TEMPORARY COVER SHEET

DOCUMENT NUMBER: 00000104

DOCUMENT TITLE: The Illiac IV Processing Element VOL I

AUTHOR: Theofanis Economidis

DATE ISSUED: April, 1974

INSTITUTE FOR ADVANCED COMPUTATION

IAC DOC. NO. PO-I1100-VOL I-A

THE ILLIAC IV PROCESSING ELEMENT

VOLUME I

THEOFANIS ECONOMIDIS

AUGUST 1973

REVISED: FEBRUARY, 1974

VOLUME I

TABLE OF CONTENTS

VOLUME I

LIST OF TABLES

VOLUME I

LIST OF FIGURES

SECTION A: ILLIAC IV PROCESSING ELEMENT CHARACTERISTICS

I. INTRODUCTION

The ILLIAC IV Computer, conceived and developed at the University of Illinois [1], is considered a milestone in the computer industry because of its fundamental proposition for parallel processing. It is an array of 256 electrically, mechanically, and functionally identical Processing Units with four Control Units, each one responsible for the operation of 64 Processing Units. One Control Unit with its associated array of 64 Processing Units constitutes a quadrant.

Because to date there is only one quadrant available, the ILLIAC IV System (Figure 1) consists of:

a. one Control Unit which decodes those instructions that specify the commands to the Processing Units (FINST/PE Instructions) and those instructions for the operation of the Control Unit itself (ADVAST Instructions).

b. 64 Processing Units each of which functions as an arithmetic-logic unit.

c. the ILLIAC IV Disk File System which consists of two disk files and 13 storage units (disks). Each disk file consists of an Electronics Unit, a concentrator, and necessary circuitry to read and write on any one of its up to 16 disks, whose capacity is approximately 79×10^6 bits per disk, and has a transfer rate of about 500×10^6 bits per second.

d. the ILLIAC IV Input/Output Subsystem used as the interface between the Control Unit and its 64 Processing Units with the Central System and the ILLIAC IV Disk File System. This subsystem consists of a Descriptor Controller (DC), an Input/Output Switch (IOS), and two Disk File Controllers (DFC). The DC is used to receive control words from the Central System over the Scan Bus and to transmit back result descriptors over the same

---

[1] A number enclosed in square brackets signifies a particular document referenced at the end of this manual.

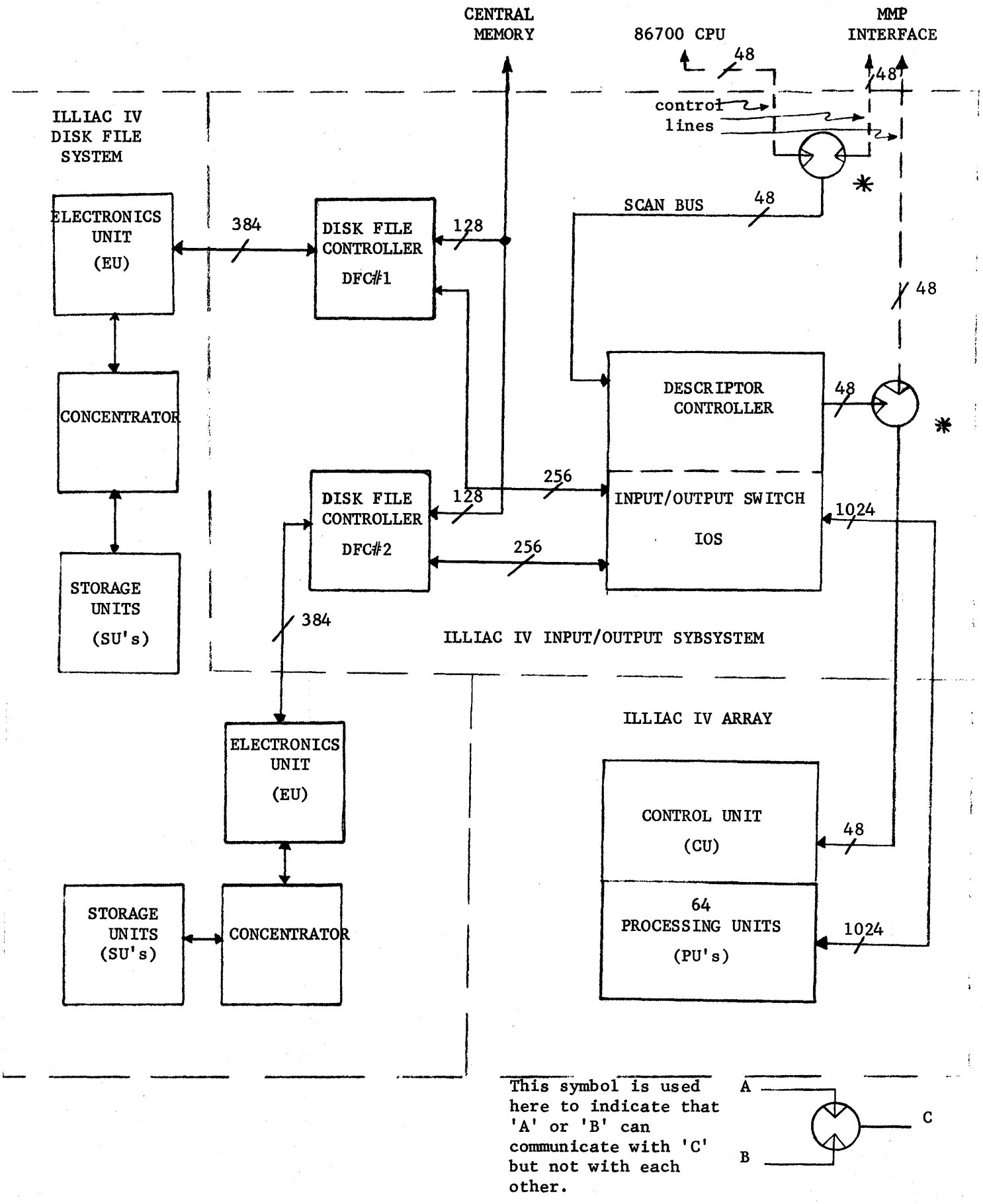


Figure 1. ILLIAC IV System, Functional Block Diagram

path (48 bidirectional lines). The IOS is used to control data transfers between each Disk File Controller and the array. This transfer of data is made through 256 bidirectional lines with each DFC and 1024 bidirectional lines to and from the array. Because the complete ILLIAC IV Computer provides for four quadrants, there are provisions for an expansion to 4096 bidirectional lines of interface between the IOS and the array. The DFC is used to provide the interface for transfers between disk and array, disk and central memory, central memory and array, and and real time link and array.

In order to be consistent with the available literature on ILLIAC IV, it must be mentioned that the ILLIAC IV System is subdivided into two subsystems, namely, the ILLIAC IV computer consisting of one Control Unit and 64 Processing Units, which is usually called a "quadrant" (Figure 2), and the ILLIAC IV Input/Output Subsystem consisting of the ILLIAC IV Disk File System and Input/Output Subsystem, which is usually called "IO."

The PE (Figure 3) is an integral part of the Processing Unit (Figure 4), which operates under the command of the ILLIAC IV Control Unit. Each PU consists of a PE, a Memory Logic Unit (MLU), a Processing Element Memory (PEM), and a dual Power Supply Shunt Regulator (Figure 4). The MLU and PEM have been described separately in NASA/Ames Research Center manuals entitled "The ILLIAC IV Memory Logic Unit" and "The ILLIAC IV Processing Element Memory." Whenever applicable, relevant parts of the Dual Power Supply Shunt Regulator were described in these manuals. Therefore, only those features of the Dual Power Supply Shunt Regulator that concern the power distribution to the PE will be described herein.

The PE contains the necessary logic to execute a full repertoire of instructions under the complete control of the Final Station of the Control Unit (FINST) which accepts these instructions from the Advanced Station of the Control Unit (ADVAST) and, after converting them into microsequences, broadcasts them to the PE. These instructions allow 64-bit, 32-bit, or 8-bit operands to be manipulated by the PE. All operations are fully synchronized through a clock provided to the PE by the Control Unit (CU). This synchronization is accomplished through a Receiver-retiming Register (TUB) which synchronizes the Enables from CU with the clock to the PE before these enables are distributed within the PE. Two mode control

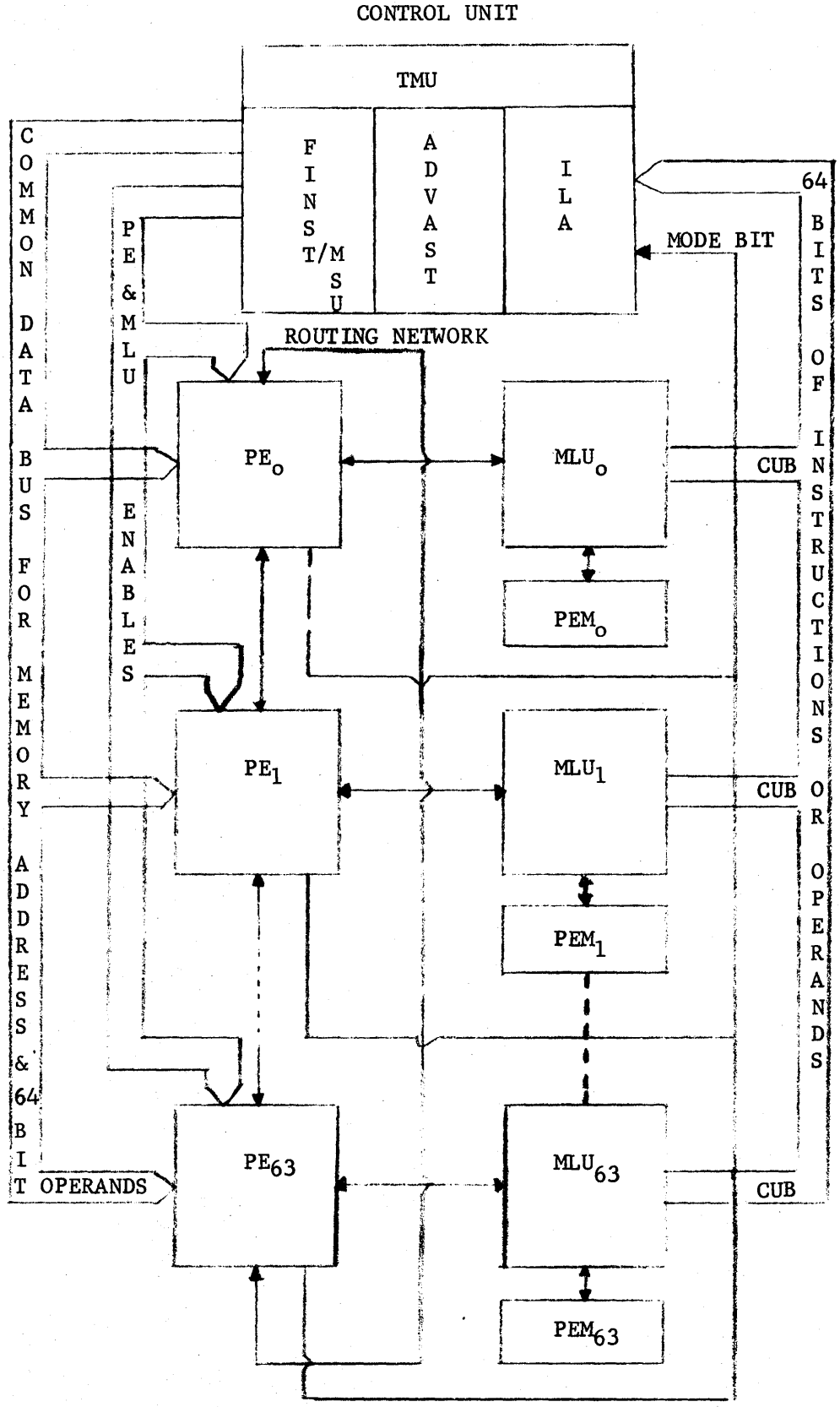


Figure 2. ILLIAC IV Quadrant Functional Block Diagram

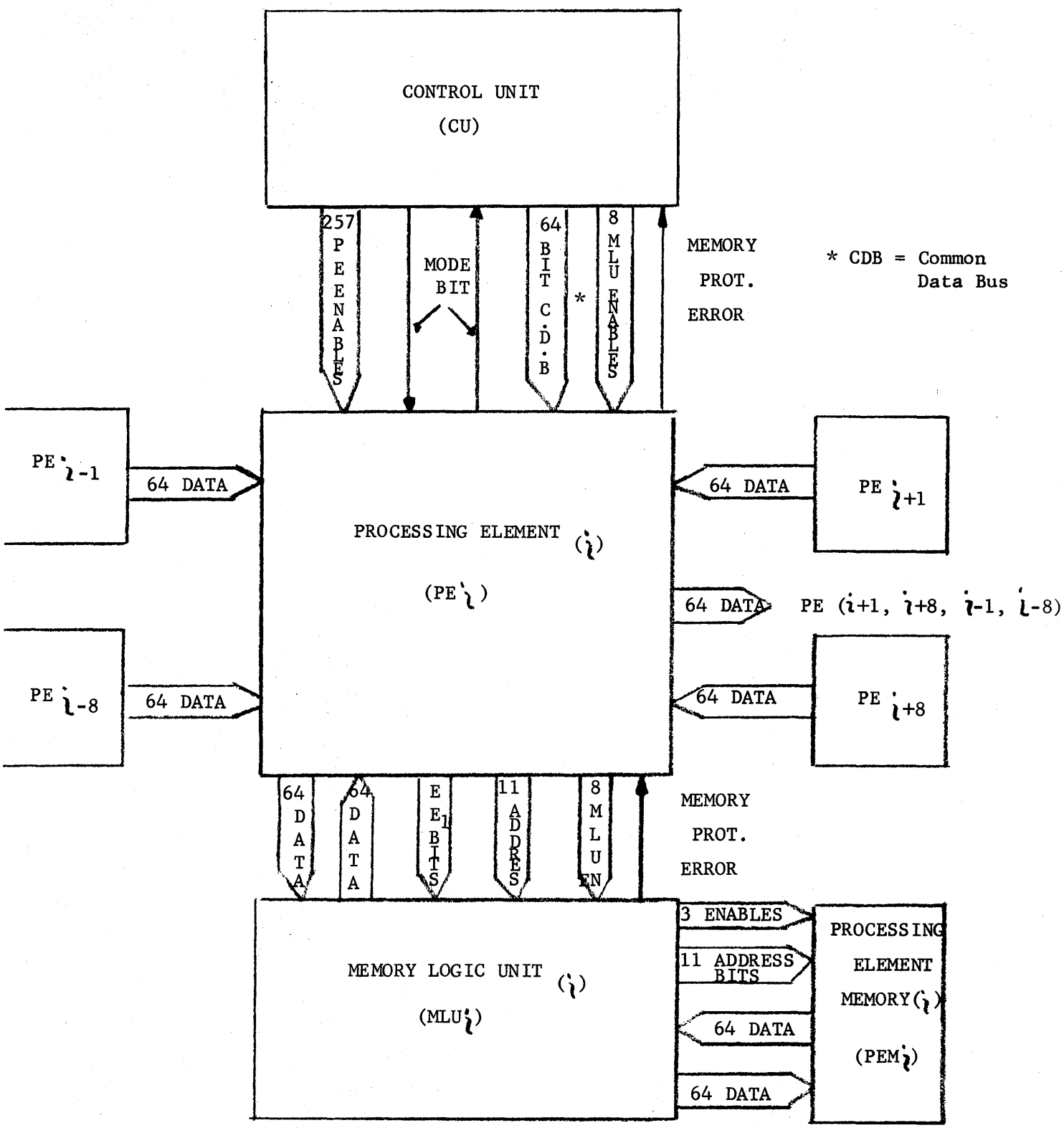


Figure 3. Processing Element Interface Block Diagram

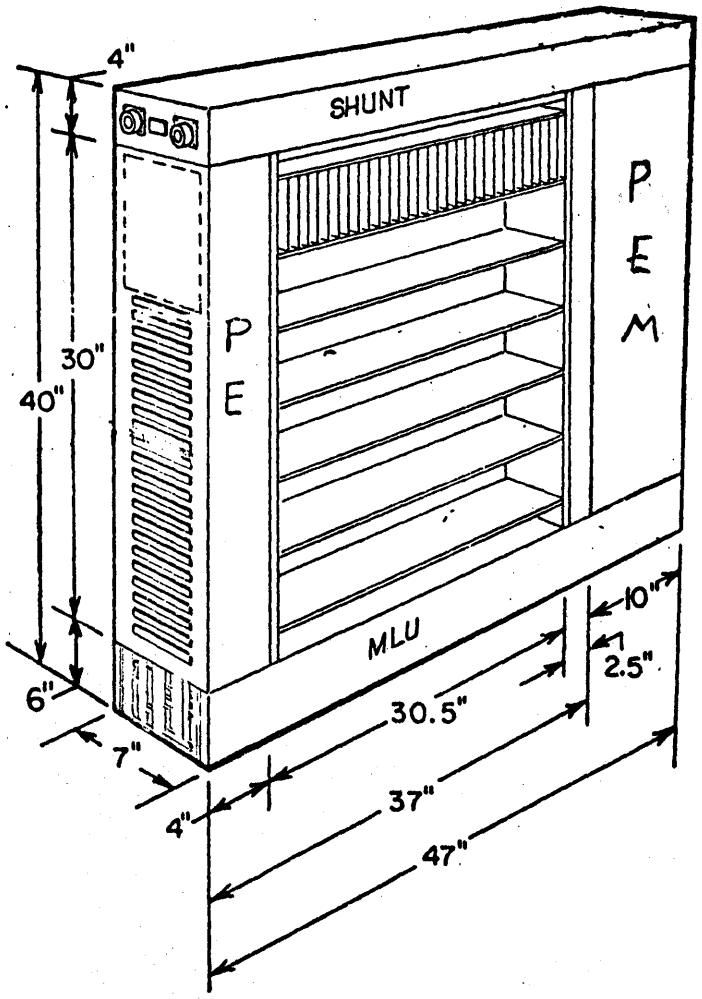


Figure 4. ILLIAC IV Processing Unit

bits protect the contents of the X, S, and A registers (see Section B) by preventing clocks from strobing data into these registers. Each mode bit also controls a separate 32-bit data path through the MLU.

The logic elements of the PE include registers for handling data, a Memory Address Register whose content may be indexed before the address is transmitted to the respective MLU, a Carry Propagating Adder with Carry Look Ahead, a Barrel Switch for shifting operations, a Leading ONES Detector to control the Barrel Switch, a Logic Unit for Boolean and other miscellaneous operations, data Receivers and Drivers for interfacing with neighboring PE's, and other circuits for special Arithmetic and other miscellaneous operations.

The PE is designed to operate at a maximum frequency of 16 MHz. Thus a clock period of 62.5 ns mimimum is used and most of the controls for operation of the PE originate in the FINST portion of the CU.

Because the intent of this manual is to describe how the PE performs the basic four Arithmetic operations (addition, subtraction, multiplication, and division), a brief description concerning the family of logic used by the PE, the type of registers, and other circuits participating in each operation and other PE hardware is provided so that description of the theory of operation of the PE which constitutes the core of this manual will be as simple as possible.

II. SUMMARY OF PE LOGIC CHARACTERISTICS

A. Basic PE Logic

All PE logic circuits belong to the emitter-coupled logic (ECL) family. The basic ECL gate configuration is shown in Figure 5. Although several other ECL circuits are in use in the PE, this circuit helps illustrate some of the general ECL characteristics discussed below, as it represents the basic logic gate of the PE circuits.

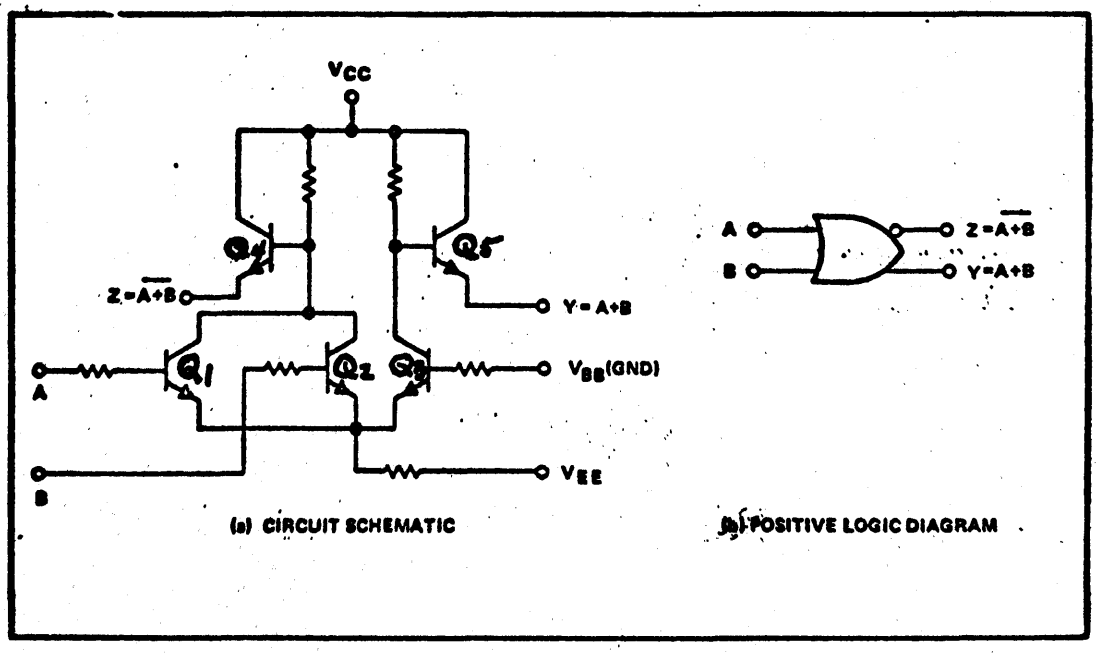


Figure 5. Basic ECL Gate

B. ECL Characteristics

1. Logic Levels.* Typical logic levels employed by the basic ECL gate are 400 mV and -400 mV, when V_{CC} = 1.32 V, V_{EE} = -3.2 V, and V_{REF} = 0 V. Minimum levels when operating at 25°C and loaded with 50 ohms to ground and 270 ohms (pulldown) to -3.2 V are ±350 mV. These logic levels are ensured with inputs at +200 mV, which provide 150 mV of dc noise margin. Since the actual threshold is approximately 150 mV and typical output levels are 400 mV, typical noise margin in excess of 200 mV can be expected. Transfer characteristics for the basic gate are shown in Figure 6.

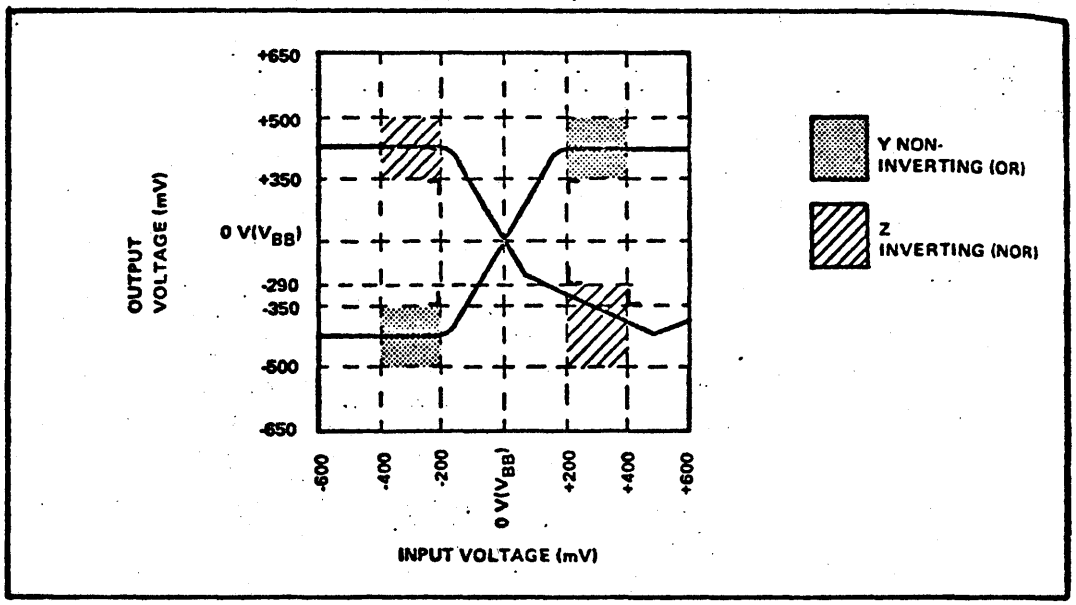


Figure 6. Transfer Characteristics of Basic ECL Gate

For gating functions which have emitter dots (wired OR), the relative-high level is increased to 450 mV; the relative-low level is also increased by 50 mV to -350 mV.

*Information is taken from *The Integrated Circuits Catalog*, Texas Instruments, Incorporated, Dallas, Texas, pp. 4a — 7.

2. Logic Convention. In general, PE logic elements are seen as performing positive logic functions. For this reason, the more positive signal values (+400 mV) are considered the logic ONE levels and the more negative signal values (-400 mV) are considered the logic ZERO levels. Because, however, the signals pass through different gates where in many instances an inversion takes place, the following table clearly indicates at what level the signal is active.

Table 1. PE Signal Representation

PE SIGNAL NAME	LEVEL	LOGIC
PLW-WXX--1	High	1
PLW-WXX--1	Low	0
PLW-WXX--0	High	0
PLW-WXX--0	Low	1

3. Gate Speed.* Switching time performance at 25°C, with various capacitive loadings, is described in Figure 7. This capacitive loading is directly relatable to ac fan-out, assuming 4 to 5 pF per gate input. Delay time degradation with increasing fan-out approximates 75 ps per additional load. Switching-time waveform definitions and output terminations used for testing are shown in Figure 8. Typical propagation time through a single ECL gate is 4 ns from leading edge to leading edge and 4 ns from trailing edge to trailing edge.

*Information is taken from *The Integrated Circuits Catalog*, Texas Instruments, Incorporated, Dallas, Texas, pp. 4a-10, 4a-11.

