MINISTRY OF RADIO INDUSTRY OF THE USSR
ACADEMY OF SCIENCES OF THE USSR
STATE COMMITTEE FOR SCIENCE AND TECHNOLOGY
Under the Council of Ministers of the USSR

V. A. Myasnikov, M. B. Ignatiev, V. A. Torgashev

RECURSIVE COMPUTERS

Report at a scientific and technical seminar
"Multiprocessor Computing Systems"
Moscow, November 21-22, 1977

Preprint No. 12, 1977

Moscow

Recursive computers /RC/ represent a new class of computers, the structural and software organization of which is determined by the following basic principles proposed in [1]:

1. The internal language of RC includes recursively defined program elements (generalization of machine instructions) and data elements (generalization of machine words), which can be of arbitrary complexity.

2. The order of execution of program elements of any program presented in the RC can be specified implicitly using functional (in the general case, recursive) relations and dynamically determined directly during the execution of the program, thereby realizing recursive-parallel control.

3. The internal memory of the RC consists of blocks, the elements of which can be both cells and memory blocks, which themselves, in turn, can consist of blocks, and the connections between the memory blocks can be programmatically rebuilt.

4. The external physical structure of the machine is uniquely determined for an arbitrary number of elements using a finite number of recursive relations.

5. The internal structure of the machine, which determines the connections between the computational processes that are implemented in the RC, is flexible, programmable and dynamically reflects the structure of the problems being solved at each moment in time.

The implementation of these principles makes it possible to create multiprocessor computers consisting of arbitrarily large number of processors. The effective performance of such machines is proportional to the number of processors for wide

classes of problems being solved, while the complexity and efficiency of a software system does not depend on the number of processors.

The above principles are very general in nature, so that on their basis it is possible to build an infinite number of the most diverse RCs. Below we will consider only a limited subclass of the general class of recursive computers, focused on the current state of microelectronics.

By its structure, any RC corresponds to a distributed or consolidated computer network, in the nodes of which there are processors of various types, terminals, external devices and recursive machines of lower levels. The main difference between RC and any modern computer network or multiprocessor computing system is the organization of the internal language. Modern computer networks in most cases are focused on individual processing of information in each node of the network, which usually corresponds to a fairly autonomous computer with a traditional structure. The internal language of any network is reduced to a small number of communication commands and is not organically connected with the internal computer languages that correspond to the nodes of the network. The main mode of operation of the network is that each computer of the network solves its own separate problem, using for this mainly only its own resources. If a task cannot be solved on any computer in the network, then either the operating system of the network, or even more often the user must split the task into separate parts, each of which can be solved in one node of the network, and this division is carried out in advance before the beginning of solving the problem. The efficiency of such parallelization drops sharply if the subtasks into which the task is divided turn out to be interconnected due to the difficulties of mutual synchronization of these subtasks and delays in the exchange of information between network nodes.

While the internal languages of most of the currently known computer networks and multiprocessor systems are based on the concept of a sequential computer with centralized control, the internal RC language is primarily targeted to the implementation in a multiprocessor machine with decentralized control. Therefore, the problem of parallelization is solved in RC not by external means using complex operating systems, but is carried out in a natural way when interpreting language elements.

The main control method adopted in the internal language of the RC is based on an asynchronous approach to control of a computing processor [2, 3] and the principle of control from data [4-9]. The essence of this method lies in the fact that the order of execution of operators in the program is not preset in the form of an explicitly presented sequence of operators, but is dynamically determined at each moment of time by checking the conditions of readiness of operators to work. The most common condition for an operator to be ready is the presence of operands. This control method allows us to

preserve the natural parallelism inherent in almost any task during program creation and easily use the natural parallelism in program execution, completely eliminating the synchronization problem at the same time. However, the direct implementation of this method is associated with significant difficulties due to the large number of language elements (both operators and data) that must be checked for readiness. To solve this problem, it is necessary, firstly, to be able to parallelize the control process and, secondly, to minimize the number of checked elements. Both conditions can be met with a single tool: deep structuring of both program and data.

Let's first consider the issues related to data structuring. Suppose we want to add two sequences of numbers, A and B, each containing 10,000 numbers. If only individual numbers can be elements of the machine language, then two queues of operands will be lined up to a single addition operator, each of which must contain a sign of readiness. The operator is ready to work when the first pair of numbers arrives at its inputs. After each addition, the pair of operands involved in the operation is destroyed and a check is made for the presence of the next pair of operands at the input of the operator. Thus, each operation of addition of two numbers corresponds to the operation of checking the readiness of the operator to work, and the latter operation may require significantly more time than addition. In addition to a significant increase in the time spent on control of the computational process, in comparison with the usual sequential control method, in this case it is difficult to realize (at least at the machine language level) the potentially high possibilities of parallelizing the computational process due to the lack of information about the length of the operands in the program of the operator's interpretation and the difficulty of dividing queues from operands into separate parts.

