JPRS L/9607

13 March 1981

USSR Report

CYBERNETICS, COMPUTERS AND AUTOMATION TECHNOLOGY

(FOUO 9/81)

FBIS FOREIGN BROADCAST INFORMATION SERVICE

FOR OFFICIAL USE ONLY

APPROVED FOR RELEASE: 2007/02/08: CIA-RDP82-00850R000300090029-8

NOTE

JPRS publications contain information primarily from foreign newspapers, periodicals and books, but also from news agency transmissions and broadcasts. Materials from foreign-language sources are translated; those from English-language sources are transcribed or reprinted, with the original phrasing and other characteristics retained.

Headlines, editorial reports, and material enclosed in brackets [] are supplied by JPRS. Processing indicators such as [Text] or [Excerpt] in the first line of each item, or following the last line of a brief, indicate how the original information was processed. Where no processing indicator is given, the information was summarized or extracted.

Unfamiliar names rendered phonetically or transliterated are enclosed in parentheses. Words or names preceded by a question mark and enclosed in parentheses were not clear in the original but have been supplied as appropriate in context. Other unattributed parenthetical notes within the body of an item originate with the source. Times within items are as given by source.

The contents of this publication in no way represent the policies, views or attitudes of the U.S. Government.

FOR OFFICIAL USE ONLY

JPRS L/9607

13 March 1981

USSR REPORT

Cybernetics, Computers and Automation Technology

(FOUO 9/81)

Contents

 [III - USSR - 21C S&T FOUO]

APPROVED FOR RELEASE: 2007/02/08: CIA-RDP82-00850R000300090029-8

HARDWARE

UDC 681.3.01:681.3.06

THE EL'BRUS SYSTEM

Moscow PROGRAMMIROVANIYE in Russian No 6, 1980 pp 72-86

[Article by B.A. Babayan and Yu.Kh. Sakhin]

[Text] The article is devoted to a description of the architecture and software of a new high performance family of multiprocessor computer systems, the El'brus-1 (E-1) and El'brus-2 (E-2) MVK developed at the Institute of Precision Mechanics and Computer Technology imeni S.A. Lebedev.

The general principles of the El'brus system are outlined in the article and the various components of the system are described; in addition the new features and versions of implementation are outlined in more detail.

The El'brus system is a multiprocessor modular system with a high degree of dynamism, stack organization for calculations and a tag configuration and hardware implementation of the procedural mechanism.

I. General Concepts

When developing a computer system, one must take into account and balance the different characteristics (operating speed, cost, reliability, convenience of use and so on). To retain conceptual unity, it is primarily necessary to "prioritize" these characteristics in a specific manner. The main principle from which the remaining principles ensue must be determined for this purpose. The main principle in the El'brus design is the principle of programming simplicity.

As design practice has demonstrated, this principle harmonizes well with the remaining aspects of the problem and its consistent fulfillment leads to correct resolution of most of the main problems arising during development. Let us explain this.

1. One of the most important features of the system is its dynamism. Generations of previous machines and programming languages accustomed programmers to very "static" utilization of resources and data types, which made programming complicated. The machine program is also an algorithm whose structure has been adapted to the configuration of the machine and has been limited by it. In this sense programming consists of translating an abstract algorithm to a program. The more

dynamic the system is, the more precisely it can be adapted to the structure of the algorithm and the simpler programming is. The degree of dynamism should be sufficient so that there are no indications to physical resources (for example, to arithmetic registers) in the machine languages and programs, but so that there are algorithmic concepts (for example, local data procedures). The use of tags [6] provides realization of the dynamism of data types. Good dynamic properties make the system adaptable to terminal work.

2. Another important principle which facilitates programming is the modular nature. Modularity is a method of controlling the information complexity of programming and it permits one to conceal technical details from the programmer, requiring only a knowledge of the interface from him. Modules should meet the following requirements:

a) modules should correspond to programming requirements. To do this, for example, the machine instruction set should support in hardware such concepts as procedure and data array;

b) the functions realized by the module should be sufficiently complete. For example, the procedures should be recursive, which requires implementation of stack discipline for calculations. Parameters should be transferred without limitation, arrays should be dynamic and so on.

c) the modules should be independent of resources;

d) modules should be independent of errors. Context security using tags has been realized in practice for the first time in the El'brus.

e) the effectiveness of programs based on the modular principle should be high, which is achieved by powerful hardware support.

3. The most important characteristic is effectiveness. There is the opinion that the effectiveness and convenience of programming are mutually contradictory. Development of the El'brus convinced us of the opposite. For the case of high performance machines a system with maximum convenient programming permits realization of high productivity. Actually, many algorithmic concepts are naturally embodied in the hardware (such as procedure, loop, array, expressions stack and so on) for ease of programming, which makes apparatus optimization of calculations possible.

4. Programming is simplified considerably when oriented to high-level languages. Other programming is not used in the El'brus system.

We note in conclusion that the means available to the user are usually employed for the needs of systems programming as well:

a) systems programming is accomplished in high-level languages;

b) the user has available those means of parallel programming, by use of which the operating system of the El'brus is written;

c) the user and systems programmers have available the same memory distribution equipment (dynamic arrays and so on);

d) the user and systems programmers employ the same El'brus file system mechanisms.

With this approach, systems programmers find themselves under as comfortable conditions as users.

2. Memory

A graphical depiction of the relationship of various schemes for organization of virtual memory is shown in Figure 1.

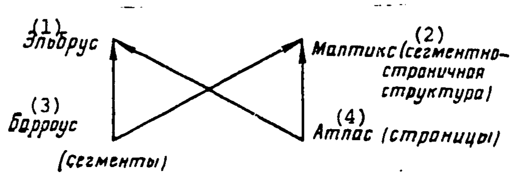


Figure 1

Key:

1. El'brus
2. Multics (segment-page structure)
3. Burroughs (segments)
4. Atlas (pages)

Unlike the Atlas [3] and Multics [4, 1] where the logical and physical memory are divided in pages of constant length, the memory in B6700 and B7700 machines of the Burroughs Company is distributed by segments of varying length according to the natural size of the file being ordered.

As in the Atlas and Multics, there is a logical memory in the El'brus. A page of logical memory contains 512 words. A word contains 64 information and 8 service digits (tags, parity). However, whereas the segments are of constant length in the Multics and consist of a fixed number of pages, there is no concept of segment in the logical memory in the El'brus. The segments exist only in the physical memory. If the requested array is less than a page, the system allocates the next logical page completely to the given user. In this case, exactly as much physical memory is allocated as the user requested. If the user requested an array larger than a page, the system issues the entire number of sequential logical pages to him. In the physical memory as many words as necessary for placement of the user array are allocated. Thus, the scheme presented above illustrates the fact that the logical memory in both the El'brus and Atlas is divided into rigid pages, but the physical memory is distributed in segments of arbitrary length according to the actual size of the array. Allocation of the physical memory according to the requirements of the algorithm is convenient for the programmer, leads to efficient use of the memory and reduces the size of transfer arrays. As is known, the average size of an array declared by a programmer equals to 50-100 words [5]. An attempt to arrange this array, for example, in pages of 256 words each (2 kilobytes), like IBM, leads to poor utilization of memory. An attempt to reduce the size of the page leads to an increase of the number of data transfers and to an increase of tables.

