REPORT NO. DOT-TSC-FAA-71-15

LARGE SCALE SYSTEMS A STUDY OF COMPUTER ORGANIZATIONS FOR AIR TRAFFIC CONTROL APPLICATIONS

JOHN DUMANIAN
DAVID CLAPP
TRANSPORTATION SYSTEMS CENTER
55 BROADWAY
CAMBRIDGE, MA. 02142

JUNE 1971

TECHNICAL REPORT

Prepared for

FEDERAL AVIATION ADMINISTRATION
WASHINGTON,D.C. 20546

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. DOT-TSC-FAA-71-15	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Large Scale Systems - A Study of Computer Organizations for Air Traffic Control Applications		5. Report Date June 15, 1971 6. Performing Organization Code TCC
7. Author(s) John Dumanian & David Clapp		8. Performing Organization Report No.
9. Performing Organization Name and Address DOT/Transportation Systems Center Computer Technology Division 55 Broadway Cambridge, MA 02142		10. Work Unit No. FA-03 11. Contract or Grant No.
12. Sponsoring Agency Name and Address FAA-SRDS-RD-123 FOB 10A, 800 Independence Ave., S.W. Washington, D.C. 20546		13. Type of Report and Period Covered Technical Report 14. Sponsoring Agency Code FAA RD-123

15. Supplementary Notes

16. Abstract

Based on current sizing estimates and tracking algorithms, some computer organizations applicable to future air traffic control computing systems are described and assessed. Hardware and software problem areas are defined and solutions are outlined. System evaluation criteria are presented.

Section 1: delineates the objectives and approach, and furnishes definitions of computer hardware and software;

Section 2: presents the ATC data processing requirements: the anticipated traffic, the computer processing rates, and the methods for analyzing computer performance;

Section 3: describes current computing systems with capabilities for usage in near future ATC applications;

Section 4: denotes the algorithms which are to be used in the projected ATC programs;

Section 5: sums up the future prospects in ATC data processing, assesses the risks and points out some future work efforts.

17. Key Words Computers NAS Stage A Data Processing ARTS III Data Processing	18. Distribution Statement

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 152	22. Price

Form DOT F 1700.7 (8-69)

TABLE OF CONTENTS

ADVANCED AIR TRAFFIC CONTROL COMPUTER SYSTEMS

1. OBJECTIVES AND DEFINITIONS

1.1 INTRODUCTION

The study effort under FA-03-01, Large Scale Systems, has resulted in this Final Report describing large computing systems and software anticipated for the early eighties. Features expected to be found in large real-time computers of that era which could handle the large ATC data processing loads forecasted are emphasized. Software advances which will affect the situation at that time are also included. Extrapolations are made from current computer hardware, which is described in detail. Criteria are developed for evaluating the future systems.

1.2 OBJECTIVES AND APPROACH

The objective of this report is to provide information on hardware/software systems applicable to fourth generation ATC data processing systems. The approach is through studies of large real-time computers, ATC computations and algorithms, and criteria for evaluating large-scale systems.

Following a discussion of ATC data processing requirements in Section 2, a detailed presentation is made in Section 3 of the significant features of a number of commercial, military and experimental machines having capabilities responding to future ATC needs. Computers described are the IBM 9020, 360/195, 370/165, ARTS III, CDC 7600, STAR, TI Advanced Scientific Computer, PEPE, the Goodyear Associative Processor and ILLIAC IV. Correlation and tracking algorithms are reviewed and compared in Section 4 with some discussion of possible improvements.

It is hoped that the repertory of techniques presented here will be useful in defining new systems to meet future ATC requirements as they evolve.

The remainder of Section 1 is devoted to brief definitions of some of the terms used in present-day descriptions of computation systems.

1.3 COMPUTER HARDWARE

The major components of computers will be reviewed. These include memories, processors, I/O processors, I/O controllers, and I/O devices.

1.3.1 Memory. Memory can be broadly divided into two classes: main (memory) storage characterized by high speed, and secondary (auxiliary) storage characterized by large storage capacity.

Main storage is a general-purpose, addressable, read/write storage medium used to hold programs and data. It may be thought of as an array of N, M-bit, storage registers. Information in a given register may be accessed by specifying the address or location of the information in the array. The contents of one of the m-bit registers is the basic unit of storage. Main storage is random access, i.e., the time required to access a basic unit of information is independent of the location of the information in the array. Each processor is provided a dedicated processor-to-memory data path, independent of I/O, to access information in main storage.

Random access memories are either read-only (ROM) or read-write (RWM). The information in a ROM is inserted at the time the memory is manufactured. It can only be changed by physically altering the memory. For equivalent size and speed, read-only memories are usually lower in cost when mass produced in identical patterns, than read-write memories. Read-write memories which are electrically alterable, sometimes are operated as ROM's in computer systems.

Memories are considered volatile if they lose information during lapses of power, non-volatile if they do not. Memories which use magnetic storage elements (cores, plated wire, etc.) are usually non-volatile and memories which use active storage elements (semiconductors) are usually volatile.

Non-volatile memories can be further classified as either destructive read-out (DRO) or non-destructive read-out (NDRO) memories. When information is read from a location in a DRO memory the information stored in that location is destroyed by the read-out process. To preserve the stored information, each read-out operation is normally followed by a restore (write-back) operation. This extra operation lengthens the cycle time of the memory.

Core memories operate in the DRO mode, semiconductor memories in the NDRO mode, and plated-wire memories in both modes. Computer main memories have cycle times between 1/2 and 10 microseconds, and access times between 1/4 and 5 microseconds. The size of these memories is between 10^4 and 10^8 bits. Ferrite core memories still rule as the chief form of computer main memory.

Recently, IBM[1] has employed semiconductor memories as high-speed main storage buffers. The largest of these has a 32K byte capacity and a 54 nano-second cycle time. Plated wire memories have found use in military computer systems where severe environmental constraints limit the other technologies.

Secondary storage made up of magnetic disks, drums, and tape, are of large capacity, have long access times and are accessed via input-output instructions. Data on secondary storage devices are transferred to main storage, a block of words at a time. Information is stored in volumes. A volume is a unit of auxiliary storage accessible to a single read-write mechanism. A dismounted volume is referred to as external storage.

A sequential-access secondary storage device (magnetic tape) accesses data by linearly positioning the volume relative to the read-write mechanism, i.e., moving the volume forward or backward until the desired physical record is found. The time required to access a record is therefore dependent on the position of the desired record relative to the record presently under the read-write mechanism.

Direct access secondary storage devices (disk, drum, etc.) allow random access to data stored on a volume. Information is accessed by referencing its location in the volume. In this respect, direct access devices are similar to main storage. The time required to access information may or may not be dependent on the location of the information. For example, the time required to access information on a disk includes seek time, the time required to position the read-write mechanism over the correct disk track, and rotational delay--the time required for the information on a track to rotate to a position under the read-write mechanism.

---

1 IBM Systems Journal, Vol. 7, No. 1, 1968, Structual Aspects of the System/360 Model 85 II The Cache by J.S. Liptag.

Secondary storage devices may be characterized by three parameters. They are: access time--the time required to position the first basic unit of desired information under the read-write mechanism; transfer rate--the rate in bits per second at which information can be read/written from/into a volume once access has been obtained; storage capacity--the number of bits of information that can be stored on a volume.

The characteristics of present day secondary storage devices are listed below.

DEVICE	CAPACITY	ACCESS TIME	TRANSFER RATE
Magnetic Tape	50-400 M bit	10 MS-3 MIN	8K-2M bit/sec
Disk	10-6000 M bit	20-500 MS	0.5-16 M bit/sec
Drum	1-150 M bit	8-60 MS	2K-20M bit/sec

In addition to main storage and secondary storage, a third form of storage known as an associative (content addressable) memory has recently come into use. Information in an associative memory is accessed by content rather than by address. Each word which is loaded into an associative memory contains one or more keys which will be used to access the information stored in the remainder of the word.

KEY	DATA

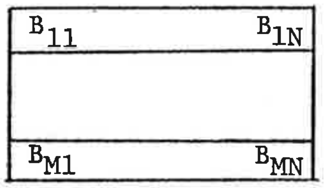
ASSOCIATIVE MEMORY DATA WORD

To access a data word the key is presented to the associative memory. The memory then replies by providing the data word that matches the key. If two or more items in the memory can have the same key then the associative memory must perform some form of multiple match resolution so that only one reply results. Associative memories are very useful when table lookup or fast searches must be carried out.

The purpose of an associative memory is to provide an aid to a computer with its powerful simultaneous parallel search capability. The associative processor is a further extension to the associative memory in that a limited (serial) arithmetic capability is added. There are several recent applications of both associative memories and associative processors.[6-12]

The basic cell of an associative array is a flip-flop and a match circuit. The cells are arranged in the array as they would be in a normally addressed memory array, N bits per word, M words long, but with columnar linkages in each bit column so that each bit of an argument word can be matched with the equivalent bit in each of the memory words. Where the totality of the argument bits match the memory word bits, a match condition is met and a flag is raised. It then remains for a multiple match resolver to home in on one. To express this mathematically and symbolically, it can be shown that:

given word $A_1, \ldots A_N$, an N bit word to be matched, from a memory B_{MN}



$$\text{Totality of Matched Words} = \sum_{j=1}^{M} \prod_{i=1}^{N} A_i \oplus B'_{ij}$$

$$\text{Totality of Matched Words} \rightarrow \begin{matrix}\text{Multiple}\\ \text{Match}\\ \text{Resolver}\end{matrix} \rightarrow \text{one matched word}$$

Plated wire memories were used to fabricate associative memories. Now semiconductor chips are being made, but they are expensive. They are 20 times the cost of ordinary semiconductor memory chips. However, the price is due to come down shortly and the use of them is spreading. It will not be long before associative memories and associative processors become integral parts of a computer.

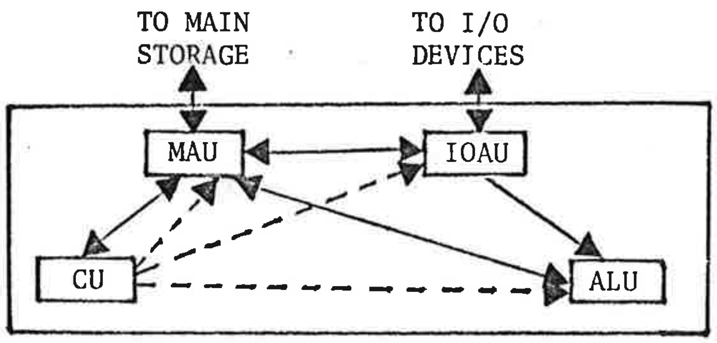
1.3.2 Processors. The processor units of a computer are those elements which decode and execute the instructions of a program. Processors can be classified as general-purpose processors, which are capable of solving a large class of problems, or special-purpose processors which are capable of solving a restricted class of problems. A general-purpose processor can be thought of as several special-purpose processors operating under common control. Processors contain one or more of the following units: memory access units (MAU), computational or arithmetic and logic (ALU), input/output unit (IOAU), control or instruction unit (CU). The number of units of each type used and the number of data paths interconnecting the units are important in determining the instruction execution rate of processors. The control unit decodes instructions and issues the micro-orders (control pulses) needed to cause execution of the decoded instructions by the processor subunits. The micro-orders cause the processor to fetch the next program instruction from memory, decode its operation code, fetch any required operands, carry out the operation, and store the result in the proper location. Thus, the control unit in conjunction with the processor automatically sequences through the program in main storage. A processor may have a mixture of general-purpose computational units and special-purpose units. Special-purpose units are used to decrease the time required to perform certain operations, e.g., complex arithmetic operations. The memory access unit and the I/O access unit handle all processor-to-memory and processor-to-I/O device communications.

Fast processors are composed of special-purpose execution and control elements that can be operated concurrently. Concurrent operation is achieved by providing each concurrent unit with its own local storage (hardware registers) and a local control unit. Operation of the various concurrent units is synchronized by the main control unit of the processor.

There is a large disparity between processor computational speed and I/O device operating speed. Therefore, most processors contain one or more special-purpose subunits known as I/O processors or channels. These carry on I/O operations in parallel with processor computations. They do this by executing I/O programs which are setup in advance by the computation element of the processor. On some computer systems with more than one processor, a complete general-purpose computer is dedicated to the task of performing I/O operations. When it is used in this fashion, the general-purpose computer is called an I/O processor.

1.3.3 Processor Organizations. In this section several processor organizations will be discussed.

The organization of the conventional processor is presented in Figure 1. The processor contains a memory access unit (MAU), input/output access unit (IOAU) arithmetic & logic unit (ALU), and a control unit (CU).



_____ DATA PATH

----- CONTROL PATH

CONVENTIONAL PROCESSOR ORGANIZATION

FIGURE 1

It processes program instructions in a sequential manner. A typical sequence of operations required for processing an instruction proceeds as follows:

1. The address of the next instruction to be fetched is generated from information in the processor's control unit at the time of the last instruction executed.

2. The instruction is fetched from memory and placed in the control unit.

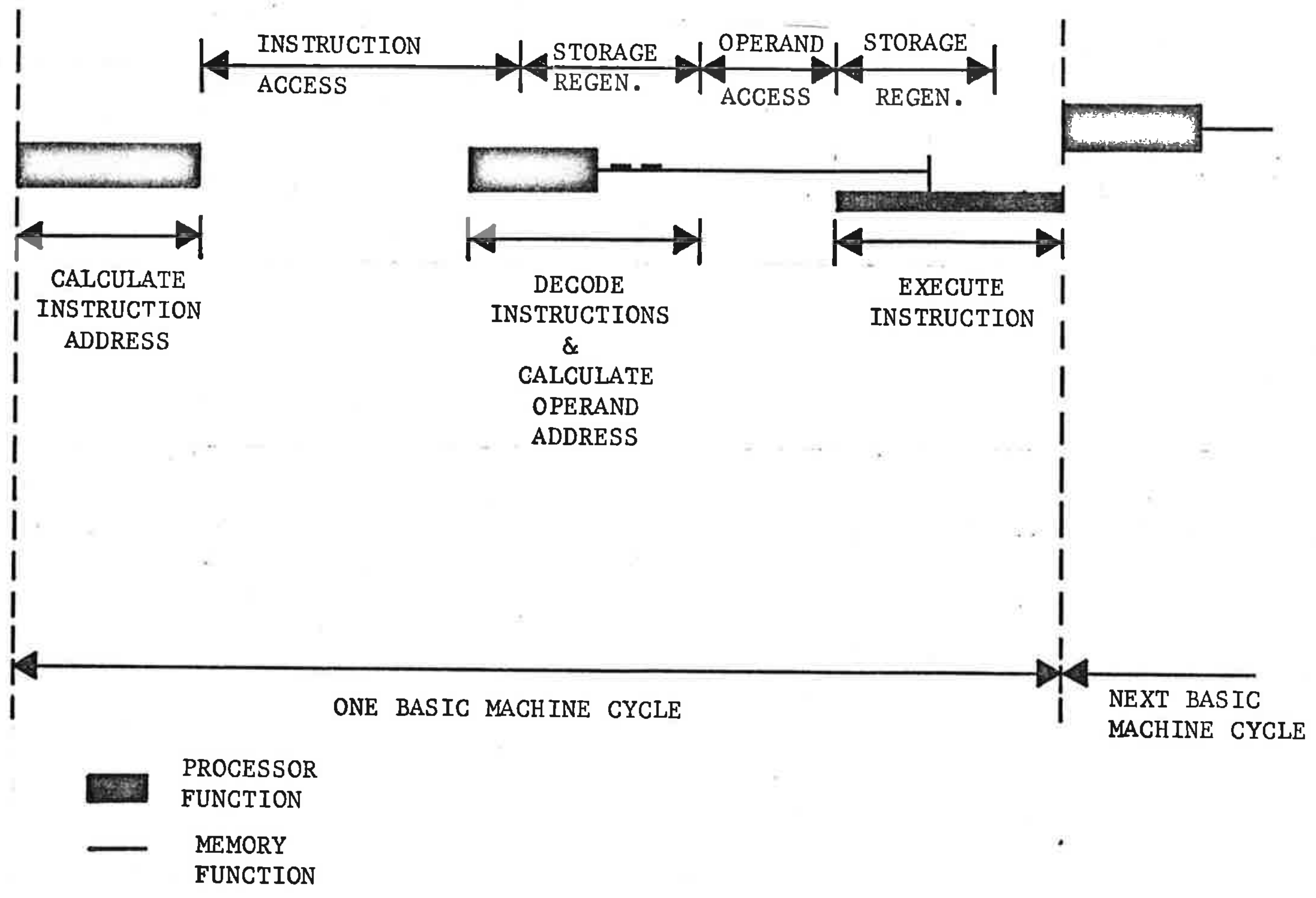
3. When the instruction arrives at the control unit, it is decoded. If an operand is required, its memory address is resolved.

4. The operand is fetched from memory and placed in the ALU.

5. The result of the operation is stored in the proper location, either in main memory or in a hardware register.

6. Steps one thru five are repeated.

Steps 1 through 5 represent the basic cycle of the machine. A timing diagram which graphically depicts both memory and processor functions during two basic processor cycles of operation is shown in Figure 2.



TIMING DIAGRAM FOR PROCESSOR BASIC CYCLE OF OPERATION

FIGURE 2

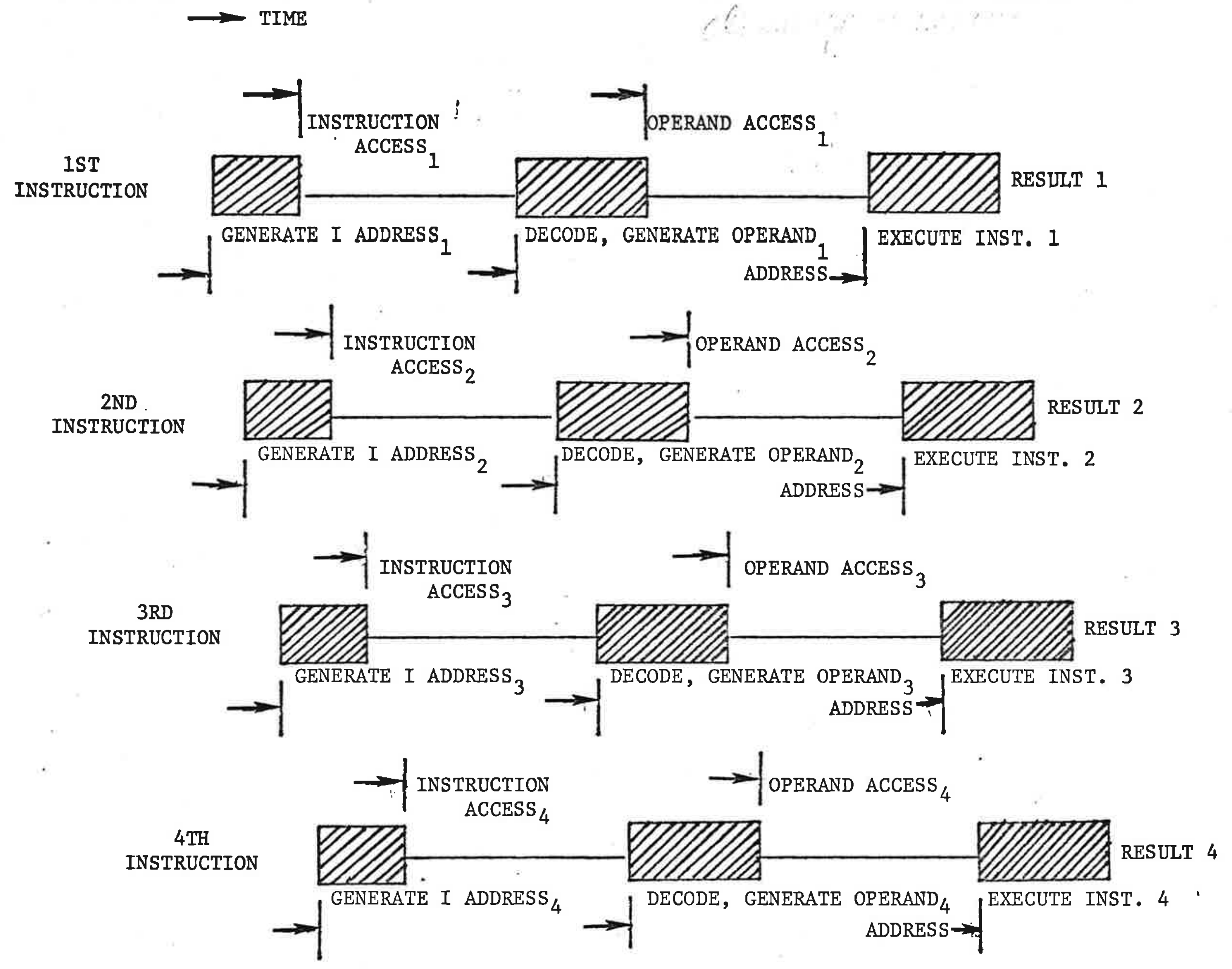


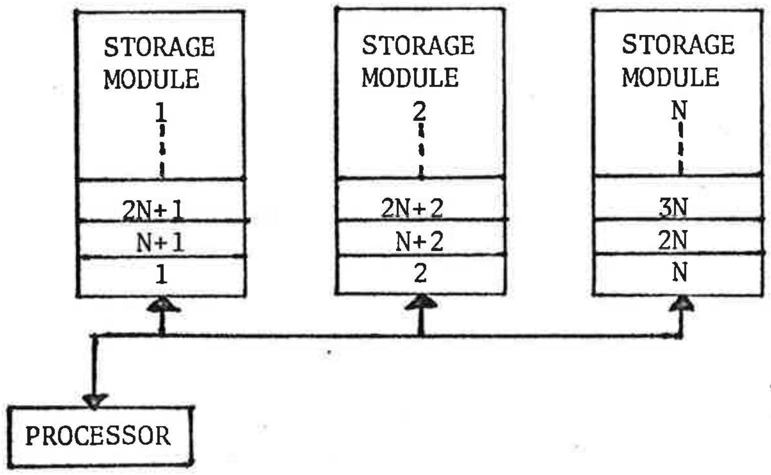
ILLUSTRATION OF CONCURRENCY AMONG SUCCESSIVE INSTRUCTIONS

FIGURE 3

As can be seen, an appreciable portion of the basic cycle is spent waiting for instructions, operands, and results from operations. Two bottlenecks to more efficient utilization of processors resources are apparent from this diagram. They are the memory-to-processor data rate and the long execution time of the instructions.

The concurrent processor uses organizational techniques to eliminate these bottlenecks. The concurrent processor allows several instructions to be in various stages of processing within the processor at the same time. A timing diagram which illustrates the concurrency among successive instructions is shown in Figure 3. As can be seen by comparison with Figure 2, the effective instruction execution rate has been increased many times at the cost of some additional hardware.

The organizational techniques used to achieve concurrency are: memory interleaving, instruction fetch-execution overlap, execution concurrency and pipelining. Memory interleaving (see Figure 4) is a technique used to increase the effective data transfer rate between main storage and the processor.

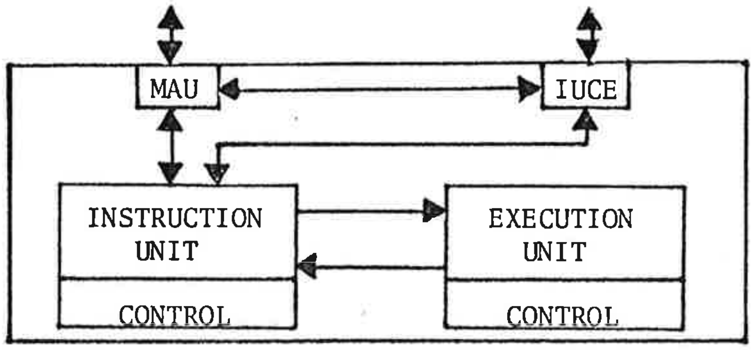


INTERLEAVED MEMORY ADDRESSING STRUCTURE

FIGURE 4

It is implemented by splitting main storage into several independent modules and arranging the addressing structure so that n successive words are fetched from n different modules in one memory cycle. Due to limitations on hardware registers, the n words from the n different modules are stored in registers during submultiples of the memory cycle. A delay of one memory cycle is still incurred when fetching and storing information. But by assigning successively used instructions and data to successive storage locations (i.e., different modules) whenever possible, advantage may be taken of the memory interleaving. If program branches are infrequent, then the processor-to-memory data transfer rate can be increased by a factor that approaches the degree of memory interleaving. The degree (or amount) of interleaving required for a certain amount of processor concurrency is dependent on the memory cycle time, processor request rate and the desired effective storage access time.

Within the processor, the natural way to increase instruction execution rate is to overlap instruction fetching and execution. This is accomplished by partitioning the processor (see Figure 5) so that instruction fetching, decoding and operand fetching are carried out by one unit, and actual execution by a second unit. Each has its own local control unit so that it can operate in an independent fashion. The units operate as follows. The instruction unit prepares an instruction for execution. It fetches any required operands from memory. When the necessary preprocessing is complete, the instruction unit tests the execution unit to see if it is busy. If the execution unit is free, the instruction unit tells it to begin processing the instruction that was just prepared. Immediately after acceptance of the command the instruction unit starts preparation of the next instruction. Both units are now processing concurrently. If the instruction unit finishes preparation of the next instruction before the execution unit completes its processing, it must wait. If the execution unit finishes first, then the instruction unit can initiate the execution of the next instruction and start the preparation of another instruction. Whenever a branch instruction is encountered, the instruction unit assumes the branch will not be taken and continues to fetch instructions from the same sequence in storage. If the branch is taken, the instruction unit discards the now-irrelevant data and initiates processing at the branch location.



PROCESSOR WITH FETCH-EXECUTION OVERLAP

FIGURE 5

The technique just discussed is known as a single instruction overlap. Sometimes, this is not enough. The techniques of pipelining, internal buffering and execution concurrency can be used to overcome the limitations imposed on the inability of the instruction unit and execution unit to perform at a desired rate.

Execution concurrency is a technique used to increase the processing capability of the execution unit. To achieve it, the execution unit is divided into several special-purpose processing elements, etch capable of executing a subset of the processors total instruction set. Each element is provided with its own local unit so it can operate in an independent manner. Operations in the individual elements are initiated by commands received from the instruction unit. The synchronism required to insure that instructions are executed in the correct sequence is provided by tags and interlocks within the various units and the instruction unit.

Since some elements are designed to execute special instructions, execution concurrency, i.e., several instructions being executed at the same time, can be achieved when a proper instruction mix is applied. For instructions which require long execution times, a special execution unit is implemented; and to balance the total operation, a certain number of common execution units according to the average mix of instructions are used.

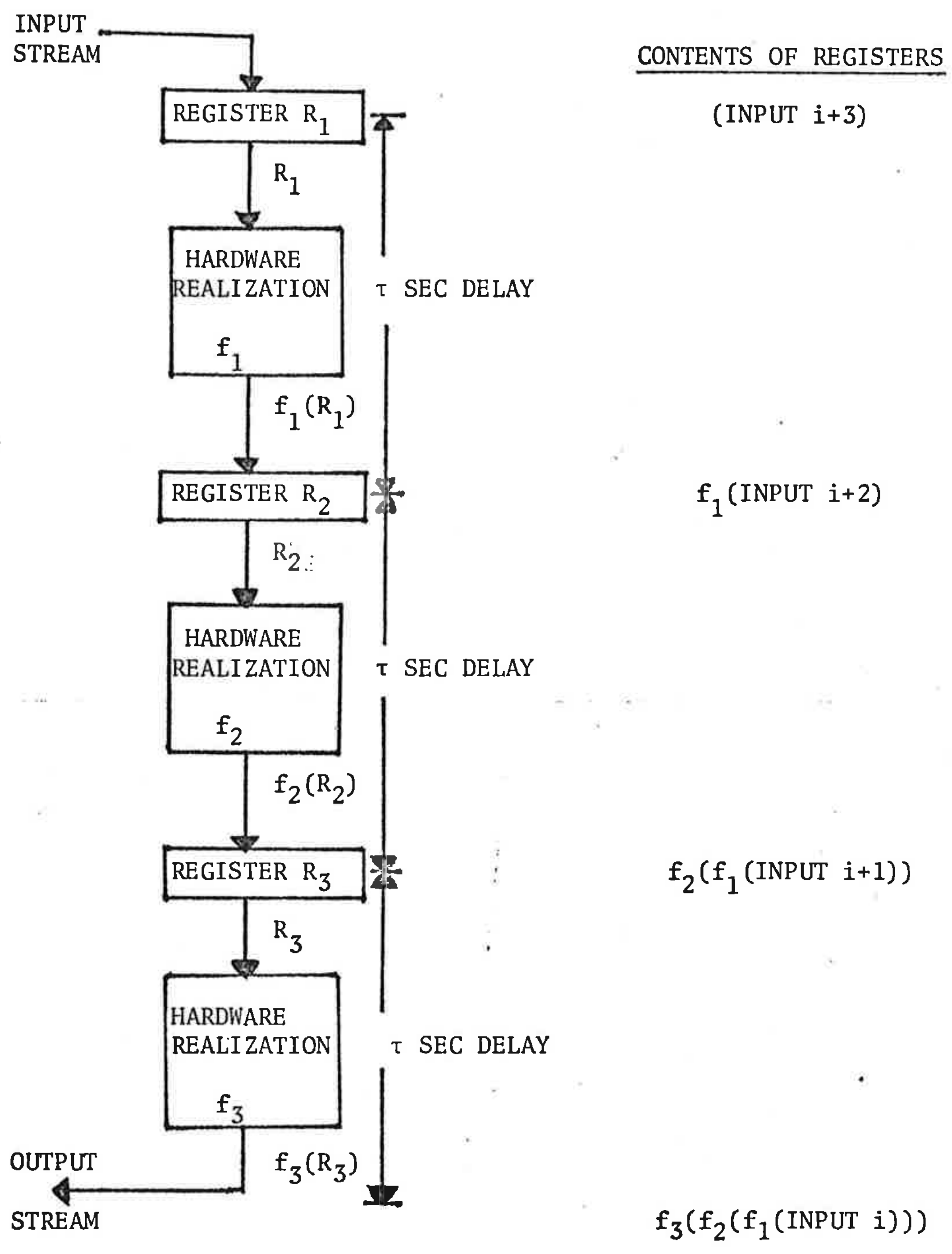
The ability to process more than one instruction at a time in the instruction unit and in individual execution elements can be achieved by using pipeline techniques.

To implement a pipeline execution element or instruction unit (see Figure 6), the function to be performed by the unit, must be broken into several processing steps, f_1, f_2, f_3, that are hardware implementable. Temporary storage registers R_1, R_2, R_3 are then installed between the hardware units that perform the between-step processing. Processing proceeds from time, $t = t_o$, when the first instruction and data set is loaded into register R_1. At time $T = t_o + \tau$, where τ equals the pipeline interstage delay, the second instruction and data is loaded into R_1 and the output result from the first processing step, $f_1(R_1)$ is baded into register R_2. At time $T = t_o + 2\tau$, a third instruction and data set is loaded into register R_1 and $f_1(R_1)$ resulting from the second instruction and data set is loaded into R_2 and $f_2(R_2)$ of the first instruction set is loaded into register R_3. Three instructions are now ar various stages of processing within the pipeline unit. At time $t = t_o + 3\tau$, a fourth instruction set is loaded into register R_1 and $f_1(R_1)$ of the third instruction set is loaded into register R_2 and $f_2(R_2)$ resulting from second instruction set is loaded into R_3 and the final result of the first instruction $f_o = f_3(R_3)$ is made available at the output of the pipeline unit. The pipeline is now full and as long as a steady stream of instruction and data are available, results will come out of the pipeline at intervals of τ seconds.

In the example presented, a three-stage pipeline unit, τ would be approximately one-third the time required to process an instruction and data in a non-pipelined unit. This means that once the startup transient has died away, a pipeline processing element can process instructions and data at approximately N times the rate of a non-pipeline unit, where N equals the number of stages in the pipeline. A processor which uses pipeline techniques to achieve execution concurrency is called a pipeline processor.

When using any of the previously mentioned techniques, internal buffering in the form of high-speed registers, is usually required to smooth the flow of data and instructions between the various processor elements. The number of stages of buffering required depends on the organization of a processor, the capability of the instruction unit, the desired rate of instruction execution, and the techniques used to preserve the proper order between instructions.

Processors which use any or all of the above techniques are called highly serial processors since they process instructions in a sequential manner.



FUNCTIONAL BLOCK DIAGRAM OF A PIPELINE PROCESSING UNIT

FIGURE 6

The Control Unit (CU) in Figure 1 decodes machine-language instructions and interprets them at the circuit level in terms of the data transfers and control signals required to execute the instructions. In many systems a separate hard-wired logic circuit is used to implement each instruction in the machine language. Microprogramming can be used to simplify the design of the CU, however, and is finding increasing use in modern machines. The essential idea is to use a stored program of microinstructions, each of which sets gates to define a path along which data or control signals will travel. Microprogramming not only reduces the amount of circuitry required for the CU but also makes it economical to provide a richer instruction set at the machine-language level.

A completely microprogrammed instruction set also makes it possible for one computer to emulate another, i.e., for one computer to perform calculations and logical operations according to the system architecture and instruction set of some entirely different type of computer.

Microprogramming may be implemented using read-only storage, or writable storage. In the latter case changes or additions to the microcode are quickly made, whereas in the read-only case changes involve mechanical replacement of storage units. If a volatile semiconductor medium is used for high speed, it must be reloaded each time the power is turned on, as e.g., in the IBM 370/135 and 145 which use a special console disk unit for this purpose.

1.3.4 I/O Channels and Processors. Early computers carried on I/O operations by having the central processor execute the I/O instructions. Whenever I/O instructions were encountered, the processors would become dedicated to the I/O task which would mean operating at a reduced rate. Handling the I/O greatly reduced the effective speed of processor operation.

To overcome this, computers were adapted with hardware units known as channels which allowed I/O operations, once initiated, to proceed in parallel with processor computation. In the simplest form, a channel is capable of executing a single I/O instruction transferred to it from the central processor. More sophisticated channels are capable of executing strings of instructions which are stored in memory. Some computers even have small dedicated processors known as I/O, peripheral, or satellite processors which carry out the processor-to-memory data transfers.

Most medium and large-scale computers have one or more channels that carry on data transfers under the control of a channel program which is executed independently of the central processor program. The central processor exercises control over the operation of the channels. To do this, it has facilities (hardware and software) for initiating the operation of a channel and for the receiving of completion signals (interrupts) from the channel. The processor on receipt of the interrupt stops processing and analyzes the cause of the interrupt. The cause of an I/O interrupt may be due to a normal task completion or due to some form of error of hardware failure. If the interrupt was a normal termination then both the data and the channel become available for use by the processor.

The instruction executed by the channels are of the following types: data transfer commands, sense commands, control commands, change of sequence commands.. Data transfer commands are read or write commands which cause data to be transferred between the I/O devices and main storage. These commands specify where in memory the data is to be placed and the amount of information to be transferred. Sense commands carry information into main storage concerning the status of a channel. Control commands specify device operations such as backspace, rewind tape, advance paper, etc. Change of sequence commands specify the location in main storage from which the next instruction is to be fetched. Both conditional and unconditional branches are handled in this manner. Channels are usually one of two types--selector or multiplexer.

A selector channel is a type of channel which is capable of carrying on data transfers to one device at a time on a dedicated basis--several devices may be connected to a selector channel, but only one may be active during a given channel program. Transfers are initiated by the central processor, which supplied the address of the first instruction of the channel program that will control the transfer and the identification of the device that will be used for the transfer. The channel then takes control and executes the channel program concurrent with CPU processing. When the program is completed, the channel notifies the CPU so that another I/O operation can be scheduled. A selector channel is said to operate in a burst mode because the channel is tied up for the entire duration of the transfer including the time spent waiting for the device to complete its mechanical operation.

