JPRS-UCC-89-004
27 JUNE 1989

FOREIGN BROADCAST INFORMATION SERVICE

JPRS Report

Science & Technology

USSR: Computers

JPRS-UCC-89-004

27 JUNE 1989

SCIENCE & TECHNOLOGY

USSR: COMPUTERS

CONTENTS

Molecular Computer: Hopes and Doubts

18630125a Moscow KHIMIYA I ZHIZN in Russian No 11, Nov 88 pp 14-21

[Article by N.G. Rambidi, doctor of chemical sciences: "A Molecular Computer: Hopes and Doubts," published under the rubric "Problems and Methods of Modern Science"; the first paragraph is an epigraphic quotation]

[Text] The only difference between optimism and pessimism is in setting the date of the end of the world. — Stanislaw Jerzy Lec

In December of 1959, when electronic technology was making its first timid steps toward microminiaturization, the famous theoretical physicist R. Feynman presented a paper entitled "Still a Lot of Room at the Bottom" at the annual meeting of the American Physical Society. He made a detailed analysis of the potential applications of the microscopic phenomena and superminiaturized devices, up to devices of atomic dimensions for electronic technology. Anticipating much of the current realities, Feynman spoke of the opening prospects for manipulating huge amounts of information with an active approach to solution of important research problems, a more profound understanding of the informational essence of biological processes, etc. Just under three decades has passed since then. Today, technology and engineering, planning and record keeping that support the functioning and development of modern society are unthinkable without the "information industry" which relies on the methods and tools of modern electronics. Rapid progress in this field has been and still is based on inorganic semiconductor materials and the fabrication of microminiature electronic products consisting of two-dimensional (planar) structures. This technology made desktop computers with huge random-access memories widely available and made possible the creation of computer complexes with information processing speeds of up to 100 million arithmetic operations per second.

As they say, when things are going well, why look for greener pastures? However, although the potential of the planar semiconductor technology is far

from being exhausted, many scientists today are seeking new solutions and essentially new materials that will further expand the capabilities of information-processing devices. The need for expanding these capabilities has been growing rapidly over the past few years.

Well and Poorly Defined Problems

Practical computation problems can be divided into two main categories: well defined and poorly defined.

For solving a well-defined problem it is possible to devise an algorithm that is a sequence of unequivocally defined operations; such an algorithm not only allows finding a solution, but can also answer the question whether what is found is a solution.

Problems in this category include calculations for and planning of physical, chemical and mathematical experiments, numeric solutions of all kinds of complex equations, a variety of problems in economic forecasting, industrial process control and many other practically important areas. Even though the capacities of existing computers may fall short of certain well-defined but extremely complex problems, rapid progress in computers with conventional architecture (operating according to the principle of sequential information processing) with growing operation speeds of computer elements and random-access memory capacity as well as the introduction of new computers with progressive architecture (operating according to the principle of parallel processing) give hope that the degree of complexity of problems that can be solved in this category will grow continuously.

For poorly defined problems, an algorithm may be developed, but a priori there is no strict informational criteria to determine that the problem has been solved; this group includes some problems for which no universal solution algorithms are possible.

Most problems that one has to solve in daily life belong to the poorly defined group; the fact that they exist means that we need artificial intelligence systems. A simple example is the choice of a move in a game of chess which, as detailed studies show, is never predetermined unambiguously.

Modern computers have trouble handling problems of this kind and sometimes cannot cope with them. The difficulties seem to be intrinsic and presumably connected with the properties of conventional components (incidentally, these are the same properties that make them so suitable for solution of well-defined problems). Specifically, these properties include the simplicity of elementary logical operations, the limited set of such operations, and the rigidity (invariability) of operations executed by each component. For this reason, scientists in recent years have seriously

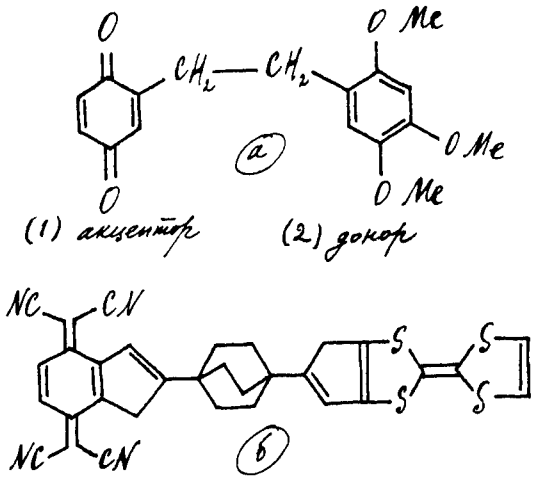


Figure 1. Aviram and Ratner molecule (*a*) and a real synthesized rectifier molecule (*b*).

Key: 1 — acceptor; 2 — donor.

discussed the possibility of using single molecules or relatively small groups of molecules as logical elements of computer devices as an alternative to existing semiconductor components.

A Rectifier Molecule

The existence of relatively weak intermolecular bonds is a characteristic property of organic and element-organic substances in a condensed state (molecular crystals and polymers, ordered films and liquid crystals). An organic molecule in a condensed phase largely retains its individuality; as a result, the properties of an organic crystal are a peculiar tangle of properties of individual molecules and the collective properties of the crystal. Large organic molecules, such as proteins or nucleic acids, are even more peculiar. E. Schrödinger, one of the the greatest physicists of modern times, aptly called such molecules aperiodic crystals. These properties give us hope that on the basis of organic and element-organic substances, we will be able to build information-logical devices with essentially new capabilities which will be different from conventional devices based on inorganic semiconductors.

The first publication to describe a possible molecular component of a computer device was a 1974 paper by the American chemists A. Aviram and M. Ratner entitled "Molecular Rectifier." They described a molecule built of two fragments separated by a system of methylene bridges (fig. 1); one fragment (donor) can supply electrons while the other fragment (acceptor) can receive electrons. If a potential difference is applied to the opposite ends of such molecule, the electron conduction from acceptor to donor was

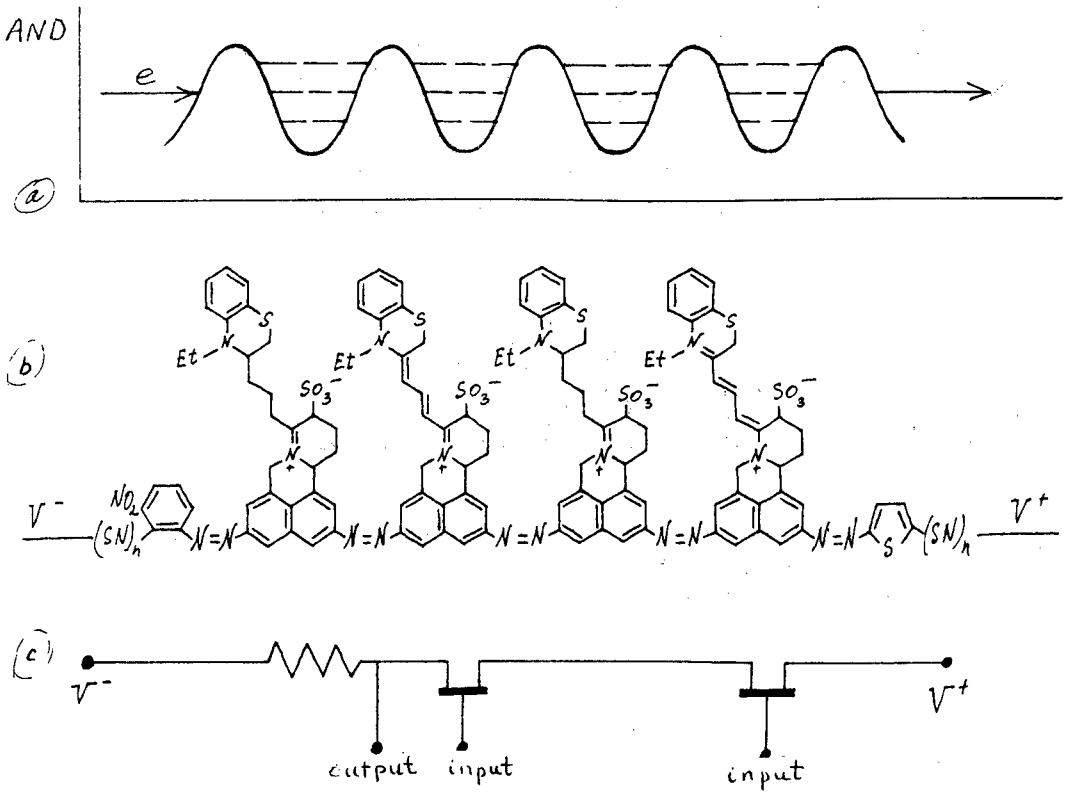


Figure 2. A NOT-AND molecular logical element: *a* — the system of barriers and potential wells traversed by an electron when its energy matches one of the permitted energetic levels inside the well; *b* — structure of a molecule capable of performing a logical operation; *c* — equivalent electrical circuit of a molecular element.

calculated to be quite different from that in the opposite direction. In other words, such a molecule would let electrons through with different degrees of ease in different directions, i.e., it could operate as a molecular analog of a rectifying component.

Wells and Solitons

More specific proposals concerning the development of a molecular component base were advanced in the past few years by F. Carter of the US Navy Research Labs. The ideal was to utilize resonance tunnel conductivity in a system of potential wells (i.e., conductivity based on an electron's capacity of jumping in space from one energetically efficient position to another) and controlling this conductivity by shifting energy levels in one of the wells.

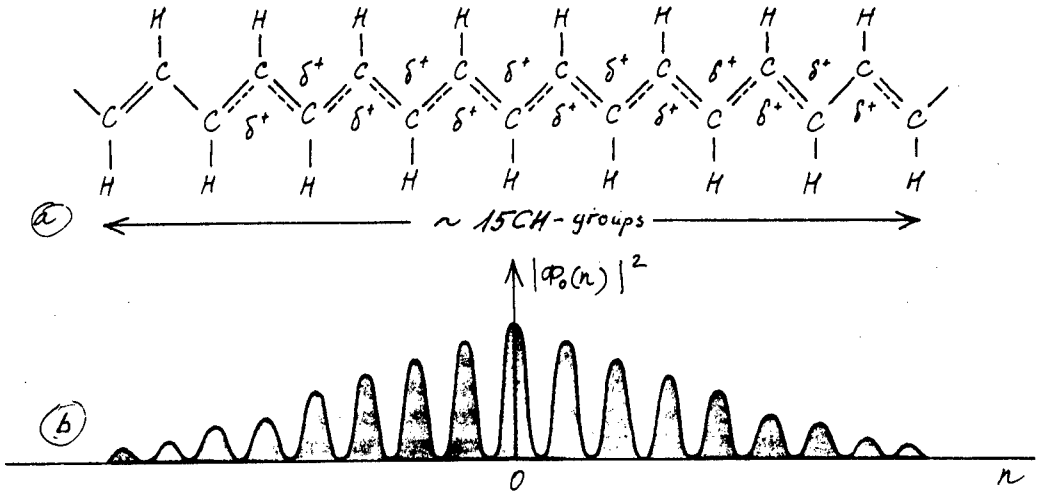


Figure 3. Formation scheme of a soliton: a local excitation traveling along a chain (*a*) and electron density distribution in a soliton (*b*).
Key: 1 — ≈15 CH-groups

In a potential well (or a set of wells) electrons can exist only at certain permitted energetic levels. If the energy of an electron coincides with the energy of a level inside the well, the electron can jump into the well unobstructed, as though bypassing the energy barrier.

Figure 2 shows a structure of a molecular component which makes use of this effect to implement the logical NOT-OR function. The positively charged aromatic heterocyclic groups operate as potential wells for electrons; the diazo groups linking them function as barriers between the wells. A conduction electron can pass freely along a chain of potential wells (i.e., along the molecule) if its energy coincides with energy of a level in a well. If an additional electron is fed to one of the inputs of such a molecular device, the entire electron structure of the fragment will be changed: the shape of its potential well and the precision of energy levels in it will be altered, greatly reducing the system's capacity for transmitting a conduction electron.

Computer networks can be built also on the basis of the soliton mechanism of transmitting excitation along a protracted molecular chain (i.e., the motion of an energy quantum — a soliton — along a molecule); along its path, the excitation modifies the electron structure of the molecule.

The soliton mechanism was first proposed by A. S. Davydov, a member of the Ukrainian Academy of Sciences, to explain the transmission of excitation along a chain in protein molecules. Carter then proposed using this mechanism for transmission of soliton excitations that can arise in trans-polyacetylene molecules and their use in switching components.

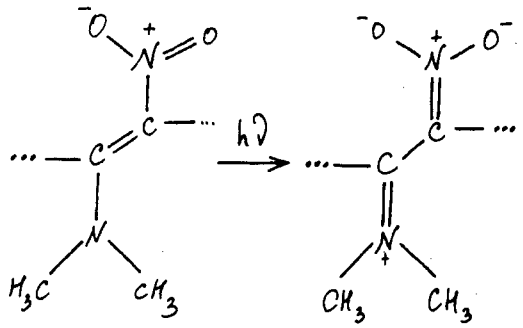


Figure 4. Photoisomerizing group capable of controlling the passage of solitons through a conjugated chain.

Several investigators demonstrated that structural defects can arise in trans-polyacetylene and move along a chain virtually without energy scattering (fig. 3); importantly, the transmission of the defect changes the alternation sequence of double and ordinary bonds. In addition, positively and negatively charged solitons can be formed in transpolyacetylene besides a neutral defect.

The concept of soliton switching developed by Carter is based on linear conjugate systems where soliton excitation energy can be transmitted. Control groups are built into such systems which can break at the coupling chain and thus block the possible transmission of a soliton. An example of such a group is shown in fig. 4: light isomerizes it and the central double bond is replaced by an ordinary bond. If such a group is built into a conjugate system of double bonds, we can initiate in the control fragment isomerization reaction before the transmission of a soliton and modify the structure of adjacent portion of the chain; as a result, the soliton could not be transmitted. Conversely, after the passage of the soliton through the main chain, that is after double bonds are replaced by ordinary bonds, no photoisomerization will take place, which can be utilized to register the soliton. Carter and other scientists have suggested various complex conjugated molecular networks with all kinds of built-in control groups; such networks, in principle, could execute highly complex logical operations.