The program segments in the El'brus are also of arbitrary length (with accuracy to a word). However, logical memory is not allocated to them and access occurs through physical addresses. Taking into account that all programs in the El'brus are re-entrant, this permits one to use a single copy of the program from different jobs. This organization does not lead to scanning (review) since all the physical addresses (segment bases) are collected at one point (in the segment dictionary).

Special hardware operations for working with lists are provided in the system to distribute the physical memory by segments of arbitrary length. Each occupied segment is framed by two service words by use of which they are intertwined into lists. Each list contains segments, the numbers of the logical pages of which are comparable module 2^k (see Figure 2).

There is associative memory of pages (AZUS-32 registers) in the processor, which is used to recode the logical address to a physical address.

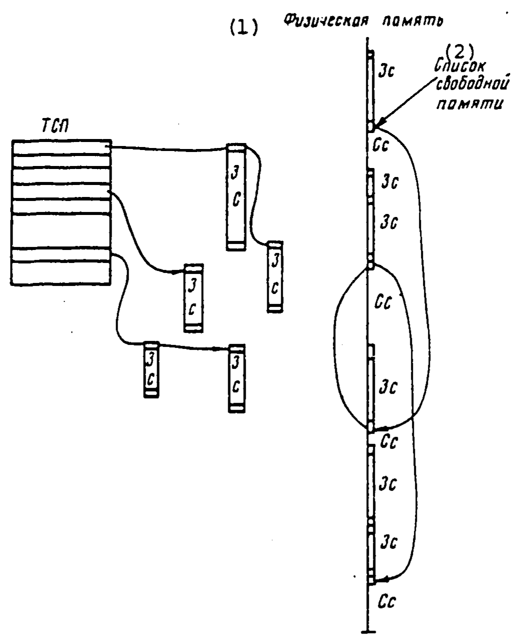


Figure 2. TSP--user page table; ZS--occupied segment; SS--free segment

Key:

1. Physical memory
2. List of free memory

If the required logical address is not present in the AZUS, the apparatus provides an automatic search by list and carrying of the found address to the AZUS. The free memory is also organized into lists. The advantages of this organization of memory compared to Burroughs [2] are related to the absence of physical addresses in descriptors, which frees one of the necessity of scanning upon return to the physical memory system. This is manifested in the following:

a) there is no overhead for scanning in the mechanism of the virtual memory when data are sent to secondary memory since the presence of logical memory permits only table correction during evacuations;

b) descriptors may specifically describe subarrays or part of an array initially declared by the user; this is impossible in the Burroughs since this would lead to a significant complication of scanning;

c) if there are mathematical pages, no problem of array placement arises since the arrays are automatically divided into pages;

d) the Burroughs scheme does not permit one to locate the descriptors in the virtual evacuated segments. All the descriptors should be in the memory for the case of scanning;

e) the memory is freed of local arrays upon completion of the procedure. When the descriptors of these arrays are sent to global regions in the Burroughs they will indicate a freed memory which can be allocated to another job. This may lead to disruption of protection. It is inefficient to scan the regions of global data upon return from procedures due to concepts of efficiency. The presence of a logical memory in the El'brus removes these problems since the freed memory is not used again;

f) the presence of physical addresses and scanning the Burroughs leads to complex problems upon dumping and repeated call-up of the problem (SWAP) in the case of time-sharing. Actually, if the job is activated after dumping, the memory segments are located in another physical zone which requires rather complex correction of all descriptors. The scheme adopted in the Burroughs for controlling this complexity provides for allocation of the entire memory of the interactive job contiguously to simplify correction. As a result, interactive and batch jobs are statistically separated in the memory, which in turn leads to the undesirability of combining interactive and batch modes in time. These difficulties are absent due to the presence of a logical memory in the El'brus. The possibility of indicating the chain of memory regions in a single exchange operation contributes to more rapid dumping of the job.

The El'brus system provides operation with local and dynamic arrays. The size of the dynamic array can vary during problem-solving. The memory occupied by the local array is freed automatically by the operating system upon completion of the procedure. Organization of dynamic arrays in the described scheme is simple and requires only a single surplus word of memory per array as overhead. The presence of dynamic array generation during problem-solving ensures automatism of memory resets.

As already noted, the most important property of the entire system and of the memory system specifically consists in organization of data security. Context security based on tag configuration is realized in the El'brus. One of the most important designations of tags is dynamic detection of nonconformity between an operation and the type of data at program execution time. This control specifically permits one to strictly observe semantically correct actions with address information and to detect any errors in the user programs which violate this regularity. For example, it is impossible to add a number to the address, but a descriptor can be indexed, obtaining a reference to the file element described by the descriptor. With this type of indexation, an apparatus check is made whether the index of the file size described by the descriptor is exceeded. Other operations with descriptors are provided which correspond to semantically correct actions with the file, for example, "take the subfile," "write by address" and so on. A descriptor cannot be arbitrarily created by the user. It is issued to the user together with issuance of the corresponding new file by the system. All this is dynamically monitored by the apparatus due to the fact that it "understands" types of data by using tags. There is no address in the machine instructions. Instead, indexes are used by means of which the descriptors are indexed. The initial descriptors used to form the context (address environment) of the current procedure are found in the base registers. Their contents are formed by apparatus through input and output instructions from the procedure. Besides regions described directly by base registers, the context of current procedure may also contain files whose descriptors are located in these regions and so on. Consequently, regardless of what the code of a given procedure is, it is incapable of counting or changing something beyond the bounds of its own context without starting other procedures.

Thus, all procedures are protected from each other in the El'brus system if of course they do not share some regions of context with each other at the desire of the program authors. There are a special type of data--procedure markers with their own tag--for changing to a different context. Having some marker in the context, one can omit this procedure, thus converting to a different context. However, one cannot return to the context of this procedure whose marker is in the current context while working in the old procedure. This protection is reliable. Specifically, if the allocated procedure is omitted from the allocated program, this procedure can either be cycled or can be returned to the point of call-up. It becomes possible in this case to analyze the results.

All errors in the procedure work are formulated in terms of the interface of a given procedure. It has no side effects of any kind that cannot be checked. Since protection is reliably provided automatically, the user is actually freed of the concerns related to protection. It should be noted that there is essentially no need for a privileged instructions mode in this system. It is introduced in the El'brus only to enhance the efficiency of some operations.

