 = -5; A + (-A) = 10000_2 = 2^4.$$

Here $-A$ is a complement of A to 2^4 . At this condition, the highest four-digit number is $0111 = 7$, and the lowest is $1000 = -8$.

Let the two codes are added: $0111 + 0111$. The result should be 14, but we get $1110 = -2$. So, we have an overflowed result in terms of the signed number representation. In most processors such overflow situation is fixed in the overflow flag $V = 1$ (**oV**erflow), as well as in the **Carry** flag $C = 1$. Otherwise, when performing, for example, $0011 - 0111$ we get the right result -1100 , so for him $V = 0, C = 1$. Each computer also has a **Zero** result flag Z, and a **Negative** result flag N.

When multiplying integers, the unsigned multiplication, and signed multiplication are distinguished. For example, in the first and second cases, we have $0001 * 1111 = 00001111$ and $0001 * 1111 = 11111111$, respectively.

Accordingly, the unsigned, and signed multiplication instructions are distinguished. The same features are applied to the division operations. In

many computers, the multiplication and division operations are performed as unsigned. Then the operand signs are taken into account by means of additional calculations, which require an appropriate excess of time.

Similarly, we must distinguish signed, and unsigned comparison operations, because, for example, $1110 > 0111$ (unsigned), and $1110 < 0111$ (signed).

1.5.4 Shift operations

The operation of the shift by k bits can be regarded as a multiplication by the coefficients of 2^k when shifting to the left or by the coefficients of 2^{-k} when shifting to the right.

The arithmetic and logic shifts to the right are distinguished, of signed and unsigned integers, respectively. For example, the following instructions of the I80x86 processor perform the following various shifts:

```
mov DX, 8C00h; negative number is loaded to DX register  
sar DX, 4; signed (arithmetic) shift right to 4 bits, result is F8C0h  
shr DX, 4; unsigned (logic) shift right to 4 bits, result is 0F8Ch  
shl DX, 4; shift left to 4 bits, result is F8C0h
```

Also, the cyclic shift operation is used, when the bits shifted out are shifted in the same word but from the opposite side.

The last shifted out digit is stored in a C flag for the further use as a condition in a conditional branch instruction. In some architectures like I80x86, several shifted out bits can be stored in a separate register, as in the instruction:

```
shld AX, DX, 4; 4 bits, shifted out left from DX, are shifted in AX
```

In the simple microprocessors, the shift operation to k bits is used to normalize the mantissa when performing the floating point calculations. To speed up these calculations, the instruction set has a special instruction of

finding the left leading bit. This instruction returns the number of k bits to shift the operand for its normalization.

1.5.5 Sign expansion operation

The operations of the format conversion of short numbers to long numbers or calculations with the short numbers are often used. If the unsigned number is carried out, then zeroes are added to it on the left. And if the signed integer is an operand, then its sign bit is expanded, and this operation is called a *sign expansion*. Most processors have special instructions for this purpose. If they are absent, it is necessary to supplement the number by zeros, shift it to the left to make the appropriate number of digits, and then shift to the right, completing the signed shift.

For example, consider the stored in memory byte: AAh, which means the signed number -86. Then the byte is loaded in the half-word register: 00AAh. Then the half-word is shifted to the left by 8 bits: AA00h. And finally, it is shifted to the right by 8 bits: FFAAh, that means the same number -86.

1.5.6 Storing numbers in the memory

Usually, the multi-byte words are computed and stored in the registers of the processor core. There are two orders of their storing in the external memory. The first of them is named as *big-endian*, ie. the lower bytes are stored at the upper addresses, the second is *little-endian*, ie. these bytes are stored at the lower addresses. Examples of storing the four byte words in these orders are shown in Fig. 1.10.

The multi-byte word is addressed, as a rule, by the address of its lower byte. So, in the architecture "big-endian", such as Apple, MIPS, the higher byte is read as the first one, and in the architecture "little-endian," the processor reads the lower byte first, as in the i80x86 architecture.

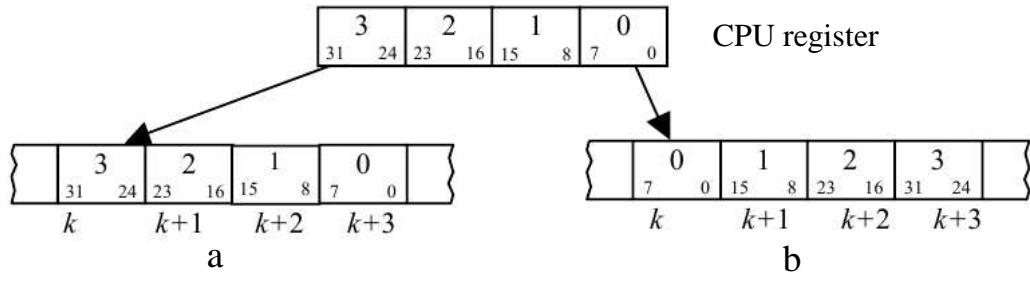


Fig.1.10. Storing a word into the memory in the big-endian order (a) and in the little-endian order(b), beginning at the address k

It should be noted that in order to provide the rapid access to the word of the length 2^k , it is written to the address in the whose code the k least significant bits are zero. In this case, we say that the word is ***aligned on a word boundary***. Thus, typically 2^k bytes can be read or written simultaneously. Otherwise, you need to read the word byte by byte or in its parts with the subsequent connection these parts in the whole word.

To ensure the architectures interoperability, the modern microprocessor architectures usually permit the proper read-write order to install before the starting the operating system.

1.5.7 Address calculations

The arithmetic of unsigned integers is usually used to calculate the data and instruction addresses. By this, a semantics acts, when the address word overflow is not taken into account. That is, the calculations are performed modulo 2^k , where k is the address word length.

Consider the base address is added to the displacement D . Then if the displacement is higher than 2^{k-1} , then it operates as the code $(2^k - D)$ is added

to the base address. For example, if $k = 16$, $D = 0FFFFh = 10000h - 1$, then $3456h + D = 3455h$, ie. it looks like a one is subtracted from the address.

For this reason, it is important to use the sign expansion operation, if the address displacement is a short negative number.

1.5.8 Decimal number representation

The decimal number can be represented as a string of letters. To do with such a number, it has to be converted to the integer before the calculations and to be converted back after the calculations.

A decimal digit is represented by a byte in the ASCII coding. Such a byte has the binary code of the decimal (**BCD**) digit in its lower nibble, and the code 3 in the higher nibble. For example, the number 25 is encoded as 3235h. Such a number is named as the **unpacked BCD** code. It is distinguished from the **packed BCD** code in that, that the bytes in the latter contain the couples of BCD digits.

The I80x86 processor architecture has instructions for processing the unpacked BCD numbers that are represented by such codes. Some computer architectures, such as IBM370, have the instructions to operate with the packed BCD numbers. All processors can operate with packed decimal codes using the respective subprograms.

1.5.9 Floating point data

The floating point data are widely used in scientific calculations since they provide the small error calculations without caring about the data scaling. The number in the floating point format consists of the sign bit S , exponent field E , and mantissa or fraction field M , so the number is equal to $(-1)^S \cdot K^E \cdot M$, where K is the exponent basis. In different computer architectures, the basis K is equal to 2, 4, 8, 10, or 16.

Prior to the application of the standardized floating point data representation, the calculations in computers were imperfect. In some architectures, if two integers are divided entirely, the corresponding floating-point numbers are not divided entirely. In other architectures, the result of $A + A$ is not equal to $2.0 \cdot A$. Also, the equation $(A - B) = -(B - A)$ is not true in some processors. A small non-zero divisor X in the operation A / X could cause the interruption of "divide by zero" type.

The IEEE-754 standard for binary floating-point arithmetic was published in 1985. It provides two basic formats: ***single*** with the 32 bit width, and ***double*** with the 64 bit width. Single and double formats are shown in Fig. 1.11. In the single format, the minimum exponent is 0000 0001, which corresponds to 2^{-126} , the maximum exponent is equal to 1111 1110 and corresponds to 2^{127} . The 24-bit fraction is coded by the 23-bit field because the most significant bit of the normalized fraction is always a one and is not stored.

S 31	E 30	M 22	0
---------	---------	---------	---

S 63	E 62	M 52	0
---------	---------	---------	---

Fig.1.11. Single and double formats of the floating point numbers

For example, code 1 1000 0000 100 ... 00 has a minus sign, the exponent field is E = 128, that is, the exponent is $2^{128-127} = 2$ and fraction is $1.10 \dots 0 = 1.5_{10}$, that is, the decimal number is -3.0 .

The standard provides four rounding modes for the result:

Round to nearest. The processor chooses the nearer of the two possible outputs. If the correct answer is exactly halfway between the two, the system chooses the output where the least significant bit of the fraction is zero. This behavior (round-to-even) prevents various undesirable effects. This is the default mode when an application starts up.

Round up, or round toward $+\infty$. The processor chooses the larger of the two possible outputs (that is, the one further from zero if they are positive, and the one closer to zero if they are negative).

Round down, or round toward $-\infty$. The processor chooses the smaller of the two possible outputs.

Round toward zero, or chop, or truncate. The processor chooses the output that is closer to zero, in all cases.

The code of the rounding mode is written in the control register of the floating point coprocessor, such as a register FPCR of the coprocessor i80x87.

The standard also offers formats with extended accuracy. In the single extended floating point format, the fraction has a bit width 32, and the exponent has a bit width 11. These figures in the double extended format are 64 and 15, respectively. These formats are used in the hardware accelerators which calculate the mathematical functions to keep all the exact digits of their results with a single or double precision.

At present, a new version of the standard IEEE 754-2008 is propagated, which proposes the use of quadruple precision binary floating-point format: binary128. Such a format has 15 bits of the exponent and 113 bits of the fraction. Even more exact format binary256 is possible.

Traditionally, the floating-point ALU prevents the overflow situation causing an interrupt. The standard offers to replace the overflowed data automatically to the special code $E = 1111\ 1111$ and $M = 00\ldots 0$, which means the positive or negative infinity ($+ \text{INF}, - \text{INF}$). Then, when such a value occurs

in subsequent calculations, it behaves in a logical manner, for example:
 $1.0 + \infty = +\infty$; $1.0 / \infty = 0.0$.

But in some cases, the result of the operations with the INF numbers may have not any sense. This, for example, the result of operations: $\infty - \infty$; $0.0 / 0.0$. In such cases, a transition to the exception or replacement of the result in the special value takes place. This special value is called "not a number", shortly, **NaN** (Not a Number), which is coded as $E = 1111\ 1111$, $M \neq 00\dots0$.

The standard requires the use of two types of NaNs — signaling and quiet NaN. If one of the operands of the operation is a **signaling NaN**, then ALU must cause a trap. When some data in the user program are not initialized, then the operation system usually initializes them as the signaling NaNs. Then the calculations do not make a sense and do not begin.

If one or both operands are **quiet NaNs**, then the result is also the quiet NaN, a trap does not occur and the calculations are continued. At the end of these calculations, the situation which causes the NaN numbers can be determined by analyzing the calculation flow.

A special situation occurs when the result of the operation in absolute value is less than the smallest number that can be represented in the floating-point format. This situation is called **underflow** or loss of accuracy. For example, the lowest normalized code is $0\ 00000001\ 0000\dots0$, which is equal to 2^{-126} . If the result of calculations is the value 2^{-128} , then it is represented by the code $0\ 00000000\ 0100\dots0$. The exponent $E = 00000000$ in it means that this number (represented by the fraction bits) has underflow. Thus, the calculations can be carried out more accurately.

For example, when the calculations $x - y + y$ take place, then if $x \approx y$, the result of the step $x - y$ is a zero when the operands are represented in a

conventional floating-point format, and the final result is y . Upon calculation of $x - y$ as an underflow situation, then the result is correct and is equal to x .

1.5.10 Problems

- 1) The word ‘continue’ and the string “continue” in the C program after the compilation have the lengths of 4 bytes and 9 bytes, respectively. Explain this.
- 2) Why are the SMS messages in Cyrillic characters twofold larger than ones in Latin characters?
- 3) Explain the behavior the AAA instruction, which is used after integer addition of the strings, representing the decimal numbers.
- 4) The string “Example” is stored in the memory, beginning at the address X, and it is moved to the address X+3. Show the results of the moving in two situations: when the moving starts at the lowest address and at the highest address. Note, that the rest of the memory is filled with a sign '#'.
- 5) Why could not occur the symbol with the code 00h in the middle of the strings after the C compilation?
- 6) All the modern computers have the 2-s complement integer data representation but not the fraction number representation. Why?
- 7) The first computers had the decimal number data representation. Why were they substituted by the binary computers?
- 8) For which features are the comparison of signed and unsigned data distinguished in the computers?
- 9) Why are needed both the carry and overflow flags in the computers?
- 10) Propose the addition algorithm of the very long integers represented by several words.

11) For which purposes have some instruction sets the multiplication instructions of three types: unsigned by unsigned, signed by unsigned and signed by signed?

12) How to substitute the multiplication to the constant, say, 47, by the addition, subtraction and shifting instructions?

13) How to add the signed byte to the signed word when the instruction set has not the sign expanding instruction?

14) Often the free program memory is filled by the NOP instructions. A program is loaded in the highest addresses of this memory. What does happen when the last instruction of this program finishes its implementation?

15) The jump instruction with the displacement code 20h is placed at the address 0fff0h of the 16-bit architecture. Where is the instruction stored, which is fetched as the next one?

16) It is a method to calculate the sum of a large number of the floating point data. Firstly, the large and small numbers are added separately. Then the sums of large and small numbers are added together. What is the purpose of this method and why does it work?

17) The single precision floating point number looks like the following: 0100 0000 1010 1010 1010 1010 1010 1010. Calculate its decimal representation. Propose the operation, which result is such a number.

18) Some scientific programs fill the initial data arrays by the NaN numbers. Explain, why?

19) Translate the decimal number 0,1 into the floating point format. Estimate the error of the number representation. For which decimal fractions is this error equal to zero?

20) The floating point data have the formats of 4 or 8 byte words. But till 90s the 6 byte floating point data were popular. Why this format of data becomes very rare format.

1.6. Addressing modes

1.6.1 Address space of the CPU

The architecture of each processor is characterized by its address space. The term of the address space means a numerical range of memory addresses that can be accessed by the processor core. The readout of the memory addresses, usually operates in bytes, starting with a null byte. Thus, a processor, that has the k bit address bus, addresses eventually space in the limits from 0 to 2^{k-1} .

Typically, the address space is divided into specific areas, which are used to store data, programs, operating system, input-output data, and others. The division of the address space is called a ***memory map***.

Each processor provides a number of data addressing modes. When a set of addressing modes is selected, the problem of the instruction set effectiveness is solved. Complex addressing mode needs the complex instruction coding and implementation. Often there is a direct relation between the memory addressing modes and the programming language. On the one hand, the existing architecture influences the creation of the appropriate language and compiler. On the other hand, the development of new architectures is connected with features of the programming languages.

1.6.2 Direct (absolute) addressing

Most languages, such as Pascal, C, Fortran, allow the programmer to declare the static data, that is to each operand a single cell in the memory is assigned during the program compilation, which is unexchangeable during the program execution. In this case, the size and address of the datum can be determined before the start of the program execution, that is, at the time of its compilation. This means that the actual (logical) address is known before the

program running. Therefore, many processors have direct addressing, which allows the programmer to put a valid address directly into the instruction.

The direct addressing is not directly implemented in many processors. But then the static address is the address within a specific memory segment, which beginning is set to the base address, ie., the direct address is actually the offset relatively the base address of the segment. Therefore, for example, in the architecture I80x86 the direct addressing is implemented when a 16- or 32-bit offset d is set directly in the instruction, and the effective address (physical address) is the sum of d and of the address code stored in the segment register DS. OS loads the value in the register DS before the program execution, which means the address of the free memory segment, which is selected by OS for this program.

So, the assembly instruction of this architecture `mov AX, data` is loading in the register AX the operand with the symbolic address `data`. Here, the operand reading is done with the direct addressing, and writing is done with ***direct register addressing*** (AX register address is set in the instruction as a code 000_2 , and the segment address is in the register DS). That is, the register addressing is the direct addressing of the registered RAM. It should be noted that the `data` identifier is a ***symbolic address*** that receives a particular numerical value at compile time.

1.6.3 Index addressing

The data array is a special structure that is used in almost all programming languages. Therefore, the modern computer hardware supports the array indexing. The indexing addressing is implemented in hardware as a scalable offset (stored in the index register), which is added to the starting address of the array, known at compile time. The arrays can store bytes, halfwords, words, etc. This means that the address is a multiple of 1, 2, 4,

If the array index is stored in the index register, then it must be multiplied by 1, 2, 4 or 8, ie. be scaled just before the use. Scaling is performed by the shift left of the index word. It is provided by the compiler or by the hardware when the instruction with the indexed addressing is implemented.

In the architecture I80x86, the index addressing is performed by adding the offset specified in the instruction to the scaled contents of the index register SI, or DI, for example, in Fig. 1.12.

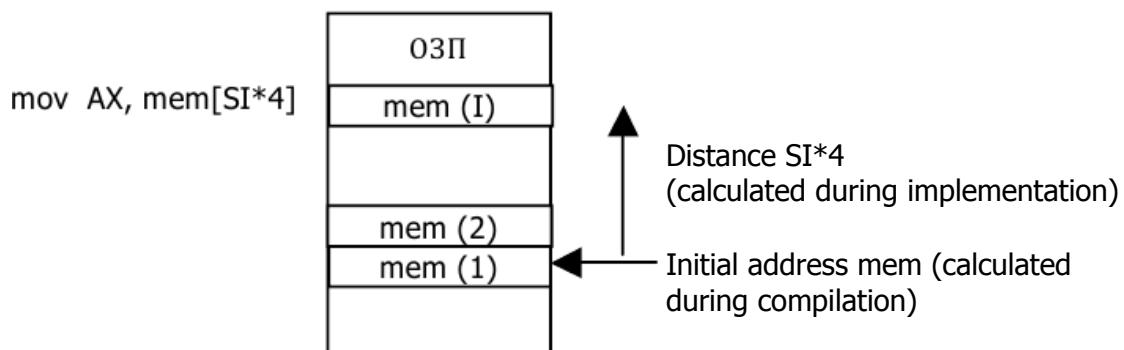


Fig. 1.12. Index addressing

Here, the array consists of a four-byte elements, and the assembler provides the array address **mem** in the object code. Since the scale factor 4 is used, the SI register contains the array index, which is loaded into it, and is modified according to the user algorithm. For example, if the 10-th array element is read, then the number 9 is stored in the SI register, as the indexing starts with zero.

1.6.4 Base addressing

This addressing is similar to the index addressing. But instead of using the index register, the base register is considered containing the address of the memory block. This is an absolute address and it does not need to scale.

This address is usually the dynamic one. In contrast to the static address, the dynamic address is unknown at compile time and is determined at runtime.

Many computers do not use the direct addressing because of the need to have relocatable programs. In this case, the base addressing is implemented in hardware. In the base register, the starting address of the program module is placed, and the absolute address is calculated as the base address plus the direct (or other) address of the instruction.

Also, the base address is used for procedure calls. With such a procedure call the parameters can be passed as the base memory address, where they are stored. In most cases, it is the address of the stack top with these parameters.

The base addressing in the I80x86 architecture is performed by the addition of the displacement X set in the instruction, to the contents of the base register BX, or BP, such as:

```
mov AX, [BP + X];
```

1.6.5 Base plus index addressing

If the data array is the dynamic one, or if the data are placed in the stack with the offset from its top, then both the base and index addressing are required. Then the absolute address is equal to $A = B + C*I$, where B is the base address, I is the index, C is the scale factor.

Many processors provide the hardware implementation of this addressing mode, which is often called as a dual-index addressing. Then the fetching, for example, an element of the array $X(I)$ is carried out by a single loading instruction. In other processors, such an operation requires several instructions to be implemented.

Generally, the procedure parameters are transferred through a stack field. Therefore, when the procedure is called, the stack pointer, its base address, and scaled operand index are used to derive the parameter address.

Such an addressing is used in the architecture I80x86 by adding the base register (AX, BX, CX, DX, BP) and scaled index register (SI, DI), as well as possible offset X, for example, in the instruction

```
mov AX, [BX + SI * 4 + X] ;
```

1.6.6 Indirect addressing

Indirect addressing occurs when the instruction refers to the operands not directly, but first reads the cell, where its address is written. For example, when the parameter is passed to the procedure, very often the parameter value is the address of the actual parameter rather than a copy of it.

Consider Q procedure (Fig. 1.13). When it is called, then the parameter **Dat** is passed to the procedure. It is not a value, but the address where the value Dat is stored. Therefore, such indirect addressing is performed in two steps.

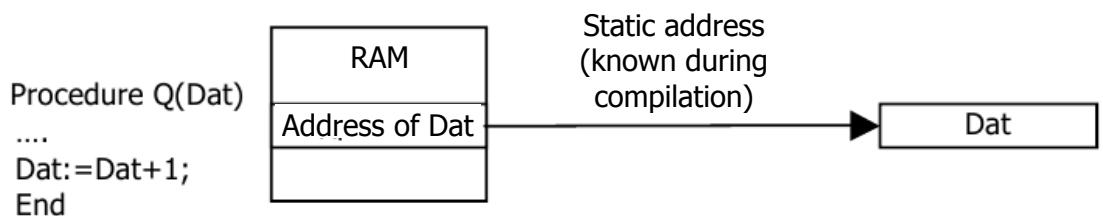


Fig. 1.13. Example of the indirect addressing

Firstly, the address of the parameter **Dat** is loaded into the base register. Then, the base addressing takes place, for example:

```
mov BX, OFFSET Dat
mov AX, [BX]
```

where the first instruction loads the address of the operand Dat in the base register BX, and the second instruction loads the operand in the register AX.

By the jump using the indirect addressing, the address is specified indirectly as the value stored in a particular memory cell, or in a register. Therefore, we can write such instruction as

```
jmp CX
```

where the register CX stores the absolute address of a jump.

In some computers, the indirect addressing is performed by hardware. Although a single instruction is needed for the data access, but two consecutive accesses to the memory remains. In the RISC computers, the registered indirect addressing is widely used. The usual RISC instruction of the data loading contains the address of the register, which holds the address of the operand to be loaded.

When the executing addresses are stored in the registers, which are pointed in the instruction, then such an addressing is named as the ***register indirect addressing***.

1.6.7 Indirect addressing with index addressing

If the parameter which is passed in the procedure during its call is an array, then the indirect addressing has to be combined with the index addressing, as in the example in Fig. 1.14.

By the use of the indirect indexed addressing, the conditional jump to multiple addresses is organized, as in the following code.

Case: mov AX, Njump; jump number loading
 mov BX, AX;
 shl BX, 3 ; index multiplied by 8
 jmp Jump_table [BX]; jump to Jump_table with the number Njump

Here the `jmp` operator performs an indirect jump to the address, which is equal to the base address plus the `Jump_table` index address, which is stored in the register `BX`.

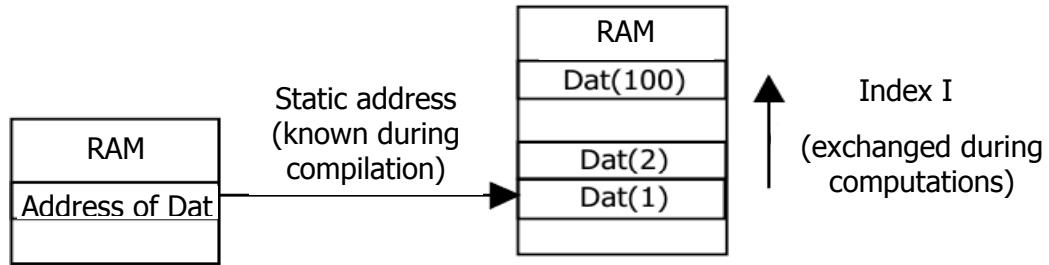


Fig. 1.14. Indirect addressing with the index addressing example

1.6.8 Base plus indirect addressing

In the previous examples, it was assumed that the pointer of the indirect addressing is the static address. But in languages, which are oriented to the stack operations (C, Pascal), the pointer itself can be placed in the stack. Therefore, to access it is necessary to add a base addressing, as in Fig. 1.15.

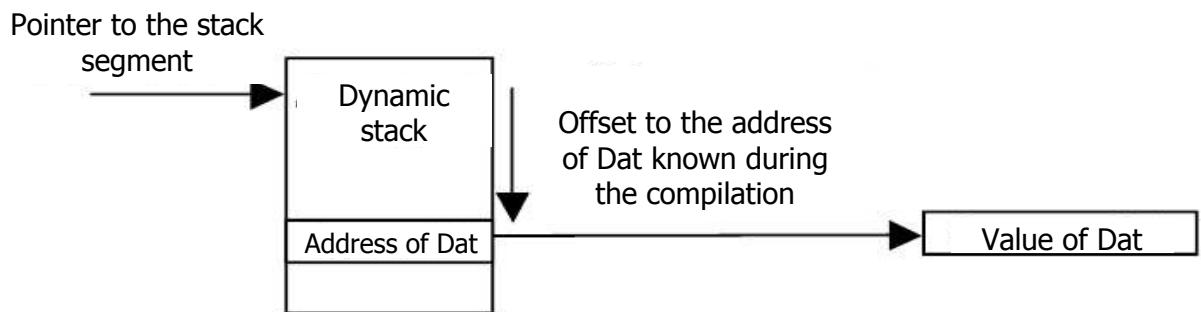


Fig. 1.15. Base plus indirect addressing

Now the addressing of the `Dat` operand includes a stack pointer offset addition to get a pointer to the `Dat`, and then this pointer is used to get the actual value of `Dat`. Such addressing may be implemented as a sequence of instructions, although some CPUs have a built-in base plus indirect addressing.