A multiplexer channel is capable of carrying on data transfers with several devices at seemingly the same time, in a time multiplexed basis. Low data rate devices then share the same channel facilities. Information is transferred to individual devices by broadcasting, simultaneously over the same set of data lines, the data and the identification of the device involved. Associated with each device is a dedicated group of storage locations for the storage of all information required to maintain an independent transfer to a device. The stored information consists of the address of the next channel instruction, channel status information, and the channel instruction presently in process, as well as information such as the base address of the data buffer in main storage, its size, and the operation being performed by the instruction. This information is updated by the channel each time a new instruction is executed or a data transfer to the device takes place. Transfers to the individual devices are scheduled automatically by the channel as the devices become available to accept or transmit data. Programs are initiated and terminated in the same manner as on a selector channel except the multiplexer channel continues to process any remaining active channel programs after notifying the central processor of completion of one channel program.

Satellite computers and I/O processors operate in a manner similar to a multiplexer channel except that the processor can also perform editing and formatting functions on the data before it is transferred. More flexible scheduling is also possible.

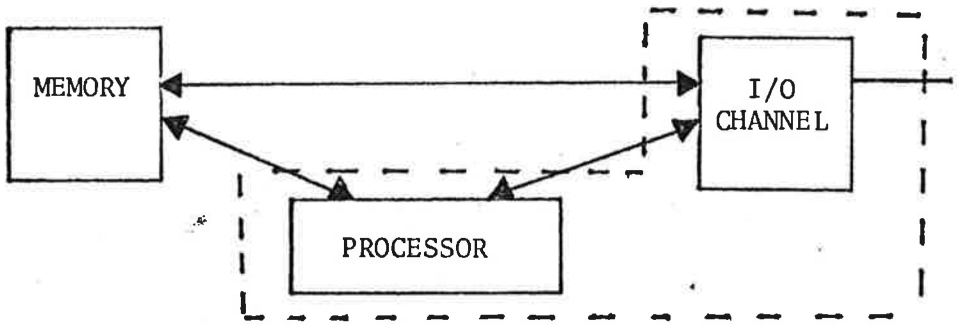
1.3.5 I/O Controllers. I/O controllers are special-purpose devices which interpret the commands broadcast by the I/O channels. They are located between the I/O devices and the channel to which they are connected. They provide the timing and sequencing signals needed to make the I/O devices carry out the operations specified by the channel commands.

1.3.6 I/O Devices. Into the category of I/O devices fall drums, disks, card readers, large core memories, etc., and transducers of various types. They are peripheral to computers and are connected to them via I/O channels. The common characteristics of these devices are response times, low data transfer rates, and varied control sequences.

1.4 SYSTEM ORGANIZATION

1.4.1 System Configurations. There are a large number of system configurations (organizations) currently in use. Among these appear four major configurations.

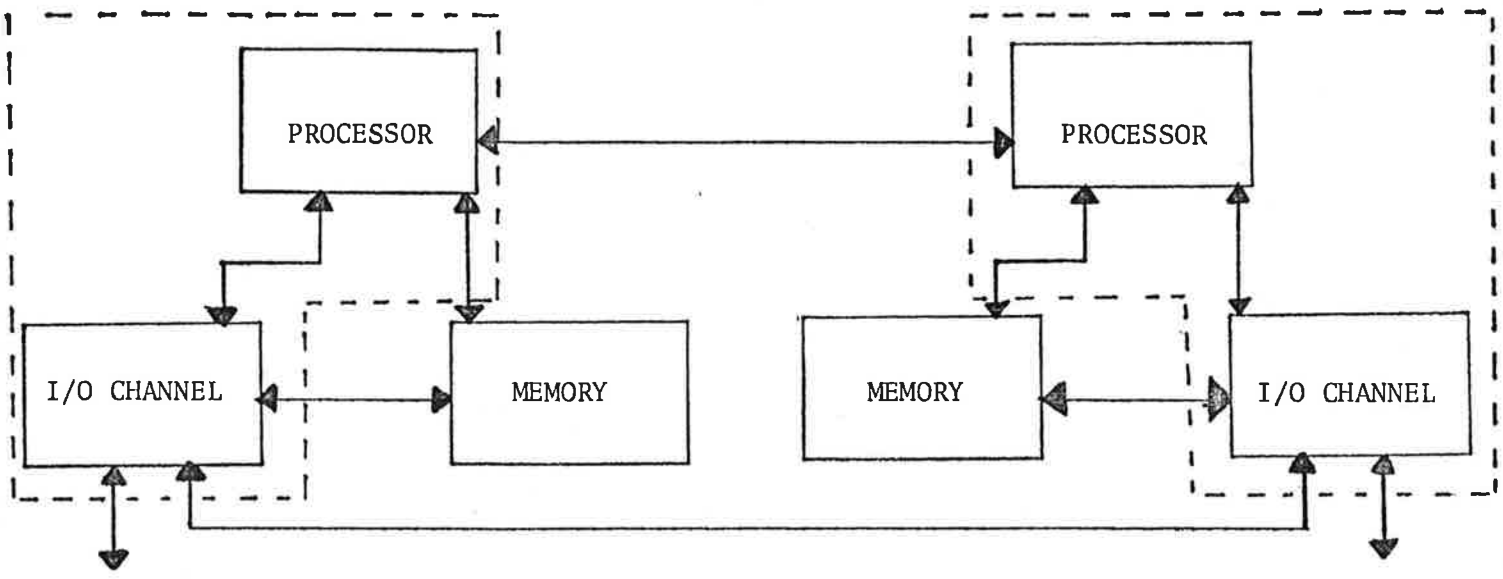
The simplest and most often used is the single computer or uni-processor system which is shown in Figure 7. The processor is connected to memory and to the I/O channel; the memory communicates with the processor and the I/O channel. Connections to the I/O devices are not shown. The processing capability of this configuration is limited by the capability of the single processor in the system.



UNI-PROCESSOR CONFIGURATION

Figure 7

To increase processing capability, computer manufacturers went to the multicomputer system configuration (see Figure 8). In this system, the processors are often of different size. Normally, the smaller of the two carries on the I/O and housekeeping functions while the larger acts as the primary computation element of the system. The two processors are linked by processor-to-processor



MULTICOMPUTER CONFIGURATION

Figure 8

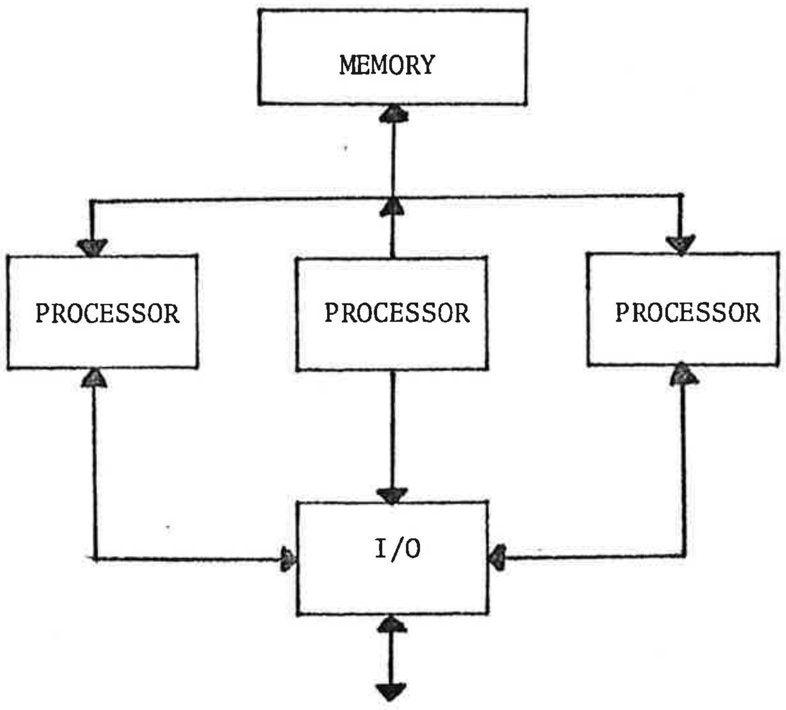
and channel-to-channel data links. The channel-to-channel link is used to transfer information between the main storage units of the two computers. I/O devices may also be shared via this link. The processor-to-processor data link permits one processor to notify the other that it is wanted to do something. It is most often used to set-up storage data transfers.

If both computers are identical and perform large amounts of computations, then the system is called a federated or loosely-coupled dual computer system. This configuration is often used in fail safe processor systems. When used as such, the processor-to-processor data link is used for status testing and failure notification.

A multiprocessor system configuration is shown in Figure 9.

Both memory and I/O channels are shared by all processors on equal basis. The processors are often identical, however, they need not be. If they are, the operating system assigns tasks to the processors as they become available. Because resources are shared, the operating system must resolve the inevitable conflicts, if the system is to operate. An efficient multiprocessor must minimize the overhead incurred in resolution of these conflicts.

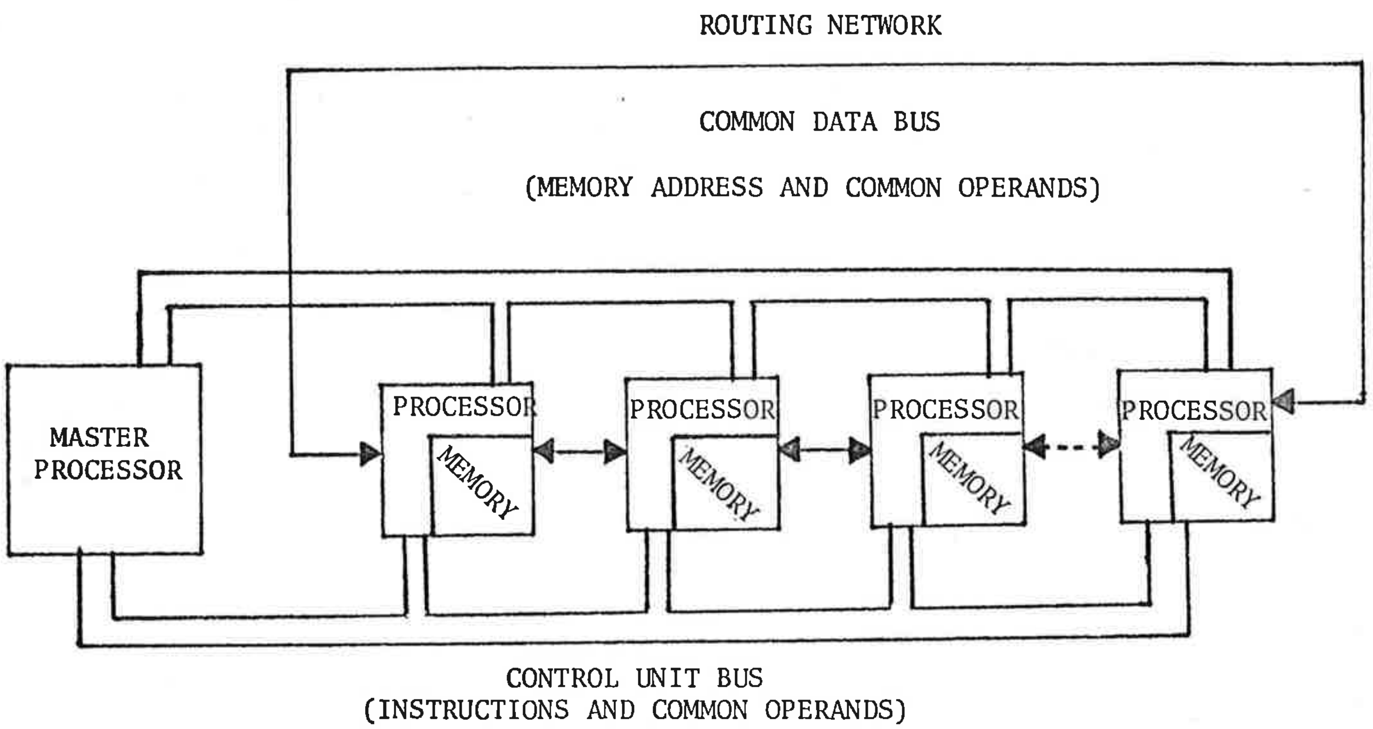
An organization which differs radically from any of those discussed previously is that of the array processor shown in Figure 10. It is composed of N processors, each of which has its own memory and is able to communicate with



MULTIPROCESSOR CONFIGURATION

Figure 9

its nearby neighbors, and a master processor which controls the operation of the system. Data is entered into the individual processor memories from the master processor. The master processor issues the commands that control the operation of the individual processors. Commands are global in nature, i.e., the command calls all processors to perform the same operation. In addition, most parallel processors have a masking capability which allows them to inhibit execution of a global command. This facility allows the processors to perform independent operations. Processors of this type are most effective when performing vector or matrix operations.

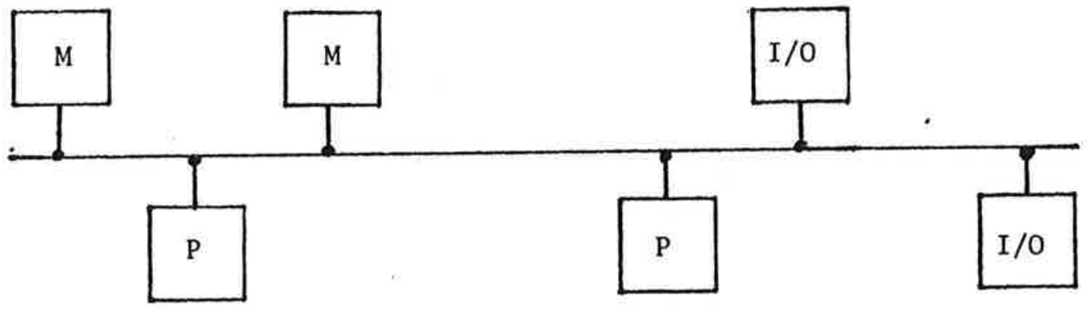


ARRAY PROCESSOR CONFIGURATION

Figure 10

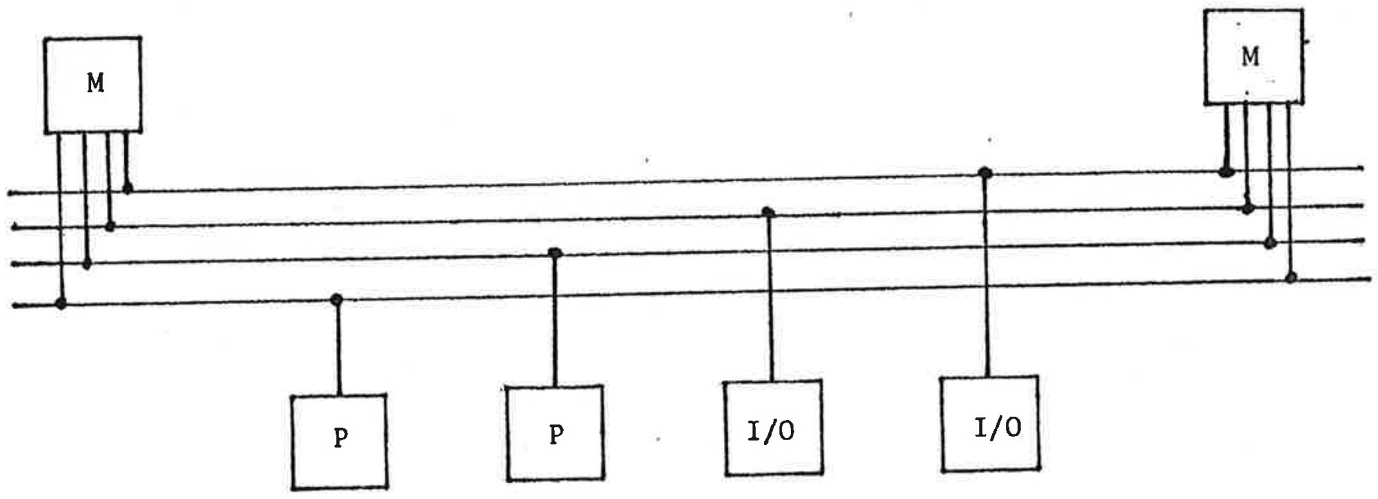
1.4.2 Interconnections. Three interconnection techniques used to tie processors, memories, and I/O processors together are shown in Figures 11 through 13. The simplest of the configurations conceptually is the common data bus shown in Figure 11. All modules are connected in parallel to the common bus. The bus may be a word, byte, or bit wide depending on the desired data transfer rate. Narrower busses require complex control hardware. Access to the bus is granted to processors, one-at-a-time, on a time multiplexed basis. Assuming a fixed memory cycle time, one possible procedure for controlling access is to allow each processor access to the bus on successive memory cycles in a round robin fashion, i.e., P_1, P_2, P_3, P_4, P_1, P_2 ... etc. If there are N processors then each will be able to access memory once every N memory cycles. Herein, lies the primary disadvantage of the common data bus, i.e., the waiting time for information increases as the number of units on the bus increases. This is offset by the ease with which additional modules may be added to the bus.

Figures 12 and 13 show two interconnection methods which are essentially the same. The multiport memories allow individual processors to access different memory modules at the same time. This is accomplished by providing multiple inputs to each memory module and separate buses for each processor. A processor's bus connects to one port on each memory module. If more than one processor requests the same memory module, then hardware switching networks within the module resolve the conflict and grant access to only one of the requesting processors. This interconnection method overcomes the data rate limitation of the common bus. However, it has the disadvantage that the number of modules which can be added to the system is limited by the number of parts which were designed into the multiport memory module. The crossbar style interconnection technique overcomes this limitation. The crossbar network is a modular switching network which can grow to accommodate new memory and processor modules. One crosspoint switching network is added for each desired processor to memory data path. A crossbar may be full or partially full depending on the number of desired data paths. The memory modules are single port. Conflicts are resolved within the crosspoint switch in a manner similar to that used in the multiport memory module on a column basis. A variety of conflict resolution techniques is possible.



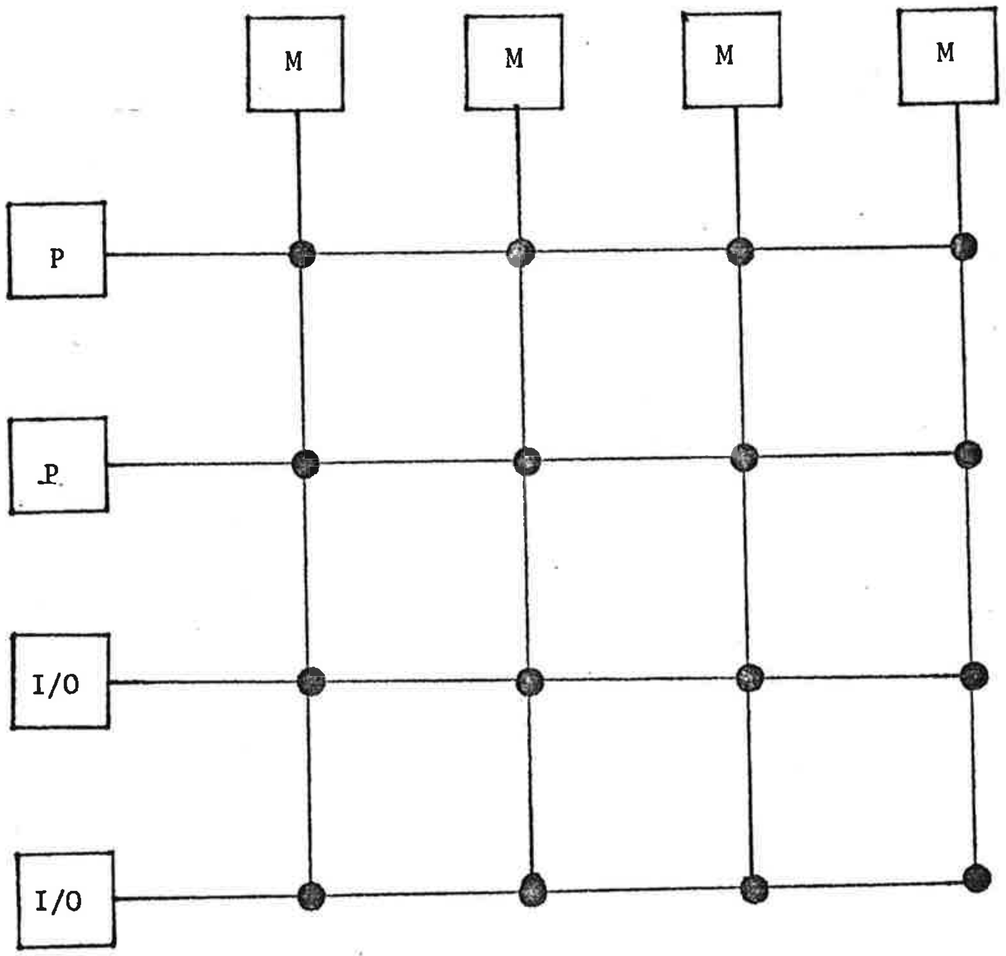
COMMON DATA BUS

Figure 11



MULTIPORT MEMORY CONNECTION

Figure 12



CROSSBAR CONNECTION

Figure 13

1.5 COMPUTER SOFTWARE

One of the sources of the considerable power inherent in general-purpose digital computer systems has been the concept of the stored program, it being set forth in virtually its present-day form by von Neumann in 1948 [1]. Then as now, the program was a series of coded instructions stored in the memory of the computer which, when interpreted in sequence, caused the computer to (1) accept data from an outside source, (2) perform calculations on the data, (3) modify its own sequence of operations on the basis of the data, and (4) present data to an outside user. Of these, it is the third--the ability to "interpret" the data and act in various ways depending on the "interpretation"--that is the key the unique power of the computer system, for with it the computer may assess alternatives and make choices based on complex sets of criteria. This, coupled with the great speed of the hardware, allows the properly programmed computer to accomplish tasks of previously unthinkable complexity.

5

1.5.1 The Development of Computer Programming. The first computer programs essentially reproduced the patterns which were stored in memory. In the binary machines such as von Neumann's, the entire program could be written as a series of zeros and ones. The octal system of numbers was soon adopted, however, for the representation of addresses and numbers in the computer, and a system of alphabetic characters was used to represent the instruction codes. Programs thus denoted were said to be written in machine language and the coding was said to use absolute addressing.

The difficulty with coding in this way was quickly apparent, for not only was the resultant series of letters and numbers an unnatural form for a man to interpret, it was also almost completely unyielding when it became necessary to make changes or to correct mistakes. The solution was to allow the man to deal in symbols for instructions and addresses and to let the computer itself make the transformation from the symbols to the absolute numbers they represented. The computer program that did this task was called a symbolic assembler; it dealt with programs prepared in assembly language using the technique of symbolic addressing.

The simple assemblers of the late 1950's were gradually extended with the incorporation of pseudo-operations (instructions to the assembler itself rather than parts of the program being assembled), conditional assembly (actions conditioned on the states of switches, the values of constants, the progress of the assembly, etc.) and in conjunction with appropriate loaders, relocatable routines (ones whose exact locations in core are determined at load time--hence ones which can be easily used as part of many different programs).

The development of macro-assemblers during the 1960's lead to today's very powerful assemblers which provide the skillful programmer with tools with which to write efficient and capable programs to handle the most complex problems. The macro-assembler is able to recognize system and programmer defined macro-instructions, or simply macros, which are names given to certain patterns of coding which are used again and again in the program. The use of these macros made coding easier by reducing the effort needed to write the code, by making the coding less error-prone and by making changes easier to incorporate. Extensions in the form of nested macros, redefinitions within macros and conditional macro expansions made the concept from a coding convenience to a useful and powerful tool.

However, the operative word in the paragraph above as applied to the programmer is "skillful". As the computer systems became larger, faster, more capable, the user of the system found himself less able to cope unless he were a professional programmer--a situation which led to intensive work in two areas: high-order languages and operating systems. The former were languages intended to allow the non-programmer specialist to write programs for computers without concern for the mass of detail implicit in an assembly language program. On the other hand, operating systems were developed to manage the expensive and complex computer systems so that they might be used in efficient and reasonable ways. After a short look at the subject of data structures encountered in computer applications, we will take a more extensive look at each of these subjects.

1.5.2 Data Structures. It is the nature of the mind to organize facts (data) into structures such that inter-relationships among the facts may become clear and may be dealt with. Computers, being highly organized creations of the minds of men, also deal with data organized in various structures. A goodly part of any computer program, sometimes explicitly and at other times implicitly because of the facilities of the operating system or the programming language, is devoted to matching the structure of the data to the structures of the media on which they are stored.

The structure of a storage medium is frequently described in terms of the order in which its contents may be accessed. Clearly, the most general of all is the so-called random access memory (RAM) where each unit of data has a unique address, for upon this structure may be imposed any more restrictive form. Next, in the usual hierarchy is the block random access memory in which each relatively large collection of data may be accessed without regard to its place among the other blocks, but within the block a second level of addressing is necessary. Many drum memories are organized this way. (Note that if the bit is taken as the unit of information, the usual word or byte addressed memory is in reality block random access, with the word or byte as the block.)

The serial access memory is the last of the three forms usually found in systems today. In this form, epitomized by magnetic tape, access to any unit of information depends on having had access to all its predecessors.

Note that despite the variety of access means, the underlying addressing structure in all these storage types has been one-dimensional. That is, there is a sequence of addresses from beginning to end such that given any address, the next may be calculated by a single elementary operation (e.g., add one). Within this framework, the computer program deals with data arranged in a number of ways, the simplest to handle being the _file_, a collection of elements either unstructured (random) or sequential. Although random files on sequential storage media have been difficult to deal with, the availability of large-scale random access memories and the simplicity of the file structure have served to make this a rather straightforward case.

The _array_ is an important data structure, especially in science and engineering. A one-dimensional array is called a _vector_ and a two-dimensional array a _matrix_. Multi-dimensional arrays, though specially named, are also frequently found. One characteristic of array processing is the frequent application of the same operation to all or many members of the array at the same time. A single sequential processor must be programmed to cycle through the array to perform the required calculations--a not inconsequential task.

Other structures frequently found are strings, lists, rings, trees and lattices. A _string_ is a sequence of elements, usually characters, of variable length in which the order of the elements is the important characteristic. This structure is found in situations involving message processing or other kinds of language interpretation. Those machines with provisions to address and manipulate characters (bytes) and character groups obviously are best suited to these uses.

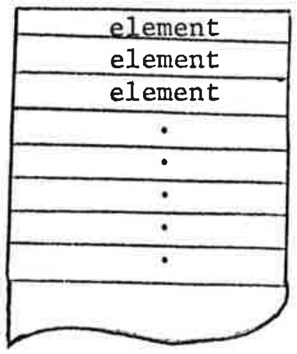
An ordered sequence of elements each of which contains a pointer to the location of the next item is called a list. Obviously, the successive elements need not be stored in successive locations in memory, but may be scattered throughout. The primary advantage of a list structure is the ease with which the list may be updated. If a new element is to be added between two pre-existing elements, it may be stored at any free location in memory and its pointer set to the location of the successor. The pointer of the predecessor is then set to this location as its new successor.

A list whose last element points back to the first, forming an endless loop, is called a ring.

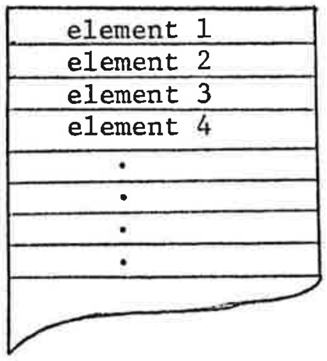
Hierarchical structures are frequently represented as trees. A tree is a two-dimensional structure each of whose elements (except first and last) has exactly one predecessor and one or more successor. The last element of each branch--the one with no successor--is called the leaf element. If the first element has a predecessor the structure is called a rooted tree, otherwise, unrooted.

If, in a tree-like structure, each element has more than one predecessor, the whole is called a lattice. The various structures described above are depicted in Figure 1.4.1.

1.5.3 Programming Languages and Applications Programming. As mentioned above, the main impetus for the development of high-order programming languages was the hope that the computer would be made more approachable by non-programmer professionals who, after all, were providing the major workload for the systems. The first of the languages so developed was FORTRAN (for FORmula TRANslator), intended to allow scientists and mathematicians to write natural and understandable programs in what looked like mathematical notation. All of the routine housekeeping functions as well as input and output were done by the compiler, based on relatively straightforward standards and conventions. That FORTRAN is now the most widely used programming language, and that it even now closely resembles its original implementation, is a tribute to the great skill with which it was conceived and executed.



a) random file



b) sequential file

$a_1, a_2, a_3, \ldots, a_n$

c) vector array

$a_{11}, a_{12}, a_{13}, \ldots, a_{1m}$

$a_{21}, a_{22}, a_{23}, \ldots, a_{2m}$

a_{n1}, a_{n2}, a_{n3}, 111, a_{nm}

d) matrix array

....NOW IS THE TIME ...

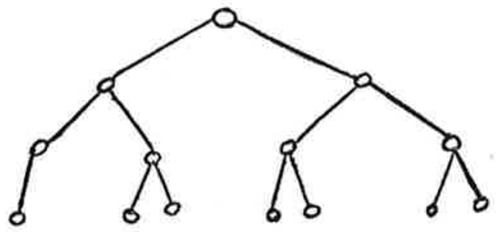
e) string

element 4 pointer
element 1 pointer
element 2 pointer
element 5 end
element 3 pointer

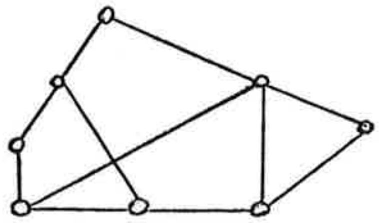
f) list

element 1 pointer
element 2 pointer
element 4 pointer
element 3 pointer

g) ring



h) tree



i) lattice

Figure 1.5.1
REPRESENTATIONS OF DATA TYPES

But good as it was, FORTRAN was not the answer to all of the needs of computer users. In particular, FORTRAN made no provision for data structures other than random files and one- and two-dimensional arrays. Business and financial data processing make wide use of sequential files and of hierarchies of random and sequential files, hence a language providing facilities for the easy use of such structures was called for. COBOL (COmmon Business Oriented Language) was such a language; it also provided facilities not found in FORTRAN for defining business oriented functions.

In 1958, the specifications of what was hoped to be an international standard mathematical language was published as ALGOL 58. Since then, new standard versions have been defined culminating in ALGOL 68 currently being touted. ALGOL is used as the standard language for publication of algorithms and has been widely accepted as a programming language in Europe. It is used to a lesser degree in the United States but some of its derivatives, notably JOVIAL, have achieved some acceptance here.

In order to allow convenient processing of lists, such languages are IPL-V and LISP were developed, and for processing strings and unstructured text, SNOBOL and TRAC, among others, were written. Furthermore, in a veritable explosion of effort, languages have been developed for a vast range of special purposes, using concepts and language peculiar to particular fields from civil engineering to making animated movies. The definitive works on many of these languages have been collected by Rosen [2].

A newer general-purpose language gaining great popularity is the IBM-backed PL/1. This language was developed during the mid-1960's as an extension to and a replacement for FORTRAN. It has capabilities in many areas and could well be a replacement for ALGOL and COBOL, as well as FORTRAN.

When, in the design of a computer system, it becomes necessary to choose the language in which the applications programs of the system are written, great care should be expended to ensure that the one(s) chosen will have the following attributes. The language should match the task--though business systems can and have been coded in FORTRAN, the extra effort hardly justifies the choice. The language should allow natural expression of the data with which the system deals--it may be that a special language, or dialect of an existing language, should be developed for the application. Finally, the language should impose no arbitrary limits on the development of the system--a built-in maximum array dimension may make a particular algorithm impossible or hard to program, for instance.

5

1.5.4 Operating Systems. As computer systems increased in size and scope it became clear that major economics would result from letting the computer assist in the sequencing of tasks and in the supplying of services. The earliest systems accepted an input stream pre-stored on tape or from decks of cards and supplied compilers, assemblers, loaders, sort programs, core dump programs, etc. from a system library on magnetic tape. The output was usually stored on tape for later off-line printing and/or punching. Their objective was to speed up operations in the so-called "closed-shop" batch operation.

A parallel development was the building of continuously running real-time systems. These systems were meant to control processes while they were taking place--hence real-time. The command and control systems from SAGE onward are examples of this kind of system.

A number of pivotal concepts have since been incorporated to produce the multi-purpose operating systems of today.

Interrupt processing. The key to real-time operation was the introduction of active interruption of the computer processing by external devices. Multi-level priority schemes allow a variety of fast- and slow-speed devices to be monitored with processing geared to the needs of each.

Multiprogramming. The servicing of slow-speed input/output devices requires only a small fraction of the time available on the computer. To make use of this time, operating systems have been devised which can schedule many tasks at once, working on each in turn while allowing the slow I/O to take place for the others. In this way, expensive equipment can be used to maximum advantage.

Multiprocessing. As the variety of tasks increased, it became apparent that some would require a very large part of the total system while others would require much smaller parts. One way to operate efficiently would be to supply numbers of each of the basic computing resources--computing modules, memory modules, I/O channels, etc.--and to assign these resources on a dynamic basis to the various jobs being run. A machine with multiple computing elements is called a multiprocessor and operating in such a way that a number of tasks are being processed in parallel is called multiprocessing.

Interactive Systems. Since the early days of command/control systems, one aim of system designers has been to build systems with which men could interact on a dynamic basis. Despite great strides, there is still a certain barrier between a human operator and the lightning-fast computer. The work goes on, however, with indications of success with direct voice communication, recognition of handwritten messages, etc.

Time-sharing Systems. The extreme speed of the computers means that a single interactive user is actually using only the merest fraction of the computers power. By using the technique of time-slicing, a single system can be shared by many interactive users, each of whom thinks of himself as the only user: the computer assigns each user a few milliseconds out of each second and rotates its attention among them. With proper hardware and operating system, each user can be assigned enough resources so that it appears that the whole machine is his. By means of the virtual memory concept, a total addressable memory space larger than the main memory of the computer can be assigned to each user. This is because most of the memory actually is stored on mass memory devices outside the machine and only the small part active at any one time is brought into the main memory.

5

1.5.5 Summary. A very large part of any computer system is the software: the operating system and the application programs. The total cost of this software will usually equal and sometimes may exceed the hardware cost. Modern techniques involving high-order languages and sophisticated operating systems can be used to prepare efficient software in an efficient manner.

References

[1] Burks, A.W., Goldstein, H.H., von Neumann, J., "Preliminary discussion of the logical design of an electronic computing instrument", (Part 1, Vol. 1), a report prepared for the U.S. Army Ordnance Dept., 1946 in collected works of John von Neumann, Vol. 5, A.H. Taub (Ed.), MacMillan Co., New York 1963.

[2] Rosen, S. (ed), Programming Systems and Languages, McGraw-Hill, New York, 1967.

REFERENCES

1. Flores, Ivan, Computer Organization, Prentice-Hall, 1969.

2. Flores, Ivan, Data Structures, Prentice-Hall, 1970.

3. Stimler, Saul, Real-Time Data Processing Systems, McGraw-Hill, 1969.

4. Gear,   Computer Organization and Programming, McGraw-Hill, 1969.

5. Martin, James, Design of Real-Time Computer Systems, Prentice-Hall, 1967.

6. Estrin, G., and Fuller, R., Algorithm for Content-Addressable Memories, Proceedings IEEE, pp. 118-130, Pacific Computer Conf., 1963.

