A PARALLEL COMPUTING SYSTEM

A PARALLEL COMPUTING SYSTEM

by

JORGE V. ROITMAN, B.Eng.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Engineering

McMaster University

April 1972

MASTER OF ENGINEERING (1972) McMASTER UNIVERSITY
(Electrical Engineering) Hamilton, Ontario

TITLE: A PARALLEL COMPUTING SYSTEM

AUTHOR: JORGE V. ROITMAN, B. Eng.
(Universidad Nacional de La Plata, Argentina)

SUPERVISOR: Dr. E. DELLA TORRE

NUMBER OF PAGES: xiv, 120.

SCOPE AND CONTENTS:

The philosophy of parallel computation is discussed. A parallel computing system has been devised and a typical cell has been implemented. Supporting systems such as mnemonic language, parallel algorithms and computer programs to operate the array have been developed. The possibility of using the parallel processor as an associative-memory is also considered. A multi-cell system has been simulated on the PDP-11 computer.

ABSTRACT

A highly parallel computing system capable of computing transcendental functions, matrix operations and iterative calculations has been devised and a typical cell has been implemented. The system consists of an array of cells, a control unit, a PDP-11 computer and an interface unit. The array uses modified SOLOMON type of communication between cells. Each cell consists of 15 words and arithmetic hardware. Arithmetic and logic operations, on words or bytes, may be performed serially between pairs of these words. Division and floating-point arithmetic are under software control. Parallel algorithms have been developed. A set of instructions and mnemonics permits a practical use of the system. The possibility of using the array as an associative-memory processor is also considered. The system has been tested by using the software package prepared. Although only one cell has actually been constructed a complete array has been simulated on a PDP-11 computer.

ACKNOWLEDGEMENTS

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

Special thanks are also due to T. V. Srinivasan and H. Debruin for their invaluable assistance in writing this thesis.

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

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AC1	accumulator 1
AC2	accumulator 2
ADD	addition
ALO	arithmetic and logic operation
ALU	arithmetic - logic unit
AUT	automatic
AV	absolute value
BAR	basic address register
B/F	busy/free
CAR	carry
CAS	cell addressing sytem
CB	central buffer
CCC	computer-cell communication
CEI	cell enable input
CEO	cell enable output
CI	conditional inhibit
CIN	control in
CL	clock
CLR	clear
CMR	cell mode register
CNT	counter
CNV	convergence
COB	column buffer
COL	column
COM	complement
CONC	concatenation
COU	control out
CPG	control pulses generator
CPU	central processor unit
CTR	control transfer
CU	control unit
DEC	decrement

D.E.C.	Digital Equipment Corporation
DIN	data in
DMPX	demultiplexer
DOU	data out
DR	data register
DSR	device status register
E	east
ENB	enable
EX	exponential
EXO	exclusive or
EXT	external
FB	first byte
FF	flip-flop
FLT	floating
HPCS	highly parallel computing system
ICH	interchange
ID	instruction decoder
INC	increment
INF	integer, floating input
INH	inhibit
INP	input
INT	integer
IR	instruction register
LOR	logical or
LSB	least significant bit
LSI	large scale integration
MAI	move AC2 inverted
MOV	move
MPX	multiplexer
MPY	multiplication
MSB	most significant bit
MSG	move $\overline{\text{sign}}$ to AC2
N	north

NS	negative sign
NZ	no zero
OD	one detected
OUT	output
PR	preset
PS	positive sign
ROB	row buffer
ROT	rotate
RST	reset
S	south
SAV	sign/absolute value
SB	second byte
SEL	selector
SHF	shift
SHR	shift or rotate
SPC	special operation
SSR	shift serially to the right
SUB	subtraction
TCM	two's complement
TFB	transfer from buffer
TTB	transfer to buffer
TTL	transistor-transitor logic
UI	unconditional inhibit
W	west
WCC	without clear carry
WR	write
YZ	yes zero
ZD	zero detected

CHAPTER 1

Introduction

1.1 Background

Since the advent of the first commercial electronic computer, a great deal of improvement has been realized. The technological developments in semiconductors and magnetic materials have allowed an increase in speed, reliability and versatility of the digital computer, together with a reduction in physical size and cost.

On the other hand, architectural changes have been introduced in order to improve the computation speed. The efforts have been mainly directed to process information in a parallel fashion. The different degrees of parallelism that have been used are:

MULTIPLE BIT PROCESSING PARALLELISM

The multiple bit processing parallelism consists in considering each word as a unit of information so that all the bits of a word are processed simultaneously. The idea is used in most of the modern computers. Some of them use this feature only for addition, subtraction and logical operations. The more sophisticated machines use the same principle also in multiplication, division and even floating-point operations.

MULTIPLE FUNCTION PROCESSING PARALLELISM

The multiple function processing parallelism consists in transferring the information between the units under device and channel control,

while the CPU is processing data. The functional relationship among the units is optimized and a high degree of overlapping is obtained, especially between the I/O devices and the CPU. The "third-generation" computers make use of this feature.

MULTIPLE FUNCTION STREAM PROCESSING PARALLELISM

The multiple function stream processing parallelism consists in distributing a sequence of instructions over a sequence of "small computers" so that many subprograms can be simultaneously executed, as suggested by Holland [1]. Many "small computers" or modules, each with their own processor unit, a bank of memory and a control unit are interconnected among themselves forming an array. Because the complexity of efficient programming and prohibitive cost, no practical application of the Holland machine has yet been found.

HIGHLY PARALLEL COMPUTING SYSTEM (HPCS)

The HPCS makes use of single instruction stream which is executed simultaneously by many identical arithmetic-logic processors, each operating with a different set of parametric values of the same data type. It seems to be an approach to the solution for problems in which a very large set of data should be processed in a given time or for problems with inherent parallelism.

The concept of HPCS was suggested by Unger [2]. His machine has been conceived as a means of data manipulation where spatial configuration has significance (e.g., identification of edges, corners, curvatures and closed regions, solution of Karnaugh maps, etc.).

Lee and Paull [3] make use of a particular form of content-addressable memory. They developed a machine which is very useful whenever data must be identified by its contents rather than by its location in memory (e.g., to find identical patterns, to search for values between certain specified limits or to find an item closest to a given one, etc.).

The SOLOMON (Simultaneous Operation Linked Ordinal Modular Network) machine [4] is a general purpose parallel computer which has more practical applications than any of its predecessors. It can be used in weather forecasting, nuclear physics problems, large hydrodynamics problems, character recognition, optimization, and so forth.

Different versions of this computer have been constructed by Knapp [5], Litton [6] and Della Torre and Ho [7]. The most sophisticated version, the Illiac IV [8] is now under advanced construction. The machine proposed in this thesis is also a modified version of the SOLOMON computer.

Many other parallel machines have been suggested and/or constructed but a sharp classification cannot be made. For example, the Berkeley Array Processor [9] is a special purpose computer designed to perform the operations of correlation, convolution, recursive filtering, matrix manipulations, etc. The modified Holland machine [10] is an improvement over the basic Holland machine while the Davies-Associative Processor [11] and the Goodyear's Associative Processor [12] can be considered a further step on the Lee and Paull machine [3].

1.2 Philosophy of the Proposed System

In an attempt to build a highly parallel computing system, it has been found that the SOLOMON structure is a good starting point because:

a) it is a general purpose computer;

b) the development of the integrated semiconductor electronics in the past few years may lead to the construction of a large number of identical cells at low cost, using the LSI techniques; and the cylindrical magnetic domains (bubbles) in certain uniaxial magnetic materials [13], seem to have natural applications in performing memory and logic functions for large quantities of data, at very low cost and physical size;

c) most of the large scale computing jobs are generated from the repetitive execution of the same algorithm over and over on different pieces of data;

d) only one control unit is needed, regardless of the number of cells used, therefore the computing power is increased with a comparatively small increase in hardware;

e) associative-memory capabilities can be easily introduced.

The range of the degree of complexity of the cells in the different versions of SOLOMON computer is very wide. It varies from the relatively simple Della Torre and Ho's cell[7], with 3, 12-bit memory words and single-bit processing, to the sophisticated Illiac IV [8]

with 2048, 64-bit memory words and a very elaborate multiple-bit arithmetic unit. The greater the degree of complexity of the cells, the fewer of them that can be built for a given price. A compromise has been made in this thesis.

As a base for the design, the following specifications must be satisfied.

a) The system should be compatible with existing minicomputers, particularly with the PDP-11 available.

b) The structure of the array must allow easy handling of matrices.

c) Each cell should be the simplest possible but complex enough for performing any arithmetic or logic operation. It should be able to perform floating-point arithmetic and compute transcendental functions under software control.

d) Each cell should have some degree of autonomy in the way that they can accept of reject the common instruction, according to local tests.

e) Each cell must be able to communicate to the control unit that some degree of "convergence" between certain particular data under processing has been obtained.

f) The system should have some content-addressable memory capabilities.

g) The control unit and the instruction set should allow a practical use of the array.

1.3 Outline of the Thesis

The aim of this thesis is to present a new HPCS suitable for solving problems in which a matrix or mesh of numerical values as well as information retrieval are involved. Such operations are encountered in communication, character recognition, hydrodynamics, heat flow, optimization, weather forecasting and air traffic problems.

The general organization of the system as well as the array architecture are presented in Chapter 2.

Chapter 3 deals with the philosophy of the cell design and its configuration. It includes a discussion of the data representation systems used, the techniques for doing arithmetic operations and the conditional branch capabilities. The Chapter ends with a section in which a description of the cell is made.

The operation of the cell, from the operands selection point of view is described in Chapter 4. The instruction set and a mnemonic language are also proposed in this Chapter.

In Chapter 5, the implementation of the Control Unit is discussed. Design considerations of the individual subunits (instruction decoder, control pulses generator and cell addressing system) are treated.

The interface between the computer and the rest of the system is described in Chapter 6.

The software used in this project is described in Chapter 7, where several algorithms are proposed. A program to control the cell and the microprogramming of the algorithms are discussed. Finally, a

routine which allows the PDP-11 to simulate an array of cells is also presented in this Chapter.

The possibility of using the array as an associative memory processor is discussed in Chapter 8. A technique for carrying out the comparison and the detection of the cell addresses is proposed.

Chapter 9 summarizes the contents of the preceding Chapters and suggests future improvement of the system proposed in this work.

CHAPTER 2

General Organization

2.1 System Organization

The general organization of the system is shown in Fig. 2.1. It consists of four main parts:

1. A PDP-11 COMPUTER

   The PDP-11 computer is assigned the following functions.

   a) Storage of data and instructions.

   b) Executive control of the execution of array programs.

   c) External I/O processing and supervision.

   d) Compilation of programs.

2. AN INTERFACE UNIT

   The interface unit makes possible the communication between the PDP-11 and the rest of the system by

   a) solving some compatibility problems, and

   b) acting as a temporary storage of information.

3. A CONTROL UNIT (CU)

   The CU has the following functions.

   a) To receive instructions from the computer, to decode them and to generate enable signals which are broadcast to all the cells in the array.

   b) To generate the control pulses transmitted to the cells for

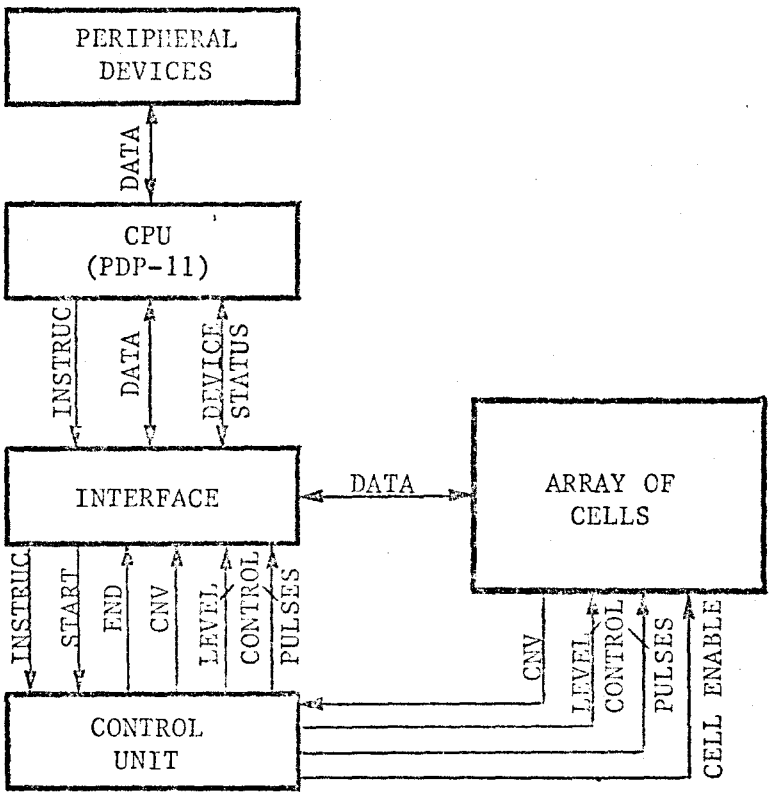


FIGURE 2.1. SYSTEM ORGANIZATION

instruction execution.

c) To receive and compare convergence information from the array of cells.

d) To address the particular cell (or cells) in which transfer of information as well as arithmetic and logic operations (ALO) are enabled.

e) To notify the computer (by setting a flag) about the convergence condition of the system.

f) To notify the computer (by setting a flag) that the execution of the instruction has finished.

4. THE ARRAY OF CELLS

The array consists of a set of cells interconnected among themselves by data interchange lines. Every cell can store and process information. The execution of any arithmetic or logic operation in the enabled cells depends upon:

a) the level of the control lines during the execution of the operation,

b) the presence or absence of appropriate pulses in the control pulses lines in the precise moment, and

c) the state of the control bits of the cell.

Whereas the control bits depend on the history of that particular cell, the other levels and pulses are generated in the CU and applied to all the cells simultaneously.

The cells are enabled by the cell address lines. Information about the convergence condition of every cell is sent to the CU where

the inequality signals are OR-ed, giving a convergence state for the whole array.

2.2 Array Architecture

2.2.1 Geometrical configuration

Similar to the SOLOMON organization, from the data transference point of view, the cells are interconnected forming a basic two dimensional rectangular matrix in which each cell is an element of it. Each cell is connected by serial data busses to its four nearest neighbors: the cell immediately to the north, N; south, S; east, E; and west, W.

Different neighbourhood relationships can be assumed for the "edge" elements of the basic two dimensional rectangular matrix, and the array can be geometrically

a) a planar rectangular matrix, assuming no N neighbors for the first row; no S neighbors for the last one; no E neighbors for the last column; and no W neighbors for the first one.

b) a horizontal cylinder, considering the elements of the first and last rows as adjacent.

c) a vertical cylinder, considering the elements of the first and last columns as adjacent.

d) a torus, assuming that conditions b) and c) are verified simultaneously.

The configuration a) has been selected for this thesis. However, the user can change from one configuration to another by simple wiring of the "edge" elements. It would be possible to incorporate in the system a circuit which allows changes of geometrical configuration under program control.

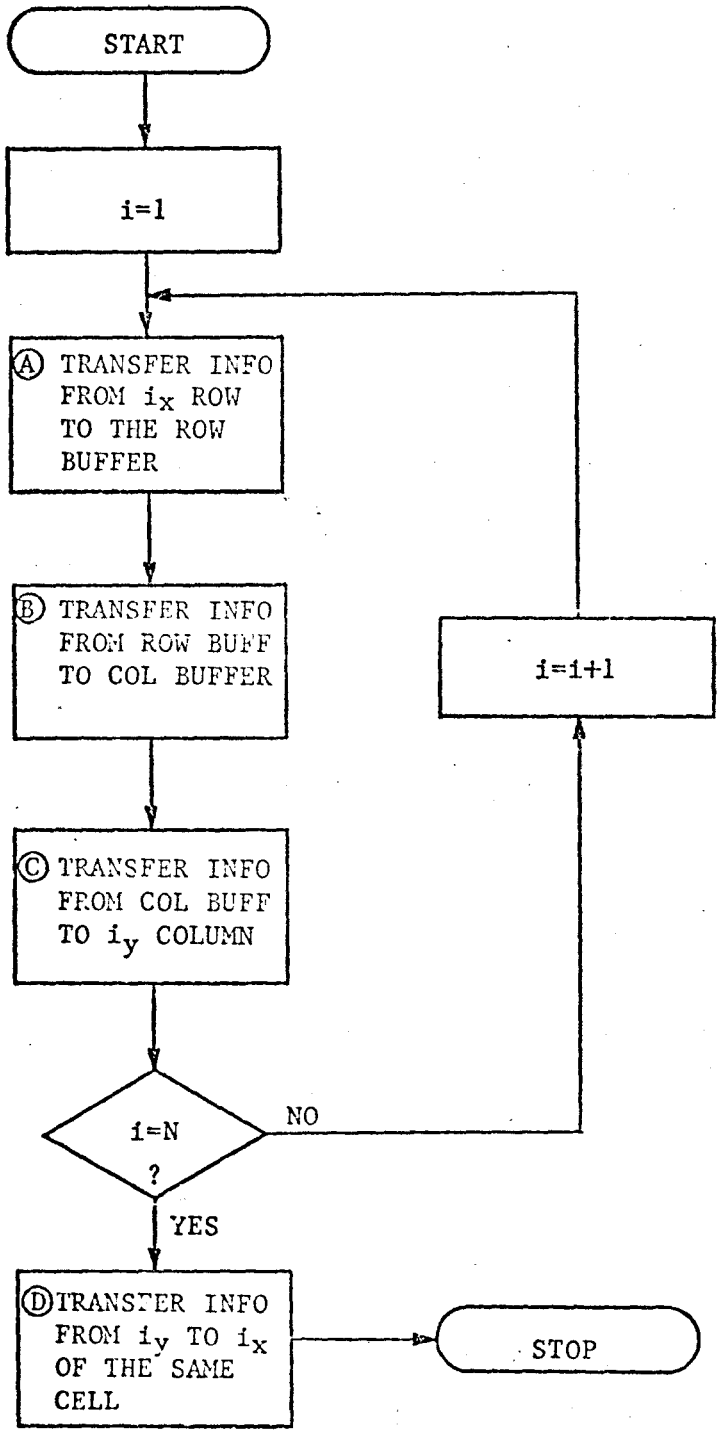
2.2.2 Row and Column Buffers

The mathematical operations in which matrix manipulations are involved can be greatly simplified by adding a row and a column buffer to the basic geometrical configuration.

The buffers are cells like the other ones, but with different neighbourhood interconnections. Each cell of the row (column) buffer has bidirectional communication with the corresponding cells in all the rows (columns). So the buffers become the fifth and sixth neighbors of each cell.

Suppose we wish to transpose an nxn matrix, where every element of it is the word x of a cell of the array. The possible steps are illustrated in the flowchart of Fig. 2.2. Step B could be skipped if a unique cell can simultaneously be an element of the row and the column buffers.

Transfer of data between the matrix of cells and both buffers is never performed simultaneously. So, by using a common row and column buffer, a higher degree of parallelism efficiency can be obtained.



Nb. i_x stands for word x of i
i_y stands for word y of i

FIGURE 2.2. TRANSPOSITION OF AN n x n MATRIX

2.2.3 Central Buffer

There are many calculations in which all the cells use a common operand. Memory space can be saved if these operands are stored in a central buffer (CB), instead of being stored in each cell. The CB is a register located in the interface unit and it is loaded directly from the CPU. It has connection with all the cells of the array and it can be considered the cell's seventh neighbor.

In conclusion, each cell can operate with:

a) its own data;

b) the four nearest neighbors data;

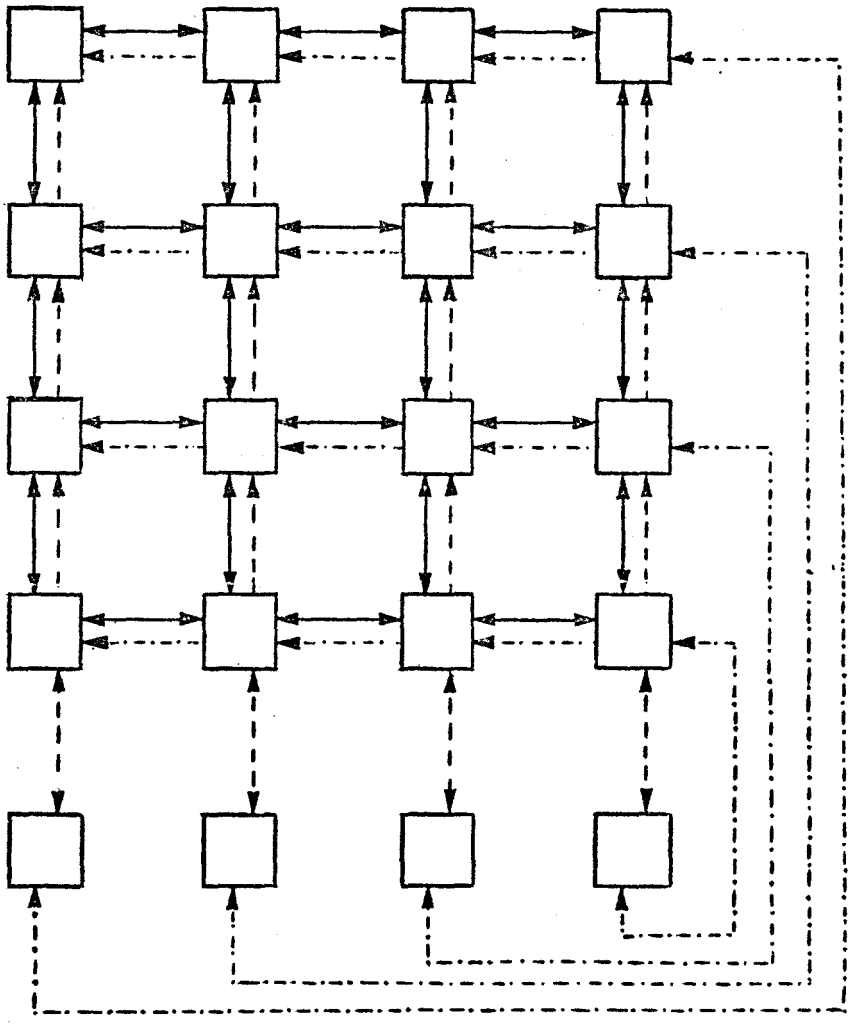
c) the corresponding row or column buffer data; and

d) the central buffer data.

The cells data communication is illustrated in Fig. 2.3.

2.3 Control Bus

The control bus is a unidirectional bus carrying the control levels and pulses from the CU to all the cells. The CU sends a common instruction to every cell of the array. However, the cell addressing system allows certain degree of exclusiveness in a particular cell or set of them. Each cell operates with its own data and in accordance with the required inhibit conditions.



——— CELLS COMMUNICATION

-·-·-·- COLUMN BUFFER - CELLS COMMUNICATION

------ ROW BUFFER - CELLS COMMUNICATION

Nb. The connection of all the cells to a CB is not shown.

FIGURE 2.3. CELLS DATA COMMUNICATION

CHAPTER 3

The Cell

3.1 Philosophy of the Design

In order to have a practical array several cells should be built. Because of cost limitations, it will be possible to do so only by using LSI technology, but this problem is out of the scope of this thesis. The cell discussed here is intended to be a feasibility study rather than a commercial unit so small and medium scale integration technology is used.

The very first point in the design of the cell should concern its complexity and size. Multiple bit parallelism and floating-point arithmetic hardware are the cornerstones for any high-speed scientific computer but this hardware is very expensive. In this feasibility study and because of cost limitations, it has been decided to use serial bit arithmetic and to do division and floating-point arithmetic under software control.

A study of the algorithms necessary to calculate transcendental functions in both integer and floating-point modes has been done. It was found that a cell with 15 words memory bank is complex enough to compute these functions as is shown in Chapter 7. A 16-bit word memory size has been chosen for this project because it is compatible with the PDP-11 computer used as well as it is easy to get in the TTL memory market.

The arithmetic-logic unit (ALU) should be able to perform the basic arithmetic and logic operations, in serial fashion, within the contents of the memories of the cell itself or the neighbors.

Several selectors should be able to select, from the memory bank, the corresponding source and destination word.

In addition to the ALU, the memories and their associated selectors, other devices such as accumulators and control bits have been incorporated in the cell and their use is discussed in the following paragraphs.

3.2 Representation of Data

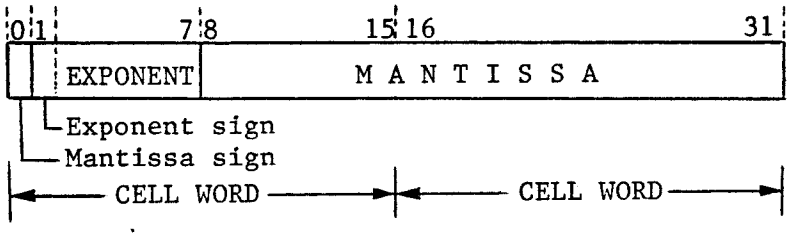
The cell is essentially a 16-bit word processor. Each word represents a logical variable or an integer.

Three different binary data representations are used in electronic digital computers: 1's complement, 2's complement (TCM) and sign/absolute value (SAV). The 2's complement data representation was chosen for this cell, because its arithmetic requires less hardware whenever addition or subtraction is performed [14].

Sophisticated and expensive circuits would be required to perform hardware division. So the fourth arithmetic operation is done under software control. Because the SAV representation makes the division easier, the system provides facilities to change from one data representation to another.

Using 16 bits, TCM representation, the range of an integer is from -32,768 to +32,767. This is a very restricted interval. In order to allow the user more flexibility the cell must have the capability of

performing floating-point operations, under software control. Two 16-bit words are used to represent a floating-point number X, according to the following scheme:



$$X = .\text{MANTISSA} \times 2^{\text{EXPONENT}}$$

The mantissa is in SAV representation. After a non-zero mantissa has been normalized, its absolute value is a fraction in the range

$$\tfrac{1}{2} \leq |\text{MANTISSA}| < 1$$

The exponent of base 2 is in TCM representation. It is any integer in the range

$$-64 \leq \text{EXPONENT} \leq 63$$

From the preceding considerations, it is seen that a cell word can represent the following types of variables:

a string of 16 logical characters.	LOGICAL
a two's complement integer.	S INTEGER
a sign/absolute value integer.	S INTEGER
a part of the AV of the mantissa.	MANTISSA
the sign and part of the AV of the mantissa and TCM exponent.	S S EXPONENT MANTISSA

A set of instructions, a control unit and the cell itself have been designed in order to be able to manipulate any of these data representations.

3.3 Integer Arithmetic

3.3.1 Serial Operations

The cell has been structured around the ALU. Because no more than two operands are involved in any ALO, the ALU has two data inputs, X and Y; and one output, Z. It can perform the following operations: addition, subtraction, logical and, logical or, exclusive or, complement, two's complement, increment, decrement and move (interregister transfer).