1.6.9 Indirect plus base and index addressing

Finally, consider the example where the array is disposed in a dynamic stack, and is transferred into the procedure as a parameter. Then the access to the array element $D(I)$ is performed in three steps (Figure 1.16.):

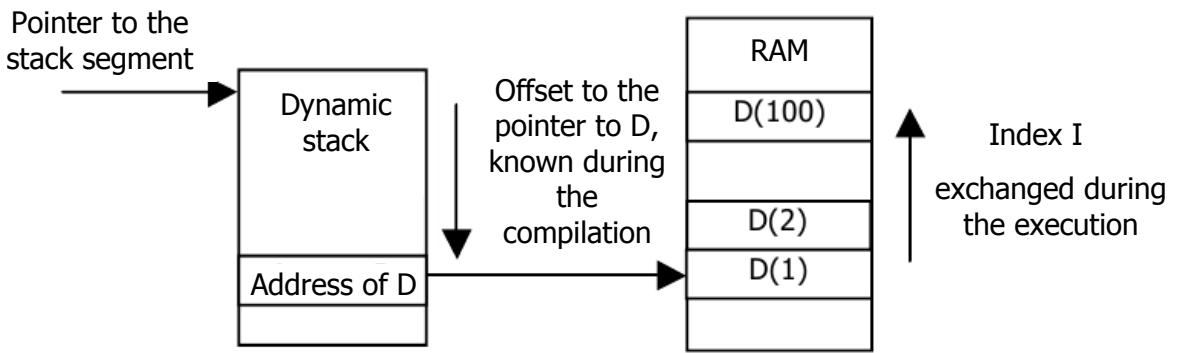


Fig.1.16. Indirect plus base plus index addressing

1.6.10 Stack addressing

In the addressing modes described above, the stack segment of the memory somewhere was considered. Really, the stack memory is used for the sake of the stack addressing. The stack memory is organized as a set of cells, which form the layers from top to bottom. Each operation with the stack is performed through its top layer, or shortly, top. Initially, the stack is empty, and has only a single top layer. When the operand is stored, it is written in the top. And the previously stored data are shifted down increasing the number of layers. When the operand is read from the top, it is pushed out, and the number of layers is decreased (see Fig.1.17).

The most of the programmable computer architectures have a couple of instructions, which deal with the stack: PUSH and POP. The first of them stores the data into the stack, and the second one reads the data. The main feature of the stack addressing consists in that, that the addressing is represented in the instruction implicitly. Therefore, the PUSH and POP

instructions are usually null or one address instructions, they are very short and very quick.

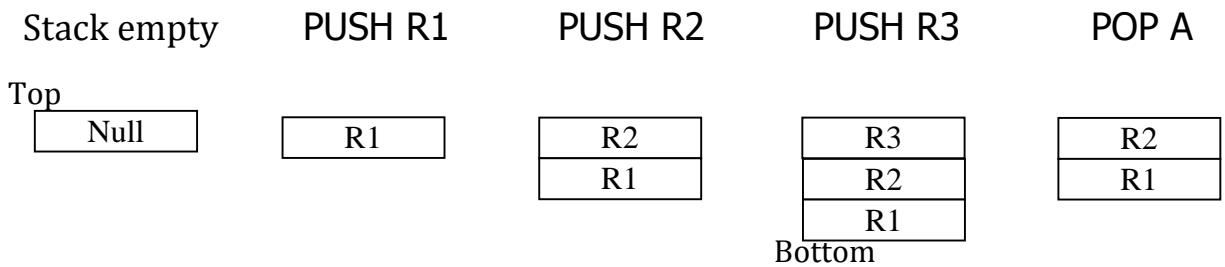


Fig.1.17. Stack addressing

The PUSH and POP instructions are often used to save the processor register contents, named the *program context*, during the procedure calls, interruptions, program switchings. In many program language realizations, the parameters to the procedures are passed through the stack using these instructions. To speed-up the context saving and restoring, some architectures have the special instructions for the group transfer the data between registers and the stack. So, the PUSH, POP instructions can store the given group of registers in the stack in the ARM architecture.

The stack handling is convenient during the expression calculating. Therefore, the mathematical coprocessor I80x87 has the stacked architecture. So, the arithmetical operation is implemented with the operands in the top register of the stack and in this register but the next one.

Some architectures are organized around the stack memory. They are distinguished in the short instructions and programs, and in the frequent use of the procedure calls at the cost of the slowed performance.

Usually, the stack is organized as a special segment in the main RAM. Therefore, the stack memory can have the access as the usual RAM. Then two pointers are held on the processor. The first pointer assigns the bottom of the stack, it is usually written by the operational system. The second one does the

top and is named as the ***stack pointer***. Depending on the architecture or on its control, the stack pointer contents can be increased or decreased during the stack filling. This means that the bottom pointer can specify either the lowest or highest address of the stack segment.

In the I80x86 architecture, the stack pointer register is SP (ESP) and is automatically incremented during the **PUSH** instruction implementation. The value of the increment +1, ..., +8 depends on the width of the data pushed in.

1.4.5 Problems

- 1) What feature do the index and base addressings distinguish?
- 2) What is the dynamic addressing? Why is the indirect addressing needed for its implementation?
- 3) What kind of addressing is needed for the dynamic array access?
- 4) Propose the algorithm of the formula $y = a*x^2 + b*x + c$ processing using the stack as in the I80x87 coprocessor.
- 5) How to program the case statement on the assembly language?
- 6) Program in some assembly language the loading in the accumulator the array element $a[i,j]$, which is a word, using the proper addressing modes.
- 7) The ***attached addressing*** is the addressing mode, when the resulting address is formed by the concatenation of two address code words. To which kind of addressing does this addressing mode belong and why?
- 8) What features has the register direct addressing? Why is it widely used in the modern computers?
- 9) What features has the register indirect addressing? Why is it widely used in the modern computers?
- 10) In many RISC processors, the register R0 stores a constant zero. What kind of the addressing mode does the instruction with the register R0? For which feature is such a register introduced in the architecture?

1.7 Intel 8051 architecture

The Intel 8051 microcontroller was developed by the Intel company in 1980 for the use in embedded systems. Its architecture is termed as MCS-51. The microcontrollers of original versions were popular in the 1980s and early 1990s. Its enhanced compatible derivatives remain popular today and are the most popular 8-bit microcontrollers. Several companies offer I8051 derivatives as Intellectual Property cores for the use in FPGA or ASIC. Despite its age, this architecture remains the actual, and will be in the use in the next decades.

The I8051 architecture is worth to be familiarized in educational purposes because it is simple, contains the most of the features of the CISC processor architectures. The programming in the assembly language and modeling of this architecture helps to understand the essence of programmable computers. Let consider the architecture features, described in this chapter, in the short description of the I8051 architecture.

1.7.1 Data types

The I8051 microcontroller has only one data type. It is 8 bit word or a byte. And the size of each register is also 8 bits. The byte can represent both unsigned and signed integer in the ranges 0 — 255 or -128 — 127, respectively. But in the most cases, the data are considered as unsigned.

The bytes can represent the characters as well. But this feature has not any relation to the architecture.

The halfword data type is used exceptionally as the two byte address word. This hafword is stored to the 16 bit Data Pointer (DPTR) register to have the access to the outer data memory.

The I8051 architecture is intended for the frequent bit manipulation in the bytes. Therefore, a bit can be considered as a special data type as well.

1.7.2 Address spaces

The 8051 architecture provides the user with three physically distinct memory spaces which can be seen in Fig. 1.18. The 8051 microcontroller's memory is divided into program memory and data memory. Program memory (CODE) is used for permanent saving program being executed, while data memory (DATA, XDATA, IDATA, SFR) is used for temporarily storing and keeping intermediate results and variables.

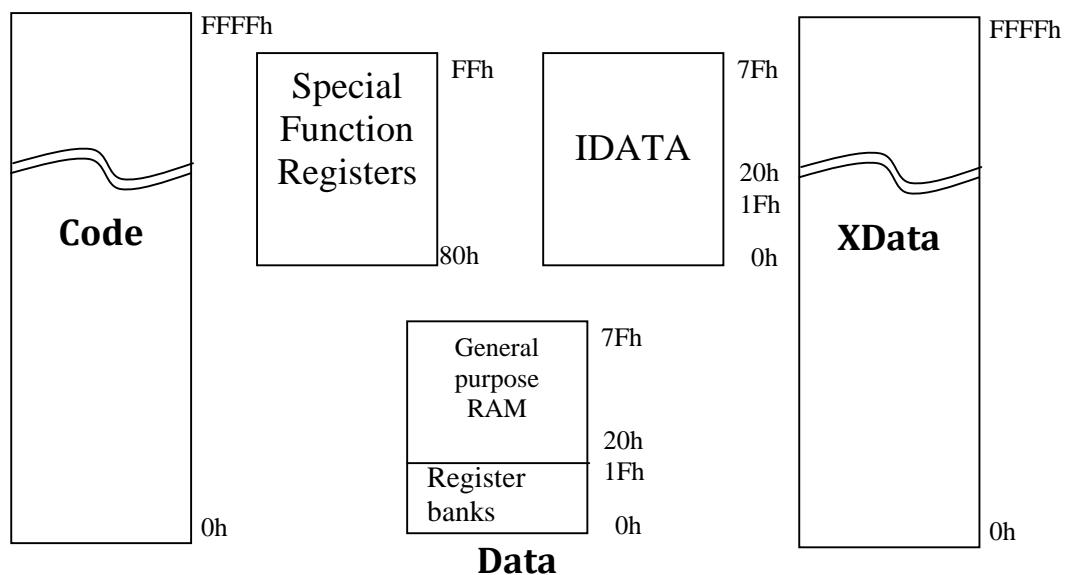


Fig. 1.18. Address spaces of I8051

The first memory space is the **CODE** segment, in which the executable program resides. This segment can be up to 64K (since it is addressed by 16 address lines). The processor treats this segment as readonly. Many embedded systems these days are using EEPROM, which allows the memory to be overwritten by an external device.

In addition to executable code, it is common practice with the I8051 to store fixed lookup tables in the CODE segment. To facilitate this, the I8051 provides instructions, which allow rapid access to tables via the data pointer

(DPTR) or the program counter with an offset into the table optionally provided by the accumulator.

The second memory space is the 128 bytes of internal RAM on the I8051. This segment is typically referred to as the **DATA** segment. The RAM locations in this segment are accessed by the direct addressing. The variables stored in the DATA segment, can also be accessed indirectly via R0 or R1 register.

The DATA segment contains a subsegment, which consists of the four sets of register banks, which compose the first 32 bytes of RAM. The 8051 can use any of these four groups of eight bytes as its default register bank. The registers of each bank are referred to us as R0, R1,...,R7.

The selection of register banks is changeable via the RS1 and the RS0 bits in the **Processor Status Word (PSW)** register. The register bank switching allows not only for quick parameter passing, but also opens the door for simplifying task switching on the 8051.

The **Special Function Register (SFR)** segment samples the ALU registers, control registers and the peripherals on the I8051 at locations 80h and above. Many of them are bit addressable. Such a register has the zeroed least significant nibble of its address. For example, the accumulator register ACC has the address E0h. The bits in the bit addressable SFRs can either accessed by name, index or bit address. Thus, using the bit accessing instructions, you can refer to the EA bit of the Interrupt Enable SFR as EA, IE.7, or 0AFH.

A memory map of the SFRs is shown in Table 1.2. The bit addressable registers are marked in bold. The free cells of the map are usually filled by the special registers, which are added to the architecture in some expansion clones of the I8051.

Table 1.2. SFR segment map

Address	0	1	2	3	4	5	6	7
F8								
F0	B							
E8								
E0	ACC							
D8								
D0	PSW							
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2		
C0								
B8	IP							
B0	P3							
A8	IE							
A0	P2							
98	SCON							
90	P1							
88	TCON	TMOD	TL0	TL1	TH0	TH1		
80	P0	SP	DPL	DPH				PCON

Certain I8051 family members, such as the 8052, contain additional 128 bytes of internal RAM, which reside at RAM locations 80h and above. This segment of RAM is typically referred to as the **IDATA** segment. Because the IDATA addresses and the SFR addresses overlap, address conflicts between IDATA RAM and the SFRs are resolved by the type of memory access being performed, since the IDATA segment can only be accessed via indirect addressing modes.

The final 8051 memory space is 64K in length and is addressed by the same 16 address lines as the CODE segment. This space is typically referred to as the external data memory space (**XDATA**). This segment usually consists of some sort of RAM and the I/O devices or external peripherals, to which the I8051 must interface via its bus. Read or write operations to this segment are performed using either DPTR, R0, or R1 registers.

1.7.3 Structure and registers

The processor structure is shown in Fig. 1.19. The narrow rectangles in Fig. 1.19 represent the 8-bit data/address common bus and 16-bit address bus. The legend in it mean the following: BUF – buffer, AR – address register, PROM – program read-only memory, DAR – data address register, DAA – decimal adjust, and constant unit, T1, T2 – temporary registers 1,2, SFRU – special function register unit, BSC – synchronization and control block. The other abbreviations are described below.

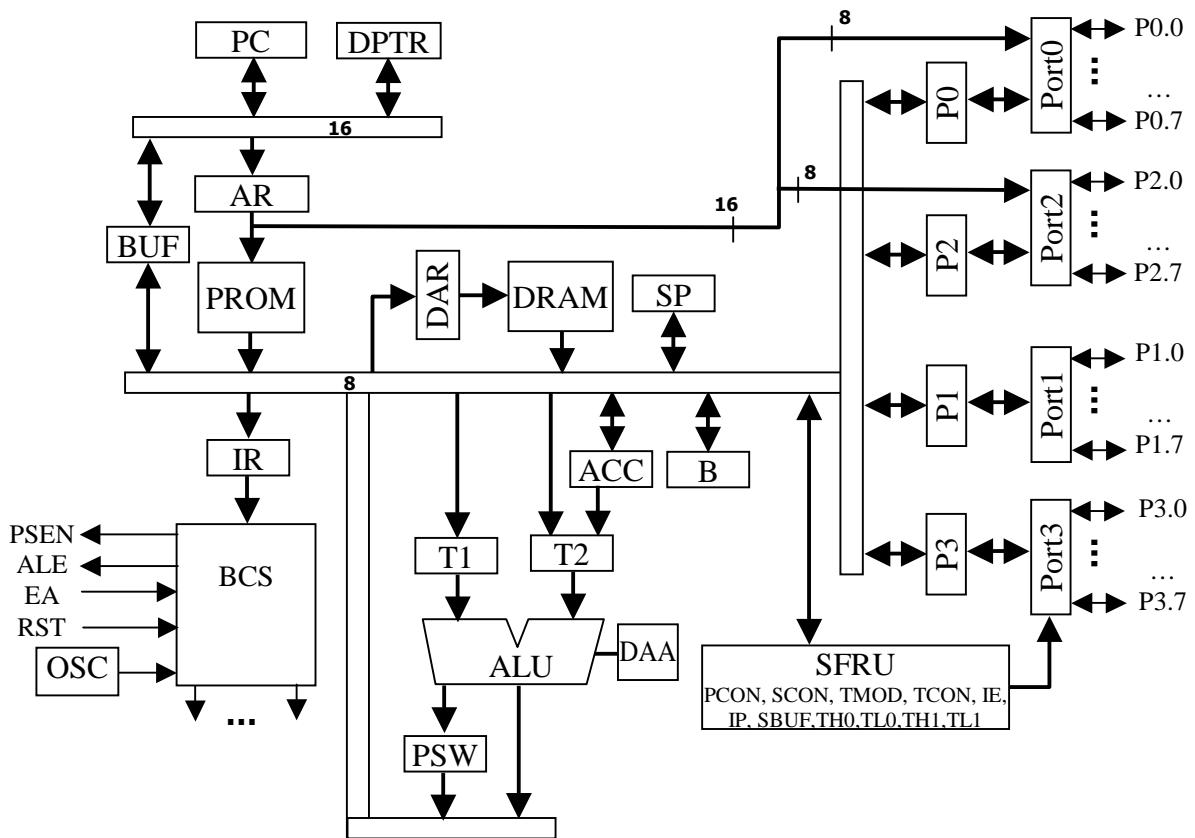


Fig.1.19. I8051 microcontroller structure

This structure shows both the architecture and microarchitecture of the I8051. A set of so-called architecture registers like ACC, P0,...,P3, SP, DPTR, memory units, which are used in the instructions, describe the architecture.

The registers, modules, which are invisible to the programmer, like IR, T1, T2, ALU, busses, oscillator (OSC), etc. describe the microarchitecture.

The executed instruction, read from PROM, is stored in the ***Instruction Register*** (IR). The instruction decoder in BSC decodes the opcode and identifies the type of the instruction to execute. After that, a sequence of the control signals for the instruction execution is read from the microprogram memory of BSC.

ACC is the ***accumulator*** register. The instructions, which access the accumulator, use frequently its short name 'A'. For example, loading the port 2 content to the accumulator is coded as MOV A, P2. The name 'ACC' is used, for example, for the bit-wise addressing of the accumulator. Thus, the symbolic name of the fifth accumulator bit is codes as ACC.5.

B register is used for the temporary storing the operand, for example, during the multiply and divide operations.

The Program Status Word (**PSW**) register contains information about the program state. Its structure is shown in Fig. 1.20. The meanings of its bits are shown in the annex to this book. The carry bit CY or C in it is widely used in computations. The bit OV means the result overflow. The bit AC means the carry from the 3-th bit to the 4-th bit of the result and is used in the binary-decimal calculations. The parity bit P signales about the parity of any data in the accumulator. The bit F0 can be used as the programming flag. Bits RS1, RS0 set the working register bank.

The 16-bit ***Address Register*** (AR) stores the address of memory segments CODE or XDATA.

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	OV	—	P

Fig.1.20. PSW register

The **Stack Pointer** (SP) is the 8-bit register, whose contents is incremented before writing data to the stack when performing the instructions PUSH and CALL. Initial state of SP is 07h and it points to the stack area in data RAM, which is starting with 08h. By overriding SP, the stack region can be placed in any place of the data RAM, if necessary.

The **Data Pointer** (DPTR) consists of a high byte (DPH) and the low byte (DPL). It stores a 16-bit address when accessing the external memory XDATA. It can be used as a 16-bit register or as two independent eight-bit registers.

Special function registers P0, P1, P2, P3 are parts of the microcontrollers porst **Port 0,...,Port 3**, respectively.

The **Serial port Buffers** SBUF are two separate registers: the transmitter buffer and receiver buffer. When the operand is written to SBUF, then it enters the transmitter buffer, and this writing initiates its transmission through the serial port. When the operand is read from SBUF, it is fetched from the receiver buffer.

The **Timer** registers are the pairs (TH0, TL0) and (TH1, TL1), which form the 16-bit registers of the timer/counter 0 and timer/counter 1.

Special function registers IP, IE, TMOD, TCON, SCON, and PCON are the control registers and contain control bits and bits of the system interrupts, timers/counters, serial port control, circuit energy supply switching.

1.7.4 Datapath and control block

In the I8051 microarchitecture, the **datapath** is formed by ALU, Decimal Arithmetic Adjust (DAA) unit, registers T1, T2, ACC, B, PSW, as well as the busses which connect them together. The registers ACC, B, PSW logically belong to DRAM and SFRU, because they are SFR registers, and they can be accessed as the DRAM cells. The temporary registers T1, T2 store intermediate operands, and could not be accessed by the program.

8-bit **ALU** can implement 51 different operations like addition, subtraction, multiplication, division, logic operations and cyclic shifts. It is designed to operate with nibbles (4-bit operands), swaps nibbles in a byte, adjusts the nibbles when operating with the binary-decimal numbers.

During a single instruction cycle, ALU can perform a sequence of operations, for example, incrementing the 16-bit coupled registers. During some jump instructions, ALU increments the PC register three times and compares two operands.

ALU can operate with bits. For this feature, some people say about the “Boolean processor” built in I8051. This helps to design the control applications, which check and exchange the separate pins of the microcontroller chip. So, ALU can operate with four data types: Boolean (1 bit), nibble (4 bits), byte (8 bits), address word (16 bits).

The **control block** BCS is intended for generating the synchronizing and control signals, which provide the coordination of the mutual operation of the microcontroller blocks in all their modes. BCS consists of the timing sequence generator, input-output logic, instruction decoder and programmable logic device (PLD). The words fetched from PLD represent the microinstructions which control all the units including datapath.

The most of the instructions are performed for one or two cycles. The multiply and divide instructions last four cycles. A single instruction cycle has six states S1 - S6, each of them contain two phases P1 and P2. A single phase lasts a cycle of the clock signal generated by the inner clock Oscillator (OSC) or enters the chip through the pin BQ. If, for example, the outer clock frequency is equal to $f_{BQ} = 12 \text{ MHz}$, then the instruction cycle lasts $T_{MC} = 12/12 \text{ MHz} = 1 \mu\text{s}$.

1.7.5 Addressing modes

The immediate, direct, indirect and implicit addressing modes are used in I8051. The immediate byte or halfword operand is present in the instruction word. Some operand addresses are present in it implicitly. They are accumulator A, registers B, DPTR, PC, carry bit C.

The access with the direct addressing can be performed only to the DATA segment and to SFRs. During the indirect addressing of the DATA segment, eight bits of the address are fetched from the index register R0 or R1. The indirect addressing is performed through the registers DPTR, PC, R0 and R1 to the IDATA, XDATA segments as well.

To get the data from the CODE segment, the **PC relative addressing** is used. Then the offset code D is loaded in the accumulator, and the instruction MOVC A, @A + PC loads to the accumulator the data at the address, standing at a distance D from the address in the PC register.

The I8051 instructions allow many combinations of addressing modes, making them flexible and versatile, which is shown below.

1.7.6 Instruction set

The I8051 instruction set contains 111 instructions, which are distinguished by their functionality to five groups: data transfers, arithmetic operations, logic operations, jump-type instructions and Boolean operations. The most of the instructions have one or two byte width and are implemented for one or two instruction cycles. The first instruction byte contains the opcode. The second and third bytes contain either operand addresses or direct operands. Below some instructions are considered, which formats are shown in Fig.1.21. A list of all instructions is represented in the addendum.

Data moving instruction examples, except AJMP, are represented in Fig.1.21. We can see the variety of instruction codings and address modes on

these examples. Due to the assembly language syntax, the first word of the instruction is its mnemonic, the second and third words, separated by a comma, represent the operands. The first of them is the destination address, and the second one is the source address of the operand to be moved.

MOVX A, @DPTR	11100000		
MOV A, Rn	11101 rrr		
MOV A, @Ri	1110011 i		
MOV A, ad	11100000 ad		
MOV Rn, #d	01111	rrr	#d
MOV ad, @Ri	1110011	i	ad
MOV add, ads	10000101	add	ads
MOV DPTR, #d16	10010000	#d16	
AJMP ad11	a ₁₀ a ₉ a ₈	00001	a ₇ ...a ₀

Legend:

- Rn — register from R0 to R7
- rrr — three bit register address
- i — index register address
- ad — eight bit data address in DATA segment
- #d, #d16 — eight and sixteen bit immediate operands
- add, ads — destination and source addresses in DATA segment
- ad11 = a₁₀...a₀ — twelve bit jump address in CODE segment

Fig. 1.21. Instruction formats

The instruction MOVX A, @DPTR loads a byte from the segment XDATA to the accumulator. The prefix character '@' means the indirect 16-bit address, which is stored in the DPTR register in this example. It is the null address instruction and has the implicit indirect addressing.

The instruction **MOV A, Rn** loads the register content to the accumulator, is one address instruction, and has the implicit and direct addressing. The instruction **MOV A, @Ri** loads the DATA memory cell content with the address, which is stored in an index register R0 or R1, to the accumulator. It is one address instruction, and has the implicit and indirect addressing.

The instruction **MOV A, ad** loads the operand from the DATA segment to the accumulator, is one address instruction, and has the implicit (accumulator) and direct addressing. The instruction **MOV Rn, #d** loads the constant #d to the register. Here the prefix '#' means the immediate operand, and d can be any eight-bit number. Therefore, it is the direct and immediate addressing instruction.

The instruction **MOV ad, @Ri** moves the operand from the DATA segment by the address, stored in register Ri , to the address ad in the same segment. It is the two address instruction with the direct and indirect registered addressing.

The instruction **MOV add, ads** transfers the data from one to another cell of the DATA segment, and is two address instruction with the direct addressing. The instruction **MOV DPTR, #d16** loads the 16-bit constant to the register DPTR.

The variety of data moving instructions is formed by exchanging the types of the first and second operand of the **MOV** instructions. All of them are represented in the addendum. Among them are the instruction **MOVC A,@A + PC**, that loads the data from the CODE segment, stack handling instructions **PUSH** and **POP**, instructions **XCH**, which swap the data as well.

The programmer usually selects the proper moving instruction trying to optimize both the performance and the length of the program.

24 instructions form a group of ***arithmetic operations*** (see Addendum), performing the addition, decimal correction, increment,

decrement bytes. Some instructions perform subtraction, multiplication, and division of bytes.

ADD and ADDC instructions allow the addition the accumulator with a large number of operands. Similarly to ADDC, there are four SUBB instructions that make it easy to calculate a subtraction of multi-byte binary numbers. The I8051 implements an expanded list of instructions of incrementing / decrementing bytes, including the increment of 16-bit data pointer register DPTR.

The group of ***logic instructions*** is formed by 25 instructions, implementing the logical operations on bytes. It is possible to make the "XOR" operation with the contents of the port registers. The XRL instruction can be effectively used to invert individual bits of a port.

A distinctive feature of the ***bit instructions*** is that they operate with the one-bit operands. As such operands, the individual bits of some SFR registers can serve, as well as 128 software user flags. To address these bits, the straight eight-bit address is used. There are the bit reset (CLR), set (SETB) and inversion (CPL), as well as conjunction and disjunction of the carry flag instructions.

