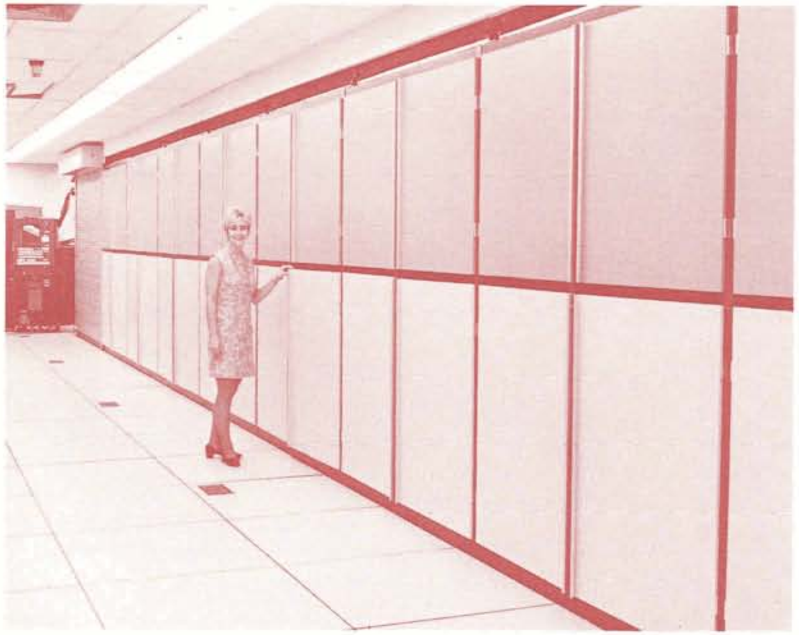
ILLIAC IV

ILLIAC IV

The ILLIAC IV System represents a fundamentally different approach to data processing. The limitation imposed by the velocity of light, once thought to be an absolute upper bound on computing power, has been stepped over by several approaches to computer architecture, of which the ILLIAC IV is the most powerful by as much as a factor of four.

The conquest of the limitations of the velocity of light was foreseen by Herman Kahn and A.J. Wiener in 1967, when they wrote: ". . . . .over the past fifteen years this basic criterion of computer performance has increased by a factor of ten every two or three years . . . . . While some will argue that we are beginning to reach limits set by basic physical constants, such as the speed of light, this may not be true, especially when one considers new techniques in time sharing, segmentation of programs to add flexibility, and parallel processing computers. . .(such as). . .the ILLIAC IV . . ."


ILLIAC IV Quadrant

ILLIAC IV represents a significant step forward in computer systems architecture offering

– greatly improved performances:

- 200 MIPS computation speed
- 10^9 bits/sec I/O transfer rate
- 10^6 bytes of high-speed integrated circuit memories
- 2.5×10^9 bits of parallel disk storage

– contemporary technology:

- ECL circuits
- semiconductor memories
- belted cables

– and a new approach to the art of computing using parallelism, which offers an opportunity to programmers to utilize the vast power of the system as effectively as possible.

MAJOR SYSTEM ELEMENTS

As shown in the accompanying system diagram, the major elements of the ILLIAC IV System are the Array Subsystem, the I/O Subsystem, the Disk File Subsystem, and the B 6700 Control Computer Subsystem.

The main computing power resides in the Array Subsystem. The ECL circuit family is used to implement the logic in the Array Subsystem. In the array is a Control Unit (CU) directly governing 64 identical Processing Units (PU). Each PU is principally a combination of a Processing Element (PE) and a Processing Element Memory (PEM). The PE has no independent control except for mode, some data dependent conditions, and addressing within its own memory. Mode control permits a PE to accept or ignore a broadcast control sequence from the CU, depending on the current status of its mode bit. The PE is essentially a four-register arithmetic unit capable of executing a full repertoire of instructions having 64-bit, 32-bit, and 8-bit operands. Directly associated with each PE is a PEM having 2048 words of 64-bits each, 4096 words of 32 bits each, or some combination of both sizes.

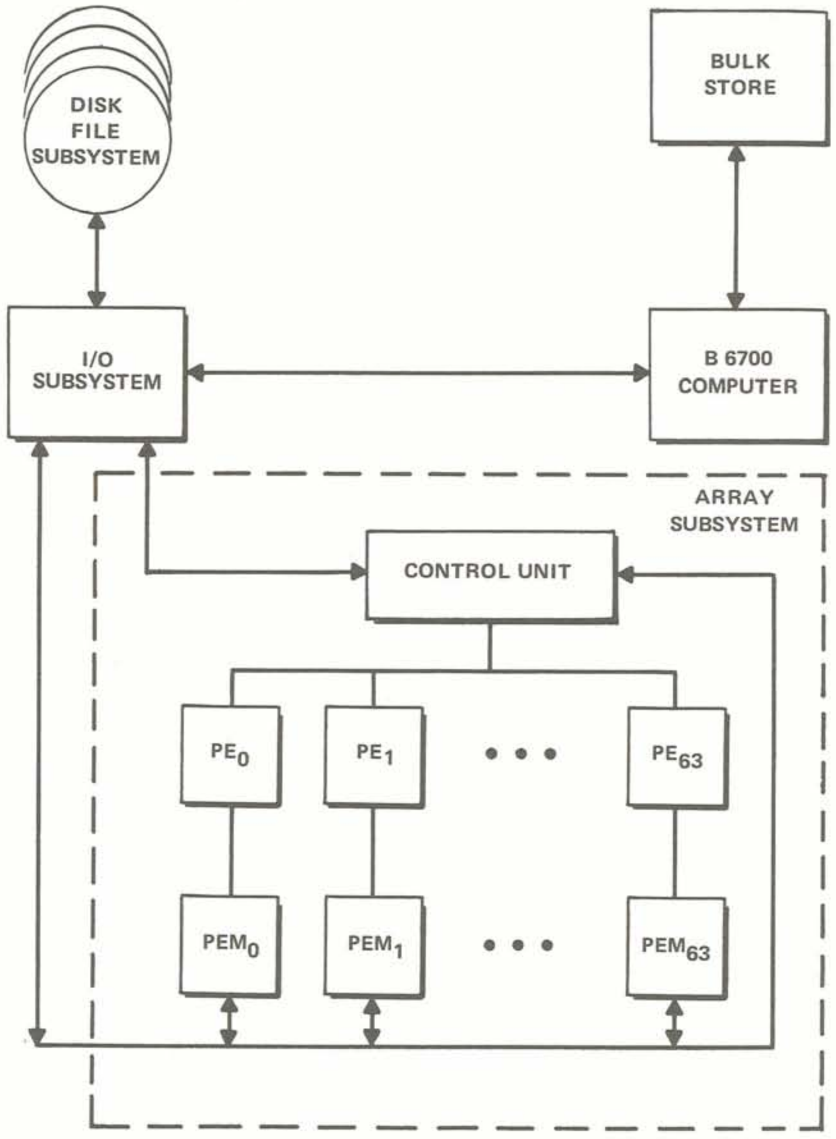
The I/O Subsystem controls the routing of data among the other major elements of the system and a 1024-bit wide interface that may be used for a variety of purposes, depending on the application.

The Disk File Subsystem provides an intermediate data storage for the array having a storage capacity up to 2.5×10^9 bits of storage and a transfer rate up to 10^9 bits per second.

A B 6700 is the control computer for the ILLIAC IV System. This computer provides executive control, facility allocation, peripheral equipment control, I/O initiation and control, fault recovery, and program assembly and compilation.

The logic of the I/O Subsystem, the Disk File Subsystem, and the B 6700 Control Computer Subsystem is implemented in the CTL circuit family.

Each of these elements of the system is discussed in more detail on the following pages.



ILLIAC IV Functional Diagram

FEATURES OF ARRAY OPERATION

Efficient operation of parallel array programs requires machine features that are unfamiliar to designers and users of conventional serial machines. The following discussion, which highlights the operation of the array at the individual instruction level, has several sections. The first presents the conventional aspects of array operation, and hence, discusses what appears to be a conventional instruction set. The other sections discuss the capabilities which have been added to facilitate array processing, such as on-off control of the Processing Elements, routing of data among Processing Elements, one-clock alignment and normalization of floating point numbers, independence among PE's of the address field of the instruction, and broadcasting.

CONVENTIONAL INSTRUCTION SET

There are 65 computers in the ILLIAC IV array. Of these, 64 are identical, and are called the Processing Elements (PE). The 65th computer is imbedded in the Control Unit (CU). Most instructions are conventional, such as add, multiply, fetch, store, and are either PE or CU instructions. A PE multiply instruction, for example, causes the contents of every PE accumulator to be multiplied by a second operand specified in the address field of the instruction. A CU add instruction adds the contents of one of the words of the local operand store to the contents of one of the four accumulators in the CU. The CU computer is used for purposes like loop control. The major burden of the data processing is on the PE's, whose instruction set is relatively conventional, containing instructions such as add, multiply, logical OR, divide, fetch, store, and register-to-register moves. Typical instruction times are given on page 8.

