AN ASSOCIATIVE PROCESSOR

Richard G. Ewing and Paul M. Davies
Abacus Incorporated, Santa Monica, California

1. INTRODUCTION

This paper describes the computer system designed under an Air Force sponsored study program to develop a non-cryogenic Associative Processor organization and to study its possible use in a variety of Aerospace applications. Two approaches were considered to this problem: one in which an associative memory would be added to a more or less conventional computer and another in which a new organization would be developed around the principle of memory distributed logic. The latter approach was chosen because it appears to result in a more efficient form of parallel processor.

Because of the nature of the intended use of the processor, emphasis was placed on network simplicity, on reduction of size and power, and especially, on reliability. While the processor organization was designed in terms of a particular mechanization-wire memory and integrated circuitry—the organization and algorithms are described here in general terms, and questions of mechanization are postponed to a final section.

When the fundamental limits of electrical and optical signal propagation speeds are reached, there are just two ways to further reduce the time to perform a given computation. One of these is by making things smaller, and the other is by performing parallel processing. But efforts to achieve efficient parallel processors have encountered several difficulties. First is the problem of providing sufficient memory and computing capability within a single module. Some parallel processors, such as the Holland machine, have employed relatively simple modules, but the memory capacity and computing capability of each module were limited. Others, such as the Solomon Computer, provide greater memory capacity and computing capability in the module, but each module approaches the complexity of a small computer.

Another serious problem is that of communication. For a periodic computing structure to be useful, it is essential that there be efficient paths for the communication of control signals and operands among the modules. In some parallel processors, the communication networks are more complex than the processing modules themselves.

The associative memory suggest itself as a basis for another approach to the problem of parallel processing. Logical operations are performed within the individual memory cells of this memory, and communication within the structure is particularly efficient. Extension of these principles to permit full logical and arithmetic capability within each memory cell would provide a high degree of processing parallelism.

We shall call an associative memory structure and its control logic, which is capable of performing such distributed computation, an Associative Processor.

In addition to the parallel computing capability, there are several other advantages which one may expect to achieve in the Associative Processor. These are:

1. The data storage and retrieval capabilities of the Associative Memory, which greatly simplify or eliminate such common data manipulations as sorting, col-
1. Sorting, searching, matching, cross-referencing, updating and list processing;

2. Programming simplifications based upon the possibility of ignoring the placement of data in memory and the extensive use of content addressing and ordered retrieval;

3. The periodic structure of a large portion of the processor. Periodicity of structure lends itself to integrated circuit techniques and batch fabrication. Interconnections between components become shorter and less tangled, reducing propagation delays and simplifying layout and checkout. Since the structure is periodic, it can easily be expanded in size;

4. Fault Tolerance. The periodic structure may permit an organization which is tolerant of memory or circuit element failures. If a cell fails, it may be possible to avoid its further use with little loss to the system capability. A program for an associative structure makes little or no reference to a unique cell so that loss of a cell would not confuse the program.

Two approaches have been taken in the past to solve the problems of parallel processing by using associative processing techniques. Rosin and Fuller have considered an associative memory under control of a general purpose computer. In Fuller's work, algorithms for a variety of arithmetic operations are built up as sequences of elementary operations performed by the rather limited word logic of the associative memory. In Davies, more extensive word logic is provided, and the control is integrated into the associative processor. The present paper represents an attempt to achieve the higher speed of the second approach with a considerably simpler logical structure, which could be mechanized from non-cryogenic components.

The design which was adopted provides a random access memory for program storage and a bit serial associative memory for data storage and parallel processing. The ability to write tags (i.e. to simultaneously write data in a selected bit position of a number of selected words), coupled with simplified word logic networks, permits relatively efficient bit serial algorithms for many kinds of parallel searches, parallel arithmetic and ordered retrieval. Methods were developed for treating certain classes of memory and circuit failures. For these cases, the processor can continue to operate in spite of a failure with only slight impairment of the overall system capability. In the area of communication, methods were developed for treating operand pairs in a variety of relative locations.

