

A Fast, Flexible, Highly Parallel Associative Processor

JOHN E. SHORE AND FRANK A. POLKINGHORN, JR.

*Intercept and Signal Processing Branch
Electronics Warfare Division*

November 28, 1969



NAVAL RESEARCH LABORATORY
Washington, D.C.

CONTENTS

Abstract	iii
Problem Status	iii
Authorization	iii
Glossary of Abbreviations	iv
1. Introduction	1
2. Associative Devices	2
3. Introduction to the Proposed Associative Processor	2
4. Description of the AP	3
4.1 The Syllable Definition Register	4
4.2 The Comparand	5
4.3 The Search Command Registers and the Associative Search Operation	5
4.4 The Multioperation Register	8
4.4.1 The Multiwrite	9
4.4.2 The Multiadd	9
4.5 Conventionally Addressed Operations	10
4.6 Simultaneous Operations	10
4.7 Complex Arithmetic Operations	12
4.7.1 Logical Operations	12
4.7.2 Multiplication of all AM Words by the Comparand	13
4.7.3 Addition of Two Syllables Within Every Word	14
4.7.4 Multiplication of Two Syllables Within Every Word	14
5. Design of the Associative Processor	15
5.1 The Chip	15
5.1.1 Addition	16
5.1.2 The Associative Search	19

5.1.2.1	The "Don't Care" Search	20
5.1.2.2	The Exact Match Search	20
5.1.2.3	Search for Larger Elements	20
5.1.2.4	Search for Smaller Elements	21
5.1.2.5	The SDR and Its Use in the Associative Search	22
5.1.3	The AM Read	22
5.1.4	The AM Write	22
5.1.5	Simultaneous Operations	23
5.2	The Response Store	23
5.2.1	The RS and the Associative Search	24
5.2.2	The RS and the Multioperations	25
5.2.3	The Conventional Read, Write, and Add	25
5.2.4	The Response Bit Complement Operation	25
5.2.5	The RS and Simultaneous Operations	26
5.3	Response Resolution Network	26
5.4	Timing Considerations	30
5.4.1	The Associative Search Time	31
5.4.2	Addition Time	33
5.4.2.1	Multiaddition Time	33
5.4.2.2	Conventional Addition Time	34
5.4.3	AM Write Time	34
5.4.3.1	Multiwrite Time	34
5.4.3.2	Conventional Write Time	35
5.4.4	AM Read Time	35
5.4.5	Reduced Operation Times	35
5.4.6	Simultaneous Operation Times	36
5.4.7	The Response Resolution Time	37
5.4.8	Practical Considerations	37
5.5	Interword Communication – Future Developments in the AP Communication Topology	38
APPENDIX A	– Boolean Equations for the Associative Chip	41
APPENDIX B	– Boolean Equations for the Response Store	42

ABSTRACT

The logical design and operation of a general purpose associative processor (GPAP) is described. The basic circuit consists of a powerful associative cell which may be combined with an integral number of identical cells and implemented in MSI or LSI at a reasonable cost. A complete description of this cell, together with logic diagrams, Boolean equations, and a detailed timing analysis are presented.

The processor obtained by connecting these cells together in quantity has both variable syllable and variable instruction capability. That is, the total associative word length can be split on a software basis into any number of syllables (fields), each of arbitrary length. The search criteria (greater than, less than, greater than or equal to, less than or equal to, exact match, and don't care) can be specified independently for each of these syllables. The search time depends only on the width of the largest syllable and is typically less than 1 microsecond (assuming gate delays of approximately 20 nanoseconds).

The problem of multiple responses (priority resolution) is considered in detail. The design of a high-speed, inexpensive (in terms of the number of gates required per word) response resolution network is presented.

Other GPAP operations include conventionally addressed read, write, add, multiwrite, multiadd, and logical operations. Simultaneous operations, such as write-on-match and add-on-match, are also possible. These capabilities, as well as several other unique properties of the design which contribute to its general purpose character and high speed of operation are described in detail.

GPAP will be controlled by a microprogrammable processor with timing produced by a clock inhibit network which counts down a compiler-produced operation-time parameter.

PROBLEM STATUS

This is an interim report on a continuing NRL Problem.

AUTHORIZATION

NRL Problem R06-41
Project NavAir System A37-533-000/6521/WF08-151-702

Manuscript submitted July 14, 1969.

GLOSSARY OF ABBREVIATIONS

AM	Associative memory
AMAR	Associative memory address register
AMOB	Associative memory output buffer
AP	Associative processor
CIN	Clock inhibit network
COMP	Comparand register
GPAP	General purpose associative processor
IR	Instruction register
J	"J" input to flip-flop
K	"K" input to flip-flop
MOR	Multioperation register
MPP	Microprogram processor
Q_n	Flip-flop output before clock pulse $n + 1$
Q_{n+1}	Flip-flop output after clock pulse $n + 1$
RB	Response bit
RRB	Response resolution bit
RRN	Response resolution network
RS	Response store
SC1	Search command one register
SC2	Search command two register
SDR	Syllable definition register

A FAST, FLEXIBLE, HIGHLY PARALLEL ASSOCIATIVE PROCESSOR

1. Introduction

Over the years since the introduction of the modern general purpose computer, processing speeds have increased by several orders of magnitude. Most of the improvement has resulted from better hardware and better programming techniques. Only a small part is the result of modifications to the organization of the machines. Even today, general purpose computers on the market for the most part still follow the basic organization of the von Neumann machine and perform calculations sequentially. Such organizational modifications that have been made for the most part involve an increase in parallel processing capability.

These departures from the classic von Neumann organization have generally followed one of two basic paths. In one, several (von Neumann) general purpose computers are tied together into a multiprocessing system, the first of which was the Burroughs D825 system. In this type of configuration, different branches of one program (or several independent programs) are executed independently. That is, different multiprocessor elements execute different instructions on different data bases. It is this extremely free structure that limits the size of multiprocessing systems, as expressed by the number of individual processors. This is due both to the expense (each element is a general purpose computer) and to the software problem of tying many sophisticated processors together so freely.

Many general problems, both military and commercial, cannot be treated with currently available sequential processors or multiprocessors. Examples include air traffic control, numerical weather forecasting through solution of the atmospheric equations, atomic reactor calculations, mapping and charting, complex pattern recognition, ocean surveillance, and signal processing. The inability to ever meet these real time processing requirements on a sequential machine relates to the intrinsic limitations of switching speeds, memory speeds, and signal propagation delays. Only with the development of an advanced parallel processing capability can we expect to effectively treat these problems. All of the problems mentioned have in common that they require a processing system in which a large number of processing elements execute the same instruction on different data bases (as compared with different instructions on different data bases in multiprocessing). This is the second path taken in departures from von Neumann machine organization, and is illustrated by the SOLOMON computer and ILIAC IV.

One way of describing the degree of parallelism in such a machine is by the size of the individual data bases. For example, in ILIAC IV each processing element works on a 2048-word (64-bit) local memory. Clearly, overall processing capability reaches a maximum when each processing element works on a data base of only one word (assuming that the processing capability of each element remains constant). One way of providing this one-on-one processing capability is through use of associative (or content addressable) memories and processors.

2. Associative Devices

Associative memories have been generally described as a collection of data storage elements which can be accessed in parallel on the basis of data content as opposed to conventional addressing techniques. In most configurations the basic operation of such a device is the exact match search, in which all associative elements are simultaneously compared to a given word (the comparand) and response store bits are set to indicate which words in associative memory are equal to the comparand. A response resolution mechanism then supplies the control device with a list of addresses that have responded. Usually, a mask register is used to inhibit portions of the comparand, thus permitting only certain bits to be used as the search criteria.

Extending the comparand to include the ability to use previously set response bits as criteria for additional searches, as well as adding to the basic hardware of the associative memory, permits a wide range of increased capability. Single or compound searches that might be implemented include:

equal	not equal
less than	greater than
less than or equal	greater than or equal
maximum value	minimum value
between limits	not between limits
next higher	next lower.

If arithmetic and multiwrite circuitry is also included, operations such as write-on-match and add-on-match may also be made available.

An additional level of sophistication and processing capability is available through use of a variable instruction technique in which the search criteria for each syllable can be specified independently and simultaneously. (A syllable may be defined as a contiguous sequence of bits within a word.) Suppose, for example, we have a 48-bit associative machine divided (through hardware constraints) into six eight-bit syllables. Then in one cycle we might search for all words having their first syllable equal to the first syllable of the comparand, their second syllable greater than the second syllable of the comparand, etc. Let us call such a device a fixed syllable, variable instruction associative processor.

To go one step further, we might remove all hardware constraints on syllable size and place the definition of each syllable under software control, thus obtaining a variable syllable (or floating syllable), variable instruction machine.

3. Introduction to the Proposed Associative Processor

The main body of this report describes the design of a proposed associative processor, concentrating on the solid state associative chip that represents the major developmental item. For research purposes, to investigate the application of associative techniques to the problems enumerated in Section 1, one would like to have as sophisticated and flexible an associative processor as possible. With a device having every capability described in Section 2 one could apply the full range of associative techniques to each problem in order to determine realistic design criteria (in terms of required associative capability) for practical, special purpose devices to be built in the future. In addition, such a device would represent a prototype general purpose associative processor.

Against this background, we have considered many associative processor (AP) designs to solve a large number of existing naval requirements. To meet all of these requirements at a reasonable cost we propose a design for a solid state, variable syllable, variable instruction AP whose basic element (the associative chip) combines as many one-cycle searches and operations as possible (nine total), including a multiwrite and multiadd (both keyed on the current state of the response store, i.e., on the results of previous searches). Compound searches and complex operations would be controlled by microprograms located in a high-speed solid state (flip-flop) memory. Examples include adding two syllables within all words and storing the result in a third syllable, multiplying all words by the comparand, etc. (see Section 4.7). Macroinstructions (one AP microprogram corresponds to one macroinstruction) might be delivered to the AP by the main processor on an instruction-by-instruction basis. Alternatively (or in addition), complete AP programs could reside locally with respect to the AP. In the latter case locally stored AP routines could run independently while the main processor continues to work on nonassociative tasks. For example, in an air traffic control application self-contained AP programs periodically initiated by the main processor might include collision avoidance checks, flight planning routines, etc.

4. Description of the AP

Those parts of the AP that combine to produce its associative properties are shown in Fig. 1. The syllable definition register (SDR), comparand (COMP), search command one (SC1), search command two (SC2), multioperation register (MOR), associative memory output buffer (AMOB), and the associative memory address register (AMAR) are seven (flip-flop) registers under the control of a microprogram processor (MPP) that

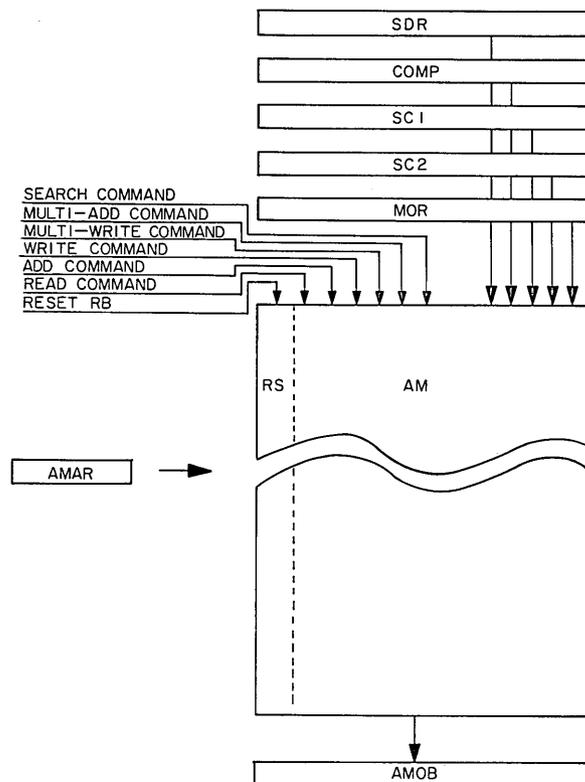


Fig. 1 - The basic components

combines with the associative memory (AM) to form the AP. Strictly speaking, the inclusion of the MPP does not increase the associative capabilities of the AM, since the seven registers in Fig. 1 could be directly under the control of the main processor. However, including the additional level of control represented by the MPP, as indicated in Fig. 2, enables frequently used complex associative operations to proceed in parallel to the continuing operation of the main processor. In such a configuration, an associative macroinstruction passed to the MPP from the main processor would initiate a microprogram located in the MPP. Also, this configuration enables us to exploit the speed of the AP by operating the MPP-AM combination with a cycle time independent of that required by the main processor.

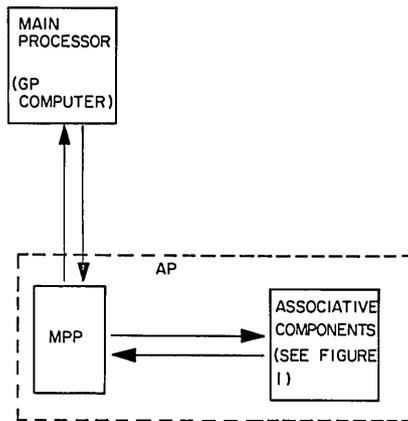


Fig. 2 - System organization

In what follows, the term "word-slice" refers to the contents exposed by any horizontal cross section through the right-hand side of Fig. 1. Similarly, a "bit-slice" refers to the contents exposed by a vertical cross section through Fig. 1.

