Chapter 21

A Productive Implementation of an Associative Array Processor: STARAN[1]

Jack A. Rudolph / Kenneth E. Batcher

Introduction [Rudolph]

The associative or content-addressed memory has been an attractive concept to computer designers ever since Slade and McMahon [1957] described a "catalog" memory. Associative memories offered relief from the continuing problem presented by the typical coordinate-addressed memory which requires that an "address" be obtained or calculated before data stored at that address may be retrieved. The associative memory could acquire in a single memory access any data from memoy without pre-knowledge of its location. Ordered files and sorting operations could be eliminated. Unfortunately, early associative memories were expensive, hence none found their way as the "main frame" memory into any commercial computer design.

The organization of an associative memory (AM) requires that each n-bit physical word of the memory be connected to a dedicated processing element (PE) which performs the compare function between a bit read non-destructively from the word and a corresponding input bit from a query word. The PE's for all words are driven by a central controller, thus a single query bit is simultaneously compared with the corresponding stored bit in every word of the AM. With the ability to simultaneously write back the state of each PE into a specified bit position of each word it became possible to perform bit-serial arithmetic between fields of bits within each physical memory word. An array of associative memory words could then be viewed as an array of simple computers—an associative array processor—with all the simple computers in the array simultaneously executing the same instruction obtained from a common control unit as is done in the more complex ILLIAC-IV design.

An alternative AP design provides a PE at each bit of each physical memory word. This design, though complex in terms of logic and interconnection requirements, permits a simultaneous compare of all bits in a query word with all bits of the memory word rather than the serial-by-bit operation described earlier.

Due to the early high cost of semi-conductor memory and logic elements none of the many associative processor designs described in the literature were attractive enough to warrant development. However, it has now become commercially feasible to construct a computing system embodying "main frame" memory content addressability coupled with array arithmetic capability operating under a more or less conventional stored program control system.

Several proprietary versions of the associative processor (AP) are being developed. The first working engineering model known to the author, built for USAF by Goodyear Aerospace Corporation, was demonstrated during a Tri-Service contract review in June, 1969 at Akron, Ohio [Fulmer and Meilander, 1970]. The same machine, modified to include a larger instruction memory, was loaned by USAF in 1971 to the FAA for conflict detection tests in a live air traffic control terminal environment at Knoxville, Tennessee operating in a multi-computer configuration with a Univac 1230 conventional computer [Rudolph, Fulmer and Meilander, 1971]. The original test objectives were achieved by December, 1971 and additional experiments involving terrain avoidance processing were completed successfully in June, 1971.

The lessons learned in programming and testing the USAF AP model resulted in a new design called STARAN S which was committed to production in 1971. This first commercial AP was publicly introduced in a series of live demonstrations in May, 1972 at the TRANSPO exhibit in Washington, D.C. and in June, 1972 at Boston, Mass.

This paper describes STARAN S and its programming language, provides examples of its applications, and discusses measures of AP cost-effectiveness.

STARAN Description

A configuration diagram of STARAN S is shown in Fig. 1. Studies have shown that initial uses of AP's would be weighted toward real-time applications involving interface with a wide variety of sensors, conventional computers, signal processors, interactive displays, and mass storage devices. To accommodate all such interfaces the STARAN system was divided into a standardized main frame design and a custom interface unit. A variety of I/O options implemented in the custom interface unit includes conventional direct memory access (DMA), buffered I/O (BIO) channels, external function channels (EXF) and a unique interface called parallel I/O (PIO).

A top-cut diagram of the STARAN main frame is shown in Fig. 2. It consists of a conventionally addressed control memory for program storage and data buffering, a control logic unit for sequencing and decoding instructions from control memory and from one to thirty-two modular AP arrays. A typical AP array is shown in Fig. 2.

To accommodate both bit-slice accesses for associative process-

[1]This chapter is compiled from Rudolph [1972], *Proc. FJCC*, 1972, pp. 229–241; and Batcher [1974], *Proc. NCC*, 1974, pp. 405–410.

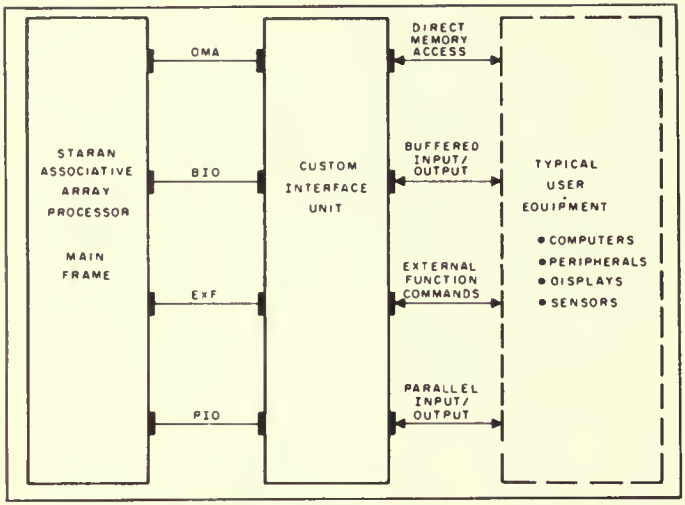

Fig. 1. STARAN system configuration.

ing and word-slice accesses for STARAN input/output (I/O), the data are stored in a multi-dimensional access (MDA) memory (Fig. 2).[1] It has wide read and write busses for parallel access to a large number (256) of memory bits. The write mask bus allows selective writing of memory bits. Memory accesses (both read and write accesses) are controlled by the address and access mode control inputs; the access mode selects a stencil pattern of 256 bits, while the address positions the stencil in memory.

[1]The passage beginning with this paragraph is from Batcher [1974].

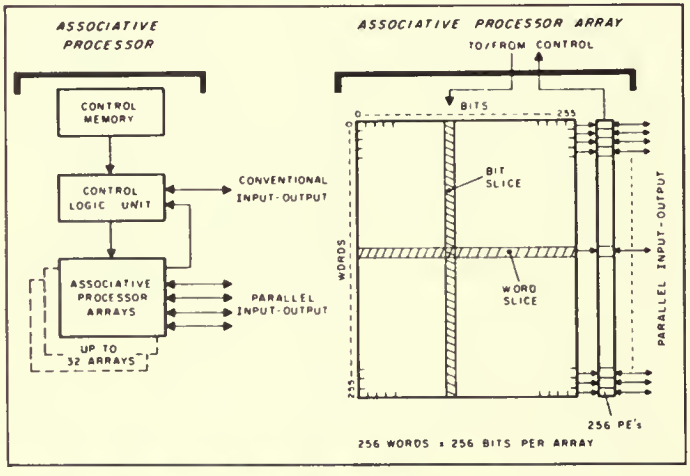

Fig. 2. Associative processor diagrams.

For many applications, the MDA memory is treated as a square array of bits, 256 words with 256 bits in each word. The bit-slice access mode (Fig. 2*a*) is used in the associative operations to access one bit of all words in parallel, while the word acess mode (Fig. 2*b*) is used in the I/O operations to access several or all bits of one word in parallel.

The MDA memory structure is not limited to a square array of 256 by 256. For example, the data may be formatted as records with 256 8-bit bytes in each record. Thirty-two such records can be stored in an MDA memory and accessed several ways. To input and output records, one can access 32 consecutive bytes of a record in parallel (Fig. 3*a*). To search key fields of the data, one can access the corresponding bytes of all records in parallel (Fig. 3*b*). To search a whole record for the presence of a particular byte, one can access a bit from each byte in parallel (Fig. 3*c*).

The MDA memories in the STARAN array modules are bipolar. They exhibit read cycle times of less than 150 nsec and write cycle times of less than 250 nsec.