Suppose now that the elements of machine language can be not only individual numbers, but also arrays of numbers. Each line, in addition to the sign of readiness, must include in its composition an indicator of the number of elements in the array. Now, instead of a queue of 10,000 pairs of operands, there will be only two operands at the input of the addition operator, and accordingly, to add all these numbers, it is enough to check the readiness of the operator for operation only once. Here, structuring the data by introducing only one additional level made it possible to minimize the number of readiness checks, but, however, worsened the dynamic characteristics of the computational process, since if earlier the computation began with at least one pair of numbers in the machine, now it is necessary to have ready immediately all elements of arrays A and B. Although the parallelization of computations in this case can be performed directly during the interpretation of the addition operator, this parallelization is of a static nature and imposes rather high control requirements (it is necessary to know the number of free processors before starting the execution of the operator and,

accordingly by number, divide the operands into parts, forming a description for each part, and then glue the results of adding these parts into one common result).

Let's carry out an even deeper structuring of the data.

Let $A = A_1, \ldots, A_i, \ldots, A_{10}$; $B = B_1, \ldots, B_i, \ldots, B_{10}$; for any $i = 1, \ldots, 10$; $A_i = A_{i1}, \ldots, A_{ij}, \ldots, A_{i10}$; $B_i = B_{i1}, \ldots, B_{ij}, \ldots, B_{i10}$; for any $j = 1, \ldots, 10$; $A_{ij} = A_{ij1}, \ldots, A_{ij10}$; $B_{ij} = B_{ij1}, \ldots, B_{ij10}$. Here the variables $A, B, A_i, B_i, A_{ij}, B_{ij}$ are independent elements of the language, each of which has its own description containing the sign of the readiness of the element and the number of its components (the description or passport of the element may contain other information about the element, for example, is the element terminal or not, numeric, logical or symbolic, for numbers, the radix and the way of representation can be determined, etc.). Execution of the addition operator over such operands leads to recursive replication of this operator up to the terminal level, that is, if $C = A + B$, then $C_i = A_i + B_i$, where $i = 1, \ldots, 10$ and $C_{ij} = A_{ij} + B_{ij}$, where $j = 1, \ldots, 10$.

Thus, in the process of interpreting the addition operator, it will naturally be transformed into 100 of addition operators, but defined over terminal elements, each of which corresponds to an array containing 100 numbers. Here parallelization is dynamic and, which is very important, requires minimal control, being a natural part of the operator interpretation process. A nonterminal data item is considered ready to work if it contains at least one ready component. Therefore, in order for the operator to start executing, it is sufficient that each of its operands has at least one terminal element ready for operation, that is, the dynamic characteristics of the computational process remain high enough, although worse than in the case of unstructured data.

RC language and program structuring play a role that is no less important than the role played by data structuring. The number of operators in a program can be quite large, but at any given time, only a certain part of them will be ready for work, and often whole groups of operators can be identified in the program, which obviously cannot be executed at the moment. For example, in the simplest network of operators shown in Fig. 1,a, until the variables A and B are ready, not a single operator will be ready for execution. Therefore, due to structuring, it is possible to significantly reduce the number of software elements to be analyzed at each moment of time, So, if the network of operators shown in Fig. 1,a, is represented as one software element that directly includes two elements, the first of which is contains operators 1-4, and the second operators 5-8, then until the variables A and B are ready, the readiness check will be carried out not for eight, but only for one program element.