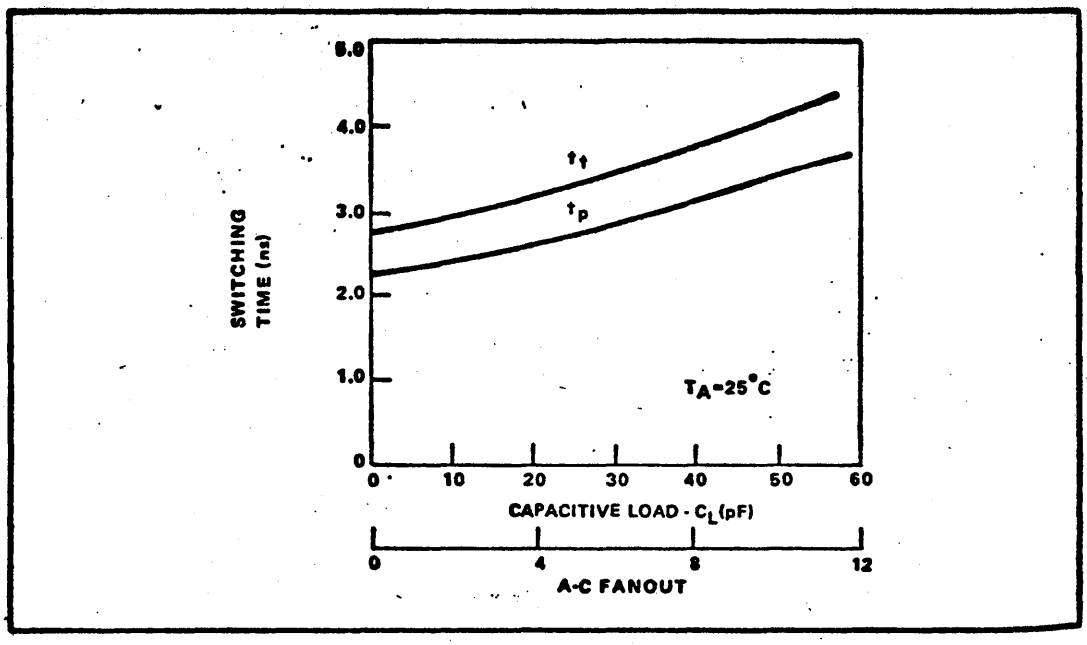


Figure 7. Switching-Time vs. Loading

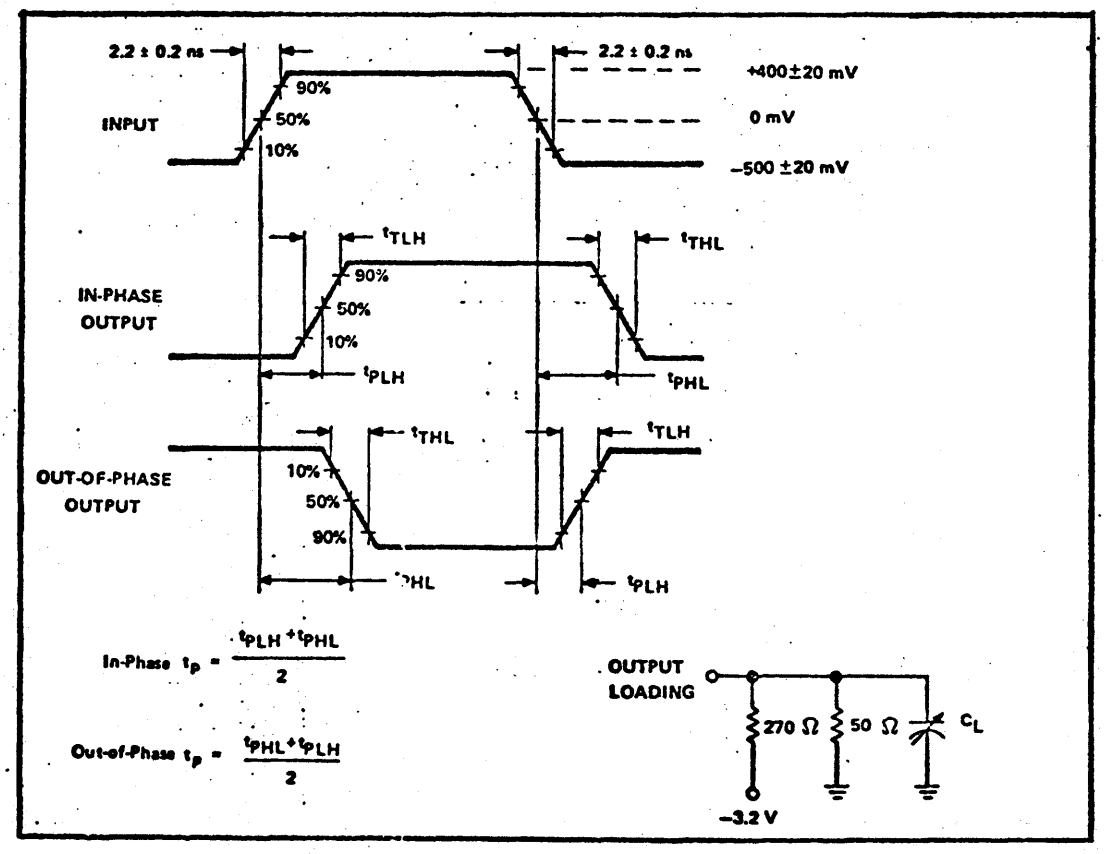


Figure 8. Switching-Time Waveforms

4. Nonsaturation. ECL circuits operate in the nonsaturated mode. That is, the transistors in each gate are never fully cut off or in a saturated state. This is the chief reason for the high switching speed that is characteristic of these circuits. Because the transistors are always conducting, even when the inputs to the gate are false, the inputs do not have to pass a threshold before the logic decision is made. Since the output transistor does not have to be saturated for the output signal to be considered true, there is no switching delay caused by the need to overcome capacitance in the output transistor. For both of these reasons, the output of an ECL gate is able to follow the inputs almost immediately.

5. Complementary Outputs. Many integrated circuit packages included in the ECL family provide dual, complementary outputs. This results in the AND/NAND, OR/NOR, and AND-OR/NOR functions illustrated in Figure 9. Propagation time through these circuits is the same for both outputs; that is, both outputs become valid at the same time.

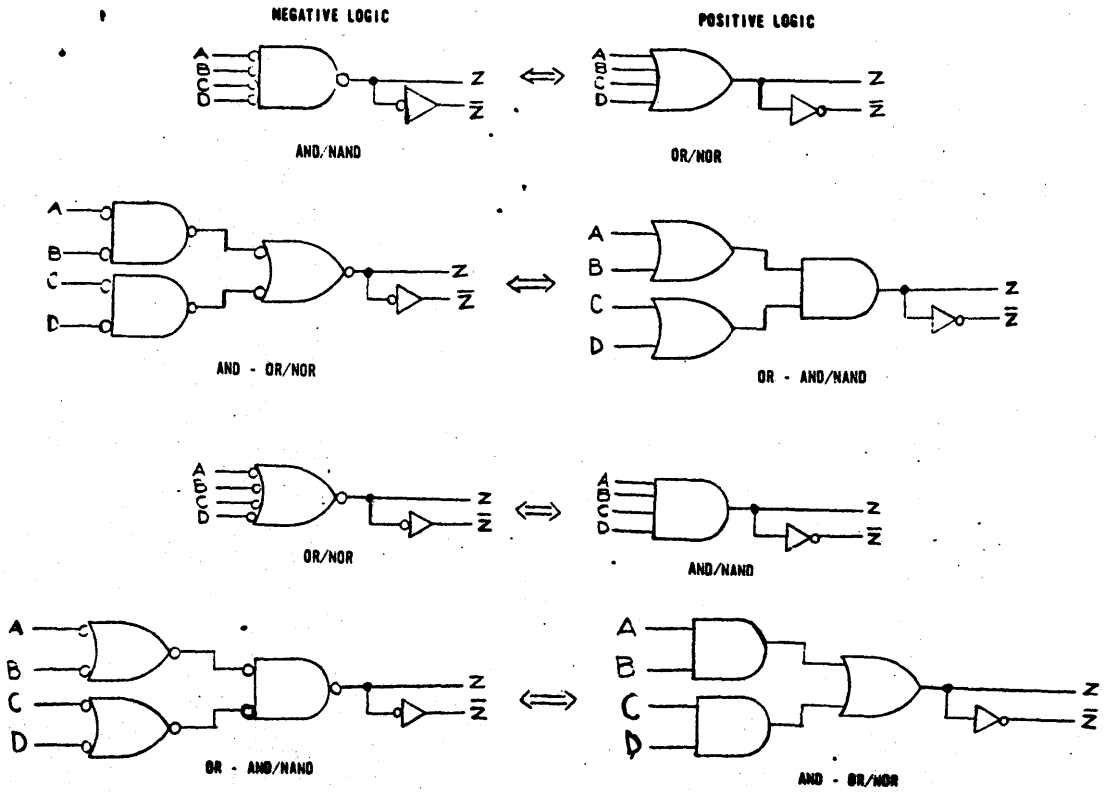
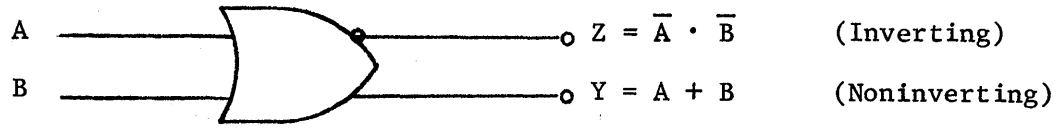


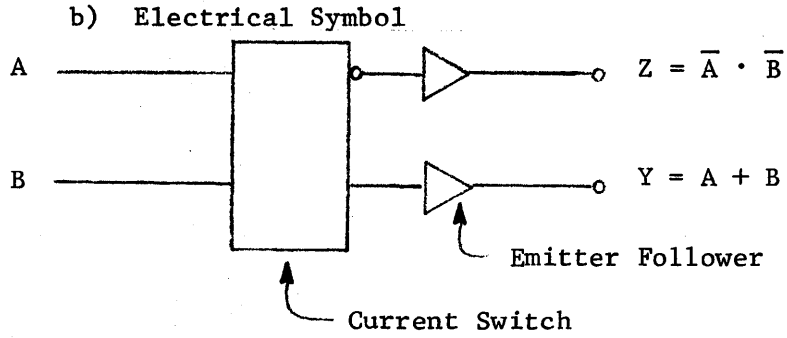
Figure 9. ECL Logic Function with Dual Outputs

6. ECL Symbols. Transistors Q1, Q2, and Q3 of Figure 5 constitute a current switch. The whole circuit, however, can be represented by two symbols:

a) Logic Symbol



b) Electrical Symbol



The ECL circuits can provide additional logic functions by tying collectors together or by tying emitters together. In the case of a collector tie, a clamp circuit is needed to keep the transistors from saturating. In practice, the electrical symbol has an advantage over the logic symbol, because it is easier to show collector and emitter ties and the number of current switches and emitter followers as well. However, it is easier to read logic schematics that employ logic symbols consistently and therefore the ECL gates are represented on PE schematics by logic symbols.

The relatively small swing of ECL logic signals (±400 mV) constitutes a disadvantage because of the associated low noise immunity. Thus on the long signal lines between the CU and PE and those of the routing network between PE_i and PE_{i+1}, PE_{i-1}, PE_{i+8}, PE_{i-8}, where these paths are exposed to relatively high noise transients and also where they pass through areas with significantly different temperatures, signals are transmitted in their true and complement form by line drivers and received by differential line receivers to suppress common-mode noise.

III. SUMMARY OF PE INSTRUCTIONS

In the Introduction, it was mentioned that the Control Unit decodes two types of instructions, namely ADVAST and FINST/PE instructions. The main difference between the two types is that the ADVAST instructions are used to control internal operations of the CU itself, whereas FINST/PE instructions specify functions to be performed by PE's in the quadrant.

Because the FINST/PE instructions involve both the FINST portion of the CU concerned with the transmission of data and commands to an individual PE and requests to the PE to respond to this instruction, and the description required is lengthy, no attempt is made here to describe the FINST/PE instructions; the reader can find this information in [2]. Since FINST/PE instructions call for operations involving data, the word format for both instructions and data is partitioned as follows.

A. Instruction Word Format (FINST/PE)

Figure 10 shows the various fields in a FINST/PE instruction word which is 32 bits in length. The meanings of these fields are as follows:

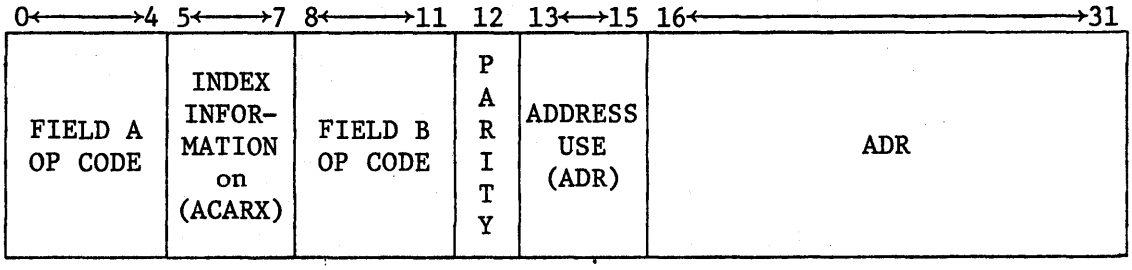


Figure 10. FINST/PE Instruction Word Format

1) Field A OP Code: First part of operation code. Bit #0 is always "1". See Table 4-1 of [2].
2) ACARX: If bit #5 is "1" the contents of one out of four ADVAST accumulator registers specified by bits #6 and #7 must be added to the ADR field. If bit #5 is "0" the value contained in bits #6 and #7 is ignored.
3) Field B OP Code: Second part of operation code. See Table 4-1 of [2].

4) Address Use: The state of these bits specifies the use of the ADR field as shown in Table 2.

Table 2. Truth Table of ADR Use Field and Specified Action

ADR USE			ACTION BEING TAKEN	REMARKS
BIT #13	BIT #14	BIT #15		
0	0	0	CU is transmitting a literal	Bit #14 is ignored
0	0	1	No indexing is required	
0	1	0	CU is transmitting a literal	Bit #14 is ignored
0	1	1	Index ADR by the content of RGX	
1	0	0	CU is transmitting a register code	Bit #14 is ignored
1	0	1	Index ADR by the content of RGS	
1	1	0	CU is transmitting a register code	Bit #14 is ignored
1	1	1	Index ADR by the content of RGS	

5) ADR: This field, depending upon the type of instruction to be executed, designates the location of an operand (for both read and write operations), shift count, amount of indexing and routing distance. The latter is explained later during the description of PE drivers and receivers. Throughout this manual, whenever "content" of ADR in instructions other than shift, indexing bit value and routing distance is mentioned, the content of a register specified by ADR is meant.

B. Data Word Format (FINST/PE)

The FINST/PE instruction repertoire provides various options with regard to data word formats. Figure 11 shows data word formats involving operands partitioned into multiples of eight-bit bytes on both floating and fixed point arithmetic instructions. Figure 11(a) and (b) represents a number in floating point partitioned into two sections which represent the exponent and the mantissa.

1) Exponent: The exponent is represented by an excess or offset code instead of by sign and magnitude for a number of reasons, the most important of which are the following:

   a) Avoidance of the representation of +0 and -0.
   b) Avoidance of recomplementation of the exponent as is required in sign and magnitude representation.

Because every number in floating point is artificially partitioned into exponent and mantissa, the number zero may be represented by a zero mantissa and by an exponent which does not necessarily have to be zero. The zero number, whose exponent does not have the minimum value that a particular register can hold, is called "dirty 0" [3]. A zero number that is represented not only by a zero mantissa, but also by the smallest exponent (all zeros) the machine can store in the proper register, is called "clean 0". If a number A were added to a "dirty" zero number, trouble might be encountered during the alignment step because the number having the smallest exponent would have to be shifted end off to the right as many places as the difference of the two exponents and if A happened to have the smallest exponent significant bits would be lost. This problem is not encountered if "clean 0's" are used. It is then assured that

$$A + 0 = A$$

Of course "clean 0's" can be produced from "dirty 0's" but this would require additional hardware or programming — use of the normalize instruction (NORM) will convert "dirty zeros" to "clean zeros."

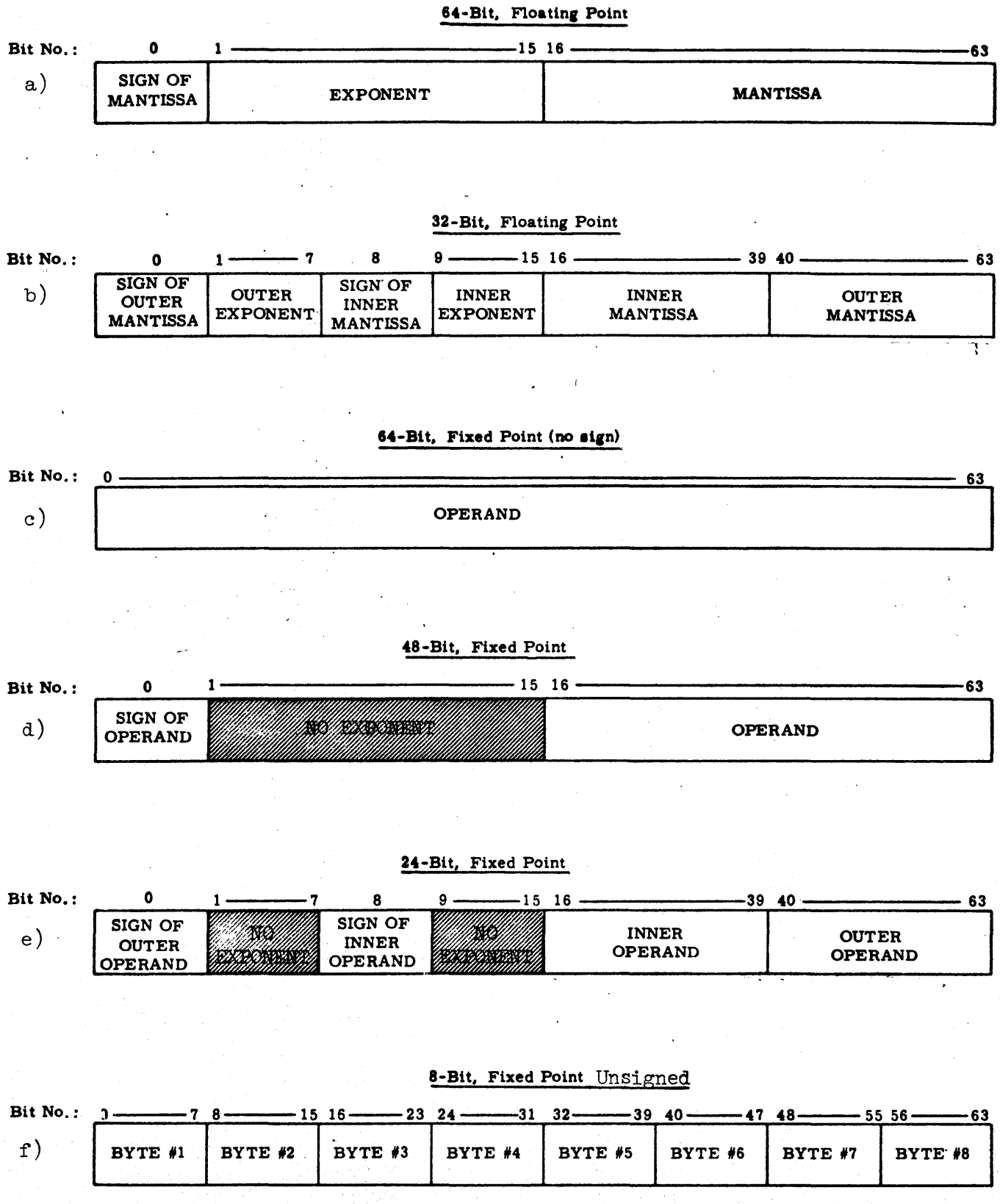


Figure 11. FINST/PE Data Word Formats

In the excess code notation the zero exponent is represented by placing a "1" in the most significant bit position of the exponent field (Figure 11(a) and (b)) and all "0's" in the remaining part of the exponent field. Positive exponents are formed by adding to the value of the excess code the value of the exponent, while for negative components the absolute value of the exponent is subtracted from the value of the excess code.

With this convention, a "1" in the most significant bit position of the exponent field means positive exponent, while a "0" means negative component.

The value of the exponent varies as follows:

64-Bit Mode

EXPONENT FIELD	EXPONENT VALUE
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Maximum Exponent Value
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Zero Exponent Value
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Negative Exponent Value (-1)
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Minimum Exponent Value

32-Bit Mode

EXPONENT FIELD	EXPONENT VALUE
1 1 1 1 1 1 1	Maximum Exponent Value
1 0 0 0 0 0 0	Zero Exponent Value
0 1 1 1 1 1 1	Negative Exponent Value (-1)
0 0 0 0 0 0 0	Minimum Exponent Value

Since

64-Bit Mode

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	represents 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	represents +1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	represents -1

and

32-Bit Mode

1 0 0 0 0 0 0	represents 0
1 0 0 0 0 0 0	represents +1
0 1 1 1 1 1 1	represents -1

it can be said that the exponent is offset by

$$1\ 0\ 0\ 0\ 0\ 0\ 0 = +64)_{10}$$

in the 32-bit mode and by

$$1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 = +16384)_{10}$$

in the 64-bit mode. This means that the actual value of the exponent is the content of the exponent field minus $16384)_{10}$ or minus $64)_{10}$ depending on the bit mode being used.

With the following definitions:

E = content of the exponent field
D = $16384)_{10}$ or $64)_{10}$ for 64- or 32-bit mode, respectively
T = E-D

the exponent can take on the values from $T = 2^{+14} - 1 = 16383)_{10}$ to $T = -2^{+14} = -16384)_{10}$ in the 64-bit mode and from $T = 2^{+6} - 1 = 63)_{10}$ to $T = -2^{+6} = -64)_{10}$ in the 32-bit mode as tabulated below.

		E	D	T = E-D
64	Max	$2^{+15} - 1$	2^{+14}	$2^{+14} - 1 = +16383_{10}$
	Min	0	2^{+14}	$-2^{+14} = -16384_{10}$
32	Max	$2^{+7} - 1$	2^{+6}	$2^{+6} - 1 = +63_{10}$
	Min	0	2^{+6}	$-2^{+6} = -64_{10}$

2) Mantissa: The mantissa is assumed to be a quantity less than 1, having the binary point in front of the first digit. The exponent tells how many places the binary point must be moved from its assumed position toward the right in order to give the true value of the operand. Since a shift of the binary point to the right by one place is the same as by multiplying by 2, the exponent indicates by which power of 2 the mantissa is to be multiplied. The mantissa is represented in sign and magnitude form, which means that numbers having the same absolute value (magnitude)

are identical, but differ only in the sign bit. This notation was chosen to facilitate the implementation of certain arithmetic instructions (particularly multiplication [4]).

Therefore, a number in floating point can be represented as follows:

$$X = (-1)^{Xo}\ 2^{T}\ \left[\sum_{i=1}^{48} 2^{-i}X_i\right] \qquad \text{in 64-bit mode}$$

$$X = (-1)^{Xo}\ 2^{T}\ \left[\sum_{i=1}^{24} 2^{-i}X_i\right] \qquad \text{in 32-bit mode}$$

where

X_0 = sign of mantissa ("0" = +, "1" = -)

2^T = exponent value

$\sum_{i=1}^{48} 2^{-i}X_i$ or $\sum_{i=1}^{24} 2^{-i}X_i$ = mantissa field in binary fractional form in 64-bit and 32-bit mode, respectively

2^{-i} = the weight of the vector X_i

X_i = the vector X in the ith position of the register which can take on the binary values "0" or "1"

The FINST/PE instructions can be classified into two general categories: the transfer of data and the modification of data.

C. Transfer of Data

These instructions involve the PE, the CU, and the PEM as follows:

1. Transfer of data from CU to PE.
2. Transfer of data from PE register to PEM (Table 3).
3. Transfer of data from PE to CU; this operation is known as PE to CUB transfer [5].
4. Transfer of data from PEM to PE register; this is known as READ operation [6] (Table 4).
5. Transfer of data from CU to PEM; because there is no other path available, this transfer is made through the PE and it is similar

to (2) above with the exception that the E, E1 bits are overridden no matter what their status is [5].

6. Transfer of data from one register to another within an individual PE (Table 5).

Table 3. Transfer of Data from PE Register to PEM

MNEMONIC CODE	OPERATION
STA	Store (Write) from RGA to PEM
STB	Store (Write) from RGB to PEM
STR	Store (Write) from RGR to PEM
STS	Store (Write) from RGS to PEM
STX	Store (Write) from RGX to PEM

Table 4. Transfer of Data from PEM to PE Register

MNEMONIC CODE	OPERATION
LDA	Transfer (Read) from PEM to RGA
LDB	Transfer (Read) from PEM to RGB
LDR	Transfer (Read) from PEM to RGR
LDS	Transfer (Read) from PEM to RGS
LDX	Transfer (Read) from PEM to RGX

Table 5. PE Register to PE Register Transmit Instructions

SOURCE OF DATA	DESTINATION OF DATA					
	RGA	RGB	RGD	RGR	RGS	RGX
RGA	—	LDB	*	LDR	LDS	*
RGB	LDA	—	LDD	LDR	LDS	LDX
RGD	—	LDB	—	—	—	*
RGR	LDA	LDB	—	—	LDS	LDX
RGS	LDA	LDB	—	LDR	—	LDX
RGX	LDA	LDB	—	LDR	LDS	—

*No direct path available.

7. Transfer of data between PE's (RTL Instruction). This is a very important feature of the ILLIAC IV System, because it allows full data word communication between the PE's of the quadrant. This communication is called routing; it is used to transmit the contents of any specified register (except the mode register) of any PE to register R of PE(i+D) modulo 64, where i = Initial PE number and D = routing distance specified in bits 22 to 31 of the ADR field (Figure 9).

   NOTE: In transfers 1) through 6) the source register retains the data.

The PE's in the quadrant are numbered from 00 to 63 as shown in Table 6. There is a connection between PE_i with PE_{i+1}, PE_{i+8}, PE_{i-1}, and PE_{i-8} and, because in the quadrant there are only 64 PE's, the routing distance along with PE number where the data is initially found must be modulo 64 in order to make the rotation of data among the PE's possible. The register of PE_i from which the data is transmitted is specified in bits 17 to 21 of the ADR field (Figure 9) and it can, along with D, be indexed only by an ACAR (Advanced Station Accumulator Register) and not by X or S PE registers. The specified PE register for transfer of data to the R register of the PE's designated by the routing distance and its corresponding address bit (ADR field) are as follows:

PE Register	ADR Field Bit
A	17
B	18
X	19
S	20
R	21

If, for example, a route of +15 is requested, the routing action involves two routes with distances of +8 and one route of -1. The data transfers mentioned in 1) through 6) with the exception of 5) are known as transmit instructions, while 5) is known as store instruction.

Table 6. One-Quadrant Array Configuration (Octal Numbering)

PE	PUC 0	PUC 1	PUC 2	PUC 3	PUC 4	PUC 5	PUC 6	PUC 7
↑	00	01	02	03	04	05	06	07
	10	11	12	13	14	15	16	17
	20	21	22	23	24	25	26	27
	30	31	32	33	34	35	36	37
	40	41	42	43	44	45	46	47
	50	51	52	53	54	55	56	57
	60	61	62	63	64	65	66	67
↓	70	71	72	73	74	75	76	77

D. Modification of Data

This class of instructions is subdivided into nonarithmetic and arithmetic instructions.

1. Nonarithmetic instructions: They include —

a) Boolean instructions (Tables 7, 8, 9): Each one of these instructions performs a logic operation on two operands (ADR and RGA). The result is placed into the A register (RGA).[2]

Table 7. Boolean Instructions

MNEMONIC CODE	OPERATION PERFORMED
AND	Logic AND of RGA with ADR
ANDN	Logic AND of RGA with complement of ADR
EOR	Logic exclusive OR of RGA with ADR
EQV	Logic equivalence of RGA with ADR
NAND	Logic AND of complement of RGA with ADR
NANDN	Logic AND of complement of RGA with complement of ADR
NOR	Logic OR of complement of RGA with ADR
NORN	Logic OR of complement of RGA with complement of ADR
OR	Logic OR of RGA with ADR
ORN	Logic OR of RGA with complement of ADR

[2]In subsequent discussion, a symbol (such as RGA or RGX) is used alternatively to mean either the register (A or X) or the content of the register.

NOTES: 1. Instructions AND, ANDN, NAND, and NANDN perform the logic AND of the content of the A register with the content of ADR bit by bit. The content of ADR is first placed in the B register through the Common Data Bus (ADR is more general than CDB data in that it may be a PEM or other PE register word), and then the instruction is executed. (The contents of ADR may also be a PE or PEM word.) Because the result of this logic function involves two registers (A and B), the truth table (Table 8) shows the four combinations of the bits in the A and B registers and the results.

2. Instructions NOR, NORN, OR, and ORN perform the logic OR of the content of the A register with the content of ADR which is placed into the B register before the instruction is executed. The truth table (Table 9) shows the four combinations of the bits in the A and B registers and the results.

Table 8. Truth Table of Boolean Functions

STATE OF A REGISTER BIT	STATE OF B REGISTER BIT	LOGIC FUNCTION PERFORMED			
		AND	ANDN	NAND	NANDN
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Table 9. Truth Table of Boolean Functions

A REGISTER BIT	B REGISTER BIT	LOGIC FUNCTION PERFORMED			
		NOR	NORN	OR	ORN
0	0	1	1	0	1
0	1	1	1	1	0
1	0	0	1	1	1
1	1	1	0	1	1

b) Comparison instructions (Tables 10 and 11): These instructions test as to whether the contents of the A or X or S registers of a PE are greater, equal to, or less than the contents of ADR; they also check to see if the contents of A are equal to logic 1 or logic 0. The result of this comparison is stored in the I or J bit of the mode register (RGD) in 64-bit mode or in the I/G or J/H bits in the 32-bit mode.

Table 10 appears on page 26

Table 11. Nonarithmetic Instructions
(Arithmetic Comparison)

MNEMONIC CODE	OPERATION PERFORMED
(I/J) A (G/L)	Determine whether the content of RGA is arithmetically (A) greater or less than ADR.
IAG	Place result of test for RGA arithmetically greater than ADR into I.
IAL	Place result of test for RGA arithmetically less than ADR into I.
JAG	Place result of test for RGA arithmetically greater than ADR into J.
JAL	Place result of test for RGA arithmetically less than ADR into J.

NOTE: These instructions are executed in either 64-bit or 32-bit mode. In 64-bit mode the result is placed into I or J bit of the mode register, while in 32-bit mode the result is placed in I or J for the Outer word and in G or H for the Inner word.