4.1 The Syllable Definition Register

For many applications it is convenient to store a complete set of parameters $\{P_n\}$ in each AM word. For example, in an air traffic control application, each AM word might represent one aircraft by a set of parameters such as identification number, position, time, airspeed, heading, altitude, etc.

To take a general example, suppose that we are using a sixteen-bit machine and have stored five parameters in each word according to the following specification:

bits 1 - 3:	P_1 ,
bits 4 - 5:	P_2 ,
bits 6 - 8:	P_3 ,
bits 9 - 11:	P_4 ,
bits 12 - 13:	P_5 .

Bits 14 through 16 are taken up by the response store (RS), the operation of which will be described in Section 4.3. Bit positions are labeled from right to left (as shown in Fig. 3). We note that in an actual application both the AM word length and the parameter specifications would probably be considerably longer.

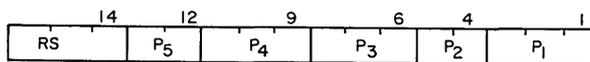


Fig. 3 - AM word format for
the example in the text

Addition and nonequal comparisons (less than; greater than) require indication of the intrasyllable limits. This is the function of the syllable definition register (SDR), which should contain a "1" in the lowest numbered (least significant) bit of each syllable. For the preceding example, the SDR would be loaded as in Fig. 4.

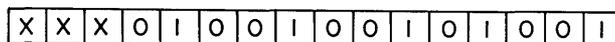


Fig. 4 - SDR configuration for
the example in Fig. 3

There are several reasons for including this variable (or floating) syllable capability. First, since we can allocate syllables according to the requirements of the specific application, we are able to make efficient use of the AM word-slice length. Second, since the SDR is under software control, we can go a step further and allocate syllables on a real time basis as a function of the data being processed. Finally (as will be seen in Sections 4.7.1 and 5.1.2.5) special purpose manipulation of the SDR results in a marked increase in the overall associative capability of the AP.

4.2 The Comparand

The comparand (COMP) contains the reference word for an associative search. Also, during a multiadd operation the COMP is the addend and during a multiwrite operation the COMP acts as the input buffer. The exact nature of each of these operations is contingent on the contents of the other control registers.

4.3 The Search Command Registers and the Associative Search Operation

When a search command is given (by raising a line from the MPP) every word in the AM is simultaneously compared to the COMP. Each of these comparisons is performed in a bit parallel fashion. There are four basic types of searches, each of which can be specified for any sequence of bits (including a sequence of one). These are:

1. greater than the COMP,
2. less than the COMP,
3. exactly matching the COMP,
4. don't care.

The selection of one of these is determined, on a bit-by-bit basis, by the contents of the SC registers. The code for this specification is given in Fig. 5. A contiguous series of either type 1 or type 2 (in the list above) carries the implication that the specified series of bits is to be considered as one (numerical) value in each AM word-slice for

SC1	SC2	TYPE OF BASIC SEARCH
0	1	GREATER THAN THE COMP
1	0	LESS THAN THE COMP
1	1	EXACT MATCH TO THE COMP
0	0	ANYTHING WILL DO (DON'T CARE)

Fig. 5 - Search specification code

purposes of the search. In the event that such a series crosses an intrasyllable division, the SDR automatically splits it into two separate series. If a particular word in the AM satisfies all criteria specified by the COMP, SC1, and SC2, the word-slice is said to respond, and a bit is set in the response store (RS) section of that word-slice. The RS is a section of each AM word-slice used to store information regarding the search history of that word-slice. We note that no SDR specification is needed for those bits in RS.

During an associative search operation, the RS section of each AM word-slice is also compared to the corresponding COMP bits. However, there are only two RS search types, namely, exact match and don't care, instead of the four types available to the non-RS section of the word-slice. The search specification code for the RS is given in Fig. 6.

SC1	SC2	TYPE OF BASIC SEARCH
0	1	DON'T CARE
1	0	DON'T CARE
1	1	EXACT MATCH TO THE COMP
0	0	DON'T CARE

Fig. 6 - Response store search specification code

In a particular search, requiring one or more matches in the RS section is equivalent to including the criteria of a previous search or series of searches. It is this capability that enables us to build complex search types (e.g., between limits) from our basic set.

The number of bits in the RS is a hardware variable as far as the present design is concerned. While two bits will suffice to accommodate any length series of ANDed searches, each search being conditional on the results of the previous search, additional RS bits are required if we also wish to maintain a record of when (i.e., between which searches) each word-slice ceased to satisfy a continuing series of searches. Thus, the RS should be made long enough to record the longest expected series of the latter type. Note that if only a count of the number of satisfied searches is required, this can be accomplished by using a syllable within each word-slice.

As was mentioned previously, when a word-slice responds to a particular search, a bit is set in the RS section of that word-slice. The particular bit to be set is specified for each search by the occurrence of the control register pattern reproduced in Fig. 7. The intersection of this bit-slice with every word-slice defines the response bit (RB) for the search being performed. We note that this combination of a "1" in the COMP with a "don't care" specification would not otherwise be useful.

Fig. 7 - Specification of response bit

COMP	1	
SC1	0	
SC2	0	

The extreme-left-hand bit in the RS is called the response resolution bit (RRB). Its output is connected to the response resolution network (RRN). The RRN is designed to supply the microprogram processor (MPP) with a list of addresses of those AM word-slices that have a "1" set in the RRB. Thus, whenever we are actually interested in working with those word-slices responding to a particular search, we should specify the RRB as the response bit for that search.

To clarify these procedures let us return to our example. Suppose the AM is filled with information as in Fig. 3. Suddenly we become interested in that set of AM words satisfying

$$A < P_3 < B,$$

$$C < P_4 < D,$$

$$E < P_5 < F.$$

If, for example, P_3 and P_4 are spatial coordinates and P_5 is time, then this is equivalent to specifying a particular two-dimensional area and a specific time period. Suppose, in addition, that the code for P_2 has been constructed in such a way that all AM words of present interest will have a "1" in bit 5 of Fig. 3. For example, if P_2 were used to indicate aircraft type in an ATC application, bit 5 might distinguish between airline and non-airline traffic.

Loading the control registers as in Fig. 8, we raise the associative search line to the AM. As a result, bit 14 is set in every word with bit 5 equal to "1" and with

$$P_3 < B,$$

$$P_4 > C,$$

$$P_5 < F.$$

Reloading the control registers to correspond to Fig. 9, we search again. The search criterion that bit 14 be equal to "1" restricts the set of possible solutions to those words in AM that have already satisfied the search criteria defined in Fig. 8. Thus, as a result of our second search, bit 16 (the RRB) is set in every word that satisfies our previous search and the criteria:

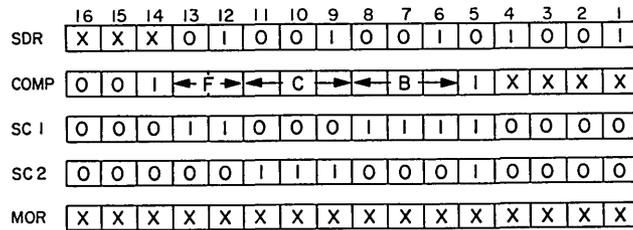


Fig. 8 - Control register configuration for the search example

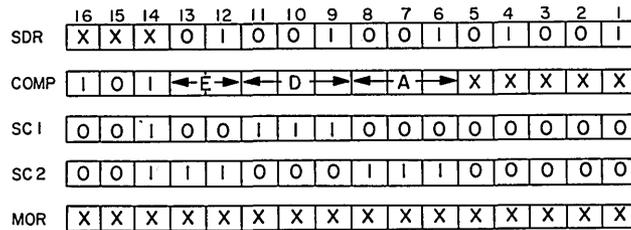


Fig. 9 - Control register configuration for the search example

$$P_3 > A,$$

$$P_4 < D,$$

$$P_5 > E.$$

This combination is equivalent to our original specification. The RRN now supplies the MPP with the addresses of the responding AM words. The total time required to set the RS according to the above criteria is only two AM cycle times (independent of AM size) plus the time it takes to fill the COMP, SC1, and SC2 before each search. This can be reduced to one cycle time by doubly listing each parameter to allow simultaneous comparison of both the upper and lower limits at an obvious increase in hardware cost.

Of the four basic search types (listed in Fig. 5) we note that only the "greater than" and "less than" search types depend on the contents of the SDR. The exact nature of this dependence is best explained in connection with the digital design of the associative chip (and will be explained in Sections 5.1.2.3 through 5.1.2.5). We mention, however, that the analysis of this dependence results in the addition of two basic search types: less than or equal, and greater than or equal, resulting in a total of six basic search types.

4.4 The Multioperation Register

The multiwrite and multiadd operations are symmetric in two respects. First, when either is initiated, those bits in the COMP, SC1, and SC2 that correspond to the RS (bits 14 through 16 in our example) search the RS with the search specification code of Fig. 6. The operation is then executed for those word-slices responding to this RS search. Thus, like the associative search, the two multioperations can be conditioned on the results of any previous set of associative searches.

The second symmetry of the multioperations concerns the multioperation register (MOR). A write or add operation for any word-slice involves that word-slice and portions of the COMP, in particular those bits of the COMP which have a "1" in the corresponding bit of the MOR. Those COMP bits with a "0" in the corresponding MOR bit are ignored during the multioperation.

We note that the search portion of either multioperation does not result in the setting of an RS bit (the RB). This is because the RS search does not extract any additional information from the AM.

4.4.1 The Multiwrite

In most cases the multiwrite operation is used to copy certain portions of the COMP simultaneously into every word in AM that has satisfied a particular combination of previously executed searches. If a particular word-slice satisfies the RS search criteria, each bit of the COMP which has the corresponding MOR bit set to "1" is copied into the word in the AM. Those bits in the AM which correspond to bits in MOR set to "0" are left unaltered. This procedure includes the RS section of every word-slice.

Returning to our previous example, suppose that we wish to reset P₁ to zero in all words selected by the searches described in Section 4.3. This is accomplished by loading the control registers as in Fig. 10 and raising the multiwrite line.

Including the RS in the multiwrite operation implements a convenient, one-cycle clear-response-store operation, depicted in Fig. 11.

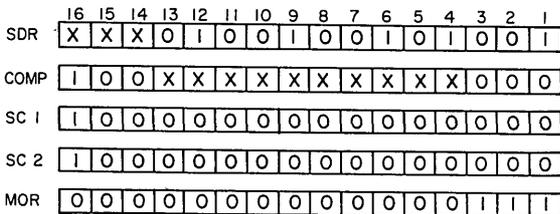


Fig. 10 - Control register configuration for the multiwrite example

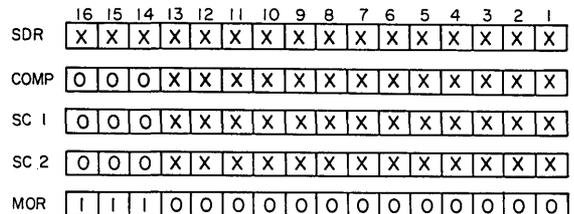


Fig. 11 - Control register configuration for a one-cycle clear-response-store operation

4.4.2 The Multiadd

When the multiadd line is raised, the COMP (masked by the MOR) is simultaneously added to each word responding to the RS search (the addition itself is bit parallel). Individual syllable integrity is maintained by operation of the SDR, which inhibits carry propagation across each intrasyllable division. The addition does not include the RS.

Continuing the above example, instead of performing the multiwrite described in the last section, we might decide to replace P₁ and P₅ according to

$$P_1 + C_1 \rightarrow P_1$$

$$P_5 + C_5 \rightarrow P_5$$

in all words selected as described in Section 4.3, where C_1 and C_5 are constants. This may be accomplished by loading the control registers as in Fig. 12 and raising the multi-add line.

In general, we can perform 2s complement integer arithmetic on a syllable-by-syllable basis (as defined by the SDR). If required, overflows can be detected by an algorithm similar to that used with a conventional 2s complement adder. In the AP this detection can be performed simultaneously for every syllable in every word-slice.

4.5 Conventionally Addressed Operations

The associative memory (AM) can also be accessed by means of the AM address register (AMAR) and a conventional address selection network. When the read line is raised, that AM word-slice whose address is located in the AMAR is gated (in parallel) onto the AM output buffer (AMOB). When the write line is raised, the comparand (COMP) is written into the AM word-slice whose address is located in the AMAR. Finally, when the add line is raised, the COMP is added (in parallel) to that AM word-slice whose address is located in the AMAR. In both the add and the write operation, the COMP is masked by the MOR as described in Section 4.4.

4.6 Simultaneous Operations

The AP has been designed so that many combinations of the operations that have been described individually can be performed simultaneously (from a programming point of view). Of the six individual operations (corresponding to the six lines into the AM) there are $\binom{6}{2} = 15$ possible combinations of two operations. To perform two operations simultaneously, the two corresponding lines into the AM are raised.

Defining each of the 15 combinations as one binary operation, we may state that every binary operation is possible, in the sense that unique and unambiguous results are obtained when any two lines are raised simultaneously. Of course, not all of the binary operations are equally useful. In general, the most useful will be those binary operations which are a combination of two "orthogonal" single operations. Two operations may be said to be "orthogonal" if they are functionally unrelated, i.e., if the two have no effect on each other.