3. Processes and Problems

A logical memory is allocated to each problem in the El'brus system. The problem has a number of attributes which are worked out during development and can be simplified or changed during its existence.

Prior to execution, batch jobs pass through the planning mechanism of a multi-program mixture. Each problem can have parallel processes provided by the user within it. No restrictions are placed on the number of these processes except general restrictions by resources. A dynamic stack corresponds to each process on a one-to-one basis. Creation and starting of a parallel process are almost equivalent in expenditures to simple announcement of a file in the user program. There is general priority of processes from different jobs ready for execution in this system. This order is regulated by priorities. Each processor of a multiprocessor system takes on execution of stacks from this order after being released from routine work.

Signals are selected as the mechanism for synchronization of processes. The authors do not feel that this mechanism is convenient. However, the schemes for synchronization proposed at present are less satisfactory. For example, let us note the monitor scheme most widely distributed in the theoretical literature. This scheme requires compulsory fulfillment of rather complex programs compared to the El'brus realization of critical signal sections. In operational systems one must rather frequently cover two or more signals simultaneously, which leads to unnatural complications in the monitor schemes.

The effectiveness of parallel processes is of primary significance in synchronization schemes. Actually, no user begins to use parallel processes without a special need. Only concepts for increasing efficiency usually force him to do this. Therefore, it is illogical to talk about low-effective synchronization schemes.

The first most important "user" of parallel processes is the operating system itself of a multiprocessor machine. Highly effective synchronization is especially important here. The developers of the Multics conducted an experiment, reducing or enlarging the critical sections in the operating system, and concluded that enlarged sections are more economical. This is the result of high overhead for input and output from critical sections. High overhead for synchronization is in itself poor and reduces the productivity of a multiprocessor machine. Moreover, enlargement of critical sections occurs as a result and this in turn leads to the fact that processors begin to interfere to a large degree with each other due to closed signals for a prolonged period. The mechanism of synchronization is completely realized in the apparatus in the El'brus due to tag configuration and therefore is very efficient. This realization is somewhat different in the E-1 and E-2. Let us ascribe the E-2 scheme as the most perfected one. It should be noted that the described scheme was first developed and realized in the El'brus system.

The signal in the El'brus system is a word with a special tag which protects the contents against improper use and modification. The signal is located in the user data. The operation of opening and closing signals is a machine operation. Each of them is fulfilled during two accesses to the memory: reading the signal with simultaneous setting per unit of service bit (if this bit were in the unit prior to this, the processor waits until it is cancelled) and modification of the signal contents in the processor and inverse writing to the memory with cancellation of the service bit. There is a signal bit in addition to the service bit. If this bit is zero prior to the operation "close signal," the processor writes a one in it the operation is completed with this and the next instruction--the first instruction of the critical section--begins to be carried out. If the signal bit is one (the signal is closed), a special interruption occurs in which the system establishes the process to be carried out by program in the queue to the given signal (the beginning of the queue begins in the signal itself), making use of the fact that the service bit is closed; the service bit then opens up. In the typical case when the open signal is closed prior to this, the operation is highly efficient. In like fashion, the operation to open the signal reads the signal contents (with interlocking of the service bit), checks the presence of a queue and in the typical case of its absence opens the signal with removal of service interlocking. The operation is completed with this. If there is a queue, a special interruption occurs and the queue to the signal is processed in the procedures.

Operations of closing of write and read signals are introduced in the El'brus. Several processes may close a single read signal and at the same time operate in critical sections. And only closing of a write signal blocks the operation of processes of awaiting the opening of a given signal. This measure is also called upon to reduce the interference of processors in a multiprocessor system.

Correct work with general data plays an important role in organization of parallel calculations in a high-speed multiprocessor machine. The main problem here includes the following. Rapid calculations at modern speeds require organization of local super RAMs (SOZU) for each processor. If some general data are stored in the local memory prior to the next input to the critical section on the same processor, an attempt to read them in the given critical section will provide the local variant of these data, whereas they can be announced in the memory by another processor between these critical sections.

Systems designed without regard to multiprocessor function and which utilize local memories (for example, IBM) are forced to review the SOZU of all processors with any notation from any processor or channel to the general memory and to destory local copies of general data. This is one of the most important factors of why multimachine configurations are more popular in these systems and multiprocessor configurations sharply inhibit work when the number of processors is increased.

A new solution of the indicated problem is realized in the El'brus. For simplicity let us describe the solution of this problem in the E-1 complex. The complex contains two special registers for time-code storage. Moreover, each cell of the SOZU has a bit which indicates what information in this cell is related at a given moment from the time registers. The corresponding bit is set at one upon access to any cell of the SOZU during writing or reading. If it turns out at some moment that the bits are in the position of the one the current time is entered in the time register corresponding to the unit bit and the old contents are rewritten to another register. All the bits are set to zero. This system permits rought determination of when the last access to data in a specific cell of the SOZU occurred. Moreover, the time of opening is stored in all open signals. If some processor now attempts to close the signal, the times in the two service registers are then compared to the time the given signal was opened. In the typical case when opening of this signal was earlier than access to the data in the processor (the signal time is less than the time in the register, corresponding to zero bit), nothing is done with the information in the SOZU and the processor enters the critical section. If access to some data occurred prior to opening of this signal, the cells occupied by them are freed. An attempt to read them in the critical section leads to reading from the general memory.

The complex of proposed measures leads to a system in which the processors essentially do not interfere with each other.

4. The File System

Each file in the El'brus system contains data and file attributes. There are drum, disc, magnetic tape, punch card, ATsPU [Alphanumeric printer], punch tape, alphanumeric display and typewriter files. Graphical files will be subsequently realized. The main operations on files are generation, destruction, opening and closing of a file, reading and modification of attributes and various types of exchanges. Direct and butter exchanges are provided. A direct exchange occurs between file data and user files. Direct exchanges can be synchronous and asynchronous.

With buffer exchange, data transmission occurs between the file and one of the systems files (buffer). A descriptor of the operating buffer is issued to the user after the exchange. Thus, the user can work directly on systems buffers.

There are two types of buffer exchanges--single-buffer and multibuffer. A single-buffer exchange is an ordinary sequential buffered exchange. A multibuffer exchange is used only for drum and disc files and permits the user to receive descriptors for several buffers simultaneously containing different regions of the given file.

System responses which are completed by a "situation" output in some "emergency" cases (see below), are provided during exchange with different errors. The user can prescribe his own error-processing algorithm.