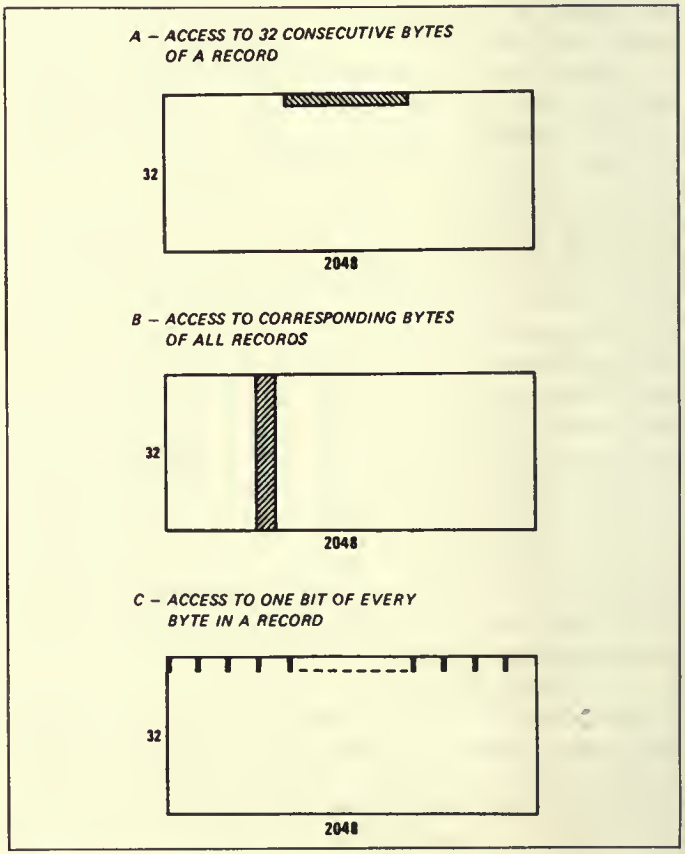

Fig. 3. Accessing 256-byte records.

STARAN Array Modules

A STARAN array module (Fig. 4) contains an MDA memory communicating with three 256-bit registers (M, X, and Y) through a flip (permutation) network. One may think of an array module as having 256 small processing elements (PE's), where a PE contains one bit of the M register, one bit of the X register, and one bit of the Y register.

The M register drives the write mask bus of the MDA memory to select which of the MDA memory bits are modified in a masked-write operation. The MDA memory also has an unmasked-write operation that ignores M and modifies all 256 accessed bits. The M register can be loaded from the other components of the array module.

In general, the logic associated with the X register can perform any of the 16 Boolean functions of two variables; that is, if x_i, is the state of the ith X-register bit, and f_i is the state of the ith flip network output, then:

$$x_i \leftarrow \phi(x_i, f_i) \; (i = 0, 1, \ldots, 255)$$

where ϕ is any Boolean function of two variables. Similarly, the logic associated with the Y-register can perform any Boolean function:

$$y_i \leftarrow \phi \; (y_i, f_i) \; (i = 0,1, \ldots, 255)$$

where y_i is the state of the ith Y-register bit. The programmer is given the choice of operating X alone, Y alone, or X and Y together.

If X and Y are operated together, the same Boolean function, ϕ, is applied to both registers:

$$x_i \leftarrow \phi \; (x_i, f_i)$$
$$y_i \leftarrow \phi \; (y_i, f_i)$$

The programmer also can choose to operate on X selectively, using Y as a mask:

$$x_i \leftarrow \phi \; (x_i, f_i) \qquad \text{(where } y_i = 1\text{)}$$
$$x_i \leftarrow x_i \qquad \text{(where } y_i = 0\text{)}$$

Another choice is to operate on X selectively while operating on Y:

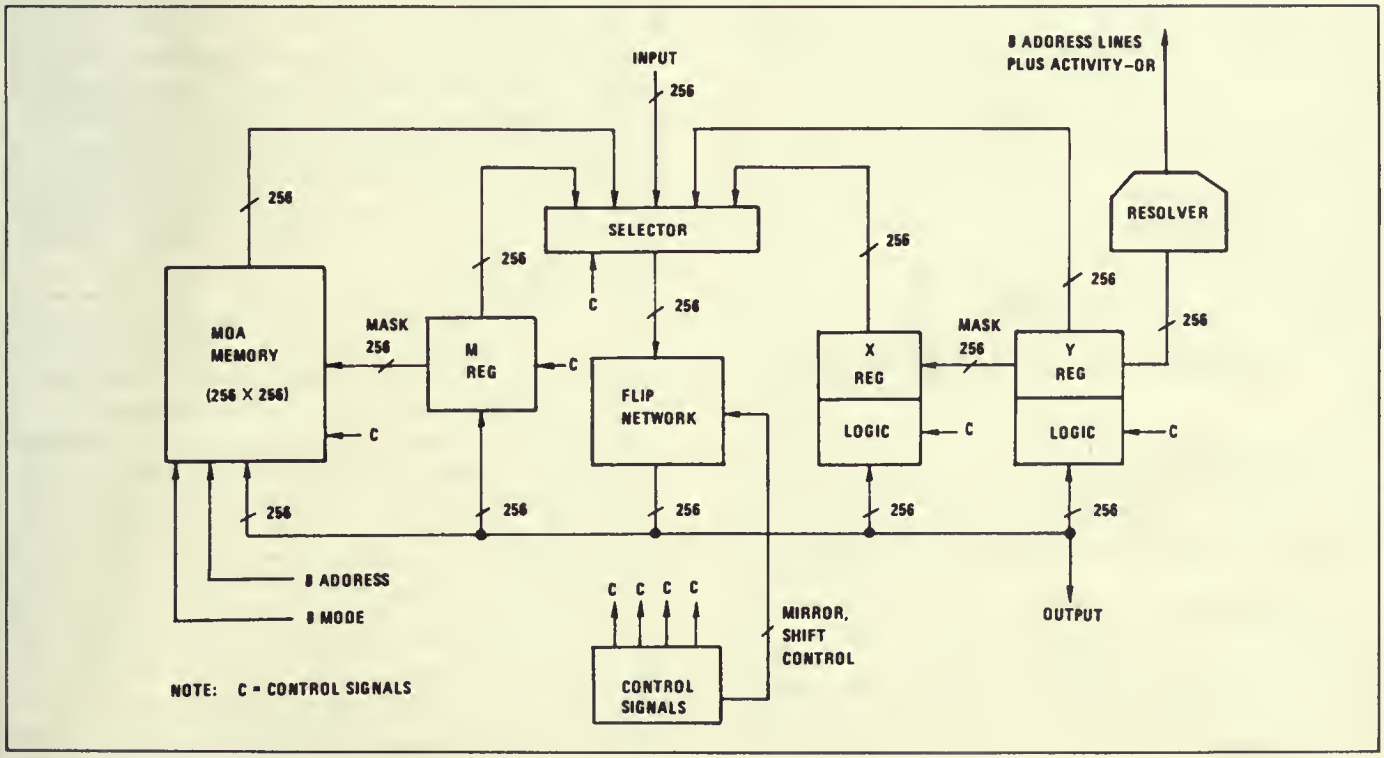

Fig. 4. STARAN array module.

$x_i \leftarrow \phi\ (x_i, f_i)$ (where $y_i = 1$)
$x_i \leftarrow x_i$ (where $y_i = 0$)
$y_i \leftarrow \phi\ (y_i, f_i)$

In this case, the old state of Y (before modification by ϕ) is used as the mask for the X operation.

For a programming example, the basic loop of an unmasked add fields operation is selected. This operation adds the contents of a Field A of all memory words to the contents of a Field B of the words and stores the sum in a Field S of the words. For n-bit fields, the operation executes the basic loop n times. During each execution of the loop, a bit-slice (a) of Field A is read from memory, a bit-slice (b) of Field B is read, and a bit-slice (s) of Field S is written into memory. The operation starts at the least significant bits of the fields and steps through the fields to the most significant bits. At the beginning of each loop execution, the carry (c) from the previous bits is stored in Y, and X contains zeroes:

$x_i = 0$
$y_i = c_i$

The loop has four steps:

Step 1: Read Bit-slice a and exclusive-or ($\oplus$) it to X selectively and also to Y:

$x_i \leftarrow x_i \oplus y_i a_i$
$y_i \leftarrow y_i \oplus a_i$

The states of X and Y are now:

$x_i = a_i c_i$
$y_i = a_i \oplus c_i$

Step 2: Read Bit-slice b and exclusive-or it to X selectively and also to Y:

$x_i \leftarrow x_i \oplus y_i b_i$
$y_i \leftarrow y_i \oplus b_i$

Registers X and Y now contain the carry and sum bits:

$x_i = a_i c_i \oplus a_i b_i \oplus b_i c_i = c'_i$
$y_i = a_i \oplus b_i \oplus c_i = s_i$

Step 3: Write the sum bit from Y into Bit-slice s and also complement X selectively:

$s_i \leftarrow y_i$
$x_i \leftarrow x_i \oplus y_i$

The states of X and Y are now:

$x_i = c'_i \oplus s_i$
$y_i = s_i$

Step 4: Read the X-register and exclusive-or it into both X and Y:

$x_i \leftarrow x_i \oplus x_i$
$y_i \leftarrow y_i \oplus x_i$

This clears X and stores the carry bit into Y to prepare the registers for the next execution of the loop:

$x_i = 0$
$y_i = c'_i$

Step 3 takes less than 250 nsec, while Steps 1, 2, and 4 each take less than 150 nsec. Hence, the time to execute the basic loop once is less than 700 nsec. If the field length is 32 bits, the add operation takes less than 22.4 microsec plus a small amount of setup time. The operation performs 256 additions in each array module. This amounts to 1024 additions, if four array modules are enabled, to achieve a processing power of approximately 40 MIPS (million-instructions-per-second).