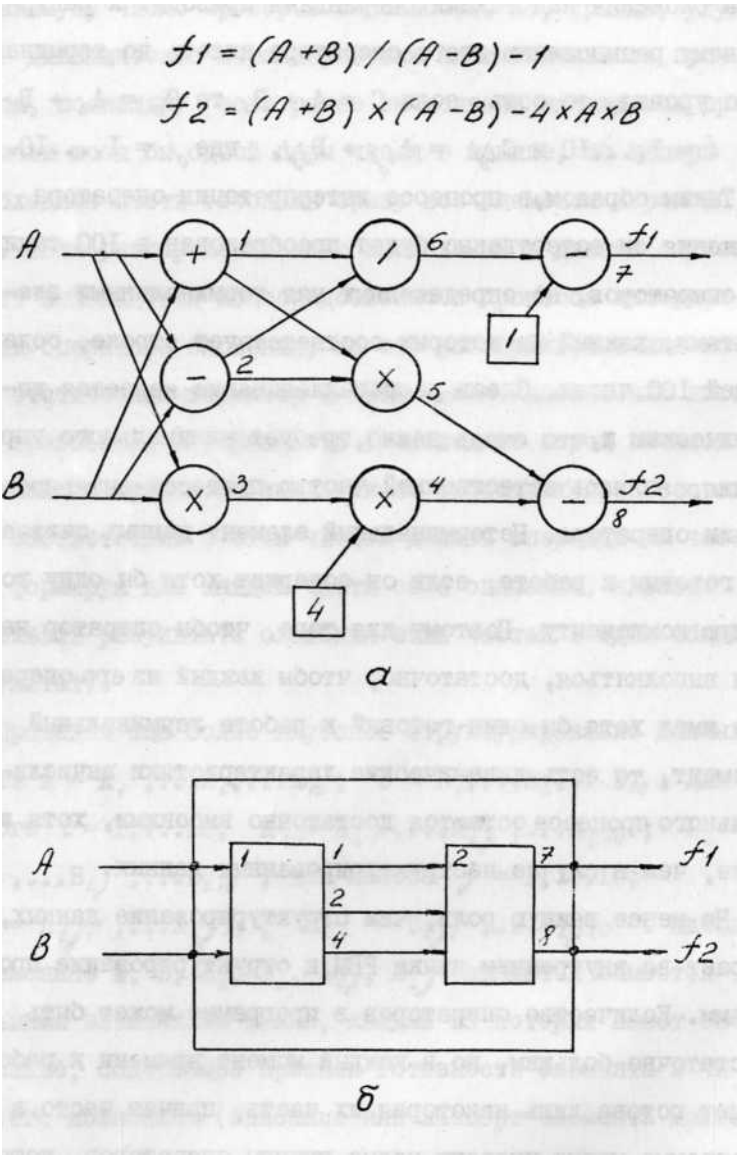

Fig.1

A non-terminal program element in the internal language of the RC is called a block. A block can include blocks or statements, or both. The organization of the computational process at the block level uses a recursive-parallel procedure for interpreting blocks, which is natural in this case. As soon as the block interpretation process finds a program element ready for execution among the blocks immediately contained in it, a new block interpretation process is recursively generated, to which the

element ready for execution is transferred for interpretation. The process will unfold recursively until it reaches the terminal elements of the statements. For example, when variables A and B are ready, the interpreter of the external block (Fig. 1,b) recursively generates its own copy, which will interpret the first internal block, and both interpreters can subsequently work simultaneously (if the recursive generation of a new interpreter was accompanied by the allocation processor).

So, thanks to the replacement of the sequential principle of control with the principle of asynchronous control from data in combination with deep structuring of programs and data in the internal RC language, automatic dynamic parallelization becomes a natural part of the process of interpreting language constructs and can be carried out without any involvement of the operating system.

However, it should be noted that the application of the above control method in its pure form in many cases turns out to be impractical due to a number of disadvantages of this method (for example, the need to use recursive processes instead of iterative ones, a large amount of control required even for purely sequential processes, the need to regenerate data when they are reused, there is a deep discrepancy between traditional algorithms and programming languages oriented to a sequential computer, this control method, etc.). Therefore, a number of additional mechanisms are built into the internal language of the RC, which make it possible to eliminate or compensate for the indicated disadvantages. Such mechanisms, in particular, include: special control operators that provide blocking (temporary shutdown from work) and activation (unblocking) of any program elements and data elements; operators for erasing variables; control variables that enable and disable operators and blocks; descriptions of variables that can be included in the composition of blocks (and, accordingly, not included in the composition of the variables themselves, and a number of other mechanisms, for the description of which a more detailed consideration of the internal language of the RC is required). In addition, in recursive machines, it is allowed to use not one, but four different methods of controlling the computational process, each of which corresponds to a certain type of block: a compiler block, a sequential block, a function block, and a free block.

The block compiler follows the traditional sequential control method. The execution of such a block begins with the transformation of all its program elements into a single program that does not require further interpretation. Although such a transformation of a program is carried out within the framework of one language, in its essence it is equivalent to translation (compilation) of a program from a high-level language to a low-level language. At the end of compilation, the resulting program is transferred to a single executive processor, where it is executed, starting from the first instruction, just like a normal program in a traditional computer.

The sequential block also implements the sequential control method, but in the interpretation mode. Such a block allows only one of its program elements to be controlled at a time, while explicitly defining the next element to be executed after the given one. Control functions during the execution of a sequential block are always performed by one processor, but several processors may be required to execute the elements of this block, and if the executed element is again a block, then at least one of these processors will perform control functions. Thus, the sequential block, unlike the compiler block, allows parallelism at its lower levels.

To execute the block function, each program element of the block is associated with one or more processors (depending on the type of the element and on the types of input data of this element); all dedicated processors are interconnected in accordance with the block structure, after which the computational process is launched for execution. All control of the block-function after its start is reduced to pumping input variables and pumping out output variables. The block function provides static parallelization of the computational process, similar to the way it is done in analog computers, digital integrators, and homogeneous computing environments.

A free block selects from its composition those program elements for which the data is ready. Each of these elements can be allocated its own processor (if there are free processors), which ensures the execution of the element. In contrast to the block-function, a free block provides dynamic parallelization of the computational process, since at each moment of time processors are allocated only for those program elements that can actively use these processors.