The cell operates in serial fashion, so one operation cycle of 16 bit cycles is necessary to perform any of these integer operations, one bit at a time. "Shift to the right" is equivalent, in binary representation, to divide by 2. It is serially performed by 'adding' a number to itself, but with an operation cycle running from the most significant bit (MSB) to the least significant bit (LSB).

For any of these operations, the data should be serially accessed from the memory bank. Sixteen-bit TTL memories with direct address, non-destructive read-out and high speed characteristics have been chosen. They satisfy the design requirements.

3.3.2 Multiplication

The "add/shift/inhibit" technique is used. It is a modification of the classic "add/shift" technique. For the operation AxB=C the

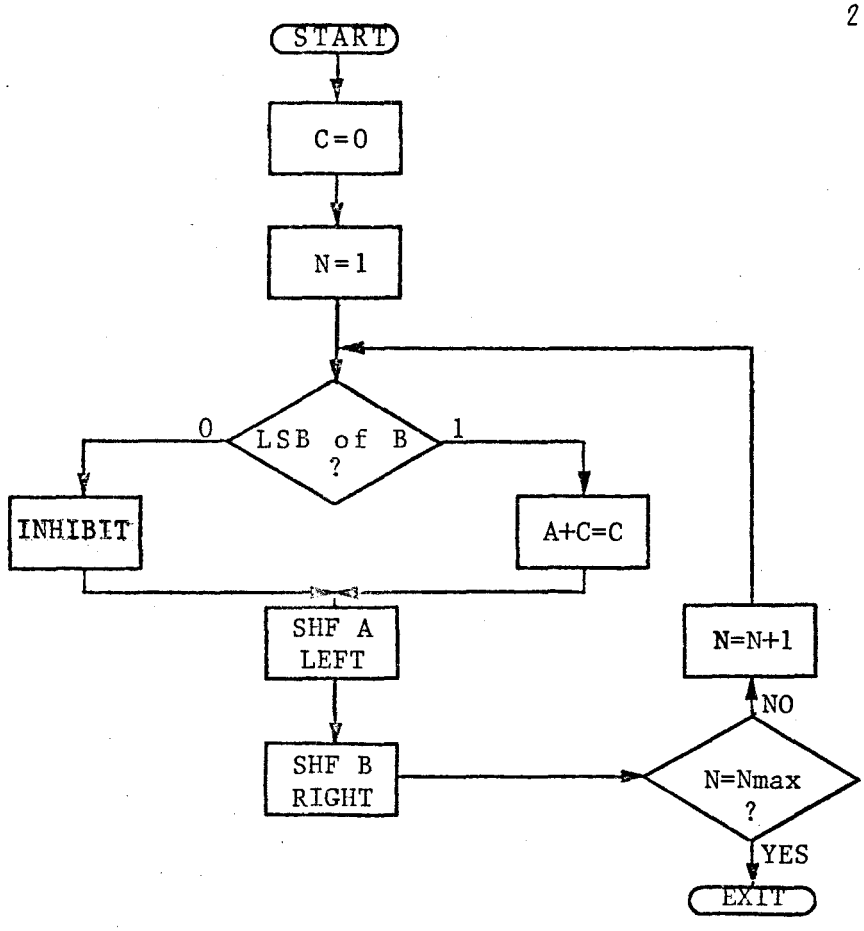


FIGURE 3.1. MULTIPLICATION. "ADD/SHIFT" TECHNIQUE

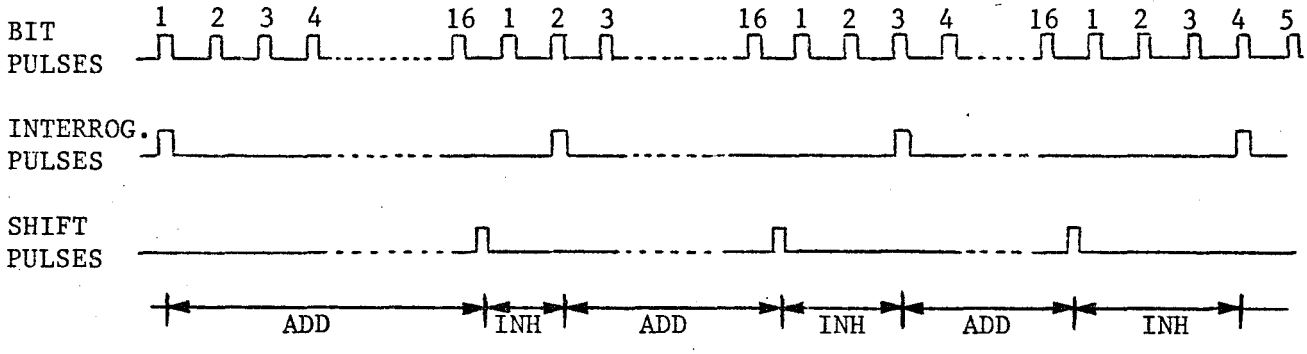


FIGURE 3.2. MULTIPLICATION. TRAIN OF PULSES

"add/shift" technique consists of the steps shown in the flowchart of Fig. 3.1. When the "Nth" less significant bit (LSB) of B is being interrogated "N-1" left shifting in A have already been performed. Therefore, the "N-1" LSB's of A are equal to zero and no change is produced in the "N-1" LSB's of C during the add cycle, A+C=C.

The "add/shift/inhibit" technique consists in inhibiting the part of the add cycle in which no change occurs in C and to start the add cycle in bit N, after the multiplier B interrogation has been performed. The necessary train of pulses is shown in Fig. 3.2. If parallel shifting can be performed, this algorithm does multiplication very efficiently, because addition, shifting and multiplier interrogation can be performed in one operation cycle. The size of the answer cannot exceed 16 bits, so a multiplier and a multiplicand of 8 bits each are assumed. Therefore, the complete multiplication cycle in integer mode consists of 8 operation cycles.

The operation "shift B right", shown in the flowchart is done by shifting the bit of B being interrogated each cycle. In order to perform parallel shifting of A, the multiplicand must be in a 16-bit shift left register. It differs from the others direct-address words and is called accumulator 1 (AC1).

3.3.3 Division

The technique used consists of four parts:

1. Conversion of the negative numbers from TCM representation to SAV representation.
2. Normalization of the divisor.

3. Restoring division.

4. Conversion of the answer from SAV representation to TCM representation.

The logic used is shown in the flowchart of Fig. 3.3. In the first step, the numbers (except the MSB) must be 2's complemented if they are negative; otherwise they should remain unaltered. A "sign detector" in ACl is incorporated for this purpose. Hardware for manipulating only 15 of the 16 bits is necessary in order not to change the MSB.

In the normalization part, the divisor must be left shifted until a "1" reaches the MSB. Again, ACl is utilized, adding a "1 detector" in the MSB.

In the division itself, the restoring method is utilized. A cycle is defined as the processing required to generate a single bit of the quotient. For dividing $M_A \div M_B = M_C$ the technique consists of subtracting $M_A - M_B$. If the answer is positive, the subtraction is successful. The answer is the new value of M_A for further cycles and a "1" is obtained as the next MSB of the quotient. On the other hand, if the subtraction is unsucessful ($M_A - M_B < 0$), M_A is not changed and a "0" is obtained as the next MSB of the quotient. Finally, M_B is shifted to the right one bit and a new cycle begins.

The number of cycles should be equal to the number of shifts during the normalization part. In order to do so one of the registers acts as a counter. After being reset to "0" it is increased by "1" every shifting performed. Then a "1" is subtracted from the register every division cycle until a "0" is detected. A "zero detector" in ACl will do the job.

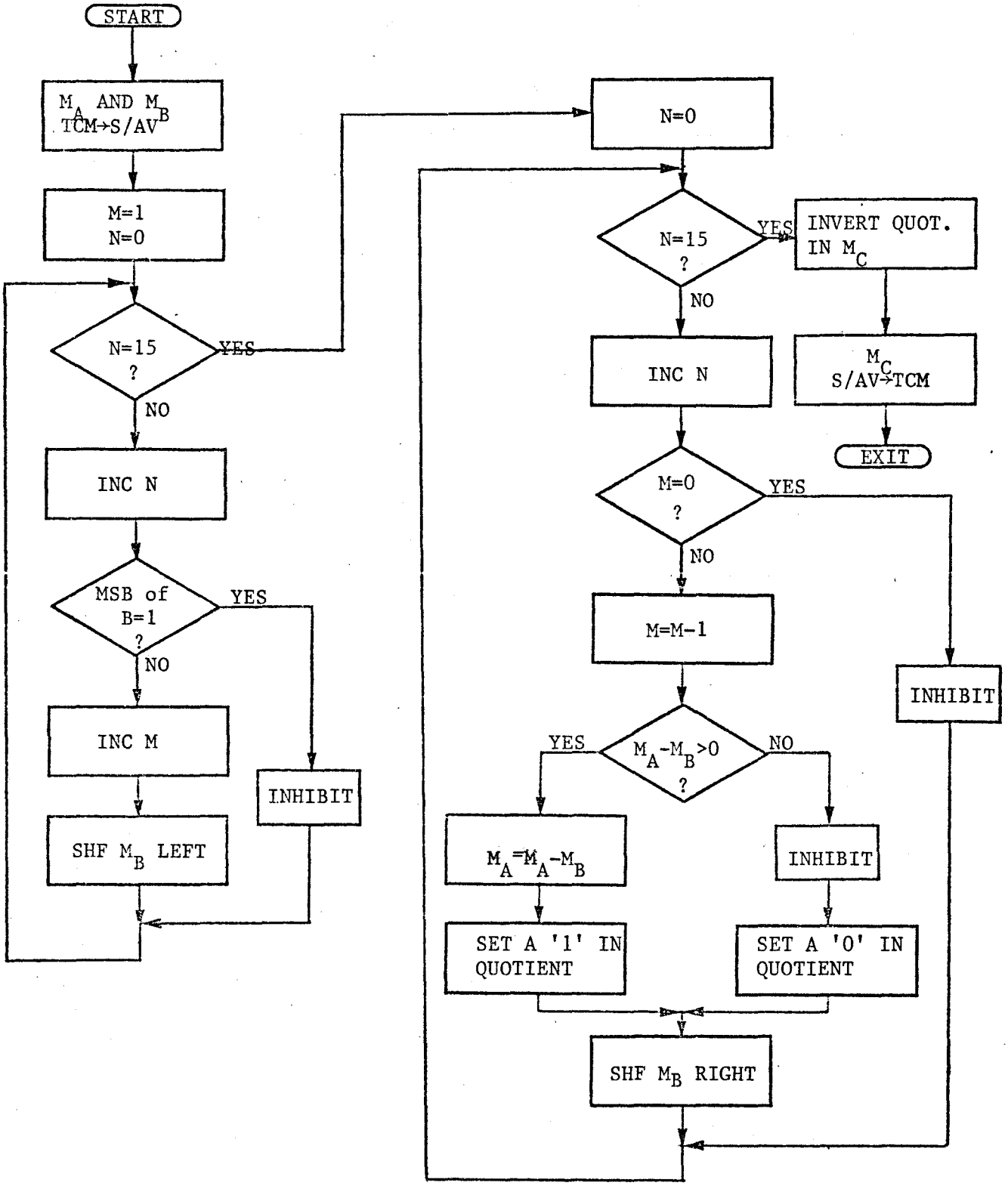


FIGURE 3.3. INTEGER DIVISION FLOWCHART
($M_A \div M_B = M_C$)

In addition, a new 16-bit shift register is necessary in order to store the quotient, which is obtained, one bit per cycle from the "sign detector". It is called accumulator 2 (AC2). A "move $\overline{\text{sign}}$ of AC1 to AC2" logic will transfer the information available at the "sign detector" to the AC2.

Unlike the other operations, in division the answer is obtained starting from the MSB and its absolute value (AV) is shown up inverted in AC2. An "invert AC2" logic will invert the AV of AC2 and will store it as an AV in the 15 LSB's of a specified register. If the EXO of the dividend and the divisor had been previously stored in this register, its MSB would contain the sign of the answer and, after the inversion of AC2, the content of this register would be the SAV of the answer.

The last step in the division consists in the conversion of the answer from SAV to TCM representation.

Many of the operations discussed so far make use of the special capabilities of AC1. For that reason, the user may wish to transfer the contents of any word of memory to AC1. Using the "move" operation, the content of the accumulator would be lost unless it is previously saved in another word. In order to save extra memory and extra number of steps, an "interchange AC1" logic is included. It interchanges the contents of any memory word with the contents of AC1.

3.4 Floating-Point Arithmetic

Before any floating-point operation, the number must be normalized. This is accomplished by transferring the 24-bit mantissa to a

24-bit shift-left accumulator and shifting it until a "1" is detected in the MSB. Every left shifting must be accompanied by a subtraction of "1" in the exponent. On the other hand, every right shifting must be accompanied by an addition of "1" in the exponent. Otherwise, the number would change.

Multiplication between two numbers is performed by multiplying the mantissas and adding the exponents. The sign of the answer is the exclusive-or of the two mantissa signs.

Addition is carried out by shifting the mantissa of the number with smaller exponent to the right a number of times so that the exponents of both addend are equal. The addition of the mantissas is the mantissa of the answer and the common exponent is the exponent of the answer. Because the mantissas are represented in SAV, a conversion to TCM is necessary before the addition.

Division is performed by subtracting the exponents, dividing the mantissas and EXO-ing the signs.

All the floating-point operations discussed can be done, under software control, extending both accumulators, with the same "zero", "sign" and "1" detectors capabilities to a 24-bit register. Each accumulator has available two inputs, "integer" (INT) and "floating" (FLT) which allow incoming data to use the accumulator as a 16-bit or 24-bit register. Aside from this, it is necessary to have a hardware capable of operating in different modes, as follows:

In multiplication:

a) INTEGER MULTIPLICATION, where bits 0-8 are interrogated, and the "inhibit/add" cycles last 16 bit cycles.

b) FLOATING MULTIPLICATION, where bits 0-8 are interrogated, and the "inhibit/add" cycles last 24 bit cycles.

In the others ALO:

a) INTEGER, where bits 0-15 are processed and the INT input is used.

b) FIRST BYTE, where bits 0-7 are processed and the FLT input is used.

c) SECOND BYTE, where bits 8-15 are processed and the FLT input is used.

d) EXPONENTIAL, where bits 8-14 are normally processed, bit 15 is EXO-ed and the FLT input is used.

e) ABSOLUTE VALUE, where bits 0-14 are processed and the INT input is used.

f) INTEGER FLOATING INPUT, where bits 0-15 are processed and the FLT input is used.

When an overflow occurs in any arithmetic operation, the carry flip-flop is set on. Normally it is cleared before any new operation starts, but when two words operation is required it should be used as a link between both. This is the case of the mantissas in floating-point representation. Hence, the capability to operate without clear the carry has also been incorporated to the hardware. This feature also allows the cell to perform double-precision arithmetic under software control.

3.5 Conditional Branch

In a sequential computer, there is an interaction between the data and the control unit. The conditional branch instructions allow

the user to alter program flow according to data tests. In a HPCS a direct interaction is impossible because different sets of data are operating with the same instruction stream.

The problem has been solved by using the "inhibit" and "convergence" system.

Inhibit is the property of each cell to enable or disable, according to local tests, local execution of a command. Conditional inhibit permits inhibit of the operation according to data tests (e.g., inhibit if AC1 = 0). Unconditional inhibit permits ask for the inhibition of a cell for its position in the array, independently of its data content (e.g., inhibit boundary cells in solving Laplace's equation).

Convergence is a property of the array. Each cell compares new data being introduced with the content of a particular word. If the comparison gives equality in a given range in all the cells simultaneously, a "flag" is set. The CPU can be microprogrammed to interrogate the "flag", allowing the array, in iterative process to self-determine the number of passes until a desired convergence is reached in all the cells.

3.6 Cell Description

The cell, shown in Fig. 3.4, contains the following elements:

Two 24-bit accumulators.

Thirteen 16-bit direct-address memories.

A 1-to-16 lines demultiplexer.

Two 16-to-1 line multiplexers.

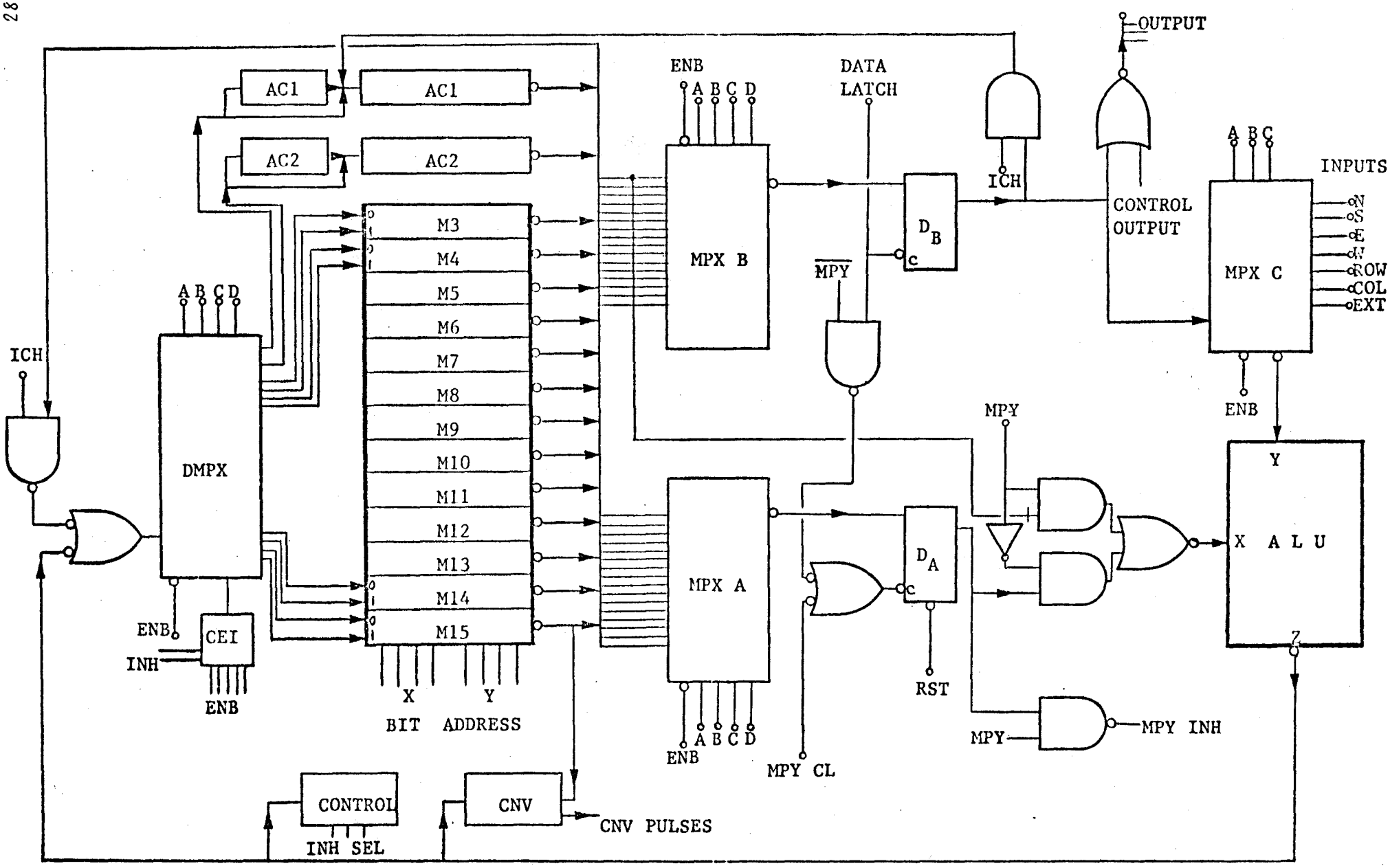


FIGURE 3.4. THE CELL

An 8-to-1 line multiplexer

An arithmetic and logic unit.

Two data latch flip-flops.

Four control flip-flops.

A 4-to-1 line inhibit multiplexer.

Several gates.

The characteristics and functions of these elements are described in the following sections.

The cell can perform the operations listed in Table 4.1. Addition, subtraction, and, exclusive-or, logical or, complement, two's complement, increment, decrement, move, shift serially to the right, invert AC2 and control transfer are performed in serial fashion, bit-by-bit, through the ALU. Interchange is also done serially, but without intervention of the ALU. Parallel shifting is performed in parallel, all the bits at a time, in the accumulators, while multiplication is carried out by combining both, serial and parallel operations. Computer-cell communication as well as buffer-array communication are performed serially, but only in the cells that have been selected by the cell addressing system. Finally, move $\overline{\text{sign}}$ to AC2 is done by using the AC2 and a few gates.

3.6.1 Accumulators

Two accumulators, AC1 and AC2, have been designed. Their size, as well as their shifting and testing capabilities are in accordance to previous discussions. They can also be used like any of the other

13 words whenever the special accumulators capabilities are not required.

Each accumulator consists of a 24-bit shift register although only 16 bits are enabled when the FLT input is not used, as is shown in Figs. 3.5 and 3.6. The "writing" is done serially through the left side (write 1 line) and the "reading" is performed serially through the right side (read line). The ROT line controls the flow of information into the input. If ROT is high, the output is fed-back to the input and the information is recalled in non-destructive mode. This allows the accumulators to be used for reading without losing their contents, like in the other 13 words. This also enables the accumulators to perform circular shifting, where the empty places produced by the shifting are replaced by the bits that fall off the end. On the other hand, when $\overline{\text{ROT}}$ is activated new information can be written in the accumulators whenever one of them has been chosen by the de-multiplexer (DMPX) for storing the answer of an operation or new data. Also, logical shifting, where the empty places are replaced by 0's are performed with $\overline{\text{ROT}}$ control high. In that case, SHIFT (SHF) line should be held high to make sure that the empty places are filled with 0's.

The accumulator 1 can be left or right shifted, according to the level of the SHR DIRECTION line. Clock (CL) AC1 is activated every time AC1 is selected for operations (read, write, shift) but it cannot reach the accumulator clock whenever the cell is inhibited. Information from the output line is sent directly to the 16 LSB's of AC1, whenever interchange (ICH) line is held high.

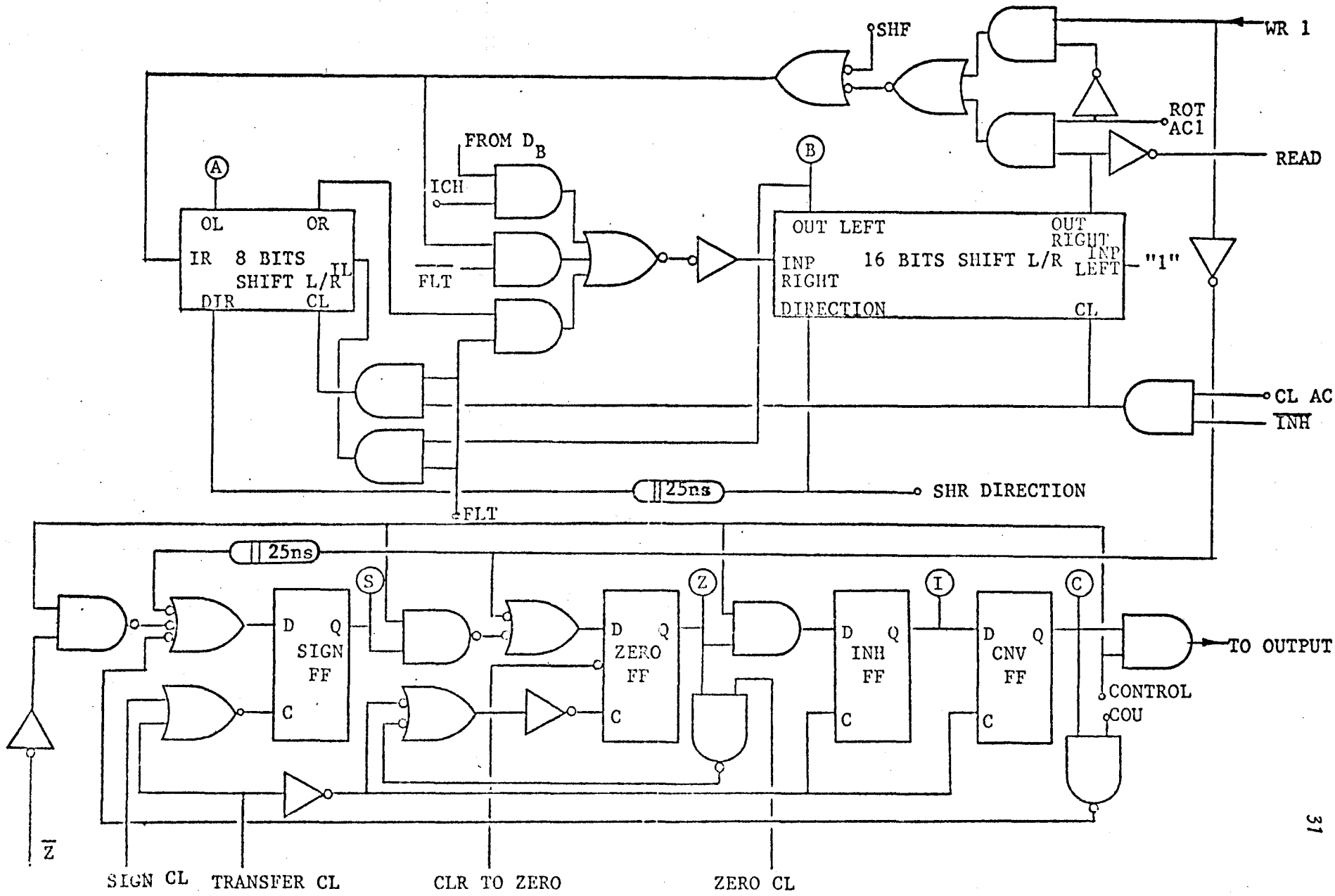


FIGURE 3.5. ACCUMULATOR 1 AND CONTROL BITS

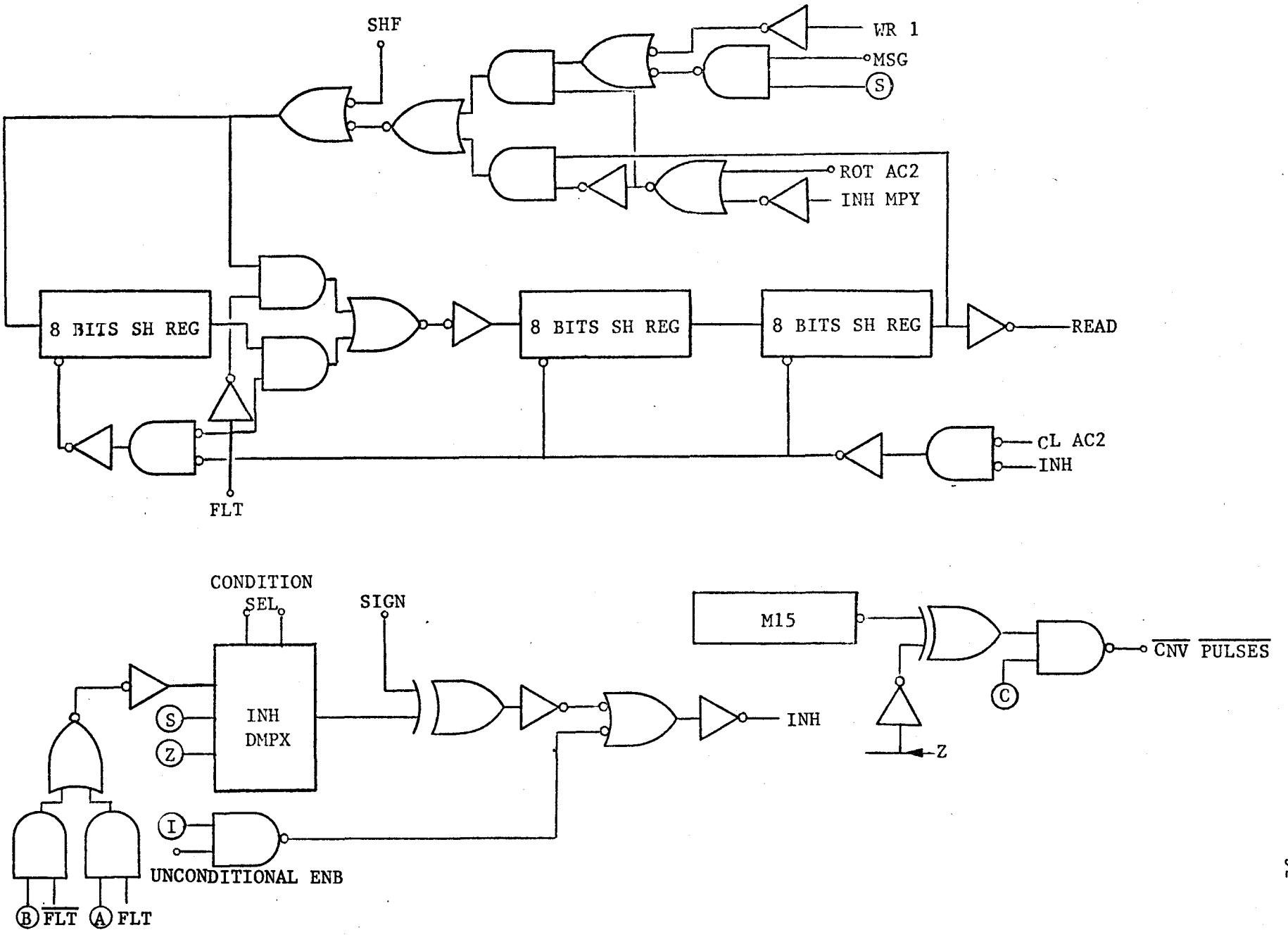


FIGURE 3.6. ACCUMULATOR 2. INHIBIT SYSTEM. CONVERGENCE SYSTEM

The accumulator 2 can only be shifted to the right. No applications for both directions shifting have been found and therefore considerable amounts of hardware can be saved. It works with an independent clock which is activated whenever AC2 is selected for operations, but it cannot clock it when the cell is inhibited. The value of the $\overline{\text{sign}}$ of AC1 is written in AC2 using the control line move $\overline{\text{sign}}$ to AC2 (MSG).

