PROCEEDINGS

OF THE

1980 INTERNATIONAL CONFERENCE

ON

PARALLEL PROCESSING

Cosponsored by the

Ohio State University
Columbus, Ohio

and the

IEEE Computer Society

IEEE Catalog No. 80CH1569–3

VASTOR: A Microprocessor Based Associative Vector Processor for Small Scale Applications*

W.M. Loucks, W.M. Snelgrove and S.G. Zaky

Dept. of Electrical Engineering,
University of Toronto

ABSTRACT

A word-parallel, bit-serial associative processor built around an array of 1-bit wide microprocessors is introduced. It is intended as a low-cost auxiliary processor in small scale computer systems. Data are organized in an array of fixed number of elements, variable word-length vectors. Processing proceeds in parallel on all elements of a vector. Information about the location and word-length of these vectors is stored in a small general-purpose computer which is used to control the storage and processing array.

I. INTRODUCTION

The parallel processing capabilities of an associative processor are highly attractive in many non-numeric applications. Operations such as searching and sorting are inherently parallel in nature, since they may be regarded as a sequence of basic operations such as compare, shift, and mark performed in parallel on a large number of operands. Many organizations have been proposed for associative processors [8, 10]. Of these, the word-parallel, bit-serial, or vertical [9], organization has received considerable attention. This is due to the fact that the bit-serial organization leads to a considerable simplification of the hardware in comparison with fully parallel schemes.

Because of the hardware intensive nature of associative processors, they tend to be economically viable only in large, high capital cost systems. The purpose of this paper is to introduce an associative processor that is meant for relatively small applications. It is based on an array of commercially available 1-bit wide microprocessors. Machine organization is word-parallel, bit-serial. Data is stored and processed in the form of vectors consisting of a fixed number of elements. The machine has been dubbed VASTOR for Vector Associative Store TO-Ronto.

VASTOR is intended as a special purpose processor to be attached to a conventional mini-computer system. In what follows, the minicomputer will be referred to as the host. In such a system, VASTOR would handle those parts of the work load that can benefit from its associative and vector capabilities. Use of associative processors in this manner has been suggested by many authors, e.g. [5]. Also many potential applications have been studied [3]. The main feature of VASTOR is that it represents an associative structure and its implementation that are economically viable in a minicomputer system environment. A prototype processor has been constructed and tested.

The main constraints in the design of VASTOR were low cost and modularity. This required that readily available components be used, that internal communication and control be kept simple, and that VASTOR should not overload the computer to which it is attached. Modularity also meant that backplane interconnections between modules should be kept simple and easily expandable.

The VASTOR processor, figure 1, consists of two main components, namely the processing array and the controller. The processing array contains all the storage and processing elements of VASTOR. The controller translates high level commands received from a scalar machine -the host- into sequences of control signals for the processing array. This paper presents a practical implementation of the array and its controller, and describes input/output transfers between the array and the host computer. Algorithms that may be implemented on vector oriented machines such as VASTOR are readily found in the literature [2, 3 and 7].

II. MACHINE STRUCTURE

The organization of the VASTOR array is illustrated in figures 2 and 3. The storage section in the array is an n-word memory, with a word length of several kilobits. Operations are performed on vectors of data elements, figure 2, when the elements of a given vector occupy the same bit positions in all words. While the number of bits per element is the same for all elements of a given vector, it may vary from one vector to another. A 1-bit wide processing element PE is a part of every word. Shift-register SH provides the main mechanism for data transfer

---

* This work was partially supported by the Natural Sciences and Engineering Research Council of Canada under research grant #A8994

among VASTOR words, as well as between the array and the outside world.

VASTOR's architecture, depicted in figures 2 and 3, has the properties both of an associative processor and of an array processor, in the sense in which those terms are defined in [10]. It is an SIMD machine, as are both of these types (note that opcode lines are shared by all cells in figure 2). Each cell contains a storage element which may be used to mark individual words. The I/O structure enables the host to read from and write to marked words in the memory. This allows VASTOR to be used as a content-addressable memory for the host machine. Each cell also has the ability to perform logical and arithmetic operations on its memory under the control of the mark bit, so that one may operate (in parallel) on all data elements satisfying some arbitrary condition. The above features give VASTOR the properties of an associative processor.

On the other hand, one may leave all words selected and use VASTOR as an array of processors. Its I/O structure allows large quantities of data to be transferred to and from the host machine via the parallel port on the right of figure 2. I/O data transfer rate ranges from 0.5 to 8 Mbit/s, as will be discussed in section V. Each cell C can perform data manipulation operations on one word of the memory M. From this point of view, VASTOR is an array processor. Inter-processor communication within the array enables handling of data organized in the form of a one-dimensional array, hence the word "vector" in the machine's name. Thus associative operations may be seen as a particular case of array processing, in which a preliminary computation is used to select data in certain cells for further processing or output to the host machine.