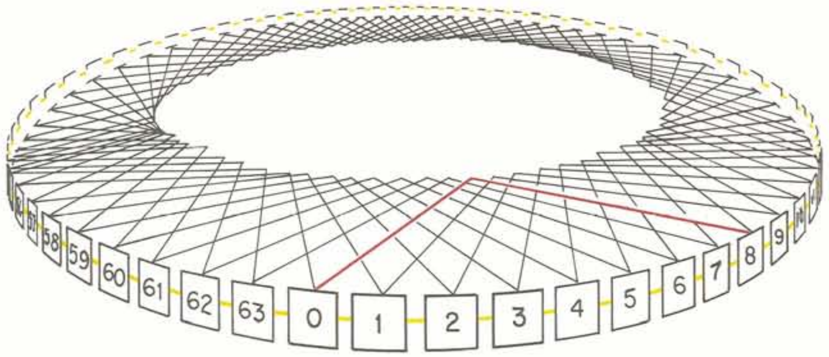
ONE CLOCK NORMALIZATION

All PE's must operate together with essentially no control response back to the CU. For example, when normalizing sums from floating point addition, every PE has a different shift amount. If the various PE's took different times to shift, the CU would have to wait a minimum of 240 nanoseconds until all PE's had reported the completion of shifting. The 240-nanosecond period is the round trip time for the cabinet length of over 50 feet.

A barrel switch is, therefore, provided in the PE that can shift any amount in one-clock time. The CU, giving the command to normalize, does not wait for a response before going on. Likewise, all shift instructions take one clock time since they use the same barrel switch.

ROUTING

Routing is a mechanism whereby the PE's can exchange data rapidly and simultaneously. The instruction that accomplishes this is called the "route" instruction. Routing consists of taking 64 words of data in the 64 registers in the 64 PE's and shifting it among the PE's by distance N modulo 64. That is, data starting in PE_0 winds up in PE_N; data starting in PE_1 winds up in PE_{N+1}. All such shifts are done simultaneously, so that all 64 words of data are transferred in one shift time. Shift time is a function of N, being the minimum of 125 nanoseconds for distances of either 1, 8, -1, or -8.



The above figure shows this end-around routing connection schematically. In addition to the neighbor-to neighbor linkages which form the PE's into a ring, there are connections of PE's eight apart such that data can leapfrog intermediate PE's when the distance to be covered is large.

ON-OFF CONTROL

On-off control of the PE is effected with control bits called mode bits. When the mode bit is set, the PE is operating normally; when the mode bit is reset, the PE will not execute the current instruction. Resetting the mode bit is, therefore, a mechanism whereby the PE can branch forward in the instruction stream.

There are two ways of controlling the PE's with the mode bits. Mode bits may be set by the result of tests in the PE's, or they may be set unconditionally by the CU. To illustrate the first method, assume a case where the PE should control its own performance (Let P be the PE number):

```
        IF X(P) ≥ Y(P) THEN DO STATEMENT A,
        ELSE DO STATEMENT B.
```

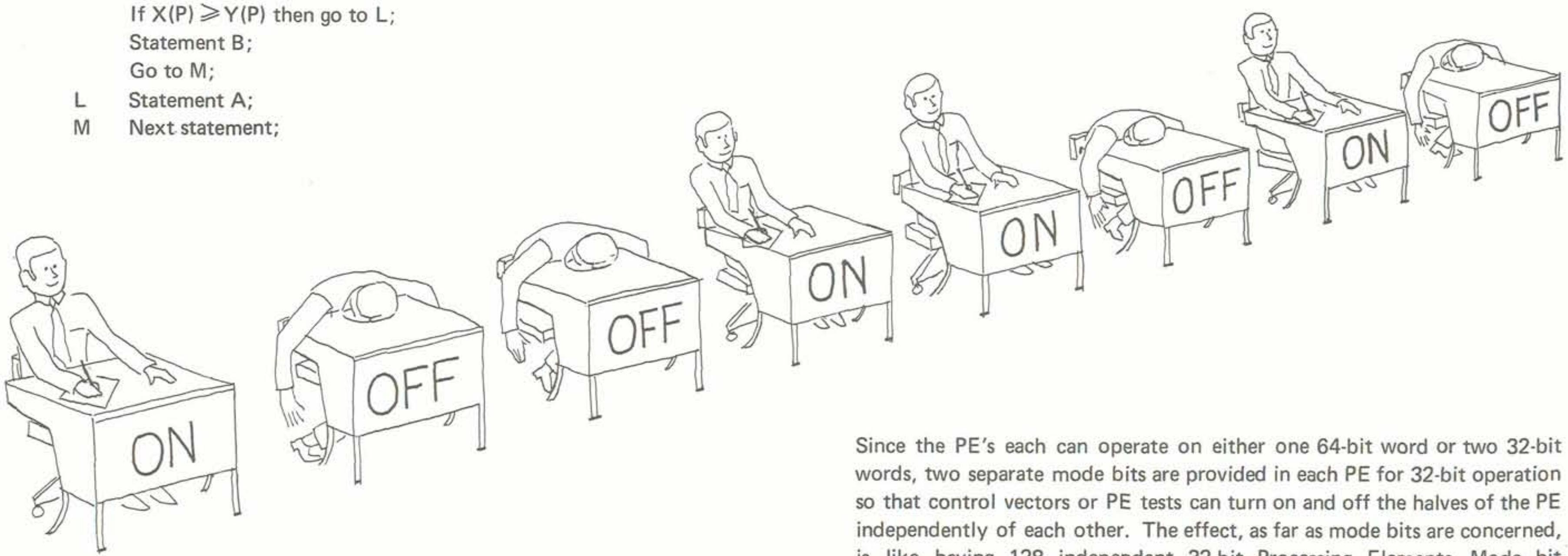
In conventional computers the code would read something like the following:

```
        If X(P) ≥ Y(P) then go to L;
        Statement B;
        Go to M;
   L    Statement A;
   M    Next statement;
```

In the ILLIAC IV, the code would read as follows, and 64 passes through *the code would be executed simultaneously.*

```
        If X(P) ≥ Y(P) set mode bit (P) to ON;
        Statement A;
        Complement mode bits;
        Statement B;
        Set all mode bits on;
        Next statement;
```

In the extended FORTRAN being implemented for ILLIAC IV, PE's can be turned on and off by "control vectors." These vectors have one bit per element, and are used to mask out any given operation. The control vectors are fetched to the CU and sent directly to the mode bits of each PE.



Since the PE's each can operate on either one 64-bit word or two 32-bit words, two separate mode bits are provided in each PE for 32-bit operation so that control vectors or PE tests can turn on and off the halves of the PE independently of each other. The effect, as far as mode bits are concerned, *is like having 128 independent* 32-bit Processing Elements. Mode bit control does not interfere with routing.

INDEPENDENCE OF ADDRESSING AMONG PE MEMORIES

Although each PE instruction is for 64 PE's, it contains only one address field. However, each PE has its own independent memory (PEM). To achieve independence of addressing within the PE memories, each PE is provided with an index register. The contents of this index register can be specified to be added to the content of the address field.

The simplest example of the utility of the index register lies in the fetching of a matrix that is stored skewed (See illustration below). By not indexing, a matrix row is fetched. By inserting a function of the PE's own number into the index register, and then indexing the address, a column of the matrix is fetched into the PE's (Column 4 is shown in black below left).

	PE_0	PE_1	PE_2	PE_3	PE_4	PE_5	PE_6	PE_7
Address 7	$a_{8,2}$	$a_{8,3}$	$a_{8,4}$	$a_{8,5}$	$a_{8,6}$	$a_{8,7}$	$a_{8,8}$	$a_{8,1}$
Address 6	$a_{7,3}$	$a_{7,4}$	$a_{7,5}$	$a_{7,6}$	$a_{7,7}$	$a_{7,8}$	$a_{7,1}$	$a_{7,2}$
Address 5	$a_{6,4}$	$a_{6,5}$	$a_{6,6}$	$a_{6,7}$	$a_{6,8}$	$a_{6,1}$	$a_{6,2}$	$a_{6,3}$
Address 4	$a_{5,5}$	$a_{5,6}$	$a_{5,7}$	$a_{5,8}$	$a_{5,1}$	$a_{5,2}$	$a_{5,3}$	$a_{5,4}$
Address 3	$a_{4,6}$	$a_{4,7}$	$a_{4,8}$	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{4,5}$
Address 2	$a_{3,7}$	$a_{3,8}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$
Address 1	$a_{2,8}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
Address 0	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$	$a_{1,8}$

Indexing at the PE level is also allowed on shift counts of shift instructions, and on the bit number of the bit-manipulation instructions, providing for versatility in nonnumerical processing. In double precision arithmetic, for example, the normalization correction, once developed, is inserted into the index register and used to control the double-length shift that accomplishes normalization.

BROADCASTING

Every PE instruction is sent to the PE's from the CU as 266 enable levels to each and all PE's. Accompanying this set of enable levels is a 64-bit "common data bus" that also goes to all PE's. Depending on the specific instruction and variant thereof, the data on the common data bus is used within the PE's as a 64-bit literal, or the least significant 16 bits may be used as the address of an operand in memory, or to designate a register in the PE, or as a shift distance. Memory addresses and shift distances may be further indexed in the PE, as described above.

This facility to transmit 64-bit literals is used in the software to transmit or "broadcast" those variables to the PE's that are constant across all PE's, as in the expression:

$$C(P) \leftarrow A \cdot B(P)$$

where P is the PE number.

Variable A will be fetched to the CU and broadcast, being the same for all PE's; whereas, B(P) is distinct in each PE and will be fetched out of the PE's memory.