7. Ewing, R.G., and Davies, P.M., An Associative Parallel Processor with Applications to Picture Processing, Proc. FJCC, pp. 105-115, 1965.

8. McKeever, B.T., The Associative Memory Structure, Proc. FJCC, pp. 371-388, 1965.

9. Flynn, M.J., Very High Speed Computing Systems, Proc. IEEE, Vol. 54, No. 12, pp. 190-1909, Dec. 1966.

10. Lewin, M.H., Retrieval of Ordered Lists from A Content Addressed Memory, RCA Review, June 1962, pp. 215-229.

11. Kautz, W.H., An Augmented Content-Addressed Memory Array for Implementation with Large-Scale Integration, J.A.C.M., Vol. 18, No. 1, Jan. 1971.

12. Wald, L.D., An Associative Memory Using Large-Scale Integration, NAECON 70 Record, pp. 277-281.

13. Engles, R.W., Concepts and Terminology for Programmers, IBM TR 00.1663, October 2, 1967.

14. Lyons, R.E., The Application of Associative Processing to Automated Air Traffic Control, A White Paper submitted to the FAA, Feb. 14, 1969.

15. Cannel, M.H., et.al., Concepts and Applications of Computerized Associative Processing, MTR-1735, MTR-863, ESD-TR-70-379.

16. Thurber, K., The Application of Parallel Processing Techniques To Air Traffic Control,Proprietary Report submitted to the FAA, March 16, 1970 by Honeywell Systems and Research Center.

17. Meilander, et.al., Applicability of Parallel Processing Techniques for ATC System Modernization, Final Report submitted to the FAA Feb. 24, 1970 by the Goodyear Aerospace Corp.

2.0 ATC DATA PROCESSING REQUIREMENTS

2.1 INTRODUCTION

In this chapter we will discuss three aspects of the ATC system evaluation process. In the first section, we discuss the recent efforts to predict the size of the computing facility needed to carry out the ATC functions in the 1980's. We do this by considering the estimations of traffic levels, by considering the effects of increased traffic on the data processing loads and finally by looking at the estimations made of the size of the data processing system needed to do 1980 ATC.

The second section will be devoted to a general assessment of the near future development of air traffic control systems in terms of the hardware and software and their interactions. Large capacity and high reliability are the characteristics stressed and developments which lead in their direction are discussed.

The third section is devoted to the evaluation techniques to be applied to ATC systems. A description of evaluation measures is followed by a discussion of the way these measures are applied in system design and evaluation and to the approaches used to develop these measures.

Finally, a few conclusions are drawn concerning profitable directions for future efforts in ATC system design and evaluation.

2.2 ESTIMATIONS OF ATC DATA PROCESSING LOADS FOR 1980

2.2.1 Estimations of Traffic Levels. Probably the best and most recent forecast of air traffic levels currently available is that done by the working group (Group IV) on users needs for the DOT Air Traffic Control Advisory Committee (ATCAC) and reported by Ashby [1,2]. Their estimates were derived by collecting from the FAA and others [see e.g. 3-6] counts of numbers of types of aircraft and their utilizations in hours. for the then most recent year [1968] and forecasts of the same numbers for the years 1980 and 1995. The basic data used by the group in deriving their estimates are probably as good as can be obtained; that is not to say, however, that the data base was entirely adequate, as pointed out by Ashby, himself. [1, p. 469]

The measures sought in this study were the annual number of operations (takeoffs and landings) and the peak number of airborne aircraft for the years 1968, 1980 and 1995. The numbers derived were: for annual operations in millions, 128, 222, 519 and for peak airborne aircraft, 12800, 22200, 54400. Approximately stated, the values recorded for both measures indicates that air traffic levels will double between 1968 and 1980 and will double again between 1980 and 1995.

An increase in air traffic does not automatically imply a proportionate increase in demand for ATC services. As a matter of fact, under rather modest assumptions as to IFR usage after 1980, the study group estimated that there would be an increase in IFR enroute flights of three times the 1968 rate by 1980 and eight times by 1995. The load on terminal area control operations, including possible intermittent positive control (IPC) and taking into account the fact that some airports are near saturation now, is estimated to reach 10 to 15 times 1968 levels by 1995.

One could fault these figures only on the basis of having three years more worth of data available at this time. If one were to work with FY 71 data when available and take into account the recent and current drop in airline activity, somewhat different numbers might be derived. The demise of the U.S.-built SST, as well as the slowdown in general aviation because of economic factors, could also be included in a new traffic level forecast.

In sum, the numbers produced for ATCAC as estimates of traffic levels are the best available at the moment, are probably adequate for the time being but could undoubtedly be sharpened in light of more recent data.

2.2.2 Effects of Increased Levels on System Operation. There are two major ways in which increased levels of traffic affect the operation of the air traffic control system: by causing delays and by causing accidents. Delays are the result of saturated facilities--mostly airport runways and loading docks. Queues start to appear at relatively low system loadings--of the order of half of capacity--and build up rapidly as various parts of the system approach saturation. Cost penalties aside, a system operating in a highly loaded state is subject to stresses caused by transient phenomena which strain its capabilities to the point where complete breakdown is a distinct possibility.

Safety is, of course, an end in itself--no cost penalties or system degradation factors need be invoked. Studies done in support of the ATCAC report indicate that near mid-air collisions (NMAC) and probably mid-air collisions (MAC) themselves occur in proportion to the square of the aircraft population in terminal areas unless both aircraft are under ATC control. [7]

Alexander [8] showed that aircraft interactions per unit time per aircraft occur about in proportion to the density of aircraft in the regions, as an upper bound. At the same time, he showed that even with a highly ordered flight environment, aircraft densities could not exceed about one per square mile without catastrophe. Fortunately, while aircraft densities in some areas are at the level where collision avoidance techniques will soon be mandatory, densities at their worst are not expected to exceed the levels at which extensions of present ATC techniques will suffice.

It should also be mentioned that in the situation described so far, it will not be enough merely to increase the number of controllers and consoles as the traffic loads increase. There is already a great deal of interaction among controllers, which will increase as the system becomes loaded. The required reaction time will decrease, so that each controller will have more to do in less time. This implies automation of certain functions to both relieve the controller load and provide for the added ATC services.

Thus, it is clear that ATC data processing loads will not simply be increased in proportion to the traffic levels, but will expand faster than traffic levels, both because of new control tasks and non-linear expansion of old ones.

2.2.3 Estimations of Data Processing Requirements. The most extensive recent estimate of data processing requirements for air traffic control in the future was the one published in the ATCAC report [9] and summarized in the IEEE Proceedings [10]. In preparing their estimate, the Computer Sizing Group of ATCAC made a number of basic assumptions which served to make their task at least tractable. They first of all created a model of the air traffic control system of the future, including a data acquisition system, a control organization and a set of control functions. Next, they chose a particular terminal and enroute area (Los Angeles basin) for analysis on the basis that it will be the area with the highest level of activity in 1980-1995, and estimated traffic loads for that time frame. On the basis of these assumptions, they projected variously obtained estimates of terminal and en route instruction rates and storage requirements to the expected level and obtained overall processing and storage requirements for the future maximally loaded terminal and en route centers.

It is difficult to quarrel with the model they have chosen for the ATC system, for it is reasonable in all detail. It does, however, give only "one point in solution space", as the committee puts it, with no indication of whether it is a typical point or an extremum. Their selection of Los Angeles as the area with heaviest air traffic activity was a good one, but their estimate of traffic levels for 1990-1995* was based on earlier FAA projections of three-fold increases per decade for the next twenty years. They therefore chose as a basic size assumption, a peak count of 100,000 air craft airborne simultaneously--ten times the 1969 count.

* The Sizing Committee made their estimates for 1990, while the Users Needs groups chose 1995 as their target date. As summarized in the IEEE Proceedings, the reports of both groups referred to 1995: evidently an editorial change was made for a more uniform presentation.

The Users Needs group, however, in their estimate of traffic loads, predicted the 1995 peak airborne level as 54,400 [1, p. 473]. This factor of two could be looked upon as a "factor of safety" or as an unnecessarily conservative "hedge factor". In either case, it should be kept in mind when gauging the merit of the data processing load estimate.

The actual estimates of instruction rate and storage requirements were obtained by the members of the committee from various studies and proposals on individual subsystems, e.g., collision avoidance, data acquisition, flow control and command/control. These estimates were made for the terminal system, the en route system and, in the case of flow control, the national system. Storage estimates were presented in words of fast memory and of mass memory with a portion assumed for monitor use. The instruction rates were estimated for each subsystem in terms of actual executed instructions per unit time to carry out the function for the postulated system loading. These instruction rates were then multiplied by factors which were meant to take into account (1) the fact that different instructions require different times on most processors, (2) the monitor operates some of the time to supply services to the traffic control programs, and (3) the programs will probably be written in a higher-level language (FORTRAN, JOVIAL) whose compiler produces less than optimum programs.

Granted that the storage estimates are reasonable, they being based at least in part on actual counts of existing programs, there is a certain arbitrariness in the computing power estimates that casts a pall on their outcome. Even if one were to accept the derived values of instruction rate,* he should question the values of the three factors applied to them, which seem more intuitive than objectively derived. So much so, that the two published versions of the study use different values for the compiler inefficiency factor--3.0 and 1.5--giving final estimates of system computational power that are different by a factor of two.

Perhaps an even more valid criticism is that the choise of a single measure, MIPS, to describe the computational power of the system is an over-simplification which could lead to erroneous conclusions. This point is discussed below at some length.

* There are some inconsistencies between values reported in [1] and [2]--the data acquisition value changed by 10% for the en route case and 17% for the terminal. Furthermore, the estimate given for the en route command/control already contains the compiler degradation by a factor of 2. See [11], p. 4-7.

The study concludes by considering the kinds of computer equipment which will be available by about 1975 and asking whether an ATC system built from these equipments would be adequate for the 1980-1990 traffic loads. Even on the basis of very conservative technological forecasts, they conclude that the answer is yes and recommend basic configurations of multiple processors with cache memories, large fast memories and I/O processors for the en route, terminal and flow control centers. Their estimates include costs based on projections of current prices to the 1975 period.

2.3 SYSTEM DESIGN CONSIDERATIONS

There are three facets to any computer system design which can be examined for trends: the overall system architecture, the hardware which makes up the physical system and the software which directs its operation. These three aspects are so inter-related, however, that is is difficult to discuss any one without reference to the others. In the following sections, the attempt is made starting with the system architecture and then returning to it after the looks at hardware and software.

2.3.1 System Architectures - I.

One of the recommendations of the ATCAC report was that the computers at the various centers should be of the multi-processor configuration. This choice was made both because of the requirement for a range of processing power for different sized terminals and also the requirement for very high reliability. A modularly expandable system can meet both requirements: the number of processing modules can be tailored to the expected load plus an extra to be used in case of a failed module.

Unfortunately, the computing power of a multi-processor configuration does not grow linearly with the number of processor modules. There are two reasons for this: increased overhead associated with more complex configuration control and delays introduced because of conflicts between processors attempting to access a common memory module simultaneously.

Multi-processor configurations will be made up of varying numbers of four major components: input processors, computational processors, output processors and memory units. In addition, there will be other elements such as interconnection switching units, reconfiguration units and system and maintenance consoles with associated keyboard entry and printing devices. Some mass storage--such as magnetic tape, drum or disc--will have to be provided for storage of large amounts of data and for backup to the main storage. There will also be the entire data acquisition systems with their interfaces, the display output systems with their interfaces and message entry devices of various kinds. Other output will be communications to other centers and possibly, via data link, to the aircraft.

1. Input Processor. Input processors of varying capabilities will be used in any future system to provide for the preprocessing of inputs from the various sensors, other centers and message entry points within the center. These equipments will act as data concentrators, regulating the flow by multiplexing the inputs and providing for quantization, formatting, queuing and other services. The present-day Data Acquisition System (DAS), Data Receiver Group (DRG) and Common Digitizer (CD) fall into this category.

The Input/Output Control Elements (IOCE's) and Processors (IOP's) currently in use also are in this group. Their functions of controlling channels and peripheral equipment will be made easier by the increased use of common, uniform interface modules, such as the Peripheral Adapter Module (PAM). These processors will be provided with computing capability sufficient to carry the load of error checking and correction and data validation independent of the central processing system. Certain routine computational functions, such as coordinate conversion, could be performed in the input processor portion of the system.

2. Central Processor. The central processor module will continue to be the computational point in the system, but because of increased parallelism, each module will become relatively less important to the system. Since conflict between processors attempting to gain access to a common memory unit will be a major factor in the use of multi-processors, efforts on many levels will be made to minimize it. At the system architecture level, a high-speed buffer, or "cache" memory may become part of each processor. If these memories are combined with wide data paths to main memories, the conflicts between processors can be significantly reduced.

If the buffer memory were large enough, say on the order of 500K words, and fabricated using content-addressable memory chips, it could be organized as an associative memory. Then, various operations such as matching, greater than or equal, less than or equal, add fields and multiply fields could be conducted in parallel with maskable arguments. This type of organization could overcome the chief problem encountered with associative memories and processors--the time absorbed in shuttling data across the interface between them and the serial processor used as a control.

Extensions of the instruction repertoires of central processors will make them better able to carry out their major ATC functions, as well as enabling diagnosis of failures and reconfiguration of the system. Increased word length will be required to implement the new functions and will also serve to increase effective data transfer rates into and out of the processors.

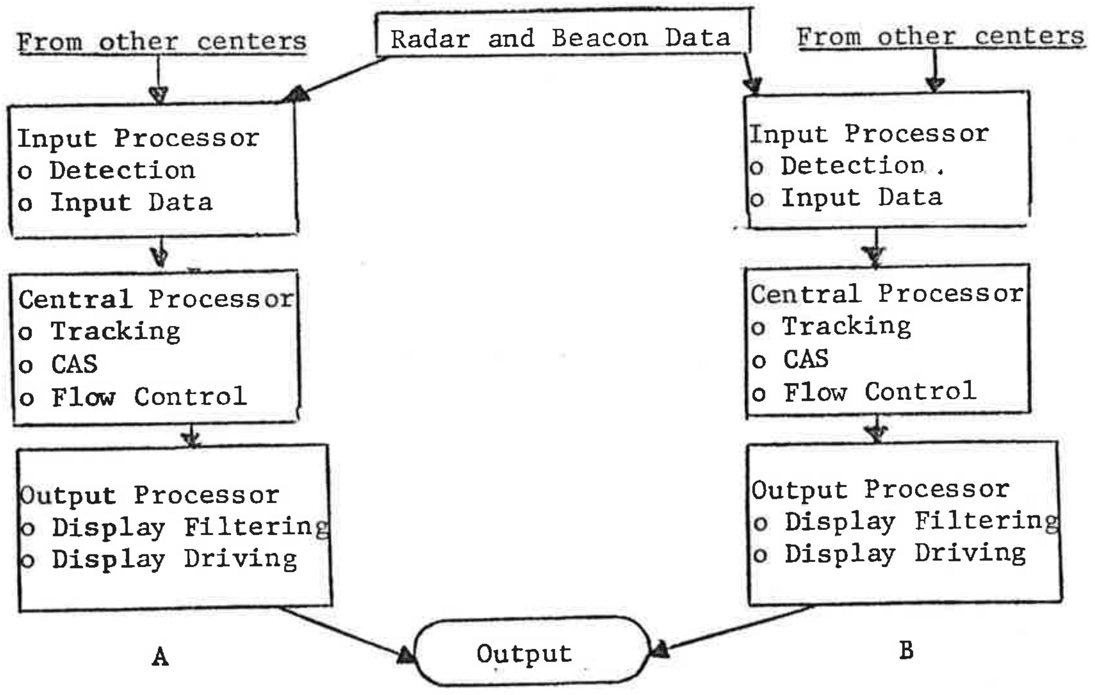
3. Output Processor. All of the output decisions and data will be processed in the output processor which will be specialized toward driving displays according to adaptable display filtering algorithms. Operator requests for display changes will be passed from the input processor through main memory to the output processor without intervention by the central processor, thus relieving the load at that point.

The current Computer Display Channel (CDC) and Display Element (DE) are examples of rudimentary output processors.

4. Memory Units. The trend in memory technology has been toward faster and larger units to the point where the fastest memories are compatible with the best CP cycle times and the largest units strain the addressing capability of all but the longest word machines. Unfortunately, speed and size have up to now not been achieved simultaneously. This fact, along with the trend toward multiprocessing, has led to the development of modular memory units organized in a hierarchical fashion. Memory hierarchies and buffer memories are not new, of course, but it should be pointed out that the hierarchies being projected will be entirely random-access memories with only access time and size differentiating the various levels.

Modularity of memory will serve the same purposes as multiple processor units serve--namely, gaining the ability to partition the system into complete subsystems for independent operations for reliability purposes and the ability to reconfigure with redundant units in case of failure.

As an example of how reliability could be achieved, consider the following adaptation of the SAGE duplex system.



Here, the Input Processor handles the detection process as well as coordinate conversion and bias correction; it also accepts and formats input data from other centers. The Central Processor, which might include a large semi-conductor memory, performs the tracking, collision avoidance, metering and spacing and flow control functions. The Output Processor does display filtering and the driving of displays. Output to other centers and to aircraft for Intermittent Positive Control (IPC) or other collision avoidance schemes is also handled by the Output Processor.

In this use of a duplex configuration, both systems will be doing identical computations on the same data set, with one system--say A--being designated as the main, or on-line, system and the other, B, acting as the auxiliary, or backup, system. If an error is detected in the output of system A, it is switched off-line for fault diagnosis while B is made the on-line system. The error detection process could well include an automatic comparison of the outputs of A and B and some combination of automatic and human resolution of detected differences.

The success of any such simple plan requires a degree of circuit reliability that is expected to be realizable as the result of hardware developments discussed below. In general, however, system designers will have recourse to many hardware features which will be used to increase system reliability. Included will be independent power systems, switch-over capability, failure detection and correction, redundant organization and provisions for preventive maintenance.

2.3.2 Hardware Developments. The development of logic devices, circuits and components in the near future will be aimed at characteristics mentioned above: capacity and reliability. Very high-speed logic components with switching speeds of one nanosecond and under will be available using advanced emitter-coupled logic (ECL) and Schottky-barrier transistor-transistor logic (TTL). Such speeds are already being approached in commercially available lines. (See [12] and [13] for instance.) At the same time, very high packing densities will be attained with resulting speed-power products of the order of 25 pico-joules and lower. The development of other concepts, such as charge-coupled devices (CCD), could lead to even better speed-power products.

Large-scale integration (LSI) should be an important factor in the market within a few years and its arrival should influence profoundly the reliability as well as the complexity of computer circuits. There will be a reciprocal relationship, however, between the development of new widely used circuits and the continued development of LSI. If modular circuits, such as random access semi-conductor memories, can be standardized and requirements generated for large quantities of them in LSI packaging, the production techniques will be developed which will result in very much lower cost for a very much more reliable product. Part of the reliability will come from the order of magnitude reduction in external connections in the circuits and part through the error detection and correction circuitry within the module. The fact that fabrication cost increases far less than linearly with circuit complexity makes it possible to produce very complex circuits economically, if produced in large numbers.

Semi-conductor memories could be a replacement for all magnetic memories except for very large bulk storages. The one disadvantage of semi-conductor memories that could prevent this trend is their volatility, i.e., the characteristic that information stored in them disappears when power is interrupted. An extensive backup will be required or some new ideas devised to solve this problem before dependence can be placed on these memories in the ATC environment with its demand for uninterrupted service. The only backup at the present time which could provide the instant response required would be a compeltely redundant memory with independent power supply.

In the case of control memories, the volatility problem could possibly be solved by use of a technique suggested by three IBM research scientists [14]. They point out that due to slight manufacturing variance, within tolerance, each cell of a memory will tend to return to one of its states more often than the other when power is turned off and then restored. They propose to use this natural tendency but accentuating it--in ways which depend on the actual circuit in question--so that the cell will always return to a selected state. This state will be determined during manufacture, limiting the use of the technique to permanent ROM's. It is expected that the only effect on the performance of the circuit will be a negligible increase in power dissipation.

Since complexity of circuitry will be an essential attribute of LSI implementation, it will be possible to design and build modules with new capabilities--for instance, semi-conductor RAM's which are not only location-addressable, but also content-addressable. Thus, parallel-search capability will be part of each standard processor. At the same time, array processors of various types and capabilities will be much less expensive and much more reliable.

Still newer, presently experimental techniques may reach fruition in the next few years and bring new capabilities to an ATC system. An example is the so-called bubble memory under development at Bell Telephone Laboratories. A memory unit using this technology could replace a moderately sized disk file--10 million to 100 million bits, while occupying 2 to 3 inches of space, using only about 10 watts and providing an access time an order of magnitude less than the disk.

The Domain Tip Technology (DOT) memories could provide essentially the same performance gains if developed.

2.3.3 Software Developments. Over the past few years, it has been possible to observe a number of trends in software development and it is safe to say that these trends will continue. On the one hand will be the continued development and refinement of existing systems--the application of current techniques to newer equipment and to new situations. On the other hand will be the non-evolutionary development of new languages and systems to take advantage of non-traditional hardware and to apply to unusual situations. The programming for ATC purposes can be expected to be derived from some combination of both trends.

The ATC software of the future will have to exhibit three characteristics if it is to be successful: reliability, efficiency and usability. These are desirable features in any system, of course, but they assume great importance in the real-time control environment where human lives are at stake.

The one development which will have the largest impact on all three is that of new high-order programming languages which will allow the programmer to produce both efficient and error-free code. Increased use will be made of interactive incremental compilers for these languages which have analytic powers to test the coding as it is written both for correctness and completeness. Optimization passes will be made on the generated code to produce routines of maximum efficiency.

In the area of system design, strides will be made in the production of supervisory systems that are tied very closely to the hardware design so that the overhead inherent in their use will be minimized. The interest in supervisory systems is quite intense at the present time and research in that area should lead to more rational and scientific methods in the design of the hardware/supervisor system.

As a contribution to the overall system operation, the system software will incorporate more, better and faster error handling techniques. Again, these routines will be very closely interlinked with the newer hardware implementations to produce extremely efficient operation.

2.3.4 System Architectures - II. As a result of the hardware and software developments outlined above, the presently observable trend toward new and unorthodox system architectures should continue. Control memories of semi-conductor ROM chips will make microprogramming relatively easy and inexpensive to implement, so that its use should spread from the central processor to other parts of the system. This implies a decentralization of the computing power so that memories, peripherals and terminals can relieve the load on the CP's by doing code translation, data formatting, coordinate conversions, scaling and the like. This increased parallelism will also make it possible to construct centers of many sizes and capacities from the same units and with the same programming.

Rewritable control memories will also contribute to system reliability by allowing a standard microprogrammed unit to act in a number of capacities. One might envision a collection of identical processors, each specialized for a particular job by virtue of its control memory contents, with a few spares ready to be put on-line in any capacity as soon as its control memory was properly loaded.

As an example of how far this trend might go, consider the Fairchild Symbol II R computer being installed at Iowa State University [15, 16]. In this experimental machine, the compiler for a high-level language, called Symbol, and a time-sharing monitor system are hardwired into the machine. As a result, the compilation rate is said to be an order of magnitude faster than conventional machines, simply because there is essentially no operating system overhead involved in the process. Implementation of the monitor system in the hardware also leads to great savings in overhead, since operations, such as input/output processing for example, are accomplished directly rather than through routines which require elaborate calling and parameter passing sequences.

The extensive use of large, fast semiconductor control memories whose contents can be easily changed could result in whole systems being implemented into the so-called "firmware". In systems that remain relatively constant in function, such as ATC systems, there might be no stored programs as we know them, memory being used for data and parameter lists only.

2.4 EVALUATION TECHNIQUES AS APPLIED TO ATC SYSTEMS

2.4.1 Defining System Performance. The traditional approach to system design has been to analyze the set of inputs and the required set of outputs and to define, in terms of these sets and the transformation between them, a cost-performance function to be optimized. It is in the definition of the cost function that the greatest difficulty generally occurs, since, for most systems, one is required to assign monetary value to such unsubstantial qualities as customer impatience or the effect of delay in treatment of a sick hospital patient. For real-time systems, one is constrained, moreover, to provide a cost-effective system which is able to remain in synchronism with an external source of data, accepting a (usually) very high percentage of it and producing output within a (usually) short time. A real-time control system, for instance, is generally required to produce output within such time that this output may be fed-back essentially before the next input is generated.

An air traffic control system is a real-time control system with additional constraints imposed on it: namely, that is never be overwhelmed by the input data (or at least that it have satisfactory provisions for handling an overload) and that it never fail (or at least that it fail in a "controlled" or "graceful" way).

Thus, there are three characteristics of the ATC system which require definition and measurement. The first is a measure of the capacity, or throughput, of the system. This could be taken to be merely the peak number of aircraft that can be tracked by the system without degradation of response. This measure, of course, would relate the capacities of two systems only if both provided the same functions such as collision avoidance, etc. In fact, the introduction of additional functions into the system complicates the definition of capacity; one must develop throughput measures which reflect the multiplicity of tasks within the system, the complexity of each task and also the relative amounts of input to and output from each.

A second characteristic needing definition is response, or reaction, time. In many systems, each input message triggers a particular output message, so that reaction time can be measured directly as the interval between them. In many control systems, the response to an input is observable at some point in the system and again is directly measurable.

Some facets of the ATC system also exhibit an observable response time. For instance, the radar correlation and tracking functions work synchronously with the scan of the radar and produce an output for each track during the scan--clearly observable. However, a collision avoidance system (CAS) might produce output only when collisions were imminent. The CAS function's response time might be the time between its detection of collision producing conditions and the issuance of corrective instructions, while the system response time might be the time between the arrival of the relative radar data and the receipt of the instructions by the pilot(s).

It is clear then that any definition of response time will have to take into account a number of factors, some of which are not easy to quantify.

Finally, any ATC system will have to exhibit a very high degree of reliability not required of most systems. There are many facets to this characteristic, as well: the hardware must be reliable in the sense of long MTBF and short Mean Down-time and the programs must be reliable in the sense of being fully checked-out. In addition, there must be provision for maintaining system integrity by replacing failed modules with redundant ones, by switching to backup modes of operation, etc. There must also be an extensive capability to detect and react to error and fault conditions, both in software and hardware.

It is interesting to note that surveys of computer system performance measurement and evaluation published nearly four years apart say very nearly the same thing about the state of the "Theory of Computer Performance". In early 1967: "Progress is designing and applying information handling systems has outraced progress in evaluating their performance. As system complexity has increased, the relative adequacy of existing evaluation tools has decreased." [17, p. 12] In late 1970: "Today, ...the knowledge necessary to develop such a theory [of computer performance] is, at best, only rudimentary. ...The need...is substantial." [18, p. 22].

The very extensive literature on the subject shows that a real and considerable effort is being made to develop the tools needed for computer systems design, performance measurement and evaluation. In the next section, we will discuss some of them, but as a basis for conclusions about their applicability to air traffic control systems, we will first look at some criteria which ATC systems must meet as regards capacities and response time.

1. Input rate - There will be a variety of input/output requirements imposed on the ATC center; one of the most important of these is input rate. The input stream will be made up of essentially four components: (1) very high rate stream from primary data acquisition system (DAS), (2) medium rate stream from other centers, terminal and/or en route, (3) low rate stream from back-up DAS and (4) low rate stream from input within the center. The actual rate in the first stream will depend on the acquisition system chosen for implementation but will probably be of the order of 300-500 thousand bits per second (KBPS) for a terminal center and 100-200 KBPS for an en-route center. The second component will be made up of flight-plan data, hand-over and cross-tell information, weather data, etc. and should be of the order of 50-100 KBPS for any one center. Back-up DAS will no doubt be of low capacity compared to the main system and might contribute about 10 KBPS to the input rate. Finally, the lowest input rate will occur from data entry from keyboard, light pen, track ball and other devices within the center itself; some will be controller actions but others will be flight-plan, weather, configuration and other data entries. The peak rate from this activity might approach 10 KBPS, but the average will be well below that level. These estimates are paraphrases of data given in [9] and [10].

2. Input formats - Somewhere in the total system, in the acquisition subsystem, the processing subsystem or their interfaces, the variable formats of input messages must be rectified. It would, naturally, be preferable from a system design point of view to have compulsory standard instrumentation in all aircraft (or at least equipment which produced standard format messages), but this may never be a political reality. The matter of non-standard messages will therefore be a criterion on which the system design will be based.

The back-up DAS may present the same problem unless it relies entirely on primary radar for its data. A solution for the main system should apply to the back-up system as well, however.

Input formats for messages from other centers and within the center itself are presumably under the control of the system designer and can be manipulated to provide best performance.

3. Input response - The only input streams with stringent real-time response requirements are from the main and back-up DAS. Even so, since these messages will presumably be time-tagged and since the processing programs will presumably work on a sector basis, there will be a natural opportunity for buffering, so long as the cycle rate is fast enough to keep up with the stream. Other inputs may either be accepted character-by-character on multiplexed channels of the input processor, or buffered at the input device and accepted in a burst transmission.

4. Operating priorities and time constraints - The real-time aspect of ATC systems comes to the fore when one considers the matters of priorities and constraints. Certain programs within the system must operate when called (as, for instance, a collision avoidance algorithm when an imminent collision is detected) and certain others must operate within given time frames (as, for instance, all the radar correlation/ tracking algorithms which must run once per scan of the radars). These conditions will be the prime determinants of the system design; they will dictate the hardware configuration, executive philosophy and operating procedures of the successful design.

5. Output rates - The major output streams from the center will roughly correspond to the input streams: (1) medium data rate components to aircraft via data link for control purposes, (2) medium rate stream to other centers, terminal and/or en route, (3) medium rate stream to the devices within the center and (4) low rate streams in the form of control information to the data acquisition systems. Estimates of the sizes of these streams--except that out to other centers which should balance the amount of input--will depend on decisions as to the amount of control exercised by the system, the nature of the data acquisition system and the nature and amount of display to be generated in the centers. Some of these decisions will be part of the system design process, but others will be part of the system specification.

6. Overall response time - As indicated earlier, response time of a real-time system is paradoxically extremely important and frequently difficult to define and measure. It is tempting to say that the system "must do the job" but, of course, this is not enough. It is possible, and perhaps even enough, to list the inputs to the system and assign outputs to them in such a way that a reaction to each may be calculated,

but an overall, inclusive measure by which two different systems may be compared is yet to be enunciated.

On a more positive note, and in line with the preceding suggestion, certain response times can be listed. The reaction of the correlation/ track prediction algorithms to radar data input has already been noted. Certainly the reaction to controller inputs should appear to the man, himself, to be instantaneous. For instance, if he were to request that a certain track be put in SUSPEND status, the indication on his display should appear to have been created by a direct connection between the button he presses and the face of the scope. Again, there must be no more than a few seconds' delay between the receipt by the DAS of an emergency code and the appropriate ramifications at the controller's station (actually the delay might be more meaningful if measured from the moment of first transmission by the aircraft).

In all, this is an area where careful thought might be most rewarding.

7. Reliability - The question of reliability has been touched upon earlier and is extensively treated elsewhere*, so let it suffice that ATC data processing equipment in the 1990's will have MTBF figures of greater than 10000 hours. In addition, so-called "fail-soft" and "fail-safe" modes will be included so that the ATC system essentially never fails.

8. Capacity - The last criterion to be listed is the elusive one of capacity, again, already mentioned. If one were able to measure any quantities he wanted within a system and to take as many samples as he pleased, he could still not produce a number which could be called the system capacity. The reasons were mentioned above in general terms, but let us look at one commonly quoted system measure, MIPS, or millions of instructions per second.

The chief advantage of MIPS as a measure of capacity is that it is easy to derive and simple to apply. It is usually defined as the reciprocal of the length of time in microseconds required to execute a representative instruction, such as ADD. Unfortunately, only very similar machines may be compared with this number because there is no account taken of word length, addressing structure, special instructions or special architectures. Indeed, to cite an absurdity, a computer with a 2-bit word length, 1 nano-second cycle time and an instruction code consisting of just ADD would outscore the ILLIAC IV in computing power as measured in MIPS. Further remarks on this topic may be found in [17, pp. 14-16].

* See "ATC Computer System Reliability and Recovery" Report of FA-03, Task 3.

N_bulous as some of them are, these are the criteria by which ATC systems will be judged as they are developed.

2.4.2 Evaluation Methodology. The techniques to be used in evaluating system designs should be chosen on the basis of the evaluation criteria, or to put it another way, on the basis of the questions about the designs for which answers are desired. It is possible to draw up representative sets of questions such that the members apply to different aspects of the design/evaluation process. One such set might be:

1. Given the expected load, how big should the system be? What form should the system take?

These questions must be asked, and answered, before and during the preliminary design phase; the answers will be couched in terms of such measures as throughput, storage capacities, I/O channel capacities, etc. and of such descriptive terms as number of central, input and output processor, memory hierarchy organizations, I/O bus layouts, etc.

2. Given an expected load and a proposed system design, will it do the job? Given two or more system designs, which one will do the best job?

Once the preliminary design of a system has been prepared, it is possible to expect answers to this second set of questions, in terms of response times, capacities, measures of merit and performance ratios. Note that these questions are in a sense the converses of the first set, dealing as they do systems at about the same level of detail and with similar performance measures.

3. Given an overall system design, what is the optimum detailed design? Given a system cost and/or load, what is the optimum design?

The final selection of components--type, size and speed--will depend on answers to these questions, however phrased. The answers will be given in terms of characteristics of specific components; as, a two-level memory hierarchy with 10 nano-second and 500 nano-second cycle times and capacities of 64K words and 1024K words, respectively. Presumably, this is the level at which the final design is decided upon, complete except for fine tuning.

4. Given a complex system, what is its internal behavior? Given a complex system, what are its optimum parameter settings?

This is the fine tune and final evaluation stage, at once the easiest and most difficult part of system design and evaluation. Easiest because the system exists and is operative, can be loaded and run and either operates or fails. Most difficult because in its existence, it embodies all of the complexity and detail, most of which is hidden from view and little of which can be even conceptualized, much less observed and measured.

There are basically four approaches to system design and evaluation, or to the search for answers to the questions posed above: analysis, simulation, synthesis and measurement. These approaches are not competitive nor mutually exclusive, in general. Each has its place in the process, as we shall show, and each can be applied and the many levels of the design/evaluation process.

Analysis - The use of analysis--the representation of physical processes by means of mathematical equations and formulae and their subsequent solution--has a long history in system design. It is the first thing one turns to when conceptualizing the system and is useful during portions of any phase of the design.

There are two areas of difficulty in the use of analytical techniques in system design, indeed aspects of them are common to other techniques, as well. The first is in the formulation of the model: in identifying the important members of the input set, in selecting the dominant internal transformations and in choosing the critical output quantities and then in expressing the relationships among these in cogent form. The other difficulty is in solving the resulting equations to obtain some sort of useful result, the most desirable of which would usually be a general closed solution in terms of easily measurable variables.

But the closed solution is rarely possible except in very simple, or highly simplified, situations, so that even when it is found, it is of limited usefulness. Other solutions, such as those obtained by Monte Carlo simulation methods, are available and frequently lead to useful results.

In the later stages of the design/evaluation process, analysis has a place in predicting the performance of subsystems, whose relationships are intrinsically simpler than the overall system; in evaluating alternatives in various parts of the system, where these alternatives can be clearly stated; and in choosing values for parameters, where the effect of the parameter is reasonably well understood.

Since analysis, by its nature, is well understood, it is probably the most reliable tool for system design and evaluation in spite of its sometimes difficult aspect.