A free block is the main block type used in PBMs. The rest of the block types can be used at the request of the programmer only in special cases. So, for example, a block-compiler can be used to emulate traditional computers on recursive machines, as well as in cases where only one processor can be allocated to execute a program. The block function is used when it is required to ensure maximum performance when solving vector problems. It is advisable to use a sequential block with limited RAM resources or if the program corresponding to the block is known to be sequential.

It is interesting to note that most of the additional internal language mechanisms discussed above are aimed at introducing sequential elements into the language, violating the natural order of program execution, which corresponds to maximum parallelism.

One of the main distinctive features of the RC is the wide use of queues both in the interpretation of programs and in the processing of data, and, what is especially important, the order of elements in these queues does not play a significant role. Indeed, for the block interpreter it does not matter in what order to check the readiness of the

internal program elements of the block for operation, and when executing statements, the order makes sense only at the lower, terminal levels of data structures, which correspond either to individual words or small strings. Due to this property, delays in the transfer of information from one processor to another do not significantly affect the performance of the RC, provided that the throughput of interprocessor communication channels is not lower than the throughput of the processors themselves (otherwise the processors will be idle, and queues will appear in the communication channels).

Any RC consists of two parts: a commutation field and an operating field. The switching field contains switching processors of one or several types, interconnected so that they form a certain recursive structure (the simplest version of a recursive structure is an iterative or homogeneous structure, for example, a matrix type), which determines the external structure of a recursive machine [1, 10-14]. The operating field consists of control and executive processors and external devices connected to the switching field, and, most often, not directly related to each other. Connections between the elements of the operational field of the RC are carried out during the operation of the machine by laying and rebuilding connections in the switching field, similarly to how it is done in modern automatic telephone exchanges. These connections determine the internal structure of the RC, which dynamically changes in the process of solving the problem, and as a whole at each moment of time reflects the structure of the problem.

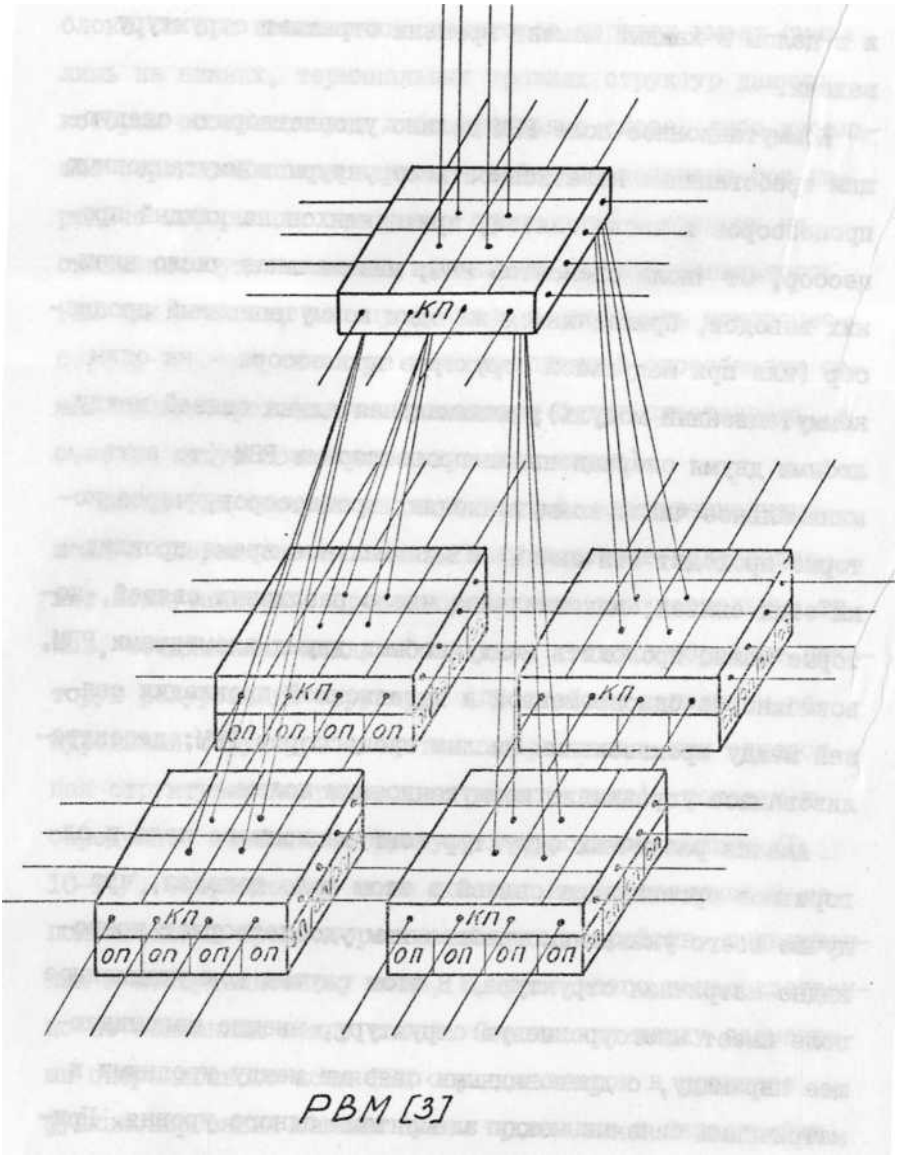

Fig.2

The switching field of the RC must satisfy the following requirements: the independence of the structure of the switching processors and the number of connections per processor from the number of elements of the RC; the minimum number of external pins per switching processor (or with a modular processor structure per switching module); the minimum length of links between any two operational processors of the PBM (that is, the minimum number of switching processors through