Table 10. Nonarithmetic Instructions (Logic Comparison)

MNEMONIC CODE	OPERATION PERFORMED
(I/J) (L/M) (E/G/L)	Determine whether the logic word (L) or mantissa (M) part of RGA is equal to (E), greater (G), or less (L) than ADR.
ILE	Place result of test for RGA logically equal to ADR into I.
ILG	Place result of test for RGA logically greater than ADR into I.
ILL	Place result of test for RGA logically less than ADR into I.
IME	Place result of test for RGA mantissa logically equal to ADR into I.
IMG	Place result of test for RGA mantissa logically greater than ADR into I.
IML	Place result of test for RGA mantissa logically less than ADR into I.
JLE	Place result of test for RGA logically equal to ADR into J.
JLG	Place result of test for RGA logically greater than ADR into J.
JLL	Place result of test for RGA logically less than ADR into J.
JME	Place result of test for RGA mantissa logically equal to ADR into J.
JMG	Place result of test for RGA mantissa logically greater than ADR into J.
JML	Place result of test for RGA mantissa logically less than ADR into J.
(I/J) (S/X) (E/G/L)	Determine whether the content of RGS (S) or RGX (X) is equal to, greater, or less than ADR.
ISE	Place result of test for RGS logically equal to ADR into I.
ISG	Place result of test for RGS logically greater than ADR into I.
ISL	Place result of test for RGS logically less than ADR into I.
IXE	Place result of test for RGX logically equal to ADR into I.
IXG	Place result of test for RGX logically greater than ADR into I.
IXL	Place result of test for RGX logically less than ADR into I.
JSE	Place result of test for RGS logically equal to ADR into J.
JSG	Place result of test for RGS logically greater than ADR into J.
JSL	Place result of test for RGS logically less than ADR into J.
JXE	Place result of test for RGX logically equal to ADR into J.
JXG	Place result of test for RGX logically greater than ADR into J.
JXL	Place result of test for RGX logically less than ADR into J.
(I/J) (L/M) (O/Z)	Determine whether the logic word or mantissa of RGA is equal to all ONES (O) or all ZEROS (Z).
ILO	Place result of test for RGA logically equal to all ONES into I.
ILZ	Place result of test for RGA logically equal to all ZEROS into I.
IMO	Place result of test for RGA mantissa logically equal to all ONES into I.
IMZ	Place result of test for RGA mantissa logically equal to all ZEROS into I.
JLO	Place result of test for RGA logically equal to all ONES into J.
JLZ	Place result of test for RGA logically equal to all ZEROS into J.
JMO	Place result of test for RGA mantissa equal to all ONES into J.
JMZ	Place result of test for RGA mantissa equal to all ZEROS into J.

NOTES: (for Table 11)

1. Instructions (I/J) (L/M) (E/G/L) are executed either in 64-bit full word or in mantissa parts. In 64-bit mode the result is stored in either I or J bit of the mode register while in 32-bit mode the result is stored in I or J (Outer word) and in G or H (Inner word).
2. Instructions (I/J) (S/X) (E/G/L) are used to compare the 16-bit index register (RGX) or the 16 least significant bits of RGS with the 16 least significant bits of ADR.
3. Instructions (I/J) (L/M) (O/X) are executed either in 64-bit full word or in mantissa parts. In 64-bit mode the result is stored in the I or J bit of the mode register, while in 32-bit mode the result is stored in I or J bit (Outer word) and G or H bit (Inner word).

c) Modify and test index instructions (Table 12): These instructions modify the content of the X register of a PE by adding or subtracting the content of ADR to or from it. If an overflow results, the overflow bit is stored in either the I or J bit of the mode register; the type of instruction determines whether it is the I bit or the J bit.

Table 12. Nonarithmetic Instructions
(Modify and Test Index)

MNEMONIC CODE	OPERATION PERFORMED
(I/J) XGI	Add the least significant 16 bits of ADR to RGX and store the carryout (overflow) in I or J bit of mode register.
IXGI	Add ADR to RGX and store overflow in I.
JXGI	Add ADR to RGX and store overflow in J.
(I/J) XLD	Subtract the least significant 16 bits of ADR from RGX and store the complement of the carry-out into I or J bit of mode register.
IXLD	Subtract ADR from RGX and store complement of overflow in I.
JXLD	Subtract ADR from RGX and store complement of overflow in J.
XI	Add the least significant 16 bits of ADR to RGX and place the result in RGX modulo 16.
XD	Subtract the least significant 16 bits of ADR from RGX (2's complement) and place the result in RGX modulo 16.

d) Modify bit of A register (Table 13): These instructions set, reset, or complement a selected bit of A register in 64-bit mode or two bits in 32-bit mode.

Table 13. Nonarithmetic Instructions
(Modify Bit of RGA)

MNEMONIC CODE	OPERATION PERFORMED
CAB	Complement bit(s) in RGA
CHSA	Change sign(s) in RGA
RAB	Reset bit(s) in RGA
SAB	Set bit(s) in RGA
SAP	Reset sign(s) in RGA
SAN	Set sign(s) in RGA

e) Transmit bit of A register (Table 14): These instructions transmit a selected bit of A register to the I or J bit of the mode register in 64-bit mode or two bits in 32-bit mode in which case one bit is transmitted to I or J and the other to G or H bit of the mode register.

Table 14. Nonarithmetic Instructions
(Transmit Bit of RGA)

MNEMONIC CODE	OPERATION PERFORMED
(I/J) (B/SN)	Transmit RGA bit (B) or sign (SN) to mode register:
IB	Transfer RGA bit(s) to I bit (and G bit in 32-bit mode).
ISN	Transfer RGA sign(s) to I bit (and G bit in 32-bit mode).
JB	Transfer RGA bit(s) to J bit (and H bit in 32-bit mode).
JSN	Transfer RGA sign(s) to J bit (and H bit in 32-bit mode).

f) Eight-bit byte (Table 15): These instructions add or subtract ADR and A register and also test as to whether the content of A register is greater, less, or not equal to the content specified by ADR. These operations are performed in eight-bit bytes whose format is shown in Figure 11(f). Results of the test instructions are left in RGA. The least significant bit of each byte is the result; all other bits are ZERO.

Table 15. Nonarithmetic Instructions
(Eight-Bit Byte)

MNEMONIC CODE	OPERATION PERFORMED
GB	Test for RGA greater than ADR
NEB	Test for RGA not equal to ADR
LB	Test for RGA less than ADR
ADB	Add ADR to RGA
SBB	Subtract ADR from RGA
OFB	Transmit overflow bits of previous 8-bit byte instructions from RGC to RGB

g) Modify exponent (Table 16): These instructions load, add, or subtract ADR exponents into, to or from the exponent field(s) of A register in both the 64- or 32-bit modes. No change of sign(s) or the mantissa(s) takes place.

Table 16. Nonarithmetic Instructions
(Modify Exponent)

MNEMONIC CODE	OPERATION PERFORMED
LEX	Load ADR exponent(s) into RGA exponent field.
ADEX	Add ADR exponent(s) to RGA exponent(s).
SBEX	Subtract ADR exponent(s) from RGA exponent(s).

h) Shift instructions (Table 17): There are basically ten shift instructions whose general characteristics are as follows:

1) Right shift count with indexing: The shift count is sent to the Address Adder through the Common Data Bus (CDB) and Operand Select Gates (OSG) where it may be indexed by X or S registers. The content of ADA is stored into the shift count register (LOD #4) as modulo 64 or 32 depending upon the mode of operation. Because the original shift count may be the sum of ADR and the content of one of the ACAR's, it can be said that the shift count $N = ADR + ACAR_i + X$ or S register, where $i = 0, 1, --, 3$. If $ACAR_i$, X or S register is not specified for indexing then they are assumed to be zero in the above equation [2].

2) Left shift count with indexing: As will be explained later when the Barrel Switch is discussed, all shifting is actually performed to the right. In order to perform a left shift, the ADA receives (through the same channel as in right shifting) the shift count minus one (N-1). This number may be indexed at ADA and is transmitted to shift count register (LOD #4). Then the output of LOD #4 is complemented (1's complement) and the result is equivalent to a right shift number as shown in Table 17.

Table 17. Right and Left Shift Count Equivalence

OUTPUT OF CU		OUTPUT OF LOD #4	
Right Shift (N)	Left Shift (N-1)	Right Shift (N)	Left Shift ($\bar{N}$)
0	63	0	0
1	0	1	63
⋮	⋮	⋮	⋮
63	62	63	1

3) E bits disabled: Because the mode register has not yet been discussed, in particular the role of the E, E1 bits, it is only mentioned here that whenever one or both the E bits are not enabled the part of the register corresponding to the disabled E bit is unchanged by the shift instruction.

4) End around shifts: In addition to the six bits specifying the shift count (N), the shift count register receives from the CU (through the CDB and OSG) two bits, the status of which specifies the direction of shifting, that is, end around, right end off, or left end off. During the end around instruction, whatever bits are shifted out of the right end of one register reappear at the other end of the register.

5) End off shifts: During this instruction, whatever bits have been shifted out of the right or left end of the register do not reappear at the other end, but instead a number of "0's" equal to the number of the "shifted off" bits are forced into the other end of the register.

6) Mantissa shifts: These instructions refer to the mantissa part of the register which may be shifted left or right end off or end around. In left or right end off shifts, if the shift count is >48 in 64-bit mode or if the shift count is >24 in 32-bit mode the mantissa portion of the register is forced to zero.

7) Logic shifts: These instructions enable the whole word in 64-bit mode or half of the word in 32-bit mode for shifting operations left or right end off or end around.

8) Double and single length shifts: When a double length shift is requested, A and B registers of the PE are treated as one 128-bit register and their contents may be shifted left or right end off. These shifts are valid only for 64-bit mode and may include logic shifts or mantissa shifts. In this case, E = E1. When single length shifts are requested, the content of a register is shifted left or right end off or end around and may include logic shifts or mantissa shifts. The single length shifts are valid for both 64- and 32-bit mode.

Table 18. Nonarithmetic Instructions (Shift)

MNEMONIC CODE	OPERATION PERFORMED
RTAL	Rotate (shift left), end around, logic, single length
RTAR	Rotate (shift right), end around, logic, single length
SHABL	Shift left, end off, logic, double length
SHABR	Shift right, end off, logic, double length
SHABML	Shift left, end off, mantissa only, double length
SHABMR	Shift right, end off, mantissa only, double length
SHAL	Shift left, end off, logic, single length
SITAR	Shift right, end off, logic, single length
SHAML	Shift left, end off, mantissa only, single length
SHAMR	Shift right, end off, mantissa only, single length

NOTES: 1. The meaning of the variants for shift instructions is the following:

RT = Rotate (implies end around)
SH = Shift (implies end off)
A = RGA (single length)
AB = RGA and RGB (double length)
M = Mantissa part only
L = Left
R = Right

2. Whenever the letter M does not appear it is assumed that a logic shift is requested, which means that all bits of the word (or half word) are enabled.

i) Mode register instructions (Table 19): These instructions are subdivided into two categories:

1) Load instructions: These instructions load the specified mode register bit(s) with a bit from CU (ACAR). ACAR may be indexed as previously explained. In this instruction the ADR-use field of the instruction word (Figure 10) is not used. If the LOAD EEl instruction is requested, however, the particular bit of ACAR will load both the E and El bits of the mode register of the corresponding PE.

Table 19. Nonarithmetic Instructions (Mode Register)

MNEMONIC CODE	OPERATION PERFORMED
(LD)	Load mode register from ACAR as follows:
LDE	Load mode register E bit from ACAR
LDE1	Load mode register E1 bit from ACAR
LDEE1	Load mode register E, E1 bits from ACAR
LDG	Load mode register G bit from ACAR
LDH	Load mode register H bit from ACAR
LDI	Load mode register I bit from ACAR
LDJ	Load mode register J bit from ACAR
(SET)	Set mode register bit with the result of a logic function of two bits specified in the ADR field as follows:
SET E	Set mode register bit E
SET E1	Set mode register bit E1
SET F	Set mode register bit F
SET F1	Set mode register bit F1
SET G	Set mode register bit G
SET H	Set mode register bit H
SET I	Set mode register bit I
SET J	Set mode register bit J

2) Set instructions: These instructions force a particular bit of the mode register to be set with the result of a logic function of two bits. These two bits (B1 and B2) and the logic function occupy bit positions in the instruction word (Figure 10) as shown in Table 20.

Table 20. Mode Register Set Instructions

LOGIC FUNCTION		MODE BIT B2		MODE BIT B1	
CONTENT	INSTRUCTION WORD BIT	CONTENT	INSTRUCTION WORD BIT	CONTENT	INSTRUCTION WORD BIT
B1 or B2	16	$\overline{E1}$	20	H	24
$\overline{B1}$ or B2	17	E1	21	G	25
B1 and B2	18	$\overline{E}$	22	J	26
$\overline{B1}$ and B2	19	E	23	I	27
				E1	28
				E	29
				F1	30
				F	31

j) Miscellaneous instructions (Table 21): So far all of the FINST/PE instructions have been arranged according to a functional group which is briefly introduced to the reader by some general comments. Because the set of instructions shown in Table 21 covers a variety of operations (though equally as important as the grouped instructions), they are referred to as "miscellaneous."

Table 21. Nonarithmetic Instructions (Miscellaneous)

MNEMONIC CODE	OPERATION PERFORMED
ASB*	Place sign(s) of RGA into sign(s) position of RGB
CLRA*	Clear RGA
COMPA*	Complement RGA
SWAP*	Interchange RGA and RGB
SWAPX*	Interchange the inner and outer words in RGA
T3A	Transfer contents of RGC to RGA
EAD	Recover extended precision after addition in floating point arithmetic
ESB	Recover extended precision after subtraction in floating point arithmetic

*See [2] pp. 4-13.

2. Arithmetic instructions: These instructions are the most important and will be treated separately later in the theory of operation. They can be separated into four general categories, each one including a limited number of options. These instructions involve operands whose formats are shown in Figure 11. The programmer, in addition to having to deal with the formats of Figure 11, which involve floating, fixed point, and unsigned operands, has the option of normalization and rounding. The basic arithmetic instructions are:

1) Addition (Table 22)
2) Subtraction (Table 23)
3) Multiplication (Table 24)
4) Division (Table 25)

and the variants being used for the above instructions are specified as follows:

Suffix	Meaning
A	Unsigned
M	Fixed point
N	Normalized result
R	Rounded result

When unsigned operands are dealt with, their mantissa signs are forced to look alike (positive) but the original sign(s) of RGA are retained.

When fixed point arithmetic is requested, the exponent field of the operand is ignored. If a number A is to be added to a "zero number" the order of magnitude is not important. Therefore whether "dirty" or "clean" 0's are being dealt with is of no concern because no alignment is involved (due to a difference in the exponents of A and the zero number). Thus, even if the order of magnitude of the zero number (not zero exponent) is greater than the order of magnitude of A, it is assumed that

$$A + 0 = A$$

When the normalize variant is used, the result of addition, subtraction, multiplication, and division of two operands (floating point arithmetic) must be normalized. That is, the leading ONE of the mantissa field should be brought into bit position 16 in 64-bit mode or 32-bit mode Inner word or in bit 40 in 32-bit mode Outer word (Figure 11(a) and (b)), and the exponent reduced accordingly. Also the programmer might normalize (NORM

instruction) the operands before the actual arithmetic operations begin. This is an option which is only mandatory in division, where the divisor must be normalized before the recursive process starts. If the mantissa part of the operand to be normalized is zero, the leading ONE detectors will be unable to generate a shift amount for shifting the mantissa to the left and subsequently to have the exponent reduced. Under these circumstances the mantissa sign and exponent part of the operand or result is forced to zeros. This zero operand is called "clean" or true zero. Using the excess or offset code when representing the exponent, it was shown previously that the exponent value is 2^T where T can vary from $2^{+14}-1$ to -2^{+14} in 64-bit mode and from $2^{+6}-1$ to -2^{+6} in 32-bit mode. In excess code, all zeros in the exponent field of the operand or of the result represents the minimum possible exponent value; therefore, the zero operand or result can be represented by

$$X = (-1)^{Xo} 2^{-2^{+14}} \cdot 0 \qquad \text{in 64-bit mode}$$

$$X = (-1)^{Xo} 2^{-2^{+6}} \cdot 0 \qquad \text{in 32-bit mode}$$

where

X = operand or result to be normalized
Xo = mantissa sign (0 = +, 1 = -)
0 = zero mantissa.

The rounding option (R) is very important when high precision is needed because it saves significant bits (in the case of addition) which otherwise could be truncated. However, because rounding is treated differently in every individual arithmetic operation, it is described explicitly in the "Theory of Operation" section of this manual.

Since the intent of this manual is to describe how the PE operates as an arithmetic unit from the hardware point of view, the reader is urged to read Chapter IV of [2] which covers the FINST/PE Instruction Repertoire quite extensively. For convenience, a "FINST/PE Instruction Index," which was borrowed from [2], is provided in Table 26 of this manual. A list of reference pages, which is included in the original table as it appears in [2], has been omitted from the table.

Table 22. Arithmetic Instructions (Addition)

MNEMONIC CODE	OPERATION PERFORMED
	Add the content of ADR to RGA. Variants are: A, M, N, R
AD	Add in floating point
ADA	Add in floating point two unsigned numbers
ADM	Add in fixed point
ADMA	Add in fixed point two unsigned numbers
ADN	Add in floating point and normalize
ADNA	Add in floating point two unsigned numbers and normalize
ADR	Add in floating point and round
ADRA	Add in floating point two unsigned numbers and round
ADRN	Add in floating point, round and normalize
ADRNA	Add in floating point two unsigned numbers, round and normalize
ADD	Add ADR to RGA. The operands are 64-bit, fixed point, and unsigned

NOTES: 1. If M (fixed point) is specified, the operands and their results are treated as fixed point numbers. The original content of RGA exponent field is retained in the result.

2. The content of ADR specifies the source of addend which is brought into B register before the operation begins.

Table 23. Arithmetic Instructions (Subtraction)

MNEMONIC CODE	OPERATION PERFORMED
	Subtract the content of ADR from RGA. Variants are: A, M, N, R
SB	Subtract in floating point
SBA	Subtract in floating point two unsigned numbers
SMB	Subtract in fixed point
SBMA	Subtract in fixed point two unsigned numbers
SBN	Subtract in floating point and normalize
SBNA	Subtract in floating point two unsigned numbers and normalize
SBR	Subtract in floating point and round
SBRA	Subtract in floating point two unsigned numbers and round
SBRN	Subtract in floating point, round and normalize
SBRNA	Subtract in floating point two unsigned numbers, round and normalize
SUB	Subtract 64-bit, fixed point number of ADR from RGA

NOTES: 1. If M (fixed point) is specified, both the operands and their results are treated as fixed point numbers. The original content of RGA exponent field is retained in the result.

2. The content of ADR specifies the minuend which is placed in B register before the operation begins.

Table 24. Arithmetic Instructions (Multiplication)

MNEMONIC CODE	OPERATION PERFORMED
	Multiply the content of RGA by the content of ADR. Variants are: A, M, N, R
ML	Multiply in floating point
MLA	Multiply two unsigned numbers in floating point
MLM	Multiply in fixed point
MLMA	Multiply two unsigned numbers in fixed point
MLN	Multiply in floating point and normalize
MLNA	Multiply in floating point unsigned numbers and normalize
MLR	Multiply in floating point and round
MLRA	Multiply in floating point unsigned numbers and round
MLRM	Multiply in fixed point and round
MLRMA	Multiply in fixed point unsigned numbers and round
MLRN	Multiply in floating point, round and normalize
MLRNA	Multiply in floating point unsigned numbers, round and normalize
MCM	Execute one cycle of multiplication
MULT	See [2] pp. 4-72

NOTES: 1. MCM and MULT are special instructions and are treated differently than the ML instruction.

2. When M is specified, the values of the two operands and their results are treated as numbers in fixed point. The original content of RGA exponent field is retained in the result.

3. The content of ADR specifies the multiplier which is placed in B register before the operation begins.

Table 25. Arithmetic Instructions (Division)

MNEMONIC CODE	OPERATION PERFORMED
	Divide the mantissa of RGA and RGB (double length) by the content of ADR which is in RGR. Variants are: A, M, N, R
DV	Divide
DVA	Divide unsigned numbers
DVM	Divide numbers in fixed point
DVMA	Divide unsigned numbers in fixed point
DVN	Divide and normalize quotient field
DVNA	Divide unsigned numbers and normalize quotient field
DVR	Divide and round
DVRA	Divide unsigned numbers and round
DVRM	Divide, round and normalize quotient field
DVRMA	Divide unsigned numbers in fixed point and round
DVRN	Divide, round and normalize
DVRNA	Divide unsigned numbers, round and normalize

NOTES:
1. If M (fixed point) is not specified the division is in floating point.
2. If both N (normalize) and M are not specified the division is an unnormalized floating point operation.
3. If M is specified the two operands and their results are treated as fixed point numbers.
4. The content of ADR specifies the divisor which is brought into R register before the division process begins.

Table 26. FINST/PE Instruction Index

Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page	Mnemonic Code	Octal Code	Ref. Page
AD	3504	4-17	IXL	2310	4-59	NORN	2307	4-31
ADA	3505	4-17	IXLD	2712	4-62	OFB	2506	4-76
ADB	2606	4-22	JAG	3715	4-52	OR	2304	4-31
ADD	2604	4-23	JAL	3717	4-52	ORN	2306	4-31
ADEX	2500	4-24	JB	3503	4-54	RAB	3701	4-36
ADM	3414	4-17	JLE	3517	4-55	RTAL	3513	4-87
ADMA	3415	4-17	JLG	3315	4-55	RTAR	3512	4-88
ADN	3404	4-17	JLL	3317	4-55	RTG	2413	4-77
ADNA	3405	4-17	JLO	3311	4-57	RTL	2412	4-77
ADR	3506	4-17	JLZ	3313	4-57	SAB	3702	4-36
ADRA	3507	4-17	JME	3515	4-55	SAN	3702	4-38
ADRN	3406	4-17	JMG	3115	4-55	SAP	3701	4-38
ADRNA	3407	4-17	JML	3117	4-55	SB	3704	4-79
AND	2704	4-27	JMO	3111	4-57	SBA	3705	4-79
ANDN	2706	4-27	JMZ	3113	4-57	SBB	2607	4-82
ASB	2507	4-26	JSE	2513	4-59	SBEX	2501	4-83
			JSG	2113	4-59	SBM	3614	4-79
			JSL	2313	4-59	SBMA	3615	4-79
CAB	3700	4-33	JSN	3503	4-54	SBN	3604	4-79
CHSA	3700	4-35	JXE	2511	4-59	SBNA	3605	4-79
CLRA	2411	4-39	JXG	2111	4-59	SBR	3706	4-79
COMPA	2211	4-40	JXGI	2711	4-61	SBRA	3707	4-79
DV	3304	4-41	JXL	2311	4-59	SBRN	3606	4-79
DVA	3305	4-41	JXLD	2713	4-62	SBRNA	3607	4-79
DVM	3214	4-41	LB	2107	4-63	SCM	2104	4-85
DVMA	3215	4-41	LDA	2617	4-104	SETE	2514	4-69
DVN	3204	4-41	LDB	2700	4-104	SETE1	2515	4-69
DVNA	3205	4-41	LDD	2212	4-104	SETF	2516	4-69
DVR	3306	4-41	LDE	2114	4-69	SETF1	2517	4-70
DVRA	3307	4-41	LDE1	2115	4-69	SETG	2714	4-70
DVRM	3216	4-41	LDEE1	2116	4-69	SETH	2715	4-70
DVRMA	3217	4-41	LDG	2314	4-69	SETI	2716	4-70
DVRN	3206	4-41	LDH	2315	4-69	SETJ	2717	4-70
DVRNA	3207	4-41	LDI	2316	4-69	SHABL	3711	4-89
EAD	2010	4-45	LDJ	2317	4-69	SHABML	3713	4-91
EOR	2505	4-29	LDR	2701	4-104	SHABMR	3712	4-92
EQV	2504	4-30				SHABR	3710	4-90
ESB	2410	4-48				SHAL	3501	4-93
GB	2106	4-50	LDS	2702	4-104	SHAML	3511	4-95
IAG	3714	4-52	LDX	2703	4-104	SHAMR	3510	4-96
IAL	3716	4-52	LEX	2117	4-64	SHAR	3500	4-94
IB	3502	4-54	ML	3104	4-65	STA	2612	4-97
ILE	3516	4-55	MLA	3105	4-65	STB	2613	4-97
ILG	3314	4-55	MLM	3014	4-65	STR	2614	4-97
ILL	3316	4-55	MLMA	3015	4-65	STS	2615	4-97
ILO	3310	4-57	MLN	3004	4-65	STX	2616	4-97
ILZ	3312	4-57	MLNA	3005	4-65	SUB	2605	4-99
IME	3514	4-55	MLR	3106	4-65	SWAP	3103	4-100
IMG	3114	4-55	MLRA	3107	4-65	SWAPA	3303	4-101
IML	3116	4-55	MLRM	3016	4-65	SWAPX	3703	4-102
IMO	3110	4-57	MLRMA	3017	4-65	T3A	2105	4-103
IMZ	3112	4-47	MLRN	3006	4-65	TCY	3100	---
ISE	2512	4-59	MLRNA	3007	4-65	TCYS	3101	---
ISG	2112	4-59	MULT	2213	4-72	TCYX	3102	---
ISL	2312	4-59	NAND	2705	4-27	XD	2503	4-107
ISN	3502	4-54	NANDN	2707	4-27	XI	2502	4-108
IXE	2510	4-59	NEB	2210	4-73			
IXG	2110	4-59	NOR	2305	4-31			
IXGI	2710	4-61	NORM	2013	4-74			

SECTION B: PROCESSING ELEMENT ORGANIZATION

I. INTRODUCTION

Functionally, the PE logic elements (Figure 12) are partitioned into three sections: registers, data transfer and modification units, and interface units. Because the Processing Unit is considered a general purpose computer with the PE functioning as an arithmetic unit, the PE logic elements are used for the execution of the FINST/PE instruction repertoire. Knowing, however, the variety of the FINST/PE instruction, one should expect that these logic elements differ in size and logic and for this reason a brief description of these elements is provided in this section.

II. LOGIC ELEMENTS AND PE ORGANIZATION

A. PE Logic Elements

1. PE Registers. These are the logic elements that are used to hold data whose word format is shown in Figure 11. These data may be an operand or the result of an operation (arithmetic or nonarithmetic). The PE registers are the following:

a) A Register (RGA): This register holds
   (1) the augend in addition
   (2) the minuend in subtraction
   (3) the multiplicand in multiplication
   (4) the dividend in division (most significant 48 bits)

It is also used as an accumulator because it receives the result from the Carry Propagating Adder at the end of each arithmetic operation and the result of the nonarithmetic operations from the logic elements involved.

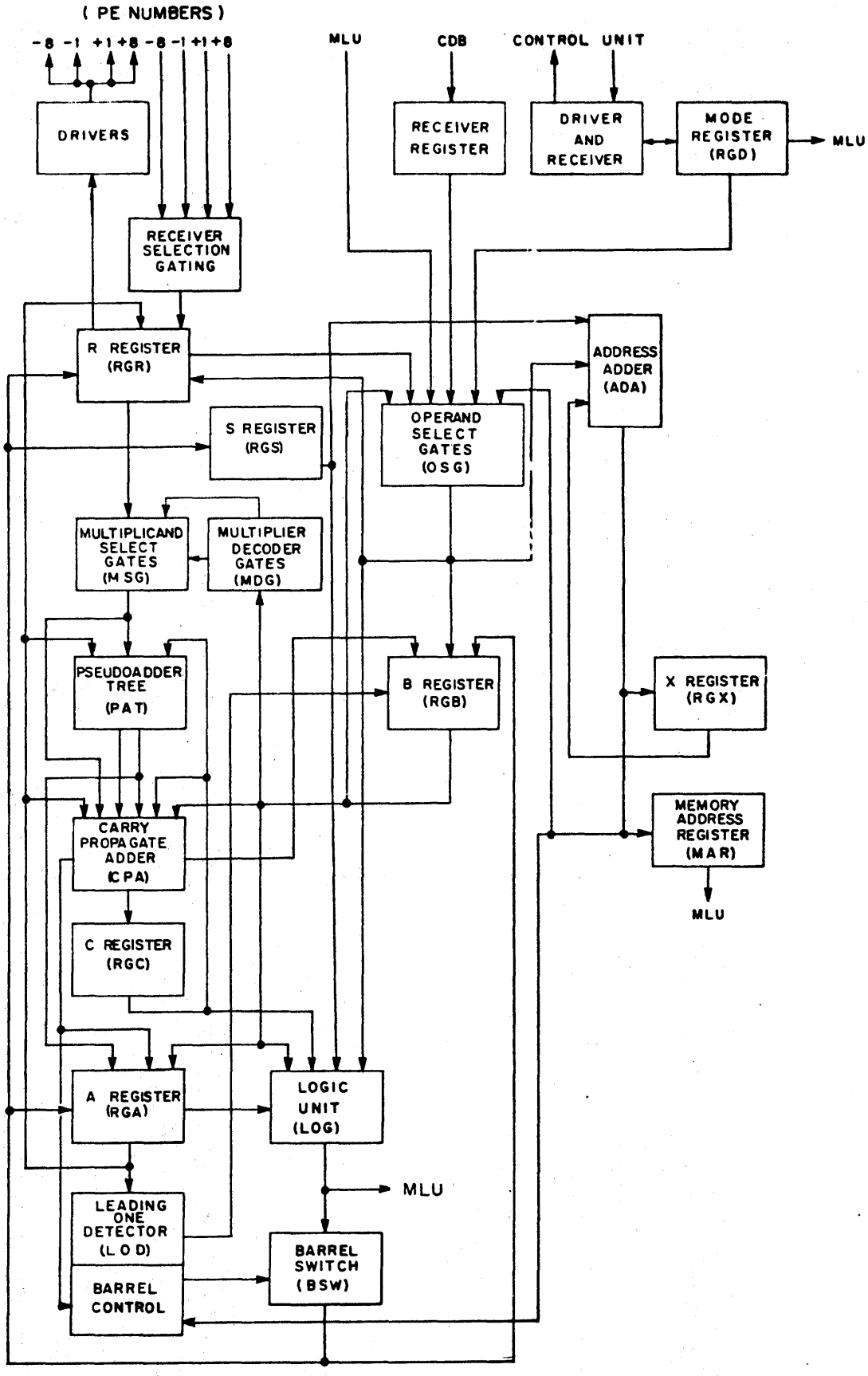


Figure 12. Processing Element Block Diagram

There are eight cards in the PE comprising the RGA, as follows:

RGA 1, RGA 2 — These are A01-A type cards holding the exponent part of the word and also providing special gating for the sign of the mantissa(s).