There have been many attempts to automate the calculations required for the thorough analysis of a computer system. One such attempt resulted in a proprietary program of TRW called QUEST, which accepts as its input, sets of tables of computer characteristics, routine, or program, characteristics and parameters governing the particular run. From these, the program produces tables and histograms of data which are useful to the analyst in making his design choices. Even in its present rudimentary state, and it is being extended in scope, this is a valuable tool which both systematizes and automates some of the essential calculations.

Simulation - Whenever the processes of a system under consideration are represented by different but hopefully similar processes instead, the technique is called simulation. Two processes are said to be similar whose inputs can be shown to correspond in some fashion, whose outputs can be shown to correspond and whose concomitant transformations correspond. Presumably, the second system, or model, made up of these "similar" processes, will be easier to analyse and to build than the primary one and hence judgments about it will be easier to make than about the primary system. The closeness with which the system and its model correspond will have a profound effect on the confidence which can be placed in the simulation results.

Simulation can take many forms; exponents of the technique with different implementations often do not even recognize each others work as simulation. These implementations range from full-scale mockups of space modules or aircraft cockpits subjected to motions and stresses expected during real operation to mathematical equations solved on computers using large numbers of random inputs. Likewise, in system design, large-scale simulation of the entire system, using massive computer programs and many peripheral equipments with recorded real inputs or carefully simulated inputs, stands at one extreme and quite modest, back-of-an-envelope simulations, perhaps by means of a time-shared terminal, stands at the other.

Most simulations of large-scale computer systems will involve the use of a computer program written for a large-scale computer. This raises the question of when it is better to build the simulation model before building the system. Clearly, if they are of the same degree of difficulty, take the same length of time to program and can be controlled and measured with the same ease, there is no use for the simulation. (An interesting description of an example of the case in point is given in [19] on pp. 904-905.)

If, on the other hand, the simulation is, to use an imprecise term, "believable" and it is simple and quick to implement relative to the base system, then its use is, as clearly, highly to be desired.

Synthesis - The synthesis of a system is the building of it from its component parts. It is often possible to infer a great deal about the performance of a complete system, given the subsystem performances. Hence, given that the system may be so broken down, it may be of value to collect such figures and attempt to combine them. As a matter of fact, a frequent starting point for a system simulation is the set of subsystems represented as "block boxes".

There is no requirement, other than that of maintaining a consistent level of precision and accuracy, on the manner in which performance figures for the subsystems are gathered. Some may be obtained analytically and some from simulation. The performance of certain subsystems may be so difficult to predict that it is easier to actually build and test them.

This frankly eclectic method is probably the most powerful means for system analysis available today, suffering as it does only one major weakness, namely, that the final combination of the parts--the synthesis--requires knowledge of, or determination of, the interactions between these parts. Whether the synthesis is by simulation, as mentioned above, or by analysis, the final results will depend on the correctness of the representations of the interactions.

In the extreme, perhaps the only way to compare a number of system designs is to implement and measure them. One computer competition using this technique is being conducted under the auspices of the Army Ballistic Missile Defense Agency (ABMDA) in the area of ballistic missile tracking. Here, the software for the process is being written for each computer so as to take advantage of its particular forte, and the running time and accuracy of the solution is being measured. Obviously, this is a very expensive procedure, rarely justifiable.

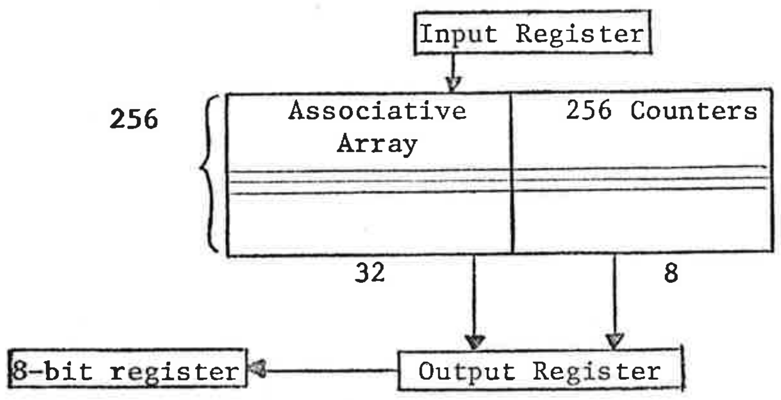
Measurement - The performance of the system itself may be measured either as a last resort--analysis having been overwhelmed and simulation having proved too slow or costly, for instance--or as a part of the final design or evaluation phases. Except for minor corrections and adjustments, this is an after-the-fact exercise: the system, or subsystem, has been built and is at least partly operational.

In general, two techniques have been developed for system measurement; one, using extra hardware, monitors and records various facets of the computer and peripheral operation while the other, using extra software, monitors and records various facets of the program operation. Both techniques have drawbacks but both can provide valuable information about the performance of the system.

The hardware technique has the supreme advantage of not interfering with the operation of the system in question but has the disadvantages of inflexibility and relative lack of power. That is, the connections of the measuring device to the system are semi-permanent and difficult to reconfigure, in general, and the quantities that can be measured are frequently not easy to identify with program operation.

The software technique has corresponding but inverse characteristics: it is eminently flexible with many changes being possible on-the-fly, during system operation via a terminal. Also, since it is implemented from programming, it is able to monitor program activity in its own terms. (Of course, hardware activity can be measured only indirectly by this technique.) The disadvantage of a software measuring technique is that it always, to a greater or lesser extent, interferes with the operation of the system and makes any conclusion dubious which is dependent on or is affected by timing.

One hardware device that seems not to have been described earlier, is a small associative processor set up as a computer performance measurement device. Using passive sensors and detecting the contents of the instruction register of a computer, the associative processor can be used to create an updated file on the number of times each instruction has been used during an application program. The construction of the associative processor is given below.



At each computer execute-time the contents of the instruction register are transferred to the AP input register and a search is made for a match in the associative array. If a match is made the contents of the adjoining 8-stage counter is incremented. These counters are fabricated in 4-stage chips which have an 18 nsec delay per chip. A total of 512 counterchips will be used. An alarm bit per counter will signal if the 8-stage counter overflows. The search-plus-increment time will consume 61 nanoseconds which is faster than the CPU clock time of most computers.

An extra tag column attached to the associative array can be used to denote which words are presently in use. Whenever a new or non-previously used instruction enters the input register it is deposited in the lowest available empty word. When 255 words are filled, the 256th word is used for miscellaneous instructions that are different from the 255th.

When the program is completed the AP memory can be read out by means of the output register shifted out in 8-bit bytes. A simple analysis program in the computer can then assemble the data and prepare histograms and other output.

2.5 CONCLUSIONS

From the above three sections--on data processing loads, design considerations and evaluation techniques--three general conclusions can be drawn.

First, there is not available at the present time a really definitive study of data processing system requirements in the 1980-90 decade. While new traffic estimates may be desirable, the real need is for an up-to-date study of the shape, form and size of the computing system needed in the near future.

Second, the ATC system of even the very near future could be quite different from anything being used or built today if the latest developments were pursued. Most of these developments are of an evolutionary nature, building on current technology, but many could have great effect on overall system design.

Third, there is no single method by which systems may be designed, measured or evaluated. Of the four--analysis, simulation, synthesis, measurement, each has a place in some phase of the process. The most that one can say is that (1) analysis is indispensable, during the early stages especially, (2) small-scale simulation is very useful, in the middle stages in particular, (3) large-scale simulation is sometimes justified despite its great cost, (4) synthesis, when used with small-scale simulation and analysis, is probably the best way to arrive at the final design, and (5) measurement of system performance is at present in its very rudimentary stages but will have to become much more sophisticated if systems are to be properly evaluated.

References:

[1] Ashby, W.L., Derivation of Aircraft Activity Estimates, Report of DOT Air Traffic Control Advisory Committee, Vol. 2, Appendix G-1, pp. 467-477, December 1969.

[2] Ashby, W.L., Future Demand for Air Traffic Services, Proc. IEEE 58.3, (March 1970), pp. 292-299.

[3] FAA, FAA Statistical Handbook of Aviation, yearly editions.

[4] CAB, Forecast of Domestic Passenger Traffic for the Eleven Trunk-line Carriers: Scheduled Service - 1968 - 1977, CAB Research Study, Sept. 1968.

[5] FAA, FAA Air Traffic Activity: Fiscal Year 1968, FAA Office of Management Systems, August 1968.

[6] ATA, Air Transport Association Airline Airport Demand: Industry Report, July 1969.

[7] Graham, W., Terminal Air Traffic Model With Near Mid-aid Collision and Mid-air Collision Comparison, ATCAC Report, Vol. 2, Appendix C-3, pp. 151-164, December 1969.

[8] Alexander, B., Aircraft Density and Mid-air Collision, Proc. IEEE 58,3, (March 1970), pp. 377-381.

[9] Nelson, J.C., Final Report of the Computer Sizing Group for the 1980 Air Traffic Control Committee, Report of DOT ATCAC, Vol. 2, Appendix D-1, pp. 221-240, December 1969.

[10] Blake, N.A. and Nelson, J.C., A Projection of Future ATC Data Processing Requirements, Proc. IEEE 58, 3 (March 1970), pp. 391-399.

[11] Holland, F.C., Computer Sizing of En Route Command and Control for the ATC Advisory Committee, WP-8272, MITRE Copr., 22 April 1969.

[12] Vacca, A.A., The Case for Emitter-Coupled Logic, Electronics 44,9 (26 April 1971) pp. 48-52.

[13] Franson, P., Schottky TTL Blunts ECL Growth, Electronics 44,5 (1 March 1971) pp. 69-72.

[14] News Item in Electronics 44,5 (1 March 1971) pp. 19-20

[15] News Item in Electronics 44,4 (15 February 1971) p. 25 and 44,8 (12 April 1971) pp. 36-37.

[16] Session 34, SJCC 1971 The New Technology: Computer Architecture, four papers on Symbol IIR to be published in Proceedings SJCC 1971.

[17] Calingaert, P., System Performance Evaluation: Survey and Appraisal, Comm. ACM 10,1 (January 1967) pp. 12-18.

[18] Johnson, R.R., Needed: A Measure for Measure, Datamation (15 December 1970) pp. 22-30.

[19] Campbell, D.J., and Heffner, W.J., Measurement and Analysis of Large Operating Systems During System Development, Proc. FJCC 1968, pp. 903-914.

3. SURVEY OF SYSTEMS AND TECHNIQUES

3.1 INTRODUCTION

The architecture of individual computer systems, some of which can handle today's and some which even handle tomorrow's ATC data processing requirements, will be described in detail. Four types of systems are included. These are uniprocessors (CDC STAR and TI ASC are examples), multiprocessors (IBM 9020, ARTS III), array computers (ILLIAC IV, PEPE), associative processors (Goodyear AP, New AF AP, HAM-H832).

As explained in the previous section, the evaluation of these, e.g., reliable performance per unit cost, where cost includes long range as well as initial expenditures, is a difficult chore. It has been proven in the past that extra cost produces a greater performance per unit cost ratio, but since performance sufficient to solve the air traffic control problems of the 1990's is all that is required, extra capability is not warranted. The best available evaluation techniques are simulation and measurement hardware and software. All the major manufacturers rely on efficiency measuring hardware to produce systems which make the maximum utilization of components. Some even offer maintenance (error-detection) software, i.e., monitors which help isolate failed subsystems and provide continued service until their service facilities can make repairs.

Intrinsically, the uniprocessor system suffers from the reliability standpoint and a duplex configuration is needed to make it viable. The same may be said for the associative processor, which up until now has worked in tandem with a serial processor. The multiprocessor competes with the duplex configuration (either with uniprocessors or associative-serial combinations). Parallelism will work its way in the design of all future computers; the trend toward large-scale integration circuits will insure this.

3.2 COMPARISONS OF STRUCTURES

The uniprocessor, array processor and associative processors are structures which utilize a single instruction stream, and the effort has been to speed up this stream. The multiprocessor uses multiple instruction streams and the effort here has been to reduce lock-ups or interferences and make the processing operate more smoothly. The uniprocessor employs a single data stream and a single instruction stream while the others employ multiple data streams with a single instruction stream or a single data stream with multiple instruction streams. Multiple data streams controlled by a single instruction stream is the mode which is common with array and associative processor.

The uniprocessor often relies on parallel organization to speed up the throughput of the execution element. Memory look-ahead and separate I/O processors relieve the execution element. A large main memory saves going to secondary storage. It is not astonishing to find that the costly item of the uniprocessor is the central processor. Adherents to this structure feel that it can do nearly all jobs most efficiently; keeping in mind that greater performance per cost is achieved (Grosch's law) by paying more for a "speeded-up" processor.

Array processing assumes that it is more efficient if several data streams are processed simultaneously. In the standard type, multiple processing units are stimulated by a primary processing stream. The individual processing element has its own stored program which can be tailored to its specific data stream. The second type, or associative processor, depends upon logic-in-memory, i.e., each memory bit has a match circuit attached which permits content-addressing. This type of memory costs more, about one-half of the system, but permits fast parallel search operations.

Array processing becomes inefficient when processing elements are idle most of the time. The processor arrays become more productive when decision-making and logic capabilities exist at the lower levels.

The multiprocessor contains several processors, each capable of performing the tasks alone, sharing memories. The total problem is segmented and each segment is performed autonomously. There is, therefore, a dynamic allocation of segments of the problem among the available processors and a substitution of processors in the event of the failure of one. Since there is an overhead charge (both hardware and software) for this type of system structure and since the concept of substituting several small computers for a large one is a violation of Grosch's law, the multiprocessing advantages are reduced. The prime justification for this structure is reliability. There is not much evidence as to the actual reconfigurability that has been obtained.

After this prologue, the individual computing systems which will influence future ATC data processing systems will be reviewed.

3.3 IBM 9020 MULTIPROCESSOR

The 9020 is a special configuration of IBM 360 components especially modified and assembled to operate as a multiprocessor at the NAS ARTCC's. As such, it is an application of third-generation computer technology to a specialized function, namely air traffic control.

The computer hardware system was planned to operate with the monitor system provided (the NAS monitor) so that its particular features make sense only in the particular environment in which it will be found. Hence, this discussion will treat the computer-monitor combination as a whole.

There are three versions of the 9020 extant, the A, D and E models: the 9020A is the original version using 360/50 components and the D and E are later versions using 360/67 components. The original plan, which would have used the 9020A for the complete processing at a center, has given way to a new arrangement in which the major computing load at a center (radar correlation, tracking, flight plan processing, etc.) is done by a 9020D which passes data to a 9020E for display preparation and generation.

3.3.1 System 360 Features. Since the 9020 has been developed from 360 components, it shares with that system a number of architectural features, which we will discuss at this point. The most obvious involve the combination byte/word orientation of the machine and the general register approach to the computing element. Memory is addressed in 8-bit bytes while arithmetic is done, for the most part, on 4 byte, 32-bit words in a group of 16 registers which serve as accumulators, index registers or base registers. The familiar 360 order code, including binary, decimal and floating-point arithmetic and logical operations, is implemented in the computing elements. The 360 I/O pattern is also familiar: start and stop instructions in the CE's, with I/O channel commands in main storage accessible by the channel controllers. The interrupt capability and arrangement are likewise as implemented on the 360 system. Finally, the two states, supervisor and problem, have been implemented, so that the monitor can run in the supervisor state in the familiar way.

To the programmer, the 9020 is basically a 360 with additional capabilities; in fact, the 9020 can operate in the 360-mode by suppressing these special features. Hence, many useful programs have been carried over from earlier efforts to the 9020 and can operate unchanged.

3.3.2 9020 Features. All three 9020 systems have certain characteristics or features that have been developed by IBM to deal with the special environment in which they will be used. The objective of these features is to provide a system that is ultra-reliable and which is modular, for adaptation to particular sites. Both requirements were met by making the 9020 a multiprocessor system with redundant components and a reconfiguration capability. That is, the system (including the Operating System) is able to detect errors and failures and to initiate remedial action automatically.

In the first phase, the so-called "fail-safe" mode, a failing element is replaced by a spare element of the same kind and the system is reconfigured to carry on as before the failure. The whole operation should take no longer than 30 seconds and be accomplished without operator intervention. The second phase is called the "fail-soft" mode and is really a graceful degradation of a system which cannot continue in complete operation. By means of a pre-established schedule, the system reconfigures as best it can and drops out non-essential functions while slowing down the less critical functions until the processing load is reduced to the limited capability of the partially failed system.

A. Multiple Computing Elements. With this as background, one can examine the 9020 features required over and above the 360 capability. First is the requirement for multiple processors: in order to provide varying amounts of computing power at different sites and, at the same time, allow for redundant computing elements, it is most economical to provide a single size of CE which can be used in a multiprocessor configuration with varying numbers of the common CE working together or serving as backup. The system has been organized so that the control program can be operating in any or all of the CE's simultaneously, that is, there is no master-slave relationship among the processors. In general, the operational subprograms--the ones which do the ATC work--operate in the CE's, while the control program takes control only to answer interruptions from I/O operations, end-of-job, interval time-outs, etc.

Special instructions have been implemented on the 9020E processors to enable them to process radar and weather data and produce display tables for storage in the Display Elements, described below, and eventual use in driving operator displays.

B. Storage Elements. The main memory of the 9020 system consists of modules called storage elements (SE).

These are of several sizes and speeds. The original elements contained 32K 4-byte words with a 2.5 usec read and restore cycle time. Elements of the same type with twice the capacity (65K) were later made available. The SE's used in the D/E series have 65K 8-byte double words with a double-word read-restore cycle of 0.8 usec.

Each byte in these memory units is directly addressable and special hardware registers are provided to make automatic translation of logical addresses to physical addresses. Thus, it is possible to configure the SE's in any order by changing the contents of the logical-physical correspondence registers.

The cycle time of the SE on the 9020 D/E is not directly related to that of the CE, so that each can operate at its optimum speed. Data are transferred to the CE a double-word (eight bytes) at a time and to an IOCE (see below) one word at a time. Further, memory interleaving for odd/even double-words is provided. These three--independent cycle times, double-word transfers and memory interleaving--will tend to reduce the impact of memory interference between two CE's trying to access one SE simultaneously by allowing the SE to grant accesses at its maximum rate not constrained by the earlier access, by reducing the number of separate accesses required by each CE and by allowing essentially simultaneous access about 50% of the time.

C. Input-Output Control Elements. Input and output are under control of the Input-Output Control Elements (IOCE) in the usual 360 fashion. The CE's issue start and stop instructions to the IOCE's which then read from a sequence of pre-stored I/O commands in memory and initiate the I/O actions independent of the CE. Completion of the I/O processes cause I/O interruptions to be generated which are routed to the proper CE for interpretation. Each IOCE has one multiplexer (shared) channel and two selector (single-device) channels, all of which it can operate simultaneously. (There is an option to add a third selector channel to as many as two IOCE's in the system.)

Originally, the control of I/O was the whole of the task assigned to the IOCE's. It was seen, however, that since they were really heavily adapted CE's, they possessed a great deal of processing capability that was going unused. This capability was made available to the system by a direct control path from the CE's and implementing an instruction in the CE which would start the IOCE processor and an interrupt from IOCE to CE to signal end-of-job or other condition.

The IOCE in these later models can be thought of as two distinct elements sharing some common logic and a 32K word 2 usec internal memory--the so-called Maintenance and Channel (MACH) memory. The two elements, the IOCE-processor and IOCE-channel controller, can operate independently and simultaneously. The IOCE-processor can neither initiate its own or other IOCE-processor operation nor initiate I/O operations. It can, however, under control of a CE execute a subset of the 9020 instruction code accessing either main storage or MACH storage for its instructions or data. Decimal and floating-point arithmetic are not part of the IOCE-processor instruction set.

D. Peripherals. Various kinds of peripheral equipment must be operated by or must supply data to the 9020 complex. Most of these devices operate through the channels of the IOCE after interfacing with various special and standard adapter units. In particular, a Peripheral Adapter Module (PAM) is supplied in order to interface the various radar, controller keyboard, interfacility and other inputs. Tape Control Units and Data Adapter Units interface with Magnetic Tape Units and radar keyboard multiplexers. Finally, a system console communicates with a multiplex channel on the one hand and a printer and card-reader/punch on the other.

E. Display Element. A component of the 9020 system unique to the 9020E is the Display Element (DE), a memory of 32K 8-byte words. This memory differs from the usual SE in that it has special features to enable it to operate as a I/O controller, in effect, for a group of Display Generators (DG) connected to it. Each DG contains 6 Character Vector Generators (CVG), each of which, in turn, controls a Plan View Display (PVD). Eight Display Generators may be connected to a DE, but only four of them may be active at any one time; each DE, therefore, may be controlling the displays on as many as 24 PVD's at a time. (Note: the DG's, CVG's and PVD's are not part of the 9020E system, but are supplied by other manufacturers for connection to the 9020E.)

In order that all of the display devices are refreshed at the maximum rate and at the right times, the DE has been constrained to operate synchronously. The CVG's request data asynchronously but requests are granted on the basis of a fixed priority scan. Data are transmitted 16 bytes at a time to the CVG's at their assigned time if they are requesting service at that time. If a CVG does not use its time slot, it is made available to a requesting CE, if any. If there is none, the slot remains unused. (The CE's also are allowed an access after every 8 CVG accesses.)

The displays are refreshed at a maximum rate of 55 times per second by requiring that initialization take place only at fixed times clocked at that rate. If regeneration takes longer than 55^{-1} seconds, it is allowed to occur continuously to maximize the refreshment rate.

To the CE, the Display Element looks like any other Storage Element.

3.3.3 Operational Configurations. The minimum configuration must obviously have at least two of each element for redundancy purposes. The normal maximum configurations plus the expansion possibilities are given in Table I. The expansion possibilities require additional logic as well as down-time for installation in the field in most cases.

TABLE I

Maximum Normal Configuration and Expansion Possibilities

	9020D	9020E
CE	3 + 1	3 + 1
SE	8 + 2	4 + 1
DE	-	4 + 1
IOCE*	3	2 + 1
TCU	3	2 + 1

* Each IOCE has two selector channels and one multiplexer channel which is expandable to three selector channels and one multiplexer if the total of selector channels is not greater than eight.

Each element of the system contains a Configuration Control Register (CCR) which is set by a Set Configuration (SCON) instruction issued by a CE. Which CE's can issue the SCON and which elements will react to it depend on the contents of the CCR's of the elements involved. The CCR consists of three fields: the state field, the set configuration field and the communication field.

The state field puts an element into one of four states, 3, 2, 1 and 0, which correspond roughly to on-line active, on-line redundant, off-line available, and maintenance. A CE can issue the SCON instruction only if it is in states 3 or 0. The system can be so configured that an element check (ELC) in the active CE will cause a redundant CE to be made active (its state changed to 3) and be interrupted causing a transfer to an error routine which will then cause the newly active CE to take over the functions of the failing CE.

The set configuration field determines the CE's which are allowed to set configurations, i.e., to change CCR's. The communication field determines the elements to which the element in question will "listen". For two-way communication, element A listens to element B and element B in turn must be able to listen to A.

Communication between CE's may be accomplished in three ways. First is through the sharing of a storage element, wherein one CE writes and the other reads. Premature or interfering accesses are prevented by the use of a TEST AND SET instruction, which allows a CE to gain access to the memory and lock out any other CE until it has finished whatever processing it needs to do.

Secondly, a Direct Access connection is implemented which allows one CE to interrupt another and to transfer a byte of information directly.

Lastly, a channel-to-channel adapter has been provided which allows CE's to establish communication via two IOCE's properly configured. In fact, it is this feature which is used to connect a 9020D in the Central Computer Complex (CCC) to a 9020E in the Display Channel Processor (DCP) of a complete ARTCC.

	9020A	9020D	9020E
word length (bits)	32	32	32
word length (bytes)	4	4	4
instruction rate (MIPS)	.8(1)	2.1(1)	2.1(1)
failure rate MTBF (hrs)	61.7(2)	-	-
effective memory cycle time for 32 bits (nsec)	2500	200	200
processor cycle time (nsec)	500	200	200
add time (usec)	5.00	1.63	1.63
multiply time (usec)	28.38	5.03	5.03
memory parity bits (bits/word)	4	4	4
main memory size (1K words)	65-384	260-1300	260-650
error detection in logic	yes	yes	yes
automatic instruction retry	no	no	no
automatic reconfiguration	yes	yes	yes
hardware subsystem failure detection	yes	yes	yes
compilers	FORTRAN, COBOL JOVIAL	FORTRAN, COBOL JOVIAL	FORTRAN, COBOL JOVIAL
operating system	NAS OS/360	NAS OS/360	NAS OS/360
mean up time (hrs)	33543	>1000(3)	>1000(3)
mean down time (hrs)	0.29	<1.0(3)	<1.0(3)

(1) Based on 2 CE's and 2 IOCE's--the latter at 80% rated speed.

(2) From "Final Report of IBM 9020 System Reliability Determination" June 1967 for 325 flight, A1 system.

(3) From specifications: FAA-ER-NS-100-1

3.4 ARTS III MULTIPROCESSOR

The computer complex being developed for the ARTS III program is a moderately sized multiprocessor consisting of relatively conventional components. The computer is being developed in stages, starting with a single processor configuration and progressing toward a multiprocessor configuration composed of a number of unlike elements. The basic computer is a development by UNIVAC of its military 1230 computer. It is called the ARTS III Input-Output Processor (IOP) in its current manifestation.

In its final form, the system will have multiple IOP's, as well as a number of units, specialized for computation, called Central Processor Modules (CPM's). Core memory is supplied in 16K modules. The maximum configuration is limited by certain register lengths to eight processors (a mixture, presumably, of CPM's and IOP's) and 16 memory modules. Input and Output are done through 16 input and 16 output channels built into the IOP.

To provide for continuity of operation and/or fail-safe, fail-soft operation, the complex is connected to a Reconfiguration and Fault Detection Unit (RFDU). This device can detect and report on failures of units in the system and can reconfigure the complex on command of the system executive.

The multiprocessor executive is, of course, a fundamental element of the whole system, controlling as it does the allocation of system resources to operating tasks within the system.

We discuss each of the system elements in turn.

3.4.1 IOP-processing function. The IOP is essentially a general-purpose computer with both computational and input-output capability. The latter will be discussed in the next section.

3.4.4 Memory Modules. The memory of the ARTS III system is made up of modules of 16K 32-bit words each. Physically, each module is made up of four 4096 word core stacks with access time of 275 nanoseconds and read cycle time of 650 nsec. Each word is divided into upper and lower halves of 15 data bits plus one parity bit. The whole word may be accessed or either the upper or lower half separately. As many as 16 modules may be incorporated into the ARTS III system, giving a total memory of 262K words.

3.4.5 Multiprocessor modifications. In order for the system to be used in a reasonably efficient way as a multiprocessor, certain important hardware functions have been added to the basic IOP/CPM system. These functions, in general, provide for logical-physical unit identifications, memory read/write protection and provision for an "executive call" function. The implementation takes the form of additional control registers and additional processor instructions. The so-called "Relative Address Allocation and Memory Protection" feature adds 110 nanoseconds to the effective memory cycle time when enabled.

3.4.6 Multiprocessor additions. At least two additional units of hardware must also be developed in order to make the ARTS III multiprocessor system viable: a Reconfiguration and Fault Detection Unit (RFDU) and a Centralized Memory Access Module (CMA).

The RFDU is a dual-purpose device, as its name implies, to which all hardware error indications are passed and stored, and which can make and break connections between memory units and processing units. When a fault has been detected in a unit, the RFDU can interrupt a non-faulted processor for action on the fault. This processor then issues instructions to reconfigure the system by cutting out the unit in error and bringing a replacement unit on-line. If insufficient redundancy is not available, the unit can only reduce the configuration, leaving the adaptation to the executive program.

3.4.7 Multiprocessor executive. In order that the system operate effectively, it must be under the control of a supervisory, or Executive, program which controls scheduling of jobs and allocation of facilities among the jobs. Such a program is being written for the ARTS III system. It will provide for efficient processing within the Air Traffic Control environment in its initial version and will be developed to provide for automatic fail-safe/fail-soft operation in later versions.

ARTS III MULTIPROCESSOR

	*	**
work length (bits)	30	30
instruction rate (MIPS)	1.2†	3.9††
failure rate MTBF (hours)	N/A	N/A
effective memory cycle time (nsec)	750	1000
processor cycle time (nsec)	750	750
add time (nsec)	1.5	1.750
multiply time (usec)	6.8 - 13.0	6.8 - 13.0
memory parity bits (number)	2	2
main memory size (1K words)	131K (8 x 16K)	262K (16 x 16K)
error detection in logic (yes or no)	yes	yes
error correction in logic (yes or no)	yes	yes
automatic instruction retry (yes or no)	no	no
automatic reconfiguration (yes or no)	yes	yes
hardware subsystem failure detection (yes or no)	yes	yes
compilers (type)	macro assembler	macro assembler
operating system (type)	ARTS III executive	ARTS III executive
mean up time (hours)	>830[x]	>830[x]
mean down time (hours)	<0.5[x]	<0.5[x]

* based on 2IOP, 8 memory module configuration
** based on 4 CPM, 4 IOP, 16 memory module configuration w/CMA
† rate for one processor + .8 (rate for one processor)
†† 2(rate for one processor) + 6 x .8(rate for one processor) (with memory protection)
x requirement of FAA-TD/S-120-801 for "Simplex system"

3.4.8 System Improvement. In order to measure, analyze, and improve the total hardware and software aspects of the ARTS III computer system, four measures have been taken by UNIVAC. One is a Memory Queuing Model (MQM) written in the discrete high-level simulation language employed by the UNIVAC Simulation Package (USP) 2.0. This model determines the queues and queue times existing at a shared memory referenced by two IOP's, the main processors in the present version of the ARTS III computer, utilizing a distribution of referencing rates to the shared memory. The model is composed of two IOP's and two memory banks and has the capability of expanding to 3 IOP's. Peripheral (I/O) activities were not included in this model. Instruction processing was considered sufficiently valid for studying the queue characteristics at the IOP-memory interface. An ATC Instruction Mix, a modified Gibson Mix was used to exercise the MQM.

The second measure was a Hardware Characteristics Model (HCM) that was used to investigate the ARTS III Data Processing Subsystem (DPS) hardware performance and to examine the effects of various DPS operational characteristics, loads, and other aspects of its intended environment. The model contains representations of an IOP, Memory, Data Acquisition Subsystem (DAS), Interfacility Communications Adapter (ICA), Magnetic Tape Unit (MTU), Typewriter (TW), and Display Consoles. A similar (as with the previous model) Gibson type mix of instructions were used to exercise the HCM. Both models are presently being converted to the UNIVAC System Simulation Language (USSL) to increase run time economy and overall utility.

The third endeavor at improving the system performance was a manual method of analyzing the time spent in performing subtasks.

The fourth was an automation of the third. It is a tracer program for the timing of functions in the programs. The tracer counts the types of instructions, and records where branches take place.

3.5 CURRENT ATC EXECUTIVE SYSTEMS

The NAS and ARTS III air traffic control systems, as described in Sections 3.3 and 3.4, are dependent for their correct operation on the Executive Systems that have been designed for them. In both cases, the actual air traffic control operations will be conducted by a set of programs--called variously applications subprogram, problem or work programs or, simply, tasks--which will not be discussed in this section. These are the programs which will do the radar data correlation, tracking, display preparation, flight plan processing, etc. The Executive Systems, on the other hand, have the responsibility for scheduling of tasks, allocation of resources, fault detection and handling and input/output control. In a sense, then, the Executive is the software framework on which the whole system is built.

There are a number of design principles which must be observed in constructing an executive system, and more especially, a multiprocessor executive system. In some systems, provision is made in the design of the hardware for these principles, so that the software design is relatively easy and system performance is high. Other systems achieve the required capabilities by adding circuitry to the hardware, retaining the programming convenience but sacrificing speed. Finally, it would be possible to program around most requirements but the result would be a complex and slow executive and an extremely limited system in terms of capability and capacity.

The inference to be drawn from these remarks is quite clear; it is, indeed, not a new idea at all but has been a basic tenet of disciples of good design practice for a long time. It is that all aspects of the system must be developed together to produce the most efficacious design. This is particularly true when considering development of fourth-generation systems because new technologies have blurred the traditional lines between the provinces of hardware and software. We will talk here about principles which have been involved in present-day system design but which, from our point in time, would seem to carry over to the newer systems coming up.

The two general classifications into which the principles fall are protection and control. The system must have the means to protect itself from damage or destruction by actions of the software, either the application programs or the executive, or by actions of the hardware, through a machine fault or by the reaction to bad data from whatever source. Furthermore, the system must be able to exercise control over the actions of its component parts in order to maintain efficient operation of the system as a whole. No one application program, for instance, must be allowed to monopolize resources needed periodically by other programs, or to prevent the operation of some other function by suppressing the event which triggers it.

The paragraphs below state--in rather general terms and with no attempt to be rigorous--four "principles" and some of the possible implementations of them.

1. There must be in the system at least two classes of processes, or programs, of which one must consist of the executive, alone. The executive (class) must be a privileged class with sole and direct control over those operations which change the "status" of the system in certain fundamental ways. Typically, the instructions are included that control i) input/output operations, ii) changes to the system configuration and iii) transfers of control among independent programs.

The simplest implementation of this idea involves the establishment of two states--say, supervisor and problem--with a flag whose set condition indicates supervisor state, for instance. Setting and clearing of the flag may only be done from the supervisor state; any attempt to do so from the problem state causes an identifiable interrupt. Certain other instructions (I/O, etc. as mentioned above) are also valid only when the flag is set and attempts to execute them cause another identifiable interrupt. One instruction must be available to the problem to allow transition back to the supervisor state: this instruction would set the flag and cause a special identifiable interrupt. Following sections will describe the implementation in the NAS and ARTS systems.

2. The executive must control the allocation of resources among the programs in the problem class (es). The resources in question include computations, storage and input/output units. In general, the allocation should be done dynamically, i.e., a) the executive operating both periodically and at the end of each problem program schedules the problem programs as needed, b) each program makes requests for additional resources as it needs them and these requests are filled as the resources become available.

In order that the program-controlled allocation be possible, a number of hardware provisions must be made. The problem programs must be able to request resources, implying a special supervisor call interrupt, possibly the one described above. The executive must have a means for identifying the processor on which it is operating so that it will know which processors are available to assign to other programs. Finally, address translation registers must be provided so that the executive can assign memory units as it wishes to the problem programs and have correct addressing done automatically.

3. A mechanism must exist by which storage resources may be protected against unwarranted uses. That is, when a region of memory has been assigned to a particular program other programs must be prevented from writing there or executing the code there if these actions would interfere with the first program. The establishment of read only and execute only states in conjunction with the addressing hardware mentioned above can provide this protection.

Some areas of storage will need to be accessed by two independent programs but not simultaneously. The uninterruptable TEST AND SET instruction would allow a program to gain access to a data table, say, while at the same time denying access to any other program.

4. Transitions among programs must be done in an orderly fashion under control of the executive. This implies that the interrupt system must be controlled by the executive. One way to implement this would be to have all interrupts lead directly to the executive which will then decide on the basis of type of interrupt and stored status and environment information where to transfer control. Another possibility would be to allow the executive to direct the interrupts to the appropriate problem program on the basis of a priori information.

The accompanying figure is a simplified diagram of the system showing the major transfers of control. In the following, we will discuss each part of system as shown, starting with the applications subprograms and progressing through the various parts in the order in which control is passed from one to another. By proceeding in this way, the startup portion is described last--a seeming anomoly. It is hoped, however, that the presentation introduces concepts and ideas in a more natural way than would a strictly chronological development.