PERFORMANCE

Algorithms for estimating the "speed", "power", or "throughput" of computers have been the most elusive and illusory in the industry. Among the simpler means employed have been the clock rate, add time, and more recently, "Mips" (million instructions per second). Bench mark programs have also been used with more success. Presented here is a discussion of the clock rate, the instruction times for some typical instructions, and a discussion of Mips. Such oversimplifications are recommended for use on only the grossest basis for comparison among various computers, since the actual performance on any given application is dependent on factors such as the memory allocation algorithm used, the compiler efficiency, as well as the application itself and the efficiency of the application programs. Listed below are salient characteristics of ILLIAC IV which depict the "speed", "power", and "throughput" of the machine. The following data are based on a 64-PU system. Future copies can also be built with 8, 16, or 32 PU's:

Data Rates

Between the 64 PE's and their 64 PEM's – 9,362,285,714 bits/sec (64 paths at 64 bits each 312.5 to 437.5 nanoseconds)

Between array memory and parallel disks – 1,004,000,000 bits/sec (980,468 transfers/sec of 1024 bits each)

Word bus of the B 6700 – 40,000,000 bits/sec

Clock Rates (MHz)

Array Subsystem	16
I/O Subsystem	10
B 6700 Subsystem	5
Clock track on parallel disk file	4.41

Memory Sizes

Array Subsystem (May be expanded in future copies)	8,388,608 bits or 1,048,576 bytes or 131,072 words of 64 bits or 262,144 words of 32 bits
B 6700 Subsystem (Modular in increments of 16,384 words)	524,288 words of 48 bits
Parallel disk file (Modular in increments of 78,643,200 bits)	2,516,582,400 bits

Memory Times

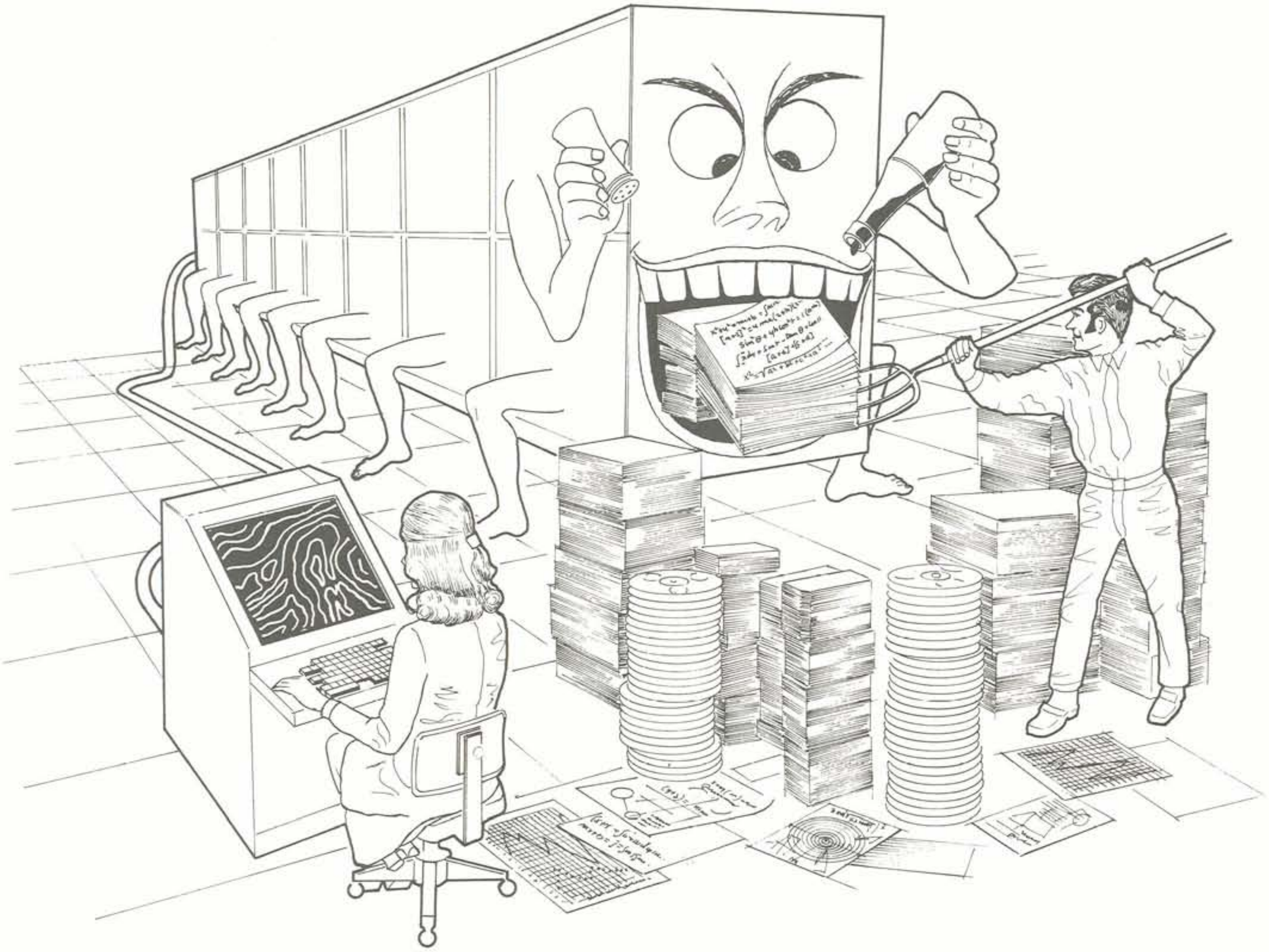
Array – cycle time	200 nanoseconds (bare memory) 312.5 nanoseconds (in system)
B 6700 – cycle time	1.2 microseconds
Parallel disk system – access time (Average access time for a single item)	19.6 milliseconds

FLOATING POINT ADD AND MULTIPLY TIMES

Floating point add time, including alignment of operands, addition, and normalization of the sum is 4.9 nanoseconds per pair of 64-bit operands, and 2.9 nanoseconds per pair of 32-bit operands. This is based on the time to execute one add instruction simultaneously on 64 pairs of 64-bit operands in 312.5 nanoseconds, and on the execution of one add instruction simultaneously on 128 pairs of 32-bit operands in 375 nanoseconds.

Floating point multiply time, including normalization of the product, is 8.8 nanoseconds per pair of 64-bit operands and 4.9 nanoseconds per pair of 32-bit operands. This is based on the time to execute one multiply instruction simultaneously on 64 pairs of 64-bit operands in 563 nanoseconds, and on executing one multiply instruction simultaneously on 128 pairs of 32-bit operands in 625 nanoseconds.

These add and multiply times assume that the operands are already in place in the registers of the PE's. This was true for 50 percent of the add instructions and for over one third of the multiply instructions in the codes surveyed. For the other 50 percent add instructions and the more than 60 percent multiply instructions, a memory access time of 375 nanoseconds must be added. Overlap in the Control Unit will hide an undetermined fraction of this memory access time.



ILLUSTRATIVE KERNELS

To illustrate ILLIAC IV's speed of operation the times required to evaluate some simple arithmetic expressions are given.

For 64 variables,
A_0 to A_{63}, find A_j^n for all j - - - - 0.563n microsecond

Find $\sum_{k=1}^{n} A_j^k$ for all j - - - - - - - - - - - - - 0.875n microsecond
(Method of nested polynomials)

MIPS

"Mips" (million instructions per second) is a measure that has been widely used to describe the computing capabilities of various machines. A figure of 200 Mips is used for the ILLIAC IV, based on a large set of assumptions which are necessary as there is no agreement in the industry as to what constitutes an instruction. These assumptions include:

- The instruction frequencies counted from 12 different codes are typical.
- The overlap in the Control Unit is 95 percent efficient and the proportion of Control Unit instructions is less than 25 percent of the total.
- The code uses the full 64 Processing Elements efficiently.

For codes using only one of the 64 Processing Elements at a time, the figure of 4.5 Mips is used.

THE EFFECT OF PARALLELISM ON THROUGHPUT

As is well known, the throughput of ILLIAC IV is problem dependent. To oversimplify, assume some fraction, x, of a problem is stubbornly serial and that the rest of the problem is perfectly parallel and fits the 64-PE width. Also assume that the problem takes a time, T_s on a hypothetical one-PE machine, then it takes a time, T_p, on the parallel machine:

$$T_p = (x + (1 - x)/64)T_s$$

Thus, if 90 percent of the problem is perfectly parallel and 10 percent is stubbornly serial, one expects a speed improvement by a factor of 8.8 on ILLIAC IV.

The above analysis is an oversimplification because, in general, portions of problems are neither perfectly parallel nor perfectly stubborn about being serial. Among the items that can be done to allow some parallelism of operation in a problem that appears to be serial are:

- Adopt some variation in the algorithm that allows parallelism. In some cases the variant algorithm will be less efficient mathematically than the stubbornly serial algorithm. However, because it allows parallelism, it causes a net gain in efficiency on the ILLIAC IV. Matrix inversion and data transfers offer examples of this sort.
- Reallocate the memory so that the variables that were all in one PE, and therefore could only be treated one at a time, are spread across many PE's.

Thus, programs that are executed on the ILLIAC IV, in comparison to the same problems being solved on a hypothetical one-PE serial machine, run with an efficiency that is a combination of many effects. For example:

- Some small portion of the problem really is stubbornly serial, and runs with no speed-up over the one-PE comparison case.
- Some larger portion of the problem runs efficiently in parallel, approaching a 64:1 speed-up over the one-PE comparison case.
- Much of the program runs in parallel on the machine, with varying degrees of efficiency. This variation is due to parallel operations that do not use 64 PE's and also to the cases in which the parallelism was bought at the price of a less efficient algorithm. For these sections of the code, the speed-up over the one-PE comparison case is much greater than 1:1 but less than 64:1.

