IMPLEMENTATION OF A

CELLULAR COMPUTING MEMORY ARRAY

IMPLEMENTATION OF A CELLULAR COMPUTING MEMORY ARRAY

BY

FRANCIS K.W. HO, B.ENG

A Thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the Degree of MASTER OF ENGINEERING

MCMASTER UNIVERSITY

February 1971

MASTER OF ENGINEERING (1971)
(Electrical Engineering)

McMASTER UNIVERSITY
Hamilton, Ontario.

TITLE: IMPLEMENTATION OF A CELLULAR COMPUTING MEMORY ARRAY

AUTHOR: FRANCIS K.W. HO, B. ENG. (McMaster University)

SUPERVISOR: Dr. E. Della Torre

NUMBER OF PAGES: x, 98.

SCOPE AND CONTENTS:

The theory of Cellular Logic in Memory Arrays is discussed. Such an Array has been implemented and its system design is described.

Supporting systems such as control and computer programs to operate the Array have been developed and are described.

ABSTRACT

A Cellular Computing Memory Array consisting of twenty cells of three words plus logic per cell has been constructed and interfaced to a digital computer. Both arithmetic and logic operations can be performed between words of a cell and those of adjacent cells, the results of which may be stored in any of the three words of the cell. The cells are organized in a two dimensional array so that each cell can communicate with the four nearest neighbours. In addition to the ordinary communication between cells to perform operations on data in adjacent cells, data can be transferred in the memory plane on a mass basis.

ACKNOWLEDGEMENTS

The author wishes to thank Dr. E. Della Torre for his guidance and encouragement in the preparation of this thesis.

Thanks are also due, to V. Diciccio and J. Harris Jr. , who as summer assistants helped in constructing the system.

The financial assistant provided by the Department of Electrical Engineering is gratefully acknowledged.

This work is dedicated to my parents and to D. Duggan, a friend.

TABLE OF CONTENTS PAGE

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Since the birth of the electronic computer some twenty five years ago, and particularly in the last ten years, great advancements have been made particularly in the speed of calculations and in the techniques of using computers. Computer languages and software have been developed greatly, allowing easy use of complex computer systems, and computation speeds have increased tremendously due to the development of solid state technology. One area, however, which has resisted any significant change, is the basic organization of the computer. The conventional organization where a large memory is associated with a sophisticated central processor is unquestionably, very useful and will remain so for a long time to come.

There are some types of problems, however, where despite the great speeds of present computers, the sequential processing of data is inappropriate and dreadfully slow. Such problems usually involve large masses of data on which identical operations are to be performed. For such problems, the drawback is due, not just to having to repeat a large number of computations but also to the cumulative transfer times of data from memory to processor and then, of the results or data back to memory for storage. Improvements can be made, of course by producing more efficient software or by using high speed hardware. But such steps would not be really significant and will be limited by the state of the technology. The access time of data from memory, for example, could be reduced by replacing the popular ferrite core memories by the faster solid state semiconductor memories.

The solution, not surprisingly lies with reorganizing the computer or part of the computer to operate in a parallel fashion. The concept is quite simple; increase the number of arithmetic processors, each working with a different section of memory and the overall computation time is shorterned accordingly. The full power of this concept becomes apparent when we extend it to the limit, associating one processor with one memory word or more practically with a unique set of memory words. This arrangement makes possible the concept of operating on data in situ, without having first to transfer data from memory to processor. The time saving here is twofold, in eliminating the need to repeat sequences of operations and in not having to move data in and out of memory.

Such an organization was proposed as early as 1958 by Unger (1) when he saw that problems involving blocks of data which are locally correlated could be handled with much greater efficiency by operating on the data in a parallel fashion without having to sequentially scan neighbouring data for correlated information. In 1962 Slotnick (2), based on Unger's work, proposed Solomon. Solomon was to be a highly parallel computer organized in a rectangular array where each cell of the array had arithmetic capabilities and was capable of communicating with its four nearest neighbours. These, together with proposals made by Lee (4), Paull (5) and Holland (6) formed the basis which led to a steady development in the concept of cells with arithmetic capabilities, and organized in arrays, also known as CLIM arrays (Cellular Logic in Memory Arrays).

The proposals of Lee and Paull have been commercially realized and are easily available now, but only a few experimental versions of the more sophisticated Solomon type systems have been realized. Some of these include the ILLIAC IV (9), the Berkeley Array Processor (10) and a cellular APL Computer (14). These were designed with specific areas of application in mind, indicating the possible wide areas of application of CLIM arrays. In particular, the Berkeley Processor was designed to perform the operations of correlation, convolution, matrix multiplication and a variation of the Fast Fourier Transform; the APL (A Programming Language) computer was designed to perform efficient execution of APL programs and the ILLIAC IV was designed as a more general purpose machine to process data in bulk.

GENERAL THEORY

We see then that the concept of parallel processing shows promise in the critical areas of data processing which requires identical operations performed on large sets of data, and typical of this area are problems in matrix operations, scaling and partial differential equations, which even the fastest of conventional computers are unable to solve in a reasonable amount of time. What makes the concept of parallel processing even more exciting is its potential to operate in real time, on the time averaging functions such as correlation, autocorrelation, convolution, Fast Fourier Transforms and recursive filtering. These characteristics makes the concept of parallel processing very attractive in a vast variety of applications such as in communications, optimization, control requiring fast response such as in guidance, trajectory calculations and even weather forecasting.

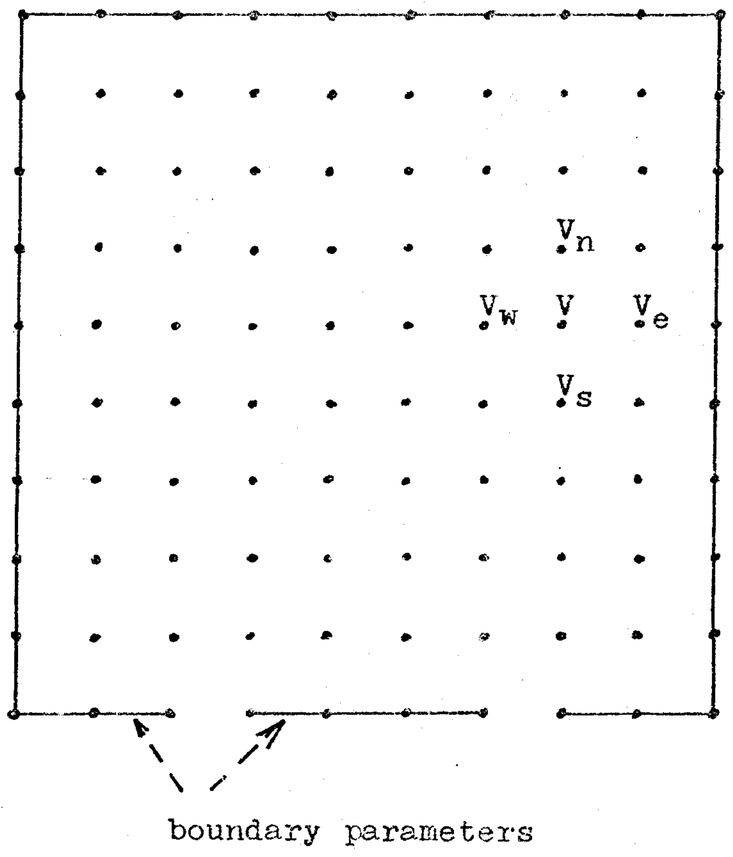
For the various functions, of course, different requirements are put on the way in which the processing array is organized and also on the capabilities that each cell of the array must have. We may then characterize CLIM cells and arrays under the following headings.

CELL COMPLEXITY

In general, depending on the nature of the problem, the complexity of the cell may vary from the simplest logic function capability to a complete system capable of elaborate arithmetic and logic operations. The type of cell, for example, required to solve partial differential equations as proposed by Slotnick is considered fairly complex. Each cell has two storage words and possess fixed point arithmetic and some logic abilities. The iteration procedure for solving a two dimensional Laplace equation with cartesian coordinates involves relaxing the value at each point in a bounded region, to the average value of its four neighbours, to the north, the south, the east and west. This is illustrated in Fig. 1.1. An example of a simple cell is the associative memory proposed by Lee (4), where associated with each memory word is a logic unit which allows for fast retrieval of data from that word by associating an address with part of that memory word.

ARRAY GEOMETRY

Several geometric organizations of cells are possible, the simplest of which is the linear array. This organization, shown in Fig. 1.2, is being used in the associative memories of Lee and Paull and in the Berkeley Array processor. The linear array should be suitable for time averaging functions and other applications where one variable dimension is involved. Another geometry is the rectangular array which is suitable to operate on matrices and on

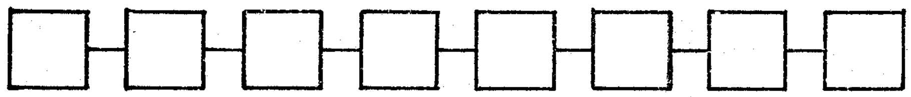


Assuming that the values of the boundaries are known, the numerical solution to the Laplacian Field problem can be calculated at each matrix point(representing a cell) by successive relaxation.

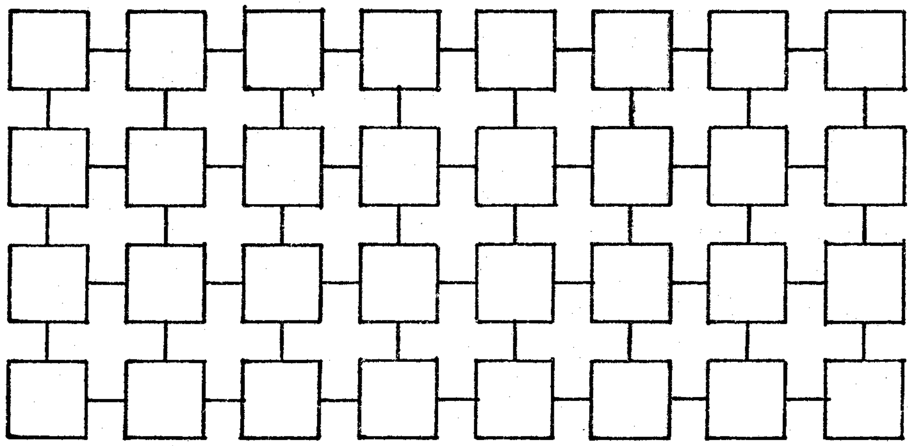
An arbitary point 'V' is relaxed by :-

$$V = 1/4 (V_n + V_s + V_e + V_w)$$

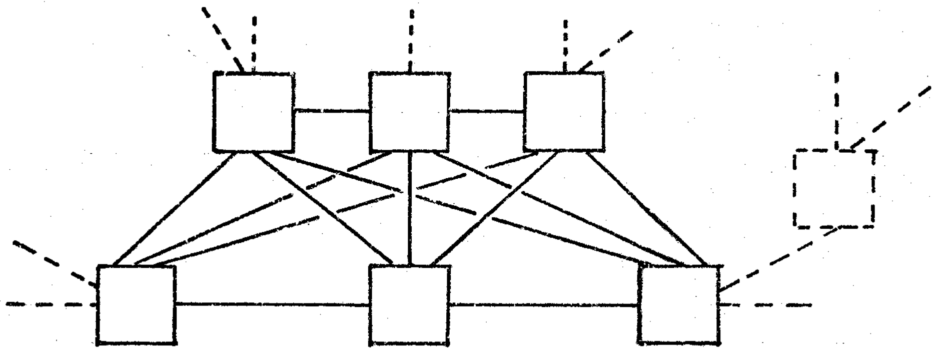
FIGURE 1-1. SOLUTION OF A LAPLACIAN FIELD PROBLEM USING A 10X10 ARRAY OF CELLS.



a) LINEAR ARRAY OF CELLS



b) RECTANGULAR ARRAY



c) N-DIMENSIONAL ARRAY

FIGURE 1.2 CELL GEOMETRY

partial differential equations. An n-dimensional geometry is also possible where cells may be arbitrarily connected. This type of structure could show great promise in a multi-variable problem where, for example, computations can be made while all dimensions are varied simultaneously, to converge to an optimum. Cells for such a structure would probably be highly complex.

The above arrays represent very basic structures, and variations of these are possible. The rectangular array can be extended to a cubic structure, and branches or sub-arrays can be added on to the linear array.

NEIGHBOURS

An important characteristic of parallel processing arrays is the 'neighbour' relationship between cells. The term 'neighbour' is rather loosely defined, but generally includes those cells with which communication is possible for the exchange of control or data information. In other words, mere geometric proximity does not make two cells neighbours. In the rectangular array for example, a cell can have four neighbours, to the north, south, east and west also defined as above, below, left and right, as in the array used to solve Laplace equations. However, if the application requires it, the north-east, north-west, south-east and south-west cells could also be included in the neighbourhood. An interesting proposal, by J. Holland (6) is an example of a structure in which the neighbourhood is redefinable. Here, by program control, new paths are built by making connections via intervening cells. This would give us a versatile array, but would involve highly complex cells.

This concept of being able to communicate with neighbour cells is a prime contribution to the power of array processors especially when the data sets are locally or spatially correlated.

CONTROL

Another characteristic of parallel processing arrays is the degree to which control of the cell functions is distributed between a central controller and a localized in cell controller. The simpler and more common is where control is highly centralized. In this type of system, all cells perform the same operations as determined completely by the central controller. Included in this group are the processors that handle matrices, partial differential equations and time averaging functions mentioned earlier. At most, the above type of cells may possess the ability to ignore control instructions to limit the area of operation. The proposals of Unger, Lee and Slotnick are all of type. The Holland proposal is an example where control is highly localized. Here, the cell operation executed is determined mainly by control information contained locally in the cell. This gives it the advantage that independent computations may be executed simultaneously, and at the same time enjoy the other benefits of array organization.

GENERAL

One obvious characteristic that a parallel processor should have, for versatility and for ease in implementation, is that all cells should be identical. They should vary in function only in what control information is put in them or is presented to them. This then suggests that cells should be designed with general purpose applications in mind, and may, for some applications be more complex than required.

It is with this general purpose application in mind that this thesis project was pursued. The cell to be described, referred to as a Computing Memory Cell possesses full arithmetic and logic capabilities as well as a degree of local control.

SUMMARY

The aim of this project is to implement a workable system based on the parallel processing characteristics of

CLIM arrays. A rectangular array was built and interfaced to operate under the control of a PDP8 computer. A set of operate programs was developed to control the functions of the array. Since the control of the array was quite elaborate, a special purpose control unit was implemented to perform as much of the controlling as possible. The actual cell was designed around a set of arithmetic and logic parameters, this set being as extensive as was practical, and had to at least enable the array to be used in the solution of Laplace's equation.

In perspective, the system developed has a moderately complex cell which can perform both arithmetic and logic operations between words of the cell. Control is essentially centralized, but has more local autonomy, for example, than the cell proposed by Slotnick. The geometry is a rectangular array with a fixed neighbourhood where the neighbours consists of the cells to the north, south, east and west.

CHAPTER 2

THE COMPUTING MEMORY CELL

The work on the concept of Cellular Logic In Memory Arrays that was done by Lawrence (15) in 1969, consisted of the design of a computing memory cell, four of which were implemented in order to investigate the workings and limitations of such cells. These cells consisting of two memory words associated with two processors were organized in a linear array. Each cell was capable of performing fixed point arithmetic operations between the two words under suitable control. The arithmetic functions included addition, subtraction, division by 2^n ($n=1,2,--10$) and multiplication. This project was developed to a state whereby the cell functions could be performed, largely by manual setting of the cell control lines, and initiating the operation cycle, also manually.

In attempting to improve the cell design and to develop a workable array, it was necessary to investigate the basic questions of what a cell must be capable of doing and what structure and intercommunications to organize the cells.

Obviously, the basic set of arithmetic operations used by Lawrence, also proposed by Slotnick and others, was necessary. In addition, a set of Boolean operations would be necessary if the design was to be used as a general purpose device. It was decided to improve intercell communication by enabling operations to be performed between data in neighbouring cells without having to transfer the data. It was also decided to add a 'convergence check' capability. This would be used for terminating iterative operations when there are no longer significant changes in data.

Each cell needs a minimum of one arithmetic and logic processor and at least two memory words to be workable, but the two word per cell organization of Lawrence proved to be awkward, particularly when multiplication had to be performed and when intercell data transfers were to be made. Adding a third word would be a great improvement, and since there was no apparant need for a fourth word, a three word cell was decided upon. These then, were the basic parameters around which the computing memory cell was designed.

The cells were organized in a rectangular array so that it would be suitable for solving two dimensional problems. However, the array could be converted to a linear array quite simply. Since control was to be highly centralized, an elaborate control unit was built, which was capable of controlling the execution of all arithmetic, logic and input - output operations automatically. All this was interfaced to a PDP8 computer which served as the main controller, storing all data and iteration programs.

The complete list of the basic characteristics of the realized system is as follows:-

1) Each cell can perform the following functions between words of the cell.
    a) addition
    b) substraction
    c) multiplication
    d) division by 2^n
    e) logical AND
    f) logical OR
    g) logical EXCLUSIVE OR
    h) complementing
2) Each cell has an 'inhibit' feature which enables the cell to ignore control instructions
3) Each cell has a 'convergence check' feature which enables the cell to test for significant changes in computed values.
4) The cells are organized in a rectangular array.
5) The neighbours of a cell, except for the boundary cells, are the cells to the north, south, east and

west.

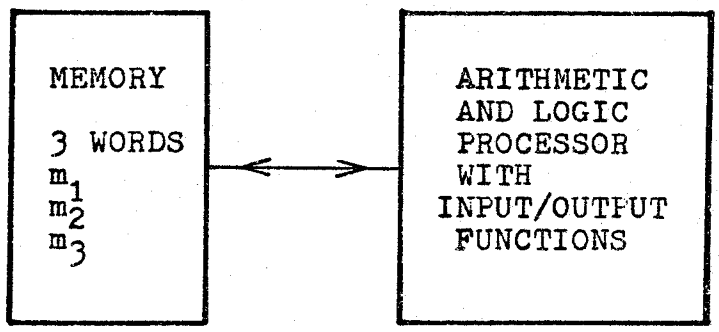
6) Control of the array is highly centralized.

7) The control unit generates all the necessary logic states to execute the cell functions.

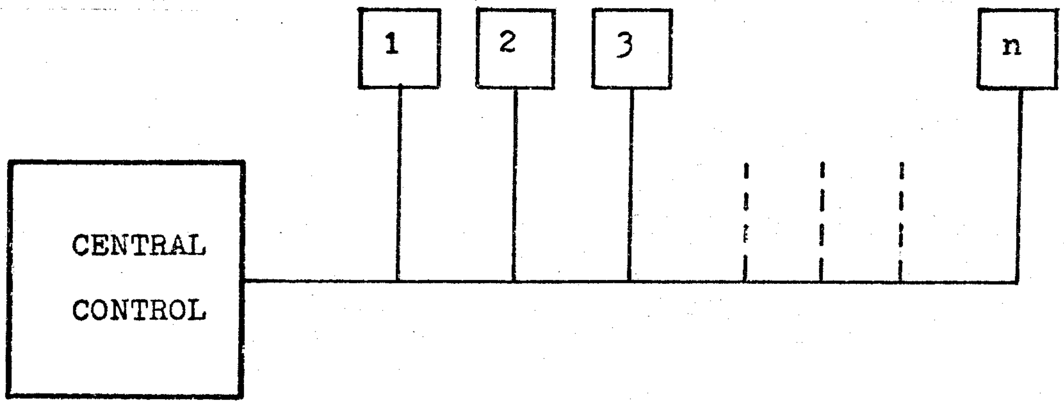
8) The realized system is interfaced to a PDP8L computer which serves as the overall system control. The details of the design follows.