Various other molecular systems that can be used in information processing devices and memories of an extremely high capacity have been studied in the past few years. We should bear in mind that they are all speculative and usually not supported even by indirect experiments. None of the investigators examine the difficulties of chemical synthesis of such devices, although even in elementary cases such difficulties must be formidable. Finally, these schemes simulate components of conventional sequential digital computers.

Another Fly in the Ointment

A discussion of the feasibility of a molecular component base must include an evaluation of its possible advantages over existing technologies. It should be considered also that even most optimistic experts do not expect first molecular information-logical devices to appear before the late 1990's. An evaluation of the advantages of a molecular component base should, therefore, make comparisons with the future devices likely to appear in the next decade rather than with the existing semiconductors.

A typical example of a modern integrated circuit is the Motorola 68020 (United States) microchip with a speed of 2.5 million operations/s; this processor contains 192,000 elements arranged on an 80 mm^2 crystal. The degree of microminiaturization of semiconductor devices has been growing rapidly in recent years and we can expect that over the next decade the typical dimensions of elements could be reduced by about an order of magnitude, down to 0.25 μm. In addition, it is expected that innovative technologies will make it possible to build semiconductor devices of 0.01-0.025 μm.

Large biological molecules such as proteins are built of several (from one to four) structural fragments — domains, each consisting of 100-150 amino acid residues rapped up into a globule of some 25 Å across. Accordingly, an average molecular logical element should be about 100 Å large or 0.01 μm, which is not all that different from the semiconductor elements of the future.

With respect to memories, the physical limit of placement of memory elements on magnetic domains is on the order of 0.05 μm, which would create a memory with a density of 4×10^{10} elements/cm^2. On the other hand, the effective packing density of a molecular memory is similar: 10^{10}-10^{11} elements/cm^2.

The speed of modern semiconductor devices is characterized by switching delay, which in best devices today is 5×10^{-10} s; in principle, the delay could be reduced to 10^{-14} s. With molecular elements, the speed is limited by the time of a light quantum absorption by a molecule in the basic electron state, which is estimated at 10^{-15} s.

A comparison of the expected characteristics of molecular elements and the semiconductor elements of the future (assuming that their functions and architectures are similar) suggests that transition to molecular element base would indeed increase the degree of microminiaturization. However, considering the expected progress in semiconductor technology, this improvement will be much less dramatic than one could expect from an superficial appraisal.

The Cell as a Computer

Despite the absence of decisive physical advantages of molecular logical elements over semiconductor structures, even today there are two promising aspects of molecular systems. One is the study of the architecture of natural molecular systems and the information processing methods implemented on their basis that could be utilized to devise promising semiconductor devices; the second area is the harnessing for the purposes of information processing processes that occur during restructuring of large molecular structures acted upon by physical factors.

The processes that take place with participation of complex biological molecules correspond to logical functions equivalent to a large combination of elementary logical operations such as NOT, AND-AND, OR-OR carried out by semiconductor elements. For example, during the course of an enzymatic reaction, within about 100 μs, the enzyme molecule recognizes a specific object among a large number of surrounding molecules: the molecule of the substrate. It bonds that molecule, helps its rapid transmutation into the product and releases the molecule of the product. The amount of heat released during one such complex logical act is just 10-100 kT (where k is the Boltzmann constant and T is absolute temperature) which is much less than the amount of heat $\approx 10^{10}$ kT per cycle released by typical semiconductor transistor elements.

The complexity of logical functions performed by large biological molecules in their vital activities gives us hope that on their basis it will be possible to build information-logical devices capable of solving problems beyond the reach of modern computers or even computers of the next generation.

The feasibility of this path of development of computer technology can be demonstrated by a hypothetical model device which became known as the Brown computer.

An example of such a Brown machine is the process of copying information encoded in a deoxyribonucleic acid (DNA) molecule and its transfer to a ribonucleic acid (RNA) molecule by means of a reaction accomplished by a special enzyme (fig. 5). The function of the enzyme is to get attached to a DNA molecule initiating the reading process, determine which base (adenine, guanine, cytosine or thymine) is to be next, find a transport molecule of the required base in the immediate environment and help that base to be attached to the growing RNA strand.

The process constitutes a complex sequence of information-logical operations. It is reversible, since an enzyme can not only attach a new base to RNA but also split off a previously attached base and return one step back in the

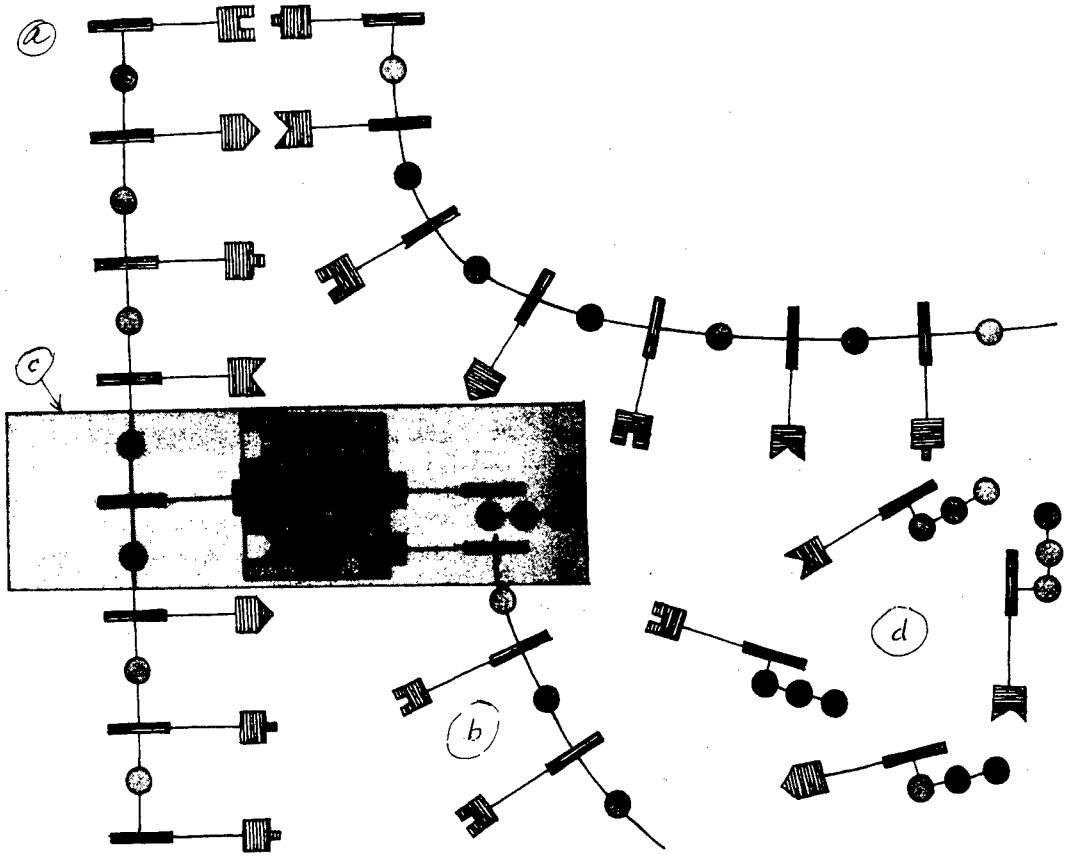


Figure 5. Processes of information reading from DNA: *a* — DNA molecule; *b* — RNA molecule being synthesized; *c* — enzyme complex; *d* — free nucleotides.

sequence of DNA bases. The driving force bringing the system from one state to another is the increase or decrease of reactive component concentrations.

The sequence of logical operations executed during DNA information reading can be represented by a flowchart (fig. 6) not unlike one for a computer program. The blocks in the chart represent extremely complex logical actions (for example, pattern recognition), which are the object of artificial intelligence research. In case of enzymes, pattern recognition occurs by means of change of the internal structure of the macromolecule, which adjusts itself optimally to a particular task. It is an elementary example of a functionally flexible molecular device, whose potential capabilities and efficiency are actively debated today.

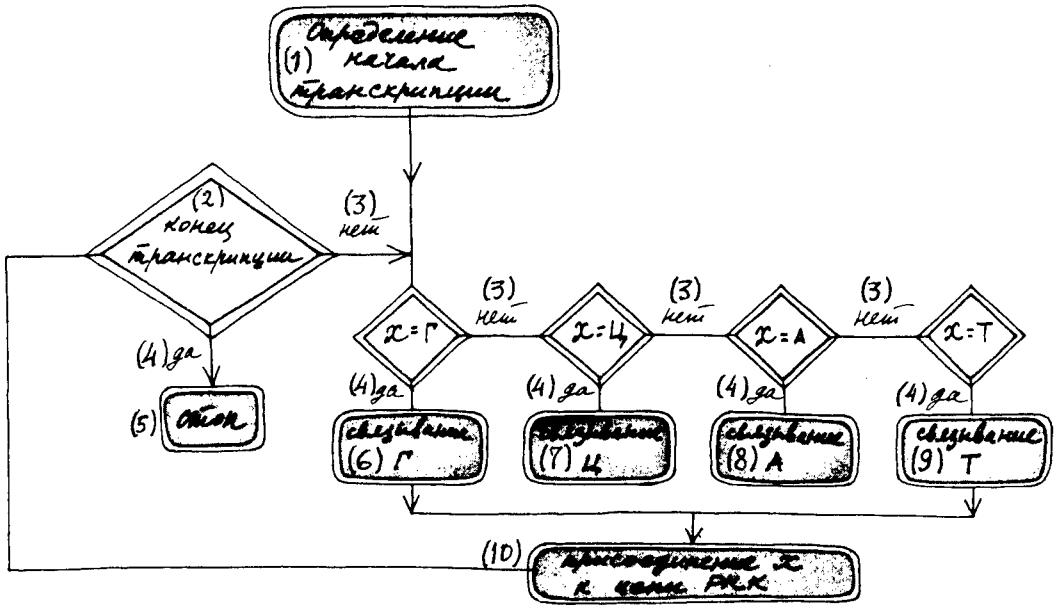


Figure 6. Flowchart of logical operations performed during the course of information reading from DNA.

Key: 1 — setting the start of transcription; 2 — end of transcription; 3 — no; 4 — yes; 5 — stop; 6 — G-bonding; 7 — C-bonding; 8 — A-bonding; 9 — T-bonding; 10 — attaching x to RNA strand.

No Quantum Leap

A detailed analysis of the computational capabilities that could be derived when large biological molecules are used as the component base was made recently by the famous american biophysicist M. Conrad. In his study he proceeded from an exclusion principle formulated by him at an axiomatic level: an information-logical system cannot be effectively programmable, capable of evolution through variability in selection and computationally effective all at the same time.

Never trying to prove this principle, Conrad gives three general considerations to support it. First, a universal rigidly programmable system should be capable of handling problems in all physical situations and could not be as effective as a system designed specifically for a narrow group of tasks. Secondly, a rigid program assumes sequential execution of operations that is an inefficient use of computer resources. Thirdly, a rigid program can fail easily, and the system does not allow for a gradual change of functions in case of a gradual change of the programs structured.

The existing information-logical devices on a semiconductor component base, which efficiently solve well-defined problems, are based on the principle of

a rigid structural organization combined with effective programmability. On the other hand, large biological molecules which take part in vital functions and perform complex sequences of logical operations (which lead to successful solutions of poorly defined programs) are narrowly specialized, functionally flexible systems capable of variability and the maximum efficiency in execution of their functions.

Speaking of unconventional development trends in computer technology, we should acknowledge that the idea of functionally flexible molecular information-logical devices is one that primarily deserves practical implementation.

If we succeed in building computers with flexibly readjustable inner structure on the basis of large organic molecules, it will be essentially different from even most promising devices based on semiconductors. By the same token, molecular and semiconductor computers will be used to solve different kinds of problems and will not compete with but effectively supplement each other.

List of Suggested Reading

1. Rambidi, N.G., and V.M. Zamalin, "Molecular Microelectronics: Sources and Hopes," ZNANIYE. SERIYA FIZIKA, No 11, 1985.

2. Valiyev, K.A., MIKROELEKTRONIKA: DOSTIZHENIYA I PUTI RAZVITIYA (Microelectronics: Accomplishments and Development Paths) Moscow, Nauka, 1986.

3. Davydov, A.S. SOLITONY V BIOENERGETIKE (Solitons in Bioenergetics) Kiev, Naukova Dumka, 1986.

4. Lord, N., et al., COMPUTERS OF THE FUTURE [Russian translation], Moscow, Mir, 1987.

5. Rambidi, N.G., et al., "Molecular Component Base of Future Information-Logical Devices," VINITI, ITOGI NAUKI I TEKHNIKI, SERIYA ELEKTRONIKA (Moscow), Vol 22, 1987.

Replica of Human Intelligence

18630125b Moscow KHIMIYA I ZHIZN in Russian No 11, Nov 88 pp 22-26

[Article by O.I. Larichev, doctor of technical sciences: "A Replica of Human Intelligence," published under the rubric "Problems and Methods of Modern Science"]

[Text] ## 1. A Thinker or an Idiot Savant?

Anyone who has to deal with computers is struck by their fast growth. Computers are getting smaller and more powerful; they are getting faster and cheaper (although, alas, not in the Soviet Union). They are spreading to ever new fields — from household appliances to space research. This leads to a natural question: where does it all lead to? Will the science fiction fantasy of an artificial intelligence superior to human mind come true? The debates have been going on for years whether a computer is (or will be) able to think. The argument was about whether a computer is a "creator" or an "idiot savant." I believe these debates had a positive result: they made scientists pause and ponder about the ways we think. In principle, a computer can be built to accomplish tasks as well as a human but in an entirely different way. This was the path adopted by certain serious investigators who supported their choice by arguing that, in engineering, machines and devices do not simulate human motions but execute physical work much more effectively than a man can do. Other scientists (including the Nobel Prize winner G. Simon) have been trying to gain insight into the information processing as performed by man and to build computer programs utilizing certain human mental techniques. The rationale of this approach is also understandable: before trying to surpass a human, the machine should learn to do his job as well as he does.