RGA 3, RGA 4, RGA 5, RGA 6, RGA 7, RGA 8 — These are A01 type cards and are used to hold the mantissa part(s) of the word whose format is shown in Figure 11.

Both types of cards take care of eight bits each as follows:

RGAi	RGA1	RGA2	RGA3	RGA4	RGA5	RGA6	RGA7	RGA8
Bit Position	0—7	8—15	16—23	24—31	32—39	40—47	48—55	56—63

The logic of A register is shown in "Logic Schematic Gated Register A01 and A01-A."

b) B Register (RGB): This register holds

(1) the addend in addition

(2) the subtrahend in subtraction

(3) the multiplier in multiplication*

(4) the dividend in division (least significant 48 bits)†

*In multiplication the multiplicand is temporarily stored in R register while B register is used to provide the space for the partial product and also to provide the inputs to the multiplier decoding gates.

†In division the divisor is stored in R register and B register holds the least significant 48 bits (mantissa) of the dividend, which is usually 96 bits long (only the mantissa part). If the option of rounding, however, is used, then B register receives half of the quantity or all of the divisor from R register but still this quantity is considered as part of the dividend. This register is also used to hold the result whenever operations involving double length operands are performed.

There are eight PE cards comprising the RGB, as follows:

RGB 1, RGB 2 — These are A01-A type cards holding the exponent part of the word and also providing special gating for the sign of the mantissa(s).

RGB 3, RGB 4, RGB 5, RGB 6, RGB 7, RGB 8 — These are A01 type cards and are used to hold the mantissa part(s) of the word whose format is shown in Figure 11.

Both A01 and A01-A cards take care of eight bits each as follows:

RGBi	RGB1	RGB2	RGB3	RGB4	RGB5	RGB6	RGB7	RGB8
Bit Position	0——7	8——15	16——23	24——31	32——39	40——47	48——55	56——63

c) C Register (RGC): This register is used for saving carries (partial) from the Carry Propagating Adder during the execution of multiplication. These partial carries are fed back to the PAT during each iterative cycle, but in the final cycle the carries are brought into CPA in order to form the final sum (product).

There are four PE cards of A08 type which can take care of 16 bits each as follows:

RGCi	RGC1	RGC2	RGC3	RGC4
Bit Position	16—31	32—47	48—63	64—79

The logic of C register is shown in "Logic Schematic Gated Register A08."

d) R Register (RGR): This register is used for

(1) communication with other PE's (+8, +1, -8, -1); this is known as routing.

(2) temporary storage of one of the operands (i.e., the multiplicand in multiplication).

(3) holding the divisor in division.

(4) extended addition and subtraction.

There are eight PE cards of A01 type which can take care of eight bits each as follows:

RGRi	RGR1	RGR2	RGR3	RGR4	RGR5	RGR6	RGR7	RGR8
Bit Position	0——7	8——15	16—23	24—31	32—39	40—47	48—55	56—63

The R register communicates with PAT through the MSG during multiplication, with the CPA and B register during division (see division process), and with the receiver selection gating and drivers during routing. Because R register is not protected by E, E1 bits, the programmer should be very careful about how the operand stored in R register is used. The logic of R register is shown in "Logic Schematic Gated Register A01."

e) S Register (RGS): This register is a spare register, which may be used for temporary storage of an operand for subsequent instructions. This results in saving time (memory cycles). The 16 least significant bits of this register (48 — 63) may also be used for indexing purposes whenever an additional index is required.

There are four PE cards of A08 type in the S register and each card accommodates 16 bits as follows:

RGSi	RGS1	RGS2	RGS3	RGS4
Bit Position	0——15	16—31	32—47	48—63

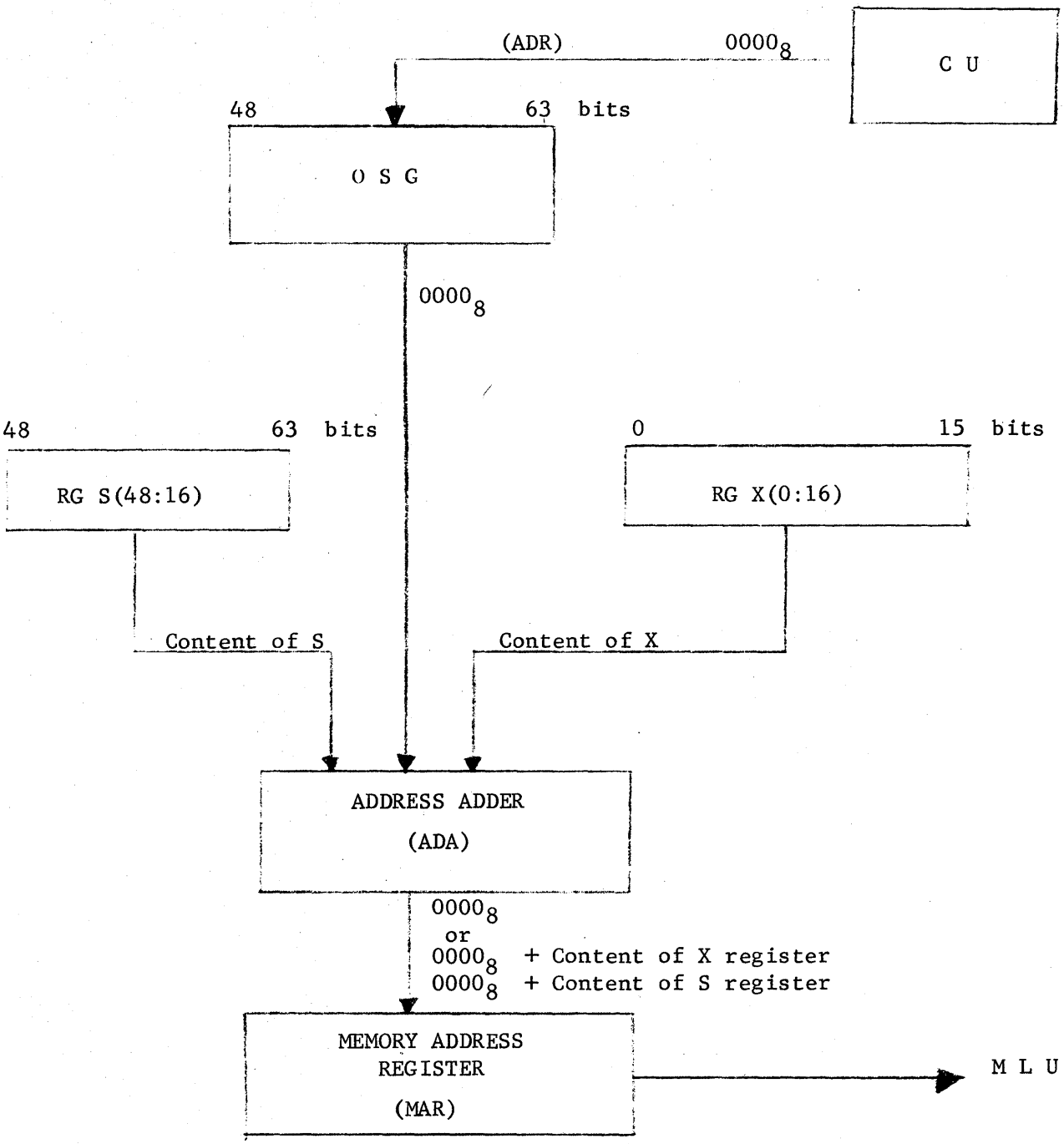
The S register communicates with the Address Adder (16 least significant bits) to index the address, with the logic unit (LOG) in order to provide the operand which has been stored temporarily in the appropriate register, and with the barrel switch in order to receive the operand for storage from RGA, RGB, RGR, RGX, or PEM.

The logic of S register is shown in "Logic Schematic Gated Register A08."

f) X Register (RGX): This is a 16-bit register used for indexing purposes. The type of card used is A08 which takes care of 16 bits through 16 latches.

The X register communicates with RGB, RGR, RGS, and PEM through the Address Adder (ADA) and Operand Select Gates (OSG). Because it has only 16 bits, whenever transfers are being made during the transmit instructions, only the 16 least significant bits of the source register are enabled into X register. For transfers from X register to the above-mentioned registers, their 48 most significant bits are cleared and not loaded.

As shown in Figure 13, the content of X register may specify indexing of the memory address, in which case the output of ADA (ADR+RGX) is brought into the Memory Address Register (MAR) and thereafter into MLU or it may specify indexing of shift count N (see details in shift instructions) in which case the output of ADA (N indexed by RGX) is brought into the Shift Count Register (LOD4). Because the index amount for memory address from ADA can be brought into X register, it can be said that the content of RGX = ADR or (ADR + content of RGX) or (ADR + content of RGS).



NOTE: In this illustration, the ADR value 0000_8 was used arbitrarily as an example. Continuing this example, the following instructions have the meanings shown:

(a) TCY: Transfer 0000_8 into MAR
(b) TCYX: Transfer 0000_8 + content of RGX into MAR
(c) TCYS: Transfer 0000_8 + content of RGS into MAR

Figure 13. Memory Address Chain (Example)

g) Memory Address Register (MAR): This is an A08-type-card, 16-bit register used to drive the 11 (presently used) bits of address to the PEM through MLU.

The content of MAR (Figure 13) may be the address specified by ADR field indexed by the contents of RGX or RGS. If indexing is not required, it is evident that the content of MAR is ADR itself.

h) Shift Count Register (SCR): This eight-bit register holds the shift amount and direction of shifting. It controls the Barrel Switch Controls (LOD1, 2, 3) which in turn control the three levels of the Barrel Switch and is used in both 64- or 32-bit modes of operation.

The shift count N (see equation in shift instructions) is received by the Operand Select Gates (OSG) over the Common Data Bus (CDB) path in bits 58 through 63 and from there it is brought into the Address Adder (ADA) where it may be indexed by the contents of RGX or RGS. The output of ADA (bits 10 through 15) is enabled into the six least significant bits of the shift count register, but always modulo 64 for the 64-bit mode. If the shift is done in 32-bit mode, the Inner and Outer words are acted upon separately and the shift amount is the shift count modulo 32.

The Barrel Switch has been designed to always shift right, but a left shift can be accomplished if the amount of left shift is converted appropriately to a right shift that produces the same result. The bits which specify the direction of shift (Table 27) are enabled into the OSG (bits 56, 57) over the CDB path and from there they are brought directly to the SCR (LOD4) at bit positions specified by SHL and SHR (Table 28). For shift left the SCR receives from CU through ADA the shift count N-1. This number may be indexed at ADA as in the case of right shift. Because the 2's complement of the left shift number is required, the output of the SCR is complemented (1's complement) and then applied to LOD1, 2, 3 in order to control the Barrel Switch levels.

Table 27. Shift Count Register Bit Organization

FROM OSG BIT 56 LEFT SHIFT	FROM OSG BIT 57 RIGHT SHIFT	← OUTPUT OF ADA →					
		bit 10	bit 11	bit 12	bit 13	bit 14	bit 15
		← OUTPUT OF CPA WHEN ALIGNED →					
		bit 74	bit 75	bit 76	bit 77	bit 78	bit 79
		66	67	68	69	70	71
			a1		a2		a0

Table 28. Shift Direction Truth Table

STATE OF BIT AT SHL	STATE OF BIT AT SHR	SHIFT DIRECTION PERFORMED
0	0	Not applicable
0	1	Right end off
1	0	Left end off
1	1	End around

The SCR not only generates the shift amount received from CU when a shift operation is requested through a CU decision, but also stores the shift amount, in the case of alignment (addition or subtraction), received from CPA, bits 74 through 79 in 64-bit mode or 32-bit mode for the Inner word or CPA bits 66 through 71 in 32-bit mode for the Outer word.

Since there are actually three levels and four possible displacements in each level of the Barrel Switch, the shift count (SC) for a right shift may be described by the following equation:

$$S.C.\ right = \sum_{i}^{2} a_i r^i = a_0 + a_1 r^1 + a_2 r^2$$

where the value of a_i (i = 0, 1, 2) may be any of the combinations of "0" and "1" at the SCR (Figure 12) and therefore may equal 0, 1, 2, or 3. The

symbol r represents a number with a base of four, because as was mentioned earlier, the SCR sends the shift amount to LOD1, 2, 3 to control the Barrel Switch shifting levels which can shift by 0, 1, 2, 3 (4th level), 0, 4, 8, 12 (3rd level), and 0, 16, 32, 48 (2nd level). Thus the above equation can be written in a more explicit form as follows:

$$\text{S.C. right} = \sum_{i=0}^{2} a_i r^i = a_0+4a_1+16a^2$$

where a_0 controls the 4th level of the Barrel Switch, a_1 controls the 3rd level of the Barrel Switch, and a_2 controls the 2nd level of the Barrel Switch.

In order to left shift, the content of the SCR is complemented (1's complement) and therefore the shift count for left shift may be described by the following equation:

$$\text{S.C. left} = 64_{10} - \sum_{i=0}^{2} a_i r^i$$

i) Mode Register (RGD): This is an A13-A-type-card, eight-bit register which is used to store results of instructions executed in the PE or results coming from the CU. It is also used to specify the status of a PE.

(1) E, E1 bits: These bits are called enable bits, because they control the gating clocks (clear, load) for the word (Figure 11) stored in PE registers A, S, and X as follows. When E bit is disabled, the Outer word (bits 0-7, 40-63) of A and S registers and the contents of the index register RGX are protected (that is, the Outer word contained in the register is not affected by any instruction). When E1 is disabled the Inner word (bits 8-39) is protected. In 32-bit mode the E, E1 bits are independent of one another, while in 64-bit mode they must both be set programmatically to the same state (E = E1).

The E and E1 bits control the movement of data from the PE to the PEM (PE write data) and from the PE to the CUB (transfer data). The E bit enables one half (32 bits) of the PE write/transfer data path through the MLU; the E1 bit enables the other half of that data path through the MLU. The E and E1 bits protect the PEM only when a Write operation from the PE to the PEM is performed, but they are ignored regardless of their state if a Write operation from the Input/Output Subsystem (IOSS) or the CU is requested. Table 29 lists the various states of the E, E1 bits and the indicated operation.

Table 29. E, E1 Bits Truth Table

ENABLE BITS		64-BIT MODE (FULL WORD)	32-BIT MODE	
E	E1		INNER WORD	OUTER WORD
0	0	Is disabled	Is disabled	Is disabled
0	1	These conditions should be avoided because they cause undefined results.	Is enabled	Is disabled
1	0		Is enabled	Is disabled
1	1	Is enabled	Is enabled	Is enabled

The E, E1 bits can be set or reset by the LD(E, E1, EEL) instructions, in which case the E, E1 bits are loaded from ACAR, and by the SET(E, E1) instructions, in which case the E, E1 bits are set with the result of a logic function (see mode register instructions).

(2) F, F1 bits: These bits are used to indicate a fault due to any of the following conditions:

- Exponent overflow
- Exponent underflow if normalization takes place and the resultant mantissa is not zero, or in floating point multiply or divide
- Mantissa overflow in fixed point arithmetic
- Zero divisor
- Unnormalized divisor (the divisor is always assumed to be normalized)

The F bit, when present, indicates a fault in 64-bit mode or in 32-bit mode for the Outer word, while the F1 bit indicates a fault in the Inner word (in the 32-bit mode). The setting or resetting of the F, F1 bits can be made by the SET(F, F1) instructions, in which case these bits are set with the result of a logic function (see mode register instructions) independently of the state of E, E1 bits. The setting of the F and F1 bits, due to the presence of a fault, as a result of the above conditions, depends upon the state of the E and E1 bits.

(3) G, H, I, J bits: These bits are used to store results of certain instructions (compare, indexing test, etc.) They can be used individually or combined in pairs as follows:

- I, G: When the instruction involves operands in 64-bit mode, I is used to hold the result. In 32-bit mode, however, the I bit is used when the Outer word is involved, while G holds the result when the Inner word is involved.
- H, J: In 64-bit mode the J bit is used to hold the result, while in 32-bit mode J bit holds the result if the Outer word is involved or H is used for the result if the Inner word is involved.

For more details, see the instructions involving the use of the mode register.

At the beginning of this manual it was said that the mode register bits can be controlled by signals from the CU. Table 30 shows these (code) signals and the resulting status of the mode bits.

Table 30. CU (Code) Signal for Mode Register

CU (CODE) SIGNALS	MODE REGISTER BITS							
	E	E1	F	F1	I	G	J	H
FYELD1IH-T	1	0	1	0	1	0	1	0
FYELD1OH-T	0	1	0	1	0	1	0	1
FYELD1EHJT	1	1	0	0	0	0	1	1
FYELD1IHJT	0	0	0	0	1	1	1	1

In summary, the mode register is the means by which the CU knows how many PE's are in 64-bit or 32-bit mode; i.e., by the state of the mode bits (E, E1). For this and other reasons explained previously, the mode bits can be sent to or received from the CU over the mode lines, so that constant monitoring can be achieved. The mode bit may be conditionally cleared by any PE (as a result of any arithmetic or logic operation).

2. PE Data Transfer and Modification Units: These are logic elements, which are used to modify data according to a particular instruction or to transfer this data from one logic element to another. The PE data transfer and modification units include Adder (CPA and CLA), ADA, Barrel Switch, LOD, LOG, OSG, PAT, MSG, and MDG.

a) Adder (Carry Propagating Adder and Carry Look Ahead): During the execution of the arithmetic operations in ILLIAC IV, the Carry Propagating Adder (CPA) is used to add two operands and the Carry Look Ahead (CLA) adder is used to determine whether a "Carry" is required, depending upon the two operands to be added. If there is a Carry it is properly fed into the CPA, which produces the final Sum within a clock time period. It is apparent, however, that during the time when the Carry is generated and which takes approximately nine ECL gates time delay (55 ns), the two operands to be added must be present at the Select gates of the CPA.

As indicated above, the Adder is subdivided into two parts, namely, the CPA which consists of 16 cards (A05) with each Card (group) taking care of four bits at a time and the CLA which consists of two cards (A11) with each card taking care of two sections (eight groups) at a time.

Because the Adder participates in the formation of the result during the execution of the basic arithmetic operations, it needs, in addition to the two operands to be added to one another, Control Signals to enable the Adder to perform each specific arithmetic operation. At this point, only the mechanization for the formation of the Sum of two operands is discussed. Later, when the multiplication and division processes are described the Adder (CPA in particular) is again discussed with regard to those peculiar characteristics that are essential to the implementation of the basic arithmetic operations.

The Adder can be looked upon as a five-stage grid where each stage, with the exception of the last one, participates in the formation of the bit carries, which, along with the original operands, form the final sum within one clock time period (Figure 14).

Stage #1

This stage generates 64 "Bit Transmits" (BT) and 64 "Bit Generates" (BG) according to the following equations:

$$BT_i = A_i \oplus B_i$$

$$BG_i = A_i \cdot B_i$$

where

A_i = ith bit of A register
B_i = ith bit of B register
$i = 0, 1, \ldots, 63.$

Stage #2

Each CPA card takes care of four bits and the output of Stage #1 is brought into Stage #2 to form 16 "Group Transmits" (GT) and 16 "Group Generates" (GG) according to the following equations:

(1) Mantissa Part

$$GT_i = BT_{j+15} \cdot BT_{j+16} \cdot BT_{j+17} \cdot BT_{j+18}$$

$$GG_i = BG_{j+15} + BG_{j+16} \cdot BT_{j+15} + BG_{j+17} \cdot BT_{j+16} \cdot BT_{j+15} + BG_{j+18} \cdot BT_{j+17} \cdot BT_{j+16} \cdot BT_{j+15}$$

where $i = 1, 2, 3, \ldots, 12$ and $j = 1 + 4(i-1) = 4i-3$

(2) Exponent Part

$$GT_i = BT_j \cdot BT_{j+1} \cdot BT_{j+2} \cdot BT_{j+3}$$

$$GG_i = BG_j + BG_{j+1} \cdot BT_j + BG_{j+2} \cdot BT_{j+1} \cdot BT_j + BG_{j+3} \cdot BT_{j+2} \cdot BT_{j+1} \cdot BT_j$$

where $i = 13, 14, 15, 16$ and $j = 4(i-13)$.

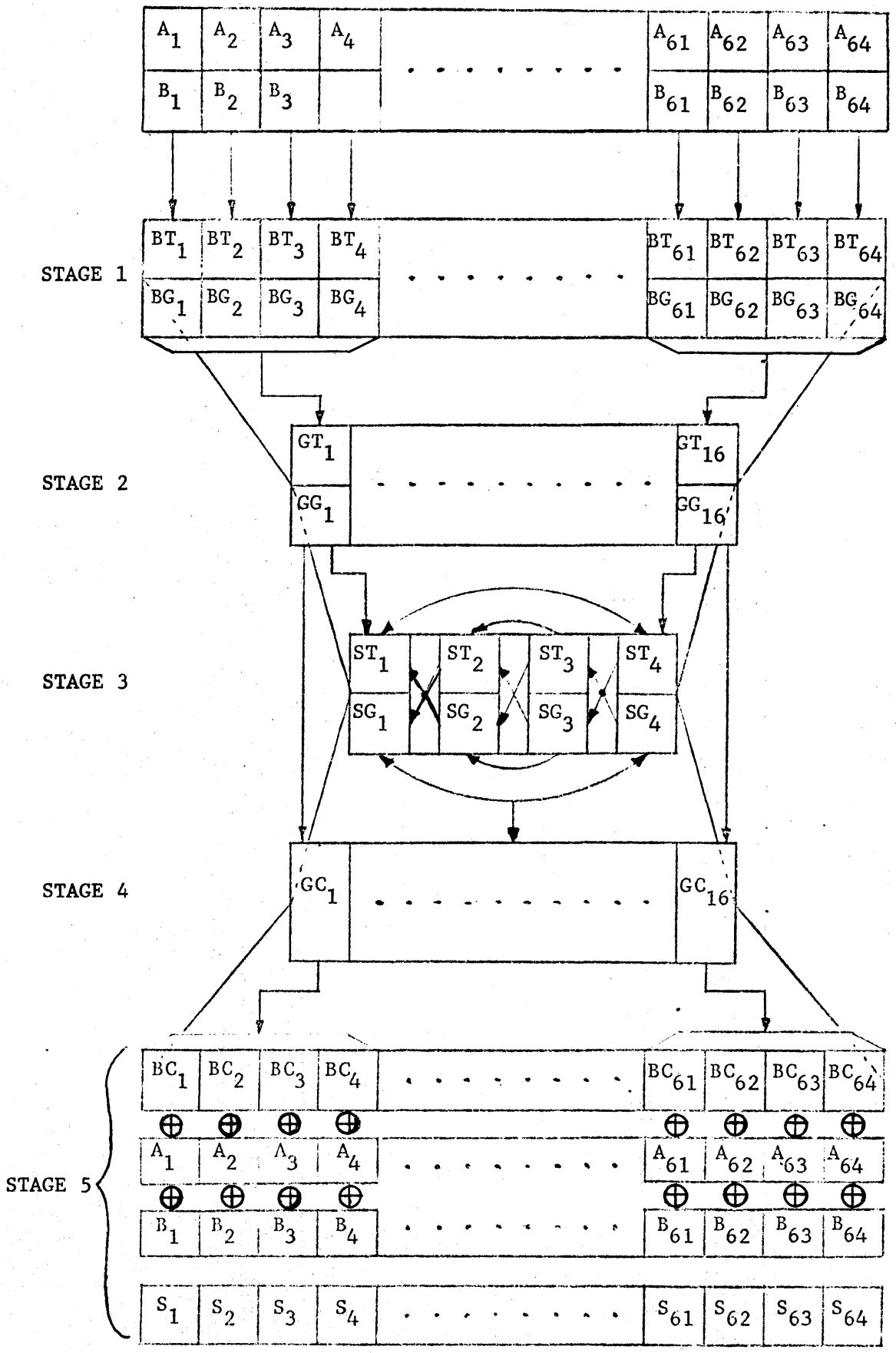


Figure 14. ILLIAC IV Processing Element Adder

The above equations indicate that, on each CPA Card, there is only one Group Transmit and one Group Generate gated out and at that stage there is no communication among the CPA Cards for the formation of the Group Transmit and Group Generate. However, each Group Transmit and Group Generate is a function only of the operand bits taken care of by the particular CPA Card and is not influenced by the output of its preceding CPA Card, where the order of precedence is from right to left.

Stage #3

The 16 Group Transmits and Group Generates from the CPA are gated into a different type of card, namely, the Carry Look Ahead (CLA) consisting of four Sections where each Section takes care of four Group Transmits and Group Generates. These Sections interchange information among themselves so that the Group Carries generated in this stage are a function not only of the Section Transmits and Generates but also of the Carry into each Section. To be more specific, each Section produces a "Section Transmit" (ST) and a "Section Generate" (SG) according to the following equations:

$$ST_i = GT_j \cdot GT_{j+1} \cdot GT_{j+2} \cdot GT_{j+3}$$

$$SG_i = GG_j + GG_{j+1} \cdot GT_j + GG_{j+2} \cdot GT_{j+1} \cdot GT_j + GG_{j+3} \cdot GT_{j+2} \cdot GT_{j+1} \cdot GT_j$$

where $i = 1, \ldots, 4$ and $j = 1 + 4(i-1) = 4i-3$.

These Section Transmits and Section Generates feed into all Sections of the Carry Look Ahead in order to form the Carry (Incoming Carry) for each Section, which, along with the Group Transmits and Group Generates, forms four Group Carries per Section.

The Incoming Carry for Section i (ICS_i) is given by the following equation:

$$ISC_i = SG_{[i+1]} + SG_{[i+2]} \cdot ST_{[i+1]} + SG_{[i+3]} \cdot ST_{[i+2]} \cdot ST_{[i+1]} + SG_{[i+4]} \cdot ST_{[i+3]} \cdot ST_{[i+2]} \cdot ST_{[i+1]}$$

where $i = 1, 2, 3, 4$ and [] = modulo 4.

The reader should bear in mind that since there are only four Sections in the Carry Look Ahead, the above general equation is consistent

with the implementation of the Adder only when it is assumed that the Section with subscript 0 on the right side of the equation is the same as the Section with subscript 4. Also, it is important to note that the Section Transmits and Section Generates are used only on Stage #3 to form the Incoming Carry for each Section; they are never gated into Stage #4, which belongs to the CPA.

Stage #4

The output of the CLA, in terms of the 16 "Group Carries" (GC), is fed back to the CPA, where the formation of 64 Bit Carries takes place. Because all the Group Carries within the same Section do not have the same number of terms, there are four general equations that describe the 16 Group Carries.

$$(1)\quad GC_i = GC_{i+1} + GG_{i+2} \cdot GT_{i+1} + GG_{i+3} \cdot GT_{i+2} \cdot GT_{i+1} + ICS_j \cdot GT_{i+3} \cdot GT_{i+2} \cdot GT_{i+1}$$

where $i = 1, 5, 9, 13$ and $j = 1 + (i-1)/4 = (i+3)/4$.

$$(2)\quad GC_i = GG_{i+1} + GG_{i+2} \cdot GT_{i+1} + ISC_j \cdot GT_{i+2} \cdot GT_{i+1}$$

where $i = 2, 6, 10, 14$ and $j = 1 + (i-2)/4 = (i+2)/4$.

$$(3)\quad GC_i = GG_{i+1} + ISC_j \cdot GT_{i+1}$$

where $i = 3, 7, 11, 15$ and $j = 1 + (i-3)/4 = (i+1)/4$.

$$(4)\quad GC_i = ICS_j$$

where $i = 4, 8, 12, 16$ and $j = 1 + (i-4)/4 = i/4$.

Stage #5

Each of the 64 "Bit Carries" (BC) is a function of the Group Carry into each CPA and the Bit Transmits and Bit Generates which precede that particular bit within the CPA. As in the case of the Group Carries, however, where the lower-order Group Carry within the Section, with the order of significance taken from left to right, is a function only of the

Incoming Carry for the Section to which the particular Group Carry belongs, the least significant Bit Carry within the Group is only a function of the Group Carry for the Group (CPA Card) to which the particular Bit Carry belongs.

For the reasons explained in the case of Group Carries, the 64 Bit Carries are described by the following equations:

(1) Mantissa Part

- $BC_i = BG_{i+1} + BG_{i+2} \cdot BT_{i+1} + BG_{i+3} \cdot BT_{i+2} \cdot BT_{i+1} + GC_j \cdot BT_{i+3} \cdot BT_{i+2} \cdot BT_{i+1}$

where $i = 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60$ and $j = 1 + (i-16)/4 = (i/4) - 3$.