CELL ORGANIZATION

As shown in Figure 2.1, each cell resembles a small computer, containing its own memory, arithmetic processing unit and input - output circuits. These cells receive instructions from a central controller simultaneously, also shown in Figure 2.1. Each cell has three memory words (m_1, m_2, and m_3), two of which are intended for operand and operator and the third for the results of cell computations. Each word is a 16 bit serially addressed memory. The processor consists basically of gating circuits and a full adder, also designed to operate sequentially. At the present state of technology, parallel arithmetic would be prohibitive both in cost and labour. Figure 2.2 illustrates the logical flow of data in a cell. There are two selector circuits which drive the input of the processor unit (P.U.). Selector circuit 1 chooses either the contents of word m_1 of this cell or of any neighbour cell. Selector circuit 2 chooses

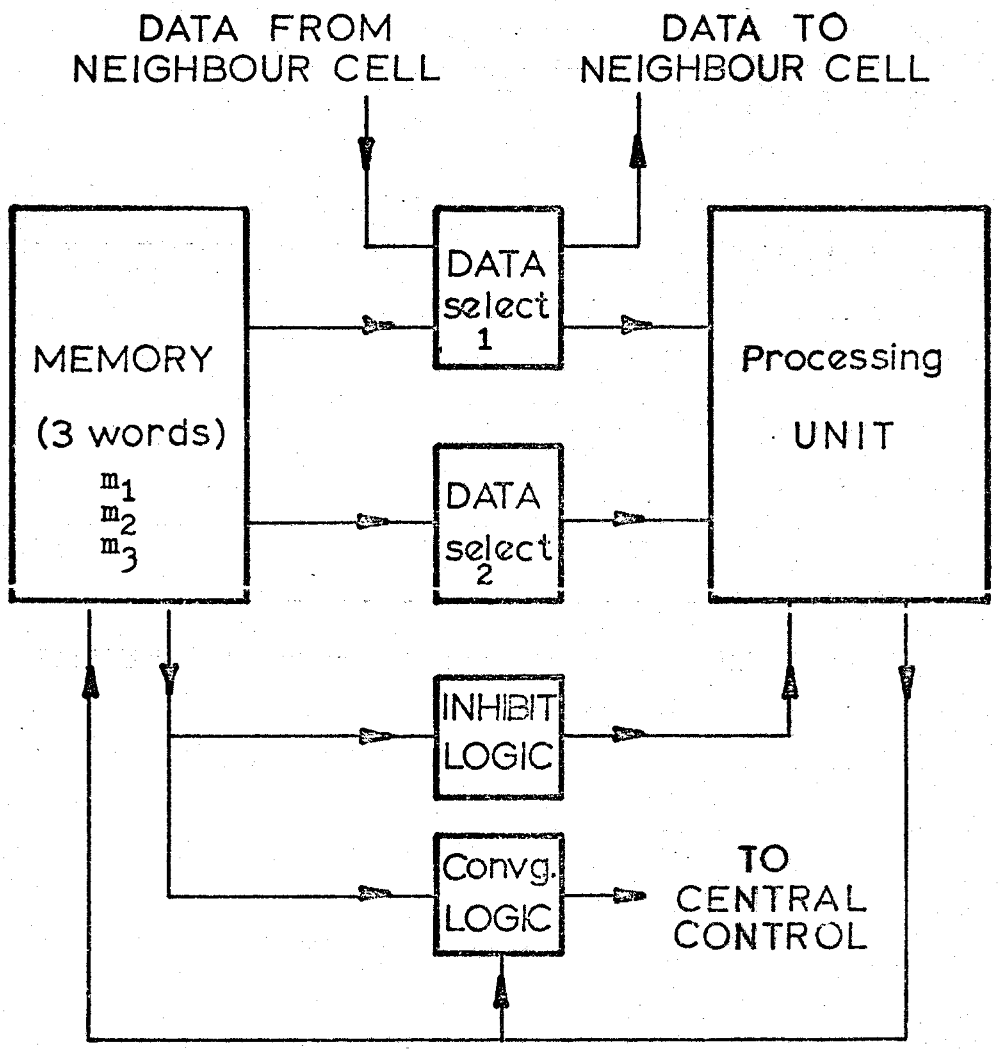


A CELL WITH MEMORY AND LOGIC



CELLS UNDER A CENTRAL CONTROLLER

FIGURE 2-1. GENERAL ORGANIZATION OF CELLS.



<u>FIGURE 2.2</u> COMPUTING MEMORY CELL:

BLOCK DIAGRAM

one of the three words in this cell. The processor performs the arithmetic and logic operations listed in Table 2.1 subject to the state of the inhibit logic, which effectively either accepts or ignores instructions from the central controller depending on its setting. The output of the processor can be fed to any combination of the three words of the cell for storage. This data flow organization allows us to perform operations between data in the cell or between data of this, and another cell or an external source, the results of which can be stored in the cell.

The 'convergence' logic is designed to compare previous and present values of data for a prescribed number of bits. If, for example, the value from the previous computation is stored in m_3 and the P.U. is computing the present value, then the convergence circuit will produce an output when these data differ. The comparison is performed at the same time while the P.U. output is being stored in word m_3. If any enabled 'convergence' circuit produces an output, this is detected in the central control and can be used to recycle an iteration.

REALIZED CIRCUIT

Figure 2-3 shows the overall circuit for one cell. Not shown, are the eight memory address lines and driving buffers. In the model constructed, each cell requires 17 I.C. chips to implement (this includes the address line buffers) and requires about 1.9 watts to drive. These two

Arithmetic Functions

$m_1 + m_1$	Results of these operations can be stored in m_1 or m_2 or m_3.
$m_1 + m_2$	
$m_1 + m_3$	
$m_1 - m_1$	
$m_1 - m_2$	
$m_1 - m_3$	
$m_1 \times m_2 = m_3$	'A' is set external to the array, and operates on the whole data plane.
$m_1 \times A = m_3$	
$m_1 \div 2^n$	$n = 0, 1, 2, \ldots, 11$ This operation also operates on the whole data plane.
Shift data 1 bit left	Operates on m_1 only.
Shift data 1 bit right	

Logic Operations

$m_1 \cdot m_2$	The results can be stored in m_1, m_2 or m_3.
$m_1 \cdot m_3$	
$m_1 + m_2$	
$m_1 + m_3$	
$m_1 + m_2 + m_3$	
$m_1 \oplus m_2$	
$m_1 \oplus m_3$	
Complement (m_1, m_2, m_3)	

Note: All the above functions can be inhibited in any selection of cells by setting control bits 0, 1 or 2.

Comparison

$m_3(t - 1)$ vs $m_1(t)$	This means that current data in m_3 can be compared with results of computations being performed. If data are equal the output of the comparison is '0', if not, a sequence of 1's.
$m_3(t - 1)$ vs $m_2(t)$	
$m_3(t - 1)$ vs $m_3(t)$	

TABLE 2.1 CELL FUNCTIONS

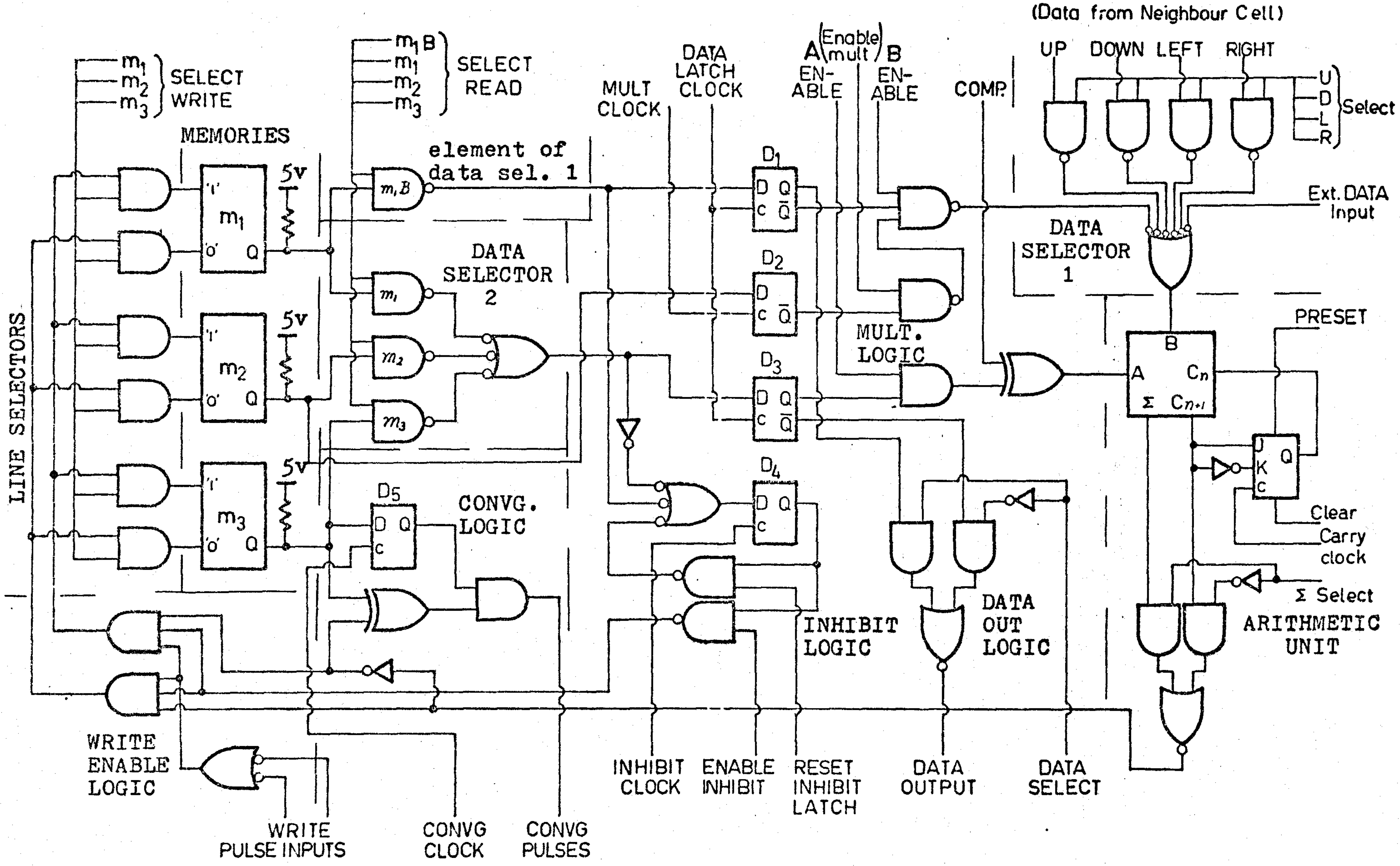


FIGURE 2.3 CELL SCHEMATIC

factors alone limit the magnitude of the array to a small but useful 20 cells. Such a cell would be an attractive organization for realization with L.S.I.. Should this be done, a cell with 50 or so input-output pins and perhaps only a few I.C. chips would result.

MEMORIES

The memories used in this design have the disadvantage of requiring seperate inputs for writing logic '1' and logic '0'. This means that the line selectors (see Fig. 2-3) which consists of 2-input AND gates, have to be duplicated, three gates for writing 'i's' and three for writing '0's'. However, only three 'select write' lines are required.

DATA SELECTORS

DATA SELECTOR$_1$ selects data either from m_1(B) or from any of five external sources. In our present array organization, the four external sources are the immediate neighbours of the cell and the fifth is the DATA INPUT. Excluded from this selector are the data from m_2 and m_3. Their inclusion appears redundant and would also increase the complexity of the circuit.

DATA SELECTOR$_2$ selects data from any combination of m_1, m_2, or m_3 for processing or for operation with data from DATA SELECTOR$_1$.

ARITHMETIC UNIT

The arithmetic unit consists of a one bit binary full adder and a J.K. flip-flop for delaying the 'carry out'. Besides 'addition', the AU can be made to perform the EXCLUSIVE OR function by merely disabling the 'carry' flip flop and taking the output from Σ. The logical AND function can be performed, also be disabling the 'carry' flip-flop but taking the output from 'C_o' the CARRY-OUT output. Note that these two follow from the fact that in a binary full adder, the carry-out is the AND operation of the two data inputs and the Σ output is merely the EXCLUSIVE OR operation. Also, by presetting the carry flip-flop, ie. by setting the CARRY IN to '1', data can be incremented by one.

DATA LATCHES

The rest of the circuit can be considered the Logic Unit. The D-type latches are used to hold information for a part of, or a whole 16 bit address cycle. D_1 and D_3 are data latches which hold data intended for the AU, for only 1 bit (1/16 th) of the cycle. D_2 is used for multiplication and holds data from m_2 only, for a full 16 bit cycle. The address cycle is treated in greater detail in chapter 3.

INHIBIT LOGIC

D_4 and its associated logic gates form the 'INHIBIT OPERATION' logic. This circuit is designed to interrogate bits 0, 1 and 2, of any of the words m_1, m_2 or m_3 selected for operation. If a logic '1' is detected in either bits 0, 1 or 2, the INHIBIT CIRCUIT will disable the 'WRITE ENABLE' logic. This prevents writing over existing data currently stored in m_1, m_2 or m_3, and in effect is equivalent to inhibiting an operation for the cell in question.

WRITE LOGIC

The 'WRITE ENABLE' logic basically is made of two 3-input 'AND' logic gates, one each for writing 1's and 0's. One of the three inputs is for data, another for the inhibit logic discussed in the previous paragraph and the third, for the WRITE strobe pulses. The 2-input OR logic allows for two sets of WRITE STROBE pulses. One set is common to all cells and thus operates on all cells simultaneously and the other set which is unique to each cell, (by line selecting/demultiplexing) allows operations in the selected cell only.

CONVERGENCE LOGIC

D_5 and its associated circuitry perform the 'Convergence Check' operation. Data stored in m_3, presumably from the previous computation, is compared with incoming data from the current computation by the EXCLUSIVE OR gate. If incoming data in the range being compared is different from the previous data, a pulse will be generated which informs the control unit that the desired convergence has not yet been attained. This, for a relaxation type problem for example, enables the computer to repeat a self-determined number of passes until the desired convergence is reached. This operation is enabled by storing a logic '1' in control bit 3 of m_3. D_5 serves to interrogate this bit (bit 3) in the first cycle of operation to see if a convergence check is required.

DATA READ OUT

The DATA OUT logic serves merely to select data from D_1 or D_3. D_1 is selected only when we wish to transfer m_1 data from one cell to a neighbour for simultaneous operation with data of the neighbour cell. D_3 is selected for normal data Read Out from m_1, m_2 or m_3 of any cell.

SUMMARY

From Figure 2.3 we can see that the cell design is really rather straightforward considering that it can perform all the

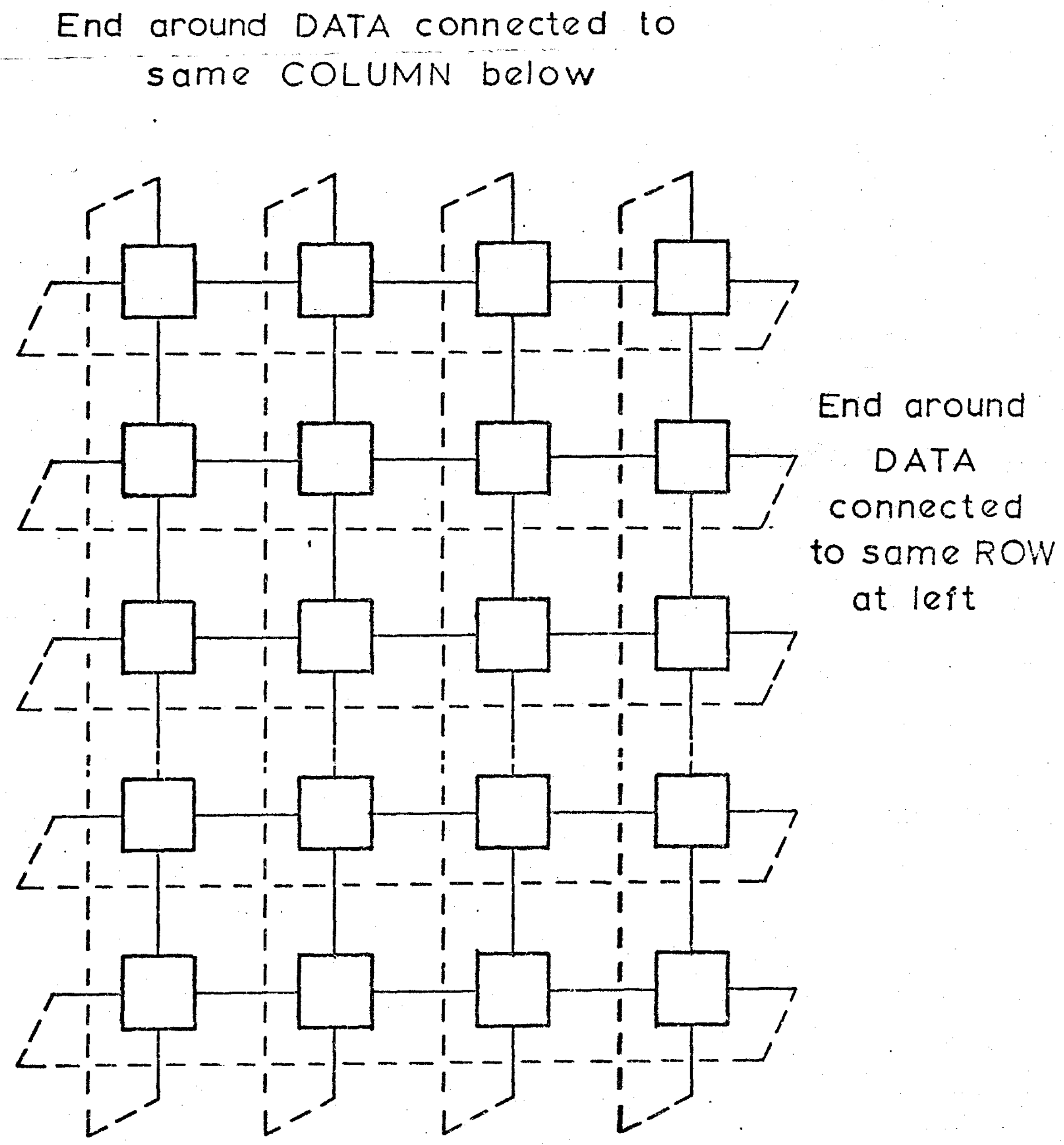

End around DATA connected to same COLUMN below
End around DATA connected to same ROW at left

FIGURE 2.4 ARRAY ORGANIZATION OF CELLS SHOWING COMMUNICATION BETWEEN CELLS

functions listed in Table 1. One reason for this relative simplicity in design is the fact that the computations are done serially, one bit at a time. Thus a cell function takes a minimum of 16 bits or a full address cycle. The heart of the processor section is the binary full adder. It performs most of the major functions, and with supporting gating logic, the full complement of arithmetic and logic functions can be performed.

Twenty of these cells were built on five printed circuit boards and are organized in a rectangular array as shown in Fig. 2.4.. Each cell communicates with its four neighbours. The boundary cells communicates with its three nearest neighbours and is connected 'end-around' to communicate with its corresponding boundary cell on the opposite side of the array.

CHAPTER 3

CONTROL UNIT:- CLOCKING LOGIC

In a system where control is highly centralized such as this is, the controlling unit has to be elaborate by necessity. The CLIM array cells described in chapter 2 operate entirely under central control. The control unit sets the logic state of 41 control lines, the state of which enables the appropriate functions. The total control actually extends beyond the control unit to be described, to include the general purpose computer to which the system is interfaced. The general organization is illustrated in Fig. 3.1. Certain aspects of control such as data storage, setting of the data field and programming of sequential iterations are intended as being in the domain of the general purpose computer. On the other hand, certain other aspects like storage of instruction states, system timing and clocking, decision logic and microprogramming are common to the domain of both the specially designed control unit and the general purpose computer.