The array module components communicate through a network called the flip network. A selector chooses a 256-bit source item from the MDA memory read bus, the M register, the X register, the Y register, or an outside source. The bits of the source item travel through the flip network, which may shift and permute the bits in various ways. The permuted source item is presented to the MDA memory write bus, M register, X register, Y register, and an outside destination.

The permutations of the flip network allow inter-PE communication, A PE can read data from another PE either directly from its registers or indirectly from the MDA memory. One can permute the 256-bit data item as a whole or divide it into groups of 2, 4, 8, 16, 32, 64, or 128 bits and permute within groups.

The permutations allowed include shifts of 1, 2, 4, 8, 16, 32, 64, or 128 places. One also can mirror the bits of a group (invert the left-right order) while shifting it. A positive shift of mirrored data is equivalent to a negative shift of the unmirrored data. To shift data a number of places, multiple passes through the flip network may be required. Mirroring can be used to reduce the number of passes. For example, a shift of 31 places can be done in two passes: mirror and shift 1 place on the first pass, and then remirror and shift 32 places on the second pass.

The flip network permutations are particularly useful for Fast Fourier transforms (FFT's). A 2^n point FFT requires n steps, where each step pairs the 2^n points in a certain way and operates on the two points of each pair arithmetically to form two new

points. The flip network can be used to rearrange the pairings between steps. Bitonic sorting [Batcher, 1968] and other algorithms [Stone, 1971] also find the permutations of the flip network useful.

Each array module contains a resolver reading the state of the Y register. One output of the resolver (activity-or) indicates if any Y bit is set. If some Y bits are set, the other output of the resolver indicates the index (address) of the first such bit. Since the result of an associative search is marked in the Y register, the resolver indicates which if any words respond to the search.

Other STARAN Elements

Figure 5 is a block diagram of a typical STARAN system with four array modules. Each array module contains an assignment switch that connects its control inputs and data inputs and outputs to AP (associative processor) control or the PIO (parallel input/output) module.

The AP control unit contains the registers and logic necessary to exercise control over the array modules assigned to it. It receives instructions from the control memory and can transfer 32-bit data items to and from the control memory. Data busses communicate with the assigned array modules. The busses connect only to 32-bits of the 256-bit-wide input and output ports of the array modules (Fig. 4), but the permutations of the array module flip networks allow communication with any part of the array. The AP control sends control signals and MDA memory addresses and access modes to the array modules and receives the resolver outputs from the array modules.

Registers in the AP control include:

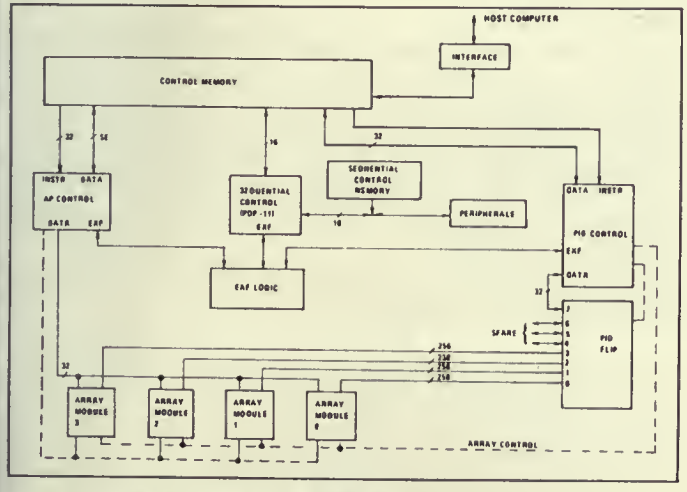

Fig. 5. Typical STARAN block diagram.

1 An instruction register to hold the 32-bit instruction being executed.

2 A program status word to hold the control memory address of the next instruction to be executed and the program priority level.

3 A common register to hold a 32-bit search comparand, an operand to be broadcast to the array modules, or an operand output from an array module.

4 An array select register to select a subset of the assigned array modules to be operated on.

5 Four field pointers to hold MDA memory addresses and allow them to be incremented or decremented for stepping through the bit-slices of a field, the words of a group, etc.

6 Three counters to keep track of the number of executions of loops, etc.

7 A data pointer to allow stepping through a set of operands in control memory.

8 Two access mode registers to hold the MDA memory access modes.

The parallel input/output (PIO) module contains a PIO flip network and PIO control unit (Fig. 5). It is used for high bandwidth I/O and inter-array transfers.

The PIO flip network permutes data between eight 256-bit ports. Ports 0 through 3 connect to the four array modules through buffer registers. Port 7 connects to a 32-bit data bus in the PIO control through a fan-in, fan-out switch. Ports 4, 5, and 6 are spare ports for connections to high bandwidth peripherals, such as parallel-head disk stores, sophisticated displays, and radar video channels. The spare ports also could be used to handle additional array modules. High bandwidth inter-array data transfers up to 1024 bits in parallel are handled by permuting data between Ports 0, 1, 2 and 3. Array I/O is handled by permuting data between an array module port and an I/O port. The PIO flip network is controlled by the PIO control unit.

The PIO control unit controls the PIO flip network and the array modules assigned to it. While AP control is processing data in some array modules the PIO control can input and output data in the other array modules. Since most of the registers in the AP control program are duplicated in PIO control; it can address the array modules associatively.

The control memory holds AP control programs, PIO control programs, and microprogram subroutines. To satisfy the high instruction fetch rate of the control units (up to 7.7 million instructions per second), the control memory has five banks of bipolar memory with 512 32-bit words in each bank. Each bank is expandable to 1024 words. To allow for storage of large programs, the control memory also has a 16K-word core memory with a cycle

time of 1 microsec. The core memory can be expanded to 32K words. Usually the main program resides in the core memory, and the system microprogram subroutines reside in bipolar storage. For flexibility, users are given the option of changing the storage allocation and dynamically paging parts of the program into bipolar storage.

A Digital Equipment Corporation (DEC) PDP-11 minicomputer is included to handle the peripherals, control the system from console commands, and perform diagnostic functions. It is called sequential control to differentiate it from the STARAN parallel processing control units. The sequential control memory of 16K 16-bit words is augmented by a 8K×16-bit "window" into the main control memory. By moving the window, sequential control can access any part of control memory. The window is moved by changing the contents of an addressable register.

The STARAN peripherals include a disk, card reader, line printer, paper-tape reader/punch, console typewriter, and a graphics console.

Synchronization of the three control units (AP control, sequential control, and PIO control) is maintained by the external function (EXF) logic. Control units issue commands to the EXF logic to cause system actions and read system states. Some of the system actions are: AP control start/stop/reset, PIO control start/stop/reset, AP control interrupts, sequential control interrupts, and array module assignment.

The design of STARAN allows it to be connected to other computers (host computers) as a special-purpose peripheral. The interface can take many different forms. One could connect to an I/O channel of the host. Alternately, one could connect to the memory bus of the host so that it can address STARAN memory directly and/or allow STARAN to address its memory directly. For example, the STARAN at Rome Air Development Center [Feldman] is connected to an I/O channel of a Honeywell HIS-645 computer. At Goodyear Aerospace, another STARAN is interfaced to the direct memory access port of an SDS Σ 5 computer.