2. MEMORY DISTRIBUTED LOGIC

One of the fundamental features of the associative memory is that logical operations are performed within the memory cells. However, even in the random access memory a limited amount of logic is performed in the memory cell. The boolean function \( X_i \cdot Y_j \cdot S_{ij} \) is performed, where \( X_i \) is the selected X address coordinate, \( Y_j \) the selected Y address coordinate, and \( S_{ij} \) the bit stored at location \( ij \). The value of the function is read out on the sense line. In an associative memory, the memory logic is extended to permit selection of a memory cell on the basis of stored data. In some associative memories, this is accomplished by the function

\[
(S_i \leftrightarrow R_i) \cdot (S_j \leftrightarrow R_j) \cdots (S_n \leftrightarrow R_n)
\]

which is mechanized in each memory word cell. “\( S_i \leftrightarrow R_i \)”, the equivalence function, is the same as \( S_i \cdot R_i + S_i \cdot R_i \). \( S_i \) is the bit stored in the \( i \)-th bit position of a typical word, while \( R_i \) is the corresponding bit of a reference word stored in an external register. The function selects all words whose stored contents match the reference word. This can be improved to permit masking of selected bits as follows:

\[
[(S_i \leftrightarrow R_i) \cdot M_i] \cdot [(S_j \leftrightarrow R_j) + M_j] \cdots [S_n \leftrightarrow R_n] + M_n
\]

where \( M_i \) indicates whether the \( i \)-th bit is to be ignored in the comparison.

In addition to providing logic in each memory bit position, it is also profitable to have logic associated with each word cell. This is the case in certain word organized random access memories and in associative memories. In the first case, there is the word driver which may be a magnetic or semiconductor amplifier which responds to X and Y coordinate amplifier which responds to X and Y coordinate selection lines just as the typical bit cell does in a coincident current memory. In associative memories, there is usually a match detector with each
word which responds to the match logic described above. Ordinarily, the match detector has memory. These operations at both the bit level and the word level suggest the possibility of providing sufficient distributed logic to permit parallel computation throughout the memory structure.

In arriving at an Associative Processor capable of such parallel computation a number of important decisions must be made. One basic choice is whether to use a separate random access memory or the associative memory itself for program storage. The first choice is probably more practical since the random access memory is less expensive; furthermore, it will be easier to protect the associative portion from fault if the program is kept separate.

A second choice to be made is between bit parallel and bit serial operation. Certain associative operations such as the matching of fields for equality can be performed in bit parallel. On the other hand, to perform the more complex functions of arithmetic, it appears more convenient to use the bit serial approach, simplifying the bit cell by time sharing one logic module among all bits of a word.

A third problem is that of communication. To perform parallel computation, one must have access to the operands and operand pairs. In some cases, the operand pairs are stored together in the same word. In other cases, they are in adjacent words, while in still others, they are in non-adjacent words, but always some fixed number of words apart. Another common requirement involves operand pairs in which the first operand of each pair is common while the second operands are distinct. In this case, the common operand, in an external register, must be communicated to the others, stored in various memory cells. Still another communication problem is based upon the fact that while large portions of a problem may be susceptible to parallel processing, other parts may be essentially sequential. These also must be performed efficiently by the Associative Processor if they are not to offset the advantages gained in the parallel processing.

Techniques for solving these communication problems include the following:

1. Transmission of a common operand to all memory word cells.
2. Flexible control of field selection to permit operation on pairs of operands in the same words.
3. Use of shift registers for communication between words. These can be uni-directional or bi-directional and can be extended to two or more dimensions to give greater flexibility.
4. Forms of entry-exit ladder networks which permit rapid communication between non-adjacent word cells.

The following sections will describe the Associative Processor, which is based upon specific choices of these options.

3. ORGANIZATION

A block diagram of the Associative Processor is shown in Figure 1. It contains both a conventional random access memory (RAM) and an associative memory. The RAM provides storage for instructions and constants; it is accessed parallel by bit and serial by word. In processing operations, the Associative Memory is accessed parallel by word and serial by bit. In the organization under consideration, RAM contains 4000 twenty-four bit words, and the Associative Memory contains 500 ninety-six bit words.

Instructions accessed from RAM are transferred to the Instruction Register where they are held during execution. The D-Register,
which has the same length as a RAM word, serves as temporary storage for operands which participate in associative operations. For instance, the D-Register may hold the argument of a search, may receive data being retrieved from the Associative Memory, or may communicate with the external world. Data originating from outside of the Associative Processor can be transferred directly to either the Associative Memory or RAM. Direct input to the memories is under an automatic interrupt control.

In the Associative Memory, only one bit column at a time may be operated upon. The particular bit column is selected by either the A Counter, the B Counter, or the C Counter. Associated with the A and B Counters are the A and B Limit Registers. Each may contain a value which serves to define a maximum or minimum value of its companion counter. Together, each counter and limit register define a field which can be any length up to the number of bits in the Associative Memory word, and may overlap the field defined by the other counter and limit register.