Since the PDP8 computer, serving as the general purpose computer, has only a small 4K memory bank, the control unit was designed to do as much of the control as practical. The PDP8, to be discussed in chapter 5, is used mainly to store data,

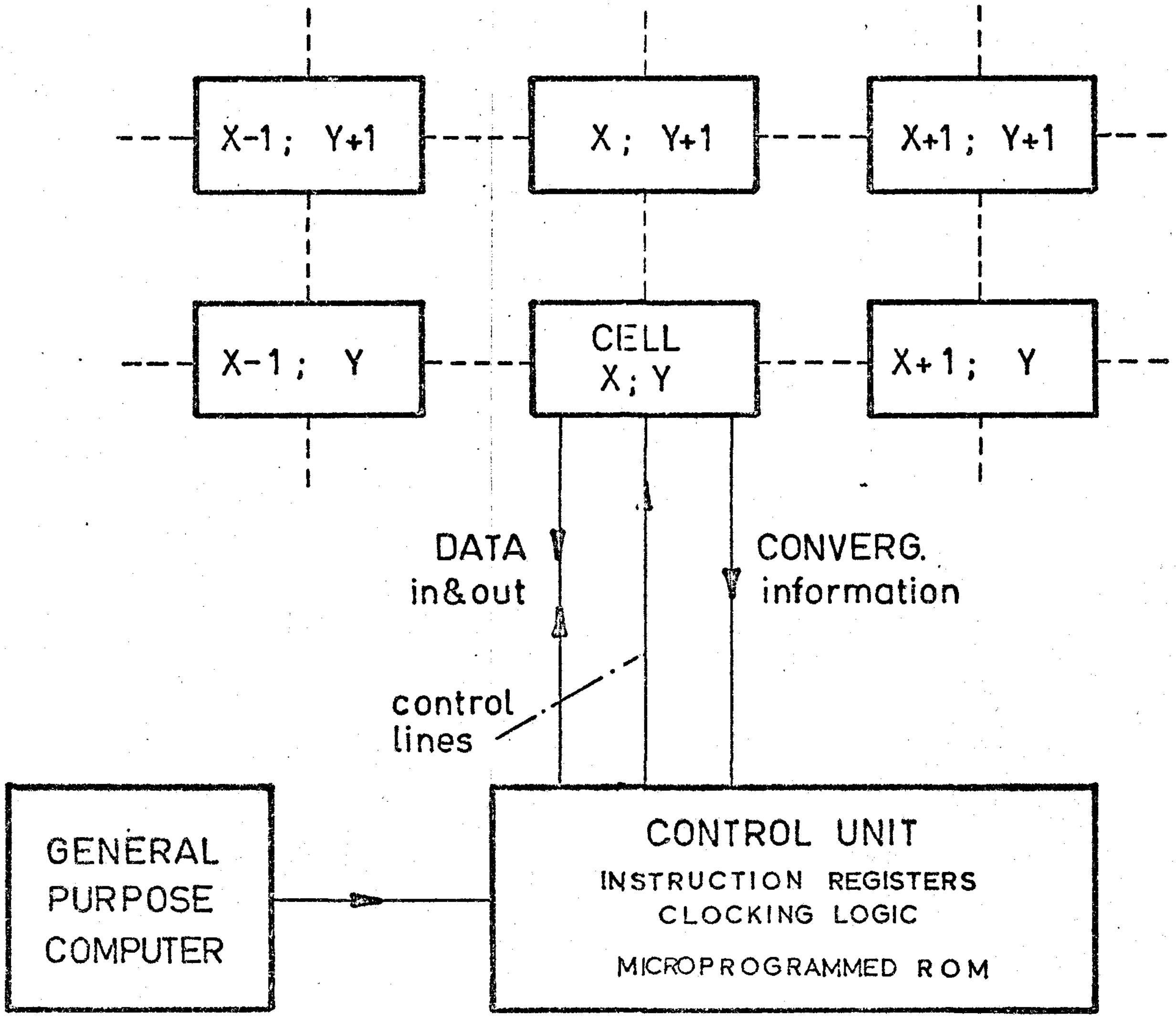


FIGURE 3.1 GENERAL ORGANIZATION: SHOWING COMMUNICATION WITH ARBITARY CELL

sequentially iterate programs and perform the microprogrmas.

The control unit is discussed in two parts, the instruction registers and system interface in chapter 4, and the system timing and clocking logic follows.

CLOCKING LOGIC FUNCTIONS

Basically, the function of the clocking logic unit is to provide the necessary pulse trains to execute the operations in the computing memory cell. These basic functions include:

1) addressing the 16 cell memory bits serially.
2) producing appropriate clocking pulses for D-type flip flops, to latch information stored in memory locations currently addressed. These are for the purposes of interrogating and latching 'Inhibit' information, of holding 'multiplier' bits for a complete address cycle of multiplicand data, of interrogating and latching 'convergence' information and of holding operator and operand data for the cell functions.
3) generating 'carry' pulses for the JK flip flop associated with the full adder of Figure 3.3 and also to generate properly timed pulses for the 'preset' and 'clear' inputs of the carry flip flop.
4) generating write strobe pulses for writing data into addressed memory locations.

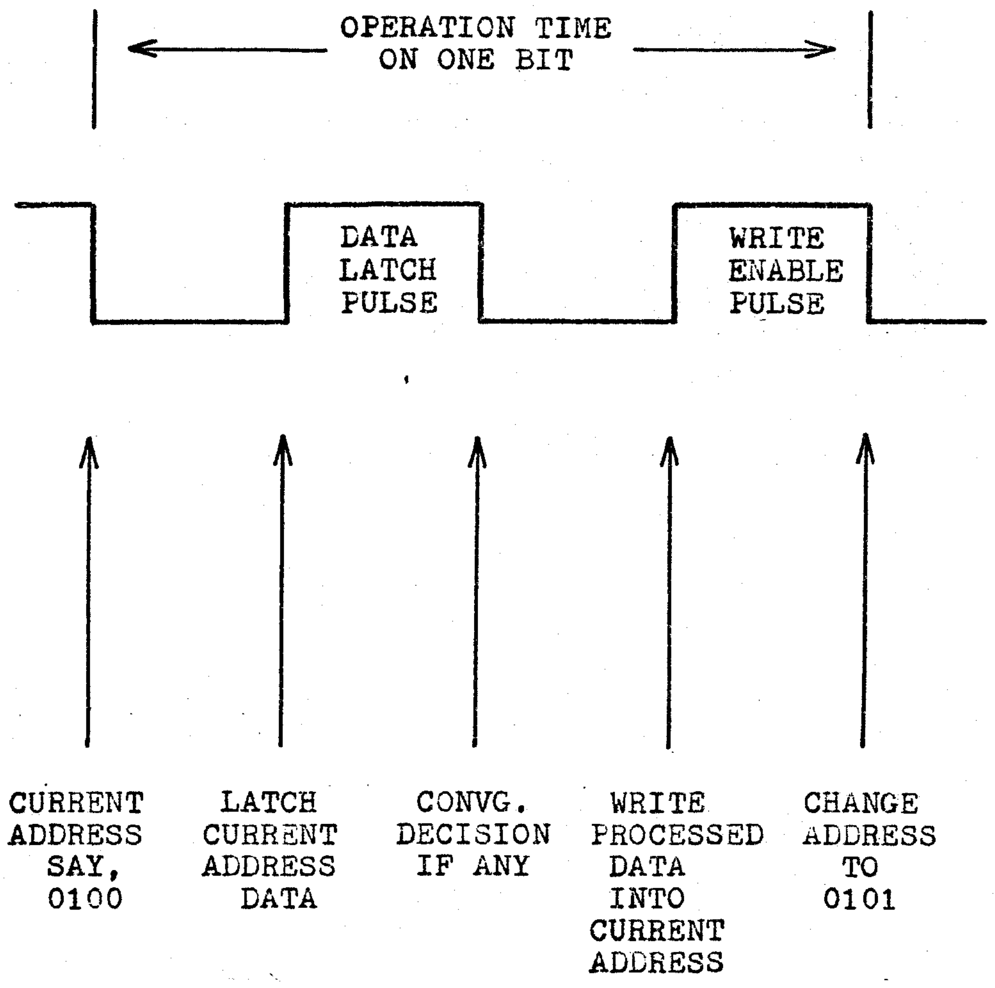
5) providing logic
   a) to control the ADD/SHIFT cycle used in multiplication
   b) to indicate Busy or Free mode of the system
   c) to control the interrogation range of the convergence check.
6) generating Flag pulses to indicate completion of an operation.

ADDRESS CYCLE

Starting from a basic master-clock pulse with mark-space ratio of one, a typical operation on one bit is as follows:-

1) latch information from location currently addressed.
2) perform logic decisions, as in convergence, if any.
3) write processed information into currently addressed bit of selected memory.
4) change address to the next location.

These four operations can be defined in the time spanned for two clock pulses, having a total of 2 alpha and 2 beta edges, one edge for each of the operations. (See Figure 3.2). Thus, a full 16 bit operation, being one address cycle, would span over 32 clock pulses. Currently, the longest time needed for operation, propagation and settling, is approximately 100 ns between the operations 1) and 2). This allows the clock to run at a maximum rate of 5 MHz. The system is currently set



<u>FIGURE 3-2</u>. BASIC OPERATION ON ONE BIT.

to run at 2 MHz.

PULSE TRAINS

Based on the function requirements listed earlier and on the cell circuitry, repeated in Figure 3.3 for convenience, the basic pulse trains illustrated in Figure 3.4 can be derived.

DATA LATCH PULSES

Of the 32 clock pulses for 1 cycle, the 16 odd numbered pulses are used for data latch. The first one being used to latch contents of memory location '0000' (binary '0') which is the location addressed in the 'clear state', the 2nd one for bit 0001, and so on.

INHIBIT LATCH PULSES

Only three INHIBIT Latch pulses are needed, the first one to interrogate location 0000 the next, location 0001 and the third, location 0010. (Recall from Chapter 2 that bits 0, 1, 2, of the memories are used for Inhibit operations).

RESET AND PRESET PULSES

The Reset Inhibit Latch logic allows the inhibit circuit to erase an 'inhibit' state carried over from the previous

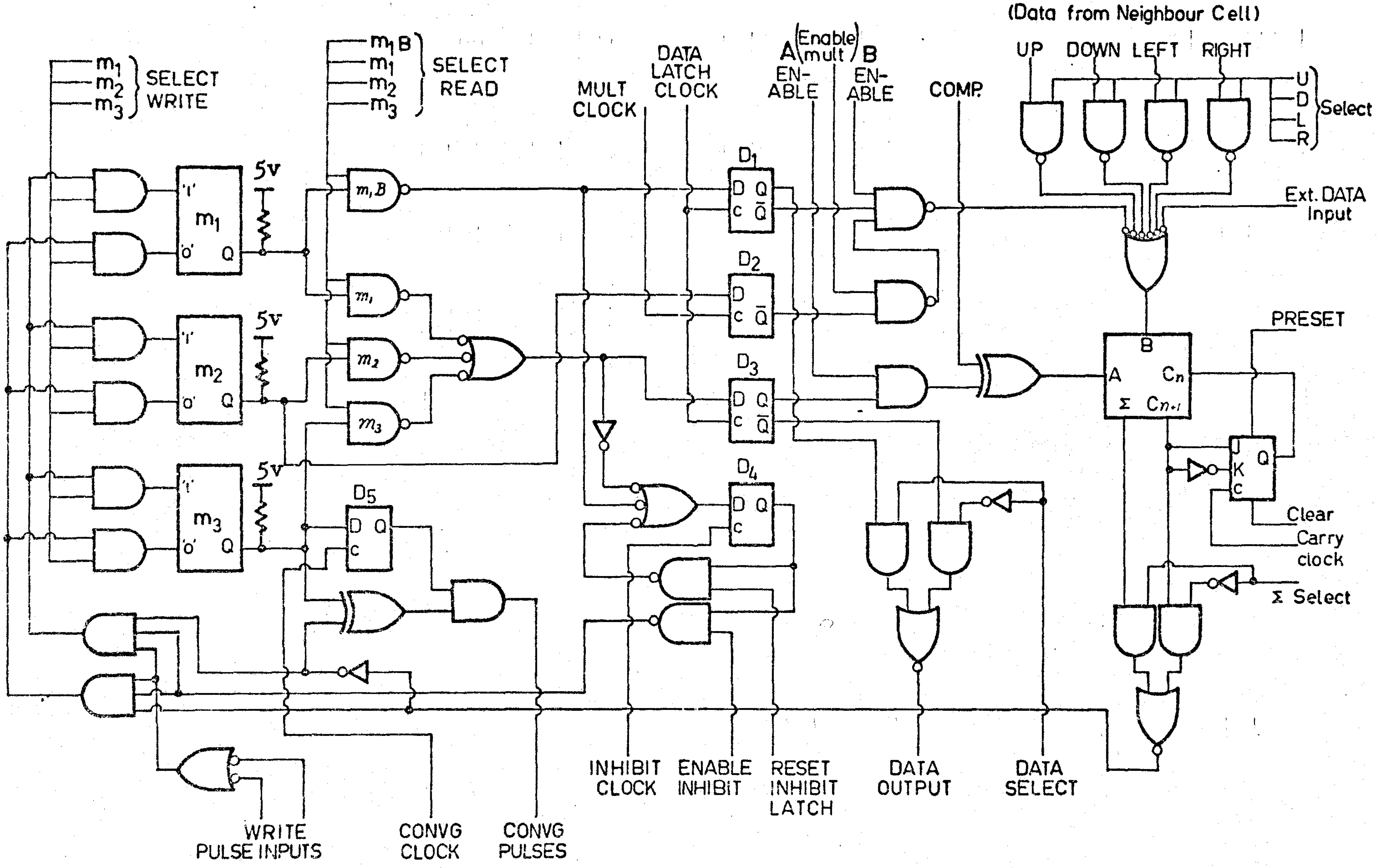


FIGURE 3.3 CELL CIRCUITRY

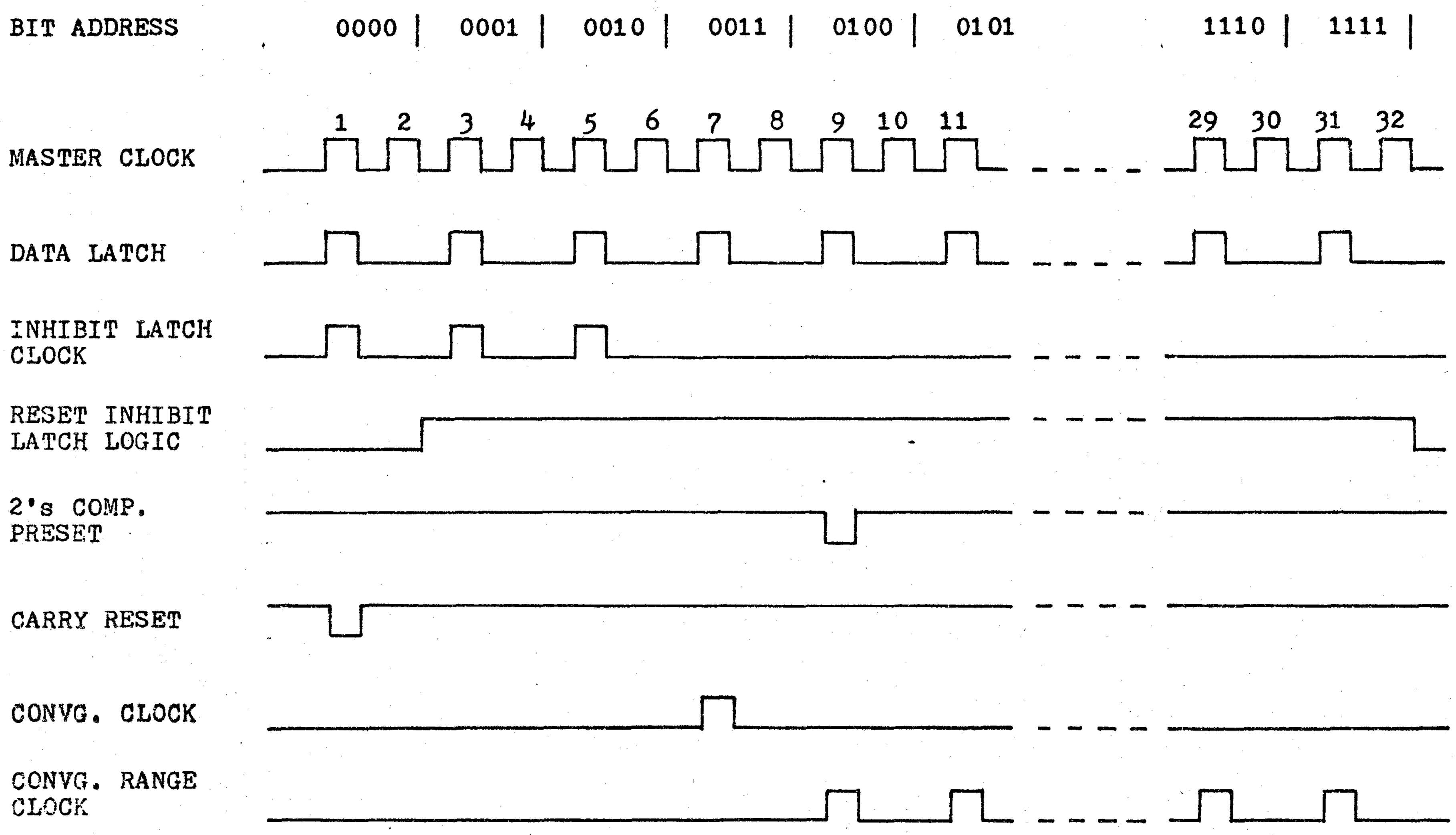


FIGURE 3.4 CONTROL BOARD PULSE TRAINS (contd.)

