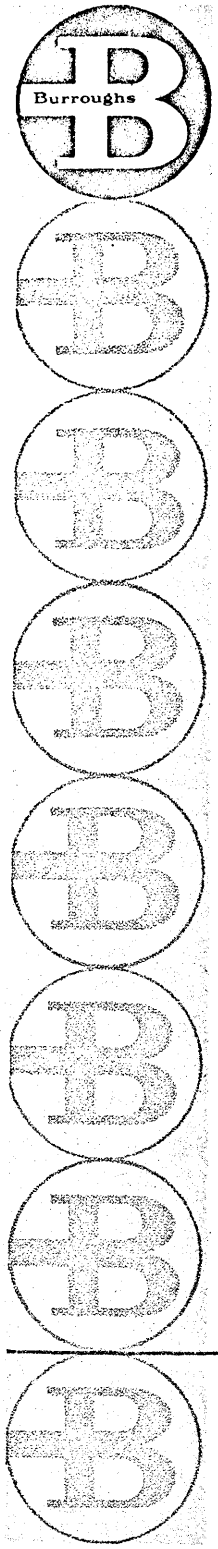
OCTOBER 13, 1966



ILLIAC IV SYSTEM STUDY

PROGRESS REPORT NO. 2

SUBMITTED TO

UNIVERSITY OF ILLINOIS

UNIVERSITY OF ILLINOIS PURCHASE ORDER NO. 09852-B

Burroughs Corporation

CONTENTS

CONTENTS (Cont.)

LIST OF ILLUSTRATIONS

Table

LIST OF ILLUSTRATIONS (Cont'd)

SECTION I

INTRODUCTION

This report is the second of three reports to be submitted during the Phase I effort of the ILLIAC IV Program. At this measured milestone in the development of the ILLIAC IV system real progress is indeed evident.

There exists now a detailed specification of the Processing Element for which a fully compatible and complete instruction repertoire has been developed. In addition, the data transfer paths establishing the links between the Input-Output and the Array, between the Control Unit and the Array, and between the elements of the Array have been specified.

Progress in defining the system hardware has also been made. A family of logic circuits has been selected and is currently undergoing an intense packaging effort. The size, type and speed of the memory system for the Array has been selected.

Progress in defining the applications areas, particularly that progress made at the University of Illinois, is especially evident.

The contents of this report in conjunction with the first report contain much of the rationale for the present system definition.

For the remainder of Phase I, much work remains. Although most of the functions and specifications of the I/O and the Control Units have been identified some additional work is needed.

In the area of packaging, both at the cabinet level and the logic level, detail design remains. Such items as power distribution, system cooling and general installation detail must be specified. Finally the entire procurement must be matched to a comprehensive program plan of schedule and delivery which is mutually acceptable .

SECTION II

THE ILLIAC IV SYSTEM

This section contains discussions of specific systems problems which are considered to be of major importance in the design of the ILLIAC IV System.

ROUTING

One of the instructions in ILLIAC IV is to transfer data residing in the PE array to new locations in that array. This is called the "routing" instruction. One version of this instruction will take the data from the nth PE and transfer it to the (n + m)th PE, where m is an indexable variant contained within the instruction, and n runs over all PE's. Disabled PE's are not to receive any new data or lose any of the data they are currently holding as a result of this instruction.

The immediate reaction to such a requirement is to implement all the required various paths by brute force. Such a solution is not only expensive, but unnecessary, and of inferior performance. To find a superior method of implementing routing, it will help to consider individual properties of the routing process. From a functional viewpoint the four properties of concern for routing are the level, the timing, the modulo, and the increment.

In the array of 256 PE's the levels of shifting with which we are concerned may be divided into shifts between quadrants, shifts between the same elements of a quadrant and shifts within a Processing Element.

The timing refers to the execution time of the shift and its concurrency with other array instructions.

The modulo is the end-around size of the shift which can be variable up to a given maximum size. For instance a 64-bit shifter may be designed to shift 64 bits end-around or eight 8-bit bytes end-around depending on the modulo control.

The increment is the smallest shifting amount of which all shifts are a multiple. Bit shifts would have an increment of 1, byte shifts would require 8, etc.

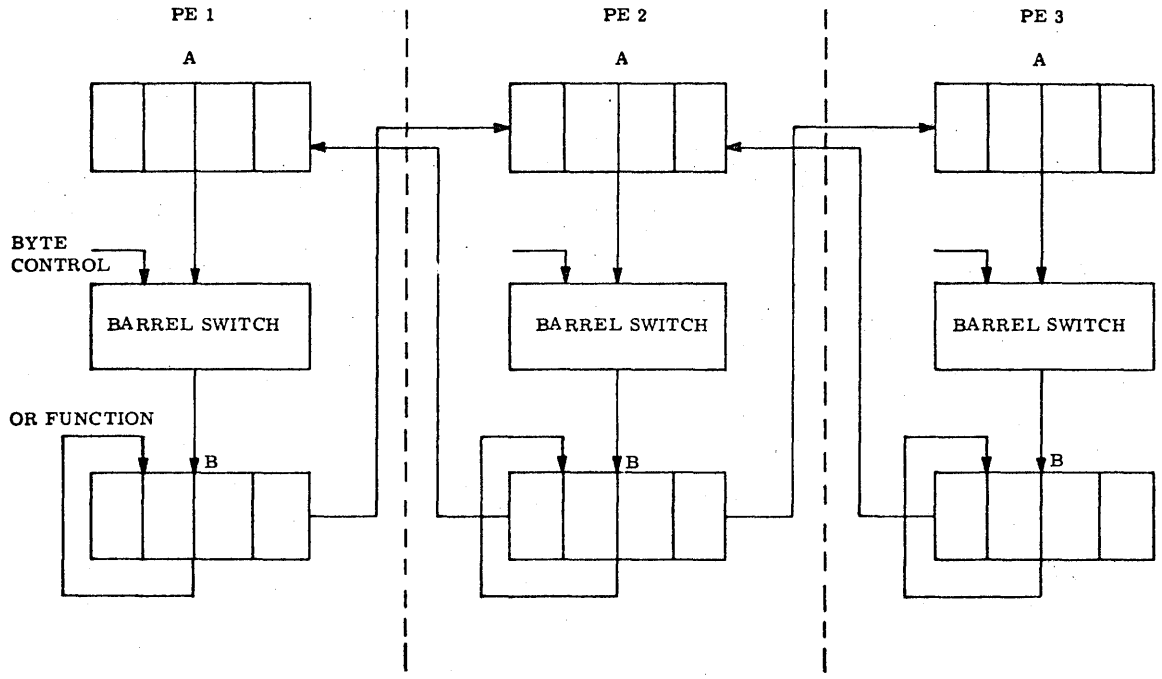


Figure 2-1. PE Level Shifting

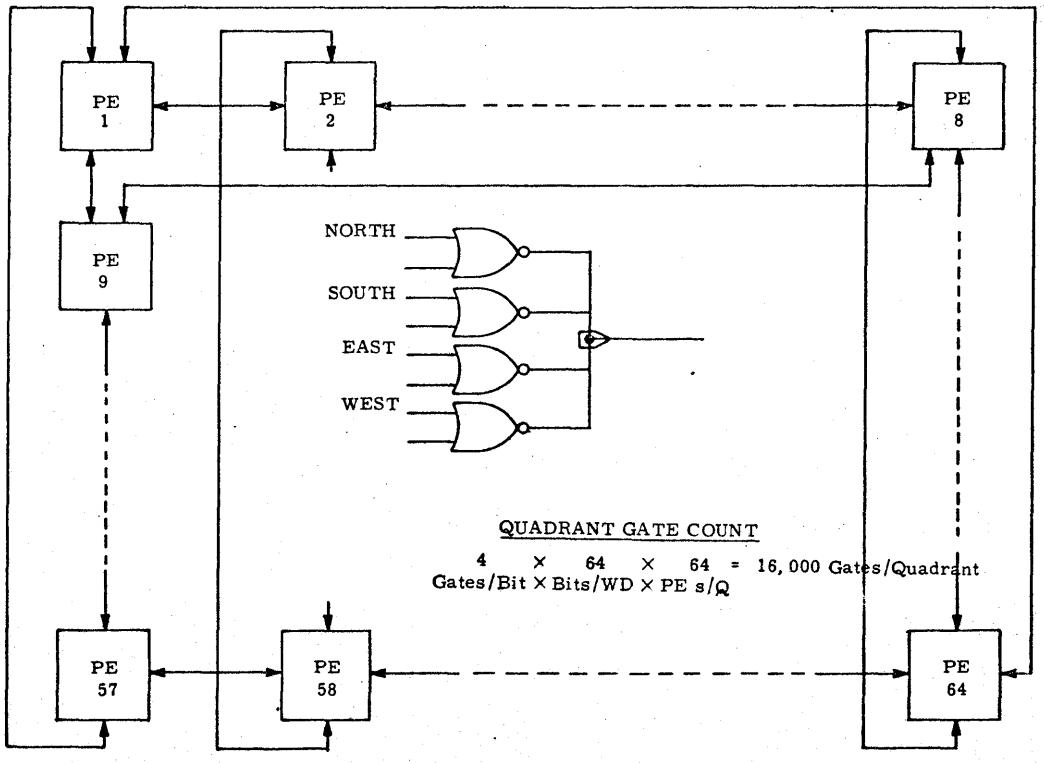


Figure 2-2. Quadrant Nearest Neighbor Connections

The variations on these properties are many, and since the function involves a very large number of bits the different variations involve substantial differences in cost.

Shifting at the PE Level

The first level of consideration is the shifting of operands within the PE. Here a barrel switch is provided which will shift in increments of 1-bit-multiples at the 64-bit-operand level.

A modulo control may be employed by adding an additional logic input to the first level of gating in the barrel switch and an additional gate to the receiving register input. Such modulo capability may include 8-, 32-, and 64-bit bytes and be extended to link PE's by a full word transfer at the end of the switching cycle (figure 2-1).

The execution time of this shifting will require one pass through the barrel switch for modulo 64 shifts, two passes through the barrel switch for end off switching, and four passes for end-around. Otherwise a transfer of Register B to the neighboring Register A will extend this capability between PE's. For the most part this logic capability exists within a PE, and there is no concurrency of execution at this level.

Shifting at the Quadrant Level

At this level of shifting, considering the variety of desirable shifting properties available, there are many possibilities. Two seemingly practical forms at the Quadrant level are nearest neighbor connections and the quadrant barrel switch. These two shifting approaches represent the practical minimum and maximum cost for the ILLIAC IV System.

NEAREST NEIGHBOR CONNECTIONS - Figure 2-2 shows the necessary data paths and gates to implement this shift approach. It has the capability of shifting left or right 4096 bits in increments of 64 and 512 bits. Increments smaller than this may be shifted with the local barrel switch if desirable since concurrency is not possible here.

All shifting must be done in sequence with all arithmetic and logic instructions; all shifted amounts are combinations of the basic two increments. Shifting of a modulo size smaller than the whole quadrant is done by mode control of the PE's.

THE QUADRANT BARREL SWITCH - The main advantage of the quadrant barrel switch is its ability to execute any shifted amount (in increments of 8) in two passes through it. Each shift can be executed concurrently with other instructions

in the PE. Modulo control is accomplished by controlling the contents of the Routing Register between partial shifts.

A COMPARISON OF THE TWO APPROACHES -

1. The two methods can produce identical results with differences in execution time and cost. Ignoring time and cost the methods are functionally equivalent.

2. There is an approximate cost difference of six to one. If we ignore the requirement for special circuits to transmit signals long distances, the simple gate count for the quadrant barrel amounts to 20% of the total system.

3. Physical distance is an important consideration. Figure 2-3 depicts a psuedo physical layout of the quadrant in which the interconnected PE for the Nearest Neighbor Connections may be reasonably close. Perhaps a maximum distance of 6 feet may be realized.

With the quadrant barrel however some paths are long. Considering a centrally placed barrel switch in the middle of a 27 foot quadrant cabinet row, one row per quadrant, the worst-case distance would be about 30 feet.

This means that every shift, however short, will involve 30 feet of cable delay plus logic. The following computation shows anticipated delay times.

Barrel Switch

Cable: 30' × 1.7 nsecs/ft. = 51.0 nsecs.

Logic: 8 gates × 3 nsecs/levels = 24.0

75.0 nsec/shift

Nearest Neighbor

Cable: 6' × 1.7 nsecs/ft. = 10.2 nsecs.

Logic: 3 gates × 3 nsecs/gate = 9.0

19.2 nsecs/shift

Average Barrel Switch Time = 75.0 nsecs.

Average Nearest Neighbor Time = 4* × 19.2 = 76.8 nsecs.

*Refer to table 2-1, page 2-5.

Table 2-1. Shift Table, Showing Shifts by One's and Shifts by Eight's

Desired Shift	Shift by One's	Shift by Eight's	Total	Desired Shift	Shift by One's	Shift by Eight's	Total
0	0	0	0	32	0	4	4
1	+1	0	1	33	+1	4	5
2	+2	0	2	34	+2	4	6
3	+3	0	3	35	+3	4	7
4	+4	0	4	36	-4	-3	7
5	-3	+1	4	37	-3	-3	6
6	-2	+1	3	38	-2	-3	5
7	-1	+1	2	39	-1	-3	4
8	0	+1	1	40	0	-3	3
9	+1	+1	2	41	+1	-3	4
10	+2	+1	3	42	+2	-3	5
11	+3	+1	4	43	+3	-3	6
12	+4	+1	5	44	-4	-2	6
13	-3	+2	5	45	-3	-2	5
14	-2	+2	4	46	-2	-2	4
15	-1	+2	3	47	-1	-2	3
16	0	+2	2	48	0	-2	2
17	+1	+2	3	49	+1	-2	3
18	+2	+2	4	50	+2	-2	4
19	+3	+2	5	51	+3	-2	5
20	+4	+2	6	52	-4	-1	5
21	-3	+3	6	53	-3	-1	4
22	-2	+3	5	54	-2	-1	3
23	-1	+3	4	55	-1	-1	2
24	0	+3	3	56	0	-1	1
25	+1	+3	4	57	+1	-1	2
26	+2	+3	5	58	+2	-1	3
27	+3	+3	6	59	+3	-1	4
28	+4	+3	7	60	-4	0	4
29	-3	4	7	61	-3	0	3
30	-2	4	6	62	-2	0	2
31	-1	4	5	63	-1	0	1

Average number of shifts = $\frac{255}{64} \doteq 4$

4. However, real problems are not random with average shifts. The data to be shifted often falls into patterns which, with proper programming, can be made to somewhat fit the available shifting patterns of the partial shifting scheme, and save time over the average time taken for transfer of randomly placed data. Some examples of such savings follow.

(a) The Cooley-Tukey algorithm for spectrum analysis in one form involves swapping data between PE's which are half an array apart at the first swap, a quarter of an array apart at the second swap, an eighth at the third, and so on. Over each 64-PE quadrant, these transfers are accomplished with four, two, and one partial shifts respectively, or an average of 2.333 partial shifts for each actual data transfer. This is considerably faster than the four partial shifts required for a random data transfer, and is faster than the data transfer effected through the all-possible-paths scheme.

(b) Another use for the data transfer paths which cover the array is in the computation of global variables ("Global" here means a variable which acquires its definition from data which covers all the PE's). Maximum, minimum, logical AND, across-word parity, sum, product, are examples of functions which one might want to perform on corresponding words in all PE's, more or less in parallel across the array, producing a one-word result. Since PE's are only capable of two-operand operations, a combination of the 64 corresponding variables from the 64 PEs of one quadrant into one resulting variable must take at least six operational steps, consisting of 32 operations on the variables by pairs, then a pairwise combining of the 16 resulting pairs, and so on. Between

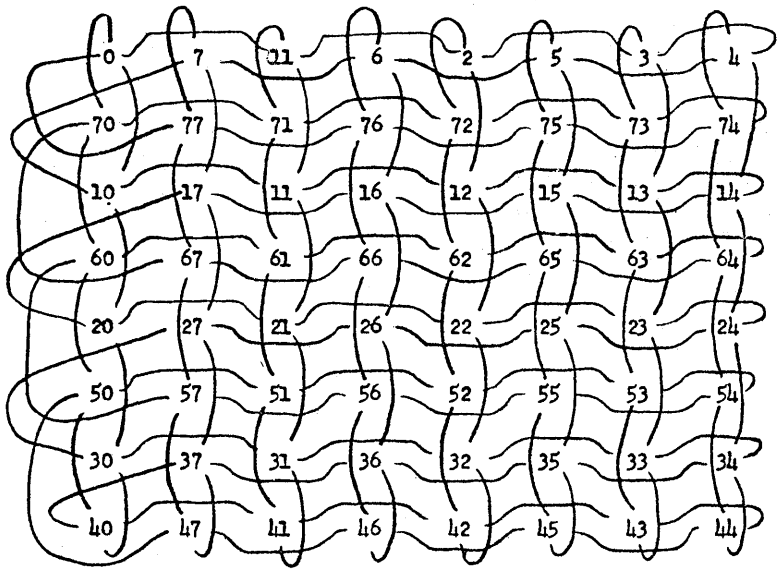


Figure 2-3. Pseudo Physical Layout of a Quadrant

0	17	1	16	2	15	3	14	4	13	5	12	6	11	7	10
360	377	361	376	362	375	363	374	364	373	365	372	366	371	367	370
20	37	21	36	22	35	23	34	24	33	25	32	26	31	27	30
340	357	341	356	342	355	343	354	344	353	345	352	346	351	347	350
40	57	41	56	42	55	43	54	44	53	45	52	46	51	47	50
320	337	321	336	322	335	323	334	324	333	325	332	326	331	327	330
60	77	61	76	62	75	63	74	64	73	65	72	66	71	67	70
300	317	301	316	302	315	303	314	304	313	305	312	306	311	307	310
100	117	101	116	102	115	103	114	104	113	105	112	106	111	107	110
260	277	261	276	262	275	263	274	264	273	265	272	266	271	267	270
120	137	121	136	122	135	123	134	124	133	125	132	126	131	127	130
240	257	241	256	242	255	243	254	244	253	245	252	246	251	247	250
140	157	141	156	142	155	143	154	144	153	145	152	146	151	147	150
220	237	221	236	222	235	223	234	224	233	225	232	226	231	227	230
160	177	161	176	162	175	163	174	164	173	165	172	166	171	167	170
200	217	201	216	202	215	203	214	204	213	205	212	206	211	207	210

Figure 2-4. "Nearest Neighbors" Arrangement of Full Array, Showing Quadrant Subarrays

each operation a transfer of data occurs to place it in position for the next operation. Thus a global function takes six binary operations and five data transfer times when implemented within the system which has all possible paths. The above description of a sequence of steps is correct even if a single instruction calls forth the whole sequence.