Therefore, it can be seen that the speed-up factor available on the ILLIAC IV is dependent on the application and on the cleverness of the programmer; it will vary between 1:1 and 64:1 over a hypothetical machine with only one PE. It is also true that better speed-up is expected in practice than that predicted by a naive separation of problems into "serial" vs. "parallel".

APPLICATIONS

The power of parallel processing is most effectively applied to solving problems when they are arranged in a parallel manner. To illustrate the methods involved, two illustrations (one computational, one non-computational) are given.

FOURIER TRANSFORMS

As an analytical technique, transformation to the frequency domain has too many applications to catalog. Fairly recent developments have produced a collection of "fast Fourier transform" methods. Perhaps, the first widely read of these is from the article by Cooley and Tukey.

Fast Fourier transforms turn out to be ideal for ILLIAC IV, operating at almost full efficiency, thereby opening up all those problems amenable to solution in the frequency domain to efficient attack. These include linear differential equations, filtering, simulation and signal analysis.

The fast Fourier transform method is described as follows:

Let A_k, where k runs from 0 to 2^m-1, be the 2^m (=N) samples of some time function that we wish to transform. The transformed function is given by

$$X_j = \sum_{k=0}^{N} A_k W^{jk}$$

where $0 \leqslant j \leqslant 2^m-1$ and $W = e^{2\pi i/N}$

Fragment the indices into their individual bits, setting

$$j = j_{m-1}2^{m-1} + \ldots + j_1 2 + j_0$$

$$k = k_{m-1}2^{m-1} + \ldots + k_1 2 + k_0$$

The method is based on the observation that W^{jk} is the product of factors related to the fragmented indices, and that these individual factors (such as $W^{j_{m-1}}$ or W^{k_0}) are repeated in a regular way in the 2^{2m} values of W^{jk}. There are only 2^m of these factors.

The transformation process proceeds as follows:

The A_k, multiplied by appropriate W factors, are combined pair-wise into a vector $B_{1,k}$. The vector $B_{1,k}$ is combined pair-wise into a vector $B_{2,k}$, and so on until a vector $B_{m,k}$ is formed.

The vector $B_{m,k}$ is the desired transform X_k, except for the ordering of the index. If we reverse the order of the bits in the index k (call that k'), then we find that $B_{m,k'} = X_k$.

The pair-wise combination of the elements of each succeeding vector to form the succeeding vector involves elements of the vector that are successively closer by factors of two. Consider an example where m = 10; there are 1024 points in the prime function. The formation of the elements of the vector B_1 involves elements of A_k that are 512 samples apart. Each element of B_2 is the sum of two elements of B_1 that are 256 samples apart in the index, down to B_{10}, whose elements are the sum of neighboring elements in B_9.

The equations for the vectors $B_{q,n}$ are, writing the indices in their fragmented form;

$$B_{1,j_0,k_{m-2},\ldots,k_0} = \sum_{k_{m-1}} A_{k_{m-1},\ldots,k_0} W^{j_0 k_{m-1} 2^{m-1}}$$

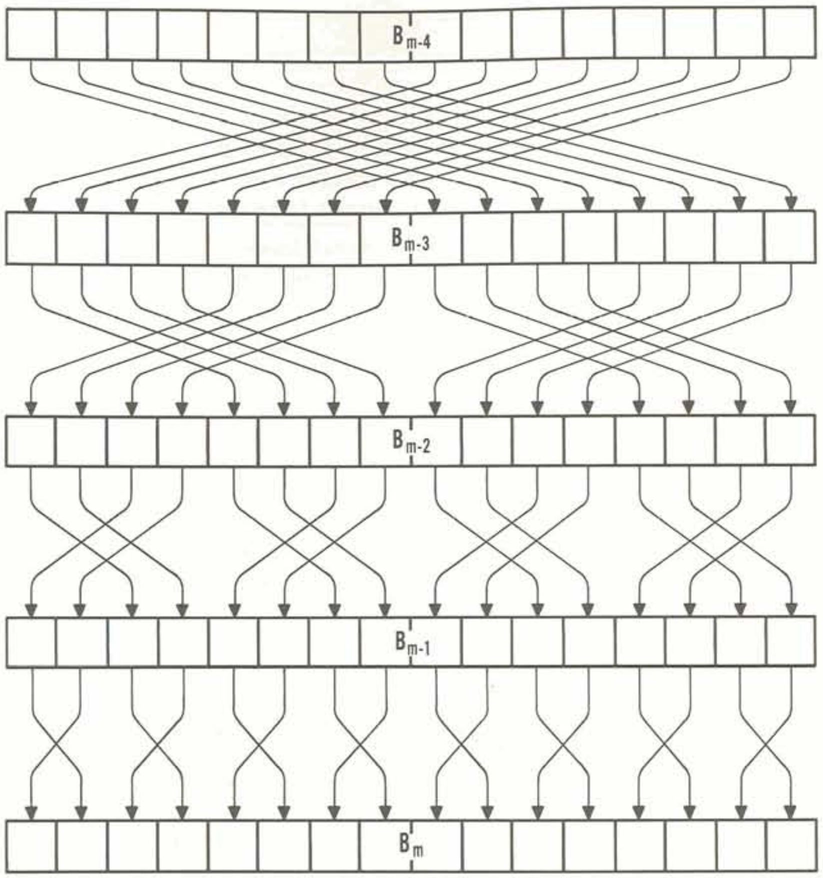
$$B_{q,j_0,\ldots,j_{q-1},k_{m-q-1},\ldots,k_0} =$$

$$\sum_{k_{m-q}} \left(B_{q-1,j_0,\ldots,j_{q-2},k_{m-q},\ldots,k_0}\right) \cdot$$

$$\left(W^{(j_{q-1}2^{q-1} + \ldots + j_0)k_{m-q}2^{m-q}}\right)$$

On ILLIAC IV, the A_k elements are divided into 64 equal pieces. For the example of 1024 time samples, there are 16 per Processing Element (PE). The first PE contains the 1st, 65th, 129th, etc. time sample; the second PE contains the 2nd, 66th, 130th, etc. time sample, and so on throughout all PE's. Computations for all but the last 5 steps are therefore carried on within the individual PE's with no interaction between PE's. For the 5th from the last step, we must swap operands between PE's that are 32 apart as indicated in the top layer of the accompanying figure. This is a simple route by distance 32. For the 4th from the last step, we must swap operands between PE's that are 16 apart as indicated in the next to the top layer of the figure. Half the operands are routed by a distance +16; half by a distance -16. The third from last step routes half the operands by distance +8, half by distance -8. This continues through distances 4 and 2 to form the results.

As described above, the results are produced correctly, but in scrambled order. Rearrangements of the data can be accomplished by fetching, routing, and storing. Approximately 160 or less route instructions are required to move the 1024 result points into their proper locations.



Data Transfer Paths, Fast Fourier Transform (16 PE Example)

TABLE LOOK-UP

Table look-up is an excellent example of the versatility of parallel processing. Investigation reveals several powerful techniques for implementing table look-up. The choice depends mainly on the behavior of the search key and the table size and regularity. Among these techniques are the following:

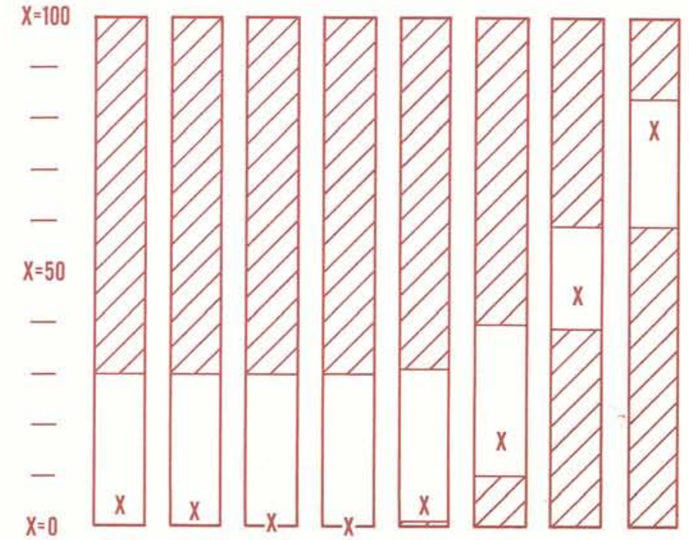
Replicate the Table – 64 copies of table. Tolerable only if the table is small.

Repeat the Table Several Times and Route the Computation – If the table were repeated eight times, for example, the first section would appear in PE's 0, 1, 2, 3, 4, 5, 6, and 7. The second section would appear in PE's 8 through 15; the third, in PE's 16 through 23 and so forth to the last section, which would appear in PE's 56 through 63. The table look-up would now be performed eight times, once for each section of the table. At each performance the routine would recognize whether it was in the proper section of the table in each PE. By the time the eight look-ups were complete, 64 values would be obtained. The actual number of repetitions for a particular problem depends on the relative size of the table and the amount of computation time available as a trade-off.

Compute – Some tables can be replaced by computation. When this is true, it is because, in a serial machine, the computation takes longer than the table look-up. In a parallel machine, the computation runs 64 times as fast, so that the trade-off may favor computation.

Interpolate More Intelligently – Use of more complicated interpolation formulas results in fewer entries being needed in a table. This is a special case of substituting more computation for less table.