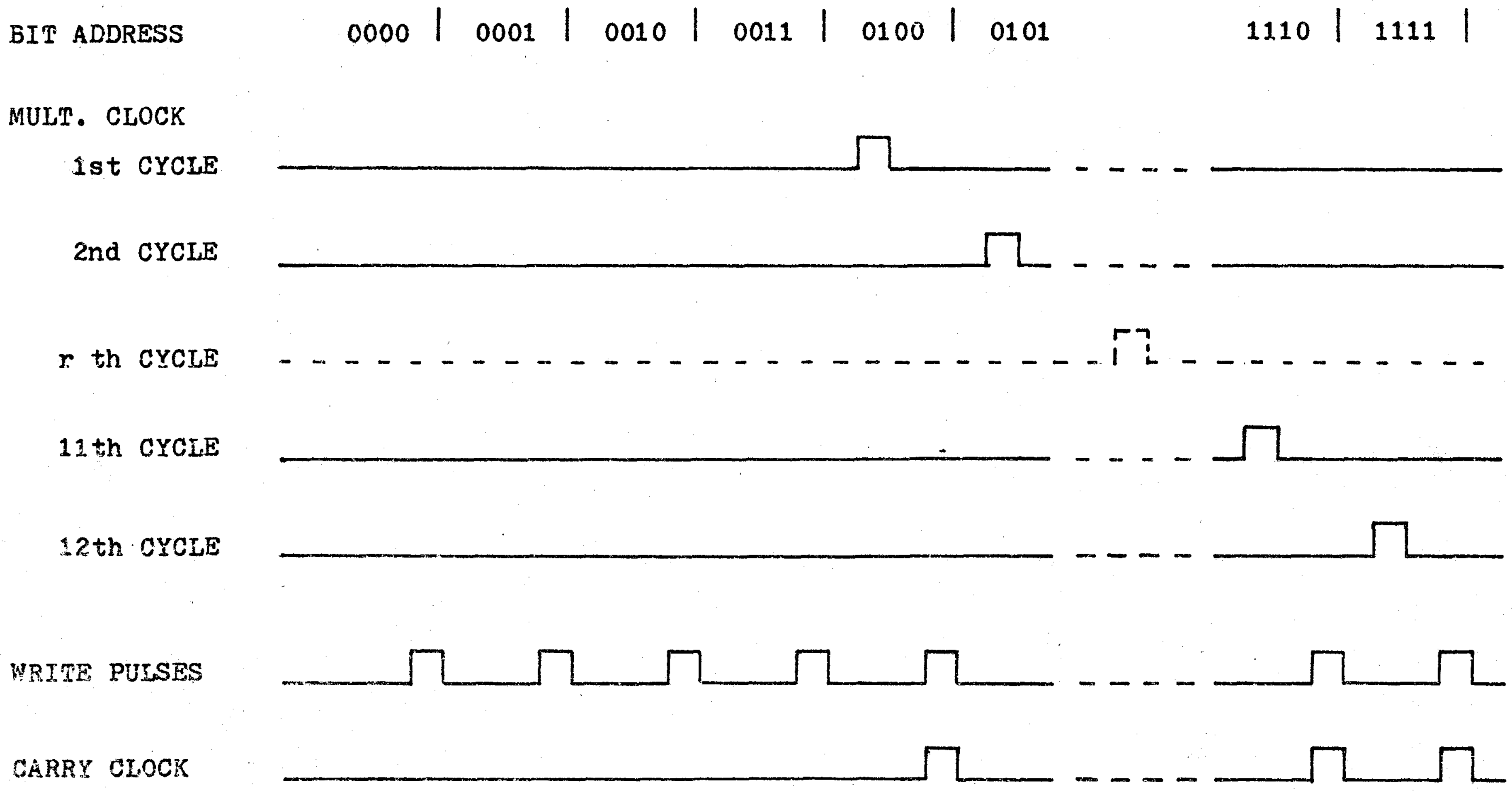


FIGURE 3.4 (contd)

address cycle. Using D-type latches, an erase can be performed only by clocking the latch and presenting a logic '0' at the data input, or in this case by presenting new data which performs the dual function of erase old data and latch on to new data.

The 2's complement preset occurs at data bit 4 after the control bits 0, 1, 2 and 3 have been processed (note that data bits for arithmetic operations are stored in bits 4 to 15 only). It sets the carry-in of the full adder to '1' which increments incoming data by 1.

The Carry Reset occurs at the very beginning of a cycle to erase any carry over from a previous operation.

CONVERGENCE CLOCK PULSE

The Convergence Clock is timed to interrogate bit 3 of the control data. If a '1' is detected, the cell in question will perform a convergence check on its data. The basic Convergence Range pulses are available from bits 4 to 15. These pulses generated in the convergence circuit on the CONTROL BOARD, determine the number of data bits being interrogated for convergence. If, for example only Convergence Pulses 8 to 15 are generated, then Convergence is checked only for data bits 8 to 15, ignoring changes in the less significant bits 4, 5, 6 and 7. (The circuit for selecting this range is described later).

MULTIPLICATION PULSES

The technique for multiplication used here is the ADD/SHIFT technique. During the first ADD/SHIFT cycle (equivalent to 2 address cycles) a pulse is generated at address bit 4, during the 2nd, at bit 5 during the third, at bit 6, and so on, until the 12th and final cycle where a pulse is generated at bit 15. The reason for this is that during the first ADD/SHIFT cycle bit 4 of the multiplier is interrogated and held for the rest of the cycle to operate on the multiplicand, during the 2nd cycle, bit 5 of the multiplier is interrogated for the same purpose and so on until the full multiplication cycle (12 ADD/SHIFT cycles or 24 address cycles) is complete. This operation is illustrated in an example for multiplication between two 4 bit binary numbers, presented in the next page.

WRITE PULSES

Whereas the DATA LATCH pulses were composed of the 16 odd numbered master clock pulses, the WRITE pulses are composed of the other 16 even numbered pulses. Recall that each bit operation consists of the interval occupied by one DATA LATCH pulse and one WRITE pulse. These WRITE pulses are available in 8 selections, the particular sequence selected being dependent on the desired operation. For example if control data bits 0,1,2 and 3 only are to

```
      0 1 1 0        A register
      0 1 0 1        B register
      -------
      0 1 1 0
    0 0 0 0
  0 1 1 0
0 0 0 0
-------------
0 0 1 1 1 1 0        C register
-------------
```

(a) 'Long Multiplication' of two binary numbers

Now, to illustrate in-cell multiplication, let :-

A = 0 1 1 0

B = 0 1 0 1 = $b_4\ b_3\ b_2\ b_1$

C = 0 0 0 0

<u>1st ADD/SHIFT cycle</u>

$b_1 = 1$

A = 0 1 1 0

C = C + A = 0 0 0 0 + 0 1 1 0

= 0 1 1 0

Shift A 1 bit left: A = 0 1 1 0 0

<u>EXAMPLE 3.1</u> CELL MULTIPLICATION

(contd)

2nd ADD/SHIFT cycle

$b_2 = 0$

since $b_2 = 0$, addition is inhibited.

Shift A, 1 bit left: A = 0 1 1 0 0 0

3rd ADD/SHIFT cycle

$b_3 = 1$

A = 0 1 1 0 0 0

C = C + A = 0 1 1 0 + 0 1 1 0 0 0

= 0 1 1 1 1 0

Shift A, 1 bit left: A = 0 1 1 0 0 0 0

4th ADD/SHIFT cycle

$b_4 = 0$

since $b_4 = 0$, addition is inhibited

C = 0 1 1 1 1 0

Shift A, 1 bit left: A = 0 1 1 0 0 0 0 0

The Multiplication is now complete, the answer is in 'C'

where:- A X B = C = 0 1 1 1 1 0

EXAMPLE 3.1 (contd) CELL MULTIPLICATION

be operated on, then a sequence enabling pulses 0 to 3 only, of the 16 WRITE pulses, is selected. Note that new data can be written into memory only when the appropriate WRITE pulse occurs. For all arithmetic operations, it is necessary to enable bits 4 to 15, these being the 12 data bits. The full sequence 0 to 15 is selected when new information has to be written into all 16 bits of memory. Selections of single pulses 0,1,2 or 3 are used if it is required to alter just one control bit. The final selection is where all WRITE pulses are disabled, as one would require when nondestructive data read out is desired. These selections are tabulated in Table 3.1.

CARRY PULSES

The final pulse train is the CARRY pulses. These operate the 'carry' flip flop when the cell is performing addition, and hence are required only for bits 4 to 15.

REALIZED CIRCUIT

The circuit that was constructed produces all the pulses described above, and is illustrated in Fig. 3-5. The heart of this circuit is a free running clock and a string of counting elements consisting of 3 flip flops and two 4 bit binary counters. The output of the first flip flop, which serves to generate a mark-space ratio

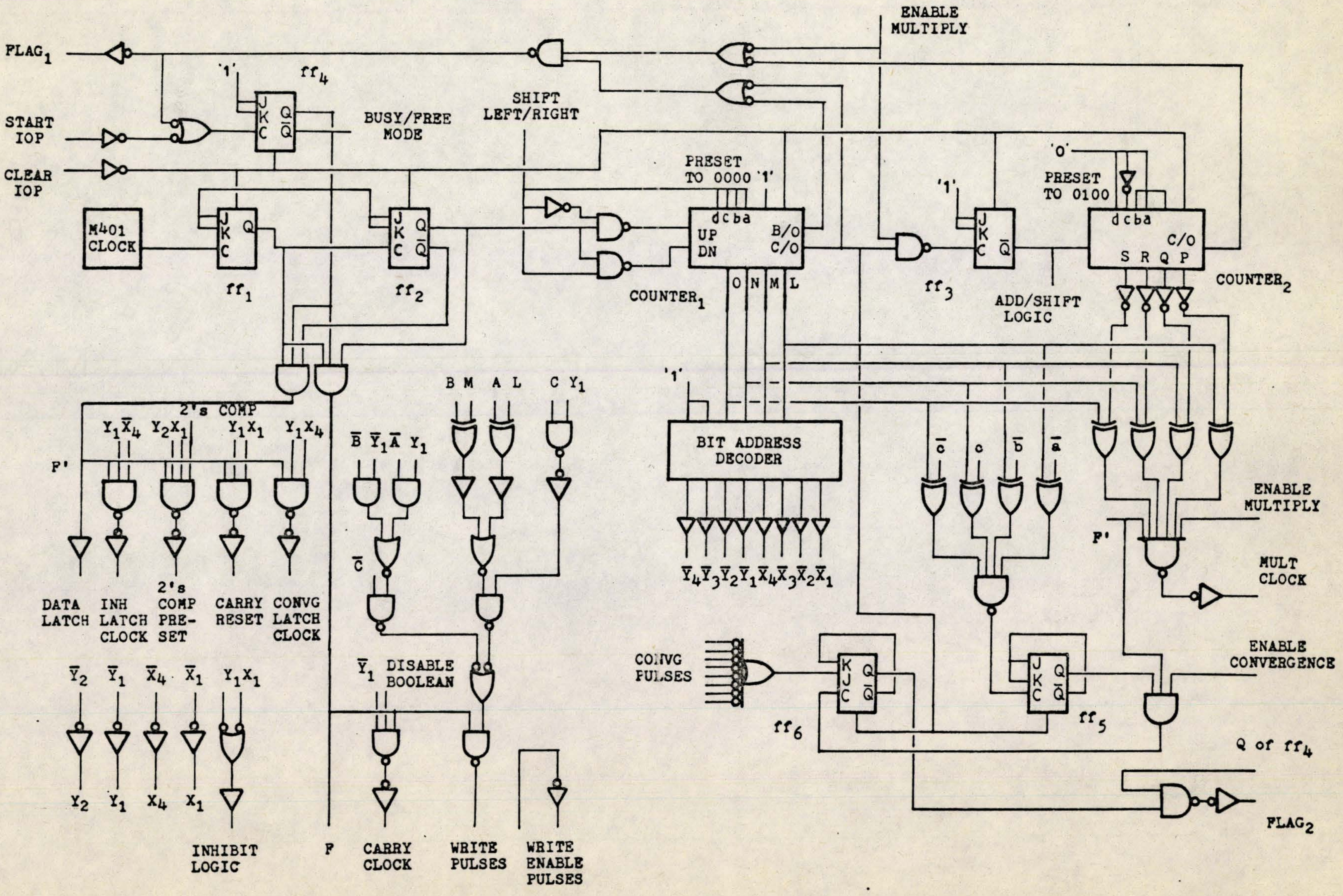


FIGURE 3.5 CONTROL CIRCUIT

of one for the clock pulses, is considered the master clock. The second flip flop and the first counter (total of 5 bits) control the address cycle which occupies 32 master clock pulses. The third flip flop and the second counter (total of 5 bits) is used to control the multiplication cycle which occupies 24 address cycles.

The CARRY OUT pulse from the first counter is used as a flag to indicate the end of a multiplication cycle. These pulses are fed to flip flop 4, which starts and stops the master clock and hence controls the cycle. A cycle is iniated by the START IOP, generated by the PDP8, which triggers flip flop 4 and starts the master clock. The CARRY OUT pulse from counter 1 or 2, serves to reset flip flop 4. It also serves as the system FLAG 1 which informs the PDP8 that the operation is complete.

Prior to any operation, the PDP8 generates a CLEAR IOP which clears or sets all logic on the control board to an 'initial' state. The output from each bit of the first counter is decoded via a binary to a 'two out of eight' decoder. This gives 4 'X' lines and 4 'Y' lines to address the 16 bits of the memories. For example, bit 0 is addressed by Y_1 X_1, bit 1 by Y_1 X_2, etc, and bit 15 by Y_4 X_4.

PULSE TRAIN LOGIC

Logic for producing all the pulse trains of Fig. 3.4 are available from the counters and decoder so far discussed. The following pulse trains are fairly straightforward and the logic realizations should be obvious from Fig.3.4 and Fig.3.5:-

DATA LATCH	=	F'
INHIBIT LATCH CLOCK	=	$F'.Y_1.\overline{X}_4$
RESET INHIBIT LOGIC	=	$\overline{Y}_1 + \overline{X}_1 = \overline{Y_1.X_1}$
2's COMP PRESET	=	$F'.Y_2.X_1.$(2's COMP ENABLE)
CARRY RESET	=	$F'.Y_1.X_1$
CONVERGENCE CLOCK	=	$F'.Y_1.X_4$
CARRY CLOCK	=	$F'.\overline{Y}_1.$(BOOLEAN DISABLE)

The WRITE logic has three control lines A, B, C, connected to the IR, which allows the selection of eight pulse trains as illustrated in **Table** 3.1. The WRITE function can be produced by :-

$$\text{WRITE} = F.\overline{C}.(\overline{B}.\overline{Y} + \overline{A}.Y) + F.Y.C(\overline{B \oplus M + A \oplus L})$$

The MULTIPLICATION clock pulses are intended to enable D_2 of each cell to latch on to multiplier data bits, each bit for a complete ADD/SHIFT cycle. As explained earlier, since multiplier data is stored in bits 4 to 15, the multiplication clock must be designed to interrogate bit 4 during the 1st

IR setting C B A	logic enabled	pulse train	purpose
0 0 0	$F \cdot 0$	nothing	no operation
0 0 1	$F \cdot Y_1$	bits 0-3	4 control bits
0 1 0	$F \cdot \overline{Y}_1$	bits 4-15	ARITH OPERATION
0 1 1	$F \cdot 1$	bits 0-15	all of memory
1 0 0	$F \cdot Y_1 \cdot X_1$	bit 0	INHIBIT bit
1 0 1	$F \cdot Y_1 \cdot X_2$	bit 1	control bit 1
1 1 0	$F \cdot Y_1 \cdot X_3$	bit 2	control bit 2
1 1 1	$F \cdot Y_1 \cdot X_4$	bit 3	CONVERG. bit

The circuit to produce the logic of column two can be minimized by replacing X_1, X_2, X_3, X_4, by logic from the 2 lsb. of COUNTER_1, where :-

	X_1	X_2	X_3	X_4
M	0	0	1	1
L	0	1	0	1

The pulse trains of column 3 can be generated by the WRITE function :-

$$\text{WRITE} = F.\overline{C}.(\overline{B}.\overline{Y} + \overline{A}.Y) + F.Y.C.(\overline{B \oplus M + A \oplus L})$$

TABLE 3.1. WRITE PULSES SELECT

ADD/SHIFT cycle, bit 5 for the next cycle and so on, and finally bit 15 for the 12th and last ADD/SHIFT cycle. As illustrated in Table 3.2, this pulse train can be realized by the logic :-

$$\text{MULT} = F'.(L \oplus \overline{P}).(M \oplus \overline{Q}).(N \oplus \overline{R}).(O \oplus \overline{S})$$

Note, in Table 3.2 that since COUNTER_2 only needs to count 12 cycles, it is always preset to binary 4 before any operation.

The CONVERGENCE RANGE pulses are also produced by using EXCLUSIVE OR logic :-

$$\text{CONVG.} = F'.(\overline{a} \oplus L).(\overline{b} \oplus M).(c \oplus N).(\overline{c} \oplus O)$$

The selection of pulse trains is presented in Table 3.3. These pulse trains are directed to the clock of flip-flop_6 which is designed to detect a logic 1 at the 'J' input for the duration of the CONVERGENCE RANGE selected. The input to 'J' of flip-flop_6 is the 'OR' function of the outputs from the CONVERGENCE logic circuit of all cells in the array.

If in the selected range, a logic 1 is detected from any of the cells, which are enabled for Convergence check by setting bit 4 of m_3 to '1', a pulse is generated which is called Flag_2 and is used to instruct the PDP8 computer to recycle the iteration.

COUNTER$_1$

ADDRESS BIT	O	N	M	L
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

COUNTER$_2$

ADD/SHIFT CYCLE	S	R	Q	P
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0
10	1	1	0	1
11	1	1	1	0
12	1	1	1	1

By comparing these two 'truth' tables we can see that

$$\text{MULT} = F'.(L \oplus \overline{P}).(M \oplus \overline{Q}).(N \oplus \overline{R}).(O \oplus \overline{S})$$

is 'true' when Address bit 4 coincides with ADD/SHIFT cycle 1, bit 5 with cycle 2 and so on.

TABLE 3.2. MULTIPLICATION PULSES

IR SETTING c b a	PULSE TRAIN	# of significant bits
0 0 0	bits 4-15	12
0 0 1	bits 5-15	11
0 1 0	bits 6-15	10
0 1 1	bits 7-15	9
1 0 0	bits 8-15	8
1 0 1	bits 9-15	7
1 1 0	bits 10-15	6
1 1 1	bits 11-15	5

The Convg. interrogation function can be generated by using logic from COUNTER_1 alone together with 3 externally set 'select' lines. The function :-

$$\text{CONVG.} = F'.(\bar{a} \oplus L).(\bar{b} \oplus M).(c \oplus N).(\bar{c} \oplus O)$$

will produce a pulse when COUNTER_1 logic coincides with the 'select' setting. This pulse marks the beginning of the convergence interrogation, and is used to trigger flip-flop_5 of FIG.3.5., which will enable F' pulses from this time on till the end of the address cycle.

TABLE 3.3. CONVERGENCE PULSES SELECT

SUMMARY

This section of the control unit serves to generate the various pulse trains to execute the cell operations, which are controlled by the state of the IR.

The waveforms shown in Fig. 3.4 were formulated, based on the logical sequences required for the various cell functions decided upon. The circuit of Fig 3.5 is merely a logic circuit realization to produce the waveforms of Fig. 3.4.

CHAPTER 4

CONTROL UNIT - INTERFACE DESIGN AND INSTRUCTION REGISTERS

In conjunction with the clocking logic, the control lines to be described serve to enable or disable critical logic gates in the cells in order to allow execution of the desired operation. These lines control functions such as:-

1) selecting array location from which data is to be read.
2) selecting array location to which data is to be written.
3) selecting the neighbour from which data can be read from or sent to.
4) enabling the decision logic to inhibit or allow an operation.
5) enabling a path for multiplication, addition or complementing.
6) selecting data either from the sum output or the Carry-Out output of the full adder.

Unlike the control lines described in Chapter 3, these lines are held in the same state for the whole address cycle. The state of this group of control lines uniquely define a path of information flow in the cells for a full cycle, consequently controlling the function to be performed.

Most of the pulse trains described in Chapter 3 are closely tied to the function to be performed and therefore also to the setting of these control lines. In these cases, the relevant pulse trains are enabled depending on the state of these lines. These lines then can be thought of as Instruction Lines, their various states as Instruction Sets and the registers (in this case, D-type latches) which hold the Instruction Set, as Instruction Registers.

Not all the pulse trains are tied to the basic instruction lines. Those which require a measure of versatility like the write pulses, Convergence 'interrogation range' pulses and the Inhibit pulses are selected as required by their own allocated set of registers. They also form part of the IR (Instruction Registers).

The control lines from all the cells are tied in parallel to the IR. The actual instruction set is stored in the PDP8 computer which transfers the instructions to the IR as required. The IR then has to be interfaced to communicate with the PDP8. For this purpose a straightforward and simple design is utilized where a one to one link is established between a limited number of registers and the accumulator of the PDP8. No instruction-set coding is utilized, and the result is a simple hardware design which requires elaborate instruction and data loading procedures.