This group of ***jump-type instructions*** includes the instructions of the conditional and unconditional branch, subroutine call and return, and the no operation NOP instruction as an exclusion. The most instructions use the direct addressing. There are 3 types of jump instructions which are distinguished in the branch code width.

There are two ***long jump*** instructions: jump instruction LJMP and routine call LCALL. In these three byte instructions, the full 16-bit address occupies two bytes. This address is loaded in the PC register during the jump or call. The ***absolute jump*** occurs within a program memory page with the

volume of 2048 bytes. These instructions, like AJMP in Fig. 1.21, have the 11-bit absolute address code ad11 in the 2-byte code. During the jump, the 11 least significant bits of the PC register are substituted by this code.

Short *relative jump* performs the branch in the range from —128 to 127 bytes relatively to the next instruction address. There are the short unconditional jump instruction SJMP and a set of jump instructions under the condition of zeroed accumulator (JZ), not zeroed accumulator (JNZ), carry bit is one (JC) or zero (JNC), selected bit is one (JB) or zero (JNB).

The *decrement and jump* instruction (DJNZ) is usually used for the loop programming. It decrements the selected memory cell, and if the result is not zero, performs a jump.

The instruction JMP @A + DPTR provides a jump to the indirect address, which is calculated as a sum of addresses in the accumulator and DPTR register. This instruction is useful to make a jump to several directions as in the Case operator of the programming language.

The instructions LCALL and ACALL perform the *subroutine calls* to the long and short address respectively. The instruction RET implements the return from the subroutine, and the instruction RETI does the return from the interrupt routine.

The other features of the I8051 architecture are considered later.

1.7.7 Architecture summary

The I8051 architecture is distinguished in the following.

This is the CISC-architecture. It has a set complex instructions of different lengths. For example, the instruction JBC performs a jump if the bit is set, then this bit is cleared, which affords a lot of machine cycles to perform. Moreover, during this operation implementation, the access to some SFR

control registers is prohibited (from outer processes, timers, etc.), because the selected bit control can cause the unintended behavior.

This is a 8-bit architecture. The byte is the main data format, however, the selected bits and halfwords take part in the calculations.

The architecture has the powerful addressing mode set. The large set of data moving and arithmetical-logic instruction has multiple possibilities to select the sources and destinations of the data including the direct, indirect, implicit, immediate addressing. This signs to the CISC nature of the architecture as well.

The architecture has several fixed memory segments like DATA, XDATA, CODE. For the fact, that instructions are stored in the CODE segment and are fetched independently on the data access in the DATA segment, it is named as the ***Manchester architecture***. This provides the immunity to the program damage because it is stored in the independent ROM. In the highest variations of the I8051 architecture, this improves the instruction speed because instructions and data are accessed in parallel.

The memory segments, mentioned above, have comparatively small volume, which is less than 65536 cells. This disadvantage demonstrates itself in recent times when the embedded operation systems, the data arrays and files become megabytes of the volume. But in the new microcontroller models, the memory volume is enlarged by the addition of the page registers, which something decreases this disadvantage.

The I8051 microcontroller remains the “working horse” in the embedded systems for its small hardware complexity, small program length, high reliability, and energy effectiveness, huge ***ecosystem*** (accessible compilers, libraries, development and debugging tools, the infrastructure of the chip production, distribution, engineer education). This is mostly due to its effective architecture.

1.7.8 Problems

- 1) Why is the memory space in the I8051 architecture divided to the separate DATA and CODE segments?
- 2) What advantages to the architecture does the four register banks give, which are placed in the DATA memory?
- 3) For which purposes is the PC relative addressing effectively used?
- 4) For which purposes are the MOVC instructions used? What constraints are applied to the parameters of the instruction MOVC A, @A+PC?
- 5) How to add some new registers to the SFR segment of the memory of the I8051 architecture?
- 6) Why the bit addressable SFR registers have the lowest address bits which are equal to 0h or 8h?
- 7) How to program the access to the IDATA segment?
- 8) How can the page register, which is added to the I80x51 architecture, increase the XDATA memory volume?
- 9) Propose an assembly program piece, which implements the moving the array from the XDATA segment to the DATA segment.
- 10) Propose an assembly program piece, which implements the function $y = f(x)$, where x is the one byte data.
- 11) Propose an assembly program piece, which tests the 0-th bit of the port P0, and if it is equal to 1, resets it.

1.8 RISC and CISC processors

1.8.1 History of the RISC processors

In 60-80 years, the architects of new CPUs have tried to add in the instruction sets the instructions, which were supposed to accelerate the implementation of some calculations, word processing, compilation. These instructions have been difficult to perform and they were performed by multi-cycle microprogrammes. Therefore, the instruction set for each new computer generation was characterized by more and more complex instructions.

On the other hand, the interests of programmers differed from the computer developer's ideas which are built in the instruction set on many issues. Many instructions (for example, the TRT instruction of the string checking, decimal arithmetic instructions in IBM-360) did not take part in the compilation. They could be used only in certain cases, and the additional conversion of data formats was needed. It was also found, that for the processors with such an instruction set, it is difficult to perform the automatic optimization of the programs under compilation.

To support the software portability, the instruction sets became more complicated with each new processor generation. To minimize the hardware volume, the rarely used instructions were implemented using subroutine call using the TRAP instruction.

Thus, the computer with the complex instruction set (**CISC-processor**) always has the excessive complexity of the control blocks, and as a consequence, it has the relatively low speed to hardware volume ratio. Through the nonoptimality of the compiled programs, it is also characterized by large hardware downtime.

The idea of the RISC processor is based on the fact, that the complex multi-cycle instructions can be substituted by the chains of one-cycle simple

instructions. Hence their name — ***Reduced Instruction Set Computer*** — means, that their instructions have reduced complexity.

For the first time, the principles of the RISC processor were introduced in the CDC 6600 computer in the early 60-ies. When S. Cray, the creator of this computer, has found that in the scientific calculations he can use the simplified instruction set, which is easy to decode. It helped to create an effective Fortran compiler as well. The novelty was that:

- only data read-write instructions can access the RAM, the RAM address is stored in the register, and the indirect addressing is used;
- the arithmetic instructions operate with the two source and one destination registers;
- the instruction format is simple and uniform;
- several functional modules process the data in parallel, and the instruction execution is pipelined, to perform the parallel instruction implementation, the functional modules loading is dispatched.

The IBM-801 project was the first attempt to develop an RISC architecture intentionally. The aim of the project was to create a computer that can quickly switch the context during interrupts. Its development began in 1974 and in 1980 the prototype was ready. This scientific development had a great influence on the formation of RISC architectures in the world.

In 1980-1983 in Berkeley the first RISC microprocessors **RISC I** and **RISC II** have been developed under the direction of D. Paterson. At the same time, at Stanford, the another RISC microprocessor named **MIPS** (Multiprocessor without Interlocked Pipeline Stages) was developed. The ideas, which have been laid out in these microprocessors, were then introduced in a series of SPARC and MIPS processors, respectively.

1.8.2 Principles of the RISC processor design

The RISC processors are designed on the base of a set of following principles.

One instruction per a clock cycle. Simple instruction has to be executed in a single clock cycle. This principle is contradictory to the functioning of the von Neumann processors. According to it, the beginning of the next instruction is due to the result of the previous instruction. The instruction could not begin before the end of the previous one, it must pass four successive stages of its implementation (see. Von Neumann architecture). Therefore it can not be executed in a single cycle, from start to finish. But the following architectural features of the RISC processors are intended to support the processor speed which is equal to one instruction per a cycle.

Pipelining. Each instruction goes through several stages: instruction fetching (IF), instruction decoding (ID), reading the operands (RO), the operation with operands (OP), writing the results (WR). If the neighboring instructions are independent on each other, these steps may be performed in an instruction pipeline — one step per cycle as in the timing diagram in Fig. 1.22.

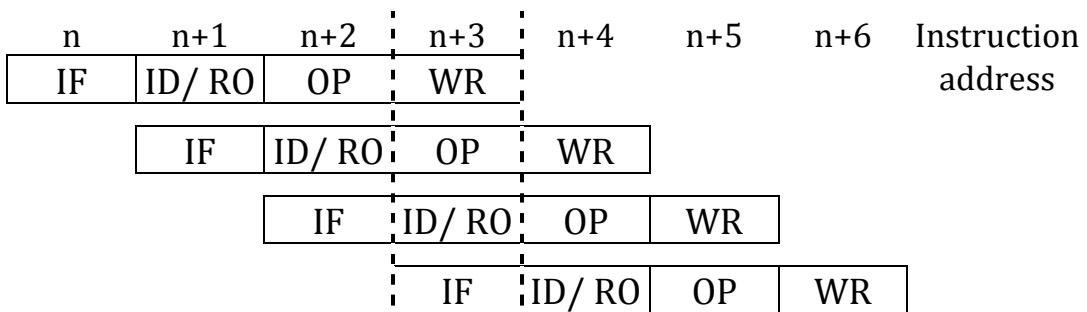


Fig. 1.22. Implementation the instructions in the 4-staged instruction pipeline

Register memory. If the operands of the neighboring instructions are placed in the register memory, then these instructions can be executed in a single cycle, not to take into account the fetching and decoding these

instructions. Then the result, calculated in this cycle, may participate as an operand for another instruction in the next cycle. This phenomenon is referred to as the ***operand forwarding*** (Fig. 1.23).

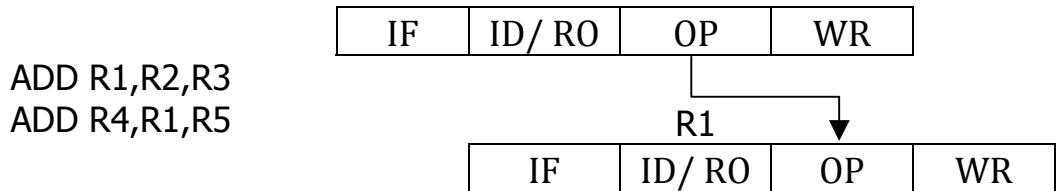


Fig. 1.23. Operand forwarding from one instruction to the other through the register R1

Therefore, the instruction set of the RISC processor is divided into a subset of registered data processing instructions, a subset of the data transfer instructions and a subset of control instructions.

Suppose we have a program:

```
L R2, A
L R3, B
NOP
ADD R4, R2, R3
L R5, C
L R6, D
NOP
ADD R7, R5, R6
```

Here the empty NOP instructions (No OPeration) are inserted to perform the delay, during which the data A, B, C, D have time to be read from the external memory by the load instruction L. To do without the NOP instructions, the instructions are rearranged so that the delays are filled naturally:

```
L R2, A
L R3, B
L R5, C
L R6, D
ADD R4, R2, R3
ADD R7, R5, R6
```

Here the instructions are performed with the overlapping. The compiler optimizes the instruction order, so the instructions in the chains perform the ***software pipelining*** without the interlocking the data. This means that the data are pre-loaded into the registers, so, the data arrive in time the instructions, which use them. Besides, the results are uploaded from the registers on the computing background. Also, the operation schedule is performed in such a way, that the data are processed and the intermediate results are longer stored in registers and not copied back to the RAM to minimize the exchanges between the registers and memory.

Running the instructions during the jump instruction processing.

When the jump instruction is performed, after its decoding, the program pipeline has to be cleared. I.e. the instructions that follow it, and were loaded into the pipeline, should be rejected, since these are not the instructions that are executed in accordance with the program flow. This raises the jump delay and the pipeline stall, while the correct instruction is loaded to the pipeline. So, the ***branch delay slot*** occurs during the branch-type instruction execution. But in some RISC processors, the pipeline is not cleared during jumps and implements up to N instructions after the jump instruction, which are named as the ***delayed branch instructions*** (see Fig 1.24).

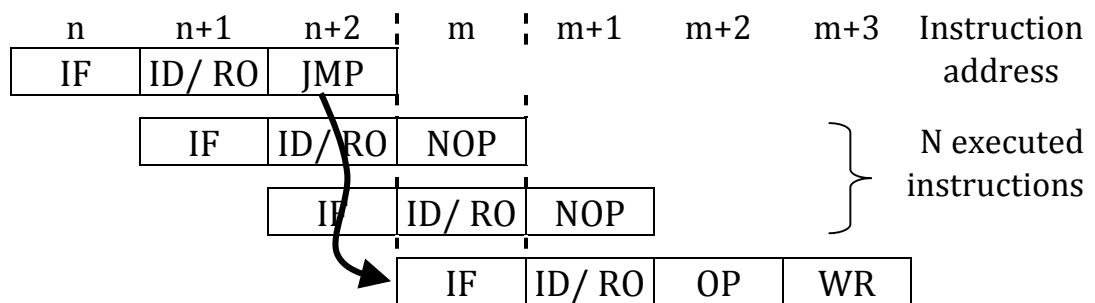
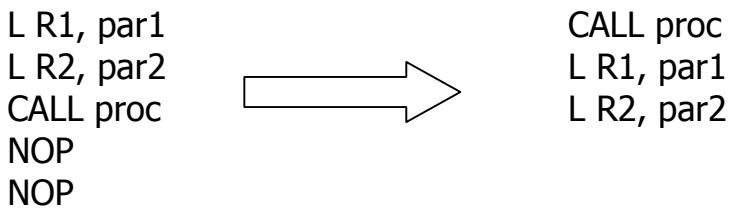


Fig.1.24. Running the jump instruction in the RISC instruction pipeline

To preserve the semantics of the program, these N instructions have nothing to do, then they are the NOP instructions. But in many cases, the

compiler can replace them by a chain of instructions that perform the useful actions. For example, after the subroutine call instruction it can insert the first N instructions from the called subroutine. Then the call branch is performed round the removed instructions. Then the return from the subroutine is done to the instruction, which stays after the chain of these N instructions.

The another way is to place the instructions of the parameter transfer to the subroutine after the CALL instruction to it as in the following example:



As a result, at the cost of the delayed branch, the branch instructions can be performed averagely for a single clock cycle.

Simplified addressing. The simplified instructions must have the simplified addressing. First of all, the data in the memory must be aligned to the address boundary. For example, the address of 4-byte words always has two zeroed least significant bits. Then the reading of this word can be performed by a single instruction, performing only one transfer via the data bus. If the word is not aligned to the address boundary, then it has to be read in three steps: lowest and highest parts are read and then assembled together.

Furthermore, this feature simplifies the cache RAM handling. Note that the cache RAM always stores the copy of the data string from the main memory. And this string is always aligned to the address boundary. When some word is not aligned then the conflict situation is possible, when only a part of this word is stored in the cache RAM.

So as a direct address can not fit into the instruction as a whole (32-bit address of a 32-bit instruction), the indirect addressing is most commonly used in the RISC processors. The indirect addressing is performed rather simple. One instruction loads or modifies the executing address in some

register, and other instruction makes the memory access with it. This couple of instructions is implemented more quickly and simple than the equivalent complex instruction in the CISC-processor.

Simple instruction format. Simple, even logically redundant, instruction format, including an opcode, address fields, immediate operands, provides a simple and therefore, fast decoding of instructions and addresses. Thereby this minimizes the period of the clock interval.

All instructions in the RISC processors tend to have the same length and the minimum instruction formats. In this case, the position of the opcode and address fields in the neighboring instructions is known in advance. This makes it possible not only to simplify the decoder but to decode multiple instructions simultaneously. This simplifies the forecasting the access to the RAM, so the processor can read the operands and instructions from it in advance.

The RISC processors have the progressive architecture, which provides a high performance — power, performance — hardware costs ratios and therefore, it is ideal for its realization in modern chips.

1.8.3 Comparison of RISC and CISC processors

Since the decoding of complex instructions is difficult, then they could not be quickly fetched from the RAM and their execution parallelization is complex or impossible. Therefore, the CISC processors tend to have a low speed in comparison with the RISC processors. As a result, the perfect CISC architecture Intel-432, as well as the architecture DEC VAX, were unable to compete with the RISC architectures in the mid-80s.

The CISC-architecture Intel 80x86 is an exception that proves the rule. This architecture loses the RISC architectures, such as PowerPC, PARISC, MIPS, SPARC for speed. But, firstly, at a time, when the RISC machines appeared on the market, the personal computers with the I80x86 processors

have already gained a leading role in the PC market. Besides, tens of thousands of applications for them were expanded over millions of PC users. Therefore, the vast majority of PC users have chosen the I80x86 architecture, as it was uncomfortable and unprofitable to move to the RISC architecture. The exceptions were the workstation users, which were few. Secondly, the architecture I80x86, since I80486, has evolved with the purchase of attributes of the RISC processors, by increasing the number of registers, the addition of simplified instructions, pipelining the datapath.

It is important to evaluate the influence of the compilers to the architecture choice. The fastest program is a program written in assembly language. But now it is rarely used programming language. Therefore, the computer performance depends not only on its hardware speed as from the compiler effectiveness. The principle of the compiler operation is recognition of operations (or idioms) in the program and substituting them by the library procedures, described by the machine codes. The optimizing compiler selects such substituting procedures, which must comply with the minimum number of executed instructions. That is, the program optimization problem is considered as the difficult combinatorial problem, which complexity substantially increases with the increase of the set of the objects to be selected. I.e., if the instructions are complex, then the number of instruction chains, that must be selected, becomes very huge.

This optimization problem is simplified if the instruction set has a small number of instructions. Thus, it was found that the C compiler for the CISC processor 68020 uses only 30% of all the instructions of the set. In addition, the similar CISC instructions usually use the different operand types. Then the compiler must always monitor the consistency of types of executed data and have a large set of library routines with a variety of input and output data types.

The RISC architecture has a simple instruction format, minimized instruction number, a lot of data registers, which use is uniform, etc. All this makes it possible to construct a simple compiler, and to do the software optimization more efficient and deeper.

1.8.4 Problems

- 1) Why is the clock frequency of the RISC processor much higher than one of the CISC processor? Estimate the ratio of these frequencies.
- 2) Propose the situations, when the von Neuman processor could not start the next instruction implementation before the finish of current one and explain why.
- 3) Propose the methods which increase the clock frequency of the RISC processor to the extremum values.
- 4) Two RISC processors have the 4 and 8-staged pipelines and operate at 1 and 1,5 GHz, respectively. Compare the throughputs of them in the situations, when they perform the program without the branch instructions and the program, in which each 4-th executed instruction is the branch.
- 5) Take an example of the program pipelining in some PC program.
- 6) Propose the algorithm of the 4-byte word reading when this word position is not aligned to the address boundary.
- 7) Propose the instruction set of the RISC processor, which has the indirect registered addressing in the 3-address instructions. The instruction number is 32, the register RAM has 16 registers.
- 8) Why are the delayed branch instructions used in some RISC processors and why are they overrided in other processors?
- 9) For which features was the MIPS processor named as the Microprocessor without Interlocked Pipeline Stages ?

1.9. Interrupt system

1.9.1 Concepts and purpose of the interrupt system

In the 60s of the last century, the introduction of the interrupt system in the computer architectures was an extremely important step in the development of the computer technology. Through the use of an interrupt system, it was possible not only to compute the response to external events but also to implement automatic control of the computing process by the operating system, as well as to implement a number of parallel computational processes in a single processor.

The interrupt system is a hardware and software system, which provides the computer response to a variety of events (interrupts) that are internal or external to the computing process. This reaction is performed as the transition to the instruction sequence, that calculates it and as the return to the interrupted process. To understand precisely the essence of the interrupt system the next common definitions are introduced.

The dispatcher is an operating system software which is used to control the interrupt signal processing, to analyse the interruption causes and storing the information, which is necessary for the restoration of the interrupted program operation. The dispatcher is also engaged in the dynamic allocation of tasks or software threads between computing resources.

The real time processor operation is the implementation of user tasks in the processor so that the user requests (interruption signal) were fulfilled immediately or with a permissible delay. Here, the user is considered not only the computer operator but also the controlled object (if the computer is included in the control system circuit), peripheral device (eg., disk drive controller), resident process (for example, the keyboard driver service, WEB-browser), and the like.

The time-sharing processor operation is the simultaneous processing of multiple processor tasks, which are initiated by different users. However, users are waiting for responses to their requests in coordination with the work in real time. That is the processor "shares" its CPU operation time between users.

Thus, the real-time operation requires that the service of the i -th type interrupt request requires the minimum permissible time delay T_{Pi} considering that multiple requests may occur simultaneously. For example, when the interrupt signal is inputted from the keyboard, then T_{Pi} can be less than 0.1 s but the reading from the magnetic hard disk requires $T_{Pi} = 10 \mu\text{s}$. When a set of interrupt requests with the equal priority occur simultaneously, then these requests are queued. This queue can be serviced for the average time T_Q . Then if the computer works in real time, then the requirement to its performance is $T_{Pi} \leq T_Q$.

1.9.2 Interrupt signal classification

Hundreds of thousands of signals that cause the interrupts may be involved in the computers. All these signals are not equal to each other, and they require the different algorithms of their service. The classification of the interrupt signals is below.

- 1. Signals about the events that occur in the computer*** (both in the hardware and in programs). They are the signals of failures in a variety of devices, which are fixed by the special detectors. For example, this is the signal of the reducing of the supply voltage, which causes the interrupt of the state saving before the computer is turned off. When the failure occurs in the transmission channel or by the memory reading, an error interrupt program is called, which retransmits the data or repeats the memory reading.

The software interrupts are associated with certain events, which are caused by the program implementation, for example, with the errors in the programs. They are caused by the signals like incorrect instruction code, incorrect addressing, division by zero, number overflow, impaired memory protection conditions, the use of the prohibited instructions in the program (for example, privileged instructions).

The programmable interrupt is the software interrupt that is caused by the implementation of special instructions (**traps**), which are intended for the detection of certain specified conditions. For example, such an interrupt is executed when debugging or by executing instructions that are not implemented in the hardware. When the user program makes the access to the OS resources via the interrupt instructions, such interrupt is called the programmable interrupt as well. For this purpose, the interrupt instruction INT 21h is widely used in the architecture I80x86, for example, for the input-output operations.

2. **Signals from the timers.** Such signals are used to arrange the real time operation of the computer in the time division mode or when the computer controls the objects. Such a timer is the clock IC or the pulse generator. To prevent the program deadlock, the tracking sensor named as a **watchdog** is used. When the main program does not load periodically the watchdog counter, then it is considered to be deadlocked. Then the watchdog counter, which is overflowed in some period of time, interrupts the computer, and it falls in the restart mode.

The software timer is a memory cell that is incremented by OS and cause the interrupt in the case of equality of its content to a given value. For example, such a timer controls the planned running of some programs like updating the software.

3. **Signals from the input-output devices.** These signals are generated by the synchronization signal receiving from the input-output devices. Such

signals indicate the readiness, failure of the peripheral device, or the operator's inference to the computer operation. The specific interrupt routine corresponds to a particular I/O device, and it reacts to these interrupt signals. Such a routine is called a ***driver*** of the I/O device.

4. ***Request signals.*** These signals arrive into the computer from various external sources. In the control systems, they are the signals of events in the controlled objects. In the multiprocessor systems, they are the synchronization signals from other processors.

1.9.3 Interrupt system algorithm

The interrupt register and the priority encoder are specific elements of the interrupt system. The first of them stores the interrupt flags and second one selects the interrupt source. They are defined as follows.

The interrupt flag is a hardware element, which latches a separate interrupt signal. It is set to 1 when the interrupt signal occurs and is set to 0 after the interrupt service.

The interrupt register is a register, combining the outputs of all the interrupt flags. Unlike the ordinary register, this one changes the states of its bits independently at arbitrary moments of times.

The priority encoder is a hardware or hardware-software unit that decides which of the fixed interrupt source should be served first.

The common interrupt handling algorithm is shown in Fig.1.25. The following actions are performed during the algorithm implementation.

- 1) One or more bits of the interrupt register capture the interrupt signal presence.
- 2) The program stops its execution. The moment of the permitted interrupt can be set by one of these methods: the instruction code indicates

whether it is possible to interrupt this instruction; interrupt is possible at the end of any instruction; interruption in any cycle of the instruction.

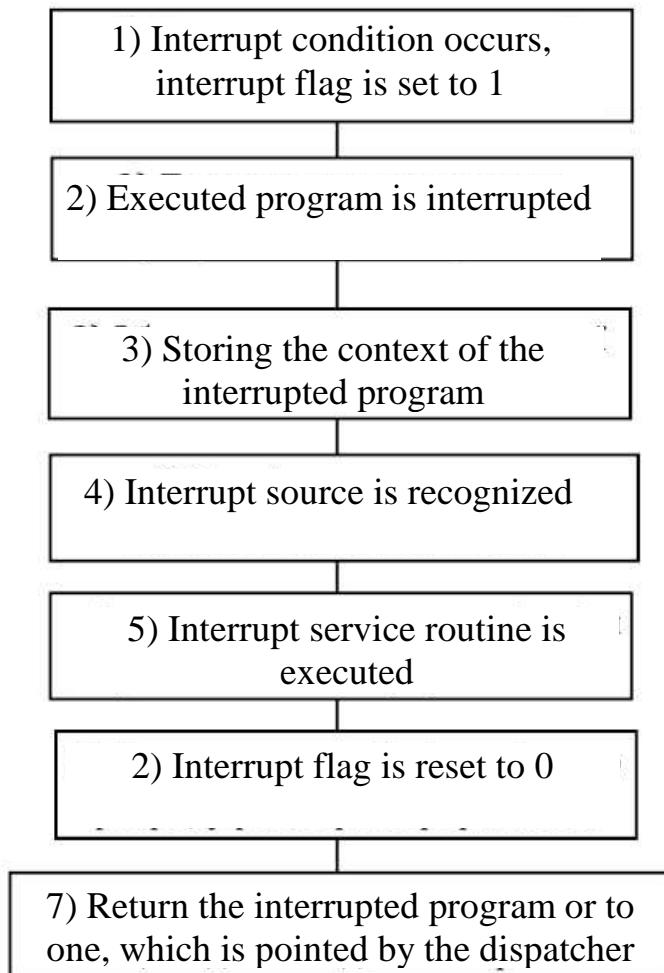


Fig. 1.25. Interrupt service algorithm

In the RISC processors, when the interruption occurs, the instruction pipeline must be stopped and the states of its stages must be stored, or the interrupted instruction must be computed to its end, not to load the next instructions. That is, in the RISC processors, the interrupt handling mechanism is much more complicated.