The design of the Associative Processor is sufficiently general to permit implementation by a variety of memory elements and logic techniques. Therefore, the following description of the Associative Memory, shown in Figure 2, will present those characteristics which are essential to the design of the Associative Processor.

Storage for one bit is provided at each intersection of a word and a bit line. A pulse on a bit line causes a signal to be emitted by each bit on that line. The signals are transmitted through the word lines to the sense amplifiers.

The equivalence function is obtained in one of two ways depending upon the particular memory element. In some memories it is sufficient to exercise control over the polarity of the interrogating pulse, thereby achieving a signal output for a match and no output for a mismatch. In these cases, the bit element itself performs the equivalence function, \( S \rightarrow R \). In other memories, the stored bit is merely read out; the reference bit is transmitted to all sense amplifiers and logic associated with each sense amplifier generates the equivalence function.

Writing at a particular bit location is accomplished by passing a current through the intersecting bit and word lines. The polarity of the current in the word line, or in some cases the word and bit lines, determine the state of the written bit. By energizing all the word drivers and one bit driver, one bit of each word can be written into. The latter operation, which is sometimes referred to as “tagging”, plays a significant role in the design of the Associative Processor.

The logic associated with each word gives great power to the Associative Processor. This logic is identical for all words and consists of a sense amplifier, storage flip-flop, write amplifier, and control logic. Refer to Figure 3.3. The sense amplifier is bistable and remembers the match state from one interrogation to the next. The output of the sense amplifier determines the state of the storage flip-flop in various ways as determined by the control signals, \( E_s, E_r, \) and \( E_c \). In addition, the contents of each storage flip-flop can be shifted to the storage flip-flop in the word above under control of the signal, \( E_{sh} \). This provides communication between words.

One of the functions of the storage flip-flops is to control writing. In this operation, the storage flip-flops in the “1” state select the words that are to be written into, while the signals, \( W_1 \) and \( W_0 \), determine whether “1’s” or “0’s” are written by the selected write amplifiers. In addition, the output of each storage flip-flop is “ANDed” with the output of the corresponding sense amplifier. The outputs of these AND gates are ORed together to provide an output channel from the Associative Memory to the remainder of the Processor.
Control of the word logic networks is exercised through a Control Unit. This unit interprets the contents of the Instruction Register and D Flip-flop to determine the control signals, Es, Er, Ec, Ess, W1, and W0 which are transmitted to all word logic networks.

The D flip-flop and D Register are the data link between the Associative Memory and the Random Access Memory. Data, such as the argument for a search, are transferred from RAM in parallel to the D-Register. Each bit is then shifted into the D Flip-flop where it participates in the search operation. Data retrieved from the Associative Memory are transferred through an adder to the D Flip-flop and then to the D Register.

The Associative Processor offers a variety of processing options in terms of operand location and processing speed. The following list illustrates some of the possibilities:

1. \( D + M \rightarrow D \)
2. \( D + M_i \rightarrow M_i \)
3. \( M_j + M_i \rightarrow M_i \)
4. \( M_j + M_k \rightarrow M_k \)

(1) represents an operation occurring between the D Register and one selected word in the Associative Memory. The result goes to the D Register. (2) illustrates a process between D Register and many words in the Associative Memory. The third operation occurs between pairs of operands, each pair stored in a separate word. (4) represents an operation occurring between operands in different words. The same operation may simultaneously occur in many such pairs of words. In addition to these operations, many variations are possible, e.g. the operands may be located in different words with the results going to a third word.

The capability of the storage flip-flops to act as a shift register provides the communication link between adjacent words. Another use of this shift register occurs in counting the number of words which satisfy a search algorithm. This is accomplished by operating the storage flip-flops as a shift register and counting the number of "1's" shifted out. Each "1" corresponds to a word that satisfies the search.

