# Chapter 45

# The TI ASC: A Highly Modular and Flexible Super Computer Architecture[1]

*W. J. Watson*

## Introduction

Early in 1966, a large computer development program was begun by Texas Instruments. The goal for this effort was to provide needed capacity for supporting seismic processing, plus offering a general super computer capability in the support of new markets.

This development has resulted in the Advanced Scientific Computer (ASC)—a highly modular system offering a wide spectrum of computing power and configurability.

## Overview of the System

The major subsystems of a typical configuration are shown in Fig. 1: the central memory, the central processor, the peripheral processor, on-line bulk storage, a digital communications interface, plus a selection of standard peripherals.

The peripheral processor has been designed for executing the operating system. The central processor has been designed expressly to provide high computing power for large arrays of data. The central processor operates as a slave to the peripheral processor. This design approach was chosen to maximize the overlapping of system overhead tasks with the execution of user programs. In operation the job stream is analyzed by the peripheral processor. The language processors, plus user object code, are executed by the central processor. System control and I/O tasks are processed by the peripheral processor. I/O is routed through high-speed, head-per-track disc storage. A data communications interface for the common carriers is provided for the support of remote batch and interactive terminals. Standard types of peripherals are also provided. The central memory serves as the common access communications and access storage medium for these subsystems.

## Central Memory

The ASC central memory consists of a memory control unit (MCU) and appropriately sized modules of high-speed or medium-speed central memory. Optionally, a medium-speed central memory extension can be used in conjunction with a high-speed memory.

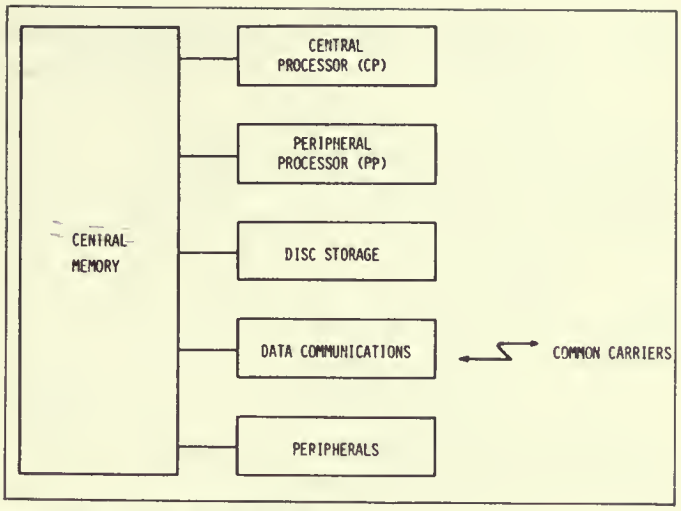


**Fig. 1. Major ASC subsystems.**

The MCU is organized as a two-way, 256-bit/channel (8-word) parallel access traffic net between eight independent processor ports having full accessibility to all memories. The nine memory buses are organized to provide eight-way interleaving for the first eight buses with the ninth bus used for the central memory extension. The MCU provides the facilities for controlling access from the eight processor ports to a CM having a 24-bit address space (16 million words). A port expander can be utilized to expand the number of processor ports. Fig. 2 illustrates this structure.

The MCU is designed to operate asynchronously, independent of cable delays, processor clock rates, and memory unit access and cycle times. This capability allows for a great deal of flexibility to accommodate improvements in memory or processor technologies which may be desired. The MCU is capable of handling a maximum data transfer rate of 80 million words per second per port, giving a total transfer capacity of 640M words per second. Therefore, a significant capacity beyond today's memory and processor speeds is available in the MCU.

The semiconductor high-speed central memory modules have a cycle time of 160 ns and a read time of 140 ns. Additionally, all transfers are 256 bits (eight 32-bit words) with a Hamming code providing single-bit error correction and double-bit error detection for each 32-bit word. High-speed central memory is typically divided into eight equal sized modules which permits eight-way interleaving. A patch board within the MCU controls the memory address decoding and sets the interleaving pattern.

The optional central memory extension provides for large

[1]*Proc. AFIPS FICC*, 1972, pp. 221–228. The section on software beginning on p. 759 is excerpted from Dean [1973].