2. Our Knowledge and Skills

Whatever a man can do is determined by either his knowledge or skills. Knowledge is what can be extracted from books or learned in a classroom. Skills are acquired with experience and in course of work, with or without tutelage.

In certain vocations knowledge is more important than skills; in others, the opposite is true. But in most both are needed. A locksmith must know how to use tools, a mechanical engineer should know how to calculate beam stresses, a physician should know how to examine a patient and make a correct diagnosis. However, there are no distinct and unequivocal (not to say quantitative) algorithms of diagnosis and treatment. So, in a medical profession, skills acquired with experience still play the primary role. This is why it takes so much longer to train a doctor than an engineer and why a doctor has to continue to learn through his entire career. On the other hand, one cannot be a good machine-tool operator without knowing the properties of materials, a good designer engineer should be able to visualize a future machine before making the blueprint, and all this requires both knowledge and skills.

Knowledge is transmitted from generation to generation: in old people's tales, chronicle books and manuals. Skills have always been transmitted in one way: from teacher to pupil. It is a long and bumpy road, and the results depend on the personalities of the giver and the receiver. Often the road broke off and skills were lost. We know, for example, that Tibetan medicine could cure almost all existing diseases. Lists of herbs and complete recipes have been preserved. But we do not know for sure how Tibetan doctors treated patients and with an enormous effort have to decipher the ancient rules and laws of treatment. Leonardo da Vinci was a talented engineer, but we can only guess how he arrived at his designs that were so much ahead of his time.

3. The Base of Human Skills

An individual who can accomplish complex tasks has a good memory that stores both knowledge and skills — experience with successes and failures in the past activity and the resulting generalized rules of action. Simon claims that the memory of an individual who has become a master of his trade contains dozens of different situations. He has estimated that a chess grandmaster remembers about 50,000 positions and their estimates. An experienced physician should remember about as many situations.

Memory is just one of the foundations of a skill. Another foundation is capacity for rapidly recognizing a situation and finding familiar traits. A

chess player — a grandmaster or a champion — remembers within a few seconds a complex multipiece situation that contains a certain meaning for him and can reconstruct it without effort much later.

An additional important foundation of a skill is the (usually innate) capacity of transforming unknown situations, supplementing them by analogy — adding characteristics that are lacking and reducing the new situations to those familiar past situations that are the closest to them. For example, an experienced engineer can recognize a new quality of a structure in an unusual combination of familiar characteristics.

We see that a skill is founded on memory generalization, rapid recognition and transformation of situations. Natural talent is not enough for mastering a skill: it takes years of learning. A talented chess player has to practice for eight to ten years before he becomes a grand master (this is amply illustrated by the biographies of the current world champion and his predecessors); it takes 10 to 15 years for a physician to become a reliable diagnostician; an outstanding designer engineer takes no less time to realize his full potential. Even child prodigies that amazed their contemporaries, such a Mozart, had to learn eight to ten years before reaching the peak of their mastery.

What is this time spent on? All these are, obviously, talented people. The answer is clear: it is spent on memorizing thousands of situations that can be later used flexibly and creatively.

4. Capabilities and Limitations

Imagine a computer which incorporates in its logic a replica of a recognized human expert. Such a computer should be able to plan experiments, analyze results, make diagnoses, etc. It would be a reproduction of an individual's skill that would not be lost after the death of the original owner: the copy will remain.

The problem can be formulated as follows: a field of activity is given; find facts and rules used by the individual who can solve problems in that field. They must be provided in a form understandable to a computer: a set of parameters and values. How can one find such rules and facts?

At a first glance, the task appears simple: one should interview an expert, a master or a professional whose skills we want to copy in the computer. Unfortunately, much as he would try, no expert can tell all we want to know. There are important psychological limitations. First of all, an individual who has a skill cannot fully verbalize it: the crucial transition from facts to action (a decision) often occurs subconsciously. In a real situation, an expert makes a correct decision without hesitation, but he may be nonplussed

when asked to formulate general rules prescribing actions for certain situations. This is why any difficult skill takes a long time to learn and why specialists are trained with the aid of specific examples.

Any professional field can contain hundreds and thousands of specific situations. It would take months, if not years, for an expert to appraise all of them. It is the extremely rare occasion that an expert may need the entire set of rules for explicit typical situations. Indeed, any recent medical school graduate can list all characteristic features of appendicitis, but it takes an experienced physician to make a correct diagnosis in complicated atypical situations which are so frequent.

We should also remember that to err is human. Even experienced people sometimes make mistakes because of fatigue, lapse of attention or complication and novelty of the situation (for the particular individual).

Experts' errors are often due to complexity of problems at hand, which is natural. If an individual is overloaded by the description of a situation, he inevitably omits part of information to simplify the problem before making a decision. This course of action is well justified. Information processing (including decision making) is known to occur in short-term memory and its capacity is limited: try, for example, to memorize at once two new telephone numbers. One has to find heuristic techniques and use simplifications, which, in turn, may be a source of error. It happens that a minute detail would tell an experienced diagnostician more about the patient or his diseases than volumes of a case history. What if that detail has been omitted to simplify the description…

5. Dialogue With an Expert

These difficulties of building a system to simulate the logic of an expert mean that system design could not be limited to techniques of computational mathematics but has to rely on psychological research. Such was the conclusion arrived at by a group of scientists at the All-Union Scientific Research Institute of System Studies, which included the present writer.

The general idea was to build a computer system which, after some tuning, could entertain a dialogue with an expert: ask questions, describe specific situations and step by step collect information to create a complete and consistent data base of the expert's skills. A certain strategy is needed to collect such information.

The questions to be asked of an expert are chosen in such a way as to derive as much useful information as possible. One is reminded of a game where one player has to guess a name thought of by his partner and familiar to both players. With this strategy, the number of questions and the guessing time

can be reduced by a factor of 5 to 10. Another important basic idea was to continually test the expert during the course of the dialogue for consistency of his decisions. The questioning is conducted in a way to compel the expert to estimate states of an object (directly or indirectly) several times; for example, an expert is given properties of a substance. The expert places the substance in a group A. Some of the properties are then modified so as to make them even more typical of that group, and a new set of properties is presented to the expert. Each answer is compared with the preceding one; the expert is alerted to contradictions that may occur.

In this fashion, occasional errors are eliminated, the basic rules are detailed and the operation logic of the expert is verified.

The dialogue is conducted in a language comprehensible to the expert; for example, a physician reads a case history displayed on the screen. The information load is dosed according to the human capacity of information perception and processing. In this study we had guidelines derived from an earlier analysis of a subject's capability for solving classification problems correctly and consistently.

The system we have developed (its acronym is CLASS, for CLASsification System) can be attuned to any desirable problem. It conducts a dialogue with an expert until building a complete (for all conceivable cases) and consistent database of the expert's skills relevant to the problem at hand. The entire information extracted in the course of the dialogue is memorized by the computer. Thus, one can obtain at any desired point an exhaustive explanation why a particular decision was made by the expert.

6. The Skills of CLASS

Interacting with an expert, the CLASS system generates in three to four days up to a thousand decision-making rules, including rules for exact diagnosis. By setting a hierarchy for decision rules, it is possible to classify millions of states, that is, bring to a system, formalize and describe the sphere of human experience, skills and techniques from an area that appears mysterious and defies formal description. What is the purpose of all this?

Today we know of fields with an urgent need for such systems: these are so-called expert systems and decision-making support systems. Besides primary data and mathematical models, they contain an important unit: the knowledge base (a better term would be a database of skills). Such data bases takes years to build and are hardly ever complete. Feigenbaum, a well-known American scholar, describes this problem as the bottleneck of artificial intelligence; Japanese scientists working on fifth-generation computers have set themselves the goal of learning by 1990 how to build systems containing from 10,000 to 15,000 decision rules.

Our methodology for development of skill databases offers a new approach to diagnostic system design. Instead of developing instrumental systems to be filled with knowledge and skills, we decided to simulate the logic of an expert. A system of this kind can be completed speedily: after two or three months of dialogue, the system is ready to operate.

The experience placed into the computer memory is professional advice not unlike a consultation given by an experienced professional to a young colleague. Such consultations are useful and especially valuable when there is no one around to consult with; a rural doctor or a ship's physician on a long voyage are often in this situation. They can be used not only in medicine, but also for rapid and error-free diagnosis of complex technological objects such as nuclear reactors, chemical plants and power engines.

7. Anecdotes and Responsibility

Thus, the problem has been solved. The knowledge and the skills of most knowledgeable and skillful professionals can be entered into a computer which can provide a faultless answer in any conceivable situation. But, does it really cover all situations? And, has the problem really been solved?

An American scientist specializing in expert systems recently told of this episode at a conference. A physician examined a patient using the help of an expert system designed specifically for diagnosis of diseases which involve back pain; he was unable to determine the disease. The patient continued to complain of back pain. When the doctor accidentally became distracted from sorting out the alternatives, he noticed a bulky wallet in the patient's hip pocket. The hard object exerted a constant pressure on the patient's lower back and was the source of the mysterious disease. What computer system could have anticipated such an eventuality?

Such anecdotes are rare and they are not the main source of limitations of computer capabilities. Systems simulating human skills utilize a prespecified set of features. Experts may claim that such a set as unquestionable but it is a priori limited. A specialist in practical work can almost always use features from a skill database but the "almost" remains. Thus, an artificial system can never do as much as its creator.

An important factor is the construction of the skill database and the careful selection of characteristics. We know from practice that the use of such systems by professionals induces them to be attentive and to survey the entire set of features rather than picking the first hypothesis and forgetting competing alternatives. This is a positive factor that can make the system useful to its creators. One should bear in mind, however, that advice from a computer equipped with other people's knowledge and skills is

no more than a consultation with a competent but absent professional who is never at one's side to see the case with his own eyes.

Another important difficulty with computer consultations has arisen in the United States, where expert systems are used in practice. Who is responsible for the decision made: the individual who trusted the computer or the system itself, that is, its designers or the experts? There have been attempts at placing blame on them. We believe that whatever the circumstances, the decision maker should be responsible for the diagnosis, the structure or any decision made whether with or without the aid of a computer.

8. Predicted and Unpredictable

Several decades ago science fiction writers predicted that man would build systems that could be a substitute for human intelligence. The prediction is probably coming true, but as we see this happens only partially. Without human intelligence, no "intelligent" system would have noticed the wallet in the hip pocket. One should not rely on them fully and mindlessly, but learn to operate with them; use their advice, but remain the master and the controller.

Besides, systems simulating human skills are not likely ever to be able to solve all of our professional problems. For one thing, there are practical areas where it is impossible or hard to find an expert who would transmit his skills to a computer. In medicine, for example, these problems include early diagnosis of diseases such as cancer. The opposite situation is one where experts are available, but there is not enough users to justify building an expert system. An obvious example is the skill of operating some unique and expensive instrument.

Nevertheless, there is a large and growing number of practical tasks where a correct decision can only be found with experience, skills and intuition of a master. In all epochs there has been a shortage of true masters and the modern one is no exception. Let masters transfer their experience to the computers — it will amplify their skills and capabilities.

Suggested Reading on Design of Expert Knowledge Databases

1. Larichev, O.I., Mechitov, A.I., Moshkovich, I.M., and Furems, E.M., "Systems for Identification of Expert Knowledge in Classification Problems", TEKHNICHESKAYA KIBERNETIKA, No 2, 1987.

2. Larichev, O.I., and Moshkovich, I.M., "Direct Classification Problems in Decision Making", DOKLADY AKADEMII NAUK USSR, Vol 286, No 6, 1986.

Control Systems and Machines

18630290a Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 88 pp 1-2

[Table of contents and selected articles]

TABLE OF CONTENTS

COMPUTING SYSTEMS AND PERIPHERALS

MICROPROCESSORS AND MICROTECHNOLOGY

CONTROL OF THE COMPUTING PROCESS IN COMPUTERS AND NETWORKS

PROGRAMMING LANGUAGES AND TECHNIQUES

DATABANKS, KNOWLEDGE BASES AND EXPERT SYSTEMS

UDC 681.3

Multilevel Organization of Microcomputers

18630290b Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 88 pp 8-11

[Article by M. A. Gladshteyn]

[Text] Computerization, as witnessed in recent years, is reaching almost every domain of human activity. Widespread use of computer resources would be impossible without cheap and compact microcomputers built from microprocessor modules.

The most difficult task in developing microcomputer-based systems is the design of the software. Software as a product is characterized by distinct quality indexes. These include the time required to solve tasks and the memory volume occupied by the program. In recent years, due to the rapid product obsolescence, program maintainability is becoming increasingly urgent. The concept of maintainability includes easy comprehension of the program by other experts, simple testing and debugging, and possibilities for modernizing the program.

According to [1, 2], program logic is accomplished by embedding typical program structures: succession, branching, repetition, choice, and choice-repetition. Each such structure has one input and one output, which greatly simplifies understanding the links in the program. However, it cannot be said that the use of structured programming provides a total solution to the problem of creating effective software. A particular drawback of this approach is the fact that the complexity of the program logic determines the depth of embedding. Although the use of main memory stacks places almost no restrictions on the depth of embedding, it becomes difficult to comprehend a program with more than five or six levels. All of this adversely influences possibilities for on-line testing and debugging.

The situation is exacerbated by the fact that nearly all present-day microcomputers have a single level of machine instructions (since the programmer does not have access to the microprogram level). Therefore, a multilevel program for execution in the computer should be represented on the solitary level of machine instructions. This renders the process of testing and debugging the developed program more difficult, for the program loses its hierarchical structure (in which it was conceived during the design process) on the machine level.

Thus, to enhance programming effectiveness, the hierarchical structure of the program should be made to conform with the hardware on which the program is to be implemented. This can be done by organizing a multilevel computer. A multilevel computer can be defined as a machine having a series of levels of machine instructions (i.e., a series of machine languages). Such computer, consequently, must have a special program counter for each level. The operation of a multilevel computer is supported by a group of relatively independent programs of different levels which are linked together by a hierarchical interpreter during the course of execution. Consequently, the chief difference between a multilevel computer and a traditional computer consists in the specification of several program counters of various levels and the presence of a hierarchical command interpreter. On this foundation, one may conclude that a multilevel computer can be organized in any given microprogrammable computer having a sufficient number of general-purpose registers to accommodate the program counters and sufficient microprogram memory to contain the hierarchical interpreter. The main task in setting up such a machine consists in developing the hierarchical interpreter.