4. COMMAND STRUCTURE

There are two types of instructions in the Associative Processor. Instructions which exercise control over the Associative Memory shall be referred to as associative instructions. Instructions which provide access to the Random Access Memory or that perform control transfers shall be referred to as non-associative instructions. A list of non-associative instructions follows:

- LA Load the contents of memory location \( M \) into the A-Counter and Limit Register.
- LB Load the contents of memory location \( M \) into the B-Counter and Limit Register.
- LC Load the contents of memory location \( M \) into the C-Counter.
- LD Load the contents of memory location \( M \) into the D-Register.
- LM Load the contents of the D Register into Memory Location \( M \).
- TD Transfer to location \( M \) if the D flip-flop equals "zero", otherwise proceed sequentially.
- TO Transfer to location \( M \) if the output of the OR gate equals "zero", otherwise proceed sequentially.
- TI Transfer to the location specified by memory location \( M \).
- TU Unconditionally transfer to location \( M \).
- SH Up-shift the storage flip-flops a number of times equal to \( M \).
- SC Up-shift the storage flip-flops a number of times equal to \( M \). The C Counter counts the number of ones shifted into the highest level storage flip-flop.
- CD Transfer the contents of the C Counter to the D Register.
- ID Input data word from external device to the D register.
- OD Output data word from the D register to external device.

*The input and output commands generally work in conjunction with the automatic interrupt facility. An external device requests an interruption by turning on an interrupt flip-flop. This causes the Processor to complete the present instruction, store the contents of the Instruction Address Counter in memory, and jump to an Input or Output routine. These routines can transfer I-O data between the D Register and either the RAM or the Associative Memory.
Each associative instruction controls the processing during a single bit time, except when it is executed in a Repeat Mode. The instructions are divided into a number of fields, each of which specifies the control of a separate part of the Processor. Figure 3 summarizes these fields which are described below:

**Column Select (CS):** The contents of this field determine what bit column of the Associative Memory is to be interrogated or written into, either by specifying the A, B, or C counter, which in turn selects the bit column, or by directly specifying one of the four bit columns. The four directly specified columns are ordinarily used for the storage of tag bits.
- A A counter
- B B counter
- C C counter
- T1 Column 1
- T2 Column 2
- T3 Column 3
- T4 Column 4

**Counter Control (CC):** The contents of this field determine whether the counter selected in CS will be modified. A counter may be decremented or incremented by one.
- IN Increment
- DE Decrement
- NC No Change

**Transfer Control (TC):** In general, instructions are accessed from sequential memory locations in RAM. To facilitate exiting from a subroutine, it is desirable to be able to transfer to another location when the contents of a counter become equal to the associated Limit Register.
- Ta Transfer to memory location 0 when the A-Counter becomes equal to the A Limit Register
- Tb Transfer to memory location 1 when the B-Counter becomes equal to the B Limit Register
- NC Proceed sequentially.

**Adder Control (AC):** The contents of this field control the manner in which the output of the OR gate is transferred into the D Flip-flop.
- L OR gate output copied into the D Flip-flop.
- C Complemented OR gate output copied into the D Flip-flop.
- A OR gate output added to the D Flip-flop. The carry is stored in a flip-flop associated with the Adder.
- S OR gate output subtracted from the D Flip-flop
- NC No transfer

**D-Register Shift (RS):** The D Register can be made to shift one bit in either direction. The shift is end around when the contents of the AC field indicate that no transfer is to take place.
- R Right shift
- L Left shift
- NC No Change

![Figure 3. Word Logic.](From the collection of the Computer History Museum (www.computerhistory.org))
Interrogate Control (IC): Upon interrogation, the sense amplifier responds with a "1" output to a match condition between the interrogated bit and what previously has been labeled a reference bit. The IC field defines the reference bit:

1 Interrogate for "1". If the stored bit is equal to "1", the sense amplifier will be set.
2 Interrogate for "0". If the stored bit is equal to "0", the sense amplifier will be set.
3 If the D flip-flop is equal to "1", interrogate for "1", if "0", interrogate for "0".
4 If the D flip-flop is equal to "0", interrogate for "1", if "1", interrogate for "0".

Write Control (WC): This field specifies writing to occur in the words for which the storage flip-flop is equal to "1". During writing, the IC field is available to determine whether "1's" or "0's" are to be written.

W—Write
NC—Do not Write

Storage Flip-flop Control (SC): This field specifies the state of the control signals which are common to the input logic of each of the storage flip-flops. This logic influences the transfer of data from the sense amplifiers to the storage flip-flops.

NC 0→Es, 0→Er
   No transfer takes place.
Es 1→Es, 0→Er
   The Storage flip-flop is set if the sense amplifier is equal to "1".
Er 0→Es, 1→Er
   The Storage flip-flop is reset if the sense amplifier is equal to "0".
Esr 1→Es, 1→Er
   The state of the sense amplifier is copied by the storage flip-flop.
D 1→Es, 0→Er,
   if the D flip-flop is equal to "1".
0→Es, 1→Er,
   if the D flip-flop is equal to "0".