A significant typical feature of the file system is its dynamic nature. Unlike many other systems, files can be generated within any procedure and not only at the level of the problem-control language. During operation, any drum or disc file can be lengthened or shortened arbitrarily. A dynamic distribution system of the physical drum and disc memory is realized to maintain these properties. With this dynamic nature of resource distribution (memory and devices), something must be said about the approach to the problem of critical situations (dead lock) in the El'brus. It could be solved by requiring instructions from the programmer about all the necessary resources or the work of a specific program segment. Another well-known solution is realization of the "banker algorithm." Both these solutions violate modularity since one must know the necessary resources to it and the resources of all programs caused by it to start the problem.

A different solution, more convenient to the user and which does not contradict modularity, has been adopted in the El'brus. Any problem solved in the system can be "dumped" to slow devices at any moment, thus freeing all the resources required for operation of it and it can then be continued at a later time.

Another important property of the El'brus file system is introduction of address information or of interfile references to the files. This approach is logically necessary in the El'brus system since it permits one to apply the context security described in the memory section to the file system. The information stored in the files is more permanent in nature, i.e., it has no specific lifetime (as, for example, do files which exist for the longest time until the end of the problem). Moreover, a very large volume of information stored in the files must be considered. This nature of files requires in some cases the possibility of "review" of all addresses which link files to each other. The following solution has been adopted in the El'brus in this regard: all references are concentrated in the file headings which are collected into a single general file unified for all drums and for each set of discs. Any heading contains a special key consisting of interfile references, each of which is the relative address of the file heading in the headings file. Any object in the file system can be reached only through the references. Since the entire headings file is directly inaccessible to the user, the system changes references only through the user's requirement. Context security in the file system is realized in this case.

A number of complexities which arise when working with address information and methods of solving them in the El'brus must be noted here. A counter of references for the given file is introduced in each file to facilitate the problem of file destruction. If the counter is equal to zero, the file can be destroyed at any time. All the objects to which the given file refers are reviewed upon destruction of a file and if the given reference was the only one (the reference counter is equal to one), this object is also destroyed. This process is carried out recursively.

If the user wants to destroy a file, the reference counter of which is not equal to zero, the given file is destroyed together with the heading (with possible recursive destruction described above), but a "keeper" measuring one drum segment (32 words) in which the reference counter is stored remains in place of the heading. This reference is subsequently destroyed upon an attempt at access to the given file through any reference and the counter is reduced to one. This finally leads to elimination of the "keeper." This technique of "keepers" is used when it is necessary to transfer headings to another point within a headings file in the case of expansion of the file and in this case the "keeper" contains the address of the new location of the heading.

The described work with address information may lead to the appearance of "garbage" in the system, although the system of recursive destruction carried out above and a number of other measures significantly reduce the number of these cases. A background "garbage collector," which operates without stopping the solution of the problem beforehand has been developed to clean up this "garbage." The following functions are entrusted to this "garbage collector": destruction of the "garbage," elimination of "keepers," determination of the "oldest" files and candidates for pushing to slower devices, destruction of all user files excluded from the system and destruction of references earlier created by a different user to the "host" files (if desired). This "garbage collector" is rather economical and its work is related to review of one or two files.

Interfile references and context security, so far as is known to the authors, were developed and implemented for the first time in the file system of the El'brus.

The mechanism of references described above could be sufficient to create a context of already translated programs if one could get along without identifiers, but identifiers are required for communication with man and for creation of the correct information environment for him.

The information mechanism has been introduced in the system for working with identifiers. Technically this is a hierarchical index table with variable number of levels, with blocks of constant length and with using the technique of table splitting. All information references on the drum are concentrated in a single systems file. In like fashion, there is an individual information file for each set of disc packs. From the external viewpoint, the information manual contains pairs: identifier-interfile reference to the file or to another information manual. Thus, directories are also included in the reference system and utilize standard techniques ("keepers," "garbage collector" and so on). The structure of the relationship between directories is not at all rigid in nature (for example, tree-like similar to the Multics [1] and to other systems [2]). Directories are generated and linked to each other and to the files by rather arbitrary system and user programs.

Using the technique of directories, each program file--an objective code file (FOK)--can be referred to other files (program or data files) not only directly by the reference in its heading but by using a so-called external name or identifier. To do this, the possibility of linking each FOK through a switch to a specific directory which creates the context of the external names of a given FOK is provided. If it is now necessary to have access to the file indicated by using an external name when completing a given program, the system will search for it in the file according to the found reference. This access does not violate context security. Thus, directories can again be referred to directories and the external name can be polysyllabic. However, single-syllable names are most commonly used in the El'brus system since the search is carried out "from below" from the local directory rather than from the root.

It should be noted that only the reference to the object is stored in the directory and in the file switch and all characteristics of the object are stored in the directory itself. However, since there can be as many references as desired from different directories or file switches as desired, some information related to specific access to this object through a given input is stored in the directory. For example, this information is writing restriction. The same object of a single user can simply be an ordinary file which permits modification of data and attributes, lengthening, shortening and destruction. This file may look like a permanent file (constant) in the context of a different user. This is achieved by the information mentioned in the reference. Another example is the right only to execute the FOK. This permits the program author to transfer it to the context of other users, only authorizing execution of it, having prohibited even readout of the FOK contents.

Moreover, this information is used for the following purposes. If the element of a directory with specific identifier is referred to a different directory, this, from the user's viewpoint, may create a hierarchy of directories, access to which is possible by a polysyllabic external name. However, if the feature of "indirect" context is contained with the reference from directory to directory, this means that both directories seemingly create a unified context of identifiers. Thus, if the search for some identifier in the first directory is unsuccessful, indirect contexts begin to be reviewed in alphabetical order (this process can be continued recursively).

Of course, any directory can be an indirect context for several other directories simultaneously. Such a case is a global directory of a system which is either contained directly or through a series of indirect contexts in the contexts of all users. This mechanism permits one to get along in a large number of cases with a single-syllable name and also abstract and make independent the program from a specific structure of directories.

The developed mechanisms and the structure of the archive described above permit one to find a highly efficient method of starting one independently compiled program from another. This is done in most cases by interfile references through switches without resorting to a search by identifiers. This search is reduced to the maximum since there is no search "from the root," as in most other systems. Moreover, introduction of interfile references and the described method of program context formation permit one to solve correctly the problem "rules of search" existing in all systems (see, for example, Multics) [4].

Besides the two mentioned system files (the headings file and the directory file), there are two other systems files: the operations file and the user file. The documentation of problems in different stage of execution (in input queues, in the process of fulfillment, in output queues and in the process of printing out the results) is stored in the operations file. Each problem is referred to several files from this file by using interfile references: the assignment file (if the problem is a packet type) is an FOK translated from an assignment packet and input and output files (files first read from input devices and files in which the output is stored). The user file contains extensive information about each user.