The orthogonality relationships between the six single AM operations are expressed in matrix form in Fig. 13. That the matrix is valid will be apparent after we present the digital design of the AM, starting in Section 5. For the present, however, the matrix may be understood by applying the following principle: Single operations are of two types,

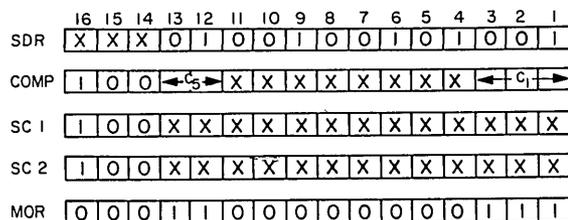


Fig. 12 - Control register configuration for the multiadd example

Suppose a binary operation is composed of the two single operations O_1 and O_2 , which normally execute in times t_1 and t_2 respectively. In Section 5.3.6 it will be shown that the binary operation will require at least as much time as the larger of t_1 or t_2 but never as much as five gate delays more than this. Therefore, the inclusion of binary operations gives an additional processing speed advantage to the AP.

An analysis similar to the above could be carried out for all combinations of three primary operations. In general, there are

$$\sum_{n=0}^{n=6} \binom{6}{n} = 2^6 \quad (1)$$

possible simultaneous AM operations of any number that can be arrived at by the above procedure (one might have to consult the digital design to determine the outcome of some of these). This suggests that one six-bit syllable in each instruction of the MPP be reserved as an AM instruction code. The output of the corresponding flip-flops of the MPP instruction register could be connected directly to the AM control lines. Other microprogram syllables would specify (directly or indirectly) the control register contents for that AM instruction, etc.

4.7 Complex Arithmetic Operations

In Section 3 we indicated that the flexibility of the AP would be greatly increased by making complex arithmetic and logical operations available as stored microprograms in the MPP. To illustrate this capability, we offer the following four examples.

4.7.1 Logical Operations

The three most often used logical operations — AND, OR, and EXCLUSIVE OR — are easily implemented as AP operations by proper manipulation of the control registers.

If A and B are two binary numbers and we wish to replace B according to

$$A \text{ OR } B \rightarrow B,$$

we may proceed as follows: For each bit in A, if that bit equals "1" we write this value into the corresponding bit in B. If a bit in A equals "0", we leave the corresponding bit in B unaltered. This procedure modifies B so that it is now equal to the logical OR of A and B. Thus, if we load both the COMP and the MOR with A and then perform a multiwrite, those word-slices selected by the RS search are modified to become the logical OR of the COMP and the word-slice. To take a particular example, let $A = 110010$. We load the control registers as in Fig. 15 and raise the multiwrite line. As a result, the last six bits of each word-slice selected by the RS search now represent the logical OR of A and their previous contents. The RS search criteria have been left blank in Fig. 15 to indicate that the word-slices can be selected for the OR operation according to any specification

Continuing the above example, suppose we wish to replace B according to

$$A \text{ AND } B \rightarrow B.$$

We may proceed as follows: For each bit in A that equals "0" we write this value into the corresponding bit in B. For those bits in A that equal "1", we leave the corresponding

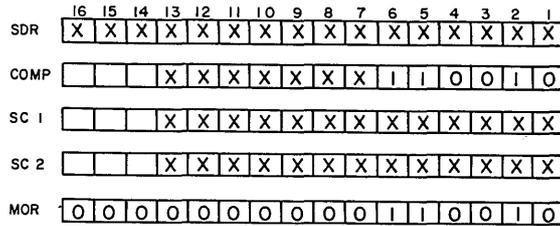


Fig. 15 - Control register configuration for the logical OR example

bit in B unaltered. This procedure modifies B so that it now represents the logical AND of A and B. We can accomplish this in the AP by loading the COMP with A, loading the MOR with the bit-by-bit complement of A, and raising the multiwrite line (Fig. 16). As a result, the last six bits of each word-slice selected by the RS search now represent the logical AND of A and their previous contents.

Finally, suppose we wish to replace B according to

$$A \text{ EXCLUSIVE OR } B \rightarrow B.$$

Since, for any two bits, their EXCLUSIVE OR is simply the sum of the two bits, we may accomplish this logical operation by loading the control registers as in Fig. 17 and raising the multiadd line. Note that the SDR specification results in the inhibition of all carries between bit-slices. As a result, the last six bits of each word-slice selected by the RS search now represent the logical EXCLUSIVE OR of A and their previous contents. We note that since both the AND and the OR involve an AM write operation, they may be performed simultaneously on different syllables, specifying each syllable according to the above.

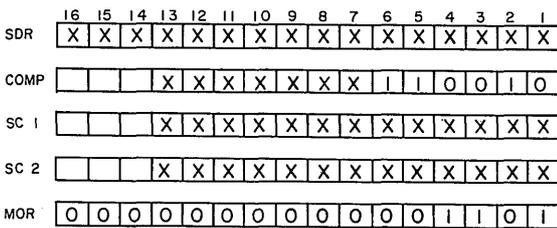


Fig. 16 - Control register configuration for the logical AND example

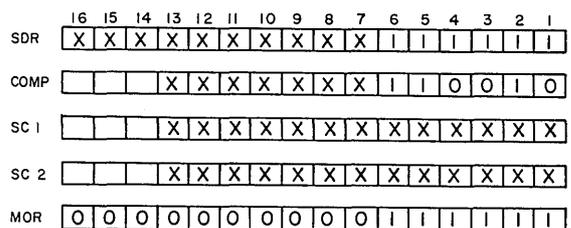


Fig. 17 - Control register configuration for the logical EXCLUSIVE OR example

4.7.2 Multiplication of All AM Words by the Comparand

The multiplication of all AM words by the COMP is based on the algorithm flow-charted in Fig. 18. Since all multiadds can be conditioned on the results of any combination of previous searches this capability is already built into the multiplication (specifically, at operations 3 and 5 of Fig. 18). This algorithm will work for multipliers that are nonzero in one syllable only (there is no preferred syllable).

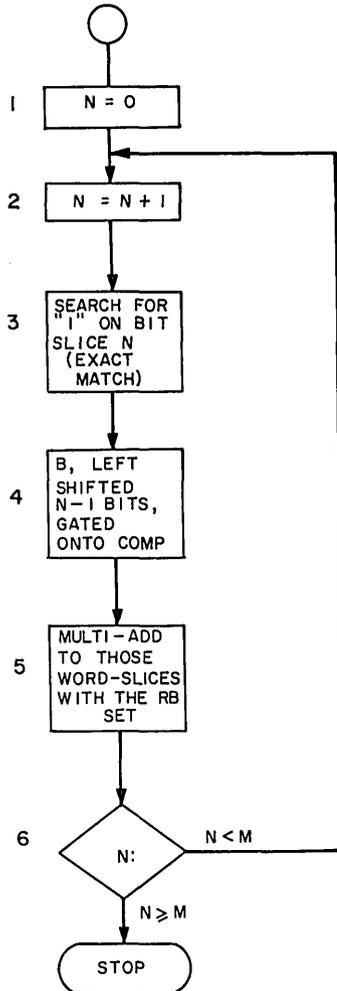


Fig. 18 - Multiplication of all AM words by B. M is equal to the width, in bits, of the largest number in AM to be multiplied by B.

4.7.3 Addition of Two Syllables Within Every Word

The addition of two syllables within every word is based on the fact that the addition of two n-bit words, A and B, may be performed in a serial fashion by expressing one word in the form

$$A = \sum_{i=0}^{n-1} c_i 2^i \quad (2)$$

and then adding A to B on a term-by-term basis:

$$A + B = B + \sum_{i=0}^{n-1} c_i 2^i. \quad (3)$$

Suppose we wish to add the syllable of bits 3 through 5 to the syllable of bits 8 through 12 within every word-slice of AM. As our first step, we load the registers as in Fig. 19 and raise both the associative search and the multiadd lines. For each word-slice this operation adds a "1" to B if A has a "1" in its least significant bit. Next, the registers are loaded as in Fig. 20 and the operation is repeated. This time, a "10" (binary) is added to B if A has a "1" in its second bit. The procedure is iterated for each bit in A. This completes the addition according to the algorithm described.

We note that adding two syllables and storing their sum in a third syllable ($A + B \rightarrow C$) can be easily accomplished by writing zeros into C, adding A to C, as above, and then adding B to C. Since, throughout, we may have required that an RS bit be set (e.g., bit 14) this entire procedure is word-slice selective according to the results of previous searches.

4.7.4 Multiplication of Two Syllables Within Every Word

The algorithms of the previous two sections can be combined, enabling us to multiply two syllables together and store their product in a third syllable ($A \cdot B \rightarrow C$). Specifically, operation 5 in Fig. 18 is replaced by the algorithm of the last section. The entire procedure may be described as follows: To begin with, zeros are written into syllable C. The lowest order bit-slice of syllable B is copied into an RS bit-slice (by performing an exact match search with a "1" in the COMP bit corresponding to the lowest order bit-slice of B). Syllable A is added to C (as in the last section) in every word with the RB set. Next, the second lowest order bit-slice of syllable B is copied into the RS bit-slice. Syllable A, left shifted by one bit, is added to C in every word with the RB set. Syllable A is shifted left, virtually, by one bit merely by displacing by one bit the correspondence between bits in A and C as specified in the algorithm of the last section. Specifically, the sum of A (left shifted by one bit) and B is given in the notation of Eqs. (2) and (3) by

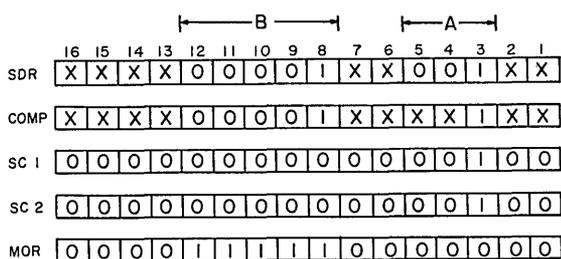


Fig. 19 - Control register configuration for the first step in addition of parameter A to parameter B within all AM words

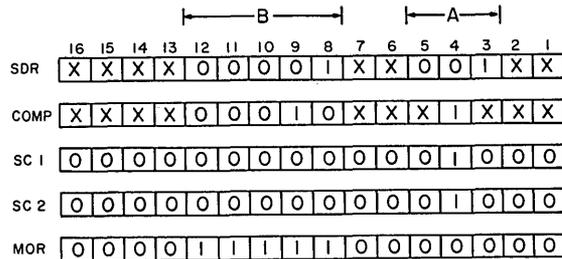


Fig. 20 - Control register configuration for the second step in the addition initiated by the operation described by Fig. 19

$$A \text{ (left shifted by one bit)} + B = B + \sum_{i=0}^{n-1} c_{i-1} 2^{i-1} \quad (c_{-1} = 0) \quad (4)$$

Thus, in Fig. 19, the correspondence would be between bit-slices 3 and 9, instead of between bit-slices 3 and 8 as shown for the unshifted example.

As in all of the AP operations that we have been describing, perhaps the most significant advantage lies in that their execution times are all independent of AM size. That is, whether we multiply only one set or many thousands of sets of syllables together bears no relation to the operation time. In addition, the operations involve simultaneous logical word-slice selection, permitting operation only on those words that satisfy a given set of criteria. This flexible logic capability is available without any penalty in execution time.

We note that the logical operations described in Section 4.7.1 can also be performed between syllables within the same word-slice by a method analogous to that of intersyllable addition. Adding these to the other intersyllable operations described, we obtain a set of operations that actually result in a small processor within every word, thereby giving the AP unprecedented logical and computational power.

5. Design of the Associative Processor

The key design element of the AP is the integrated circuit chip which forms its basic component. Each chip contains an integral number of AM bits. The AM is constructed by horizontally and vertically stacking this chip. The RS section can be formed by adding external logic to the basic chip and stacking this combination. Alternatively, the RS can be implemented as a separate integrated circuit chip.

The number of bits per chip N is a variable in the following design. As N increases, so does the number of gates and control lines per chip. The limit can be determined on purely technical grounds.

5.1 The Chip

The design to be presented requires $36N + 2w$ gates and $6b + 9w + 3$ control lines per chip, where b is the number of bit-slice sections, w is the number of word-slice sections, w is the number of word-slice sections, and $N = bw$. As the number of gates

available in a particular technology increases (MSI and LSI), it becomes increasingly important to optimize b and w so that the total number of control lines is minimized. This may be accomplished by setting $w = 2b/3$.

To simplify the following discussion, we consider a chip which contains a number of contiguous bits from the same word-slice (that is, $w = 1$ and $b = n$). A schematic of one bit in this associative chip is shown in Fig. 21. The various control lines are summarized in Fig. 22. (The clock, power, and ground lines are not included in Fig. 21.)

The circuit in Fig. 21 can likely be further minimized. Also, in terms of another canonical logic operation (NOR) or some technologically possible combination of operations, it may be possible to arrive at lower gate and control line requirements. Therefore, it is convenient to specify the chip design as a set of Boolean equations. Such a specification is given in Appendix A.

5.1.1 Addition

A NAND logic implementation of the addition scheme is depicted in Fig. 23. The horizontal dashed lines represent limits of the chip. The dotted vertical lines represent the division of the chip into individual bits. Only one bit is shown in Fig. 23. The extreme left and right bit division lines become chip limits as well, with the width in bits to be determined as was discussed.