INTERFACE

The total interface design consists of 40 D-type latches, two 16 bit shift registers and some miscellaneous logic. Of the 40 latches 31 serve as the IR, 5 as array location address registers and 4 are spares. One of the 16 bit shift register is used to store data intended to be written in a selected array location and the other is used to store data to be read back to the PDP8. Both shift registers have parallel data input and output facilities. This is necessary because the PDP8 accumulator handles data in parallel whereas the computing memory cells handles data serially.

INSTRUCTION REGISTERS

The IR (see Figure 4.1) is organized in 4 rows of 10 latches each. The input of the 4 rows are connected in parallel as illustrated and are presented with data from the PDP8 accumulator simultaneously. In any one instruction loading cycle only 10 of the 12 PDP8 accumulator bits serve to hold control information and the other 2 is used to select one of the four rows of latches to which control information is transfered. Thus a typical instruction set has to be loaded in four cycles.

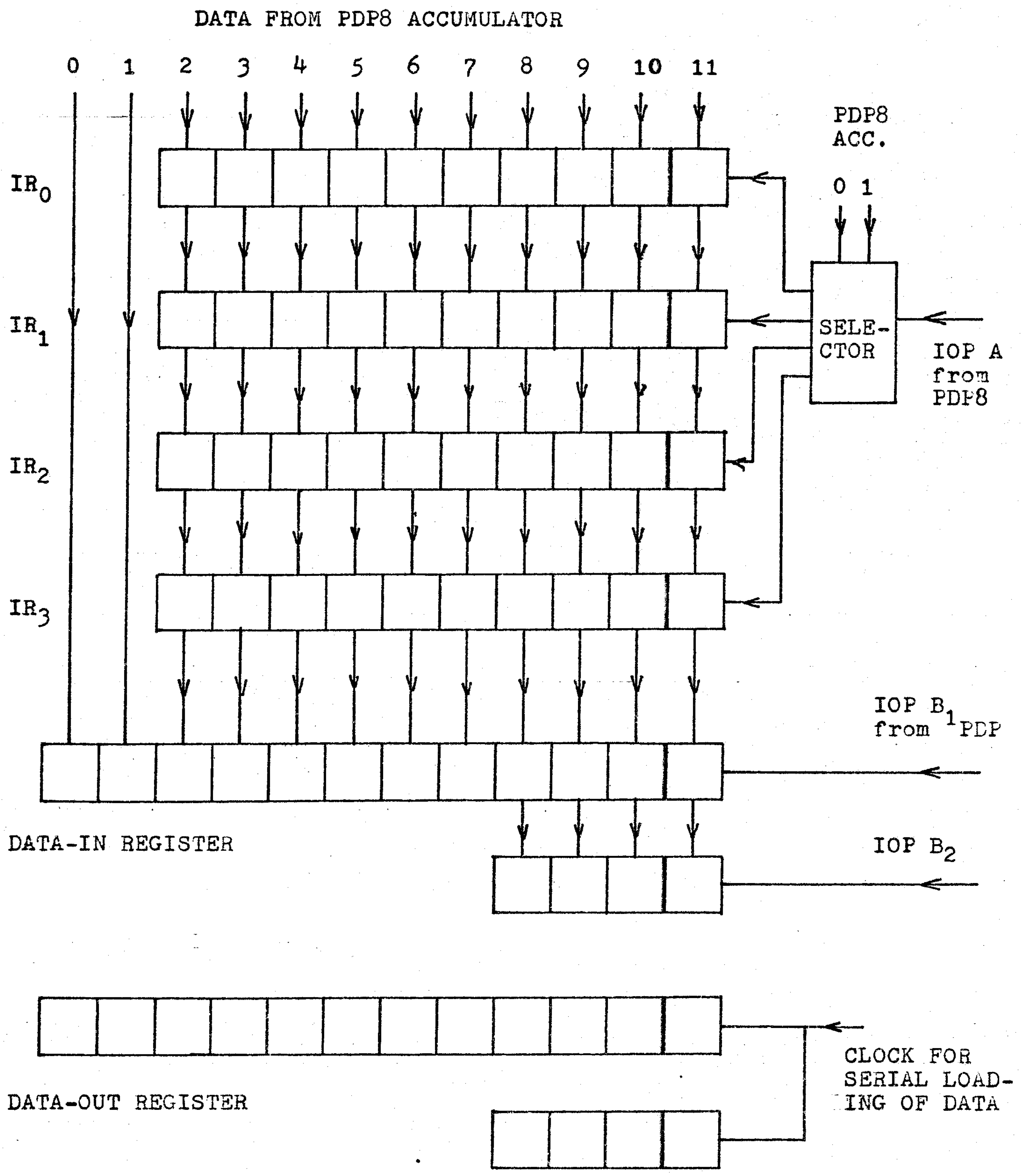


FIGURE 4.1 LAYOUT OF INSTRUCTION REGISTERS AND DATA REGISTERS.

For example the following state:-

```
(ROW SELECT) (    10 INSTRUCTION BITS    )
    0  0     0  1  1  0  1  0  1  0  1  1
```

loads the 10 instruction bits into the first row. The second row will be selected by the leading code '0 1', the third by '1 0' and the fourth by '1 1'.

DATA REGISTERS

Both Data Registers are 16 bit devices. This incompatibility to the 12 bit PDP8 accumulator makes it necessary to have two loading cycles for any data transfer. For the DATA IN register, the first cycle loads 12 data bits into the 12 msb of the register (most significant bits, bits 4 to 15), and the second cycle loads the 4 control bits, from a new set of accumulator data, into the 4 lsb of the register. Each cycle is executed by its assigned IOP (input-output pulse) generated by the PDP8. In a similar manner, for the DATA OUT register, the 12 data bits are written into the PDP8 accumulator in the first cycle by one IOP and the 4 control bits are written during the second cycle by another IOP. To write data into a selected array location, data from the PDP8 Accumulator is loaded in parallel into the DATA IN shift registers (See Figure 4.2)

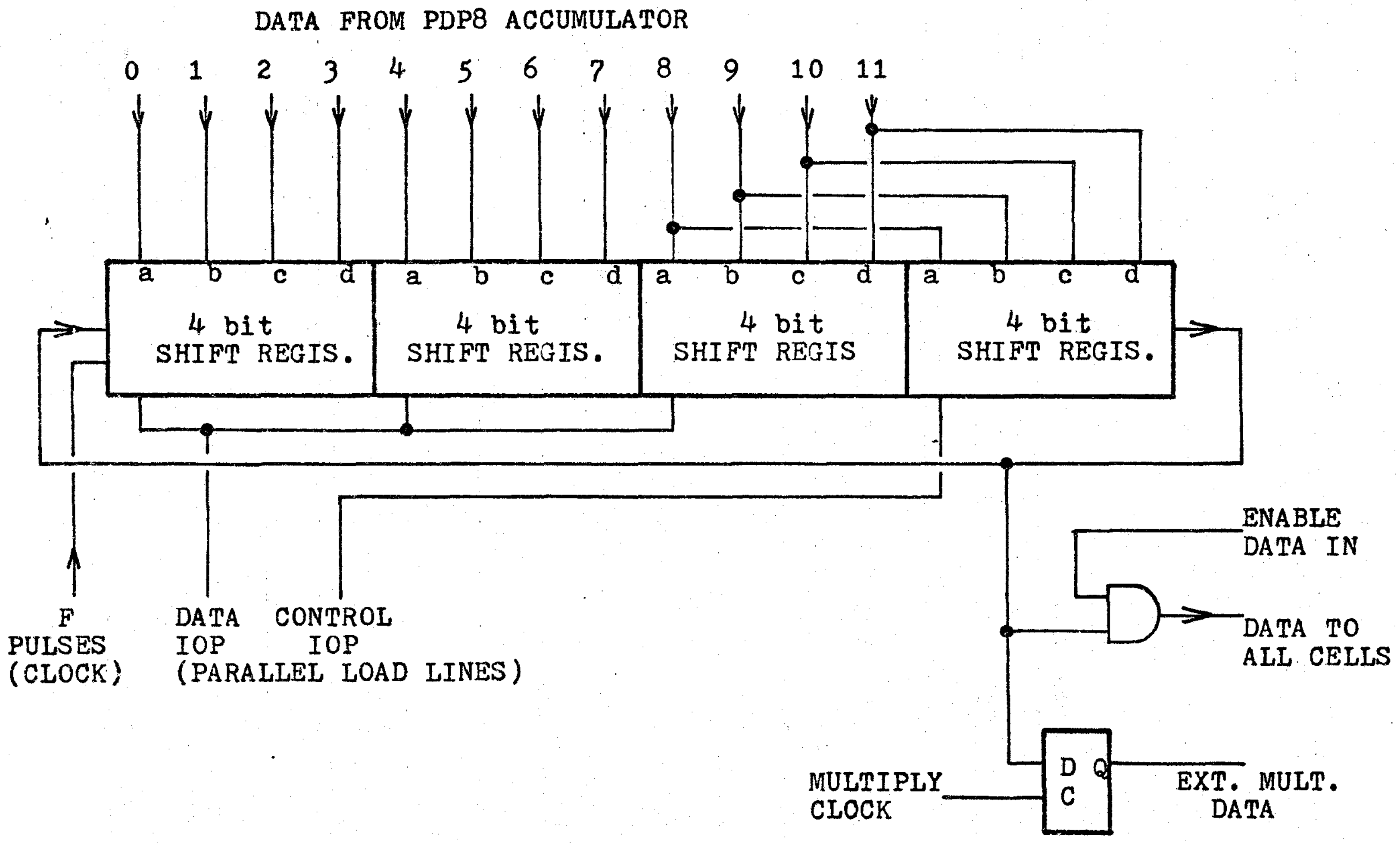


FIGURE 4.2 DATA IN REGISTER

Then a loading cycle is initiated which writes data into the selected location one bit at a time. This procedure is repeated for each location. Reading data out from the array is the reverse. One cycle reads data from the selected location into the DATA out register serially (See Figure 4.3). After this, data is read out in parallel to the PDP8 accumulator. One bit in the Instruction Register is used to select reading either the 12 data bits or the 4 control bits.

ADD SHIFT CYCLE LOGIC

Recall from Chapter 3 that multiplication was performed by a sequence of ADD cycles and SHIFT cycles. During each phase of the ADD/SHIFT cycle, different control lines have to be activated in order to enable an ADD operation after which the lines have to be reset to perform the Shift operation. The circuit to perform this is illustrated in Fig. 4.4.

During the ADD cycle the A/S line is held 'High', and this enables Read m_1, Read m_3, Write m_3. This setting allows the data flow '$m_1 + m_3$ store in m_3'. Thus the contents of m_1 are added to the contents of m_3. During the SHIFT cycle, the A/S line is held 'Low', and this enables Read m_1 and Read m_1B and write m_1. This allows the operation '$m_1 + m_1$ store in m_1'. This is equivalent to multiplying the contents of m_1 by 2, and since m_1 is a binary number, it is also equivalent to shifting m_1 by 1 bit towards the msb. The ADD/SHIFT cycle is repeated

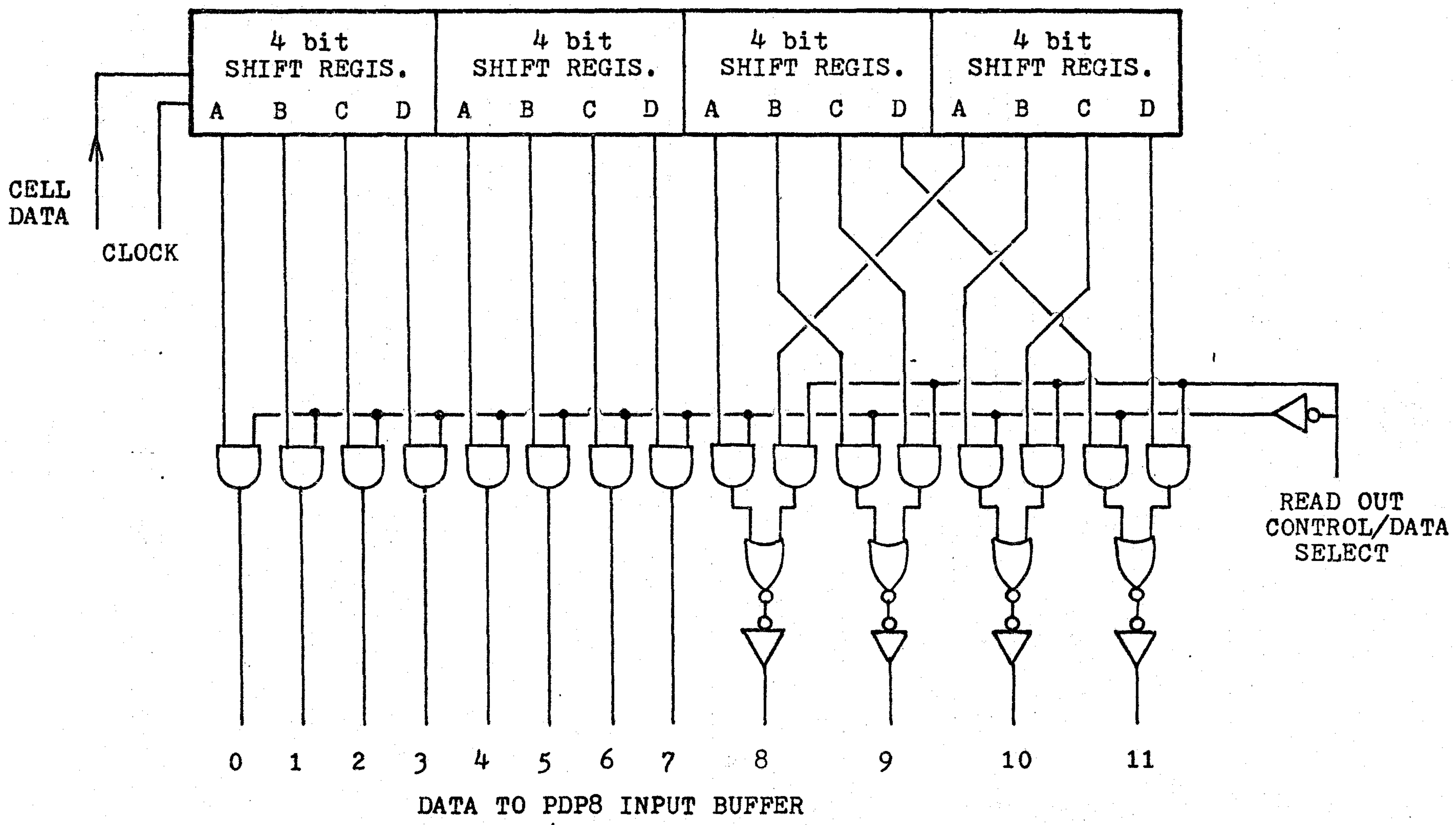


FIGURE 4.3 DATA OUT REGISTER

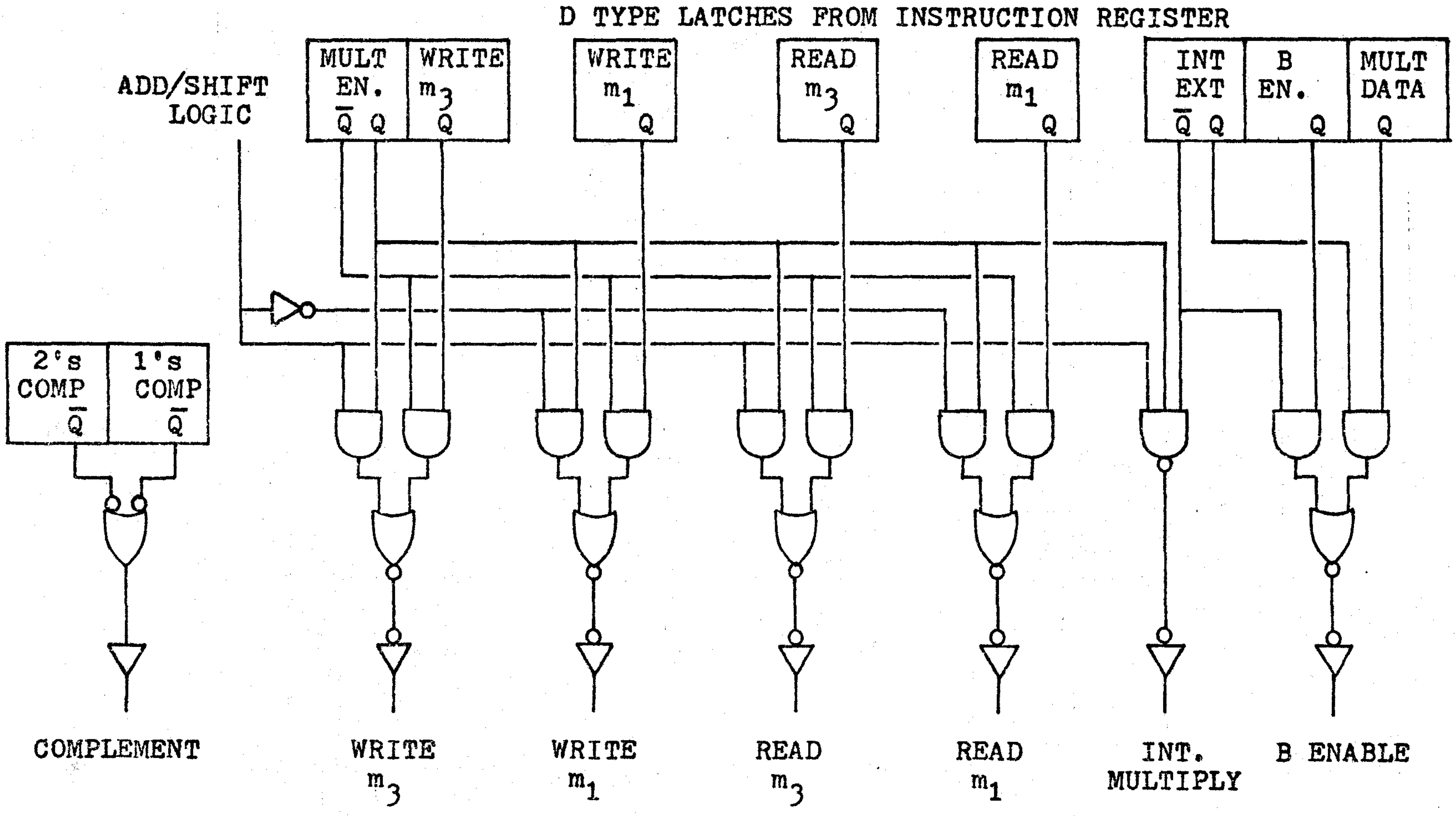


FIGURE 4.4 ADD-SHIFT LOGIC AND COMPLEMENT

12 times to complete the multiplication.

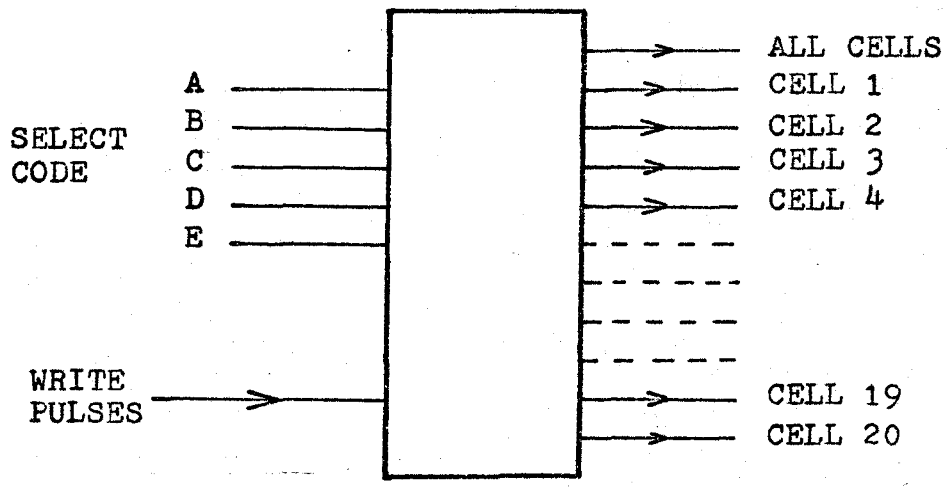
MULTIPLICATION BY A CONSTANT

A simple addition to the interface board allowed multiplication of data in the array by an externally set constant. The multiplier in this case is stored in the DATA IN register of Fig. 4.2, and the D type latch serves to hold multiplier data for a complete address cycle in much the same way as explained for in-cell multiplication.