A directory of users which is found in the context of an operating system exists in the directory system. The input identifiers of this directory are the names of the users and the reference to entering a given user in the user file is put into agreement to each such name. The possibility of determining the structure of documentation in the user file within broad limits is provided in the El'brus system for administration of the computer center. It typically contains the following information: passwords, checks of access to the system, budget information, reference to the directory of a given user and the file context which determines it. This context is found as a context reference for external names for a packet FOK problem of a translated task of a given user. This context is used for dialogue work to interpret external names coming from the terminal. A possible pattern of connections and references in the El'brus file system is presented in Figure 3.

Let us analyze the process of establishing a new user and entering it into the system. Upon establishment of a new user, a new input to the user directory with name of a given user is created by the operator and a reference to the newly established documentation of the given user is entered in it. A directory is created which will serve as the context of this user. A reference from the user documentation (reference to context) is established for it. One or several global directories are entered as an indirect context in this directory.

The user's authority is checked upon input of it into the system and if this is a batch job it is then translated, user context is assigned to it, the problem documentation is created in which the reference to the newly created assignment FOK is written and finally the problem documentation is entered in one of the preliminary planning queues.

The same checks are carried out with interactive input into the system and the user context found in the documentation, as noted above, is used to interpret the external names assigned by the user from the terminal. Restoration of system files is provided for the case of breakdowns and failures of apparatus and also errors in system programs.

An input-output system having an exchange speed up to one billion bits/s in maximum configuration and which takes the load off the operating system to a significant

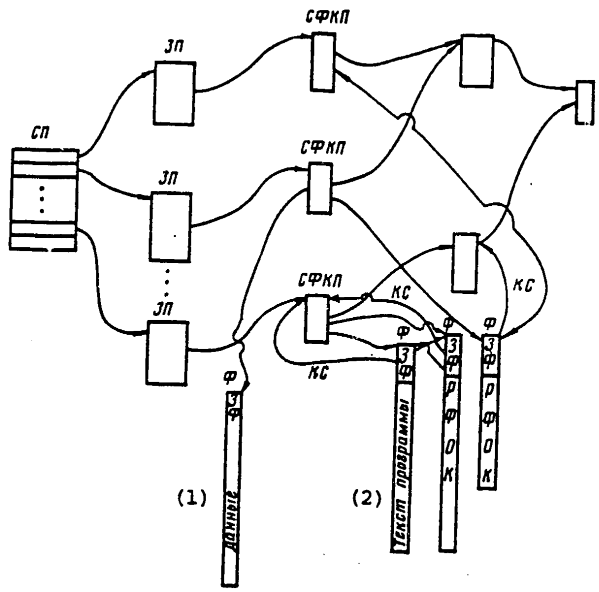


Figure 3. SP--user directory; ZP--user notation; SFKP--file context directory; ZF--file heading; KS--context reference; RFOK--expanded file of object code

Key:
1. Data
2. Program text

degree from working with the exchange queues, with dynamics and search for different access routes to the same object have been developed to ensure operation of files and the mechanism of virtual memory in the El'brus.

5. Programming Language

Programming languages play a special role in the El'brus project. This is indicated by the main principle which is the basis of development--programming simplicity. This explains the high affinity of the configuration decisions of the El'brus to high-level programming languages.

Work on configuration was to a significant degree work on language. We dwell below on the most important features of the basic Autocode programming language of El'brus. Selection of the name of the language is explained by the fact that this language, not inferior to existing programming languages in level, is the "lowest" programming level in El'brus and programming in it is equivalent in efficiency to Assembler programming in other systems.

Turning to description of the language itself, it must be noted that its most important features (dynamism and modularity) are based on the corresponding features of the system as a whole. Autocode is the first highly effective language in computer practice that is dynamic by types.

Many dynamic languages (APL, Euler, Lisp, Gedanken, POP-2) were created in the past which are convenient from the programmer's viewpoint, but are hardly effective on existing machines. The presence of tag configuration permits one to make the dynamic language highly efficient. It was possible to write the operating system in Autocode only due to the fact that Autocode is a dynamic language. Actually, the procedures of the operating system are constantly concerned with statically nonspecific data users, the types and structure of which are determined dynamically during operation (by tags).

Autocode is a recursive language and this property is maintained by the stack configuration and nonaddress instruction system. As indicated by design practice, it is this approach, unlike machines with directly addressible registers, that permits one to achieve maximum efficiency of calculating expressions.

It is interesting to trace the history of this problem. Directly addressible registers were initially introduced into computer configuration to increase the speed of calculations and this was justified to a sufficient degree (if one is diverted from programming convenience) for machines of the early 1960s when the problem of equipment economy became acute. A classical example of this configuration is IBM. However, registers were only an encumbrance under conditions of extensive overlapping of operations in the IBM model 91, when many different functional AU appeared in a single processor since information was transferred directly from one arithmetic device to another and the numbers of the registers were actually for identification of information and their current position was associative. Under these conditions the presence of the register number in the instruction set only led to superfluous conflicts which slow down the work.

Registers for calculation of expressions (16 for the E-1 and 32 for the E-2)--the apparatus apex of the stack (the number is not limited by any digit grid and is selected only by concepts of efficiency)--are provided in the El'brus. But these registers are not addressed directly from the program. During instruction decoding, the processor translates a nonaddress system to a three-address register system, dynamically designating the numbers of the registers. This approach is yet another example of matching programming convenience and efficiency of calculations.

A significant improvement in the El'brus and consequently in the Autocode compared to traditional computers is related to the FOR loops. Registers of control variables which service several embedded loops are automatically introduced for them. They are initialized by a special operation "beginning of loop" and a one is added to or subtracted from them by a corresponding operation "end of loop". There are packed files which are described by a certificate consisting of an ordinary descriptor and the aggregate of measuring steps arranged adjacent to it. Each step corresponds to an increment of the packed file index upon variation of the corresponding measurement per unit of index. This corresponds in accuracy to the technique used in the translator. The operations "take the file element" and "take the address of file element" can be used within the loop with instructions of which control variables index each dimension of the file. Upon first fulfillment of this instruction or after interruption of the work of a loop (for example, starting a procedure within a loop), the index of a packed file is calculated by multiplying the control variables by the corresponding steps of the packed file with subsequent addition. This index is then corrected by addition alone instead of multiplication during subsequent repetitions of this loop by using special associative registers in which the current significance of the index for the packed file is stored.

This technique permits one to summon the values of the file elements for several of the following loops to the special associative memory of the files since the law of index variation is well known to the apparatus. The described mechanism may serve as yet another illustration of the fact that programming efficiency and convenience assist each other.