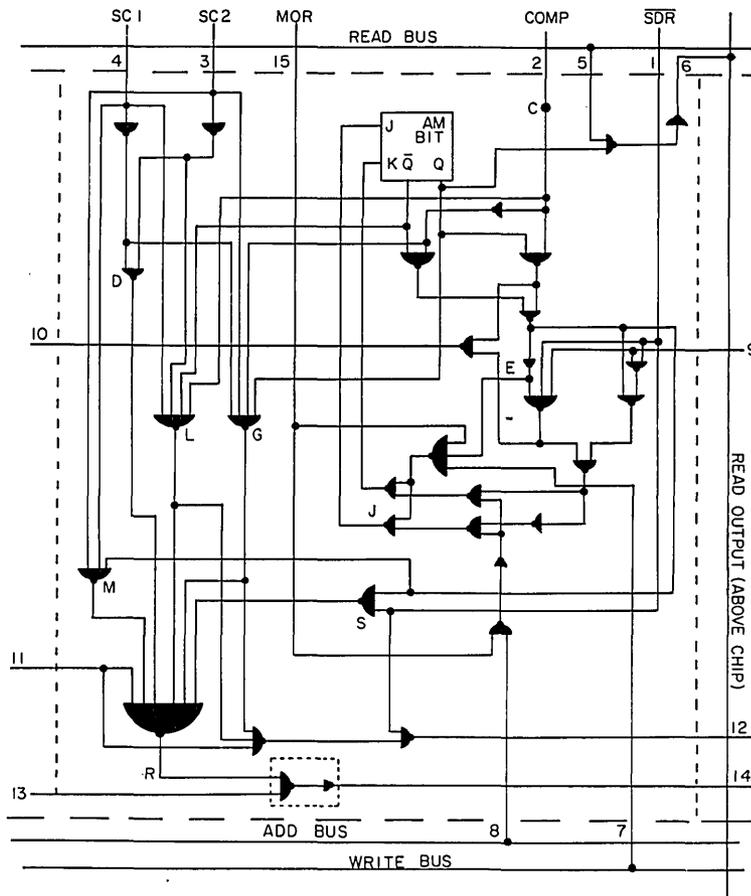


Fig. 21 - NAND logic diagram of one AM bit

NO.	DESCRIPTION OF THE LINE	Occurrence
1	complement of the SDR (input)	once per bit-slice
2	from COMPARAND	once per bit-slice
3	from SC1	once per bit-slice
4	from SC2	once per bit-slice
5	connection from read bus	once per word-slice
6	flip-flop output when read bus is high	once per bit-slice
7	connection from write bus	once per word-slice
8	connection from add bus	once per word-slice
9	carry from bit $n - 1$	once per word-slice
10	carry to bit $n + 1$	once per word-slice
11	set all remaining responses in syllable (in)	once per word-slice
12	set all remaining responses in syllable (out)	once per word-slice
13	response line from more significant bits	once per word-slice
14	response line output	once per word-slice
15	MOR input	once per bit-slice
16	clock (not shown)	once per chip
17	power (not shown)	once per chip
18	ground (not shown)	once per chip

Fig. 22 - Control lines of the circuit in Fig. 21

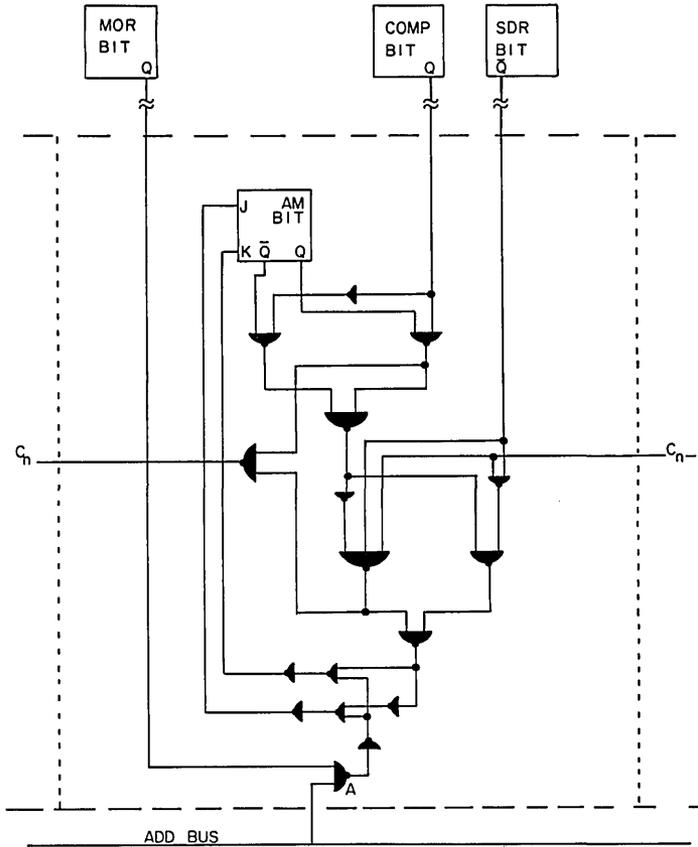


Fig. 23 - NAND logic diagram of the AM addition circuitry (one bit shown)

The square box within the chip represents a J-K clocked flip-flop (six gates have been allowed for this flip-flop). The truth table requirements for this flip-flop are given in Fig. 24. The states J, K, Q_n , and \bar{Q}_n (output and its complement) are states at the leading edge of the clock pulse. Q_n changes to Q_{n+1} at the trailing edge of the clock pulse. Q_{n+1} remains unchanged at least until the trailing edge of the next clock pulse. The use of a clocked flip-flop eliminates all race-time problems and is the source of our ability to include so many functions on one chip. Operationally, race-time problems are eliminated if we make sure that the next clock pulse after the initiation of an operation does not come before all J-K inputs have reached a steady state. This procedure may be compared to that of measuring the water level in a complex plumbing system. After opening or closing a set of valves that are part of system, level ambiguity is eliminated if we do not make our measurement until enough time has elapsed for the water to reach its natural level.

Addition is keyed to the state of the add bus. The bus will be high if this word-slice has been selected for addition (either by conventional addressing or by response to the

J	0	1	0	1
K	0	1	1	0
Q_{n+1}	Q_n	\bar{Q}_n	0	1

Fig. 24 - Truth table for the J-K flip-flop

RS search of the multiadd operation). If the add bus is high, the COMP (masked by the MOR) is added in a bit parallel fashion to the AM word-slice involved. The input from the MOR inhibits the add bus input unless, as required by Section 4.4, the MOR bit is "1" (at point A in Fig. 23). The key to the add operation is the logical EXCLUSIVE OR:

$$a \oplus b = a \cdot \bar{b} + \bar{a} \cdot b .$$

If a_n and b_n are the contents of the n th bit in the COMP and AM word-slice respectively, let s_n be the contents of this bit in AM after the next clock pulse. Then,

$$s_n = (a_n \oplus b_n) \oplus c_{n-1} ,$$

where c_{n-1} is the carry from bit $n-1$. The carry (from bit n to bit $n+1$) is given by

$$c_n = a_n \cdot b_n + c_{n-1} \cdot (a_n \oplus b_n) .$$

A truth table for all values of a_n , b_n , and c_{n-1} is given in Fig. 25. The reader may verify that the circuit in Fig. 23 satisfies this truth table if the add bus and the MOR input are both high.

a_n	b_n	c_{n-1}	s_n	c_n
0	0	0	0	0
1	0	0	1	0
1	1	0	0	1
1	1	1	1	1
0	1	0	1	0
0	1	1	0	1
0	0	1	1	0
1	0	1	0	1

Fig. 25 - Addition truth table

Recalling that the syllable definition register (SDR) contains a "1" in the least significant bit of each syllable, we see that the SDR acts to inhibit the propagation of a carry from the end of one syllable to the beginning of the next. Consequently, any syllable overflow will result in loss of the most significant bits.

5.1.2 The Associative Search

Each type of associative search is performed in a bit parallel manner. The "exact match" and "don't care" searches are bitwise independent; i.e., for each bit the search depends on the contents of that bit only. The "greater than" and "less than" searches are similar to the addition (described in the preceding section) in that the outcome of the search depends to some extent on the contents of the other bits in the syllable. (Addition depends on the less significant bits. These searches depend on the more significant bits.)

Referring to the one-bit schematic in Fig. 21 (Fig. 23 corresponds approximately to the upper-right-hand corner of Fig. 21), the state of the line at the bottom left marked "R" indicates the search response of this bit. If R is high, this bit satisfies its particular search criterion. If it is low, the criterion has not been met. The response of the previous bit (the one on the left) is connected to input line 13. Clearly, line 14 will be high if and only if both line 13 and R (response of this bit) are high. If these conditions are not satisfied, and line 14 is low, every such line 14 in all bits standing to the right will be forced to the low state. Hence, line 14 from bit 1 of the word-slice will be high if and only if every bit in the word-slice has satisfied its search criterion.

To avoid unnecessary gate delays, in actual practice the two gates near R that are surrounded by the dotted box should be included only in the extreme-right-hand (least significant) bit of each chip. If each chip is to contain N bits, then the first gate within the dotted box would be an (N+1)-input NAND gate, one input for every R line in the chip plus one from output line 14 of the previous chip (input line 13 of this chip).

5.1.2.1 The "Don't Care" Search

If the "don't care" criterion has been specified, the output of gate D (Fig. 21) will be in the low state. Following the line down we see that this condition will force R high, as required.

5.1.2.2 The Exact Match Search

Since our method of addition involves the bitwise logical EXCLUSIVE OR of the COMP and AM, it is advantageous to make as much use of this logical function as possible. If A and B are two single-bit binary numbers, we note that $A \oplus B$ is zero if and only if $A = B$. Thus, $\overline{A \oplus B} = 1$ if and only if $A = B$. In Fig. 21 the state of point E corresponds to the EXCLUSIVE OR of the COMP and the bit under consideration. Tracing all inputs to gate M (lower left), we see that if the complement of the EXCLUSIVE OR is high and if the exact match code has been specified, then the output of gate M will be low and R will be forced high, as required.

5.1.2.3 Search for Larger Elements

Whether an AM syllable is larger than the corresponding syllable in the COMP might be determined as follows: We look first at the most significant bit of the syllable in AM. If this bit is larger than the corresponding bit in the COMP, we know that the search has been successful and that we can disregard the remaining bits of the syllable. But if the bit in the COMP is larger, then we already know that the syllable does not satisfy the search. If the two bits are equal, we must repeat this procedure for the next most significant bit, etc. If every bit is equal, then the two syllables are equal and the AM syllable does not satisfy the search, since it is not greater than the corresponding syllable in the COMP.

The algorithm used in the associative chip is derived from the preceding, with modifications that enable the comparison to be made in a bit parallel manner. Thus, the algorithm flow-charted in Fig. 26 is executed simultaneously in every bit. This algorithm makes use of the above procedure and the fact that if R remains low in only one bit the entire word-slice will not satisfy the search.

To implement this algorithm we again make use of the following: If A and B are both single bit binary numbers we note that

$$\begin{aligned} \overline{A \cdot B} &= 0, \text{ if } A > B, \\ &= 1 \text{ otherwise.} \end{aligned}$$

The output of gate G in Fig. 21 is equivalent to this function with $A = \text{AM bit}$ and $B = \text{COMP bit}$, if a "greater than" search code has been specified for this bit. Following the line down, one sees that if the output of gate G is low, R is forced high, as required. In addition, unless this is the least significant bit of the syllable (indicated by input line 1, the SDR), output line 12 is forced low. Output line 12 is connected to input line 11 of the

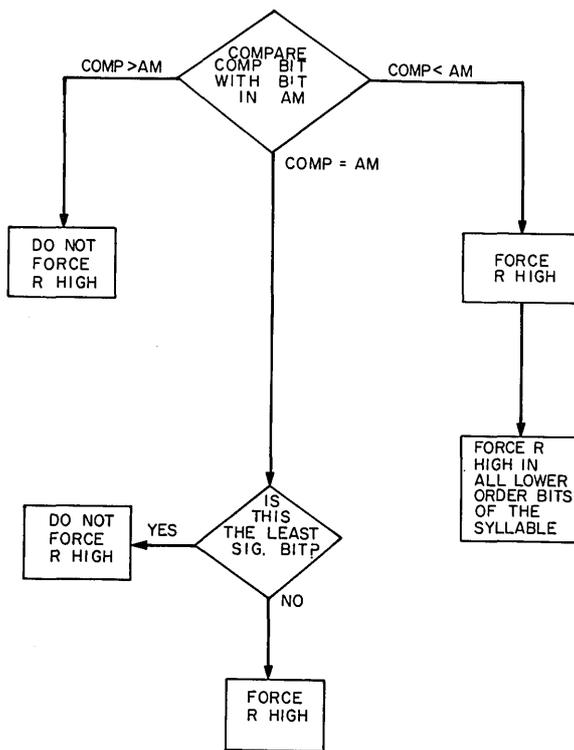


Fig. 26 - Algorithm for the "greater than" search

next bit. If line 11 is low, then R is forced high. Output line 12 is again forced low, if this is not the least significant bit of the syllable in question. This completes the right-hand branch of Fig. 26. Also, if the AM bit and COMP bit are equal, R is forced high unless the AM bit is the least significant of the syllable (point S). Finally, we see that if $A < B$, no action is taken and R stays low (unless a more significant bit has satisfied $A > B$). In this manner each branch of the flow chart in Fig. 26 is satisfied.