The closing sections will discuss those aspects of the system design that are unique to or required by a multiprocessor executive.

3.5.1.1 Applications Subprograms. This class of program occurs in all systems, though under various names: problem programs, functional subprograms, operational programs, etc. It consists of that group of programs which do useful work on the data base of the system--those that apply the transformations to the input data to produce output data in its prescribed form. In the NAS system, an applications subprogram gains control of a CE when the monitor dispatcher executes a Load Program Status Word (LPSW)whose instruction address points to the program in question. How this particular program was selected for operation will be discussed below.

The applications subprograms operate in what is called the "problem state", while the executive system operates in the "supervisor state": this means, essentially, that the former cannot (1) issue input/output instructions or (2) change the operational configuration or state of the machine. This necessary restriction imposes no hardship on the applications subprograms because indirect means are made available to them to carry out any legitimate operation of these kinds. The indirect means referred to is the Supervisor Call (SVC) interrupt which may be triggered by any program executing the SVC instruction with an eight-bit code appended. Control is transferred to the executive and the code passed along, thus providing a mechanism for indicating which service is being asked for of the nearly 30 that are provided. As with interruption schemes

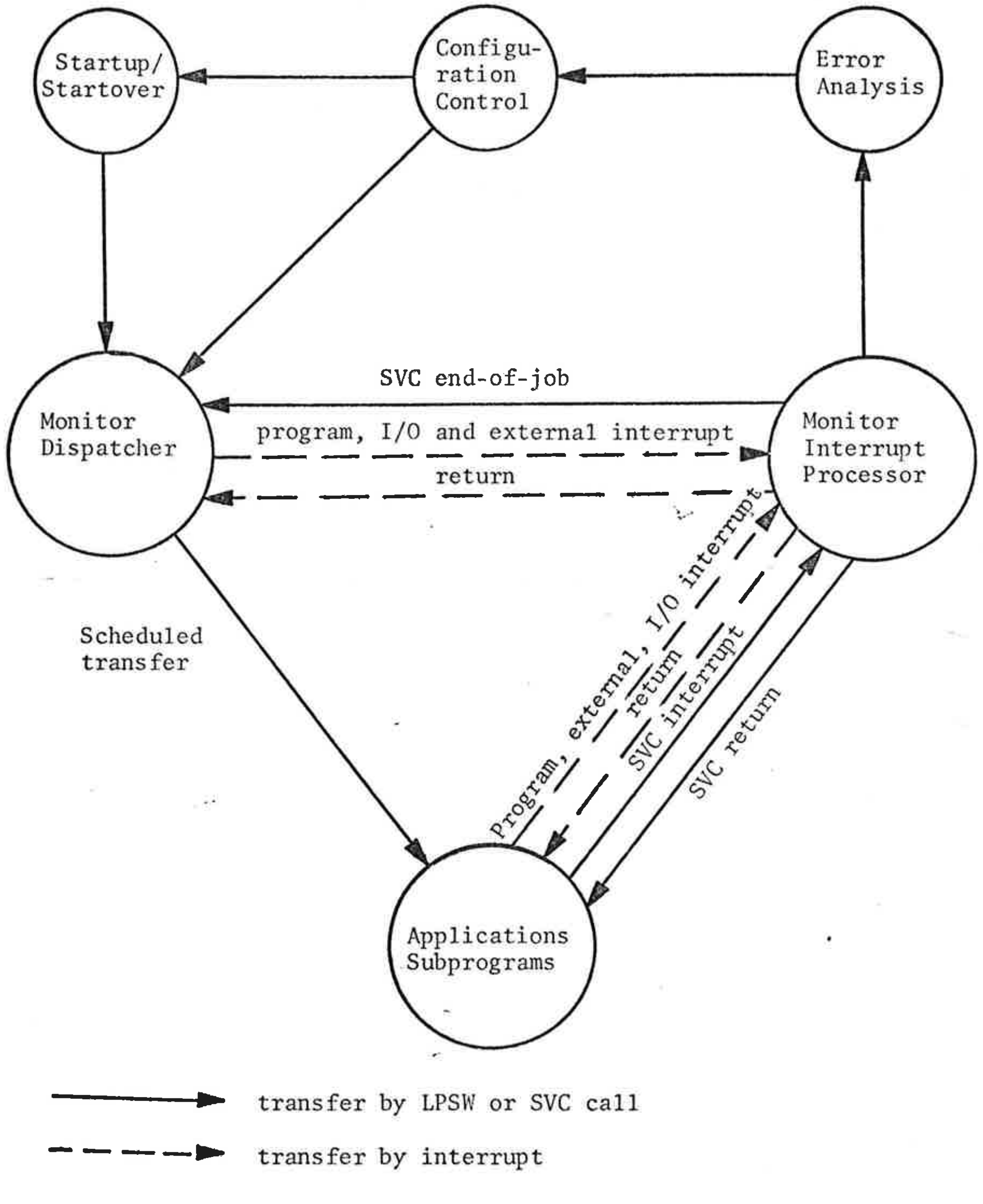


Figure 3.5.1
SIMPLIFIED DIAGRAM OF CONTROL FLOW IN NAS SYSTEM

in general, the state of the machine at interruption is saved so that a return to that point is possible; it is not necessary, however, as one call, SCV FINIS, is used to signal the end of operation of an applications program. Some of the other calls request I/O operations, status indications, resource allocations and housekeeping functions.

Transfers out of the applications programs also occur on other interruptions which, with the SVC interruption, are directed to the Monitor Interrupt Processor. There are, in general, two conditions which cause interruptions: on the one hand, the interruption may signal the occurrence of an event which, expected or not, is within the normal purview of the system. Events such as end of I/O operation, input of data from external source, and arithmetic overflow fall in this class. On the other hand, the interruption may signal an error condition in either the hardware or the program. Incorrect parity detected during data transfer, attempts to write in a protected area of storage and illegal operation attempts are examples of this condition.

In the case of interruptions such as those involved with I/O operations, the Interrupt processor will return control to the subprogram after taking the appropriate action. Interruptions such as arithmetic overflow are the result of conditions that can be handled by the subprogram, so control is returned immediately to a portion of the subprogram set up to handle the situation. In the other cases, there is real problem in the subprogram, or system, operation which must be handled by the executive.

3.5.1.2 Monitor Interrupt Processor. The Monitor Interrupt Processor, really a set of loosely-linked programs, stands by ready to act upon the occurrence of any input/output interruption or any "exceptional" condition, to use the IBM term. The general reaction of this program to an interruption was described in the section above; a more detailed look follows.

3.5.1.2.1 <u>SVC Interrupts</u>. The SVC interrupts come only from the applications subprograms and carry with them a code indicating the monitor action desired by the subprogram. Each type of request is handled by a separate routine designed to handle that particular job. The SVC FINIS routine, for instance, frees all storage resources used by the subprogram, cancels its execution and transfers control to the Monitor Dispatcher.

One group of SVC's is associated with the allocation of storage areas to the subprogram. Three types of storage are defined: areas, blocks and lines. An <u>area</u> is a section of storage which contains data or code which is used by two or more subprograms; areas must be protected against simultaneous access by two subprograms to avoid loss of data or use of erroneous data. A <u>block</u> is a section of storage to be used either as a temporary work area by a subprogram or a storage area for data put into an I/O or subprogram queue. A <u>line</u> is a section of storage used to store identification data used in transferring a block from one subprogram to another through its queue or to an I/O device through its queue.

A subprogram gains control of areas, blocks and lines by executing the SVC LOCK, LEASE and RESERV, respectively; it returns them via the SVC UNLOCK, RELEAS and CANCEL, respectively.

During its operation, a subprogram, having a certain number of blocks and lines assigned to it, can develop its data in the blocks and transfer control of them to other subprograms and/or I/O devices by setting up the lines with the proper information (block number and I/O or subprogram queue identification) and by then executing the appropriate SVC. When subprogram A passes a block to subprogram B via a line, A executes SVC SEND which causes B to be initiated. B, in turn, executes SVC ACCEPT. When B is finished with the block, it executes SVC DELETE, although it could pass the block along with SVC RELAY. SEND and RELAY may be used to pass data to I/O devices, as well.

A second group of SVC's is concerned with passing input/output requests to the Executive, which has sole capability of initiating I/O operations. These SVC's range from requests for particular services (CONTRL asks for execution of an indicated I/O control operation on a specified device) to requests for complete operations (READ requests that a record be read from the indicated device).

The third category of SVC's provides for various services such as scheduling of other subprograms, supplying of status information, causing so-called Legal and Analysis recording, etc. Clearly, the response to each of these is tailored to the particular request, and clearly, other SVC's can be added to the system with very little effort.

In general, the Interrupt Processor will return control to the subprogram, sometimes immediately and at other times upon the occurrence of a prescribed event, such as an I/O operation.

3.5.1.2.2 Other Interrupts. Various other conditions, none directly controlled by the subprogram, cause control to be transferred to the Interrupt Processor. These are classified in the usual S/360 manner as program, input/output, machine-check and external.

The program interrupts are caused by an occurrence of an exception of either of two types, as mentioned above. One type arises when the data being handled by a program is faulty or atypical. Typical of these faults are the overflow and underflow exceptions which could occur if a word of input data were affected by noise in the transmission line, say. Then, if there were no checking of the data before use and the word was outside the acceptable range, the exception might occur when the word was used.

The program using the data should be written in such a way that it can deal with situations of this type, so when this kind of exception occurs and the interruption comes from an applications subprogram, the interrupt processor checks to see if a return was requested. If so, control is returned there.

On the other hand, if the interruption was from the Monitor Dispatcher, or was from a subprogram with no return requested or was of the second type, transfer is made to the Error Analysis routine, discussed below. The second type of program exception mentioned is that type which arises by an error in the program or some kind of machine malfunction. Typical of these are attempts to execute illegal instructions, attempts to address a location outside of available memory and attempts by an applications subprogram to execute a privileged instruction. One, the SE-stopped exception, prevents system hangup if, for some reason, a storage element fails. All of these conditions, when detected by the Interrupt Processor, lead to a transfer of control to the Error Analysis program.

Input/Output interruptions occur either upon completion of a requested operation or the occurrence of an abnormal condition connected with an I/O operation. In the first case, the Interrupt Processor is able to take appropriate action because of prestored information and prearranged queuing facilities whereupon it transfers control to the Monitor Dispatcher. In the second case, the Interrupt Processor will, in general, attempt to retry the I/O operation, perhaps with a different I/O device of the same type as the one giving trouble. If such attempts are not possible--or are unsuccessful--control will be passed to the Error Analysis Monitor or to the Dispatcher, depending on the exact nature of the error.

Machine-check Interruptions are passed to the Error Analysis Monitor for disposition, while External Interruptions, the remaining type, are handled according to whether they are "normal" or "abnormal". Those whose origins are in devices which use the interrupt as an attention-getting device are handled according to prearranged plans, while those which arise from error conditions such as On-Battery-Signal are passed to the Error Analysis Monitor for disposition.

3.5.1.3 Monitor Dispatcher. The Monitor Dispatcher is the third part of the main path of control within the NAS Executive. It is here that control resides when the system is in a dormant state and through which control passes when tasks, either new or interrupted, are to be scheduled. The Dispatcher has access to tables which show which subprograms are due to run at scheduled times, etc. There are also tables of resource requirements for each subprogram, as well as lists of available resources in the system. Using all this information, the Dispatcher attempts to keep the system as a whole operating at maximum efficiency.

3.5.1.4 Operational Error Analysis, Configuration Control, Startup/Startover. When an error condition is detected by the Monitor Interrupt Processor, it will, as described above, transfer control to the Error Analysis program, a set of subprograms specialized for particular kinds of error processing: machine-check analysis, I/O error analysis, reconfiguration, etc. In broad terms, these programs will identify the source of the error, record the conditions under which the error took place and attempt recovery from the error state.

In analyzing the error and arranging for recovery, there are two questions to consider: (1) can the system be reconfigured so that it retains its normal processing capacity and (2) has the data base remained essentially valid. Again in broad terms, there are four cases: if the answers are both yes, then the system can be reconfigured quickly and automatically and control passed to the Monitor Dispatcher, whence system operation continues as before.

If the data base has not been affected by the error condition but the processing power has been reduced, then the Configuration Control can arrange to drop non-essential items from the task list and to process other tasks less frequently than the normal rate. An algorithm to schedule these service cuts in a rational way is a part of the Configuration Control program.

If the data base, including the programs themselves, is damaged or suspect, then the system must be restarted whether or not the processing power is intact after reconfiguration. The startup/startover program is able to reload the system from its resident tape and reload the environmental data from the tape on which such data has been periodically recorded during operation of the system. If the new configuration is equivalent to the old one, recovery is complete. If, however, the processing capacity is lower, then an austere and slowed-down version of the system is put into operation.

3.5.1.5 Multiprocessor Executive Features. The design of the 9020 provides for two states in which the machine may run as regards "privilege": the supervisor and the problem states. By making the supervisor state be the province of the Multiprocessor Monitor alone and requiring that applications subprograms operate in the problem state, the designers of the NAS system have provided the Monitor with direct control over I/O operations; over the Program Status Word (PSW), Preferential Storage Base Address Register (PSBAR) and configuration control registers and over direct read/write between processors. By making sure that all new PSW's after interrupts put the processor into the supervisor state, they ensure that the Monitor is given complete control of the interrupt system.

The Supervisor Call (SVC) instruction causes a special interrupt which carries with it a code from the instance of the instruction; this provides the problem program with a means for transferring data to the Interrupt Processor. By following pre-established conventions, the application subprogram can use this means to request storage facilities and I/O operations of the Monitor, etc.

System deadlock is a condition in which two or more processes are permanently prevented from operating because each needs a resource being held by the other (s). The occurrence of this problem has been prevented in the NAS system by implementing the following rule:

Given an ordered set of resources R_i $(1 \leq i \leq N)$ and a set of processes, system deadlock will be prevented if the following holds: a process may request control of a resource R_K only if it controls no resource R_j $(K \leq j \leq N)$.

In the NAS system, the resources in question are the storage types: lines, blocks and areas in that order. In practice, if a program--either application subprogram or monitor--needs storage of a type, it must suitably release all units of that type and higher. That is, to get an area requires that the program UNLOCK all its present areas, to get a block requires also that the program RELEAS all its blocks and to get a line requires that the program CANCEL its lines as well. This implies that the program must make its initial requests in the order: lines, block, areas.

The 9020 design provides two other features which are used in the multiprocessor design. Storage protection is provided by a storage key associated with each block of 2048 bytes. This key is compared with the key in the Program Status Word when reading or writing operations are performed; a mismatch causes an interrupt and consequent return to the Monitor Interrupt Processor.

Finally, an extensive diagnostic facility has been provided, such that the Monitor can execute a DIAGNOSE instruction and cause control to pass to a selected routine stored in Read-only Storage (ROS). These routines, or "kernals", then cause selected registers to be stored, others to be reset, etc., thereby providing a quick, automatic reaction to a fault situation.

3.5.2 ARTS III Multiprocessor Monitor. The ARTS III system will operate under a number of versions of its monitor, or executive, system, each reflecting an increase in the capability of the underlying computer complex. The system is currently in the design and initial implementation stage of the so-called Multi-IOP Executive, a relatively limited program planned to operate in a complex of two IOP's plus peripherals. The final product--the one to be discussed here--is to be the Fail Safe/Soft Executive which will operate in a complex of central Processor Modules (CPM) and Input/Output Processors (IOP), both with multiprocessing expansion hardware. Included also would be the Reconfiguration and Fault Detection Unit (RFDU) and, presumably, the Centralized Memory Access Module (CMA).

Since this system has not yet been written, one can only describe the objectives of the design and the proposals for implementation. It might be remarked here that the designers of this system have not attempted to develop a general-purpose operating system which could be used in a wide range of environments, nor have they taken a general-purpose OS and adapted and modified it for the ATC environment. Both of these approaches could lead to valid systems and in some circunstances might lead to superior systems.

In this case, however, the system to be developed will be specialized in a number of ways to the ATC milieu: this is the point in the design process where judicious trade-offs must be made. There are two factors to balance; one is the need to provide in the design enough flexibility and adaptability to permit reasonable changes in and additions to the requirements placed on the system. It is clear that no system designed to meet today's problems can exactly meet tomorrow's--the system designer keeps this in mind.

On the other hand, generality is expensive. A system that is ready for anything must, before reacting, find out what is happening and then must choose from its wide range of possible reactions, the one which fits. The tests, the logic for choosing and the inventory of reactions are all overhead, paid for by extra memory requirements, slowed reaction time and reduced capacity.

The ARTS-III system design is predicated on a cooperation between operational task programs and the executive and on a leanness in the executive design that are expected to keep overhead relatively very low. The implication is that this is a dedicated system--dedicated in this case to air traffic control--whose individual tasks will be compatible with the system and will be compatibly coded.

3.5.2.1 Operational Task Organization. Considerable thought has been given to the organization of the Operational Tasks within the system, for it is on how well these tasks behave that the success of the system as a whole depends.

It is assumed that the system operation is cyclic, being tied to the basic scan rate of the radar. It is further assumed this scan can be divided into semi-independent "cycles", each of which is made up of a number of tasks. The tasks in each cycle are interconnected by a predecessor and successor relationship, forming a lattice structure. The analogy between this structure and the familiar PERT chart is quite apt.

In each cycle, certain tasks have no predecessor tasks, or 'pretasks' as they are called; these are the first ones to be operated in each cycle. Other tasks have no successors or 'post-tasks', which means that their completion signals the end of the cycle. In between, the selection of a task to be run is dependent on (1) its being a post-task of some completed task--so that it gets put into the queue--and (2) all of its pretasks having been completed--so that it is eligible for operation.

There is a special class of tasks called "pop-up" tasks which are initiated asynchronously from the regular tasks. These tasks are those whose operation is so infrequent is to make scheduling of them inefficient, or whose operation is required at an unpredictable rate. Examples might be hourly updates of weather conditions, operator messages concerning changes to peripheral units or the requirement to handle fault indications.

A cycle may be ended in one of two ways: by request of a task or by being timed out by the executive. In either case, the critical tasks must be accomplished before the change of cycle.

Communication between tasks and executive is carried out via an Executive Service Request (ESR), a macro which has as its first word an instruction which causes the instruction, presumably a Return Jump, stored at a special location in memory to be executed. The next two words of the macro then tell what service is being requested and pass parameter values along. Various ESR macros will be defined, including requests for I/O services, error handling services and normal exit.

A very important facet of the ARTS design is the way in which input/output interrupts are handled. In this system, the operational tasks, themselves, will handle the interrupts. This will have the very worthwhile effect of reducing the overhead usually associated with this activity--an overhead that has severely limited the capability of many systems in the past. But the price for this gain is a loss of control by the executive: a task program handling an input/output interrupt could cause loss of data simply by taking too much time. The task programs are constrained by mutual agreement to return control after no more than 250 microseconds.

3.5.2.2 <u>Interrupt Processor</u>. Except for the interrupts already mentioned--I/O and ESR--all interrupts are handled by a module of the executive called the Interrupt Processor. The portions of the module that handle the Intraprocessor Hardware Error Interrupts (memory address parity, etc.) and the Program Fault (illegal op code) Interrupt reside in the NDRO memory of each processor. The remaining portions reside in main memory and consist of reentrant code so that a single routine for each type of interrupt may be accessed by all processors simultaneously.

3.5.2.3 Scheduler. The Scheduler works with the two classes of tasks mentioned above: the planned and the pop-up tasks. The planned tasks are assigned for operation according to elaborate relationships to the other planned tasks, while the pop-up tasks appear asynchronously. Some tasks, as a matter of fact, can be classed as both, having a planned task entry and pop-up task entry; this allows them to be regularly scheduled yet be available for quick reaction.

The planned tasks are listed in a table, called the Post Task Pointer (PTP) table which is continually scanned by the schedular; if a task is listed in the table and an eligible processor is available, the task is initiated. Tasks are put into the table whenever all of their pre-tasks are completed. That is, each time a task is completed, a check is made of all of its post-tasks; if all of the other pre-tasks of one of these have been completed, the task is eligible and is put into the PTP table.

Selection of post-tasks and checking of pre-tasks is done via the Lattice Description Table (LDT). This is a doubly-threaded list of the tasks, pre-tasks and post-tasks. That is, for each task there is an entry and part of this entry is a list of pointers back to the table entrys for the pre-tasks and part is a list of pointers forward to the entrys for the post-tasks. Included in each entry are indications of the status of the task (not eligible, eligible, complete) and of the pre- and post-tasks (not complete, complete), which are updated by the scheduler each time a task is completed.

Whenever the tasks of a lattice are all completed, the cycle is complete and the scheduler must go to the Cycle Table (CT) for the address of the LDT corresponding to the next cycle.

Meantime, the scheduler must have been checking the pop-up task list to see that none of this category is eligible for execution. Each pop-up task in the list has assigned to it an expected start time and tasks are initiated according to that time. The pop-up scan has a higher priority than the planned task scan to ensure that no task is late.

3.5.2.4 Fail-safe/Fail-soft. The fail-safe and fail-soft capabilities of the ARTS multiprocessor executive will be added onto the operating system after initial implementation. Consideration has been given to these features in the original design, so the addition should be possible with minimal extra work. Moreover, the original scheme required modular, expandable design for the purpose of changing and expanding the system as experience dictated. For this reason, the fail-safe/soft capacities should come easy.

There are two facets to the problem of operating a system in a manner which ensures continuity: detection of the error conditions and recovery from them. Errors can be detected both by the hardware and by the programs. The hardware detected errors in the ARTS system are the usual ones built into the IOP/CPM, which cause interrupts to occur and control to pass to pre-stored error routines. These error conditions are also passed to the RFDU where they are available to processors other than the one involved with the error condition.

The executive program through a module called the Device Verification Test (DVT) will attempt to detect malfunctions in the various processors and memories by executing test sequences of instructions, writing and reading test patterns of words, etc. The DVT will make sure that all devices are tested by requiring that each processor run tests periodically while monitoring the occurrence of the tests on the other processors. If a processor is late, it can be interrupted and forced to run the diagnosis.

Each of the operational tasks will be expected to check the validity of the data it handles and to pass any error indications on to the executive via special ESRs. This policy is consistent with the original design philosophy of requiring the task programs to assume responsibility for the proper operation of the system. It is worth repeating that this imposes unusually stringent requirements on the persons doing the detailed design of the operational tasks: they must thoroughly understand the operation of the executive and of the system as a whole.

Having detected a problem and determined its identity and extent, the system must react to it in a constructive way. The reaction to any particular error will quite naturally depend on the error, but the aim of the recovery process is always to return the system as soon as possible to as nearly complete capability as possible.

A part of the ARTS executive has the responsibility through the RDFU of reconfiguring the system hardware to leave any failing units off-line. Given sufficient redundant resources, this would leave a complete configuration ready for the recovery process. If less than a complete configuration is available, a reduced system will be reloaded, for a fail-soft recovery.

The recovery process can be conveniently described, as it has elsewhere, as having three parts: reloading, restoring and restarting. The reloading is straightforward unless a fail-soft load is called for. In that case, the executive must choose, according to preset criteria, which of the more critical tasks will either fit in core or will not use more processor capacity than is available.

The restoring of the data base is the next part of the recovery process. It will be facilitated in this system by having critical data redundantly stored in core--assuming that only one memory module would fail at one time, one up-to-date copy of this critical data would always be available.

Restart of the system would encompass entering each task of the system in turn at a special recovery entry point so that they could do whatever housekeeping tasks might be required before the scheduler resumes normal or fail-soft operation of the system.

3.5.2.5 Multiprocessor Executive Features. The ARTS-III multiprocessor executive has been planned from the start to be a "lean" system with much reliance being placed on the cooperative behavior of the operational tasks in the system. At the same time, all of the necessary design features have been provided for.

Firstly, there are the two classes of processes in the system: the privileged executive and the operational tasks. Although current documentation is not entirely clear on the point, certain instructions implemented on the CPM will be executable only when in the executive mode. Since one of the design objectives of the executive system was to make it operable on either the CPM or IOP, it would seem that this, and other, special facilities would have to be implemented on both machines.

In any case, it appears that the executive will have control of a number of registers which control such things as enabling/disabling of interrupts, steering of interrupts, relative address assignments, memory lockout, etc. Presumably, input/output instructions will also be privileged, although this may be only by convention: the task programs being constrained to avoid I/O instructions but not being actually prevented by hardware. This would naturally introduce the risk of a machine error whereby an instruction is mistakenly decoded as an I/O instruction and executed as such.

As to the allocation of resources, the second of the principles of multiprocessor executive design, the ARTS executive will control the allocation according to rather rigid prearrangements. A scheduled task will be assigned to a processor selected from a set assigned at load time (and this set might consist of some cases of a single unique processor). Furthermore, all tasks as well as the data base, both permanent and transient, will be assigned to permanent locations in memory. This will eliminate the executive coding and overhead associated with dynamic allocation--undoubtedly a great saving. Note that the decision to proceed this way was made despite the existence of relative addressing registers which could be used to provide the dynamic storage assignments.

The use of static allocation of storage requires that a very good prediction of requirements be made during the system design phase. While it may be relatively easy to predict average loads and to provide facilities to handle them, the provisions for peak loads will have to be either exorbitant (by providing enough storage to cover the peak needs of all tasks simultaneously) or likely to be exceeded at too high a rate. On the other hand, the cooperative use of facilities by a number of tasks would solve the dilemma by supplying the peak needs of any program (but not all simultaneously) while keeping total requirements to a minimum.

One objective of the scheduler design and the task structure is to introduce regularity into the system operation--it is hoped that the tasks will operate on a nearly fixed schedule. As a result, control will certainly be retained by the executive, but the system as a whole will assume a certain rigidity such that its success in operation will depend directly on the ability of the system designers to predict the requirements placed on the system.

The third system principle involves protection of resources from incursion or unauthorized use by tasks to which they have not been assigned. In the ARTS system, hardware has been provided to allow read and write protection to storage and instructions have been provided which allow control of access by more than one processor to common tables. Further control is provided by the earlier mentioned requirement of fixed locations for all programs and tables and the requirement that programs not modify themselves, allowing them to exist in write-protected areas. Lastly, a TEST AND SET instruction can be used to synchronize access by two programs to on data set.

Finally, the transition among programs which must be allowed for in multiprocessor design is provided in the ARTS system by the elaborate scheduling algorithm described earlier, by assigning all interrupts-except I/O-to the executive and by provision of interrupt-steering hardware which allow the executive to set logical-physical relationships among the processors for interrupt handling purposes.

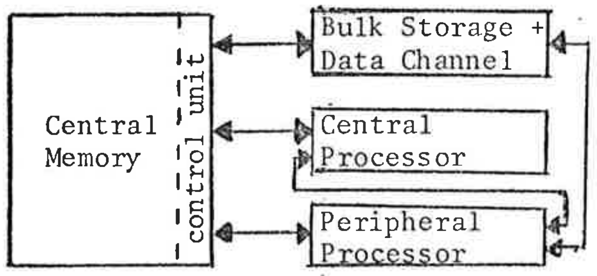
3.6 TEXAS INSTRUMENTS ADVANCED SCIENTIFIC COMPUTER (ASC)

3.6.1 General Description. This is a pipeline type uniprocessor similar to the CDC 7600 but employing LSI or MSI integrated circuits in the logic and memory areas. It is rated at between 58 and 280 million instructions per second. It is to be delivered early in 1971 to an internal Texas Instruments customer, Geophysical Services Inc., for the purpose of making seismic calculations.

The four main blocks of this computer are Central Memory, Peripheral Processor, Central Processor, and Bulk Data Storage and Channel.

The central memory associated with a single Memory Control Unit consists of up to one million 32-bit words. The memory is made up of modules of 2048 256-bit words and during each memory cycle, 200 nsecs, a 256-bit word or 8 32-bit words are provided. Actually, the raw cycle time is 160 nanoseconds which means one 32-bit word per 25 nanoseconds, and since 4 memory modules are interleaved, this means one 32-bit word every 6.25 nanoseconds.

The Memory Control Unit provides for the simultaneous operation of all memory modules and permits simultaneous operation with the Central Processor, Peripheral Processor, Bulk Storage-Data Channel; see below.



Central Memory is partitioned into 4K word-pages to facilitate dynamic relocation. This is much better than 1K word-pages which is said to cause excessive page turning, once per 10 to 100 instructions. Memory map hardware (an associative memory application) is provided at the MCU which maps the user's virtual memory page number into an actual memory location for any or all of the MCU memory ports as specified by the Peripheral Processor. References by Central Processor or the BS Data Channel to pages not allocated are treated as errors.

Each memory access port contains hardware for protecting up to three segments of virtual memory. These segments are defined by furnishing a start and finish addresses of the protected areas. They may be designated as write only, read only, or execute only. Each memory port is set up this way.

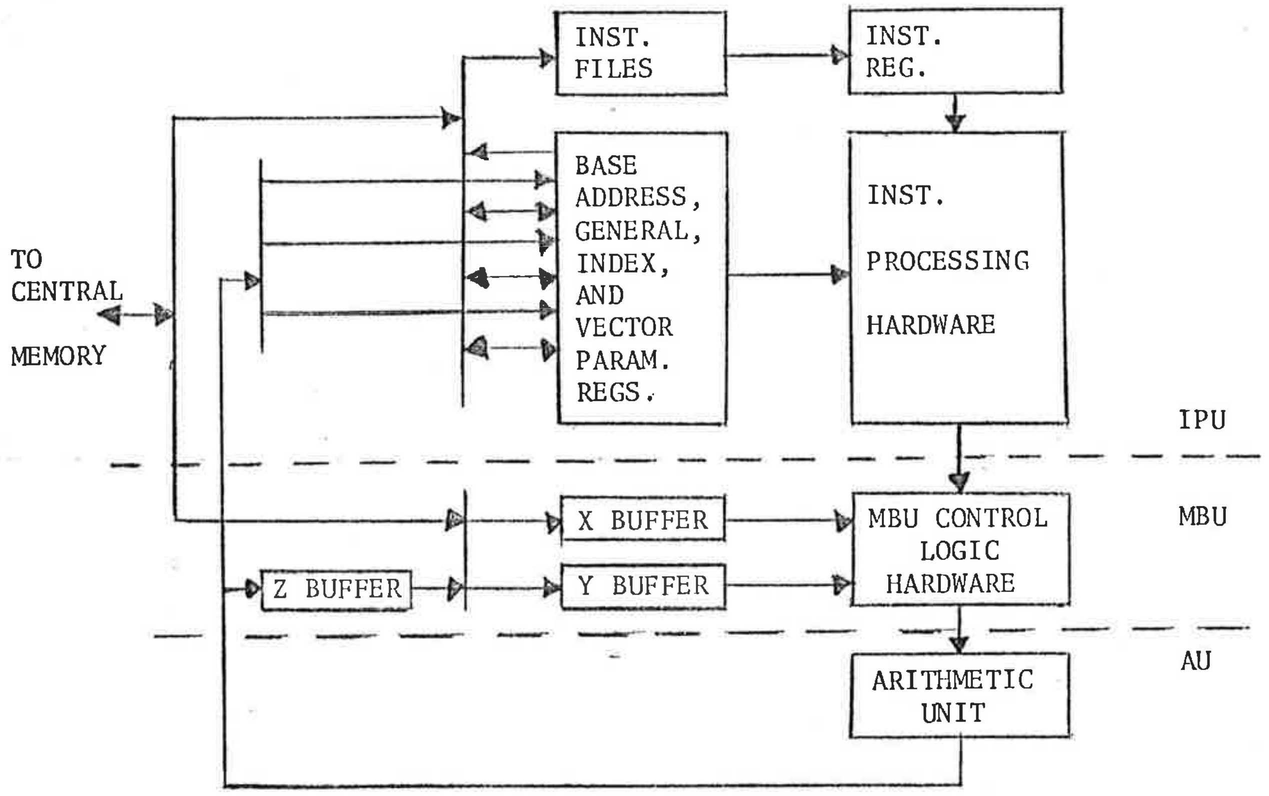
The Central Processor (CP) employs pipelining techniques for instruction processing as well as in the arithmetic unit. Five different instructions can be in various stages of completion at the same time. The advantages gained by pipeline processing of the instruction stream are dependent on the ability to handle two classes of difficulties, branches and instruction dependencies. Look-ahead hardware is provided so that the assembly language programmer can put a command at the beginning of a loop to anticipate a branch back command to return to the beginning of the loop, rather than proceeding to the next instruction after the branch instruction. This feature will anticipate correctly the next octet of instructions. On the last pass through the loop the need for the following instruction octet will not be recognized in time to use the look-ahead feature. The ASC hardware further minimizes this class of problems through the following provisions: branch-address development at the highest possible pipe level, branch condition determination at the highest possible level, drastic reduction in loop structure of programs by using automatic loops through the vector instructions. The compiler is optimized to effectively use the advantages afforded by the pipeline structure.

3.6.2 Central Processor. The Central Processor is made up of three units, the Instruction Processing Unit (IPU) which generates the effective address, the Memory Buffer Unit (MBU) which fetches and stores operands, and the Arithmetic Unit (AU) which executes the indicated operation.

The arithmetic pipeline of the AU is a 64-bit parallel operating unit which is split into 32-bit halves, similar to CDC STAR. The least-significant (left) half can be halved to do 16-bit operations, using the left most 16-bits. The same AU is used for fixed and floating-point arithmetic. Fixed-point numbers are represented as signed integers in 2's complement notation. Floating-point numbers are in sign and magnitude with a base 16 exponent represented by an excess 64 binary number.

To enable the programmer to handle two-dimensional arrays (matrices) of data as well as one-dimensional arrays (vectors), the vector processing hardware can be instructed to iterate the basic operation in a way analogous to three nested DO loops.

Below is the block diagram of the CP.



INST. FILES
INST. REG.
BASE ADDRESS, GENERAL, INDEX, AND VECTOR PARAM. REGS.
INST. PROCESSING HARDWARE
TO CENTRAL MEMORY
IPU
X BUFFER
Y BUFFER
Z BUFFER
MBU CONTROL LOGIC HARDWARE
MBU
ARITHMETIC UNIT
AU

CP BLOCK DIAGRAM

3.6.3 Peripheral Processor Unit. The PPU provides communication with I/O devices. Eight programs are time-shared in the PPU (at the bit time of 65 ns) using 8 hardware "Virtual Processors", VP_n. The Operating System for the prototype system has the complete monitor function in the PPU. These monitoring functions include assignment of system control parameters, allocation of VP units to system program tasks, allocation of the CP to user programs (steps), and the monitoring of the progress of all programs including CP programs.

In addition to the eight virtual processors, the PPU has an Arithmetic Unit (AU), a Read-Only Memory (ROM), a file of Communication Registers (CR_n) and a file of Single-Word Buffers (SWB) which provides access to the central memory. See below for a diagram showing their relationships.