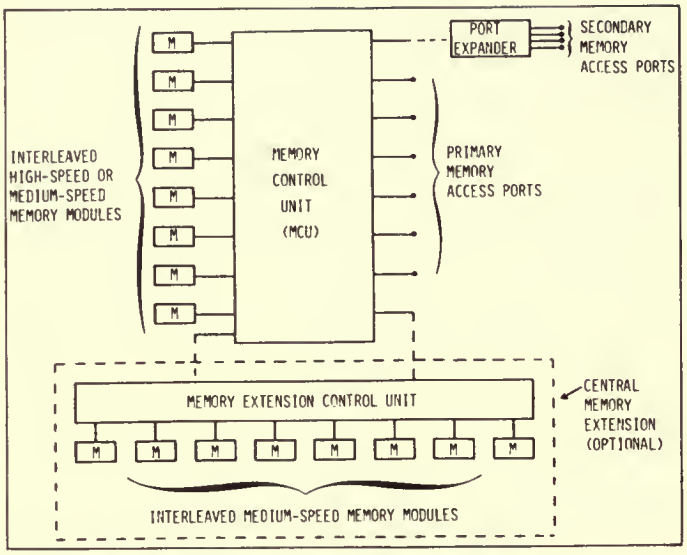


**Fig. 2. Modular structure of the ASC central memory.**

amounts of relatively economical medium-speed memory to be utilized in support of the high-speed central memory. The memory extension uses 1 μs semiconductor technology and is also accessed in 8-word increments. Single-bit error correction is provided at the 8-word level. The central memory extension is included in the address space of the central memory and, therefore, can be addressed by a processor or channel controller for instructions or operands. It is also possible to effect block transfers of data between high-speed memory and the memory extension. This is possible because both a normal memory bus and a memory access port are provided. Block transfers are initiated by the peripheral processor with the specification of the source starting address, the destination starting address, and the block length. The block transfer proceeds automatically at 40M words per second, and the peripheral processor is notified upon completion.

The central memory size is limited only by the 24-bit address (16 megawords). The proportions of fast memory and memory extension may be varied in order to balance memory capacities to suit the particular system requirements. The present high-speed memory module is modular from 16K to 128K 32-bit words. This permits memories from 128K to one million words to be configured.

Central memory management and access control of memory ports is achieved through the use of two facilities: map registers and protect registers. Each user program has its own unique page address map. Page addresses not required by the program are mapped into absolute page zero which is not accessible to the CP. When a program is loaded into memory, it will likely be loaded into discontiguous memory pages. During program execution, program developed page addresses are converted, without execution time penalty, to actual page addresses by the map registers. Because a reference to page zero is denied and the relevant processor notified, the map registers provide for inter-user memory protection. Figure 3 shows the mapping scheme. Desired page sizes depend on the amount of central memory and the problem mix of a particular installation. Four different page sizes may be specified for an ASC system, varying from 4K to 256K words. A program may utilize any one of the page sizes available.

The protect registers allow for intra-user protection. These registers consist of three pairs of bounds registers for defining the upper and lower addresses of access for read, write, or execute areas. The five combinations of protection presently used by the system software with the bounds registers are:

- Execute Only
- Read Only
- Execute, Read, No Write
- Read, Write, No Execute
- Read, Write, Execute

An attempt to reference an area out of bounds for a particular control state is denied and the processor notified of the attempted violation.

In large ASC systems, more processors and control units require additional access ports to memory. In these cases memory port expanders are utilized to provide additional ports and are utilized to service the devices not requiring the full bandwidth of a memory port. Each memory access port expander provides a 1:4 expansion with a maximum bandwidth degradation of ten percent;

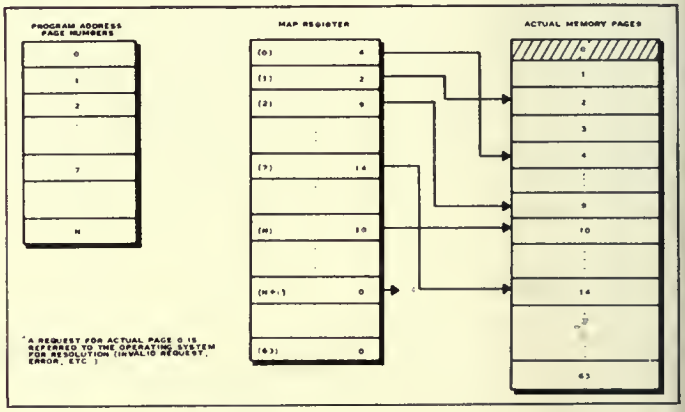


**Fig. 3. Memory mapping.**

i.e., from 80 million 32-bit words per second to approximately 72 million 32-bit words per second. These expanders can be concatenated to provide further increases in connectivity. Priorities at the single access port interface are resolved on either a fixed or distributed basis. The mode is selected by patch card wiring in the expander hardware.

## Central Processor

The central processor (CP) provides both scalar (single operand) and vector (array) instructions at the machine level. The basic instruction size is 32 bits, with 16-, 32-, or 64-bit operands. The single instruction stream, which contains a mixture of scalar and vector instructions, is preprocessed by the instruction processing unit.

The central processor design is such that one, two, three, or four execution units or "pipes" can be provided. These units employ the pipeline concept in both scalar and vector modes. A single execution unit can have up to twelve scalar instructions in process at one time. From one to four vector results can be produced every 60 ns, depending on the number of execution units provided.

The CP has 48 program-addressable registers. This group of 32-bit registers consists of sixteen base address registers, sixteen arithmetic registers, eight index registers, and eight vector parameter registers. This last group is used to extend the instruction format for the complete specification of vector instructions. The basic instruction format is shown as it relates to these register groups in Fig. 4.