Use Regularities of the Independent Variable – Suppose a large table is storing f(x), where x is the independent variable. In many problems, x will vary smoothly from one PE to the next. We keep in any given PE, say PE number "p", only that portion of the table that refers to values of f(x) for x near the value x_p contained in that PE. As x_p changes, we come to the case that $f(x_p)$ is no longer contained in the portion of the table in PE number p. Then we digress to a table rearranging routine, which, in general, will only have to move copies of values from neighboring PEs.



PE NUMBER		PE 0	PE 1	PE 2	PE 3	PE 4	PE 5	PE 6	PE 7
TABLE STORED OVER A RANGE OF X, $f(X_1)$ TO $f(X_2)$	X_1	0	0	0	0	1	10	34	59
	X_2	25	25	25	25	26	35	59	84
VALUE OF X (AT THIS STAGE OF THE COMPUTATION)		4.5	3	0.5	0.0	4.5	17	40.5	78

Illustration of Table Stored Using Regularities of the Independent Variable

The accompanying figure shows an example of such a table having 100 items with only 25 items required to be stored in each PE of the 8-PE example array. Only as much of the table as corresponds to the range of x is stored in immediately accessible memory (84 of the items in the table of the example). When x goes out of range of the immediately accessible table (above 84 in the example), more table must be brought in (Perhaps, a copy stored in array memory but in more compact form).

SYSTEM ELEMENTS

ARRAY SUBSYSTEM

The concept of the ILLIAC IV is the use of an array of separate and identical Processing Units. This architecture leads to programming flexibility that is not found in the competing architectures. For example, access time to memory is shorter and addressing memory is made flexible by the existence of a separate address in each Processing Unit. In addition the replication leads to economy of design and manufacture.

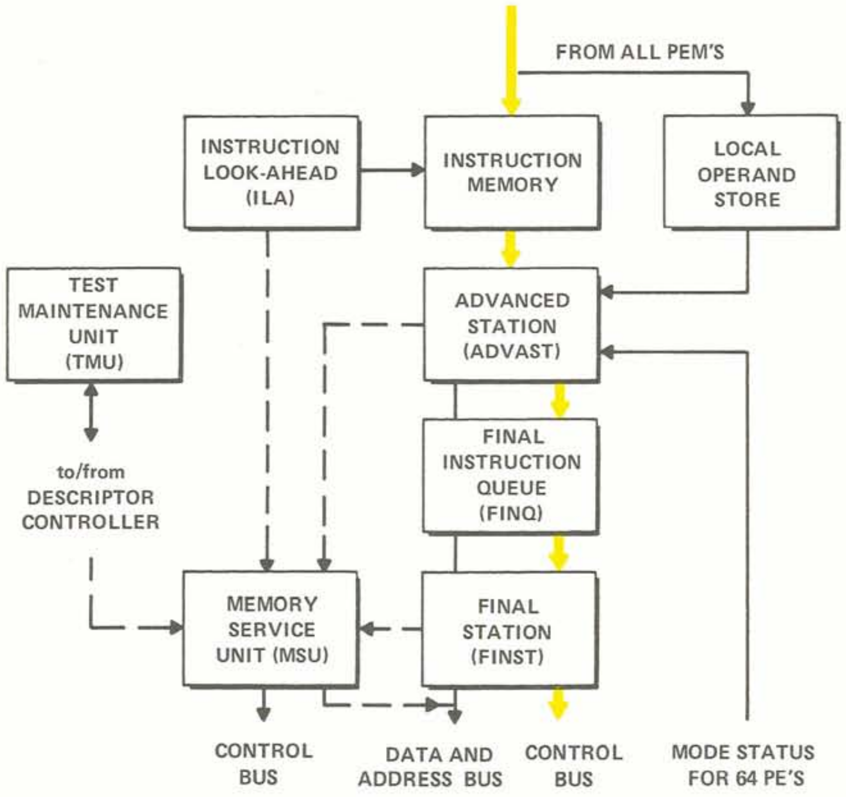
Control Unit (CU)

The Control Unit is the portion of the ILLIAC IV System that performs the initial processing of instructions up to and including the generation of detailed instruction microsequences for a step-by-step control of the Processing Elements (PE). All execution of instructions in the array is controlled by the instruction decoding stations in the CU.

The flow of instructions through the CU is shown in the figure by the green arrows that form a path vertically descending through the figure. The instruction stream enters the CU by being fetched from the array memory (PEM's) to the instruction memory that is local to the CU. Fetching is done in blocks of 16 instructions each. Management of this instruction memory is assigned to an autonomous section of the Control Unit called the Instruction Look Ahead (ILA). As the Advanced Station (ADVAST) is finished with each instruction, the ILA sends the next instruction to ADVAST for initial decoding. The ILA checks that the next instruction is in the instruction memory, and if it is not, the ILA fetches it and its whole block of 16 instructions. Since the ILA keeps track of the instructions in terms of their memory addresses, its operation is completely transparent to all programmers.

The instruction set has two general types of instructions: those used primarily to control the internal operations of the CU (ADVAST instructions) and those used primarily to control Processing Unit (PU) operations (FINST/PE instructions). The instructions that specify the CU operation (ADVAST instructions) are used for such functions as loop control and interrupt control. The FINST/PE instructions, which control the operation of the 64 PE's executing in parallel, may require some preliminary operation to be performed by the ADVAST (e.g., operations such as address arithmetic or the obtaining of a literal to append to the instruction).

When the instruction reaches ADVAST, it is decoded partially to determine which type it is, and ADVAST operations are performed on it as required. ADVAST has access to four accumulators, a 64-word local operand store, and miscellaneous registers. If the instruction concerns these registers only, it is executed entirely at ADVAST. If the instruction calls for PE operations, ADVAST may be required to do some preparation,


FROM ALL PEM'S
INSTRUCTION LOOK-AHEAD (ILA)
INSTRUCTION MEMORY
LOCAL OPERAND STORE
TEST MAINTENANCE UNIT (TMU)
ADVANCED STATION (ADVAST)
to/from DESCRIPTOR CONTROLLER
FINAL INSTRUCTION QUEUE (FINQ)
MEMORY SERVICE UNIT (MSU)
FINAL STATION (FINST)
CONTROL BUS
DATA AND ADDRESS BUS
CONTROL BUS
MODE STATUS FOR 64 PE'S

Control Unit

especially in the address/literal field of the instruction. Instructions that have been completed by ADVAST are then discarded. Those that are intended for execution by the array of PE's are passed to the Final Instruction Queue (FINQ) with the ADVAST-prepared address/literal field. This revised instruction stream is now fed to the Final Station (FINST) which issues commands to the PE's on the basis of the instructions it receives. The commands issued by the FINST cause similar operations to take place simultaneously in all the PE's.

ADVAST instructions affect nothing but ADVAST itself, and most ADVAST operations on other instructions are such that ADVAST and FINST can operate most of the time independently of one another. Consequently, the ADVAST operations can be carried on at the same time as the executions of some previous instruction in the FINST and PE's. The FINQ is composed of eight instruction storage positions that allow time-smoothing between ADVAST and FINST. This overlap between ADVAST and FINST causes the Processing Elements to be kept continuously busy, as long as the number of ADVAST instructions is not too great in any given segment of code.

An occasional instruction may require cooperation between ADVAST and FINST, or the PE's. These instructions will cause ADVAST to wait until all previous instructions are completed, at all stations, before their execution can proceed.

ADVAST executes those portions of the code which can be called housekeeping and has a number of facilities to aid it in this task. One of these is the local operand store which serves several functions. For example, it may contain index words that are designed primarily for the control of loops of instructions; it may contain numerical variables that are to be broadcast to the PE's in parallel; or it may contain "control vectors" (words containing one bit per PE) that are destined to be transmitted to the PE mode bits for on-off control of the PE's. These functions are not explicit in the hardware, but the data in the local operand store is put to explicit use only as a function of the program being executed in ADVAST. The machine language of ILLIAC IV provides a number of instructions for exercising this control. For example:

- **LOAD** – Transfer one word from array memory to the local operand store.
- **TX– –M** – Test index (greater, less or equal as specified by the letters supplied for "– –") against the limit contained with the index word, and modify the index by adding the signed increment also contained within the index word.
- **LDL** – Transfer a word from the local operand store to one of the accumulators from which preparation of the address/literal field of PE instructions takes place.
- **LDE** – Transfer each bit of the accumulator (assumed to contain a control vector) to the mode bit of the corresponding PE.
- **SETC** – Transfer the mode bit (a bit in the PE which turns it on and off) from each PE to the corresponding bit number of the accumulator.
- **LEADO** – Convert the leading ONE of the accumulator (assumed loaded by a SETC instruction) into its corresponding bit number (which is now the PE number of the successful PE).

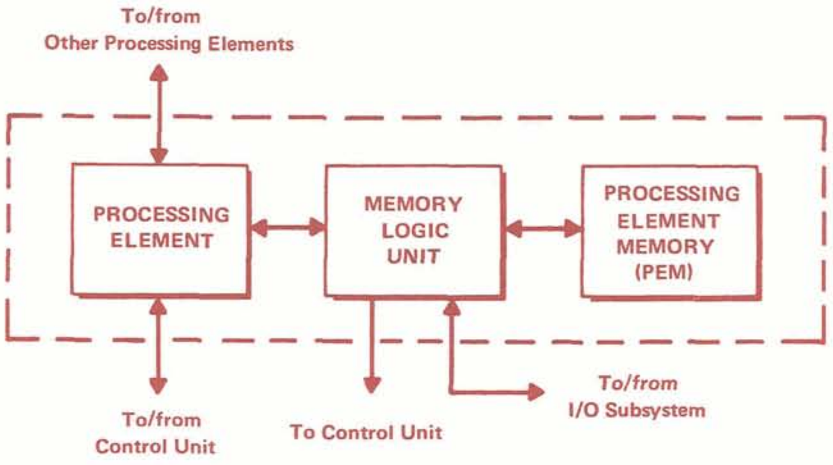
The Memory Service Unit (MSU) resolves the conflicts of the three users of array memory: I/O, FINST, and ILA. It also transmits the appropriate address to memory and exercises control over the memory cycle.