Associative Processor Software [Rudolph]

The STARAN software system consists of a symbolic assembler called APPLE (for *A*ssociative *P*rocessor *P*rogramming *L*anguag*E*), and a set of supervisor, utility, debug, diagnostic, and subroutine library program packages. An associative compiler has not yet been developed for STARAN. Early applications of STARAN must therefore be accomplished by assembly language programmers. Programmers find APPLE a convenient language to use, however, and write significantly fewer instructions to program a suitable application on STARAN than would have to be written for a conventional machine since APPLE's command structure reflects the content addressability and processing characteristics of the associative arrays the language controls. For example, although the programmer must explicitly define his record formats via field definition statements, he usually need not be concerned with physical record location in the arrays. Also, he need not order data tables by key, since any desired datum may be located in one parallel search operation. A third example of APPLE convenience is the elimination of the conventional programming loop which requires advancing a list pointer, examination of an exit criterion, and making a decision for each pass over different data sets. The APPLE array instruction processes all pertinent data sets simultaneously and does not require initialization of an index register with the count of data sets to be processed.

Internally, all software packages with the exception of array diagnostics and the subroutine library operate on the SC. In the minimum STARAN configuration the software packages are furnished on paper tape for input via the SC tape reader. Where STARAN is installed with interface to a conventional computer system in a multicomputer configuration, APPLE and supporting software can be input to STARAN using the existing peripherals of the conventional computer.

The usual load, store, test, branch, and control instructions required for sequential execution of an application program are present in APPLE. Where APPLE departs most from conventional assemblers is in the search and arithmetic array instructions. A representative set of fixed point standard instructions is shown in Table 1 with the approximate timing formulas. Hardware floating point is available on special order.

Associative search and arithmetic instructions are of two types, "argument register" and "field." In the first an operand (32 bits max) stored in the argument register of AP control is used as the search or arithmetic argument against a specified field in all array words simultaneously. Instructions of the field type perform similar operations but between specified fields within each array word.

Instruction execution times are dependent upon n, the number of bits in the operands (fields) involved in the instruction executions, but are not functions of the number of operands being processed, which relationship is exactly the opposite of that existing in the conventional computer. This characteristics dependence of execution time on operand or field length is a consequence of the word-parallel bit-serial design of the associative arrays discussed earlier.

From the programmer's point of view, Table 1 has interesting connotations, some of which are:

1 In real-time applications the programmer can easily time out his initial flow diagram since programming loops in the conventional sense are eliminated. This single consequence of associative processing can save much of the reprogramming effort invariably found necessary during the testing phase of conventional attacks on real-time problems.

2 He can conserve on execution time (and array memory

Table 1 Typical APPLE Associative Fixed Point Instructions

Mnemonic	*Instruction*	*Approx. execution time (μs)** *Formula*	*n = 16*	*n = 32*	*MIPS† per array for n = 32*
	Argument register instructions				
EQC	EXACT MATCH COMPARAND	0.6 + 0.15n	3.0	5.4	47
GTC	GREATER THAN COMPARAND	0.7 + 0.15n	3.1	5.5	47
LTC	LESS THAN COMPARAND	0.7 + 0.15n	3.1	5.5	47
ADC	ADD AR TO FIELD	2.8 + 0.85n	16	30	8.5
	Field instructions				
EQF	EXACT MATCH FIELDS	0.6 + 0.43n	7.4	14	18
GTF	GREATER THAN FIELDS	2.3 + 0.43n	9.1	16	16
LTF	LESS THAN FIELDS	2.3 + 0.43n	9.1	16	16
MAXF	MAX FIELDS	0.6 + 0.68n	11	23	11
MINF	MIN FIELDS	0.6 + 0.68n	11	23	11
ADF	ADD FIELD TO FIELD	2.8 + 0.85n	16	30	8.5
MPF	MULTIPLY FIELD BY FIELD	5.8 + 2.9m + 0.85mn + 0.4	277	980	0.26

*n or m equal number of bits in operand.

†Max execution rate of specified instructions for single array with all 256 PE's active.

space) by defining fields to use only as many bits as are required by the application.

3 He has no need for overhead-generating techniques such as indexed file constructions, linked lists, or sort and merge operations usually needed in a conventional computer. This capability results in a significant reduction both in the number of instructions which must be written and executed and the amount of memory required.

Array Storage Allocation

The concept of a file of related records as used in associative processing requires some discussion. In conventional approaches to file generation one thinks of the distinction between a logical file and a corresponding physical file; that is, a logical collection of records, usually ordered by some key, is placed as a block of contiguous addresses in a physical file. The conventional operating system keeps track of the beginning address and the block length for the file whether stored in core or on external stores. Thus in most cases logically different files are stored in physically separate areas of store.

The associative approach differs from the conventional approach in several ways: the records within the logical file need not and usually are not ordered by any key; records within a logical file usually are not stored in contiguous locations in an area of the array or on external devices; and the operating system generally is not required to keep track of individual file beginning addresses and block lengths.

In STARAN, records belonging to different logical files may be physically intermixed in the array as well as being logically unordered. Within each record format, in addition to defining the item fields, the programmer defines a set of control tag fields. How these tags are used is described below.

When new records are added to a logical file the update program writes the new, properly formatted record into the first available empty array location. Since empty array locations usually are not contiguously located within the array, records belonging to a specific file are scattered throughout the array in random locations. This characteristic is illustrated in the array map example of Fig. 6.

Empty array memory locations are identified by executing an EQC on a one-bit activity tag field using a "0" as the search criterion. The execution time for this search (see Table 1) is less than one microsecond at the end of which time all processing elements for physical memory words containing a 0 in the activity field will be in the "ON" state. At the conclusion of the search a hardware pointer automatically points to the PE having the lowest physical address in the array (or arrays). The new record, with its activity field set to a "1," is written into this first empty location. The hardware pointer then moves to the next available empty memory location for writing another record if a batch of new entries must be loaded. If no empty locations are found the program will exit to whatever routine the programmer has chosen for handling this type of error—for example, if appropriate to a specific application, the program may select an age test of all records in a particular file, purging the oldest to make room for