3.6.2 Direct Address Memories

Sixteen-bit active-element memories arranged in a 4 by 4 matrix are used. Four X and four Y lines permit the addressing of one bit at a time. The memories have non-destructive read-out. "Write 0" and "Write 1" are independent inputs which are accessible when the word has been selected for writing. Information can be written or read in any bit, changing the bit address. The memories cannot be used to provide information of the state of a bit while writing in the same bit is performed.

3.6.3 Demultiplexer (DMPX)

The DMPX selects the word in which the information is being written. The decoding function is performed by using 4 control lines to address the 16 output lines, as is shown in Fig. 3.7. The "write 0" or "write 1" pulses are sent to the selected word through any of the two AND gates that the decoder output lines maintain available. If AC1 or AC2 are selected, only the "write 1" line is used because the absence of write pulses during a bit cycle is equivalent to write

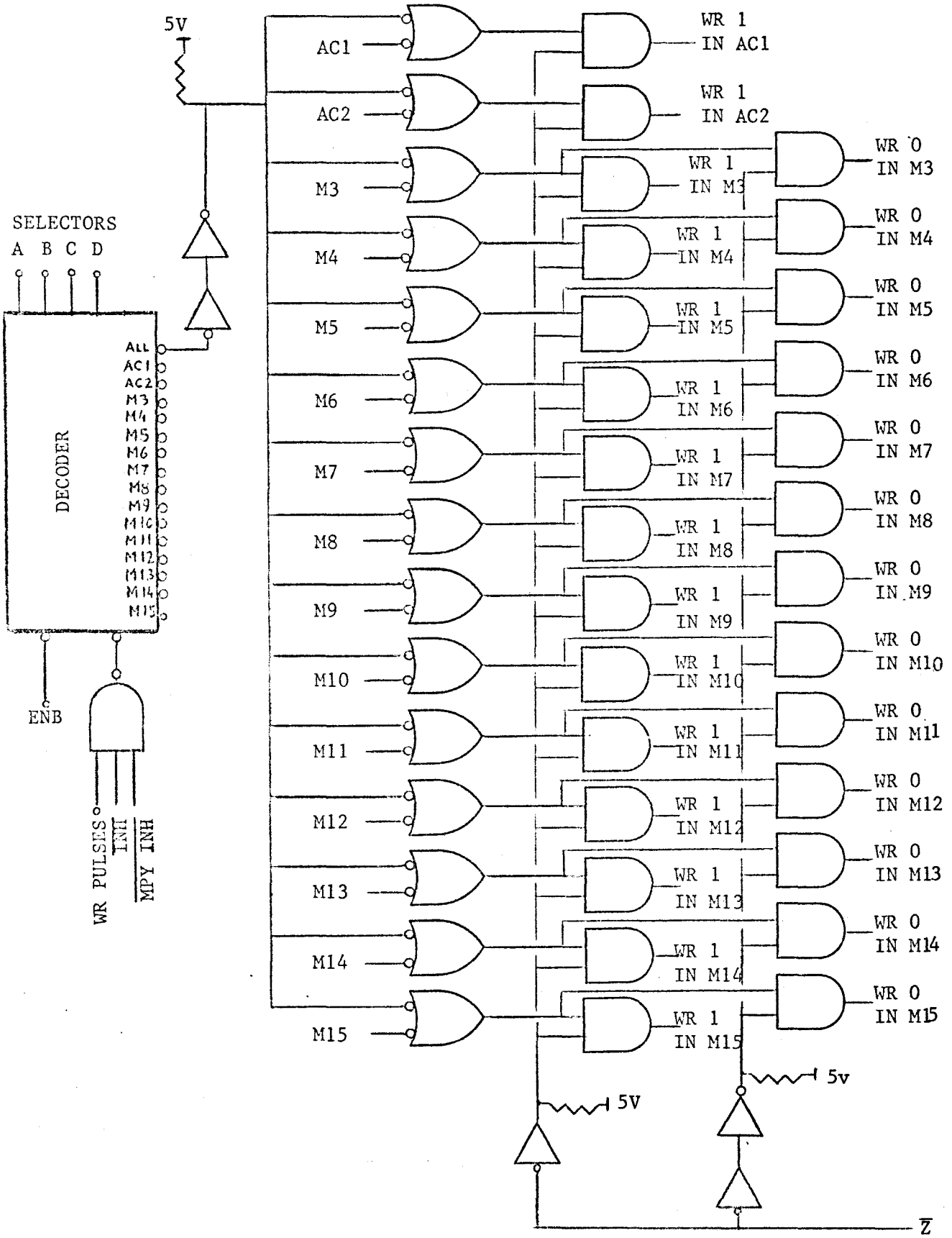


FIGURE 3.7. DEMULTIPLEXER

a "0" in the accumulators. When the output 0 is selected by the decoder, the information is sent to all the words. This can be particularly useful in clearing the cell.

If any of the inhibit condition occurs, the DMPX produces no output and no change of the contents of the memories are possible. It is also inhibited, using the D ENB line, in operations in which the DMPX is not involved.

3.6.4 Multiplexers A and B (MPX A, MPX B)

The multiplexers select one of the 15 words that are sent to the data latch flip-flops, bit-by-bit (Fig. 3.4). A selection of "0" means that none of the words are selected. For some operations in which the multiplexers are not involved, their operations are inhibited by using the corresponding ENB lines.

3.6.5 Multiplexer C (MPX C)

The MPX C is the input selector. Three control lines select the information available in D_B of the cell itself or any of its four neighbours (N, S, E or W), the ROW or COL buffers, or the central buffer. The output of this MPX is directed to the Y input of the ALU. In operations in which MPX C is not involved, its output is inhibited by the C ENB line.

3.6.6 Data Latch Flip-Flops (D_A and D_B)

The data latch flip-flops are used as a temporary storage of information. It is possible to take information from a word and write

the answer in another word almost simultaneously. But the characteristics of the memories do not allow one to write new information in the same word which is simultaneously being read. The problem is solved by dividing the bit cycle into a read and a write subcycle. During the read subcycle, the information available at the outputs of the MPX A and MPX B are latched by the "data latch" clock in the corresponding flip-flops. While the write subcycle is performed the data is taken from the data latch flip-flops rather than from the memories themselves and new data can be entered into the selected memory word.

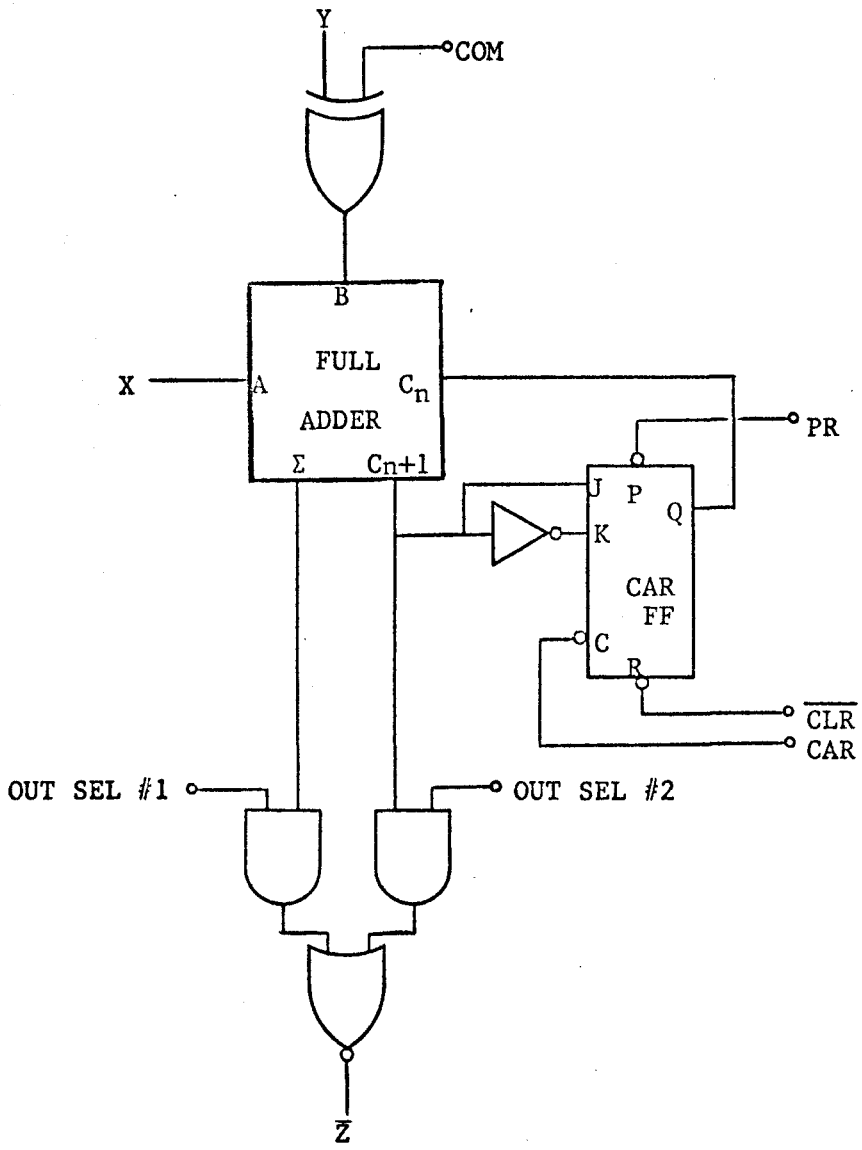
The information available in D_B can be sent to the Y input of the ALU of the same cell, its four neighbors, the row or column buffers or to the CB. However D_A output is only sent to the X input of the ALU or to the MPY INH line of the same cell.

When MPY is performed, D_A holds the multiplier during the add cycle. When the cycle finishes, reset D_A (RST D_A) clock is activated and the flip-flop (FF) remains in "zero" during the inhibit cycle, until a new interrogation is performed by the MPY clock. The output of D_A controls the MPY INH line level.

3.6.7 Arithmetic-Logic Unit (ALU)

The ALU is the heart of the cell because all the serial arithmetic and logic operations (ALO) are performed there. The operations are carried out in serial fashion, one bit at a time.

The ALU, as is shown in Fig. 3.8, is composed of:



		ADD	SUB	AND	EXO	LOR	COM	TCM	INC	DEC	MOV
LEVEL CONTROL	COMPLEMENT	0	1	0	0	0	1	1	0	1	0
	OUT SEL #1	1	1	0	1	1	1	1	1	1	1
	OUT SEL #2	0	0	1	0	1	0	0	0	0	0
CARRY PULSES	PRESET	0	1	0	0	0	0	1	1	0	0
	CLEAR	1	1	1	1	1	1	1	1	1	1
	CLOCK	1	1	0	0	0	0	1	1	1	0
INPUTS	X	1	1	1	1	1	0	0	0	1	0
	Y	1	1	1	1	1	1	1	1	0	1

FIGURE 3.8. THE ARITHMETIC AND LOGIC UNIT

a) A 1-bit binary full adder, that performs the addition of the three inputs; A, B and carry in (C_n). Two outputs are available: sum ($\sum$) and carry out (C_{n+1}).

b) An exclusive-or gate that transfer the Y input of the ALU to the B input of the adder in "true" or "complement" form, according to the level of the COM line.

c) A carry flip-flop, controlled by the carry clock (CAR CL), that holds the output C_{n+1} in order to be used as the input C_n in the next bit cycle. It can be preset and cleared, according to the desired operation.

d) A 2-to-1 output selector, that selects the adder output to be transferred to the Z output of the ALU.

Assuming no carry is present in C_n (C_n=0), the 1-bit binary full adder gives the following outputs:

$$\sum = A \not\wedge B \quad ; \quad C_{n+1} = A \wedge B$$

The output selector can select any of these outputs to send to the 2 output line.

The following arithmetic and logic operations can be performed in the ALU:

EXCLUSIVE-OR OPERATION (EXO)

Transferring the Y input of the ALU to B in "true" form and selecting the $\sum$ output (OUT SEL 1 = 1, OUT SEL 2 = 0), the output Z is:

$$Z = \sum = A \veebar B = X \veebar Y$$

$$\therefore \quad Z = X \veebar Y \tag{3.1}$$

AND OPERATION (AND)

In the same conditions, but selecting the C_{n+1} output:

$$Z = C_{n+1} = A \wedge B = X \wedge Y$$

$$\therefore \quad Z = X \wedge Y \tag{3.2}$$

OR OPERATION (LOR)

Selecting both $\sum$ and C_{n+1} outputs, the Z output is the OR-ing of them:

$$Z = (X \vee Y) \vee (X \wedge Y) = (\overline{X} \wedge Y) \vee (X \wedge \overline{Y}) \vee (X \wedge Y)$$

$$= (\overline{X} \wedge Y) \vee X \wedge (\overline{Y} \vee Y) = X \vee Y$$

$$\therefore \quad Z = X \vee Y \tag{3.3}$$

MOVE OPERATION (MOV)

Performing and exclusive-or, but with the X input inhibited:

$$Z = 0 \veebar Y = Y$$

$$\therefore \quad Z = Y \tag{3.4}$$

COMPLEMENT OPERATION (COM)

In the same conditions, but complementing the Y input:

$$Z = 0 \veebar \overline{Y} = \overline{Y}$$

$$\therefore \quad Z = \overline{Y} \tag{3.5}$$

ADDITION OPERATION (ADD)

By using a CAR CL pulse the carry C_{n+1} is fed back to C_n. It is added in the next bit and $\sum$ gives the arithmetic addition of A and B. Selecting the $\sum$ output:

$$Z = X + Y \tag{3.6}$$

INCREMENT OPERATION (INC)

Presetting the carry flip-flop and inhibiting the X input:

$$Z = Y + 1 \tag{3.7}$$

TWO's COMPLEMENT OPERATION (TCM)

In the same condition, but complementing the Y input:

$$Z = \overline{Y} + 1 \tag{3.8}$$

SUBTRACTION OPERATION (SUB)

Adding the X input to the last operation, Z becomes:

$$Z = X + \overline{Y} + 1 = X - Y \tag{3.9}$$

DECREMENT OPERATION (DEC)

Inhibiting the Y input and holding COM line high, the B input is equal to 111 ...1. In two's complement representation, this is the number -1. Adding A + B, the Z output is:

$$Z = Y - 1 \tag{3.10}$$

In conclusion, the ALO that the ALU performs depends upon:

a) the level of COM, OUT SEL 1 and OUT SEL 2 during the cycle.

b) the absence or presence of the carry pulses (preset, clear and clock) in the proper moment.

c) the enable inputs.

These conditions are indicated in the table of Fig. 3.8, where a 1 indicates; level high, pulse present and input enable; and a 0 indicates the complementary conditions. The enable input lines are controlled by the enable lines of MPX A and MPX C.

3.6.8 Control Bits

There are four control bits (Fig. 3.5). Two of them, "zero" and "sign" are set automatically. They give an indication of the zero/non zero and sign of the last data written in AC1. On the other hand, "inhibit" and "convergence" bits are unconditionally preset in the chosen cells from the CPU during the loading.

New data enters AC1 through its "write 1" line and a sample of this information is sent to the inputs of the sign and zero FF's. When AC1 has been selected as a destination word and before the operation cycle starts, a "clear to zero" pulse is generated and the zero FF is reset. The "zero clock" interrogates every bit being written in AC1. If a '1' is detected, the zero FF goes to '1' and no further changes are allowed during the operation cycle. It can remain in '0' only if all the bits that have been written are equal to '0', that means, AC1=0.

While the bit 15 is being written, "sign clock" is generated latching its value in the sign FF. Because 2's complement representation is used, bit 15 contains the sign information (0 positive, 1 negative).

In exponential mode of operation, bit 14 is interrogated, because it is the MSB of the exponent and it contains the exponent sign.

Loading new data from the computer and transferring from the neighbors cells are performed through the "write" line. Read out and transfer to the neighbors are carried out through the "output" line. In both cases, the "control" level must be high, and four "transfer clock" pulses must be generated, allowing the shifting of information. In addition, "control out" (COU) must be high whenever read out is performed, so the information is fed back to the input and non-destructive reading is done.

3.6.9 Inhibit System

The inhibit system selects the inhibit conditions which should be applied to the cell. The inhibit control levels are available at the output of the three first control flip-flops. When their levels are high, they indicate (Fig. 3.5):

(S) that the last number written in AC1 is negative.

(Z) that the last number written in AC1 is non-zero.

(I) that unconditional inhibit can be performed in the cell.

The "1 detectors" of AC1 are also part of the inhibit system and they indicate:

(A) that a '1' has been detected at the MSB of AC1, when FLT mode operation is performed.

(B) that a '1' has been detected at the MSB of AC1, when $\overline{\text{FLT}}$ mode operation is performed.

One of the conditional inhibit lines, B (A in FLT mode), S, or Z, can be selected using the 4-to-1 inhibit multiplexer (INH MPX) for performing conditional inhibit (Fig. 3.6). An EXO gate at the output of the INH MPX, controlled by the "sign" line, permits the user to complement the inhibit condition, i.e., inhibit when (A) or (B) detects a '0', or when AC1 $\geq$ 0, or when AC1 = 0.

If INH MPX selects the 0 input line, no conditional inhibition is allowed.

Unconditional inhibit flip-flop can be interrogated simultaneously with any of the conditional ones. The INH line is the OR-ing of the conditional and unconditional inhibit. It disables changing of information in memories and shifting of accumulators. The INH MPY line also inhibits the change of information in memories, but the accumulators are enabled for shifting.

3.6.10 Convergence

All incoming data are compared with the corresponding bit stored in M_{15} in the EXO convergence (CNV) gate (Fig. 3.6). If they differ and the cell has been selected for convergence tests (i.e., CNV FF=1), convergence pulses are sent to the CU, where they may or may not be interrogated, according to program control.

CHAPTER 4

Operation of the Cell

4.1 Operand Selection

According to the number of operands involved in the operations, the cell can be considered as a triple, double, or single-address machine.

4.1.1 Triple-Address Instructions

In the operations ADD, SUB, AND, LOR and EXO, two operands (X and Y) are involved, according to the characteristics of the operations themselves. Both source operands and the destination word are selected by the programmer and therefore three addresses must be specified in the instruction.

The X operand is selected from the bank of memory by MPX A. The Y operand is selected from the bank of memory of the cell or its neighbors by a combination of MPX B and MPX C. However, when MPX C selects the EXT input, the Y operand is taken directly from the CB, without intervention of MPX B. The answer Z can be directed to any of the words or all of them, according to the selection performed by DMPX. From the operands selection point of view, both accumulators are considered like any one of the other words.

4.1.2 Double-Address Instructions

Four groups of operations can be distinguished in that category:

a) Complement, two's complement, increment and move.

b) Shift serially to the right (SSR).

c) Decrement.

d) Multiplication.

In groups a), b) and c), only one operand is involved, according to the characteristics of the operations themselves. Therefore, the operand and the destination word must be specified in the instruction.

In operations of the group a), the operand is selected by the combination of MPX B and MPX C in a similar fashion to that for triple-address instructions, and the same selective capability is available. However, MPX A is inhibited and no signal in the X input of the ALU is obtained.

In SSR operation (group b), although only one operand is involved, the technique used consists in adding the number to itself, but starting from the MSB. Both multiplexers, MPX A and MPX B, must select simultaneously the word to be shifted, and MPX C has to select the internal (INT) input. Only "in-cell" operations are allowed.

In DEC (group c), MPX C is inhibited. The word is selected "in-cell" by MPX A and it is directed to the X input of the ALU.

In the three groups (a, b and c), DMPX selects the destination word.

Although MPY (group d) is an operation in which two operands (multiplicand and multiplier) are involved, the cell is limited to operations in which AC1 is the multiplicand. The programmer must specify solely the multiplier and the places he wishes to store the

answer. The MPX's and the DMPX will perform the selection.

Suppose it is wished to perform the multiplication AC1 x M_X = M_Y. When MPY line is activated, the multiplicand AC1 is sent directly to the X input of the ALU (Fig. 3.4). The MPX A selects the multiplier, M_X. It is interrogated and latched in D_A during the add cycle, for inhibit purposes. The MPX B selects automatically the same word that has been chosen by DMPX for storing the answer, and thus allows the cell to perform AC1 + M_Y = M_Y, during the add cycle. Whether the addition is performed or inhibited, depends on the state of D_A FF.

4.1.3 Single-Address Instructions

Parallel shiftings are included in this category. Only one operand is involved (AC1 or AC2) and the answer is stored in the same operand. These operations are carried out in the accumulators, without using any of the multiplexers or demultiplexers of the cell.

The operations "Interchange AC1" and "Invert AC2" are also single-address instructions. Because one of the operand is implicit in the instruction itself, the programmer has the freedom to choose the other operand. The selection is physically realized by the DMPX.

Finally, "Control transfer" is a one-address instruction, but the neighbor, rather than the word, should be chosen as an operand. The selection is done by MPX C.

4.1.4 Operate Instructions

Operate instructions are those in which no operands need to be

specified. The special operation "Move $\overline{\text{sign}}$ to AC2" is the only one in that category.

The "Computer-cell communication" (CCC) and "Buffer-array communication" instructions have not been included in any of the preceding groups because the cell addressing system is involved in the selection.

4.2 Instruction Set

The minicomputer used is a 16-bit word machine. Its interface characteristics make it possible to transfer instructions from the CPU to the interface unit in parallel fashion, 16 bits simultaneously. But because of the diversity of the functions the array must perform, two computer words are necessary for coding most of the operations. A 32-bit instruction register (IR) in the interface unit is provided. The complete instruction is stored there and it is available for decoding.

The IR is loaded in two halves by the CPU, in the following sequence:

1st)	bits	16 - 31	(second half)
2nd)	bits	0 - 15	(first half)

Simultaneously with the loading of the first half, a "start" pulse is generated and the array operation begins.

But it has been found that for some operations, no more than 16 bits of the IR content change from one instruction to the next one. In that case, two cycles would be spent to fill up the 32 bits of

the IR when actually only half of them have changed. It is a waste of time and storage capacity of the minicomputer. The problem has been solved with an efficient design of the instruction set such that the part of the instruction that most frequently remains unchanged from one operation to the next one is in the second half. The part of the instruction corresponding to neighbors, mode, inhibit, convergence and carry has been placed there. Hence, if during a particular operation, the second half of the instruction should be the same as that of the preceding instruction, only the first part should be loaded.

A particular but frequent case is when the operation must be performed "in cell", in integer mode, clearing the carry, and without inhibit or convergence test. In our instruction set it corresponds to all the bits of the second half equal to zero, making easier the programming, and shortening the compiling time.

In the operations for which flow of information between the computer and the cells are involved, only 16-bit instructions are required. They are distinguished from the ALO by the value of bit 16.

The complete instruction set, shown in Table 4.1, is described in the following sections.

4.2.1 Arithmetic and Logic Operations

The bit 16 = 0 indicates ALO. In that case:

Bits 0-3, OPCODE; select the desired operation.

Bits 4-7, DESTINATION; select the word in which the answer of the operation is stored.

Bits 8-11, SOURCE A; select one of the operand.