The Test Maintenance Unit (TMU) provides the control channel to the CU from the B 6700 and the manual maintenance panel.

Processing Unit

A Processing Unit (PU) functions as a general-purpose computer under the direction of the CU. All of the 64 PU's in the ILLIAC IV System are electrically, mechanically, and functionally identical. Each PU consists of a Processing Element (PE), a Memory Logic (MLU), and a Processing Element

Memory (PEM). Data and control inputs from the PE and MLU are shown below.



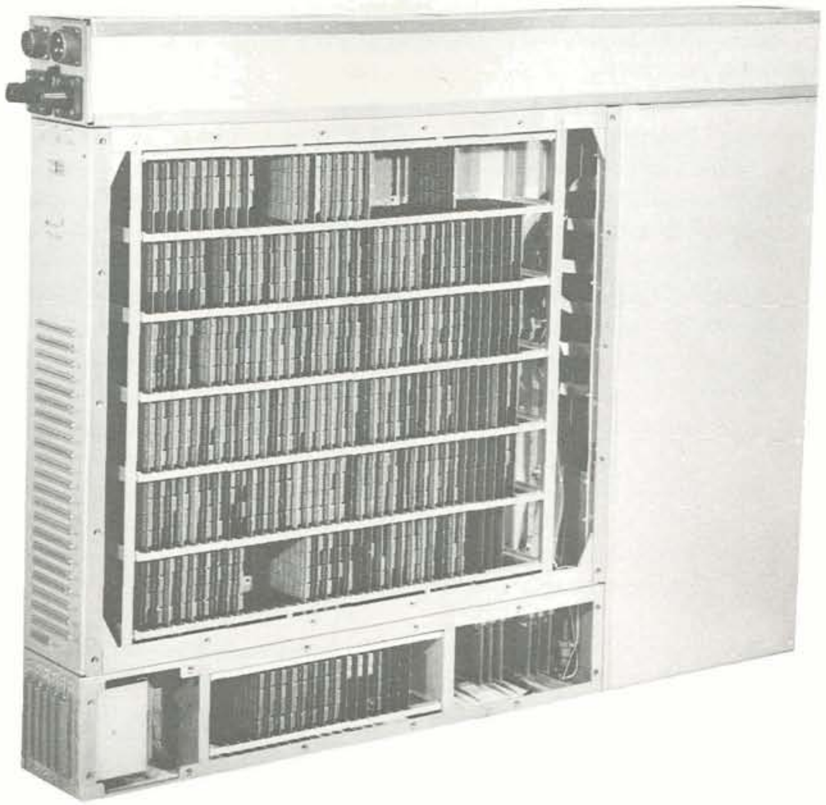
Processing Unit Data and Control Paths

For control, the PE and MLU receive enable signals from the CU for the sequential enabling of data paths and logic during instruction execution and for controlling the reading and writing in the PEM. In addition, the CU monitors and controls the status of the PE's by using one input and one output of the mode logic in each PE. Similarly, it monitors the memory protect status of the PEM's by using an output from each MLU.

Processing Element

The PE is the execution portion of the PU and is devoid of all independent control with the exception of mode, which may be set by some data dependent conditions. Mode control permits a PE to accept or ignore a broadcast control sequence from the CU, dependent on its current status. The PE is essentially a four-register arithmetic unit, as shown in the figure, capable of executing a full repertoire of instructions having 64-bit, 32 bit, or 8-bit operands. Further, operations involving 64-bit and 32-bit words can be done in either fixed-point or floating-point representation.

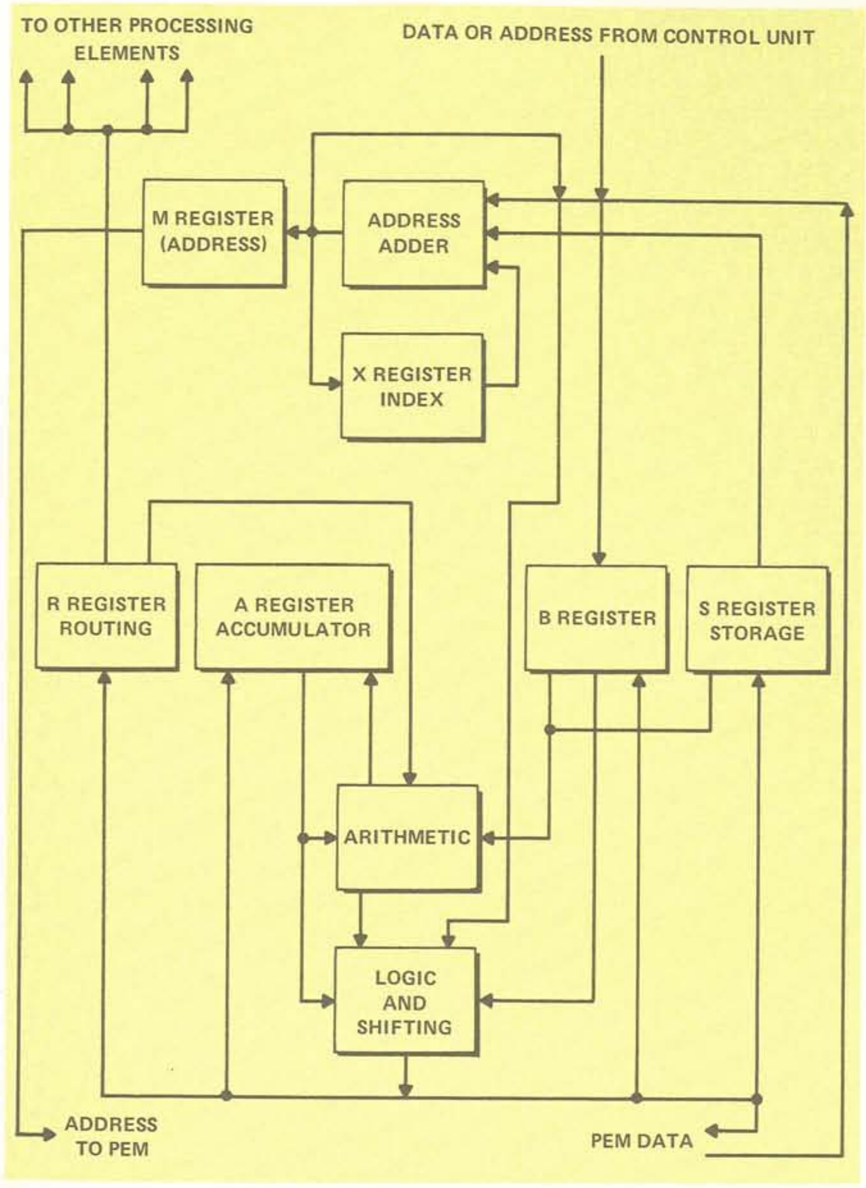
An arithmetic unit in the PE combines a carry-save adder tree and parallel adder with carry look-ahead logic to give a floating-point multiply in 9



Processing Unit

clocks and a floating-point add in 5 clocks. Both times include post-normalization.

The instruction set of the PE is that of a large-scale, general-purpose digital computer. Floating point arithmetic in both 64-bit and 32-bit words is provided, with options for rounding and normalization. Full word operations, 8-bit byte operations, operations ignoring exponents, operations using exponents only, and operations ignoring the signs are among the instructions provided in the arithmetic group. A full set of tests is generated by making all registers addressable and providing all possible comparisons. Test results are set into a mode bit which may be used to



Processing Element

direct the flow of the instruction execution. Bit manipulation, shifts and logical operations are also included in the instructions set.

Processing Element Memory

The PEM provides the high speed, random access, primary storage for the ILLIAC IV. Each PEM provides storage for 2048 64-bit words, which totals 131,072 words. To obtain the high speed necessary for array operation, the PEM has been implemented using semiconductor memory techniques. The PEM interfaces with, and is directly controlled by, the MLU.

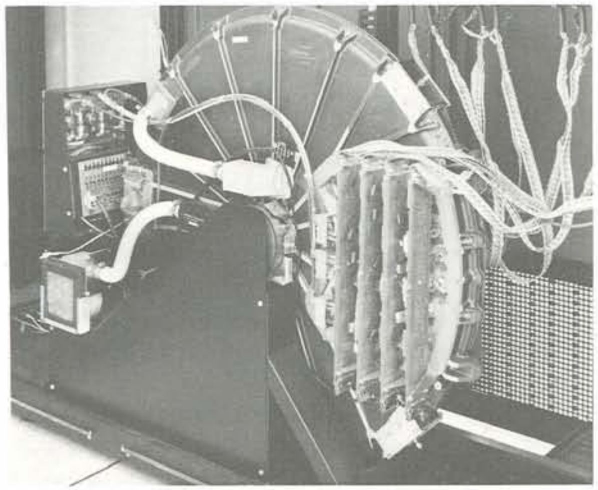
The first 128 words of each PEM can be write-protected by setting an appropriate control bit. If a write is attempted in any of the word locations 0 through 127 when the bit is set, the memory cycle will not occur.