The CP scalar instruction repertoire includes an extensive set of Load and Store instructions: halfword, fullword, and doubleword instructions, with immediate, magnitude, and negative operand capabilities. Ability to load and store register files and to load effective addresses is also available. Arithmetic scalars include various adds, subtract, multiply, and divide for halfword (16-bit) and fullword (32-bit) fixed point numbers and fullword and doubleword (64-bit) floating point numbers. Scalar logical instructions are provided as are arithmetic, logical, and circular shifts. Various comparison instructions and combination comparison-logical instructions are provided for halfword, fullword, and doublewords. Many combinations of test and branching instructions with incrementing or decrementing capability are also available. Stacking and modifying arithmetic registers can be done with single instructions. Subroutine linkage is accomplished through Branch and Load instructions. Format conversion for single and doublewords, as well as normalize instructions, are available.

The vector capabilities of the CP are made available through the use of VECTL (vector after loading vector parameter file) and VECT (assumes parameter file is already loaded) instructions. The vector repertoire includes such arithmetic operations as add, subtract, multiply, divide, vector dot product, matrix multiplication, and others for both fixed point and floating point representations. Vector instructions are also available for shifting; logical operations; comparisons; format conversions; normalization; and special operations—such as Merge, Order, Search, Peak Pick, Select and Replace, among others.

One important characteristic of the vector instruction capability is the ability to encompass three dimensions of addressability within a single vector instruction. This is equivalent to a nest of three indexing loops in a conventional machine.

The basic structure of the CP, shown in Fig. 5, has three major components: the instruction processing unit (IPU) for non-arithmetic stages of instruction processing for the CP instruction stream, the memory buffer unit (MBU) to provide operand

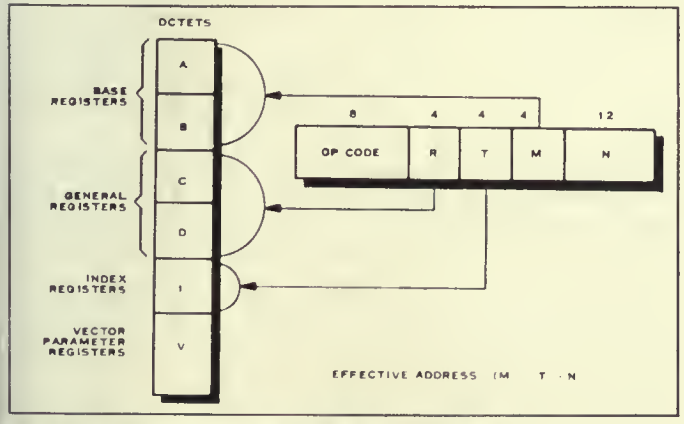


Fig. 4. Instruction format and register groups.

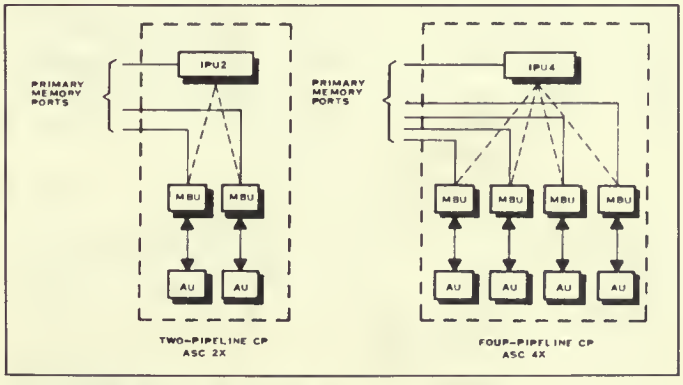


Fig. 5. Basic structure of the CP.

interfacing with the central memory, and an arithmetic unit (AU) to perform the specified arithmetic or logical operations. Figure 5 shows a CP diagram for 2- or 4-pipeline CP's, each with a corresponding number of MBU-AU pairs. Note that a memory port is required for the IPU and, in addition, one memory port for each pipeline (MBU-AU pair) in a CP.

A significant feature of the CP hardware is an operand look-ahead capability which causes memory references to be requested prior to the time of actual need. Double buffering in multiple 8-word (octet) buffers for each pipeline provides a smooth data flow to and from each arithmetic unit. The pipelined AU achieves its highest sustained flow rate in the vector mode, typically a result each 60 ns per AU.

### *Instruction Processing Unit*

The primary function of the instruction processing unit (IPU) is to supply a continuous stream of instructions for execution by the other parts of the CP. One Central Memory port is required to provide the instruction stream. Two 8-word (octet) buffers are utilized to achieve a balanced stream of instructions from memory to the IPU. Instructions are transferred from memory in octets as are all other references to memory for fetching or storing of information.

The following functions are performed by the IPU: (1) instruction fetch, (2) instruction decode, (3) register operand selection, (4) effective address development through indexing and/or indirect addressing, (5) immediate operand development, (6) branch address development, (7) determination of branch condition, (8) storage of AU results into the register file, (9) scalar hazard and register conflict resolution, (10) generation of vector starting addresses, and (11) transmittal of vector parameters to the MBU during vector initialization.

Up to 36 instructions in various stages of execution can be overlapped within the 4-pipe CP. There are twenty positions for instructions in the 2-pipe CP and twelve positions for instructions in the 1-pipe CP. Four levels are contained within the IPU, and eight levels are contained in each arithmetic pipeline (MBU-AU pair). In addition to the previously mentioned functions, the IPU performs routing of instructions to the MBU-AU pairs based on an optimum use of arithmetic unit capability.