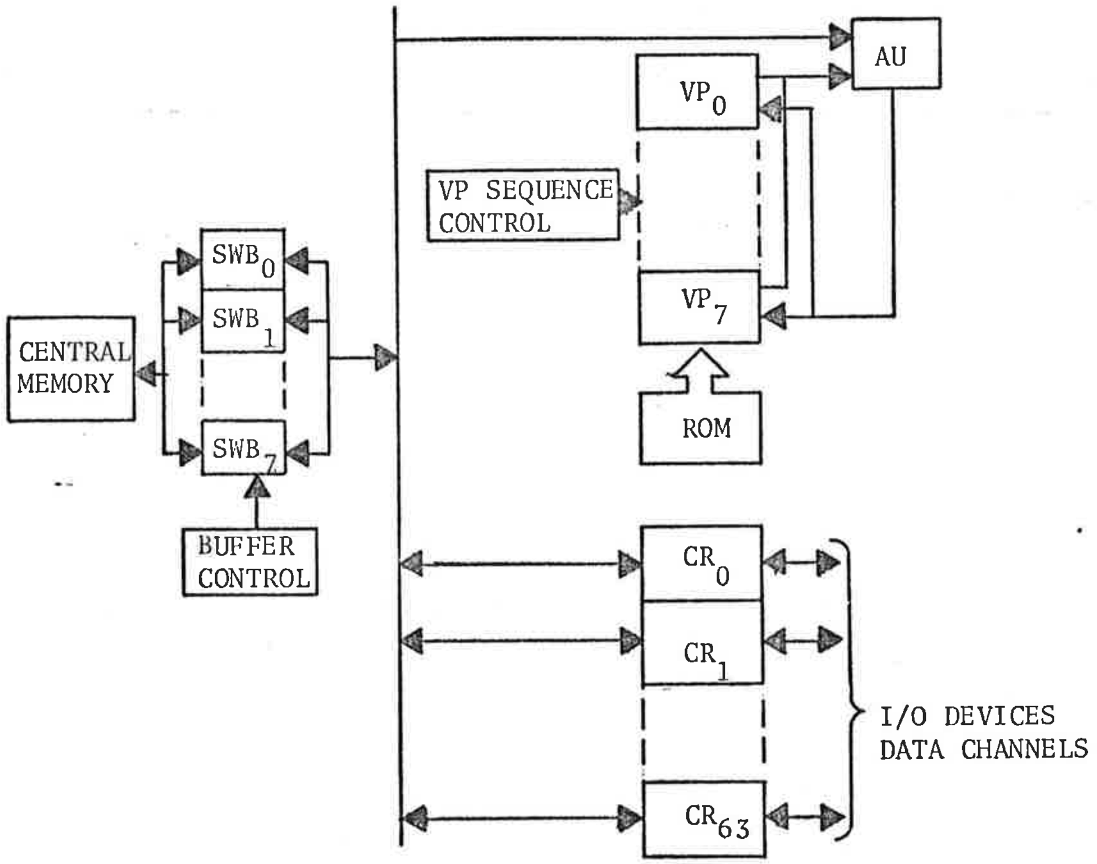


DIAGRAM OF THE PPU

3.6.3.1 Read-Only Memory. The ROM holds a pool of programs and is accessed by reference from the program counter. In the pool are primitive operating system sub-programs and at least one control program for each I/O device connected to the system. The ROM provides 32-bit instructions to the VP units and it has an access time of 20 ns. The ROM is organized into 256-word modules and is expandable up to 4096 words. Since the ROM contains the I/O device programs which include control functions as well as data transfer functions, the motion of mechanical devices can be controlled directly by the program.

3.6.3.2 Virtual Processors. The 8 VP share PP elements. Time is divided in cycles of 16 time slots (bauds). Each baud (one bit period) is 65 ns. One or more time slots can be assigned to a VP. The time slots are assigned by a master VP. The source of instructions for a VP can either be the ROM or the Central Memory.

Communication Registers. Sixty-four Communication Registers (CR's) each containing 32 bits provide control and data links to all peripheral equipment. The CR's are addressable by VP's and by the individual device to which each is connected.

3.6.4 Data Channel Unit. The Data Channel Unit (DCU) brings the high-speed mass data devices into central memory. There is a 256-bit parallel transfer of data between the DCU and central memory and a 32-bit parallel transfer between devices and the DCU. The Peripheral Processor controls the DCU through two of the communication registers.

Each data channel interfaces with a disc controller having a transfer rate of one-half million 32-bit words per second. For the prototype ASC each data channel will service two disc modules. A disc module stores 25 million 32-bit words. There is one read/write head per track. The maximum access time is 32 ms. A software optimization algorithm will be used to execute disc requests in the order of their rotational position on the disc rather than in the order in which they are executed by the user. This will increase the efficiency of disc channel usage and reduce the average access time.

3.6.5 Hardware Modularity Features

3.6.5.1 Memory. Because of the asynchronous nature of the memory control unit, high and low speed memories can be mixed in the 2^{24} word address space. With four-way interleaving and 160 us cycle time modules, a bandwidth of 200×10^6 words per second is attained. Eight-way interleaving is also possible.

3.6.5.2 Central Processor. The memory buffer unit can be replaced by one which could support two to four arithmetic units. Thus the equivalent vector rate of 65 MIPS can be increased to 130 or 260 MIPS by modular expansion. No software changes will arise.

Another expansion possibility is to use multiple central processors, i.e., employ multiprocessing. By using four central processors to access the same memory control unit the processing rate could be pushed to 1000 MIPS.

3.6.5.3 Peripheral Processors. Additional Peripheral Processors can be added; however, the present operating system cannot accommodate this change. A newer O.S. will be able to cope with it.

3.6.5.4 Mass Memory Systems. Since each Virtual Processor can sustain an I/O rate of 100,000 words per second which exceeds the rate of current mass memories, there is considerable room for expansion. There should be no difficulty in adapting new mass memories, e.g., the magnetic bubble memory, when they become developed. One interesting mass memory which TI is working on is a holographic memory which stores 10^8 bits per plate and has an access time of 1 microsecond and a data transfer rate of $.25 \times 10^6$ words per second. A complete memory will store 10^{14} plates having a maximum access time of 10 seconds and a total storage capacity of 10^{12} bits.

3.6.6 System Software

3.6.6.1 Operating System. The OS provides control, scheduling and resource allocation for the orderly flow of jobs through the computer system.

As mentioned before, the OS executes within the Peripheral Processor. A job is entered through on-line terminals or through an input device. A Job Specification Language (JSL), a simple, user-oriented language is used to define jobs. A design goal of this language is the minimization of the number of specifications required for the user while permitting an escalation to more details if a user wishes. JSL is used to specify file transfer between peripheral devices and central memory; file management services, execution and the order of execution of CP programs including compilations and assemblies, and change of control within a job. The four main components of the ASC operating system are the Master Controller, Command Controller, Task Controller, and Disc I/O Controller.

The Master Controller allocates all processors including the CP. It furnishes the software interface between the CP and the PPU. It monitors all peripheral devices for attention. It allocates processors to programs (but does not do the scheduling) and it can terminate any program under execution.

The Command Controller does the scheduling. System resources such as disc space and central memory requirements are scheduled and/or reserved by the Command Controller. Gross balancing of system resources and interpretation of a user's job specification language is done by the Command Controller. The Command Controller operates interpretively on the job specification which permits modification of system flow without changing the external language. A user request, a single job specification statement, such as catalog a file, will require several actions (or commands) by the operating system. Each JSL statement contains one or more commands and each command contains one or more tasks. In addition to scheduling, the Command Controller schedules commands within a job and passes the request for the execution of the command to the Task Controller.

The Task Controller assigns central memory for code and activation records needed by tasks scheduled by the Command Controller. It manages all communication and transfer of control between tasks. It informs the Command Controller that a task has made a Disc I/O request or when a sequence of tasks has terminated. It preprocesses all Disc I/O requests and monitors the Disc I/O load.

The main function of the Disc I/O Controller is to manage the hardware channel controllers and to optimize the order of the Disc I/O requests.

3.6.6.2 Support Functions. The ASC Fortran Compiler compiles programs for execution on the computer, which are written in an extension of the IBM S/360 Fortran IV language.

The ASC Assembler is a meta-assembler, a machine independent assembly program. The input to the assembler consists of a definition of the assembly language, a definition of the object language, and a program to be assembled which is written in the defined assembly language.

The ASC File Management handles file cataloguing, file security, library management and disc management.

The ASC Link Editor is a system service program (within the OS) to link together the object module of the language processors into modules suitable for loading.

TI ADVANCED SCIENTIFIC COMPUTER

word length (bits)	32
instruction rate (MIPS)	58 to 280
failure rate MTBF hours	N/A
off memicycle time (ns)	6.25*
processor cycle time (ns)	50
add time (ns)	≈ 15ns
multiply time (ns)	
memory parity bits (number)	
main memory size	up to 1M words/Mem Control Unit
disc unit transfer rate 10^6 words/sec	.5
disc module size million words	100
max. disc access time (ms)	34
page size (words)	4096
compilers types	Fortran Meta-assembler
operating system type	TI ASC OS uses JSL Job Specification Language

* each high-speed memory module puts out one 256-bit word every 200ns and four interleaved modules then result in one 32-bit word/6.25 nsec.

Note: Although the TI ASC is a pipelined uniprocessor, it can be configured as a multiprocessor.

3.7 THE GOODYEAR ASSOCIATIVE PROCESSOR

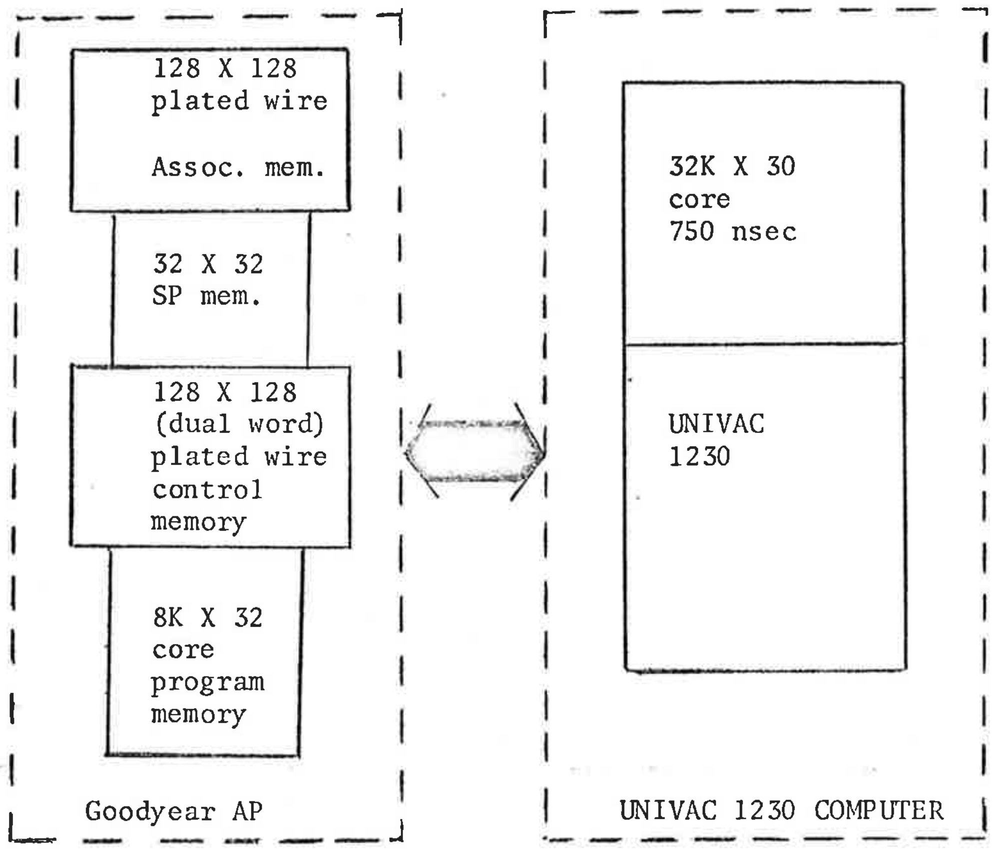
3.7.1 Introduction. The GAP, Goodyear Associative Processor, has been called other things, e.g., STARAN IV and has stood for a variety of hardware implementations for the past two years. It is hard to say what really was. It is known that in the summer of 1970, a demonstrator was exhibited in Washington, D.C. and possibly elsewhere which was composed of a 128 X 128 plated wire associative array and a 32 X 32 semiconductor control memory, which was fed by a paper tape program. This version made a big impression on many on-lookers although in many respects, it was and is a primitive machine, but it was the culmination of 10 years of work by a group of people at Goodyear Aerospace Corporation. They had seen the limitations in the conventional digital processor: that it operated sequentially, requiring that the memory be addressed one location at a time, and that only one operation could be performed on one data stream at any instant of time.

The present GAP was arrived at after the design and fabrication of an Associative Memory for the Rome Air Development Center; actually the Goodyear Associative Processor was developed for RADC also.

The clinching argument which favors associative processors is that they expend constant processing time as the number of repetitive operations is increased (to a finite limit) whereas the conventional processor follows a linear slope over increasing time, actually crossing the constant level line of the associative processor after only a few repetitive operations.

3.7.2 The Goodyear Associative Processor In The Knoxville Experiment. The description of Goodyear Associative Processor and Univac 1230 Computer Configuration and the problems to be solved as outlined in the Univac Technical Proposal of October 1970 have been altered.

An initial (60 day) contract was signed on December 18, 1970 obligating Goodyear to begin to supply the Associative Processor which is to interfaced with the UNIVAC 1230 and to supply the software to do the tracking and conflict detection routines. In four months, the hardware is to be shipped to Knoxville and two months of installation and testing will be required. As a result of the close schedule, conflict detection algorithms suggested by other companies will not be included.


128 X 128 plated wire Assoc. mem.
32 X 32 SP mem.
128 X 128 (dual word) plated wire control memory
8K X 32 core program memory
Goodyear AP
32K X 30 core 750 nsec
UNIVAC 1230
UNIVAC 1230 COMPUTER

The above is a block diagram of the Knoxville configuration.

The upper block in the Goodyear Associative Processor is 128 X 128 plated wire associative memory which was part of the demonstration processor that was exhibited during the summer of 1970. The next block is a 32 X 32 semiconductor memory, formerly used as the control memory in the summer demonstration and which was fed by a paper tape program for operating the demonstrator AP, now used as a scratch pad memory for holding index registers and data. The 32 X 32 memory is actually the upper section of the AP control memory. The 8K X 32 core memory (900 nanosecond memory cycle time) makes up the third major block in the AP. It contains the programs for tracking and conflict detection.

The way the associative processor and UNIVAC 1230 serial processor work to handle air traffic is described below. Target data returns from the Data Acquisition System are stored in data buffer storage areas in the 1230 Main Memory. As many as 60 targets (3 UNIVAC words per target) can be stored in the main memory during a 4 second scan. The target data is accumulated in sectors and is entered into the AP 8K X 32 core memory; it takes 18 microseconds per target. At the conclusion of this data transfer, the AP is allowed to execute the tracking algorithms, correlation, smoothing, and prediction. Meanwhile the target data for the next sector is being accumulated in a buffer storage area in the UNIVAC Main Memory.

The Tracking Program in the AP 8K X 32 main memory is composed of associative processor microcoded instructions. At a later date some of the most used strings of micro-code instructions, e.g., GREATER THAN will be implemented in the AP control memory.

When the tracking program is concluded the UNIVAC 1230 is signalled, and the results are transferred back to the 1230 main memory "result" buffer area. The 1230 CPU which has been monitoring the AP, decides whether there is time left (before the next sector requires "tracking") to perform conflict detection. This and a few other executive decisions are really the only CPU functions the 1230 is to perform. For the most part, it functions as an IOP, it inputs target data from the DAS and it outputs to the displays (for which it does do coordinate conversions). The AP on the other hand is permitted to do all the tracking and conflict detection.

Later on it is expected that the AP will be called on to do display output filtering, radar reinforced beacon tracking, radar only tracking.

3.7.3 Associative Array Operation. The Goodyear 128 X 128 associative array is arranged into six fields and 9 individual tag bits. Below is a chart of some of the associative processor instructions.

Instructions	Execution time (microseconds)
Exact match	.1(n + 4)
Less than	.1(n + 4)
Greater than	.1(n + 4)
Maximum value	.2(n + 2)
Minimum value	.2(n + 2)
Next lower	.3(n + 3)
Next higher	.3(n + 3)
Add common	.8(n + 1)
Add fields	1.1(n + 1) + .1
Multiply common	.8b(a + 3.2)
Multiply fields	1.1b(a + 2.7)

n,a,b = number of bits in pertinent data field

It is to be noted that the execution time is independent of the number of words but only on the number of bits in the fields involved. Fields A, B, D, and E are 7 bits each. Field C is 9 bits and field F is 13 bits. Tag bits are designated for Between Limits, Next Higher, Next Lower, Maximum, Exact Match, Less Than, Less Than or Equal, Greater Than, and Greater Than or Equal.

To illustrate the operation, suppose the instruction was given to tag all words in field A that are greater than or equal to a certain supplied argument (and possibly a mask of it). 1.4 microseconds after the instruction, mask and argument was given the tags for "greater than or equal to" for all the words in field A which qualified would be "set". Subsequent read instructions could interrogate each of these tagged words individually. A comparable action in a serial processor (or conventional computer) would take milliseconds. To describe another instruction, consider Add Fields. Suppose the instruction was issued to add field B to field C and store the results in field C. It would take 8.9 microseconds to obtain the sums in all 128 words. It would take at least a half millisecond in a serial processor.

It is of course the ability to conduct search operations (field-limited in the Goodyear AP, but not in others) over all the words (128 in the GAP) which make the associative processor an intriguing prospect in the future design of computers. By virtue of the parallel organization, speeds comparable to that of much larger serial machines can be obtained. In addition, their reliability is greater due largely to the fewer numbers of parts compared to these large machines.

Following is a classification of processors.

Class	Type	Examples
Single instruction stream, single data stream	Sequential (pipeline) processor	CDC STAR, 7600 TI ASC IBM 360/195
Single instruction stream, multiple data stream	Array processes	Solomon machine ILLIAC IV, PEPE Associative processor
Multiple instruction stream, multiple data stream	Multi-processor	Holland machine IBM 9020, ARTS III

3.7.4 Associative Array Costs. The present cost for bipolar associative memory chips (8 bits per chip, 16 bits per chip sizes) runs between \$1 to \$2 per bit. These chips have cycle times of 35 nsecs and the power dissipation is 30 milliwatts per bit. Thus, fast large systems having cycle times of less than 70 nanoseconds can be contemplated.

The present cost for 128-bit MOS associative memory chips with cycle times of 100 nanoseconds runs about \$.50 per bit--3 mi-liwatts per bit. This allows large system search times of about 300 nanoseconds. The plated-wire associative memory cost runs somewhere between the MOS and bipolar associative memory costs but the plated-wire AM configuration is bit-serial, that is, it takes 100 nanoseconds per bit during search.

There is no question that the semiconductor associative memory chips which permit distributed logic and therefore faster search operations will come down in price by a large factor (10) in the next few years. Greater speed and lower power dissipation will result with the newer types of MOS memories, e.g., CMOS, Silicon-on-Sapphire, etc.

The associative array in the Knoxville experiment plus the additional circuitry (but exclusive of 1230 computer) costs in excess of \$2M.

3.8 TWO NEW ASSOCIATIVE PROCESSORS

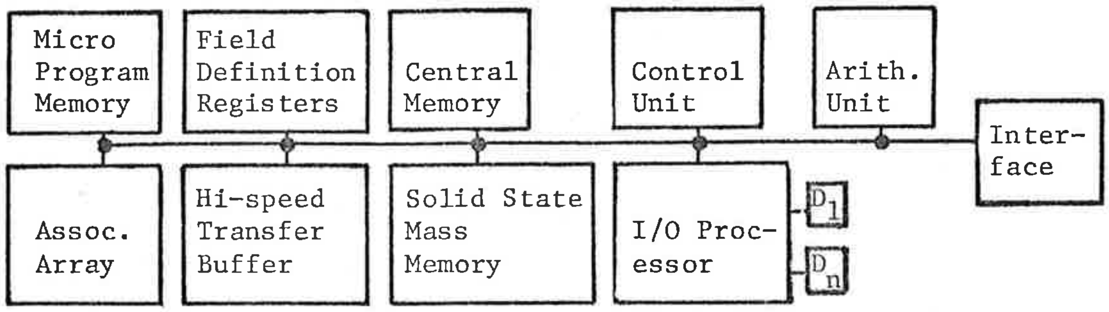
3.8.1 The New Air Force Associative Processor. The Air Force Rome Rome Air Development Center has been funding associative processor development work since 1961. They have funded the development of cryogenic, osephson device , ferro-electric, electro optic, plated-wire, silicon-on-sapphire semiconductor memory elements and the system development of cryogenic and plated-wire types of associative memories and processors.

In an AWAC study the feasibility of an associative processor to process 1600 tracks and perform the IFF, radar preprocessing, and display functions as well, was demonstrated. The additional functions required only 57% more of the associative processor's time; thus, collision avoidance and flow control could also be added at little extra expense. It is this concept of doing everything in one computer, except possibly radar and beacon data preprocessing which result in economics in transmission, that makes the associative processor so attractive. The usual MIP rate factor used to rate serial processors is not adequate in defining the ability of the AP; although "equivalent" MIP figures are used to describe the AP.

The Naval Air Systems people have a similar but shorter record of effort in the same area. They have also funded a number of element and system efforts over an even more varied scope than has the Air Force; speaking, of course, in the development of associative processors.

After experiences with Goodyear in the fabrication of a plated-wire bit-serial associative memory and a similarly constructed associative processor, RADC decided to undertake a new procurement effort at a new associative processor. It is shortly expected that an RFP will be issued for a design study phase (one year) for the hardware and software specifications of the new associative processor. High reliable programming techniques will be part of the study. After this one year work-study program, a new RFP will be announced for the fabrication and software development. The winner of the first RFP will be excluded from the second one. Following this is a description of the new AF AP.

The functional diagram of this processor is shown below.



The suggested size of the associative array is 4096 words by 288 bits. The performance requirements are such as to allow the plated-wire technology to compete. Less than 100 nsec are permitted for bit-slice searching and less than 500 nsecs for bit serial addition. This means that a search of 288 bits will be 28.8 microseconds and an addition of two half words is 72 microseconds. Bit-slice read and write are 100 and 300 nanoseconds.

Besides being content-addressable the words in the array are capable of normal addressing. With each word in the array will be a number of control bits or tags which can be searched rapidly. These extra bits can be used by the programmer and/or executive. They can be used for array segmentation and memory protection. While this array has many commendable features, it lacks the distributive logic search capability with its speed advantages.

In addition to the argument and mark registers is the field definition registers which define fields in multiple field operations.

The microprogram memory is the store which holds the sequences of microinstructions which make the macro or machine language instructions.

The high-speed buffer memory is an array of 4096 by 72 bits with capability of 4096 long bit slice read or write in less than 50 nsecs. An ECL semiconductor memory seems to be the solution to this memory.

The solid-state mass memory will probably be fabricated MOS memory chips which currently afford 1024 bits per chip at a 1 microsecond cycle time. Unfortunately, at this stage of technology, they are not beam-leaded, but it is expected that they soon will be as well as an increase in bits per chip to 2048.

For the purpose of control of the AP, a CPU is used. An electrically alterable microprogram memory, a central program memory, an arithmetic unit, logic and other registers are included in the central processor unit.

An IOP, Input-Output Processor, is used to handle peripheral devices and communication interface.

Other features are that the machine should have multiple address and operation breakpoints.

3.8.2 Honeywell Associative Processor. The preferred model of Honeywell for an associative processor for the ATC problem is a combination of a distributive logic associative memory, e.g., the 256 X 64 NASA Associative Memory, and a bit slice random access memory which operates like the Goodyear Associative Processor. With this implementation the fast search capability of the associative memory, 500 nsecs, is coupled with the fast arithmetic capability of the bit slice associative processors which is capable of performing addition on a bit slice to bit slice basis in 400 nsecs. A host computer is required with the associative processor.

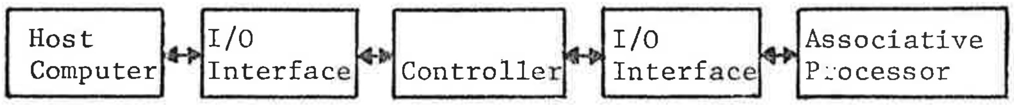
Two main types of associative processors exist. They are the distributed-logic type and bit-slice-logic type. The distributed-logic type has logic at every bit position while the bit-slice logic type has logic only on a per-word basis. The following table shows the differences of these two types.

Operations	Distributed Logic	Bit Slice
Equality Search	Parallel-By-Bit	Serial-By-Bit
Other Search Operations	Serial-By-Bit	Serial-By-Bit
Arithmetic Operations	Serial-By-Bit	Serial-By-Bit
Word Write	Parallel-By-Bit	Serial-By-Bit
Word Read	Parallel-By-Bit	Serial-By-Bit

*This table appears in the report "An Associative Processor for Air Traffic Control" by Kenneth Thurber, Honeywell Systems and Research Center, St. Paul, Minnesota.

The distributed-logic associative processor has distinct speed advantages over the bit-slice AP for the equality search and read/write operations since operations are performed simultaneously over all bits of every word. The bit-slice AP will usually have the speed advantage for "bit-slice" read and write operations and for arithmetic operations.

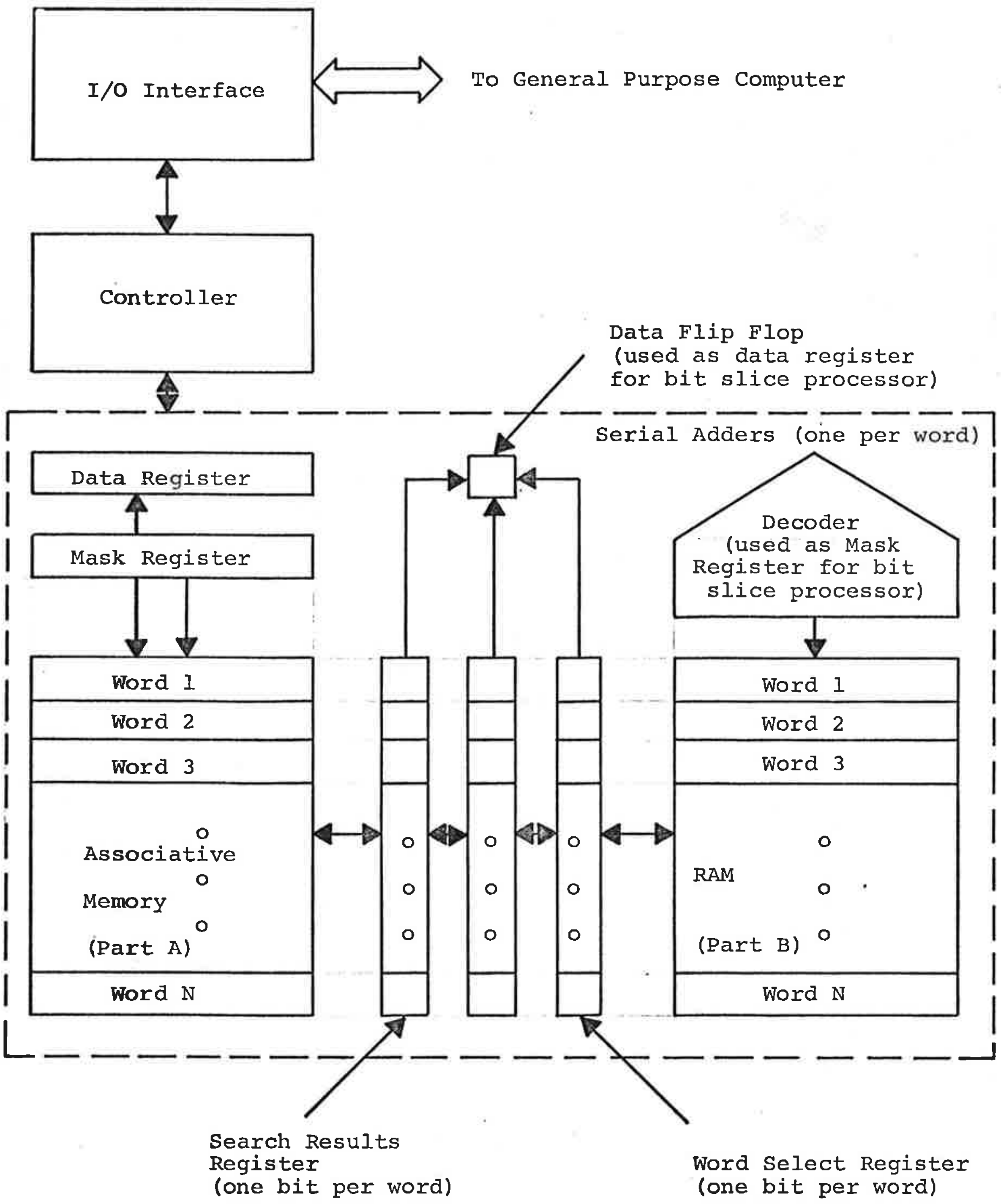
Honeywell has combined the best features to form a "hybrid" or "combination" AP which will outperform either of the two discussed types for a mixed problem such as is encountered in air traffic control. Another problem is also solved by the Honeywell AP approach, i.e., host computer-associative processor incompatibility. The usual arrangement for the Host Computer-Associative Processor system is shown below.



An interface exists between the AP Controller and the host computer which is usually a general purpose sequential computer and another exists between the controller and the AP. The interface units are necessary since the data is handled differently in the neighboring units. For example, the bit-slice AP handles the bits in a serial fashion whereas the host computer I/O is usually handled in word parallel.

The 3 main sections of the Honeywell Associative Processor are the input-output interface, a microprogrammed controller and the dual logic type processing element.

The interface contains registers, voltage level shifters, routing logic and word formatters. The microprogrammed controller receives instructions from the host computer and instructs the processing element to execute them. The controller has both a read-only memory and a random access memory. The ROM holds microinstructions, e.g., less-than-search and some fixed data. The RAM holds sequences of microinstructions specially written for host computer macroinstructions. This arrangement allows flexibility as the host/AP interacting software can be written independently, i.e., without constraints.


I/O Interface
To General Purpose Computer
Controller
Data Flip Flop (used as data register for bit slice processor)
Serial Adders (one per word)
Data Register
Mask Register
Decoder (used as Mask Register for bit slice processor)
Word 1
Word 2
Word 3
Associative Memory (Part A)
Word N
Word 1
Word 2
Word 3
RAM (Part B)
Word N
Search Results Register (one bit per word)
Word Select Register (one bit per word)

HONEYWELL ASSOCIATIVE PROCESSOR

The associative processing element is made up of a distributive-logic associative memory, an end-driven random access memory, and a centrally shared core of adders and results registers. The AM part has the advantage of reading writing and searching words in parallel and the end-driven RAM allows for the efficient bit serial (column) operations that are typical of arithmetic instructions. Thus in halves of AP words, bit-slice and distributed logic operations take place. The RAM section is made of conventional RAM chips. The AM section is made up of conventional CAM chips. The central core is made up of standard logic integrated circuits.

The following table gives the speeds for several fundamental associative processor instructions for the distributed, bit-slice and combination types of associative processors. The combination (Honeywell Associative Processor) column gives the speeds for each half of the word, i.e., Part A speeds represent the speeds of the AM, Part B speeds represent the speeds of the RAM.

The final table summarizes the speed comparisons of the three types of associative processors using normalized units for speeds, for 4 typical types of instructions.

	Distributed	Bit Slice	Combination	
			Part A(AM)	Part B(RAM)
Bit Slice Read	300ns	100ns	300ns	100ns
Bit Slice Write	200ns	100ns	200ns	100ns
Bit Slice Search	300ns	100ns	300ns	100ns
Parallel Maskable Equality Search	300ns	Not available (100ns/bit)	300ns	Not available (100ns/bit)
Equality Search	300ns	100ns/bit	300ns	100ns/bit
Parallel Word Read	100ns/word	Not available (100ns/bit/word)	100ns/ word	Not available (100ns/bit/word)
Parallel Word Write	100ns	Not available (100ns/bit/word)	100ns	Not available 100ns/bit/word)
Add Bit Slice to Bit Slice and Store in a Bit Slice	900ns	400ns	900ns	400ns
Multiple Match Resolve	100ns	100ns	100ns	100ns

COMPARATIVE SPEEDS FOR THREE TYPES OF ASSOCIATIVE PROCESSORS

	Distributed Logic	Bit Slice	Combination
Equality Searches	10 units/second	1 unit/second	7 units/second
I/O	20 units/second	1 unit/second	10 units/second
Arithmetic	1 unit/second	3 units/second	2-3 units/second
Bit Slice Processing	1 unit/second	3 units/second	2-3 units/second

SUMMARY OF SPEED COMPARISONS FOR THREE TYPES OF ASSOCIATIVE PROCESSORS

For the ATC application, Honeywell advocates the use of the combination type associative processor with 512 words of 104 bits of distributed logic associative memory and 128 bits of bit slice type memory. One track would be assigned to each 232 bit word. The controller would need 2000 words of read/write memory, and 500 words of read only memory. The host computer would be the ARTS III computer.

The three main subfunctions of tracking, namely, correlation, correction, and prediction would be performed by this AP.

Correlation includes getting the targets from the host, performing correlation of range and azimuth of targets against tracks stored in the associative processor, tagging the target reports for prediction and/or correction and storing the target report in the track file. All these subfunctions are ideally suited for the AP. A queue of targets accumulated in a subsector are to be sequentially processed for correlation.

For correction, the four standard correction equations are solved. These are

(1) $(X_c)_n = (X_p)_n + \alpha(X_R - X_p)_n$

(2) $(Y_c)_n = (Y_p)_n + \alpha(Y_R - Y_p)_n$

(3) $(\dot{X}_c)_n = (\dot{X}_c)_{n-1} + B/t(X_R - X_p)_n$

(4) $(\dot{Y}_c)_n = (\dot{Y}_c)_{n-1} + B/t(Y_R - Y_p)_n$

For all targets in the subsector the corrected positions and velocities are calculated simultaneously. It was assumed that coordinate transformations to the X,Y grid were previously performed in the host computer.

The two linear prediction equations

$$(X_p)_n = (X_c)_{n-1} + (\dot{X}_c)_{n-1}T$$

$$(Y_p)_n = (Y_c)_{n-1} + (\dot{Y}_c)_{n-1}T$$

are solved for all targets in a subsector simultaneously.

The turning track equations

$$(X_t)_n = N[V\sin(\theta \pm RN) - \dot{X}_c] + X_p$$

$$(Y_t)_n = N[V\cos(\theta \pm RN) - \dot{Y}_c] + Y_p$$

will be used for predicting the position of aircraft which are turning.

Conflict detection is proposed for the AP by doing rectangular and circular area searches and solving the law of cosines. A large rectangular area surrounding an aircraft is first searched. Next, a semicircle, inside the large rectangle is searched for other aircraft. This search is for eventual hazards. Then, a small square area is searched. Next, a small square is searched before an inner circle of danger is searched. A more efficient use of the AP is made when rectangular area searches are made before circular ones. Finally, the law of cosines relationship is solved for aircraft contained in the inner circle.

It is expected that the AP transmits to the host computer the data to be displayed. The host computer performs the filtering of data to the proper displays. However, the host computer is to be aided by the AP which uniquely codes each target to be displayed.

To fabricate an AP such as the Honeywell AP, the T.I. 4000JC 128-bit content-addressable memory chip can be used. It costs 15¢ per bit when purchased in 250 unit quantities (500 are needed for the AM considered here). Sense amplifiers (104 of them) are also needed for the AM and this along with other circuitry should push the price per bit closer to 20¢ per bit. The TMS 4000JC is a high-speed MOS LSI circuit. It has a settling time of 50ns, interrogate access time of 80 nsecs, a read access time of 60nx, and a minimum write time of 60 nsecs.

Other CAM chips, mostly bipolar, are presently being produced. Although they are faster, they are fewer bits per package and as a result are more expensive. The Intel 3104 is a 16-bit CAM which is quite fast. The search-to-match output delay is 25ns and the address-to-output delay is 25ns. No sense amplifiers are needed with the bipolar CAM's.