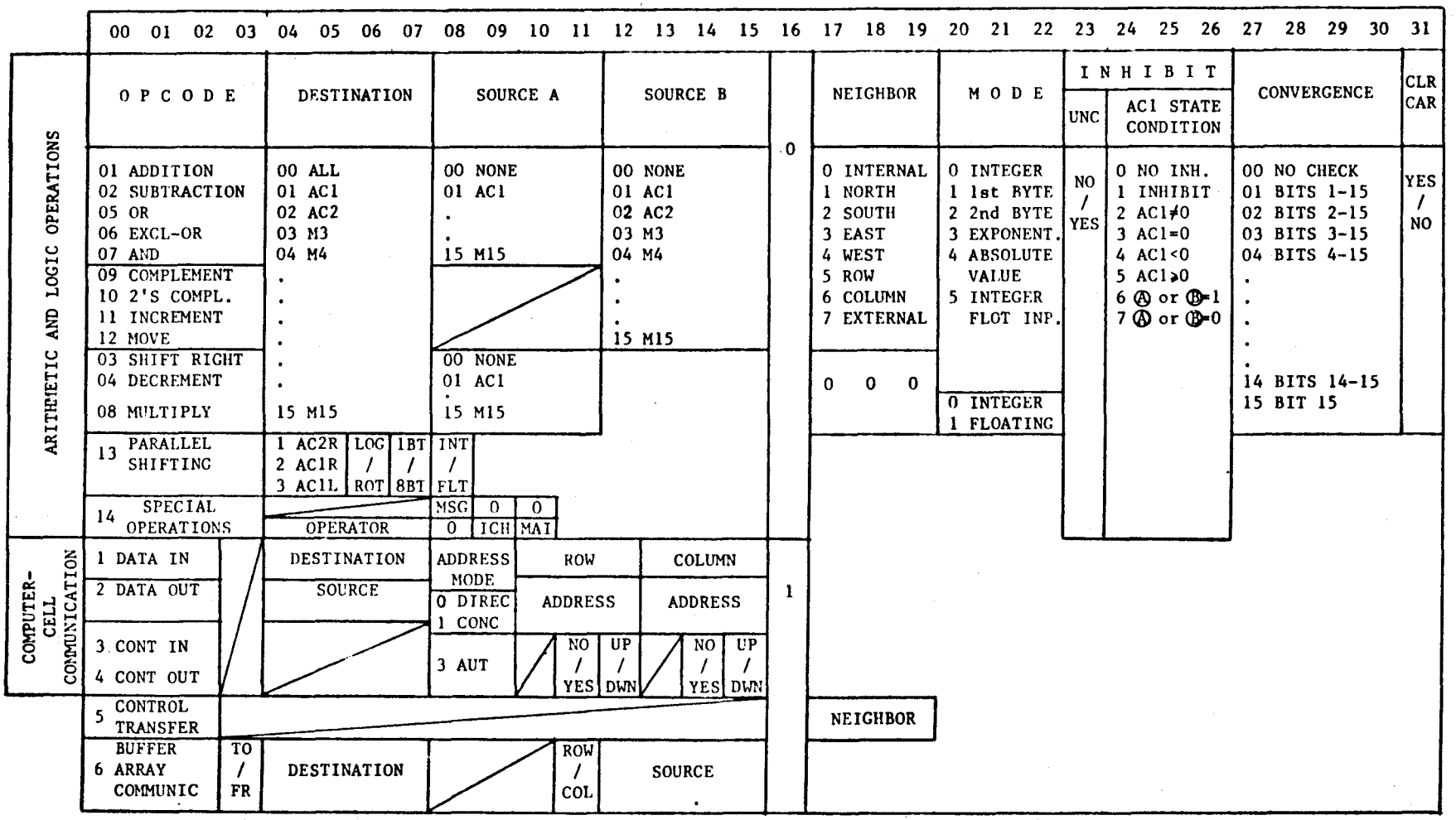


TABLE 4.1. INSTRUCTION SET

Bits 12-15, SOURCE B; select the other operand.

Bits 17-19, NEIGHBOR; select the neighbor cell.

Bits 20-22, MODE; select the mode of operation, as it was discussed in section 3.4.

Bit 23, UNCONDITIONAL INHIBIT; selects whether the unconditional INH control bit is interrogated or not.

Bits 24-26, CONDITIONAL INHIBIT; select the AC1 state condition being interrogated.

Bits 27-30, CONVERGENCE; select the range of CNV interrogation.

Bit 31, CLEAR CARRY; selects whether the carry should be cleared or not before the operation cycle starts.

When OPCODE = 13, PARALLEL SHIFTING is performed. In that case:

Bits 4-5 select accumulator and shifting direction.

Bit 6 selects logical shifting or rotation.

Bit 7 selects the shifting of 1 or 8 bits.

Bit 8 selects the INT or FLT input.

Bits 23-26, INHIBIT; select the inhibit condition.

The other bits are meaningless.

When OPCODE = 14, SPECIAL OPERATIONS are performed. In that case:

Bit 8 selects the MSG operation

Bit 9 selects the ICH operation

Bit 10 selects the MAI operation.

In the last two cases;

Bits 4-7 select the operator word.

4.2.2 Computer-Cell Communication Operations

For CCC operations, should be bit 16=1.

Bits 0-2, OPCODE; select the desired operation.

Bits 4-7, select the desired word, in the case of Data in (DIN) or Data out (DOU) operation.

Bits 8-15, CELL ADDRESS; select the cell, as will be discussed in Chapter 5.

4.2.3 Control Transfer Operations

For control transfer operations, should be bit 16=1 and bits 0-2=5. In that case:

Bits 17-19, NEIGHBOR; select the neighbor from where the transfer is produced.

Although bits 3-15 are not used, it is convenient to maintain the same bits used in ALO for neighbor selection. This is because the control transfer operation usually follows a data transfer (MOV operation). In that case it is not necessary to change the neighbor bit from one instruction to the next one.

4.2.4 Buffer-Array Communication Operations

For buffer-array communication operations, should be bit 16=1 and bits 0-2=6. In that case:

Bit 3 selects transfer to or from buffer.

Bits 4-7 select the destination word.

Bit 11 selects the row or column buffer.

Bit 12-15 select the source word.

4.3 Mnemonic Language

In an attempt to facilitate the communication between the user and the array, a set of mnemonics is proposed.

4.3.1 Serial Arithmetic and Logic Operations

For serial ALO the instructions have the format:

OPCODE, D, A, B, N, M, UI, CI, V, W

where

OPCODE indicates the desired operation. It is a three characters mnemonic, and can be any of the following:

ADD, addition

SUB, subtraction

EXO, exclusive-or

LOR, logical-or

AND, logical-and

COM, complement

TCM, two's complement

DEC, decrement

INC, increment

MOV, move

SSR, shift serially to the right

MPY, multiplication

D indicates the "destination" word address. A "0" stands for all of the words of the cell and M_1 (or AC1), M_2 (or AC2), M_3, M_4...M_{15} indicate a particular word selected.

A indicates the "source A" word address. A "0" stands for none operand, and M_1 (or AC1), M_2 (or AC2), M_3, M_4,...,M_{15} select the particular word.

B indicates the "source B" word address. Analogously than A, it can have the values 0, M_1, M_2,...,M_{15}. (A or B are skipped in the double-address instructions, according to the table 4.1).

C indicates the neighbor selected. It can be

0, in-cell

N, north

S, south

E, east

W, west

ROW, row

COL, column

EXT, external

M defines the mode of operation. It can be:

0, integer

FB, first byte

SB, second byte

EX, exponential

AV, absolute value

INF, integer with floating input

FLT, floating multiplication

UI indicates that unconditional inhibit bit is checked.

CI indicates the conditional inhibit. Its value can be

0 (none), no inhibit required

YZ (yes zero), inhibit if (AC1)=0

NZ (non zero), inhibit if (AC1)≠0

PS (positive sign), inhibit if (AC1)≥0

NS (negative sign), inhibit if (AC1)<0

ZD (zero detected), inhibit if a '0' in "1 detector"

OD (one detected), inhibit if a '1' in "1 detector"

V indicates the convergence range. Its value can be

0, no convergence required.

CNV1, CNV2,...,CNV15, selects the convergence range.

W indicates whether the carry is cleared or not before the operation starts. When it assumes the value WCC, operation without clear the carry is assumed. Otherwise, the carry is cleared.

4.3.2 Parallel Shifting Operations

For parallel shifting, the general format is:

OPCODE, RD, N, M, UI, CI

where

OPCODE indicates the desired operation. The mnemonics are

SHF - for logical shifting

ROT - for rotation.

RD indicates register and shifting direction, as follows:

AC2R, accumulator 2 to the right.

AC1R, accumulator 1 to the right.

AC1L, accumulator 1 to the left.

N indicates the number of bits, as follows:

0, shift 1 bit.

BIT8, shift 8 bits.

M indicates the mode of shifting, as follows:

0, integer (using 16-bit accumulators).

FLT, floating (using 24-bit accumulators).

UI indicates unconditional inhibit, as in the ALO.

CI indicates the conditional inhibit, as in the ALO.

4.3.3 Special Operations

For special operations, the following formats are used:

MSG indicates move $\overline{\text{sign}}$ to AC2.

MAI, D indicates move (AC2) inverted to D (with D varying between M_3 and M_{15}).

All the inhibit mnemonics used for serial ALO are also valid for special operations.

4.3.4 Computer-Cell Communication Operations

For CCC operations, the general format is:

OPCODE, U, M, RA, CA

where:

OPCODE indicates the desired operation. The mnemonics are:

DIN, data in.

DOU, data out.

CIN, control in.

COU, control out.

W indicates the word selected. It indicates the destination word in DIN and the source word in DOU operations, and it is not specified in CIN and COU operations.

M† indicates the cell addressing mode, as follows:

0, direct.

CONC, concatenation.

AUT, automatic.

RA indicates:

the row address (0 to 7) in direct or CONC mode;

the AUT increment (ROW UP or ROW DOWN) in AUT mode.

CA indicates:

the column address (0 to 7) in direct or CONC mode;

the AUT increment (COL UP or COL DOWN) in AUT mode.

4.3.5 Control Transfer Operations

For control transfer, the format is:

CTR, N

where

---

†M, RA and CA are related to the CAS and the details of them will be discussed in section 5.4.

CTR is the opcode for the operation, and

N indicates the neighbor source selected.

4.3.6 Buffer-Array Communication Operations

For buffer-array communication, the general format is:

OPCODE, D, S, B

where

OPCODE indicates the desired operation. The mnemonics are:

TTB, transfer to buffer

TFB, transfer from buffer.

D indicates the destination word (as in ALO).

S indicates the source word (as in ALO).

B indicates the selected buffer, as follows:

ROW, row buffer

COL, column buffer.

CHAPTER 5

The Control Unit

The aim of the Control Unit (CU) has been pointed out in Chapter 2. The CU is composed of three subunits, as is shown in Fig. 5.1.

1. The Instruction Decoder (ID)

   The ID has the task of decoding the 32-bit instruction loaded in the Instruction Register (IR) and generating all the necessary control levels; which are broadcast to the array of cells, the other CU subunits and the Interface Unit.

2. The Control Pulses Generator (CPG)

   The CPG has the task of generating all the necessary trains of pulses. It is divided into two main parts:

   a) The Clock Unit. The Clock Unit receives information from the Device Status Register (DSR) about the "busy/free" (B/F) condition of the system; and it keeps sending pulses to the Sequencer as long as the DSR remains in "busy" state.

   b) The Sequencer. The Sequencer receives the clock pulses, as well as the decoded levels from the ID, and according to the desired operation it generates the required trains of pulses. They are transmitted to all the cells in parallel. In addition, the Sequencer receives information about the convergence condition of every cell and it sends the general convergence

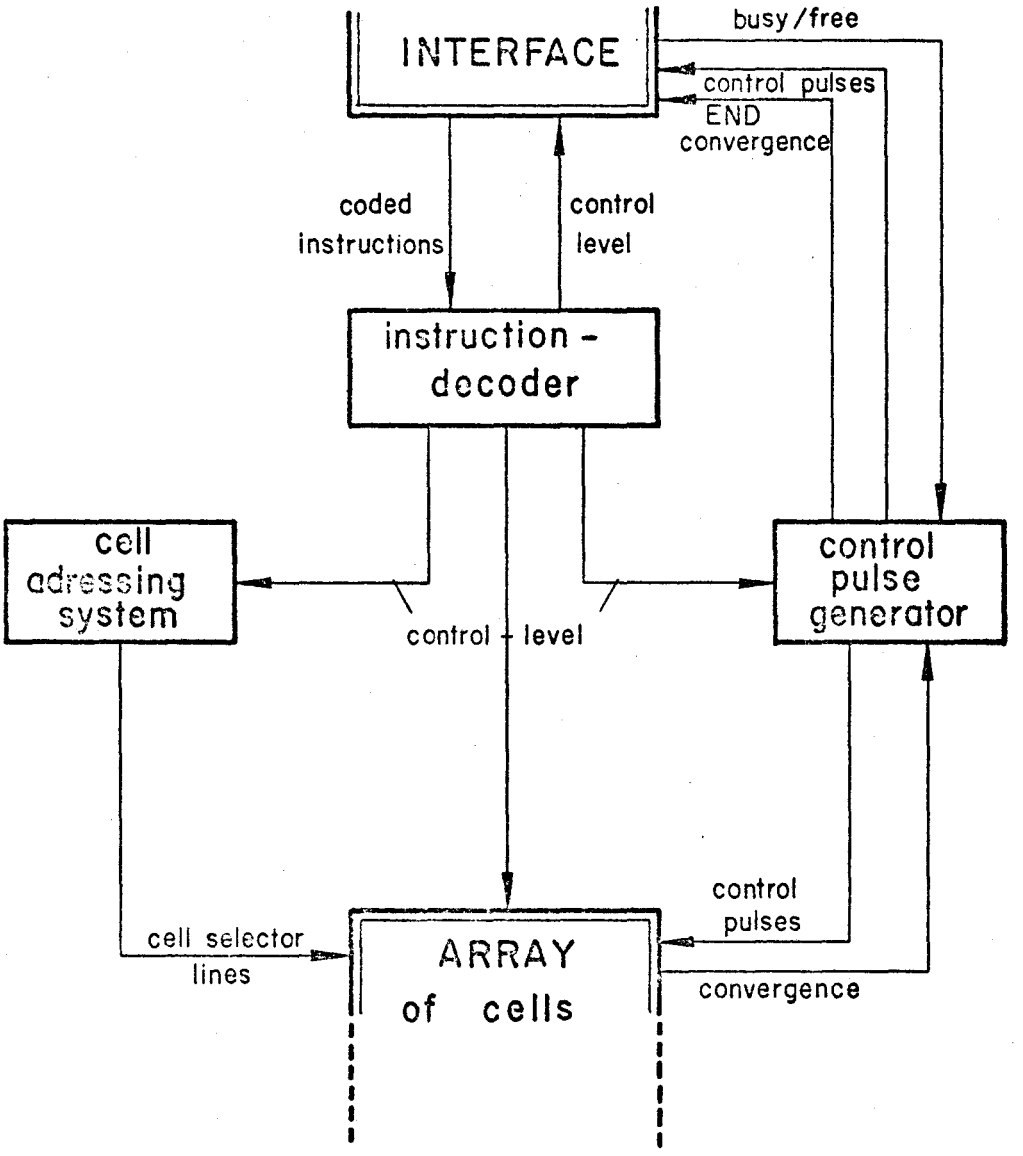


FIGURE 5.1. THE CONTROL UNIT
BLOCK DIAGRAM

condition of the system as a whole to the DSR. Finally, the Sequencer controls how long the operation should last, by sending "end" pulses to the DSR.

3. The Cell Addressing System (CAS)

The CAS has the task of selecting a particular cell (or cells), whenever it is required by the operation. The necessary information is taken from the ID, the Basic Address Register (BAR) and the Cell Mode Register (CMR). Selective control levels are sent to the cells of the array, allowing certain degree of exclusiveness in some operations.

The operation of the CU subunits, as well as the actual implementation are described in the following paragraphs.

5.1 The Instruction Decoder

Once the 32-bit IR is loaded with the corresponding coded instruction, certain logic levels in the control lines should be created in order to perform the actual instruction.

According to the discussion of Chapter 3, about the logic of the cells, the Table 5.1 has been constructed. The logical levels of the control lines for any possible operation are shown there. The following convention has been adopted.

a 0 indicates low level or pulse absence,

a 1 indicates high level or pulse presence,

an S indicates that the level or pulse presence are selected

OPERATION	MPX A ENB	MPX B ENB	MPX C ENB	DMPX ENB	MPX A SEL(4)	MPX B SEL(4)	MPX C SEL(3)	DMPX SEL (4)	MPY	COM	OUT SEL #1	OUT SEL #2	FLT INP	SHR AC1 DIR.	ROT AC1	ROT AC2	SHR AC1	SHR AC2	SHR 1BIT	AC1 ENB	AC2 ENB	CONTROL	CCU	SR AC1	CNT DOWN	MSG	INT MPY	FLT MPY	BYTE OPER.	FB	INH SEL (4)	CNV SEL (4)	EX	AV	ICH	CLOCKS PR CAR	CLOCKS CAR	CLOCKS CL CAR
ADD	1	1	1	1	S	S	S	S	0	0	1	0	S	0	S	S	0	0	0	S	S	0	0	S	0	0	0	0	S	S	S	S	S	S	0	0	1	S
SUB	1	1	1	1	S	S	S	S	0	1	1	0	S	0	S	S	0	0	0	S	S	0	0	S	0	0	0	0	S	S	S	S	S	S	0	1	1	S
LOR	1	1	1	1	S	S	S	S	0	0	1	1	S	0	S	S	0	0	0	S	S	0	0	S	0	0	0	0	S	S	S	S	S	S	0	0	0	S
EXC	1	1	1	1	S	S	S	S	0	0	1	0	S	0	S	S	0	0	0	S	S	0	0	S	0	0	0	0	S	S	S	S	0	S	0	0	0	S
AND	1	1	1	1	S	S	S	S	0	0	0	1	S	0	S	S	0	0	0	S	S	0	0	S	0	0	0	0	S	S	S	S	S	S	0	0	0	S
COM	0	1	1	1		S	S	S	0	1	1	0	S	0	S	S	0	0	0	S	S	0	0	S	0	0	0	0	S	S	S	S	0	S	0	0	0	S
TCM	0	1	1	1		S	S	S	0	1	1	0	S	0	S	S	0	0	0	S	S	0	0	S	0	0	0	0	S	S	S	S	0	S	0	1	1	S
INC	0	1	1	1		S	S	S	0	0	1	0	S	0	S	S	0	0	0	S	S	0	0	S	0	0	0	0	S	S	S	S	0	S	0	1	1	S
MOV	0	1	1	1		S	S	S	0	0	1	0	S	0	S	S	0	0	0	S	S	0	0	S	0	0	0	0	S	S	S	S	0	S	0	0	0	S
SSR	1	1	1	1	S	=A	I	S	0	0	1	0	S	0	S	S	0	0	0	S	S	0	0	S	1	0	0	0	S	S	S	S	0	S	0	0	1	S
DEC	1	0	0	1	S			S	0	1	1	0	S	0	S	S	0	0	0	S	S	0	0	S	0	0	0	0	S	S	S	S	0	S	0	0	1	S
MPY a/s	1	1	1	1	S	=D	I	S	1	0	1	0	S	0/1	1	S	0/1	0	0	1	S	0	0	0	0	0	S	S	0	0	S	S	0	0	0	0	1	S
SHR	0	0	0	0					0				S	S	S	S	S	S	S	0	0	0	0		0	0	0	0			S				0			
MSG	0	0	0	0					0				0			0	0	1	1	0	0	0	0		0	1					S				0			
ICH	0	1		1		=D		S	0	0	0	0			0	0				1															1			
MAI	1			1	2			S	0	0	1	0			0	0	0	0	0	0	0	1	0	0	0	1	0	0	0						0	0	0	S
DIN	0	0	1	1			E	S	0	0	1	0	0	0	0	0	0	0	0	S	S	0	0	S	0	0	0	0			S	S	0		0	0		S
DOU	0	1	0	0		S			0				0	0	S	S	0	0	0	S	S	0	0		0	0	0	0							0			
CIN	0	0	1	0			E		0	0	1	0				0	0	0	0	0	0	1	0		0										0	0		
COU	0	0	0	0					0							0	0	0	0	0	0	1	1		0										0			
CTR	0	0	1	0			S		0							0	0	0	0	0	0	1	0		0										0			

TABLE 5.1

LOGICAL LEVELS OF THE CONTROL UNIT LINES

Nb: 0, low level or pulse absence
1, high level or pulse presence
S, selected by the user
I, 'internal' input

=A, MPX A selection
=D, DMPX selection
'E, 'external'input
2, AC2 selected

by the user, either directly or indirectly, and

a "blank" indicates that the level of this particular line does not affect the operation.

During the MPY cycle, a line can assume two different levels - one during the INH/ADD cycle and another during the shifting. This condition is also indicated in the table by 0/1. The 2, INT and EXT are fixed values that the corresponding multiplexers selector lines assume during the operation. Finally, the table shows that in ICH, MPY and SSR, the MPX B selector lines are set equal to the value of the selector lines of DMPX or MPX A. That means that although the value is selected by the user, it is done indirectly. Thus one can select the value of DMPX or MPX A selector lines, and MPX B automatically takes these values for its selector lines.

An example will clarify the use of the table. Let us suppose an addition is being carried out. The three multiplexers and the DMPX are enabled. The value of their control lines are selected by the user in his instruction. All the lines indicated by "0" must be held low during the operation, while the "output selector 1" (OUT SEL 1) line should be held high. The CAR CL should appear in the precise moment. The user can choose directly the level of the "byte operation", FB, FLT INP, INH, CNV, AV and EX lines, as well as the presence or absence of the "clear carry" (CLR CAR) pulses. The selection should be done in accordance with the characteristics of the addition the programmer wishes to perform. The other lines marked by "S" are indirectly chosen. For example "write AC1" or "write AC2" are set automatically at "1" whenever AC1 or AC2 are respectively selected

by the DMPX; otherwise they are equal to "0". Similarly, AC1 ENB and AC2 ENB are set automatically at "1" whenever the corresponding accumulator has been selected as a source or destination word.

The set of instructions (Table 4.1) gives the values of the decoder input levels, while the Table 5.1 gives the output levels desired. The combinational logic circuits that produce these output levels have been designed and are illustrated in Figs. 5.2, 5.3, 5.4 and 5.5.

5.2 The Clock Unit

The Clock Unit generates the basic trains of pulses that are used by the Sequencer. Because the characteristics of the cell, as were discussed in Chapter 3, two trains of pulses, shifted 180° in phase are necessary to complete a cell operation. Basically, one of the trains produces the "Read" subcycles (clock A) and the other one, the "Write" subcycles (clock B). Both are derived from a Master Clock train of pulses. During the part of an operation in which no delay between read and write is required, the computation is speeded up by using the Master Clock.

The three trains of pulses as well as the circuit used to generate them are illustrated in Fig. 5.6. The pulses are allowed to reach the Sequencer whenever the system is in "busy" state.

The primary source of pulses is a D.E.C.-M401 Variable Clock which is running at 4.5 MHz.