Vector processing is altered by software in order to distribute segments of the vector for multiple pipe systems.

Several features are provided to alleviate the potential problems of branches and instruction dependencies in the instruction pipeline. The Prepare-to-Branch instruction, used extensively by the Fortran compiler, increases the execution speed of branches, particularly important in loop iterations. This instruction provides the IPU control hardware with advance address information to facilitate uninterrupted instruction processing. Instruction dependencies are recognized by the hardware. It scans the instruction stream and distributes the independent instructions across MBU-AU pairs to insure proper, yet efficient, execution sequences.

### *Memory Buffer Unit*

The memory buffer unit (MBU) provides an interface between central memory and the arithmetic unit. Its primary function is to supply the arithmetic unit with a continuous stream of operands from memory and to provide for the storing of the results back to memory. Note that all references to memory, whether for fetching or storing, are made in 8-word increments (octets).

The MBU has three double buffers, one octet per buffer, called the "X" and "Y" buffers for input and the "Z" buffers for output. This double buffering is provided so that pipeline processing can be sustained at a high rate with minimal memory access conflicts. These buffers are illustrated in Fig. 6.

During scalar operations, data specified by effective addresses developed in the IPU are fetched or stored as required. The Z buffer can be transferred directly to the X or Y buffers so that memory references are not necessary for scalar operands which reside in the Z buffer.

For most vector operations, two operand data strings are fetched, while a result data string is stored. Addresses for sustaining the vector operations are computed in the MBU using parameters initially specified by the vector parameter file.

### *Arithmetic Unit*

The primary function of a CP arithmetic unit (AU) is to perform the arithmetic operations specified by the operation code of the instruction currently at the AU level. There is one AU per pipeline