OR 1→Er
   if the output of the OR gate is equal to "1".
Ec 1→Ec
   The storage flip-flop is complemented if the sense amplifier is equal to "1".

Instruction Type (IT): This field appears in both associative and non-associative instructions. The contents designate the instruction as being associative or not.

A Associative
Â Non-associative

Repeat Mode (RM): It is sometimes desirable to repeat an instruction during the execution of a simple search.

R Repeat until the counter specified by CS becomes equal to its limit register.
R Do not repeat instruction.

The time required to execute one associative instruction is measured from the time the instruction is transferred into the Instruction Register to the time the sense amplifier outputs are transferred into the storage flip-flops. During this time, the next instruction is accessed, and the previous output of the storage flip-flops can be transferred to the D flip-flop. This time will be referred to as a "bit time". Associative instructions are accessed at a rate of one per bit time. It should be noted that the AC field of the associative instruction will control the disposition of storage flip-flop data that resulted from an interrogation specified by the previous associative instruction. Non-associative data transferring instructions require two bit times for execution (Both an instruction and an operand must be accessed from the Random Access Memory). Non-associative instructions which transfer control require one bit time for execution.

5. MICROPROGRAMMED ALGORITHMS

The method by which associative instructions are controlled constitutes one of the
major factors contributing to the flexibility of the Associative Processor. Each field of the instruction directly specifies some control function, so that numerous associative instructions can be micro-programmed by the appropriate selection of fields. Despite the large number of options available, the central control unit is very simple since the control functions are obtained directly from the instruction fields.

Following, is a description of several important categories of associative instructions with typical microprograms; algorithms are also given for more complex data retrieval and arithmetic processes built up for microprogrammed associative instructions.

Possibly the most often used algorithm is ordered retrieval. A number of algorithms for retrieval have appeared in recent publications. Among these, the algorithm presented by Lewin appears to be fastest, making its' implementation in the Associative Processor an attractive consideration. However, in view of its' unique hardware requirement, (i.e., the equivalent of a three state sense amplifier on each bit column), an algorithm was developed which utilizes logic of a more general nature. In fact, the development of this algorithm greatly influenced the design of the word logic. The algorithm, presented in detail later in this section, retrieves one bit of data for each bit time of execution.

The ordered retrieval algorithm is used whenever data is to be retrieved from the Associative Memory or whenever it is necessary to select a word in which to write data. Since the time required to identify a word is related to the number of bits that must be searched, it is often desirable to have stored in each word a compact address field. Having such a field also provides a convenient way to distinguish between two words which might otherwise contain the same data.

In most instances, before the execution of an associative search, it is necessary to precondition the storage flip-flops either by setting or resetting them all, or by setting those corresponding to the set of words which is to be searched. Accomplishing this last operation requires transferring the contents of the tag bit (or whichever bit position holds the information) into the sense amplifiers and then copying the states of the sense amplifiers into the storage flip-flops. These operations can be executed with a single associative instruction.

\[
\begin{array}{cccccccccccc}
CS & CC & TC & AC & RS & IC & WC & SC & IT & RM \\
T1 & NC & NC & NC & NC & 1 & NC & Esr & A & R \\
\end{array}
\]

Setting (or resetting) all the storage flip-flops requires two associative instructions. The procedure is to interrogate the same bit column twice; once for "1's", and once for "0's". The following instructions reset the storage flip-flops.

\[
\begin{array}{cccccccccccc}
CS & CC & TC & AC & RS & IC & WC & SC & IT & RM \\
1. T1 & NC & NC & NC & NC & Z & NC & Er & A & R \\
2. T1 & NC & NC & NC & NC & 1 & NC & Er & A & R \\
\end{array}
\]

The following associative searches have been microprogrammed:

- Equality
- Maximum value
- Less than
- Minimum value
- Less than or equal
- Similarity
- Greater than
- Ordered Retrieval
- Greater than or equal

Except for Similarity, the execution time or each search is one bit time for each bit of the argument.

The object of most searches is to leave the storage flip-flops in a state which defines the locations of those words which meet the conditions of the search. However, it is possible to obtain the complementary set of words, as well as the set obtained by ORing or ANDing the results of several different searches. Following are a few examples of search microprograms:

Equality Search

\[
\begin{array}{cccccccccccc}
CS & CC & TC & AC & RS & IC & WC & SC & IT & RM \\
A & IN & TA & NC & R & D & NC & Er & A & R \\
\end{array}
\]

This instruction searches the words in memory whose storage flip-flops are initially true. At the end of the search, the storage flip-flops identify those words containing a field exactly matching the field in the D Register. The field in memory is defined by the A Counter and Limit Register. Each bit of the field is interrogated starting with the least significant bit.
The D flip-flop specifies the match conditions. A mismatch will cause the appropriate storage flip-flop to be reset.