Global functions are in the same category as the Cooley-Tukey algorithm as far as data transfer is concerned. The data is shifted by 2 and combined with an unshifted copy, and so on up to a shift of 32, for the 64-element subarray. For the nearest neighbors scheme, an average of 2.33 partial shifts per actual shift is called for. The nearest neighbor connections would thus seem to have a slight advantage in speed over the quadrant barrel in executing global functions such as maximum, minimum, global AND, global OR, and across-word parity.

5. The conclusion is that we do not buy anything by implementing schemes which tranfer data across from one side of the array to the other with a single step. The time it takes for the data to travel across the array is sufficient to permit several logical operations. It is possible to find a set of partial data transfer paths which result in simpler mechanization and lesser wiring, and which do not degrade performance. Furthermore, by avoiding the barrel, we also avoid the necessity for allowing maximum delay on each transfer, and when the data is ordered, or partially ordered, a speedup is accomplished which would be impossible in the more expensive system.

Inter-Quadrant Transfers

Shifting between quadrants is dependent on the type of intraquadrant shifting selected, but the considerations at the interface are similar. The choice of transferring all words of a quadrant as a column (row) may be made for either case.

Here also a similar layout of PE as was done for the quadrant (figure 2-3) can make edge switching faster than quadrant switching. Figure 2-4 shows an arrangement of the whole array, 256 PE's, with the same properties for the whole array that figure 2-3 has for the 64 PE's of the single quadrant. PE's which differ in number by ±1 or ±16 from a subject PE are physical neighbors of it, just as in figure 2-3, where PE's which differ in number by ±1 or ±8 of a given PE are physical neighbors thereof. Furthermore, each quarter of the diagram of figure 2-4, as indicated by the dotted lines, is a copy of figure 2-3. To allow one to see this latter point, the PE's in figure 2-4 are numbered from 0 to 377 in octal. To translate from an entire array PE numbering, as shown in figure 2-4, to quadrant numbering, one may suppress the first and fifth bit of the binary equivalent of the octal PE number. When this is done, each quarter of figure 2-4 is like figure 2-3, with those on the right reflected about the vertical axis, and those on the bottom reflected about the horizontal axis. The lower right quadrant, for example, is like figure 2-3 both upside down and backwards.

Machine Instruction

The machine instruction to use this equipment thus requires considerable equipment to translate the instruction into the various shifts required. The number of shifts will differ, depending upon how far around a given quadrant, or whole array, the data is to be shifted. Variants of the shift instruction will be able to call upon the intraquadrant shift even when the entire array is being used. This is of advantage during one possible implementation of the Cooley-Tukey algorithm, for example. A shift of one in the 32-bit-word mode also involves a transfer of half-words between neighboring PE's, since each one is now acting as two half-word PE's.

The shift instruction therefore calls up, possibly an internal barrel shift in the PE, possibly a half-word transfer between neighboring PE's, a variable number of east-west neighbor shifts, and a variable number of north-south shifts. To preserve the contents of the A and B registers of the disabled PE's, intervening steps must avoid the A and B registers, which may be only an initial source of data, or the destination at the final shift. Either the MDR or the S register, therefore, is involved in routing, with strong preference to the MDR, since it is desirable, even though maybe not as absolutely necessary, to save the S register as well as the A and B.

THE MULTIPLY ALGORITHM

The most important single logic function from the standpoint of both performance and cost is the multiply. The emphasis placed on this instruction in its design and application singles it out for special discussion.

When first considering the many different ways to implement the multiply the ILLIAC array itself offers the first direction. There is a class of algorithms which takes advantage of the statistical nature of the ONE and ZERO trains in the multiplier. The average execution time of such a multiply is always less than a worst-case pattern of ONE and ZERO in the multiplier and, therefore, in the course of a program run, the multiply time is the average multiply time.

However, because of the lock-step synchronous operation of the Array which handles up to 256 pairs of operands simultaneously, the average execution time becomes the worst-case time. Such methods therefore are not applicable to the ILLIAC system.

Following the above, the next decision to be made is how many bits of the multiplier are to be examined and disposed of simultaneously during a single step in the cyclic multiply sequence. Apart from circuit speeds, this is the single basis for determining the speed of the multiply.

Two separate, yet interdependent, techniques, are available to do this. One technique is to encode fields of the multiplier into a larger number base*; and the second technique is to combine manifold summands selected by the conditions of the multiplier bits. In practice these two techniques are combined into a single comprehensive design. The following general relation is useful in determining the size of the partial multiplier:

$$\text{Execution time: } \left(\frac{MB}{XB}+3\right) \quad G \cdot \Delta_t$$

Where:

MB = Number of bits in the mantissia

XB = Number of bits in the partial multiplier

G = Number of gates in typical delay chain

Δ_t = Nominal gate delay including loading, wire length, etc.

Reasonable values for the above parameter are:

MB = 48 bits

G = 10 gates selected as typical PE logic chain

Δ_t = 3 nanoseconds/gate

Execution time = memory cycle time = 250 nanoseconds.

Therefore: XB = 8 bits.

Table 2-2 shows the relative speed-cost relation for the range of possible partial multiply sizes. The important feature in this table is the gate count differences for the different selected sizes. In terms of a 10K-gate Processing Element the 8-bit selection appears to be a reasonable maximum hardware investment.

*MacSorley, O. L., "High-Speed Arithmetic in Binary Computer." Proceedings of the IRE, pp. 67-71, January 1961.

Wallace, C. S., "A Suggestion for a Fast Multiplier, "IEEE Transactions on Electronic Computers, V. EC-13, February 1964.

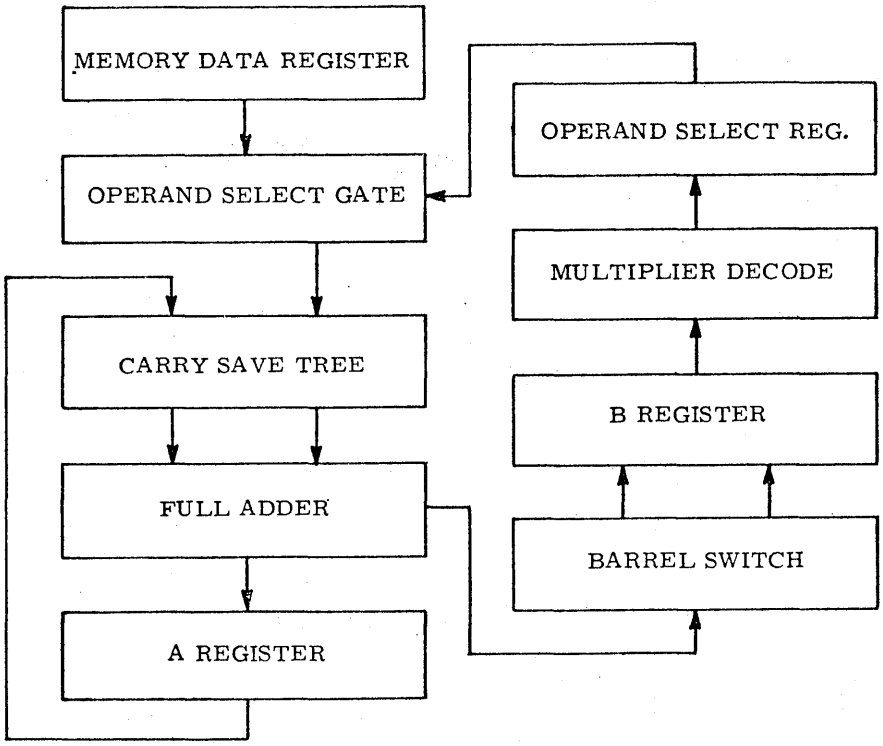


Figure 2-5. Logic Elements Involved in Multiply Algorithm

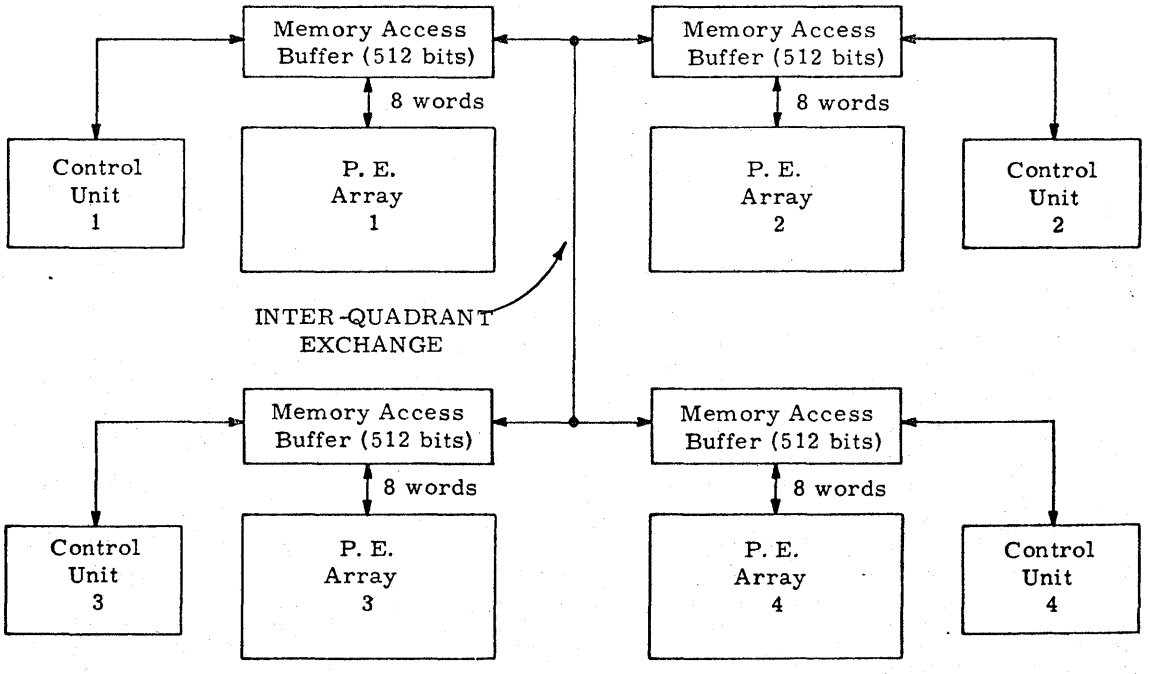


Figure 2-6. System for Handling Data From Control Unit to Processing Element, Block Diagram

Table 2-2. Relationship of Gate Size to Multiplier Size

Number of Partial Multiplier Bits	Logic Gate Count for Multiply Only	Estimated Execution Time (nanoseconds)
6	2,000	330
8	2,800	270
12	4,800	210

Having decided what constitutes a reasonable number of multiplier bits to handle in a single cycle (in this case 8 bits), the next consideration is to determine how to maintain a 10-gate delay for the worst-case logic chain. To evaluate this criteria the following substeps in the multiply must be completed within a 10-gate chain:

1. Decode the partial multiplier.
2. Add the next summand to the partial product.
3. Shift the multiplier and the partial product.
4. Normalize the result.

Figure 2-5 is a block diagram of the logic elements involved in the execution of this algorithm. The delay chain involved is the time to go from register to register.

The multiply algorithm selected first decodes the 8 bits of the multiplier into a base-4 representation into which 0, +1, -1, and +2 values of the operand are selected. This decode is stored in the Operand Select Register (OSR). The OSR is 12 bits long for storing the fully decoded four conditions for each of four operands.

Because the 8 bits of the multiplier are taken as bit pairs, four summands must be added to the partial product in a single cycle. The carry save tree, which contains three full length carry save adders (2 less than the total number of summands) combines four summands plus the partial product into a new sum and carry value. The full adder now combines this sum and carry into a new partial product.

In terms of worst-case delay, the logic chain through the operand select gate, the carry save tree, and the full adder represents the worst-case delay.

Detailed logic design analysis has shown that a nominal 30-nanosecond delay through the chains is possible. A breadboard of the actual hardware must be built and operated to obtain a true final result.

EIGHT-BIT WORD LENGTHS

The ILLIAC IV is to be built with several different word lengths. Each 64-bit PE is capable of being split, effectively, into at least two 32-bit PE's. Another desirable word-length breakpoint is at eight 8-bit effective PE's within any given PE. To evaluate this possibility we require, first, to know what it costs to expand PE capability to handle eight independent 8-bit words, and second, we need to estimate the worth of the extra capability produced by such an expansion over and above the capability already inherent in the 64-bit PE for handling 8-bit pieces. The conclusion is that all logical operations, probably even add and subtract, are easily programmed for the 64-bit PE so that they make effective use of the parallelism of the 64-bit machine. The only feature that is lacking is the independent mode control of the 8-bit sections of the 64-bit PE.

The price for 8-bit words is mainly in the fragmentation of the controls which results. Existing circuit design contemplates a buffer capable of driving 24 loads. With independent mode control on every 8-bit slice, each 8 bits of a 64-gate transfer command would have to be independently controlled, thus requiring 8 gates instead of the 3 gates required by straightforward implementation of the 64-bit PE. Since there are estimated to be 150 command lines, and the 64-gate transfer command is typical of them, we estimate that 750 gates per PE are added by the fragmentation of the command lines. In addition, the barrel controls multiply, from the three control signals per level required by only one word size, to 24 control signals per level. However, only two levels of control are used in the module eight barrel, so that 42 additional gates are added from this account.

The extra mode register bits for 8-bit operation imply extra lines to the control unit, and extra drivers and receivers for them Twenty-eight such bits per PE will require 28 drivers and 28 receivers in the PE itself, and 3,584 extra drivers and receivers in the control unit. The design difficulties of cable bundles of this bulkiness are considerable at the speeds under consideration.

The 8-bit operation contemplated here assumes we still have no more than one index register per PE. To get eight 8-bit words out of independently specified memory addresses requires eight memory accesses, a situation which is exactly the same as though 8-bit fields were programmatically extracted from64-bit words in 64-bit operation.

The most promising design of the memory currently contemplated for the ILLIAC IV, representing the best compromise between cost and cycle time, is a nondestructively read memory. Writing a small field, such as 8 bits, into a memory word will require two memory cycles, one to read those portions of the word which are not to be changed, and one to write back the word, one 8-bit field of which has been changed. A store instruction in 8-bit operation, if independent addressing is called for, on each 8-bit word, will require, therefore, 16 memory cycles.

The above hardware operations are hardly better than programmatic implementation in 64-bit mode. Apparently, the chief virtue of implementing 8-bit operations in the PE hardware is that mode register control over the individual operations can be achieved.

All the above circuitry amounts to an additional 800 gates in the PE for 8-bit operation. Probably 1000 gates is a better estimate. From this we conclude that adding 8-bit operation to the 64-bit PE adds about 10% to the equivalent gate count, and presumably also adds 10% to the estimated cost of the PE exclusive of memory.

In addition to the direct cost, there is degradation of the performance in normal 64-bit mode. Were the equipment packaged as a solid volume, lengths would be increased by 3.2% as a result of the 10% increase in components. In a well-matched system, equally sensitive to memory and logic speeds, the result would be a net 1.6% slowdown. Power and cooling requirements also follow the component count.

The opposite question is to enquire to what extent can 8-bit operations be carried on in parallel in a 64-bit PE without the assistance of specific 8-bit operations in the hardware.

Certain operations, which are expected to be frequent operations in 8-bit programming, are independent of word length, such as all bit-by-bit Boolean operations, compare all words against zero, set to specified value, read from memory, store to memory, and others.

Certain operations are programmable in such a way as to make use of much of the parallelism of the 64-bit machine even though the entities being manipulated are only 8 bits long. Add, subtract, and routing, are examples of this class of operation. Add, for example, on words of 7 bits plus sign each, can be handled by removing the sign bit and using the resulting space for overflow control, thus allowing as many adds in parallel as the adder is long. Routing can be partially implemented on all eight 8-bit words at once by means of the barrel. End-around shifts can be implemented by two shifts, which are then partially blanked in accordance with a mask which could be held in the S register, and the two shifted words then ORed together.

The only operations of any signifigance which are not included in the above paragraph are either things such as multiply or independent indexibility, which were never intended for 8-bit operation because of their expense, and mode control.

To program mode control, without actually having it in the hardware, requires manufacturing masks, which blank out whole 8-bit segments of the 64-bit word, in response to decision bits which usually will show up in the sign-bit position of the 8-bit words.

This programming, while facilitated by the one-clock-time barrel, by the complete set of Boolean operations, and by the fast single-bit instructions, will not be trivial.

COMMON DATA PATHS

This section describes the various paths by which information is transferred between the PE and its memory both to the input-output interface, and to the control unit. As a result of the conflicting requirements on transfer rates, frequency of use, and destination, there are three segments of the equipment provided for data transfer. These segments are an input-output buffer, a memory access buffer with each control unit, and direct lines from the control unit to and from the PE. The rest of this section describes this hardware in terms of its use in each class of signals which must be handled into and out of each PE.

Input-Output Buffering

Present plans are to provide an input-output buffer of 4096 bits for each quadrant of the array. Each bit of the I/O buffer has connection to one bit of the memory data register of one PE. When data is to be transferred to or from a PE memory from the I/O buffer, one memory cycle is taken from PE operations, and all 64 PE's insert one data word into their memory at that memory cycle.