3) At the time of interruption of the program, the states of the main registers are stored in order to restore correctly its operation. The data in these registers are named as the ***program context***.

4) The cause of the interruption is determined and the interrupt handling program is selected for a particular interrupt flag. If there are several

interrupt sources (hundreds or thousands), then the hardware selects a group of sources, using the priority encoder, and the particular interrupt source is determined by the software.

If there are several interrupt requests, they should be served in the manner specified by the priorities. The decision, which of interrupt signals to serve first, makes the priority encoder under the dispatcher control. The discipline of the interruption service, which is incorporated in the dispatcher and the priority encoder, sets the sequence of handling multiple interrupt flags in the interrupt register.

5, 6) The implementation of the identified interrupt routine. Resetting the corresponding interrupt flag, i.e. the cause of the interruption is eliminated.

7) The dispatcher provides a continuation of the interrupted program by using the recovery of the processor register contents or starts handling the next interrupt request (depending on the circumstances, the interrupt service disciplines, etc.).

1.9.4 Methods for storing of the interrupted program

When using the software method of the interrupt handling, the contents of all registers, i.e. the program context, is stored by the dispatcher in a separate memory segment. The contents return to the registers is also performed by software.

To speed-up this process, the storage area is made as the high speed RAM. This process also speeded up, if not to save the state of all registers, but only those that can be changed at the interrupt handling.

By the hardwired saving method, the contents of registers are automatically rewritten to the stack memory or to the specialized memory segment. In the I80x86 architecture, during the interruption, more than 104

bytes of data from the main registers are automatically saved in the task state segment (TSS). In many RISC-systems, the data and address registers are duplicated. And during the interrupt, one group of registers, that store the context of the interrupted program, it switches to another group of registers, which are used for the interrupt processing. The advantages of this method are in high speed. But the shortcomings are the lack of flexibility, limited number of parallel interrupts, increased hardware expenses.

In the hardware-software method, the basic registers (accumulator, state register, program counter, etc.) are saved by the hardware, and the rest of the context is save by the software.

1.9.5 Main characteristics of the system interrupts

interrupt system determines the overall computer performance. Therefore, to optimize the architecture for speed, it should determine the characteristics and parameters of the interrupt system. The timing diagram of the interrupt handling process is shown in Fig. 1.26.

It identifies the following time characteristics of the interrupt system:

$T_{R\text{and}}$ — reaction time;

$T_{S\text{Si}}$ — state storing time;

$T_{D\text{and}}$ — time for the interrupt source discrimination;

$T_{P\text{rand}}$ — preparation, response time, is equal to the absolute delay of the interruption, and characterizes its speed;

$T_{I\text{P}\text{and}}$ — interrupt processing time;

T_{li} — common time of the interrupt service.

The derivative parameter is the loss factor $K_L = T_{I\text{P}\text{and}} / T_{li}$ which characterizes the relative performance of the interrupt system.

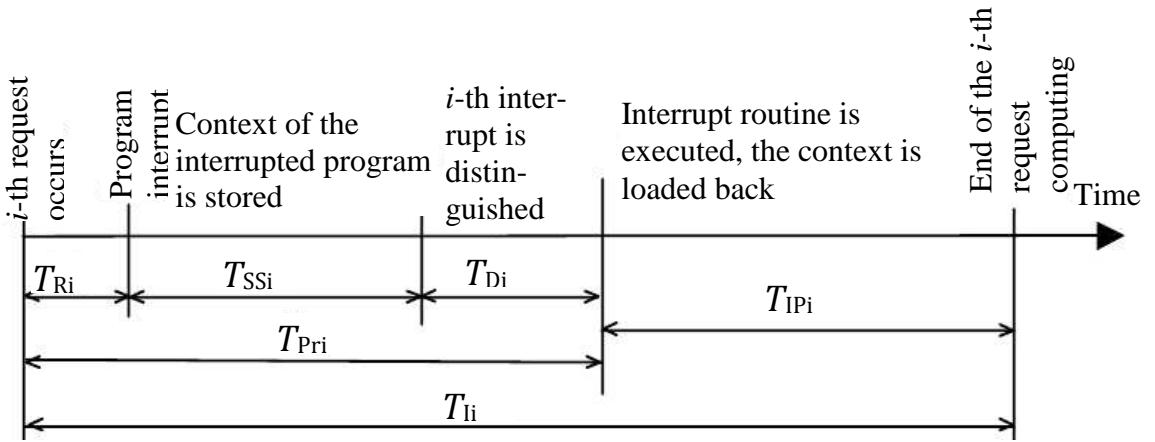


Fig. 1.26. Timing diagram of the interrupt service

The interrupt system saturation is the state, when the flow of interrupt requests could not be served in time, that is, when $T_{li} > T_{Pi}$.

1.9.6 Interrupt service disciplines

The interrupt system should be able to allocate one interrupt flag among the plurality of simultaneously occurred flags according to the established discipline of interrupt service. The interrupt requests may have varying degrees of importance, depending on the particular circumstances, which may vary. Ideally, the system allocates a request (flag), which is the most important. The **optimum service** is a service when the interrupt system correctly and quickly responds to the importance of requests and switches the processor to process the most critical situation.

In computers, the following interrupt service disciplines are the most common.

1. The discipline of service **with the cyclic queue** is a cyclic polling of all flags of the interrupt register. The first not zeroed flag is serviced. After processing of the i -th flag, the $i + 1$ st flag is tested and is serviced if it has the

value one. If the polling time is sufficiently small, a synonym of this discipline is ***first-come, first-served***.

2. The discipline of service ***with the relative priorities***. At this discipline, the most important request is selected from a plurality of queries in accordance with the assigned priorities. If during its handling even more important request (with the higher priority) comes, it waits until the end of this request processing. The next request, which is handling, is the most important request in the moment of time of the previously processed request finishing.

3. The service discipline ***with absolute priorities*** means that each time the most important request is processed, no matter how many less important requests are waiting for their processing. That is, the interrupt of another interrupt is possible.

The priorities are assigned to either individual requests or groups of similar requests. The highest priority is given to the hardware failure interrupts, then to the timer interrupt. The requests from the high-performance units are given the higher priority than the requests from the slow units.

If the computer is running with ***dynamic priorities***, the priorities can be re-evaluated and reassigned depending on the situation.

The interrupt protection is a mechanism for the temporary disabling of the interruptions from the selected sources. It is performed using the ***interrupt mask register***. The code 0 in the i -th bit of this register disables the i -th interrupt flag. Such a protection is used to prohibit the interruptions during the critical application running, such as recording in the HDD or the privileged task of the operating system.

As a rule, a separate bit of the mask register controls the disabling of all interrupts, that is a bit of a ***global interrupt enabling***.

1.9.7 Interrupt system classification

The most characteristic feature of the interrupt system is a rule about the organization of choice for service the interrupt request. On this basis, these systems are divided into the systems with:

- polling (eg., cyclic or priority selection);
- an absolute priority;
- the relative priorities;
- the mixed priorities.

Interrupt source selection systems are divided to the systems with:

- the software implementation of the priority scheme;
- the hardware implementation of a priority scheme;
- the hardware and software priority scheme;
- the multi-staged interrupt system, when there are a lot of interrupt sources.

The interrupt register fixes the request in the flag p . The code in the interrupt mask register masks some or all the bits of the interrupt flags. The priority encoder selects the highest priority number p among several established and unmasked flags and generates the common interrupt request signal. The selected interrupt number p to be serviced is used in the formation of the input address, which is fed in the ***interrupt vector table***. This formation consists in adding the base address of the interrupt vector table to the scaled code p , i.e. the code $2^k p$.

1.9.8 Hardware implementation of the interrupt service

Hardware implementation of the interrupt — is an essential condition for ensuring high performance system interrupts. A typical hardware implementation of the interrupt circuit shown in Fig. 1.27.

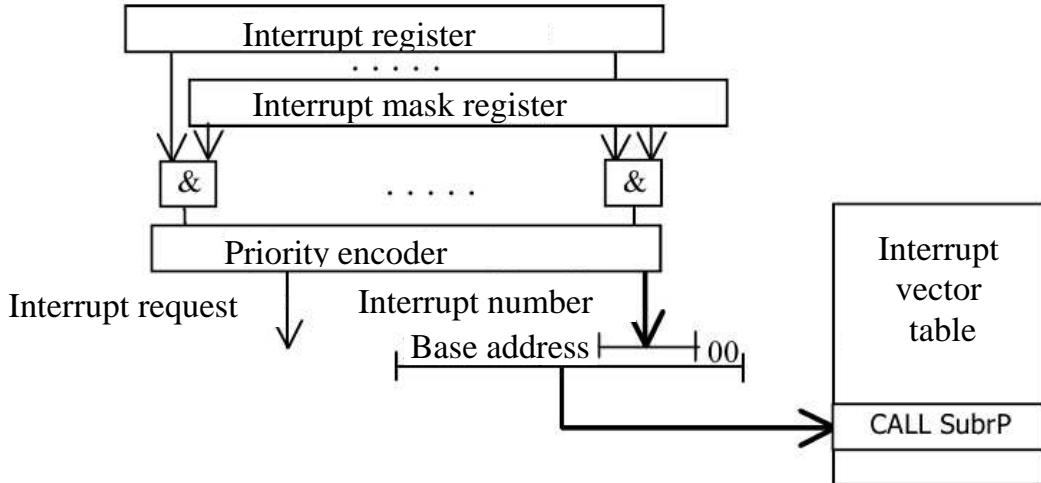


Fig. 1.27. Hardware implementation of the interrupt service

The p -th row of the interrupt vector table stores the short 2^k -byte routine of the interrupt service, the instruction of the interrupt routine call or the absolute address of the routine. Such a routine is located in a residential area of the OS. In the latter case, the interrupt subroutine call is implemented in hardware, and the address of the subroutine code is, in fact, the ***interrupt vector***. After the end of the interrupt subroutine, the interrupt return instruction returns the control to the program, which was interrupted. The effect of this instruction is different from the action of the usual return instruction, at least, by the fact that after it the p -th flag, which caused the interrupt, is automatically reset to 0.

In the architecture I80x86, when working in real mode, the base address of interrupt vector table is zero. The table stores the addresses — the 4-byte interrupt vectors. The number of vectors is 256. Table 1 shows the sources of standard hardware interrupts I80x86 architecture.

When working in protected mode, the architecture I80x86 has the interrupt descriptor table. Each descriptor in it contains not only the entering address to the interrupt routine but the information about the access

rights to it which serves to the resource protection. More detailed description of the interrupt service in this architecture is represented in the second part of this teaching book.

Table 1. DOS hardware interrupts

Nº	Interrupt source
IRQ 0	System timer
IRQ 1	Keyboard controller
IRQ 2	Video clock signal, in connection with 9-th
IRQ 3	COM2 / COM4
IRQ 4	COM1 / COM3
IRQ 5	Free
IRQ 6	FDD controller
IRQ 7	LPT1
IRQ 8	Real Time Clock
IRQ 9	Video clock signal, in connection with 2-nd
IRQ 10	Free
IRQ 11	Free
IRQ 12	Mouse controller PS / 2
IRQ 13	Mathematical coprocessor
IRQ 14	Controller IDE HDD (first channel)
IRQ 15	Controller IDE HDD (second channel)

1.9.9 Interrupts in the I8051 microcontroller

The 8051 provides 5 interrupt sources. The external interrupt signals

INT0 and INT1 enter the chip through the port pins P3.2 and P3.3. They can each be either level-activated or transition-activated depending on bits IT0 and IT1 in the Timer Control register TCON (Fig.1.28). These signals are fixed in the flags IE0 and IE1 of the register TCON. The overflows of the Timer 0 and Timer 1 are fixed in the flags TF0 and TF1 of the register TCON. The Serial Port generates the interrupt signals RI and TI just after receiving and translating a data byte, respectively, which are fixed in the Serial port Control

register SCON (Fig. 1.28). These signals are ORed and then considered as the interrupt flag RI+TI.

So, the bits IE0, IE1, IT0, IT1 and RI+TI form the flags of the interrupt register. They are set by the interrupt signals and reset after the interrupt computing by the hardware. They can be set or reset by the user program providing the software interrupt as well.

Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in the SFR Interrupt Enable register IE (Fig.1.28). Then the bit ES enables the serial port, ET1 does the first timer, EX1 does INT1 and so on. IE contains also a global disable bit, EA, which disables all interrupts at once.

Each interrupt source can be individually programmed to one of two priority levels by setting or clearing a bit in the SFR Interrupt Priority register IP. A low priority interrupt can itself be interrupted by a high-priority interrupt but not by another low-priority interrupt.

TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
IE	EA	—	—	ES	ET1	EX1	ET0	EX0
IP	—	—	—	PS	PT1	PX1	PT0	PX0

Fig.1.28. Registers TCON, SCON, IE and IP

If two requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If the requests of the same priority level are received simultaneously, an internal polling sequence

determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence, as in Fig.1.29.

Nº	Source	Priority within level	Vector address
1	IE0	highest	0003h
2	TF0		000Bh
3	IE1		0013h
4	TF1		001Bh
5	RI + TI	lowest	0023h

Fig. 1.29. Interrupt flags, their priorities and vector addresses

The interrupt flags are sampled at the end of every machine cycle. The samples are polled during the following machine cycle. If one of the flags was in a set condition at the end of the preceding cycle, the polling cycle will find it, and the interrupt system will generate an artificial LCALL instruction. This LCALL instruction calls the appropriate service routine. But it is blocked by any of the following conditions:

- an interrupt of equal or higher priority level is already in progress;
- the current (polling) cycle is not the final cycle in the execution of the instruction in progress, i.e. the instruction must be finalized before the interruption;
- the instruction in progress is RETI, or any write to the IE or IP registers, i.e. by this conditions, at least one next instruction must be executed before the interruption.

The polling cycle is repeated with each machine cycle. Note, if an interrupt flag is active but not being responded to for one of above conditions, and is cleared before this blocking condition is removed, this interrupt will not be serviced. In other words, the fact, that the interrupt signal was once

active but not serviced, is not remembered. The polling cycle and LCALL sequence are illustrated by Fig.1.30.

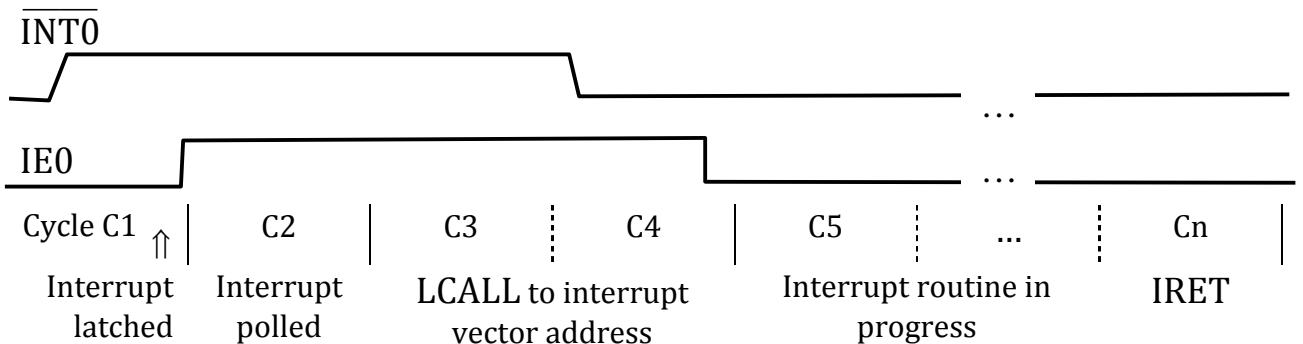


Fig. 1.30. Interrupt response timing diagram

The hardware-generated instruction LCALL in most of cases also clears the flag that generated the interrupt. But it never clears the RI+TI flag. This has to be done in the user's software. It clears an external interrupt flag IE0 or IE1 only if it was transition-activated. The LCALL instruction pushes the contents of the program counter PC onto the stack and reloads it with an address that depends on the source of the interrupt being vectored to, as shown in Fig. 1.29. Note, that LCALL does not save the PSW or other program context. This task must be done by the interrupt software.

This software execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the control hardware that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the PC. The execution of the interrupted program continues from where it left off.

So, if an interrupt request is active and conditions are right for it to be acknowledged, the hardwired LCALL to the requested service routine will be the next instruction to be executed. It takes two cycles. Therefore, a minimum of three complete machine cycles elapse between activation of an external

interrupt request and beginning of execution of the first instruction of the service routine, as shown in Fig. 1.30.

The interrupt vector table (see Fig.1.27) consists of sets of eight memory cells. This volume is enough to put there a simple interrupt subroutine, which, for example, updates an event register-counter. If the longest subroutine is programmed then the jump instruction is placed in the vector table cell, which performs a jump to the long subprogram.

A longer response time would result if the request is blocked by one of the 3 previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time depends on the nature of the other interrupt service routine. If the instruction in progress is not in its final cycle, the additional wait time cannot be more than 3 cycles, since the longest instructions, like MUL, DIV, are 4 cycles long. If the instruction in progress is RETI or an access to IE or IP, the additional wait time can not be more than 5 cycles (one to complete the instruction in progress plus 4 to complete the next long instruction like DIV). Thus, the response time T_{Prand} is more than 3 cycles and less than 9 cycles (see Fig. 1.26).

1.9.10 Conclusions

The operating system media contains only the basic interrupt routines including ones for the drivers of the most popular I/O devices. But the system programmers often need to develop a special program for the specific interrupt service to the computer, providing the optimum performance, functionality, when attaching the specific modules. Therefore, they need to know and be able to use and modify the interrupt system of the target architecture.

1.9.11 Problems

- 1) The interrupt system distinguishes the computer speed. Prove it.

2) Three interrupt requests have the permissible delays of their service, which are equal to 10, 20, and 30 ms, respectively. What is the average interrupt service time if the computer is working in real time?

3) Why is the interrupt from the watchdog not the usual interrupt but does cause the reset mode?

4) Give the examples of the interrupt sources which need the priority services.

5) What are the purposes of the driver program?

6) How to arrange the processor architecture, which provides the long instruction execution interrupt?

7) What data represent the program context of the I80x51 architecture? In which registers are they stored?

8) The program context of some program of the I80x51 architecture is stored in 8 data registers and in PSW. How much time is needed to save and restore this context?

9) Why is the first interrupt vector address in the I80x51 architecture equal to 0003h ?

10) Calculate the maximum average frequency of interruptions per second of the microcontroller i8051. The clock frequency is 60 MHz. The interruption of the first type is executed by 2 jump instructions and 1 data transfer instruction. The interruption of the second type is executed by a single jump instruction. The likelihood of the first type interruption is equal to 90 %, while of the second type — 10%. The average interrupt execution time must not exceed the average execution time of the main program.

11) The i8051 microcontroller operates at 60 MHz clock frequency.

The impulses of the frequency F_s enter its input $\overline{\text{INT}0}$. What is the maximum frequency F_s of the impulses, which can be counted by the two-byte counter programmed in the microcontroller?

2. PROCESSOR ENVIRONMENT ARCHITECTURE

2.1. Interface basics

2.1.1 Data movements in the computers and interfaces

The first generation computers had a centralized structure in which all the blocks, such as memory, magnetic disks, tapes, drums, input-output terminals were attached directly to the central processor unit (CPU) and were working under its control. Since the speeds of the CPU and peripheral devices were relatively small, the CPU spent a greater part of its time on data exchanges with peripheral devices. In the development of the second and third generations of computers, it became clear, that the high-speed CPU does not perform useful work most of the time, serving the slow peripherals. An idea occurred to share the tasks between a CPU and peripherals, and allow them to work more independently. This problem was solved by introducing the concepts of the system interface and the system interrupts.

The common bus interface was first introduced in the PDP-8 computer in 1965. The **common bus** made it possible to operate the CPU and peripheral devices independently, as well as to send data to each other. In this interface, all devices are connected to the same bus in the same way through a standard hardware interface. These devices should be subject to the same rules when sending signals to the bus and receiving them from it. These rules are called the **interface protocol**.

With the introduction of the multiplexed and selector interfaces (channels), the IBM-360 architecture has allowed the unified attachment of ten different models of the CPU with four dozen different types of peripheral devices as desired. This was one of the reasons for the high commercial success of this architecture.

Thus, the history of the first three generations of computers has shown the need for interfaces. It allowed:

- efficient use of the CPU and peripheral devices;
- changing the structure of the computer (and sometimes even during operation), depending on the needs;
- through the standardization, to develop and produce a variety of computer devices at an arbitrary time by different manufacturers, as well as complete the computers optimally.

The interface is considered as the hardware means, which are serving for the uniform connection of several blocks, and provide reliable communication therethrough and connected these blocks to the system. The interface description consists of descriptions of its hardware and protocol.

The hardware part of the interface is described as a structure comprising a set of hardware connectors, units, and links between them, as well as the requirements for connectors and electrical signals, which are distributed in the links.

The communication link is called as the **interface bus**. Its description, as a rule, is the interface connector description. Also, there are physical limitations on the design of the bus, i.e. the design of its wires and their maximum length. The other physical limitations apply to the design of the bus signal generators and receivers. These are the levels of voltages and currents in wires that represent bits of information, as well as the timing diagrams of the signals in them.

The interface bus consists of the address bus, data bus, control bus, power bus and the reserve wires.

The interface protocol is a system of rules to handle with the interface in time. They are the rules for the interface connection, arbitration, and transmission of data in it. The protocol is given as an algorithm of sending

(receiving) the signals through the address bus, data bus, as well as of the power management.

2.1.2 Common bus, masters, and slaves

The common bus is one of the simplest interfaces. The access to it is shared among all devices connected to it. Its advantages are low price and versatility. At the same time, only a single data source can be connected to it. Among the bus sources and destinations are the masters and slaves. **The master** is the initiator of the data transfer. It sends a request to the interface for one act of using the interface (**the transaction**) and after the request confirmation, it seizes the interface control. **The slaves** are the separate units, which are connected to the interface and perform the instructions of the master like reading or writing a single data word or a data set for the given address.

After the reading or writing operation, the master is disconnected from the bus and finishes its management, i.e. it completes the transaction. If the common bus is connected to the multiple masters, the arbiter unit is used in this interface. **The arbiter** decides, whom of the masters to give the control when the simultaneous requests occur from multiple masters.

The main drawback of the common bus is its limited throughput, since it is impossible to perform the transactions of several masters at the same time. This drawback is removed in more complex interfaces. But the categories of the master, slave, arbiter and transaction are used in the most of computer interfaces.

2.1.2 System and local interfaces

Traditionally, the interface busses are divided into the busses, which connect the processor with the memory, and input-output (IO) busses. The **IO**

bus may be longer, to maintain the connection the devices of many types. As a rule, such a bus corresponds to one of the standard interfaces. The **processor — memory bus**, on the other hand, is relatively short. It is often specialized and therefore, it provides the maximum bandwidth. At the stage of development of the computer system, all the types and parameters of devices, which are connected to the processor — memory bus, are known in advance. Therefore, the maximum required bandwidth is provided. However, the input-output bus developer has to deal with devices that have different, often unknown delays and other parameters.

In order to reduce the hardware cost, some computers have a single bus both for the memory and input-output devices. It is often called as a **system bus**. The personal computers of the first generations were based on a system bus in an ISA or EISA standard.

The need to maintain a balance between the speed of CPU and throughput of IO devices has led to a two-staged organization of the busses in the personal computers based on the local bus. The **local bus** is electrically connected to the CPU core. It integrates the processor, memory, buffer to the system bus, as well as some auxiliary circuits. Typical examples of the local bus is VLBus and PCI.

2.1.3 Bus implementations

Consider a typical bus transaction. The bus transaction has two parts: sending the address by the master to the slave, the reception (or transmission) of the data.

The bus transactions are defined by the interaction with the memory: the "read" type transaction transfers the data from memory (CPU or IO device), the "write" type transaction writes data to the memory or in the register in a peripheral device. In the "read" type transaction, firstly, the

memory address together with control signals are transferred, indicating that it is the read transaction. Then the memory responds by the outputting the required data to the bus with the appropriate control signals. During the "write" type transaction, the CPU sends to the memory the address and data and usually does not wait for the completion of the recording.

Two types of the interface busses are distinguished depending on the switching method: the circuit-switched bus and the packet-switched bus.

In the ***circuit-switched bus***, the master requests the bus, after the arbitration sets the data address, and blocks the bus until the end of the request service. The bus switching system makes a route from the source to the destination for the whole transaction. But most of the time, spent in this case, may be the memory delay. That is, the circuit-switched bus may be used inefficiently.

When transmitting data messages via the ***packet-switched bus***, the data packet is divided into two or more packets, which are transmitted sequentially. Moreover, the packet forwarding routes may not be the same. That is, several packets can be forwarding simultaneously, which provides the high network throughput.

The minimum message consists of address and data portion. The data transfer transaction can be divided into two parts: a bus request with the address forwarding and reply with the data transfer. This technique is called as the ***split transaction***. So, the read transaction is divided by the request subtransaction, containing the address, and the memory response subtransaction, containing the data. Each transaction must be marked (tagged) for the purpose that the CPU and memory can recognize the transaction type.