- $BC_i = BG_{i+1} + BG_{i+2} \cdot BT_{i+1} + GC_j \cdot BT_{i+2} \cdot BT_{i+1}$

where $i = 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61$ and $j = 1 + (i-17)/4 = (i-13)/4$.

- $BC_i = BG_{i+1} + GC_j \cdot BT_{i+1}$

where $i = 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62$ and $j = 1 + (i-18)/4 = (i-14)/4$.

- $BC_i = GC_j$

where $i = 19, 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63$ and $j = 1 + (i-19)/4 = (i-15)/4$.

(2) Exponent Part

- $BC_i = BG_{i+1} + BG_{i+2} \cdot BT_{i+1} + BG_{i+3} \cdot BT_{i+2} \cdot BT_{i+1} + GC_{j+13} \cdot BT_{i+3} \cdot BT_{i+2} \cdot BT_{i+1}$

where $i = 0, 4, 8, 12$ and $j = i/4$.

- $BC_i = BG_{i+1} + BG_{i+2} \cdot BT_{i+1} + GC_{j+13} \cdot BT_{i+2} \cdot BT_{i+1}$

where i = 1, 5, 9, 13 and j = (i-1)/4.

- $BC_i = BG_{i+1} + GC_{j+13} \cdot BT_{i+1}$

where i = 2, 6, 10, 14 and j = (i-2)/4.

- $BC_i = GC_{j+13}$

where i = 3, 7, 11, 15 and j = (i-3)/4.

Stage #6

This is the final stage where the 64 Bit Carries from Stage #5, along with the two 64 operand bits that are still present at the CPA Select gates, produce the final sum according to the equation $S_i = A_i \oplus B_i \oplus C_i$ which when reduced gives

$$S_i = A_i \cdot B_i \cdot C_i + \bar{A}_i \cdot \bar{B}_i \cdot C_i + A_i \cdot \bar{B}_i \cdot \bar{C}_i + \bar{A}_i \cdot B_i \cdot \bar{C}_i$$

where A_i = ith bit of A register, B_i = ith bit of B register, C_i = ith Bit Carry, and i = 0, 1, ..., 63.

The sum (S_i), once it is formed, is fed into the A register, which plays the role of an Accumulator. The Select gates of the CPA for the Addition of two operands allow only the Bit Carries and the two operands into the exclusive OR gate for the formation of the sum and, if there are any Carries as the result of that Add operation, they are ignored.

In the case of Multiplication, however, during the iterative cycles where the partial sum is generated, the Carries from the Pseudo Adder Tree (PAT) are fed into the Select gates, which provide the same path as for the Bit Carries in the case of Addition. In the same way, the partial sum (PATS) is fed into the gates which are used for the "A_i path" of the above equation and Word #4 (WD4) is gated into the gate used for B_i of the same equation. During these iterative cycles the Carries out of the CPA are allowed to be stored in the C register; in this case the CPA acts as a Carry Save Adder while the sum is stored in the A and B registers. In

the final cycle of Multiplication the partial sum in the A register is brought into the appropriate Select gate, while the Carries from the C register are brought into the Select gates that take care of WD4 during the iterative cycles of Multiplication or B register during Addition. The final sum is produced in the same way as explained for the Addition of two operands.

It is evident, therefore, that during the iterative cycles of Multiplication where the partial product (SUM) is produced only the CPA (operating as a Carry Save Adder) is used, but in the last cycle, where the final product (SUM) is produced, the Adder (CPA and CLA) is used in the same way as for the Addition of two operands. However, if there are any Carries out of the CPA they are not allowed into the C register because they are not needed.

During the recursive process for the formation of the quotient field and remainder as a result of the Division of two operands, the Adder (CPA and CLA) is used in a way similar to its use in the case of Addition with the exception that, if the Subtraction (Division) is successful, the remainder, which is nothing else but the sum in the CPA, is brought into the A register (through the wires which are effectively one position to the left with respect to the wires bringing the sum into the A register for any Arithmetic operation other than Division). In this case the CPA is like a shift register that allows a "one-bit" left shift.

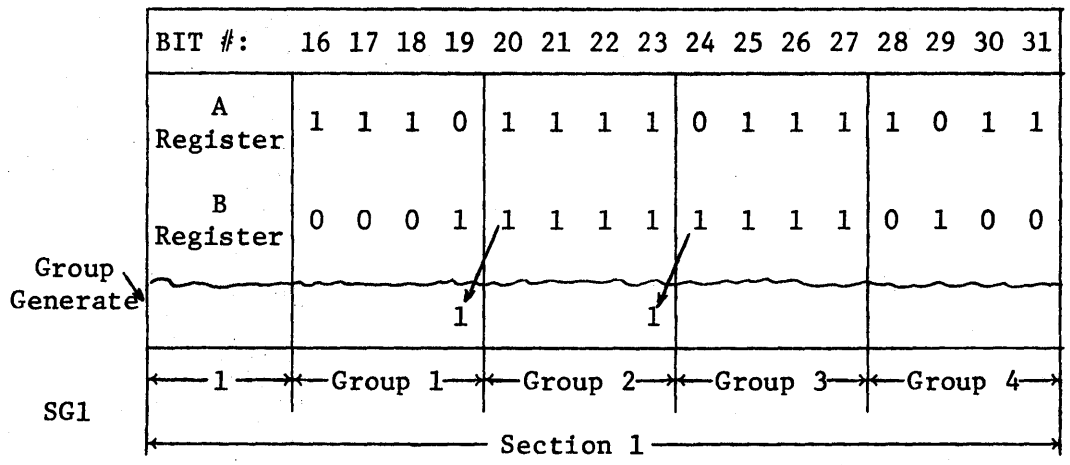
Example

REGISTER	16															31
Given A =	1	1	1	0	1	1	1	1	0	1	1	1	1	0	1	1
Given B =	0	0	0	1	1	1	1	1	1	1	1	1	0	1	0	0

Given the operands A and B above show the state of Section Generate One (SG1).

Solution

The presentation below shows A and B operands and Carries (Group and Section) in the CPA and CLA.



$$BG_{16} = A_{16} \cdot B_{16} = 1 \cdot 0 = 0$$
$$BT_{16} = A_{16} \oplus B_{16} = 1 \oplus 0 = 1$$
$$BG_{17} = A_{17} \cdot B_{17} = 1 \cdot 0 = 0$$
$$BT_{17} = A_{17} \oplus B_{17} = 1 \oplus 0 = 1$$
$$BG_{18} = A_{18} \cdot B_{18} = 1 \cdot 0 = 0$$
$$BT_{18} = A_{18} \oplus B_{18} = 1 \oplus 0 = 1$$
$$BG_{19} = A_{19} \cdot A_{19} = 1 \cdot 0 = 0$$
$$BT_{19} = A_{19} \oplus B_{19} = 1 \oplus 0 = 1$$

$$\begin{aligned} GG_1 &= BG_{16} + BG_{17} \cdot BT_{16} + BG_{18} \cdot BT_{17} \cdot BT_{16} \\ &\quad + BG_{19} \cdot BT_{18} \cdot BT_{17} \cdot BT_{16} \\ &= 0 + 0 \cdot 1 + 0 \cdot 1 \cdot 1 + 0 \cdot 1 \cdot 1 \cdot 1 \\ &= 0 \end{aligned}$$

$$\begin{aligned} GT &= BT_{16} \cdot BT_{17} \cdot BT_{18} \cdot BT_{19} \\ &= 1 \cdot 1 \cdot 1 \cdot 1 \\ &= 1 \end{aligned}$$

By the same procedure as above,

$$GG_2 = 1 \qquad GG_3 = 1 \qquad GG_4 = 0$$
$$GT_2 = 0 \qquad GT_3 = 0 \qquad GT_4 = 1$$
$$SG_1 = GG_1 + GG_2 \cdot GT_1 + GG_3 \cdot GT_2 \cdot GT_1 + GG_4 \cdot GT_3 \cdot GT_2 \cdot GT_1$$
$$= 0 + 1 \cdot 1 + 1 \cdot 0 \cdot 1 + 0 \cdot 0 \cdot 0 \cdot 1 = 1$$

In other words, the Group Generates and Group Transmits of the CPA are fed into the Carry Look Ahead whose Group Carry outputs are fed back to the CPA which finally performs the sum.

$$ST_1 = GT_1 \cdot GT_2 \cdot GT_3 \cdot GT_4$$
$$= 1 \cdot 0 \cdot 0 \cdot 1$$
$$= 0$$

For this example, $SG_1 = 1$ which means that the next Section will have a Section Input Carry coming from Section 1 and therefore the Group Carries of the next Section are a function of this Input Section Carry and the Group Generates and Group Transmits which precede that particular Group within the next Section.

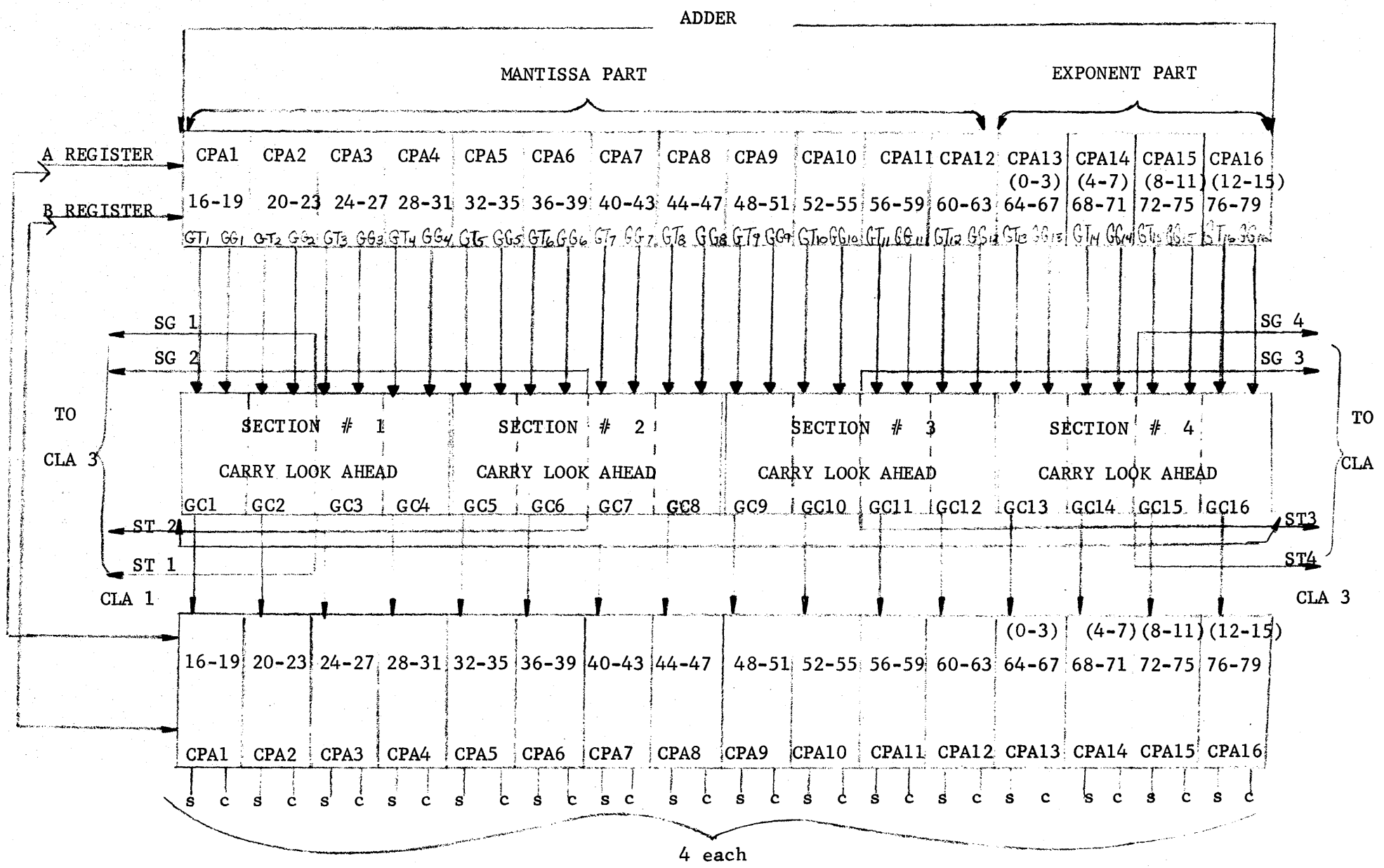
The sequence of operations of the CPA and Carry Look Ahead for the addition of two operands can be described as follows (Figure 15):

(1) In the CPA
   (a) the Bit Transmit and Bit Generates are produced.
   (b) the Group Transmits and Group Generates are produced. They are fed into the Section Carry Look Ahead.

(2) In the Carry Look Ahead the Group Carries are produced as a function of:
   (a) the Group Transmits and Group Generates from the CPA.
   (b) the Section Transmits and Section Generates from all the other Sections of the Carry Look Ahead.

(3) The CPA receives the Group Carries from the Carry Look Ahead and produces the Bit Carries as a function of:
   (a) the Incoming Group Carry from the Section Carry Look Ahead.
   (b) the Bit Transmits and Bit Generates which precede that particular bit within the Group (CPA).

(4) The CPA takes the Carries and, in conjunction with the two inputs from the operands to the particular bit position of the CPA, performs the addition, thus producing the sum in one clock time because there is no recycling of Carries involved.



s = sum
c = carry

Figure 15. Functional Block Diagram of CPA and CLA

b) Address Adder (ADA): This is a 16-bit register consisting of four A05 and one A11 type cards. The operation of ADA is exactly the same as that of CPA and CLA with the exception that its inputs are the outputs of the OSG, S, or X registers and its output is the input to the X register, the S register through OSG, LOD4 (Shift Count Register), and MAR (Table 1). The ADA receives the 11 bits of address from the OSG and if indexing by X or S is not required it passes the address through to the Memory Address Register or, if necessary, it can store the address in the X register. If indexing is requested, the amount of indexing from X or S registers is added to the address coming into the ADA from the OSG in the same way as was explained when describing the Adder. Because the ADA uses the same type of cards and operates in the same fashion as the Adder, it can be said that ADA1 through ADA4 and CLA5 comprise the Address Adder. Since there are only four ADA cards (similar to the one the CPA uses) it is evident that there are Group Transmits and Group Generates coming out of the ADA cards. However, Section Transmits and Section Generates are not used for the formation of the Incoming Carry for the Section, because there is only one Section (CLA5). In order to avoid confusion, however, the Section Transmit and Section Generate out of CLA5 are not used when the ADA is used as an address adder; but whenever two positive numbers being added produce a Carry, this is sent to the mode register as a Section Generate to indicate that an overflow has occurred. For specific instructions calling for comparison of X with other registers, the Section Transmit and Section Generate, which are strictly a function of the four Group Transmits and Group Generates, participate in the formation of the result which is stored in either the I or J bit of the mode register. If the Address Adder performs subtraction rather than addition, the quantity to be subtracted is complemented (2's complement). This is accomplished by forcing a ONE into the least significant bit of the ADA by the use of the FYE--Z3LD1 signal. No end around Carry is generated thus eliminating the need for additional logic which normally would be required to take care of a possible carry out of the ADA.

The operation of ADA is exactly the same as the one described for the Adder, but the reader should bear in mind that, even though the addition of two bits produces a Bit Transmit and Bit Generate according to the following equations

$$BT_j = K_i \oplus OSG_i$$

$$BG_j = K_i \cdot OSG_i$$

where

K_i = ith bit of the X or S registers

OSG_i = ith bit of ADR field through OSG

$$i = \begin{cases} 0, 1, \ldots, 15 & \text{if X register is gated into ADA} \\ 48, 49, \ldots, 63 & \text{if S register or OSG is gated into ADA} \end{cases}$$

the value of j depends upon the register gated into ADA (X or S register).

The Group Transmits and Group Generates are given by the following equations:

$$GT_i = BT_{j+47} \cdot BT_{j+48} \cdot BT_{j+49} \cdot BT_{j+50}$$

$$GG_i = BG_{j+47} + BG_{j+48} \cdot BT_{j+47} + BG_{j+49} \cdot BT_{j+48} \cdot BT_{j+47} + BG_{j+50} \cdot BT_{j+49} \cdot BT_{j+48} \cdot BT_{j+47}$$

where i = 17, 18, 19, 20 and j = 1 + 4(i-17).

These four Group Transmits and Group Generates form the Section Transmit and Section Generate and, along with the ONE forced into the least significant bit position of ADA, generate the Group Carry. For the reasons given in the explanation of the operation of the Adder, there are four equations to describe each one of the four Group Carries.

(1) $GC_{17} = GG_{18} + GG_{19} \cdot GT_{18} + GG_{20} \cdot GT_{19} \cdot GT_{18} + GC_{20} \cdot GT_{19} \cdot GT_{18}$

(2) $GC_{18} = GG_{19} + GG_{20} \cdot GT_{19} + GC_{20} \cdot GT_{20} \cdot GT_{19}$

(3) $GC_{19} = GG_{20} + GC_{20} \cdot GT_{20}$

(4) GC_{20} = FYE--Z3LD1

Once the Group Carries are formed in the Carry Look Ahead (CLA5), they are fed back to ADA cards to form the Bit Carries as follows:

$$(1)\quad BC_i = BG_{i+1} + BG_{i+2} \cdot BT_{i+1} + BG_{i+3} \cdot BT_{i+2} \cdot BT_{i+1} + GC_{j+16} \cdot BT_{i+3} \cdot BT_{i+2} \cdot BT_{i+1}$$

where $i = 0, 4, 8, 12$ and $j = 1 + i/4$.

$$(2)\quad BC_i = BG_{i+1} + BG_{i+1} \cdot BT_{i+1} + GC_{j+16} \cdot BT_{i+2} \cdot BT_{i+1}$$

where $i = 1, 5, 9, 13$ and $j = 1 + (i-1)/4 = (i+3)/4$.

$$(3)\quad BC_i = BG_{i+1} + GC_{j+16} \cdot BT_{i+1}$$

where $i = 2, 6, 10, 14$ and $j = 1 + (i-2)/4 = (i+2)/4$.

$$(4)\quad BC_i = GC_{j+16}$$

where $i = 3, 7, 11, 15$ and $j = 1 + (i-3)/4 = (i+1)/4$.

Because the ADA is an extension of CPA in that ADA 1 through 4 corresponds to CPA 17 through 20, the Groups (Transmits and Generates) follow this notation, while the Bit Carries are numbered from 0 to 15 in order to be consistent with the output of ADA.

It must be mentioned, however, that the content of ADA (indexed or not) does not always represent the PEM address, but may represent the Shift Count N (see Shift Count Register, subsection 1.h).

The 16 Carries along with the two operands (content of X or S registers and ADR field through OSG) which are still present at the Select Gates of ADA 1 through 4 produce the final sum described by the following equation:

$$Z_i = K_i \oplus OSG_i \oplus C_i$$

where K_i = ith bit of X or S register, OSG_i = ith bit of ADR field through OSG, C_i = ith bit Carry, and $i = 0, 1, \ldots, 15$.

c) Barrel Switch (BSW): The shifting operation is accomplished by the use of the Barrel Switch (BSW). The Barrel Switch has four levels and can shift from 0 to 63 bits to the left or right, either end around or end off in one clock period. The end around shift is performed when both the controls for left and right shift are true.

The Barrel Switch receives from the Logic Unit (LOG) a parallel input of 64 bits and passes these inputs through its first level without any shifting. It can, however, swap bytes by 0, 24, and 32, and then through the next three levels force four displacements for each bit in each level (Figure 16).

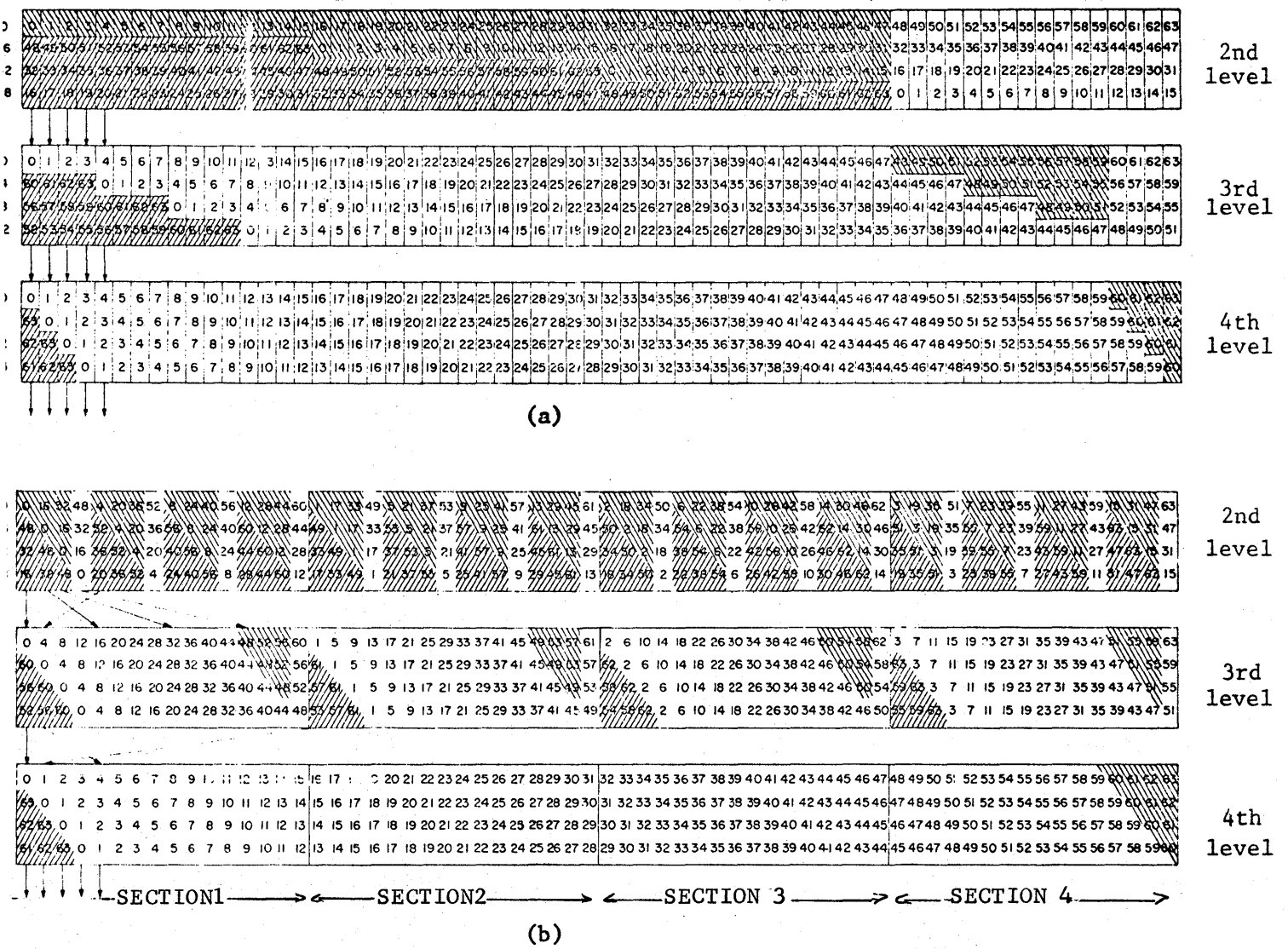


Figure 16. PE Barrel Switch: a) Description, b) Physical Configuration

Every level of the Barrel Switch is divided into four sections, each section accommodating 16 bits. The following tabular information, containing information about the specific level, quantity, and type of PE cards being used, bits being accommodated, and amount of shifting, is given to provide a better picture of the Barrel Switch. Information about the way the levels are connected between one another is also given below.

FIRST LEVEL OF THE BARREL SWITCH				
1st Section (Card)	2nd Section (Card)	3rd Section (Card)	4th Section (Card)	Amount of Shift
Bit Position	Bit Position	Bit Position	Bit Position	
0	1	2	3	
4	5	6	7	
8	9	10	11	
12	13	14	15	
16	17	18	19	N
20	21	22	23	
24	25	26	27	O
28	29	30	31	
32	33	34	35	N
36	37	38	39	
40	41	42	43	E
44	45	46	47	
48	49	50	51	
52	53	54	55	
56	57	58	59	
60	61	62	63	
Note: A06 card type used for all four sections				
SECOND LEVEL OF THE BARREL SWITCH				
0	1	2	3	M
4	5	6	7	U
8	9	10	11	L
12	13	14	15	T
16	17	18	19	I
20	21	22	23	P
24	25	26	27	L
28	29	30	31	E
32	33	34	35	S
36	37	38	39	
40	41	42	43	O
44	45	46	47	F
48	49	50	51	
52	53	54	55	1
56	57	58	59	6
60	61	62	63	
Note: A07-A card type used for all four sections				

THIRD LEVEL OF BARREL SWITCH				
1st Section (Card)	2nd Section (Card)	3rd Section (Card)	4th Section (Card)	Amount of Shift
Bit Position	Bit Position	Bit Position	Bit Position	
0	1	2	3	
4	5	6	7	M
8	9	10	11	U
12	13	14	15	L
16	17	18	19	T
20	21	22	23	I
24	25	26	27	P
28	29	30	31	L
32	33	34	35	E
36	37	38	39	S
40	41	42	43	
44	45	46	47	O
48	49	50	51	F
52	53	54	55	
56	57	58	59	4
60	61	62	63	
Note: A07-B card type used for all four sections				
FOURTH LEVEL OF BARREL SWITCH				
0	16	32	48	
1	17	33	49	M
2	18	34	50	U
3	19	35	51	L
4	20	36	52	T
5	21	37	53	I
6	22	38	54	P
7	23	39	55	L
8	24	40	56	E
9	25	41	57	S
10	26	42	58	
11	27	43	59	O
12	28	44	60	F
13	29	45	61	
14	30	46	62	1
15	31	47	63	
Note: A07-C card type used for Section 1	Note: A07-B card type used for Sections 2 & 3		Note: A07-C card type used for Section 4	

The sections of the Barrel Switch are signified by BS, followed by two decimal digits. The first digit refers to the level of the Barrel Switch and the second digit specifies the particular section (i.e., BS11 = first section of the first level of the Barrel Switch.

The levels of the Barrel Switch, by section, are connected as follows:

BS11 → BS21 → BS31
BS12 → BS22 → BS32
BS13 → BS23 → BS33
BS14 → BS24 → BS34

} Bit-by-bit correspondence of the Barrel Switch levels

BS31
BS32
BS33
BS34
} → BS41 bits 0 to 15

BS31
BS32
BS33
BS34
} → BS42 bits 16 to 31

BS31
BS32
BS33
BS34
} → BS43 bits 32 to 47

BS31
BS32
BS33
BS34
} → BS44 bits 48 to 63

For special shifting purposes there are additional interconnections between the levels of the Barrel Switch.

(1) The second level of the Barrel Switch is connected to two places of the third level of the Barrel Switch for bit positions listed below:

BS21 → BS31 bits 52 56 60
BS22 → BS32 bits 53 57 61
BS23 → BS33 bits 54 58 62
BS24 → BS34 bits 55 59 63

(2) The third level of the Barrel Switch (except Section 1 of that level) is connected to the fourth level as follows:

BS32 →	{ BS41 BS42	for bit 13
BS33	{ BS41 BS42	for bit 14
BS34	{ BS41 BS42	for bit 15
BS32	{ BS42 BS43	for bit 29
BS33	{ BS42 BS43	for bit 30
BS34	{ BS42 BS43	for bit 31
BS32	{ BS43 BS44	for bit 45
BS33	{ BS43 BS44	for bit 46
BS34	{ BS43 BS44	for bit 47
BS32	{ BS41 BS44	for bit 61
BS33	{ BS41 BS44	for bit 62
BS34	{ BS41 BS44	for bit 63

The physical configuration of the Barrel Switch as given by R. Davis of [4] (Figure 16) shows that any number (bit position) can be shifted a number of places to the left or right using the above information with respect to the interconnections between the levels of the Barrel Switch.

In order to be able to describe the shifting action, it is best to deal with one bit position and trace its path from the beginning to the end. Suppose bit position 32 is to be shifted 27 places to the right. This means that it will be shifted 16 to the right by the second level of

the Barrel Switch, the third level will shift it 8, and the fourth level 3, a total of 27 places, thus bringing the 32 bit position into bit position 59.

The input to the Barrel Switch is through the logic unit (LOG). It then passes through the first level of the Barrel Switch without being shifted at all and is then brought into the first section of the second level of the Barrel Switch where it is shifted 16. At the left end of BS21 (Figure 16(b)) the number 16 represents the amount of shifting. Following this row toward the right the fourth number (bit position 32) which was brought into this level goes into position 48 which is straight up one position on the previous (first) row. Since there is a bit-by-bit connection between the second and third levels of the Barrel Switch and because the third level is to shift it by 8, the output of the second level (bit position 48) is found on the third row of the third level still as bit position 48. After the shift (by 8) the output of the third level is bit position 56 (first row of the first section of the third level of the Barrel Switch). Since there is a connection of BS31 and BS44 for bits 48 to 63, the output of BS31 (bit position 56) becomes the input to the fourth section of the fourth level of the Barrel Switch, on the fourth row because it is to be shifted by 3. The output of this level and therefore the output of the Barrel Switch is bit position 59 (first row of the fourth section of the fourth level of the Barrel Switch).