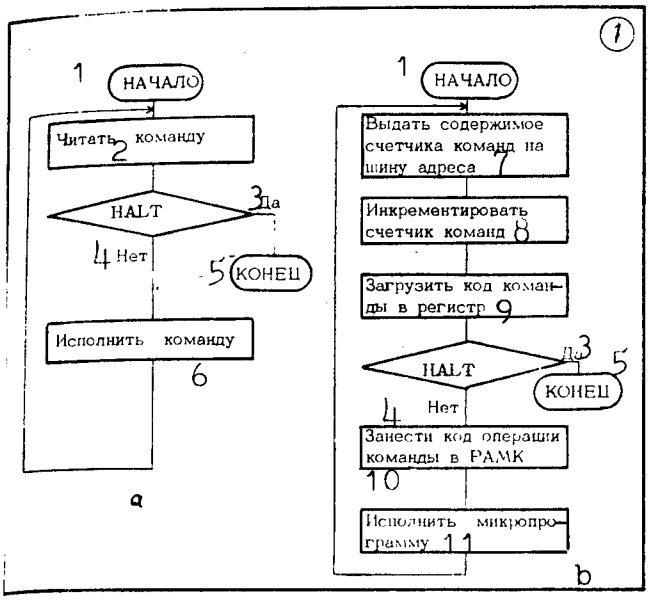


Fig. 1. Algorithm for interpreting instructions in a traditional computer: main cycle (a), detailed diagram (b).

Key:

1. Start
2. Read instruction
3. Yes
4. No
5. End
6. Execute instruction
7. Output contents of instruction counter to address bus
8. Increment instruction counter
9. Load instruction code into register
10. Enter instruction operation code to RAMK
11. Execute microprogram

The algorithm for interpreting instructions in a traditional computer is illustrated by the flowchart (Fig. 1, a). The essence of this process is a cyclical repetition of the commands READ INSTRUCTION and EXECUTE INSTRUCTION [3] until the HALT instruction is detected. This process is accomplished in a microcomputer by executing the microoperations indicated in the detailed flowchart (Fig. 1, b). Characteristic of a microprogrammable computer is the fact that initiation of instruction execution is done by entering the operation code of the instruction in the microinstruction address register (RAMK) [4].

Use of a similar approach for high-level instructions means that to execute the instruction of level i it is necessary to dispatch the operation code of the instruction to the program counter of level i-1. On the basis of these considerations one may construct a hierarchical interpretation flowchart (Fig. 2). This flowchart is a regular structure with sequential relations for progressive calling of instruction codes as the level diminishes until the instruction of the zeroth execution level is invoked. When the code of the end of operation instruction END_i is detected at the i-th level, the next-highest level is consulted for the following instruction. Detection of the instruction END_n at the top level signifies the end of the computations.

The commands READ INSTRUCTION OF LEVEL i should also be implemented by analogy with the flowchart in Fig. 1, b. Any level of instruction may contain, besides the main operation code, a set of parameters arranged in consecutive memory cells. These parameters are used by the lower execution level, which should have corresponding instructions for access to all program counters. This also enables implementation of conditional and unconditional jumps on all program levels.

As an example, consider the organization of a multilevel computer based on a series 589 microprocessor configuration (or the foreign-made Intel-3000). The structure of the computer can be organized in the standard way [5], as shown in Fig. 3, where the following abbreviations are used: BMU - microprogram control unit (K589IK01) [4]; MPP - microprogram memory; OU - operational unit (for a 16-bit computer, eight K589IK02 microcircuits and one K589IK03 microcircuit are necessary [4]; ZU - storage for programs of all levels and data; UVV - input/output device. The inputs and outputs of the structural elements have the international designations. To simplify the diagram, we do not indicate the synchronization and gating inputs, the synchro-pulse generator, and so forth.

The microinstruction structure of this microcomputer contains a control field of the OU indicating the function code F and the value of the mask K; a control field of the BMU, indicating the control codes for the flag logic FC and the address of the jump AC; and a control field for the bus, memory, and input/output, indicating the condition signals of the data bus (IN or OUT), reading from or writing into memory (MEMR or MEMW), input or output (I/OR or I/OW).

In accordance with the flowchart shown in Fig. 2, a microprogram for a five-level interpreter was developed, shown in the table. The symbolism of the

operations is consistent with macroassembler [4]. The program counters of levels 4, 3, 2, 1 and 0 are organized in the respective general-purpose registers R4, R3, R2, R1 and R0. The memory address register Pa is used for transmission of the address. The arrangement of the microprogram in the address matrix is shown in Fig. 4. The volume of the microprogram is 16 microinstructions. In other words. 3 percent of the address space of the BMU is used to implement the five-level interpreter.

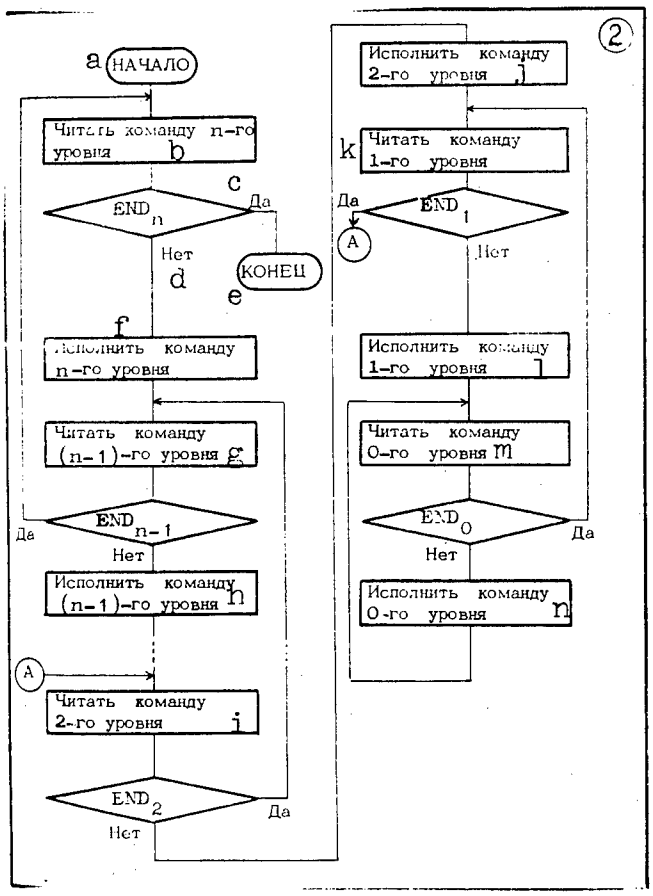


Fig. 2. Algorithm of hierarchical interpretation of instructions.
Key:

a. Start
b. Read instruction of n-th level
c. Yes
d. No
e. End
f. Execute instruction of n-th level
g. Read instruction of (n-1)-th level
h. Execute instruction of (n-1)-th level
i. Read instruction of 2-nd level
j. Execute instruction of 2-nd level
k. Read instruction of 1-st level
l. Execute instruction of 1-st level
m. Read instruction of 0-th level
n. Execute instruction of 0-th level

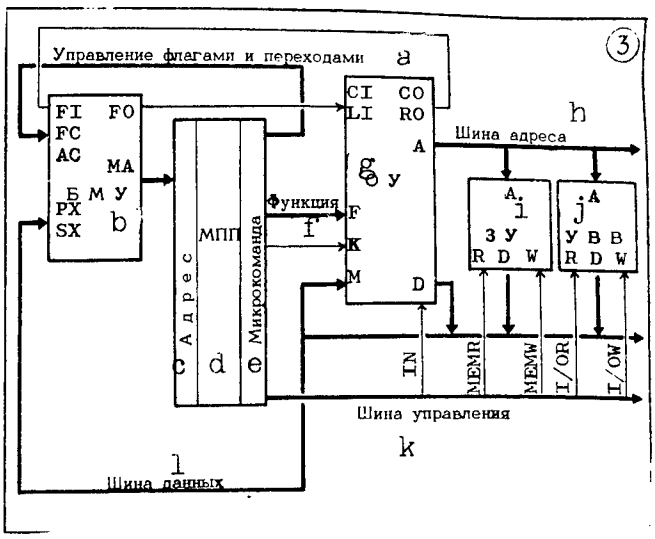


Fig. 3. Structure of a microcomputer built from the series 589 microprocessor configuration.
Key:

a.	Control of flags and jumps	g.	OU
b.	BMU	h.	Address bus
c.	Address	i.	ZU
d.	MPP	j.	UVV
e.	Microinstruction	k.	Control bus
f.	Function	l.	Data bus

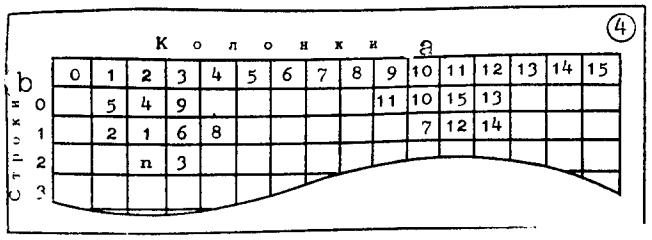


Fig. 4. Arrangement of the microprogram of the multilevel instruction interpreter in the microprogram memory of a microcomputer.
Key:

a.	Columns	b.	Rows

MICROPROGRAM OF THE MULTILEVEL INSTRUCTION INTERPRETER

(a) Номер МК	b Адрес МК: стр. c	кол. d	e Функция	f Маска	Шина данных	Управление памятью и g вв./выв. h	Управление флагами i: K	L	m Переход	Комментарий (n)
1	Ø1	2	LMI R4	Ø	IN		HCZ	FF1	JCR 1	Содержимое счетчика команд 4-го уровня переслать в PA, счетчик инкрементировать (o)
*										
2	Ø1	1	LTM AC	1	IN	MEMR	HCZ	FFØ	JFL 2	Загрузить команду 4-го уровня, перейти, если END_4 (p)
3	Ø2	3	SDR R3	1	IN		HCZ	FF1	JZR 2	Занести код в счетчик команд 3-го уровня (q)
4	ØØ	2	LMI R3	Ø	IN		HCZ	FF1	JCR 1	Содержимое счетчика команд 3-го уровня переслать в PA, счетчик инкрементировать (r)
*										
5	ØØ	1	LTM AC	1	IN	MEMR	HCZ	FFØ	JFL 1	Загрузить команду 3-го уровня, перейти, если END_3 (s)
6	Ø1	3	SDR R2	1	IN		HCZ	FF1	JCR A	Занести код в счетчик команд 2-го уровня (t)
7	Ø1	A	LMI R2	Ø	IN		HCZ	FF1	JCR 4	Содержимое счетчика команд 2-го уровня переслать в PA, счетчик инкрементировать (u)
*										
8	Ø1	4	LTM AC	1	IN	MEMR	HCZ	FFØ	JFL Ø	Загрузить команду 2-го уровня, перейти, если END_2 (v)
9	ØØ	3	SDR R1	1	IN		HCZ	FF1	JCR A	Занести код в счетчик команд 1-го уровня (w)
10	ØØ	A	LMI R1	Ø	IN		HCZ	FF1	JCR 9	Содержимое счетчика команд 1-го уровня переслать в PA, счетчик инкрементировать (x)
*										
11	ØØ	9	LTM AC	1	IN	MEMR	HCZ	FFØ	JFL 1	Загрузить команду 1-го уровня, перейти, если END_1 (y)
12	Ø1	B	SDR R0	1	IN		HCZ	FF1	JZR C	Занести код в счетчик команд 0-го уровня (z)
13	ØØ	C	LMI R0	Ø	IN		HCZ	FF1	JCC 1	Содержимое счетчика команд 0-го уровня переслать в PA, счетчик инкрементировать (aa)
*										
14	Ø1	C	LTM AC	1	IN	MEMR	HCZ	FFØ	JFL Ø	Загрузить команду 0-го уровня, перейти, если END_0 (bb)
15	ØØ	B	NOP	Ø	OUT		HCZ	FFØ	JPX...	Переход на микропрограмму выполнения команды 0-го уровня (cc)
...	...	...	...	...	...	...	...	...	...	...
n	Ø2	2	NOP	Ø	IN		HCZ	FFØ	JCR 2	Останов (dd)

Key:

a. Number of microinstruction
b. Address of microinstruction
c. Row
d. Column
e. Function
f. Mask
g. Data bus
h. Memory and I/O control
i. Flag control
k. Input
l. Output
m. Jump
n. Remarks
o. Transfer contents of 4th-level instruction counter to PA, increment counter
p. Load 4th-level instruction, move on if END_4
q. Enter code in 3rd-level instruction counter
r. Transfer contents of 3rd-level instruction counter to PA, increment counter
s. Load 3rd-level instruction, move on if END_3
t. Enter code in 2nd-level instruction counter
u. Transfer contents of 2nd-level instruction counter to PA, increment counter
v. Load 2nd-level instruction, move on if END_2
w. Enter code in 1st-level instruction counter
x. Transfer contents of 1st-level instruction counter to PA, increment counter
y. Enter 1st-level instruction, move on if END_1
z. Enter code in 0-level instruction counter
aa. Transfer contents of 0-level instruction counter to PA, increment counter
bb. Load 0-level instruction, move on if END_0
cc. Go to microprogram for execution of 0-level instruction
dd. Stop

This example illustrates that only a negligible amount of the microcomputer's resources is used in multilevel organization. Yet it may achieve a substantial increase in efficiency.

The formalized technical task assignment method and the method of modular programming [6]--using a hierarchical series of problem-oriented algorithmic languages--are ideal for software development. When translating a multilevel program, the language commands of each i-th level will code to instructions of the corresponding level. Transcription of a multilevel program can also be done by a traditional assembler, the program modules of the upper levels being designed with use of the assembler pseudoinstructions alone. Thus, for example, when using the macroassembler 8080/8085 [7] to translate the programs, a typical notation for the instruction of i-th level ($i \neq 0$) will be written as follows:

[LABEL:] DW INSTRI, PARAM1, ..., PARAMN,

where LABEL is optional label (only the first instructions of the program modules and the jump points are indicated); DW is a pseudoinstruction designating the data words; INSTRI is the symbol for the i-th level instruction (this same identifier labels the first line of the program module of (i-1)-the level, implementing this instruction); PARAM1, ..., PARAMN are parameters, each of which may be described in accordance with the rules of th macroassembler by a statement, i.e., they may include constants, letters, variables, and so on; the number of parameters is limited $N \leq 7$.