5.1.2.4 Search for Smaller Elements

The procedure used to search for syllables in the AM smaller than the corresponding syllable in the COMP is exactly symmetric to that of the search for larger syllables. Thus, in Fig. 26 the left- and right-hand branches are simply reversed. Likewise, the implementation makes use of the fact that

$$\overline{A \cdot B} = 0, \text{ if } A < B,$$

$$= 1 \text{ otherwise.}$$

where again A is an AM bit and B is the corresponding COMP bit. The output of gate L is low if this function is zero and if the "less than" code has been specified for this bit. Tracing the output of gate L we see that R is forced high if this output is low, as required.

5.1.2.5 The SDR and Its Use in the Associative Search

A complete understanding of the operation of the associative chip can lead to many sophisticated programming techniques through careful manipulation of the five control registers. These techniques can be used to increase the flexibility of the AP.

For example, suppose a parameter of interest normally will require not more than M bits. Let us instead reserve $M + 1$ bits for this parameter (by means of the SDR) but use only the M most significant bits of the syllable. Specifically, suppose we reserve bits 6 through 12 (inclusive) for a six-bit parameter and store the parameter in bits 7 through 12. Now, if we wish to search for all values less than or equal to a particular value, this may be accomplished by the control register specification in Fig. 27, which will set bit 14 of responding word-slices.

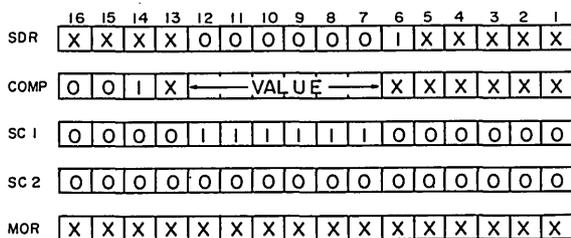


Fig. 27 - Control register configuration for "less than or equal to" search example

If we become interested in a strict "less than" search this may be accomplished by replacing bit 7 of the SDR with a "1". Clearly, we can similarly introduce the "greater than or equal to" search. In general, if one extra bit is reserved for every syllable we can consider these two new searches as part of our basic set. Distinguishing between a "less than" and a "less than or equal to" or between a "greater than" and a "greater than or equal to" is accomplished through manipulation of the SDR.

5.1.3 The AM Read

AM read operations are controlled by the read bus of each word-slice. The read bus is connected to the chip through input line 5. If the read bus is high, the output of each AM bit is connected to output line 6. External to each bit, output line 6 is connected to the bit-slice read line. Each bit in the bit-slice is connected in parallel to this line, which terminates at the bottom of the AM as an input to one bit of the AMOB. When the read line to the AM is raised the read bus of that word-slice specified by the AMAR is raised and this word-slice is written in parallel into the AMOB (Fig. 1).

5.1.4 The AM Write

AM write operations are controlled by the write bus of each word-slice. From the point of view of the associative chip, it does not matter whether the write command was initiated by a conventional write or a multiwrite.

The write bus of a particular word-slice is connected to each chip through input line 7. If this line is raised, each bit executes the following algorithm: Suppose A and B are two single-bit binary locations and we wish to write the contents of B into A . Recalling that $A \oplus B = 1$ if and only if $A \neq B$, we may use the result of this operation (which is in the chip already) to write B into A by complementing A if $A \neq B$, otherwise doing nothing.

Thus, if the write bus is high and the corresponding MOR bit is "1", the COMP bit is written into the AM bit (the change, if any, will occur on the trailing edge of the next clock pulse). In Fig. 21 the circuit implementation may be verified by tracing the input lines 7 and 15 as well as the line from point E. (The J-K input code is given in Fig. 24.)

5.1.5 Simultaneous Operations

As mentioned in Section 4.6, each of the 15 binary operations is possible, in the sense that unique and unambiguous results are obtained when any two lines are raised simultaneously. We now see that this fact derives from use of the clocked J-K flip-flop (discussed in Section 5.1.1).

The relationships described by the binary operation orthogonality matrix (Fig. 13) derive directly from the digital design of the associative chip (Fig. 21). For each pair of single operations that are said to be orthogonal, the data paths followed within the associative chip are completely independent. For those operations which are not orthogonal, the individual data paths interact within the chip. For example, according to Fig. 13, the conventional write and add, as well as the multiwrite and multiadd, are not orthogonal. It was stated in Section 4.6 that this results from the fact that both operations are simultaneously attempting to change the output of the AM bit, perhaps to different values. Referring now to Fig. 21, we see that individual data paths followed by the write and add operations interact at the two NAND gates surrounding point J (each data path is one input). Configured as in Fig. 21 these two gates operate so that the J and K inputs are both high if either the write or the add operation tries to set the flip-flop to "1". Thus, as mentioned in Section 4.6, the result of binary add/write operation, as well as the binary multiadd/multiwrite, is the logical OR of the component operations.

5.2 The Response Store

Throughout the design of the associative chip the requirements of the response store (RS) elements of the AM were kept in mind so that the RS could be constructed by adding a minimum amount of logic to this chip. The associative chip is thus the primary unit in the RS, and any integral number of these represent a convenient length for the RS. If a particular application requires a RS length that does not correspond to an integral number of chips, the bits left over may be permanently ignored by grounding their input lines 3, 4, and 15. For each bit in the RS section of the word-slice, input line 1 should be permanently grounded. This automatically divides the RS into a series of one-bit syllables.

A NAND logic implementation of a two-bit RS (built around a two-bit associative chip) is depicted in Fig. 28. This design requires the addition of $7M + 23$ NAND gates, where M is the number of bits in the RS. (One expects that the circuit in Fig. 28 can be further minimized.) As mentioned in Section 5.1, it is useful to specify digital designs as a set of Boolean equations. A set of equations describing the RS operations is given in Appendix B. We note that the combination of Appendixes A and B forms a complete description of the AM design.

Depending upon the size of the AM to be built, the additional RS logic may be individually wired for each word-slice or implemented as a second integrated circuit chip. In the design presented here, only one word-slice section per chip can be modified into RS. The additional RS logic is of two functional types:

1. Referring to Fig. 23, that section of logic to the right of the associative chip is independent of RS length and is designed to drive the RS and the various buses according to the operational requirements described throughout Section 4.

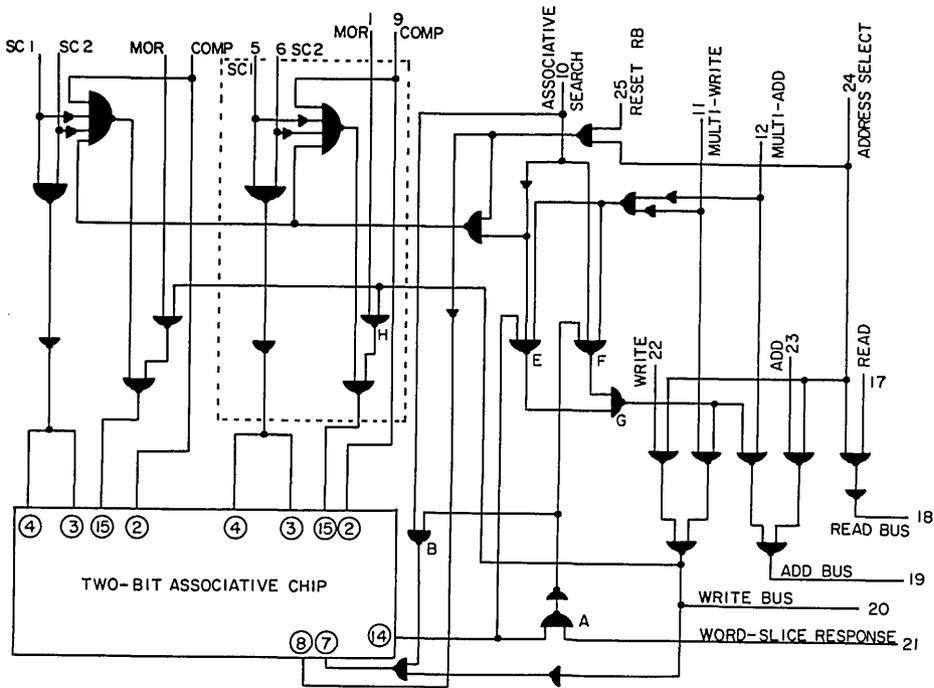


Fig. 28 - NAND logic diagram of a two-bit response store

2. The logic network above the associative chip and enclosed in dotted lines appears once for every bit in the RS. This network arranges the bit inputs to the associative chip to induce the differences in behavior between a bit in RS and other bits in the rest of the word-slice, again as described throughout Section 4.

The operation of the additional logic in Fig. 28 is most easily described by considering individually the various AM operations.

5.2.1 The RS and the Associative Search

We recall that whenever the RS is searched (as part of a number of operations) the search specification code, given in Fig. 6, is different from the code satisfied by non-RS sections of the word-slice, which is given in Fig. 5. Since the associative chip was designed to satisfy the latter, one of the functions of the additional RS logic is to effect a transformation between these codes. By tracing the SC1 and SC2 inputs of a RS bit (inputs in the code of Fig. 6), one can see that chip inputs 3 and 4, what the chip sees as SC1 and SC2, give the proper specification when interpreted as in Fig. 5.

From Section 4.3 we recall that whenever the associative search line to the AM is raised, a bit may be set in the RS section of every word-slice satisfying that search. The particular bit to be set is specified by the occurrence of the command register pattern shown in Fig. 7. Referring to the bit-associated network in Fig. 28 (within the dotted lines), this requirement is implemented by the four-input NAND gate near the top. Thus, if the associative search line is high and if the control register pattern for that bit is that of Fig. 7, the MOR input to the associative chip is raised. Unless a write operation is being performed, the rest of the network inhibits the line from the actual MOR.

Therefore, if the associative search line is raised, and the write or multiwrite lines are not, the only bits with a nonzero MOR input (chip input 15) will be those with a control register pattern as in Fig. 7.

Turning to the RS drive network, we see from Fig. 28 that the results of the RS search is combined with the response of the rest of the word-slice (output 14 of bit 1) at gate A. Therefore, gate B will have a low output if and only if the associative search line is high and the entire word-slice (including the RS) satisfies all search criteria. A zero output from gate B will result in a raised chip input 7, which is the "write" input of the associative chip. Thus, any bit with a MOR input of "1" will have the COMP written into it. Combining this fact with the results of the previous paragraph, we see that the response bit (RB), as specified according to Fig. 7, will be set in every word-slice satisfying all search criteria, as required by Section 4.3.

5.2.2 The RS and the Multioperations

We recall from Section 4.4.1 that when the multiwrite line is raised alone (or from Section 4.4.2 that when the multiadd line is raised alone), the COMP, masked by the MOR, is written (or added) into every word-slice satisfying search criteria on the RS section. Referring to Fig. 28, gate E will have a zero output if either the multiwrite or the multiadd line is raised, if the associative search line is not raised, and if the RS portion of the word-slice satisfies its search criteria (which can be seen by tracing chip output 14). If the output of E is zero, then the output of gate G is forced high. If the output of G is high and the multiwrite (or multiadd) line is high, then the write bus (or add bus) is raised as required in Section 4.4.1 (or 4.4.2).

We note that if the write bus is high, the write input (input 7) of the RS chip is also raised and MOR the contents are gated into the chip (see gate H). This satisfies the requirement that the multiwrite be extended into the RS. It is this provision that permits the "clear response store" operation depicted in Fig. 11.

5.2.3 The Conventional Read, Write, and Add

In the conventional read, write, and add operations, word-slice selection is specified by the AMAR. The address in the AMAR is decoded so that the address select line of the corresponding word-slice in AM is raised. Any one of the conventional address selection techniques may be used here. All of them trade off the primary parameters of speed, fan-in, fan-out, and circuit complexity (diode count). This tradeoff can be optimized only if one has a small range of memory sizes in mind.

From Fig. 28 we see that if the address select line of a word-slice is raised and either the write, add, or read line to the AM is raised, then the appropriate bus of this word-slice (write bus, add bus, or read bus) is forced high. As in the multiwrite, the write operation includes the RS bits. The read and add operations do not.

5.2.4 The Response Bit Complement Operation

When the RB complement line to the RS is raised, the RB (as specified according to Fig. 7) of that word-slice whose address is in the AMAR will be complemented. This operation is primarily used to reset the response resolution bit (RRB) as will be described in Section 5.3.

Referring to Fig. 28 we see that if the RB complement line and the address select line are both raised, the bit-associated networks will operate as described in Section 5.2.1. Also, input 8 (the add input) will be raised. Since the RB is the only bit with a nonzero MOR input, the COMP will be added to the RB alone. As the RS is divided into a series of single-bit syllables (as stated in the first paragraph of Section 5.2) this results in complementing the RB.

5.2.5 The RS and Simultaneous Operations

The logic described so far is not capable of handling the situation where either the multiadd or the multiwrite line is raised at the same time as the associative search line (as specified in Section 4.6). The reason for this is that the add (or write) bus can no longer be raised solely on the condition of the RS search response. Instead, the add or write is to be conditioned on the combined response of the RS and the rest of the word-slice (as in the associative search). This condition is satisfied by the combined logic of gates A, F, and G (Fig. 28). We note that in the case of a simultaneous search and multiwrite operation, the COMP will be written into those RS bits with a "1" in the MOR as well as into those with the search command pattern of Fig. 7.