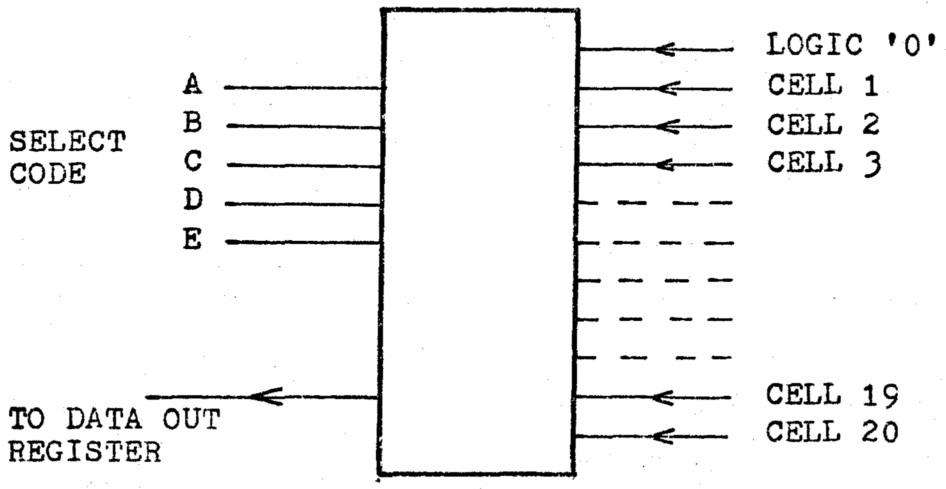
MULTIPLEXING

Communication of data between the PDP8 and the array is performed for one cell at a time. This then necessitates the use of multiplexing and demultiplexing techniques to communicate with each cell uniquely. For this purpose we have a section of multiplexers and demultiplexers capable of handling 32 lines each, of which only 20 of each are used since we only have 20 cells. As illustrated in Fig.4.5, both multiplexer and demultiplexer are controlled by 5 select lines set at the interface IR.

DATA OUT lines, one from each cell, are fed to the multiplexer/line selector, where data from one line only, selected by the IR setting is read into the DATA OUT register. This data is then read out to the PDP8 for



DIRECTING WRITE PULSES TO CELLS
BY DEMULTIPLEXING.



SELECTING CELL FOR DATA READ OUT
BY MULTIPLEXING.

FIGURE 4-5. CELL SELECTION FOR DIRECTING WRITE PULSES AND FOR DATA OUTPUT.

storage and eventually for Type-Out Display.

Reading data into the cells is somewhat different. WRITE pulses, rather than input data, are fed into the demultiplexer and directed to a cell selected by the IR setting. Input data is presented to all cells simultaneously but only the selected cell can perform writing data into memory, because only it has **WRITE** pulses. This technique has a very definite advantage over the alternative technique of feeding data to the demultiplexer, as this would require additional inhibiting operations in the cells not chosen. Recall that the absense of **WRITE** pulses or the inhibiting of **WRITE** pulses in effect inhibits a cell operation. The organization is such that line '0', selected by binary '00000', is fed simultaneously to all cells whereas lines 1 to 20 are fed to one cell each. Thus when line '0' is selected, **WRITE** pulses are presented to all cells and allows the array to perform operations in unison.

SUMMARY

The incompatibility of the PDP8's 12 bit accumulator to the 16 bit memory words of the cells pose an inconvenient problem when it is necessary to transfer data between the PDP8 and the array processor. However, the main point to note here is that in spite of the primitive interface design, the capabilities of the array are not hindered in any way. The only major disadvantage is that the programming required to perform any operations will be more complex.

It is only after a more complete and more efficient instruction set is developed that an efficient interface control system can be designed. The required instruction set would be dependent on the function capabilities required.

CHAPTER 5

OPERATE PROGRAMS

The PDP8 serves as the general purpose computer linking the array processor and the user. All information to be directed to the array must first be stored in allocated locations in the PDP8, after which the appropriate loading routine will be executed to transfer data from the assigned locations to the array on a one to one basis, where data from one assigned location will go to a particular address in the array. In a similar manner the complete instruction set is stored in the PDP8 and special routines have to be initiated which loads the IR and starts the operation. Normally one should look on the array as a special peripheral to a general purpose computer, however in this case where a significant part of the controlling is performed by the PDP8, the array becomes the main device and the PDP8 merely a programmable controller.

The OPERATE programs consisting of the subroutines, the instruction set and the allocated data storage locations is fairly elaborate, and at present occupies nearly two thirds of the PDP8's core memory. With the present distribution of functions, the PDP8 is completely tied up. Only when the interface-control design is improved to have its own memory to store microprograms and instruction set data, only then can the PDP8's role be reduced significantly. This would be desirable since the general purpose computer can then be liberated to perform other functions while the array is performing some lengthy iterations.

Following will be the description of the instruction set used to control the cell functions. Each instruction set consists of four octal numbers which are derived by grouping the 12 PDP8 accumulator bits which represents the instructions, into four groups of three bits each.

BASIC CELL CONTROL LINE LOGIC

All functions in the array processor are controlled via the INSTRUCTION REGISTER, and this in turn is set by software programs in the PDP8. The IR holds information to perform one operation only, the operation being determined by the state of the IR.

Table 5.1 shows the setting of the cell control lines to perform the cell functions defined. In the table, a '1' represents a high voltage level (3 volts) and a '0', a low voltage level (0 volts). An 'S' is used to indicate that

CONTROL LINES

CELL FUNCTIONS	SET MULT	2's COMP	ENABLE A	ENABLE B	1's COMP	Σ	DATA TRNS	READ SELECT	WRITE SELECT	WRITE PULSES
READ DATA IN	00	0	0	0	0	1	0	0000	S	011
READ DATA OUT	00	0	0	0	0	1	0	S	0	000
ADDITION	00	0	1	1	0	1	0	S	S	010
SUBTRACTION	00	1	1	1	0	1	0	S	S	010
m_1 X m_2 = m_3	11	0	1	1	0	1	0	1100	101	010
m_1 X A = m_3	01	0	1	1	0	1	0	1001	101	010
SHIFT LEFT	00	0	1	1	0	1	0	1001	001	010
SHIFT RIGHT	00	0	1	1	0	1	0	1001	001	010
LOGICAL AND	00	0	1	1	0	0	0	S	S	010
LOGICAL OR	00	0	1	0	0	1	0	S	S	010
EX-OR	00	0	1	1	0	1	0	S	S	010
1's COMP	00	0	1	0	1	1	0	S	S	010
DATA TRANSFER	00	0	0	0	0	1	S	S	S	010

__KEY__

'1'	Logic 1
'0'	Logic 0
'S'	lines to be selected as required.

__TABLE 5.1__ CONTROL LINE LOGIC FOR BASIC CELL FUNCTIONS.

one variation of the same control function is to be selected. For example, in the column for DATA TRANSFER, the 'S' indicates that the desired direction of transfer is to be set as selected by the user. An 'S' in the WRITE SELECT column indicates that one or more of the three words, m_1,m_2 or m_3 are to be selected to store the results of the computation.

INSTRUCTION SET

The instruction register layout, organized in four rows of ten bits per row, is illustrated in Fig. 5.1. When it is required to transfer an instruction from the PDP8 to the IR, the instruction consisting of 12 bits of information is first loaded into the accumulator. Of the 12 accumulator bits from the PDP8, the 2 msb are used to select one of the registers; 0 0 for IR_0, 0 1 for IR_1, 1 0 for IR_2 and 1 1 for IR_3, as shown at the left side of each row. An IOP then transfers the other 10 bits of the accumulator into the selected IR, to be used as control information.

The basic instruction set is presented in Table 5.2. The 12 accumulator bits are written as 4 octal numbers. This table is derived from Table 5.1 and Fig. 5.1. Here again, 'S' is used when a selection is required for one or more variation of the same control function. Each instruction set consists of four 12 bit instructions, one each for IR_0, IR_1, IR_2, and IR_3 where in each case, the 2 msb of the

SELECT CODE	ENABLE DATA IN		DATA SELECT	READ OUT		CELL ADDRESS				
00 IR_0	IN - 1 OUT- 0		A - 0 B - 1	DATA-0 CONT-1		16	8	4	2	1

SELECT CODE			MULTIPLY				SHIFT			
01 IR_1	CONVG. CHECK		EXT- 1 INT- 0	MULT EN.		2's COMP	L - 1 R - 0	A ENABLE	B ENABLE	INHIBIT ENABLE

SELECT CODE	BOOLEAN			DATA TRANSFER				CONVERGENCE RANGE		
10 IR_2	1's COMP	AND- 1 Σ - 0	EX-OR	E RIGHT	W LEFT	N UP	S DOWN	4 c	2 b	1 a

SELECT CODE	READ SELECT				WRITE			WRITE ENABLE PULSES		
11 IR_3	m_1 B	m_3	m_2	m_1	m_3	m_2	m_1	4 C	2 B	1 A

FIGURE 5.1 INSTRUCTION REGISTER LAYOUT

CELL FUNCTION	IR state setting in octal			
	IR_0	IR_1	IR_2	IR_3
$m_1 + m_1 = m_3$	0000	2007	4000	7142
$m_1 + m_2 = m_3$	0000	2007	4000	7242
$m_1 + m_3 = m_3$	0000	2007	4000	7442
$m_1 - m_1 = m_3$	0000	2027	4000	7142
$m_1 - m_2 = m_3$	0000	2027	4000	7242
$m_1 - m_3 = m_3$	0000	2027	4000	7442
$m_1 \times m_2 = m_3$	0000	2307	4000	7552
$m_1 \times \text{const} = m_3$	0000	2107	4000	7552
$m_1 \div 2$	0000	2017	4000	7112
SH 1 Bit Left	0000	2007	4000	7112
SH 1 Bit Right	0000	2017	4000	7112
LOGIC OP.				
$m_1 \cdot m_2$	0000	2006	4400	7242
$m_1 \cdot m_3$	0000	2006	4400	7442
$m_1 + m_2$	0000	2004	4000	6342
$m_1 + m_3$	0000	2004	4000	6542
$m_1 + m_2 + m_3$	0000	2004	4000	6742

TABLE 5.2 BASIC INSTRUCTION SET

(contd.)

CELL FUNCTION	IR_0	IR_1	IR_2	IR_3
$m_1 \oplus m_2$	0000	2006	4200	7242
$m_1 \oplus m_3$	0000	2006	4200	7442
COMP. m_1	0000	2004	5000	6142
Rd DATA IN	10SS	2000	4000	60S3
Rd DATA OUT	00SS	2000	4000	6S00
CONTROL OUT	01SS	2000	4000	6S00
TRANSFER	0000	2000	4SS0	7043

note: Four octal numbers form one set of instructions. Each of the octal numbers represent the 12 accumulator bits of the PDP8.
Of the 12 Acc. bits, only 10 are used as information for the IR, the 2 msb. are used to select the relevant IR.

In the Table, an 'S' is used to indicate that the instruction code is to be selected, based on the IR shown in FIG. 5.1, as desired by the user

TABLE 5.2 (contd.) BASIC INSTRUCTION SET

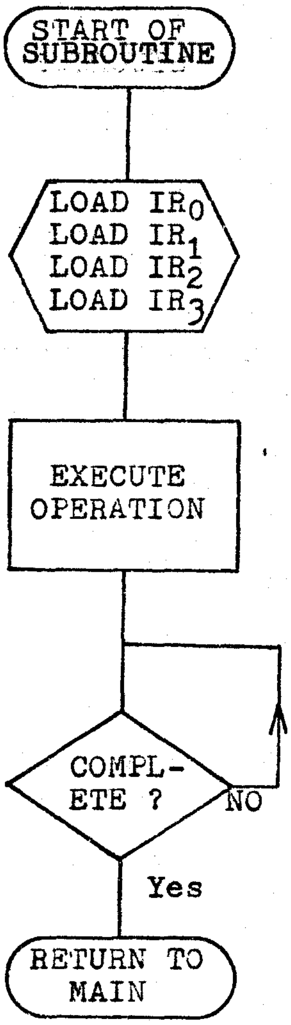
12 bit instruction are used to select the appropriate row of register .

OPERATE PROGRAMS

To facilitate the use of the array processor, a subroutine library has been prepared which enables the user to perform any cell operation, input-output transfer or type-out display by just using a mnemonic code of up to six characters which calls the relevant subroutine. This library of programs, called the OPERATE PROGRAMS is completely listed in the appendix. The language which is used for these programs is PAL. Since the PDP8 has only a 4K memory, PAL which is the most efficient, is most appropriate.

The basic subroutine to perform a cell operation is illustrated in the flow chart of Fig. 5.2. Basically, it involves loading the appropriate instructions into IR_0, IR_1 IR_2 and IR_3, and generating an IOP (input-output pulse) from the PDP8 to initiate the cycle. The program, also illustrated in Fig. 5.2 is typical of all the programs for cell functions. The data for the instruction registers is available from Table 5.2.

Programs for the communication of data between the array and the PDP8 is somewhat more involved. In these programs, a link must first be made between the PDP8 and the cells of the



```
START, 0000
       CLA
       TAD I0
       LOAD
       TAD I1
       LOAD
       TAD I2
       LOAD
       TAD I3
       LOAD
       CLA
       EXECUTE
       FLAG
       JMP .-1
       JMP I START

   I0, 0000
   I1, 2307
   I2, 4000
   I3, 7552
```

FIGURE 5.2. TYPICAL CELL FUNCTION ROUTINE, FLOW CHART AND PROGRAM FOR $m_1 \times m_2 = m_3$, USING PAL.

array, one cell at a time. For example, the program for writing data into the array performs the following steps:-

1) Load data register
2) Select cell address
3) Execute WRITE routine

These steps, illustrated in the flow chart of Fig. 5.3, are repeated for each cell location.

The program for reading data out from the array, is illustrated in the flow chart of Fig. 5.4. It performs the following steps for each cell:-

1) Select cell address
2) Transfer cell data to DATA OUT register
3) Transfer data to PDP8.

The complete list of routines for writing data into the array and reading data out from the array is listed in the appendix.

Another type of routine in the OPERATE PROGRAMS is the DISPLAY routines. These, also listed in the appendix, are used to display via the teletype machine, any selected set of data from the cells of the array. Currently, the data is typed out in a rectangular 5 X 4 array similar to the array organization of the cells. Actually, these programs display data stored in allocated locations of memory. Data answers computed in the array, have to be transferred to these locations first, by using the programs for reading data out of the array.

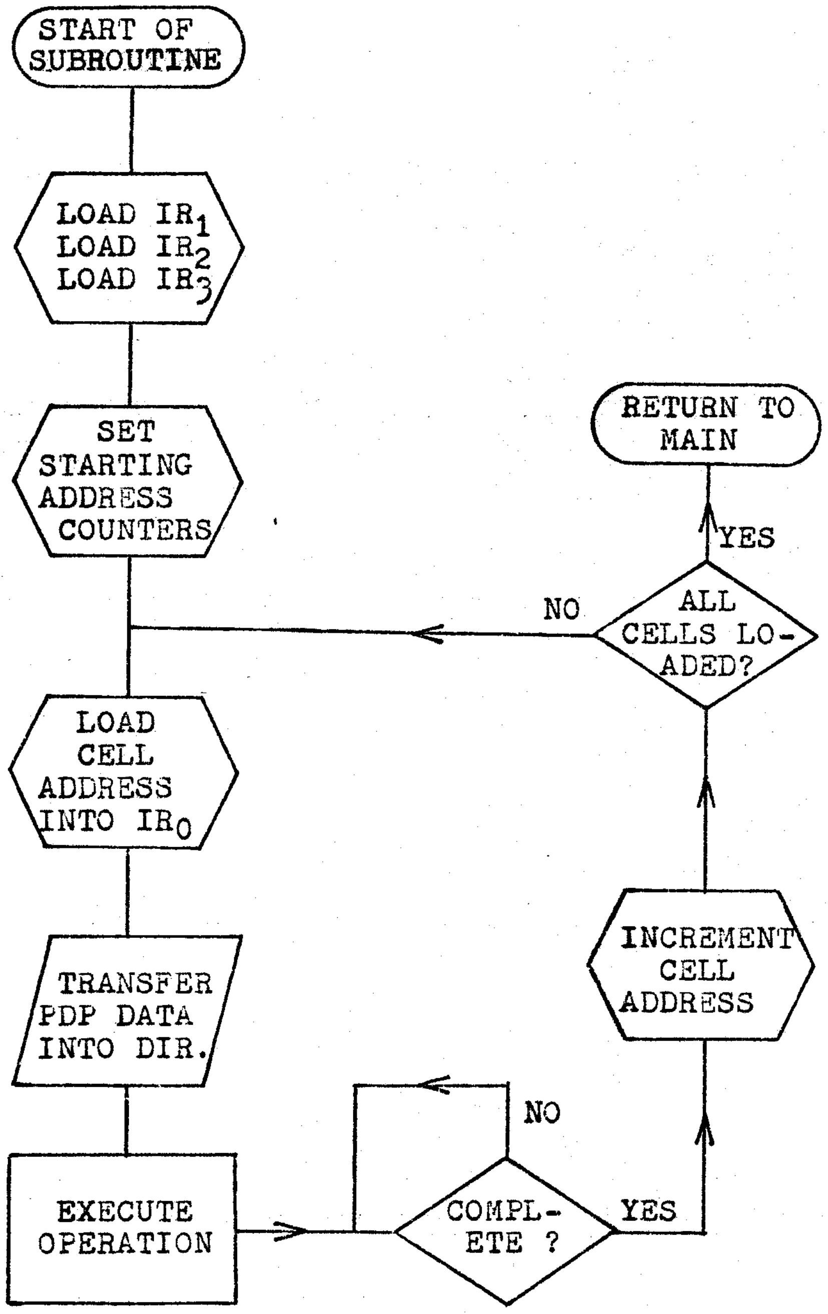

START OF SUBROUTINE
LOAD IR_1 LOAD IR_2 LOAD IR_3
SET STARTING ADDRESS COUNTERS
LOAD CELL ADDRESS INTO IR_0
TRANSFER PDP DATA INTO DIR.
EXECUTE OPERATION
COMPL-ETE ?
NO
YES
INCREMENT CELL ADDRESS
ALL CELLS LO-ADED?
NO
YES
RETURN TO MAIN

FIGURE 5.3. FLOW CHART FOR WRITING DATA INTO THE ARRAY.