In the advanced busses, the long data transfer transaction can be splitted to a set of subtransactions. Between them, the short transactions with the higher priority can be hold.

The bus is often performed as the ***pipelined bus***. The pipeline registers are set along the data and address transfer routes in such busses. Due to this, the bus clock frequency can raise to the extreme values. The split transactions can fully load such a bus providing both the high throughput and the high hardware effectiveness. But the latent delay, i.e. the delay from the beginning to the end of the transaction, is increased in such busses.

If the ***bus*** is ***synchronous***, it includes the synchronization signals in the bus lines and the fixed control protocol, which specifies the location of address signals and data signals with respect to the synchronization signal.

Fig.2.1 illustrates the interaction of a master and a slave in the ***synchronous bus***. The master generates clocks C and sends them through a separate line to all the slaves. It sets the slave address on the bus AB strobing it by the address acknowledge signal AAK, and then it sends the request signal RRQ for data reading or writing. AAK and RRQ can be combined in time. The slave outputs the data in the data bus DB as a response to the master's signals. The synchronous mode in this example consists in that, that all the signals have to appear precisely in accordance with the common clock signal.

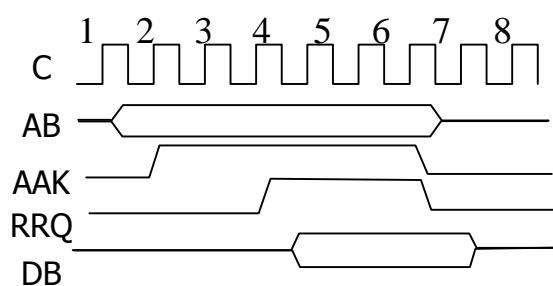


Fig.2.1. Synchronous bus timing

Such a bus can be quick and cheap. But it has two disadvantages. Firstly, all the acts in it must be performed with the same synchronization frequency, i.e. with the maximum speed of this bus. If the slave could not output the data in time, then it has to activate the line "wait" and deactivate it if the data is

ready.

Secondly, because of the phase skew of the signals, occurring in adjacent bits of the parallel bus, the synchronous bus can not be long, and it could not be connected to many devices. In general, the processor — memory bus is synchronous one.

In the modern high-speed serial busses, such as PCI-E, SATA, USB, Ethernet, the synchronization signal is implicitly specified in the data signal, and it is reproduced precisely in the special receiver circuits of the bus. Therefore, such busses can be considered as synchronous ones.

The **asynchronous bus** is not clocked. It uses a start-stop transmission mode and the handshaking protocol. By the handshaking, the receiver after the message reception sends an acknowledgement signal back to the transmitter, which reports that this receiver has received the message and is ready to receive the next one. This scheme makes it possible to easily adapt the devices to different performance to the bus and increase the length of the bus wires without the synchronization problems.

Fig.2.2 illustrates the interaction of units attached to the asynchronous bus. The master unit outputs address to the line A and the reading request RRQ. The last one is simultaneously an **acknowledge** signal of the address correctness.

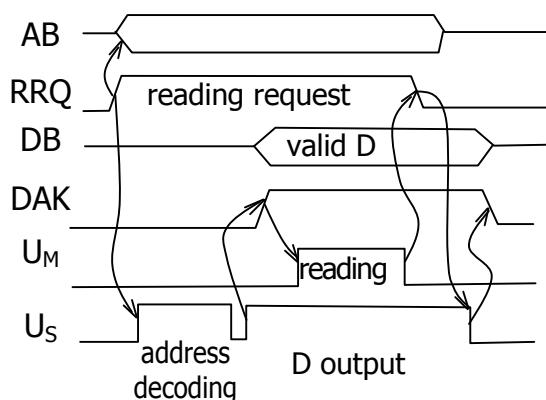


Fig.2.2. Asynchronous bus timing

The master activity is marked as U_M . All the slave units, which activity is marked as U_S , decode the address. Only selected unit, which address satisfied, outputs the data D to the bus DB , and it acknowledges the data correctness by a signal in the line DAK . This signal strobes the data input in the master.

After data receiving, the master resets the signal RRQ signaling that the data is already not affordable. By this signal, the slave releases the bus, setting it in the state of the high impedance, and it resets the signal DAK . It shows that the reading transaction is finished.

The examples of the asynchronous bus is the serial bus COM, as well as the bus PS2, which are running in a start-stop mode. The source is transmitted a single byte as the 10-bit character. The first and the last bits are the start of the character (zero) and the stop of it (one), and the rest bits are the byte bits, starting with the least significant bit (LSB). The duration of each signal pulse in the bus is determined by a predetermined symbol transmission frequency. This frequency is set the same in all the COM interfaces with an error of not more than 3%. For example, at 9600 baud rate (i.e., 9600 symbol bits per second) the pulse duration is equal to $1/9600$ s or 104 microseconds. That is, the maximum transfer rate is equal to 960 bytes per second.

The synchronous bus usually has the increased bandwidth comparing to the asynchronous one due to the lack of the synchronization overhead (acknowledgment), and to the possibility of the pipelining. The bus type selection (synchronous or asynchronous) affects not only the throughput but also the capacity of the input-output system (the bus length and the number of devices that can be connected to the bus). Consider next the common system busses, which had improvement with the development of the personal computers.

2.1.4 System bus

One of the most popular PC busses was the system bus **XT-Bus**, i.e. the XT architecture bus, which was installed in the first personal computers IBM PC. It supports the moving of 8-bit data in the 20-bit address space and operates at the frequency of 4.77 MHz.

ISA (Industry Standard Architecture) is the main bus of the PC AT-type computers. It is the extension of the XT-Bus bus. Its bitwidth is 16/24 (16-bit bus and 16 MB address space), the clock frequency is equal to 8 MHz and the maximum bandwidth is 5,55 MB / s.

EISA (Enhanced ISA) is a functional and meaningful expansion of the ISA bus. The bit width is 32/32 (32-bit bus, address space is 4GB), clocked at 8 MHz. The maximum throughput is 32 MB / s. In contrast to the previous busses, which are only controlled by the processor, this bus supports the **Bus Mastering** mode.

This mode provides control of the bus by the arbitrary device connected to the bus, due to the arbitration system and the possibility of the IRQ channel separation and the **direct memory access (DMA)** mode. This mode is used to release the processor from the forwarding operation commands and / or data between two devices on the same bus. For example, in the DMA mode PC AT architecture processor performs DMA controller, which is common to all devices. When the Bus Mastering, each device-master has its own controller.

VLB (VESA Local Bus) is a 32-bit addition to the ISA bus. Its clock speed is 25—50 MHz, the maximum transfer rate is 130 MB / s.

PCI (Peripheral Component Interconnect) is the modification of the VLB. Its expanded version has the bitwidth 64/64 and clock speed reaches 66 MHz. This provides the throughput up to 528 MB / s. The bus is divided to the segments which are connected via the **bridges**. The number of connections to

a single segment is limited to four. The segments may be connected with a different structure topology (tree, star, etc.). At present, it is the most popular bus system.

PCI-X, i.e., **PCI eXpanded**, is the high-speed parallel bus with a packet switching mode, which is the modernization of the PCI bus. The PCI-X device can be inserted into the PCI connection and vice versa, since these interfaces have the interchangeable hardware and software. The signals in the PCI-X pass through the pipelined registers and the signal edges are reproduced by a **phase locked loops (PLL)**, thereby obtaining a high data rate. The throughput of this bus is limited by 1064 MB / s. Such a throughput is achieved when the most of transactions are the block transfers, which length is higher than 128 bytes.

The system that requires a speed of more than 1 GB/s, is implemented in the version PCI-X 2.0, which implements the protocols **Dual Data Rate (DDR)** and **Quad Data Rate (QDR)**. PCI-X 2.0 has the clock frequency of 133 MHz. Thanks to the additional pulse strobes, two (DDR) or four (QDR) data are transferred per clock cycle. Due to this the throughput reaches up to 4256 MB / s. But at the high speed the bus user number is reduced to two, which makes the bus connection like **point to point**. The additional generation and verification of the control bits allows to correct the errors during the transfers, which allows to maintain both the high speed and high reliability.

PCMCIA (Personal Computer Memory Card International Association) is the external bus for the NoteBook class computers. The another name for the PCMCIA module is **PC Card**. This simple bus supports the address space up to 64 MB, the automatic setup and the possibility to connect and disconnect the devices while the computer.

2.1.5 peripheral interfaces (IO busses)

AGP bus. The PC busses which output the image data are constantly being improved. Thus, an accelerated graphics port (AGP) is an extension of the PCI bus to process the large data sets of the 3D-graphics. Intel has developed the AGP to solve two problems. Firstly, for the playback of the 3D-graphics, as much memory is needed for the texture map storing and the z-buffer handling. Under normal circumstances, the z-buffer contains the information about the three-dimensional image and uses the same memory, as the texture maps. This creates a conflict with the CPU. Secondly, the graphics performance and memory throughput are limited by the physical characteristics of the PCI bus.

The AGP bus performs the point to point connection between the graphical subsystem and the system memory, and the subsystem shares the memory with the CPU.

USB bus. USB (*Universal Serial Bus*) is designed to provide the data exchange between the computer and the connected peripheral device under the dynamic (hot) system configuration changes. When designing this interface the next features were taken into account:

- easy system configuration changes;
- low cost solutions;
- adaptation to the existing infrastructure and PC software;
- stimulate the development of new classes of PC devices.

The USB capabilities come from its technical specifications:

- high exchange rate of 12 Mbit / s;
- maximum cable length for the high exchange rate up to 5 m;
- maximum number of connected devices (including hubs) is 127;
- possibility to connect the devices with different throughput rates;
- the supply voltage through the bus for the peripheral devices is 5 V;

- the maximum supply current to the device is 500 mA.

At the physical layer, the bus topology has the tree structure like the multi-level star. At the root of the tree is the root hub, which provides the communication of the peripherals with the computer (host). The host has a set of point to point connections to other nodes of the tree.

The developers of the unusual peripheral units should include their drivers in the level of the USB bus software. The configuration and identification software system is supplied by the OS developer or other software vendor. This system controls all the network nodes. The USB driver from the OS developer performs the scheduling of the bus activity.

The USB bus utilizes the ***time division multiplexing (TDM)*** to provide the parallel access to several devices. The data are transmitted in the packets. The packet size is variable and depends on many factors. The host controller combines the packets, which are transmitted to multiple devices, in the frames with a duration of 0.001 s.

The USB-2 bus is the upgrade of the USB bus with data transfer speeds of up to 480 Mbit / s. Now the third generation of the USB bus, i.e., the USB-3 is gradually introduced, which speed is increased in ten and even twenty times depending on the version.

IDE and SCSI busses. Under the term **IDE** (Integrated Drive Electronics), which is the synonym of **ATA** (AT Attachment) a simple and cheap interface to the PC AT computer is understood. The IDE devices are usually the HDD or other high volume peripheral memory devices. A built in controller provides the functions of storage management, and the 40-bit cable performs the simplified segment of the AT bus (ISA). The simplest IDE adapter has only the address decoder, and other signals are repeated from the ISA slot.

The main IDE mode is the programmable input-output (PIO) under control of the CPU. But all the HDD support the data moving in the DMA mode using the bus mastering.

Under the term **SCSI** (Small Computer System Interface), the standards developed by the American National Standards Institute (ANSI) is understood that sets the mechanism for implementing the data transmission line between the CPU system bus and the peripheral devices, like HDD. The main its feature is the placement of an intelligent SCSI controller in the peripheral subsystem. The SCSI-1 standard peripheral bus is based on the 50-wire cable.

PCI Express, shortly, **PCI-E** is the third generation of the input-output busses for the peripheral device connection. Remember, that ISA, EISA, VESA and Micro Channel busses belong to the first generation of such interfaces, and PCI, AGP and PCI-X belong to the second one. But in contrast to the multi-bit bus of the previous generations, in which multiple users are connected in parallel, PCI-E is organized as a point to point connection with serial data lines. Due to the high clock frequency and special encoding, the typical transfer rate achieves 2.5 Gbit / s and higher.

Multiple devices are connected to the bus using the built-in multiplexers. Serial data transmission makes it possible to significantly reduce the number of contacts in the connector and of wires in the cable to four, as well as perform many port VLSI circuits. PCI-E consists of point-to-point connections in the multiplicity of x1, x2, x4, x8, x12, x16, or x32. Each connection is called as a **lane**. One lane consists of a pair of wires, which transmit signals in the forward direction and the other pair, which transmits a signal in the opposite direction.

During its transmission through the lane, the byte is encoded by ten bits that are transmitted sequentially, i.e. by the **8b/10b encoding**. This encoding is characterized in that the bitstream sequence contains no more

than five equal neighboring bits, and the difference between the counts of ones and zeros in a string of at least 20 bits is no more than two. Due to this, as well as to additional polynomial convolution of the bit sequence, the signal in the lane becomes a noise-like signal with the reduced ***electromagnetic interference (EMI)***, has a low power, high noise immunity and ensures reliable recovery of the synchronizing clock signal in the receiver.

Furthermore, the transmitter and receiver are electrically disconnected. The signal has no DC component, and therefore, it passes through the capacitors mounted near the connections. This increases the reliability of the communication and makes it possible to perform the ***hot-plug*** or ***hot-swap*** connection of peripheral devices.

Some combinations of ten bits or ***symbols*** are prohibited and are evidences of the transmission error, and some of them are ***control symbols***. For example the control symbols NACK, ACK serve as the acknowledge signals, the symbol IDLE means that the bus is free. Thus, PCI-E bus signal is transmitted continuously, which allows the receiver to quickly synchronize the data reception.

The data transmission via the serial interface is performed using a packet protocol. During the hardware initialization, the connection is initialized with the automatic selection of the number of lanes and the clock frequency without the intervention of the operating system.

SATA or ***Serial ATA*** is the standard interface, which replaces the IDE, SCSI and ATA busses for the connection of the HDD. Its structure at the physical and logical levels is very similar to the structure of the PCI-E bus. Unlike the USB bus, SATA provides significantly increased response of the peripheral device and, accordingly, a large bandwidth due to the fact that it has no temporal data multiplexing.

2.1.6 AMBA interface

A new approach to SOC designs is the use of platform technology. The platforms can be implemented both in ASICs and in FPGAs. These platforms have libraries that contain pre-designed and pre-verified intellectual property (IP) cores. An **IP core** is a documented project of a module, which can be adapted to the customer needs when it is customized in the SOC. Users can mix-and-match the functional IP core from the library to assist in design of the SOC. To connect IP cores together successfully, they have to be arranged with the standardized interfaces and communicate with a particular bus protocol.

At present, the Xilinx Zynq and Altera hard processor system (HPS) as the SOC platforms are well-known. Both platforms are based on the two-core processor Cortex™-A9, FPGA fabric and the AMBA interface of the fourth generation. Below the AMBA interface is described as a bright representative of on-chip busses, which are widely used in the SOC design.

The ARM processor is a most widely used RISC architecture built in the SOC. The ARM processor is provided with the AMBA bus, which is an open specification from the ARM corporation. This bus can be used not only with the ARM CPU but also with another CPU cores and application specific devices. The properties of the AMBA bus are similar to ones of other standard busses like IBM CoreConnect bus, Altera Avalon bus, VSIA Virtual Component Interface, and others. In the AMBA bus architecture there are three distinctive busses: advanced system bus (ASB), advanced peripheral bus (APB), and advanced high-performance bus, called the AHB.

Several masters and slaves can be connected to the AHB, but at a time only one master is allowed access. The master to be allowed access is selected by an arbiter. The AHB-APB **bus bridge** serves as a slave on the AHB, and the only master in the APB. The various low performance peripherals on the APB serve as the APB slaves. ASB is an alternative system bus suitable for use

where the high-performance features of AHB are not required. Below the AHB is considered, which shows the main properties of the AMBA bus. Its signals are the following:

HCLK – bus clock. All signal timings are related to the rising edge of HCLK;

HRESETn – bus reset signal, is active low and is used to reset the system;

HADDR[31:0] – 32-bit system address bus, which is given by the master;

HTRANS[1:0] – transfer type. Master indicates the type of the current transfer, which can be Nonsequential (code 10), Sequential (code 11), Idle (code 00) or Busy (code 01);

HWRITE – transfer direction, when high, master indicates a write transfer and when low – a read transfer;

HSIZE[2:0] – transfer size, master indicates the size of the transfer, which is typically byte (code 000), halfword (code 001) or word (code 010). The protocol allows for larger transfer sizes up to a maximum of 1024 bits;

HBURST[2:0] – burst type, master indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping;

HPROT[3:0] – protection control, master provides additional information about a bus access, which intended for use by any module that wishes to implement some level of protection. The signals indicate if the transfer is an opcode fetch or data access ($HPROT[0] = 0$ or 1), as well as if the transfer is a privileged mode access or user mode access ($HPROT[1] = 1$ or 0). For bus masters with a memory management unit these signals also indicate whether the current access is cacheable ($HPROT[3] = 1$) or bufferable ($HPROT[2] = 1$);

HWDATA[31:0] – write data bus. It is used to transfer data from the master to the slaves during write operations. Minimum data bus width of 32 bits is recommended. However, this may easily be extended up to 1024;

HSEL_x – slave select. Decoder each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave. This signal is simply a combinatorial decode of the address bus;

HRDATA[31:0] – read data bus. It is used to transfer data from bus slaves to the bus master during read operations;

HREADY – transfer done. When high, slave indicates that transfer has finished on the bus. This signal may be driven low to extend a transfer;

HRESP[1:0] – transfer response. Slave provides additional information on the status of a transfer. Four different responses are provided, Okay, Error, Retry and Split (HRESP = 00, 01, 10 and 11, respectively);

HBUSREQ_x – bus request. Master _x signales to the bus arbiter that it requires the bus. There is such an signal for each bus master in the system;

HLOCK_x –locked transfers. When high, master indicates that it requires locked access to the bus and no other master should be granted the bus until this signal is low;

HGRANT_x – bus grant. By this signal the arbiter indicates that bus master _x is currently the highest priority master. Ownership of the address or control signals changes at the end of a transfer when HREADY is high, so a master gets access when both HREADY and HGRANT_x are high;

HMASTER[3:0] – master number. These signals from the arbiter indicate which bus master is currently performing a transfer and is used by the slaves, which support split transfers to determine which master is attempting an access;

HMASTLOCK – locked sequence. Arbiter indicates that the current master is performing a locked sequence of transfers;

HSPLITx[15:0] – split completion request. This 16-bit split bus is used by a slave to indicate to the arbiter which bus masters should be allowed to re-attempt a split transaction. Each bit of this split bus corresponds to a single bus master.

Some bus lines can be absent, if needed. For example those, which support the split transfers, can be removed.

The AMBA AHB bus protocol is designed for the use with a central multiplexor interconnection scheme. All bus masters drive out the address and control signals indicating the transfer they wish to perform and the arbiter determines which master has its address and control signals routed to all of the slaves. A central decoder is also required to control the read data and response signal multiplexor, which selects the appropriate signals from the slave that is involved in the transfer. Fig.5.3 illustrates the structure required to implement an AHB design with two masters and two slaves.

The multiplexor MUXA selects address and control information from the masters to the slaves, the multiplexers MUXDW and MUXDR transfer the data when writing and reading, respectively.

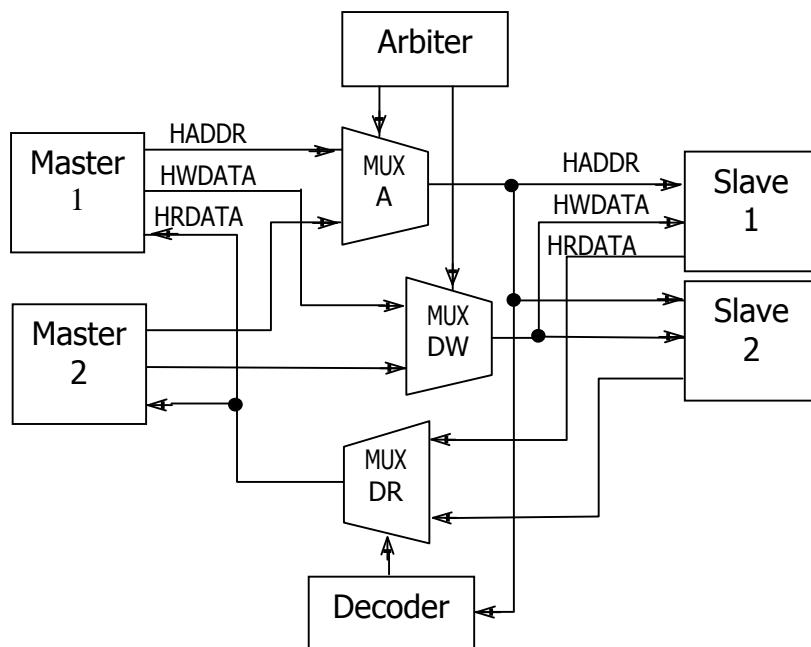


Fig.2.3. AMBA interface structure

All transfers must be aligned to the address boundary equal to the size of the transfer. For example, word transfers must be aligned to word address boundaries (that is $A[1:0] = 00$). For transfers that are narrower than the width of the bus, for example a 16-bit or 8-bit transfer on a 32-bit bus, then the bus master only has to drive the appropriate byte *lanes* (upper or lower halfword, 3-th,..., or 0-d byte). The slave is responsible for selecting the data from the correct lanes.

The AMBA protocol allows the ***burst*** transfers by a master, which has been granted the bus access. The individual transfers within a burst are called as ***beats***. Four, eight and sixteen-beat bursts are defined in the AMBA AHB protocol ($HBURST = 010, \dots, 111$), as well as undefined-length bursts ($HBURST = 001$) and single transfers ($HBURST = 000$). Both incrementing and wrapping bursts are supported in the protocol. ***Incrementing bursts*** access sequential locations and the address of each transfer in the burst is just an increment of the previous address. For ***wrapping bursts***, if the start address of the transfer is not aligned to the total number of bytes in the burst ($\text{size} \cdot \text{beats}$) then the address of the transfers in the burst will wrap when the boundary is reached. For example, if the start address of the transfer is 0x34, then it consists of four transfers to addresses 0x34, 0x38, 0x3C and 0x30. Bursts must not cross a 1kB address boundary. The first beat of the burst transfer has to be of Nonsequential type, and the others – of Sequential type.

The address and data of the different beats in a single burst are transferred in a pipelined fashion. A write burst which writes data D_1, D_2, D_3 to addresses A_1, A_2, A_3 respectively is shown in Fig.2.4. Note that data D_i and address A_{i+1} are transmitted in the same clock cycle on the HADDR and HWDATA lines. Thus the address and data phases of consecutive beats within a burst can overlap.

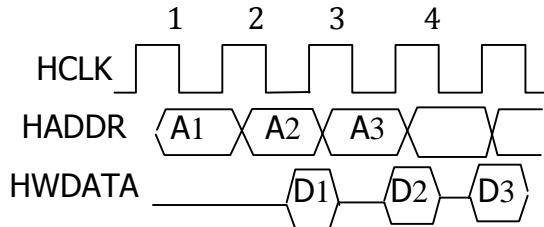


Fig.2.4

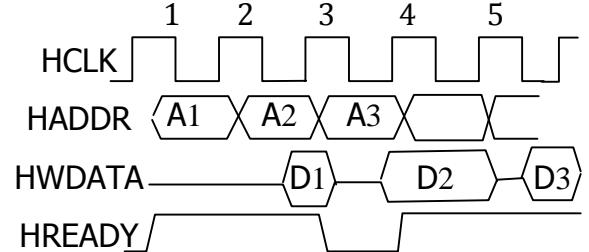


Fig.2.5

The protocol allows a slave for inserting the wait cycles by deasserting a HREADY signal if the slave is not ready to service a transfer. This extends the data phase of a transfer. Due to the pipelined nature of the bus, the address phase of the next transfer also has to be extended. Fig.2.5 shows the writing of D_1, D_2, D_3 to addresses A_1, A_2, A_3 with the insertion of a single wait cycle in the transfer of D_2 .

In order to prevent an excessive number of wait cycles, the protocol allows the release of bus access to the other masters. This is co-ordinated by the slave which either informs the arbiter of its temporary inability to service a master (a Split response) or informs the master to retry the transfer (a Retry response). The provision of ***split transfers***, that is, temporarily suspending a transfer and resuming it later when the slave is ready, raises many important questions. The pipelined nature of the AMBA bus further complicates the situation.

The AHB protocol specifies a certain behaviour that must be respected. Firstly, the master must perform pipelined accesses: every transaction must be performed in two phases. First comes an ***address phase*** during which the address and control signals are driven. At the end of this phase, the slave selected by the address samples the address and control signals and begins its response during the ***data phase***. The response includes the driving of certain control signals and either the emission of the read data or the sampling of the write data at the end of the cycle. This rule is referred to as the ***Pipeline rule***.

Then the slave can drive HREADY low to stretch the length of a bus cycle. The master must be able to stall its execution to respect the slave's request. This is the ***Stretch rule***. The AHB protocol also has provisions for several bus masters: only one master can have access to the slaves at a given time. All of the other masters must wait until the bus is assigned to them. This is the ***Arbitration rule***.

Several accesses must be made in an uninterrupted, or atomic, fashion. The AHB protocol offers the possibility of performing locked access sequences, that the arbiter cannot interrupt to grant the bus to another master. Before the first address phase, the master must warn the system that the locked transfers are about to begin by asserting the HLOCK signal. The signal must be de-asserted during the address phase of the last access. This is the ***Lock rule***.

Finally, a slave can issue various responses to a master's request. The Error response indicates that the access has failed. The Retry and Split responses must be handled in a special way: every clock cycle, the master must observe the status of HRESP. If one of the two aforementioned responses is given, then the master must immediately drive the Idle value on its HTRANS output. This cancels the address phase that followed the one that caused the response. On the following bus cycle, the master must retry the access that had caused the unusual response. This rule is referred to as the ***Exception rule***.