All other simultaneous operations (given in Fig. 13) are adequately handled by the RS logic of Fig. 28. In general, the actions of the individual operations that make up the simultaneous operation are executed independently.

5.3 Response Resolution Network

The response resolution network (RRN) is designed to supply the microprogram processor (MPP) with a list of addresses of those AM word-slices that have a "1" set in the response resolution bit (RRB), as was described in Section 4.3. An alternate approach might supply the MPP with the responses themselves instead of their addresses. However, since information storage in the AM is relatively expensive, it is advantageous to use the absolute address of a word-slice as a pointer to auxiliary files. It is possible to store the address within the word-slice itself, so that it is read out with the response. However, with the RRN of the present design this is generally more expensive than decoding the address, as will be shown.

As mentioned in Section 4.3 the RRB is the extreme-left-hand bit in the RS. Its output is connected to the RRN by permanently raising the read input (input line 5) into the RS chip containing the RRB. Output line 6 of the RRB is then connected to the RRN. Therefore, when we are interested in actually working with those word-slices that have responded to a set of associative searches, we should specify the RRB as the RB of the last search.

Specifically, the RRN operates as follows: When a "get response" signal is passed to the RRN from the MPP, the RRN gates onto the AMAR the address of the lowest addressed word-slice with the RRB set. By simultaneously raising the AM read and RB complement lines, the MPP can read out the contents of that word-slice and at the same time reset its RRB to "0". A subsequent "get response" signal to the RRN will gate the address of the next higher responding word-slice onto the AMAR, etc. In this manner, the MPP can obtain a full list, both of the responses and their absolute addresses.

One of the RRN outputs is a line that goes high if at least one of the RRB's is set. This indicates to the MPP, directly after a search, whether any word-slice has responded and also indicates when the last element in the response list has been extracted by the above method. Clearly, knowledge, before reading out this list, of the total number of

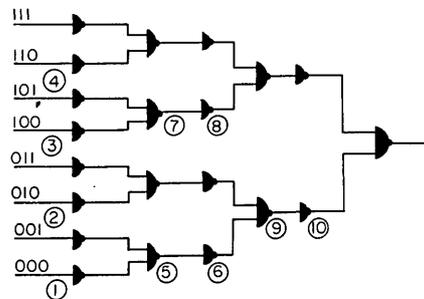
AM responses would be of great value to the MPP. While obtaining an exact number of responses would take as much time as reading out the full list, an approximate number is quickly available by passing all the inputs to the RRN (the RRB outputs) through a summation amplifier and performing an analog-to-digital conversion on the resulting voltage.

A number of RRN's have been proposed for early model associative devices. Almost all of them have performed some sort of linear search. For these older equipments a linear search was adequate due to the relatively slow associative search times. However, in the present design a linear RRN would severely degrade the performance of the AP, so that we must match its fast associative search time with a faster than linear RRN.

A symmetric (graph theoretic) tree has the property that the height of the highest branch is equal to the logarithm of the number of branch tips. One would therefore like to construct such a tree (out of two-input NAND gates) whose root is high if any response store bit is high. To resolve the ambiguity between multiple responses, one must climb the tree, pruning off the branch associated with a higher address whenever an ambiguity is found. When one reaches a branch tip (RRB), the resulting subgraph defines a unique chain connecting the base of the tree (any response) with the lowest addressed word-slice with the RRB set.

A circuit whose output is high if at least one RRB is set is depicted in Fig. 29 for an eight-word AM. Each of the eight inputs is to be connected to the RRB of the word-slice with the indicated address. If the output of this circuit is high and if a "get response" signal is generated by the MPP, we must decode this tree to determine the lowest addressed word-slice that has responded. This is the function of the circuits in Fig. 30, each of which decodes one bit of the required address. Thus, the output of the circuit in Fig. 30a is the most significant bit of the lowest addressed word-slice with the RRB set, etc. The output of each of these circuits is gated into the corresponding bits of the AMAR if the "get response" line (not shown) is raised by the MPP.

Fig. 29 - Primary circuit of the base 2 response resolution network



The gate requirements of these circuits may be calculated as follows: For an AM of 2^N words, the basic circuit requires

$$3(1 + 2 + \dots + 2^{N-1}) = 3 \sum_{i=0}^{i=N-1} 2^i = 3(2^N - 1) \tag{5}$$

gates. Decoding the most significant bit of the lowest responding address requires no additional gates. The second most significant bit requires $1 + 3(1)$ gates. The third requires $1 + 3(1 + 2)$. In general, the Mth most significant bit requires

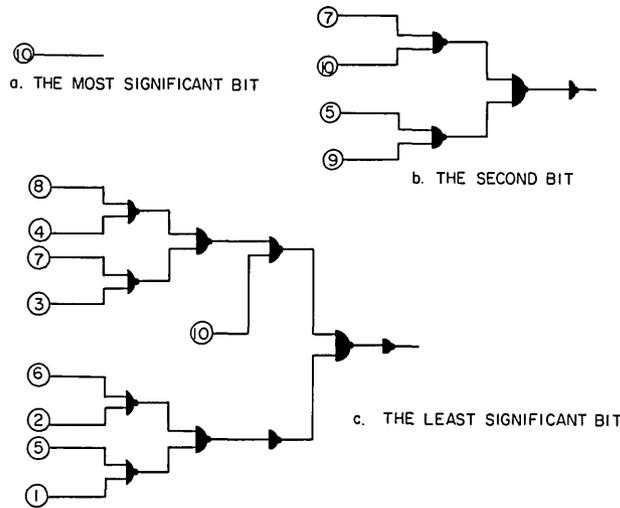


Fig. 30 - Address decoding circuits for the base 2 RRN

$$1 + 3(1 + 2 + \dots + 2^{M-2}) = 1 + 3 \sum_{i=0}^{i=M-2} 2^i = 1 + 3(2^{M-1} - 1) \quad (6)$$

gates. Thus, decoding the entire address requires

$$(N - 1) + 3 \sum_{M=1}^{M=N} (2^{M-1} - 1) = 3(2^N) - 2N - 4 \quad (7)$$

gates. Adding Eq. (5), we see that the entire RRN requires a total of

$$G_2 = 6(2^N) - 2N - 7 \quad (8)$$

gates for an AM of 2^N words. In the limit, the RRN thus requires only six gates per word-slice. We note that if the RRN is implemented in AND or OR logic, or some combination of the two, this is further reduced by a factor of between 1/3 and 1/2, with an appropriate increase in speed.

As was mentioned, an alternate method of obtaining the address is to store it within each word-slice and read it out with the response. A modified tree circuit that reads out only the lowest addressed response may require only one or two gates per word-slice. However, storing the address within the word-slice requires at least N additional gates. For memories of practical size, therefore, this alternate method is clearly more expensive.

The principles just applied in the base 2 RRN can be extended to an RRN of any base. For example, for a memory of 4^N words the primary circuit of a base 4 RRN is shown in Fig. 31 (corresponding to Fig. 29 of the base 2 system). Secondary circuits to decode each bit in the address of the lowest responding word-slice are shown in Fig. 32. The basic circuit (Fig. 31) requires

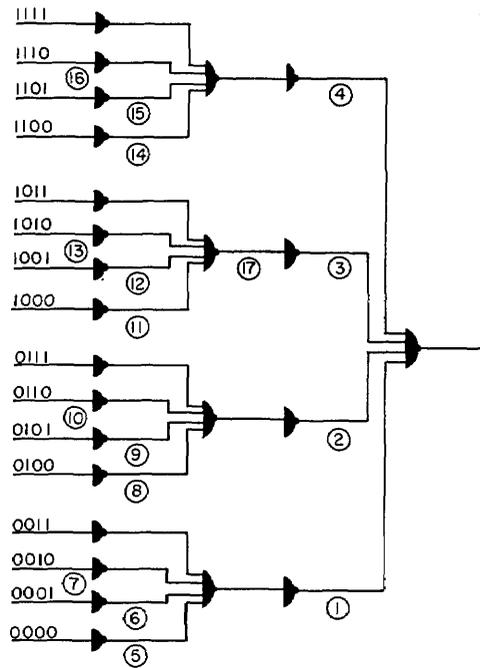


Fig. 31 - Primary circuit of the base 4 RRN

$$5 \sum_{i=0}^{i=N-1} 4^i = \frac{5}{3} (4^N - 1) \tag{9}$$

gates. Each word-slice in a memory of 4^N words has an address $2N$ bits long. Labeling each successive pair of bits with an index M , there are N pairs of bits, where $M = 1$ stands for the two most significant bits of the address, etc. To decode each pair of bits requires

$$4^{M-1} + 4 \sum_{i=0}^{i=M-1} 4^i = 4^{M-1} + \frac{4}{3} (4^M - 1) \tag{10}$$

gates ($M = 1$ corresponds to Figs. 32a and 32b). Decoding the entire address requires

$$\sum_{M=1}^{M=N} 4^{M-1} + \frac{4}{3} \sum_{M=1}^{M=N} (4^M - 1) = \frac{4^N - 1}{3} + \frac{4}{3} \frac{4^{N+1} - 1}{3} - (N + 1) \tag{11}$$

gates. Adding Eq. (9), we see that the entire base 4 RRN requires a total of

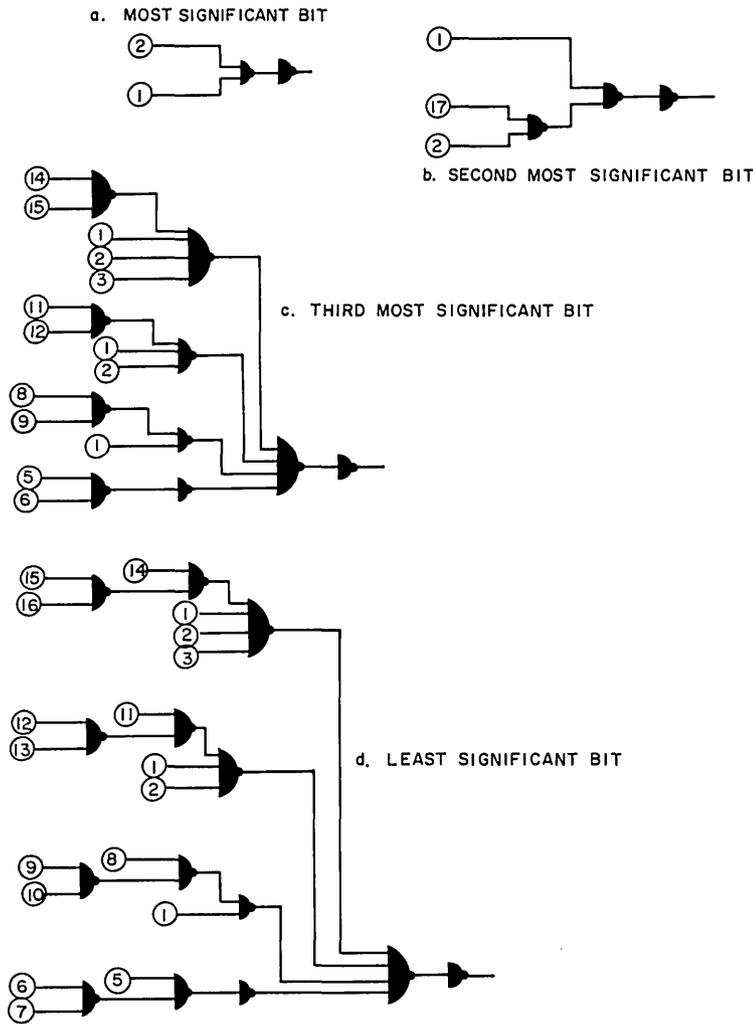


Fig. 32 - Address decoding circuits for the base 4 RRN

$$\begin{aligned}
 G_4 &= 2(4^N - 1) + \frac{4}{3} \frac{4^{N+1} - 1}{3} - \frac{4}{3} (N + 1) \\
 &= \frac{34}{9} (4^N - 1) - \frac{4}{3} N
 \end{aligned} \tag{12}$$

gates. In the limit, this reduces to 3-7/9 gates per word-slice. Similar to the base 2 system, different logic technologies can further reduce this by a factor of between 1/3 and 1/2, bringing the total down to one or two gates per word-slice.

5.4 Timing Considerations

To obtain a rough idea of the speed at which the AP will operate, it is useful to analyze the timing of the circuits described in Sections 5 through 5.3. In the following analysis we have made several implicit assumptions that should be kept in mind. First, operation times are calculated according to the circuit implementation of Figs. 21 and 28

through 32. We have already pointed out that, when constructed, these circuits may look quite different, especially if they are derived from Appendixes A and B. Second, we have assumed that a single NAND gate delay is a constant, independent of the number of inputs. Finally, we have ignored the need for line drivers or sense amplifiers, both of which will be required at various points in the AM.

In Section 5.1.1 it was mentioned that race-time problems are eliminated if we make sure that the next clock pulse after the initiation of an operation does not come before all J-K inputs have reached a steady state. Thus a convenient definition of operation time is the time, in gate delays, from the initiation of an AM operation until it is safe to inject the next clock pulse (thereby terminating this operation and, perhaps, initiating the next one). As might be expected, AM operation times depend not only on the specific type of operation (search, multiadd, etc.) but also on the contents of both the control registers and the AM itself. We shall examine the nature of this dependence on an operation-by-operation basis.