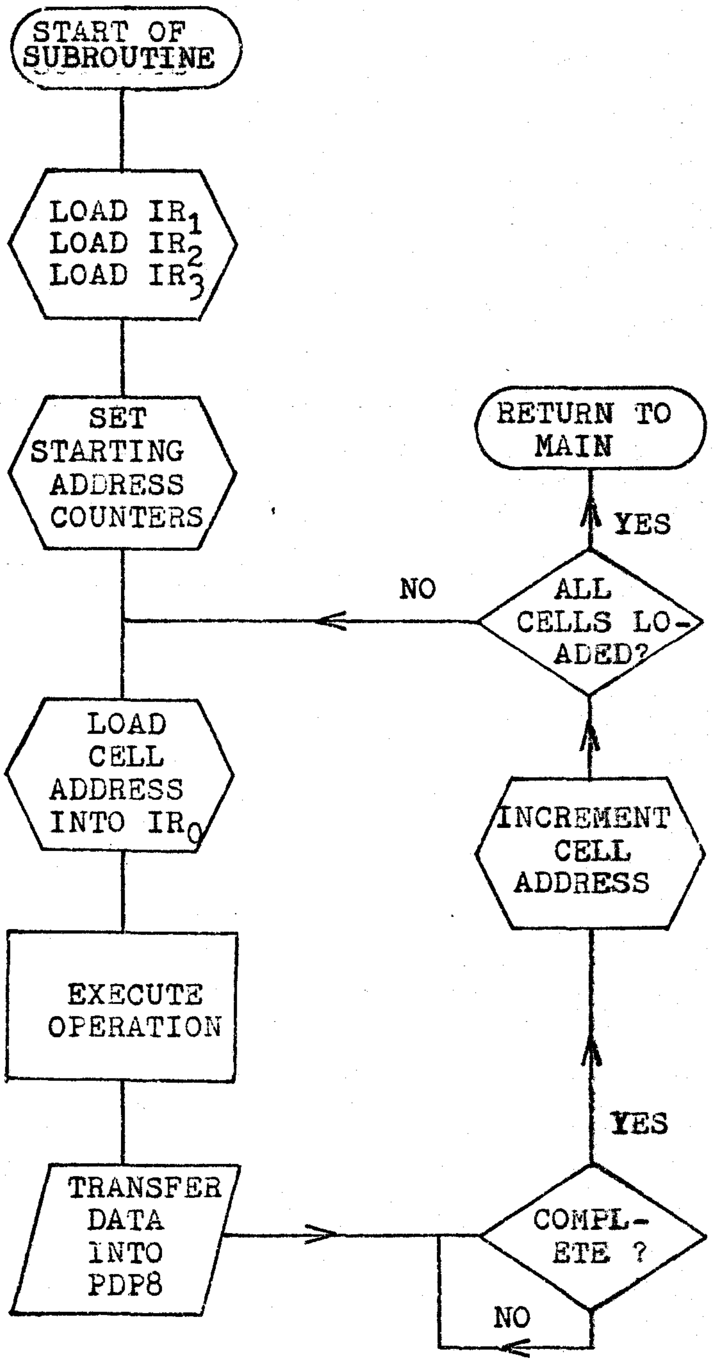


FIGURE 5.4. FLOW CHART FOR READING DATA FROM THE ARRAY.

STORAGE ALLOCATIONS

Prior to any computations, all necessary data and control information together with the OPERATE programs and the main program must be stored in allocated sections of the PDP8 memory. After any computation, data answers from the cells are also read into allocated locations of the PDP8 before they can be displayed. The idea of of using the PDP8 memory as a buffer storage for input data and for output answers, though seemingly inefficient and redundant is necessary to allow for easier programing.

The complete list of memory allocations is presented in Table 5.3. Note from the table that the 16 bit words of the cells are stored in two sections, one section for 12 bits of data and the other section for the 4 bits of control information. This arrangement is necessary because of the incompatibility of the PDP8's 12 bit memories to the 16 bit words of the cells of the array.

Note also, from Table 5.3, that the OPERATE programs and the data storage locations together occupy about 60% of available memory, leaving less than 3000_8 locations for the Main programs. If the array processor is to be used as a permanent peripheral to the PDP8 or any other small computer, this situation will be most inconvenient. It can be alleviated only by using a more sophisticated interface and control unit design.

MEMORY LOCATIONS	CONTENTS		
0 - 177	Program Constants		
200 - 2777	Main Programs		
3001 - 3025	m_1	12 data bits, answers from the array	
3031 - 3055	m_2		
3101 - 3125	m_3		
3201 - 3225	m_1	4 control bits, read out from the array	
3231 - 3255	m_2		
3301 - 3325	m_3		
3401 - 3425	m_1	12 data bits, data for the array	
3431 - 3455	m_2		
3510 - 3525	m_3		
3601 - 3625	m_1	4 control bits, data for the array	
3631 - 3655	m_2		
3701 - 3725	m_3		
4000 - 4777	INPUT/OUTPUT		OPERATE PROGRAMS
5000 - 5777	DISPLAY ROUTINES		
6000 - 7600	CELL OPERATIONS		
7600 - 7777	PDP8 Loader Programs		

Nb. the memory locations are in octal numbers.

TABLE 5.3 STORAGE ALLOCATIONS

EXAMPLE

The rather extensive list of OPERATE programs however, enables us to write very simple main programs. One such program used to demonstrate multiplication between m_1 data and m_2 data, is presented in the following example, where all array operations are executed by calling the subroutine with a **mnemonic** code of up to six characters.

The following test program (in PAL) :-

1) clears all cells.
2) writes Data and Control information, already stored in the allocated sections of the PDP8 memory, into m_1 and m_2 of all cells.
3) multiples m_1 by m_2 and stores the results in m_3.
4) reads answers in m_3 to the PDP8.
5) types out a display of the answers in m_3.

*200	starting address of main program.
CLA	clear PDP8 accumulator.
CLEAR	clear all memories in the array.
WR1D	write data into m_1.
WR1C	write control bits into m_1.
WR2D	write data bits into m_2.
WR2C	write control bits into m_2.
MULT	multiply m_1 by m_2, store in m_3.
RD3D	read m_3 data into PDP8.
RD3C	read m_3 control bits into PDP8.
TP3D	display m_3 data via teletype.
TP3C	display m_3 control via teletype.
HLT	halt.

SUMMARY

The main purpose of the OPERATE program library is to have available all subroutines which would make usage of the array processor easier. Typically, all the cell functions which are stored in subprograms can be executed by merely transferring program control to the subroutines. At present, the list of OPERATE programs is complete in that it has available, all the functions we require. Of course if any other functions are introduced, the list can be expanded accordingly. What could be a major addition to this list of programs are routines which command a sequence of cell operations to perform some standard sequential iteration. At present, such programs like relaxation solution for Laplace equations are considered as main programs. Other such programs could include for example, programs to perform correlation between data in the array, various matrix operations, and in fact any algorithms that may be useful.

This chapter has served as the final descriptive chapter of the array processor. The chapters 2, 3, 4 and 5 are intended to explain the overall system from the basic cell to the program usage of the system. It is expected that the user, in addition to this thesis, become familiar with the PDP8 computer, and with programming in PAL, in order to make use of the system.

CHAPTER 6

CONCLUSION

The field of parallel processors is still very young. Although a fair amount of work is being done in it, each effort appears only to open up more areas of application and to reveal how primitive the state of developments still is in this area. To date, except for the simple associative memory of Lee and Paull, all parallel processors, notably the Berkeley Array processor, the cellular APL computer of Montana State University and the ILLIAC IV of the University of Illinois, have no commercial application and are mainly special purpose, experimental models.

This project was intended mainly to provide a base from which investigations can be made on the applications of parallel processing arrays to the fields already mentioned in this thesis such as time averaging functions, partial differential equations with Laplacian Field problems in particular, and possibly for operation on multidimensional problems such as those encountered in optimization.

It is only with a good understanding of the basic capabilities and limitations of the basic cell, that a realistic approach can be made in investigating the areas mentioned above, and it is hoped that this thesis will serve to foster such understanding.

The project itself has by no means, come to an end. Rather, it was a step in developing a system with which it is expected that a great deal more work can be done. Thus far, the system that was implemented consists of a workable array of 20 cells, arranged in a rectangular geometry, with fixed neighbourhood, where the neighbours are the cells to the north, south, east and west. The cell consisting of three memory words and one processor is fairly complex, being able to do arithmetic and logic functions between data in the cell. Control of the array is essentially centralized but each cell has some local autonomy. This is all interfaced to a PDP8 computer. A set of programs has been developed which executes the complete list of cell operations. Except for test programs and the very popular program for solving Laplacian Field problems, no other programs or algorithms have been developed. This then is the area in which a great deal more work can be done. Most immediate, especially with the Digital Instrumentation Group in mind, are the algorithms needed to perform correlation, autocorrelation, convolution and the Fast Fourier Transform and possibly even the Walsh Function. At this stage, any algorithms developed can at most be demonstrated using stored waveform data. If functions are to be processed in real time, an efficient data interface design must be implemented to allow very high input-output data exchange rates. In the present design, all the cell functions are performed sequentially and those functions such as addition and subtraction which can be performed in one address cycle can be performed at a optimum time of approximate 4 microseconds. Multiplication however which takes 12 ADD/SHIFT cycles will require 96 microseconds. These may very well be the limiting factors when real-time processing is considered. The solution to this problem lies with the development of cells which performs the cell function in parallel. Such cells would be very complex and massive. However, with the development of Large Scale Integration, a cell with parallel arithmetic and logic could be readily realized. In connection with this concept, we note that the cellular array techniques for multiplication and division such as those proposed by J. Majithia (16) would be quite appropriate for this use, particularly if and when L.S.I. can produce these arrays.

The existing interface design discussed in Chapter 4 is simple and crude. The Instruction Register which controls the function to be executed is set directly by the PDP8 computer, and only one function state can be set at a time. Moreover, the instruction register which is 40 bits long requires 4 cycles from the computer, whose accumulator has only 12 bits, in order to set it. This requires lengthy programming and slows down the operations. This is an area where the present system can be greatly improved.

It calls for a interface-control design where the instruction register can be reduced to a compatible 12 bits if the PDP8 is used or 16 bits if a machine is available with a 16 bit accumulator. This would involve implementing a network which decodes instruction register information to set the control lines of the cell array. Consequently a convenient instruction set will have to be developed which allows easy programming. In addition, it is proposed that a small memory be incorporated in any new interface-control design for the purpose of storing microprograms. The microprograms may include, not just basic cell operations instructions but possibly, subprograms for complete sequences of iterations as well. This would liberate the general purpose computer for other function, while array computations are being executed, and would represent a major improvement to the present system. The size of the array itself can be expanded by simply wiring in more cells. However, it is expected that the 20 cells array will be adequate for present requirements.

It appears quite certain now, that along with the great speed with which the general field of digital devices, instruments and systems are advancing, that the concept of a cell with memory and logic has an important role. Its ultimate development depends very much on the advancement in L.S.I. technology and the development of algorithms that can make effective use of the sophisticated nature of the Computing Memory Cell.

APPENDIX

CONTENTS PAGE

COLUMN NUMBER 7	6	5	4	3	2	1	ROW NUMBER
7400	7430	7400	7430	R13	7408	7400	1
7400	7400	7400	7481	7481	7408	7416	2
7400	7410	7410	7481	7481	7408	7416	3
7451	74H183	7486	7481	7481	74H11	7451	4
	7476	7475	7475	7404	74H11	7475	5
7400	7430	7400	7430	R63	7408	7454	6
7400	7400	7400	7481	7481	7408	7416	7
7400	7410	7410	7481	7481	7408	R81	8
7451	74H183	7486	7481	7481	74H11	7451	9
	7476	7475	7475	7404	74H11		10

Nb. All the above I.C. chips have the prefix 'SN'.

Each chip location is identified by the ROW number followed by the Column number.

TABLE A.1 P.C. BOARD LAYOUT FOR 4 CELLS

COLUMN NUMBER

ROW NUMBER	7	6	5	4	3	2	1
1		R16	R15	R14	R13	R12	
2	7417	7417	7417	7417	7417	7404	
3	7416	7416	7416	7416	7400	7400	
4			74193	MC 4040P	74193	7473	
5		7430	7404	7486	7400	7473	
6		7410	7486	7420	7473	74H11	
7		7410	7400	7400	7451	7406	
8			7404	7404	7430		
9							
10							

Nb. 'MC' specifies a MOTOROLA I.C. Chip.

<u>TABLE A.2</u> LAYOUT OF CONTROL BOARD

COLUMN NUMBER

7	6	5	4	3	2	1	ROW NUMBER
	74156	7475	7475	7475	7475	7475	1
	7475	7475	7475	7475	7475	7475	2
R37	7416	R35	R34	7417	R32	7417	3
7416	7416	7416	9495	7495	7495	7495	4
7451	7451	7410	7404	7451	7451	7451	5
7416	7495	7495	7495	7495	7408	7408	6
7400	R76	7416	7416	7416	R72	7416	7
		R85		R83		R81	8
							9
							10

Nb. 'R' stands for 'Resistor Platform'
R85, for example, is the Resistor Platform
of row 8 and column 5.

TABLE A.3 LAYOUT OF INTERFACE BOARD

FUNCTION	IOPs	CODE
CLEAR IOP and START IOP	5-1, 5-4	6105
LOAD IR, CLEAR ACC.	7-1, 7-4	6107
LOAD DATA IN REGISTER DATA BITS	IOP 2	6112
LOAD DATA IN REGISTER CONTROL BITS	IOP 4	6114
READ DATA BITS INTO ACC.	IOP 1	6111
READ CONTROL BITS INTO ACC.	IOP 1	6111
FLAG 1 CYCLE TEST.	1-1	6101
FLAG 2. CONVERGENCE	2-2	6102

TABLE A.4 LIST OF IOPs

LIBRARY OF OPERATE PROGRAMS

The neumonic code at the left is the symbolic code of the subroutine in question.
The rest of each line is the command in PAL to jump to the appropriate subroutine.

RD1D = JMS I 20	READ DATA FROM THE ARRAY
RD2D = JMS I 21	
RD3D = JMS I 22	eg. RD3D means, READ m_3 DATA.
RD1C = JMS I 23	
RD2C = JMS I 24	
RD3C = JMS I 25	
WR1D = JMS I 26	WRITE DATA INTO THE ARRAY
WR2D = JMS I 27	
WR3D = JMS I 30	eg. WR2C means, WRITE m_2 CONTROL
WR1C = JMS I 31	
WR2C = JMS I 32	
WR3C = JMS I 33	
TP1D = JMS I 34	TYPE-OUT DISPLAY OF ANSWERS
TP2D = JMS I 35	FROM THE ARRAY
TP3D = JMS I 36	
TP1C = JMS I 37	eg. TP3D means, TYPE m_3 DATA
TP2C = JMS I 40	
TP3C = JMS I 41	
TP1DX = JMS I 42	TYPE-OUT DISPLAY OF INITIAL
TP2DX = JMS I 43	DATA FOR THE ARRAY
TP3DX = JMS I 44	
TP1CX = JMS I 45	eg. TP2CX means, Type m_2 CONTROL
TP2CX = JMS I 46	data intended for the array
TP3CX = JMS I 47	
CLEARPAM = JMS I 50	CLEAR m_1, m_2, m_3 of all cells
ROTL = JMS I 51	SHIFT m_1 data one bit left of right
ROTR = JMS I 52	
CMULT = JMS I 53	Mult. m_1 by externally set constant
MULT = JMS I 54	Mult. m_1 by m_2, store in m_3

TABLE A.5 LIST OF OPERATE PROGRAMS

ADD121 = JMS I 55	ADDITION
ADD122 = JMS I 56	
ADD123 = JMS I 57	eg. ADD131 means, ADD m_1 to m_3
ADD131 = JMS I 60	
ADD132 = JMS I 61	store in m_1
ADD133 = JMS I 62	
TCOMP1 = JMS I 63	
TCOMP2 = JMS I 64	2's Comp. of m_1, m_2, m_3,
TCOMP3 = JMS I 65	and store in same memory
COMP = JMS I 66	1's Comp. of m_1
EXOR = JMS I 67	EXCLUSIVE-OR $m_1 \oplus m_2 = m_3$
BAND = JMS I 70	BOOLEAN AND: $m_1 . m_2 = m_3$
OR = JMS I 71	BOOLEAN OR $m_1 + m_2 = m_3$
SUB123 = JMS I 72	
SUB133 = JMS I 73	SUBTRACTION
TRN12U = JMS I 150	TRANSFER DATA FROM ONE CELL LOCATION TO ANOTHER
TRN12D = JMS I 151	
TRN12L = JMS I 152	
TRN12R = JMS I 153	eg. TRN12R means, TRANSFER m_1 to m_2 of the cell to the RIGHT.
TRN32U = JMS I 154	
TRN32D = JMS I 155	
TRN32L = JMS I 156	
TRN32R = JMS I 157	
CLEAR3 = JMS I 160	CLEAR m_3 only, of all cells
SUB1 = JMS I 161	
SUB2 = JMS I 162	
SUB3 = JMS I 163	spare subroutines
SUB4 = JMS I 164	
SUB5 = JMS I 165	
SUB6 = JMS I 166	

TABLE A.5 (contd) LIST OF OPERATE PROGRAMS

LIST OF ABBREVIATIONS

A.U.	Arithmetic Unit
P.U.	Processing Unit
IR	Instruction Register
IS	Instruction Set
CLIM	Cellular Logic in Memory
COMP	Complement
2's COMP	Two's Complement
CONVG.	Convergence
IOP	Input/Output Pulse
L	Left
R	Right
OP.	Operation
Rd	Read
Sh.	Shift
lsb	Least Significant Bit
msb	Most Significant Bit
ff	Flip Flop
MHz	Mega Hertz
const.	Constant
ARITH.	Arithmetic
Fig.	Figure
ns	Nanosecond
Ext.	External
Int.	Internal