2.1.6 Problems

- 1) What factors does the length of the interface busses infer and why?
- 2) What does occur when two sources are connected to the common bus simultaneously?
- 3) What is the role of the arbiter in the interface?

4) For which purposes have the modern high-speed interfaces the point-to-point configuration?

5) Why have the interfaces with the split transactions much higher throughput?

6) Why does the DMA mechanism increase the overall computer speed?

7) The high-speed serial interfaces like PCI-E, SATA, Ethernet, USB are robust and provide the hot plug-in mode. Explain, which features provide it.

8) Why is the common bus architecture of the modern microprocessors based on a set of multiplexers but not on the tristate buffers?

9) Why have the high-speed serial interfaces much higher throughput than the parallel interfaces have?

10) The high-speed serial interfaces can have much longer bus wires than the parallel interfaces. Explain, why.

11) The high-speed serial interfaces can have much longer latent delay than the parallel interfaces. Explain, why.

2.2 Memory in computers

2.2.1 Memory types and their properties

All the computers require the variable storing in the memory cells and reading therefrom as required. These cells are registers, or they form blocks called memory units which can store millions of data bits. There are different types of memory units with different characteristics depending on their application. The memory unit with the writing and reading modes, which stores the data indefinite period of time at the arbitrary address, and frequently outputs it, if necessary, is a ***random access memory (RAM)***.

The ***read-only memory (ROM)*** stores the information for a long time, and often before the ROM is used in the system. There is a kind of memory, called the ***programmable ROM (PROM)***, where the user can store the desired data using the recording procedure. But it requires the use of a special electronic writing device and is executed within a certain time. If PROM erasure is performed electrically, such a memory is called an ***electrically erasable programmable ROM, (EEPROM)***.

Increasing the memory volume increases its cost and reading-writing delay. Therefore, the computers use the memory units having the different volume, speed, which form a hierarchical system. The main memory unit in the computer is the RAM. It is usually built on the ***dynamic RAM chips (DRAM)*** or on the ***static RAM (SRAM)***. The prices for the memory chips of the equal volume are reduced by about 30% per year. The DRAM is typically five times cheaper than SRAM of the same volume.

Also, the DRAM power consumption is about four times less. But as a rule, the SRAM speed is 2—3 times higher than the speed of DRAM. In the modern synchronous DRAM (***SDRAM***) the DRAM of the large volume and the

high-speed SRAM buffer are combined. Thanks to the pipelined ***burst mode***, the SDRAM provides the average access time less than 2 ns.

There is a tendency that the RAM speed doubles every five years. But the speed of the CPU increases more rapidly. Thanks to the Moore's Law, the CPU speed doubled for two years. This leads to an increase in the gap between the slow RAM and high-speed CPU. A compromise solution to this problem is the use a small intermediate memory with the high speed. This memory is called as the cache memory or ***cache RAM***. The cache RAM speed depends on the distance to the CPU. The most effective way is placing this RAM near the CPU on the same chip. The amount of such cache RAM is limited by the IC technology and is equal to near 0.1 — 10 megabytes. To increase the efficiency of the cache RAM, it is performed as a hierarchical RAM. The RAMs, which are integrated into the chip, are the first and second levels of the cache RAM.

When CPU requires data and it does not find them in the cache RAM of the first level, the data are copied to it from the second level cache RAM. To increase the speed of data transmission, the cache RAMs are operating in parallel at different levels: the portion of data is read from the RAM of the first level, while another portion of data is rewritten to it from the second level cache RAM.

Most often, the instructions and data are located in different memory areas. After reading, they are loaded into the various parts of CPU: the instructions enter the control unit, and data enter the datapath. Therefore, the cache RAM is often divided into two parts: data cache RAM and instruction cache RAM. This separation allows the CPU to read both data and instructions simultaneously. In addition, the protection of data and programs is going on in different ways. The architecture, in which the data and instruction RAMs are formed as separate units, historically was named as the ***Manchester architecture***.

The large amounts of data are stored in the external storage, which store them after the power is turned off, i.e. in the ***non-volatile RAM (NVRAM)***. Such a RAM has a large volume ($10^{10} — 10^{16}$ bytes), but it has a low speed. They are often based on the magnetic disks and sometimes on the magnetic tapes. Many modern NVRAM in portable devices, such as mobile phone, digital camera, based on the flash EEPROM.

The ***hard disk drive (HDD)***, which is still referred to as the Winchester, is the most common NVRAM. Its access time is equal to 1—10 ms, and the data transfer speed is 2—200 MB / s. In order to equalize the speed of RAM and the speed of the NVRAM, the HDD uses a buffer memory of average volume and average performance, which is called as **HDD-cache**.

The system of all the memory units, which are used by the CPU architecture, is called as the computer memory. This memory consists of several levels, which is seen by the user as the virtual memory. Each layer of such a memory is controlled by a special control unit, which provides the automatic data transfer between different levels of the memory. Such a unit usually uses some strategy, that minimizes the average data access time. In accordance with some access strategy, the virtual memory behaves as a memory unit with the volume of some NVRAM ($\sim 10^{12}$ bytes) and the access time of SRAM (~ 10 ns).

2.2.2 Static memory

The integrated memory circuit may be part of the CPU or a system on the chip. The memory cell in it is implemented as a trigger, which network is shown in Fig.1.5 or in Fig.2.4. Therefore, such a memory unit could not have a large amount (more than $10^6 — 10^8$ bytes), and it spends most of the system energy. This is the reason that the large volume memory is usually manufactured as the separate chips. Thanks to a special technology, special

design of the memory cell, read-write amplifiers, multiplexers and decoders, the separate memory IC has the minimized power consumption, high speed and volume up to $\sim 10^9$ bytes. The memory chips are divided into SRAM, DRAM and EEPROM, according to their properties and technology.

The abbreviation RAM means that the memory access is random, i.e. any address can be accessed at any time. Recall that in the computers of the first generation, the memory units were formed by the delay lines, that is they had the sequential access mode.

Title static RAM (SRAM) has the property that as soon as the data is stored in RAM they are save there until power is turned off. Fig.2.6 illustrates the CMOS SRAM with the volume of 2 kbytes. The principles of designing of such a memory are the same to SRAMs of other construction.

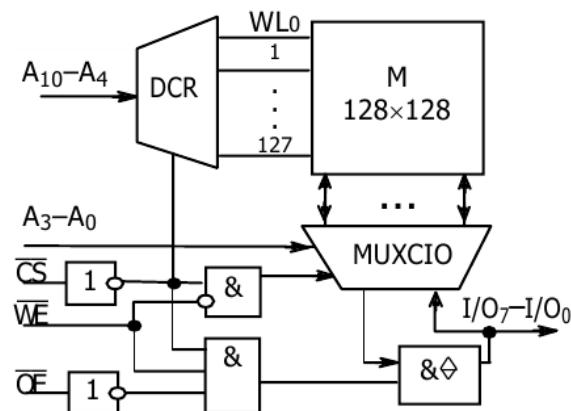


Fig.2.6. The SRAM structure with the volume of 2 kbytes

2.2.3 Dynamic memory

DRAM is similar to SRAM in that the data are stored under the same cell addressing law. On the contrary, in DRAM it is necessary to organize the periodic restoration of the memory state (**refresh**), to keep the data from being lost. The difference between the two types of the memory is the cell structures, which are shown in Fig. 2.7 and 2.8, respectively. The dynamic memory cell consists only of one transistor and a storage capacitor C_s . This

enables the high integration density and large circuit volume that achieves $\sim 10^{10}$ cells.

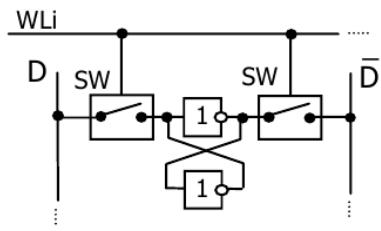


Fig. 2.7. SRAM cell

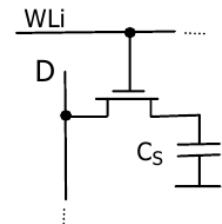


Fig. 2.8. DRAM cell

When the line selecting signal is $WLi = 1$ (see Fig.2.8), the transistor operates as a closed switch and allows the memory to read or write. Storage condition is activated when this signal is $WLi = 0$ and the path between the data line D and the capacitor C_s is terminated. The closed transistor, having a small, but significant leakage current, promotes the discharge of the capacitor. Therefore, the data can be stored in the DRAM for only a short period of time (< 1000 ms), and therefore they must be periodically restored.

The restoring operation (**refresh**) is performed by a special circuit which amplifies the charge in C_s . The refresh period is usually should be no more than a millisecond. For one refresh cycle, one row of the memory array is restored. To restore the DRAM, the refresh procedure has to go through all the rows. Therefore, the lines in a large DRAM are looked over with a frequency of several megahertz. To ensure the automatic refresh, the DRAM has a **refresh counter**, which counts the recoverable lines and is a part of the row address register.

When accessing the DRAM, the row addressing and column addressing is performed separately. Suppose we have a DRAM with the structure like one in a Fig.2.6. The Fig. 2.9 shows the timing diagram for the writing therein. The RAS and CAS signals indicate the line selection (Row Address Select) and the

column selection (Column Address Select). When $\overline{\text{RAS}} = 1$, the circuit stores data. When $\overline{\text{RAS}} = 0$, $\overline{\text{CAS}} = 1$, then the row address bits are stored in the internal register address. At the same time, all the relevant cell lines are refreshed. When $\overline{\text{CAS}} = 0$, then the column address bits are decoded, and the writing or reading operation is performed, depending on the signal $\overline{\text{WE}}$. This operation is ended when $\overline{\text{CAS}} = 1$.

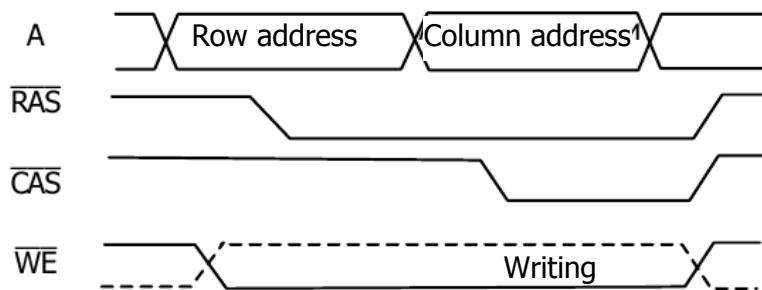


Fig. 2.9. Timing diagram of the DRAM writing

As we can see, the address is loaded in the DRAM in two cycles. Thanks to this, the address pin count in the DRAM chip is twice less than in the SRAM. The disadvantage of this mode is the need for an external address multiplexer, which generates the row address and column. The access time to the DRAM cell is minimized when the row address is stored once, and then all the cells of the selected row are available at the different column addresses.

The microprocessors usually have a special DRAM controller, which provides the proper flow of addresses and control signals, as well as the control of the time slots, chip initialization, corrupted data recovery. Therefore, the DRAM flaws are "invisible" to the user.

Due to the large address decoding circuitry, the signal amplifier delay and the two-cycle addressing mode, the DRAM has an access time of tens of nanoseconds. The modern DRAM ensures its increased speed due to the

following improvements. A number of memory banks which are placed in the chip has the independent control circuits and decoders. Then the address word is the concatenation of the row, column, and bank address fields. The bank address can be changed rather quicker than the access to the selected cell. Therefore, the series of N accesses to the same cell but in the different banks can be fulfilled in N adjacent clock cycles.

In the SDRAM the pipelined access is performed, that is input data, address, read data, and even data arrays are read by the row address and are stored in the pipelined buffer registers. Therefore, the read operation requires 2 — 10 cycles. But thanks to this, the clock frequency can be increased up to several hundreds of megahertz.

The DRAM chips have the bi-directional data bus with the bit width of 36 and more bits. To provide the quick access to a portion of the data (byte, nibble) the ***data bit mask*** (DQM) is used.

If a set of equal operations is performed, such as the sequence of readings, then the average speed can be rather high. For this purpose, the ***fast-page mode (FPM)*** or the ***burst mode*** is used. In this mode, the row address is stored and the access is performed for a sequence of adjacent column addresses. This is provided by the column address register, which is implemented as a counter. Therefore, when the CPU writes or reads the array of data, then the burst mode is usually used.

To control the FPM mode, the access delays are programmed in the DRAM by the specification of four numbers. For example, a set of 5-1-1-1 means that the first access requires five clock cycles, and the other accesses one clock cycle do. The DRAM control unit must distinguish, that the DRAM operates in different modes (initialization, reading, writing, fast-page mode, refresh, downloading, etc.) the corresponding sequence of control signals is provided. The signals RAS, CAS, WE, DQM and others form the control word,

which is directed to the address bus and is stored in the DRAM by the front of the clock signal. During the computer initialization, the DRAM is initialized as well. By this process, all the DRAM cells must be refreshed during several milliseconds. Only after that, the DRAM is guaranteed to work properly.

2.2.4 Read-only memory

EPROM is an Erasable-Programmable ROM. The EPROM programming consists in the charge transfer to the gate area of the MOS transistor, which serves as a memory element, due to the increased voltage on the gate and the electron tunneling effect.

If this charge is present, then the transistor is permanently opened. This charge is stored for a long time, because it is "stuck" in the gate dielectric, or the auxiliary gate, which is immersed in a dielectric, the so-called ***floating gate***. In the devices of this type, the data erasure occurs when these transistors are irradiated with the ultraviolet light for several minutes.

The ***Electrically Erasable ROM (EEPROM, E2PROM)*** have the advantage that the data can be erased in the circuit. To clear the bit, the voltage is applied with the reverse polarity to the gate of the storage transistor. The new technology makes it possible to perform a large number of erasing and write operations (up to a million) in a short time. For the high erasure rate as compared to EPROM, these devices are called as ***flash EPROM***, where "flash" means the speed with which the data can be erased. Today, a single transistor can store 2 or even 3 bits, which are coded as the voltage level, which was programmed in it.

This kind of the memory is very advanced. At present, the 3D chips of the flash EPROM are manufactured, which have up to 64 layers of the storage transistors. The capacity of such a chip achieves hundreds of gigabytes. Its structure is similar to one of the SDRAM — the static RAM buffer provides the

high speed of the data reading. Therefore, the flash EPROM is the base chip of the ***solid-state drive (SSD)***, which substitutes the HDD in the applications, where the high speed, small dimensions, and high reliability are needed.

But the erasing in the flash EPROM is performed with the whole blocks of the memory. After some set of writings (several thousand) such blocks start to degrade and after some period of time, this ROM failures.

Ferroelectric RAM (FRAM) retain the advantages of DRAM (high volume and speed) and the NVRAM (the data is saved after a power failure). The RAM cell resembles a DRAM cell. But the bits of data stored in it is not a charge, but a sign of the dielectric polarization. To do this, the capacitor is made of a ferroelectric. When the bit reading is performed, then the capacitor is recharged. And if the capacitor while the recharging changes the polarization, it is considered, that a bit one was written. But this process changes the polarization. Therefore, the reading process is completed with the writing this bit back.

2.2.5 High-speed memory

The access time in the ***high-speed memory*** is less than in conventional RAM. Such a memory is connected between the RAM and CPU to reduce the average access time. This increases the speed of the CPU, because it directly accesses the high-speed memory. Furthermore, the access to the high-speed memory and an ALU operation can be implemented in parallel.

The memory types of immediate, associative, pipelined and stack addressing are distinguished. The memory with the direct addressing has the cell addresses 0 to $m - 1$. The cell address is placed directly in the address field of the instruction. When m has a small value (up to 128 — 256), the memory is called as the ***Registered RAM (RRAM)***.

In the ***associative*** addressing mode, the operand has not the address, but the label, tag or keyword. When the tag is inputted to the associative RAM, it outputs one or more words, which coincide with the input tag, or nothing if the tag does not match. This addressing mode is used in the cache memory, which is discussed hereinafter.

Two or more processor units are often connected through the memory buffer with the ***pipelined addressing mode***. Such a memory is formed a set of registers, which are connected to the pipeline. The source writes the data to this memory, and the receiver reads these data in the same order in which data is supplied to the memory. This mode is called ***first-in first-out (FIFO)***. Therefore, this kind of the memory is referred to as FIFO.

FIFO is often used when the source provides the data at irregular intervals, or at a different frequency than the receiving unit receives them, for example, in the communication systems.

In the ***stack*** mode, the memory unit is considered as a register stack. The user only has the access to the top register of the stack and can ***push*** data to it or remove (***pull***) the data from it. The read operand is that has been written in the stack as the last one, and therefore, this kind of the memory is called as ***Least Recently Used (LRU)***. Often we have the access not only to the top register, but also to the next one, or to the arbitrary register (by its address). For example, so the stack of the I8x87 floating point coprocessor is constructed.

2.2.6 Cache memory

The automatic exchange of data between RAM and the high-speed memory is achieved in the associative addressing mode. The structure of the cache RAM is shown in Fig. 2.10. It has a data memory block M, a ***context***

associative memory (CAM), an address comparator CMP, a control unit CU, an address register RGA, and the data register RGD.

Each data, which is written in M, has a tag, which is equal to its address, and which is stored in the CAM. When the CPU performs the data reading or writing, the data address is written in RGA, the read RD and write WR signals enter the CU. The address in RGA is compared with the addresses in the CAM using the comparator CMP. If the address is equal to a certain address in the CAM, then we see about the **cache hit**. In this situation, the data read from M is written in RGD or from RGD is stored to M. If the address is not equal to any address in the CAM, we talk about the **cache miss**. Then CU organizes the access to the external RAM.

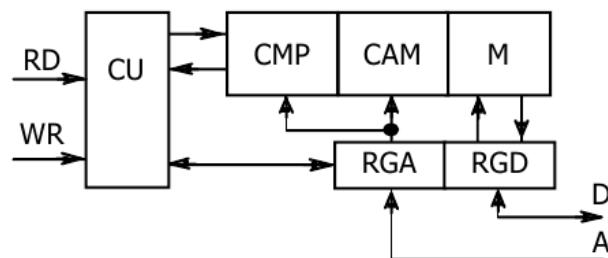


Fig. 2.10. Cache RAM structure

When the cache miss, the data from the RAM is written to RGD and in the empty cell of the block M, and its address is stored in the respective cell of CAM. By the next reading of this address, the real reading occurs from the cache RAM but not from the outer RAM.

When the data is written, the writing process is performed due to some cache consistency strategy. Due to the **write-through** strategy, the data is stored both to the outer RAM, and to the respective cell in the cache RAM. The **write-back** strategy is also called store-in or copy back one. In contrast to the write-through strategy, write-back only modifies the cached data. By the use of the dirty-bit, that cache line can be marked as dirty, the cache memory

indicates for the CU, that the line, when being replaced, must be written back to the next level of the memory.

When the cache memory is in action, all its cells are filled. Therefore, to access the new address, some cache memory cell must be released. It is natural, when the cell is released, which has the smallest number of the access times. To determine it, the CU has a network that selects it according to some strategy.

The next strategy is commonly used. Let the latch T_i is associated with the i -th cell of the CAM. Initially $T_i = 0$ for all i . If the access to the i -th cell hits, then $T_i = 1$. This means that at least a single access to this cell occurred. If it misses, then the nearest cell is selected for the release, for which $T_i = 0$. When $T_i = 1$ for all addresses i , then all the cells are released, and for them $T_i = 0$ except one cell which is selected for the current access. The strategy of the random cell selection or the round-robin discipline are used for the cell release as well.

A disadvantage of the associative memory is its hardware complexity when the address number is large. Its effectiveness increases when the ***set associative mode*** is used. Many microprocessors have the set associative cache RAM. When a cache miss occurred, then not a single word is moved from the outer RAM but a set of bytes which form the whole line of data. This minimizes the hardware for the tag comparing and storing. Besides, in many cases, the address sequence has the natural order or the addresses of a set of accesses belong to the same RAM line. For example, the data form an array, the next instruction in the program usually has the near location. Then only a single miss occurs for the whole line of data or instructions, which is moved using the high-speed burst mode. As a result, the average access time can be reduced several times. The length of the cache line is equal to 64, 128 or 256 bytes.

2.2.7. Problems

- 1) The fast memory usually has the small volume, the high volume memory usually is slow. What is the cause of this situation, and how is it resolved in computers?
- 2) The ***memory splitting*** is a method of its speed increase. It is based on parallel access to several memory units. Consider the i -th array data are stored in the memory units with the number $i \bmod 4$. Then up to 4 adjacent data can be read simultaneously or with the small time shift. I.e., the memory speed is increased up to 4 times. Declare the limitations of the splitted memory use. Where can we find it now?
- 3) Which features make the throughput of DRAM be increased?
- 4) The Micron company has proposed the Hybrid Memory Cube (HMC) DRAM architecture. Such a memory unit consists of a set of DRAM chips which are connected to the CPU through the lanes of the PCI-E-type interface. Explain, which features has this kind of a memory contra the usual DDR3 DRAM and why.
- 5) Why the SSD drive has much higher speed as the HDD drive has?
- 6) The hash function is a function, which maps homogeneously the N -bit argument to the n -bit result, $N \gg n$. How to build the associative memory using the hash function?
- 7) The coprocessor i8087 is based on the 8-level stack, the top and the next register of them are forwarded to ALU. Propose the algorithm for this coprocessor, which calculates the formula $y = a \cdot x^2 + b \cdot x + c$. Show the states of the stack.
- 8) The stack segment of RAM is usually used for the context saving when the subroutine calls occur. Explain, why.

9) Consider a program, which computes the large square array of data. When the rows of the array are computed then the computing speed is much higher than when the columns are. Explain, why.

10) The experience shows that the programs which have preferably short jumps are executed much quicker than ones which have long jumps. Explain, why.

11) For which purposes is the program optimized to have the minimized number of accesses to the RAM?

12) Why does the cache RAM occupy more than 50% of the microprocessor chip surface?

13) A flash memory of the volume 8 GB is attached to the computer, which makes approximately 10 writes per second to it of files with the average length of 16 kB. Estimate the time of the reliable operation of this memory.

14) RAM has the volume of 16 GB, the HDD speed is 50 MB/s. Calculate the period of time to save the RAM content to HDD.

2.3. External storage devices

2.3.1 Peripheral devices

The computer peripheral devices are divided into input-output devices and external storage devices. A common characteristic of the input-output device is the data rate (the maximum rate at which data can be transferred between the input-output device and the main memory or CPU). Table. 2.1. represents the basic peripheral devices, which are used in today's computers, as well as the approximate rate of data movements, which are provided by them. Consider the most rapid of these devices — the magnetic disks.

Table 2.1. Examples of the peripheral devices

Type	Data transfer direction	Transfer speed (Kbytes/s)
Keyboard	Input	0.01
Input	Input	0.10
Voice input-output	Input-output	20.00
Scanner	Input	200.00
Laser printer	Output	500.00
Graphical display	Output	100000
Magnet tape	Input-output	2000.00
Optical disc	Input-output	5000.00
Magnet hard disc	Input-output	50000 — 500000

2.3.2 Magnetic disks

The **hard disk drive (HDD)** is composed of one or several, most commonly, aluminum disks with the magnetic layers on both sides. The **track** is the circular sequence of bits on the disc, which are written for its full turn. Each track is divided into **sectors** of fixed length. Typically, one sector

contains 512 bytes of data. Now there are 2048 and even 4096-byte sectors, providing high performance of the hard drive.

Before the data, the ***sector header*** is written, which allows the HDD controller to synchronize the magnetic head before reading or writing the data. The sector address and a set of flags are written in the header, including the sector corruption flag. After the data in the sector, the cyclical control code is written. This is typically a convolution code with a 32-bit polynomial that can reliably identify the corrupted data in the sector. This code is named as the ***cyclic redundancy check (CRC)*** code. Also in today's hard drive at the end of the sector, the ***error correction code (ECC)*** is placed. This Reed-Solomon code allows the HDD controller to fix some corrupted bytes. Between the adjacent sectors, the intersectoral gaps are placed.

The track structure is formed during the HDD ***physical formatting***. It occurs during the HDD manufacturing. Due to the sector headers, CRC, ECC and intersectoral gaps, the volume of the formatted HDD is at ca. 15% less than the unformatted volume.

The magnetic disks are placed on a single spindle, which spins at a speed of 5400, 7200 or 10,000 rpm. depending on the HDD design (Fig. 2.11).

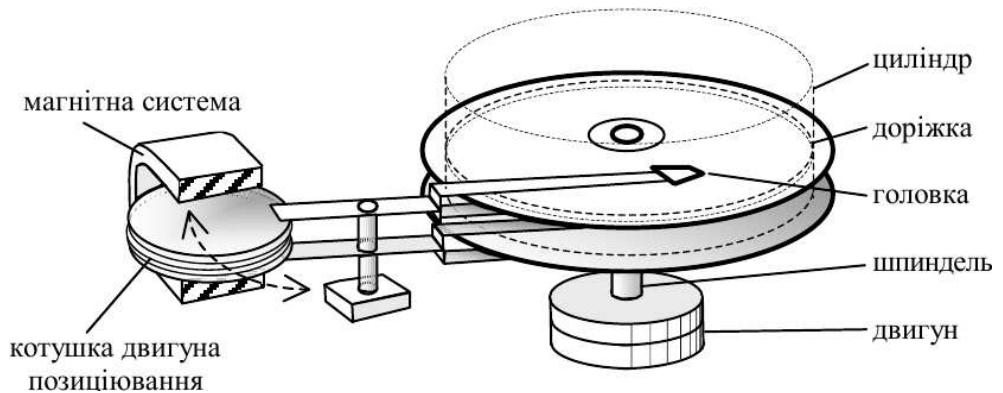


Fig. 2.11. The HDD drive with four heads

The magnetic heads are mounted on the block of light consoles, which can position the heads precisely and quickly over a certain track by the electromagnetic motor (averagely for 5 ms, the transition to an adjacent track lasts less than 1 ms). Thus, the track is a series of concentric circles, the center of which coincides with the center of the spindle. The width of each track is a fraction of a micron. The collection of tracks located at the same distance from the center referred to as a **cylinder**.