Data is transferred across the external interface of the I/O buffer in word sizes appropriate to the device found at that interface. The external device may be a buffer memory with very long words, say 4096 bits each. In this case, I/O operations are accomplished by stealing one PE memory cycle from the array for each cycle of the external buffer memory. On the other hand, the external device could conceivably be a device of lesser word size, 512/bits per word, for example, in which case an independent loading and unloading control is required to assemble the shorter external words into the 4096-bit word in the I/O buffer. This independent control communicates with the array control unit both to steal memory cycles, and to report I/O complete when a whole block of data has been successfully transferred.

The design of this interface is almost independent of the rest of ILLIAC IV.

It is not the intent of this section of this report to discuss the design of the independent I/O control unit, as this design is dependent upon matters discussed elsewhere in this report.

From Control Unit to Processing Element

Data for the use of the PE, data to be stored in the PE memory for the control unit's own use, and commands for the PE are transmitted from the control unit to the processing element. Common data lines from the control unit are used for data which goes from the control unit to the PE, whether this data is for PE use or for the control unit's private use. To avoid continually interrupting PE operations for data fetches and stores which are related to control unit purposes, many words should be fetched and stored in parallel. A reasonable compromise between amount of hardware and interference with PE operation appears to be no more than 32 words of data transmitted to the PE array in parallel. Further discussion of the buffer size is found below. We plan to provide 32 words of buffering. Of these 32 words, eight are assigned to each quadrant. A block diagram of the system for handling data is shown in figure 2-6 (page 2-10).

It is simplest to describe operation by starting with the isolated quadrant. Eight words of data are transmitted to the memory access buffer from the control unit. This transmission is overlapped with PE operations provided that the program is such that the requirement for transmitting data can be recognized far enough ahead of time. This will generally always be recognizable ahead of time when the data is simply to be stored in memory. One clock time per word is expected to be required, since the data paths within the control unit are expected to be typically one word wide. Each word in the memory access buffer has access to a column in the array. The eight words have simultaneous access to one element in every column, namely a row. As a result, the eight words can be transmitted in parallel to every row in the array. If data is to be stored in memory, one row accepts the data.

If data is being broadcast, rows receive the data. In this case, the eight words of data are eight copies of a single word which originated in the control unit.

When the array is operating as a coherent whole, all four control units operate on the same program string and data in parallel, and have identical internal states. Out of a package of 32 words to be sent, the first eight can be derived from information supplied by the first control unit, the second eight can be derived from information supplied by the second control unit, and so on. Likewise, when broadcasting data, the four control units will have four identical copies of the word to be broadcast, each of the four is copied eightfold into the memory access buffers, and each of the resulting 32 copies is then used to drive eight PE's.

Also transmitted from the control unit to all PE's in parallel are control signals. These control signals, are identical for all PE's. Retiming considerations will demand that there be a flip-flop in the control unit for each such line. Present planning calls for a receiver per cabinet for each such signal, and eight PE's per cabinet.

Addresses must also be issued from the control unit to all PE's. These will use the least significant 12 bits of each word in the memory access register, and the broadcast data bus.

Issuing of addresses and broadcasting of data require a fixed delay of several clock times. The design of the microsequences as manufactured by the control unit is such as to allow for this constant delay.

From Processing Element to Control Unit

When data is fetched from PE to control unit in the isolated quadrant, the mechanism is the reverse of the process described above for disseminating data. Namely, the information is collected in the form of a single word from each single column of the quadrant, and the eight words from each row are then transmitted to the control unit. For the instruction "store to broadcast register," the eight words are ORed together to form a single word in the control unit, very like a broadcast in reverse.

When data, or program string, are being fetched in 32-word packages from the united array an additional complication sets in, since each one of the memory access buffers contains only a quarter (eight words) of the 32-word package and each control units wants all 32 words. In this case it will be necessary to transfer data around among the memory access buffers until each quarter package of eight words has shown up once in each of the eight-word memory access buffers.

Single word fetching makes use of the same paths by disabling all but one of the words being received.

Also entering the control unit are lines from each of the mode register flip-flops in the PE's. To the control unit of one quadrant, a bit in the mode register appears as a 64-bit register to the control unit, which may be set, read, compared with other registers available to the control unit, etc. When operating in united mode, control units must cooperate in the sensing of the state or mode registers. For example, a "jump on any bit equal to ONE" means to jump if any bit in any one of the four 64-bit registers involved is ONE. Since all four control units run the same program instructions at the same time, only twelve wires, one to communicate between each pair of control units,are required to secure the necessary cooperation.

Evaluation

Some of the above described data-transferring procedures take more time than one might at first expect. One 250-nsec. memory cycle is required to load the memory access buffer. This memory cycle interferes with the action of the PE only insofar as the PE memory or memory data buffer is required.

The memory access buffer for a single quadrant can be unloaded into the appropriate area in program lookahead or local data buffer in eight clock times. However, these potentially interfere with other used of program lookahead or local data buffer. With proper implementation of the controls, these eight clock times can be largely hidden by being taken from otherwise idle time in the two buffer areas in the control unit.

When the array is in united operation, one must count not only 32 clock times rather than 8, one also must transfer the data from one memory access buffer to another.

If cables of up to 30 or 40 feet long separate the memory access buffers, then the time to transfer data from one memory access buffer to another may well be three clock times, and three transfers are needed. The conclusion is that as much as 41 clock times are needed to transfer the 32 words, read in one memory cycle time, into the appropriate areas of the four control units. Seven clock cycles in each memory cycle is a likely design choice. In this case, 41 clock cycles represents 1.46 microseconds. This 1.46 microseconds is overlapped with other operations as long as a reservoir of instructions for PE operation can be maintained in the control unit.

However, this 1.46 microseconds often gets in the way when loading the program lookahead. It is a penalty to be taken whenever the program jumps to a location not contained in program lookahead. Further, the coarser block size means that loops between 32 and 64 words long will less often be found within the program lookahead, and program fetching will take place more often when the block which is fetched is larger. Large block size also interferes with broadcast operations, since a larger delay occurs between the instruction to fetch a block of data to the data buffer and the first opportunity to broadcast one of those words, when blocks are larger.

Optimum block size is that which finds the best tradeoff between interference with the operation of the PE's in the array, and slowing of operations in the control unit. Block sizes of 8, 16, and 32 words are easily available with minor modifications of the structure here described, and a choice of a smaller block size than the 32 words here proposed can be easily made during the early, design months of the next contract.

MODE

Introductory Considerations

Each PE is supplied with a mode register, which it can change as the result of tests, and which the control unit can set at will. Instructions will be executed or not as a result of the setting of the mode register. This arrangement is in lieu of branching at the PE level, since all PE's must share the common instruction stream, and therefore cannot independently execute transfers of control.

Modes should be remembered and recoverable. At the very least, each routine which one enters must be supplied with fresh capability for setting and changing its mode, while remembering the mode of the calling sequence. Actually, it seems that considerably more flexibility is desirable. In the limit, one could specify a particular mode register, out of some set of mode registers, for each instruction.

The Problem

At issue is the question of the location of the backup storage of the noncurrent modes. The choice is between supplying back-up mode register storage in the PE's, or of supplying the control unit with the capability of reading, storing, and restoring the PE modes. The implementation of mode control is considerably different, depending on the results of this choice, so that we are really choosing between two different systems for controlling PE operations in response to mode.

Back-Up Storage In The Control Unit

The first system to discuss is that with back-up storage for other than current modes in the control unit. In this system, only one mode register's worth of flip-flops are in the PE, and the setting of the mode register, for which a given instruction will be executed in a specific PE is known at the time of that instruction. Either four bits accompany the instruction, specifying which of the possible modes permit the execution of the instruction, or else a pre-existing decision, based on the content of the mode register, controls the execution or nonexecution of instructions in the PE. Bit economy in the instruction stream favors the latter, if mode is not to change at every instruction. A decision based on speed also favors separation of the mode decision from the execution of the instruction, since then less control gating is involved for the individual instruction at the PE.

One of the savings in speed of the ILLIAC IV type of computer ought to lie in the fact that the subcommand matrix of a normal computer finds itself mostly in the control unit, so that the instruction decoding and timing which consumes one clock time per instruction in most computers can be spent in the control unit, overlapped with useful arithmetic work in the PE. For most complete overlap, the PE has an "on-off" flip-flop to control the execution or nonexecution of the next instruction. Individual instructions in such a scheme will never wait for the decoding of mode information before they can start and noninterfering microsequences can freely overlap. Occasionally a one-clock-time instruction would be needed to change the setting of the "on-off" flip-flop in response to some new interpretation of the mode register.

The "on-off" flip-flop also must respond (sometimes) to arithmetic overflow as well as to programmed tests which change the mode bits. There is an overflow flip-flop which appears to the control unit much like third mode bit.

In this system there are therefore four flip-flops per effective PE with mode-like functions. Thus there are four flip-flops per 32-bit word, or eight flip-flops per actual PE, two programmatically specifiable mode bits, the overflow flip-flop, and the "enable-disable" flip-flop. There are PE instructions to set or reset each mode bit in response to programmed tests. Each bit in the PE appears to the control unit as a 64-bit word, since there are 64 PE's per control unit. There are instructions to read, save, and set these words, the instructions being control unit instructions rather than PE instructions. The control unit is provided with high-speed register storage for such saving. It is also provided with logical instructions to manipulate the modes, namely AND, OR, and COMPLEMENT instructions which can operate on the words formed from the mode register bits. Jump instructions in the control unit would test mode bits (either "any mode bit" or "all mode bits"). They would also test old modes stored in the control unit.

This system thus has a reservoir of old modes which can be reactivated on short notice. This back-up is in the control unit. Response to old mode settings is effected in one clock time by setting the "on-off" flip-flop in each PE. The instruction stream has no mode field per instruction, but does have mode-manipulating instructions, which are decoded by the control unit in parallel with the issuing of instructions to the PE.

Storage of old modes in the control unit is backed up further by the storage of words from old mode registers back to memory. It is assumed that the control unit has some means of addressing memory, both for read and write.

Back-Up Storage In The Processing Element

The second system we discuss is that with back-up storage of old modes assigned to the PE's. In this system, where several modes are stored in the PE, a method must be chosen for choosing the applicable mode for any given stream of instructions. A settable pointer could be used, and the pointer setting changed on command from the control unit, in between actual processing instructions. Using the pointer, the operation would be fully equivalent to operation with back-up storage in the control unit except for the location of the stored bits. When a mode register address is issued with each instruction, more flexibility is obtained. As described to us, the system was used with six mode bits with each instruction: Two bits of mode register address, and four bits to interpret the contents of that register.

Arithmetic overflows are lost in the implementation unless tested for immediately by means of a jump instruction in the control unit. They cannot influence mode directly, although, clearly, when tested, they can be used to control modes. Even with back-up storage of modes in the PE's, some means of setting new modes under control unit control is required. Whether a path directly from control unit to PE for loading the mode registers is needed, or whether indirect methods suffice, has not yet been determined.

Each instruction in the PE must therefore look to the mode register for conditions against which execution is made conditional. Some instructions are made one clock time longer becuase of this. Furthermore, no overlapping of non-interfering microsequences appears possible.

Jump instructions are made dependent on a bit, one per PE, returned to the control unit. These instructions are, like any other, conditional on one of the mode registers, so that jumps can easily be made conditional on mode register settings. This capability is similar to that achieved in the competing system by testing old modes in the control unit.

The system has a limited reservoir of old modes which can be reactivated on short notice. This reservoir is limited in length, but very fast of access. The instruction stream has a 6-bit mode field per instruction.

Storage of additional modes is in PE memory. If they are to be packed effcient-ly in memory, a short mode-packing routine is necessary.

Functional Capability

Both competing systems can execute the same functions. A partial list of capabilities follows:

- Quick change of control from one mode register to another.
- Storage of several different mode registers per PE.
- Transfer of control possible in response to mode register setting.

Comparison

A list of features in which the suggested methods of supplying back-up mode registers differs reveals none in which the functional capabilities made available by the one cannot also be made available by the other. However, there are other differences.

When back-up mode storage is in the PE every microsequence requires the operation of mode gating in the PE, and overlapping of microsequences is not possible. Six bits of every instruction are expended on mode information.

When back-up mode storage is in the control unit, every change of mode requires a one t-time operation interleaved between the regular arithmetic operations. Each change of mode takes a short instruction, say 16 bits.

What is needed, to compare the advantages and disadvantages listed above, is some estimate of the number of instructions, on the average, executed without change of mode setting, or executed while ignoring mode. Even if average,

the string length is only two or three. The speed advantage would appear to lie with the system of storing modes in the CU.

Another difference is in the PE hardware. Considerable savings are expected by placing the back-up mode storage in the CU, whose long registers are much cheaper per bit than PE short registers.

At a very detailed machine language level of coding there are differences in the programming of mode control. With back-up modes stored in short registers in the PE, a different mode with every instruction is appropriate, but programming complexity mounts when mode registers above the fourth are used. With control unit back-up storage, programming complexity remains essentially constant for any depth of mode storage, although producing memory interference when depth of storage exceeds that available in the high speed registers of the control unit. The same assembly language could be used in either case, if desired, putting the above differences solely into the assembly program.

SECTION III

REVISED PE LOGIC DESIGN

This section describes the mechanization and hardware requirements for the PE. A block diagram is included containing the data paths for each item required to mechanize a PE. The logic requirements per unit are tabulated in table 3-1.

PROCESSING ELEMENT DESCRIPTION

The revised PE logic design is shown in figure 3-1. A PE is essentially a three-register system which can execute a complete general purpose computer order code as described in Section VII of TR66-3 or otherwise modified in this report. A fourth register was added to store intermediate results. Capabilities of some units may be increased or decreased to vary operation code execution times. A summary of each unit in the PE follows.

MEMORY DATA REGISTER (MDR)

This unit is a buffer register between a PE and its PE's thin film stack and the outside world. The register serves as an intermediate buffer when performing array shifts or memory stores and fetches. This register is accessible even though the PE's mode status is disabled. The MDR receives operands from the Common Data Bus, A Register, Operand Select Gates, Address Adder and its PE's memory. The register's output is connected to the Common Data Bus, Address Adder, PE's memory and Operand Select Gates located within its PE's boundaries or four nearest neighbors.

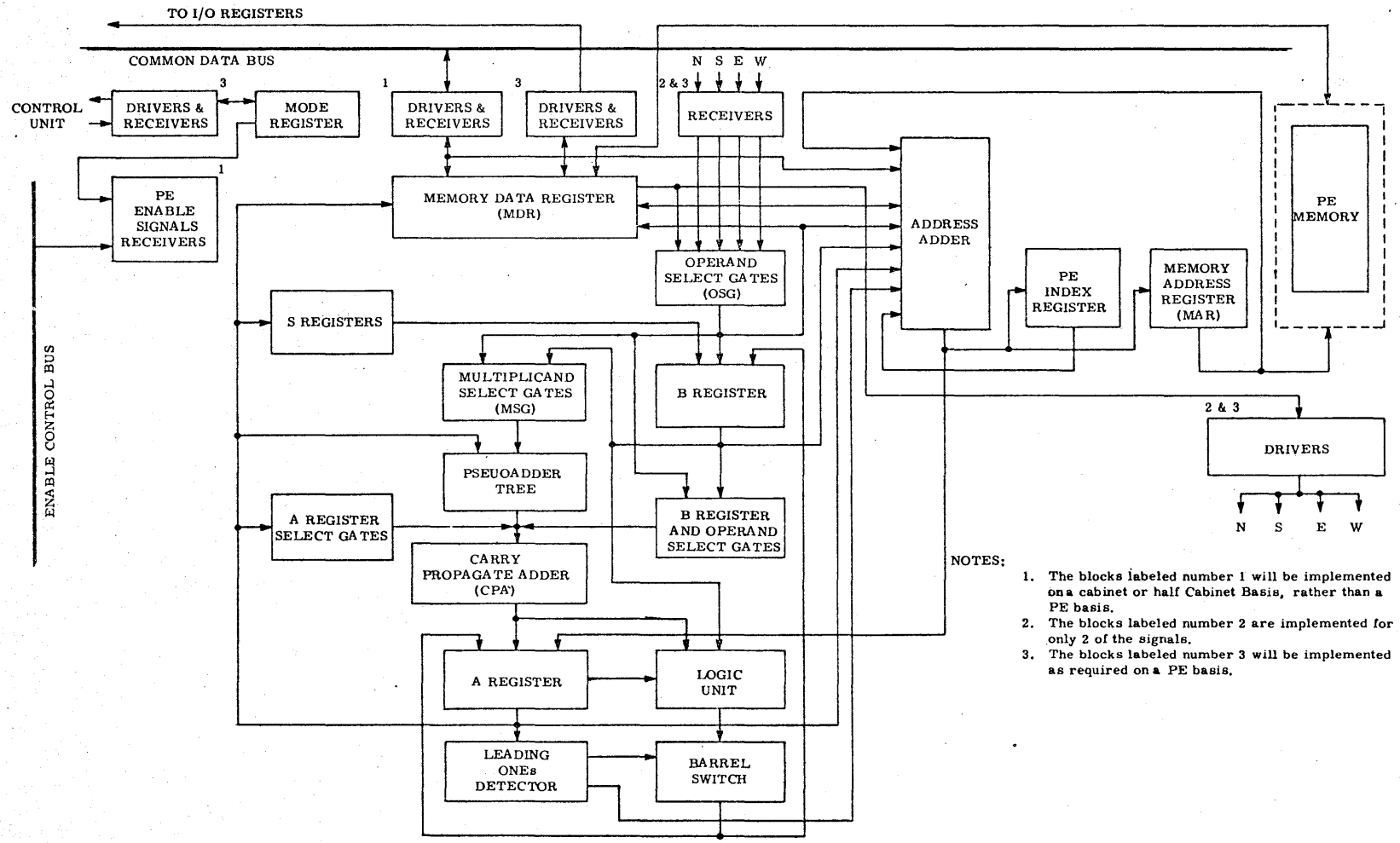


Figure 3-1. Revised PE Logic Design, Block Diagram

Table 3-1. PE Logic Requirements

UNITS	FUNCTIONS	BIT POSITIONS	GATES/ BIT	TOTAL GATES	FLIP FLOPS	DRIVERS/ RECEIVERS
Memory Data Register	Buffer register between thin film stack and logic	64	6	384	64	
	Address Adder inputs, bit positions 52 through 63	12	1	12		
	Drivers and Receivers to and from Common Data Bus					64/64
	Drivers to adjacent PE Operand Select Gates					64/00
	Drivers and Receivers to Input Output Registers					64/64
Operand Select Gates	Selects an Operand for PE processing	64	6	384		
	Receivers from two adjacent numbered PE's					00/128
Multiplicand Select	Selects 4-56 bit words based on 8 multiplier bits	56/word	16	896		
	Control word enable signal generation, some fan out			288	49	
Carry Propagate Adder	Consists of six adder sections: Receives mantissa field inputs from A-B Registers, Pseudoadder Tree and Operand Select Gates.					
	Gate and Flip-flop requirement for six sections			648	12	
	Input Gates: A Register	48	2	96		
	B Register and Operand Select Gates	48	3	144		
Address Adder	Consists of two adder sections: Perform address modification, exponent summation and supplementary sections to Carry Propagate Adder					
	Gate and Flip-flop requirement for two sections			216	4	
	Input Gates: Address Modification	12	4	48		
	Exponent Summation	16	5	80		
	Control Inputs	16	3	48		
	Output Gates:	16	3	48		
PE's Index Register	Stores Address variable	12			12	
Memory Address Register	Buffer register between thin film stack and logic	12			12	
A Register	Main operand store and Overflow flip-flop	64	7	448	65	
B Register	Auxiliary operand store	64	6	384	64	
	Address Adder inputs: positions 0 through 15	16	1	16		
S Register	Intermediate operand store	64			64	
Pseudoadder Tree	Converts 4 Words + Partial Product into CPA inputs	56	27	1512		
Logic Unit	Performs logic functions between A-B Registers	64	6	396		
Barrel Switch	Performs shifting functions.	65	14	910	1	
	Barrel Switch Control Logics			48		
Mode Register	Buffer register between a PE and the Control Unit	8			8	
	Mode control input gates			38		
	Drivers and Receivers to Control Unit					8/8
Leading One's Detector	Detects position of leading one, mantissa field			184		
PE Control Signal Rec.	Receives approximately 200 signals from the Control Unit					000/200
				7228	355	200/264

Assumptions: 1. The flip flops are constructed of 4 gates per element.
2. The Drivers and receivers are constructed of one logic gate.