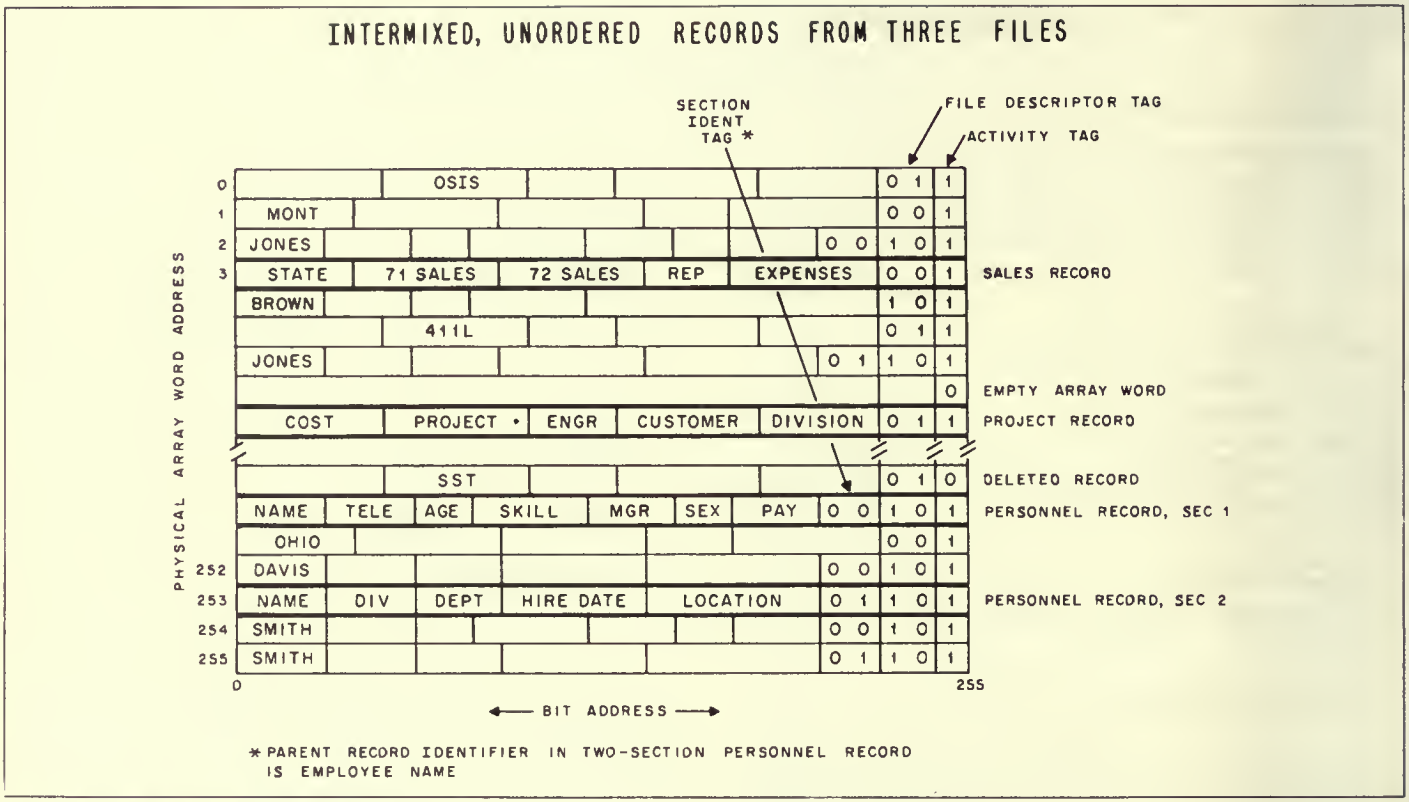

Fig. 6. Associative array map example.

the newest. A record once located may be deleted from a file by merely setting the activity bit to a "0."

When a specific file is to be processed in some manner, the scattered locations containing the file's records are activated by performing EQC's on both the activity field and an n-bit "file descriptor" tag field. If, as in the example of Fig. 6, the file descriptor field is two bits long, the entire selected file will be ready for processing in less than 2 microseconds (<1 μs for the activity bit search, <1 μs for the file descriptor field search).

Where record lengths are greater than the 256-bit length of the associative array word, several non-contiguous associative array words may be used to store the single record in sections, one section per array word. The format for each record section must contain the same activity and file descriptor fields as are used in all record formats, and in addition it must contain a parent record identifier and an n-bit "section identifier" tag field. The scattered locations containing the desired section of all records in the specific file may be activated by performing EQC's on the activity, file descriptor, and section identifier fields. All three searches can be completed in approximately 2 or 3 microseconds.

These two or three tag search operations in the AP permit random placement of records in the physical file and eliminate the bookkeeping associated with file structuring and control required in conventional systems. The same approach is used for files which exceed the capacity of the associative arrays—the records of such files are stored in a similar manner on external mass storage devices and are paged into the arrays as required.

The strategy used to allocate array storage space can have a significant effect on program execution time. An example is shown in Fig. 7, where the products of three operand pairs are required. In A, the operands are stored in a single array word. For 20-bit fixed point operands the three MPF instructions would execute in a total of 1175 microseconds. All similar data sets stored in other array words would be processed during the same instruction execution. However, an alternative storage scheme (B) which utilizes three PE's per data set requires only one MPF execution

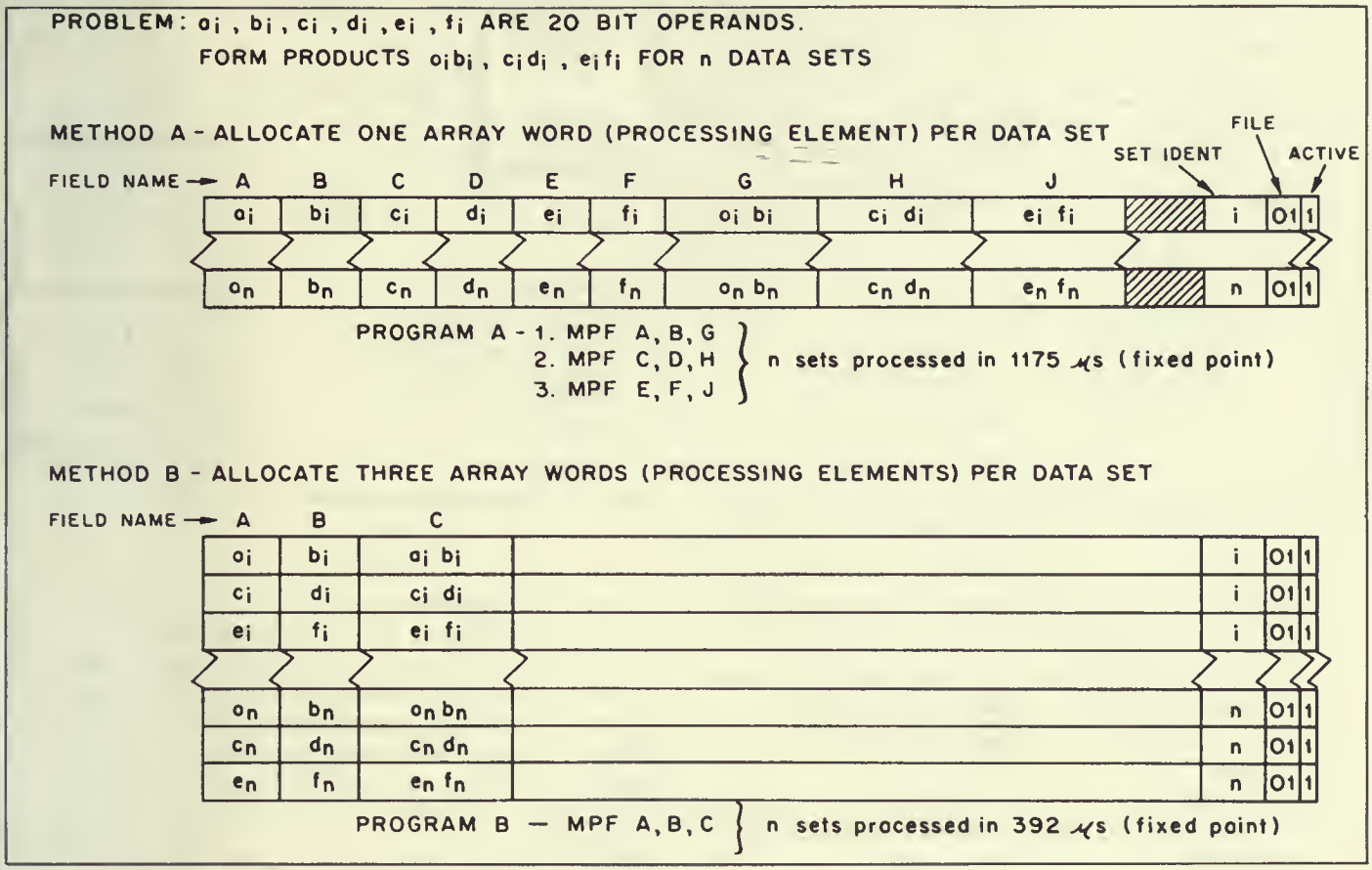

Fig. 7. Effect of array memory allocation on execution time.

to produce the three products in 392 microseconds. If one thousand data sets were involved in each case the average multiply times per product would be 392 and 131 nanoseconds, respectively, but at the expense, in B, of using 3000 processing elements. Unused bits in B may be assigned to other functions.

A last example of how array storage allocation can affect program execution time is shown in Fig. 8, where the columns represent fields. Here the sum e_1, of 16 numbers is required. If the 16 numbers are directly or as a result of a previous computation stored in the same field of 16 physically contiguous array words, the near-neighbor relationships between the processing elements can be used to reduce the number of ADF executions to four. All similar 16 number sets would be processed at the same time.

STARAN Applications

While many papers have appeared (see Minker [1971] for a comprehensive bibliography) which discuss the application of AM's and AP's in information retrieval, text editing, matrix computations, management information systems and sensor data processing systems, there are none yet published which describe actual results with operating AP equipment in any application. (But see Stillman [1972] for a recent AM application result.)