A left shift is accomplished in the same fashion as a right shift except that the amount of right shift is taken as 64_{10} minus the amount of left shift desired. For example, if bit position 16 is to be shifted left 16 places, it will be brought into bit position 0. This is the same as if it were shifted 48 places to the right (64 - 16 = 48). Bit position 16 is in the fourth row of the first section of the second level of the Barrel Switch in Figure 16(b), which shows that after being shifted by 48 places the bit is in bit position 0.

As stated previously, the Barrel Switch can shift left or right either end around or end off. In an end around shift, there is simply a displacement of the bit position equal to the right shift amount when shifting right or 64_{10} (minus the left shift amount when shifting left) and therefore none of the bits is lost. In the case of shifting end off either right or left, as many bits as the number of shifts are lost. If,

for example, the mantissa of A register were to be shifted right end off 16 places, bit position 16 would be moved to bit position 32, bit position 17 would be moved to bit position 33, and finally bit position 47 would be moved to bit position 63. Bits 48 to 63 would be lost and bit positions 16 to 31 of A register after the shifting would contain zeros.

To see how the zeros are inserted into the mantissa of A register (bit positions 16 to 31, for example), refer to Figure 16(b). In the second row of the second level of the Barrel Switch, there are crosshatch lines directed upward and to the right covering the numbers:

48,52,56,60 49,53,57,61 50,54,58,62 51,55,59,63

These numbers when placed in sequence are 48 to 63 which are lost when shifting by 16. What is really happening from the hardware standpoint is that these bit positions are never enabled when shifting by 16 to the right end off. The result of this action (blocking) is that since these bit positions are not enabled, the output of the Barrel Switch corresponding to bit positions 0 to 15 is zero. In the meantime, bit positions 0 to 15 have been moved 16 places to the right and therefore occupy bit positions 16 to 31. Since only the mantissa was to be shifted right end off 16 places, bit positions 0 to 15 of the A register were not enabled to enter into the first level of the Barrel Switch, which means that zeros were inserted at the input of the first level of the Barrel Switch for bit positions 0 to 15.

In the same way, those bits not being enabled when shifting left can be found. These bits are covered by the crosshatch lines directed upward and to the left.

Defining:

C = the amount of shifting to the right end off

T = the amount of shifting to the left end off

E = the number of bit positions which become zero (0 through E)

H = the number of bit positions which become zero (H through 63)

then:

E = C - 1 for an end off right shift

H = 64 - T for an end off left shift.

As stated previously, the first level of the Barrel Switch does not do any actual shifting, but it can swap bytes. Another feature of the first level of the Barrel Switch is the capability it possesses to block certain bits from entering as inputs to the second level of the Barrel Switch. This is interpreted as if the Barrel Switch (second, third, and fourth levels) receives zeros at these particular inputs, which, after being shifted a number of places, force the output of the Barrel Switch corresponding to these bits to be zero. One reason for having this additional feature in the first level of the Barrel Switch is because it is possible for a certain portion of a register which cannot be blocked to come into the Logic Unit, since there are no controls enabling the logic unit by parts. Instead, the whole word from the Logic Unit is enabled into the first level of the Barrel Switch. This level therefore must have control signals which can enable the word by parts into the rest of the Barrel Switch.

The Barrel Switch is one of the most important logic elements of the PE. It participates in the execution of every arithmetic instruction which, without the use of the Barrel Switch, could not be executed as rapidly.

During the description of the Barrel Switch no mention was made of what causes the different levels of the Barrel Switch to be acted upon and a shift operation to take place. This shifting operation is a combination of two logic elements, namely, the Barrel Switch and the Leading ONES Detectors.

Each level of the Barrel Switch receives special controls which determine the actual amount of shifting (as in the case of alignment). These controls, prior to their application to the Barrel Switch, originate from the Shift Count Register (LOD4), whose two most significant bits are decoded and applied to the second level of the Barrel Switch as controls for shifting by 0, 16, 32, and 48. The other two bits (bit positions 2 and 3) of the shift counter are decoded and applied to the third level of the Barrel Switch as controls for shifting by 0, 4, 8, and 12. Finally, the two least significant bits of the shift counter, after being decoded, are applied to the fourth level of the Barrel Switch as controls for shifting by 0, 1, 2, and 3 (Figure 17).

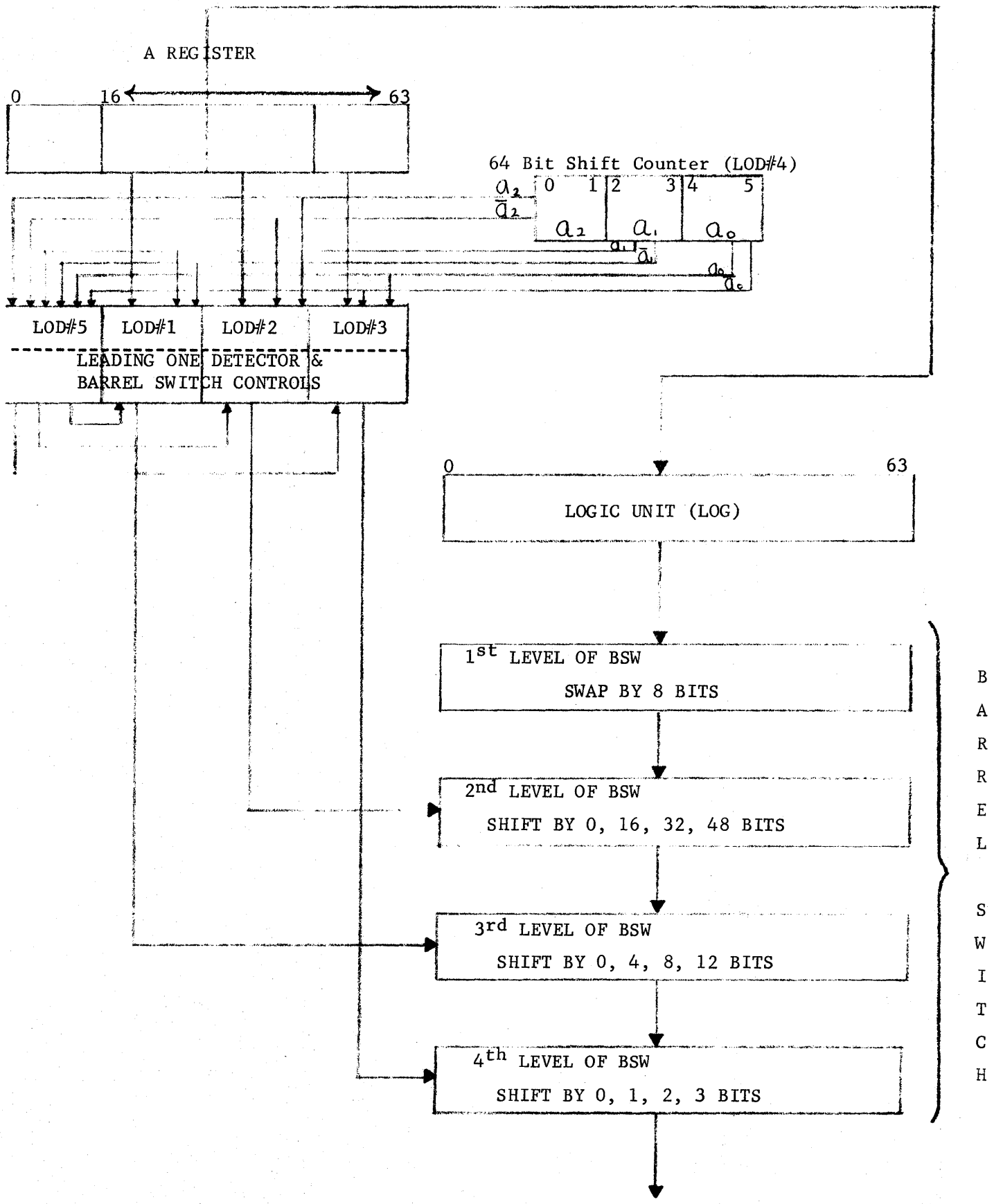


Figure 17. Functional Block Diagram of Barrel Switch and Leading ONES Detectors

The amount of shifting, which involves the Shift Count Register, is concerned with shifts for alignment or shifts due to CU decision (Shift Count N). The Barrel Switch, however, performs shift operations due to PE decisions (in case the option of normalization is used), in which case the amount of shift is determined by the Leading ONES Detectors 1, 2, 3, while the shift operation is executed by the Barrel Switch.

Example

Suppose the mantissa of A register is required to be shifted right end around by 3 (CU decision). Show illustratively the path of bit at position 20.

Solution

Figure 17 shows that the Shift Count Register (true output) sends the following number $000011_2 = 3_{10}$ to LOD 1, 2, 3. This number is decoded there and LOD3 which received $A_0 = 11$ enables the gate of the Barrel Switch corresponding to a displacement of a number by three positions. This means that, since LOD3 controls the fourth level of the Barrel Switch, the bit at position 20 passes through the second and third levels through the gates corresponding to a zero shift and "comes out" at bit position 23 (Figure 18).

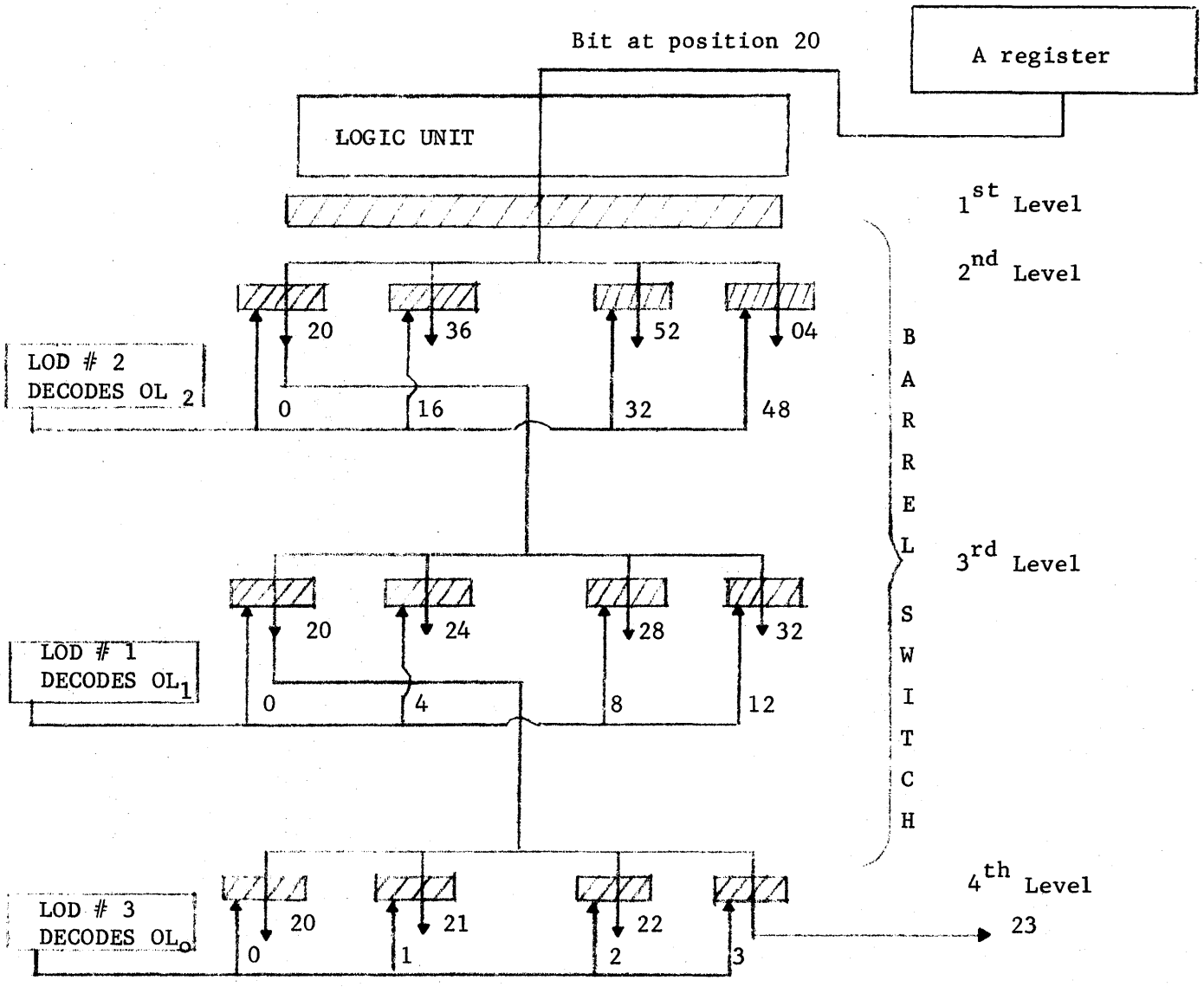


Figure 18. Path of Bit at Position 20 through the Levels of the Barrel Switch (Example).

d) Leading ONES Detector (LOD): This unit is used to control the Barrel Switch when shifting from 0 to 63 left or right, or both, end off or end around. It is also used to detect where the leading ONE in the mantissa of an operand is located and then to generate the proper controls to cause the Barrel Switch to shift left (normalize).

The previous description of the Shift Count Register (LOD4) mentions that its contents (least significant six bits) may come from the CU over the Common Data Bus path or from the CPA. The latter is the source of the contents of LOD4 for mantissa alignment (see Alignment in the subsection dealing with addition).

In order to control the Barrel Switch due to a CU decision (shift amount is received from the CU) or due to a PE decision (in which case an alignment or normalization is performed), the Leading ONES Detector, which is subdivided into six subunits, operates as follows:

(1) LOD1, 2, and 3: These are 16-bit-long units whose logic appears on A09-D, E, and F type cards schematic diagrams; LOD1 receives bits 17 to 32 from A register, LOD2 receives bits 33 to 48 from A register, and LOD3 receives bits 49 to 63 from A register (one bit not used). The logic of LOD1, 2, and 3 can be characterized as priority logic in that whenever the option of normalize is used and a ONE is detected in one of the three LOD's, and within each LOD in one of the four groups which are categorized as shifts by 0, 1, 2, and 3, these particular controls dominate and the Barrel Switch shifts left accordingly. It is evident, therefore, that if a ONE is detected by LOD1 and LOD3, LOD1 will predominate in 64-bit mode or 32-bit mode for the Inner word.

If the shifting operation is due to a CU or a PE decision (alignment), in which case LOD4 holds the shift number (amount), LOD1, 2, and 3 receive this number and direction of shift from LOD4, decode it, and generate the proper controls to cause the Barrel Switch to shift as much as requested. In the case when normalization is to be performed, the output of LOD4 to LOD1, 2, and 3 is zero,

because LOD4 does not participate in this operation at all. The Interface of LOD1, 2, and 3, LOD4, and the Barrel Switch is shown in Figure 17.

(2) LOD5: This is an A10-C type card which contains supplementary logic of LOD1, 2, and 3 and is used to communicate with LOD4 (true and complement form), thus to generate shift left and right controls for LOD1, 2, and 3, and to block the Barrel Switch from shifting any amount if the leading ONE is at bit position 16 when normalizing, and to generate a special signal (Zero Mantissa Level) for LOD6 when normalizing and the mantissa is a zero number. LOD5 also participates, when the option of Rounding is used, to save the most significant shifted off bit (right end off) during the process of aligning the mantissa when addition or subtraction is requested and the exponents of the operands are unequal.

During the process of normalizing an operand or the result of an operation, LOD5 generates proper controls for the exponent correction (see the subsection concerned with Normalization).

(3) LOD6: This unit receives the higher-order bits of the result of exponent subtraction from the CPA (bits 65 through 73) and the Zero Mantissa Level signal from LOD5. This signal, when active, clears the mantissa, sign, and exponent fields of A register when the result is being normalized and the mantissa field contains zeros.

(4) LOD15: This is an A10-B type card and receives the nine most significant bits of the exponent difference and is used to generate the proper controls so that, when the amount of shifting of an operand (mantissa) as a result of the exponent difference is greater than the mantissa field itself, the mantissa of the operand with the smaller exponent is zeroed and the Barrel Switch does not have to participate in this operation.

Because the LOD is a complex unit which participates in the shifting operations due to either CU or PE decisions, it is suggested that the reader familiarize himself with LOD logic so that when the alignment and normalization processes are discussed he will be able to follow the process of a shifting operation as implemented in the machine.

e) Logic Unit (LOG): This is an A03 type card logic element used as a select gate for data whose destination is the MIR portion of the MLU or Barrel Switch. It is also used to perform the logic "AND" and "OR" of two operands. It consists of eight cards, each one handling eight bits.

Because the basic gate of LOG is a NOR gate tied to four other NOR gates, the output of LOG undergoes an inversion before it is brought to the MIR of MLU or to the Barrel Switch.

Each two-input gate receives two lines, with one line representing the data from A, B, S, and C registers or Operand Select Gates (OSG) and the other representing the enable signal. The enable for each register is different from that for every other, but is the same for all the bits of a specific register. This means that if, for example, the output of B register is selected by LOG, the enable signal will be the same for all the gates of LOG that are wired directly to B register, but, because there are eight cards in the LOG, this enable signal feeds eight cards and is common for the eight bits of B register per LOG card. The same scheme applies to the other inputs to LOG (Figure 19). Because all the sources of data to LOG are 64 bits long, there is a bit-by-bit correspondence with respect to LOG with the exception of C register. The bits of C register are numbered from 16 to 79 and, for reasons of implementing eight-bit byte instructions (i.e., A register is less, greater than, or equal to B register, etc.), only C register bits 16, 24, 32, 40, 48, 56, 64, and 72 are sent directly to LOG bits 23, 31, 39, 47, 55, 63, 7, and 15, respectively.

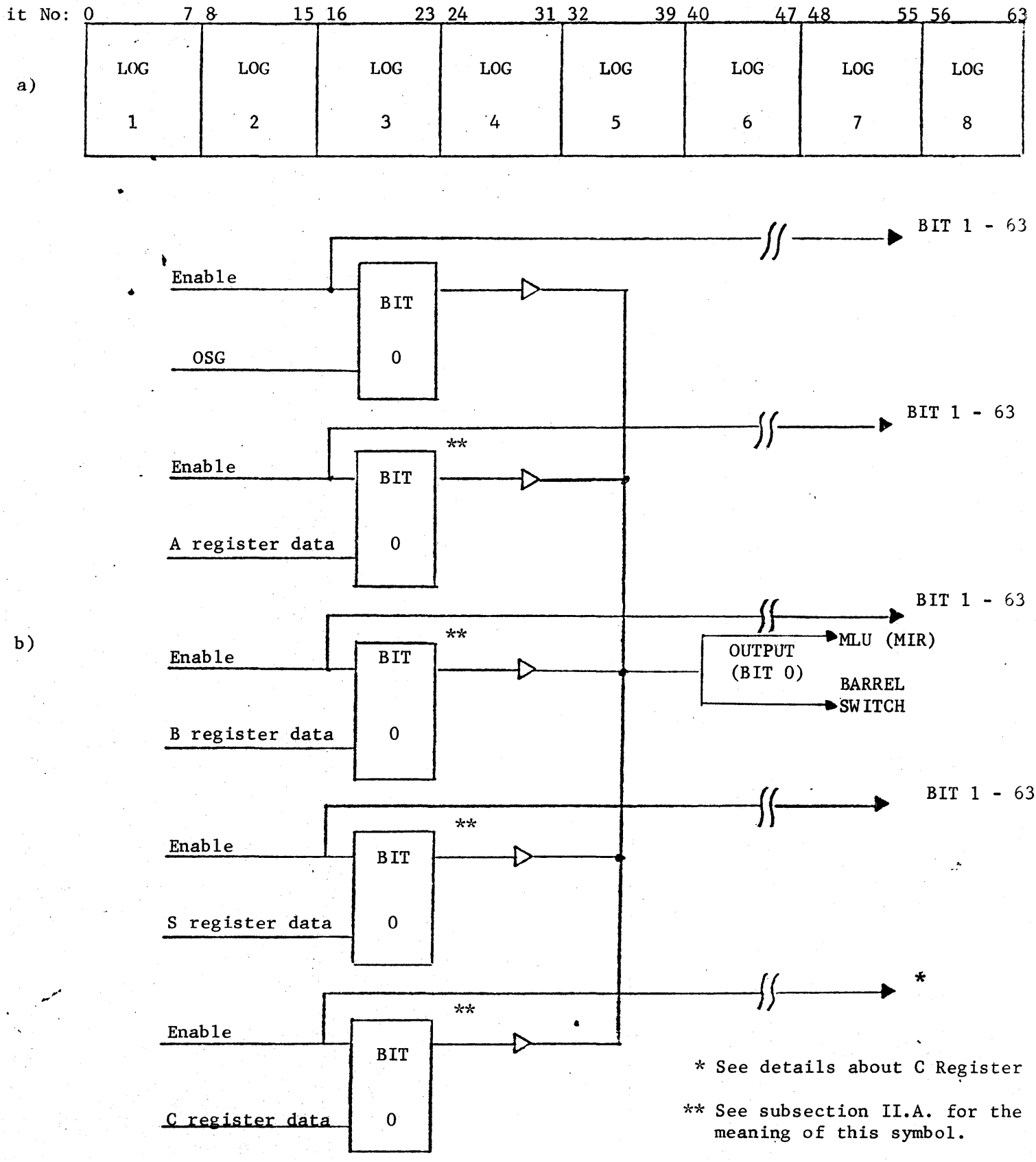


Figure 19. LOG: a) Bit Organization, b) Logic Configuration

f) Operand Select Gates (OSG): This is an A03 type card logic element used to select data transferred to the PE registers from MIR (MLU) or CU data over the Common Data Bus. Data from MLU to PE is known as "PEM read data." Data from the CU may be transferred to PE registers, or after it passes through OSG and LOG it may be sent to the MLU, in which case this data is known as "CU write data to PEM." Since OSG is an A03 type card its operation is similar to that of LOG. It is organized into eight cards, each one taking care of eight bits (Figure 20(a)).

The inputs to OSG are CDB; MIR (MLU); B, R, and mode register; and Address Adder. Data from CDB and MIR and the B and R registers may be 64 bits wide, while data from the mode register are only eight bits wide, and data from the Address Adder are 16 bits wide.

Because the OSG output consists of five two-input NOR gates tied together (Figure 20(b)), data from the mode register is wired directly to OSG #1 and data from the Address Adder is wired to OSG #7 and #8. In this way, a significant amount of logic is saved. The output of OSG is wired to LOG, B and R registers (64 bits), and to the Address Adder (16 least significant bits of OSG). This data undergoes an inversion while passing through OSG and, in order to be consistent with the description in the PEM manual, Table 31 provides signal level information before and after it passes through OSG.

Table 31. OSG Signal Representation

OSG INPUT			OSG OUTPUT			REMARKS
SIGNAL NAME	LEVEL	LOGIC	SIGNAL NAME	LEVEL	LOGIC	
TVW-WXX--0 (CDB)	HIGH	1	PDW-WXX--0	LOW	1	In actuality, the CDB input to OSG is the output of the latch following a differential receiver (Receiver and Register).
TVW-WXX--1 (CDB)	LOW	1	PDW-WXX--1	HIGH	1	
TVW-WXX--0 (CDB)	LOW	0	PDW-WXX--0	HIGH	0	
TVW-WXX--1 (CDB)	HIGH	0	PDW-WXX--1	LOW	0	

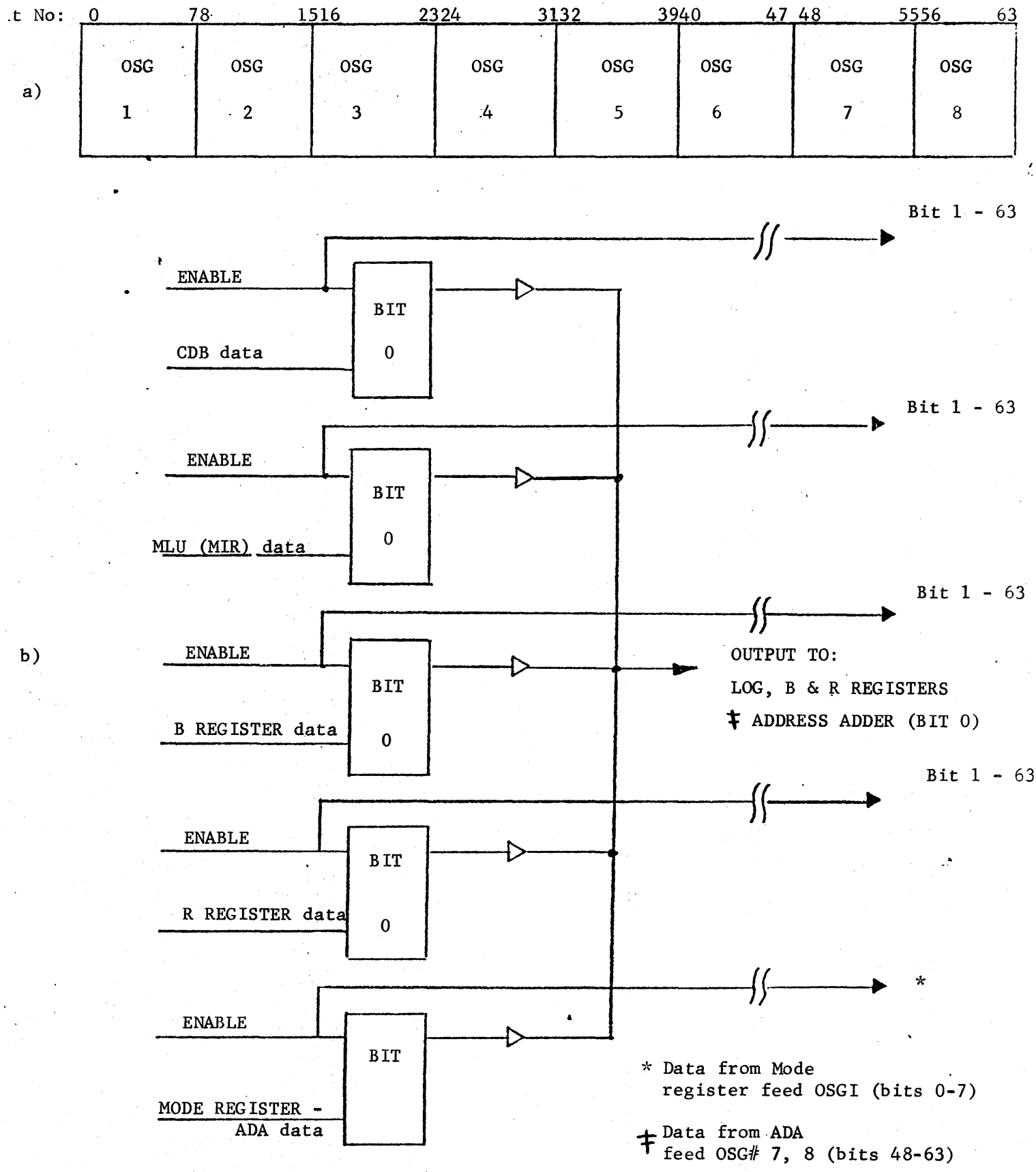


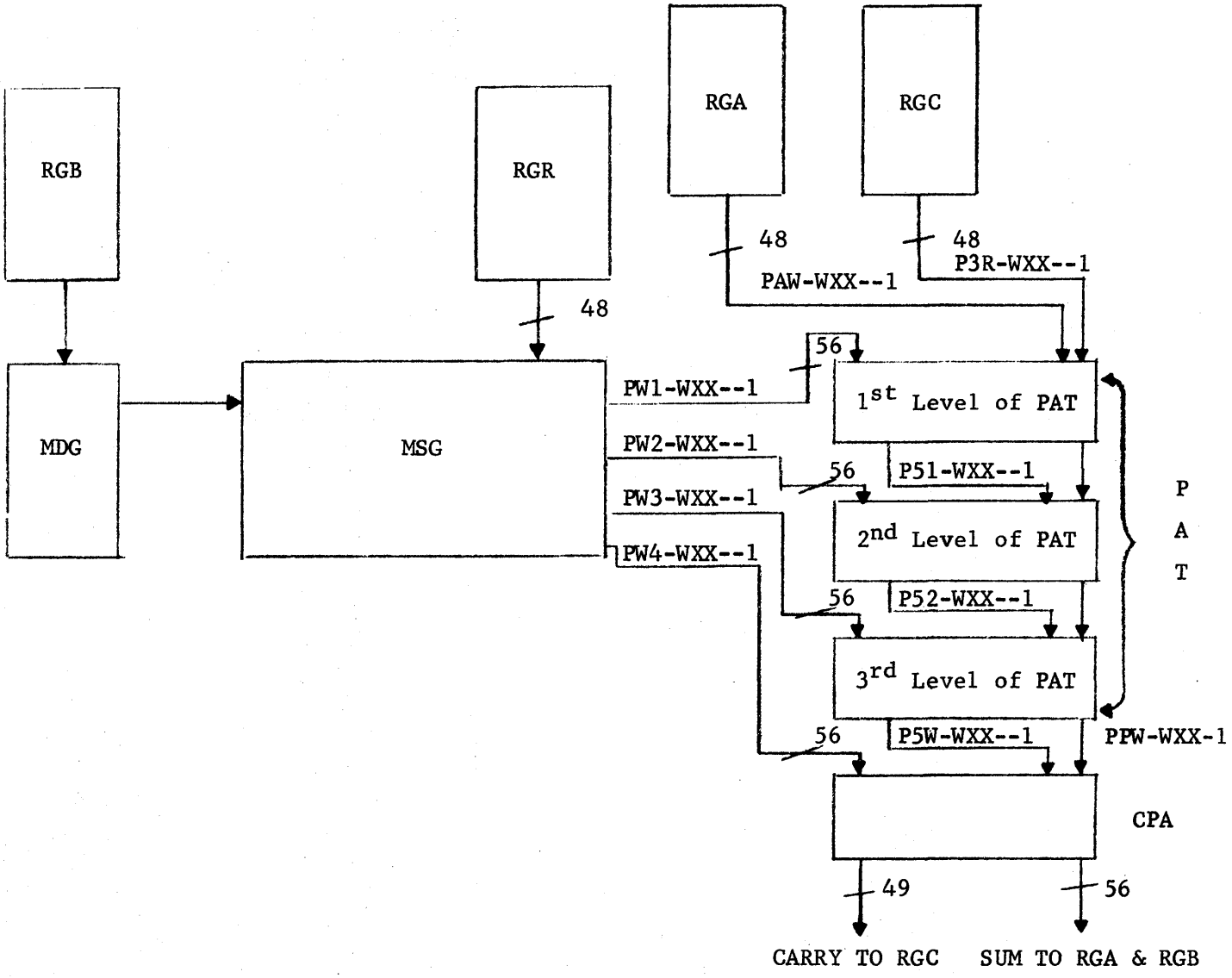
Figure 20. OSG: a) Bit Organization, b) Logic Configuration

g) Pseudo Adder Tree (PAT): This is an A04 type card logic element used primarily in multiplication and division only if the dividend is smaller than the divisor. It consists of three levels which function as a Carry Save Adder: the first level has as inputs the multiplicand or partial sum, the partial carries, and the recoded multiplicand; the second level has as inputs the sum and carry from the first level and the recoded multiplication; and the third level has as inputs the sum and carries from the second level and the recoded multiplicand.