Total gates used.

7228
1420
664
9312

OPERAND SELECT GATES (OSG)

This unit consists of one logic level of decoding gates. The enabled gate selects an incoming operand from its MDR or nearest neighbor's MDR for array transfer operations or PE processing. During an array transfer operation, the OSG output is stored temporarily in the MDR. PE processing requirements are that the selected operand be connected to the B Register input gates and Multiplicand Select Gates. Depending upon the instruction an operand, or multiple thereof, is loaded into the recipient register.

A-B REGISTER (ACCUMULATOR)

The A Register and B Register provide main operand store and auxiliary operand store, respectively. The Barrel Switch output is connected to both registers. These register inputs are used to execute operation codes that require main or auxiliary operand shifts, or logical combinations thereof. Both register outputs are routed to the Carry Propagate Adder (CPA) and Logic Unit.

In addition the A Register receives inputs from the CPA and the Modified Address Adder.* The CPA output is gated into the A Register's mantissa field. This output, depending on the algorithm, could be a partial product or remainder, a summed mantissa or just an operand transfer from the B Register or the OSG. During an operand transfer, the appropriate exponent is gated directly into the register's exponent field. The Modified Address Adder output is a summed value loaded into the exponent field. The A Register's outputs are connected to the Leading ONEs Detector, MDR, Special Register (SR) and the Modified Address Adder. The B Register receives additional inputs from the CPA, OSG, and SR. The CPA input is a completed 8-bit product obtained in a multiplication micro-sequence step. The OSG input is the incoming operand from this PE's or an adjacent PE's MDR. The SR input is the incoming operand obtained from a previous calculation. Other B Register's outputs are connected to the Multiplicand Select Gates and Modified Address Adder. The Multiplicand Select Gate inputs are multiplier bits decoded into enable signals for the next multiply step. The Modified Address Adder input is the exponent field to be summed in the adder.

LOGIC UNIT

This unit performs logic functions between the A Register and B Register. The unit receives outputs from both accumulator registers. The various logic functions are performed as specified in Section VII of TR66-3 or otherwise modified in this report. This unit provides the input gating function to the Barrel Switch unit.

---

*The Modified Address Adder term refers to the Address Adder extended 4 bit positions.

BARREL SWITCH

The Barrel Switch is a matrix of symmetrical gates which shifts a 64-bit parallel input any number of places to the left or right either end-off or end-around. Operation is started upon receipt of a 64-bit parallel input and appropriate control signals. The output is connected to both the A and B Registers.

LEADING ONES DETECTOR

The input to this unit consists of the A Register's mantissa bit positions. This unit detects the location of the most significant set bit, decodes its position as a radix 2 power and enables appropriate Barrel Switch displacement control signals. The unit also senses the absence of a set bit, thereby detecting a zero value. This decoded signal is used to zero the exponent field of the A Register.

MULTIPLICAND SELECT GATES

These gates select four multiplicand words each based on 2 bits of the multiplier mantissa only. The Multiplicand Select Gates are partitioned into two parts, one which receives the next set of multiplier bits to be decoded into multiplicand word enable signals and the other which gates the appropriate multiplicand (word) into the Pseudoadder Tree. During multiplication, four 2-bit pairs of multiplier bits are received and decoded in advance to form the enable signal that gates the required words into the Pseudoadder Tree for the next microsequence step. The decoded signal may select multiplicand multiples of times one, times minus one, or times two. This word selection enable signal is stored in flip-flops at the termination of the current microsequence step.

During an operand transfer, the word selected will enter the Pseudoadder Tree so that its output is positioned at the A Register mantissa input gates.

PSEUDOADDER TREE

The Pseudoadder Tree consists of three carry save adders per bit position. During multiplication, this unit receives inputs consisting of four words from the multiplicand Select Gates and the current partial product from the A Register. The carry save adders reduce these five inputs to two outputs, one being the summand sum and the other the summand carry. These outputs are gated into the CPA in their true and complemented form.

CARRY PROPAGATE ADDER (CPA)

Upon receipt of an input signal set from the Pseudoadder Tree or the A-B Registers, or the A register and Operand Select Gates, the CPA adds the inputs in a

microsequence step. When executing an arithmetic instruction, the extended CPA's output bit positions 0 thru 47 are gated into the A Register mantissa field. When executing the multiplication algorithm, the CPA's output positions 48 thru 53 are gated into the B Registers eight most significant locations, and positions 0 thru 47 are gated into the A Registers mantissa field.

During multiplication the CPA functions in two distinct modes: group carry save additions, and extended mantissa addition. The saved group carries are displaced 8 bit positions to the right and re-entered into that particular adder group. The group carry save addition is performed on those adder sections whose output is entered into the A Register's mantissa field. Additional adder sections are required to perform this algorithm. These adder sections receive their inputs from the most significant 16 bit positions of the B Register and group carry save bits from the previous microsequence step. The least significant section performs section carry save addition. The saved carry is re-entered into this adder displaced eight bit positions to the right. The last step in the multiplication algorithm requiring product summation is performed by the CPA operate mode of extended mantissa addition. The additional adder sections are connected to the CPA to form the extended mantissa adder. This microsequence step allows the carry to propagate throughout the adder to form the completed product.

MODE REGISTER

The Mode Register is a buffer register between the PE and the Control Unit. The PE status (operative or inoperative) is controlled by two bits of this register. These two bits are controlled exclusively by the Control Unit. The PE controls other register flip-flops when executing specific instructions.

ADDRESS ADDER

The Address Adder inputs are received from the Common Data Bus, MDR, PE's Index Register and the Memory Address Register. These added sums are connected to the input gates to the MDR, PE's Index Register and the Memory Address Register. The Modified Address Adder inputs are received from the exponent bit positions of the A-B Registers and the OSG. This added output is either decoded to form Barrel Switch control signals or entered into the A Register's exponent field.

PE INDEX REGISTER

This unit receives inputs from the Address Adder. Its output is connected to the Address Adder where compare operation or address modifications are performed.

MEMORY ADDRESS REGISTER (MAR)

The MAR is a buffer register between a PE and its thin film stack. The MAR output is the thin film stack operand address location. Depending upon the instruction the MAR can be loaded or modified with Address Adder inputs. The Address Adder input is gated into the MAR at the start of a memory cycle.

SPECIAL REGISTER (SR)

The SR Register is used to store intermediate results obtained during PE processing. The operand input is received from the A Register. The SR register's output is gated into the B Register, thus providing a buffer store for a previously computed operand without a memory fetch.

SECTION IV

THE INPUT-OUTPUT SUBSYSTEM

GENERAL CONSIDERATIONS

The following design considerations are pertinent to the ILLIAC IV I/O Subsystem.

1. A disk storage system will provide the principal on-line backup storage for the ILLIAC IV System. Present memory state of art singles out disk file storage as the only medium with the necessary volume and cost parameters to satisfy the ILLIAC IV requirements. Requirements for the disk file system are further considered at the end of this section.

2. I/O word transfers to and from the PE Array will be in the form of a 4096-bit word. This capability makes maximum use of the interrupt time of a quadrant and keeps I/O interference with the Array program to a minimum. It also provides a separate I/O path to each PE to accommodate applications which require asynchronous direct inputs to the PE memories. Some real-time problems involving large array sensor systems are typical of this application.

3. Capability to perform interlaced I/O transfers from simple descriptor operation is desirable. Considering the variety and number of peripheral devices the system may ultimately incorporate, the capability to store and rapidly fetch at least 64 I/O descriptors to control the interlaced I/O transfers is necessary.

4. Much routine I/O processing such as data packing and unpacking, descriptor assembly and updating, etc., must be performed external to and independent of the Array or Control Unit. Therefore, a processing device similar to a medium scale computer module would be most suitable. There are, however, some I/O programs such as code and format conversion which are eminently suitable to Array processing.

5. The existence of an economical core memory system separate and distinct from the Array memories is necessary for the following reasons:

a) Transfers from tape to disk, or card to disk, or disk to printer, must have a buffer area available to collect reasonable block sizes before making transfers to the disk in order to minimize latency time overhead.

b) Frequently used subroutines which overflow array storage can be kept in random access, zero latency time memory to avoid millisecond delays in processing.

c) To permit I/O lookahead transfers from disk to the array to overlap latency time with processing, a separate buffer memory is desirable as a staging area for such transfers.

It is possible that the buffer memory required to interface the disk mass memory and the PE array memory may be very large. A number of core memory manufacturers are being surveyed to identify possible candidates for core memories in the capacity range up to 16 million bits (modules independently accessible up to 8 million bits), with word-lengths in the range of 1024 to 4096 bits, and cycle times in the range of 2 to 8 microseconds. In the indicated capacity range the ferrite core memories are currently dominant from the cost standpoint. Memory organizations involving only 2 wires per core, such as linear select or variations of "2 1/2 D," are generally indicated to minimize mat fabrication cost. This is also a significant consideration here but in this case it is likely that the electronics will remain a very significant portion of the cost. For the longer word lengths (2K - 4K bits) it appears that "2 1/2 D" would offer little advantage over linear select and the large number of sense amplifiers and data bit drivers would reflect heavily on the cost per bit. It also appears that practical considerations may limit word drivers to something in the order of 1000 cores per line and hence might require multiple driver and switch matrices for the longer word lengths. Necessarily tentative extrapolations of fairly recent cost data suggests that it may be difficult to obtain costs much lower than 3 cents per bit for a core memory for this requirement.

At this time several core memory manufacturers, including the Burroughs Component Division, have been contacted about this requirement. The unusual geometry of this unit precludes the use of an existing product line item. This fact has delayed detailed responses beyond publication of this report.

DISK STORAGE

Effort to determine candidates for a disk file mass memory offering high data transfer rates and economic storage in capacities in the range of 300 million to 1 billion bits has continued. Obtaining the high data transfer rates desired in this application (greater than 300 megabits/second) requires a high degree of parallelism in the transfer, e. g. a large data storage word. The parallel mode of operation

increases the amount of read-write electronics required and it is desirable to keep this increase to a minimum. It is possible to reduce the number of read-write channels below one per bit if basic bit rate capability can be traded to effect the reduction. This tradeoff is indeed a necessity if fewer heads are simultaneously accessible than the number of bits required in parallel.

The reduction of the number of required read-write channels is effected by using a multiple zone storage format in which several bits per word time are recorded serially in the outer zones. In general a two-zone format allows the greatest reduction in the number of read-write channels for a given sacrifice of maximum bit rate while formats of three or more zones afford greater capacity utilization. More than three zones are seldom warranted since the required increase in the number of clock rates tends to cancel the attractiveness of the small, further increments to utilization efficiency. The following relations enable specific disk organizations to be evaluated. For a three-zone format the number of tracks in each zone must be related as follows:

$$2n_1 + n_2 = N_T\left[\frac{n_b}{n_h} + 2 - K\right]$$

and for a two-zone format

$$n_1 = N_T\left[\frac{n_b}{n_h} + 1 - K\right]$$

where

n_1 = the number of tracks in the outer zone

n_2 = the number of tracks in the middle zone

N_T = total number of tracks per disk face

K = number of bits per track per word stored serially in the outer zone.

$n_b = \frac{N}{n_f}$ = the number of bits per word per face

N = the number of bits per storage word

n_f = the number of disk faces used per word

The number of read-write channels required is:

$$n_c = n_h\, n_f$$

where n_h is the number of heads per face which are simultaneously selected. For a reduction of read-write channels below one per bit of the parallel storage word it is required that

$$n_b < n_h \quad .$$

In order that the available capacity of the disk file be used efficiently it is also required that n_h be an integral submultiple of the total number of tracks per disk face, N_T, and that n_f shall be an integral submultiple of the total number of faces per file, N_f.

For the usual ranges of interest the maximum packing density occurs in the innermost track of the outer zone. It is at this critical radius, R_c, that the maximum bit packing density, B, applies.

$$R_c \approx R_2 - \frac{n_1}{T}$$

where

R_2 = outermost track radius

T = the radial track density.

The number of storage words per data cylinder, that is, the number of words accessible in one disk revolution without head switching, is given by:

$$n_W = \frac{2\pi\, RcB}{K}$$

The word transfer rate, $n_W^{\cdot}$, is then determined in terms of the disk rpm as follows:

$$n_W^{\cdot} = n_W \left(\frac{RPM}{60}\right) = \frac{\pi RcB\,(RPM)}{30K}$$

The total file capacity is:

$$C = \frac{n_W N_T N_f}{n_h n_f} \text{ words/file} = \frac{n_W N_T N_f N}{n_h n_f} \text{ bits/file.}$$

The ideal packing efficiency, a measure of capacity utilization referred to the maximum possible capacity which would be available at uniform packing density in all tracks, is given by:

$$e_1 = \frac{200\ RcN}{K\ n_h n_f (R_2 + R_1)} \qquad \%.$$

where R_2 and R_1 are the radius of the outermost and innermost tracks respectively. A related packing factor based on the maximum possible capacity at a constant number of bits per track for all tracks is:

$$F = \frac{RcN}{KR_1 n_h n_f} .$$

Note that this factor is in general greater than unity because the multizone format utilizes available disk area more efficiently than would be possible with a single clock rate for the entire disk (as would be required for a straight parallel configuration).

In the multiple zone format every track in the same zone contains the same number of bits, but each zone requires a different clock rate. The clock rates for a two-zone format, for example, would be:

$K\ n_W^{\cdot}$ in the outer zone

$(K-1)\ n_W^{\cdot}$ in the inner zone.

As previously reported, the Librascope 4800 disk file series is potentially attractive because of the large number of heads per face. The model 4802 is of particular interest because the increased packing density offers higher basic bit rates and more capacity in the same basic unit. The salient characteristics are summarized here.

No. disk per file - 6 (48" dia.)

Tracks per face - 432

Packing density - 2000 bits/inch

Track density - 48 tracks/inch

RPM - 900 (35 ms avg. access)

Table 4-1. 512-Bit, Parallel Organizations for the Librascope 4802

	No Head Modifications		All Selected Heads on	
			One Disk	One Face
Format (Bits Outer Zone: Bits Inner Zone)	3:2	4:3	3:2	4:3
Head Groups/Head Assembly	1	1	3	4
Bits/Face/Word	86	128	256	512
Heads/Sel. /Face/Word	36	36	108	144
Heads/Group	12	12	4	3
Faces Sel. /Unit	6 of 12	4 of 12	2 of 12	1 of 12
Ideal Packing Efficiency	83%	86. 3%	83%	86. 3%
No. Stg. Words/Data Cylinder	83840	58000	84500	58000
32-Bit Segments/Data Cylinder	2620	1815	2640	1815
Word Transfer Rate, MHz	1. 26	0. 87	1. 27	0. 87
Read-Write Channels	216	144	216	144
Data Cylinders/Unit	24	36	24	36
Total Stg. Cap. Words/File (10^6)	2. 0	2. 09	2. 02	2. 09
Total Stg. Cap. Bits/File (10^6)	1024	1070	1030	1070
Clock Rate, Inner, MHz	2. 52	2. 62	2. 54	2. 62
Clock Rate, Outer, MHz	3. 78	3. 48	3. 81	3. 48
No. Tracks, Outer Zone	168	240	160	240
No. Tracks, Inner Zone	264	192	272	192
Head Groups, Outer Zone	14	20	40	80
Head Groups, Inner Zone	22	16	68	64
Head Sticks, Outer Zone	14	20	13 + 1/3	20
Head Sticks, Inner Zone	22	16	22 + 2/3	16
Clock Channels/Unit	12	8	4	2

The large number of accessible heads permit a number of possible disk organizations. Without sacrificing capacity the following might represent limiting transfer rate capabilities.

1. Up to 128 bits per face or 1024 bits per file at a word transfer rate of approximately 0. 8 megaHertz without modifications to head stick wiring.