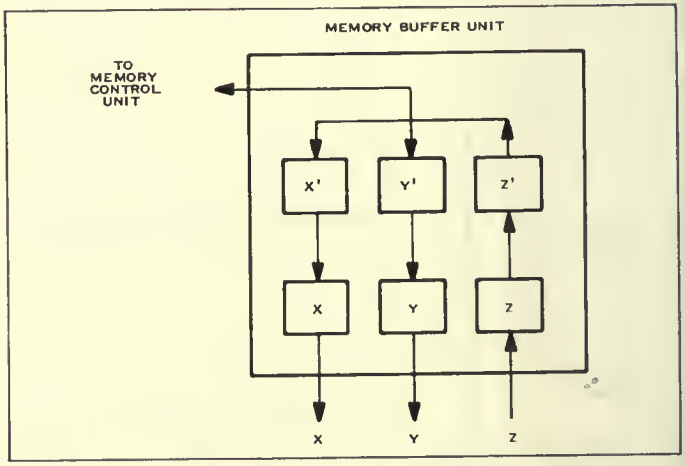


**Fig. 6. Multiple operand streams in the memory buffer unit.**

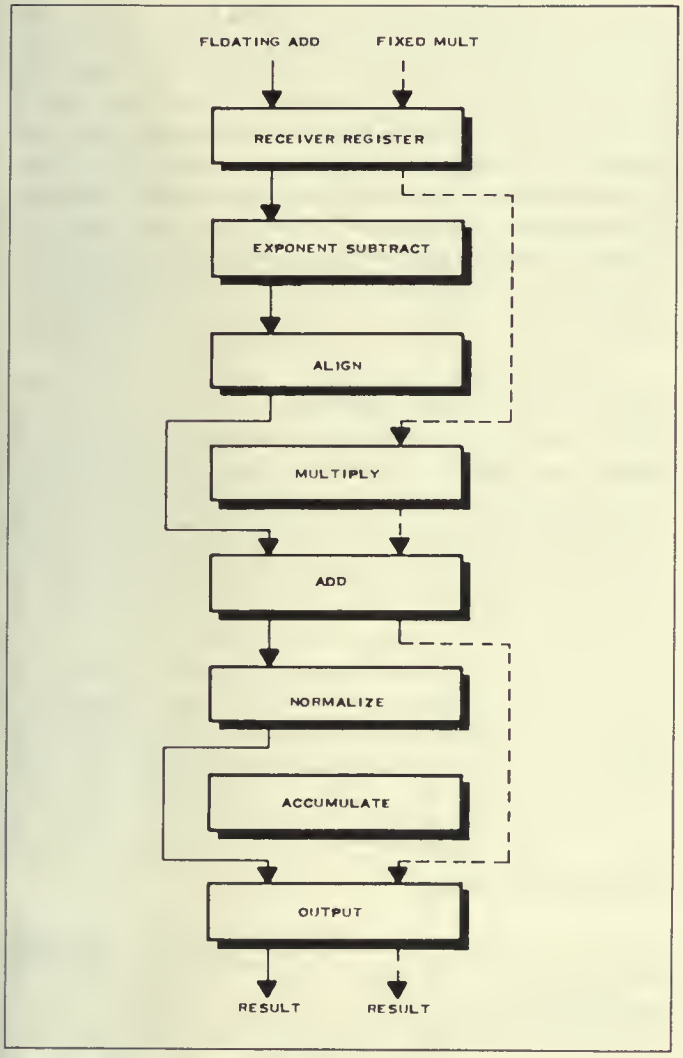


**Fig. 7. Arithmetic unit pipeline concept.**

in the CP, each having a 60 ns basic cycle time. A distinguishing feature of an AU is the pipeline structure which allows efficient execution of the arithmetic part of all instructions. There are eight exclusive partitions of the AU pipeline involved, each of which can provide an output every 60 ns. These eight sections are (1) received register, (2) exponent subtract, (3) align, (4) add, (5) normalize, (6) multiply, (7) accumulate, and (8) output. Figure 7 shows how different sections of the AU are utilized for execution of particular instructions; i.e., floating point addition and fixed point multiplication.

An AU is a 64-bit parallel operating unit for most scalar and vector instructions. Exceptions are double length multiply and all types of division. In these circumstances various combinations of the components of the AU are utilized; and therefore, more than one clock cycle is required to complete these arithmetic operations.

Fixed point negative numbers are represented in two's complement notation, and the floating point representation is hexadecimal with the exponent biased by $40_{(16)}$.

## The Peripheral Processor

The peripheral processor (PP) is a powerful multiprocessor designed to perform the control and data management functions of the ASC. Several aspects of the implementation of the peripheral processor concept greatly increase the effectiveness of the ASC system. Figure 8 shows the logical organization of the PP.

The PP is a collection of eight individual processors called virtual processors (VP's). Each VP has its own program counter along with arithmetic, index, base, and instruction registers. The eight VP's share a read only memory, an arithmetic unit, an instruction processing unit, and a central memory buffer. Use of the common units is distributed among the VP's using sixteen single 85 ns cycles. When an equally distributed sequence of time units is used, each of the eight VP's receives two 85 ns cycles every 1.4 μs. The typical PP instruction requires two 85 ns cycles for completion. The distribution of available time units can be dynamically varied to suit particular processing requirements. Figure 9 illustrates two possible distributions.

The read only memory within the PP is utilized for program

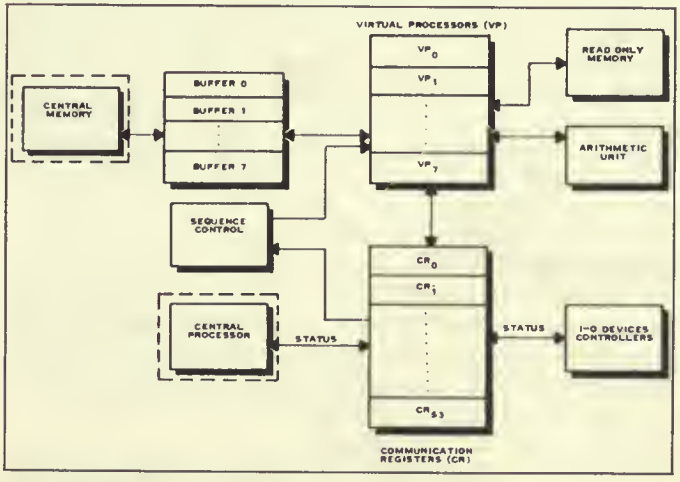


**Fig. 8. Peripheral processor.**

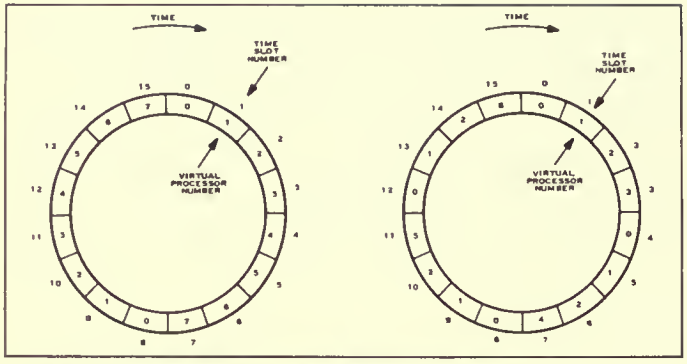


**Fig. 9. Two possible VP time slot assignments.**

storage and execution of those short routines which are highly utilized by the VP's, such as polling loops. The read only memory consists of up to 4K 32-bit words of non-volatile memory elements with a cycle time of less than 85 ns. It is modular in 256-word increments.

Because the PP is intended to perform control functions rather than execute mathematical algorithms, the instruction set is oriented toward control operations and does not require multiplication, division, or floating point operations. The instruction format is similar to that of the central processor, using a 32-bit word for each instruction. Instructions are provided for bit (1 bit), byte (8 bits), halfword (16 bits), and fullword (32 bits) operations.

Each VP has direct access to the entire central memory for program execution and data storage. Therefore, a single copy of reentrant code can be executed simultaneously by more than one VP.