An important innovation in Autocode, easily maintained by the apparatus, is the so-called structural transition which permits one to leave a loop of dynamically embedded phrases up to a point noted by the programmer in a special manner. The response which actually "replaces" the omitted phrases is provided in the point marker. This response can be parametrized from the point the structural transition occurs. The structural transition simplifies in a significant manner "programming without a transition operator." It is also extremely important to transfer processing of emergency situations from systems procedures to the user (for example, the situation "end of file"). The iteration loop is extremely simplified due to the presence of structural transition--this is simply an infinite loop with emergency exit in the Autocode.

The possibility of introducing an "arbitrary" syntax for procedure start-up (the restriction of arbitrariness possibly has a double meaning) is provided in Autocode for the case of starting up complex programs (for example, a complex variant of translator start-up) when a simple procedural syntax becomes inconvenient due to the large number and variety of parameters. This mechanism may serve to start complex applied programs. Moreover, the syntax of instructions from terminals can be determined by using this mechanism.

Autocode is used as a batch job control language. The idea of using a high-level language for this purpose (the same in which the problems are programmed) is not new, but according to our data it has been realized for the first time in industrial development. Some facilities which are in reality language orthogonalization had to be introduced into Autocode for this. The most significant of them are literal files which are actually the cards of external files which are the basis for the batch job.

As a result, a unique property of Autocode, along with its high level, is its universality which is greater than that of assemblers in existing machines (it encompasses the job control language and terminal language). However, it must be said that it is rather voluminous, like any practical language.

It should be noted that modern language-independent dynamic diagnosis can be easily realized due to the tag configuration in the El'brus. Actually, a stack can be printed at any moment of fulfillment where one can easily understand the sequence of starting the procedures at the machine level and the significance of all the local data and parameters with accuracy to types.

A complexer which permits combining of the programs written in different languages is provided in the El'brus. An important feature of the complexer is that there is no need to use it after each translation. The programs are issued by the translator in a form ready for execution and, moreover, the programs do not require loading due to the instruction set. Programs can be complexed only if the errors in a program already operating are corrected.

6. The Instruction Set

The instruction set and the internal structure of the El'brus are designed for programming only in high-level languages. The main organization of the computer process is the stack. The names of procedures as well as the intermediate results of calculating expressions are located in the stack. The top of the stack is the fast registers in the processor. These registers are distributed dynamically during problem-solving by the apparatus under the intermediate results of calculations.

There is a special table--base registers, containing descriptors in the processor. These descriptors describe the stack regions accessible to a currently operating procedure, i.e., they determine its address context. Base registers are inaccessible to the notation user and their contents are changed automatically by operations of procedural transitions according to the context of the new procedure.

Names are addressed in the stack by using an address pair--the lexical-graphical level of the procedure which determines the corresponding base register and of the shift which is the index of the name within a given procedure. Access to the file elements is gained through the file descriptor usually located in the stack. The files themselves are located outside the stack. Thus, a descriptor address and one or several indexes (in the case of a multidimensional file) are used to calculate the address of the file element.

Most of the most frequently used instructions have a length of one or two bytes. Thus, for example, local variables of current procedure (the first 32 names) are read in E-2 by a single-byte instruction while a two-byte format is used in the remaining cases.

The arithmetic and logical operations have a single-byte format. The operands for the queueing operation are the upper elements of the register apex of the stack. Multibyte instructions are usually those containing a literal constant.

The type and format of data control the operation-fulfilling algorithm due to the presence of tags. For example, whole and real operands of identical or different formats can be fed in any combination oo the input of the same arithemtic operation and the apparatus itself makes the necessary transformations prior to the operation. If it turned out during reading of the operand that indirect references lead to it, the passage through the address chain is realized by apparatus.

There are two types of branch instruction: conditional and unconditional branch and calls to procedures. The address context is changed by apparatus when making procedural transitions and control is transferred to the new program segment.

The only type of address contained in the instruction code is the address pair, which is found as a result of program translation and is actually an encoded name. This name is converted to a memory address only at the moment the instruction is fulfilled and is determined by the content of the base registers. Consequently, the addresses in the program code are not dependent either on the position of the program itself in the memory or on the position of the stack. Therefore, all programs are re-enterable in the El'brus.

Compared to other computers, the apparatus realized in the El'brus made it possible:

a) to increase the productivity of the system as a whole and the efficiency of equipment utilization and consequently to reduce the cost of calculations;

b) significantly facilitate use of the system. There are no programs written in Assembler language in the El'brus system;

c) reduce the volume of software. The total volume of software, which includes all the necessary components inherent to the more developed systems (without a data base) and which includes a programming system in three languages--Autocode, Algol and Fortran--comprised an order of 200,000 lines of Autocode text;

d) increase software reliability.

BIBLIOGRAPHY

1. Organick, E. I., The Multics System: An Examination of Its Structure, Cambridge, MIT Press, 1972.

2. Organick, E. I., Computer System Organization: The B5700/B6700 Series, New York, Academic Press, 1973.

3. Kilburn et al, "One-Level Storage System," IRE TRANSACTIONS ON ELECTRONIC COMPUTERS, EC-11, No 2, April 1962.

4. Dennis, J. B., "Segmentation and the Design of Multiprogrammed Computers Systems," JACM, Vol 12, No 4, October 1965.

5. Batson, S.-M. In and D. C. Wood, "Measurements of Segment Size," Second Symposium on Operating Systems Principles, Princeton University, October 20-22, 1969.

6. Ailif, D. J., "Printsipy postroyeniya bazovoy mashiny" [Principles of Constructing a Baseline Machine], translated from English, Moscow, Mir, 1973.

[8144/0521-6521]

6521
CSO: 8144/0521

EL'BRUS-1 IN SERIAL PRODUCTION; EL'BRUS-2 READIED

Moscow SOVIET UNION in English No 11, Nov 80 pp 6-9

[Article by I. Nekhamkin: "El'brus Comes of Age"]

[Text] El'brus-1 multi-processor computer complexes have gone into serial production in the Soviet Union. Computing speed: from 1.5 to 15 million operations a second. Memory holds about 10,000 million bits of information. El'brus-2 is next in line.

The development of an integrated computer system in the USSR was described in an article in this magazine 2 years ago (issue No 337, 1978) Letters from readers ask for information about the latest Soviet developments. Interest in this field is natural because it is advancing so rapidly: while three generations of human beings succeed one another in a century, three generations of computers have arisen in less than 40 years. Now fourth-generation computers, to which El'brus-1 (named after Mt Elbrus in the Caucasus, Europe's highest peak) belongs, are in.

Speed, one of the most important specifications of computers, is increasing steadily. Structure, storage capacity, the technological level of the machine as a whole and of its separate components, its softwear, the language of the computer, and the number of interconnections with its own components and also with outside users, are equally important. This broad range of specifications held the attention of the creators of the El'brus-1 multi-processor computer complex.