2. Up to 512 bits per face or 2048 bits per file at a word transfer rate of approximately 0. 8 megaHertz if head stick wiring is revised to enable simultaneous selection of four heads per stick.

3. Up to 256 bits per face or 1024 bits per file at a word transfer rate of approximately 1. 2 megaHertz if head wiring is revised to enable simultaneous selection of three heads per stick.

Still larger word lengths would be possible by revising head stick wiring to permit simultaneous access to all heads but the resulting increase in the number of head wires from 15 to 39 would likely be troublesome. With no fewer than 3 heads per group the number of wires per stick can be held to 23 for completely flexible head-track sparing, or to 14 wires per head stick if restriction to group sparing is acceptable. At the word lengths of interest it is necessary to either subdivide the head stick wiring or concurrently select heads over several disk faces. Both have some potential disadvantages; the latter from the possible necessity for skew correction due to relative timing errors between several faces and/or head plates, and the additional clock channels required. At a word length of 512 bits either approach is a possibility. Table 4-1 presents a summary for four different organizations: two involve no head stick revisions, and two require head revisions but involve only one disk or one disk face, respectively. Two different two-zone formats (indicated in the table as the number of serial bits per word time in outer zone: the number of serial bits per word time in the inner zone) offer words transfer rates of 0. 8 to 1. 2 megaHertz. At the initial contact with General Precision, Inc., they indicated a tentative preference for organizations involving concurrent head selection on a single face because they did not think they had adequate information at that time with respect to the skew problem.

SECTION V

PROGRAMMING

PE OPERATIONS

This section describes modifications and additions to the list of instructions executed, at least in part, by the logic within Processing Elements. All instructions not mentioned here are functionally unchanged from those described in Section VII of Progress Report No. 1 (TR-66-3; August 26, 1966). However, some changes in instruction formats are contemplated to accommodate certain new instructions. In particular, some of the new instructions will be made variants of old instructions instead of new op-codes and these old instructions will have their formats changed to incorporate variant fields.

In 32-bit mode, bits 0 and 32 are the sign-bits of the two operands in either the A or the B Register; bits 1-7 and 33-39 are the exponents; bits 8-31 and 40-63 are the mantissa magnitudes. Fixed-point operations always involve the mantissa magnitudes and the mantissa signs only. Thus, in 32-bit mode fixed-point, operations are performed on 24-bit-plus-sign quantities. When an operation uses operands from both the A and the B Registers, corresponding bits from the two registers are involved. As an example, a double-length arithmetic shift (SHD) treats bits 8-31 of the A and B Registers as one 48-bit quantity and bits 40-63 of the two registers as another 48-bit quantity. Shift counts are interpreted modulo 32 in 32-bit mode.

The PE Index Register is not affected by the word-length mode. For example, Load Index from A Register (LXA) always transmits bits 52-63 of the A Register to the Index Register.

There are now nine mode-bits per PE, designated as the W, E, E1, F, F1, G, I, and J bits. The W-Bit is always set or reset by the Control Unit and designates the word-length (ZERO implies 64-bit operands, ONE implies 32-bit operands).

In 64-bit mode the E-Bit enables or disables the changing of full operand registers and the E1-Bit is not available for program use. In 32-bit mode, the E-Bit controls the changing of bits 0-31 of the registers and the E1-Bit controls bits 32-63. In test instructions executed by PE's, any register bit which is disabled from being changed is also disabled from being tested to generate a change in any mode bit. In 64-bit mode, the F-Bit is set to ONE whenever an arithmetic fault occurs in the PE and the F1-Bit is not available for program use. In 32-bit mode, the F-Bit is set to ONE whenever a fault occurs in arithmetic performance on bits 0-31 of operands and the F1-bit is set to ONE whenever a fault occurs in bits 32-63. In 64-bit mode, the instruction list is augmented to permit specification of any one of the G, H, I, and J bits to hold the result of any test. For example, the tests for the A-Register being zero now include GAZ and HAZ (Set G/H if A Equals Zero) as well as IAZ and JAZ. The Control Unit can enable or disable a PE as a result of the condition of the G and H bits in a manner identical to the previous capability with the I and J bits. In 32-bit mode, the IAZ instruction sets the G-Bit if bits 0-31 of the A-Register are zero and sets the I-Bit if bits 32-63 of the A-Register are zero. Of course, if the E-Bit equals ZERO, the G-Bit is unchanged and if the E1-Bit equals ZERO, the I-Bit is unchanged. Similarly, JAZ affects both the H and J bits in 32-bit mode. The operations of instructions calling for setting of the G and H bits (e.g. GAZ, HAZ) are presently undefined for 32-bit mode.

The new register in each PE is designated the "S Register" (for Save Operand Register or Special Register). The contents of the S Register are not involved or altered by any arithmetic or shift instruction. Three instructions have been defined involving the S Register:

SAS — Store A Register in S Register

Copy all of the A Register into the S Register. If in 64-bit mode and E = ZERO, do not change the S Register. If in 32-bit mode and E = ZERO, do not change bits 0-31 of the S Register. If in 32-bit mode and E1 = ZERO, do not change bits 31-63 of the S Register.

IBS — Load B Register from S Register

SWAPS — Swap A, B and S Registers

A Goes to S

B Goes to A

S Goes to B

There is now a set of instructions which permit operand transmissions to begin and/or end at the MDR without involving the A and B Registers. These instructions are designed so that intermediate PE's can participate in multi-PE routing without having their operand registers altered. The Control Unit has a single instruction which causes a sequence of transfers among neighboring PE's to accomplish the desired multi-PE routing. The instructions transmitted to the PE's by the Control Unit in response to the single instruction executed by the

Control Unit are also available for the program to call upon individually. These instructions include MDR-to-MDR transmission with direction specified, storage from the A or the B Register to the MDR of the same PE, and loading of the A or B Register from the MDR of the same PE. A disabled PE does not execute those instructions which load its A or B Register but does participate in the others. This control of which PE's finally receive multi-PE transmissions, coupled with the programmed control of the path distance and directions, permit the single Control Unit instruction required. In effect, all PE's originate transmissions but, after the path control has been counted down, only enabled PE's accept the result of the transmission. The operands which arrive at disabled PE's are retained in their MDR's and may be discarded by the next instruction causing their replacement. However, some saving of transmission time may be accomplished, for certain routing patterns, if the first multi-PE transmission is followed by a new transmission that starts with MDR-to-MDR transmissions and which terminates with a different set of PE's being enabled to accept the transmission results from their MDR's into their A and B Registers.

There are now instructions which cause the clearing of the exponent field of the A or the B Register. Clearing of the mantissa field has always been available as an end-off arithmetic shift with any shift count exceeding the length of the mantissa. With the Barrel, shifting to clear a field takes no longer than the execution of a special clear field instruction. Obviously, in assembly language a clear mantissa instruction may exist which would assemble as a shift.

There is now a round variant on each of the pertinent arithmetic instructions. In floating-point, rounding is accomplished before normalization to preserve the proper significance.

There are now instructions which add address fields to the previous contents of the Memory Address Register, with the address fields being optionally provided by fields within the instruction or the PE Index Register or both. This permits straightforward, multi-level, indirect addressing when the level is known.

There is now an instruction which stores an operand from the A, B or Memory Data Register of an enabled PE to the Common Data Bus. When only one PE is enabled, this provides the required transmission of an operand from a PE to the Control Unit without requiring memory access. When two or more PE's are enabled, the result of this instruction is undefined.

There are now instructions which carry an indexable 6-bit field specifying one bit within either the A or the B Registers. The bit number is interpreted modulo 64 in 64-bit mode. In 32-bit mode, the bit number is interpreted modulo 32 and as modulo 32 plus 32, allowing specification of corresponding bits in both halves of the Register. The operations on and with the specified bit are:

- Set either the G, H, I or J bit to equal the specified bit.
  In 32-bit mode, only I and J are defined.)

- Change the specified bit.
- Set the specified bit to ZERO.
- Set the specified bit to ONE.

There are now instructions which perform arithmetic on the magnitudes of the operands.

There are now instructions which compare the magnitude of the A Register with the magnitude of the B Register. Also, the magnitude of either register may be compared with the Common Value. Comparison with the magnitude of the Common Value is not included as a separate PE instruction since this may be accomplished by having the Control Unit broadcast the magnitude only of the Common Value.

The instructions which operate upon or compare the value of the PE Index Register have been augmented so that the Index Register may be stored in, loaded from, or compared with the least significant twelve bits of the B Register or a memory word. When a memory word is involved in any of these instructions, the address in memory is indexable.

CONTROL UNIT INSTRUCTIONS

In the following numeric list of instructions, the first syllable is given in octal. Op-code "000" is interpreted by the CU as a halt. Op-codes "001-177" are executed in part by the CU and part by the PE array. Op-codes "200-377" are interpreted fully by the CU and no direct PE action results.

In ulti-syllabic instructions, the following abbreviations are used to indicate the coding of the syllables other than the first:

.	A bit which does not affect the operation of the instruction being described.
a	A bit which is part of an address or literal field.
b	A bit which is part of a field which designates a bit-number within a register.
c	A bit which is part of a shift count (in some instructions a bit-number).
C	An eight-bit syllable used to designate an address within CU local memory.
d	A bit which designates shift direction.
e	A bit which distinguishes between end-off and end-around shifts. End-around shifts are mnemonically referred to as "Rotate."

i A variant bit which is defined as ONE for the specific variant being discussed.

o A variant bit which is defined as ZERO for the variant being discussed.

L An eight-bit syllable which is part of a literal field.

M An eight-bit syllable which is part of an address field used by the CU to address Main Memory.

v A variant bit which has meaning in defining a variant described elsewhere. Examples of this are given following this list.

x A variant bit which controls indexing, by the PE Index Register, of the address, shift-count or bit-number field given in the instruction.

As an example of the use of these abbreviations in the instruction list, consider the instructions which transmit data between neighboring PE's. Each of these instructions starts with an op-code syllable equal to octal 120 and has a second syllable specifying variants. The variants permit the choice of the A Register, the B Register or the Memory Data Register as the transmission origin; the same three registers, or the Memory Address Register, may be the destination; the transmission-direction may be North, East, South, or West. Two bits are used for each of these variants, leaving two undefined bits in the variant syllable. The two most significant bits specify the originating register, the next two bits specify the destination register and the two least significant bits specify the direction. Transmission from the Memory Data Register to the Memory Data Register of the North neighbor is indicated by coding "10" for each of the register-designation bits and "00" for the direction: 1010..00, where the periods indicate the undefined bits. In the instruction list this would appear "ioio..oo". However, to avoid listing each possible variant separately, the list contains the following three entries:

```
120 iovv..vv        D-T-
120 vvio..vv        -DT-
120 vvvv..oo        --TN
```

The first entry denotes the coding of the bits that specify the origin and indicates that the other variant bits have no affect on the origin. Similarly, the second entry denotes destination and the third entry denotes direction. The mnemonic abbreviations show the characters that represent the variants and hyphens are used for character positions that are used to denote other variants. In the specific example being considered, "DDTN" means MDR-to-MDR transmission North and is encoded "ioio ..oo".

Elements of the Control Unit

The control unit is conveniently described in terms of its registers and other functional entities. The registers consist of:

- Eight mode function registers, E_1, E_2, F_1, F_2, G, H, I and J although physically located in the processing elements, are addressed by program as though they were registers of the control unit
- Index registers, each 32 bits long
- 64-bit registers in high speed scratchpad storage. Sixty-four of these can be used as the broadcast buffer.
- Program counter
- Interrupt register and mask, or interrupt control, register
- Local register address pointer, 8 bits of register address.

The above registers are addressible uniformly using an 8-bit register address in the instructions. In addition, there are several registers which are not addressible by the 8-bit field, but are implicity addressed by all instructions which are relevant to their use. They are:

- Program lookahead, for holding a reservoir of program steps independently of memory accessing.
- Address register, which serves as an accummulator for address-sized fields. It is 24 bits long.
- Accumulator (for want of a better name), a common register referenced by all data manipulating instructions. It is 64 bits long.
- A queue of instructions and data, which decouples the operations which are solely within the CU from those that refer to the PE's. This queue, like that in the B8500, has no effect on the operations of the instructions except to permit a certain amount of parallelism, and therefore is not discussed further.

Discussion of the Instructions

All registers within the CU are uniformly addressed by an 8-bit field, which is indexable. The operation of transferring the I Register to the E Register, which is "enable those PE's which had a true result on the most recent test involving the I-Bit," is accomplished by the same sequence of instructions as transferring register number 24 to register number 42 which merely moves data around the scratchpad. Similarly, a jump is accomplished by storing a new value in the numbered register which is the program counter.

A bonus which comes from this approach is that capability which is invented for the use of one special case, such as reading a PE number from the E register, can be used on any data within the CU, thus increasing programming flexibility.

The CU Accummulator is central to most CU operations. Its use is implied in most CU data transmission instructions. Whenever an instruction is transmitted from the CU to the PE array, the CU Accummulator may modify the PE instruction or otherwise participate in the operation of the PE. For example, the PE instruction "Load A Register from Common Value" (LAC) transmits the contents of The CU Accummulator to the A Registers of all enabled PE's. The accummulator also receives the transmission from the PE in the "Store A Register to Common Data Bus" Instruction (SAC).

When an instruction with an address, shift-count, or bit-number field is transmitted to the PE's, the contents of the CU Accummulator are added to this field before transmission. Thus, multiple indexing with CU index registers is accomplished by ordinary addition to the CU Accummulator before issuance of the PE instruction.

The CU also has one special index register for its own use in addressing within its local memory. The act of placing a value in this index automatically causes the addition of this value to the next instruction with a CU register number. This register is always cleared after its use.

000	HALT	Halt All Operations
002	CHSA	Change Sign of A Register
003	CHSB	Change Sign of B Register
004	SAP	Set A Register Positive
005	SBP	Set B Register Positive
006	SAN	Set A Register Negative
007	SBN	Set B Register Negative
010	CEA	Clear Exponent of A Register
011	CEB	Clear Exponent of B Register
013	CMB	Complement B Register
015	CLB	Clear B Register
016	NORM	Normalize A Register
020	SAD	Store from A Register to Memory Data Register
021	SBD	Store from B Register to Memory Data Register
022	LAD	Load A Register From Memory Data Register

023	LBD	Load B Register from Memory Data Register
024	SAC	Store from A Register to Common Data Bus
025	SBC	Store from B Register to Common Data Bus
026	LAC	Load A Register from Common Value
027	LBC	Load B Register from Common Value
030	LMA	Load Memory Address Register from A Register
031	LMB	Load Memory Address Register from B Register
032	LMC	Load Memory Address Register from Common Value
033	LMD	Load Memory Address Register from Memory Data Register
034	LXA	Load Index Register From A register
035	LXB	Load Index Register from B Register
036	LXC	Load Index Register from Common Value
037	LXD	Load Index Register from Memory Data Register
040	SXA	Store Index Register in A Register
041	SXB	Sotre Index Register in B Register
042	SXC	Store Index Register to Common Data Bus
043	SXD	Store Index Register in Memory Data Register
044	ADAX	Add A Register to Index Register
045	ADBX	Add B Register to Index Register
046	ADCX	Add Common Value to Index Register
047	ADDX	Add Memory Data Register to Index Register
050	LDAM	Load A Register as Designated by Memory Address Register
051	LDBM	Load B Register as Designated by Memory Address Register
052	STAM	Store A Register as Designated by Memory Address Register
053	STBM	Store B Register as Designated by Memory Address Register
054	LDMM	Load Memory Address Register as Designated by Memory Address Register
055	LDDM	Load Memory Data Register as Designated by Memory Address Register
060	SAS	Store A Register in S Register
061	LBS	Load B Register from S Register
062	SWAP	Swap A and B Registers
063	SWS	Swap with S Register (A to S: S to B: B to A)
064	DBA	Duplicate B Register from A Register

100	xdcc cccc	SHA	Shift A Register Mantissa
101	xdcc cccc	SHB	Shift B Register Mantissa
102	xdcc cccc	SAL	Shift A Register Logical
103	xdcc cccc	SBL	Shift B Register Logical
104	xdcc cccc	RAL	Rotate A Register Logical
105	xdcc cccc	RBL	Rotate B Register Logical
106	xdcc cccc	SHD	Shift Double-Length Mantissa
107	xdcc cccc	SDL	Shift Double-Length Logical
110	x. bb bbbb	CHBA	Change Specified Bit of A Register
111	x. bb bbbb	CHBB	Change Specified Bit of B Register
112	x. bb bbbb	SBA	Set Specified Bit of A Register
113	x. bb bbbb	SBB	Set Specified Bit of B Register
114	x. bb bbbb	CLBA	Clear Specified Bit of A Register
115	x. bb bbbb	CLBB	Clear Specified Bit of B Register
120	oovv .. vv	A-T-	Inter-PE Transmission from A Register
120	oivv .. vv	B-T-	Inter-PE Transmission from B Register
120	iovv .. vv	D-T-	Inter-PE Transmission from Memory Data Register
120	vvoo .. vv	-AT-	Inter-PE Transmission to A Register
120	vvoi .. vv	-BT-	Inter-PE Transmission to B Register
120	vvio .. vv	-DT-	Inter-PE Transmission to Memory Data Register
120	vvii .. vv	-MT-	Inter-PE Transmission to Memory Address Register
120	vvvv .. oo	--TN	Inter-PE Transmission North
120	vvvv .. oi	--TE	Inter-PE Transmission East
120	vvvv .. io	--TS	Inter-PE Transmission South
120	vvvv .. ii	--TW	Inter-PE Transmission West
121	oioo ioio	CLA B0000	Clear A Register Boolean Function 0000
121	oiii ioio	AND B0001	A AND B; Result to A Register Boolean Function 0001
121	oiii ioii	NIMP B0010	Not (A Implication B); Result to A Register Boolean Function 0010
121	oiii iiio	NRIMP B0100	Not (B Implication A); Result to A Register Boolean Function 0100