The communications register (CR) file contains sixty-four 32-bit word registers which are program addressable by the VP's. The CR file serves as the principal storage medium for control information necessary for the coordination of all parts of the ASC system. Synchronization of communications is achieved between all processors (CP, VP's channel controllers, and peripheral unit controllers) from interpretation of status bits received from all devices into the CR file.

## Disc Storage

Disc storage is the principal secondary storage system for the ASC system. Disc storage consists of head-per-track (H/T) disc systems supplemented by positioning-arm disc (PAD) systems.

### *Head-per-Track (H/T) Disc System)*

The H/T disc system is a high-performance device whose effective performance is further enhanced because the operating system utilizes a shortest-access-time-first (SATF) algorithm [Denning, 1967] for data transfers. This combination of hardware and software provides a very high effective transfer rate. Each H/T disc module has a capacity of 25 million 32-bit words with a transfer rate of approximately 500K words per second. Using the shortest-access-time-first algorithm, access time will average approximately 5 ms which results in an exceptionally fast "effective" transfer rate. The rotational period of the disc is 32 ms. Each H/T disc module has seven discs with fourteen surfaces. Two surfaces of the module are used as alternate storage for inoperative sections. For data ordering purposes, the discs are divided into bands and then further subdivided into sectors of 64 words each.

### *Positioning-Arm Disc (PAD) System)*

The PAD system, when utilized to supplement head per track, is available in a variety of configurations. Control of PAD systems is achieved by use of channel interface, disc controller, and disc interface units. From two to eight PAD disc drives may be attached to a set of control devices. The number of controllers and discs per controller will depend upon the storage and retrieval problem requirements.

The PAD system has a transfer rate of 200K words per second and a storage capacity of 25M words per disc drive. Access time is divided into two categories: positioning-arm time which is 30 ms average with a maximum of 55 ms and average rotational latency which is 8.4 ms. Thus, average total access time is approximately 38 ms.

## Data Communications

The data communication system is very modular and, thus, externally flexible in the various devices which may be utilized for communication with the ASC. Data communications are controlled by a data concentrator which, in turn, interfaces to the MCU through a channel control device.

### *Data Concentrator*

The data concentrator is a TI-980 minicomputer equipped with special-purpose hardware communication interface units on its direct memory access ports. The TI-980 is a small, general-purpose computer with up to 64K 16-bit words of memory and a one-microsecond cycle time. The data concentrator hardware is under control of a data communications operating system which executes in the TI-980. This operating system provides for the functions of buffering, reformating, routing, protocol handling, error control and recovery procedures, and system control messages. The system services multiple stations concurrently.

The data communications system presently supports communication with three types of stations: high-performance user terminals, other large computers, and remote concentrators. The system can be easily extended to support smaller terminals down to the teletype level. These stations may be either remote or local. When local, the communication link is implemented with multiple conductor cables. Since the transfer is asynchronous by word, the average transfer rate is very dependent upon cable length with a maximum transfer rate of 250,000 words per second for distances less than 500 feet.

### *Remote Links*

Remote links are presently implemented with non-switched, full duplex common carrier data transmission facilities. Data is transferred over these links synchronously at rates determined by the modems and common carrier bandwidths. The data communication system supports transfer rates up to a maximum of 240,000 bits per second. Because the system supports full duplex transmission, this capacity typically translates to the ability to support a 1200 lpm printer simultaneously with a 1000 cpm reader over a 9600 bps transmission facility.

## Peripherals

Standard types of magnetic tape drives, card equipment, and printers have been interfaced with the ASC. These interfaces are attached to primary or secondary memory ports through a variety of standard selected and multiplexed data channels.

## Summary

Preservation of global system modularity concepts in the design of the ASC has resulted in a capability for configuring systems having a very wide range of cost and capabilities.

In the memory area capacity, performance, connectivity, protection, and mapping are all variable over wide bounds. The central processor can be tailored to provide a wide range of processing power by using one, two, three, or four pipes.

The peripheral processor provides for dynamically matching the execution rates of up to eight independent instruction streams with the task requirements. The highly flexible communication register file provides a matrix of 2048 bits which can be manipulated and sensed by the eight virtual processors. Flexible hardware interfaces are provided for coupling these bits to external I/O signal lines. Finally, the modular read only program memory of the peripheral processor accommodates growth and modifications in read only memory resident operating system code.

An example of a complete system configuration is illustrated in Fig. 10.

## Software[1]

At the beginning of the software design effort, several goals were established which have directed the development effort. It was desired that the system support multi-programming, local and remote batch processing, as well as multiple users of the previously mentioned interactive terminals. It was considered important that the powerful central processor be reserved for the scientific computations for which it was designed and that as much as possible of the "overhead" function be performed in the Peripheral Processor Unit. It was determined that the first users of the ASC had a significant investment in Fortran coded programs. Fortran was thus selected as the first high level scientific language and it was important that the compiler produce highly efficient object code with no change in the source. It was required that an extensive file management system be provided with special emphasis on privacy of files. It was desired that the services and facilities of the system—both hardware and software—be made available to the user in a simple and straightforward manner. Simple jobs should require only minimal user descriptions and knowledge of the inner workings of the operating system should not be a requirement for the efficient use of the ASC system. Finally, it was recognized that each installation has somewhat different workload and priority requirements. Anticipating that some of these requirements might be over-looked in the initial design, it was thus considered important that the system be modular and easily modified to meet each installations' particular needs.