3.9 PEPE, PARALLEL ELEMENT PROCESSING ENSEMBLE

The PEPE computer is a global, highly parallel machine made up of an unstructured ensemble of processing elements under common control for real-time track data processing in advanced radar systems. Each processing element contains arithmetic logic, a random access data memory, and a correlator. Associative techniques are employed both on data input and result retrieval.

3.9.1 Introduction.

The technique involved in this novel machine is performance gain through the organization of the components and their functions rather than through increased performance of the components. Instead, a compromise is reached by sacrificing the ultimate in componentry for somewhat less, to increase reliability and make economic gains.

In the PEPE technique, hardware for computation and scratch pad storage is dedicated to each track, although spares may be substituted at any time and initially any element may be assigned to any track. The principal gain is that a central control actuates a group of processors simultaneously and causes similar computations to proceed in parallel. This works well for vector or matrix computations when all of the elements are working, but as soon as some are not being used, then the efficiency is lowered. The justification for the use of PEPE is the short time interval constraint when a lot of computation must be performed. The entire data base is accessible in parallel through content-addressable memories. The output as well as element inputs are effected by associative means. This parallel machine may be classified as a one-dimensional array processor, which uses associative memories and is managed by a host computer.

3.9.2 Machine Organization.

The following figure shows the structure of PEPE. The present prototype has N=16 elements.

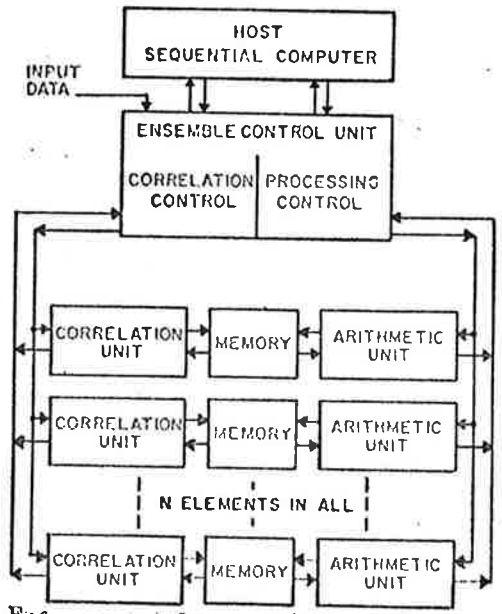


Fig. [illegible] urgar [illegible]

Figure 12. Ensemble Organization

The function of the host computer is to perform the program sequence control. It is like a program in main memory acting on a control memory. The host computer does perform other duties; those for which it is best suited, e.g., sequential portions of the programs being processed.

The ensemble control unit decodes the instructions as they are received and produces the detailed gating necessary for the ensemble. Ensemble control is divided into two parts, correlation control and processing control. For correlation control, a small memory for the internal storage of correlation programs is included. Actually, the arithmetic unit and the correlator share the use of this memory. Only a minimum of control circuitry is found in the processing element.

The arithmetic unit performs parallel floating point operations on variable length fields. The following diagram shows the ingredients of the arithmetic unit.

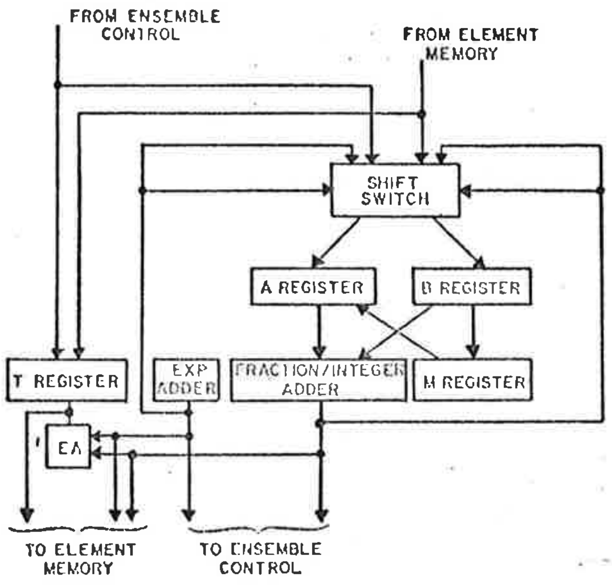


Figure 13. Clement Arithmetic Unit

The A register is the accumulator, the B is the memory operand register and M is the register used in multiplication and division. All are 32 bits long. The shift switch is capable of shifting a 32-bit input from 0 to 31 bits, right or left with 6 logical nodes of delay. The control section of the arithmetic unit has an 8-bit T register which can be loaded from memory or set as a result of tests performed on the contents of the accumulator. This register holds the activity state of the element since its contents are compared with that requested by common control. If the activity state shown by T matches the activity specification of common control then the control flip-flop EA is made active and the element is allowed to execute the instruction.

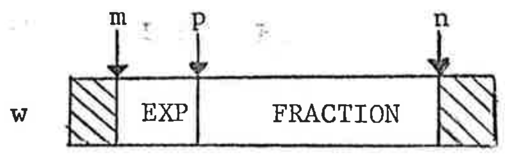
The data memory is 512 words long and has 32 bits per word. It is fabricated from the Texas Instrument Company's 256-bit MOS chip, TMS 4003JR. The memory cycle time for the data memory is less than 500 nanoseconds.

Presently, the 8 X 32 plus 8 tag-bits associative memory of the correlation unit is made up of Texas Instrument's 7400 series of gates and flip-flops but it is expected that they will soon be converted to using the TI 128 bit (16 X 8) content addressable memory, TMS 4003JC.

There is an effort underway to rebuild PEPE using MSI chips. The present average instruction time for the correlation unit is 400 nsec. This may be increased slightly.

The arithmetic unit is presently fabricated from the 7400 series of logic devices. The present average instruction time for it is 1.7 microseconds. It is expected that this unit will also be converted to MSI.

Operands are stored in the data memory as shown below.



A descriptor specifies w, the word address, m the bit position of the most significant bit, p the bit position of the boundary between exponent and fraction, actually the sign-bit position, and the bit position of the least significant bit. The exponent may be 0 to 8 bits long and the fraction 0 to 24 bits long including the sign. When read into the arithmetic unit the operand is automatically aligned and converted to an 8-bit exponent and a 24-bit fraction.

In selecting one of N outputs from N accumulators, a search of N accumulators can show agreement over a particular field for over many of the N. In order to resolve multiple response, a Lewin generator network is used.

The present host computer is the 360/65. The MSI 16 element PEPE is rated at 16 MIPS, and expected to cost 16 X $3000 exclusive of the host computer and the control unit.

3.10 CDC STAR-100

The Control Data STAR-100 is a pipeline processor capable of processing data at a 100 MIPS rate. It uses conventional integrated circuitry. The logic organization is designed to solve a wide variety of application programs. Speed is obtained by using pipelined arithmetic units and by organizing a large, slow, core memory to provide multiple consecutive words concurrently.

The core memory comes in 524K or 1,048K 64-bit words and is organized into 32 interleaved banks, each with a memory cycle time of 1.28 microseconds. Each memory bank supplies 8 64-bit words every 40 nanoseconds throughout the 1.28 microseconds. This results in a 200 million 64-bit words per second transfer between memory and processor. The STAR, String Array Processor, has a 4 trillion word virtual memory.

There is also tremendous power in the instruction repertoire. They can reorder files using 2 machine instructions. They employ three address logic. If one used all improper streaming of instructions, the 100 MIPS rate would dwindle down to 6.

A programmer addressable (different from a cache) hardware register of 256 words X 64 bits is available at memory cycle times of 40 nanoseconds.

MECL II logic is used whereas the CDC 7600 used discrete transistor circuits. For the register files, 32-bit MSI register chips are used.

The following diagram of the logical organization is not untypical of most computers; the power of STAR resides within the major components.

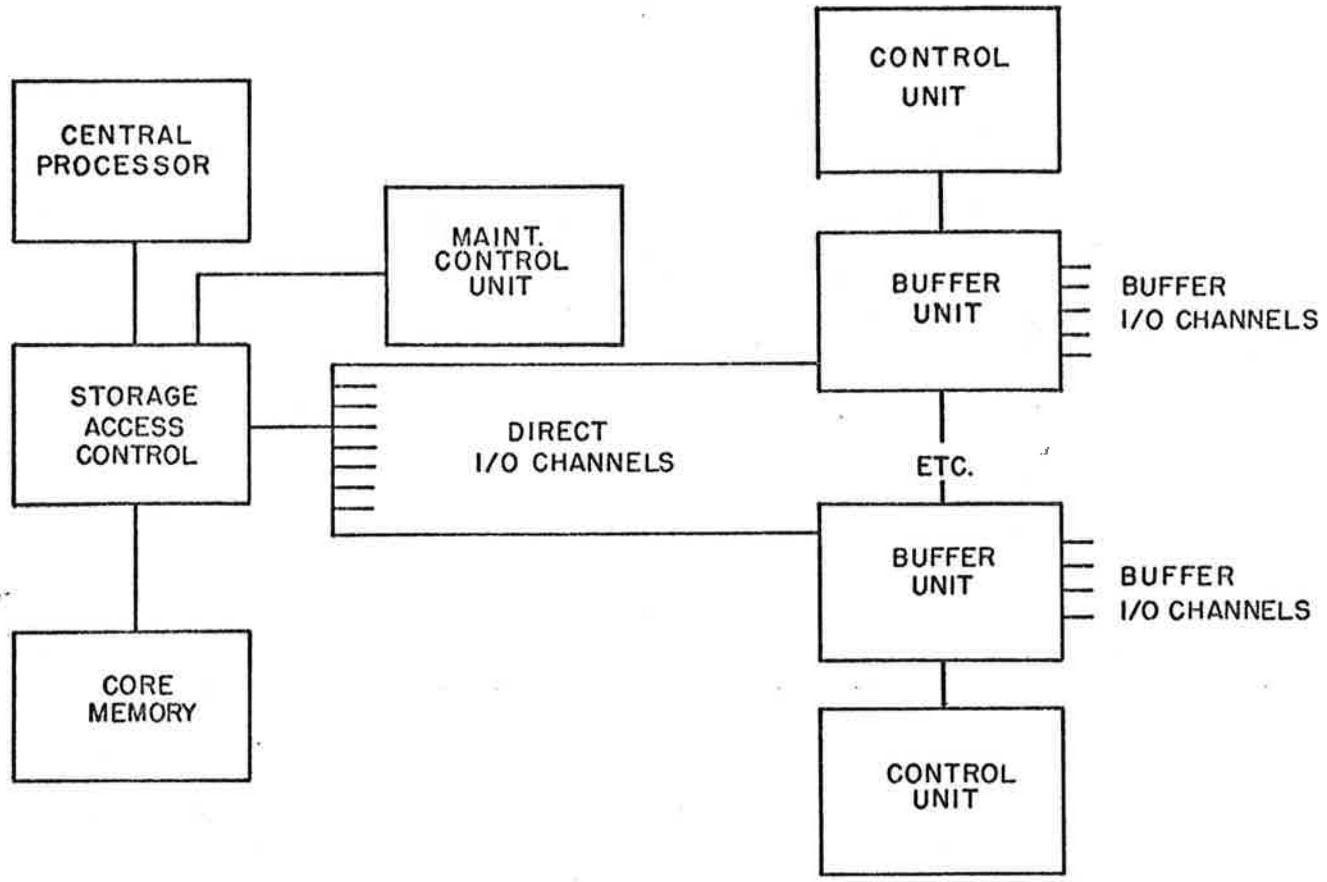


Figure 14. CDC STAR-100 Block Diagram

3.10.1 STAR Central Processor. The STAR Central Processor Unit (CPU) is made up of the Storage Access Control (SAC), Stream, Floating Point Arithmetic Units (2), and the String sections as shown in the basic central processor diagram.

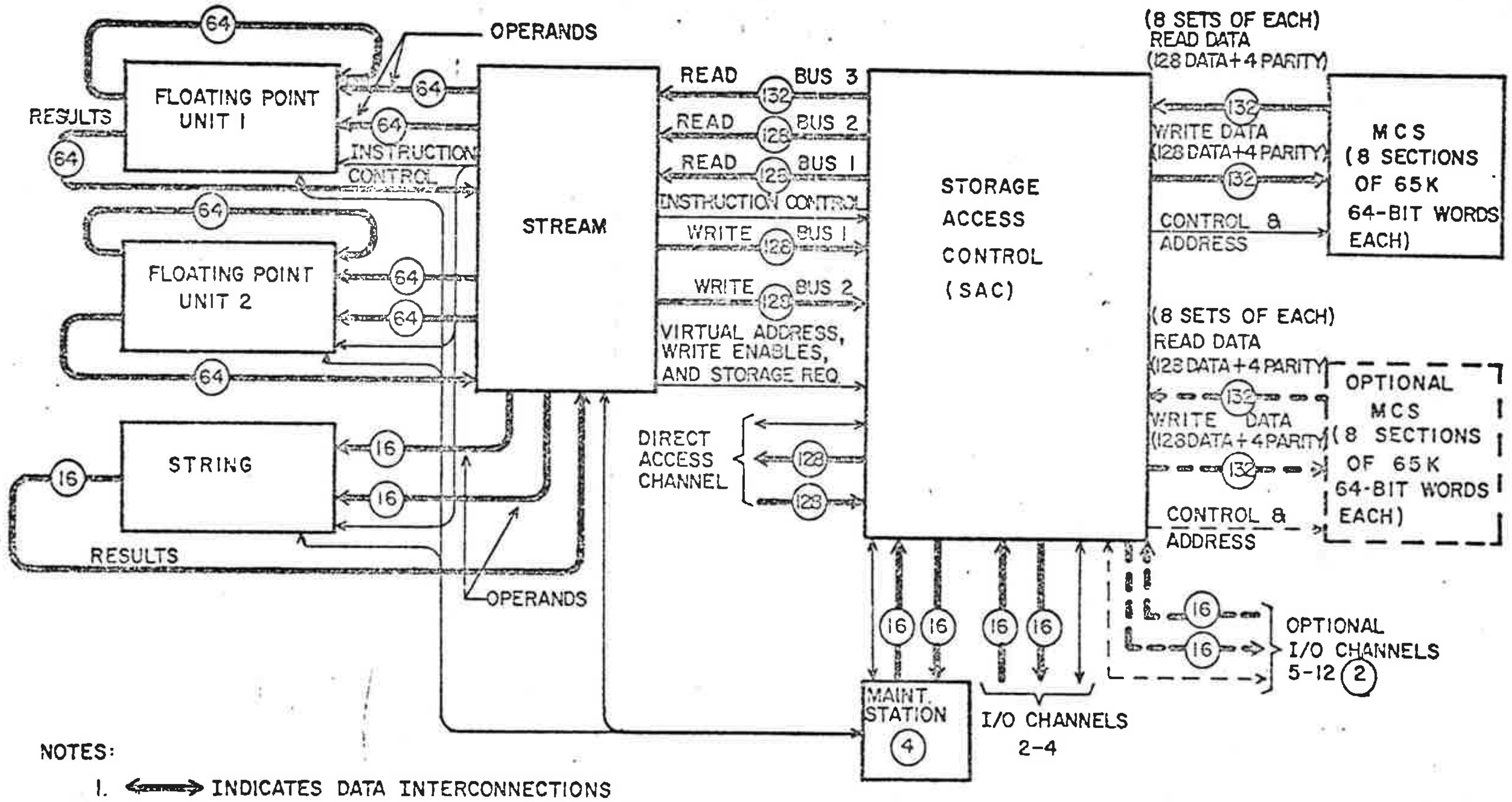
The Storage Access Control (SAC) unit controls the data transfers to and from the Magnetic Core Storage (MCS) and performs virtual address comparison and translation. It also generates parity bits for write data and checks parity on read data. SAC connects MCS to the Stream Unit, I/O Channels and the Direct Access Channel. Upon parity fault detection SAC stores the type of parity fault and the absolute address of the data causing the fault in a register which can be read by the Maintenance Station. In the case of a Stream instruction parity fault the virtual address of the fault is given by the Current Instruction Address register and means are available for obtaining the absolute address.

The types of parity faults are:

1. Access Instruction
2. Stream Instruction
3. CPU
4. Search
5. Exchange
6. I/O or Direct Access Channel

Page sizes are 65,536 and 512 for 64-bit words. The SAC unit offers storage protection in the form of locks and keys. Each page associative word contains a 12-bit lock code. Each job is assigned four 12-bit keys by monitor. If a virtual address matches the corresponding portion of the associative word, the four keys of the job are compared to the lock. There must be a match for a storage reference to be completed.

In addition to the key-lock protection feature, each of the four (job) keys is associated with a 4-bit usage lockout code. This code can lock out CPU write operations, CPU read operations, and/or CPU instruction references. When a key matches the lock but the type of usage is inhibited by the usage lockout code, an access interrupt takes place to the Monitor program which can restrict page access for a job to a particular type of reference.



NOTES:

1. ⟺ INDICATES DATA INTERCONNECTIONS
   ⟷ INDICATES CONTROL INTERCONNECTIONS
   —(64)— INDICATES NUMBER OF BITS IN DATA TRANSFERS.

(2) THE OPTIONAL I/O CHANNELS ARE PROVIDED IN SETS OF 4 CHANNELS EACH (CH. 5-8 & 9-12).

3. ALL OPTIONS ARE SHOWN IN DASHED LINES.

(4) THE MAINTENANCE STATION IS CONNECTED TO I/O CHANNEL 1. THIS CHANNEL IS A STANDARD I/O CHANNEL WITH SPECIAL CONTROL AND MONITORING CONNECTIONS.

Basic Central Processor

The Stream unit provides control in the computer. The functions it performs are:

1. Initiates all central storage reference requests for instructions and operands

2. Translates instructions and transmits control signals to the arithmetic units

3. Provides addressing for all source operands and arithmetic results

4. Buffers and positions all operands and arithmetic results between central storage and the arithmetic units.

In the Stream unit resides the 256 X 64 register storage which serves as the lower 256 words of central storage as well as for other functions. The Stream unit interfaces with Storage Access Control (SAC), Floating Point Units 1 and 2, String Unit and the Maintenance Station.

Part of the Stream unit is Instruction Control which receives all instructions from central storage or the register file (which is part of central storage) although it is located within the Stream unit. Part of Instruction Control is a 16-word, 128-bit Instruction stack into which swords (8 words) of instructions are deposited from central storage. The instructions are decoded into a sequence of micro instructions which go along with the operands and partial results through the pipeline. The entire sequence of micro instructions is repeated for many operands in the case of array instructions.

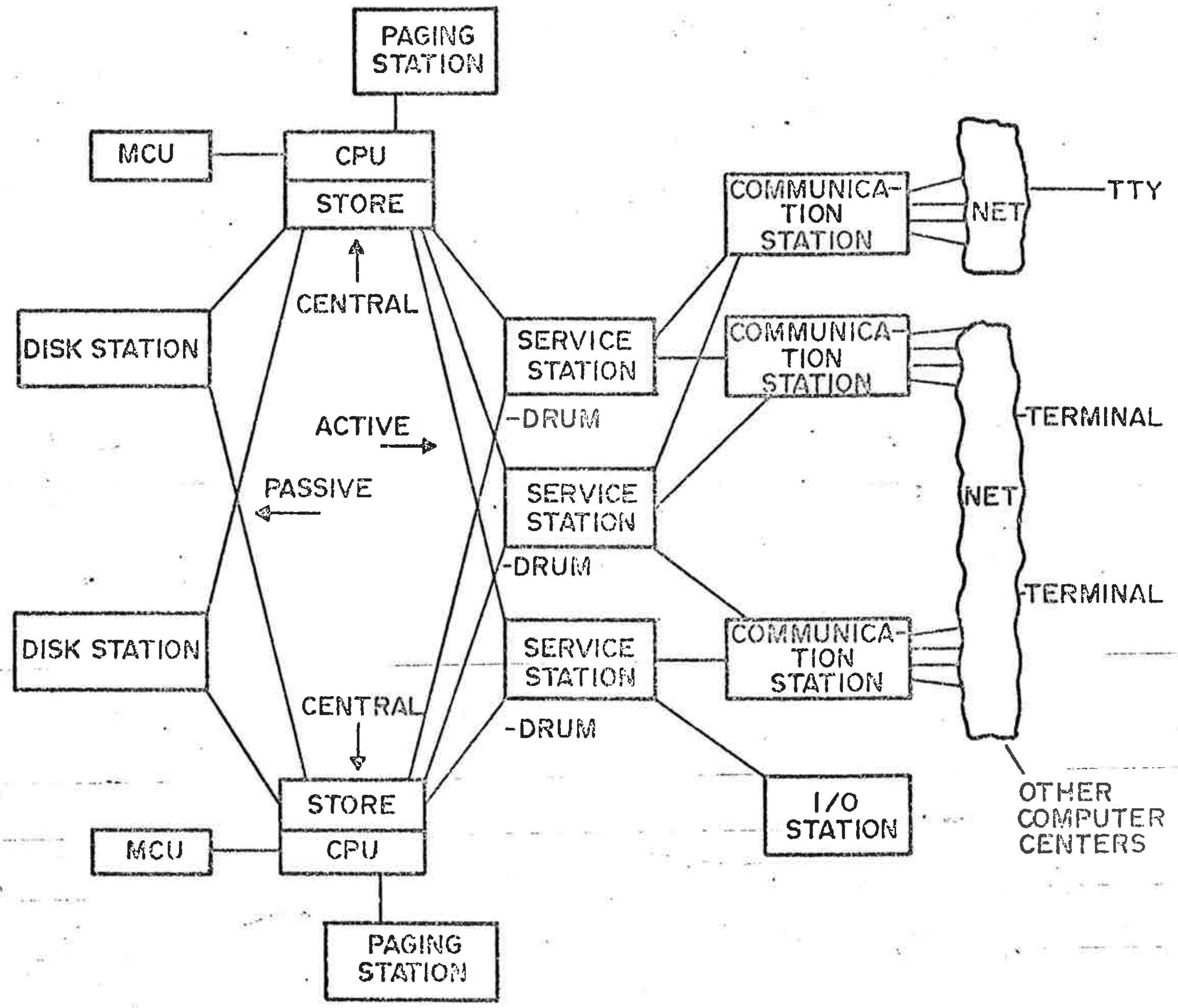
A pair of parallel independent arithmetic (pipeline) units make up the floating point arithmetic section of STAR. Pipe 1 is used for register add, register subtract, register multiply and all vector arithmetic instructions. Pipe 2 is used for register divide, register square root and all vector instructions. The speed of these units is given by the following chart.

32-Bit Results	Floating-Point Operations	64-Bit Results
100 X 10^6/sec	Add/subtract	50 X 10^6/sec
100 X 10^6/sec	Multiply	25 X 10^6/sec
50 X 10^6/sec	Divide/square root	12.5 X 10^6/sec

Another pipeline is the String Arithmetic Unit which is used to perform all of the bit logical and character string operations required in the execution of comprehensive data processing jobs. The String Unit processes all of the decimal arithmetic.

3.10.2 STAR Reliability. The STAR computer contains special hardware monitoring features that facilitate the monitoring of the system operation by the Maintenance Station. There are monitors of temperature, dewpoint and power, and also counters that count special operational activities in the CPU. The expected MTBF is 50 hours, the expected MDT is 1 hour which will be (it is hoped) reduced in the course of time to 15 minutes. By using a smaller version of STAR, EM-1B in a duplex configuration (see the diagram on the DUAL STAR SYSTEM) a fail-soft mode can be attained for an additional 20% expenditure.

3.10.3 Summary of STAR Capability and Cost. The STAR-100 is an advanced 3rd generation computer based on the use of ECL integrated circuits and system organization to give a 100 MIPS processing rate with 3 address logic and a powerful assortment of new instructions and appropriate software. The machine costs 7 million dollars. If the machine is operated (programmed) properly, it can achieve this high rate, 100 MIPS. If it is not, it can drop down to 7 MIPS. Somewhere in between lies the expected performance. The best gauge is how well it performs on a certain class of problems, e.g., air-traffic control, or the tracking of reentry vehicles. A comparative test is being conducted in the latter problem; this should furnish an indication of how good STAR-100 really is.



Dual Star System

3.11 ILLIAC IV

The ILLIAC IV is a SOLOMON-type array or parallel processor. It was to have consisted of 256 processing elements arranged in 4 arrays of 64 PE's each designed to give a 1,000 MIPS processing rate. After 5 years and 30 million dollars, including some technological mishaps, it still is not completed even though the configuration has shrunk to 1 array of 64 processing elements. There was a switch from thin-film memories to semiconductor memories for the individual PEM's, Processing Element Memories, which are dedicated 2048 X 64 data memories with memory cycle times of 200 nanoseconds.

ARPA is continuing to fund the development of ILLIAC IV which is expected to be completed by early 1972. At that time, it will be shipped to NASA-Ames Research Center, Moffett Field, California where NASA will house, operate and maintain it in return for 18% of the computing time.

From the SOLOMON Computer concept the four principal features, namely

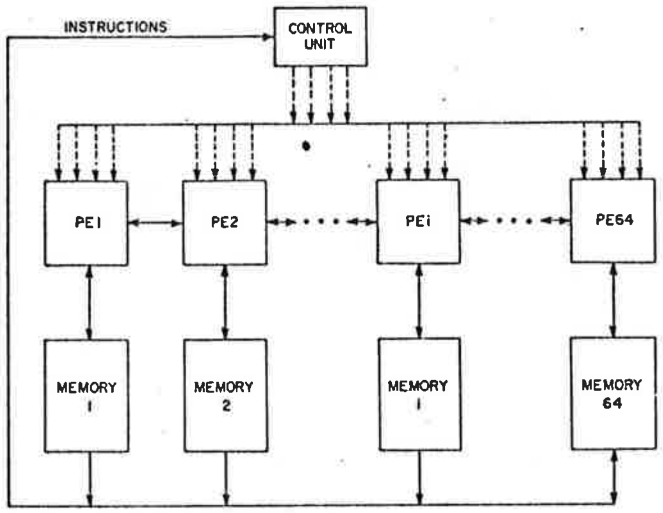
1. A single control unit, or single instruction stream sequences, a large array of arithmetic units processing many data streams

2. The central control supplies memory addresses to all units

3. Local control is achieved only by element enable or disable of the execution of common instructions

4. Nearest neighbor interchange of common instructions processing elements

were incorporated in the ILLIAC IV. The ILLIAC IV can perform an add operation (or more exactly 64 add operations) in 240 nanoseconds and multiply in 400 nanoseconds. The following block diagram shows the present configuration of the ILLIAC IV computer.

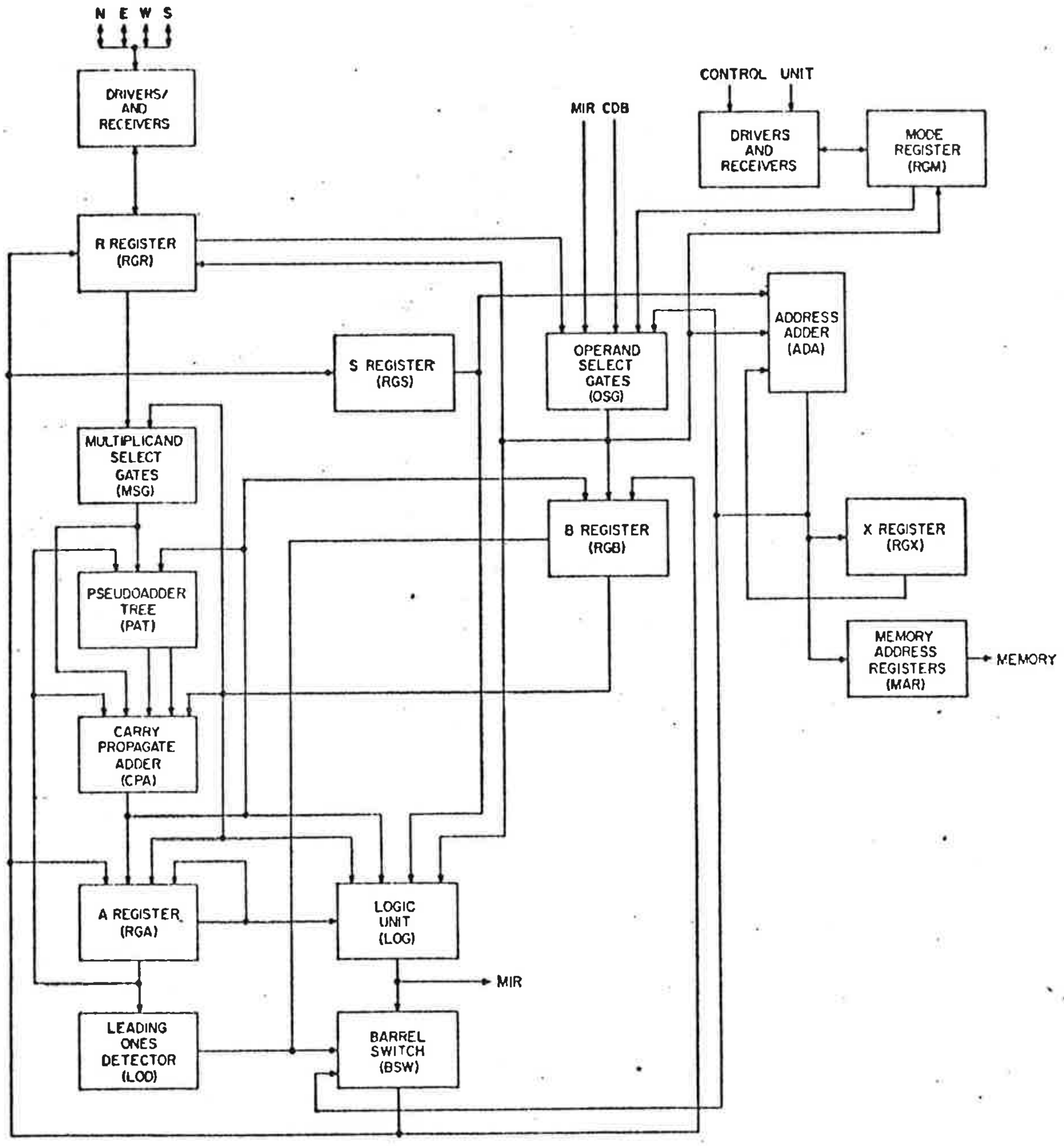


3.11.1 ILLIAC IV Processing Element. The PE is a four 64-bit register arithmetic unit. There is an A register and a B register to hold operands for arithmetic and logical operations with the A register serving as an accumulator. An S register is used to provide temporary storage and avoid repeated accesses to memory. The R register is the multiplicand register and also is used for transfer of data among the PE's. An adder/multiplier (MSG, PAT, CPA), a logic unit (LOG), and a barrel switch (BSW), which are used for arithmetic, Boolean and shifting functions, are also parts of the Processing Element. A 16-bit index register (RGX) and an adder (ADA) are used for memory address formulation. An 8-bit mode register (RGM) is used to hold the result of tests. Two bits of the mode register (RGM) control the enabling and disabling of all instructions; one of these is active only in the 32-bit precision mode and controls instruction execution on the second operand. Two other bits are set whenever an arithmetic fault, such as overflow or underflow, occurs. The fault bits of all the PE's are continuously monitored by the main Control Unit (CU) to detect a fault condition and initiate a CU trap.

Each PE has a 64-bit wide routing path to 4 of its neighbors (±1, ±8).

The PE's operate in single 64-bit word lengths, dual 32-bit word lengths, or eight 8-bit word lengths.

The following block diagram shows the ingredients of a Processing Element.



3.11.2 ILLIAC IV Control. A common control unit (CU) decodes the instructions and generates control signals for all the processing elements. By eliminating control within each PE, a saving is attained.

An index register and address adder is retained with each PE so that the final operand address for the ith processing element is determined as follows:

$$a_i = a+(b)+(c_i)$$

where a is the base address specified in the instruct, (b) is the content of the central index register, and (c_i) is the content of the local index register. This degree of freedom in operand addressing is quite effective in handling row and column operations of matrices and other multidimensional data structures.

The control unit (CU) has the following five functions:

1. To control and decode the 32-bit instruction streams.

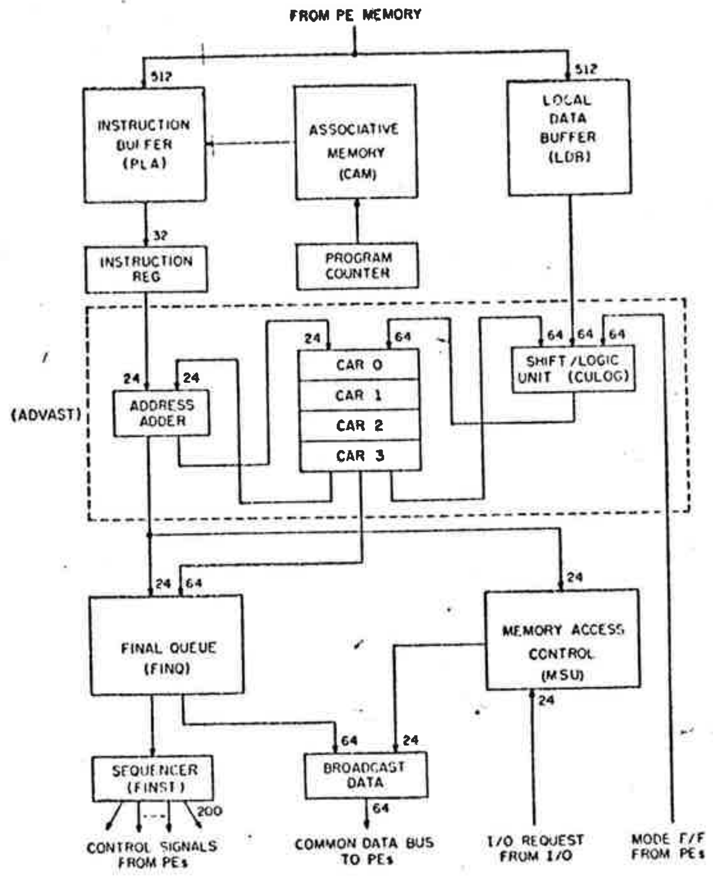
2. To generate the control pulses transmitted to the processing elements for instruction execution.

3. To generate and broadcast those components of memory addresses which are common to all processors.

4. To manipulate and broadcast data words common to the calculations of all the processors.

5. To receive and process trap signals arising from arithmetic faults in the processors, from internal I/O operations, and from the B6500, host computer.

The CU has two 64 X 64 fast access buffers, one associatively addressed which holds current and pending instructions (PLA), and the other a local data buffer (LDB). There are four 64-bit accumulator registers (CAR) which are CU internal transfer registers and hold central address indexing information and active data for logical manipulation or broadcasting. The CU arithmetic unit (CULOG) performs addition, subtraction, and logical operations. See the following diagram.

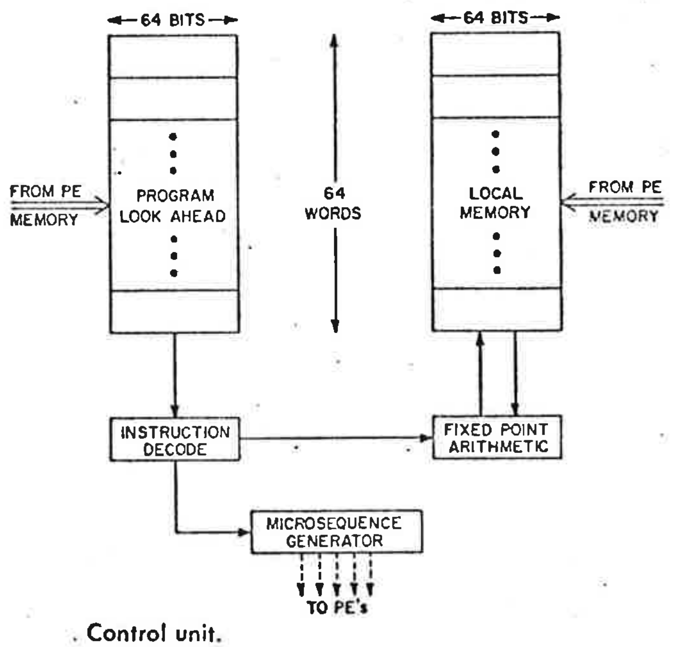


Control-unit block diagram.