which these links pass) and the minimum time for laying these links; the maximum number of different connections that can be laid between any two RC elements, the possibility of simultaneous and independent laying of connections between an arbitrary number of RC processors; decentralized control of the switching field.

Analysis of various structures of the switching field and algorithms for organizing links in this field showed that the tree-matrix structure best meets the specified requirements. In this case, the switching field has a multilevel structure that looks like a pyramid, with tree-like connections between levels and matrix connections between elements of the same level. An example of an RC with such a structure is shown in Fig. 2. This structure of the switching field, in combination with original algorithms for establishing links based on the principle of path indicators to free processors, and the widespread use of the pipeline principle of information processing in switching processors, provides a quick and effective restructuring of the internal structure of the RC. The switching field control functions are fully distributed among the switching processors, which in the course of their work use information only about their state and the states of their nearest neighbors. Due to this, the algorithms for the operation of the switching field do not depend in any way on the number of processors included in the RC. The operational field of the RC mainly consists of elements connected to the lower level of the switching field, characterized by the maximum number of connections. However, it is allowed to connect operational elements (especially control processors) at higher levels of the switching field.

In recursive machines, all internal memory is distributed among separate processors, but any operational processor can connect an arbitrary number of other processors, using them as memory blocks with extended functionality. At the same time, an increase in the memory access time due to delays in the switching field and processors that control the memory has practically no effect on the performance of the RC due to the features of the internal language discussed above. For similar reasons, a long access time to information located in external memory affects the performance of the RC much weaker than the performance of any of the modern computers.

In programs presented in the internal RC language, any addressing of both data and program elements is carried out exclusively with the help of indices and names, and the transition to physical addresses is carried out only in the process of interpreting the program, and the use of physical addresses is always limited by the limits of the own memory of each processor. Therefore, any RC program operates not with the physical, but with the virtual memory of the machine, and the efficiency of this virtual memory is much higher than that of traditional computers. Indeed, the structure of programs in these computers does not allow one to distinguish information that will be required for

the operation of the machine in the near future (and, therefore, must be in the operative memory), from information that will not be required soon (and, accordingly, can be placed in external memory). Therefore, the exchange of information between external and operational memory, as a rule, is of a random nature and, due to the high inertia of this exchange, in many cases leads to a significant decrease in computer performance. In RC, however, it is always possible to clearly distinguish between data elements and programs according to their degree of activity. The lower the activity of an element, the less likely it will be used in the near future. Therefore, elements with low activity are transferred to external memory, and more active elements are transferred in their place, perhaps long before they begin to be performed. It should be noted that the virtual memory of the RC is nonlinear and unlimited in size, while in conventional computers the size of the virtual memory is limited by the bit width of the addresses.

Due to the high level of the internal language of the RC and the recursive-parallel method of controlling the computational process, it becomes possible to significantly simplify the operating system of the machine by transferring many of its functions to the interpreters of the elements of the internal language. For example, the tasks of parallelizing the computing process, organizing multiprogramming, organizing simultaneous work with a large number of terminals, managing data, allocating machine resources and a number of others can be solved with minimal participation of the operating system (by the operating system we mean here a set of software tools that provide centralized control of the system generally). However, in RC, there are situations that require, to one degree or another, centralized control, for example, working with common for a group of users arrays of programs and data, solving several tasks with different priorities on one machine at the same time, restoring information lost during program errors, etc.

The RC operating system has the same structure in the internal RC language, like any other program, that is, it is one block, which includes other blocks. This block is external to all other programs, that is, any user programs are considered as internal blocks of the operating system. Structurally, the operating system is located in switching processors, forming a tree with inter-level connections. This tree is formed during the initialization of the machine immediately after it is turned on and then dynamically changes depending on the activity of certain parts of the operating system. Operational processors can also be included in this tree, if necessary. The RC operating system is distributed throughout the machine, and this distribution is not functional, but mainly regional in nature, that is, any of the processors included in the specified tree implements the functions of the operating system only in a limited area of the RC. The methods for parallelizing control in the operating system are the same as for any other program. If some processor that implements the OS cannot cope with its tasks, then it

automatically connects another processor to itself (perhaps, taking it away from lower-level units) and transfers to it part of its functions or part of the service queue. When the load decreases, the processors are automatically released, starting from the lower levels.