Memory Logic Unit

The MLU controls and effects the transfer of data between the PEM, the CU, the PE, and the I/O Subsystem. The MLU also enables non-memory data transfers between the CU and the PE. In addition to the timing and control logic for PEM operations, the MLU contains a memory information register used for the temporary storage of data to be written into or read from the PEM.

DISK FILE SUBSYSTEM

The ILLIAC IV Disk File Subsystem is based upon Burroughs' continuing development in head-per-track disk files. In each Storage Unit (SU), containing one independently rotating disk, 128 of these heads are simultaneously written or read, to achieve a data rate of 502 X 10^6 bits per second. Capacity is 78,643,200 bits of data storage per Storage Unit. The disk in each Storage Unit rotates at 1530 rpm, for a rotation period of 39.2 ms, and an average access time of 19.6 ms for a single item. For multiple items, the Optimizer contained within the I/O Subsystem is capable of up to



Disk File Subsystem

24 separate disk accesses within the 39.2 ms period. Up to 16 Storage Units are serviced by a single Electronics Unit (EU). The ILLIAC IV System is designed to accept two EU's, for a combined storage capacity of 2,516,582,400 bits and a transfer of 1.004 billion bits per second. Each of the 128 tracks is split into three data lines, at one third the data rate, for interface to the Disk File Controller (DFC), so that there are 384 data lines between the EU and DFC. When the EU is not busy, these same lines are used to continually transmit the present rotational position (address) of all sixteen storage disks to the Optimizer.

I/O SUBSYSTEM

The I/O Subsystem, shown in the system diagram on the right, consists of the Descriptor Controller (DC), I/O Switch (IOS), Buffer I/O Memory (BIOM), and Disk File Controllers (DFC).

The interface between the I/O Subsystem and the B 6700 control computer is designed to take advantage of the existing properties of the B 6700, while keeping simple the interface to the ILLIAC IV array. Control words are received over the scan bus interface provided from the B 6700 processor, and results are described in words transmitted back over this same interface.

Two data paths exist between the B 6700 Subsystem and the I/O Subsystem, one path being the Buffer I/O Memory (BIOM), and the other path leading directly into the Descriptor Controller (DC). The BIOM functions as a module of B 6700 memory, as seen from the B 6700, handling data transfers from the B 6700 into the I/O Subsystem. As seen from the ILLIAC IV system diagram on the next page, the BIOM can transfer data either onto the disk file or directly into the array memory. The data path to the DC uses the 48-bit word interface of the B 6700 multiplexor, which allows the DC to share a memory bus with the multiplexor.

The DFC consists of two controllers that execute descriptors held in the DC for transfers between disk and array, disk and BIOM, BIOM and array, and real-time link and array. All transfers involving the array are via the IOS.

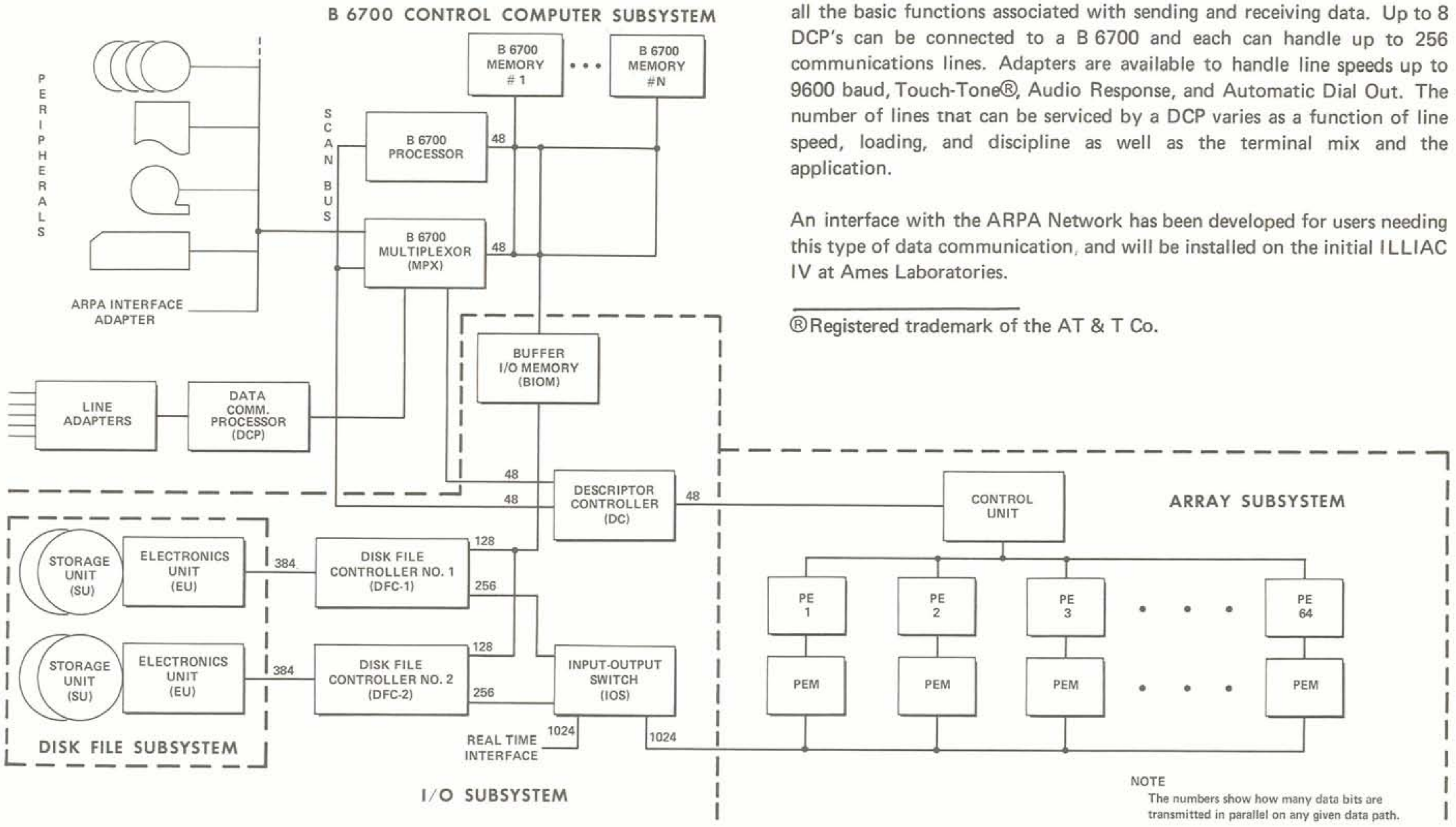
The IOS buffers and distributes data between the DFC and the ILLIAC IV array. The IOS has a 256-bit bidirectional interface with each of the two DFC units and a 1024-bit bidirectional interface with the ILLIAC IV array. The IOS also provides a 1024-bit wide external data link to the array.

The DC receives control words over the scan bus interface and fetches I/O descriptors over the multiplexor word interface in response to these control words. The DC sends result descriptors over the scan bus upon the completion of I/O transactions. Certain I/O descriptors cause the DC to send words of data, fetched over the multiplexor word interface, to the CU, where they are treated as instructions by the TMU. There is a 48-bit bidirectional interface between DC and TMU for these transfers.

B 6700 CONTROL COMPUTER SUBSYSTEM

The primary functions of the B 6700 control computer are to execute the supervisory program and prepare programs for the ILLIAC IV by assembling, compiling, etc. The supervisory program schedules jobs for the array; maintains the parallel disk files; transmits control records (descriptors) to the I/O Subsystem, which directs the I/O transactions in and out of the array; responds to interrupt conditions from the array and elsewhere; and communicates with the user and operator.

As a secondary function, the B 6700 has associated with it a full range of peripheral equipments. Consequently, the B 6700 provides the ILLIAC IV system with such peripheral capabilities as may be needed. On-line data communication may be added to the system by including a Data Communication Processor and line adapters.



ILLIAC IV SYSTEM DIAGRAM

Data Communications

Because the B 6700 is designed for multiprocessing, the system readily accommodates applications and procedures requiring data communications. The B 6350 Data Communications Processor (DCP) is the heart of the data communications network. It is a small, special-purpose computer containing the registers, logic, and translation ability to perform all the basic functions associated with sending and receiving data. Up to 8 DCP's can be connected to a B 6700 and each can handle up to 256 communications lines. Adapters are available to handle line speeds up to 9600 baud, Touch-Tone®, Audio Response, and Automatic Dial Out. The number of lines tnat can be serviced by a DCP varies as a function of line speed, loading, and discipline as well as the terminal mix and the application.

An interface with the ARPA Network has been developed for users needing this type of data communication, and will be installed on the initial ILLIAC IV at Ames Laboratories.

®Registered trademark of the AT & T Co.

DATA FLOW

The history of one program being executed in the ILLIAC IV can be tracked by tracing the sequential steps in the flow of data during the execution of a job from the gleam in the operator's eye to the production of the final printout. The steps in the data path are as follows.

Data enters the system from the peripherals of the B 6700, initially. If the data for this job is the result of output from a previous task, the data may be found in a bulk store. This data will be moved from the external source, whatever it is, to the parallel disk file before the job is executed.

For data coming from the bulk store, the data path is to the BIOM and then to the ILLIAC disk (Path 1B in the figure). For data brought in from peripheral devices, the path is shown as "1A" through the multiplexor in the figure. The B 6700 MCP controls this path. At the programmer's option, the B 6700 may perform formatting or preprocessing of the data on entrance. However, it is expected that such formatting or preprocessing can be done more expeditiously on the ILLIAC IV array, which will have the additional advantage of avoiding the potentially inefficient situation of having the array idle waiting on the B 6700.