Among the leading figures in the big team of scientists and engineers who developed El'brus-1 were pupils of Academician S. Lebedev, workers of the Institute of Precision Mechanics and Computer Technology of the USSR Academy of Sciences.

"Why is it called a complex instead of just a computer?" we asked the director of the institute, Vsevolod Burtsev, corresponding member of the USSR Academy of Sciences.

He invited us to follow him into a narrowish corridor formed by grey-blue metal cabinets roughly the size and shape of the dressers and wardrobes to be seen in an average home. Pausing in front of one of them, he said: "This is the brain of El'brus-1, its 'thinking' section, called a processor. Farther on are the memory units: magnetic tape (tape recorders, but of a special kind), magnetic discs (rolls resembling those used in early phonographs). At the side are input-output processors, through which the complex is linked up with its users. In short, this is a complex made up of modules outwardly resembling wardrobes. This, actually, is the whole complex."

We were surprised not to see the familiar console with flashing lights and hundreds of buttons and switches.

"They aren't needed," the director said. "During normal operation of the complex this room could be locked up. The 'several' thousand subscribers who use the complex--some of them in places dozens and hundreds of kilometers from here--don't suspect it functions without supervision. Nor do they imagine that while working for each of the subscribers the complex can simultaneously carry out calculations for all the others."

"You said the room could be locked up. But what if something goes wrong?"

"If a module breaks down the computer automatically switches it off and its functions are taken over by other modules. Simultaneously, after rearranging the flow of the processes, the machine reports the malfunction to the operator: it prints out the number of the faulty module and the cause of its breakdown. With that knowledge we can replace it. Modular design is one of the complex's chief advantages.

"Look at this," he continued, opening one of the cabinets. "This is the central processor. It has a computing speed of 1.5 million operations a second. A similar processor stands beside it. Together they perform 3 million operations a second. The complex is designed for as many as ten processors, with an overall computing speed of 15 million operations a second."

"Is its memory based on the modular principle too?"

"Yes it is. The memory can be built up by the addition of standard modules, from several tens of millions of bits of information to approximately 10,000 million."

When several years ago, British experts compared a computer with the human brain, they found that the human memory has a theoretical maximum capacity of 1,000 million bits of information in a lifetime, while the computer had a volume of 30 million. But in the past decade computers have made impressive progress: today their memory can hold as much information as that of dozens of sages.

The designers endowed the complex with the ability to communicate high-level languages. This has facilitated the work of the programmers and at the same time has made it a polyglot. It can communicate, if necessary, with computers that use other languages. There are plenty of subjects for conversations of that kind because the number of computers in our country is quickly growing. They are being successfully supplemented by the integrated computer system designed and built by countries of the socialist community. These machines have won high praise from experts in many countries but were met with dissatisfaction by some Western periodicals. Writing in COMPUTER WEEKLY, Rex Malik said that the Western hopes of colossal sales of computers to CMEA countries failed to come true. The Soviet Union stands for broad and mutually advantageous trade with the whole world. However, the lines above, written 7 years ago, are interesting to re-read today, when some people claim that East-West cooperation is a "one-way street."

El'brus-1 has gone into serial production," Vsevolod Burtsev said. "Its use simplifies and speeds up the designing of seagoing vessels and aircraft, complex management and information systems, and so on. In short, a computer of this class has many applications.

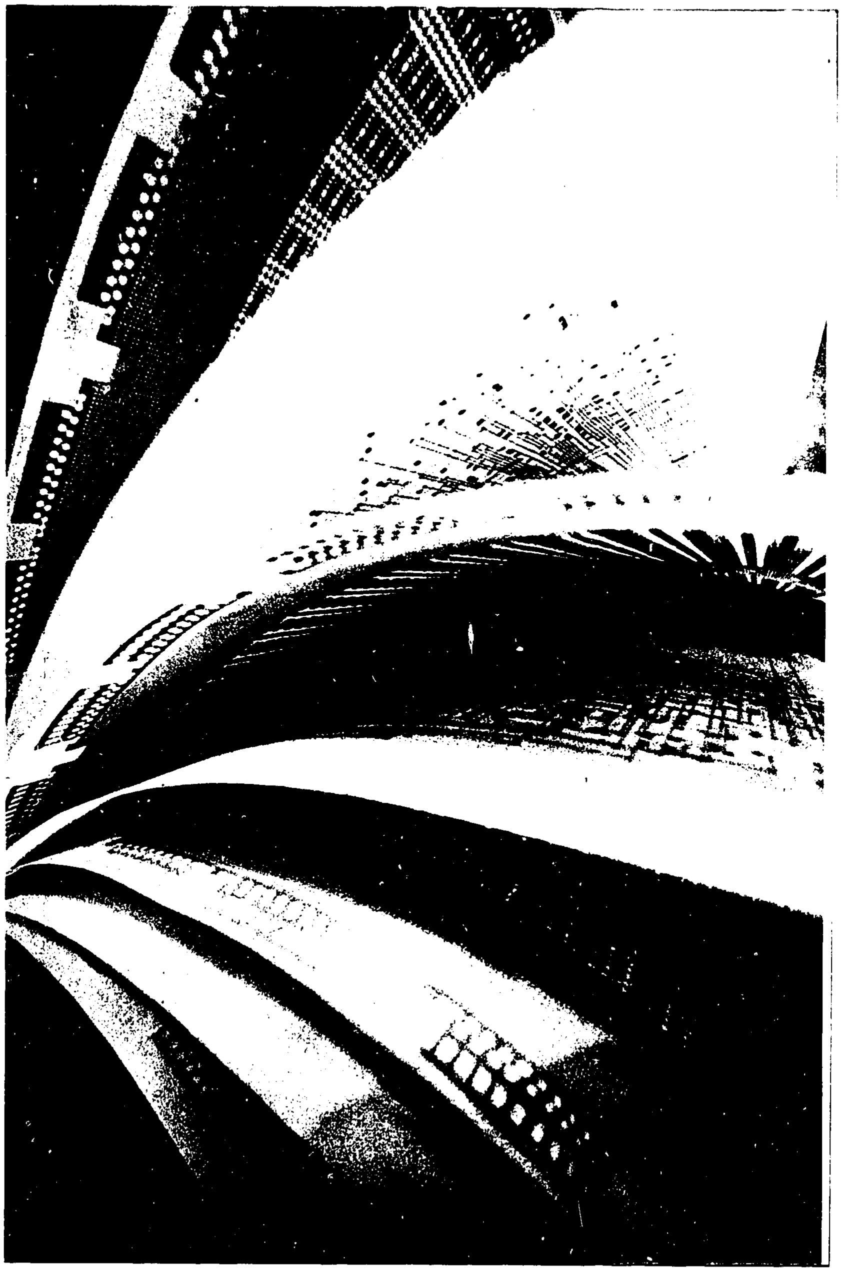
Pointing to another group of similar-looking grey-blue cabinets, he asked, "What do you think that is?"