Recursive computers are distinguished by extremely high reliability and survivability, and these abilities increase with the increase in the number of RC elements [13, 14]. Any of the switching processors periodically checks the operability of adjacent elements (operational or switching processors), and in case of failure of any element, the switching processors surrounding this element block access to it, automatically shutting down the element from work. Such disabling of the element will somewhat worsen the parameters of the machine, but will not in any way affect the execution of programs, since programs do not contain physical addresses of processors and are absolutely relocatable. The deep structuredness of programs in the internal RC language allows you to always localize any program error with an accuracy of a certain block, and thus facilitates the correction of these errors. Some of the most annoying software errors caused, for example, by a crash in the machine when control is transferred to the data area, are completely eliminated. In a recursive machine, it is possible to increase the reliability of any program or part of it, not only by repetitions, but also by repeated execution of the program in different parts of the machine.

Different recursive computers can differ from each other both in the number of elementary processors and in their nomenclature. In small RCs, it is advisable to use the same type of operating processors. In medium RCs, execution (in particular, arithmetic) processors can differ significantly from control ones in terms of data format, speed of performing arithmetic operations, and operating frequency. In this case, you can limit yourself to only two types of operating processors. In large and extra-large RCs, the range of executive processors can significantly increase due to the inclusion of high-performance or problem-oriented processors (for example, matrix or pipeline) into the machine, which differ sharply in their parameters and information characteristics from typical RC executive processors. It should be noted, however, that high-performance processors do not have to be used to build large and ultra-large RCs. High performance can be achieved even with a minimum range and the simplest structure of operating processors, including a sufficiently large number of them in the machine. However, at the same time, there may be such tasks that are of a purely sequential nature, the solution time of which will be determined not by the performance of the RC as a whole, but by the performance of a single processor.

As indicated above, a recursive computing machine can consist of any number of elementary processors. An arbitrary number of additional processors can be connected to any already working RC without making any changes to the structure of the original

machine. In this case, a new, more powerful RC is automatically obtained. Any two or more RCs with different characteristics and possibly consisting of different processors can be combined into one recursive machine without any changes in the structures of the original RCs. All these capabilities are provided through the appropriate organization of the switching processors.

Each switch processor has two communication buses: synchronous and asynchronous. Using a synchronous bus, identical switching processors are combined into a linear structure of limited dimensions. This switch strip has a single clock and provides a high speed of information transfer between elements. By means of an asynchronous bus built according to the "request-response" principle, the switching processor is connected to other elements of the RC, which may differ from it both in informational and temporal characteristics, but satisfy the following two requirements: standard electrical levels of input and output signals and the presence of signals "request" and "response", providing an asynchronous nature of data exchange.

A limited number of elements are connected to the asynchronous bus, each of which can have its own clock. As a rule, the number of these elements does not exceed two.

The standard data exchange format in the RC corresponds to one byte, but by introducing common synchronization on a group of commutation lines, it is possible to ensure the transfer of information in words of any predetermined format. Switching processors also make it possible to carry out format transformations of the transmitted data directly during transmission. In particular, any format can be converted to a standard byte form and, conversely, from a standard form can be converted to any format (including bit), and these conversions can be set programmatically.

If any elementary RC processor or external device has the timing characteristics of the asynchronous data transfer bus higher than that of the switching processor, then several switching processors can be connected to this bus and due to the multiplexing effect of these processors by dividing the request signals and constant switching in time response signals can provide information transfer at the required rate. In general, the commutation field of the RC is a universal programmable interface that provides not only the exchange of information between the individual elements of the RC, but also the coordination of their information and time parameters. At the same time, almost any standard external devices can be connected directly to the switching field (without any additional interface circuits).

Any RC has a certain number of free asynchronous buses corresponding to the peripheral elements of the machine. These buses are used when combining several RCs into one, and with the growth of the machine, the number of free buses also increases.

Recursive machines, from the smallest to the largest, are fully software compatible using the standard version of the internal language. Any program written in this version of the language does not depend on the structure of a particular RC, the number of processors included in the machine, on the nomenclature and command system of these processors (in non-standard versions of the language, the user can set the structure of the language himself using his own interpreters and control microprograms; such programs may turn out to be incompatible for different RCs due to their machine dependence). The internal language of the RC does not correspond to any particular machine, but to an idealized virtual recursive machine consisting of an unlimited number of processors and, accordingly, having unlimited memory. Reflection of a virtual machine into the structure of a specific RC is carried out directly during the execution of the program by interpreting generalized operators and blocks of the internal language into microprograms of specific processors and translating conditional indices and names into physical addresses of the internal memory of these processors. This translation can be avoided if the internal memory of the processors is associative.