Array processing starts by taking all the data collected on the disk and moving only that portion required by the beginning of the program into the array memory; that is, the PEMs are collectively treated as a single memory bank (path "2" in the figure). This move is under the control of the MCP in the B 6700.

To make all data transfers efficient, an Optimizer is included in the I/O Subsystem so that many independent blocks of data, which could be transferred together, can be included into a single transfer operation with a single access time, without the necessity of being combined under a single hardware address. The Optimizer controls will allow the MCP to put together as many as 15 disk file addresses into a single package, which, barring conflicts, will all be read during a single disk revolution of 40 ms.

As processing proceeds in the array, various data objects will be created in the array, and sent to disk (path "3" of the figure). These include restart information, overlayed information in the case of problems that do not fit in main memory, and intermediate results. In the present design of ILLIAC IV, the B 6700 MCP has the responsibility of disk space assignment, and such actions are initiated by an interrupt to the B 6700 MCP which issues the control words to the I/O Subsystem.

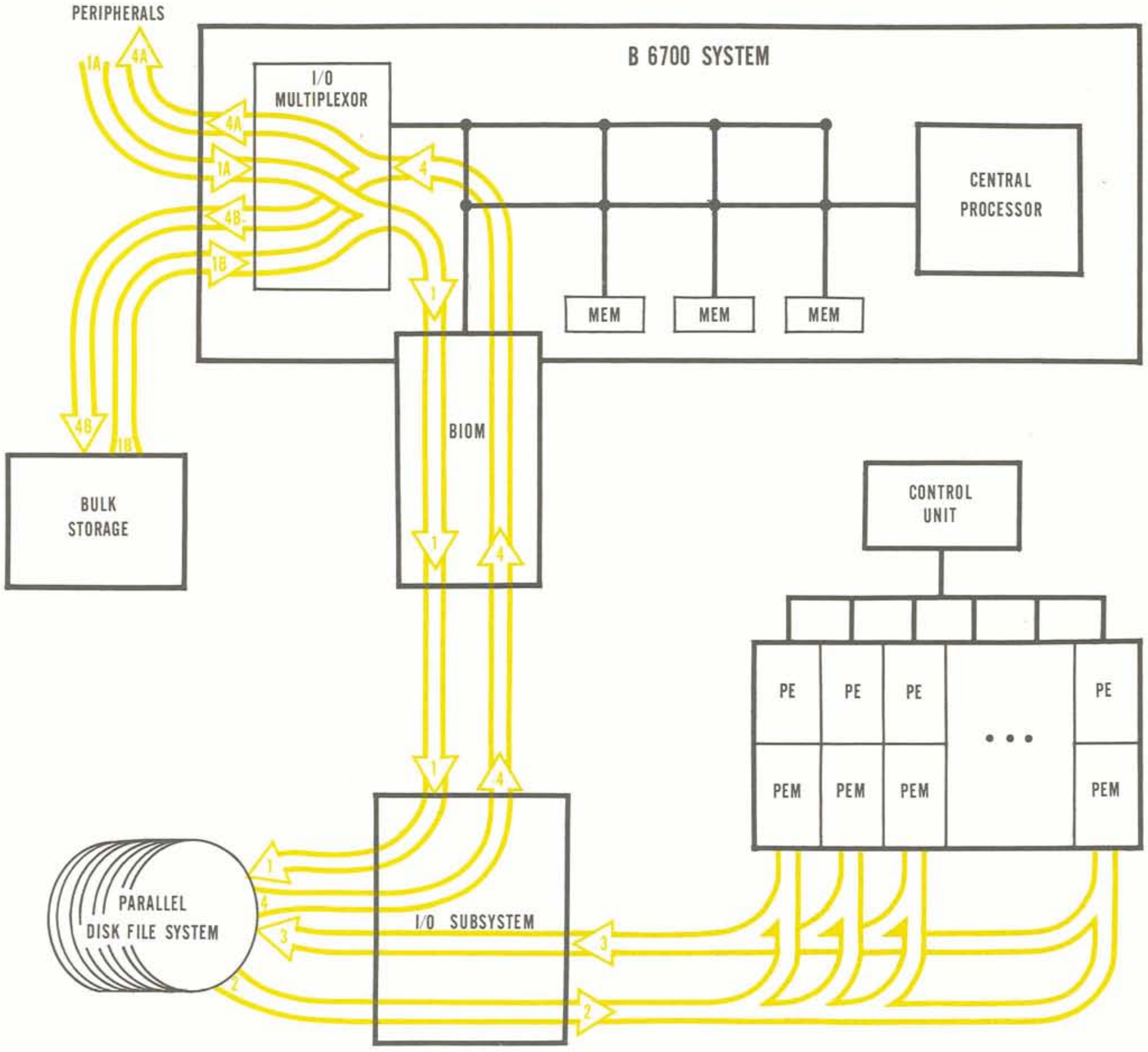
At the end of data manipulation, all results of array processing will be on disk, and outputs destined for users are transferred, via BIOM, to peripherals via the B 6700 multiplexor. Outputs destined for use by jobs for subsequent execution on the array are transferred to the bulk store. These transfers are under the control of the B 6700 MCP.

Use of the ILLIAC IV may, in some cases, involve postprocessing executed by programs in the B 6700. In general, use of the ILLIAC IV array for postprocessing is encouraged for the same reasons as for preprocessing.

The flow of both data and instructions is essentially identical up to the point of being inserted into the PEMs. The one exception is in preprocessing of data recommended for execution in the array rather than in the B 6700. The corresponding action on program instructions is compiling which will be done on the B 6700 for the program to be run on the array.

Once in the array processor, however, flow of data and instructions is different. Data generally is segregated into the individual Processing Element Memories, and is fetched to a Processing Element from its own Processing Element Memory, and restored into that Processing Element Memory. Two mechanisms break through this segregation of data into separate Processing Element Memories. The first is called "routing", the capability of the Processing Elements to shift data among themselves. The second mechanism lies in the Control Unit which sees all of the Processing Element Memories as one single memory bank. The Control Unit (a 65th, different processor, itself) can fetch any word from this memory bank. In some cases it will "broadcast" such a word of data in parallel to all Processing Elements.

Programs stored in the memory bank are seen by the Control Unit. Instructions are fetched in blocks of 16 instructions at a time, and fed, one instruction at a time, into the instruction decoding machinery of the Control Unit. Although 64 data streams can be worked on simultaneously, — one going between each Processing Element and its Processing Element Memory, only one instruction stream is being decoded (that instruction stream being fetched to the Control Unit and executed from there). Instructions are expected normally to be read-only, and to be stored in the first 6.25% of memory addresses where protection against writing is available.



Data Flow in ILLIAC IV

SOFTWARE

The ILLIAC IV software currently includes a macro-assembler, a FORTRAN compiler, an operating system, and a basic library of intrinsic functions and utility routines. A brief description of each follows:

- **ASSEMBLER** – The ILLIAC IV assembly language corresponds directly to the ILLIAC IV set of instructions. In addition, there is a set of pseudo-opcodes which handles such activities as allotting storage and loading data in the Processing Element Memories. The assembler also has numerous control options for simplifying the updating files.

- **FORTRAN COMPILER** – ILLIAC FORTRAN is an extension of ANSI FORTRAN. The additional syntax makes it possible to operate explicitly on vectors and to express the kind of parallelism exploitable on the ILLIAC IV.

  To distinguish operations or vectors from those on subscripted variables, the concept of a binary vector has been introduced. A binary vector is a string of bits; the number of bits being equal to the length of the vector with which it will be used. The bits in the binary vector can be set and reset either explicitly or implicitly by logical operations. When operations on a vector are required, a binary vector is used as a subscript. Code is emitted by the compiler so that the stated operation is performed on all those elements of the vector whose corresponding bits are set in the binary vector.

  The width of a vector is immaterial. As far as the programmer is concerned, the operation is done simultaneously on all elements specified by the binary vector.

  ILLIAC FORTRAN affords highly sophisticated matrix manipulations through its vector operations capability. The columns to be manipulated can be selected by placing a binary vector in the column subscript position. A specific row can be selected by using a scalar in the row subscript position; a different row element for each of the columns can be selected by placing an integer vector in the row subscript position. This integer vector would contain a row number corresponding to each column. Thus, for example, it is possible to operate on every other member of the diagonal of a matrix in one FORTRAN statement.

  To aid the programmer further, the compiler allows subprograms to be compiled separately and assembler code to be interspersed with the ILLIAC FORTRAN code. A wide range of file maintenance features is also included.

- **INTRINSICS AND UTILITIES** – A library of the standard intrinsic functions has been written. Included are the trigonometric functions, the natural logarithm, the exponential, and square root. They all have an accuracy of 15 to 16 octal characters. Many utility routines have been adapted for the ILLIAC IV. Among these are routines to handle Newton-Rapheson solutions, eigen-value problems, fast Fourier transforms, and Monte Carlo techniques.

- **OPERATING SYSTEM** – The operating system controls the ILLIAC IV environment. The operating system prepares a job, schedules it, moves its data and its program in and out of the array, and handles all ILLIAC disk transfers. The operating system can control the tasks for several jobs simultaneously.

ILLIAC IV is sponsored by the Advanced Research Projects Agency through Rome Air Development Center Contract USAF 30 (602)-4144 with the University of Illinois.

Burroughs Corporation
Defense, Space and Special Systems Group
Paoli, Pennsylvania 19301

Printed in U.S. America

61160