121	oooi	ioio	DAB B0101	Duplicate A Register from B Register Boolean Function 0101
121	ioii	ioio	EOR B0110	A Exclusive OR B; Result to A Register Boolean Function 0110
121	ooii	ioio	OR B0111	A OR B; Result to A Register Boolean Function 0111
121	oiii	iiii	NOR B1000	Not (A or B); Result to A Register Boolean Function 1000
121	ioii	ioii	MEQ B1001	A Material Equivalence B; Result to B Register Boolean Function 1001
121	oooi	ioii	NOTB B1010	Complement of B Register Transmitted to A Register Boolean Function 1010
121	ooii	ioii	RIMP B1011	B Implication A; Result to A Register Boolean Function 1011
121	ooio	iiio	CMA B1100	Complement A Register Boolean Function 1100
121	ooii	iiio	IMP B1101	A Implication B; Result to A Register Boolean Function 1101
121	iiii	iiii	NAND B1110	Not (A AND B); Result to A Register Boolean Function 1110
122	oovv	vvvv	----	Perform Arithmetic on Sign and Magnitude
122	oivv	vvvv	----M	Perform Arithmetic on Magnitude Only
122	vvvv	vvvo	----R	No Rounding of Result
122	vvvv	vvvi	----R	Round Result
122	vvoo	ooov	ADD	Fixed Point Add A to B; Result to A Register
122	vvoo	ooiv	UFAD	Unnormalized Floating Add A to B; Result to A Register
122	vvoo	oiov	SUB	Fixed Point Subtract B from A; Result to A Register
122	vvoo	oiiv	UFSU	Unnormalized Floating Subtract B from A; Result to A Register
122	vvoo	ioov	MUL	Fixed Point Multiply A by B; Result to A & B
122	vvoo	ioiv	UFMU	Unnormalized Floating Multiply A by B; Result to A & B
122	vvoo	iiov	DIV	Fixed Point Divide A by B; Quotient to A, Remainder to B
122	vvoo	iiiv	UFDV	Unnormalized Floating Divide A by B; Quotient to A, Remainder to B
122	vvoi	ooiv	FAD	Floating Add A to B; Result to A

122	vvoi	oiiv	FSU	Floating Subtract B from A; Result to A
122	vvoi	ioiv	FMU	Floating Multiply A by B; Result to A & B
122	vvoi	iiiv	FDV	Floating Divide A by B; Quotient to A, Remainder to B
122	vvio	ooiv	EAD	Extend Precision After Floating Add; Extension of Sum to A Register
122	vvio	oiiv	ESU	Extend Precision After Floating Subtract; Extension of Difference to A Register
122	vvio	iiov	IDV	Integer Divide A by B; Quotient to A, Remainder to B
123	iiii	iiio	GR8	Test 8-bit Bytes for A Greater than B; Result to A
123	iiii	ioii	LS8	Test 8-Bit Bytes for A Less than B; Result to A
123	ioii	ioii	EQ8	Test 8-Bit Bytes for A Equal to B; Result to A
130	oovv	vvvv	G---	Set G-Bit as Result of Comparison of A Register with B Register
130	oivv	vvvv	H---	Set H-Bit as Result of Comparison of A Register with B Register
130	iovv	vvvv	I---	Set I-Bit as Result of Comparison of A Register with B Register
130	iivv	vvvv	J---	Set J-Bit as Result of Comparison of A Register with B Register
130	vvoo	vvvv	----	Compare Sign and Magnitude of A Register with Specified State of B Register
130	vvoi	vvvv	-M---	Compare Magnitude only of A Register with Specified State of B Register
130	vvio	vvvv	-L---	Compare Logical Value of A Register with Specified State of B Register
130	vvvv	oovv	-----	Compare Sign and Magnitude of B Register with Specified State of A Register
130	vvvv	oivv	----M	Compare Magnitude Only of B Register with Specified State of A Register
130	vvvv	iovv	----L	Compare Logical Value of B Register with Specified State of A Register
130	vvvv	vvoo	--EQ-	Test for A Equal to B
130	vvvv	vvoi	--LS-	Test for A Less than B
130	vvvv	vvio	--GR-	Test for A Greater than B
131	oovv	vv..	G---	Set G-Bit as Result of Test of A or B Register
131	oivv	vv..	H---	Set H-Bit as Result of Test of A or B Register

131	iovv vv..	I---	Set I-Bit as Result of Test of A or B Register
131	iivv vv..	J---	Set J-Bit as Result of Test of A or B Register
131	vvov vv..	-A--	Test A Register
131	vviv vv..	-B--	Test B Register
131	vvoo oi..	--Z	Test for Plus or Minus Zero
131	vvvo io..	--PZ	Test for Plus Zero
131	vvvo ii..	--0	Test for ALL ONES (Minus Full Scale)
131	vvvi oo..	--S	Copy Sign-Bit to Designated Test Bit
132	oovv vv..	GX--	Set G-Bit as Result of Index-Register Test
132	oivv vv..	HX--	Set H-Bit as Result of Index-Register Test
132	iovv vv..	IX--	Set I-Bit as Result of Index-Register Test
132	iivv vv..	JX--	Set J-Bit as Result of Index-Register Test
132	vvoo vv..	-XE-	Compare Index Register for Equality with Specified Quantity
132	vvoi vv..	-XL-	Test for Index Register Less than Specified Quantity
132	vvio vv..	-XG-	Test for Index Register Greater than Specified Quantity
132	vvii vv..	-XZ	Test for Index Zero
132	vvvv oo..	-X-A	Compare Index Register with A Register
132	vvvv oi..	-X-B	Compare Index Register with B Register
132	vvvv io..	-X-C	Compare Index Register with Common Value
132	vvvv ii..	-X-D	Compare Index Regsiter with Memory Data Register
134	xobb bbbb	GBA	Set G-Bit to Designated Bit of A Register
134	xibb bbbb	GBB	Set G-Bit to Designated Bit of B Register
135	xobb bbbb	HBA	Set H-Bit to Designated Bit of A Register
135	xibb bbbb	HBB	Set H-Bit to Designated Bit of B Register
136	xobb bbbb	IBA	Set I-Bit to Designated Bit of A Register
136	xibb bbbb	IBB	Set I-Bit to Designated Bit of B Register
137	xobb bbbb	JBA	Set J-Bit to Designated Bit of A Register
137	xibb bbbb	JBB	Set J-Bit to Designated Bit of B Register

140	oovv aaaa	aaaa aaaa	GX-L	Set G-Bit as Result of Comparison of Index Register with Literal
140	oivv aaaa	aaaa aaaa	HX-L	Set H-Bit as Result of Comparison of Index Register with Literal
140	iovv aaaa	aaaa aaaa	IX-L	Set I-Bit as Result of Comparison of Index Register with Literal
140	iivv aaaa	aaaa aaaa	JX-L	Set J-Bit as Result of Comparison of Index Register with Literal
140	vvoo aaaa	aaaa aaaa	-XEL	Test for Index Equal to Literal
140	vvoi aaaa	aaaa aaaa	-XLL	Test for Index Less than Literal
140	vvio aaaa	aaaa aaaa	-XGL	Test for Index Greater than Literal
141	.ovv aaaa	aaaa aaaa	L-L	Load Specified Register with Literal
141	.ivv aaaa	aaaa aaaa	ADL-	Add Literal to Specified Register
141	.voo aaaa	aaaa aaaa	LAL/ ADLA	Specify A Register
141	.voi aaaa	aaaa aaaa	LBL/ ADLB	Specify B Register
141	.vio aaaa	aaaa aaaa	LXL/ ADLX	Specify Index Register
150	x.oo aaaa	aaaa aaaa	STA	Store A Register
150	x.oi aaaa	aaaa aaaa	STB	Store B Register
150	x.io aaaa	aaaa aaaa	STD	Store Memory Data Register
150	x.ii aaaa	aaaa aaaa	STX	Sotre Index Register
151	x.oo aaaa	aaaa aaaa	LDA	Load A Register
151	v.oi aaaa	aaaa aaaa	LDB	Load B Register
151	x.io aaaa	aaaa aaaa	LDD	Load Memory Data Register
151	x.ii aaaa	aaaa aaaa	LDX	Load Index Register
152	xooo aaaa	aaaa aaaa	LDM	Load Memory Address Register
152	xioo aaaa	aaaa aaaa	ADM	Add to Memory Address Register
152	xiii aaaa	aaaa aaaa	ADMX	Add from Memory to Index Register
200			DUP	Duplicate Non-Zero Half of CU Accummulator
201			FULL	Enter Full-Word (64-bit) Mode
202			HALF	Enter Half-Word Mode
204			ZEROF	If CU Accumulator Is Not All ZEROS, Skip Until the Next "UNSKIP" Instruction

205		ZEROT	If CU Accumulator is All ZEROS, Skip Until the Next "UNSKIP" Instruction
206		ONESF	If CU Accumulator Is Not All ONES, Skip Until the Next "UNSKIP" Instruction
207		ONEST	If CU Accumulator is all ONES, Skip Until the Next "UNSKIP" Instruction
210		SKIPF	If the Result of the Last Test was False, Skip Until the Next "UNSKIP" Instruction
211		SKIPT	If the Result of the Last Test was True, Skip Until the Next "UNSKIP" Instruction
212		UNSKIP	Resume Executing All Instructions
220		LEADO	Find Leading ONE in the CU Accumulator; Put Bit-Number in CU Accumulator
221		LEADZ	Find Leading ZERO in the CU Accumulator; Put Bit-Number in CU Accumulator
230		CCL	Clear CU Accumulator
231		CCOM	Complement CU Accumulator
232		XCUA	Index by CU Accumulator
240	aaaa aaaa	STL	Store CU Accumulator in CU Local Memory
241	aaaa aaaa	STLC	Store CU Accumulator in CU Local Memory, Complemented
242	aaaa aaaa	LDL	Load CU Accumulator from CU Local Memory
244	aaaa aaaa	EXCH	Exchange CU Accumulator with CU Local Memory
245	aaaa aaaa	EXCHC	Exchange Complement of CU Accumulator with CU Local Memory
246	aaaa aaaa	CADD	Add to CU Accumulator
247	aaaa aaaa	CSUB	Substract from CU Accumulator
250	aaaa aaaa	CAND	AND to CU Accumulator
251	aaaa aaaa	COR	OR to CU Accumulator
252	aaaa aaaa	CEOR	Exclusive OR to CU Accumulator

260	oobb bbbb		CCLB	Clear Designated Bit
260	oibb bbbb		CSBO	Set Designated Bit to ONE
260	iobb bbbb		DDHB	Change Designated Bit
261	edcc cccc		CSHIFT	Shift
262	oobb bbbb		CTSBZ	Test Bit for ZERO
262	oibb bbbb		CTSBO	Test Bit for ONE
270	aaaa aaaa		EQUALT	
271	aaaa aaaa		EQUALF	
272	aaaa aaaa		GRTRT	
273	aaaa aaaa		GRTRF	
274	aaaa aaaa		LESST	
275	aaaa aaaa		LESSF	
276	aaaa aaaa		XADD	Add to CU Index
277	aaaa aaaa		SUB	Subtract from CU Index
300	oovv	nnnn nnnn	RTA-	Route from A Registers
300	oivv	nnnn nnnn	RTB-	Route from B Registers
300	iovv	nnnn nnnn	RTD-	Route from Memory Data Registers
300	vvoo	nnnn nnnn	RT-A	Route to A Registers
300	vvoi	nnnn nnnn	RT-B	Route to B Registers
300	vvio	nnnn nnnn	RT-D	Route to Memory Data Registers
300	vvii	nnnn nnnn	RT-M	Route to Memory Address Registers
310	L L L		SLIT	Short (24-bit) Literal to CU Accumulator
311	M M M		STO	Store One Word from CU Accumulator to Main Memory
312	M M M		LOAD	Load One Word from Main Memory to CU Accumulator
320	C M M M		BIN	Block Transfer into CU Memory from Main Memory
321	C M M M		BOUT	Block Transfer Out from CU Memory to Main Memory
330	L L L L L L L L		LIT	Full-Word Literal to CU Accumulator

SECTION VI

ILLIAC IV APPLICATIONS STUDY

The implementation on ILLIAC IV of the Cooley-Tukey algorithm for the calculation of complex Fourier series has been studied. A method is described in this section.

DESCRIPTION OF THE COOLEY-TUKEY ALGORITHM

This is a method for evaluating the function X(j) for N values of the argument ($j = 0, 1, \ldots, N-1$) when we are given the N complex coefficients A(k), $k = 0, 1, \ldots, N-1$, appearing in the Fourier sum that is used to define the function X(j).

$$X(j) = \sum_{k=0}^{N-1} A(k) \cdot W^{jk} \qquad j = 0, 1, \ldots, N-1. \tag{1}$$

Here W is defined to be the principal N-th root of unity.

$$W = e^{2\pi i/N}, \qquad (i = \sqrt{-1}). \tag{2}$$

The inverse problem can also be solved by the same method, for if the N values of X(j) corresponding to $j = 0, 1, \ldots, N-1$ are given, then the Fourier coefficients appearing in equation (1) are defined by

$$A(k) = \frac{1}{N} \sum_{j=0}^{N-1} X(j)\ W^{-jk} \qquad k = 0, 1, \ldots, N-1 \tag{3}$$

which is similar in form to equation (1).

The following discussion is concerned with the problem as stated in the form given in equation (1).

The straightforward use of equation (1) is equivalent to pre-multiplying the N-component vector A(k) by the NxN matrix W^{jk} to obtain the N-component solution vector X(j). This is easily implemented on ILLIAC IV which computes the components of X in parallel. The total number of operations required would be about N^2 where an operation is considered to be a complex multiplication followed by a complex addition. The algorithm described by Cooley and Tukey* can achieve the same result with much less computation. The number of operations, in the most favorable case, is proportional to N · log N rather than to N^2. It is also economical in storage requirements. These features make it a highly desirable method for this problem, especially for large values of N.

Cooley and Tukey* showed that choosing N to be a power of 2 ($N = 2^m$) has particular advantages for computation on a binary machine. With this choice, the algorithm takes the form of generating iteratively a sequence of m N-component vectors. The first member of the sequence is derived by iteration on the vector A(k) and the final member is the required vector X(j).

It is assumed throughout what follows that the choice $N = 2^m$ has been made.

To define the sequence of vectors the indices are written in binary form.

$$\left.\begin{aligned} j &= j_{m-1} \cdot 2^{m-1} + \ldots + j_1 \cdot 2 + j_0 \\ k &= k_{m-1} \cdot 2^{m-1} + \ldots + k_1 \cdot 2 + k_0 \end{aligned}\right\} \qquad (4a, b)$$

The equation (1) can then be written

$$X(j_{m-1}, \ldots, j_0) = \sum_{k_0} \sum_{k_1} \cdots \sum_{k_{m-1}} A(k_{m-1}, \ldots, k_0) \cdot W^{jk}$$

By evaluating the indicated sums sequentially, the following definition of the sequence A_r ($r = 1, 2, \ldots, m$) is obtained. The notation used is that of Cooley and Tukey except that the iteration parameter is represented by r instead of ℓ, for ease of typing.

*J. W. Cooley and J. W. Tukey: An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation, Vol. 19 (1965). pp. 297-301.

$$A_1(j_0, k_{m-2}, \ldots, k_0) = \sum_{k_{m-1}} A(k_{m-1}, \ldots, k_0) \cdot W^{p_1}$$

$$A_r(j_0, \ldots, j_{r-1}, k_{m-r-1}, \ldots, k_0) = \sum_{k_{m-r}} A_{r-1}(j_0, \ldots, j_{r-2}, k_{m-r}, \ldots, k_0) \cdot W^{p_r} \quad (5)$$

where

$$p_1 = j_0 \cdot k_{m-1} \cdot 2^{m-1}$$

$$p_r = (j_{r-1} \cdot 2^{r-1} + \ldots + j_0)\, k_{m-r} \cdot 2^{m-r} \quad (6)$$

Writing out the two terms of the sum in equation (5) we obtain

$$A_1(j_0, k_{m-2}, \ldots, k_0) = A(0, k_{m-2}, \ldots, k_0) + (-1)^{j_0} A(1, k_{m-2}, \ldots, k_0)$$

$$A_r(j_0, \ldots, j_{r-1}, k_{m-r-1}, \ldots, k_0) = A_{r-1}(j_0, \ldots, j_{r-2}, 0, k_{m-r-1}, \ldots, k_0) + (-1)^{j_{r-1}} A_{r-1}(j_0, \ldots, j_{r-2}, 1, k_{m-r-1}, \ldots, k_0) \cdot W^{q_r} \quad (7)$$

$$r = 2, 3, \ldots, m$$

where

$$q_r = (j_{r-2} \cdot 2^{r-2} + \ldots + j_0) \cdot 2^{m-r} \quad (8)$$

The desired components of X are then defined by the last member of the sequence.

$$X(j_{m-1}, \ldots, j_0) = A_m(j_0, \ldots, j_{m-1}) \quad (9)$$

MACHINE IMPLEMENTATION

It was the suggestion of Cooley and Tukey that the value of

$$A_r(j_0, \ldots, j_{r-1}, k_{m-r-1}, \ldots, k_0)$$

calculated by means of equation (7) be stored in a location whose address is

$$j_0 \cdot 2^{m-1} + \ldots + j_{r-1} \cdot 2^{m-r} + k_{m-r-1} \cdot 2^{m-r-1} + \ldots + k_0.$$

When this is done the storage requirements are minimized and the last array calculated gives the desired Fourier sums, equation (9), in such an order that the index of an X must have its binary bits put in reverse order to yield its index in array A_m.

On any iteration, the components of A_r may be computed in parallel since the calculations defined by equation (7) may be carried out with all values of $j_0, \ldots, j_{r-2}$ and $k_0, \ldots, k_{m-r-1}$ simultaneously.

IMPLEMENTATION ON ILLIAC IV

In order to perform the calculations indicated on ILLIAC IV, it is necessary, on the r-th cycle, to have the values of both

$$A_{r-1}(j_0, \ldots, j_{r-2}, 0, k_{m-r-1}, \ldots, k_0)$$

and

$$A_{r-1}(j_0, \ldots, j_{r-2}, 1, k_{m-r-1}, \ldots, k_0)$$

available in the same PE memory. The value of W^{q_r} must also be available to the PE. One then computes, according to equation (7), the value of A_r with the same two indices.