The density of the recorded bits in the tracks differs depending on their distance from the disk center. The recording density depends also on the quality of the disc surface layer and the head, on the air purity, and it reaches 10000 bits/mm. To ensure the long life of the HDD, it is manufactured in a rigid sealed casing, the interior of which is connected to the outside air through a dense filter.

On the part of the operating system the hard drive is perceived as a fixed set of drives with the number N_h of heads, the number of cylinders N_c , and the number of sectors N_s . And the numbers of heads, cylinders, sectors are written in the preamble of the relevant sectors as the sector addresses during the physical formatting. The cylinder and head numbering start from 0 and the sectors — from 1 does.

The system **LBA** (*Logical Block Addressing*) limits the cylinder number bit width with $N_c = 16$ bits, head number — with $N_h = 4$ bits and sector number — with $N_s = 8$ bits. This allows making a hard drive capacity of up to $2^{16+4+8+9} = 131$ GB. In order to substantially remove the boundaries of the maximum amount of hard drive capacity, in 2002 the protocol **UltraDMA** has approved the access to the hard drive. According to it, the sector address bit width is limited to 48 bits, which provides the capacity up to 140 000 TB.

HDD has its own controller, which receives the data and instructions via a hard drive interface (ATA, SATA, or SCSI). Also, the controller performs

a correction of the reading errors. In addition, the controller has a cache buffer with the capacity a few megabytes, and the DMA device. Thanks to them, the average access time to the selected sector is decreased and the data transfer speed is increased up to several tens of megabytes per second.

To compress the storing information, the number of sectors on the outer cylinders is increased compared to the number of sectors in the inner cylinders. Also, the number of physical sectors per track is increased up to several thousand, which is in contradiction with the LBA addressing system. Therefore, the HDD physical disk may have up to four logical drives and the mismatching of the addresses of logical and physical sectors.

The HDD controller performs the mapping of the logical sectors addresses in the physical sector addresses. I.e. it keeps the conversion table to convert the physical sector addresses in the logic ones and vice versa. Furthermore, the controller periodically checks the integrity of the sectors. In the case of finding the damaged sector, it replaces it with a sector of the reserved area of the disk with the appropriate correction of the conversion table. This table is stored in a special EEPROM.

2.3.3 Solid-state drive

The ***solid-state drive*** (SSD), also known as a solid-state disk, really contains neither an actual drive nor disks. The SSD is based on the EEPROM integral circuits. But its architecture at the logical level repeats the HDD architecture. The SSD interface is equal to the HDD interface as well.

Due to the absence of both the moving parts and the sequential manner of the sector finding, the SSD provides the high-speed data access and the high operation reliability. The cost of SSD is many times higher than the HDD cost. But it is decreased due to the flash ROM technology improvements.

The high-speed SSD substitutes the HDD in the data centers due to its increased throughput and lower energy consumption. SSD in the form of the separate integral circuit substitute the HDD in the hand-held devices.

The SSD user has to take into account that before the writing to the EEPROM, the respective 256kB block must be erased for ca. 2 ms. Therefore, SSD can provide the high-speed reading (up to 500 MB/s) and slow writing, especially when the files are short. To improve that, the operations are implemented in parallel, the EEPROM is sometimes supplemented by the DRAM buffer. As a result of the buffering, the average access time is decreased from ca. 100 μ s to 10 μ s and is much less than one in the HDD (3—10 ms).

It must be mentioned, that the number of reliable writings to the same address of the modern flash ROM is limited by 3000—10000 times. Only some special reliable blocks of that ROM provide up to 0,1 mln of write cycles. To keep the SSD reliable, the ***wear leveling*** technique is used. According to it, each time, the writing to the same logical sector is implemented to different physical addresses, which are selected randomly by some special algorithm. As a result, the numbers of writings to the different logical sectors are averagely equal to each other. The sector mapping table is stored in the reliable blocks of ROM.

2.3.4 Compact Disc

The ***Compact Disc (CD)*** or the optical disc is developed by Philips, together with Sony in 1980 for storing and expanding the audio information. The overwhelming majority of CD has a diameter of 120 mm, the thickness of 1.2 mm, and the diameter of the middle hole of 15 mm. A thin aluminum layer, which is deposited on a transparent polycarbonate base, has a spiral track. On it there are recesses (***pits***), smaller than a μ m and surface areas (***lands***), which encode the ones and zeros of data. The pits are less than a μ m in

diameter, which provides the high information density. When playing the records, the focused laser diode beam illuminates the track, and the photodiode matrix receives significantly less light from the pits than from the lands. It also ensures that the beam is focused in the center of the track. The positive and negative transitions of the detected light level are recorded as 0 and 1, respectively.

Since 1984 the CD began to be used for the computer data storing, and it is called as **CD-ROM** (*Compact Disc — Read Only Memory*). While the data recording, every byte is encoded by the 14-bit Hamming error-correcting code. Every 24 coded bytes are stored in the 588-bit frame, of which 396 bits are used for the error correction and control. Every 98 frames form one sector. The sector begins with a 16-byte preamble, which accommodates the synchronization sequence, a three-byte sector number, at which the driver finds the sector, and the sector encoding type. Totally, a sector holds 2048 bytes of data and 228 bytes of the Reed-Solomon error correction codes.

Also, the bytes of the neighboring sectors are shuffled. Therefore, if the wide scratch occurs on the disk surface, it spoils a few dozen of neighboring bytes. Then, during the reading the corrupted surface after inverse shuffling, the bad bytes are evenly spread among unspoiled sectors. And these bytes can be recovered by using the Reed-Solomon decoding algorithm. But due to the implementation of a three-level error correction system (in bytes, frames, and sectors), the volume of recorded information is increased by 3.6 times.

To implement the capability of CD-ROM recording by the user, a **Recordable CD (CD-R)** become popular. Such a disc has an additional 0.6 mm wide groove, which directs the laser beam for the recording. A layer of dye, which darkens under the laser heating, is filled in this disc between the metal reflection layer and the disc substrate.

In the ***ReWritable CD (CD-RW)***, instead of the dye, the fusible alloy of silver, indium, antimony and tellurium is used, which has a crystalline and amorphous states with varying degrees of reflection.

CD-ROM has a capacity of about 700 Mbytes. In order to increase their capacity several times, the ***Digital Versatile Disc (DVD)*** has been developed. The sealing of the recording density therein to 4.7 GB or more is achieved by halving the size of pits, the track pitch and manufacturing a multilayer disc. Besides, the infrared laser was substituted by the red laser. In recent years, a proliferation of Blue Ray drives take place, the volume of which reached up to 70 — 128 GB by using a blue laser, which was the reason for the origin of their name.

The CD discs of the large volume are widely used for the archive data storage in the data centers. For this purposes, the glass or sapphire disks start to be used. Such discs can store up to several TB of information. They are immune to the mechanical damages, fires and therefore, they are able to store the data for millenniums.

2.3.5 Logical structure of HDD with FAT

Most of the OS modules and custom software access the hard drive, not addressing the certain sectors, cylinders and heads, but using the file system. HDD logical structure specifies the mapping files into a set of clusters.

The cluster is a predetermined number of neighboring sectors (1,2,4, ..., 128), which determines the minimum size of the file.

The file consists of one or more clusters, which optionally have a sequence order.

The file directory is a file that specifies the file name, its attributes and the number of the initial cluster, as well as a branch to the subsidiary directory.

The root cluster is a cluster that contains the root directory.

File Allocation Table (FAT) is a table, which contains the records, representing the clusters in the HDD logical drive. Each record corresponds to a single cluster, and can be signed as the file end, if it is the final cluster of the file, the number of the next cluster. It can indicate that the cluster is free, or that it is corrupted. Fig. 2.12 illustrates the state of the FAT, which stores the records of two files.

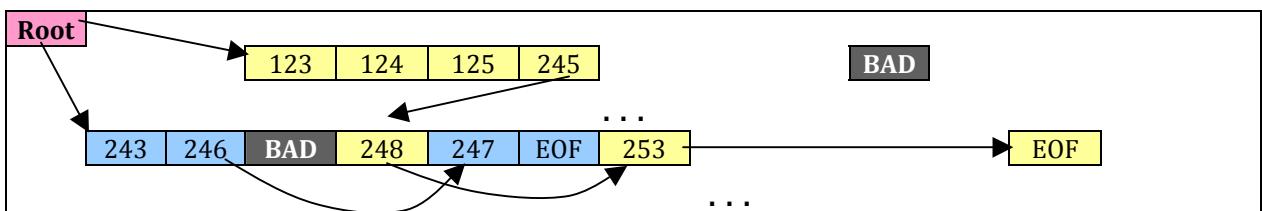


Fig. 2.12. Two files represented in FAT.

EOF – end of file cluster, Root – root directory cluster, BAD – bad cluster

The most common standard logical structure of the hard drive are FAT and NTFS. There are three varieties of FAT system with different recording measurements: FAT12 with the record bit width 12, which provides maximum 4093 clusters; FAT16 with the record bit width 16, that is, the maximum of 65533 clusters; The FAT32 with the record bit width 32, that is, the maximum of 2^{32} – 3 clusters.

Thus one HDD may have from one to four logical partitions (logical disks) with their different FAT tables.

Consider the structure of the logical disc. The first sector of a physical disk is the **master boot sector**, which contains the parameters of the physical disk. It is located on the side 0, cylinder 0 and sector 1, and contains the following information:

- sector size (usually 512, sometimes 2048 or 4096);

- number of sectors in the cluster;
- number of sectors per track;
- number of cylinders;
- error message;
- information about the dividing the physical drive into logical drives (***partition table***);

— placement of the initial sections of the logical drives and indication of the place where the operating system is stored.

The master boot sector is followed by one to four logical drives. The ***logical drive*** includes:

- boot record;
- reserved sector;
- first and second FAT table;
- root cluster directory;
- data clusters (files);
- reserved sectors.

The ***boot record*** contains the following data:

- the jump instruction, which directs to the bootloader program;
- DOS version;
- number of bytes per sector;
- number of sectors in the cluster;
- number of sectors before the FAT table;
- number of FAT tables;
- maximum number of rows in the root cluster;
- number of sectors that store DOS;
- number of sectors which are stored in the FAT table;
- number of sectors per the track;
- number of heads;

— number of hidden sectors.

Due to the lower recording density in the track zero, its reliability is the highest. Therefore, the boot sector, boot record of the logic drive C: and its FAT table are placed on this track.

The search for the file locations is performed by the first FAT table. If this table is corrupted, then the second FAT table is used as a duplicate. But if the first sector is damaged, which contains the master boot sector, the operating system could not be able to reproduce the logical structure of the hard drive, and the writing-reading of the files becomes impossible.

The logical structure of the HDD system NTFS

File system New Technology File System (NTFS) appeared with Windows NT in 1993. and took over all the positive properties of the Unix filesystem. The NTFS file system data presented in the form of files with the same structure. Smaller file occupies one cluster, and always has fields such as title, field of standard information, the file name of the stored data, and at the end of the file — the security descriptor.

The system is based on a system of official files, most of which is a directory of files, called the Master File Table (MFT). All service files are stored in the MFT-zone at the beginning of the disk, which occupies 1/8 of the entire disk volume. When the disk is full, this area is also filled with user files, resulting in delayed access.

2.3.6 Logical structure of HDD with NTFS

The file system **New Technology File System (NTFS)** has appeared with Windows NT in 1993 and took over all the positive properties of the Unix file system. The data in NTFS is presented in the form of files with the same structure. The smallest file occupies one cluster, and always has the fields

such as title, the field of standard information, file name, stored data. At the end of the file, the security descriptor is placed.

The system is based on the system files. The main system file is a directory file, called the ***Master File Table (MFT)***. All system files are stored in the **MFT-zone** at the beginning of the disk, which occupies 1/8 of the entire disk volume. When the disk is full, this area is also filled with user files, resulting in delayed HDD access.

The main file, named \$ MFT, is an array of file records, each of which describes the corresponding file or directory. One record has a length of 1 KB and includes file name, file attributes, and file indexes. The index marks the first file cluster, the length of the chain of clusters and other information. If the file is located in the chains of clusters, it indicates a reference to the first cluster, indicating the continuation of the chain length of the chain. If the file description does not fit into a single record, the reference is made to the record continue. Perhaps the file is continued in several chains so that the file has the tree form. This makes it possible to read the file in parallel flows.

The maximum cluster number is 2^{64} , although the actual length of the file does not exceed 2 terabytes. To compress the cluster coordinate encodings, they are coded by the Huffman codes. Thus, the file directory is a hierarchical list of records sorted in a tree. Then, the cluster address is searched not using the FAT table but traversing the cluster tree.

In the system files \$Bitmap and \$BadClust the free and bad clusters are marked, respectively. And the table of the free clusters takes as much bits as the number of clusters in the hard drive. Consequently, the fact that this cluster is a free one, is approved by the bit, marked as a zero. The file \$Boot stores the system bootloader.

The four main files of the MFT-area are stored in two copies for the reliable functioning of the HDD. The first of it is stored in the initial clusters,

and the second one is somewhere in the middle of the disc in the place, which is specified in the boot sector.

When some file operation runs then OS writes the parameters of this operation in the system log file \$LogFile. Then, if a computer failure occurs when performing a file operation, then during the restoring the computer, the operating system repairs the corrupted files and entries. In addition, only the OS software with the highest priority has the right of access to the system files. Thanks to these features, the NTFS file system is much more reliable than the FAT system.

2.3.7 RAID storage system

The reliable high-speed hard drive is significantly more expensive than conventional HDD. Because of this, the architecture array of inexpensive hard disks has been proposed in 1988, at which, due to its redundancy, it provides both the storage reliability and high-speed access. Therefore, it is called as the ***Redundant Array of Inexpensive Disks (RAID)***. In the RAID array, from 4 to 15 HDDs are connected in parallel to a single controller, so that the OS treats it as a single large hard drive.

There are several ways to organize the RAID array, which are called the ***levels***. For example, in one of the levels, all the sectors of HDD are divided into the strips, and these strips are duplicated in the HDDs. Because of this, the file writing lasts the same time as in a conventional hard drive. But the reading is done in parallel with multiple HDDs, and its speed is much higher. In the case of the damage of any strip, it is recovered from its copy. If one of the hard drives is failed, the RAID array does not stop the operation, and the failed HDD may be replaced with the non-defective hard drive without the power turning off.

Now in many chipsets, the RAID controllers are embedded, enabling the RAID array widespread use.

2.3.8 Problems

- 1) The 32-bit CRC code provides the finding the corrupted sector with the probability of $1.0 - 0.5 \cdot 10^{-9}$. How many corrupted sectors, which could not be found, can be stored in the 6 Tbyte HDD?
- 2) Consider the files in HDD are of the length of 1 MB and are fully fragmented, when their sectors are placed in the random order on the disc surface. How much does the HDD speed increase after the defragmenting the files? Note, that one track contains 256 sectors, the cache buffer is not taken into account.
- 3) Consider the application, that makes the reading-writing operations to the HDD 10 times during its implementation period. These operations take 50% of its working period. How much does the speed of the application increase, when the cache buffer of the proper volume is added to HDD?
- 4) In some application, the HDD transactions occupy ca. 50% of the whole application time. How much is this time decreased when HDD is substituted by SSD?
- 5) The OS implements the substantial writings to the SSD with the average period of 0,1 s. The average file length is 1 MB. SSD has no buffer and its volume is 64 GB. Estimate the maximum live time of SSD.
- 6) Estimate the width of the scratch, which can permanently damage the information in the CD-ROM. Note, that the scratch runs radially, 16 neighboring sectors are shuffled, and the Reed-Solomon algorithm can recover every 20-th byte of data.

2.4 Computer console

2.4.1 The concept of the console

The operator console, which has appeared in the first computers, was a computer control and display panel, which was cantilevered on the cabinet with the mainframe. Using the console, the data have inputted and the results have displayed. The operator started and stopped the computer operation using its console.

Since then, the traditional set of devices for direct computer control, manual input-output of the data and programs are called the console. All algorithmic languages have a separate set of functions and procedures for entering numeric and text data from the console and for the console output. Also, the traditional OS string instructions are entered from the console and OS displays the results of its reaction in the console.

The first computer console consisted of a set of switches and lamps or electric typewriter. Now, as a rule, the console is considered as the display with the keyboard and mouse-type manipulator.

2.4.2 Keyboard and mouse

The keyboard is used for manual input of the data and text. The keyboard differs on the emplacement of letters on its keys. The standard Latin keyboard emplacement is the location of "QWERTY" (by location of letters on the top key line), and the Cyrillic one is "ЙЦУКЕНГ". The standard keyboard has 101 keys.

Now the keyboard is connected to the processor through the connection PS/2 or USB. Accordingly, such a keyboard is called the keyboard of the PS/2 type or USB type. When the key is pressed, then the appropriate

letter code is generated and sent to the processor. The PS/2 type keyboard letter codes are not equal to the ASCII code but are, so-called, ***scan-codes***.

Once the key is pressed, then the respective scan-code is transferred to the processor and is stored in the BIOS port with a number 60h. If it is pressed and held, then the scan code is issued approximately every 100 ms. After releasing the key, the keyboard sends a release prefix code "F0", after which the scan-code of the released key is sent. By pressing some keys, called extended keys, the prefix code "E0" is sent before the scan code.

The mouse is a hand-held device, which in its traditional design comprises a rubber bulb at the lower side and several buttons at the upper side. The mouse was invented by D. Engelbart in 1967 during research on finding the effective interface between humans and computers.

If the mouse moves, then the ball makes a rotation, which is proportional to the distance. The codes of the distance and the button pressing events are sent to PC through the four-wire cable and the PS/2 or USB connector.

The ball is replaced by the optical system of the image sensor and LED in an ***optical mouse***. Such a mouse microcontroller calculates its movement if the motion of the LED beam is reflected from a surface with different spatial light absorption.

At each time, when the mouse is moved, it sends three consecutive code bytes, which satisfies the standard RS-232 connection, when it is connected to a PS/2 connection. These bytes define the state of the mouse and its relative moving distances along the x and y axes. In the state byte, the direction of the movement, the event that the movement speed exceeds a certain limit, as well as the button click event are set. If the mouse is moved without stopping, then it sends the codes approximately every 50 ms.

2.4.3 Video Display

The video display or monitor is based on the cathode-ray tube or a liquid crystal panel, and the control unit.

The electron gun in the ***Cathode-Ray Tube (CRT)*** generates a beam of electrons. Its vertical and horizontal deflection systems form a magnetic field in the path of the electron beam so that it is deflected and scans the phosphor screen according to the raster the left-right and top-down. The electrons, which are trapped in the screen, illuminate the phosphor to the brightness, which is proportional to the beam current. The modulation of the beam current in time corresponds to the brightness of the screen image. To obtain a color image, three electron guns and the phosphor screen of spots of three colors: red, green and blue (**R, G, B**) are used. The special mask front of the screen provides that the electrons of only one gun hit the phosphor of a certain color.

A colored dot on the screen is called a ***pixel*** (picture element — pixel). One pixel is often encoded by three-byte word. A byte in it sets the brightness of the respective color. The submission of information on the screen is controlled by the clock sync with the frequency F_c .

Thus, the words of the display pixels are supplied to the CRT modulator with the frequency F_c . This pixel stream is accompanied by a clock signal (SI), vertical VS and horizontal HS scanning signals. The signal parameters for a standard **VGA (Video graphic Adapter)** display with the image size of 640 by 480 pixels for a frame rate of 60 Hz and $F_c = 25$ MHz are presented in Table. 2.2.

The display control unit provides accurate horizontal and vertical scanning of the beams that are synchronized according to the vertical and horizontal synchronization impulses. It also provides the correct shape and sharpness of images and correct reflection of the color gamut.

Table 2.2. Timing parameters of the VGA display

Parameter	Vertical scanning VS			Horizontal scanning HS	
	Time	Cycles Fc	Rows	Time	Cycles Fc
Signal period	16.7 мс	416800	521	32 мкс	800
Time of the image	15.36 мс	384000	480	25.6 мкс	640
Impulse width	64 мкс	1 600	2	3.84 мкс	96

As a rule, there is the possibility of adjusting the display settings. The modern display control unit has the ability to transfer the information about the display state to the CPU using the I²C by bus. Using this information, the OS is able to perform the appropriate settings of the display controller.

The **Liquid Crystal Display (LCD)** panel includes a light source and a liquid crystal light modulator. The light source is based on a luminescent lamp, LED panel or electroluminescent lighting with uniform luminosity. The light modulator has a multilayer construction. Its main layers are polarizing filters, a layer of liquid crystal cells and the color filter layer in the form of raster triples segments red, blue and green colors, which number is equal to the number of pixels.

Polarizing filters, which are placed front and back of the LCD cells, have the orthogonal polarization planes. The liquid crystals between them in the usual condition rotate the polarization plane, whereby the light penetrates the cells. When a control signal is put to the cell, then the liquid crystal by an electric field ceases to rotate the polarization plane, and in the front polarization filter stops the light in the area of the cell. If the width of the control pulses and their amplitude are modulated, then the average filter throughput is adjusted over a wide range, whereby it is possible to set an arbitrary brightness and color of the corresponding pixel.

To control all the cells in parallel, the control signals are fed to them from the distributed decoder formed in a thin-film transistors layer based on

organic semiconductors. Therefore, the LCD panel is often named as the ***Thin Film Transistor (TFT)*** display.

In recent years, the displays based on the ***Organic Light Emitting Diodes (OLED)*** arrays, called OLED-displays, are widely spread. Their pixels are made up of tiny colored OLEDs.

The graphic or video signal is fed to the display of each type through a standard connection. The most common connection is the ***Video Graphic Array (VGA)*** connection, through which three analog signals of the colors R, G, B brightness, vertical VS, and horizontal HS sync signals, and signal DE of the pixel presence are transmitted in the display. When receiving a video signal over the VGA connection, the liquid crystal display converts it to the digital form by the analog-to-digital converters with a clock frequency F_c .

Another standard connection is a connector ***DVI (Digital Video Interface)***. This connection is a digital serial interface with the same principle of operation as in the PCI-E interface described above. Unlike VGA, the DVI connector provides a clear reproduction of the image with arbitrary dimensions (up to 4000·2000) and the high frame rate because the digital signal is not converted into an intermediate analog form.

The ***HDMI (High Definition Multimedia Interface)*** on the wiring diagram is similar to the DVI connection, but it is capable of transmitting not only the image but also coded sound and control signals. In addition, the image is limited to a rectangle of 1920 by 1080 pixels.

2.4.4 Display Controller

The ***display controller*** is a block that provides information for the display in accordance with the selected display mode. The structure of a typical display controller shown in Fig. 2.13.

When working in a graphics mode, the frame buffer stores the images that are displayed in the form of arrays with the same size as the display raster, for example, 640·480. A single word of this array contains the information about the brightness of the pixel colors with the appropriate coordinates. According to the settings of the display controller, the bit width of the pixel word varies from 8 to 32 bits.

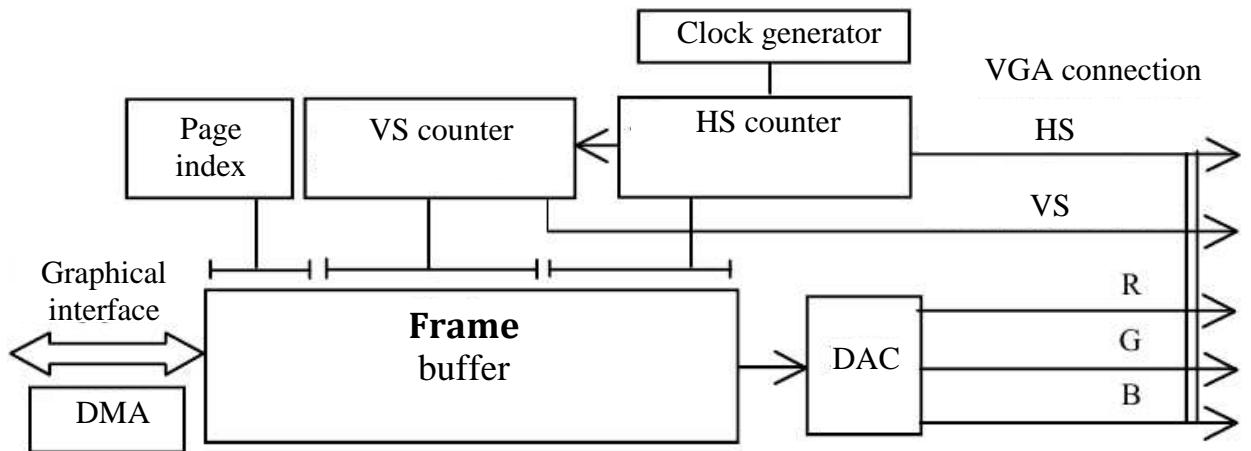


Fig.2.13. Display controller structure

The horizontal scanning Counter HS (see Fig. 2.13) iterates the pixels in the image line addresses with the clock frequency F_c . The information about a pixel brightness is read from the image buffer to the **Digital-Analog Converter** (DAC), where it is converted into the luminance signals of three colors. Also, this counter forms the **Horizontal Scanning** pulses (HS) and counting pulses for the **Vertical Scanning** (VS) counter. The last addresses the image line, which is pixel by pixel outputted through the DAC. It generates the VS impulses as well.