In the following paragraphs a description of the ASC Fortran is given.

### *ASC Fortran*

As was mentioned as a design goal, the ASC Fortran was designed to accept previously coded Fortran programs. As such, it contains ANSI Fortran and Fortran IV as a part of its language, but also contains certain extensions. The compiler is designed to optimize the execution of the object code on the ASC. It also performs an extensive diagnostic analysis complete with a set of appropriate output messages.

In the area of extensions, two new features are worthy of special mention. These are the subarray and array cross section. The following example illustrates the concepts using three dimensional

[1]The section from here to the end of this chapter is excerpted from L. C. Dean, "Texas Instructments Advanced Scientific Computer," *Informatie jaargang*, vol. fifteen, no. 4, April 1973, pp. 191–193.

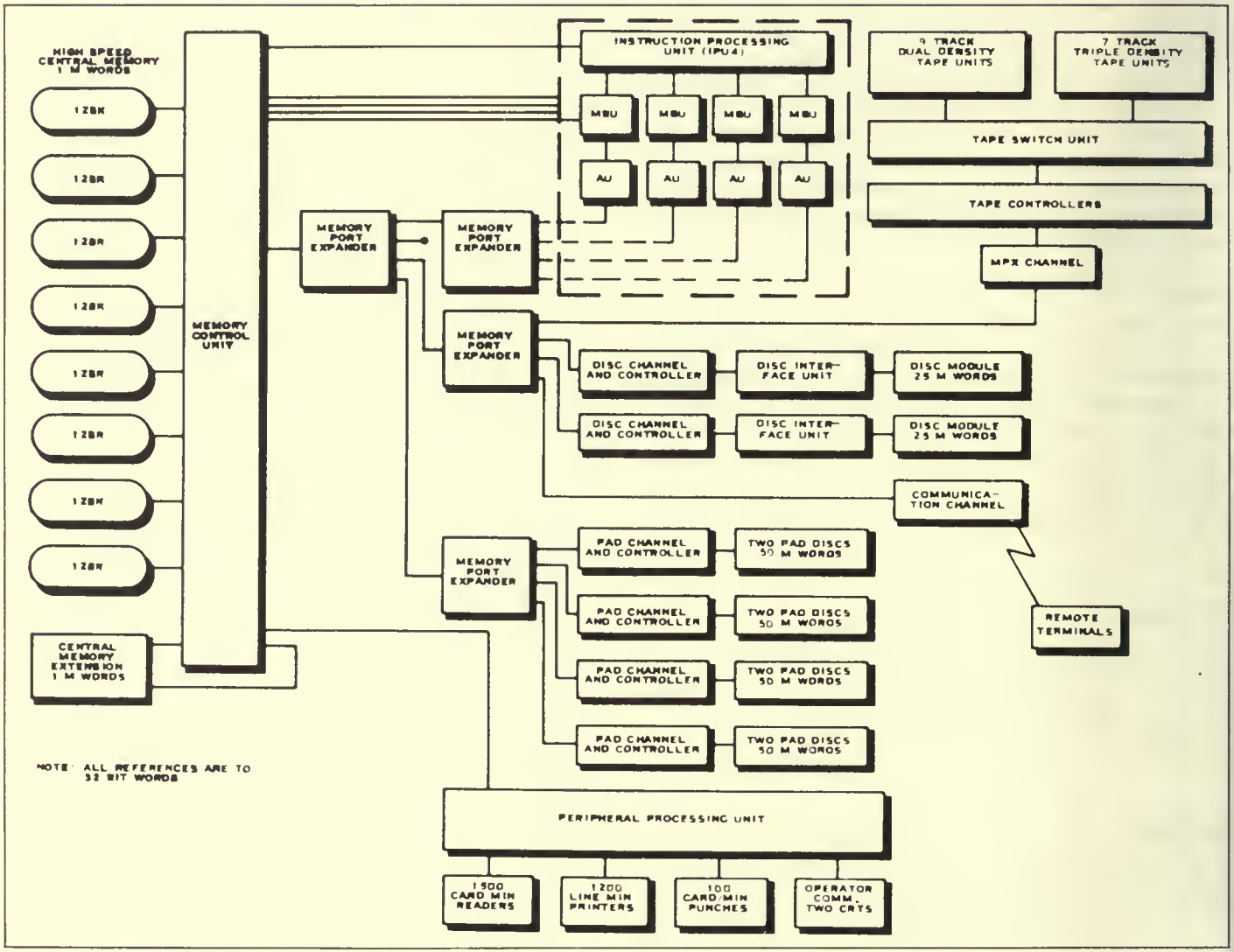


**Fig. 10. A possible ASC system configuration.**

arrays, although both subarray and array cross section may be applied to arrays up to a dimension of seven.

```
Dimension A (50, 20, 30)
    .
    .
    .
- - - - A (*, *, 13) - - -
    .
    .
    .
Subarray C (4, 5, 8) at A (23, 2, 11)
```

As shown, the "parent" array, A, is dimensioned as (50, 20, 30). If in an executable statement of the Fortran source code, the array A(*, *, 13) appears, it means the array consisting of all points lying in the 13th "plane" of the array A. Thus the asterisk in a given subscript position means to vary that subscript over its defined range of values in the conventional order. The array cross section may also be used as A (*, *, J) where the array defined depends upon the current value of J.