All instructions are 32 bits in length and are of two types: internal CU instructions, e.g., indexing, jumps and PE instructions which decode within the CU and are transmitted to all the PE's. Instructions are entered into the PLA, sixteen at a time into 8 words of the PLA. They are individually extracted from there and are sent to the advanced instruction station (ADVAST) which decodes them and executes those local to the CU. In the case of the PE instructions, the ADVAST constructs the necessary address or data operands and places them in the Final Queue Registers (FINQ) to await transmission to the PE's. Part way through the 16 instructions, an inquiry is made to ascertain whether an additional block of 16 instructions is in the PLA. If it is not, then a control is enacted for them. The PLA holds 128 instructions which is sufficient to hold the inner loop of many programs.

The control unit communicates with the PE's one of two ways. It can broadcast a 64-bit word simultaneously to all PE's; or it can broadcast a 64-bit word with one bit going to each of 64 processors. Masking can be accomplished by enabling or disabling the PE's. The control unit can fetch words from any PE, one word at a time or in blocks of 8 contiguous 64-bit words.

The transfer of information from the control unit to the processors is through the microsequence generator; see the simplified block diagram below.



. Control unit.

A PE can be disabled on command from the control unit or as the result of a local test. Once turned off, a PE cannot turn itself back on, but must await a command from the control unit.

Interrupts are provided to handle the host computer control signals and a variety of CU faults, e.g., undefined instructions, instruction parity error, etc. Arithmetic overflow and underflow in any of the processing elements is detected and produces a trap. The interrupt hardware consists of a base interrupt address register (BIAR) which is dedicated as a pointer to array storage into which status information is transferred. At the time of an interrupt the contents of the program counter and other status information and contents of CAR 0 are stored in the block pointed to by the BIAR. CAR 0 is set to contain the block address used by BIAR so that subsequent register saving may be programmed. Interrupt returns are obtained by a special instruction which reloads the previous status word and CAR 0 and clears the interrupt.

Interrupts are enabled through a mask register. During the interrupt processing, all new interrupts are ignored, although their presence is flagged in the interrupt state word.

3.11.3 ILLIAC IV Maintainability. The modularity is inherent since the ILLIAC IV is a parallel processor; this simplifies the storage of spare parts and also the diagnosis of faults as the faults can be isolated by virtue of the parallel paths, and natural duplication. By replacing total faulty processing elements and repairing them off-line, the system MDT can be held down to 15 minutes. The time to repair a subsystem may take much longer and a queue could develop which would affect the system performance. Confidence and diagnostic routines will be necessary for insuring long up times and short down times. The diagnostic routine can be designed simply to find the faulty subsystem. Off-computer routines, either hardware or software, can be used to isolate the fault in the subsystem.

3.12 SOME ADVANCED THIRD-GENERATION COMPUTER SYSTEMS

A number of computer systems have been developed in the last few years which represent the third-generation of equipment in its more advanced state. They introduce features which will be common place in future large-scale systems, such as those described in the preceding sections.

The IBM System/370 is a development of the 360 family that uses both microprogramming of part of its control and the so-called "cache memory" concept. The UNIVAC 1110 system represents a conscious attempt to provide a multiprocessor system as a concept development rather than an add-on feature. The CDC 7600 is an attempt to squeeze from an instruction stream all the parallelism it possesses and to use this parallelism to increase the processing speed.

3.12.1 IBM System/370. The System/370 from IBM is a 360-compatible family of computers that uses some of the latest system techniques to derive high-speed performance and reliability. The largest model announced so far is model 165, a replacement for the 360/65 and 75. The 360-compatibility is in the upward direction, meaning that most 360 programs run on the 370 but not necessarily vice versa. From a programming point of view, the differences consist of a few new instructions, possibly more storage and a few minor changes and additions.

The major changes to the system was in the circuitry of the central processor unit and in the storage structure. The CPU is divided into two parts, which can operate simultaneously on different instructions. The instruction unit, controlled by logic circuits, "prefetches instructions (maintaining them in sequence), decodes instructions, calculates addresses, prefetches instruction operands, and makes estimates of the success of conditional branches". [1, p.8]. A number of instructions will be in process at one time in this unit. Interestingly, when a conditional branch is encountered, instructions along both legs of the branch are fetched to minimize delay after the condition is determined.

The execution unit is controlled by a microprogram in a control store, most of which is read-only storage (ROS) but some of which is writable from a special magnetic disk cartridge. The unit can execute one instruction per 80 nanosecond cycle. Both the instruction and execution units make use of general-purpose and floating-point registers implemented in local storage with a read/write cycle time of 80 nanoseconds.

A two-level storage structure has been provided which significantly affects the performance of the system. An 8K buffer storage (with a 16K option) with an 80 nanosecond cycle time is interposed between the processing units and the main storage, which has a 2 microsecond cycle time with four-way interleaving of double words. With this arrangement, a CPU fetch from the buffer takes 160 nanoseconds, or 2 cycles, for 8 bytes, while a buffer fetch from the processor storage takes 1.44 microseconds, or 18 cycles, for 32 bytes. This latter fetch may be repeated once each two microseconds.

Buffer storage is divided into 32 byte blocks in a 4 X 64 block array, processor storage is also divided logically, in an array of 512 X 64 blocks. The 64-block dimensions, called columns, are identified together in the two storage areas and data from a column in processor storage is moved into the same column in buffer storage only. Thus, there are 512 blocks in one competing, in a sense, for 4 blocks in the other. The storage control unit keeps enough information so that when a new block is to be stored in the buffer, it replaces the block whose time-since-last-reference is the greatest.

The net effect of the various strategems is to keep the processing units operating at a high rate while allowing the use of relatively slower main memory. The actual processing speed of the System/370 is difficult to state, since in common with other new machines, its performance is strongly dependent on the actual instruction sequence and, to some degree, on the data being processed.

3.12.2 UNIVAC 1110. Based on the UNIVAC 1106 and 1108, the 1110 is a medium to large-scale computer system especially developed as a multiprocessor. The system is composed of varying numbers of seven components: Command/Arithmetic Units (CAU), Input/Output Access Units (IOAU), System Console, System Partitioning Unit (SPU), Main Storage, Extended Storage and peripheral subsystems. The maximum configuration consists of 4 CAUs, 4 IOAUs, 262K words of main storage, 1048K words of Extended Storage--with 8 Multiple Access Interfaces (MAI), 4 System Consoles and one SPU.

The Command/Arithmetic Unit of the 1110 is organized in separate components for address generation, jump condition testing, arithmetic operation and operand storage. As a result of this segmentation, it is possible to have four instructions stacked in various phases of execution, giving an effective instruction time of 300 nanoseconds under favorable circumstances. Multiple index registers and accumulaters are available which allow a properly organized program to minimize storage references, thereby contributing to the overall efficiency of the system.

The whole system was conceived with two thoughts in mind: to make it compatible with the 1106 and 1108 and to make it an effective multiprocessor. The former was taken care of by using the same instruction set, except for the addition of character/byte manipulation and certain control instructions. That the latter was achieved is evident from the fact that only multiprocessor configurations are offered, ranging from the two CAU, one IOAU minimum to the maximum mentioned above.

The 1110 system basically supplies increased computing power by supplying additional computing units.

3.12.3 CDC 7600. The CDC 7600 is one of the largest and fastest computers in use today; it derives its power and speed to a large degree from the organization of its Central Processing Unit (CPU). This unit is made of four interconnected sections which operate together on a 27.5 nanosecond clock period. The sections and their communications links are represented in Figure 3.12.

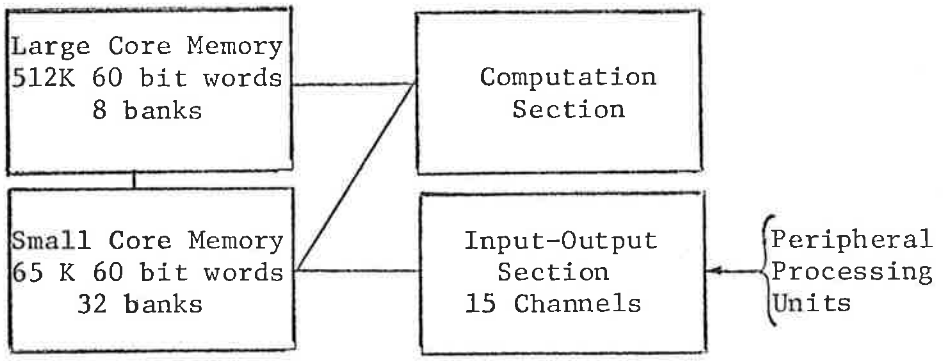


Figure 3.12.1 CDC 7600 CPU Schematic

All communications with the outside are via the 15-channel Input-Output section, which maintains buffer areas in the Small Core Memory (SCM). This core memory with 275 nanosecond read/write cycle time is organized in 32 independent banks of 2048 words each. One bank may be accessed each clock pulse so that ten of them may be in some stage of the read/write cycle simultaneously giving a maximum transfer rate of one word each 27.5 nanoseconds. The Large Core Memory (LCM) is organized in 8 banks of 64K words each, all of which may be accessed simultaneously, again with one access starting per clock cycle. Eight words are transferred each read/write cycle of 1760 nanoseconds which gives the same maximum word transfer rate as the SCM.

Programs in the 7600 may address locations in both Large and Small Core Memories, but the programs themselves operate out of the SCM. The general scheme is to store two programs with their data in the LCM and to load one of them into SCM for execution. If that job is held up for I/O or other reasons, it is swapped back into LCM for the other job. (Jobs may be unloaded onto a mass-storage device if the delays are too long, so that there is always a program in LCM ready to run.)

The CPU has an organization as shown in Figure 3.12.2 , taken from the 7600 reference manual [3].

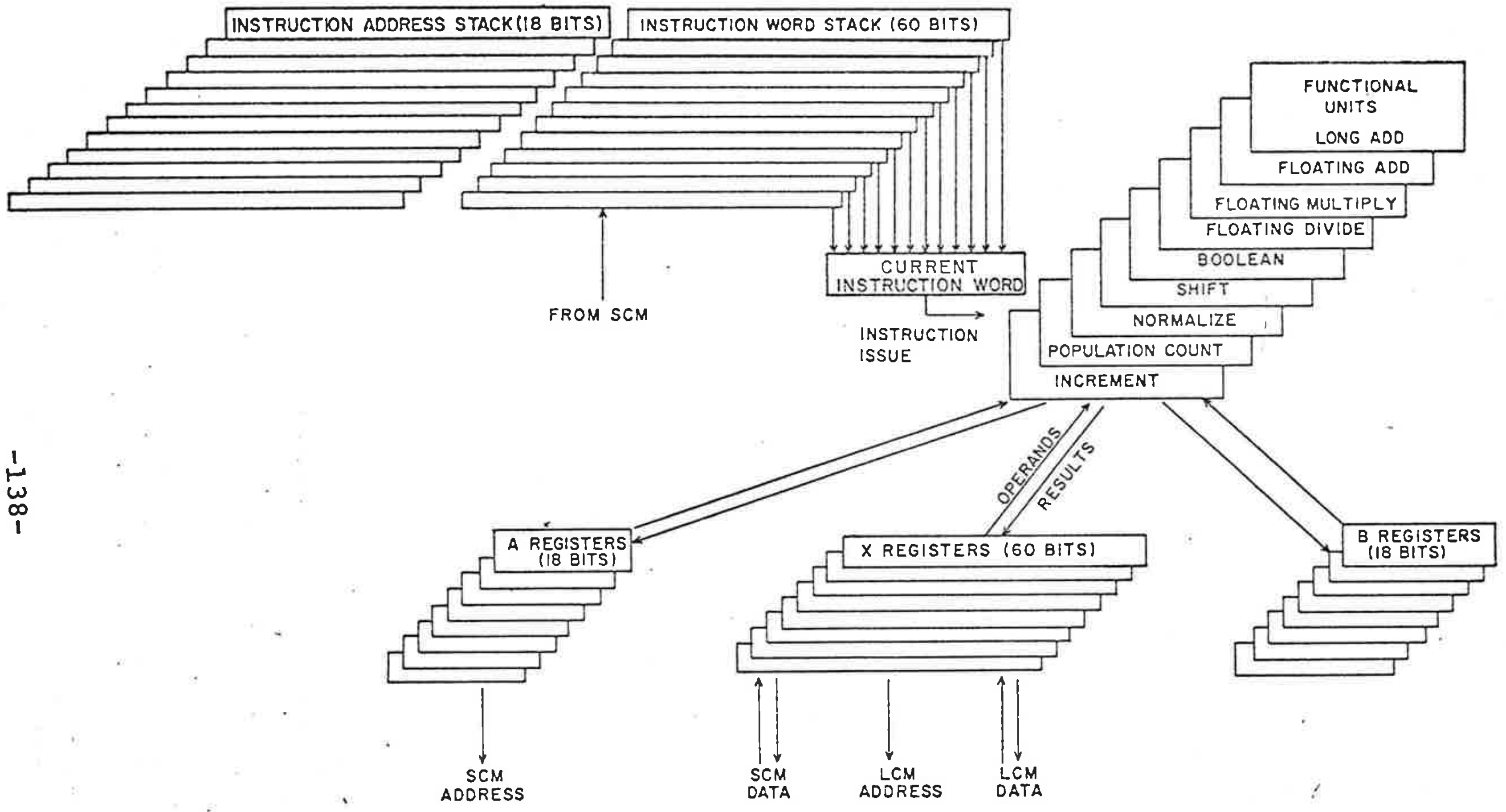


Figure 3.12.2 CDC 7600 CPU Organization

The words of the program are read from SCM into the 12-word Instruction Word Stack (IWS) with the corresponding addresses being stored in the Instruction Address Stack (IAS), with the stack being filled two words ahead of the word being currently executed. The Current Instruction Word (CIW) register is filled from the stack with the instruction next to operate, which can contain from two to four CPU instructions. If the sequence of instructions in the program is purely sequential, the IWS is like a moving window through which one can see the program two words ahead of the current location. When a branch occurs, the destination address is checked against the entries in the IAS so that any instructions already there may be used without further reference to memory. If the required word is not found, of course, it is obtained from SCM. Thus, it is possible for a short program loop to be entirely contained in the IWS requiring no memory references for its operation. It is also possible that a number of groups of instruction words will be in the stack at once.

The instruction words are interpreted in the CIW and the instructions are passed to the functional units for operation. These units operate independently and according to a fixed sequence of steps. Each one is like a "pipeline" or "delay line" into which a new set of operands, for a computation unrelated to those already going on, may be entered at each clock pulse. The only exceptions are the floating multiply which has a two clock period "segmentation" and the floating divide which does not allow segmentation because of an iterative algorithm.

Exchanges of data between the CPU and memory are accomplished through the A, X and B registers of which there are eight each. One of each, Aϕ, Xϕ and Bϕ, are used for special purposes but of the rest, the A registers are used as address registers pointing to SCM operands, the X registers as operand registers and as LCM address registers and the B as indexing registers. Reading from SCM is accomplished by loading one of A1 through A5 with an SCM address; the contents of that address will immediately be stored in the corresponding X register. Similarly, if an operand is loaded into X6 or X7 and an SCM address is loaded into the corresponding A register, then the operand is immediately stored in the appropriate place in SCM.

Single word transfers between LCM and the CPU use the X registers for address and operand and block transfers between SCM and LCM use Aϕ and Xϕ as block starting address registers for SCM and LCM, respectively.

In sum, then, the 7600 is an example of a large-scale machine which attains high performance through an architecture which allows a great deal of parallelism in its operation and a memory-register hierarchy that minimizes memory references.

References:

[1] IBM, A Guide to the IBM System/370 Model 165, GC20-1730-0, June 1970.

[2] Sperry-Rand, UNIVAC 1110 System - System Description, UP-7841, 1970.

[3] CDC, Control Data 7600 Computer System Preliminary Reference Manual, Pub. No. 60258200, 1969.

4.0 ALGORITHMS FOR ATC

The air traffic control function of any ATC system is quite naturally concentrated in the set of applications programs that is a large part of the system. These programs are called variously Task, Problem and Functional programs and subprograms, but, nomenclature aside, it is they that handle the actual data and transform it in ways that produce output useful to the ATC process. They do this by implementation of an algorithm.

An algorithm is, by definition, an ordered sequence of operations performed on a set of input data and conditions so as to produce a required set of output data and conditions. These algorithms can be expressed in the form of text, flow diagrams, mathematical equations or a combination of these. Frequently, alternate presentations are given of the same algorithm as an aid to comprehension.

An algorithm may be implemented by means of a hand calculation; a set of tables or nomographs; a collection of mechanical, pneumatic, hydraulic, electrical or electronic components; or, most important for the present purposes, a program for a digital computer system. Indeed, that these implementations are equivalent is the basis for the substitution of general-purpose digital computers, with their applications programs, for special-purpose hardware in many systems.

In the following sections, we will discuss first of all how to choose, from among the many algorithms which could perform a given function, the one best for advanced ATC application. The remaining sections will review some of the algorithms already implemented and/or proposed for implementation in future systems.

4.1 CHOOSING ALGORITHMS

One of the elemental principles in the design of systems, large or small, is that each of the various subsystems should be designed with due regard for the other components of the system. The choice of algorithms to be implemented in ATC systems should be made with this principle in mind.

The overall objective is, of course, to do the required job, but there are secondary considerations of cost and efficiency. Every task subprogram--implementation of an algorithm--must be written so as to operate in the shortest time consistent with use of the smallest set of resources (e.g., storage), thus contributing to overall system efficiency. The ideal situation, of course, would exist if there were a well-defined "cost function" for program-operation which could be evaluated for each algorithm and used as the basis for choice. No such function exists, however, since there is no general agreement, for example, about the "price" of a microsecond's running time as against an extra word in memory.

Despite the qualitative nature of the conclusions which may be drawn, however, a number of useful criteria can be developed for matching algorithm to computer system. Clearly, there are trade-offs here between special features of the computers (e.g., associative memories, array processors, microprogrammed instruction codes) and the special algorithms that take advantage of these features as opposed to the more strictly general-purpose machines and the algorithms appropriate to them. In other words, if increased speed is needed, one can attempt to improve the algorithms and speed up the computer, or one can utilize a special hardware feature at the expense of more expensive equipment less useful for general purposes.

The four qualities on which the suitability of an algorithm will be judged are, then: (1) closeness of model to the real world, or said another way, correctness of output, (2) accuracy and precision of the output, (3) the speed and efficiency of the implementation of the algorithm, and (4) its appropriateness to the computer on which it will be operated.

One can envision a computer-aided design process wherein the designer is provided with three things:

1. A realization of the algorithm expressed in a canonical form and in a machine-independent language.

2. A description of the computer system in the form of tables of parameters suitably organized.

3. A cost, or utility, function expressed in terms of the algorithm output.

With these in hand, he goes to a computer program which in some sense, runs the algorithm on the target computer and evaluates the utility function. The designer may then alter the algorithm or the computer system, or both, so as to maximize the function.

Clearly, this type of program will not be easily constructed; there are a number of problems to be solved in its implementation. Programs do exist, however, which do somewhat similar functions in the system design and simulation process, so the ground is at least broken in some areas.

4.2 RADAR CORRELATION AND TARGET TRACKING

The most obvious process in an Air Traffic Control system is that of correlating sensor data--generally from radars--with aircraft tracks maintained by the system, followed by the extrapolation of these tracks to the next period of the cycle. Since this is also the ATC process with the longest history and the one on which the most labor has been spent, we will discuss it at some length.

The correlation, smoothing and prediction algorithms for the currently implemented ARTS III and NAS systems will be described and compared in some detail, followed by remarks concerning extensions to other computer systems and the introduction of newer concepts.

4.2.1 Correlation. The correlation between radar data and tracks of aircraft is the first step in the process in both the ARTS III and NAS systems. The object is the same in each program but the implementations are not. The object is find for each scan by the radar, a one-to-one pairing of radar returns and tracks being maintained by the program. The problem is relatively straightforward, with the only complications being introduced by the fact that (1) there are both primary and beacon returns, (2) there are discrete and non-discrete beacon returns, (3) there may be multiple coverage, and (4) there may be garbled and other erroneous returns. The programs attempt to match the returns to the tracks by a number of means: search areas centered on the most likely position of the return, a hierarchy of choices from among the multiple returns, etc.

NAS

The first step in the NAS correlation process is to assign each radar return to a "radar sort box" (RSB), one of a set of rectangular areas into which the sector is divided by a grid aligned with the system axes. These boxes are nominally 25 n. mi. on a side. At the same time, each track will have been assigned to a similar "track sort box" (TSB). These areas have the same dimensions and alignment as the radar sort boxes, but are offset by one-half the length and one-half the width of the RSB's.

Now, the pairings of radar returns and tracks can be made: the process is carried out for each radar datum in turn. That is, for each radar return, an attempt is made to correlate with those tracks which are in the four track sort boxes which intersect the radar sort box in which the datum is located. This provides an initial filtering, as it were, so as to concentrate attention on a small set of tracks and radar returns. The pairings between the tracks and data are then tested for correlation and those that pass are assigned a Correlation Preference Value (CPV) depending on the type of return and its source.

Tests for correlation are of two kinds: (1) type of return vs. type of track and (2) distance. The former filters primary data from beacon tracks and vice versa, while the latter chooses the datum nearest the track. Basically, there are two types of area check: a small search area (SSA) check, which is expected to detect the return from the aircraft if it was in straight-line flight, and the large search area (LSA) check, which will detect the return if the aircraft was turning. If the track-datum pair correlates in the small search area, the large search area check is suppressed.

The end result of the correlation process is, hopefully, a single track-datum pairing for each track in the system, such that the datum has a high probability of being from that track.

ARTS

Correlation in the ARTS system also attempts to match a radar return to a predicted track position. In this system, however, the tacit assumption is made that there is a single sensor, and all correlation processing is done in (r, θ) for that radar and in terms of 32 azimuth sectors scanned by that sensor.

A list of all tracks in the sector being processed, plus those in the preceding and following two sectors, is prepared and correlation is attempted between each track and the current sector's radar returns. A search area, or bin, is constructed around the track position to be used as a distance test. The dimensions of the bin are variable and depend on the track type and its history. The history is reflected in a quantity called "firmness" which is calculated for each track: firmness increases with successful correlations and decreases with misses.

The correlation process involves two passes through the list of tracks in order to resolve as many ambiguities as possible. If there are still multiple returns correlating, a choice is made based on distance and type. If no return is found during this "primary" correlation, a "secondary" correlation with larger search area is attempted. The objective of this secondary correlation is to detect and to lock on to a turning target.

At certain values of firmness, successful or unsuccessful correlations, as the case may be, cause transitions from one type of track to another: e.g., from initial to normal, normal to turning, etc. An event worth mentioning is the creation of three tracks from one upon unsuccessful secondary correlation under certain values of firmness; the three tracks are left and right turning tracks and a straight-ahead, so-called, "parent" track.

4.2.2 Smoothing-position. Once a radar datum/track pair has been selected, it is necessary to account for the difference between the two. The track position is a predicted position subject to prediction error and the radar datum is a measured position subject also to error. The algorithms used by both the NAS and ARTS systems for the normal straight-line tracking is expressed as

$$\tilde{x}_n = \hat{x}_n + \alpha \, \Delta x_n \qquad (4.1\ a)$$

$$\tilde{y}_n = \hat{y}_n + \alpha \, \Delta y_n \qquad (4.1\ b)$$

where

$\tilde{x}_k, \tilde{y}_k$ = the smoothed position at scan k

$\hat{x}_k, \hat{y}_k$ = the predicted position at scan k

α = the position smoothing constant

$\Delta x_k = \hat{x}_k - \bar{x}_k, \ \Delta y_k = \hat{y}_k - \bar{y}_k$

and $\bar{x}_k, \bar{y}_k$ = radar measured position at scan k.

The same smoothing constant is used for x and y, which are parallel to the system axes and are treated symmetrically.

NAS

In the NAS system, the automatic tracking program, which does the position smoothing among other things, is called every 5 seconds (nominally) to operate on track/datum pairs that have been produced by small search area correlation. The smoothed position for all such is computed with equations (4.1 a,b) with the smoothing constant α being a fixed system parameter taken from a table and depending on the type of return that has been correlated with the track.

Once every ten seconds, the large search area smoothing is done. I.e., for those pairs which had a datum in the large search area, the program works only half as often. Furthermore, the smoothing equation is not the same as for SSA smoothing. The smoothing of data correlated in the large search area puts more weight on deviations perpendicular to the flight path than along it; this is consistent with the expectation that LSA-correlation detects turning targets.

The equations for LSA smoothing are

$$\tilde{x}_n = \hat{x}_n + \alpha_1 D_R \hat{x}_{n-1} + \alpha_2 E_R \hat{y}_{n-1} \qquad (4.2a)$$

$$\tilde{y}_n = \hat{y}_n + \alpha_1 D_R \hat{y}_{n-1} + \alpha_2 E_R \hat{x}_{n-1} \qquad (4.2b)$$

where α_1 = longitudinal smoothing constant

α_2 = lateral smoothing constant

$\hat{x}_k, \hat{y}_R$ = velocity components computed at scan k

$$D_k = \frac{\Delta x_k \hat{x}_{k-1} + \Delta y_k \hat{y}_{k-1}}{(\hat{x}_{k-1}^2 + \hat{y}_{k-1}^2)} = \text{longitudinal component unit vector}$$

$$E_k = \frac{\Delta x_k \hat{y}_{k-1} - \Delta y_k \hat{x}_{k-1}}{(\hat{x}_{k-1}^2 + \hat{y}_{k-1}^2)} = \text{lateral component unit vector}$$

The coefficients, α_1 and α_2, are also system parameters whose values depend on the type of return correlated.

ARTS

The ARTS system uses equations (4.1a,b)for position smoothing with the smoothing coefficient α being chosen as a function of the track type and firmness: in general, the less firm the track, the smaller the smoothing factor.

Note that correlation of radar data with tracks was done in terms of (r,θ) while smoothing of data was done in (x,y); this implies a coordinate conversion of the x, y predicted positions back to r, θ. There is a net saving in doing it this way, for only one computation per track is required as opposed to one per radar return, including both beacon and primary as well as noise returns.

4.2.3 Smoothing-velocity. Velocity smoothing, prediction or, even better, correction is done in general by the use of the equations:

$$\hat{\dot{x}}_n = \hat{\dot{x}}_{n-1} + \frac{\beta}{T} \Delta x_n \qquad (4.3a)$$

$$\hat{\dot{y}}_n = \hat{\dot{y}}_{n-1} + \frac{\beta}{T} \Delta y_{n-1} \qquad (4.3b)$$

where $\hat{\dot{x}}_k$, $\hat{\dot{y}}_k$ = the components of predicted velocity for k^{th} scan

β = the velocity smoothing coefficient

T = scan time

Δx_k, Δy_k are as before.

Note that velocity is not measured, as position is.

NAS

Velocity smoothing in the NAS system is done only for those tracks for which flight-plan information is not available. These are called "Free" tracks as opposed to Flight-Plan-Aided tracks or "FLAT". The velocity used in subsequent calculations for FLAT tracks is the flight-plan velocity.

Free tracks in the small-search area use equations (4.3a,b) with values of the smoothing coefficient taken from a prestored table according to the CPV, as in position smoothing.

Smoothing in the large serach area uses the following equations.

$$\hat{\dot{x}}_n = \hat{\dot{x}}_{n-1} + \frac{\beta_1}{T} D \hat{\dot{x}}_{n-1} + \frac{\beta_2}{T} E \hat{\dot{y}}_{n-1} \qquad (4.4a)$$

$$\hat{\dot{y}}_n = \hat{\dot{y}}_{n-1} + \frac{\beta_1}{T} D \hat{\dot{y}}_{n-1} + \frac{\beta_2}{T} E \hat{\dot{x}}_{n-1} \qquad (4.4b)$$

where β_1 = longitudinal smoothing constant

β_2 = lateral smoothing constant

Other quantities are as defined above. The values of β_1 and β_2 are again dependent on the type of radar return correlated.[1]

ARTS

The ARTS system uses equations (4.3a,b) for velocity smoothing with the value of the smoothing constant depending, as before, on the track firmness.

4.2.4 Position Prediction. In general, position prediction is done using the equation:

$$\hat{x}_{n+1} = \tilde{x}_n + \hat{\dot{x}}_n T \qquad (4.5a)$$

$$\hat{y}_{n+1} = \tilde{y}_n + \hat{\dot{y}}_n T \qquad (4.5b)$$

where all of the terms are as defined above. This is a straight-line prediction for the interval in question, but of course, any heading change by the aircraft is reflected in the changing velocity components.

NAS

The NAS system uses the equations (4.5a,b) in a straightforward way, the expectation being that the use of LSA velocity smoothing as discussed above is enough to ensure tracking during turns.

ARTS

In the ARTS system, on the other hand, a distinction is made between straight-line tracking and turning tracking. In the former, equations (4.5a,b) are used, while in the latter, the following equations are used:

$$\hat{x}_{n+1} = \tilde{x}_n + (\hat{\dot{x}}_n^2 + \hat{\dot{y}}_n^2)^{1/2} T \cos(\theta_n + RT) \qquad (4.6a)$$

$$\hat{y}_{n+1} = \tilde{y}_n + (\hat{\dot{x}}_n^2 + \hat{\dot{y}}_n^2)^{1/2} T \sin(\theta_n + RT) \qquad (4.6b)$$

where R = rate of turn
θ_k = heading of track at scan k

and the other terms are as defined above.

These equations are used on the basis that aircraft, when they turn, do so at a constant, universal rate. Apparently, this is nearly so within speed classes, with the breakpoint being about 210 knots. The rate above 210 knots is taken as 1.5 degrees/second and the rate below that speed as 3 degrees/second.

As was pointed out earlier, straight and turning tracks corresponding to the same aircraft are carried under certain circumstances.

4.2.5 Timing. Each of the systems is designed to work in synchronism with the scan of a search radar: the basic timing cycle being one scan. The NAS system assumes a nominal 12 second scan and the ARTS system a 4 second scan. The implication is that a single sensor is supplying the data (or the unlikely event that completely synchronous multiple sensors are used). The extension to multiple sensors is discussed below.

NAS

Each scan of the radar is divided into two equal subscans; the actual scan time is a system parameter initially set to 10 seconds.

The radar correlation program runs about once per second, processing whatever data happens to have been gathered in the preceding second. At the start of the second subscan, the tracking program will update the position of all tracks in the system, either updating on the basis of small search area data or extrapolating in the absence of such data. The updating is done from a point midway into the first subscan to a point midway into the second subscan.

At the start of the first subscan, the tracking program will operate on all tracks with data correlated in the small search area, then on tracks with large search area data but no small search area data during the last scan and finally those tracks with no data. The updating is done from a point midway in the second subscan of the preceding scan to the middle of the first subscan of the current scan.

The objective of this relatively complex treatment is to match the dynamics of the aircraft to that of the sensor. After all, at 550 miles per hour, the aircraft moves about 1.5 miles in the ten second radar scan.

ARTS

As implied earlier, the timing of all processing in the ARTS system is keyed to the 32 sectors into which each scan of the radar is divided. That is, each routine, such as the tracking program, for example, must operate once each 125 milliseconds, corresponding to the rotation of the antenna through one sector.

During the time the antenna is in a particular sector, say S, all of the various subprograms run (e.g., primary and secondary correlation, smoothing, prediction, etc.) but each will be working on a different set of tracks and/or radar returns, according to a fixed preset schedule. For instance, primary correlation will be working with tracks in sector (S-4), secondary correlation with tracks in sector (S-6) and prediction in sector (S-8).

Since the ARTS system operates at a relatively short range, the motion of targets in azimuth may introduce large angular errors if it is assumed that the time between radar observations is equal to the scan time. Therefore the predicted position is subjected to a second-order correction thusly:

$$\hat{x}^{1}_{n+1} = \hat{x}_{n+1} + \hat{\dot{x}}\Delta T \qquad (4.7a)$$

$$\hat{y}^{1}_{n+1} = \hat{y}_{n+1} + \hat{\dot{y}}\Delta T \qquad (4.7b)$$

where $\hat{x}^{1}_{R}$, $\hat{y}^{1}_{k}$ = corrected predicted position corrdinates

and

$$\Delta T = (\hat{A}_{n+1} - \hat{A}_{n})T \qquad (4.8)$$

where $\hat{A}_{k}$ = azimuth corresponding to $(\hat{x}_{k}, \hat{y}_{k})$.

This correction is made only if $|\Delta T| \geq 1/4$ second.

4.2.6 <u>Summary and Extensions</u>. The preceding discussion makes it clear that although the NAS and ARTS III systems make use of correlation and tracking algorithms that seem on the surface to be similar--the "simple α,β-tracker"--they do in fact use quite dissimilar and far from simple algorithms. The departures from the straight-forward representation of equations 4.1, 4.3 and 4.5 have been introduced to make what seems like a simple and reasonable formulation work in the real world.

The earliest changes and additions to the algorithms were made in the SAGE era when the concept was introduced of a "time-frame" roughly synchronized with the rotation of the main sensors. Next, attempts were made to overcome the real limitation of this tracker; namely, that the two requirements on the system, stability of tracking in the presence of noise and sensitivity to target maneuvers, are contradictory. The use of different sized and/or differently aligned search areas, variable tracking constants depending on track history and auxiliary equations for turning tracks has helped make this technique at least viable but only at the expense of vastly increased complexity, where complexity means increased computing time and storage requirements.

Many people have suggested a new look at tracking with the dual aim of providing better tracking and a more efficient algorithm from the point-of-view of computation. The technique most often mentioned is the one adapted from Kalman's work in optimum control and filter theory. This formulation can be stated as follows:

$$\underline{\tilde{x}}_n = \underline{\hat{x}}_n + \underline{K}_n \, [\underline{z}_n - \underline{h}(\underline{\hat{x}}_n)] \qquad (4.9a)$$

$$\underline{\hat{x}}_{n+1} = \underline{f}(\underline{\tilde{x}}_n) \qquad (4.9b)$$

where $\underline{\tilde{x}}_k$ = the smoothed state vector, position and velocity, at time t_k,

$\underline{\hat{x}}_k$ = the predicted state vector at t_k,

$\underline{K}_k$ = the matrix of gain coefficients defining the smoothing process at t_k,

$\underline{z}_n$ = the measurement vector at t_k,

$\underline{h}(\underline{x})$ = the function of target state defining the measurement process,

and $\underline{f}(\underline{x})$ = the function of target state defining the target motion.

There are various simplifying assumptions which can be made, such as linearizing $\underline{h}$ and $\underline{f}$ and decoupling motions in some of the directions. These simplifications will introduce errors, of course, but with the proper analysis, the errors can be kept small.

A second area in which research could produce better algorithms in the sense of efficiency is in the area of parallel processing. There is a great deal of interest being shown in the development of array and parallel processors and associatively addressed memories which could be used to implement newer forms of tracking equations. The natural view of the situation is that the correlation of data with and the tracking of all of the targets can be carried out simultaneously in parallel. This may be the correct approach, or there may be a different organization, say in terms of areas, that will give better results.