VASTOR operations are essentially word-parallel, bit-serial. The major differences between VASTOR and other serial machines, e.g. STARAN [10], stem from pragmatic considerations: component cost and backplane complexity. STARAN's memory is multi-dimensional: data may be accessed either by row (horizontally) or column (vertically) of a 256 row by 256 column memory array. These two modes of access involve a relatively complex interconnection network, which is referred to as a "flip network". Such a network is not required in VASTOR.

VASTOR uses 256 conventional 1024 by 1 bit random-access memories, all driven by the same address lines (cf. figure 2). Operations can be performed only on columns of memory. Because of this it is a "vertical" computer similar to that proposed by Shooman [9]. The I/O structure has been designed to compensate for the resulting difficulty in communicating with the "horizontal" host machine.

When the number of elements in a data vector is greater than the number of cells in a column of memory, operations can be carried out on "sub-vectors" of 256 elements each. This compromise exists in Shooman's machine also.

As mentioned earlier, development of the structure of VASTOR has been heavily influenced by interconnection considerations. The array has been designed to use only "daisy-chained" and "bused" connections between circuit boards. This allows new boards to be added at any time to increase the size of the array with minimal modifications to the existing backplane. The structure is also well suited to large-scale integration because of the small number of interconnections required between modules.

The main implication of the above restriction on backplane complexity is that it limits the inter-word and associative facilities that may be used. Hence, inter-word communication is accomplished via a shift-register, which involves a daisy-chain connection between circuit boards for both data and control information. Moreover, a single bused connection common to all words of the array combined with an analogue to digital converter (not shown) are used to provide limited accuracy associative testing.

The structure of VASTOR may be discussed in terms of three separate features: the intra-word storage and computation, the inter-word communication, and the associative testing capabilities. Each of these features is discussed briefly below.

2.1 INTRA-WORD FACILITIES

Figure 4 shows the components of a VASTOR word: two kinds of storage, a 1-bit processor and one bit of a shift register.

The random-access memory referred to in the figure as WK constitutes the 'working store'. Data are taken from this memory and returned to it during computation. A second memory, referred to as BK, for backing store, is a serial memory. Its contents are swapped with the contents of the working store in pages containing 256 bits per word. One more bit of storage is available for each word in its part of the shift-register SH. This may be used for temporary storage of operands. It should be noted that the intra-word facilities can be expanded through the use of the line marked 'B' on the figure.

The 1-bit processing element PE with which VASTOR has been implemented is the Industrial Control Unit - Motorola MC14500B. It performs a limited set of primitive operations on external data and a 1-bit internal accumulator called RR (the result register). Another internal register, output enable or OEN, contains a mask which is used to enable selective write-back into either the working or the backing store. The collection of the OEN registers in all words constitutes the output enable vector.

2.2 INTER-WORD COMMUNICATION

The shifter SH is the primary medium for inter-word communication. It is the only machine feature that defines any order to the words. The shift-register SH is divided into 8-bit segments as shown in figure 5. Each segment of SH has two parallel bidirectional ports A and B. The B port is connected to one "phrase"

of eight VASTOR words.The A ports of all segments are connected together to form an 8-bit I/O bus.

Two multiplexers CIRC and SHMODE connect the serial inputs of the segments of SH to any of a number of sources. This allows data transfer between the shifter and VASTOR words to take place in one of the following modes.

1. VASTOR to shifter - parallel mode through the B port: in this mode the source of data may be the processing element PE, the working store WK or the backing store BK.

2. VASTOR to shifter - serial mode through the SI port: in this mode up to eight bits of data may be loaded from any word of a phrase into the shifter segment. This operation takes place in parallel for all phrases.

3. Shifter to VASTOR - parallel mode: VASTOR words may be loaded in parallel from port B of the shifter SH via the processing element PE.

4. Shifter to VASTOR - serial mode: 8 bits of data can be moved serially from a shifter segment to any word in the corresponding phrase. This is accomplished via the combined use of the output enable vector OEN and the ability to circulate data within each of the 8-bit segments of SH.

We should note that in the two serial modes 2 and 4, only one word of each phrase is involved in data transfer. This reduces the parallelism in the array by a factor of eight. However, the serial modes are necessary to simplify byte-oriented data transfer between VASTOR and the host machine, as will be discussed in section V.

2.3 ASSOCIATIVE TESTS

All VASTOR operations may leave a result in register RR of the processing element. Contributions from all RR registers are summed, in an analogue fashion, onto a single line. This is a simple scheme to obtain a limited accuracy estimate of the number of responders S, i.e. the number of words with RR=1. The most useful values for this number are zero, one and more than one. A simple analogue to digital converter is used to extract this information from the analogue sum.