Less Than

<table>
<thead>
<tr>
<th>CS</th>
<th>CC</th>
<th>TC</th>
<th>AC</th>
<th>RS</th>
<th>IC</th>
<th>WC</th>
<th>SC</th>
<th>IT</th>
<th>RM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>IN</td>
<td>NC</td>
<td>NC</td>
<td>R</td>
<td>Z</td>
<td>NC</td>
<td>D</td>
<td>A</td>
<td>R</td>
</tr>
</tbody>
</table>

This instruction causes a storage flip-flop to be set if the interrogated memory bit is "0" and the D flip-flop "1" and reset if the memory bit is "1" and the D flip-flop "0". Otherwise the storage flip-flops are unchanged. Essentially, this logic is the same as borrow logic with the contents of the D Register being subtracted from the contents of each memory word. When the storage flip-flop is equal to one, the memory word is less than the data register word.

Ordered Retrieval

<table>
<thead>
<tr>
<th>CS</th>
<th>CC</th>
<th>TC</th>
<th>AC</th>
<th>RS</th>
<th>IC</th>
<th>WC</th>
<th>SC</th>
<th>IT</th>
<th>RM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>DE</td>
<td>NC</td>
<td>L</td>
<td>L</td>
<td>1</td>
<td>NC</td>
<td>Or</td>
<td>A</td>
<td>R</td>
</tr>
</tbody>
</table>

This instruction transfers to the D Register the maximum value field of the set of fields which are identified by the true storage flip-flops. Starting with the most significant bit position of the search field, each bit position is sequentially interrogated for a one. When any sense amplifier indicates a match for a field still in the search set, the storage flip-flops corresponding to those sense amplifiers indicating a mismatch are reset. Each time a bit position is interrogated and is found to contain a "1" in any of the words remaining in the search set, a "1" is transferred into the D flip-flop.

Many arithmetic and logical microprograms have been developed for the Associative Processor. Below is a partial list. The operations are identified by S when they apply to an operation between a single pair of operands. An SP refers to an operation between one operand in the D Register and many operands in the memory, and P refers to simultaneous operations between many pairs of operands in memory.

The number of bit times required for the execution of each operation appears in parenthesis.

Add \( M + D \rightarrow D \)

\[ S \quad (1 + \text{no. of operand bits}) \]

Add \( D + M \rightarrow M \)

\[ \text{SP} \quad (12 \times \text{no. of operand bits}) \]

Add \( M_1 + M_2 \rightarrow M_3 \)

\[ \text{P} \quad (12 \times \text{no. of operand bits}) \]

Multiply \( M_1 \times M_2 \rightarrow D, M_3 \)

\[ \text{S} \quad (20 \times \text{no. of multiplier bits}) \]

for a 24 bit multiplicand

Multiply \( M_1 \times M_2 \rightarrow M_3 \)

\[ \text{P} \quad (\text{no. of multiplier bits} \times \text{no. of multiplicand bits}) \]

Divide \( D / M_1 \rightarrow M_3 \)

\[ \text{S} \quad (30 \times \text{no. of operand bits}) \]

Add \( (M_1 + M_2) \rightarrow M_3 \)

Field \( M_1 \) is added to field \( M_2 \) in all words which contain a "1" in bit column (T1). Field \( M_1 \) is defined by the A counter and Limit Register. Field \( M_2 \) is defined by the B counter. Bit column T2 is temporary storage for the carry. Addition is executed in the following steps:

1. The \( j^{th} \) bit of \( M_1 \) is transferred to the storage flip-flops.
2. The contents of the storage flip-flops are added to the \( j^{th} \) bit of \( M_2 \). The carry is developed in the storage flip-flops during the addition.
3. The partial carry resulting from the carry addition (preceding step 1) to the \( j^{th} \) bit of \( M_2 \) is ORed with the partial carry in the storage flip-flop. The final carry results in the storage flip-flop.
4. The carry is added to the \( j+1 \) bit of \( M_2 \); the resulting partial carry is stored in T2.