Given such coding, the macroassembler will automatically designate the instruction operation codes INSTRI during the translation of the multilevel program.

Using a multilevel microcomputer organized in this way with traditional programming resources creates the foundation for a software engineering that satisfies virtually all the requirements formulated by V. M. Glushkov [8].

In fact, the development of multilevel programs can be organized in self-contained cycles, each program module residing in any given memory location.

In a multilevel microcomputer the structure of the machine instructions corresponds totthe hierarchy of algorithms, and therefore the computing process is easily monitored using the state of the program counters. It is possible to execute a program in instruction cycles at different levels, greatly simplifying testing and debugging.

Modifying a multilevel program requires only expansion or replacement of individual program modules and does not necessitate repeated translation or change in memory residence for the other parts of the program.

Thus, multilevel programming can produce a computer whose program memory is an open hierarchical storehouse where the user can load his program modules. Theprogram maintenance capabilities which accrue to the user in this case substantially surpass those of traditional systems.

These advantages of multilevel organization give reason to expect that it will be adopted widely in routine design.

BIBLIOGRAPHY

1. Deykstra, E., "Distsiplina programmirovaniya [The Discipline of Programming]", Mir, Moscow, 1978, 278 pp.

2. Dal, U., Deykstra, E., Khoor, K., "Strukturnoye programmirovaniye [Structural Programming]", Mir, Moscow, 1975, 248 pp.

3. Klingman, E., "Proyektirovaniye mikroprotessornykh sistem [The Design of Microprocessor Systems]", Mir, Moscow, 1980, 575 pages.

4. Berezenko, A. I., Koryagin, L. N., Nazaryan, A. R., "Mikroprotsessornyye komplekty povyshennogo bystrodeystviya [Microprocessor Configurations with Enhanced Speed]", Radio i svyaz, Moscow, 1981, 168 pp.

5. Mik, Dzh., Brik, Dzh., "Proyektirovaniye mikroprotsessornykh ustroystv s razryadno-modulnoy organizatsiey [Design of Microprocessors with Bit-Slice Organization]", Mir, Moscow, 1984, Book 1, 253 pp.

6. Glushkov, V. M., "Osnovy bezbumazhnoy informatiki [The Foundations of Paper-Free Data Processing]", Nauka, Moscow, 1982, 552 pp.

7. Myskin, A. V., Torgashev, V. A., "Programming Directions. Macroassembler 8080/8085. Software Documents," LNIVTs, Leningrad, 1980, 65 pp.

8. Glushkov, V. M., "Fundamental Research and Software Engineering", PROGRAMMIROVANIYE, No 2, 1980, pp 3-13.

UDC 621.327.28

Interactive Programmer of Programmable Logic Arrays

18630290c Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 88 pp 11-16

[Article by Ye. P. Rodionov and D. A. Strabykin]

[Text] Programmable logic arrays (PLAs) are widely used in design and modernization of automation and computer equipment. Using PLAs in construction of digital devices shortens the development time and improves the technical-economic indexes of the equipment. Adjustment of the PLA to carry out the functions of the device under design is done with the aid of PLA programmers.

In analyzing design characteristics used by present-day programmers, we should single out the devices controlled by mini- and microcomputers [1, 2]. The transfer of a sizable number of the control functions to the microcomputer simplifies the programmer apparatus, so that additional equipment assumes the form of structurally-complete modules (accessories) or boards built into the microcomputer [3]. Nevertheless, certain devices do not adequately exploit the computing possibilities of the microcomputer: the PLA programming and checking time diagrams are accomplished on the hardware, not the software level [1]; inadequate attention is paid to organization of the interaction between user and programmer; and there are no facilities for preparing data in the form of an analytical description of the PLA functions [2].

We will consider the structure and characteristics of programming PLAs, the hardware and software, and the operating regimes of an interactive PLA programmer based on the Elektronika-60 microcomputer.

PLA structure and programming characteristics. The K556RT1 and RT2 integrated microcircuits are electrically programmable logic arrays implementing 8 output functions from 16 input variables. The output functions can be either logical 1 or logical 0. The logic functions in the PLA are presented in the disjunctive normal form (DNF). The total number of conjunctions in the DNF should not exceed 48.

Figure 1 shows the PLA, consisting of two arrays. In the upper portion of the figure is the array of AND-gates; in the lower portion of the array, or OR-gates. There are 16 drivers F (d1, ..., D16) at the circuit input, allowing

both direct and inverted values of the input variables to be connected into the conjunction. The input variables can be combined in 48 different conjunctions, from which 8 disjunctions are formed by means of the OR array. By means of elements D17, ..., D24 and their corresponding links it is possible to select the lower or the upper level as the active one. Output gates for synchronization of the output of functions are formed from elements D25, ..., D32.

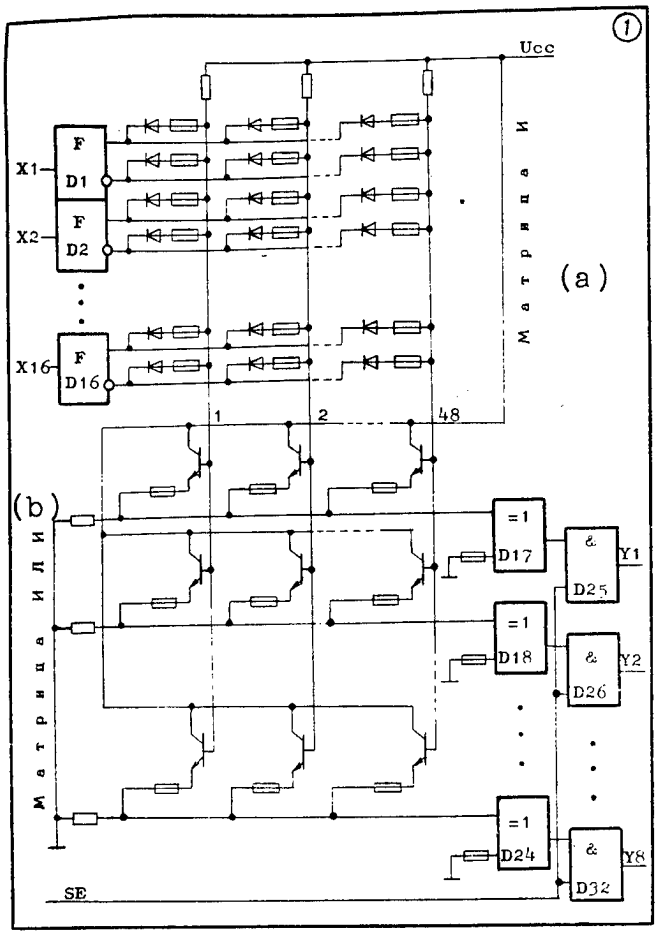


Fig. 1. PLA functional diagram.
Key:
a. AND array b. OR array

The PLA comes unprogrammed, i.e, all links are intact. In order to burn through a particular link in the array of AND-gates, the address of conjunction is specified by decoder, and voltage of logical 1 or logical 0 is applied to the particular input (depending on whether the inverted or the direct variable is to be excluded from the given conjunction). If the variable should not appear at all in the programmable conjunction, both links are burned through. To program the array of OR-gates, a second decoder specifies the address of the conjunction and a potential of 10 V is applied to the output from which the given conjunction and a potential of 10 V is applied to

the output from which the given conjunction should be excluded. Then all links corresponding to the conjunctions not pertaining to the given disjunction are consecutively burned through. To program the active lower layer of output functions, the corresponding links of the layer of output inventors (phase links) are burned through.

Thus, PLA programming resources should permit input and output monitoring of the microcircuit, programming and checking of the OR-gates, AND-gates and phase links. It is also necessary to be able to read the conditions of all links in order to check the microcircuit after burn-in and for programming the PLA of the finished product.

Hardware configuration of an interactive programmer. Analysis of the PLA programming and checking diagrams reveals that production of programming pulses of requisite length can be accorded entirely to the software. The only exception is the formation of flat-rising pulses, which is accomplished by specialized switching circuits. This approach to the development of an interactive programmer minimizes the volume of nonstandard equipment connected to the microcomputer: the programming voltage shaper (UFPN) accessory is built from a single board and installed directly in the frame of the Elektronika-60 microcomputer. This, the interactive programmer is actually an interactive control system, implemented by connecting the UFPN board to the already-present standard equipment and by augmenting the operating system (OS) with special software.

The minimum hardware configuration for PLA programming (Fig. 2) comprises the microcomputer Elektronika-60, a monitor, a floppy or hard disk drive (NMD), the UFPN, and a 20-35 V dc power supply. The interactive programmer may also include an alphanumeric printer (ATsPU) and any other peripherals (VU) connected to the OS.

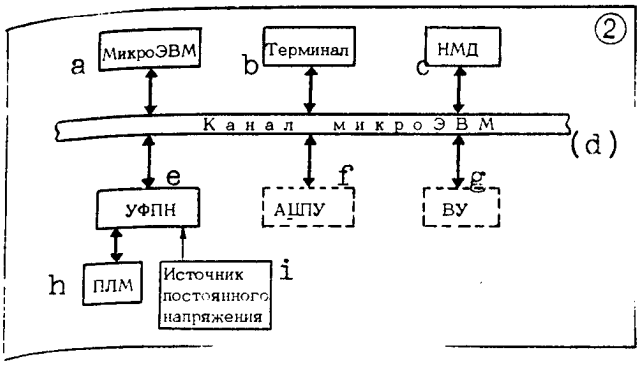


Fig. 2. Interactive programmer hardware.

Key:

a. Microcomputer
b. Monitor
c. Disk drive
d. Microcomputer channel
e. UFPN
f. Alphanumeric printer
g. Peripherals
h. PLA
i. dc power supply

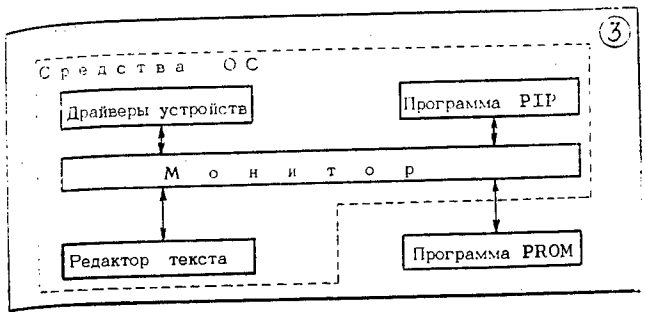


Fig. 3. Interactive programmer software.

Key:

a.	OS facilities	d.	Monitor
b.	Device drivers	e.	Text editor
c.	PIP program	f.	PROM program

The software for the programmer was developed for the RAFOS operating system, but can also be adapted to other OS used in the Elektronika-60 microcomputer. In order for the interactive programmer to work, the following software configuration is required (Fig. 3): a monitor (such as DXMNSJ), device drivers; a text editor (K52, SCREEN, EDIT, etc.), by means of which the user describes the microcircuit being programmed; a program servicing the PIP devices for copying the data from one medium to another; and the program PROM, which supports the operation of the UFPN and organizes the interaction between the user and the programmer.

The programming voltage shaper. In broad terms, the UFPN can be visualized as units which couple the buffer registers, the switches, and the voltage stabilizers to the microcomputer channel. The basis of the device's operation is software control of the voltage levels on the terminals of the microcircuit being programmed. Each switch controls corresponding stages of the program-accessible buffer register assembly.

The structural layout of the UFPN is shown in Fig. 4. The device includes the following main elements: the control unit; the address decoder; the bus driver; the PLA output reading circuit; two PLA input control registers (Rg1, Rg2); a PLA output control register (Rg3); a control signal register (Rg4); signal switching units at the inputs (BK1) and outputs (BK2) of the PLA; voltage switches at PLA terminals 28, 19, 1 (K1, K2, K3); a programming voltage switch (K4); and voltage stabilizers +17.5 V (SN1) and +10.5 V (SN2).

The device works as follows. The inputs of the program-accessible registers Rg1-Rg4 are connected to the bus A/D[0]-A/D[15] across the bus driver. Only one of the registers can be accessed per cycle. Writing is done by the signals Zp1-Zp4, which are formed by the control unit. The number of the register is determined by the signals at the output of the address decoder. A constant potential +U from an external power source is applied to the input SN1. The voltages needed to program and check the PLA are generated by SN1 and SN2 and applied to BK1, BK2 and K1-K4. The selection of voltages to be applied to the terminals of the microcircuit being programmed is

determined by the states of registers Rg1-Rg4. To check the results of the programming, the device comes with the output reading circuit, which is also used to protect the inputs of the bus driver against elevated voltage. The signal NU sets the registeres Rg1-Rg4 in a condition where a potential of 0 V is applied to the terminals of the PLA, preventing damage to the microcircuit being programmed when the power is turned on.