"An El'brus?"

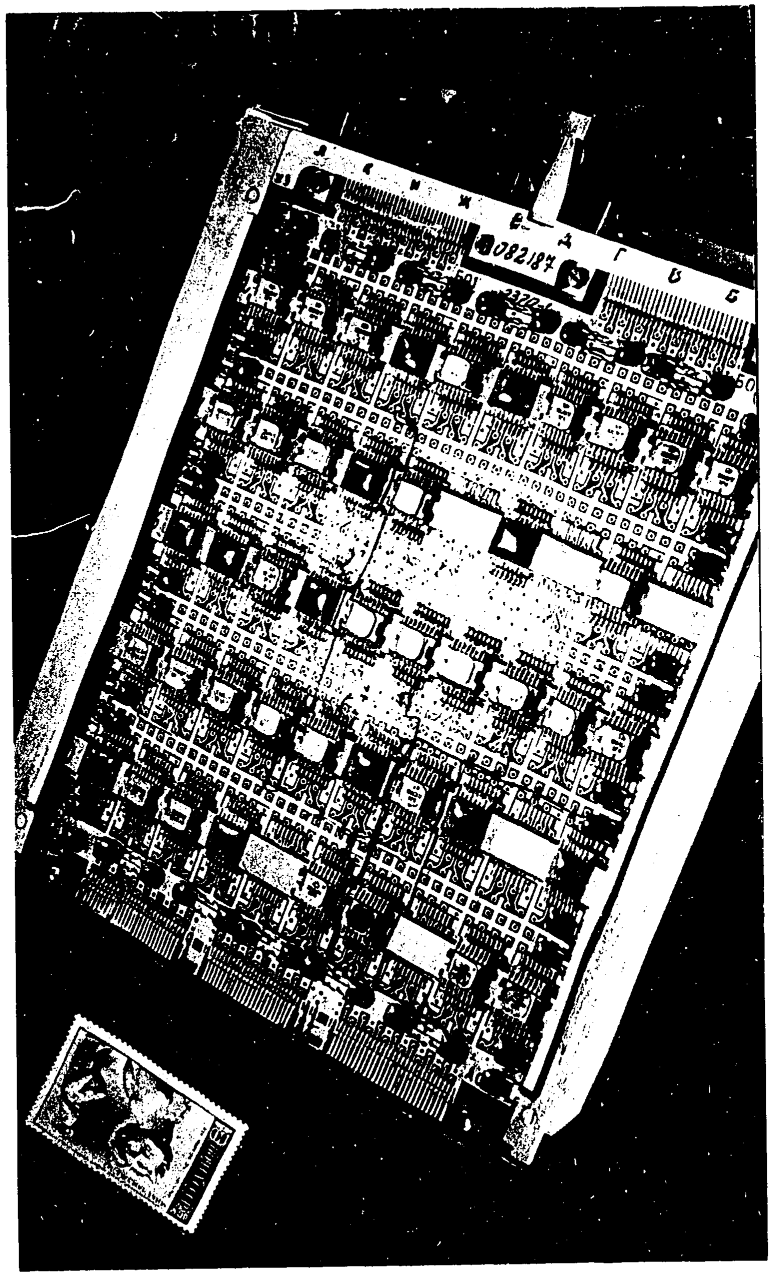
The doctor smiles. "Yes, but it's El'brus-2, the younger brother of the complex you just saw, and ten times more powerful. El'brus-2 is a multi-processor complex developed on the new design and technological basis of fourth-generation computers, having integrated microcircuits. In order to organize the serial manufacture of such powerful and super-powerful integrated circuits, precision multi-layer high-frequency ships and other components we had to master many new processes and start the mass-scale manufacture of sophisticated components. Also we had to draw up, with the help of computers, of course, an enormous amount of documentation."

The bulk of this work has now been done, and El'brus-2 is in the production stage. It will perform at the rate of 150 million operations a second. Each operation will take only between two and three ten-millionths of a second, a duration of time in which light travels a distance of only a few dozen meters.

The new complexes will enable Soviet specialists to develop still more powerful computers, needed in all spheres of life in all countries. The aim of improving and effectively applying electronic computers in all spheres of the Soviet economy, set by the 25th Party Congress in 1976, is being carried out with success.

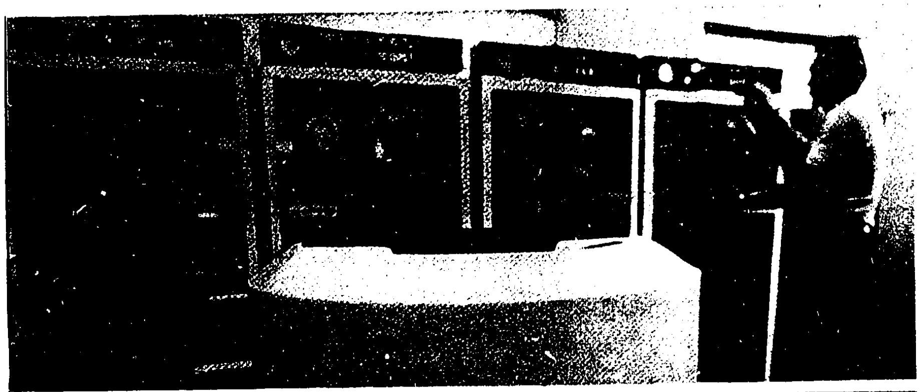


This multi-layered printed circuit board interconnects the standard replaceable elements of the El'brus-1 complex.

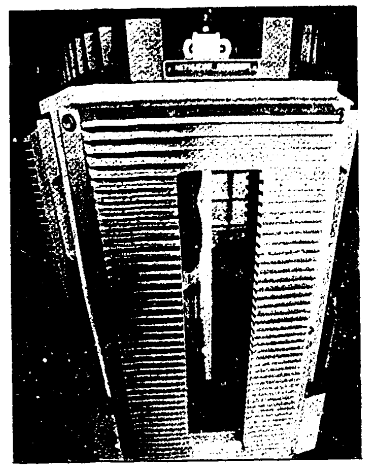


When an element (another type of which is seen in this picture) goes out of order it is replaced by a new one much in the same way as a tube in an old radio set was changed. The circuits are designed automatically by a computer

If a malfunction occurs this console indicates its nature



Part of the external storage of El'brus-1.



On the bottom we see one of its components. This cylinder with magnetic coating turns at a speed 6,000 rpm and can register 32 million bits of information, or about as many as are contained in four large volumes of 500 pages each. The computer scans all that in about 1 second.

[107-E]

CSO: 1863

END