To obtain the values of A_{r-1} with the desired pair of indices in the same PE memory it may be necessary to shift data from PE to PE during the course of the calculations.

The constant powers of W required by each PE, during the entire course of the computation, are predetermined by the method in which the original coefficients A(k) are distributed within the PE memories and by the scheme adopted for shifting data between PE memories as the computation proceeds. These details will now be discussed.

STORAGE OF THE COEFFICIENTS A(k)

Use is made of the fact that N has been chosen to be a power of 2: $N = 2^m$. It is further assumed that m is greater than 8. To determine the location in which a particular A(k) is stored, the representation of k as a binary number as in equation (4b) is used. Let the last eight bits $(k_7, \ldots, k_0)$ of this binary representation of k define the PE in which A(k) is stored. The last two of these bits (k_1, k_0) determine the number of the quadrant, the preceeding six bits $(k_7, \ldots, k_2)$ determine the number of the PE within the quadrant.

The remaining m-8 bits ($k_{m-1}, \ldots, k_8$) may be interpreted as the address of a storage location within the PE memory. To allow for the storage of both real and imaginary parts of A(k), the real parts can be stored in the indicated location while the imaginary part can be stored in the corresponding location of another block of memory, congruent to the block in which the real parts are stored. The size of each of these blocks of memory will be 2^{m-8} words.

The interpretation of the binary representation of the index k is thus as shown in figure 6-1.

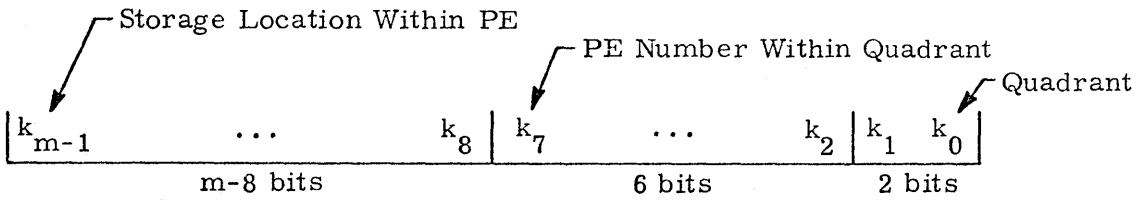


Figure 6-1. Interpretation of k as a Binary Number to Define Storage Location of A(k).

The quadrants of the ILLIAC IV array are numbered 0, 1, 2, 3. The numbering of the PE's within the quadrant is shown in figure 6-2. If p is the PE number then the last 3 bits of the binary representation of p are the column and the first 3 bits are the row numbers. When the initial data is stored as described, its distribution throughout the arrays is shown in figure 6-3. The number shown are k mod 256.

Row \ Col	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16							
3	24							
4	32							
5	40							
6	48							
7	56	57	58	59	60	61	62	63

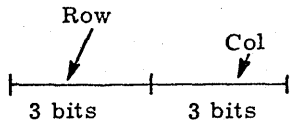


Figure 6-2. The Numbering of the PE's within a Quadrant

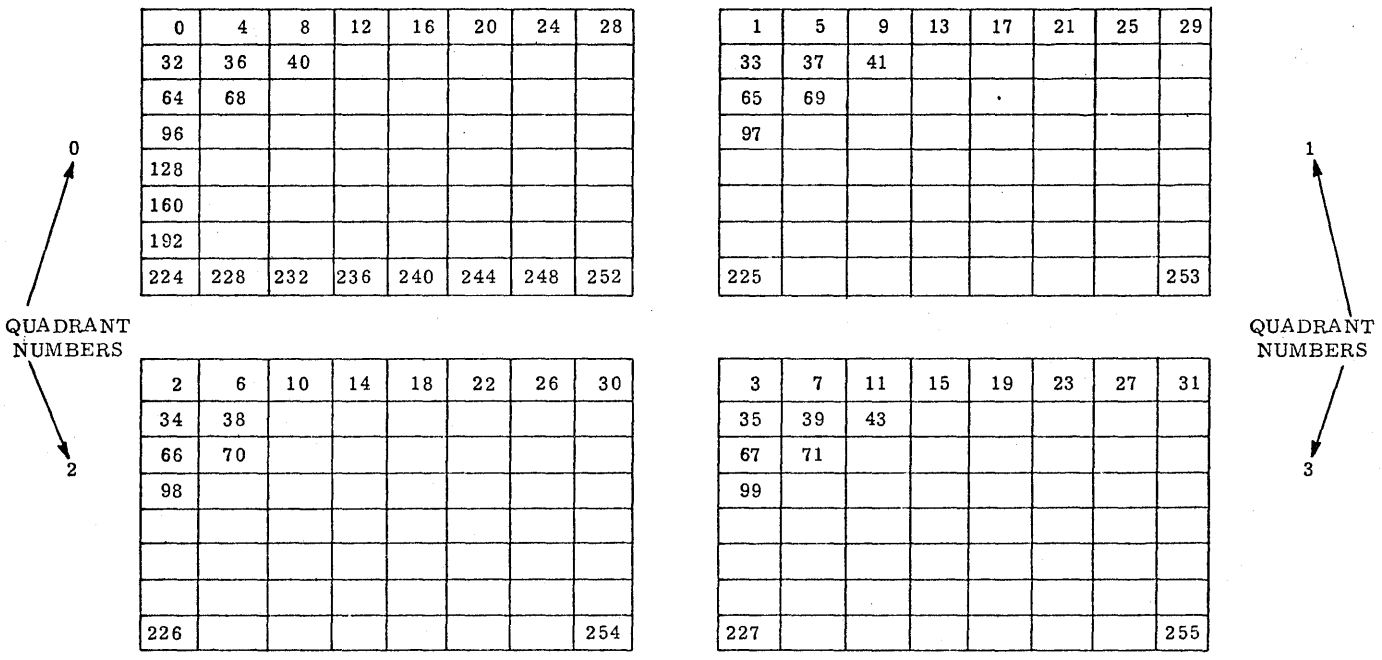


Figure 6-3. The Distribution of the Coefficients A(k) in the Array Initially

COMPUTATION AND STORAGE OF INTERMEDIATE RESULTS

The calculations fall naturally into two parts, the first m-8 iterations and the final 8 iterations. In the following description, the binary bits (always m in number) of the locations referred to are to be interpreted in the same manner as the binary representation of k just described.

The first m-8 cycles $r = 1, 2, \ldots, m-8$.

1. Calculate $A_r(j_0, \ldots, j_{r-1}, k_{m-r-1}, \ldots k_0)$, by use of equation (7).

2. Store the result in location $(j_0, \ldots, j_{r-1}, k_{m-r-1}, \ldots, k_0)$, i.e., at address $(j_0, \ldots, j_{r-1}, k_{m-r-1}, \ldots, k_8)$ of PE $(k_7, \ldots, k_0)$. Note that on the (m-8)th cycle it is address $(j_0, \ldots, j_{m-9})$ of PE $(k_7, \ldots, k_0)$ that is meant.

The calculation is performed for all values of $(j_0, \ldots, j_{r-1})$ and of $(k_{m-r-1}, \ldots, k_0)$. It is seen from equation (7) that for the computation of the A_r for any index, the storage locations defined for the quantities on both the left and right hand sides of the equation are in the same PE memory (the last 8 bits are the same). Hence no transfer of data between PE's is required. Thus all PE's compute in parallel and, on each cycle, each PE computes the value of A_r for 2^{m-8} different values of the index.

The final 8 cycles $r = m-7, m-6, \ldots, m$.

1. Shift $A_{r-1}(j_0, \ldots, j_{r-2}, k_{m-r}, \ldots, k_0)$ from location $(j_0, \ldots, j_{m-10}, j_{r-2})$ in PE $(j_{m-9}, \ldots, j_{r-3}, k_{m-r}, \ldots, k_0)$ to location $(j_0, \ldots, j_{m-10}, k_{m-r})$ in PE $(j_{m-9}, \ldots, j_{r-2}, k_{m-r-1}, \ldots, k_0)$. Note that on the (m-7)th cycle the shift meant is from location $(j_0, \ldots, j_{m-9})$ in PE $(k_7, \ldots, k_0)$ to location $(j_0, \ldots, j_{m-10}, k_7)$ in PE $(j_{m-9}, k_6, \ldots, k_0)$. Note also, on the mth cycle the shift meant is from location $(j_0, \ldots, j_{m-10}, j_{m-2})$ in PE $(j_{m-9}, \ldots, j_{m-3}, k_0)$ to location $(j_0, \ldots, j_{m-10}, k_0)$ in PE $(j_{m-9}, \ldots, j_{m-2})$.

2. Calculate $A_r(j_0, \ldots, j_{r-1}, k_{m-r-1}, \ldots, k_0)$ by use of equation (7). Note on the mth cycle it is $A_r(j_0, \ldots, j_{r-1})$ that is calculated.

3. Store the result in location $(j_0, \ldots, j_{m-10}, j_{r-1})$ in PE $(j_{m-9}, \ldots, j_{r-2}, k_{m-r-1}, \ldots, k_0)$. Note on the mth cycle it is location $(j_0, \ldots, j_{m-10}, j_{m-1})$ in PE $(j_{m-9}, \ldots, j_{m-2})$ that is meant.

Again the calculation is performed for all values of $(j_0, \ldots, j_{r-1})$ and of $(k_{m-r-1}, \ldots, k_0)$. The shift called for in step 1 is designed to bring together in the same PE the values of A_{r-1} with the two indices. They differ only in the r-th bit appearing on the right-hand side of equation (7). Once the shifts have been accomplished all PE's compute in parallel and each computes the values of A_r for 2^{m-8} values of the index.

The shifts required in step 1 are determined by the values of the bits j_{r-2} and k_{m-r}. Since only four possible combinations of the values of these bits are possible, the corresponding shifts may be tabulated.

Table 6-1. Shifts Required for Combinations of Bits j_{r-2} and k_{m-r}

Bit		Shift
j_{r-2}	k_{m-r}	
0	0	No shift.
0	1	Shift to PE of lower number (-2^{m-r}). Increase location by 1.
1	0	Shift to PE of higher number ($+2^{m-r}$). Decrease location by 1.
1	1	No shift.

Since the possible combinations of bit values occur with equal frequency it is seen from the above table that on any cycle (r) precisely half the data has to be shifted by inter-PE shifting. Of the data that is shifted, half goes to a PE of higher number ($+2^{m-r}$) and half goes to a PE of lower number (-2^{m-r}). This shifting is such that on cycle r, two PE's whose number in the array differ only in the r-(m-8) bit position exchange a word of data, for each two words they contain.

The required shifting of data between PE memories is accomplished by one or two routing instructions. In the first cycle requiring such a shift each PE in the block of 128 PE's (those numbered 0-127) sends and receives words from the corresponding PE in the second block of 128 PE's (128-255). Examination of the PE numbering system of figure 6-3 shows that the first four rows of PE's in each quadrant exchange words with the second four rows. The end-around cylindrical connection of the PE's on the North and South edges of each quadrant are such that sending and receiving can be accomplished in one instruction, since all PE's in a quadrant shift a word 4 rows South, end-around, simultaneously.

In the second cycle requiring a shift, each PE in 2 blocks of 64 PE's (0-63; 128-191) exchanges words with the corresponding PE in the corresponding block of the remaining 2 blocks of 64 PE's (64-127; 192-255). This requires the first two rows of PE's within a quadrant to exchange words with the second two rows and the third pair of rows to exchange words with the fourth pair of rows. In this case

the end-around connection cannot be used and the exchange takes place under mode control in the two parts. First, rows 1, 2, 5, 6 transmit data two rows South to rows 3, 4, 7, 8 respectively. Second, rows 3, 4, 7, 8 transmit data two rows North to rows 1, 2, 5, 6 respectively.

In the third cycle information is exchanged between adjacent rows.

The following three cycles repeat the same pattern of shifts but between columns rather than rows. Next a shift of 8 rows is required. This involves the whole array as data moves from quadrant to quadrant for the first time. Use could be made of the end-around cyclindinal connection of the whole array, as in the first shift described. The final shift is similar, being of distance 8 between columns. This pattern of shifts is listed in table 6-2.

Table 6-2. Shifting Required for the Final Eight Iterations

Cycle	Blocks of PE's		Distance of Shift Required
	Number	Size	Nearest Number
m-7	2	128	4* NS
m-6	4	64	2 NS
m-5	8	32	1 NS
m-4	16	16	4** WE
m-3	32	8	2 WE
m-2	64	4	1 WE
m-1	128	2	8* NS
m	256	1	8** WE

* Denotes that end-around connectivity may be used to allow both of the shifts to occur simultaneously.

** Shifts can also use end-around connectivity, but will require one more step in routing than "*" shifts, because of the differences in edge connectivity patterns.

We now consider the powers of W that have to be available in each PE. According to equation (7) the computation on the r-th iteration, of $A_r(j_0, \ldots, j_{r-1}, k_{m-r-1}, \ldots, k_0)$ requires the use of W^{qr} where q_r is given by equation (8). With the data storage scheme described the address of A_r within the PE determines the power of W required in its calculation. According to equation (8), on the first m-8 iterations, put the first r-1 bits of the address in reverse and multiply the resultant number by 2^{m-r}. This gives the required power of W. On the first iteration, no bits are selected by this rule--corresponding to W^0. For these iterations all PE's require the same power of the W at the same time. Thus, these powers should be broadcast, and not stored repetitively in each PE.

On the final 8 iterations one has $A_{r-1}(j_0, \ldots, j_{r-2}\, k_{m-s}, \ldots, k_0)$ stored in location $\underline{j_0, \ldots, j_{m-10}}, k_{m-r})$ in PE $(\underline{j_{m-9}, \ldots, j_{r-2}}, k_{m-r-1}, \ldots, k_0)$. After the required shifting has taken place, take the r-1 bits that have been underlined (the first m-9 in the memory address followed by the first r-m+8 in the PE number), invert these bits and multiply the number obtained by 2^{m-r}. This is the power of W required. Since these powers of W depend on the PE number they should be stored within the PE memories.

On each iteration one power of W for each pair of indices of A_r within the PE is required. The number of such pairs is 2^{m-9} and the number of iterations in this mode is 8. Thus 2^{m-6} values of powers of W are required by each PE.

As in the standard algorithm, at the completion of the m-th iteration, to find the location of X(j) in the A_m array, one interprets the bits of j in reverse as a location in the PE memories. However, after reversing the bits of j it is necessary to shift the final bit (j_{m-1}) left eight places (to between j_{m-10} and j_{m-9}) before terpreting the location as in figure 6-1.

COMPUTATIONS REQUIRED

To give some idea of the magnitudes involved some figures are given here for $N = 4096 = 2^{12}$. (m = 12).

1. Number of iterations required (m) = 12

2. Number of coefficients A(k) in each PE(2^{m-8}) = 16

3. Number of values of W stored in each PE (2^{m-6}) = 64

4. Computation required for each pair of coefficients ($2^{m-9} = 8$) per PE, is indicated by equation (7), 1 complex multiplication and 2 complex additions. This in terms of real arithmetic, amounts to 4 multiplications and 6 additions for each pair of coefficients. On the final 8 iterations data transfers occur and 2 words (real and imaginary part of one of the pair) are transmitted and 2 words received for this amount of calculation. Since there are 8 pairs of coefficients

in each PE, for each iteration a PE:

1. Transmits 16 words of data
2. Receives 16 words of data

} Not required on first 4 of the 12 iterations

3. Performs 32 multiplications
4. Performs 48 additions.

During the first 4 iterations one is effectively solving 256 problems in parallel (one in each PE), each problem being of size N = 16. During the final 8 iterations one is solving equivalently, 8 problems, each problem being of size N = 512. These last problems are distributed uniformly throughout the array.

SECTION VII

CIRCUIT DESIGN - THE ECL CIRCUIT

The basic logical circuit for ILLIAC IV is the current switch, or ECL circuit, shown in figure 7-1. Logical operations can be performed in this circuit by supplying input transistors in parallel, by tying collectors together into a common collector resistor, and by connecting together the output emitters. Gating on the input transistors appears as "not OR" for positive-going signals when seen at the inverting output "b," or as OR when seen at the noninverting output, "a." For negative-going signals, gating at the input transistors appears as "not AND" and AND, respectively, at the inverting and noninverting outputs. Logic gating can also be done by sharing a single resistor among otherwise independent collectors. Such sharing produces an OR for negative signals or an AND for positive signals, and would appear to be a way of adding another gate to the logic without adding any components or any delay. However, one must take care of the case that two collectors are delivering current into the resistor simultaneously, either by clamping the voltage, which costs components, or by ensuring in the logical design that no more than one current flows at any one time. Mixing outputs by tying the emitters together performs exactly the same function as mixing inputs of the next stage in the input transistor, namely negative AND or positive OR. Implementation of any logic equation using such gates can always be found by translating the AND-OR description implicit in the lgical equation into a NAND-NAND description, level for level. As reference to the logical design of the processing element elsewhere in this report will show, designs can often be simplified considerably from the directly translated version.

The use of ECL circuits differs depending upon the amount of wiring which must be driven and the speed to be obtained. When integrating within the semiconductor array, freedom to use the ECL gates just described is virtually unlimited. When any wiring is involved, however, the low impedance point, namely the emitter output, is used as the source when signals must be sent along conductors from one circuit package to another. Further, fanout suffers on signals which must leave the circuit package because of the need to supply damping and terminating resistors for the wiring.