One and the same program, presented in the internal language of the PBM in different machines, will execute in different ways. For example, in a small machine a program can be executed sequentially on one processor with a large number of external memory accesses, while in a large RC the same program can be executed simultaneously on a large number of processors of various types, without using any external memory accesses at all. Moreover, even in one RC with different runs of the same program, it can be executed in different ways, if other programs are EXECUTED simultaneously with it.

Among the various blocks of the internal language of the RC, only one (block-function) does not have invariance to the number of processors in the machine, since it can start its execution only after a separate processor is allocated to each of its program elements. However, in the program for interpreting the block function, it is possible to introduce a check of the correspondence of the number of program elements of the block to the limit number of processors that can be allocated in a particular RC to a given program (this number may depend not only on the total number of processors in the machine, but also on the priority of the program). In the absence of such a match, the block function is converted to a free block by simple renaming, and thus it is possible to execute it, albeit with worse timing characteristics.

An important feature of the internal RC language is the possibility of its restructuring by either modifying the standard interpreter or building a new interpreter, and the program of this interpreter can be written and debugged using all the facilities of the standard RC language. In the same program, it is allowed to use separate parts written in different machine languages. Such capabilities make it easy to emulate any

other machine in the RC. In addition, any traditional machines with their own software can be included in the RC as operational processors. This opens up the possibility of using a recursive machine as a system control device in traditional multiprocessor computing systems.

The economic efficiency of RC is determined by a number of factors, the main of which is the ability to build computers of the most varied parameters on a single technological base with the widespread use of large integrated circuits of a small nomenclature. The high degree of seriality of these LSIs determines their low cost and, accordingly, allows to significantly reduce the cost of computer production.

Since each RC consists of a large number of repetitive elements with a small number of cross-links and a small number of external leads, the nomenclature and complexity of printed circuit boards is significantly reduced, board-to-board installation and the design of the machine as a whole are simplified, machine inspection during its manufacture is greatly facilitated and technological cycles are shortened. , which also leads to a decrease in the cost of computer production.

Recursive machines form a series of computers that are practically continuous in terms of their parameters and cost. Thanks to this, the user can always purchase a machine that exactly meets his current needs, and if these needs increase, it is easy to expand this machine by connecting additional elements. At the same time, he will not have to make any changes to the structure of an already working machine and previously used programs. These properties make it possible to avoid unnecessary costs for the acquisition of a computer that is too powerful for the user's needs, reduce the effect of obsolescence of the machine (newly added elements may have better characteristics than the elements of the original RC) and, accordingly, provide high economic characteristics of recursive machines.

Recursive machines remain operational even in the presence of multiple element failures. With relatively little redundancy in performance and internal memory, the machine may not be repaired at all during its entire service life, or it may be repaired sporadically at long intervals, which significantly reduces operating costs. Since during the repair of the RC it is possible not to remove the failed elements from the machine, but only to add new ones, it becomes possible to significantly simplify the design of the RC by reducing the number of detachable elements in it.

The high level of the internal language of the RC provides a compact representation of working and control programs, which can significantly reduce the amount of RAM or reduce the intensity of information exchanges with external memory, which is equivalent to increasing the performance of a computer.

Recursive machines are easily combined into a distributed computer network, and can also be included in existing computer networks.

Recursive computers have a significantly higher degree of versatility than traditional computers, since they can adjust their internal structure and internal language to the structure of the problems being solved. Thanks to this, the efficiency of the use of the equipment is sharply increased.

RCs of different class differ from each other mainly only by the executive systems (the nomenclature and time characteristics of the executive processors and, in some cases, by the structure of direct connections between the executive processors in addition to the switching field). Moreover, within the framework of any class of RC (machines for computational or non-computational tasks or universal RC) it is possible to obtain any predetermined parameters only by increasing the number of elements.

Since the main properties of the RC are weakly dependent on the specific implementation of the machine, it is advisable to take the first step towards the implementation of recursive machines in the field of mini-computers built on the basis of microprocessors.

Distinctive features of mini-RCs, allowing them to successfully compete with other mini-machines, are:

- the ability to automatically increase the productivity and reliability of the machine by adding relatively cheap processors without any changes in the structure of the original machine, in the software system and programs compiled earlier;

- simplicity of building computer networks of any size from mini-RC;

- the ability to work with virtual memory;

- simplicity of programming and debugging programs on high-level autocode;

- the possibility of simultaneous servicing of a large number of external devices (including high-performance ones) and the simultaneous execution of a large number of programs.

In general, mini-RCs have the parameters of medium or even large computers at the prices and sizes of mini-machines.

To build a mini-RC, it is required to have two types of processors: switching and executive, each of which can also perform control functions. These processors can currently be built on the basis of bipolar microprocessor sections of the 555 series. It is advisable to make the processors 16-bit (but capable of operating with arbitrary formats of numbers and various ways of representing them) with internal memory from 4K to 64 K words.