Recent actual applications of the AP have been in real-time sensor related surveillance and control systems. These initial applications share several common characteristics:

1 A highly active data base.

2 Operations upon the data base involve multiple key

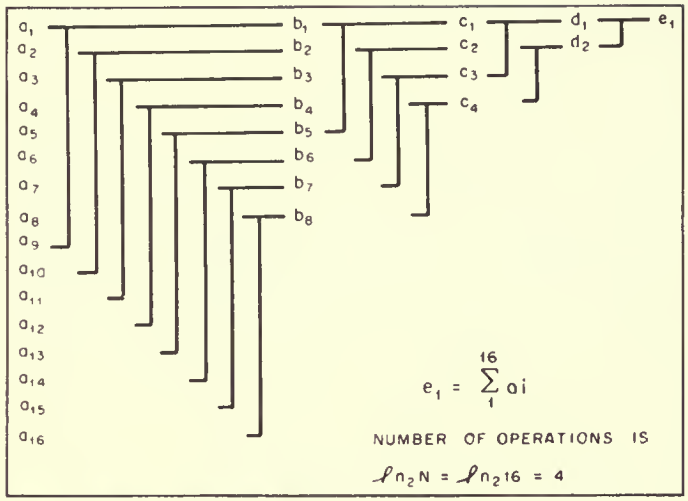

Fig. 8. Tree-sum example.

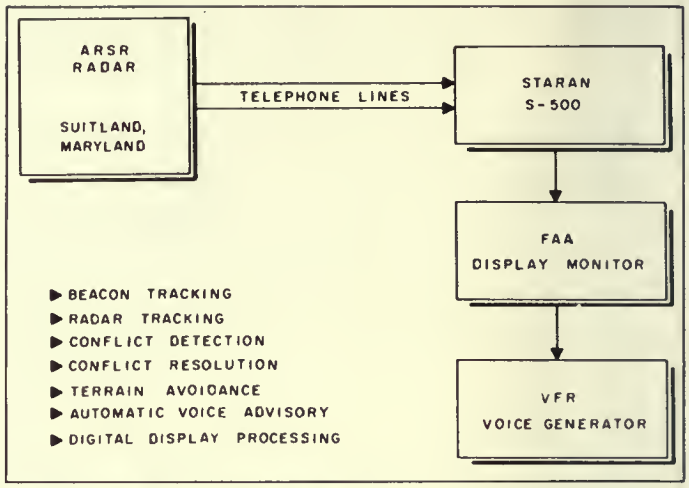

Fig. 9. Air traffic control application.

searches in complex combinations of equal, greater, between-limits, etc., operations.

3 Identical processing algorithms may be performed on sets of records which satisfy a complex search criterion.

4 One or more streams of input data must be processed in real time.

5 There is a requirement for real-time data output in accordance with individual selection criteria for multiple output devices.

A portion of the processing inherent in these applications is parallel-oriented and well suited to the array processing capability of the AP. On the other hand these same applications also involve a significant amount of sequentially-oriented computation which would be inefficient to perform upon any array processor, a simple example being coordinate conversion of serially occurring sensor reports.

Air Traffic Control

An example of an actual AP application in an air traffic control environment is shown in Fig. 9. In this application a two array (512 processing elements) STARAN S-500 model was interfaced via leased telephone lines with the output of the FAA ARSR long range radar at Suitland, Maryland. Digitized radar and beacon reports for all air traffic within a 55 mile radius of Philadelphia were transmitted to STARAN in real time. An FAA air traffic controller's display of the type used in the new ARTS-III terminal ATC system and a Metrolab Digitalk-400 digital voice generator were interfaced with STARAN to provide real-time data output. The controller's keyboard was used to enter commands, call up various control programs and select display options.

Although a conventional computer is not shown explicitly in Fig. 9 the sequentially oriented portions of the overall data processing load were programmed for and executed in the STARAN sequential controller as shown in Fig. 10. Sequential and associative programs and instruction counts for STARAN are shown in Table 2. In a larger system involving multiple sensors and displays, and more ATC functions such as metering and spacing, flight plan processing, and digital communications, the sequential and parallel workloads would increase to the point where a separate conventional computer system interfaced with the AP would be required.

The STARAN system was sized to process 400 tracks. Since the instantaneous airborne count in the 55 mile radius of Philadelphia was not expected to exceed 144 aircraft, a simulation program was developed to simultaneously generate 256 simulated aircraft tracks. Display options permitted display of mixed live and simulated aircraft. The 400 aircraft capacity is representative of the density expected as North-South traffic loads increase through the late '70s. Conflict prediction and resolution programs based upon computed track data were demonstrated and used to display conflict warning options. Automatic voice services were provided for operator-designated aircraft, thus simulating warning advisories for VFR pilots requesting the service. The voice messages, which in an operational system would be automatically radioed to the pilot, were generated by the Metrolab unit from digital formats produced by the associative processor and broadcast in the demonstration area via a public address system. A typical message would be read out in voice as, "ABLE BAKER CHAR-

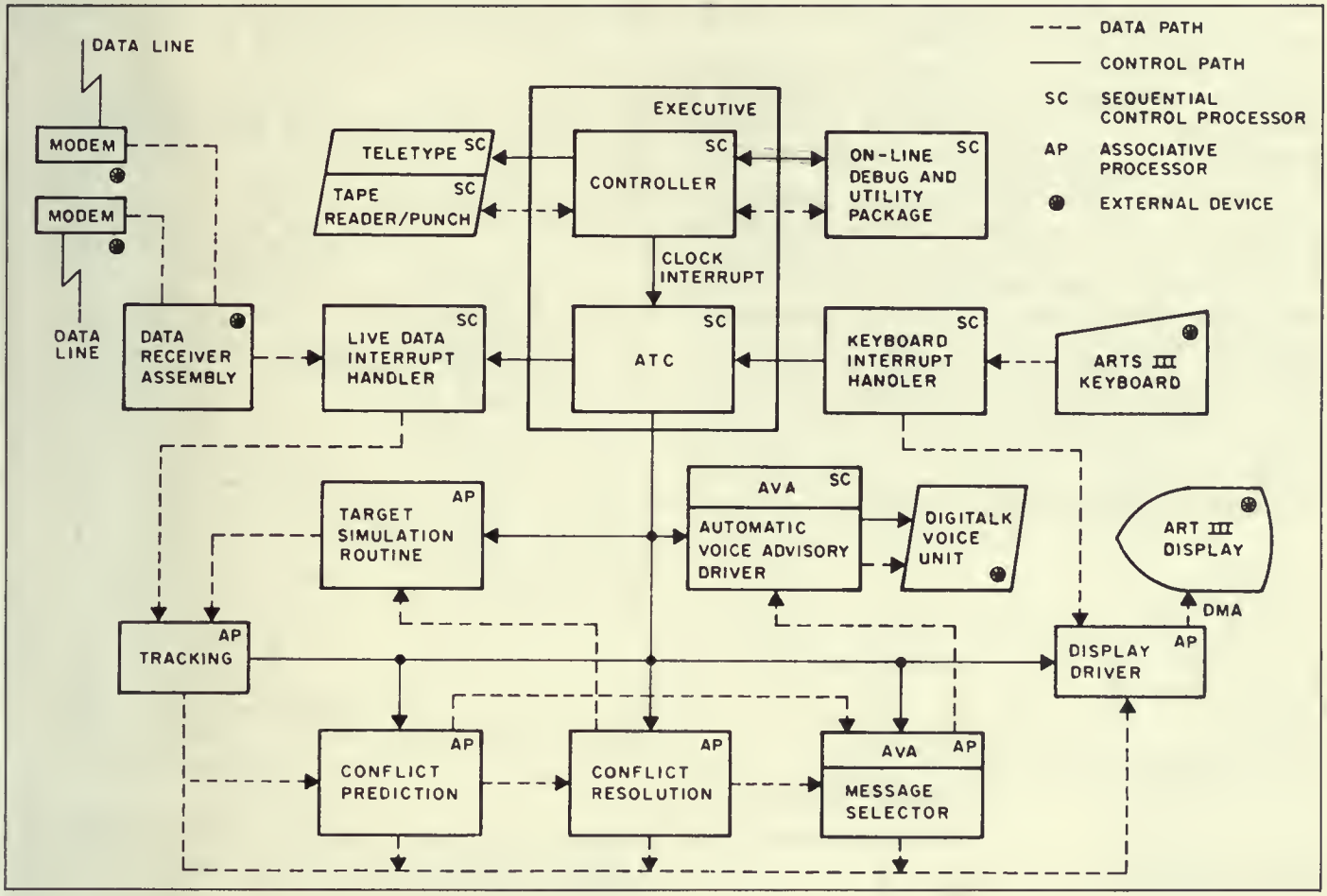

Fig. 10. ATC program organization.