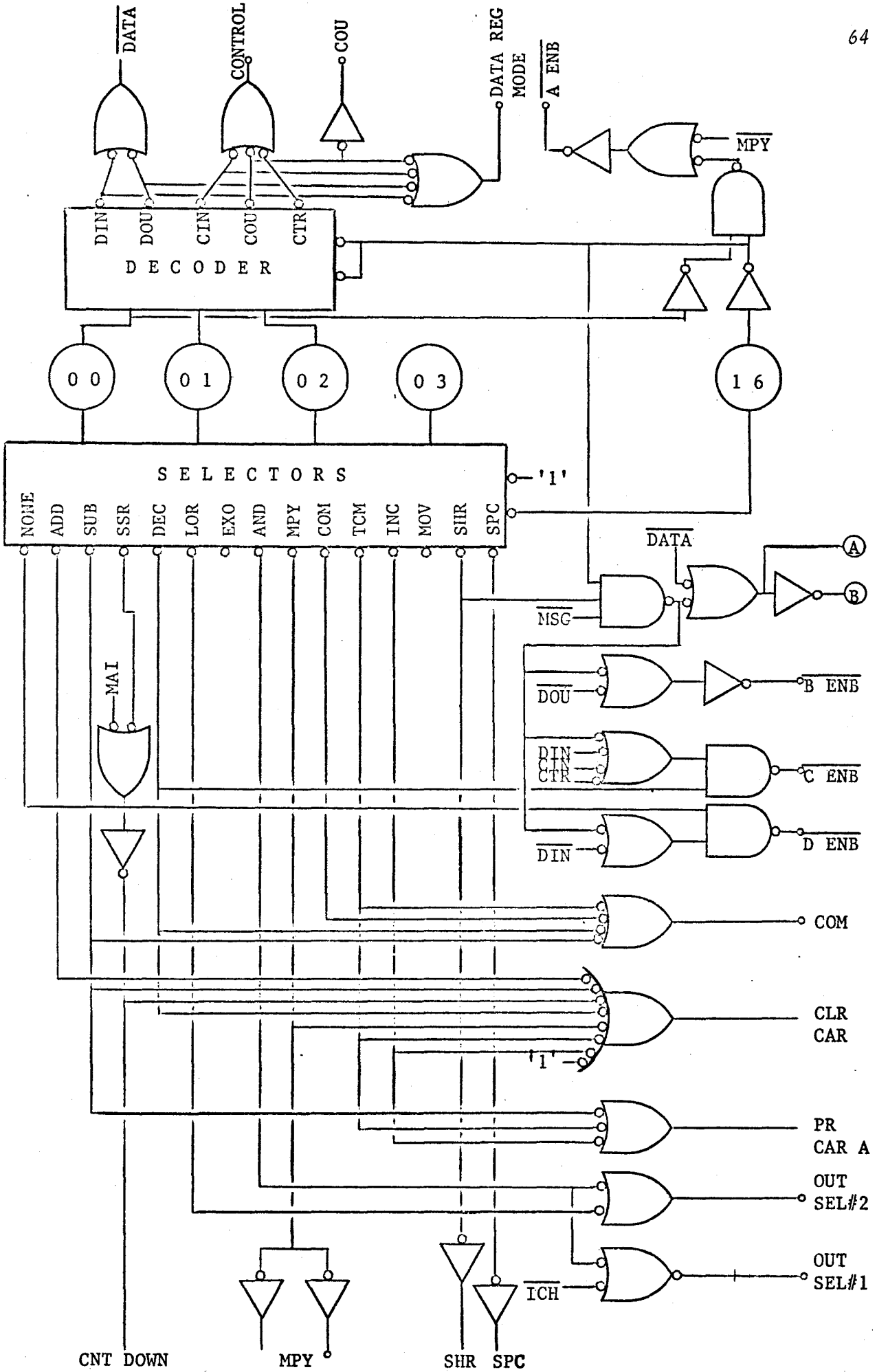


FIGURE 5.2. THE CONTROL UNIT. INSTRUCTION DECODER

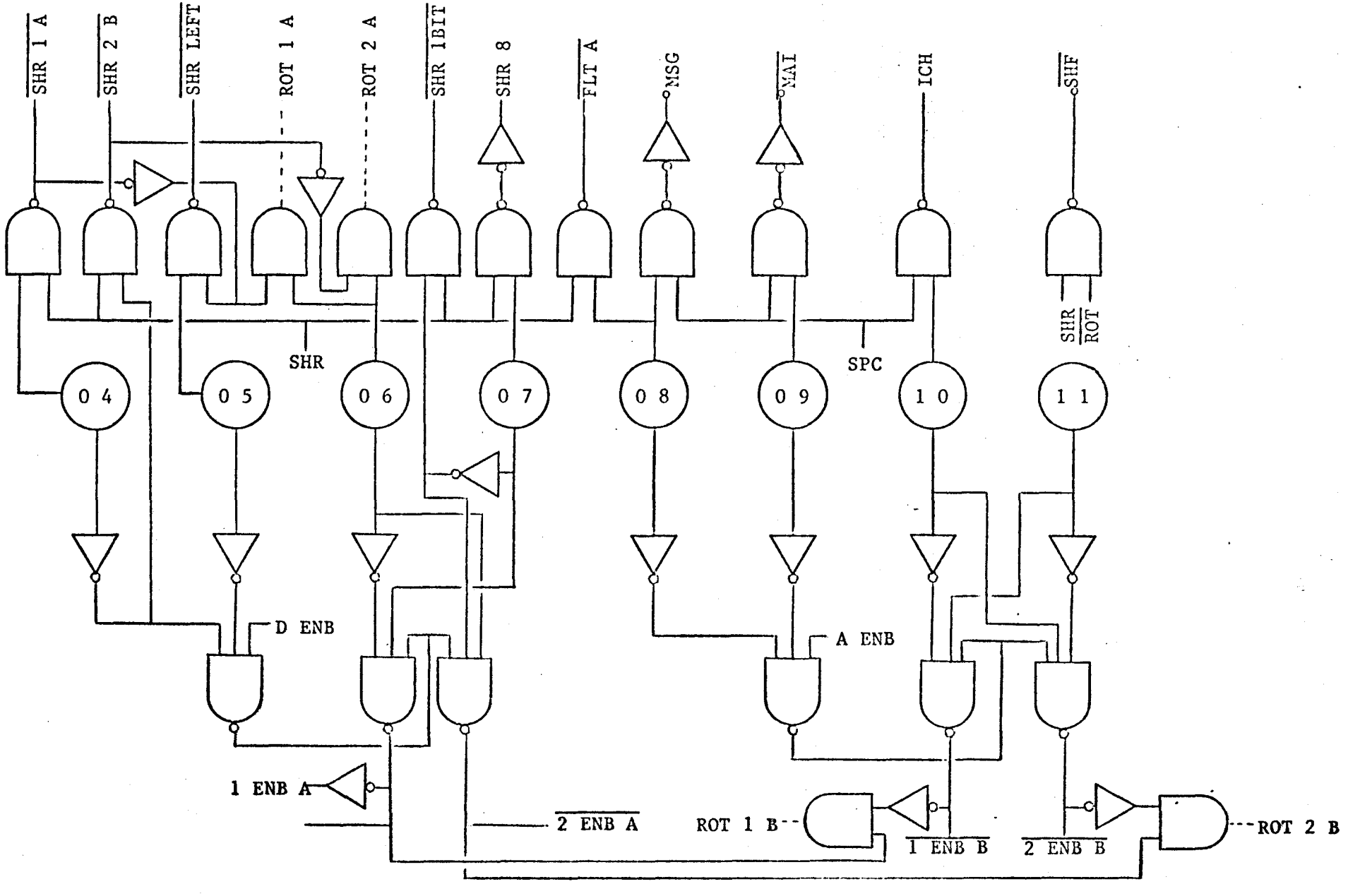


FIGURE 5.3. THE CONTROL UNIT. INSTRUCTION DECODER

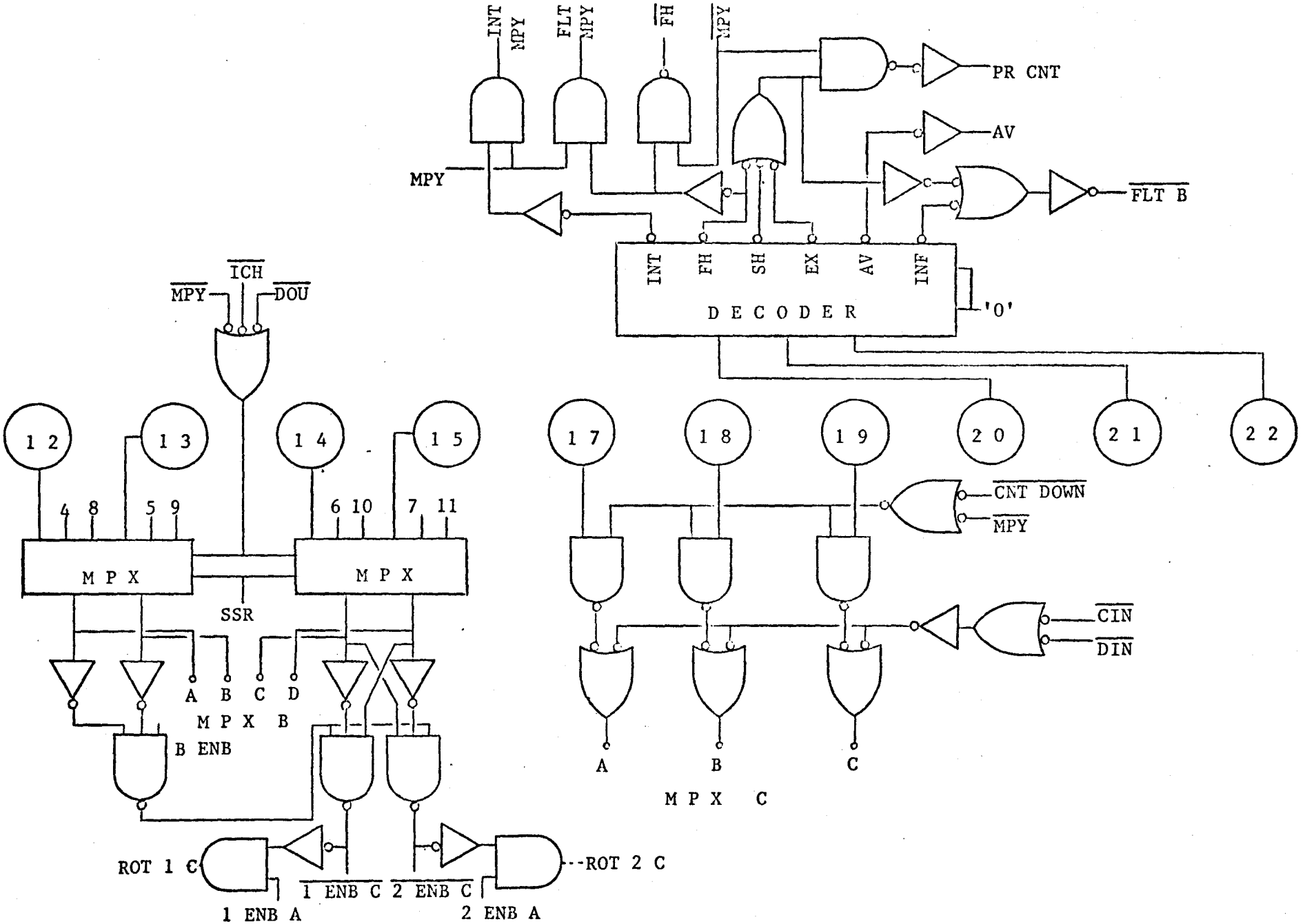


FIGURE 5.4. THE CONTROL UNIT. INSTRUCTION DECODER

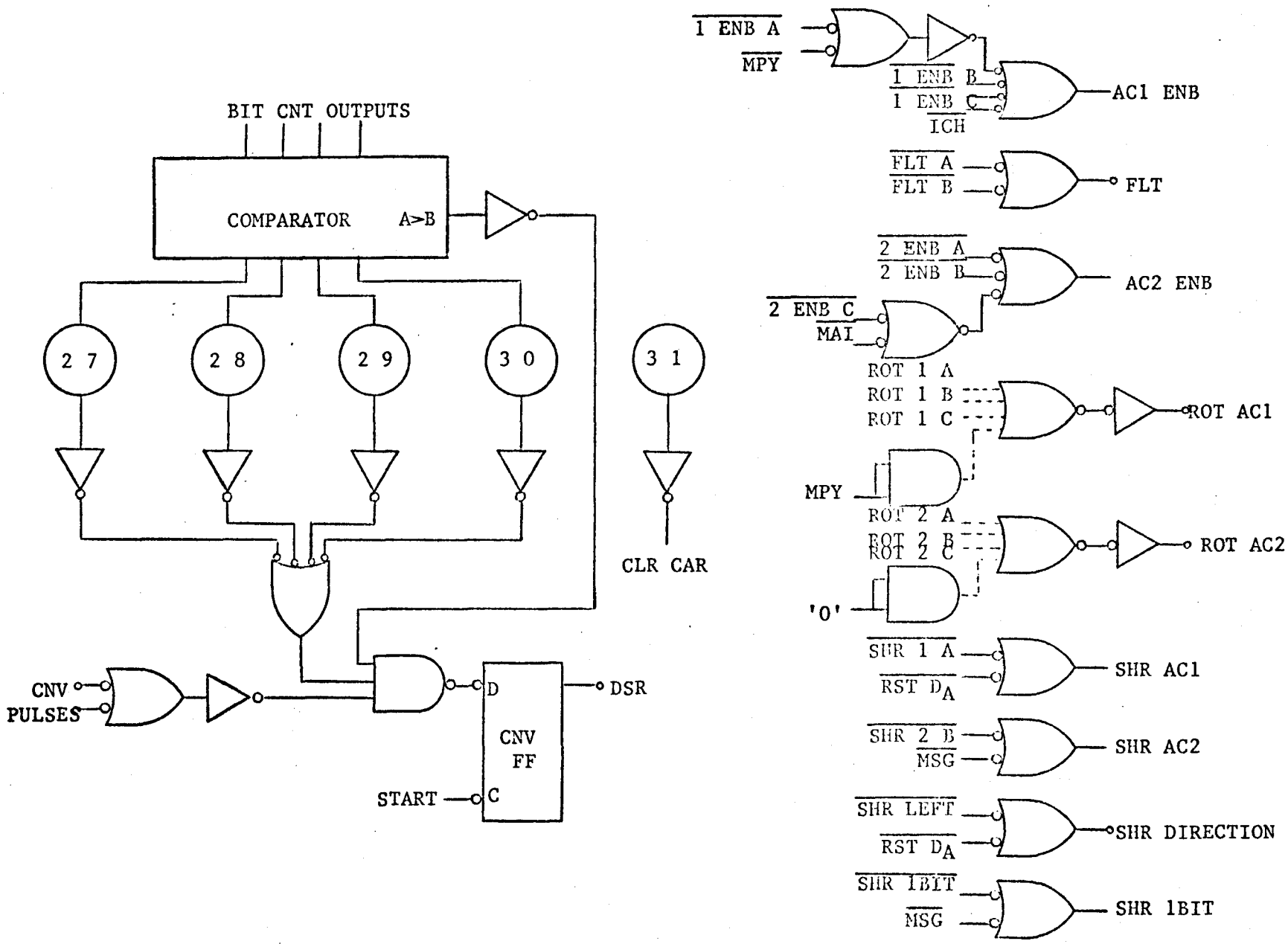
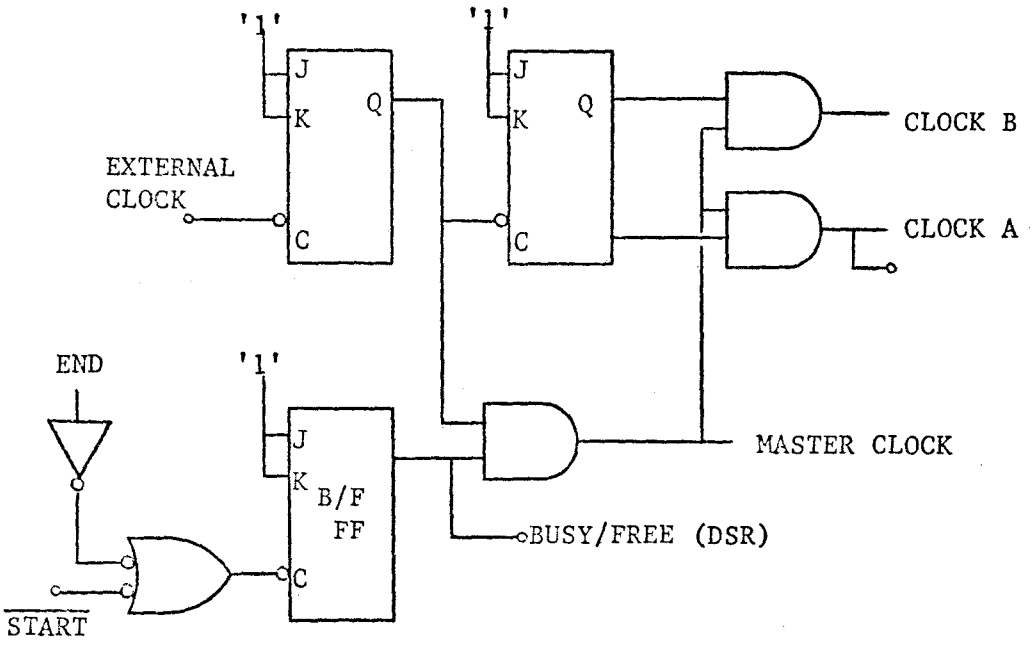


FIGURE 5.5. THE CONTROL UNIT. INSTRUCTION DECODER



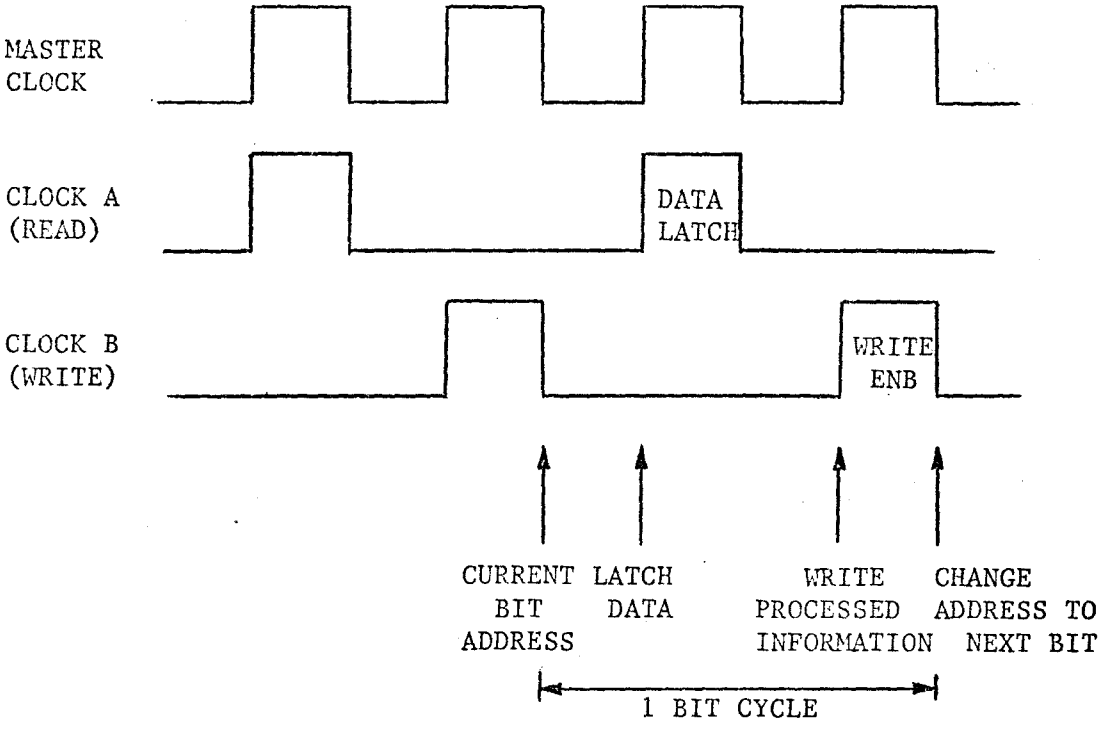


FIGURE 5.6 THE CLOCK UNIT

5.3 The Sequencer

The Sequencer receives the three trains of pulses from the Clock Unit and distributes them over the clock lines, in accordance to the operation decoded.

Before starting tht operation cycle, a preset cycle is performed. It lasts for three master clock cycles (1½ bit cycles) and it is necessary in order to clear the preset flip-flops and counters.

A Pre-counter (counter-to-4), a decoder and several gates control the preset cycle (Fig. 5.7). The Master CL pulses are used to clock the Pre-counter, which in effect controls the decoder. Sequentially, pulses to clear the carry flip-flop and to load the bit-counter, and to reset D_A, to clear to zero and to preset the carry flip-flop are generated during the first two master clock pulses received. After the third pulse arrives, the Pre-counter stops and CL B pulses are allowed to propagate to the write pulses lines and to clock the Bit-counter. If the ID has enabled the corresponding gates, CL B performs other tasks, activating the CL AC1, CL AC2, zero CL, CAR CL and transfer CL lines. In control transferring and shifting operations, CL B is directed to the corresponding lines, without waiting for the preset cycle.

The Bit-counter (counter-to-16), and the Bit-decoder (Fig. 5.8) have two main tasks:

a) to address bit by bit the 16 flip-flops of the memories; and

b) to count the number of CL B pulses that have reached the enable lines in order to detect when the operation should finish and when sign interrogation pulses must be produced.

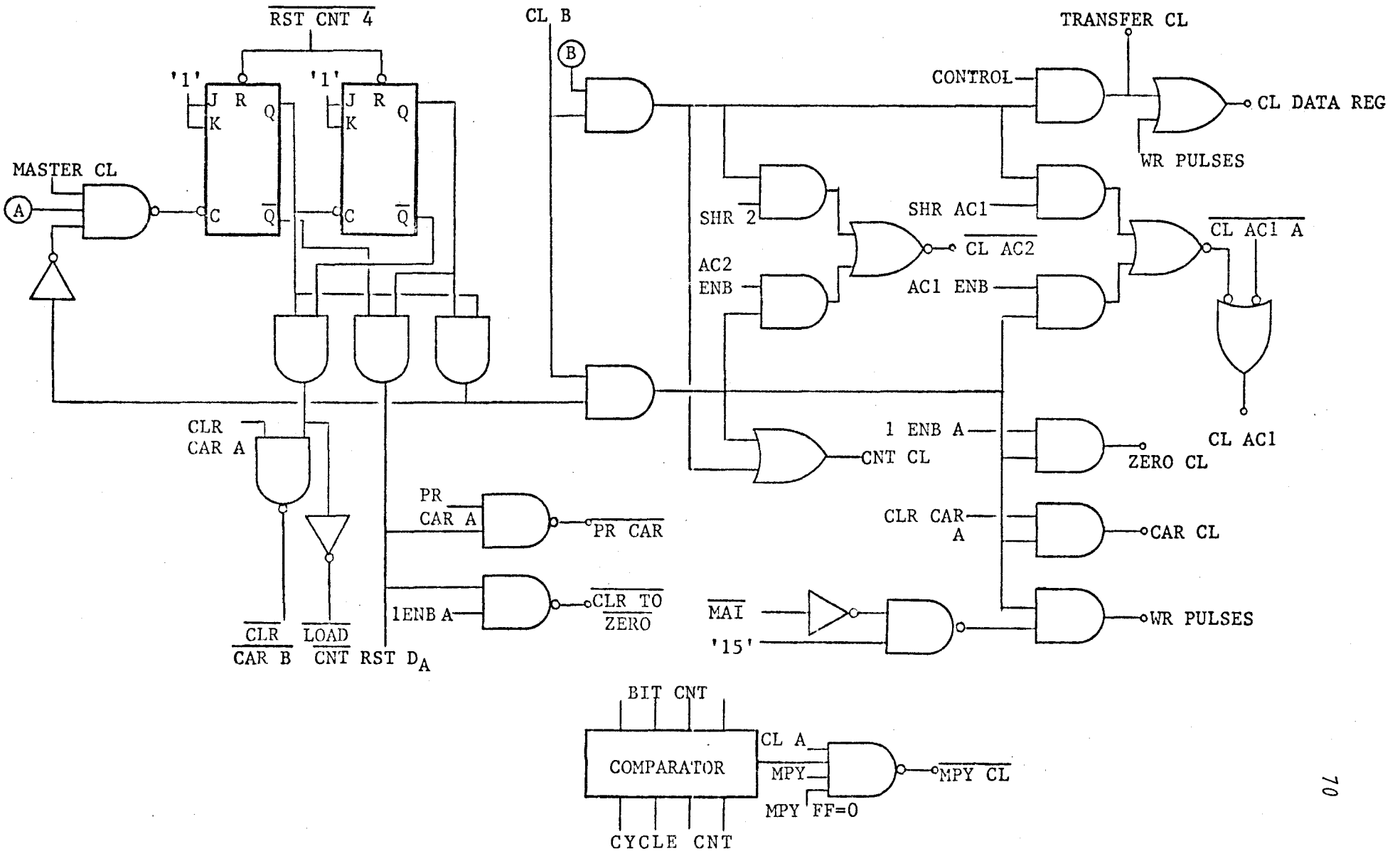


FIGURE 5.7. THE CLOCK UNIT. THE SEQUENCER

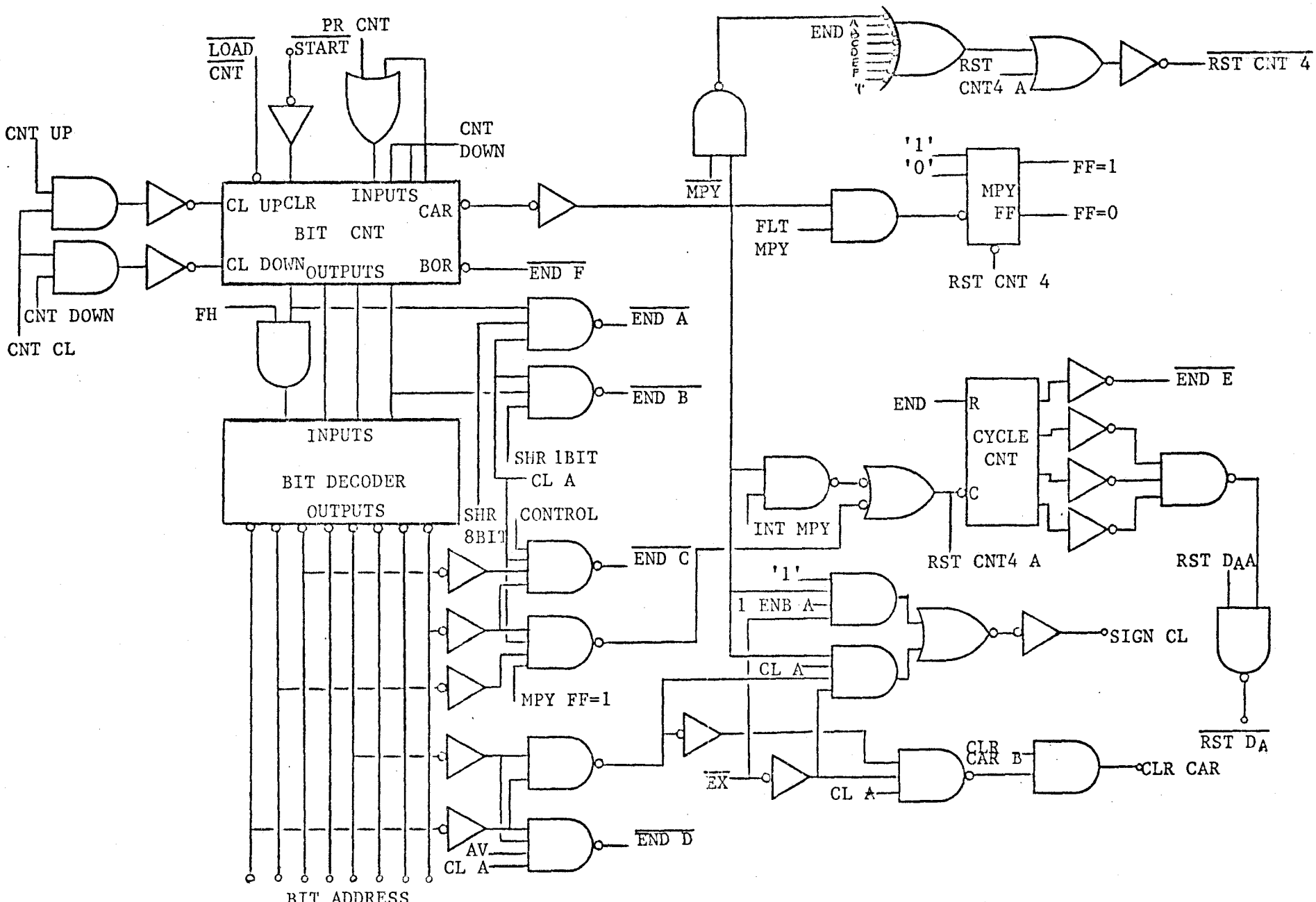


FIGURE 5.8. THE SEQUENCER

Several detectors have been set in this part of the circuit. Thus, 1, 4, 8 and 15-detectors are activated whenever SHF 1 bit, control, SHF 8 bits and AV operations, respectively, are performed. The aim of all of them is to send an "end" pulse whenever the corresponding conditions are detected.

Whenever EX operations are performed, the 15-detector has an additional task; to clear the carry in order to exclusive-or the 15 bit and to generate a "sign interrogation" pulse. Since in MAI operation, bit 15 (sign bit) must not change, the 15-detector is also used to inhibit momentarily the generation of write pulses.

If none of the previous detectors have produced an "end" pulse, then the Bit-counter finally overflows and a "carry" pulse is generated. It indicates the end of the operation, besides interrogates the sign.

If the cell is operating in the byte mode, the Bit-counter is preset to 8, producing the "carry" pulse after only eight CL B pulses; otherwise it is generated after 16. In FB mode an additional gate keeps the MSB of the decoder "low", allowing the addressing of the 8 LSB's of the memories, although the counter is running from 8 to 15.

When the Bit-counter is counting down, it is preset to 15 as an initial value and the "borrow" pulse, instead of the "carry" one, becomes the "end" pulse. This allows the writing of information starting from the MSB.

In "integer multiplication", the "carry" indicates the end of the addition cycle, rather than the end of the operation. It increases the Cycle-counter (counter-to-8), resets D_A and shifts AC1 to the left.

In "floating multiplication" the addition cycle lasts 24 bits. The MPY FF and the 8-detector extend to 24 the range of the Bit-counter. After 8 cycles, the MPY is completed. The Cycle-counter overflows and it produces an "end" pulse. The MPY CL pulses are produced whenever both Bit and Cycle-counters, have the same value. A magnitude comparator is used to detect that condition (Fig. 5.7).

The Fig. 5.9 illustrates the output of the Clock Unit, the pulses produced by the Sequencer and the value of the three counters during the preset and the operation cycle. It should be noticed that the preset cycle is omitted in some functions and that the operative cycle can start with the Bit-counter in 0 or 8. The pulses drawn in continuous lines correspond to an integer operation, while the pulses drawn in dotted lines correspond to alternative modes of operation.

5.4 The Cell Addressing System (CAS)

Although the control bus sends the same signal to all the cells of the array, there are operations in which a particular cell (or cells) is involved. These operations are:

- Computer-cell communication operations
- Buffer-cell communication operations
- Half array operations
- Specific cell, row or column operations.