III. EXAMPLES OF VECTOR OPERATIONS

This section presents two examples of vector operations in order to illustrate the capabilities of the VASTOR array. In the first example vector addition is described. The second example deals with an associative search for the largest element of a vector.

Let A and B be two vectors that are resident in the VASTOR array, Figure 6a. It is required to obtain a third vector R which represents the arithmetic sum of A and B. Information regarding the two vectors A and B is stored in a table in the controller. The table stores the relevant parameters for each vector, e.g. starting address in the array, number of elements, number of bits, etc. The ADD operation is initiated by the host computer by sending a high level command specifying the function to be performed and the two operands A and B. It is not necessary for the host computer to specify such details as the addresses of the operands, the number of elements or the element lengths. Operands are identified by means of pointers into the operand table stored in the controller. When the operation is completed, the controller returns to the host the value of the pointer corresponding to the result vector R.

Addition is performed in a bit serial, word parallel manner. The sequence of operations is given in Figure 6b. As indicated in the figure, control of the sequence of operations and address calculations are performed in the controller, while vector operations are performed in the array. The optional masking operation at the beginning of the sequence disables those words of the array for which the mask contains "0"s. This may be needed when the vectors involved contain fewer elements than the number of VASTOR words. The mask used in such operations is set up at the time vectors A and B are created.

An implementation of the binary search algorithm [3] for positive or unsigned integers is given in Figure 6c. In this case the elements of the vector are scanned starting with the MSB. A one-bit wide vector TEMP masks out the words that have been rejected at any stage of the search. The associative sum S is used to determine the first bit position where one element of TEMP contains a "1" while all other elements contain "0"s. At the end of the search TEMP contains "1"(s) in the word(s) containing the largest element(s).

The above examples illustrate the operation of VASTOR on short vectors with all bits contiguous in fields. When there are more elements in a vector than words in the array, the vector may be broken into several subvectors. Each subvector is operated on independently. It is also possible that the elements of a vector may occupy two or more non-contiguous fields in a word. In this case the controller repeats the operations on the different fields of the vector.

IV. THE CONTROLLER

The function of the controller is to reduce the control overhead required from the host machine to drive VASTOR. In order to keep the VASTOR array continuously active, 50 control bits are

needed every microsecond. That is, a control bandwidth of 50 bits/microsecond must be supported. This rate exceeds the bandwidth of the entire PDP-11 UNIBUS. Hence, it must be reduced to a level which does not prevent the host from performing operations not related to VASTOR. The controller receives high level commands from the host machine, requiring a much lower control bandwidth. These commands are then translated into the sequences of control signals needed to drive the VASTOR array.

The complexity of the commands that have to be interpreted by the controller is represented by the examples given in section III. In order to support such operations, a hierarchical approach has been adopted. Each level in the hierarchy serves to reduce the bandwidth required from the higher levels. Furthermore, interpretation of high level commands has been made relatively simple because of the use of well defined interfaces between various levels.

The hierarchical approach led to the controller organization shown in Figure 7. It consists of three distinct units. The microcontroller which performs low level looping control operations, the buffer memory which is used as a communications medium, and the microprocessor which is responsible for interpreting high level commands received from the host and for space allocation within the VASTOR array. As such, the microprocessor performs functions similar to that of the "interpreter" in ECAM [1]. The microcontroller corresponds to the iteration control logic in ECAM. The three subsystems of VASTOR's controller are discussed briefly below.

4.1 THE MICROCONTROLLER

The microcontroller UC serves to remove some of the redundancy at its output, the array control lines, in order to reduce the bandwidth required at its input. Its sophistication, and therefore cost, can be selected to provide almost any desired bandwidth at its input. We have chosen to implement a device that executes sequences of microcode stored in an internal Read Only Memory, with primitive branching and looping capability. Input commands to the microcontroller come from a buffer memory M which, in turn, is filled by the microprocessor UP.

Linear microcode sequencing provides a large reduction in the control bandwidth. Hence, it was adopted as the main sequencing mechanism in the microcontroller. The starting address for a given microcode sequence is loaded from the buffer M. Since data can be made to appear in the VASTOR array in fields of consecutive locations, further compression of the control information is obtained with a simple loop counter/index register. This counter is decremented and tested to control microprogram loops. It also serves as an index register to modify the addresses transmitted by the controller to the array memory.

Some further control bandwidth compression is obtained by introducing a data-dependent branch. The associative sum of responders is compared to a reference in the microcode. One of two branch addresses is then selected from the buffer M.