Table 2 STARAN Air Traffic Control Programs

Sequential programs		*Associative programs*	
Name	*Instruction count*		*Instruction count*
Executive		Tracking system	881
Keyboard interrupt		Track simulation system	415
Real time interrupt	1600	Turn detection	88
Live data input		Conflict prediction	488
Automatic voice output		Conflict resolution	296
		Automatic voice advisory	709
		Display processing	1140
		Total	4017
		Field definition statements included	514
Net operating instructions	1600	Net operating instructions	3493

LIE, FAST TRAFFIC SEVEN O'CLOCK, 4 MILES, ALTITUDE 123 HUNDRED, NORTHEAST BOUND."

Top level flow charts for four of the associative programs, used in the demonstration are shown in Figs. 11, 12, 13, and 14. A detailed report is in preparation describing all of the ATC programs used in this demonstration, but some comments on the four flow charts shown may be of interest.

Live target tracking (Fig. 11) is performed in two dimensions (mode C altitude data was not available) using both radar and beacon target reports to track all aircraft. Incoming reports are correlated against the entire track file using five correlation box sizes, three of which vary in size with range. Any incoming report which does not correlate with an existing track is used to automatically initiate a new tentative track. An aircraft track must correlate on two successive scans and have a velocity exceeding 21 knots to qualify as an established track and must correlate on three successive scans to achieve a track firmness level high enough to be displayed to a controller as a live target. There are provisions for 15 levels of track firmness including 7 "coast" levels.

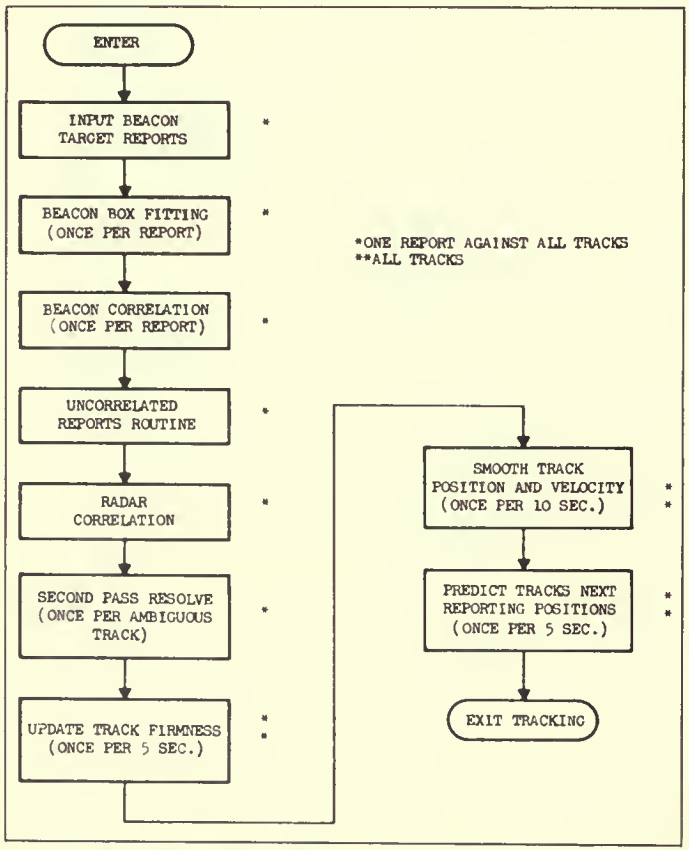

Fig. 11. Live target tracking.

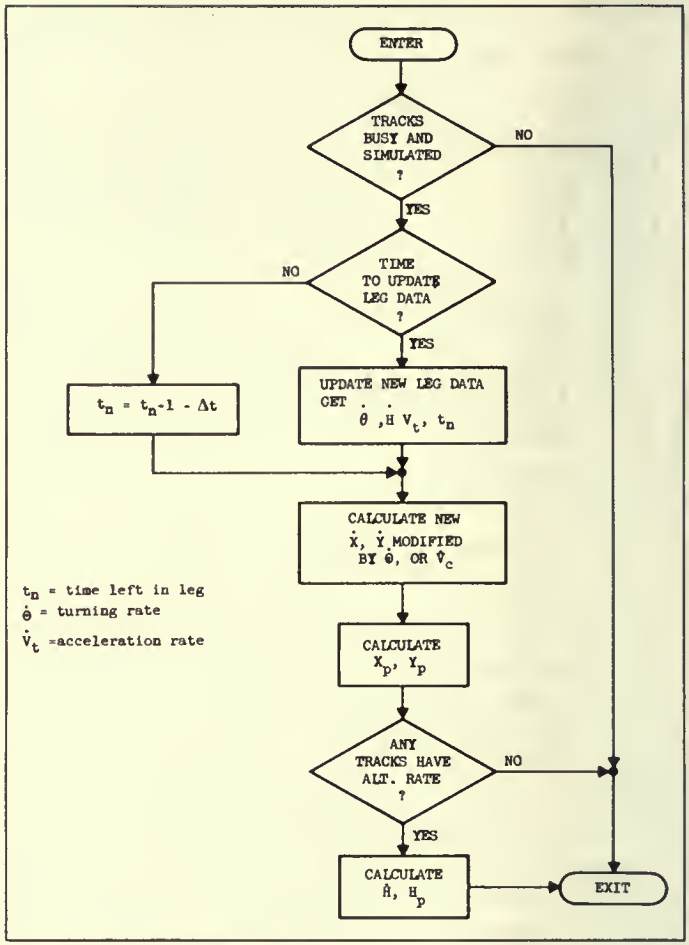

Fig. 12. Tracking simulation.

If a report correlates with more than one track, special processing (second pass resolve) resolves the ambiguity. Correlated new reports in all tracks are used for position and velocity smoothing once per scan via an alpha-beta tracking filter where for each track one of nine sets of alpha-beta values is selected as a function of track history and the correlation box size required for the latest report correlation. If both beacon and radar reports correlate with a track, the radar report is used for position updating. Smoothed velocity and position values are used to predict the position of the aircraft for the next scan of the radar and for the look-ahead period involved in conflict prediction.

Track simulation processing (Fig. 12) produces 256 tracks in three dimensions with up to four programmable legs for each track. Each leg can be of 0 to 5 minute duration and have a turn rate, acceleration, or altitude rate change. A leg change can be

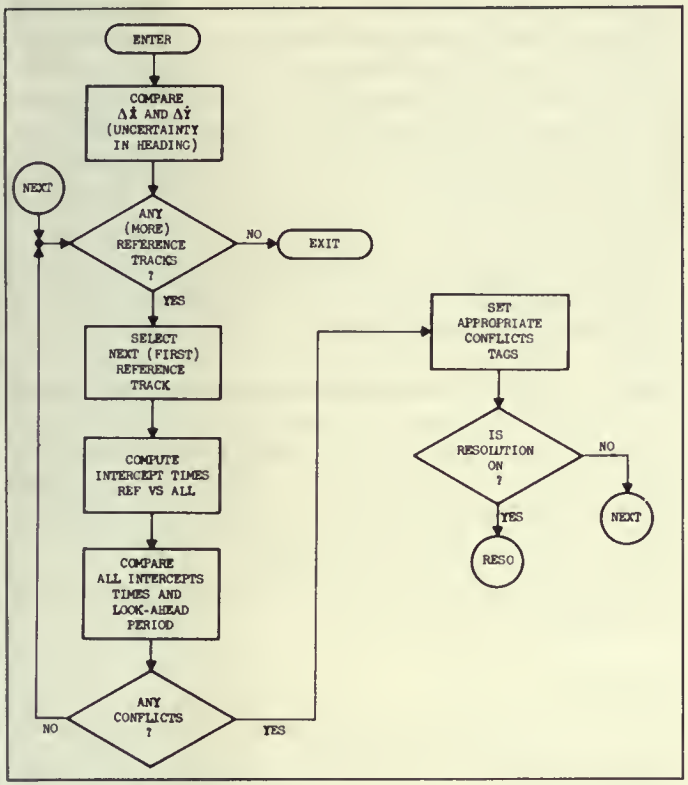

Fig. 13. Conflict prediction.