The CAS has been designed such that it can address, in a 63x63 cell matrix:

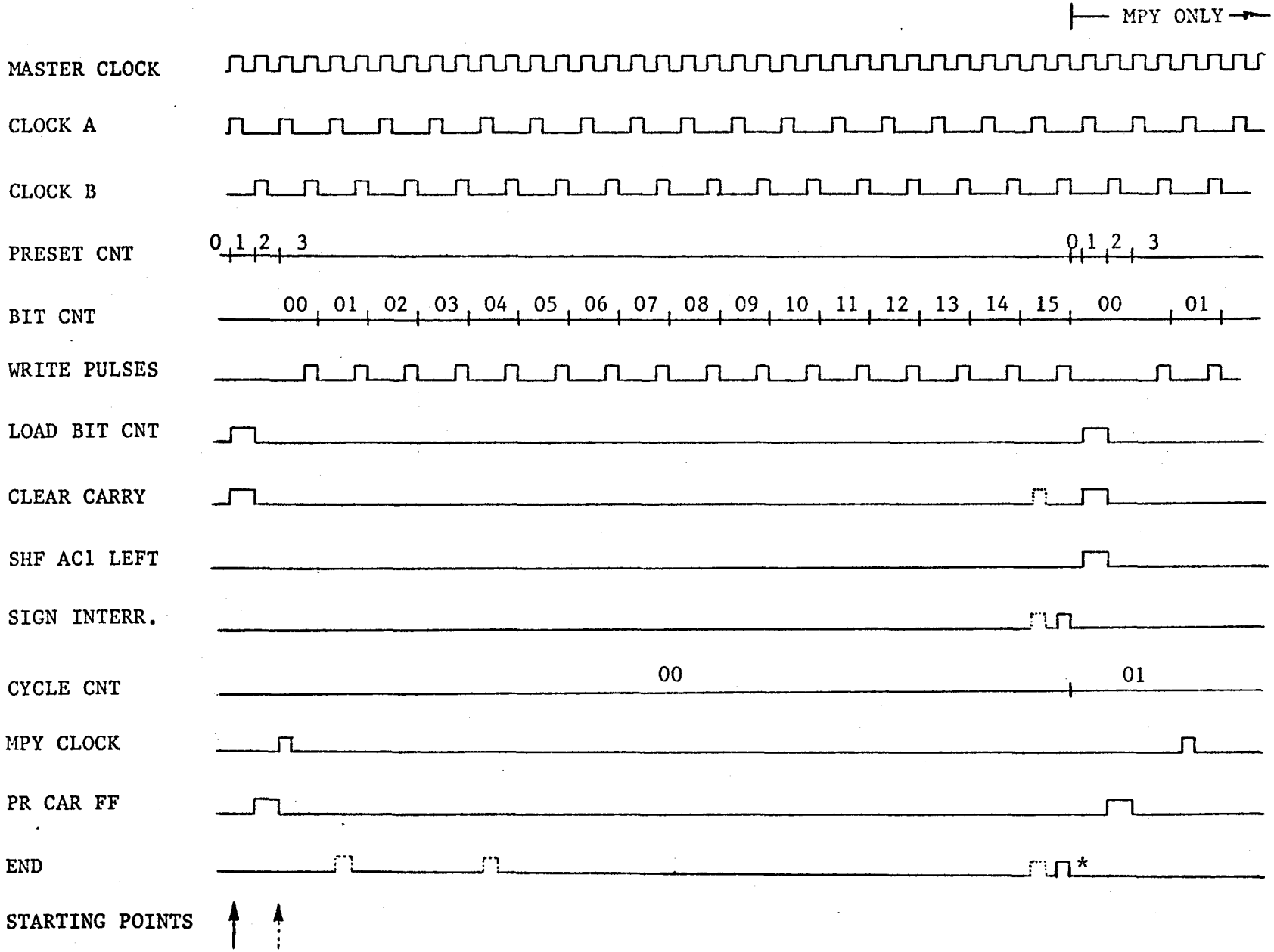


FIGURE 5.9. PULSE TRAINS AND STATE OF COUNTERS

Nb. * Except in MPY
—— Integer operation
····· No integer operation

a) a cell

b) a row

c) a column

d) all the even rows

e) all the odd rows

f) all the cells of the array.

For cell selection, the array can be considered as an 8x8 matrix of sectors. Each sector is composed of a sub-matrix of 8x8 cells, except the sectors of the left edge, upper edge and upper left corner, which are composed of sub matrices of 7x8, 8x7 and 7x7 cells respectively (Fig. 5.10).

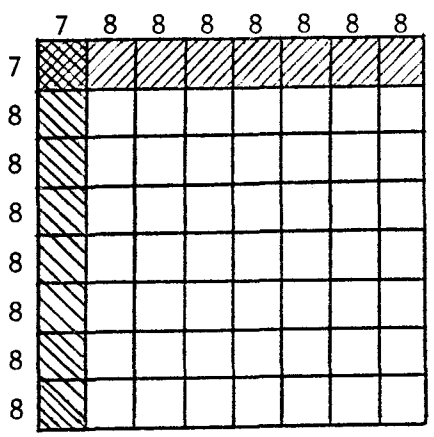


FIG. 5.10

The CAS consists of:

a 12-bit Basic Address Register (BAR)

a 12-bit Cell Mode Register (CMR)

a 6-to-64 Row Decoder

a 6-to-64 Column Decoder

a Cell Enable Input (CEI) circuit in each cell

a Cell Enable Output (CEO) circuit in each cell

The Row and Column Decoders select the Y and X coordinates, respectively. A pair of these values (i,j) defines a unique point in a 64x64 matrix as is shown in Fig. 5.13. If the first row (0) and the first column (0) of this matrix are not considered, a 63x63 matrix is obtained, in which each point of it, represents a unique cell of the array. When a "0" is selected by the Column Decoder, any point of the set $(0,Y_j)$ can be selected by the Row Decoder. That addresses all the cells of the "j" row simultaneously. Similarly, for addressing a complete "i" column, a "0" is selected by the Row Decoder and the "i" column by the Column Decoder. If the point (0,0) is selected, all the cells of the array are enabled to operate and information can be sent to all of them simultaneously.

The Basic Address Register is an index register that selects a cell (or cells) by a combination of its contents and the IR contents. It also can address a cell directly. The BAR is composed of two 6-bit counters, with preset and count up/down capabilities. One is the Row Counter and the other one, the Column Counter. It is loaded directly from the CPU. The 2-bit Cell Mode Register controls the set of cells in which any ALO is performed. It can assume any of the following four values:

0; operation in all the cells

1; operation in the cell (or cells) selected by the BAR

2; operation in all the even row cells, and

3; operation in all the odd row cells.

The Cell Enable Input circuits in each cell allow the cell (or cells) selected by the CAS to change the contents of its memory bank with the answer of an operation or new data. To inhibit the change of the contents of the memory bank in a cell is equivalent to inhibit its operation. The CEI also allows the broadcasting of new control bits.

The Cell Enable Output circuits in each cell allow the output of the cell selected to flow either to the CB or to the row or column buffers.

5.4.1 Computer-Cell Communication Operations

It is necessary to distinguish clearly between sequential and parallel operation. The PDP-11 is a sequential machine if a word is considered as a unit of information, but it is a parallel machine from the 16 bits point of view. That means that although the 16 bits are processed and, most important for our purpose, are available in the Unibus [15] simultaneously only one word is present at a time. Analogously, the computer can read into its registers 16 bits at a time, but word by word.

On the other hand, the array is a highly parallel processor, but each cell is a serial machine. Although all the cells operate simultaneously, each cell processes information in serial fashion,

bit by bit.

Obviously some incompatibilities are involved whenever communication between the computer and the array is required. This problem is solved using a Data Register (DR). It acts as a temporary storage of information for parallel-to-serial, serial-to-parallel conversion of the word. It is a shift register which is loaded in parallel with data or control bits from the CPU. From there, changing the state of its control line, serial read out is obtained at the output and the data are broadcast to the EXT input of the MPX C's of all the cells of the array. But the CEI circuits in each cell allow the information to reach the memory bank only in the cell (or cells) that has been selected by the CAS. On the other hand, the DR receives serial information from the cell selected by the CEO circuits. Information remains there, available in parallel, for further transferring to the CPU.

In addition, the DR is also used as a central buffer for arithmetic-logic operations, as has been described in Chapter 2.

Because the DR is acting as an intermediate between the CPU and the array, it has been included as a part of the Interface Unit, rather than that of the CU.

The CCC instructions have the format shown in the corresponding part of the Table 4.1. When a CCC instruction is detected in the IR, three addressing alternatives are possible, according to the Address Mode selected. They are:

1. DIRECT ADDRESSING. The cells of tht first sector (double shaded in Fig. 5.10) are selected directly by the instruction. Three

bits for the row and three bits for the column permits one to select any point of an 8x8 matrix. But because lines X=0 and Y=0 have a special purpose, the set of cells directly addressed is reduced to a 7x7 matrix. This mode permits a faster addressing in that set.

2. CONCATENATION ADDRESSING. The three LSB's of the row and column decoders are controlled by the current instruction, while the 3 MSB's are controlled by the BAR. This is very useful because the 3 MSB's of the BAR select the sector. Once the desired sector address is set in the BAR, it is possible to address a particular cell in this set directly by one instruction, in similar fashion to that in direct addressing.

3. AUTOMATIC ADDRESSING. The address of the cell is taken directly from the BAR. It is set initially to the starting address. From that point, the address is automatically increased or decreased by 1 in either row or column direction or both.

This mode of addressing is very useful whenever loading of successive points of the array are required. For example, if it is required to load some boundary conditions for solving Laplace's equation, the BAR is set to the starting address of A (Fig. 5.11). By setting in the IR the instruction "Automatic-Column Up", cells of the same row are addressed, step by step, until a point B is reached. The number of steps is controlled by the number of pulses sent by the CPU. A new instruction, "Automatic-Row Down" is loaded and the

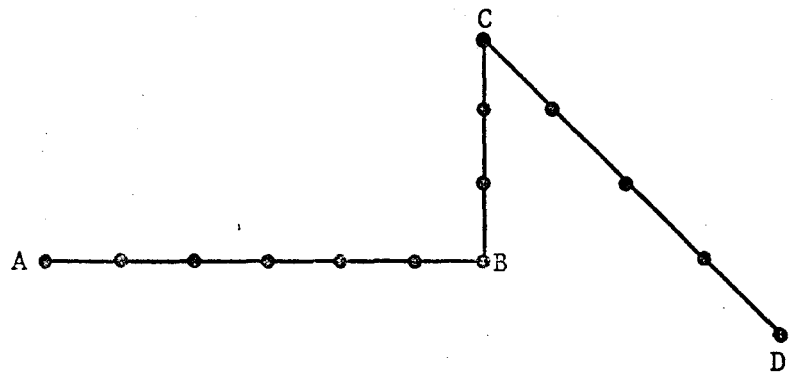


FIGURE 5.11. AUTOMATIC ADDRESSING.
LOADING OF BOUNDARY CONDITIONS

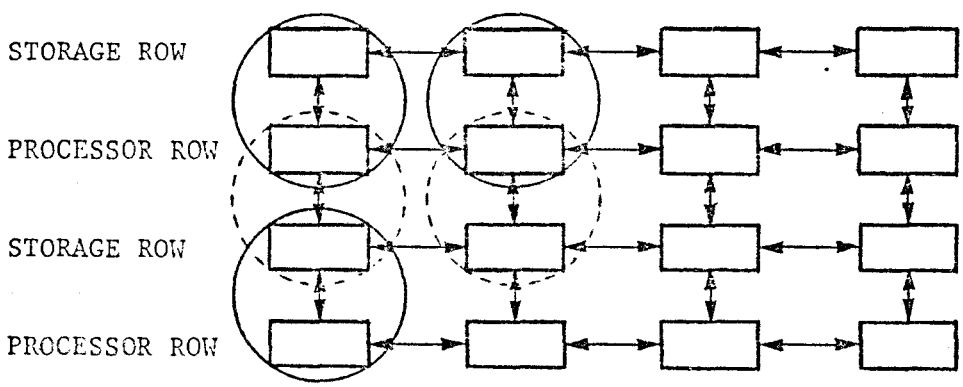


FIGURE 5.12. HALF ARRAY OPERATIONS

cells of that column are sequentially addressed until the point C is reached. Finally, an instruction "Automatic-Row Up-Column Up" is loaded; and, automatically, all the cells between the points C and D are sequentially enabled.

An interesting feature is used. When the Row (or Column) selector reaches "63" it jumps to "0" at the next step and the Column (or Row) selector is increased by "1". An exploratory scanning is performed automatically and it allows the loading or reading out of the whole array.

If DIN or DOU operations are performed, whatever is the mode for cell addressing being used, the word selection is controlled in similar fashion to that in the ALO; i.e., DMPX for loading and MPX B for reading out.

5.4.2 Buffer-Cell Communication Operations

The aim of the buffer-cell communication operations is to transfer information from a specific word of an entire row or column to a specific word of the corresponding buffers and vice versa.

In the instruction Transfer to Buffer (TTB) the word address of the source and the destination are specified as in any double-address instruction, but in addition, the row or column number must be specified too. This is carried out by specifying the desired row or column in the BAR. If a row is transferred, the row counter of the BAR is loaded with the row number and the column counter of the BAR is loaded with "0". If a column is transferred, the column counter of the BAR is loaded with the column number and the row counter of the

BAR is loaded with "0". The CEO circuits enable the cell output to reach the buffers only in the cells of the row or column selected by the CAS (Fig. 5.15). By setting CMR=1, the CAS selects the cell or cells specified by the BAR, i.e., the row or column selected.

The Transference from Buffer (TFB) is a MOV operation in which the MPX C selects the ROW or COL input. The row or column selection is done in similar fashion as that for TTB. The CAS, operating over the CEI circuits allows the transfer only to the selected cells.

5.4.3 Half Array Operations

For some operations in which the 15 word capacity of a cell is not enough, it is possible to extend the range to a 30-word memory, joining two cells together. One would be the processor cell and the other one would be used exclusively for storage. Obviously, the number of processors available would be reduced to half. This can be done assuming that the array is divided into "processor" and "storage" rows. Two neighbor cells, one in the storage row and another in the processor row can be considered as a unit or "pseudo-cell" as is shown in Fig. 5.12 in solid circle. The data flow from storage to processor is done in a similar way to that in normal operation, through the MPX C but the CEI circuits in the storage cells must be inhibited. This will prevent the transfer from processor to storage row to the neighbor "pseudo-cell". Analogously, when it is wished to transfer information from the processor to the storage cells, the CEI in the processor cells must be inhibited. When an ALO is performed, only the processor cells are enabled.

The operations are controlled by the 2-bit CMR. Usually it is in "0" and the ALO instruction fetched in the IR is performed in all the cells of the array. When it is in "2", the operations are performed only in the even rows (the odd rows are inhibited) and in "3" only the odd rows are enabled (the even rows are inhibited).

Notice that the row that is considered as a storage can be switched to processor and vice versa, according to the instruction and the contents of the CMR. Similarly, the structure of the "pseudo-cell" can be changed joining a processor row with tht other storage row neighbor, as is shown by dotted circles in Fig. 5.12.

5.4.4 Specific Cell, Row or Column Operations

The CMR gives a great deal of flexibility to the programmer. When its contents are "1" only the cell (or cells) specified by the BAR is enabled. This allows the user to perform ALO in only one cell or one row or column, as is required in certain sequential algorithms.

The CMR=1 is also used for transferring information between the buffer and the cells, as has been described earlier.

5.4.5 Cell Addressing Hardware

The Fig. 5.13 illustrates the configuration for the CAS. The BAR and the CMR are loaded together, as a unique register, directly from the Unibus of the PDP-11 using the same interface techniques as for the IR. When a CCC instruction is detected in the IR, bits 8-9 indicate the addressing mode, bits 10-12 the row selected and bits 13-15 the column selected.

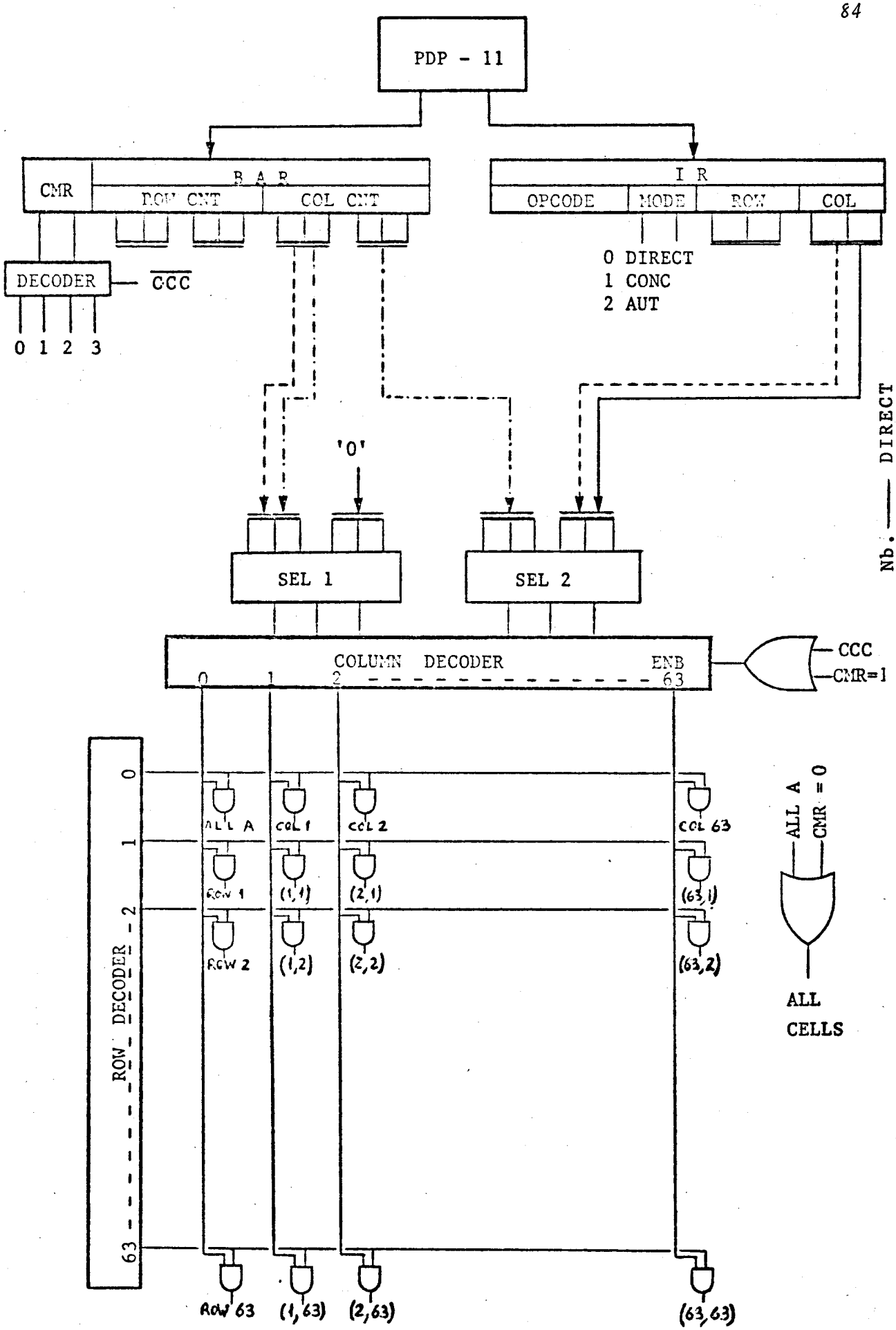


FIGURE 5.13. CELL ADDRESSING CIRCUITS

Because of the similarity between the COL SEL and the ROW SEL systems, only the former one is described (Fig. 5.14). When bits 8 and 9 are equal to "0", direct addressing mode has been selected and bits 13-15 are able to reach the 3 LSB's of the decoder, through the SEL 1. At the same time, the 3 MSB's of the decoder are set to "0", through the SEL 2.

If concatenation addressing mode is selected, the 3 LSB's are still controlled by bits 13-15 of the IR, but the control of the 3 MSB's is switched to the 3 MSB's of the COL CNT.

If automatic addressing mode is selected, the 6 control lines of the decoder are tied to the output of the BAR. The counters are able to receive start clock pulses from the CPU. Bits 14-15 have a control function. Whether the pulses reach the particular counter or not, is up to the state of bit 14, while bit 15 controls the counting direction.

For performing an exploratory scanning of the array, a "carry" of the ROW CNT produces an increment in the COL CNT. Similarly, when ROW CNT is counting down, the "borrow" pulse decrement by "1" the contents of the COL CNT.

Once both decoders have selected the proper line, a unique point is determined by the intersection of these lines. A matrix of AND gates performs this task and a 64x64 selection matrix is finally obtained (Fig. 5.13).

The Cell Mode Register is a single 2-to-4 decoder with outputs CMR=0; CMR=1, CMR=2 and CMR=3. When CMR=1, any array operation should be performed in the cell (or cells) selected by the BAR (as in AUT mode).

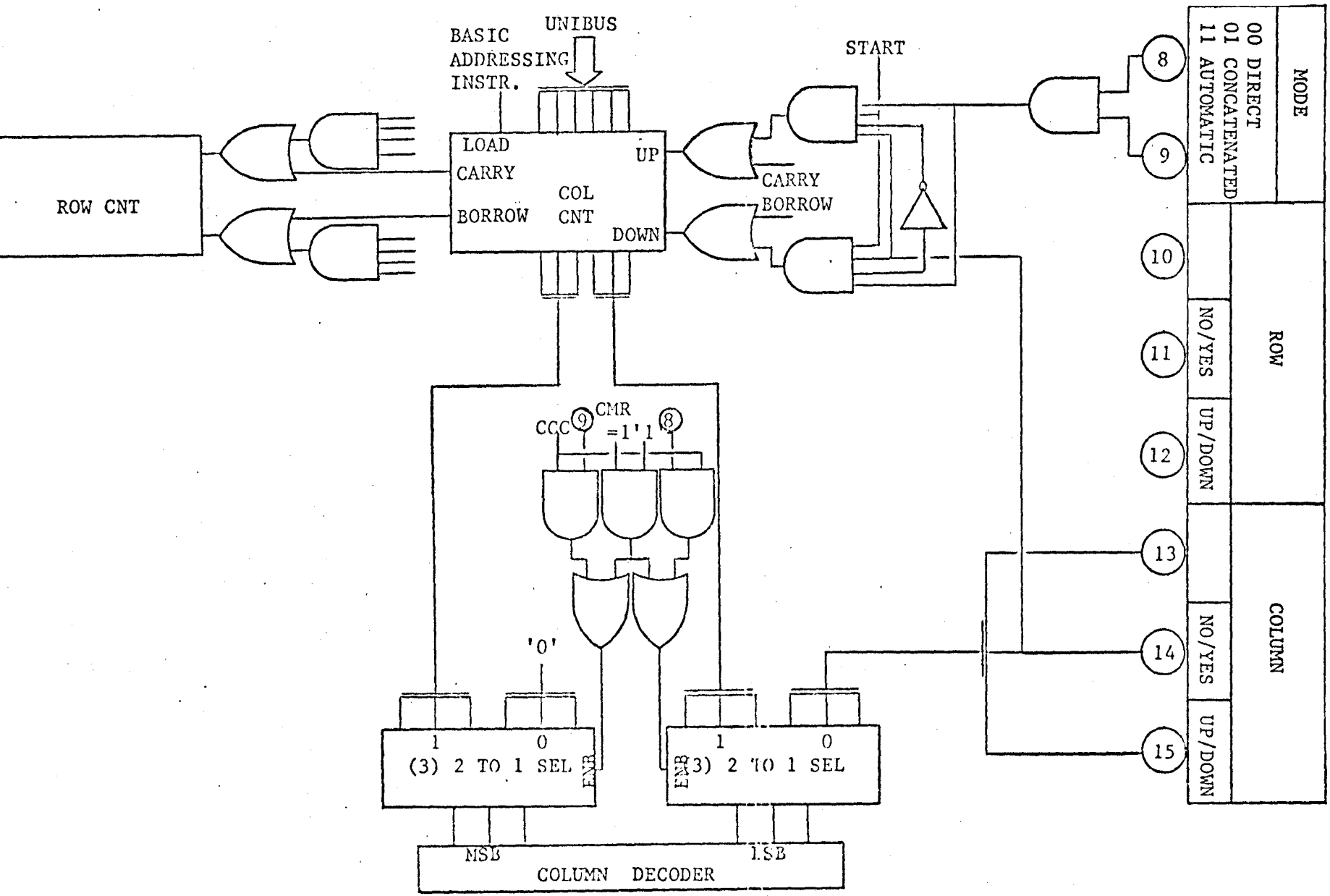


FIGURE 5.14. CELL ADDRESSING SYSTEM

Both (3) 2-to-1 selectors are set to "1" by the corresponding gates allowing the output of the BAR to control the decoder.

When CCC operations are performed, the cell selection should be independent of the CMR; hence its decoder is not enabled.

If neither CCC operations, nor operations with CMR=1 are being performed, the decoders are inhibited and no selection through the selection matrix is carried out. In that case, the cells are chosen in accordance to the CMR contents only, as follows:

For CMR=0, "All Cells" line is activated and all the cells are enabled.

For CMR=2, "Even" line is activated and all the cells that belong to an even row are enabled.