Because the multiplication process is described later in terms to which the reader has not yet been introduced, the levels are not defined here in terms of the bits they can accommodate. Also, there are signals in every level that are used specifically in PAT and which have not been defined previously. For these reasons, only the bit organization and functional Interface of PAT are provided at this point; in a later subsection concerned with multiplication a more complete description of PAT, from the functional point of view, will be given. The PAT is subdivided into 14 parts (cards), with each part taking care of four bits as follows:

PAT #:	PAT 1	PAT 3	PAT 5	PAT 7	PAT 9	PAT 11	PAT 13	PAT 15	PAT 17	PAT 19	PAT 21	PAT 23	PAT 25	PAT 27
BIT #:	16-19	20-23	24-27	28-31	32-35	36-39	40-43	44-47	48-51	52-55	56-59	60-63	64-67	68-71

The output of the third level of PAT is directly wired to the CPA and, because PAT takes care of the mantissa part of an operand, there is a bit-by-bit correspondence between PAT 1 through PAT 23 with CPA 1 through CPA 12, respectively. The purpose of PAT 25 and 27 and the exact bit configuration of each level of PAT are shown during the description of the multiplication process. Figure 21 shows the functional block diagram of PAT and its interface with the logic elements participating in the multiplication process.



NOTE: Since only 56 bit locations are accommodated by the PAT, the WXX notation refers to bits 16 - 71.

Figure 21. Registers Directly Associated with PAT

h) Multiplicand Select Gates (MSG): This is an A02 type card logic element, which is used primarily for the implementation of the instructions regarding multiplication of two operands. It consists of six cards, each one taking care of eight bits (Figure 22).

The MSG is interfaced with:

(1) the R register from which it receives the mantissa part of the word representing the multiplicand, which has been stored in R register prior to starting the multiplication process (48 bits long).
(2) the Multiplier Decoder Gates (MDG) from which the MSG receives controls forcing its outputs to represent the multiplicand times ONE, times TWO, times minus ONE, or times ZERO.
(3) the three levels of PAT, with the first level receiving the output of MSG corresponding to Word #1, the second level receiving the output of MSG corresponding to Word #2, and the third level receiving the output of MSG which corresponds to Word #3.
(4) the CPA which receives the output of MSG which corresponds to Word #4.

The Interface of MSG with the PAT and CPA is made through 49 lines instead of 48 (number of bits of mantissa part) for reasons explained in the subsection concerned with multiplication.

In division, however, where R register holds the divisor, the MSG is used as a receiver of the mantissa and, through the path for Word #4, sends the divisor to the CPA to be subtracted from the dividend (see the subsection concerned with division). Because, as mentioned in the subsection dealing with multiplication, there are functional block diagrams and bit organization figures tying all the associated logic elements together for a better understanding of the multiplication process, the reader is urged to look ahead into that subsection for any additional information about MSG.

MSG #	RGR INPUT BIT #	WORD #1 BIT #	WORD #2 BIT #	WORD #3 BIT #	WORD #4 BIT #
1	16-23	22-29	20-27	18-25	16-23
3	24-31	30-37	28-35	26-33	24-31
5	32-39	38-45	36-43	34-41	32-39
7	40-47	46-54	44-52	42-50	40-48
9	48-55	55-62	53-60	51-58	49-56
11	56-63	63-70	61-68	59-66	57-64

Figure 22. MSG Input/Output Bit Organization

i) Multiplier Decoder Gates (MDG): This is an A12 card type logic element used exclusively for the implementation of the multiply instructions. It decodes the multiplier taken eight bits at a time (each clock time) into four words, each of which may have one of three values: X1, X2, or X-1. For each word, these controls (X1, X2, X-1) are fed into the MSG which gates out the multiplicand received from the R register directly, shifted left by one or in one's complement form, respectively.

In the subsection dealing with the multiplication of two operands it is thoroughly explained how and why the multiplier is decoded by pairs (eight bits per clock time period).

The MDG consists of two cards (MDG#1, MDG#2), each of which receives four bits (two bit-pairs) and two inputs for the CARRY and CARRY NOT and generates the controls for MSG corresponding to X1, X2, and X-1 (Figure 23).

Because MDG#1 decodes the four least significant bits of the multiplier, there is a Carry if the first pair is 11 and the second pair is 10 or if the second pair is 11. This Carry is sent to MDG#2 which combines it with the third and fourth pairs of bits from the B register and generates the controls for Words #3 and #4. If, however, the third and fourth pairs of bits are such that a Carry is formed, this Carry is then brought to a gate whose true and complement outputs are brought to MDG#1 to be used in the next iterative cycle. It is evident that in the first iterative cycle MDG#1 does not receive any Carry from MDG#2.

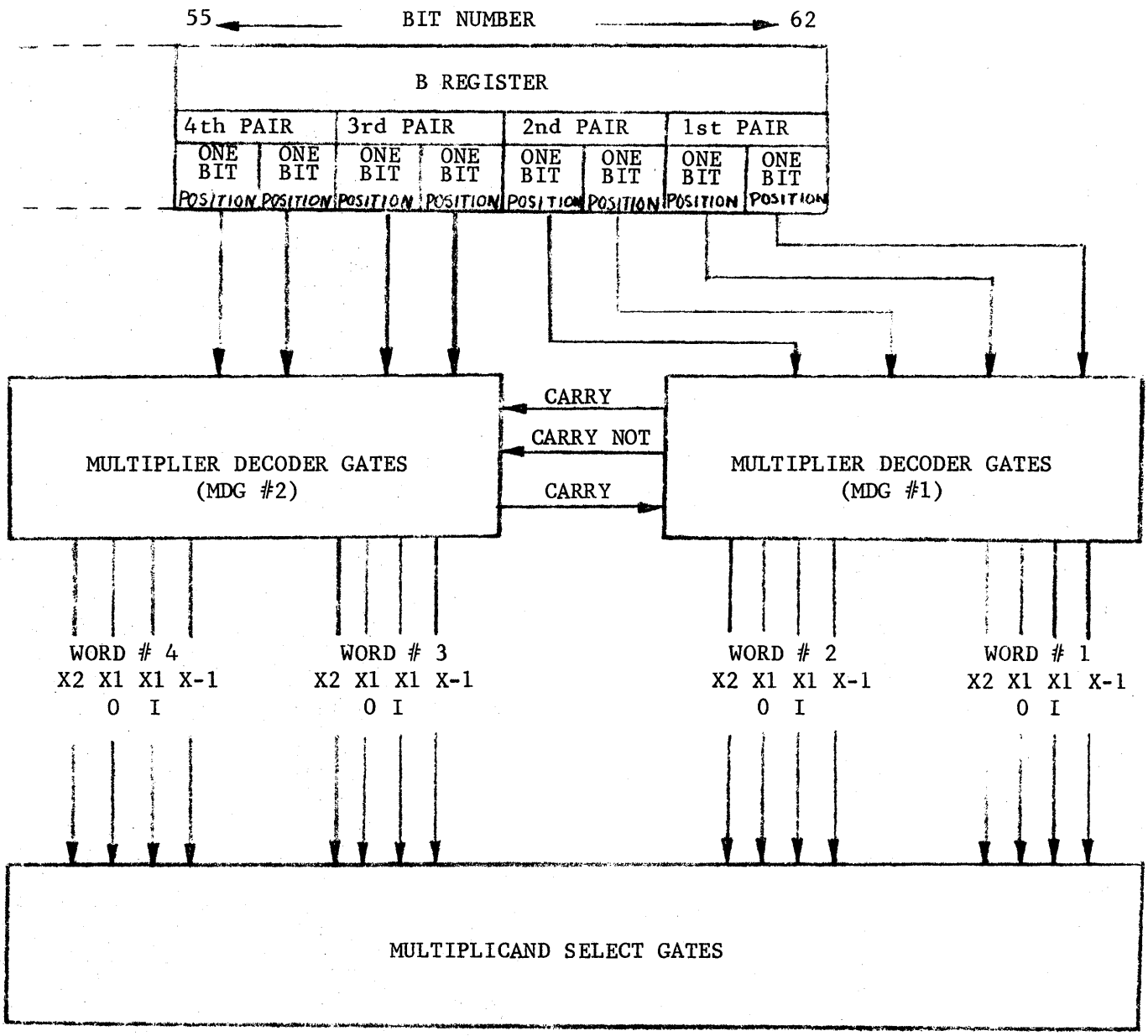


Figure 23. MDG Functional Block Interface Diagram

3. Interface Units: These are logic elements which are used to enable the Processing Element to communicate with the outside world and particularly with the neighboring PE's and CU. These logic elements are the Driver, Receiver Selection Gating, Receiver Register, and Driver and Receiver.

a) Driver (DRV): This is an A15 type card logic element, used as an interface between the R register of PE_i and PE_{i+1}, PE_{i+8}, PE_{i-1}, and PE_{i-8}. It consists of four cards (Figure 24) and can take care of 64 bits of data. This is strictly a dual line driver (true and complement outputs are used) to drive the data to the routing logic of the corresponding Processing Units. The use of the dual line driver helps to eliminate common-mode noise created on the relatively long signal lines between PE_i and PE_{i+1}, PE_{i+8}, PE_{i-1}, and PE_{i-8}. Each card consists mainly of 18 line drivers for a total of 72. Sixty-four are used for routing, one for the Mode Bit to the CU, and another for Memory Fault to the CU. Six are unused.

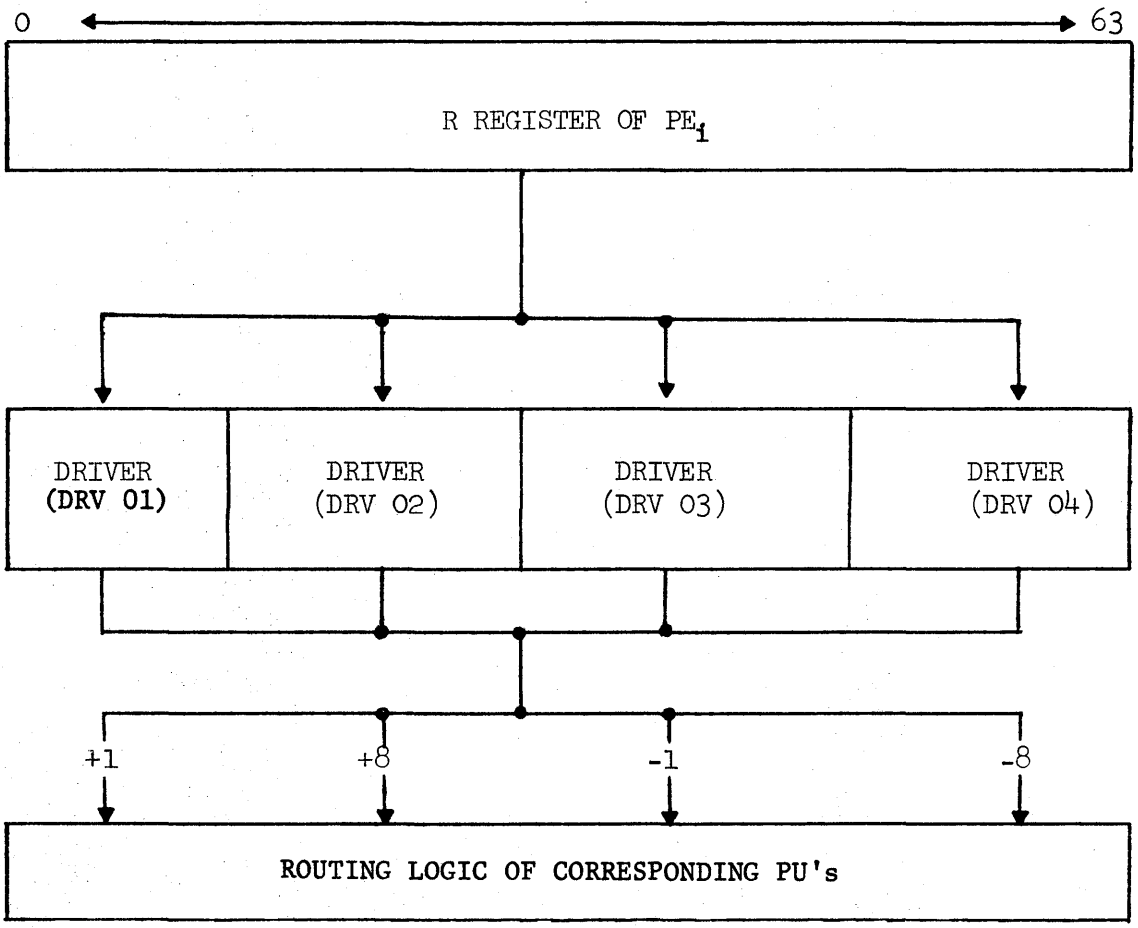


Figure 24. Driver Functional Block Diagram

b) Receiver Selection Gating (RSG): This is an A16 card type logic element. It has differential receivers and select gates. It is used to select data from PE_{i+1}, PE_{i+8}, PE_{i-1}, or PE_{i-8} and bring it into the R register. The RSG select logic consists of NOR gates tied together and controlled by enables corresponding to PE_{i+1}, PE_{i+8}, PE_{i-1}, and PE_{i-8}. The use of differential receivers provides the advantage of eliminating common-mode noise created on the long signal lines between PE_i and PE_{i+1}, PE_{i+8}, PE_{i-1}, and PE_{i-8}, which are exposed to high noise transients and significantly different temperatures. The RSG consists of 13 cards, each one able to accommodate five bits (one circuit is unused).

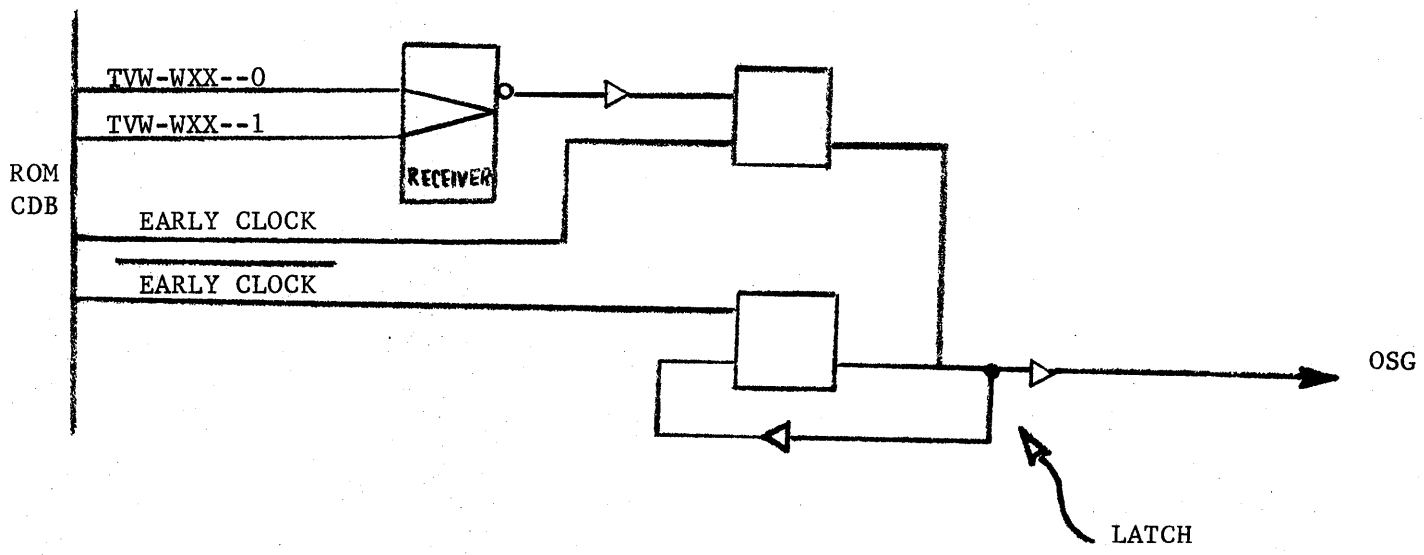
RSG 1	RSG 2	RSG 3	RSG 4	RSG 5	RSG 6	RSG 7	RSG 8	RSG 9	RSG 10	RSG 11	RSG 12	RSG 13

c) Receiver Register (RC5): This is an A14 type logic element, which consists of differential receivers and latches, and is used to store data from the CU received via the Common Data Bus path which can then be gated into the Operand Select Gates (OSG). There are four cards (Figure 25(a)) in the Receiver Register; each one consists of 16 differential receivers and 16 latches which are clocked with early clocks.

Because the true output of each latch is gated into the OSG, it is evident that the input to OSG will always have the same logic level as the output of the differential receiver (Figure 25(b)). The input signals to RC501 — RC504 are designated TVW-W(00-63)--0 and TVW-W(0063)--1.

RC5 #	RC501	RC502	RC503	RC504
BIT #	0——15	16——31	32——47	48——63

(a)



(b)

Figure 25. Receiver Register: a) Bit Organization, b) Logic Configuration

d) Driver and Receiver: This logic element consists of two parts, namely, the Driver which is a part of the DRV04 used to drive only one line back to the CU, and the Receiver part which is an A14 card type logic element used to receive the enables of the mode register (see the mode register description). The Driver and Receiver, therefore, is not a separate unit, but is mentioned separately because its function concerns an extremely different register (mode) which is constantly monitored by the CU through this portion of A14 and A15 card type logic.

4. Other Logic Elements: All of the logic elements shown in the Processing Element Block Diagram (Figure 12) have been described, but Table 32, which shows the physical location of the logic elements of the PE by type of card, lists a few logic elements in the PE which neither appear in Figure 12 nor have been mentioned elsewhere. These logic elements are:

- TUB01 — TUB17: They are used to buffer and retime the different enable signals for the operation of the PE. They are A14 card type logic elements. TUB01, 02, 05, 06, and 09 require early clocks, while the remaining TUB's require late clocks.
- CTL01 — CTL12: These 12 Control (CTL) cards are used to provide all the control signals to the PE logic, without the presence of which no operation could be implemented in the PE. Because the CTL's consist of different card types, a summary of CTL versus card type used is shown in tabular form below.

CTL NUMBER	CARD TYPE	REMARKS
CTL01	B01	
CTL02	B02	
CTL03	B03	
CTL04	B04	
CTL05	B05-A	
CTL06	B6	Clocked
CTL07	B7	Clocked
CTL08	B8	
CTL09	B9-A	
CTL10	B10	Clocked
CTL11	B11	Clocked
CTL12	B12	

- Group Look Ahead (GLA01 — GLA04): This logic element was mentioned in its use as part of the Adder, but after the logic for the generation of section transmits and section generates and group carries are distributed to the CPA and CLA, these cards are used to buffer different enable signals because of heavy loading.

- Clock Generation (CLK01): This is a C01 card type logic element which generates, after receiving the main clock from the CU, three other clocks necessary to the internal operation of the PE. These clocks are the Early Clock, the Late Clock, and the Register Clock.

- Clock Buffer (CLK E, L, R): This is a C02 card type logic element used to buffer the Early, Late, and Register Clocks from the C01 card to the PE circuitry. Since these clocks are very important, it has been proven through experience that the Early Clock measured at TUB02 (D2-13) should be about 7.0 ns wide and its leading edge should occur about 25 ns prior to the leading edge of the reference signal measured at TUB13 (D2-14). The Late Clock should have the same width as the Early Clock, but its leading edge should occur about 6 ns prior to the leading edge of the reference signal, while the Register signal should have the same measurements as the Late Clock.

- BIAS: This is a D01 card type circuit used to provide proper bias for unused gate inputs.

Table 32. PE Card Physical Configuration (Card Side)

Column Number	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Row Number
Name of Logic Element	RSG 01	RSG 02	RSG 03	RSG 04	RSG 05	RSG 06	RSG 07	RSG 08	RSG 09				CLKE 3	CLKL 2	CLKO 1	CLKR 5	CLKR 4	LOD 04	LOD 05	LOD 06								BIAS								A
Type of Card	A16	A16	A16	A16	A16	A16	A16	A16	A16				C2	C2	C1	C2	C2	A1A	A10C	A10B								D1								
Remarks													EAR	LAT	GEN	REG	REG	C		C																
Name of Logic Element	RSG 10	RSG 11	RSG 12	RSG 13	DRV 01		DRV 02	RGR 03	MSG 01	PAT 01	PAT 03	PAT 05	PAT 07	RGB 03	RGA 03	CPA 01	CPA 02	LOD 01	LOD 15	GLA 02	CPA 04	CPA 03	RGA 04	RGB 04	CTL 05	BS 42	BS 32	BS 22	BS 12	LOG 03	LOG 04	OSG 03	OSG 04	RGC 02	RGS 02	B
Type of Card	A16	A16	A16	A16	A15		A15	A1	A2	A4	A4	A4	A4	A1	A1	A5	A5	A9D	A10B	A5B	A5	A5	A1	A1	B5A	A7B	A7B	A7A	A6	A3	A3	A3	A3	A8	A8	
Remarks								C						C	C				C				C	C								C	C	
Name of Logic Element	RCC 5 01	RCC 5 02	RCC 5 03	RCC 5 04		RGR 04	MSG 03	CTL 09	MSG 05	PAT 09	PAT 11	PAT 13	PAT 15	RGB 05	RGA 05	CPA 05	CPA 06	LOD 02	CLA 01	GLA 03	CPA 08	CPA 07	RGA 06	RGB 06	CTL 08	BS 43	BS 33	BS 23	BS 13	LOG 05	LOG 06	OSG 05	OSG 06	RGC 03	RGS 03	C
Type of Card	A14	A14	A14	A14		A1	A2	B9A	A2	A4	A4	A4	A4	A1	A1	A5	A5	A9E	A11	A5B	A5	A5	A1	A1	B8	A7B	A7B	A7A	A6	A3	A3	A3	A3	A8	A8	
Remarks	C	C	C	C		C								C	C								C	C									C	C	
Name of Logic Element	TUB 01	TUB 02	TUB 03	TUB 04	TUB 15	RGR 06	RGR 07	MSG 07	MSG 09	PAT 17	PAT 19	PAT 21	PAT 23	RGB 07	RGA 07	CPA 09	CPA 10	LOD 03	CLA 03	GLA 04	CPA 12	CPA 11	RGA 08	RGB 08	CTL 07	BS 44	BS 34	BS 24	BS 14	LOG 07	LOG 08	OSG 07	OSG 08	RGC 04	RGS 04	D
Type of Card	A14	A14	A14	B14	A14	A1	A1	A2	A2	A4	A4	A4	A4	A1	A1	A5	A5	A9F	A11	A5B	A5	A5	A1	A1	B7	A7C	A7B	A7A	A6	A3	A3	A3	A3	A8	A8	
Remarks	C	C	C	C	C	C	C							C	C								C	C	C								C	C	
Name of Logic Element	TUB 05	TUB 06	TUB 07	TUB 08	TUB 16	RGR 05	MDG 02	CTL 02	CTL 01	MDG 01	RGR 08	MSG 11	PAT 25	PAT 27	RGB 01	RGA 01	CPA 13	CPA 14	GLA 01	CPA 16	CPA 15	RGA 02	RGB 02	CTL 04	CTL 06	BS 41	BS 31	BS 21	BS 11	LOG 01	LOG 02	OSG 01	OSG 02	RGC 01	RGS 01	E
Type of Card	A14	A14	A14	A14	A14	A1	A12	B2	B1	A12	A1	A2	A4	A4	A1A	A1A	A5	A5	A5B	A5	A5	A1A	A1A	B4	B6	A7C	A7B	A7A	A6	A3	A3	A3	A3	A8	A8	
Remarks	C	C	C	C	C	C	C			C	C				C	C							C	C	C								C	C	
Name of Logic Element	TUB 09	TUB 10	TUB 11	TUB 12	TUB 13	TUB 14	TUB 17	DRV 03	DRV 04						RGR 01	RGR 02	RGD 01	RGD 11	CTL 10	CTL 03	CTL 12	CTL 11	MAR 01	RGX 01	ADA 01	ADA 02	CLA 05	GLA 05	ADA 03	ADA 04			MLU 01	MLU 02	MLU 03	F
Type of Card	A14	A14	A14	A14	A14	A14	A14	A15	A15						A1	A1	A13A	A13B	B10	B3	B12	B11	A8	A8	A5	A5	A11	A5B	A5	A5						
Remarks	C	C	C	C	C	C	C								C	C		C	C			C	C	C								X	X	X	

NOTES:
1. EAR denotes early clock
2. LAT denotes late clock
3. REG denotes register clock
4. C denotes card is being clocked
5. X denotes interface connector

B. DC POWER DISTRIBUTION

a) Description

In each Processing Unit Cabinet (Figure 26) there are two power supplies that provide +4.8 V and ground, two power supplies that provide - 2.0 V and ground, one power supply that provides + 1.32 V and - 3.20 V for the routing logic and eight power supplies (pre-regulators) that provide 4.52 V (+ 1.32 V and - 3.20 V). These power supplies provide power for the eight Processing Units (PUs) that are contained in each of the eight PU Cabinets of the Quadrant. Figure 26 also depicts the physical location of each one of the twelve power supplies and location of the PUs which receive power from these power supplies.

From the PU Cabinet, the above voltages (+ 4.8 V, - 2.0 V, ground and 4.52 V) are brought into the Processing Unit in two groups. The first group brings + 4.8 V, - 2.0 V and ground; it is used exclusively for the PEM and the Up and Down converters of MLU. The second group, which brings 4.52 V to the PU, is used for the PE and MLU circuits; the 4.52 V corresponds to + 1.32 V and - 3.20 V necessary for the ECL circuits of the PE and MLU.

On the top of the PE there is a section called Dual Power Supply Shunt Regulator (Figure 27). This regulator contains two main busses used to transfer the grouped voltages into the individual subunits of the Processing Unit (PE, MLU and PEM). Both busses consist primarily of large laminated planes that are properly isolated from one another. One bus is used for + 4.8 V, - 2.0 V and ground; the other bus is used for + 1.32 V, - 3.20 V and ground.