4.2 THE BUFFER MEMORY

The buffer memory is divided into sixteen separate task control blocks. These blocks are filled by the microprocessor and interpreted by the microcontroller. Whenever the microcontroller finishes a task it interrupts the microprocessor to request the address of the next control block. Task control blocks contain up to 26 bytes of information. This includes starting and loop control information for the microcode of the microcontroller. It also includes specifications for the operands in the VASTOR array.

4.3 THE MICROPROCESSOR

Controller algorithms represented by one control block in the buffer memory take from 1 to several hundred microseconds to complete and to interrupt the microprocessor. These interrupts are usually quite simple to service but would be uneconomically frequent for the host machine. The microprocessor is therefore included to provide further compression of the control bandwidth. It simplifies the interfacing software by translating high-level operations into sequences of microcontroller tasks.

In addition to sequencing control, the microprocessor performs the storage management function. This includes allocating and freeing fields of storage, garbage collection, paging variables into the working store from the backing store, allowing the widths of elements (e.g. integers) to expand and contract, and segmenting vectors longer than the VASTOR array into manageable components.

V. INPUT/OUTPUT

Data transfer between VASTOR and the host machine is generally difficult because of the incompatibility of the addressable units in the two machines. While a host machine generally obtains all bits of a single element of a vector with one reference to its memory, VASTOR obtains one bit of each element. The transposition required to match the two machines is the source of the difficulty.

The simplest type of vector to transfer is a boolean vector, which is only one bit wide, figure 8a. In order to transfer such a vector from the host into the VASTOR array, its elements may be shifted serially by bit into the shift register SH. This is followed by a transfer from SH to a column of WK using the parallel mode (mode 3, section 2.2). If elements of the boolean vector are packed into bytes in the host machine, as is the case in some versions of APL, shift register SH may be loaded serially by byte through its 'A' port. In the current implemen-

tation, data rates for the bit-serial and byte-serial modes are 1 Mbit/s and 1 Mbyte/s respectively.

Consider now the case where data is presented to VASTOR so that some number of consecutive bits must be loaded into a single word, figure 8b. This may be achieved by first loading register RR of the ICU from the CONST line, figure 4, and then storing the content of RR in the enabled word. Due to that two-step sequence and the fact that only one word is enabled at a time, the transfer rate is limited to 500 Kbits/s.

The phrase structure may be used to increase the transfer rate of byte-organized data, as shown in figure 8c. This corresponds to mode 4 of section 2.2. The data rate achievable in this case is 2.5 Mbits/s. In this approach consecutive words from the host machine are not loaded into consecutive words of VASTOR. Rather, they are loaded into the same relative positions in consecutive phrases. A sentence structure consisting of two phrases per sentence also exists and may be used for 16-bit wide I/O transfers. The detailed procedure is given in reference [6].

VI. PERFORMANCE IN APPLICATION AREAS

This section discusses potential applications of a VASTOR processor. The primary application of VASTOR is as an auxiliary processor in a minicomputer system. In this case, it would serve to enhance the performance of the system in vector and associative operations. A second, and equally important, potential application derives from the fact that VASTOR can be regarded as a collection of 1-bit wide controllers driven in parallel by a host computer. Each of these two application areas is discussed briefly below.

Table 1.

Performance Comparison

Between VASTOR and a PDP-11/45
with Bipolar Memory in Vector Operations Involving
256-Element Vectors, with 16 Bits per Element.

Operation	Result	VASTOR Execution Time Microseconds	PDP-11/45 Execution Time Microseconds
Compare	Vector	4 us/bit * 16 bits = 64	3.225 us/word * 256 words = 825.6
Addition	Vector	10 us/bit * 16 bits = 160	1.9 us/word * 256 words = 486.4
Mark Largest Element	Vector	3 us/bit * 16 bits = 48	2.5 us/word * 256 words = 640
Compare to Scalar	Vector	3 us/bit * 16 bits = 48	2.5 us/word * 256 words = 640
Sum Reduction	Scalar	336 us/bit * 16 bits = 5376	1.5 us/word * 256 words = 384

Vector and associative operations are performed quite frequently in the operating system software of a computer. Symbol table manipulation and file management are two such examples. Also, some computer languages, such as APL and SNOBOL, are based upon the organization and manipulation of data in the form of vectors [4] or character strings [7]. A VASTOR processor is ideally suited to such tasks, and hence can take a considerable load off its host computer. Table 1 gives an estimate of VASTOR's performance in this area. The table gives execution times for a number of operations on 256-element vectors, where each element is 16 bits wide. These times are based on the current implementation using a processing element, the ICU, which runs at a 1 microsecond cycle time. For comparison, the times required to perform the same operations in a PDP-11/45 minicomputer are given. As can be seen from the data in Table 1, VASTOR is an order of magnitude faster than a PDP-11/45 when executing tasks that involve parallel operations on all elements of a vector. However, operations such as sum reduction (adding all

elements of a vector) take much more time. In this case, VASTOR's performance is limited by its inter-word communication facilities. However, when dealing with much longer vectors VASTOR's performance on sum reduction approaches its performance on vector addition. This is due to the fact that many elements of the vector would be stored in the same word of the array.