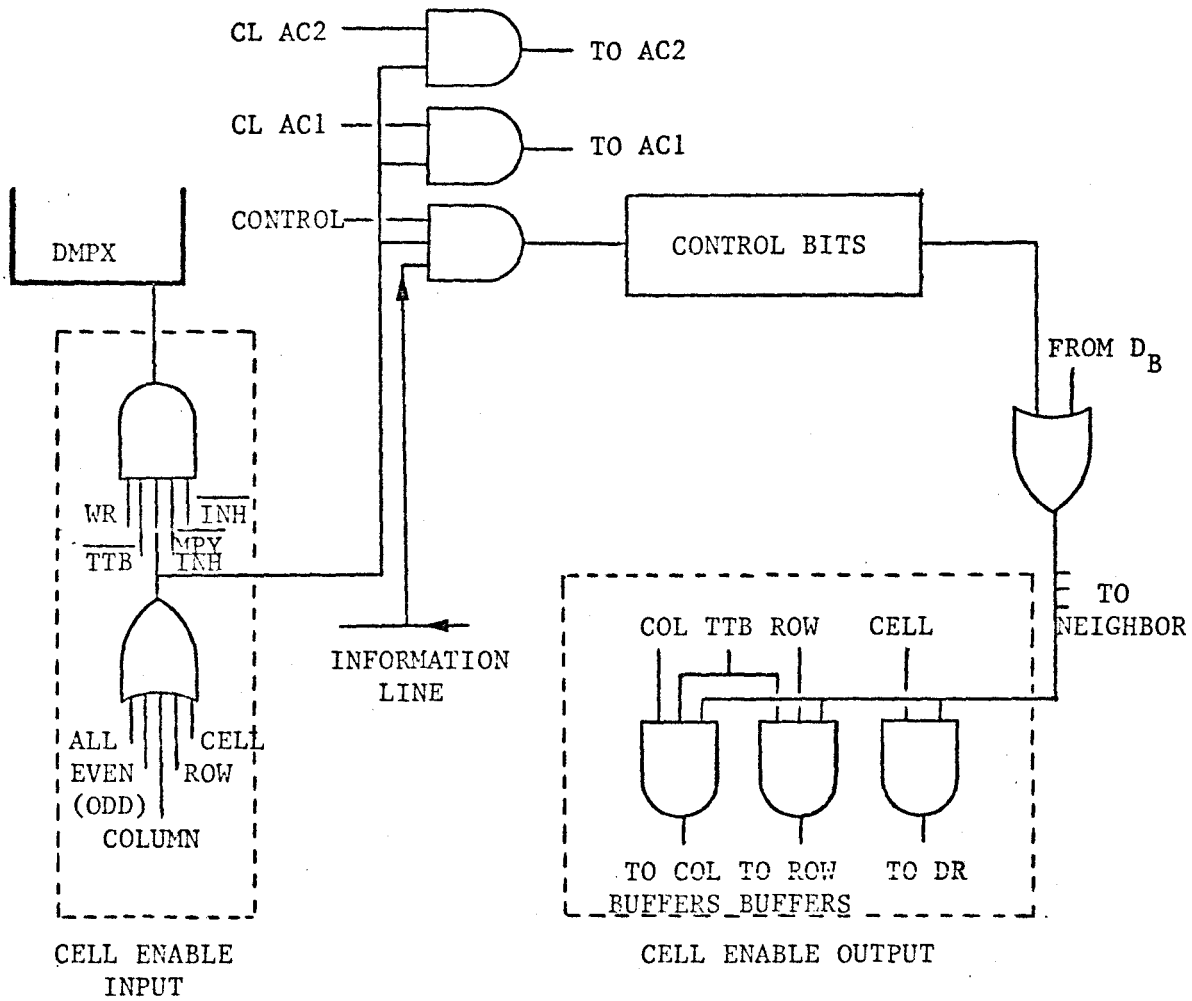
For CMR=3, "Odd" line is activated and all the cells that belong to an odd row are enabled.

5.4.6 Cell Enable Circuits

With all the previous considerations we conclude that a cell (X_i,Y_j) is selected for an arithmetic-logic or transfer operation whenever the cell itself, the corresponding row or column, all the cells of the array or all the even (odd) cells have been selected.

On the other hand, the unique DR can receive information from only one cell at a time. So the output to this register is done on the basis of a unique cell selection, while in the TTB operations, a complete row or column is selected.

The table of Fig. 5.15 shows the input and output conditions, the corresponding point in the selection matrix and the value that



	SELECTED	POINT	CMR
INPUT	CELL (X_i, Y_j)	i,j	1
	ROW Y_j	0,j	1
	COLUMN X_i	i,0	1
	ALL CELLS	x x	0
		0,0	1
	ALL EVEN (ODD)	x x	2 (3)
OUTPUT	CELL (X_i, Y_j)	i,j	1
	TTB ROW	0,j	1
	TTB COLUMN	1,0	1

FIGURE 5.15. CELL ENABLE CIRCUITS

should be assigned to the CMR. It should be remembered that in CCC operations, the selection is made solely on the basis of the selection matrix, independent of the CMR.

Each of the five input conditions correspond to an enable line selected. The five lines are OR-ing in each cell in the corresponding CEI circuit, as is shown in the Fig. 5.15. When the cell is selected, through any of its five enable lines, the "Write Pulses" can reach the DMPX, the accumulators can be clocked and control information can reach the control bits register. If none of the five lines are activated, the cell is locked and no possible change of its information is allowed.

The Cell Enable Output (CEO) circuits control the flow of information of the output of each cell.

Because only one cell has actually been constructed, the addressing system is not built, but it has been designed as an example foreseeing the construction of a 63x63 cells array (3,969 cells). Further expansion can be easily obtained by simply using a larger BAR.

CHAPTER 6

The Interface

6.1 Generalities

All communication between the PDP-11 and any external device is accomplished by a single high-speed bus called the Unibus [15]. The Unibus is composed of 56 lines (51 are bidirectional) - 16 data lines, 18 address lines and 22 control, synchronization, priority transfer and miscellaneous lines. All flow of information between a device logic and the Unibus is through the registers. Each register is assigned a bus address at which the CPU can interrogate and/or load.

From the PDP-11 point of view the array is an external device with four registers (Fig. 6.1). A device register address has been assigned to everyone of them, in accordance with the Digital Equipment Corporation (D.E.C.) specification. They are:

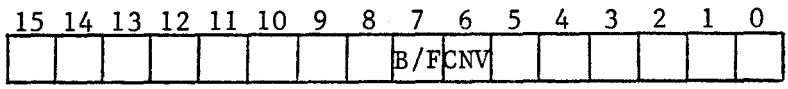
First 16 bits of the IR	(IR1)	#167770
Second 16 bits of the IR	(IR2)	#167772
Data Register	(DR)	#167774
Device Status Register	(DSR)	#167776

The instruction registers are write-only registers; the DSR is a read only register and the DR is a read/write register.

The IR is a 32-bit parallel register in which the coded instruction is loaded from the CPU and from which the ID takes the information to be decoded.

The DR is a 16-bit register that acts as an intermediate serial-to-parallel, parallel-to-serial data converter, as well as the CB, as has been described in Chapter 5.

The DSR is a 2-bit register that contains information on the Busy/Free and convergence consition of the system, according to the following format:



Both CNV and B/F flip-flops have been included in the diagrams of the CU, in Fig. 5.5 and 5.6. Whenever it is required by the program, the CPU can interrogate the DSR for further actions.

In addition to the registers and the Unibus, the interface is composed of an Address Selector D.E.C.-M105, a Gating Control circuit and a Driver to the Unibus, as is shown in Fig. 6.1.

6.2 The Address Selector D.E.C.-M105

The address selector is used to provide gating signals for up to four device registers. The selector decodes the 18-bit bus address, where:

A <00> is used for byte control.

A <02:01> is decoded to provide one of four addresses.

A <12:03> is determined by jumpers in the unit.

A <17:13> must all be in 1's.

In our case, the jumpers have been set in 0777, so that one of the four selection lines is activated whenever the CPU sets in the address bus the corresponding register addresses.

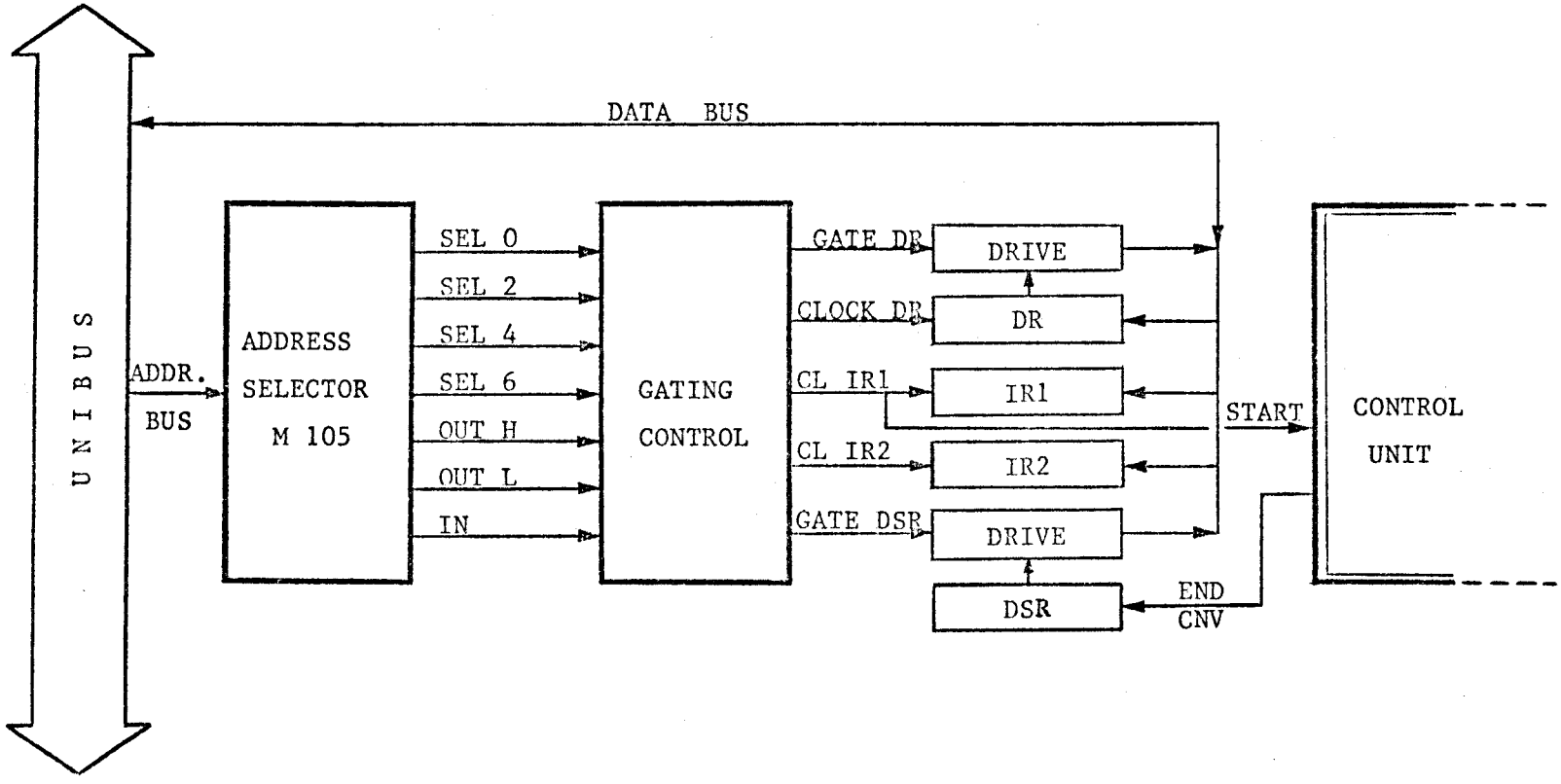


FIGURE 6.1. INTERFACE UNIT.
BLOCK DIAGRAM

Four select signals (SEL 0, 2, 4 and 6), corresponding to the four device registers are supplied by the selector. In addition, it also supplies three gating control signals:

OUT HIGH; which permits the loading of the high byte data bus.

OUT LOW; which permits the loading of the low byte data bus.

IN; which permits the reading of the register on data bus.

6.3 The Gating Control Circuits

The gating control circuits take the output of the address selector and provide the clock pulses to load the selected write register (IR1, IR2 or DR) with the value present in the data bus. It also supplies the gating signals to read the selected read register (DSR or DR) on the data bus.

The same pulse used to clock the IR1 is also used as the "start" pulse. It changes the state of the Busy/Free flip-flop.

6.4 The Driver to the Unibus

The driver to the unibus has the task of maintaining the transmission-line characteristics of the Unibus. Information transmitted on the bus must be driven by open-collector drivers.

The Fig. 6.2 shows the interface board. The Address Selector D.E.C.-M105 has not been included. It is an additional card supplied by Digital Equipment Corporation. If, in the future, the Cell Address System is constructed, a new register (BAR-CMR) would be necessary to be added at the interface.

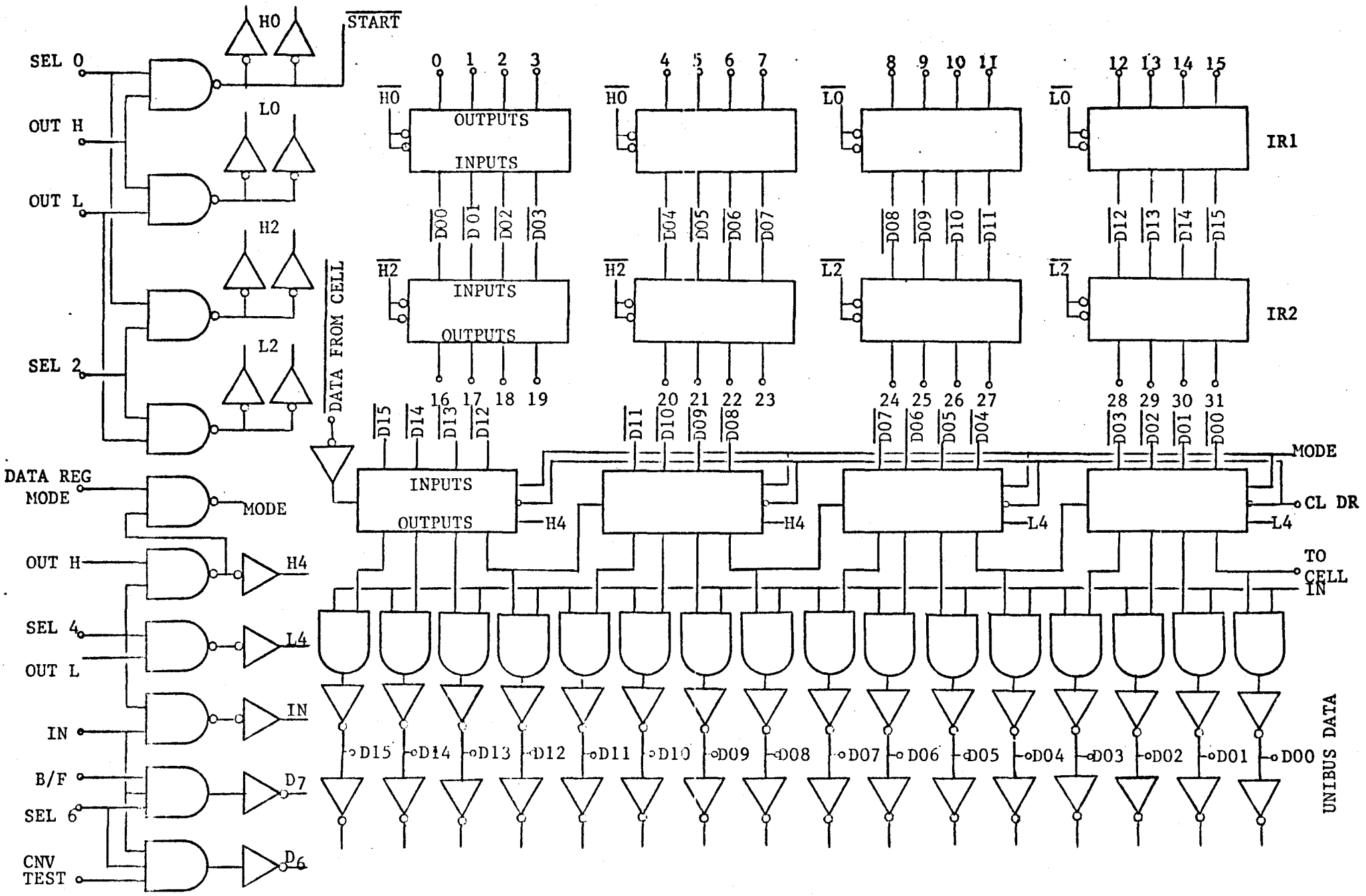


FIGURE 6.2. INTERFACE UNIT CIRCUITS

CHAPTER 7

Programming the System

7.1 Control Program

The PDP-11, acting as a control element, has been programmed to run the array. Several subroutines which are called from a main program allow the user to load the registers with the corresponding instruction (instruction register) and data (data register). The subroutines also test the state of the DSR and give operative commands, using simple mnemonic instructions. All the subroutines are grouped in a program called CONTROL and are written in PAL-11.

The array user can write the main program using very simple subroutine calls and the mnemonic language discussed in Chapter 4. The CONTROL program compiles the instruction, translating the mnemonic to the corresponding binary value of the 32-bit instruction register and storing it in a two words instruction buffer. It then remains in a waiting loop checking the DSR until the previous operation has finished. Once the B/F flip-flop of the DSR is in "free" state, the contents of the buffer are loaded into the instruction register and the array operation starts. While the array is performing the actual operation, the computer is compiling the next instruction.

The computer is also programmed to interrogate the convergence flip-flop of the DSR when iterative microprogramming is being performed.

In addition to the basic operations, several standard routines composed of many basic instructions have been grouped together under a

unique subroutine (e.g., division subroutine). This allows these subroutines to be called by a single instruction. All the subroutine calls by the main program are carried out through the same general register, R5.

Let us suppose the programmer wishes to add, in absolute value mode, (M_3) plus (M_5) of the north cell, store the answer in M_7, inhibit the operation if (AC1) < 0, not check convergence, and clear the carry. The instruction in the main program should therefore be:

```
JSR 5, SUBADD                    ; calling the addition subroutine
M7, M3, M5, N, AV, 0, NS, 0, 0
```

Subroutines to load entirely the cell, to clear the cell, to unload the cell into the output buffer, and to dump the content of the output buffer into the teletype are also provided in the CONTROL program.

7.1.1 Microprogramming

As it has been pointed out earlier, division, floating point operations and transcendental functions can be computed entirely in the cell under microprogramming control. This is also a task of the CONTROL program.

It should be considered that the same algorithms used in sequential computers cannot be applied directly in a parallel processor, due to the need of operating with several sets of data under a unique control. Several parallel algorithms have been developed and tested. They are described in the following sections.

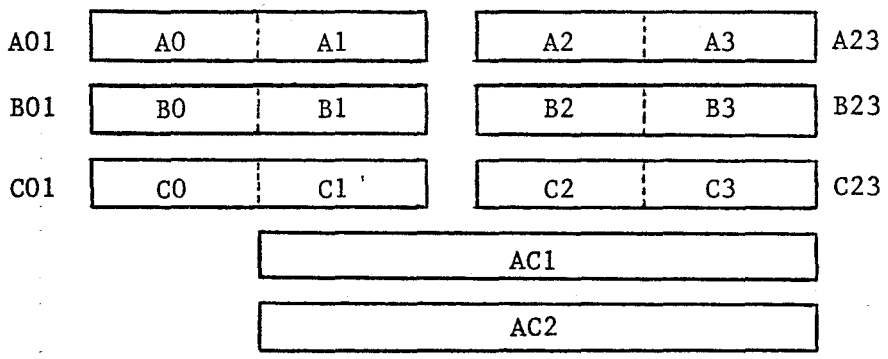
7.1.2 Integer Division

The set of instructions necessary to divide $M_A \div M_B = M_C$ is shown in Fig. 7.1. The iterative repetition of loops are under micro-programming control and the corresponding instructions have not been shown in the figure. The opcode mnemonic has been used, but in the real program, a subroutine call should be used instead. The algorithm shown in the flowchart of Fig. 3.3 has been used. The contents of M_A are lost after the operation, while M_B remains unaltered.

7.1.3 Floating-Point Operations

A floating-point number is represented by 4 bytes (2 words). One of them represents the exponent and sign of the mantissa and the other three the mantissa absolute value.

In a floating-point operation, 6 words, representing 3 floating-point numbers, and both accumulators are involved, according to the following scheme:



In any floating-point operation, the accumulators are used as a temporary register and therefore, their original contents are lost during an operation.

```
        EXO,M2,MA,MB            ;SET SIGN IN AC2
        SUB,MC,M15,M15          ;CLEAR MC

        MOV,M1,MA               ;CONVERSION TO ABSOLUTE VALUE
        TCM,MA,M1,PS
        ADD,MA,MA,MA
        MOV,M1,MB
        TCM,MB,M1,PS

        MOV,M1,MB               ;NORMALIZATION OF MB
15      SHF,AC1L,OD
times   INC,MC,MC,OD
        SHF,AC1R
        MOV,MB,M1

        SUB,M1,MA,MB            ;DIVISION LOOP. IT STORE THE
        MSG                     ;INVERTED ANSWER IN AC2
        MOV,MA,M1,NS
15      DEC,M1,MC
times   MOV,MC,M1
        SSR,MB,MB,YZ

        ROT,AC2R                ;SET PROPER NUMBER OF 0's IN
        AND,MC,M2,MC            ;THE SOLUTION
        MOV,M1,MB
        SHF,AC1L
        ADD,M2,M2,M2,YZ
15      SHF,AC1R
times   MOV,M1,M1

        MAI,MC                  ;STORE ANSWER IN MC (IN ABSOLUTE
                                ;VALUE)
        MOV,M1,MC               ;CONVERT ABSOLUTE VALUE TO
        TCM,MC,MC,AV,PS         ;2's COMPLEMENT
```

FIGURE 7.1. INTEGER DIVISION ROUTINE (MA ÷ MB = MC)

ADDITION. The addition algorithm consists of equating both exponents with the value of the larger and then, adding the mantissas. In adding A + B = X, one addend is saved while the other is lost after the addition. The steps are:

a) put the larger exponent addend in X,

b) put the smaller exponent addend in A,

c) convert X and A to 2's complement representation,

d) align exponents,

e) add mantissas, and

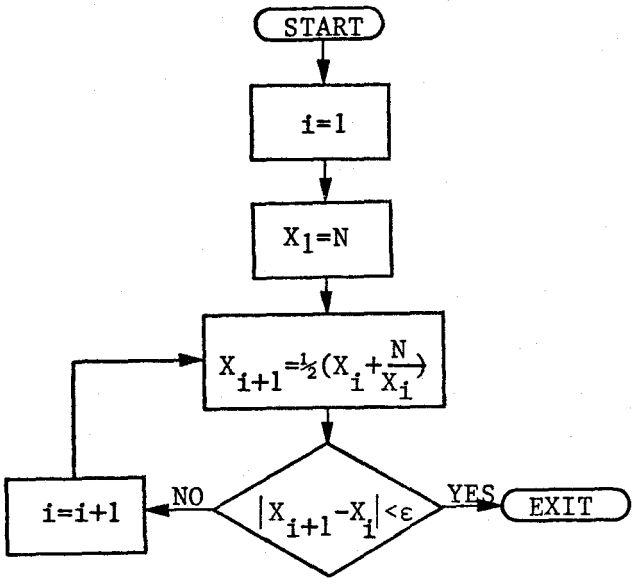
f) convert 2's complement answer into SAV representation.

MULTIPLICATION. The multiplication is carried out by adding exponents together and then performing a multiplication of mantissas. Floating multiplication hardware performs 2 byte x 1 byte multiplication, storing the 3 bytes answer in AC2. Successive shifting, additions and multiplications allow 3 byte x 3 byte multiplication. However, only the 3 most significant bytes of the answer are kept. Both operands are saved and the answer is obtained in AC1 (exponent and mantissa sign) and AC2 (mantissa absolute value).

DIVISION. The division is accomplished by subtracting the exponent of the divisor from the exponent of the dividend, and dividing mantissas. The divisor is lost after the operation.

SQUARE ROOT. The square root algorithm uses Newton's method in which an initial approximation is made and then each succeding approximation is calculated. The routine exits when the desired convergence

between two successive approximations in all the cells has been obtained. The following flowchart illustrates the routine:



EXPONENTIAL. The exponential routine uses the identities:

For X > 0

$$e^x = 2^{x\log_2 e} = 2^{N+F} = 2^N \cdot 2^F \tag{7.1}$$

where N is an integer; 0 < F < 1, and

$$2^F = 1 + \frac{2F}{A - B + BF^2 - \frac{C}{D+F^2}} \tag{7.2}$$

where A, B, C, D are fixed parameters.

For X < 0

$$e^x = \frac{1}{e^{-x}} \tag{7.3}$$

LOGARITHM. The logarithm routine uses the identities:

$Z = \log X \qquad X = 2^N F \qquad ; \qquad 1 < F < 2$

$$\therefore \quad \log X = N \log 2 + \log F \tag{7.4}$$

If $Y = F-1 \quad (0 < Y < 1)$

$$\therefore \quad \log F = \log (1+Y) = \sum_{n=1}^{8} Ai\ Yi \tag{7.5}$$

where Ai are fixed parameters.

SINE. To calculate sin α the steps are as follows:

Reduce α to α_1	$0 \leq \alpha_1 \leq 2\pi$
Reduce α_1 to α_2	$0 \leq \alpha_2 \leq \pi/2$
Consider sign α	$-\pi/2 \leq \alpha_3 \leq \pi/2$
Compute $Y = \frac{2\alpha_3}{\pi}$	$-1 \leq Y \leq 1$

Compute

$$\sin \alpha = C_1Y_1 + C_3Y^3 + C_5Y^5 + C_7Y^7 \tag{7.6}$$

where C_1, C_3, C_5 and C_7 are fixed parameters. This routine is shown with more detail in Figs. 7.2 and 7.3. The flowchart of Fig. 7.2 shows how the inhibit and convergence capabilities of the cell are widely used to reduce the original argument α to $-\pi/2 \leq \alpha_3 \leq \pi/2$. The Fig. 7.3 shows the successive steps in calculating sin α once α_3 is known, and the mapping of the memory words in each step. The final answer is obtained in M_{11} and M_{12} and the original value of α is saved in M_3 and M_4. More precision can be obtained by adding terms to the formula.

COSINE. The following identity is used:

$$\cos \alpha = \sin(\pi/2 - \alpha) \tag{7.7}$$

and hence using the sine routine.