The frame buffer is a dual-port high-speed RAM, which stores several image frames in the corresponding pages. The dynamic change of these pages provides a continuous display of complex multimedia scenes on the screen. Therefore, the buffer volume can be hundreds of megabytes. Regardless of the

reflection on the display frame, the buffer provides to the access from the CPU through a graphical interface. The graphical interface of modern display controllers is based on the PCIe bus with up to 16 lanes. The parallel transfer of the large volumes of the image data is supported by the DMA mechanism.

The display controller may operate in a text mode, for example, in the OS boot mode of the processor operation. At that time, a separate text page buffer contains the text strings in the ASCII coding, at 80 characters per line. The one character word accommodates the character code byte and the byte of information about its color, background color, presence of the cursor in the position of the character. In a separate area of the memory (RAM and ROM), the character generator firmware is stored, which contains the character image table, wherein at the address of the ASCII character the bitmap table is stored, for example, in the rectangle of 16·8 pixels.

The operation of the character generator consists in that during the displaying the page on the screen, the words of the frame buffer bits are substituted by the character images, which are selected from ROM or RAM. The character generator RAM is filled when booting the operating system, depending on the preset number of the ASCII table, such as Cyrillic ASCII table.

The luminosity of the pixels is usually non-linear. For the luminance linearization, in the simplest case, the gamma correction is used. It consists in the exponentiation of the pixel brightness code V in the power of γ . For example, the 8 bit luminance value, that is supplied to the LCD display after the linearization is equal to $256(V/256)^\gamma$, where $\gamma = 2,0 — 2,6$ is set separately for each color channel and display type.

Modern display controllers have a complex structure. In order to accelerate the multimedia applications and graphic games, namely, for the acceleration of the implementation of standardized graphics functions from libraries such as DirectX, DivX, they include parallel application specific

processors. Their structure will be discussed in the next section. Also the RAM frame buffer of the large volume (several hundred megabytes) is used for storing intermediate data, scenes and textures, and is required for rapid implementation of these library functions.

2.4.5. Problems

- 1) How does the computer recognize that a button is pushed and then is released?
- 2) Consider the mouse has the resolution of 400 points per inch. Estimate the highest mouse movement speed, when it translates the distance codes without their saturations.
- 3) The display controller in the text mode stores each symbol in the two-byte word, the first of them is the ASCII code. For which purposes is the second byte used?
- 4) The LCD display has the HD resolution, i.e. the resolution of $1920 \cdot 1080$ pixels, the frame rate of 60 frames/s. Estimate the impulse frequency in its DVI interface.
- 5) The Ultra HD display has the resolution of $3840 \cdot 2160$, the frame rate of 120 frames/s. To transfer the videotream to it through the usual HD DVI interface, the compression algorithm is used. Estimate its compression rate.

2.5. Structure of a single processor computer

The connection of CPU and the PC environment via an interface system is considered as the computer structure. To understand such a connection, it is appropriate to consider the general structure of a typical single-processor computer, which is shown in Fig. 2.14. Here CPU is connected with the fast peripherals buffer (FPB) via a high-speed local bus, which is traditionally called as **FSB (Front Side Bus)**. The FSB bus is a wide bus, which includes an address bus and bidirectional data bus, wherein the bit width is usually 64.

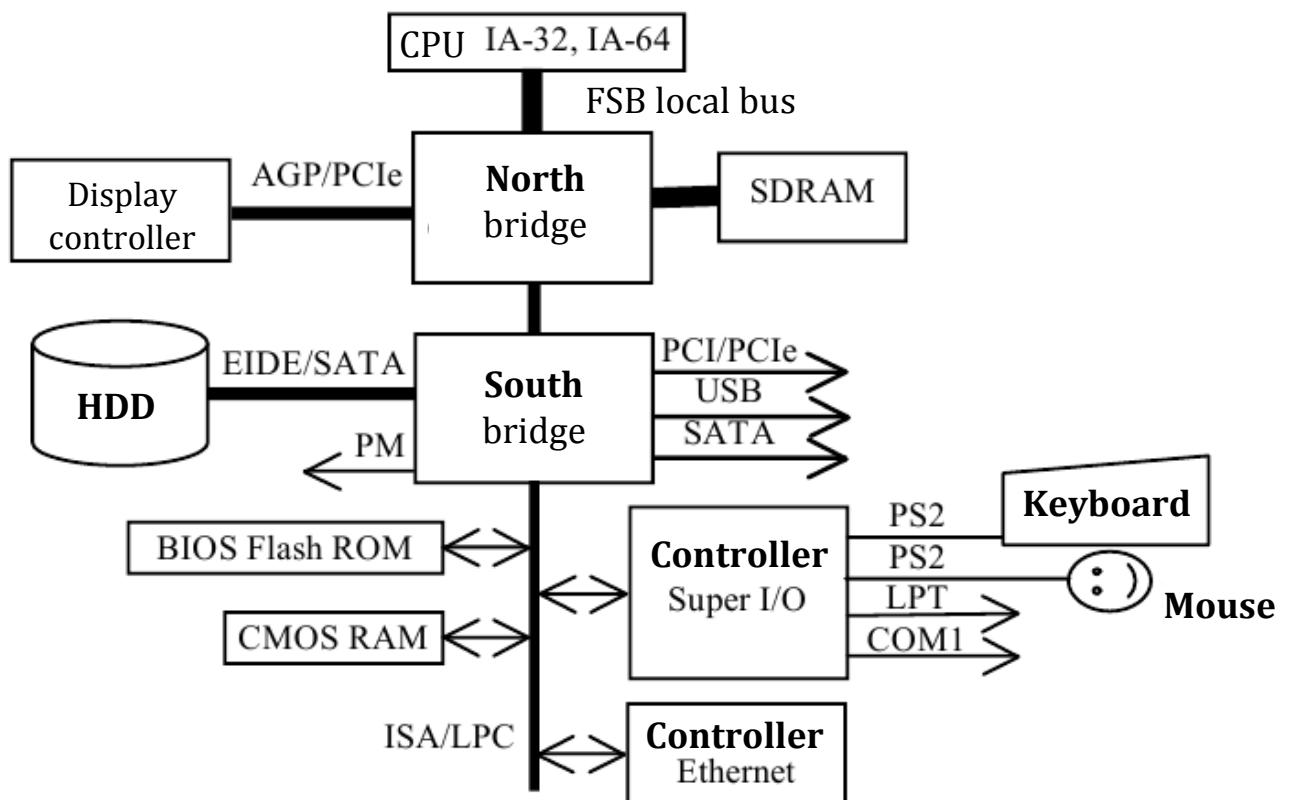


Fig. 2.14. Typical one processor computer structure

The bridge of the fast peripheral devices due to its position on the scheme is named as a **North Bridge**. The north bridge connects CPU with the RAM and the display controller, which transfer the largest data streams. It

connects CPU with the buffer of the slow peripheral devices too, which is named as a ***South Bridge***. The clock frequency F_T of the North Bridge distinguishes the computer speed. To increase the throughput of the FSB bus, the clock period is divided into four phases, during that the data are moving. As a result, the FSB bus has the throughput up to $4 \cdot 8 \cdot F_T$ bytes/s. The inner clock frequency of CPU is derived by the multiplication of the frequency F_T to the coefficient which is proportional to 0,5. For example, if $F_T = 400$ MHz, then the FSB throughput is equal to 12800 Mbytes / s, and the CPU clock frequency is equal to 3,6 GHz when the multiplication coefficient is 9.

The South Bridge contains a set of standard interfaces like PCI, PCIe, USB, SATA, EIDE. Through these interfaces, the CPU is connected to HDD and a set of peripheral devices. The bus **LPC** (Low Pin Count bus) is a separate interface, which is intended to connect the low-speed peripheral devices like ROM with the ***Base Input-Output System (BIOS)***, non-volatile RAM, which stores the basic settings of the computer structure (Setup RAM). The South Bridge also includes a real time clock, DMA controller, interrupt controller, and the power supply control unit with the Power Management input PM. This unit supervises the switching on of the computer, putting a value of the supply voltages for the motherboard chipset, saving computer power, calling interrupt on power failure.

The integral circuits of the North and South Bridges are usually designed and manufactured together and serve as a ***Chipset*** of the motherboard of the computer. Due to their high hardware complexity and high performance, they as well as the CPU dissipate a lot of energy. Therefore, for the reliable operation, they have the cooling systems. One of these bridges or both of them are integrated to the recent CPU chips due to the increased integration level and decreased energy consumption of them.

A chip of slower peripheral interfaces called the Super I/O controller is also connected to the LPC bus. Its outputs are such interfaces like PS/2, LPT (old-style printer bus), COM1, COM2 (serial ports on the RS232 standard). These interfaces connect the mouse, keyboard and other devices to PC. Often an audio multimedia interface is built in it.

2.6 Conclusions

In this teaching book, the computer architecture is considered as a model of a computer intended both for computer engineering and programming. As a basic architecture, the von Neuman architecture is studied. This architecture was a foundation of three computer generations. It is clear for achieving the essence of programmable computers. The most of the modern algorithmic languages are based on the principle of this architecture.

The explanations of the architectures are based on considering their main features like instruction set, data representation, addressing modes, interrupt system, etc. The architectures of peripheral devices and interfaces which connect them to the central processing units are considered as well.

The von Neuman architecture as such is intended for sequential, atomic implementation of the program instruction. It is not intended for the high-speed computer implementation. Moreover, it hampers the computer speed-up. Besides, its principle of handling both the data and the instructions in the same memory provides the most of computational errors and deadlocks, prevents the multiprogramming operation.

The contemporary architectures have the signs of the von Neuman architecture as well. But their features are expanded to the direction of the parallel computational processes. The organization of these processes can be many-sided and is permanently optimized. The goals of this optimization are high computer speed, multiprocessing, high reliability, low energy

consumption, programming conveniences. The computer architectures from the parallel processes point of view are studied in the next part of the teaching book.

BIBLIOGRAPHY

1. Гук М.Ю. Аппаратные средства IBM PC. Энциклопедия. — 3—е изд. — СПб.: Питер. —2006. —1072 с.
2. Гук М., Юров В. Процессоры Pentium 4, Athlon и Duron. — СПб.:Питер. — 2001. — 512 с.
3. Корнеев В., Киселев А.В. Современные микропроцессоры. — 3—е изд., перераб. и доп. — ВС. —2003. — 448 с.
4. Мельник А.О. Архітектура комп'ютера. — Луцьк: волинська обл. друк. —2008. —470с.
5. Орtega Дж. Введение в параллельные и векторные методы решения линейных систем. —М.: Мир. —1991. —365 с.
6. Организация ЭВМ. 5—е изд./ К. Хамахер и др. — СПб.:Питер. —2003. — 848 с.
7. Процюк Р.О., Корнейчук В.И., Кузьменко П.В., Тарасенко В.П. Компьютерная схемотехника (краткий курс). — К.:Корнійчук. —2006. —433 с.
8. Самофалов К.Г. и др. Цифровые ЭВМ: Теория и проектирование — 3—е изд.— К.:ВШ. —1989. —424 с.
9. Столлингс В. Структурная организация и архитектура компьютерных систем, 5—е изд.: Пер. с англ. — ВИ. — 2002. — 896 с.
10. Таненбаум Э. Архитектура компьютера. — СПб. : Питер. — 2002. — 704 с.

11. Bobda C. Introduction to Reconfigurable Computing Architectures, Algorithms, and Applications. — Springer. —2007. —359 p.
12. Bryant R.E., O'Hallaron D.R. Computer Systems. A Programmer's Perspective. —Prentice Hall. —2011. —1043 p.
13. El—Rewini H., Abd—El—Barr M. Advanced Computer Architecture and Parallel Processing. — USA, New Jersey, Canada: Wiley. —2005. — 272 p.
14. Giloi W. K. Rechnerarchitektur / Springer—Lehrbuch. —Springer. — 1993. — 485 p.
15. Patterson D.A., Hennessy J.L. Computer Organization Design. The Hardware/ Software Interface. —Elsevier, Morgan Kaufmann Publ. — 2005. —684 p.
16. Wolf W. Computers as Components. Principles of Embedded Computing System Design. —Elsevier Inc. , Morgan Kaufmann Publ. —2008. — 507 p.
17. Yiu J. The Definitive Guide to the ARM Cortex—M3. — Elsevier Inc., Newnes. — 2007. —359 p.

ANNEX 1

I8051 instruction set

Data moving instructions

Instruction name	Assembly code	OPC	B	Operation
Moving from register(n = 0 — 7) in accumulator	MOV A, Rn	11101rrr	1	(A) = (Rn)
Moving from address in accumulator	MOV A, ad	11100101	2	(A) = (ad)
Moving byte from DATA (i = 0, 1) in accumulator	MOV A, @Ri	1110011i	1	(A) = ((Ri))
Loading of a constant in accumulator	MOV A, #d	01110100	2	(A) = #d
Moving accumulator to register	MOV Rn, A	11111rrr	1	(Rn) = (A)
Moving byte with address b register	MOV Rn, ad	10101rrr	2	(Rn) = (ad)
Loading of a constant in register	MOV Rn, #d	01111rrr	2	(Rn) = #d
Moving accumulator to direct address	MOV ad, A	11110101	2	(ad) = (A)
Moving register to direct address	MOV ad, Rn	10001rrr	2	(ad) = (Rn)
Moving byte in direct address to direct address	MOV ad, ads	10000101	3	(add) = (ads)
Moving byte from DATA to direct address	MOV ad, @Ri	1000011i	2	(ad) = ((Ri))
Moving a constant to direct address	MOV ad, #d	01110101	3	(ad) = #d
Moving accumulator to DATA	MOV @Ri, A	1111011i	1	((Ri)) = (A)
Moving byte in address to DATA	MOV @Ri, ad	0110011i	2	((Ri)) = (ad)
Moving of a constant to DATA	MOV @Ri, #d	0111011i	2	((Ri)) = #d
Loading in data pointer	MOV DPTR, #d16	10010000	3	(DPTR) = #d16
Moving in accumulator a byte from CODE	MOVC A, @A + DPTR	10010011	1	(A) = ((A) + (DPTR))
Moving in accumulator a byte from CODE	MOVC A, @A + PC	10000011	1	(PC) = (PC) + 1 (A) = ((A)+(PC))
Moving in accumulator a byte from XDATA	MOVX A, @Ri	1110001i	1	(A) = ((Ri))
Moving in accumulator a byte from XDATA	MOVX A, @DPTR	11100000	1	(A) = ((DPTR))
Moving in XDATA from accumulator	MOVX @Ri, A	1111001i	1	((Ri)) = (A)
Moving in XDATA from accumulator	MOVX @DPTR, A	11110000	1	((DPTR)) = (A)
Loading in stack	PUSH ad	11000000	2	(SP) = (SP) + 1 ((SP)) = (ad)
Fetching from stack	POP ad	11010000	2	(ad) = (SP) (SP) = (SP) - 1
Exchange accumulator from pericртом	XCH A, Rn	11001rrr	1	(A) <--> (Rn)
Exchange accumulator from байтом with address	XCH A, ad	11000101	2	(A) <--> (ad)

Arithmetic instructions

Instruction name	Assembly code	OPC	B	Operation
Adding register to accumulator (n = 0 — 7)	ADD A, Rn	00101rrr	1	(A) = (A) + (Rn)
Adding byte with address to accumulator	ADD A, ad	00100101	2	(A) = (A) + (ad)
Adding byte from DATA to accumulator (i = 0, 1)	ADD A, @Ri	0010011i	1	(A) = (A) + ((Ri))
Adding of a constant to accumulator	ADD A, #d	00100100	2	(A) = (A) + #d
Adding register and carry to accumulator	ADDC A, Rn	00111rrr	1	(A) = (A) + (Rn) + (C)
Adding byte with address and carry to accumulator	ADDC A, ad	00110101	2	(A) = (A) + (ad) + (C)
Adding byte from DATA and carry to accumulator	ADDC A, @Ri	0011011i	1	(A) = (A) + ((Ri)) + (C)
Adding of a constant and carry to accumulator	ADDC A, #d	00110100	2	(A) = (A) + #d + (C)
Decimal correction of accumulator	DA A	11010100	1	If $A_{0-3} > 9 \vee ((AC)=1)$, then $A_{0-3}=A_0-3+6$, $A_{4-7} > 9 \vee ((C)=1)$, then $A_{4-7}=A_4-7+6$
Subtraction register and borrow from accumulator	SUBB A, Rn	10011rrr	1	(A) = (A) — (C) — (Rn)
Subtraction byte in address and borrow from accumulator	SUBB A, ad	10010101	2	(A) = (A) — (C) — ((ad))
Subtraction byte DATA and borrow from accumulator	SUBB A, @Ri	1001011i	1	(A) = (A) — (C) — ((Ri))
Subtraction of a constant and borrow from accumulator	SUBB A, #d	10010100	2	(A) = (A) — (C) — #d
Incrementing accumulator	INC A	00000100	1	(A) = (A) + 1
Incrementing register	INC Rn	00001rrr	1	(Rn) = (Rn) + 1
Incrementing byte with address	INC ad	00000101	2	(ad) = (ad) + 1
Incrementing byte in DATA	INC @Ri	0000011i	1	((Ri)) = ((Ri)) + 1
Incrementing data pointer	INC DPTR	10100011	1	(DPTR) = (DPTR) + 1
Decrementing accumulator	DEC A	00010100	1	(A) = (A) - 1
Decrementing register	DEC Rn	00011rrr	1	(Rn) = (Rn) - 1
Decrementing byte with address	DEC ad	00010101	2	(ad) = (ad) - 1
Decrementing byte in DATA	DEC @Ri	0001011i	1	((Ri)) = ((Ri)) - 1
Multiplication accumulator to register B	MUL AB	10100100	1	(B)(A) = (A)*(B)
Division accumulator to register B	DIV AB	10000100	1	(A).(B) = (A)/(B)

OPC – OPeration Code

B – instruction length, bytes

Logic instructions

Instruction name	Assembly code	OPC	B	Operation
Logic AND accumulator and register	ANL A, Rn	01011r rr	1	(A) = (A) \wedge (Rn)
Logic AND accumulator and byte with address	ANL A, ad	01010101	2	(A) = (A) \wedge (ad)
Logic AND accumulator and byte from DATA	ANL A, @Ri	0101011 i	1	(A) = (A) \wedge ((Ri))
Logic AND accumulator and constant	ANL A, #d	01010100	2	(A) = (A) \wedge #d
Logic AND byte with address accumulator	ANL ad, A	01010010	2	(ad) = (ad) \wedge (A)
Logic AND byte with address and a constant	ANL ad, #d	01010011	3	(ad) = (ad) \wedge #d
Logic OR accumulator and register	ORL A, Rn	01001r rr	1	(A) = (A) \vee (Rn)
Logic OR accumulator and byte with address	ORL A, ad	01000101	2	(A) = (A) \vee (ad)
Logic OR accumulator and byte from DATA	ORL A, @Ri	0100011 i	1	(A) = (A) \vee ((Ri))
Logic OR accumulator and a constant	ORL A, #d	01000100	2	(A) = (A) \vee #d
Logic OR byte with address and accumulator	ORL ad, A	01000010	2	(ad) = (ad) \vee (A)
Logic OR byte with address and a constant	ORL ad, #d	01000011	3	(ad) = (ad) \vee #d
Exclusive OR accumulator and register	XRL A, Rn	01101r rr	1	(A) = (A) \oplus (Rn)
Exclusive OR accumulator and byte with address	XRL A, ad	01100101	2	(A) = (A) \oplus (ad)
Exclusive OR accumulator and byte from DATA	XRL A, @Ri	0110011 i	1	(A) = (A) \oplus ((Ri))
Exclusive OR accumulator and a constant	XRL A, #d	01100100	2	(A) = (A) \oplus #d
Exclusive OR byte with address and accumulator	XRL ad, A	01100010	2	(ad) = (ad) \oplus (A)
Exclusive OR byte with address and a constant	XRL ad, #d	01100011	3	(ad) = (ad) \oplus #d
Resetting accumulator	CLR A	11100100	1	(A) = 0
Inversion of accumulator	CPL A	11110100	1	(A) = (NOT A)
Shifting accumulator left cyclically	RL A	00100011	1	(An+1) = (An), n = 0 ... 6, (A0) = (A7)
Shifting accumulator left through carry bit	RLC A	00110011	1	(An+1) = (An), n = 0 ... 6, (A0) = (C), (C)=(A7)
Shifting accumulator right cyclically	RR A	00000011	1	(A n) = (A n+1), n = 0...6, (A7) = (A0)
Shifting accumulator right through carry bit	RRC A	00010011	1	(A n) = (A n+1), n = 0...6, (A7) = (C), (C) = (A0)
Swapping nibbles in accumulator	SWAP A	11000100	1	(A0—3) <--> (A4—7)

Control flow instructions

Instruction name	Assembly code	OPC	B	Operation
Long jump in PRAM	LJMP ad16	00000010	3	(PC) = ad16
Absolute jump in range of 2 Kbytes	AJMP ad11	a ₁₀ a ₉ a ₈ 00001	2	(PC) = (PC) + 2 (PC ₀ —10) = ad11
Short relative jump in range of 256 bytes	SJMP rel	10000000	2	(PC) = (PC) + 2 (PC) = (PC) + rel
Indirect relative jump	JMP @A+DPTR	01110011	1	(PC) = (A) + (DPTR) (PC) = (PC) + 2,
Jump, if acc. – is zero	JZ rel	01100000	2	if (A) = 0, then (PC) = (PC) + rel
Jump, if acc. Is not zero	JNZ rel	01110000	2	(PC) = (PC) + 2, if (A) ? 0, then (PC) = (PC) + rel
Jump, if carry =1	JC rel	01000000	2	(PC) = (PC) + 2, if (C) = 1, then (PC) = (PC) + rel
Jump, if carry =0	JNC rel	01010000	2	(PC) = (PC) + 2, if (C) = 0, then (PC) = (PC) + rel
Jump, if bit =1	JB bit, rel	00100000	3	(PC) = (PC) + 3, if (b) = 1, then (PC) = (PC) + rel
Jump, if bit =0	JNB bit, rel	00110000	3	(PC) = (PC) + 3, if (b) = 0, then (PC) = (PC) + rel
Jump, if bit =1, then bit:=0	JBC bit, rel	00010000	3	(PC) = (PC) + 3, if (b) = 1, then (b) = 0 and (PC) = (PC) + rel
Decrementing register and jump, if not 0	DJNZ Rn, rel	11011r rr	2	(PC) = (PC)+2, (Rn)=~(Rn), if (Rn)≠0, then (PC)=(PC)+rel
Decrementing byte with address and jump, if not zero	DJNZ ad, rel	11010101	3	(PC) = (PC) + 2, (ad) = (ad) —1, if (ad) ? 0, then (PC) = (PC) + rel
Compare acc. with a byte with address and jump, if not equal	CJNE A, ad, rel	10110101	3	(PC) = (PC) + 3, if (A) ? (ad), then (PC) = (PC) + rel, if (A) < (ad), then (C) = 1, otherwise (C) = 0
Compare accumulator with a constant and jump, if not equal	CJNE A, #d, rel	10110100	3	(PC) = (PC) + 3, if (A) ? #d, then (PC) = (PC) + rel, if (A) < #d, then (C) = 1, otherwise (C) = 0
Compare register with a constant and jump, if not equal	CJNE Rn, #d, rel	10111r rr	3	(PC) = (PC) + 3, if (Rn) ? #d, then (PC) = (PC) + rel, if (Rn) < #d, then (C) = 1, otherwise (C) = 0
Compare byte in DATA with a constant and jump, if not equal	CJNE @Ri, #d, rel	1011011 i	3	(PC) = (PC) + 3, if ((Ri)) ? #d, then (PC) = (PC) + rel, if ((Ri)) < #d, then (C) = 1, otherwise (C) = 0
Long subprogram call	LCALL ad16	00010010	3	(PC) = (PC) + 3, (SP) = (SP) + 1, ((SP)) = (PC ₀ —7), (SP) = (SP) + 1, ((SP)) = (PC _{8—15}), (PC) = ad16
Absolute subprogram call in range 2 Kbytes	ACALL ad11	a ₁₀ a ₉ a ₈ 10001	2	(PC)=(PC)+2, (SP)=(SP)+1, ((SP))=(PC0—7), (SP)=(SP)+1, ((SP)) = (PC _{8—15}), (PC _{0—7}) = ad11 (PC _{8—15}) = ((SP)),
Return from subprogram	RET	00100010	1	(SP) = (SP) — 1, (PC0—7) = ((SP)), (SP) = (SP) — 1
Return from interrupt	RETI	00110010	1	(PC _{8—15})=(((SP))), (SP)=(SP)—1, (PC _{0—7}) = ((SP)), (SP) = (SP) — 1
No OPeration	NOP	00000000	1	(PC) = (PC) + 1

Bit handling instructions

Instruction name	Assembly code	OPC	B	Operation
Setting in 0 carry	CLR C	11000011	1	(C) = 0
Setting in 0 a bit	CLR bit	11000010	2	(b) = 0
Setting in 1 carry	SETB C	11010011	1	(C) = 1
Setting in 1 bit	SETB bit	11010010	2	(b) = 1
Complement of carry	CPL C	10110011	1	(C) = (C)
Complement of bit	CPL bit	10110010	2	(b) = (b)
Logic AND bit and carry	ANL C, bit	10000010	2	(C) = (C) \wedge (b)
Logic AND of negated bit and carry	ANL C, /bit	10110000	2	(C) = (C) \wedge (b)
Logic OR of bit and carry	ORL C, bit	01110010	2	(C) = (C) \vee (b)
Logic OR of negated bit and carry	ORL C, /bit	10100000	2	(C) = (C) \vee (b)
Moving bit to carry	MOV C, bit	10100010	2	(C) = (b)
Moving carry to bit	MOV bit, C	10010010	2	(b) = (C)