At the present stage of development of the VASTOR processor, it is very difficult to obtain an accurate estimate of the gain in performance that would result from adding a VASTOR processor to a minicomputer system. While the data in Table 1 indicate that considerable gain can be realized, this gain will be partially offset by the overhead resulting from transferring data between VASTOR and its host computer. This overhead is expected to be of the same order as that involved in transferring data between the main memory of a computer and a disk file. Therefore, VASTOR is most suited for use in applications where a number of vector operations have to be performed before a given vector is transferred back to the host machine.

Stand-alone ICU's have applications in process control and monitoring. VASTOR may be used in situations where a number of ICU's performing similar tasks are to be interfaced to a common host computer. In this case, VASTOR represents an organized way of performing I/O and control functions. Each ICU is capable of sampling data from and controlling an external device at data rates of the order of a few kilohertz. Status information and data such as minimum values, maximum values, averages, setpoints and enabling bits for each device may be kept in the corresponding working storage. The main limitation to this approach is that it is necessary to synchronize data transfer between the ICU's and the various devices.

VII. CONCLUSIONS

The VASTOR processor presented in this paper represents a trade-off between the capabilities and cost of the inter-word communication facilities in an associative processor. The result of this trade-off is a processor that allows a nontrivial associative processing capability to be incorporated in small scale minicomputer systems. The communication hardware provided in the VASTOR array enables data transfer among the words in the array without requiring costly and complicated hardware. It also results in simple backplane interconnections between different modules. The modular structure of VASTOR allows its capabilities to be expanded easily and economically.

Some of the limitations of the current implementation of VASTOR are due to the slow speed of the processing element used (the ICU). A faster and more powerful 1-bit wide processing element can lead to a considerable increase in performance without the need for any changes to the architecture. In fact, because of the low number of interconnections involved, the structure is well suited to integration. Some of the possibilities would be the implememtation of an array of 1-bit processors, or processors and memory on a single chip. Another possibility which is currently being investigated by the authors is the use of a table driven processing element made of memory only. Some other limitations of VASTOR, such as the difficulty of reordering a vector, are more fundamental. In order to perform such operations at high speed, a more complex, and hence more costly, inter-word communication scheme must be provided.

REFERENCES

1. Anderson, G.A., and Kain, R.Y., "A Content-Addressed Memory Designed for Data Base Applications", Proc. 1976 International Conf. on Parallel Processing, IEEE, New York, 1976, pp. 191-195.

2. Baudet, G. and Stevenson, D., "Optimal Sorting Algorithms for Parallel Computers", IEEE Trans. Comput., vol. C-27, pp. 84-87, Jan. 1978

3. Foster, C.C., Content Addressable Parallel Processors Van Nostrand Reinhold Co., New York, NY, 1976.

4. Grey, L.D. A Course in APL\360 with Applications, Addison-Wesley Publishing Co., Reading, Mass., 1973

5. Kaplan, A., "A Search Memory Subsystem for a General-Purpose Computer", Proc. AFIPS 1963 Fall Jt. Comp. Conf., Vol. 24, Spartan Books, Inc., Baltimore, Md., 1963, pp 193-200.

6. Loucks, W.M. and Snelgrove, W.M., "VASTOR 1978", Univ. Toronto Computer Engineering Report 13, June 1978.

7. Mukhophadhyay A., "Hardware Algorithms for Nonnumeric Computation", Proc. 5th Ann. Symp. Comp. Arch., April 1978, Palo Alto CA., pp. 8-16.

8. Parhami, B., "Associative Memories and Processors: An Overview and Selected Bibliography", Proc. IEEE, Vol. 61, pp. 722-730, June 1973.

9. Shooman, W., "Parallel Computing with Vertical Data", Proc. 1960 Eastern Jt. Comp. Conf., Eastern Jt.:c Computer Conf. 1960, pp 111-115.

10. Yau, S.S. and Fung, H.S. "Associative Processor Architecture - A Survey", ACM Computing Surveys, Vol 9, No. 1, pp. 3-27, March 1977.