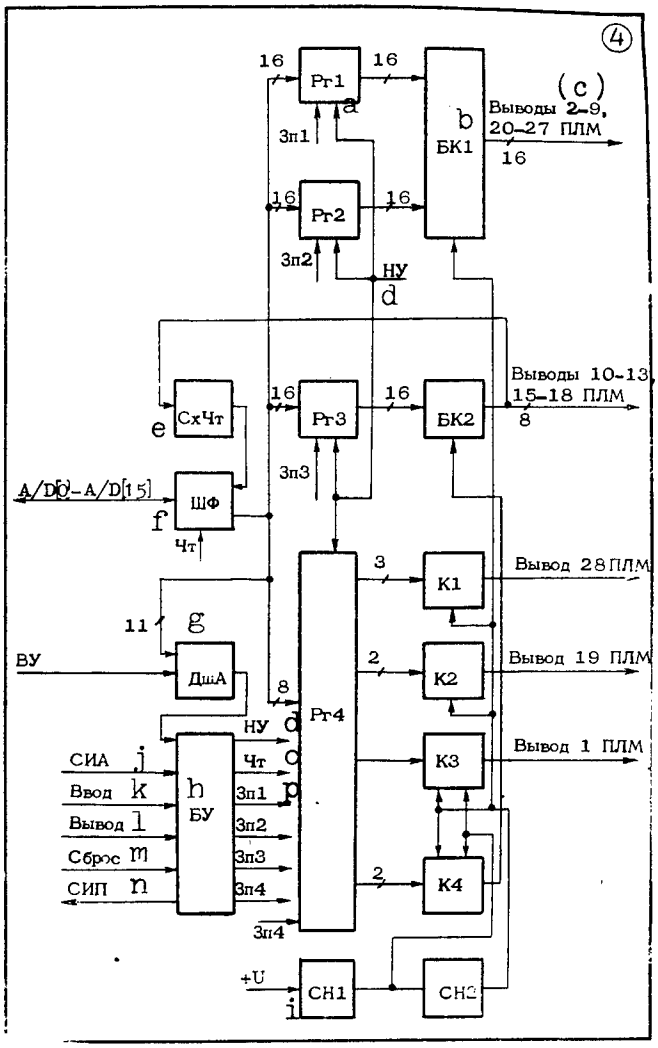


Fig. 4. Structural layout of UFPI.

Key:

a.	Rg1	i.	SN1
b.	BK1	j.	SIA
c.	PLA terminals 2-9, 20-27	k.	Input
d.	NU	l.	Output
e.	Output reading circuit	m.	Reset
f.	Bus driver	n.	SIP
g.	Address decoder	o.	Read (Cht)
h.	Control unit	p.	Write (Zp) 1, 2, 3, 4

Organization of interactive programmer software. The PROM servicing module of the programmer has a four-level hierarchical structure (Fig. 5). At the first level is the supervisor, which presents the list of operating modes of the interactive programmer on the monitor, enters the user's selection of operating mode, and calls the corresponding 2nd-level processing programs. The latter include:

- a program for entry of data from the PLA into a buffer, which controls the filling of the data buffer with information corresponding to the state of the links in the arrays of OR-gates, AND-gates, and invertors;

- a PLA check program, which compares the information written in the microcircuit with the contents of the data buffer. If a discrepancy is found,aa message with the number of the link and the type of inaccuracy is shown on the monitor;

- the PLA programming module, which performs a check of the microcircuit prior to the programming (determining if information can be written from the data buffer to the PLA), the programming proper, and a final check to see if the written data corresponds with the information in the buffer;

- a program for output of the table of matrix-wires, designed to output the condition of the matrix wires of the microcircuit in the form of a table to a system terminal or printer, from which it is possible to determine the functions provided by the PLA and whether further programming can be done;

- a program for entry of data in analytical form, which enters text files from an external medium, removes the commentaries from the descriptions, checks that the text characters correspond to the input alphabet and that the logic expressions are written properly, converts the numbers of the AND-gates and the outputs of the PLA from KOI-7 to the binary system of notation, and forms the data buffer in accordance with the description of the logic functions to be implemented by the microcircuit;

- a binary information I/O program, which organizes the exchange of data with external information media.

The third level of software consists of dependent subroutines which are used by the 2nd-level modules:

- a subroutine for checking the OR-gates, AND-gates and phase links, whereby information is written into the data buffer on the basis of an analysis of the condition of the links of the inspected PLA;

- a subroutine for programming OR-gates, AND-gates and phase links. These modules obtain control from the PLA programming program, check whether a particular link is responding to the programming pulses, and if necessary, activating the 4th-level modules;

- a subroutine for I/O of files in RAFOS OS format, insuring the exchange of data between the service program of the programmer and external devices included in the system where the interactive programmer is being used. The I/O programs support operation with any devices whose drivers are OS configured. The specific type of device is indicated by the operator in the instruction line.

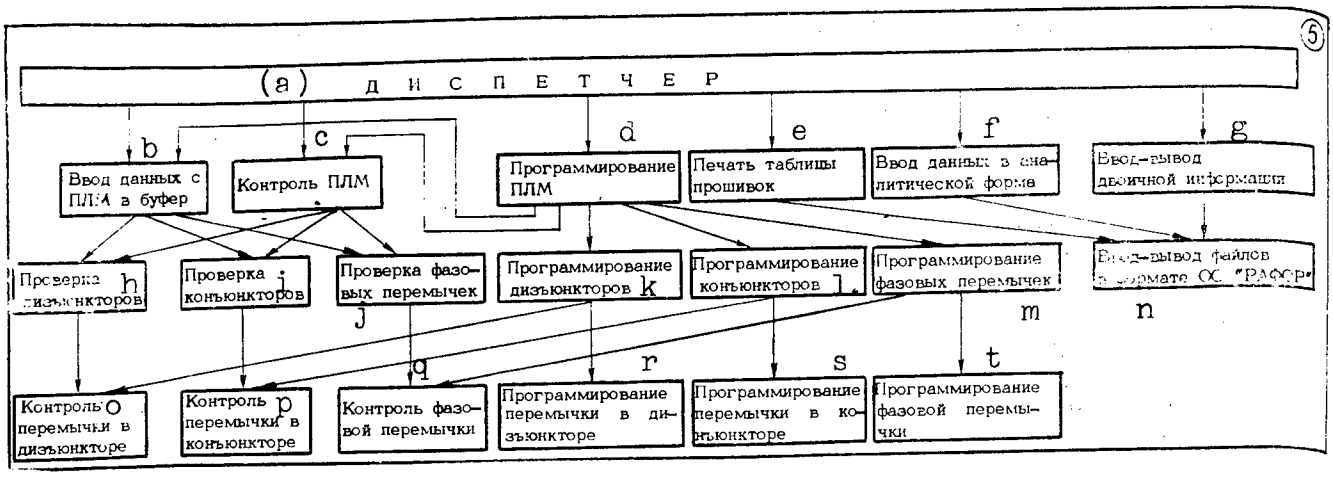


Fig. 5. Structure of the interactive programmer service program.

Key:

a. Supervisor
b. Entry of data from PLA to buffer
c. Checking of PLA
d. Programming of PLA
e. Printing of matrix-wire table
f. Entry of data in analytical form
g. Binary information I/O
h. OR-gate checking
i. AND-gate checking
j. Phase link checking
k. OR-gate programming
l. AND-gate programming
m. Phase programming links
n. I/O of files of RAFOR [sic] operating system format
o. OR-gate link checking
p. AND-gate link checking
q. Phase link checking
r. OR-gate link programming
s. AND-gate link programming
t. Phase link programming

The fourth level of software includes subroutines for programming and checking the links of the OR-gates, the AND-gates, and the phase links. The program modules are designed to control the programmer's operation. The software implements the programming and checking time diagrams on this level. The subroutines responsible for the programming consist of a controlling module and a module that forms the programming pulse. The controlling module keeps a tally of the number of programming pulses and forms a tag for the outcome of the programming, which is used by higher-level programs to inform the user as to the course of the PLA programming. The input information for the checking subroutines is the type of link and its number (address). When the subroutines are executed the data buffer stages corresponding to each given link are set or cleared.

The main working area of the programmer's service program is the data buffer. It is used in all information I/O operations, programming and checking of the PLA. There are 241 memory bytes reserved in the program to accommodate the buffer. Each link of the PLA correspondstto a definite position in the data buffer. In the I/O operations, the buffer is complemented with zero codes up to 512 bytes (the block size in the RAFOS format).

Operating modes of the interactive programmer. The programmer's distinguishing feature is its interactive operation. After starting the PROM program, the monitor displays the following menu:

```
1. ENTER DATA INTO BUFFER
   1.1. FROM STANDARD PLA
   1.2. FROM EXTERNAL MEMORY
        1.2.1. IN BINARY MEMORY
        1.2.2. IN ANALYTICAL FORM
2. OUTPUT DATA BUFFER TO EXTERNAL MEMORY
3. PRINTING DATA BUFFER AS TABLE OF MATRIX WIRES
   3.1. TO SCREEN
   3.2. TO PRINTER
4. PROGRAM PLA
5. CHECK PLA

ENTER MODE NUMBER
*
```

After selecting the appropriate mode, the user types in its symbolic notation and completes the entry by pressing the <VK> key. For example, to evoke the mode ENTER DATA INTO BUFFER IN ANALYTICAL FORM, one types 122<VK>. We shall examine the active programmer's more interesting operating modes.

ENTER DATA INTO BUFFER IN ANALYTICAL FORM. The analytical description of the PLA is obtained directly from the DNF of the logic functions implemented by the microcircuit. The description of the PLA is composed separately for the elementary conjunctions and the output functions. The elementary conjunctions are singled out from the initial DNF and described by assigning them conventional designations K1, K2, ..., K48. The descriptions of the output functions should be in the form of the logical sum of the elementary conjunctions. The resulting expressions are then described in symbolic language placed on an external medium in an initial file by means of text editor. We shall describe the procedure used to prepare the analytical description file.

The alphabet of the language is comprised of Latin letters X, K, Y, the digits from 0 to 9, and the special characters <=>, <.>, <+>, <*>, <->, <VK>, <PS>, <GT>, <PROBEL>. Any given KOI-7 symbols can be used in the commentaries. The input variables are designated X or -X, depending on whether the direct or the inverted value is to appear in the elementary conjunction. Then the number of the input variable, from 1 to 16 (depending on the number of inputs of the microcircuit), must be indicated.

The elementary conjunctions are designated K, followed by a number from 1 to 48 (the number of AND-gates in the PLA). The output functions are designated Y or -Y, depending on whether the function should be represented by the active high or the active low level, followed by a number from 1 to 8 (according to the number of outputs).

The description of each elementary conjunction and output function should end with the character <.>, and if the description does not fit onto a line it can be continued in the next one. The number of continuation lines is unlimited. The symbol for logical addition is <+>, and for multiplication, <*>. The description of the elementary conjunctions and output functions is arbitrary.

The description of the logical expressions may include the meaningless designators <PROBEL>, <GT>, <VK> and commentaries. The symbol for the start of the commentary is <;>, and the end, <VK>.

We give an example of the preparation of data in analytical form:

```
; ОПИСАНИЕ КОНЪЮНКТОРОВ: (a)
K1=¬X4*X5.
K3=¬X14.
K4=X3*X4*X6*¬X12.
K5=X13*¬X14*X15.
K6=¬X3*¬X14*X8.
K7=X9.
K8=X3*¬X16*¬X12*X14*X6.
; ОПИСАНИЕ ВЫХОДНЫХ ФУНКЦИЙ: (b)
¬Y1=K1+K4+K5+K8.
 Y2=K4+K3.
¬Y3=K4+K6+K7+K8.
¬Y4=K1+K3+K7+K8.
 Y7=K5.
```

Key:

a. Description of AND-gates
b. Description of output functions

for the following logic functions realized by the PLA:

$$\overline{Y1}=\overline{X4}\cdot X5+X3\cdot X4\cdot X6\cdot\overline{X12}+X13\cdot\overline{X14}\cdot X15+X3\cdot\overline{X16}\cdot\overline{X12}\times\times X14\cdot X6;$$

$$\overline{Y2}=X3\cdot X4\cdot X6\cdot\overline{X12}+\overline{X14};$$

$$\overline{Y3}=X3\cdot X4\cdot X6\cdot X\overline{12}+X3\cdot\overline{X14}\cdot X8+X9+X3\cdot X\overline{16}\cdot\overline{X12}\cdot X14\times\times X6;$$

$$\overline{Y4}=\overline{X4}\cdot X5+\cdot X\overline{14}\cdot+X9+X3\cdot\overline{X16}\cdot\overline{X12}\cdot X14\cdot X6;$$

$$Y5=X13\cdot X14\cdot X15$$

PRINT BUFFER AS A TABLE OF MATRIX WIRES. It is possible to print a table of matrix wires correspoinding to the contens of the data buffer to the monitor or to the printer. These operating modes of the interactive programmer are used to prepare documentation and ensure compatibility with programmers that cannot prepare data in analytical form. The table corresponds to the description ofthe PLA and consists of two parts. The upper part contains information about the links connecting the inputs of the microcircuit to the AND-gates, while the lower part contains information about the state of the phase links and the links joining the outputs of the AND-gates to the inputs of the OR-gates.

PROGRAM PLA. After selecting this mode, the microcircuit is inspected to see whether information can be recorded. The monitor shows the message CHECKING PLA. If the microcircuit's links permit recording, checking is halted. If errors are found, the screen displays information as to the missing links. After checking the microcircuit, the screen displays the message PROGRAM? (Y/N). Pressing the key Y starts the programming. If any other key is pressed, the burn-through will not be done.

Information displayed on the monitor provides information on the progress of the programming. After programming is completed, an output check of the PLA is done. The monitor displays the message CHECKING PLA, and the condition of all links of the microcircuits is analyzed. The check can detect two types of error: absence of a link and its restoration. The triple check of the microcircuit during the programming (input check,mmonitoring during burn-through of links, and final check of the PLA) greatly improves the reliability of the information writing.

Conclusion. Compared to the traditional devices, the proposed interactive programmer has a number of advantages:

- ease of use, owing to its interactive mode;

- convenient entry and editing of initial data;

- automation of programming and microcircuit checking;

-minimal nonstandard equipment and software.

Thus, the UFPN unit contains 30 microcircuit packages and a number of discrete components. It is simple to construct and use. The PROM program is written in Assembler and occupies 7.5 Kb. Programming time depends on the number of links burned through, ranging from several dozen seconds to 1-2 minutes. The interactive programmer was developed for design organizations but can also be used at enterprises manufacturing automation and computer equipment.

BIBLIOGRAPHY

1. Dianov, A. P., Shchelkunov, N. N., "Logic Circuit Programming Hardware", MIKROPROTSESSORNYYE SREDSTVA I SISTEMY, No 2, 1986, pp 77-80.

2. Butov, A. A., Grishel, S. I., Kupin, V. M., "A PLA and PROM Programmer", USIM, No 2, 1986, pp 34-37.

3. Kosarev, Yu. A., Vinogradov, S. V., "Elektricheski izmenyayemyye PZU [Electrically-Alterable ROMs]", Energoatomizdat, Leningrad, 1985, 80 pp.