The following program executes this addition algorithm:

<table>
<thead>
<tr>
<th>CS</th>
<th>CC</th>
<th>TC</th>
<th>AC</th>
<th>RS</th>
<th>IC</th>
<th>WC</th>
<th>SC</th>
<th>IT</th>
<th>RM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T1</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>Esr</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>A</td>
<td>IN</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>1</td>
<td>Er</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>A</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>W</td>
<td>Er</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>IN</td>
<td>TB</td>
<td>NC</td>
<td>NC</td>
<td>0</td>
<td>W</td>
<td>NC</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>T2</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>1</td>
<td>NC</td>
<td>Es</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>T2</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>0</td>
<td>W</td>
<td>NC</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>B</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>1</td>
<td>NC</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>1</td>
<td>W</td>
<td>Er</td>
</tr>
<tr>
<td>B</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>0</td>
<td>W</td>
<td>NC</td>
</tr>
<tr>
<td>T2</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>1</td>
<td>W</td>
<td>NC</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>12</td>
<td>Transfer to 1 (TU)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
Instruction one transfers T1 to the storage flip-flops. Instruction two interrogates the jth bit of M1 and resets the storage flip-flop where mismatches occur. The resulting contents of the storage flip-flops constitute the AND function of T1 and the jth bit of M1. Instructions three, four, and five have the effect of complementing the jth bit of M2 in words for which the storage flip-flop is equal to "1". In addition, if the jth bit of M2 were equal to "1", the storage flip-flop would remain equal to "1", thereby representing the partial carry. Instruction six OR's the carry resulting from addition of the last carry to the jth bit of M2 to the partial carry in the storage flip-flop. Instruction seven clears the T2 column in preparation for the next carry storage. Instructions eight, nine, and ten add the carry to the j+1 bit of M2 using the same technique as instructions three, four, and five. Instruction eleven stores the partial carry in T2. The routine is exited after instruction five has been executed and the addition of the most significant bits completed.

6. FAULT TOLERANCE

An interesting characteristic of this particular associative memory is its structural periodicity. Each bit driver is identical to any other, and the logic of each word is identical to that of any other. There is no addressing matrix and no ladder network. The existence of these characteristics suggest the possibility of making the system operation insensitive to local malfunctions in the memory stack and memory circuits.

There are numerous possible causes of malfunction in the Associative Memory. However, most malfunctions can be placed in one of four categories. The first is characterized by the inability of the sense amplifier to change state. The second is characterized by the inability of the storage flip-flop to change state. The third category consists of those malfunctions which cause a write amplifier to fail, and the fourth consists of those malfunctions which cause a bit driver to fail.

The procedures to be described below consist of exercising control over the functions common to the logic of each word in such a way as to gain a system tolerance to these malfunctions. Other malfunctions may occur for which the only safeguard would be the utilization of component redundancy techniques. In the following discussion it will be assumed that no more than one type of malfunction exists in any one word.

To cope with these malfunctions, it is of primary importance to guard against spurious results in the retrieval operations. It is of little importance if, in a particular word, some other operation goes awry as a result of a malfunction. This merely produces meaningless data in that word, which is unimportant if provisions are made never to write into and never to retrieve information from such a word. Since a word is selected for writing by means of the retrieval algorithm, the fundamental problem is to guard against incorrect retrieval as a result of a malfunction.

Malfunctions of the first category cause the sense amplifier to permanently store either a "1" or a "0". If a "0" is stored, the storage flip-flop can never be set. The retrieval of data from a particular word and the ability to write into a particular word are dependent upon the storage flip-flop of that word being in the "1" state. If the storage flip-flop can never be set, the word appears to be nonexistent. If information were stored in the word prior to the malfunction, it would become irretrievable; however, as long as the malfunction existed it would be impossible for data to be inadvertently stored in that location.

A "1" locked in the sense amplifier presents a different problem. If the storage flip-flop of such a malfunctioning word is true, execution of the retrieval algorithm will result in the retrieval of a word of "1's". To prevent this, it is necessary to perform an operation prior to the retrieval algorithm which will reset the storage flip-flops in words whose sense amplifiers are locked in the "1" state without altering the states of the other storage flip-flops. Furthermore, this operation must not turn on a storage flip-flop in a word whose sense amplifier is locked in the "0" state. This can be accomplished as follows: Reset the sense amplifiers by interrogating a column of "0's". Then by executing the complement control Ec, complement all storage flip-flops corresponding to sense amplifiers still in the "1" state.
Malfunctions of the second category are those in which the storage flip-flop does not change state. If the storage flip-flop is locked in the "0" state, the associated word can not participate in any reading or writing operations. Such a condition will cause that word to appear to be nonexistent. A permanently stored "1", however, may cause an erroneous readout during execution of the retrieval algorithm. To avoid this, in the case of maximum value retrieval, the affected word should be loaded with "0's" prior to retrieval. A method of determining whether any storage flip-flops are malfunctioning is first to interrogate a column of "1's" so as to set the sense amplifiers, then to execute the Es and then Ec functions. The output of the OR gate will be "1" if any storage flip-flops remain true.