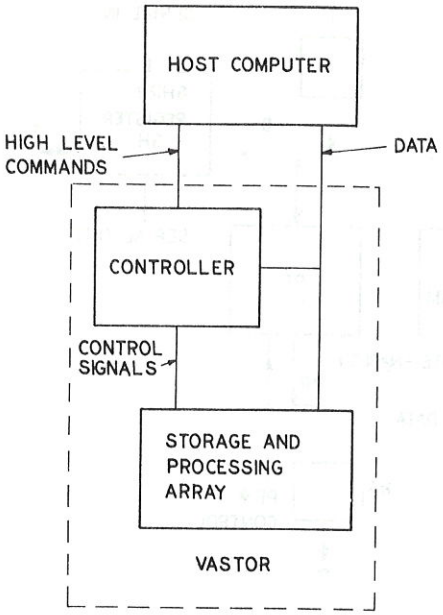

Fig. 1. The VASTOR processor

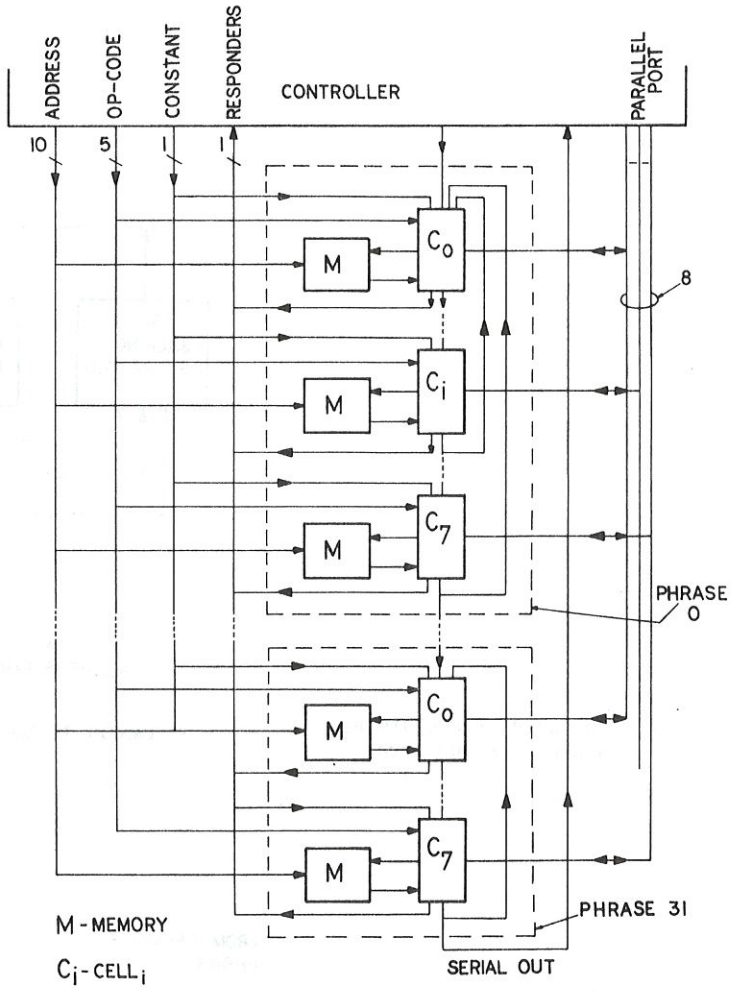

Fig. 2. Control and data paths

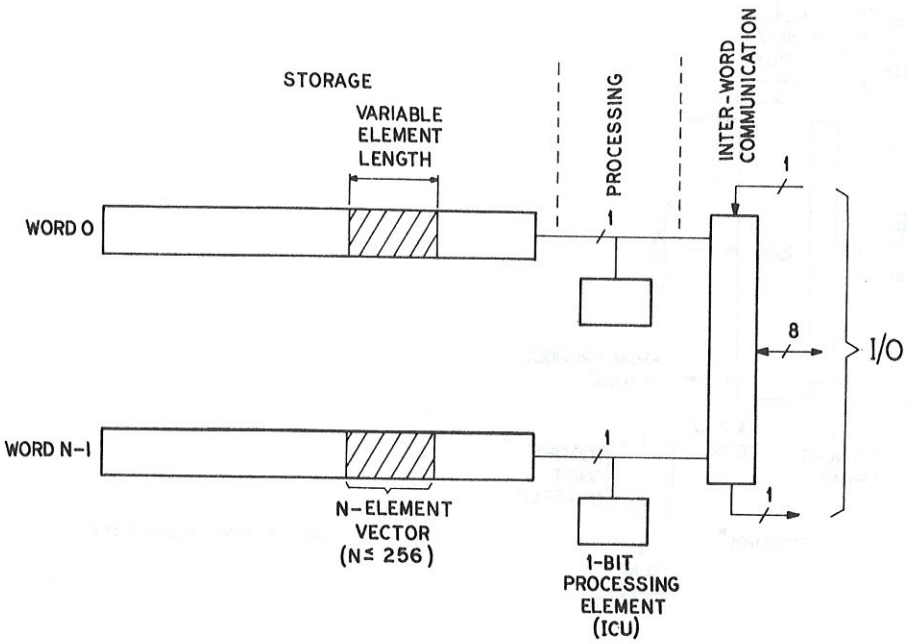

Fig. 3. Organization of the VASTOR ARRAY

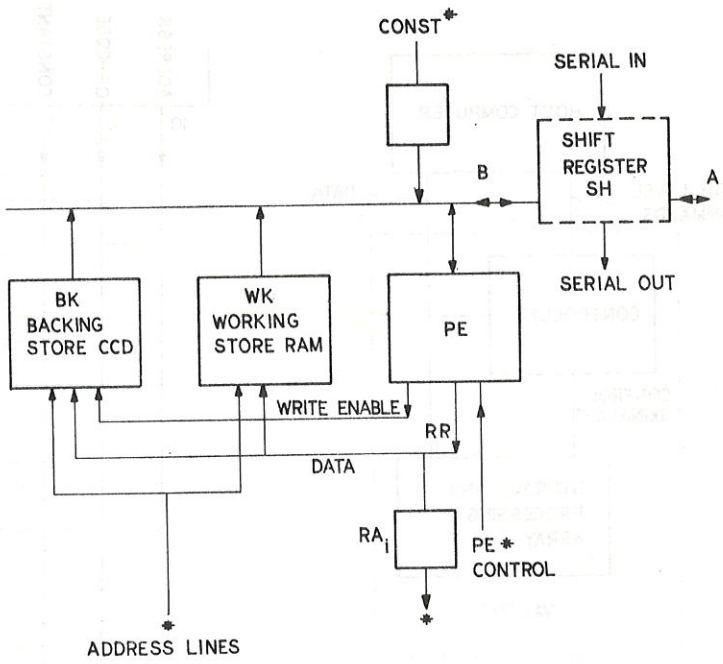

Fig. 4. One word of the storage and processing array

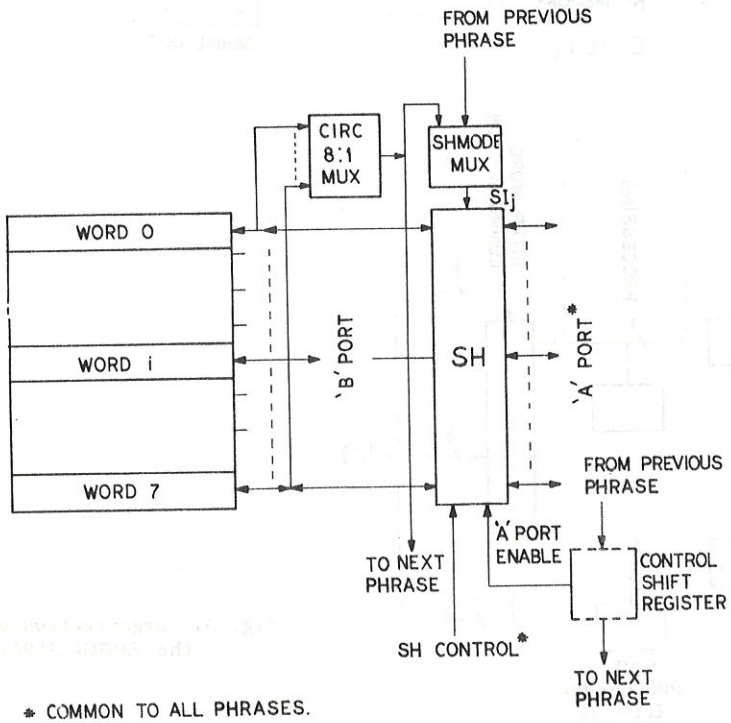

Fig. 5. The phrase structure

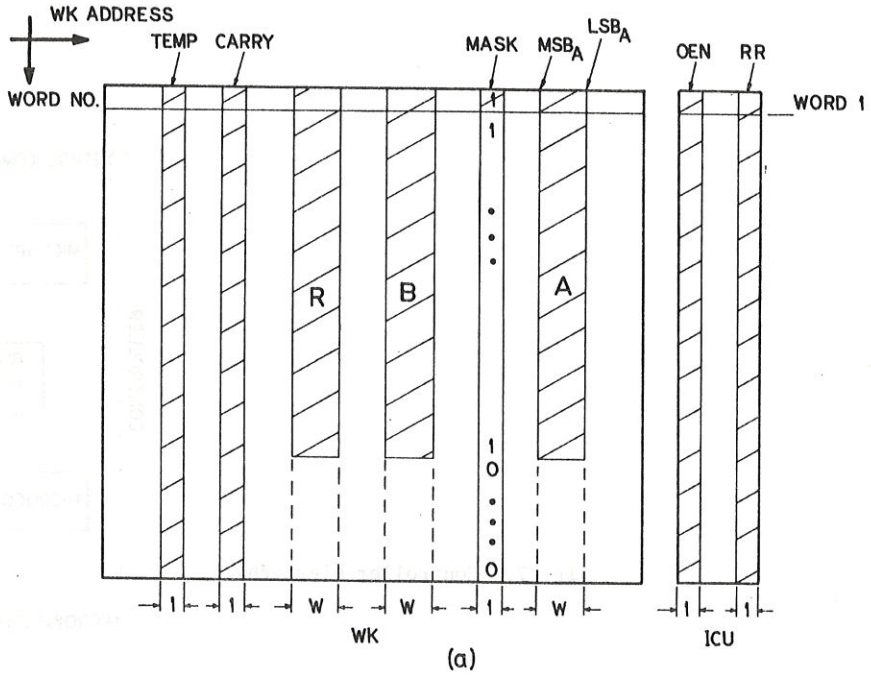

Fig. 6a. Vector addition example

```
CLEAR C                                  ; Clear carry vector - array operation (optional)
OEN ← MASK                               ; Vector array operation