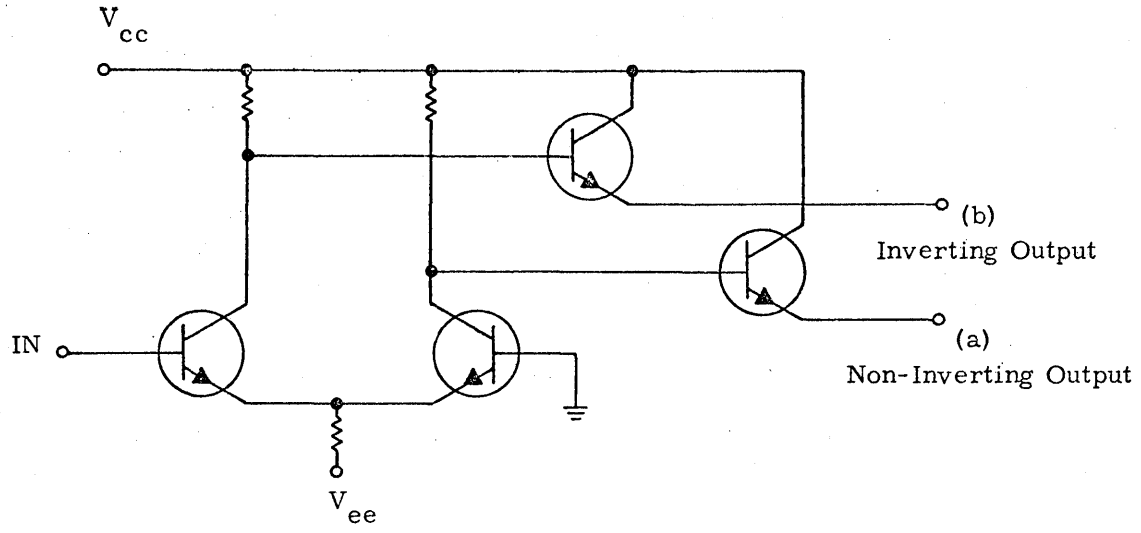
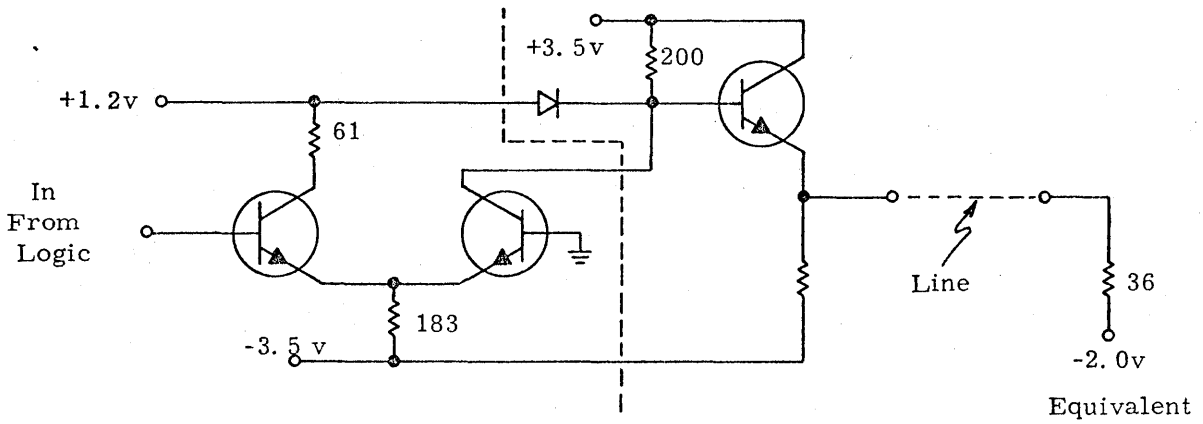
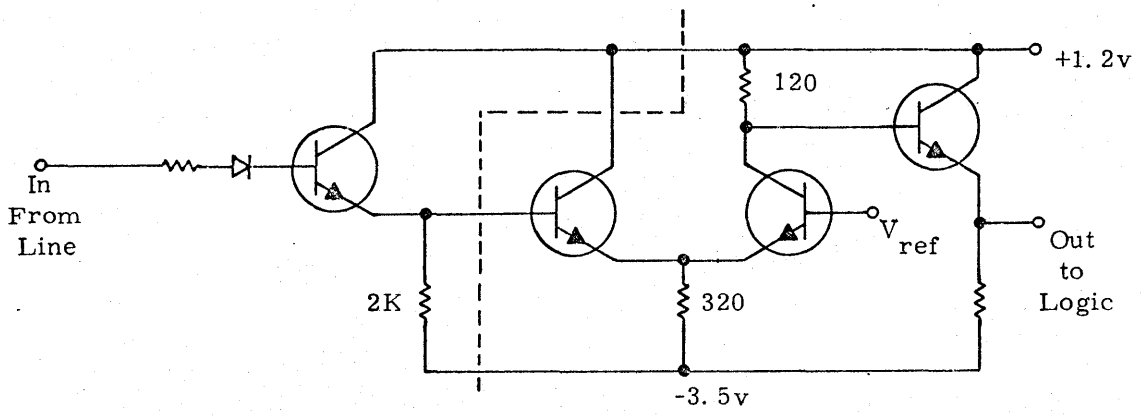


Figure 7-1. ECL Gate, Schematic Diagram



Note: All Resistance Values in Ohms.

Figure 7-2. Driver, Schematic Diagram



Note: All Resistance Values in Ohms.

Figure 7-3. Receiver, Schematic Diagram

At the collector, signal levels are approximately +1. 2v and +0. 4v. The swing is set essentially by the current in the emitter, it is related therefore to V_{ee}, V_{cc} and resistor ratios. The more positive level departs from V_{cc} by an amount determined by base current in the output transistor, a few percent at most of the "on" current. At the "output" emitter in figure 7-1, the voltage is downshifted by an amount equal to the base-to-emitter voltage of the output transistor. The output therefore swings from +0. 4v approximately to -0. 4v approximately. These output voltages are made meaningful with respect to the nominally zero volt input threshold whatever gate receives the.

Signals which must travel considerable distance, such as between cabinets, for instance, may need more margin than that supplied in the signals at the output of the ECL gates. The extra margin is needed to overcome distortions in the signal due to imperfect impedance matching in the wiring, unwanted components due to crosstalk, noise from external sources, and discrepancies in temperature and perhaps even in "zero volts" between the two cabinets. The requirements for these non-ECL signals appear to be as follows:

- Compatibility with ECL levels at the input of the driver and the output of the receiver
- Larger signal swing (3. 0v based upon previous Burroughs experience)
- Drive capability for several transmission line characteris impedances in parallel (at least two; or 36 ohms load on each signal, if 72-ohm line is used)
- Fanout of 64 (from control unit to all 64 P. E's); this implies an input impedance of over 2. 3k ohms per receiver if 32 receivers are to be attached to a single 72-ohm line.)

A driver circuit which satisfies these requirements is shown in figure 7-2. Note that the portion of the circuit to the left of the dotted line in the driver circuit is identical (except for a somewhat lower impedance level) to our standard ECL gate. When this driver circuit is implemented as a portion of a large-scale integrated array, the portion of the circuit on the left, which is "the same" as the standard ECL circuit, is available for performing some logic function. Only the non-inverted output, which feeds the large-swing signal, would be unavailable for normal ECL use.

A receiver circuit which satisfies these requirements is shown in figure 7-3. Note that the portion of the circuit to the right of the dotted line in the receiver circuit is identical to our standard ECL gate. When this receiver circuit is implemented as a portion of a large-scale intregrated array, the portion of the circuit on the right, which is the same as the standard ECL circuit, is available for performing logic functions within the array.

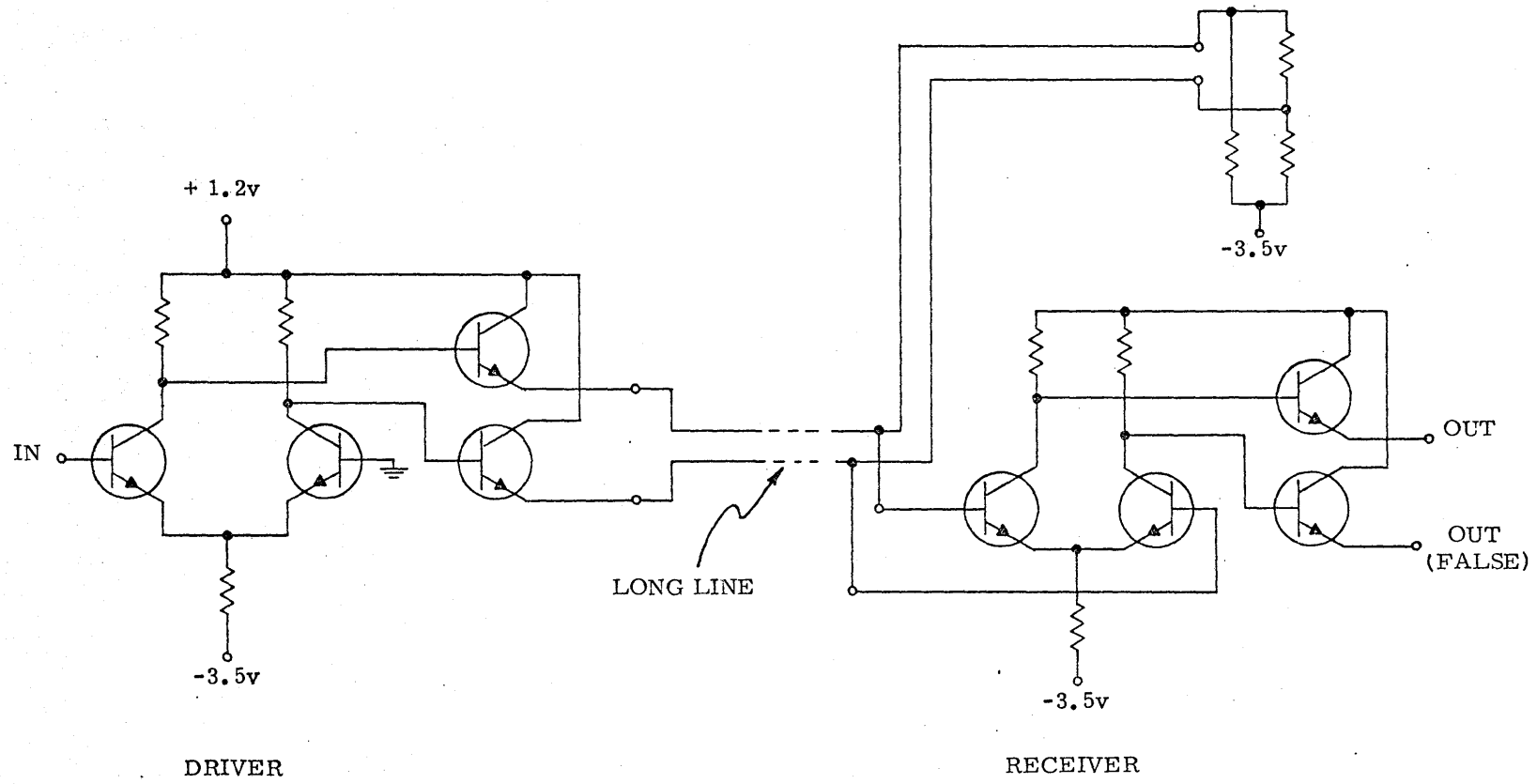


Figure 7-4. ECL Driver-Receiver, Balanced Signals, Schematic Diagram

The signal across the interface, in this system, swings from +2. 5v to -0. 4v. Threshold is at 1. 2v, at room temperature, and has a slight negative temperature coefficient. Since rise times no faster than 15 ns are of interest, the gain of the transistors at 25 mc or thereabout controls the input impedance of the receiver. With transistors whose cutoffs are in the hundreds of megacycles, gains of 30 are easily available. The +3. 5v supply can be disabled to control intercabinet transfers.

An alternative dirver-receiver circuit has been suggested in which each signal is transmitted balanced with respect to ground. This circuit is shown in figure 7-4. It has a signal swing of 1. 6v, the differential signal between the two outputs, but makes up for lowered signal swing with increased noise immunity. The noise immunity is almost equal to half the minimum signal swing, compared to a noise immunity of 30% to 35% of the signal swing in the system exemplified by the driver and receiver of figures 7-2 and 7-3. This scheme has a further advantage in rejecting noise, in that some noise sources tend to induce a common mode component in the line. This single-ended scheme rejects common mode noise essentially perfectly up to some maximum amplitude, while the scheme depends on the coupling between the two conductors of the line to induce a noise component in one conductor equal to the noise component induced on the other by some external source. The latter scheme is extremely effective, but not as effective as balanced signals.

An advantage of the balanced signal driver and receiver is that they are identical in design and fabrication with standard ECL gates.

Disadvantages of the balanced signal scheme are severe for certain proposed uses. In particular, it is not possible to put more than one driver on one signal wire. For the connections from the PE's back to the memory access buffer, it would be necessary to give each PE its own expensive wire back to its own private receiver at the memory access buffer. Using the unbalanced, 3. 0 v signal, one can combine all eight drivers on a single data line. This defect will be general, whenever several data sources converge on a common destination.

A second defect of the balanced signal scheme arises because the wiring must be balanced with respect to ground. Thus one must use pairs of wires, either twisted pair, or shielded twisted pair. Twisted pair is inferior to coaxial cable or ribbon cable in terms of crosstalk; shielded twisted pair is considerably more expensive both to buy and to install than coaxial cable and ribbon cable. Even unshielded twisted pair, when procurred in belted form, can be surprisingly expensive.

A third limitation of the balanced signal scheme arises because of the difficulty in designing for low output impedance. As described to us, the drive capability of the balanced driver was only sufficient to drive a single transmission line. In the case of data being transmitted from one PE to neighboring PE's, the data transmission paths go in both directions from the transmitting PE. For optimum performance, it is necessary to drive two transmission lines, one going in each

direction. Therefore, two balanced driver circuits are required to handle the signal. Greater fan-out at the receiver end is also available with the unbalanced design.

The conclusion is that either driver and receiver design is satisfactory for use in cables up to some maximum length, where not more than one driver per signal set is required. Ordinary twisted pair can probably be driven by the balanced driver design in medium length runs of up to 10 or 30 feet where the unbalanced driver would require coaxial cable or the equivalent. Beyond that, in either case, higher quality wire is required, such as ribbon cable using three wires per signal, coaxial cable, or shielded twisted pair. For this last class of signals, single-ended signals would be more economical of wiring, would be directly compatible with popular types of "foreign" logical circuitry which is likely to be found in external equipment, and would be more compatible with any requirement such as r. f. filtering. A table of "long lines" signals is in table 7-1.

Table 7-1 contains our conclusions on the implementation of such "long lines" signals. Either balanced or unbalanced signals will be acceptable within each quadrant, assuming each quadrant to be packaged within a single unit, a set of

Table 7-1. Signals Requiring Driver and Receiver

Class of Signals	Estimated Length (feet)	Suitable Driver Design	Comments
Pe to PE (same cabinet)	short	---	no driver needed, standard ECL signals suitable
Pe to PE (different cabinets)	6	either	----
PE to I/O Buffer	30	either	----
PE to memory access buffer	30	single ended	drivers must be ORable
PE from CU, control signals and mode control	30	either	----
CU to CU, different quadrants	80	single ended	economically more attractive wiring
CU to peripheral devices	200	single ended	compatibility with foreign signals.

cabinets bolted together. For longer runs, the lower price of coaxial cable, and the greater compatibility of unbalanced signals at a "foreign" interface seem to call for the use of 3.0v unbalanced signals between quadrants, and from the array processor to the outside.

Figure 7-5 shows an example of how the unbalanced system might be used to distribute control signals from the CU to the 64 PE's assuming that there is one receiver in each PE. A fanout of 64, as shown, may be beyond the capability of either scheme at the required speed, especially the balanced signal scheme. However, the grouping of the PE s is such that one receiver should do for four or eight PE's.

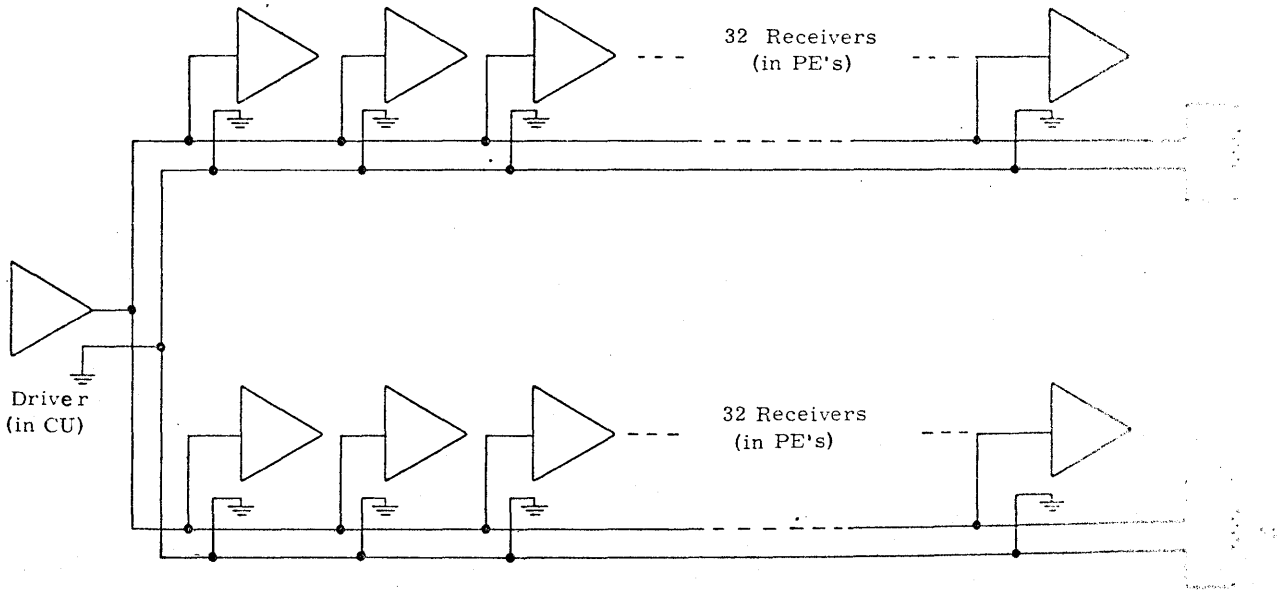


Figure 7-5. Use of Drivers and Receivers for Distributing Control Signals from CU to PE's