TABLE A.6 LIST OF ABBREVIATIONS

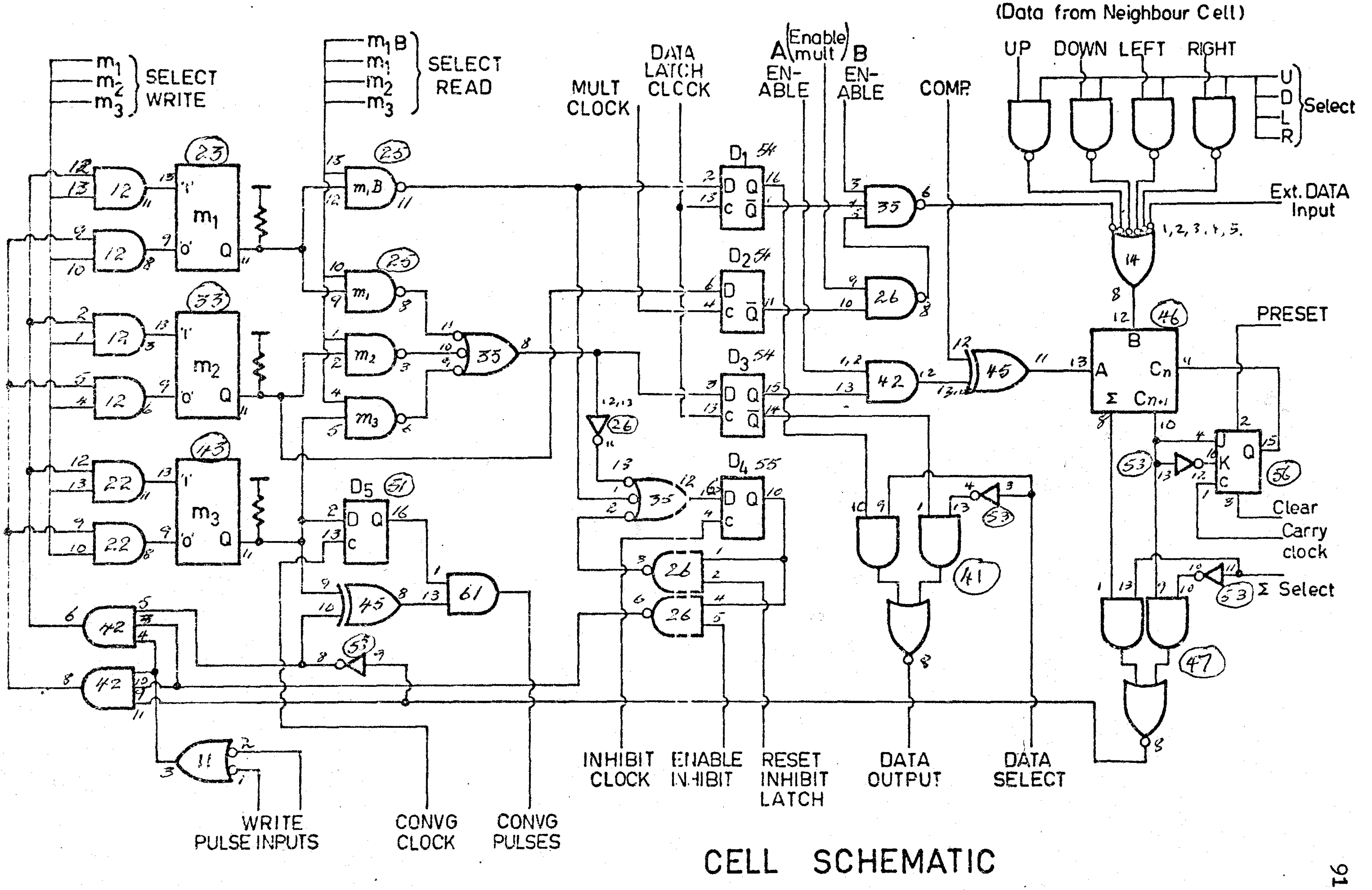


FIGURE A.1 WIRING SCHEMATIC OF CELL 1

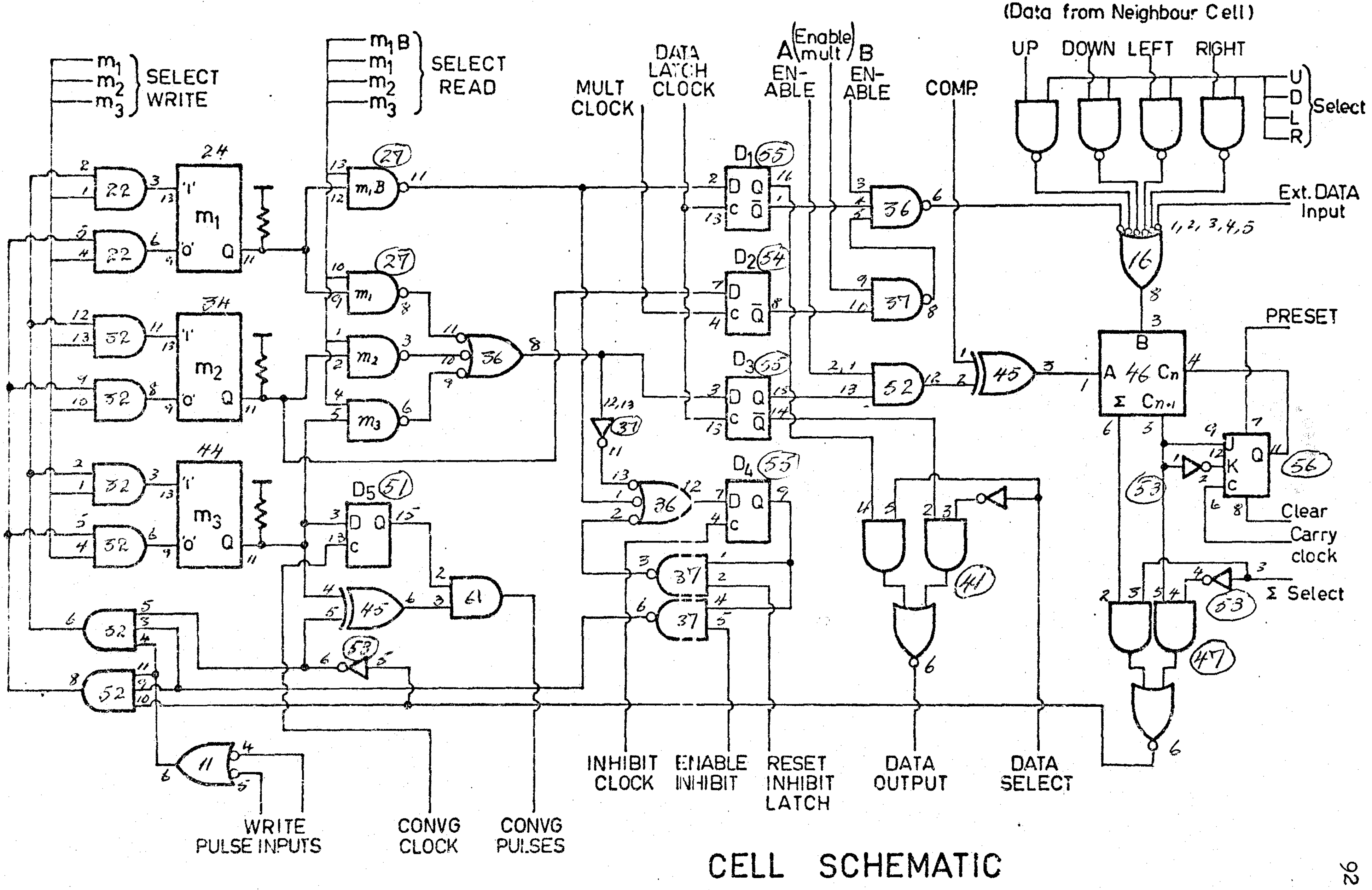


FIGURE A.2 WIRING SCHEMATIC OF CELL 2

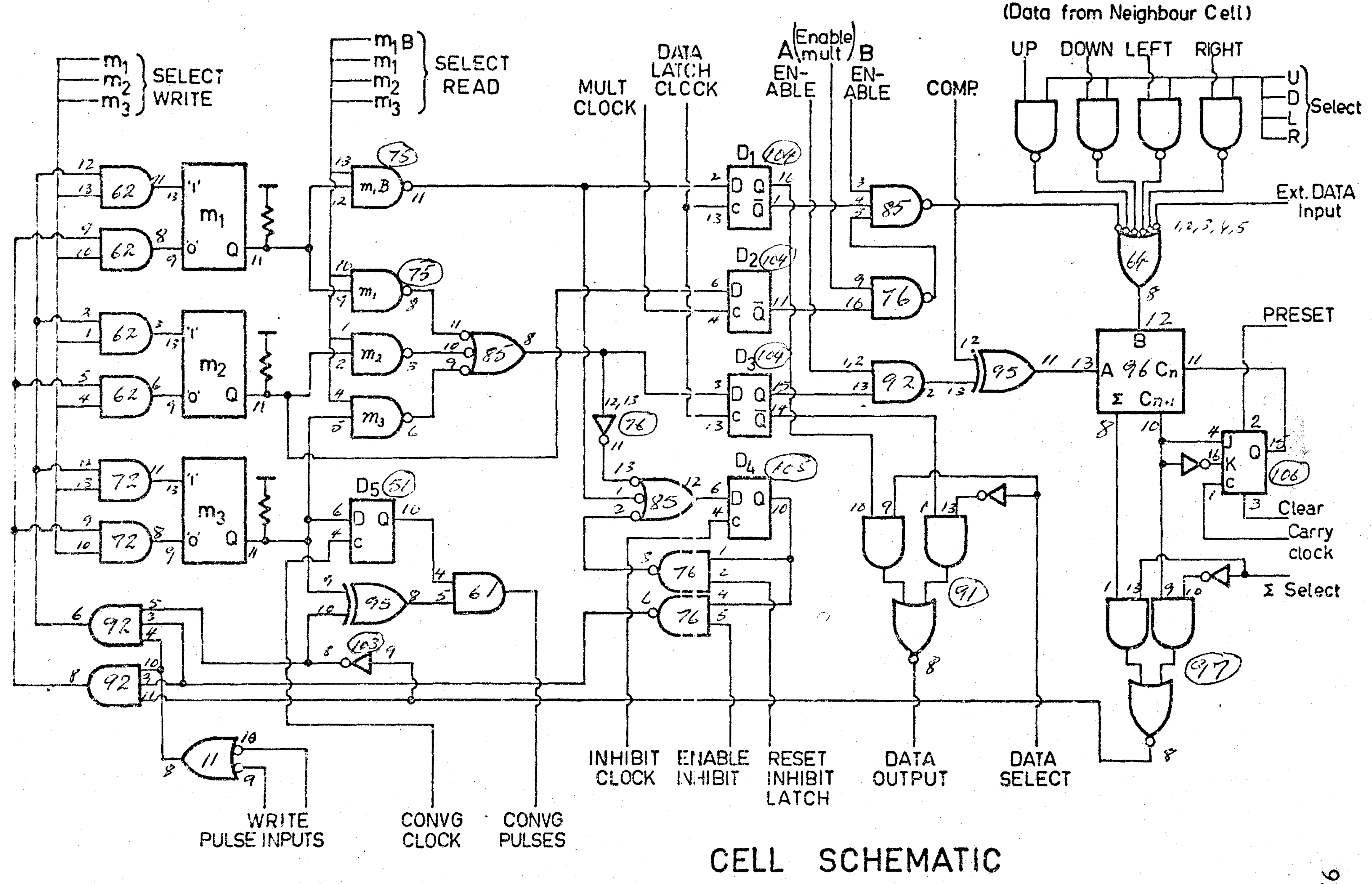


FIGURE A.3 WIRING SCHEMATIC OF CELL 3

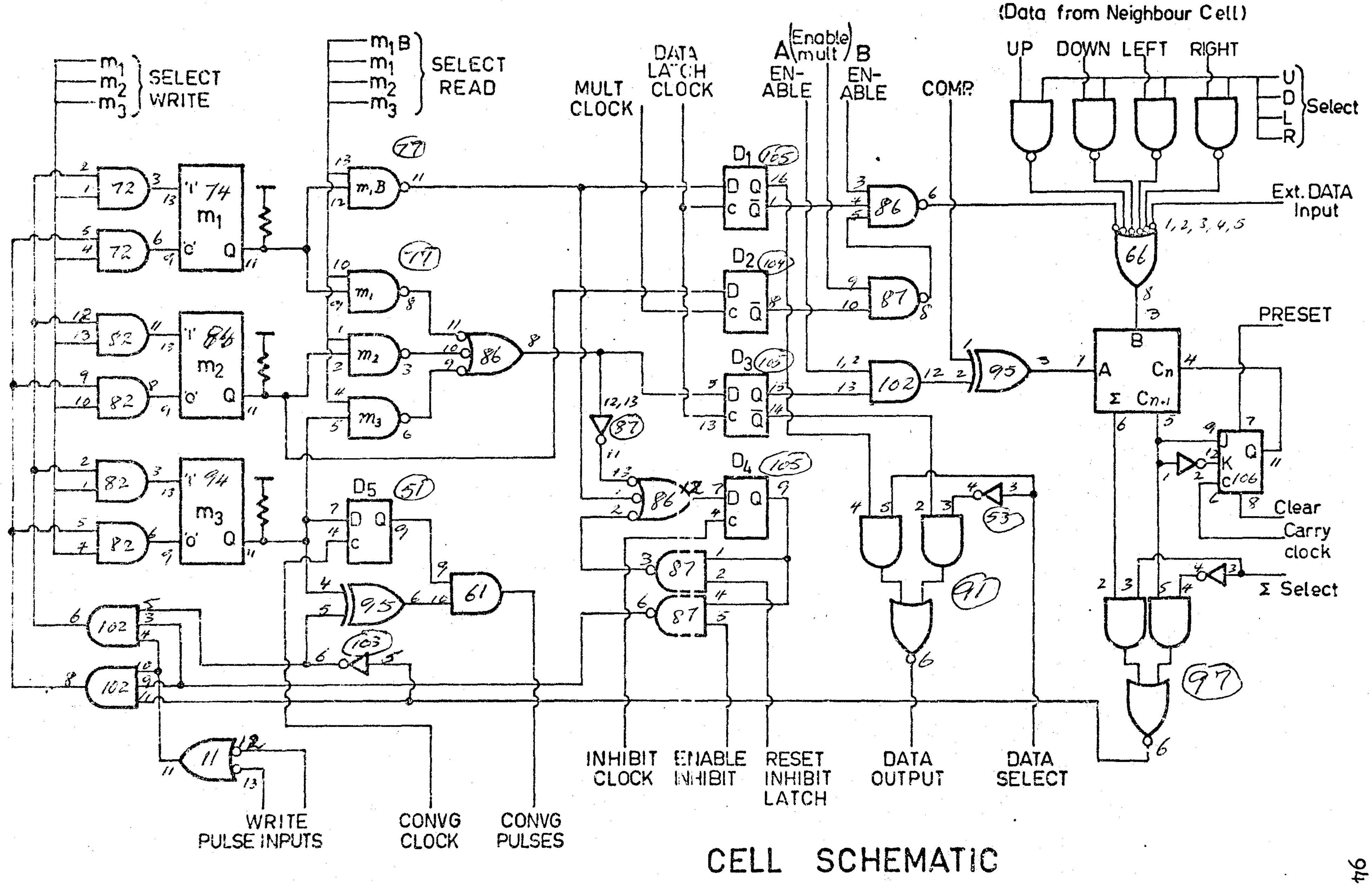


FIGURE A.4 WIRING SCHEMATIC OF CELL 4

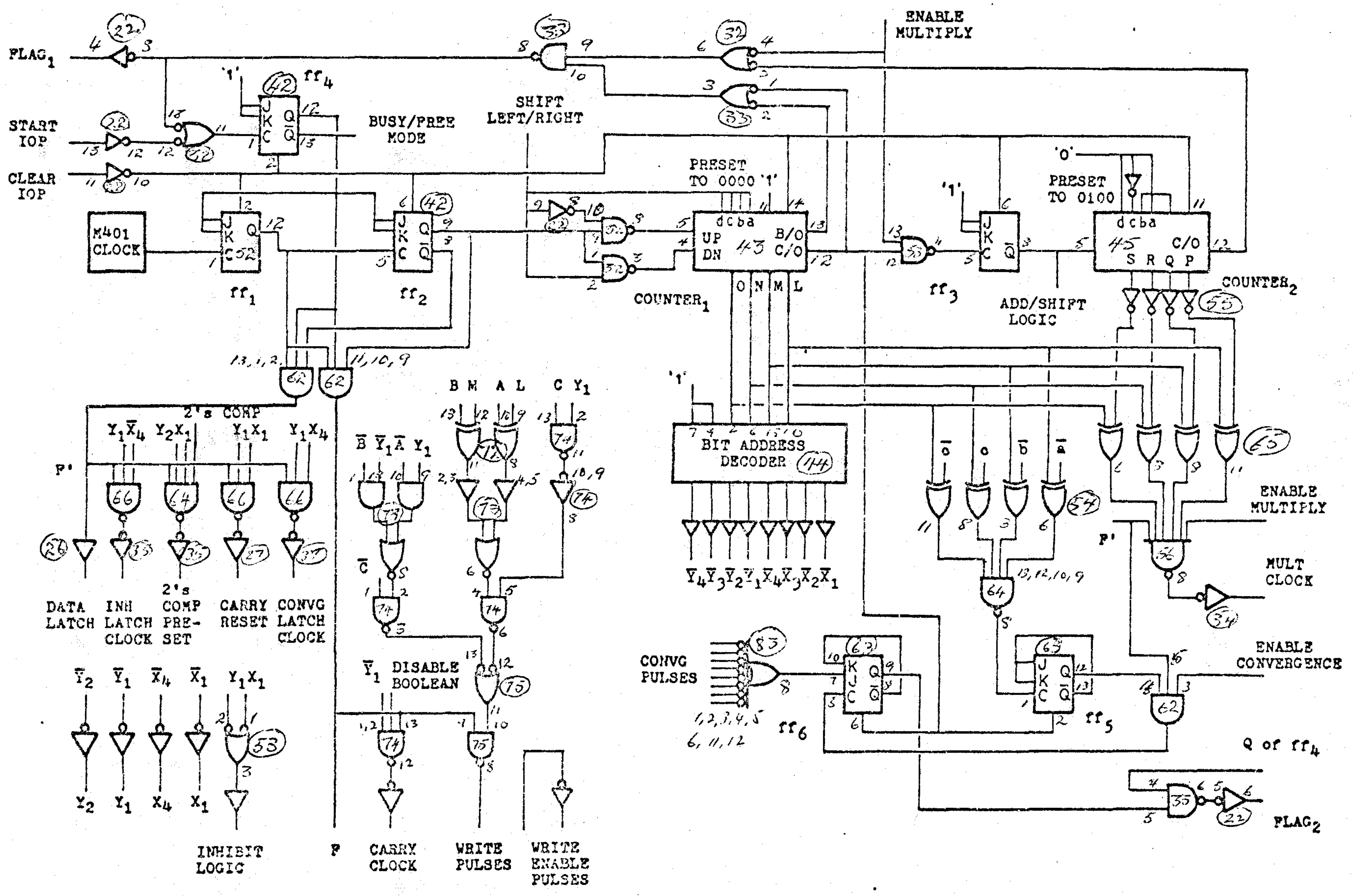


FIGURE A.5 WIRING SCHEMATIC OF CONTROL BOARD

REFERENCES

1) S.H. Unger, "A Computer Oriented towards Spatial Problems". Proc. IRE, Vol. 46, Oct. 1958. pp.1744-1750

2) D.L. Slotnick, W.C. Borck and R.C. McReynolds, "The Solomon Computer". Proc. AFIPS, Fall Joint Computer Conf., 1962, pp. 97-107.

3) J. Gregory and R. McReynolds, "The Solomon Computer" IEEE Trans. Electronic Computers, Vol. EC-15 Dec. 1961, pp. 718-722.

4) C.Y. Lee, "Intercommunicating Cells: Basis for a distributed logic computer". Proc. AFIPS, Fall Joint Computer Conf. 1962, pp. 130-136.

5) C.Y. Lee and M.C. Paull, "A Content Addressable distributed logic memory with application to information retrieval". Proc. IEEE, Vol.51, June 1963, pp. 924-932

6) J. Holland, "A Universal Computer capable of executing an arbitary number of sub-programs simultaneously". Proc. Eastern Joint Computer Conf. 1959, pp 108-113.

7) B.A. Cranes and J.A. Githens, "Bulk Processing in Distributed Logic Memory". IEEE Trans, Electronic Computers, Vol.EC-14, April, 1965, pp.186-196.

8) C.C. Yang and S.S. Yau, "A Cutpoint Cellular Associative Memory". IEEE Trans. Electronic Computers, Vol.EC-15, August, 1966, pp. 522-529.

9) G.H. Barnes et al, "The ILLIAC IV Computer" IEEE Trans. Computers, Vol.C-17, August, 1968 pp. 746-757.

10) W.Y.Dere and D.J. Sakrison, "Berkeley Array Processor". IEEE Trans. Computers, Vol.C-19, May 1970, pp. 444-447.

11) W.H. Kautz, "Cellular Logic in Memory Arrays". IEEE Trans. Computers, Vol.C-18, August 1969, pp. 719-727.

12) J.H. Huttenhoff and R.R. Shively, "Arithmetic Unit of a Computing Element in a Global, Highly Parallel Computer". IEEE Trans. Computers, Vol.C-18 , August 1969, pp. 695-698.

13) H.S. Stone, "A Logic in Memory Computer". IEEE Trans. Computers, Vol.C-19, Jan. 1970, pp.73-78.

14) K.J. Thurber and J.W. Myrna, "System Design of a Cellular APL Computer", IEEE Trans. Computers, Vol. C-19, April, 1970, pp. 291-303.

15) D.A. Lawrence, "A Computing Memory: Design and Applications with special reference to Correlation". Master's Thesis, McMaster University, 1970.

16) J. Majithia, "A Digital Moments Analyser: Design and Error Characterstics". Ph.D. Thesis, McMaster University, 1971.

17) N.R. Scott, Electronic Computer Technology , McGraw-Hill, New York, 1970.