The minimal RC is a uniprocessor one and differs from traditional minimachines only by the organization of internal software implemented at the microprocessor level. The resident part of the software system, including programs and microprograms for interpreting the basic elements of the language, the most common executive microprograms and input-output programs, is located in the internal permanent memory of the RC and partially in its random access memory. The rest of the software system, corresponding to a powerful virtual computer, resides in external memory. Due to the deep structuredness and modularity of all programs, the user has the opportunity to select from ready-made blocks such a virtual machine that exactly matches his tasks. For simple tasks, you can get high performance (4-6 million ops / s), working in compilation mode, while for complex tasks, due to a decrease in performance, you can get a service that is typical only for large modern computers. In this case, the decrease in performance can be compensated for by reducing the programming and debugging time of programs.

RC, consisting of 4 switching processors that also perform the functions of control and input-output, and four executive processors with a total memory of 256 kilobytes in performance more than 2 times exceeds the ES-1050 computer when solving purely computational problems and more than by an order of magnitude when solving non-computational problems.

Supermini-RC with an internal memory of 1 megabyte, distributed between 64 processors at a cost comparable to the cost of an ES-1060 computer, has a performance of only executive processors of more than 10 million ops/s of long operations such as multiplication of floating point numbers and more than 200 million short operations per second, such as 16-bit fixed-point addition. Moreover, executive processors are almost completely freed from control and I/O functions, which significantly increases effective performance.

The cost of an RC of any class can be significantly reduced by implementing its processors not on standard, but on specially designed LSIs.

The most effective areas of RC application are those for which structurally complex tasks with a large amount of control are characteristic, or those where it is required to simultaneously solve a large number of small tasks. These areas include various modeling tasks, control of complex production processes using robots, automated control systems, design automation, automation of scientific experiments, etc.

LITERATURE:

1. V.M.Gluskov, M.B.Ignatyev, V.A.Myasnikov and V.A. Torgashev. Recursive machines and computing technology. "Proc. IFIP Congress 74". Amsterdam-London, North Holland Publ. Comp., 1974, v.1, p.65-70.

2. Котов В.Е. Теория параллельного программирования. Прикладные аспекты. Киев, "Кибернетика", 1974, № 1, с.1-16; № 2, с.1-18.

3. Котов В.Е. О понятии типа управления в языке программирования. В кн. "Системное и теоретическое программирование". Новосибирск, 1974.

4. Дж.Деннис, Дж.Фоссин, Дж.Линдерман. Схемы потока данных. В кн. "Теория программирования". Новосибирск, 1972, с.7-44.

5. Берс А.А. Операторше структуры. В кн. "Теория программирования". Новосибирск, 1972, с.44-82.

6. Вопросы программного обеспечения ЦВМ с рекурсивной структурой. - авт.: Торгашев В.А., Андрианов В.И., Бердников Л.И., Смирнов В.Б. – В кн. "Вычислительные системы и среды" Материалы III Всесоюзной конференции "Однородные вычислительные системы и среды". Таганрог, 1972, с.453-455.

7. Управляющие операторы в РЦВМ. - авт.: Торгашев В.А., Андрианов В.И., Бердников Л.И., Канунников В.А. – В кн. "Вычислительные системы и среды" Материалы III Всесоюзной конференции "Однородные вычислительные системы и среды". Таганрог, 1972, с.455-456.

8. Торгашев В.А., Шейнин Ю.Е. Принципы организации внутреннего математического обеспечения рекурсивной ЭВМ. - В кн. "Однородные вычислительные системы и среды". Ч.1. Киев, "Наукова Думка", 1975, с.282-285.

9. Шейнин Ю.Е., Торгашев В.А., Смирнов В.Б. Принципы организации децентрализованного управления в микропроцессорной ЦВМ. – В кн. "Микропроцессоры". Рига, "Зинатне", 1975, с.128-130.

10. Торгашев В.А., Андрианов В.И., Бердников Л.И. Рекурсивно однородная структура. Авт. свид. № 371578, заявл. 19.05.1970. Бюлл. изобретений № 12, 1973.

11. Торгашев В.А., Андрианов В.И., Бердников Л.И. Регулярное вычислительное устройство. Авт. свид. № 342181, заявл. 16.11.1970. Бюлл. изобретений № 19, 1973.

12. C.J.Lipovski. A varistructured failsafe cellular

computer. "Proc. of the First Annual Simposium on Computer Architecture", University of Florida, 1975, pp.161-165.

13. Игнатьев М.Б., Торгашев В.А., Шкиртиль М.А. Архитектура сверхнадежных ЦВМ. – В кн. "VI симпозиум по проблеме избыточности в информационных средах". Л., 1974, с.46-49.

14. Торгашев В.А., Гобачев С.В., Сасковец В.Н. Обеспечение надежности СУВМ для АТС малой емкости. – В кн. "VI симпозиум по проблеме избыточности в информационных средах". Л., 1974, с.106-110.

15. Игнатьев М.Б. и др. Управление вычислительными процессами. Том 1. Изд. Ленинградского государственного университета, Л., 1973, 356 стр.