5.4.1 The Associative Search Time

We first consider a "simple" associative search consisting solely of "exact match" and "don't care" specifications, i.e., a search that is entirely bitwise independent (as was stated in Section 5.1.2). Examination of Fig. 21 shows that the response of each bit is available at point R a maximum of five gate delays after new values are gated onto the control registers. The two-gate network that combines the responses for each chip with those of the preceding chip appears once on each chip. If there are a total of K chips in the non-RS section of the word-slice, the total time until the combined response is available at output of bit 1 is given by

$$T_0 = 5 + 2K \quad (13)$$

gate delays. (Throughout the following discussion, all times are expressed in units of a single gate delay.) This final output corresponds to line 21 of the RS (Fig. 28). From the time that this output is ready it takes an additional four gate delays until RS chip input 7 has been raised (assuming that the associative search line is high). From Fig. 21 we see that another two gates are required until the J-K inputs are properly arranged. Thus, the RS action takes an additional six gate delays, and the total time necessary for the bitwise independent search is

$$T_1 = 11 + 2K. \quad (14)$$

We now consider the more complex search that involves a "greater than" or "less than" specification. To take a worst-case example, suppose that the entire N-bit (non-RS) word-slice is one syllable and that we are searching for syllables in the AM larger than the COMP. A worst-case word-slice that satisfies this search will have its most significant bit larger than that of the COMP with all remaining bits less than the corresponding COMP bits, thus requiring the affirmative response to ripple down through every bit. The response (at R) of the most significant bit is available five gate delays after initiation. The "response propagation" signal (output line 12) is available one gate delay sooner. The response of the next bit is available one gate delay after this (as seen by tracing input line 11). In general, the response (at R) of the least significant bit is available at

$$T_2 = 5 + 2(N - 2), \quad N \geq 2. \quad (15)$$

Meanwhile, the chip responses (output 14) are propagating down at the same effective speed, since the output of the summary network (enclosed in the dotted box in Fig. 21)

must always wait for the least significant bit of the chip before jumping to the next chip and waiting again. After the response of the least significant bit is available, an additional two gate delays are required before the word-slice output is available to the RS. Adding to this the six gate delays necessary for the RS operation, we see that a worst-case time of

$$T_3 = 9 + 2N \quad (16)$$

is required for the "greater than" or "less than" search.

We conclude that the time required for an associative search operation will always lie between the limits

$$11 + 2K \leq T_s \leq 9 + 2N, \quad (17)$$

regardless of the search criteria.

To take the analysis one step further, we consider that the upper limit of T_s is a function not only of the length but of the position of the largest syllable. To see this, suppose that in the N -bit (non-RS) word-slice the largest syllable subject to a nonsimple search (the largest syllable with a "greater than" or "less than" search criteria) is M bits long. It will take $5 + 2(M - 2)$ gate delays until all responses of this syllable are ready. If this syllable is located at the extreme left of the word-slice (if it occupies the most significant bits), an additional $2L$ gate delays are required for the response to reach the RS, where L is the number of chip edges standing to the right of the syllable. (This is independent of the syllable configuration of the remaining bits, since the other bits' responses will all be ready before that of the least significant bit of the largest syllable.) Adding the RS operation time, we see that the associative search time is now within the reduced interval

$$11 + 2K \leq T_s \leq 7 + 2M + 2L. \quad (18)$$

(We note that if $M = N$, then $L = 1$ and the upper limit becomes $2N + 9$, as in Eq. (16).)

If the largest syllable is located, instead, at the extreme right of the word-slice, the upper limit in Eq. (18) is reduced even further. The closed form of this new limit is a fairly complex function of M , N , and the syllable configuration of the rest of the word-slice. In general, though, the limit is less than that in Eq. (18) and lies within the interval

$$2M + 11 \leq \text{upper limit} \leq 7 + 2M + 2L, \quad (19)$$

where L is now the number of chip edges to the left.

If necessitated by processing speed requirements, the lower limit in Eq. (18) and the upper limit given by Eq. (19) can both be reduced. The procedure also eliminates the dependence of the search operation time on the position of the largest syllable (although not on its length). To accomplish these improvements (at the expense of construction costs) we permanently raise input 13 of each chip and no longer connect output 14 to input 13 of the next chip. Suppose that we have K chips in the (non-RS) word-slice. If we could pass all K response lines (output 14 from each chip) through a K -input AND gate, the output of this gate would be the overall word-slice response. Practically speaking, there is a technological limit on the number of gate inputs. Suppose that the largest AND gate available has P inputs. If we start with one such gate and take as inputs the outputs of P other identical AND gates, we have a P^2 -input AND network that involves only two gate delays. In general we can construct a P^n -input AND network requiring n gate delays. Applying this to an AM word-slice, if n is the smallest integer such that

$$P^n \cong K,$$

we may build a network of between

$$\sum_{i=1}^{i=n-1} P^i \quad \text{and} \quad \sum_{i=1}^{i=n} P^i$$

AND gates that combine the K chip responses in n gate delays. This network reduces the search operation time to within the interval

$$13 + n \leq T_s \leq 9 + 2M + n \quad (20)$$

(which may be compared with Eqs. (18) and (19)). As an example, suppose that we have available six-input AND gates, that we have less than 37 chips, and that the largest syllable with a nonsimple search specification (as defined by the SDR and the two SC registers) is 20 bits long. Then,

$$15 \leq T_s \leq 51, \quad (21)$$

the exact value depending on the search criteria. We note, that T_s is independent of both the total word-slice length and the overall AM size.

5.4.2 Addition Time

If we consider Fig. 21 as the least significant bit of some syllable (so that the carry is zero), we find that after new values become available on the control lines it takes

$$T_4 = 9 \quad (22)$$

or

$$T_4 = t_{ab} + 4, \quad (23)$$

whichever is greater, until the J-K inputs for this AM bit are properly arranged. Here, t_{ab} is the time at which the add bus is raised. The carry from this bit to the next more significant (to the left) arrives at the second bit after two gate delays. The carry to the third bit arrives after six gate delays. In general, the carry input to the n th bit is completely determined after $t_c = 2n$ gate delays, for $n > 2$. If Fig. 21 represents a bit such that $n > 2$, then we see that the addition is completed at

$$T_5 = t_c + 6 = 2S + 6 \quad S > 2 \quad (24)$$

or

$$T_5 = t_{ab} + 4,$$

whichever is greater. Hence, if the largest addition syllable is S bits wide, addition takes no longer than $2S + 6$, unless $t_{ab} > 2S + 2$. The size of t_{ab} depends upon the type of add being performed.

5.4.2.1 Multiaddition Time

Referring to Fig. 28, we see that two gate delays are required until the transformed SC contents enter the RS chip. Within the chip, it takes five gate delays until all bits

respond (the RS search is always bitwise independent, as was indicated in Fig. 6 and Section 5.3.1). Another two gate delays are required to collect the responses, so that the RS search is completed after nine gate delays. Tracing chip output 14 (Fig. 28) we see that the add bus is raised at

$$t_{ab} = 15. \quad (26)$$

Thus, the multiadd operation time will always fall within the limits

$$19 \cong T_{MA} \cong 2S + 6, \quad (27)$$

where S is the number of bits in the largest addition syllable (as defined by the SDR and MOR). Equation (27) is to be interpreted so that if $S < 5$ bits, $T_{MA} = 19$.

5.4.2.2 Conventional Addition Time

From Fig. 28 we see that if the conventional add line is raised, the add bus is raised at

$$t_{ab} = t_{dec} + 2, \quad (28)$$

where t_{dec} is the time it takes to decode the contents of the AMAR and raise the appropriate address select line. Thus, the conventional add operation time falls between the limits

$$t_{dec} + 6 \cong T_A \cong 2S + 6. \quad (29)$$

Equation (29) is to be interpreted so that if $S < t_{dec}/2$, $T_A = t_{dec} + 6$.

5.4.3 AM Write Time

From Fig. 21 we see that from the time a new value is gated onto the COMP, the time required until the J-K inputs are stable is

$$T_6 = 6 \quad (30)$$

or

$$T_6 = t_{wb} + \text{two gate delays}, \quad (31)$$

whichever is larger, where t_{wb} is the time at which the write bus is raised. Similar to t_{ab} , the size of t_{wb} depends upon the type of write being performed.

5.4.3.1 Multiwrite Time

From Fig. 28 and the discussion of Section 5.4.2.1 it is clear that

$$t_{wb} = 15, \quad (32)$$

so that the multiwrite operation time is

$$T_{MW} = t_{wb} + 2 = 17. \quad (33)$$

5.4.3.2 Conventional Write Time

From Fig. 28 and the discussion of Section 5.4.2.2 we see that the conventional write time is

$$T_W = t_{dec} + 4 \quad (\text{assuming } t_{dec} > 2). \quad (34)$$

5.4.4 AM Read Time

From Fig. 21 it is clear that T_R , the time until the contents of an AM word-slice is available to the AMOB, is given by

$$T_R = t_{rb} + 2, \quad (35)$$

where t_{rb} is the time at which the read bus is raised. From Fig. 23 we see that

$$t_{rb} = t_{dec} + 2, \quad (36)$$

so that

$$T_R = t_{dec} + 4.$$

5.4.5 Reduced Operation Times

Comparing Eq. (29) (conventional add), Eq. (27) (multiadd), and Eq. (20) (associative search), we notice that the upper limit in each equation represents a worst-case operation. For an AM of sufficient size there will always (statistically speaking) be one word-slice that is nearly worst case, so that the operation times will for the most part remain near the upper limits. Thus,

$$T_S \approx 9 + 2M + n, \quad (38)$$

$$T_{MA} \approx 2S + 6, \quad (39)$$

and

$$T_A \approx 2S + 6. \quad (40)$$

along with Eqs. (33), (34), and (37), which are

$$T_{MW} = 17, \quad (33)$$

$$T_W = t_{dec} + 4, \quad (34)$$

and

$$T_R = t_{dec} + 4, \quad (37)$$

we have six equations that describe the basic AM operation times.

In deriving these six equations we assumed that various control registers had their contents changed as the AM operations were initiated. Let us now withdraw that assumption. Operationally, this might correspond to the second of two AM operations executed with the same control register pattern or it might occur if the MPP had other operations to perform between loading the control registers and raising the appropriate operation line to the AM.

Consideration of Figs. 21 and 28 as well as the arguments leading to Eqs. (33), (34), and (37) through (40) results in the following relationships:

If the COMP, SC1, and SC2 have been stable for $5 + 2M + n$ gate delays, the search operation time is reduced to

$$T'_S = 4. \quad (41)$$

If the COMP and the MOR have been stable for $2S - 6$ gate delays and the RS portions of the COMP, SC1, and SC2 have been stable for five gate delays,

$$T'_{MA} = 12. \quad (42)$$

If the AMAR has been stable for at least t_{dec} and the COMP and MOR have been stable for $2S$ gate delays, then

$$T'_A = 6. \quad (43)$$

If the RS sections of the COMP, SC1, and SC2 have been stable for five gate delays,

$$T'_{MW} = 6. \quad (44)$$

If the AMAR has been stable for at least t_{dec} and the COMP has been stable for two delays,

$$T'_W = 4. \quad (45)$$

Finally, if the AMAR has been stable for at least t_{dec} ,

$$T'_R = 4. \quad (46)$$

We mention, once again, that all of these times are independent of both word-slice length and overall AM size.

5.4.6 Simultaneous Operation Times

Suppose a binary operation (defined in Section 4.6) is composed of the two single operations O_1 and O_2 , with operation times t_1 and t_2 respectively. From Sections 5 through 5.4.4 it should be clear that in most cases the binary operation time is simply the larger of t_1 and t_2 . This is because the operations execute independently. The only exceptions to this rule are the search/multiwrite and search/multiadd operations, the exceptions being due to the modification, described in Section 4.6, that the multioperation is executed for those word-slices satisfying the entire search, instead of just the RS search.

Consider the search/multiadd binary operation. We recall from Section 5.4.1 that six gate delays are required for completion of an associative search after the word-slice response is available at gate A in Fig. 28. Tracing the output of A through gates F and G to the add bus (in the case of a simultaneous operation) we see that the add bus is raised six gate delays after the word-slice response is available at gate A. Since an additional four gate delays are required to complete the addition (see Section 5.4.2) we conclude the following: If $T_{MA} > T_S$, the binary operation time is given by

$$T_{S/MA} = T_{MA}. \quad (46)$$

However, if $T_{MA} < T_S$, then the binary operation time is

$$T_{S/MA} = T_S + 4. \quad (47)$$

A similar analysis shows that

$$T_{S/MW} = T_S + 2 \quad (48)$$

(T_S is always greater than T_{MW} .) These increments are hardly significant. In general, then, we may say that a binary operation takes little or no more time than the longest of its two component operations.

5.4.7 The Response Resolution Time

Consideration of Figs. 29 and 31, as well as the discussion of Section 5.3 shows that for an AM of M words, the primary circuit of a base B RRN requires no more than

$$T_7 = 2 \log_B M \quad (49)$$

gate delays to stabilize after any RRB's are changed (assuming the availability of B -input NAND gates). The secondary circuits stabilize within an additional two gate delays, so that the total time to retrieve the address of the lowest responding word-slice is

$$T_{RRN} = 2 (\log_B M) + 2. \quad (50)$$

As mentioned in Section 5.3, this may be reduced as much as 50% through use of different logic technologies.