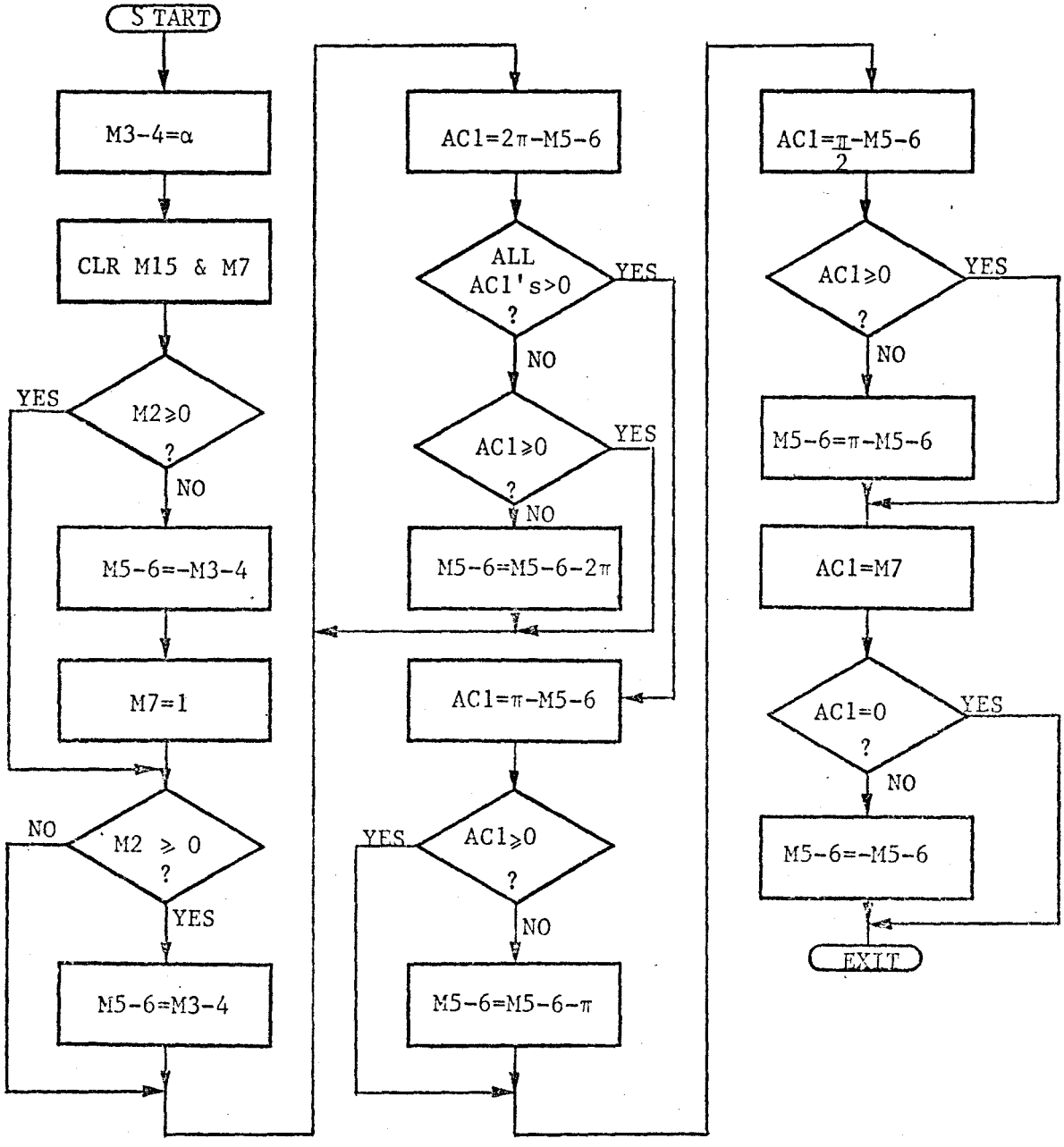


FIGURE 7.2. SINE ROUTINE. FLOWCHART

	M3-4	M5-6	M7-8	M9-10	M11-12	M13-14	M15
INITIAL VALUE	α						
AFTER REDUCING	α	α_3					
MOV,M7-8,2/π	α	α_3	$2/\pi$				
MPY,M9-10,M5-6,M7-8	α	α_3	$2/\pi$	Y			
MOV,M7-8,C_1	α	α_3	C_1	Y			
MPY,M5-6,M7-8,M9-10	α	C_1Y	C_1	Y			
MOV,M7-8,M9-10	α	C_1Y	Y	Y			
MPY,M11-12,M7-8,M9-10	α	C_1Y	Y	Y	Y^2		
MPY,M7-8,M12-13,M9-10	α	C_1Y	Y^3	Y	Y^2		
MOV,M11-12,C_3	α	C_1Y	Y^3	Y	C_3		
MPY,M13-14,M12-13,M7-8	α	C_1Y	Y^3	Y	C_3	C_3Y^3	
ADD,M11-12,M13-14,M5-6	α	C_1Y	Y^3	Y	Σ_{13}		
MPY,M5-6,M7-8,M9-10	α	Y^4	Y^3	Y	Σ_{13}		
MPY,M7-8,M5-6,M9-10	α	Y^4	Y^5	Y	Σ_{13}		
MOV,M5-6,C_5	α	C_5	Y^5	Y	Σ_{13}		
MPY,M13-14,M5-6,M7-8	α	C_5	Y^5	Y	Σ_{13}	C_5Y^5	
ADD,M5-6,M13-14,M11-12	α	Σ_{135}	Y^5	Y	Σ_{13}		
MPY,M11-12,M7-8,M9-10	α	Σ_{135}	Y^5	Y	Y^6		
MPY,M7-8,M11-12,M9-10	α	Σ_{135}	Y^7	Y	Y^6		
MOV,M11-12,C_7	α	Σ_{135}	Y^7	Y	C_7		
MPY,M13-14,M11-12,M7-8	α	Σ_{135}	Y^7	Y	C_7	C_7Y^7	
ADD,M11-12,M13-14,M5-6	α	Σ_{135}	Y^7	Y	Σ_{1357}		

FIGURE 7.3. SINE ROUTINE.
MEMORY MAPPING

7.2 Simulation Program

Having built one cell, it was possible to test its behaviour, as well as the performance of the algorithms described earlier. This was done using the CONTROL routines and several test programs. However, in order to test the full capabilities of the array such as the interconnection between cells, algorithms to solve problems involving matrix manipulations, etc., a complete array should be used.

Although economic limitations have made impossible the physical construction of such an array, it has been simulated in the PDP-11 computer by means of a program called SIMULA, which has been written in PAL-11. A 10x10 cell array plus 10 buffer cells, each with 18 computer words has been simulated and tested. Of the 18 computer words, 15 correspond to the 15 integer words of the cell, 2 to the additional extension of AC1 and AC2 and the other one contains the control bits and carry state of this cell. All the simulated cells share a common "central buffer" and a "convergence flip-flop" word. The arithmetic extended hardware facilities of our PDP-11 are used to perform multiplication, division and floating shifting at very high speeds.

SIMULA routines are called from a main program using the same instructions and mnemonic that would be used to call CONTROL routines. The program basically tests in every cell the inhibit condition of the instruction and compares it with the state of the corresponding control bit. If this cell should be inhibited, it jumps to the next cell. Otherwise, it finds the proper operands and the desired operation is performed. If convergence test is asked, SIMULA compares the answer with (M_{15}) in the specified range and the convergence flip-flop

is set or not set. Then it sets the control bits provided the destination word address is AC1. Finally, before SIMULA jumps to the next simulated cell, it stores the bits of the answer corresponding to the "mode" selected in the specified destination word. The same sequence is repeated, cell by cell until the last one is processed. At that point, the program starts again with the first cell, executing the next instruction.

The sequence used for the operation $M_Z = M_X + M_Y$ is illustrated, as an example, in the flowchart of Fig. 7.4. The same logic, with minor modifications is applied to any array operation.

Several routines which are part of the SIMULA program can load or unload the complete array, a complete cell, or a particluar word address in all the cells. A routine to dump the content of the whole array or the output buffer into the teletype has also been included in the program.

Several programs, running in conjunction with SIMULA have been tested. These include programs for general testing and programs to solve Laplace's equation and transpose matrices. The Laplace's equation program makes full use of the neighborhood relationship of the cells as well as the inhibit and convergence capabilities of the system. On the other hand, the matrix transposition program makes use of the row column/buffer as a temporary storage of information.

The simulation program and the control program are compatible in the sense that they can share a common main program. This proves that the performance of a real array can be identically imitated by

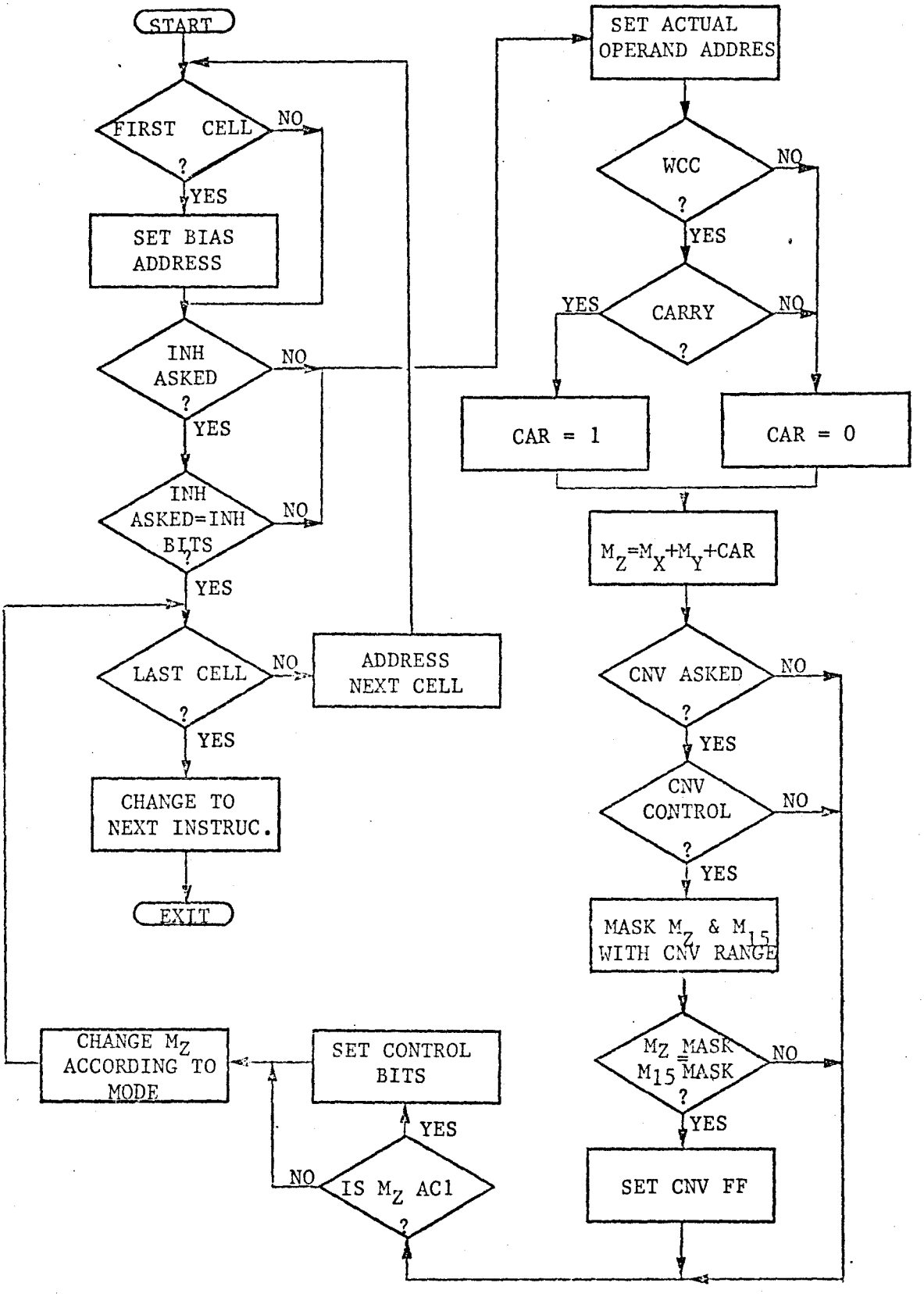


FIGURE 7.4. SIMULA PROGRAM. ADDITION FLOWCHART

using the simulation program. However, this assumption is not valid when the time factor should be considered. The simulation program is running in a sequential machine and the operations are performed "pseudo-cell" by "pseudo-cell".

All the tests realized by using CONTROL associated with several test programs have given cell performance in accordance with the specifications. Very good results have also been obtained for the programs that have been run in conjunction with SIMULA.

CHAPTER 8

Associative Memory

8.1 Introduction

The possibility of using the array as an associative memory system is discussed in this Chapter.

Alt [16] defines an associative memory as a storage device in which sufficient logic is associated with each word of memory to allow a parallel comparison of all the words in memory with a single search key. In other words, an associative memory is a system in which a word can be identified by its contents rather than by its address (i.e., content addressable capability).

The highly parallel computing system seems to be very adequate for that purpose because a set of words shares an ALU in which comparison can be carried out. Because only one cell has been constructed, no attempt to incorporate associative memory capabilities in the proposed array has been made. However, the following paragraphs will show how, with very little increase in hardware, this interesting feature can be incorporated into the system.

Although in this array a comparison between the search key and all the words of the array cannot be achieved simultaneously the search key can be compared with one selected word in all the cells. So following the Alt definition, this array is a set of 15 matrices of associative memories, with one matrix per word, rather than one

associative memory processor. Comparison between a search key and all the words of any of these matrices can be performed.

8.2 Search Key Comparison

The 16-bit search key is loaded into the DR and the comparison is done using the ALU and inhibit control bits. Some examples will illustrate this better:

Example 1. To detect the cells in which the contents of word 13 are equal to X_1 (search key).

The DR is loaded with X_1 and the instruction SUB, M1, EXT, M13; is performed by the entire array. In the cells in which the values are equal, the zero flip-flop will be set. All the words of the matrix "word 13" are simultaneously compared with the search key and the zero flip-flop indicates, in each cell, whether the required condition has been satisfied or not.

Example 2. To detect the cells in which $(M_7) > X_1$.

After loading X_1 in the DR, the instruction SUB, M1, EXT, M7 is performed. In the cells in which the condition $(M_7) >$ EXT is satisfied, the sign flip-flop will indicate negative.

Example 3. To detect the cells in which $X_1 < (M_5) < X_2$.

The search can be done in four steps.

1. Set X_2 in DR
2. SUB, M1, M5, EXT

3. Set X_1 in DR

4. SUB, M1, EXT, M5, 0, 0, 0, PS (inhibit if $(M_5) \neq X_2$)

In the cells in which the sign flip-flop indicates negative, the condition is satisfied.

In addition to the preceding examples, many different combinations are possible, such as comparing two arrays of associative memories, comparing a word with a word of the neighbors, etc.

8.3 Cell Addresses Detection

Once the control flip-flop is set, the next step is the detection of the cells in which the desired condition is satisfied. The required condition for the control flip-flops are selected by the same logic used for inhibit, but only four conditions are of interest now. They are called the content addressable conditions, and are as follows: AC1 = 0; AC1 ≠ 0; AC2 ≥ 0; AC2 < 0.

The inhibit demultiplexer, under instruction control, selects the desired condition and the information required is available at the inhibit output, where an associative flip-flop is set to "1" if the condition is satisfied or to "0" otherwise.

The problem is reduced now to detecting in which cells the associative flip-flops are equal to "1", and reading these addresses into the CPU for further processing. Because more than one cell at a time can satisfy the condition, a priority system for read out must be designed.

A possible configuration is partially shown in Fig. 8.1. A priority system detector is shown for an 8x8 matrix which can be

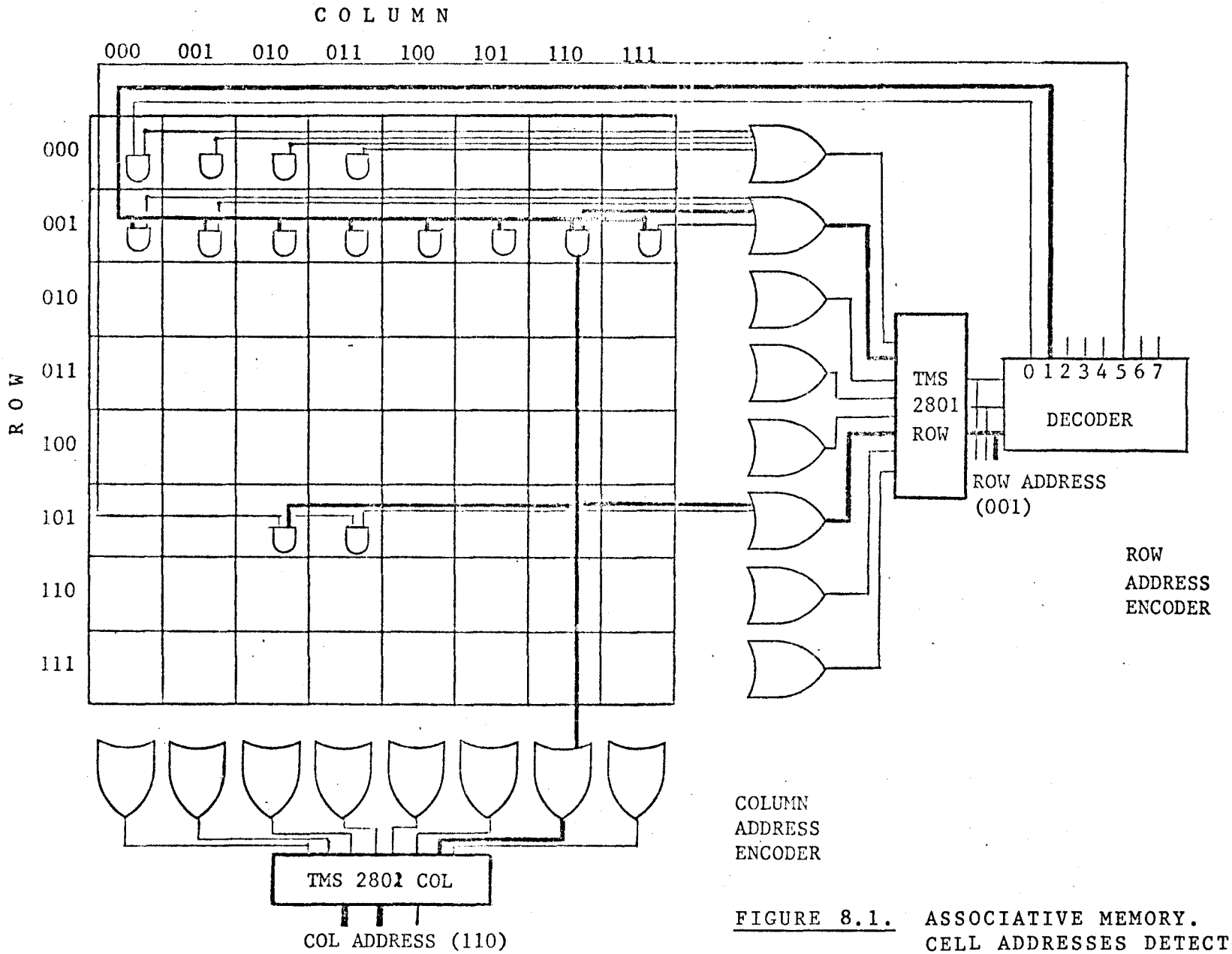


FIGURE 8.1. ASSOCIATIVE MEMORY. CELL ADDRESSES DETECTOR

easily extended to our 63x63 array. The priority is established on the basis of higher priority for lower row number and in case of the same row number, for lower column number.

The TMS 2801 is an "Eight Level Priority Encoder", manufactured by Texas Instruments. It generates an output according to the priority levels present at its inputs. Each input corresponds to a priority level. The highest priority level that is "true" produces its characteristic output code, regardless of the state of the lower priority input lines.

The outputs of the associative flip-flops of all the cells are connected to an OR gate in the "Row Address Encoder", while in the "Column Address Encoder", it is done through AND gates.

Suppose the cells that are in the intersection of row 001 and column 110 [110;001] and in the intersection of row 101 and column 010 [010;101] have the associative flip-flops "high". The "Column Address Encoder" does not have any input because the AND gates are inhibited. The TMS 2801 row, has activated the two input lines corresponding to rows 001 and 101, but it produces an output 001 (highest priority) which is available as a "Row Address". This output is decoded and all the AND gates of row 001 are activated, allowing the "Column Address Encoder" to identify the highest priority column in that row. In this case 110 is the only one. The darker lines in Fig. 8.1 represent the lines that are "true" at that moment.

The information present at both outputs of the TMS 2801's can be read into the CPU, giving a direct identification of the highest priority address of the cells which satisfy the desired conditions.

Once it is read, the computer could generate a pulse to reset the corresponding associative flip-flop and the next highest priority address can then be detected. In this example, the associative flip-flop of the cell [110;001] must be reset and the system will detect the cell [010;101] as a second highest priority.

CHAPTER 9

Conclusion

Much work has been done in the last few years in the field of parallel computers. However, no commercial parallel processor is as yet available in the market. It is expected that the availability of LSI functional units and memories will make parallelism economically attractive.

Besides cost, the main problem that has delayed a rapid development of the parallel processor has been the lack of adequate languages and parallel algorithms. Although some work is being carried out in this field [17], it is expected that more elaborate algorithms will permit more efficient use of the parallel processor, even in problems where with the present software state-of-art, it requires a sequential solution [18]. Although the high level languages used in SOLOMON structures such as Illiac IV [19] are very oriented towards a particular machine, some work in the area of general high level language has been started.

According to the discussion in Chapter 1, there are several degrees of parallelism. This project has been directed toward the design and construction of an HPCS, and the SOLOMON type of structure has been chosen. Although most of the present work is being directed toward this configuration, it is possible that further development in software will make the challenging Holland machine [1] realizable, once the formidable problem of coordination between cells is solved.

The general organization of the system - composed of a PDP-11 computer, an interface unit, a control unit and the array of cells - and the functional relationships and characteristics of the units have been discussed in Chapter 2. Concerning the array architecture, the cells have been organized in a planar rectangular matrix with additional row, column and central buffer. Each cell is related to the corresponding buffers and the four nearest neighbors. However, it is possible that because of some future problem requirements, a further expansion of the cell relationship to the eight nearest neighbors may be implemented. The extension of the present planar SOLOMON structure to a three-dimensional array could open a new area of research.

Computation speed and versatility of the array might be improved by using a more sophisticated and bigger cell. However, the size of the cell discussed in Chapter 3 seems to be very adequate. It is big enough to perform floating-point arithmetic as well as to compute transcendental functions, but small enough to allow an economic construction of a large number of cells.

The "inhibit" system permits flexibility in the control of the operations at the cell level, whereas the "convergence" system is very useful for microprogramming purposes.

Although operations in the cell are carried out bit-by-bit, the incorporation of accumulators and several detectors allows the use of techniques for performing arithmetic operations at relatively high speeds using inexpensive hardware. However, it is expected that the next step in an attempt to improve the cell itself could be the incorporation of multiple bit arithmetic.

The set of instructions and mnemonic presented in Chapter 4 works very satisfactorily in accordance with the purpose of the array and the characteristics of the minicomputer. Any operation can be easily programmed and quickly assembled by the software supplied.

The Control Unit discussed in Chapter 5 performs its task of decoding the information from the CPU and generating the control level and pulses. However, the possibility of incorporating hardware microprogramming should be considered in further improvements. Many basic operations, such as division and floating-point arithmetic are now under software control, but they could be incorporated in the hardware of the system. This feature would allow the CPU to be available for other operations while array computations are being executed besides saving computer storage.

The cell addressing system proposed would incorporate more flexibility in the array. The different addressing modes, in accordance with the nature of the problem, would allow a higher cell selection speed. The ability to operate in a half array seems to be very advantageous because a more powerful system can be obtained without hardware complications. If the nature of problems to be solved with the array would require further extension, the same principle could be used to join more than two cells together.

The system has been interfaced to a PDP-11 computer through the Interface Unit presented in Chapter 6. The unit has been designed in accordance with the computer specifications and it works satisfactorily.

The software package used in this project has been discussed in Chapter 7. The CONTROL program permits a practical use of the array. Several tests have been made using this set of routines and the perfomance of the system -hardware and software- seems to be in accordance with the specifications. The SIMULA program simulates an array of cells in the PDP-11 computer. Some parallel algorithms have been tested using this program. It is expected that further research in the developments of algorithms suitable for the proposed array or, more generally, for any parallel processor, will be made with the SIMULA programs.

The possibility of using the array as an "associative-memory" processor with very little increase in hardware has been pointed out in Chapter 8. Although some content-addressable memories are now available in the market, the increasing needs of information retrieval in business, air traffic control and the applied sciences require a more sophisticated associative-memory processor. The proposal made in this Chapter seems to be adequate and powerful enough because of the great flexibility in information comparison, as well as the very efficient system of priority address detection, without need of scanning or searching the full array.

The Interface, the Control Unit (without the addressing system) and one cell have been constructed and interfaced with the PDP-11 computer. One board per unit has been used. With this structure, the system coul be expanded by simply wiring-in more cell boards.

All the logical functions have been implemented by using integrated circuits, mainly from series 74/ and 74H/ manufactured by Texas Instruments. It was necessary to use 64 chips for the Control Unit, 29 for the Interface and 67 for the Cell. A D.E.C.-M401 variable clock, running at 4.5 MHz and a D.E.C.-M105 Address Selector have also been connected as part of the system. A 5 Volts Power Supply provides the necessary 3.2 Amperes for all the circuits.

The distribution of chips on the boards and a complete wiring diagram, not included in this thesis, are available in a "Technical Report". The report also includes the complete listing of computer programs used in this work.

REFERENCES

1. J. H. Holland; "A Universal Computer Capable of Executing an Arbitrary Number of Subprograms Simultaneously", Proc. AFIPS, Eastern Joint Computer Conf., 1959, pp.108-113.

2. S. H. Unger; "A Computer Oriented Toward Spatial Problems", Proc. IRE, Vol. 46, Oct. 1958, pp.1744-1750.

3. C. Y. Lee and M. C. Paull; "A Content Addressable Distributed Memory with Applications to Information Retrieval", Proc. IEEE Vol. 51, June 1963, pp.924-932.

4. D. L. Slotnick, W. C. Bork and R. C. Mc Reynolds; "The Solomon Computer", Proc. AFIPS, Fall Joint Computer Conf., 1962, pp.97-107.

5. M. A. Knapp; "Parallel Processing Computer System", Rome Air Development Center, Report No. RADC-TR-66-567, AD803485, Nov. 1966.

6. J. O. Campeau; "The Block Oriented Computer", IEEE Computer Group Conference Digest, June 1968, pp.57-60.

7. E. Della Torre and F. K. W. Ho; "Implementation of a Computing Memory Cell", Symposium on Computers and Automata, New York, April 1971.

8. G. H. Barnes, et al.; "The Illiac IV Computer", IEEE Transac. on Computers, August 1968, pp.746-757.

9. W. Y. Dere and D. J. Sakrison; "Berkeley Array Processor", IEEE Transactions on Computers, Vol.C-19,May 1970, pp.444-447.

10. W. T. Comfort; "A Modified Holland Machine", Proc. AFIPS,Fall Joint Computer Conference, 1963, pp.481-488.

11. P. M. Davies and R. G. Ewing; "An Associative Processor", Proc. AFIPS Fall Joint Computer Conf., 1964, pp.147-158.

12. J. A. Rudolph, L. C. Fulner and W. C. Meilander, "The coming of age of the associative processor", Electronics, February 15, 1971, pp.91-96.

13. A. H. Bobeck and H. E. D. Scovil; "Magnetic Bubbles", Scientific American, Vol. 224, No.6, June 1971, pp.78-98.

14. I. Flores; "The Logic of Computer Arithmetic", Prentice-Hall, Englewood Cliffs, N.J., 1963.

15. PDP-11 Interface Manual, Digital Equipment Corporation, Maynard, Mass.,1971.

16. F. Alt and M. Rubinoff; "Advanced in Computers", Vol. 7, Academic Press, New York, 1966.

17. J. L. Baer and E. C. Russell; "Preparation and Evaluation of Computer Programs for Parallel Processing Systems", L. C. Hobbs et al., "Parallel Processor Systems, Technologies, and Applications", Spartan Books, New York, 1970, pp.375-415.

18. J. O. Campeau; "Communication and Sequential Problems in the Parallel Processor", L. C. Hobbs et al., "Parallel Processor Systems, Technologies and Applications", Spartan Books, New York, 1970, pp.215-234.

19. David J. Kuck; "ILLIAC IV Software and Application Programming", IEEE Transactions on Computers, Vol.C-17,No.8, August 1968, pp.758-770.