FOR   i = 0, W - 1                       ; Controller operation
      ADDRESS (A_i) = LSB_A + i          ; Address calculation - controller operation
      ADDRESS (B_i) = LSB_B + i          ; Address calculation - controller operation
      ADDRESS (R_i) = LSB_R + i          ; Address calculation - controller operation
      ADDRESS (C)   = LSB_C              ; Address calculation - controller operation
      R_i ← A_i ∨ B_i ∨ C                ; Vector array operation
      C ← A_i∧B_i ∨ A_i∧C ∨ B_i∧C         ; Vector array operation
```

Fig. 6b. Implementation of vector addition

```
TEMP ← MASK                              ; Vector array operation
FOR    i = 0, W - 1                      ; Controller operation
       ADDRESS (A_i) = MSB_A - i         ; Address calculation - Controller operation
       RR ← TEMP.A_i                     ; Vector array operation
       IF (S = 1)                        ; Controller operation
                EXIT                     ; Controller operation
       ELSE IF (S ≠ 0)                   ; Controller operation
                TEMP ← RR                ; Vector array operation
```

Fig. 6c. Search for the largest element

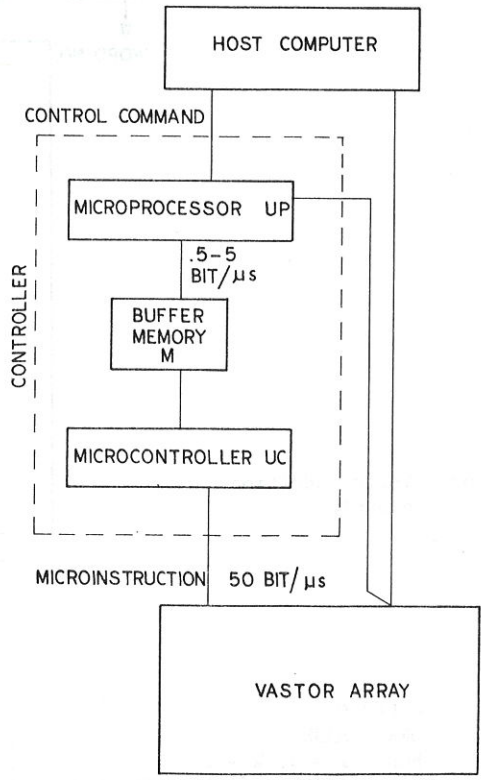

Fig. 7. Controller hierarchy

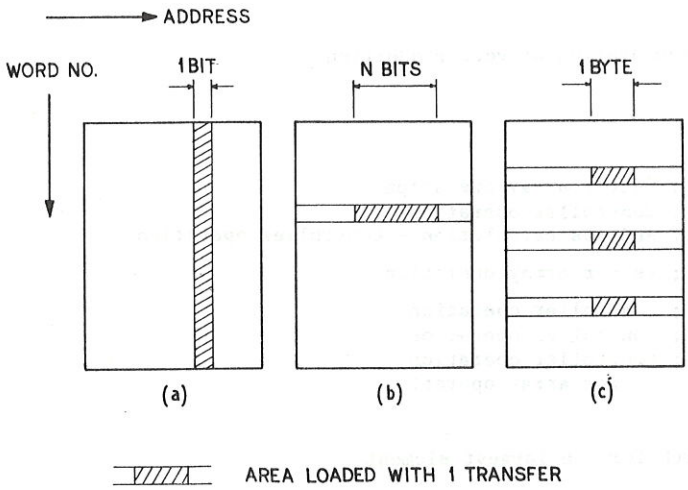

Fig. 8. Alternative modes for input/output transfers