The + 4.8 V power is tapped from the bus plane for use by the MLU level conversion circuits. A two-plane strip routes the + 4.8 V power to the MLU and provides a path from the MLU to the ground plane of the large bus. This ground path is used to shield the Cabinet Clear signal and as the ground level for the I/O circuits in the MLU.

a)

+4.8V 430AMP A2 TO B2,C2,D2,E2	+1.32, -3.2V 50AMP A3 TO ROUTE LOGIC	-2.0V 220AMP A4 TO B2,C2,D2,E2,F2,G2,H2, and J2	+4.8V 430AMP A5 TO F2,G2,H2,J2
PRE/REG 4.52V A6 TO C2	PRE/REG 4.52V A7 TO E2	PRE/REG 4.52V A8 TO G2	PRE/REG 4.52V A9 TO J2
PRE/REG 4.52V A10 TO B2	PRE/REG 4.52V A11 TO D2	PRE/REG 4.52V A12 TO F2	PRE/REG 4.52V A13 TO H2

b)

PU B2	PU C2	PU D2	PU E2	PU F2	PU G2	PU H2	PU J2

Figure 26. PU Cabinet: a) Power Supply Location, b) PU Power Distribution

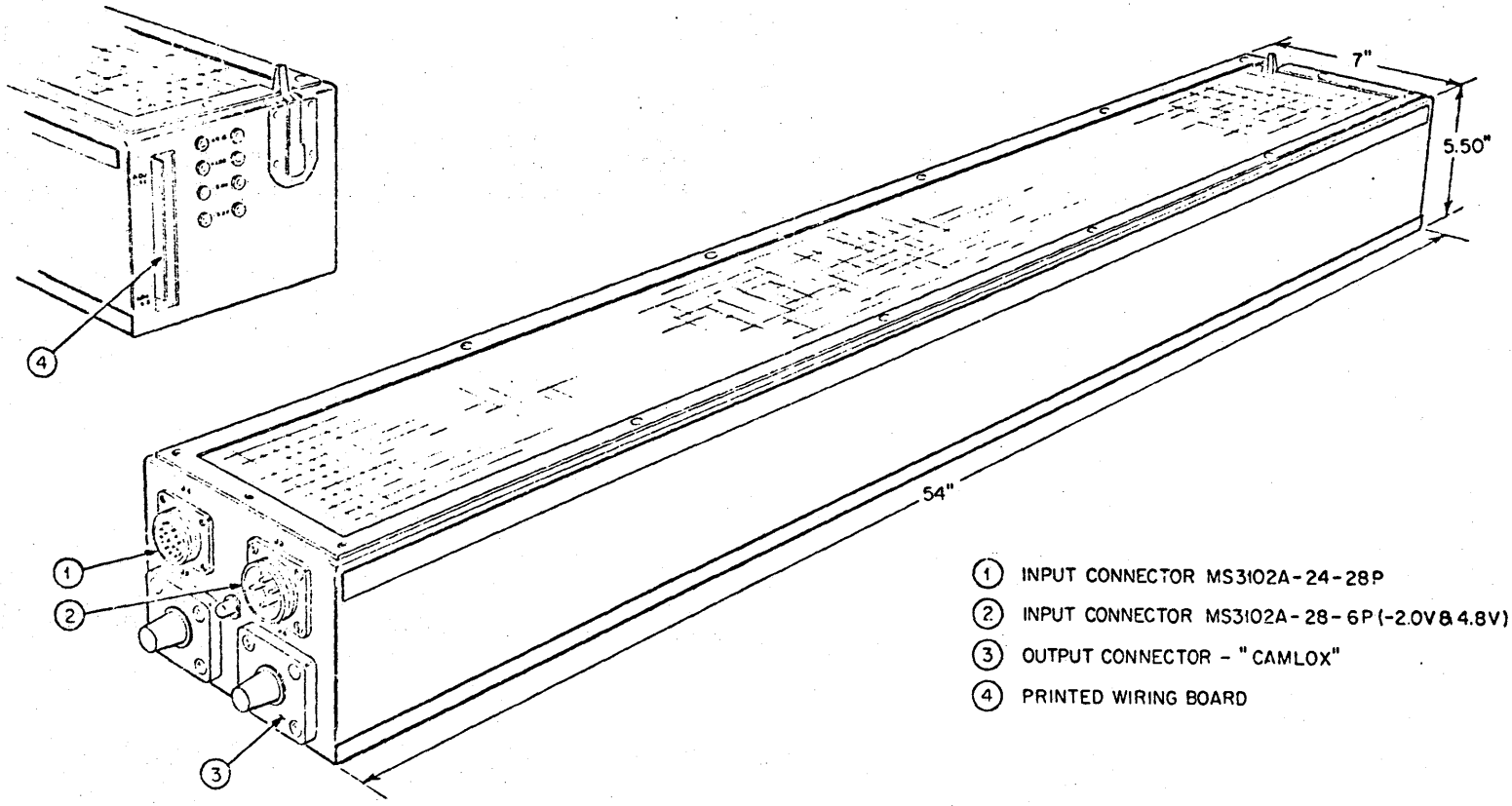


Figure 27. +1.32-Volt, -3.2-Volt, 240-Ampere Shunt Regulator, Outline Drawing

A wire between the control card in the Dual Power Supply Shunt Regulator and the MLU provides the path for the Cabinet Clear signal (MCABCLR--0); this is the signal that resets the flip-flops in the MLU when power is first applied to the PU. This is necessary because the MLU acts as a large electronic switch and it must be cleared before any action starts. In this way it is assured that when the PE or PEM accesses the MLU, the latter should contain no prior information in its memory devices (flip-flops) and it therefore must be ready to transmit the new information that the particular instruction dictates.

The ground plane of the bus for 1.32 V and - 3.20 V is connected at one end with the chassis (PU) and with the ground plane of the second bus at the other end. This assures a common ground for all the subunits of the PU.

The + 1.32 V, - 3.20 V and ground levels are provided to the MLU and PE circuits via the large, three-plane bus shown in Figure 28. These laminated planes are, of course, fully isolated from one another.

Figure 29 depicts the basic current paths involved in the distribution of the + 1.32 V and - 3.20 V. This is consistent with the basic structure of the ECL circuits of the PE and MLU.

Each power supply shunt regulator includes an overcurrent detector. This detector compares the current through a 50 amp, 50 mV shunt resistor, with a fixed reference current. If the current through the shunt resistor exceeds the reference current significantly, the detector opens the circuit breaker in the preregulator.

Similarly, the over voltage/under voltage detector in the power supply shunt regular senses an excessive or insufficient voltage level being applied to a load. If the detector determines that a voltage is outside some specified limit, it opens the appropriate circuit breaker.

Test points for the various MLU voltages are available on the MLU backplane.

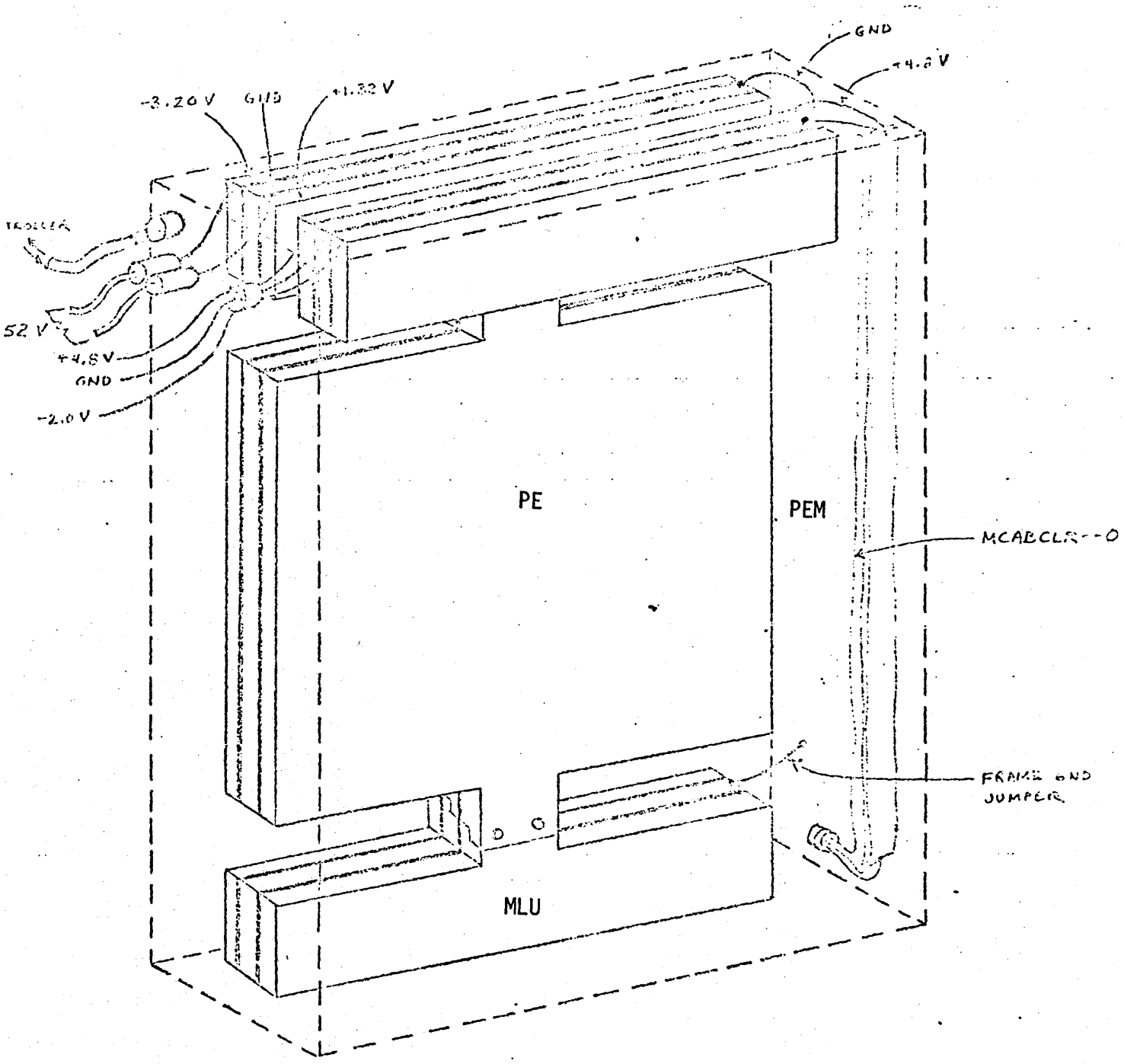


Figure 28. Power Distribution in the PU

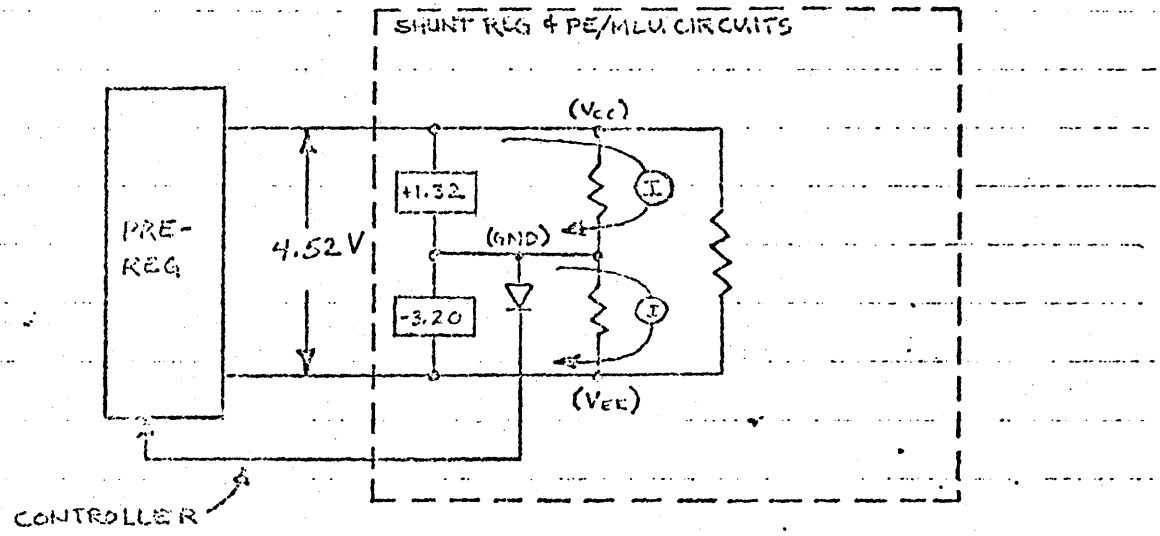


Figure 29. Current Paths for + 1.32 V and - 3.20 V Supplies

The power for the PEM is distributed from the Power Bus through the three designated leads into the power distribution board, which contains large ground and voltage planes of 2 ounces of copper. In order to reduce the voltage drop, the power base picks up the power from the power distribution board through approximately 110 contact pins and distributes the power to the control and memory boards (Figure 21 and 22) through its cam-operated connectors.

Each memory board and the control board as well use three intermeshed power grids for V_{cc} = + 4.8 V, V_{ee} = - 2.0 V, and ground for power distribution to the CTμL, TTμL, and MμL 4100 devices. In order to achieve low DC characteristic impedance, these power lines (V_{cc}, V_{ee} and ground) run parallel to one another. Figures 30 and 31 show only the main power busses, but the reader should realize that there are power distribution line picking up the power from their corresponding power busses on each board. Any sudden variation in voltage due to spikes of high frequency noise is greatly minimized by the use of high-frequency filter (bypass) capacitors, connected between V_{cc}/V_{ee} and ground and also between V_{cc} and V_{ee}.

The power grid network as it has been implemented on the control and memory boards provides an effective ground shield, which helps to obtain, in the case of relatively long transmission signal lines, a characteristic impedance of about 100Ω. It has been calculated that the total power dissipation is distributed as follows:

1) One Memory Board:

   a. CTμL, TTμL, and WRITE transistors require 3.66 amperes at + 4.8 V and 2.98 amperes at - 2.0 V.
   b. MμL 4100's require 16.00 amperes at + 4.8 V.

2) Control Board:

   a. Because the Control Board uses only CTμL devices, it requires 4.53 amperes at + 4.8 V and 3.56 amperes at - 2.0 V.

Since the above numbers represent worst-case PEM power requirements, it can be concluded that a power dissipation of about 400 watts per PEM is equivalent to 3 milliwatts per memory bit (400 watts + 131072 bits).

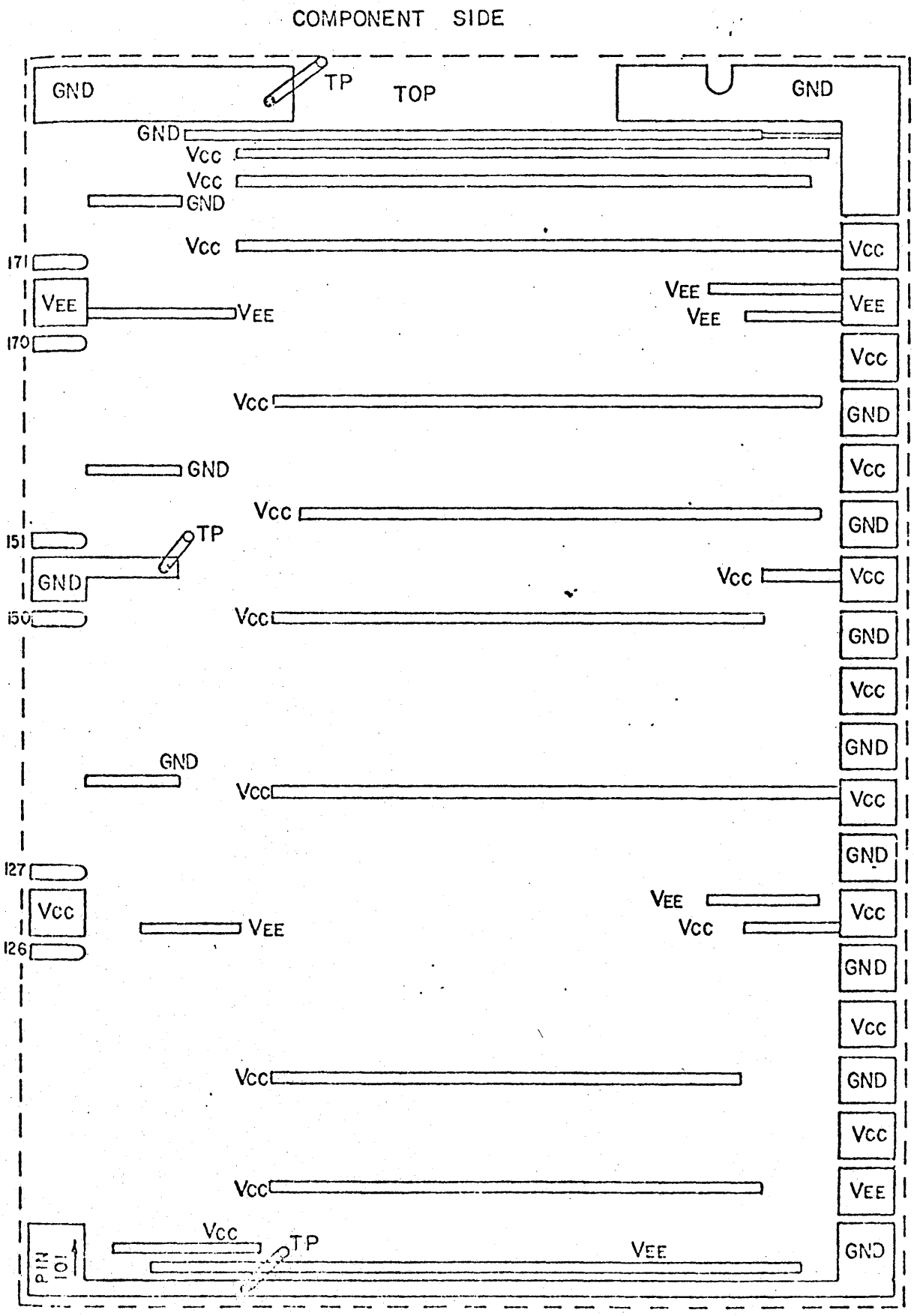


Figure 30. Memory Board Power Bus Configuration

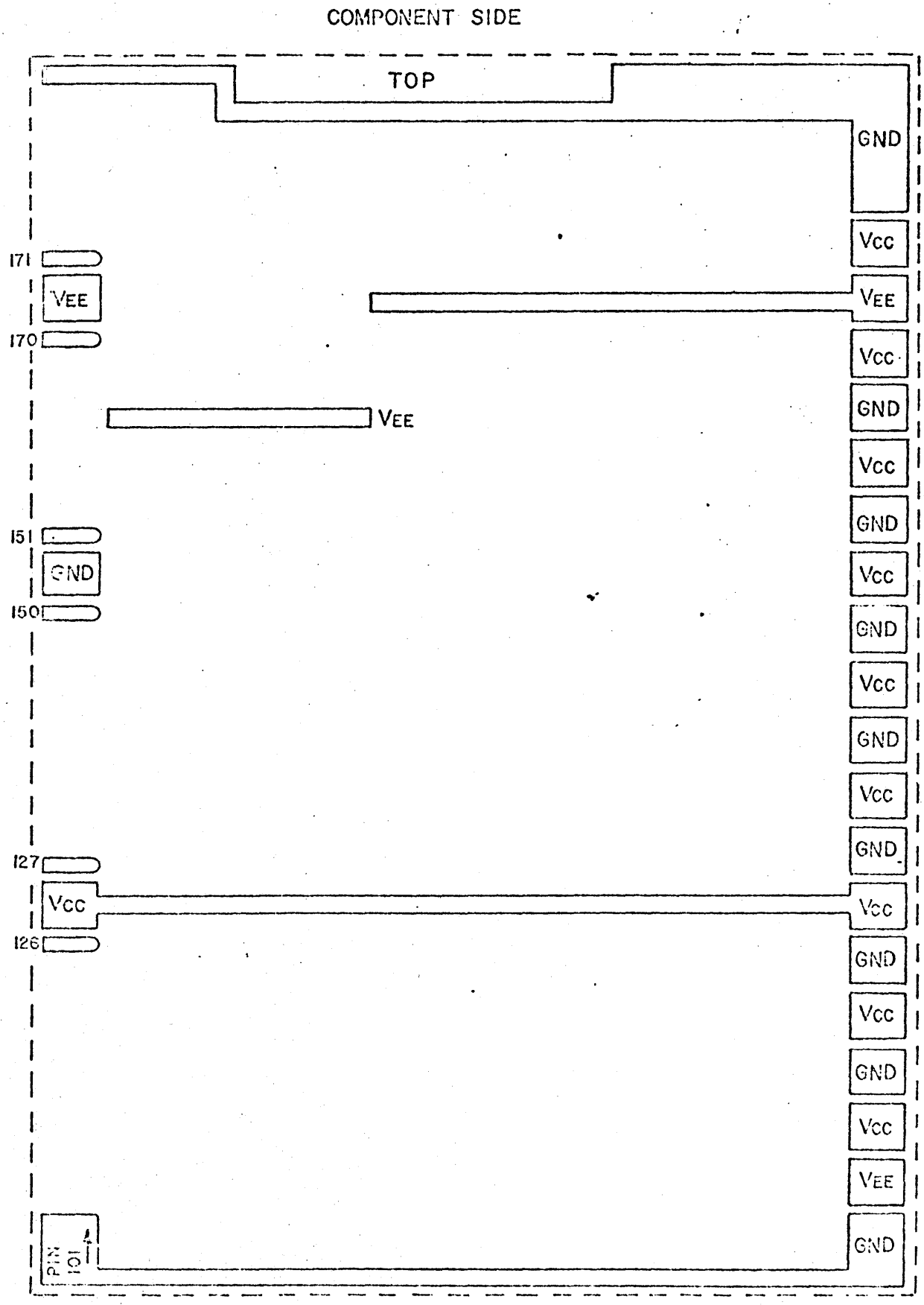


Figure 31. Control Board Power Bus Configuration

b) Grounding

Grounding is an issue of great importance because even though it seems at first a simple and "clean" area to work with, indeed it gets as complicated as the system itself. It is an area that requires particular attention, because bad grounding may result in problems that are not always easy to detect and therefore the reliability of the system is reduced significantly.

As it was explained in [9], the AC power is provided to the ILLIAC IV Computer via the Substation of the NASA/Ames Research Center facility on a Δ to Y formation. The Y constitutes the secondary of the main transformer in the facility and allows the three-phases and the neutral to be brought to the second transformer used to provide power to the computing facility.

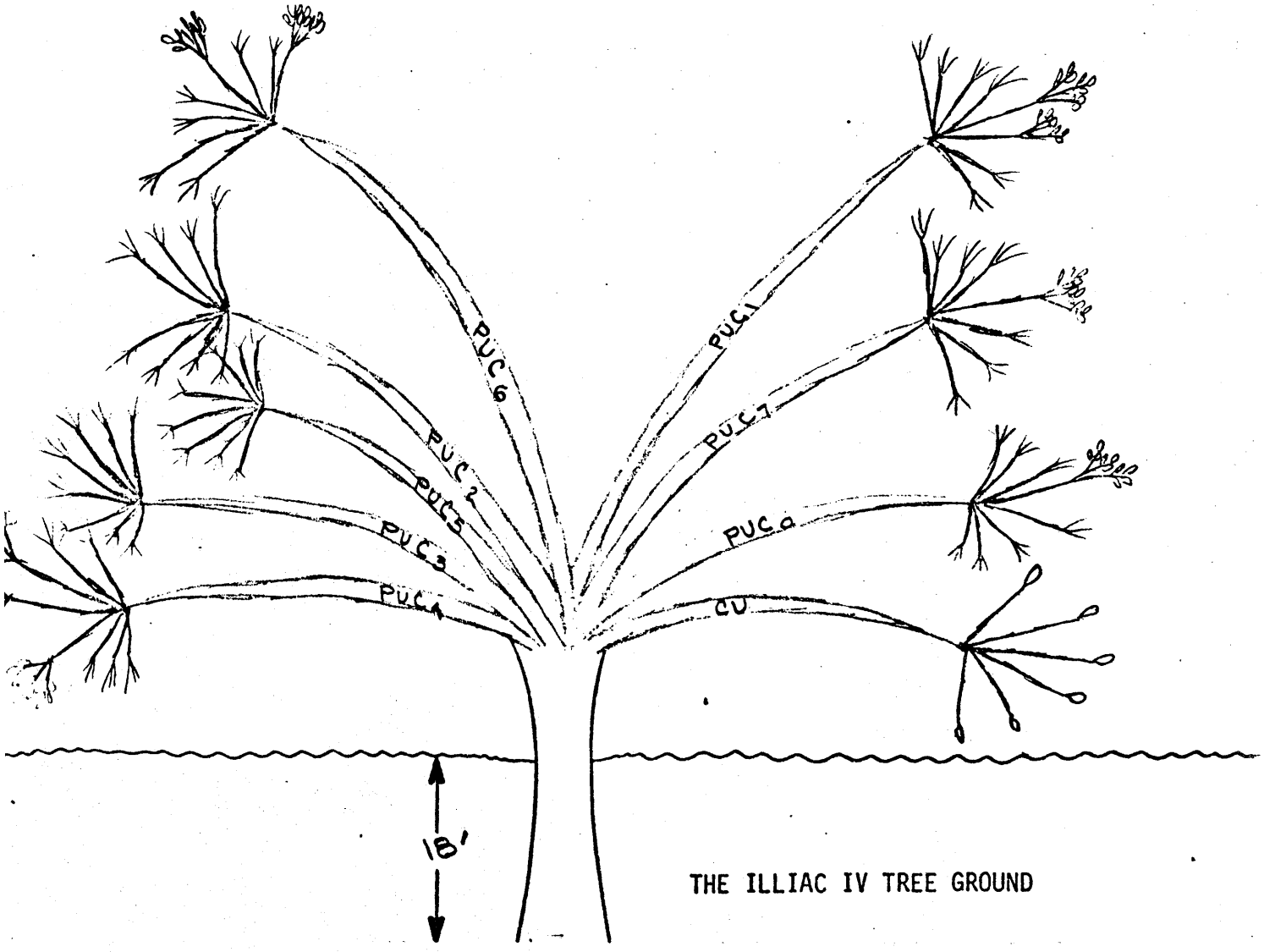
This transformer is of the Y to Y form and provides three-phases and a neutral which is grounded (earth) on the wall power panel. Normally the neutral is at a different potential than the earth ground because the loads on each phase are not ideally equal and according to Kirchoff's current law the current in the common return wire (neutral) which is equal to the sum of the three phase currents is not zero because these three currents are not equal. If the loads were equal in which case we would have a balanced system the neutral and earth ground would be the same. For safety purposes; this is a requirement of the Electrical Code, the neutral at the wall power panel is connected to earth ground. The earth ground is a point that we will talk about later on. The reason for connecting the neutral to ground is to force the circuit breaker to close if accidentally one of the phases touches the neutral. Had not the neutral been connected to ground and by accident one of the phases was touching it, a man not suspicious of the danger that the neutral carries high-voltage, could be killed instantly upon touching it because he would establish a closed circuit to earth ground via his body. This neutral along with the three phases is brought into each PUC and from there to the primaries of each transformer of each power supply (voltage regulators). The secondary winding of each transformer in

the voltage regulators provides the DC voltages that we have already talked about. These voltages are with respect to chassis ground; the ground wire form each power supply is connected to the chassis of the power supply and also to the chassis of the PU. Every PU and its associated power supply are connected via the chassis ground wire to the main frame of the PUC. At this point it must be mentioned that when the three phases and the neutral are applied to each PUC, an extra cable called Ground is brought along with it. It originates in the power wall panel and is tied to the main frame of the PUC. The other end of this cable in the wall power panel is connected to a plate made of copper which in turn connects via a long copper rod to the earth. The same configuration exists for all the PUCs and the CU. In this way the DC power ground, the chassis ground and the PUC mainframe ground are all at the same level with respect to earth ground; measurements have shown a potential of about 35 MV with respect to earth. This ground configuration serves two purposes. First, it provides safety to the personnel, since an accident could occur if the V_{cc} was shorted to the frame. A man could receive a good shot upon touching the frame if the latter was not grounded. Second, all the voltages in each PUC. are considered to be the same with respect to earth. This is important because the Quadrant sonsists of 64 processors and a CU and the operating voltages should be the same.

The Quadrant has been completely isolated from the ground floor via an insulating mayerial and it can be considered as floating with respect to the ground floor and touches the earth only via the nine earth cables through the copper plate that we mentioned before. The reason in doing this is avoidance of the ground loops which would have a termendous impact if the Quadrant was allowed to touch the ground (earth). It must be emphasized that two points connected to earth via separate cables will not have the same voltage drop across because the two grounds are different. It is almost practically impossible to achieve an ideal ground (earth). For this reason it is much more preferable to have one common point which approaches the potential of earth and connect to it different parts of a computing system than to have the system touching the earth in several points and suffer the consequences due to the presence of ground loops.

The grounding scheme of the ILLIAC IV, therefore, may be viewed as a huge tree with nine limbs each representing the earth ground of the CU and corresponding PUC. Each limb in turn contains eight main branches, representing the ground chassis of the PUs in each PUC. Each of the main branches supports three smaller branches which correspond to the PE, MLU and PEM power and signal grounds. The leaves of the individual small branches may be thought as representing the ground pins of the hardware of each subunit in the System.

The nine limbs meet the trunk of the tree at approximately the same point with its roots at the other end, found about 18 feet deep into the ground (earth).



THE ILLIAC IV TREE GROUND