The subarray statement defines a three-dimensional subset C of A, with dimensions (4, 5, 8), with the point C (1, 1, 1) at A (23, 2, 11). The subarray statement may also be used as: Subarray C(I, J,

K) at A (L, M, N) by (P, Q, R) where the current values of I, J, K determine the size of the array C, where the current values of L, M, N determine the position of C within A, and where P, Q, R determine the increments to be used on the subscripts of A to determine the values in the array C. This latter feature allows the array C to be "less dense" than the parent array A. It has particular utility when iterative procedures are used and larger "grid" spacing is desired at early stages of the iteration.

Both the array cross section and the subarray statement are powerful new tools to the Fortran programmer. They not only allow for more efficient memory utilization and minimize the memory to memory data movement, but they allow the programmer to formulate his array processing problem in a more direct and convenient manner. Both of these extensions are presently under consideration by the X3J3 Standards Committee. They have been approved "in principle" by this committee.

Optimization algorithms within the compiler include such conventional areas as constant propagation, elimination of redundant sub-expressions, reduction in operator strength, register assignment, and removal of loop constant assignment statements and loop constant expressions from DO loops. Further, it includes paired memory fetching techniques, retention of intermediate values, and extensive re-ordering of instructions for optimum "pipe-line" flow.

One of the most powerful procedures in the optimization algorithms is the automatic conversion of scalar source code into vector instructions. The most frequently occurs within DO loops and often results in the complete elimination of software indexing. Figure 11 is an illustration of this process. In Example 1 of this illustration, the source code appears in standard Fortran. In example 2, the array cross-section feature is used. In example 3, pure vector representation is employed. Regardless of the source code, the ASC Compiler will produce a single vector instruction in the object code. In addition to this vectorization, the Compiler uses vector instructions for assignment statements where possible, and provides extensive analysis to optimize memory referencing (e.g., loop reversal and re-ordering).

Figure 12 is an illustration of this type of analysis. The original source code is given in example 1 of that illustration. If no vector instructions or memory optimation were employed, the object code would require approximately 20,000 clocks. Vectorization yields a reduction to 2351 clocks, almost a 10:1 improvement. By reversing the order of the loops and using temporary storage, the execution time can be reduced to 880 clocks even though two passes through memory are employed. In example 3, the loops are reversed and one is inverted, yielding a further reduction and eliminating the need for temporary storage. Finally, in example 4, loops are both inverted and reversed yielding an execution of 437 clocks. Since 400 items are to be moved this result is probably near optimum (only 37 clocks of "overhead"). This example is obviously a simple one involving only the movement of data, but it illustrates the kind of analysis performed by the ASC Compiler in optimizing memory references.

To allow maximum use of mathematical functions within instructions which can or should be vectorized, both scalar and

```
                Dimension M (3,3), N (3,3)
                Integer K
                  •
                  •
                  •
Example 1.      DO 10 J = 1,3
                DO 10 I = 1,3
             10 N (I, J) = M (I, J) −K
                  •
                  •
Example 2.      DO 10 J = 1,3
             10 N(*,J) = M(*,J)−K
                  •
                  •
Example 3.      N = M−K
```

Fig. 11

```
1 Basic problem
      Dimension A(25,25)
      DO 10 I = 1,20
      DO 10 J = 1,20
   10 A(I,J + 1) = A(I + 1,J)

2 Reversal of loops
  Reversal of loops introduces fault. This can be circum-
  vented by introduction of temporary vector.
      DO 10 J = 1,20
      DO 10 I = 1,20
   10 T(I,J) = A(I +1,J)
      DO 20 J + 1,20
      DO 20 I = 1,20
   20 A(I,J + 1) = T(I,J)

3 Loops reversed and inverted
  This removes the fault without introducing temporary
  vector.
      DO 10 J = 1,20
      DO 10 I = 1,20
   10 A(I,22−J) = A(I + 1,21−J)

4 Loops reversed and both inverted
  This has properties of solution 3 but makes better use
  of memory.
      DO 10 J = 1,20
      DO 10 I = 1,20
   10 A(21−I,22−J) = A(22−I,21−J)
```

Fig. 12. Vector optimization.

"vector" mathematical subroutines are provided. This allows such functions as "cosine" to have meaning when the argument is an array. For example, COS (A) means: compute the cosine function for every element of A when A is an array.

In general, this will give a performance improvement of between 6:1 and 8:1 (depending on the mathematical function) over repeated calls to the scalar math routines. There is a slightly higher overhead in set up time for the vector math routines, but it appears that the cross-over point is about six elements, i.e., if the number of arguments is greater than six, it is faster to use the vector math pack version.

## References

Dean [1973]; Denning [1967].