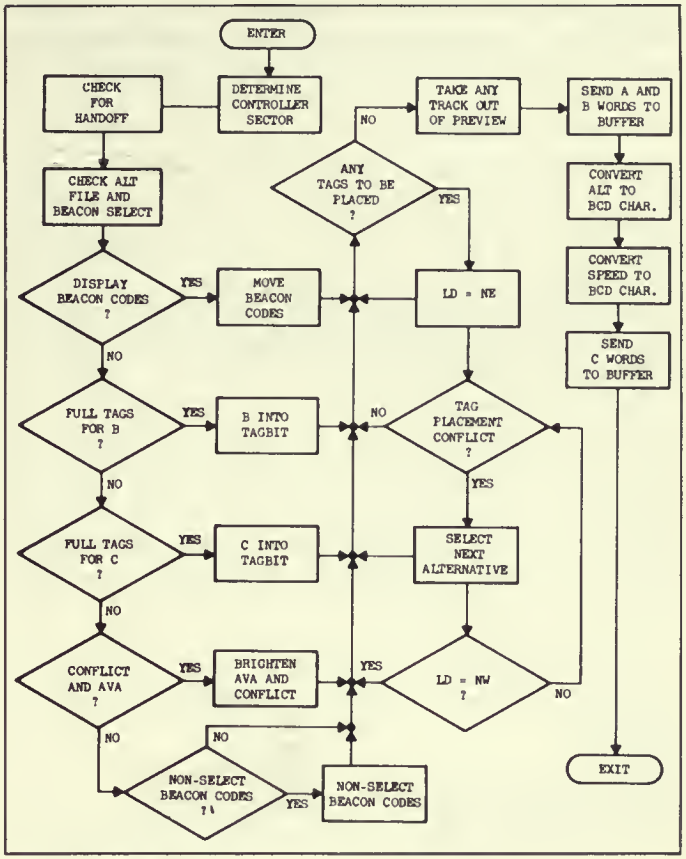

Fig. 14. Display processing.

forced by the conflict resolution program to simulate pilot response to a ground controller's collision avoidance maneuver command. Targets may have velocities between 0 and 600 knots, altitudes between 100 and 52,000 feet, and altitude rates between 0 and 3000 feet per minute.

The conflict prediction program sequentially selects up to 100 operator-designated "controlled" or "AVA" aircraft, called reference tracks in Fig. 13, and compares the future position of each during the look-ahead period with the future positions of all live and simulated aircraft and also to the static position of all terrain obstacles. Any detected conflicts cause conflict tags in the track word format to be set, making the tracks available for conflict display processing. A turn detection program not shown opens up the heading uncertainty for turning tracks.

Display processing (Fig. 14) is a complex associative program which provides a variety of manage-by-exception display options and automatically moves operator-assigned alphanumeric identification display data blocks associated with displayed aircraft so as to prevent overlap of data blocks for aircraft in close proximity to one another on the display screen. Sector control, hand off, and quick-look processing is provided.

All programs listed in Table 2 were successfully demonstrated at three different locations in three successive weeks, using live radar data from the Suitland radar at each location. The associative programs were operated directly out of the bulk core and page 0 portions of control memory since there was no requirement, in view of the low 400 aircraft density involved, for the higher speed instruction accesses available from the page memories. At intervals during the demonstration all programs were demonstrated at a speed-up of 20 times real time with the exception of the live data and AVA programs which, being real-time, cannot be speeded up. Timing data for the individual program segments will be available in the final report. The entire program executed in less than 200 milliseconds per 2 second radar sector scan or in less than 10 percent of real time. All programming effort was completed in 4½ months with approximately 3 man-years of effort. This was the first and as of this writing the only actual demonstration of a production associative processor in a live signal environment known to the author. It was completed in June, 1972. Other actual

applications currently in the programming process at Goodyear involve sonar, electronic warfare and large scale data management systems. These will be reported as results are achieved.

Fast Fourier Transform[1]

The Fast Fourier Transform (FFT) is a basic operation in digital signal processing which is being widely used in the real-time processing of radar and sonar signals. The structure of the FFT algorithm is such that it can be segmented into many similar concurrent operations. Parallel implementation of the FFT can provide orders of magnitude speed increases over sequential computer execution times. The organization of STARAN lends itself to efficient manipulation of data in the FFT.

The Air Force supplied real radar data (on tapes) to GAC to be transformed by the STARAN system. A 512-point, 16-bit FFT was performed on this real data in 2.7 milliseconds using only two MDA arrays. A 1024-point transform on real input data could be performed in about 3.0 milliseconds using all four arrays available at GAC's STARAN evaluation and training facility. For comparison purposes, the following is a list of reported execution times for a 1024-point, real input, FFT:

Sequential computers	
XDS Sigma 5	660 msec
IBM 360/67	446 msec
UNIVAC 1108	190 msec (complex)
UNIVAC 1108 (with array processor attachment)	29.2 msec (complex)
Special purpose FFT systems	
Time/Data 90 System	28 msec
ELSYTEC 306/HFFT	18 msec
SPECTRA SYSTEM '900'	9.2 msec

Sonar Post-Processing

Sensor data processing can be split into two major categories—signal processing and post-processing. Signal processing is the area of the system where operations such as the FFT are performed; post-processing involves the sorting and editing of the signal processor output data to determine tactical information such as whether a real target is in the coverage area and where the target is.

The job of sorting the spectral lines that result from the FFT operations is a formidable task, especially in a multi-sensor case. The trend has been for increasing the sensitivity of signal processing systems. The acoustic signal line sorting task that accompanies any increased sensitivity can be staggering. For instance, a 6 dB improvement in sensitivity, in a classified Navy sonar system, would result in increasing the target load by a factor of 16 and the computer processing load by a factor of 250 or more.

A digital sonar signal processing system, under development at the Naval Air Development Center (NADC), requires that subroutines operate on the target spectral lines (outputs from an FFT) and other input data to form outputs suitable for later use in classification algorithms. Since the system is a multi-sensor system, these subroutines must process a very large volume of data in real time. The content addressability feature of STARAN provides the potential for significant performance gains due to the requirement for many searches in these post-processing subroutines.

As a consequence of this potential improvement, NADC issued a contract to GAC to assess the comparative run times for the STARAN versus a large-scale conventional computer (the CDC-6600). NADC-developed algorithms for the most time consuming operations in the post-processor system were programmed on the STARAN computer. Real data was then processed on both the STARAN and, by NADC, on a CDC-6600.

The STARAN executed the programs, using the real data, 200 times faster than the CDC-6600.

String Search

A processing function used by several agencies for locating specific character strings (such as place names) in textual information, was developed for STARAN and tested on a sample data base. The same function was executed on a conventional computer (Sigma 5) for a timing comparison. The STARAN solution ran 100 times faster. This function is also applicable to nondefense applications such as patent, legal, and chemical information searches where cost of search may be a limiting parameter.

Summary [Rudolph]

Although several manufacturers are developing associative processor equipment, the first version to be produced in a production configuration was introduced in May of 1972 by Goodyear Aerospace Corporation following FAA on-site tests in 1971 at Knoxville, Tennessee of a USAF-owned engineering model built and demonstrated by Goodyear in 1969.

The processor provides full content addressability and array arithmetic capability within "main frame" memory coupled with a unique capability for wide bandwidth (over 3000 megabits/sec for a 4-array STARAN) input-output data transfers to mass data stores. The associative programming language, APPLE, provides a flexible and convenient assembler for programming array arithmetic and search algorithms without the complex and costly indexing, nested loop and data manipulation constructions required in conventional computer programming.

[1]The passage beginning here is from Batcher.

The associative processor may be viewed as a software-programmable super-peripheral, or special purpose subsidiary processor, for attachment to any general purpose conventional computer system via standard channel attachment. In this role the super-peripheral is assigned parallel oriented problem segments and data bases which would otherwise, through excess operating system software overhead, tend to choke the conventional machine.

Although first applications of the associative processor are of the real time, dedicated, command and control type, the extension to large scale data base management, on-line management information systems with immediate response to complex multiple-key queries, and large scale matrix computations await only user decision and ingenuity to accomplish now that production hardware and software has become available at the 370/145 price level.

The cost effectiveness of associative processing has yet to be proven in operational systems, but test results from initial users should accumulate rapidly now that associative processing is no longer only an interesting concept in the literature.

References

Batcher [1974]; Rudolph [1972]; Batcher [1968]; Feldman; Fulmer and Meilander [1970]; Minker [1971]; Rudolph, Fulmer, and Meilander [1971]; Slade and McMahon [1957]; Stillman [1972]; Stone [1971].