UDC 681.3

Design of Distributed Monitors for Tracking Computer Network Status

18630290d Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 88 pp 39-41

[Article by P. V. Gamin and V. G. Lyubimkin]

[Text] The development of distributed data processing systems has confronted computer network developers with the task of enhancing network operation. Effective operation of a network can be achieved by allocating the workload rationally among the nodes, choosing proper communication channel speed, changing the hook-up of the network nodes by physical communication lines, and so on.

In order to enhance the efficiency of the data exchange devices, the process of data interchange between tasks must be carefully studied. Obviously, the entire distributed data processing system must be regarded as a unified configuration of network nodes interacting with each other, and not as an assemblage of separate machines processing shared source data.

In order to analyze the operation of a distributed data processing system a large number of network parameters must be logged: the length of the queues at the communication channels, the number of logic channels existing at a given time in the nodes, the interaction of tasks in the computer network, and so on. It is important that logging be done concurrently at several network nodes; otherwise, the resulting information will not be worthy of analysis.

Monitoring problems of this type are currently handled using the control resources available to the network administrator. The administrator may obtain reference information about only one network node.

The administrator's facilities for monitoring the data exchange network include general-purpose monitors, which control the network or the printout of values of the basic network parameters.

Only a small set of data is acquired in this way, and it is impossible to combine data collected from several different nodes for comprehensive analysis and an overall picture of the network's operation.

The multi-purpose monitors available to the network administrator introduce major distortions in the operating process since they take up some of the node's resources which may cause errors in measurements of network characteristics.

In complex distributed data processing systems, special channels are set aside to reduce the error in logged network parameters and to develop adequate supervisory capability. A control subnetwork is established using the dedicated channels, which the central monitoring and control station uses to interact with the network nodes [1]. The distributed systems based on the STO/RV or the SET SM application program package do not have a separate measurement network [2]: network monitoring and control facilities operate on the communication channels used by the network. Thus the resources used by reducing network status monitors and gathering data characterizing the network's operation are current problems.

Distributed network status monitors may be used to track network parameters while the system is in operation and to obtain integrated operating indicators for the distributed system.

A distributed network status monitor is made up of peripheral monitors located at all nodes of each network segment being checked and a central monitor gathering data on the status of the nodes. The monitors are synchronized by the system time service of each node or by a reference event.

However, various sets of network parameters must be monitored during the analysis of the network's operation, while the number of parameters is limited by the problem statement of the measuring process. The design of network status monitors for the full set of parameters inevitably results in gigantic, all-encompassing programs, which is also undesirable.

This problem may be overcome by developing convenient peripheral monitor design facilities for a specific set of parameters. The network monitors should require a minimum of resources at the nodes, thereby minimizing measurement errors. If experimental requirements change or if specific network characteristics are upgraded, the peripheral monitors should be changed.

The monitor design facilities should be convenient to use and geared to specific network software.

The present article describes peripheral monitor design facilities written in MACRO-11 for distributed information processing systems based on the network software packages STO/RV and SET SM.

To shorten the time required to create a monitor, a set of macroinstructions has been developed, geared to the structure of the control network data which are inspected by the monitor.

After analyzing various versions of the network software, we have established a procedure for the structure of the network control data.

All the network data are reduced to blocks, and the blocks are combined into a complex hierarchical structure. There are two basic parts in this structure: descriptors of processes and descriptors of communication lines in the node.

The descriptors of processes (Fig. 1) are combined by a common reference table into process blocks, description PDVTB. There are five types of processes: auxiliary, AUX; communication line control, DDM (there may be several); linking to physical lines, DLX; physical line protocols, DDCMP; and network interaction control, NSP.

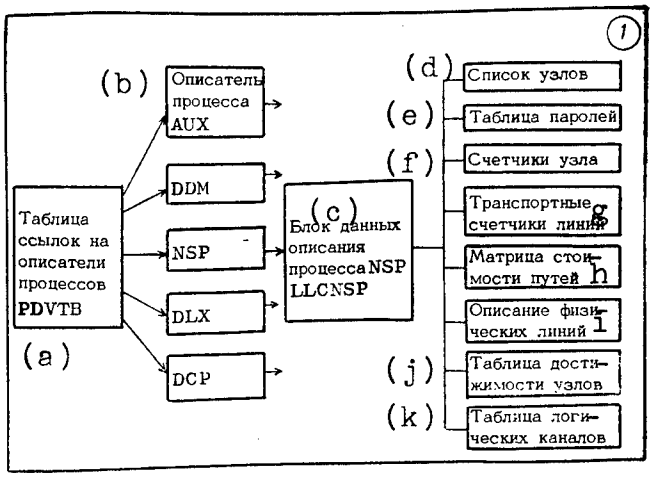


Fig. 1

Key:

a. Process descriptor reference table PDVTB
b. Process descriptor AUX
c. Block of data describing NSP process
d. List of nodes
e. Table of passwords
f. Node counters
g. Line transport counters
h. Path value matrix
i. Description of physical lines
j. Node accessibility table
k. Table of logical channels

The most important information on the status of the node is contained in the blocks of data ssociated with the NSP process. The NSP data contain information on the control of the routing, tables describing the physical lines and logical channels, the access passwords of the node, and so on.

The communication line descriptors (Fig. 2) contain information on the types of communication lines, the working variables needed to organize the operation of the protocol (DDCMP), the parameters of the controller, etc.

Thus, with access to the network control data one may determine how many network resources of the node are occupied at the given moment, and by whom.

From the error counters used in the DDCMP protocol it is possible to determine how well the communication lines are functioning while the network is in operation. The data describing the physical and logical channels can be used to graph the node connections and tasks, the channel workload, and network resources.

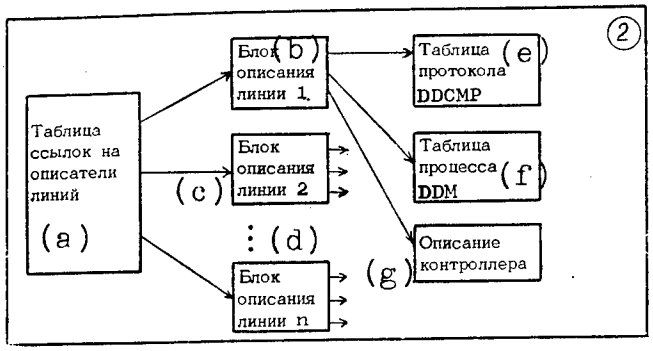


Fig. 2

Key:

a. Line descriptor reference table
b. Line 1 description block
c. Same, line 2
d. Same, line n
e. Protocol table
f. Process table
g. Description of controller

A number of macroinstructions have been developed for programming the monitors, simplifying the use of the network data structure (see table).

Macroinstructions for Constructing a Distributed Monitor

Name of Macroinstruction	Purpose
Macroinstructions for access to network control data	
DEFDF ¤	Description of network control structure data blocks
INDEX	Description of PDV table address
LIST	Macroinstruction sampling data from the network control data structure
PRCDAT	Macroinstruction determining addresses of process description blocks

[table continued on following page]

Macroinstructions for Constructing a Distributed Monitor (continued)

Name of Macroinstruction	Purpose
Macroinstructions of time fixation and event scheduling	
RTIME ¤	Determines current system time
STEVN	Time Scheduling of an event

The second group of macroinstructions is used to tag the data collection process at a particular time, stop the monitors, and determine the current system time at the node. The system time service present at each node is used to synchronize logging.

These resources were used during quality control of individual measurements of the data transmission rate in a communication channel in the network using SET SM network software. A measurement result was rejected when a fault in the physical channel was registered during the transmission of a test block. The delay factor in the channel was 53 percent.

A distributed monitor was thus developed for monitoring node status, and a graph connecting the nodes and the tasks by physical and logical channels was determined. Monitor development time was reduced twofold compared to the time needed without the use of the above macroinstructions.

The accuracy of current time registration at the nodes by the operator was approximate 1 s. The frequency of polling the peripheral monitors was 30 s. The main monitor constructed a matrix describing the graph of connections among the nodes by logical and physical channels. Monitor function was checked on a three-node network built from SM-4 type computers. The RV 3.0 operating system and SET SM network software were installed at all system nodes. The rate of transmission in the communication channels was 600 baud.

The findings revealed that the accuracy of system time service synchronization must be increased in order to fix the logical channels more precisely. To more completely register the dynamism of change in parameters it is desirable to store the gathered information at a logical node for later analysis.

BIBLIOGRAPHY

1. "Seti EVM [Computer Networks]", V. M. Glushkov, L. A. Kalinichenko, V. G. Lazarev, V. I. Siforov, Svyaz, Moscow, 1977, 280 pp.

2. Vasilyev, G. P., Yegorov, G. A., Shcherbina, N. N., "Programmnoye obespecheniye setey EVM [Computer Network Software]", Finansy i statistika, Moscow, 1983, 87 pp.

UDC 519,683,7

Development of Automated Data Processing Systems With Artificial Intelligence Features in the R-Technology Programming Environment

18630290e Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 88 pp 51-54

[Article by V. Yu. Kayurov and B. Ye. Teplitskiy]

[Text] A current task in modern computer science is incorporating intelligent features in automated information processing systems. The presence in the computer of knowledge about a subject domain will enhance the performance of computer-aided design and control facilities, thanks to a more convenient interface with the user, a higher degree of program modifiability, and use of heuristic methods to develop complicated solutions. However, present programming and design techniques and language resources are poorly adapted to creation of systems combining knowledge with traditional data processing procedures. This article describes an approach to the creation of techniques specially designed for such heterogeneous systems by expanding the R-programming technology [1].

We will present knowledge as a system of productions (SP) consisting of a group of rule-productions, a controlling structure, and an information base [2]. We simulate problem solving with the help of the SP by repeated application of the rule-productions which satisfy the current state of the information base, which is itself altered when these rules are implemented. The choice of appropriate rules is the task of the controlling structure, which carries out a particular strategy, while the latter analyzes the attainment of the ultimate target condition by the information base.

A number of algorithmic languages (REFAL, PROLOG, PLANER) have been developed so that computer programs can be written directly as a system of productions. A major drawback of such languages is the necessity of including in the SP elements (rule-products) which mirror pure computer-science and data-processing aspects of the problem and have no direct relation to the knowledge in t e subject domain. The proportion of such elements in the problems of automated information processing systems may be quite substantial.

The essence of the proposed method is incorporating production system fragments in the program's R-diagram in the form of special arcs. The R-diagram represents the general information processing algorithm, while the productions show the intellectual component of this algorithm. This makes it possible to exclude trivial calculation and data-processing procedures from the SP and to define the general data-processing conditions for application of the individual rule-products apart from the controlling strategy of the SP.

The information base of the system of productions is used to describe the situations arising during the process of solving the problem. In a generalized case it contains facts and hypotheses, the truth of which is verified on the left-hand side of the rules. Artificial-intelligence languages oriented to work with SP ordinarily introduce special means for describing an information base. For example, in the PLANER language it is a set of statements constructing a database; in the PROLOG language, the base is formed from constants, object variables, and special syntax terms.

Such specialized logic means of constructing an information base are not adequate for the present approach, which aims at unifying computational, data-processing, and logical procedures in a single program. Therefore, to describe the base, we select abstract data types (ADT), which in the R-programming technology (the RTK-2 configuration of the YeS operating system) are supported by a language-independent model processor. The semantics of the models of the RTK configuration are similar to the ADT adopted in the languages CLU, ALFARO, and MODULA-2, and comes down to the following.

The program describes data types corresponding to the concepts of the subject domain and operations upon it. No special constraints are placed on the nature of the operations: they may involve either verifying certain qualities of the subject domain (existence of relations between data) or alteration of the data themselves, for example, by computation or introduction of new values from external sources (files, terminals). Data quality verification (logic) operations are used to construct the information base of the SP: the conjunction of such operations is written on the left-hand side of the rule-productions. On the right-hand side of the rules we enter the data-altering operations (operation-actions).

We shall distinghish: A is a certain abstract data type; A·P is a logic operation (predicate) on A; A·F is an operation-action on A. Then, we write the system of productions in the form:

$$\begin{aligned}
&A_{11}\cdot P_{11} \,\&\, \ldots \,\&\, A_{1m_1}\cdot P_{1m_1} \Rightarrow A_{1m_1+1}\cdot F_{11};\ \ldots;\ A_{1m_1+k_1}\cdot F_{1l_1};\\
&A_{21}\cdot P'_{21} \,\&\, \ldots \,\&\, A_{2m_2}\cdot P_{2m_2} \Rightarrow A_{2m_2+1}\cdot F_{21};\ \ldots;\ A_{2m_2+k}\cdot F_{2l_2};\\
&\vdots \qquad\qquad\qquad\qquad \vdots\\
&A_{n1}\cdot P_{n1} \,\&\, \ldots \,\&\, A_{nm_n}\cdot P_{nm_n} \Rightarrow A_{nm_n+1}F_{n1};\ \ldots;\ A_{nm_n+k_n}\cdot F_{nl_n}.
\end{aligned} \qquad (1)$$

The result of performing the logic operation is the value "true" or "false." All logic operations A_{ij}, P_{ij} are performed during each operating step of

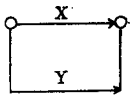
the SP (1). For the rules i^* in which the operations $A_{i^*j} \cdot P_{i^*j}$ assumed the value "true" for all $j = 1, 2, \ldots, m_{i^*}$, we perform in succession the operation-actions on the left side $A_{i^*j} \cdot F_{i^*j}$. The operation of the system of productions is complete when none of the rules can be applied. This case, obviously, includes the attainment of the target condition by the SP. It is sufficient to include in system (1) a rule whose logic operations assume the value "true" only in the desired state of the information base, while the operation-actions alter the values of the data such that no production (not even the one under consideration) can be applied.

In the generalized case, the target condition may be attained in one application sequence of the rule-products and not in another. Therefore, the languages PROLOG and PLANER provide a mechanism of returns, which permits a return to an operating step of the system of productions where it is possible to apply several rules and select the next one from them. Such controlling strategy in some instances may generate algorithms of exponential complexity, which is extremely undesirable for industrial automated data processing systems. If we exclude from the systems of productions the computational procedures and the outside information swapping operations, the fragmentation of the SP is able to reduce the complexity of the generated algorithms.