The third category consists of those malfunctions which disable a write amplifier. If a disabled write amplifier can be detected and the word uniquely marked, further operations upon that word can be avoided. Detection is accomplished by writing a pattern of "1's" and "0's" in each word. An equality search is then made using the same pattern to identify any words in which the pattern was not successfully recorded. On each successive execution of this procedure, a pattern different from the last pattern must be used. If the pattern which is read out of a given word is not identical to the current pattern, then the write amplifier driving that word is malfunctioning. The erroneous pattern cannot be used again. If the length of the pattern is N bits, then N bits in each word of the memory must be relegated to storage of the checking patterns at the time of execution of the checking procedure. The number of different patterns must exceed by two the number of malfunctions which can be tolerated.

The malfunctions of the last category are associated with the bit drivers. Once a bit driver has failed there is no way of writing into or interrogating that particular bit column. Therefore, it is necessary to isolate malfunctioning bit drivers. Detection of a malfunctioning bit driver is accomplished by designating two words of memory as test words, one of which would contain stored "1's" and the other, stored "0's". Special logic on each of these two word lines would compare the real output to the theoretical output upon each interrogation. A discrepancy would interrupt the program, thereby allowing the execution of a programmed corrective action.

7. MECHANIZATION

The critical problem in mechanizing the Associative Processor is, of course, the implementation of the associative memory. The critical requirements of this memory are the following:

1. Non-destructive readout. This is essential for the Processor described above; however, with slight modification, destructive readout could be tolerated provided the write time was comparable to the read time.
2. Small ratio of word write current to read signal. This significantly influences the complexity of the sense amplifier and write amplifier and limits the number of words in memory.
3. Short write cycle. This makes tagging operations practical.
4. Short interrogation cycle. This is especially important for a bit serial processor.
5. Limited power consumption.

A number of memories were analyzed for compliance with these requirements, including:

1. Plated Wire
2. Laminated Ferrite
3. Bi-core
4. Biax

At this time, the most promising of these for both the Associative Memory and RAM appears to be the Plated Wire Memory. It can be operated in a nondestructive readout mode, requires a word current of approximately 25 ma, and can be interrogated or written into at a 10 mc rate.

The closed flux path, rotational switching mode and lose coupling of the switched flux to the sense line contribute to the high ratio of read signal to word write current and to the low power consumption of each bit.

Our laboratory evaluation of the Plated Wire has indicated the feasibility of using integrated
circuitry for both the word logic and the bit drivers. The periodic structure of large portions of the Associative Processor and the requirement for a relatively small number of circuit types facilitate mechanization with integrated circuits.

The optimum sizes of the Associative Memory and the Random Access Memory depend greatly upon the application. For the class of Aerospace applications for which the Processor was conceived, the following dimensions and parameters were chosen:

RAM: 4096 words of 24 bits each.
Associative Memory: 512 words of 96 bits each.
Bit time = Memory cycle time = .1 μsec.

8. CONCLUSIONS

The Associative Processor possesses several virtues as a parallel processor. The basic processing module, i.e. one word of Associative Memory with its word logic, possesses considerable computing and memory capability for its size and complexity. This implies a large amount of parallel processing per dollar. Communication within the Processor is relatively efficient, especially where associative techniques can be employed. Most of the Processor is periodic in structure and, therefore, compatible with batch fabrication techniques and integrated circuitry. Fault tolerance techniques can be employed, at least to a degree. The non-periodic control structure of the Processor is relatively simple. And, finally, the microprogramming characteristics of the instructions permit and encourage programming experiments.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the support of the Space Systems Division of the Air Force under whose study contract the work was done and Space Technology Laboratories, Inc., who participated with Abacus, Inc. in the study.

We wish particularly to thank Mr. M. Waxman of S.T.L. for his work in evaluating the Associative Processor in several aerospace applications and Mr. T. Stupar who assisted in evaluating the technical feasibility of the system.

REFERENCES