Examination of Figs. 28 and 21 shows that the time necessary to reset the RRB of this word-slice (and, if desired, simultaneously read it out) is

$$T_{RRB/R} = t_{dec} + 6. \quad (51)$$

5.4.8 Practical Considerations

From the analysis in Sections 5.4.1 through 5.4.7 it is clear that AM operation times highly depend not only on the type of operation but also on the contents of the AM control registers. This suggests that to obtain maximum AP processing speed, the use of an asynchronous clock is necessary. Just how necessary is clear when you consider the clock repetition rate required by the use of a synchronous clock. From Eq. (17) we see that if the AM is 60 bits wide, the maximum operation time, which defines the fastest possible synchronous clock rate, would be 129 gate delays. Since a large percentage of AM operations take only small fractions of this time, we see that the use of a synchronous clock is unnecessarily restrictive.

A high-PRF synchronous clock can be conveniently converted into an asynchronous clock that drives the AM as efficiently as possible (wasting a minimum amount of time) by use of a clock inhibit network (CIN), shown in Fig. 33. The CIN operates as follows: In each MPP instruction, one syllable contains an AM operation time parameter. This (compiler-produced) parameter is arrived at through consideration of the AM operation type as well as the control register contents and the age of the contents. The output of the corresponding syllable in the MPP instruction register (IR) is connected to the CIN. The CIN inhibits the incoming clock pulses until the required period (as specified by the operation time parameter) has passed, at which point one clock pulse is allowed to

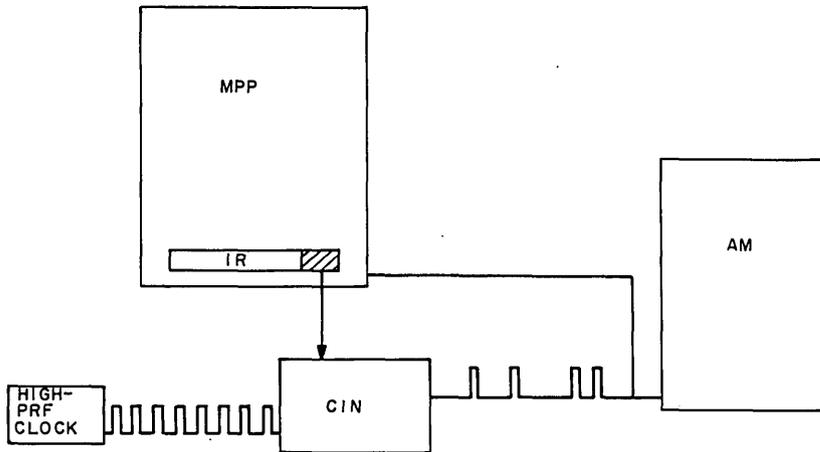


Fig. 33 - Relationship between the MPP, the AM, and the clock inhibit network

continue through to the CIN output. This output is connected both to the AM and to the MPP. When the clock pulse arrives at the MPP, the next instruction is gated onto the IR.

If a particular instruction does not involve the AM, i.e., if the AM instruction code is zero (see Section 4.6), this mechanism still enables us to control the MPP clock according to the requirements of the particular MPP instruction.

5.5 Interword Communication — Future Developments in the AP Communication Topology

A significant description of a processor design may be obtained by an examination of what communication lines exist between processing elements. In general, the more unstructured and flexible the communications topology, the more powerful (and, usually, more difficult to program) is the processing system thus described.

In the AP described herein, the principle communication link is a one-way line from the COMP to every AM word-slice. Physically, this high-data-rate channel is comprised of parallel connections between each COMP bit and every bit in the corresponding AM bit-slice.

Communication is also possible between different sections (or syllables) of a word-slice, simultaneously for every word-slice in the AM. This intraword-slice communication is necessarily bit-serial (as described in Sections 4.7.3 and 4.7.4), so that the communication channel has a lower data rate, and hence is of lesser importance, than the channel from the COMP to every AM word-slice.

There is, however, in the design presented thus far, no parallel communication between different word-slices in the AM. Stated differently, information can flow in parallel from the COMP to every point in AM and information can flow horizontally within the AM but information cannot flow vertically within the AM. We note that a communications channel from a single word-slice to all other word-slices may be established by reading this word-slice into the AMOB and then gating the contents of the AMOB onto the COMP. However, it is not possible for information to flow simultaneously between many sets of AM word-slices.

A limited (low-data-rate) vertical information channel can be established by permitting each word-slice to communicate with the word-slice immediately above or below it. This may be accomplished as follows: From the point of view of an AM word-slice it is irrelevant whether or not the COMP input (input 2 in Fig. 21) really comes from the COMP. This suggests that a new pair of external lines be used to select either the COMP, or the word-slice above, or the word-slice below, as the input that in the present design comes only from the COMP. If the input selection is made according to the code in Fig. 34, then Fig. 35 shows how this vertical communications channel might be implemented.

INPUT WORD-SLICE	CONTROL LINE # 1	CONTROL LINE # 2
COMPARAND	1	1
WORD-SLICE ABOVE	1	0
WORD-SLICE BELOW	0	1

Fig. 34 - Vertical word-slice selection code

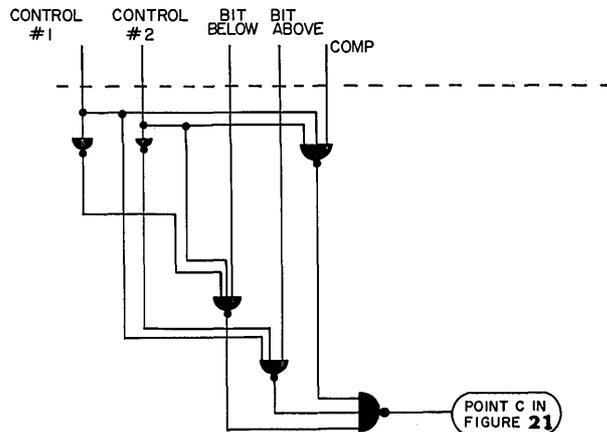


Fig. 35 - Design change for implementation of a one-dimensional vertical information flow

As a result of this design change, every operation involving the COMP and selected word-slices (see Section 4) may be performed with the same selected word-slices and the word-slice directly above or below (we note that the selection described by Fig. 34 is for the entire AM). Alternately, one might say that we have succeeded in imposing a one-dimensional vertical communication topology on a one-dimensional AM array. This additional capability might be used, for example, in statistical routines where differences between adjacent array elements are important, or in simulations of one-dimensional systems, where the behavior of any point (as described by a set of parameters stored within each word-slice) is considered a function of the parameters of adjacent points.

This immediately suggests that a two-dimensional AM array be constructed by stacking a series of one-dimensional AM's and connecting them so that a left-right selection can be added to the up-down selection specified in Fig. 34. Such a configuration

would be useful in processing two-dimensional information (e.g., optical processing) as well as in the simulation of two-dimensional systems. Going one step further, we may add an additional degree of freedom by stacking a series of two-dimensional AM's and adding a back-front word-slice selection. Such a configuration would be ideal for the simulation of any three-dimensional system whose time development may be expressed as a combination of universal and local effects (universal effects may be introduced through the COMP). Potential applications include atmospheric studies, atomic reactor calculations, and plasma simulations. As in most AM operations described in previous sections, the simulation time would be independent of the number of points being simulated.

The additional gate and control line requirements for these various communication topologies are summarized in Fig. 36.

COMMUNICATION TOPOLOGY	EXTRA CONTROL LINES	GATES REQUIRED
ONE DIMENSIONAL	$3N + 2$	$6N$
TWO DIMENSIONAL	$5N + 3$	$9N$
THREE DIMENSIONAL	$7N + 3$	$11N$

Fig. 36 - Control line and gate requirements for advanced communication topologies (N is the number of bits per chip)

The above design modifications have been isolated from the basic design both because the additional capability is more of special-purpose application than other AP operations and because, in terms of present technologies, the additional gate and control line requirements are relatively expensive.

Appendix A

BOOLEAN EQUATIONS FOR THE ASSOCIATIVE CHIP

In terms of the control lines described in Fig. 22 and the internal lines of Fig. 21 the Boolean equations for the associative chip are as follows:

$$6 = 5 \cdot Q,$$

$$10 = 2 \cdot Q + 9 \cdot 1 \cdot (Q \oplus 2),$$

$$J = 8 \cdot 15 \cdot \left[(2 \oplus Q) \oplus (9 \cdot 1) \right] + 15 \cdot 7 \cdot (Q \oplus 2)$$

$$K = 8 \cdot 15 \cdot \overline{\left[(2 \oplus Q) \oplus (9 \cdot 1) \right]} + 15 \cdot 7 \cdot (Q \oplus 2)$$

$$14 = 13 \cdot \left[A \cdot \overline{(Q \oplus 2)} + B + C \cdot (2 \cdot \bar{Q}) + D \cdot (\bar{2} \cdot Q) + 1 \cdot \overline{(Q \oplus 2)} + \bar{11} \right],$$

$$12 = \overline{\left[C \cdot (2 \cdot \bar{Q}) + D \cdot (\bar{2} \cdot Q) + \bar{11} \right]} \cdot 1.$$

In the above,

$$A = 3 \cdot 4,$$

$$B = \bar{3} \cdot \bar{4},$$

$$C = 3 \cdot \bar{4},$$

$$D = \bar{3} \cdot 4.$$

Also, Q_{n+1} is the output of the AM bit flip-flop after the next clock pulse.

Appendix B

BOOLEAN EQUATIONS FOR THE RESPONSE STORE

Operation of the response store (Fig. 28) may be conveniently described by the following set of Boolean equations, in terms of the numbered lines summarized on Fig. B1.

NO.	DESCRIPTION OF LINE
1	MOR input to RS
2	COMP input to chip
3	SC1 input to chip
4	SC2 input to chip
5	SC1 input to RS
6	SC2 input to RS
7	write input to chip
8	add input to chip
9	COMP input to RS
10	associative search line
11	multi-write line
12	multi-add line
14	response output of RS
15	MOR input to chip
17	read line
18	read bus
19	add bus
20	write bus
21	word-slice response
22	conventional write line
23	conventional add line
24	address select line
25	reset RB line

Fig. B1 - Summary of the response store control lines

$$3 = 4 = 5 \cdot 6,$$

$$2 = 9,$$

$$18 = 24 \cdot 17,$$

$$15 = \bar{5} \cdot \bar{6} \cdot 9 \cdot (10 + 24 \cdot 25) + 22 \cdot 24 \cdot 1 + 11 \cdot 14 \cdot 1 \cdot (\bar{10} + 10 \cdot 21),$$

$$7 = 21 \cdot 14 \cdot 10 + 22 \cdot 24 + 11 \cdot 14 \cdot (\bar{10} + 10 \cdot 21),$$

$$8 = 24 \cdot 25,$$

$$20 = 22 \cdot 24 + 11 \cdot 14 \cdot (\bar{10} + 10 \cdot 21),$$

$$19 = 23 \cdot 24 + 12 \cdot 14 \cdot (\bar{10} + 10 \cdot 21).$$

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Research Laboratory Washington, D.C. 20390		2a. REPORT SECURITY CLASSIFICATION Unclassified	
		2b. GROUP	
3. REPORT TITLE A FAST, FLEXIBLE, HIGHLY PARALLEL ASSOCIATIVE PROCESSOR			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) An interim report on a continuing NRL Problem.			
5. AUTHOR(S) (First name, middle initial, last name) John E. Shore and Frank A. Polkinghorn, Jr.			
6. REPORT DATE November 28, 1969		7a. TOTAL NO. OF PAGES 50	7b. NO. OF REFS None
8a. CONTRACT OR GRANT NO. NRL Problem R06-41		9a. ORIGINATOR'S REPORT NUMBER(S) NRL Report 6961	
b. PROJECT NO. A37-533-000/6521/WF08-151-702		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
c.			
d.			
10. DISTRIBUTION STATEMENT This document has been approved for public release and sale; its distribution is unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY Department of the Navy (Naval Air Systems Command) Washington, D.C. 20360	
13. ABSTRACT The logical design and operation of a general purpose associative processor (GPAP) is described. The basic circuit consists of a powerful associative cell which may be combined with an integral number of identical cells and implemented in MSI or LSI at a reasonable cost. A complete description of this cell, together with logic diagrams, Boolean equations, and a detailed timing analysis are presented. The processor obtained by connecting these cells together in quantity has both variable syllable and variable instruction capability. That is, the total associative word length can be split on a software basis into any number of syllables (fields), each of arbitrary length. The search criteria (greater than, less than, greater than or equal to, less than or equal to, exact match, and don't care) can be specified independently for each of these syllables. The search time depends only on the width of the largest syllable and is typically less than 1 microsecond (assuming gate delays of approximately 20 nanoseconds). The problem of multiple responses (priority resolution) is considered in detail. The design of a high-speed, inexpensive (in terms of the number of gates required per word) response resolution network is presented. Other GPAP operations include conventionally addressed read, write, add, multiwrite, multiadd, and logical operations. Simultaneous operations, such as write-on-match and add-on-match, are also possible. These capabilities, as well as several other unique properties of the design which contribute to its general purpose character and high speed of operation are described in detail.			

(OVER)

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computers Design Associative storage Associative processors Parallel processors Integrated circuits Semiconductor computer storage Boolean algebra Array computers Associative memories Constant addressable memories Constant addressable processors Digital design Priority resolution in associative memories Solid state memories Speed						

GPAP will be controlled by a microprogrammable processor with timing produced by a clock inhibit network which counts down a compiler-produced operation-time parameter.