To accomplish this approach it is important to have an apparatus which regards traditional program structures and the systems of productions from a unified standpoint. We introduce a certain algebraic object, a pseudoprogram, defined on arbitrary elements of the computer memory, including abstract data types [3]. In the algebra of pseudoprograms, the relation of equivalence and the operations of addition "+" and multiplication "*" are defined, making it possible to build complex structures from elementary pseudoprograms.

The pseudoprogram is a partial (possibly multiple-valued) mapping onto itself of a set of memory states and is specified by a system of equations consisting of elementary structures of the base set. A natural correspondence prevails between the expressions in the algebra of pseudoprograms and the program structures determined by R-diagrams, making it possible to regard the formal meaning of these expressions as the substantive interpretation of the R-diagrams.

In particular, we shall consider examples of typical fragments of R-diagrams, in which the symbolx X and Y designate arbitrary pseudoprograms. The expression X + Y corresponds to :

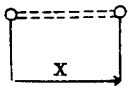


The expression X * Y corresponds to two fragments:

The pseudoprogram Z, being the minimal solution of the equation

$$Z = X * (\varepsilon + Z) + \varepsilon, \tag{2}$$

in which ε is an everywhere-defined identity pseudoprogram, corresponds to the cyclical structure:



For each pseudoprogram we introduce the concepts of the D-domain, which is the set of all memory elements whose initial states may "influence" the result of its execution, and the V-domain, which is the set of elements whose states may be altered by the pseudoprogram. Designating M as the set of memory elements, S as the set of states of these elements, and X as an arbitrary pseudoprogram, we note that:

$$D(X) \subseteq M \quad \text{and} \quad V(X) \subseteq M.$$

The domain of definition of the pseudoprogram X - DEF (X) is a subset of the set of memory states: $\mathrm{DEF}(X) \subseteq S$.

In these terms, the operations on the ADT appearing in the system of productions are pseudoprograms, so that the monadic rule-productions from system (1) can be written as:

$$C_i = (A_{i1} \cdot P_{i1} * \ldots * A_{im_i} \cdot P_{im_i}) * (A_{im_i+1} \cdot F_{i_1} * \ldots * A_{im_i+k_i} \cdot F_{il_i}). \tag{3}$$

Here, the designations $A_{ij} \cdot P_{ij}$ are considered to be the names of partly-determined pseudoprograms which do not alter the memory state. The domain of definition of such pseudoprogram includes those, and only those, program states in which the corresponding predicate $A_{ij} \cdot P_{ij}$ in (1) is true. By analogy, $A_{ij} \cdot P_{ij}$ are interpreted as the names of everywhere-defined pseudoprograms implementing the operation-actions occurring in (1).

Consider the system of equations:

$$\begin{cases} G = Q * (\varepsilon + G) + \varepsilon, \\ Q = C_1 + C_2 + \ldots + C_n. \end{cases} \tag{4}$$

The least solution G of the first equation of this system, in accordance with expression (2), is a cyclical operator. The pseudoprogram Q from the second equation of system (4) involves a selection and execution of one of the

pseudoprograms C_i (i = 1, 2, ..., n), corresponding to the individual rule-productions (3). Thus, system (4) defines a cyclical implementation of the rule-productions which continues until at least one of these rules is fulfilled. The semantic correspondence between expression (4) and the system of production is based on the fact that (as in the SP) the order of selection of the pseudoprograms C_i is not fixed (the operation "+" is commutative in the algebra of pseudoprograms), i.e., system (4) is a nondetermined structure.

Definition of the system of productions in the algebra of pseudoprograms creates the prospect for a formal analysis of programs which combine R-diagrams with a system of productions. For example, if the originally specified system of productions SP_0 after deriving from it certain operations on ADT is partitioned into an R-diagram R and a system of productions SP_1, we may check for equivalence of the resulting pseudoprogram and the pseudoprogram corresponding to the system SP_0. Of practical interest is the analysis of the D and V-domains of the pseudoprograms corresponding to operations in the rule-productions. Thus, if for a certain pseudoprogram C_i defined by expression (3) there obtains:

$$\bigcup_{j=1}^{m_i} D(A_{ij}.P_{ij}) \cap \bigcup_{k=1}^{l_i} V(A_{im_i+k} \cdot F_{ik}) = \varnothing,$$

then the i-th rule of the SP can be executed endlessly.

The complexity of the algorithms generated by the system of productions can be greatly diminished for specialized SPs: commutative and factorizable [2]. In the algebra of pseudoprograms, there is a natural formulation of the sufficient condition for commutativeness of a system of productions:

$$\bigcup_{k=1}^{m_i} D(A_{ik} \cdot P_{ik}) \cap \bigcup_{k=1}^{l_j} V(A_{jk+m_j} \cdot F_{jk}) = \varnothing \;\&$$

$$\& \bigcup_{k=1}^{l_i} V(A_{ik+m_i} \cdot F_{ik}) \cap \bigcup_{k=1}^{l_j} D(A_{jk+m_j} \cdot F_{jk}) = \varnothing,$$

for all $i \neq j$.

The meaning of this relation is as follows: if the D-domains of pseudoprograms corresponding to the conditions of application of the rules do not intersect the V-domains of operation-actions from other rules and if the V-domains of the operation-actions of different rules do not intersect each other, then one and the same result will be obtained for any given allowable application sequence of rule-productions.

In keeping with the above principles, software was developed to support a SP in the environment of the RTK-2 configuration of the YeS operating system. This is in the form of a special module (processor of the system of productions), which is included in the assignment for translating programs from the language of R-diagrams after they are converted to linear form. The

functions of this module are: checking the syntax of the rule-productions, diagnostics of the possibility of cycle-locking in the SP, analysis of conditions of commutativeness, and converting fragments of the system of productions into a program in the host language. At present, the processor of the system of productions is able to use assembler of the YeS operating system and PL/1 as the host language.

The concept of using SP processor for developing automated information processing systems software consists in the following. When a program is being designed by the R-technique, the development engineer determines the basic concepts of the subject domain and constructs models (ADT) mirroring the semantics of the subject domain in the form of operations on abstract data types. The V and D-domains of the corresponding pseudoprograms are explicitly indicated for the operations. In the R-diagram of the program, the engineer distinguishes the portions associated with the knowledge in the subject domain and forms them into fragments of the SP. These fragments in their entirety constitute the knowledge base of the system. Through the resources of the Organizator subsystem [1] of the RTK configuration, these program elements are handed over to the user with the understanding that much work remains to be done.

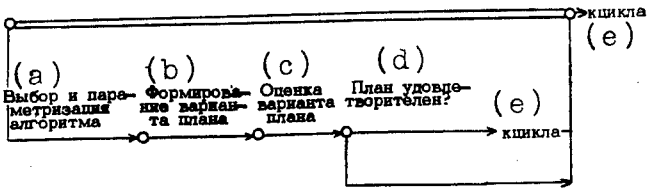
During the process of set-up and running of the problems, the user may introduce changes in the fragments (supplement or modify the knowledge base) using the basic concepts of the subject domain--the operations on ADT. After the changes, the user invokes the translation procedure for the entire program, obtaining a new version of its load module or a message stating that the changes made in the SP are incorrect.

We shall illustrate the use of this approach with the example of an automated system being developed for scheduling research and development projects. This system produces calendar schedules for work that is to be done in order to create new equipment and includes a database, created and maintained by a universal database management system; an assembly of programs implementing the algorithms of scheduling theory; and a knowledge base in the form of the above-described system of productions. During formulation of the project schedules, the knowledge is used to construct mathematical models of the problems of scheduling theory, to select the algorithm for their solution, to explain the obtained results to the users, and to issue recommendations on altering the initial data for the planning.

Let us consider a simplified fragment of the knowledge base for selecting and parametizing the algorithm for solving a problem of scheduling theory in the network form. This selection can be done by an expert having experience in the use of such algorithms, taking into account estimates of the resulting schedule alternatives from the standpoint of the system users.

The facilities of the SP processor make it possible to combine this expert knowledge and algorithms for mathematical model-based scheduling into a single cyclical program, represented by the following R-diagram:

[diagram on following page]



Key:

a. Selection and parametrization of algorithm
b. Formation of alternative plan
c. Evaluation of alternative plan
d. Plan satisfactory?
e. To cycle

In order to simulate the deliberations of the expert in the SP corresponding to the arc "Selection and parametrization of algorithm," definitions of operations on ADT embracing around 60 concepts of the subject domain are constructed during the program development phase. The SP itself, in substantive form, appears as follows:

1) QUANTITY-OF-WORK=(VERY-MUCH)→ALGORITHM:=(0)
2) QUANTITY-OF-WORK=(MUCH)&EXCEEDING-REQUIREMENTS-FOR-RESOURCES=(LARGE)& ALGORITHM=(2)&TIME-EVALUATION-OF-PLAN=(UNSATISFACTORY)→ALGORITHM:=(1)
3) QUANTITY-OF-WORK=(NOT-VERY-MUCH)&ALGORITHM:=(3)&RESOURCE-EVALUATION-OF-PLAN=(UNSATISFACTORY)→ALGORITHM:=(4)

The predicates on the left side of the rules consist of the designation of the ADT (QUANTITY-OF-WORK, ALGORITHM, etc.), the operation on them (=) and a parameter (in parentheses). During initiation of these predicates, the truth or falseness of the defined statement is verified. For example, when executing the operation QUANTITY-OF-WORK=(MUCH), the number of records of the determined type contained in the database is counted and a check is made as to whether this number corresponds to the value "much" of the linguistic variable "quantity-of-work."

The operation":=" on the ADT ALGORITHM (right side of rules) assigns to a certain variable the value of the parameter (0, 1, 2, etc.). This value is then used in the program module corresponding to the arc of the R-diagram "Formation of plan alternative", and also in the operations ALGORITHM= during subsequent movements along the arc.

The basic system version (supplied to the users) includes more than 300 rules in the SP corresponding to the arc "Selection and parametrization of algorithm." As experience is gained in the use of the system, those responsible for maintaining it may introduce changes in the SP.

It should be pointed out that changes in the SP within the framework of the ADT defined by the development engineer and the operations upon them are

quite simple and require only an understanding of the essence of the problem and the meaning of operations on the ADT. On the other hand, introducing new and important facts in the knowledge base may prove very complicated or completely impossible without a review of the entire problem-solving algorithm. Therefore, specialists in the subject domain should be consulted when programs are being developed for compilation of the original SP, so that all basic concepts of the subject domain are included in the operations on the ADT. In this case, information about the relationship of these concepts (the knowledge proper in the subject domain) may be added to the problem software at any time in its development, debugging and use.

BIBLIOGRAPHY

1. Velbitskiy, I. V., "Tekhnologiya programmirovaniya [Software Engineering]", Tekhnika, Kiev, 1984, 279 pp.

2. Nilson, N., "Printsipy iskusstvennogo intellekta [Principles of Artificial Intelligence]", Radio i svyaz, Moscow, 1985, 376 pp.

3. Kayurov, V. Yu., "Equivalence and Formal Transformations of R-Diagrams", "R-Tekhnologiya programmirovaniya i sredstva yeye instrumentalnoy podderzhki [The R-Programming Technique and Its Development Tools]", IK AN USSR, Kiev, 1982, pp 96-102.

END

[redacted]

22161
55
NTIS
ATTN: PROCESS 103
5285 PORT ROYAL RD
SPRINGFIELD, VA 22161

Foreign Broadcast Information Service (FBIS) and Joint Publications Research Service (JPRS) publications contain political, economic, military, and sociological news, commentary, and other information, as well as scientific and technical data and reports. All information has been obtained from foreign radio and television broadcasts, news agency transmissions, newspapers, books, and periodicals. Items generally are processed from the first or best available source; it should not be inferred that they have been disseminated only in the medium, in the language, or to the area indicated. Items from foreign language sources are translated; those from English-language sources are transcribed, with personal and place names rendered in accordance with FBIS transliteration style.

Headlines, editorial reports, and material enclosed in brackets [ ] are supplied by FBIS/JPRS. Processing indicators such as [Text] or [Excerpts] in the first line of each item indicate how the information was processed from the original. Unfamiliar names rendered phonetically are enclosed in parentheses. Words or names preceded by a question mark and enclosed in parentheses were not clear from the original source but have been supplied as appropriate to the context. Other unattributed parenthetical notes within the body of an item originate with the source. Times within items are as given by the source. Passages in boldface or italics are as published.

SUBSCRIPTION/PROCUREMENT INFORMATION

The FBIS DAILY REPORT contains current news and information and is published Monday through Friday in eight volumes: China, East Europe, Soviet Union, East Asia, Near East & South Asia, Sub-Saharan Africa, Latin America, and West Europe. Supplements to the DAILY REPORTs may also be available periodically and will be distributed to regular DAILY REPORT subscribers. JPRS publications, which include approximately 50 regional, worldwide, and topical reports, generally contain less time-sensitive information and are published periodically.

Current DAILY REPORTs and JPRS publications—except those which have restricted dissemination due to copyright or other reasons—are listed in *Government Reports Announcements* issued semimonthly by the National Technical Information Service (NTIS), 5285 Port Royal Road, Springfield, Virginia 22161 and the *Monthly Catalog of U.S. Government Publications* issued by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402. Information on the Official Use Only JPRS Reports must be obtained from FBIS at the address or telephone number given below.

U.S Government offices may obtain subscriptions to the DAILY REPORTs or JPRS publications (hardcover or microfiche, unrestricted and Official Use Only reports) at no charge through their sponsoring organizations. Department of Defense consumers are required to submit requests through appropriate command validation channels to DIA, RTS-2C, Washington, D. C. 20301. (Telephone: (202) 373-3771, Autovon: 243-3771.) For additional information or assistance, call FBIS, (703) 338-6735, or write to P.O. Box 2604, Washington, D.C. 20013.

Back issues or single copies of the DAILY REPORTs and JPRS publications are not available.

Both the DAILY REPORTs and the JPRS publications are on file for public reference at the Library of Congress and at many Federal Depository Libraries. Reference copies—except the Official Use Only Reports—may also be seen at many public and university libraries throughout the United States.

[redacted]