GHULAM ISHAQ KHAN INSTITUTE

CS-222 Computer Organization and Assembly Language

ADVANCE COMPUTING ENGINE

(ACE)

SUBMITTED TO:
Dr. Ghulam Abbas (Course Instructor)

PRESENTED BY:
Ahad Lodhi (2020042)
Mohsin Zia (2020244)
Syed Irtaza Haider (2020474)
Zartaj Asim (2020526)

Date of Submission: 25 May'22.

ACKNOWLEDGEMENT

We would like to extend our gratefulness and acknowledge to the support provided to us by Dr. Ghulam Abbas which enabled us to complete this report. The report has assisted us to not only understand the contents of this course, but also helped us to be able to analyze the architecture of the simple basic computer to great extent and be able to make any modifications to the original design in order to make it more efficient.

Table of contents

1. Abstract

Within this report, with the help of designing process taught in this course CS222-Computer Organization and Assembly Language we have been able to construct the Central Processing Unit (CPU) alongside the complete instruction set. The report follows the detailed steps of designing: BUS, ALU (arithmetic and logic unit), and CU (control unit); using the most appropriate registers and memory configurations; defining the format and the set of instructions of the ACE processor from a list of possible modifications. Each section contains the logical background of each component used with extensive use of block diagrams and flowcharts. The objective of ACE is the improved ALU and increased number of instructions in the instruction set. Lastly, designing a computer with many more functionalities with their appropriate micro operations other than what are already present in basic computer.

2. DESIGN AND ARCHITECTURE

2.1 Architectural Background:

An ACE processor like many other processors uses the general purpose Von Neumann Architecture as shown in figure 2.1. The Von Neumann Architecture is based on stored program concept for a computer system. In this architecture there is a processor and a CPU. Memory contains a stored program which consists of different instructions that are directly accessible by the processor and then processor executes these instructions sequentially.

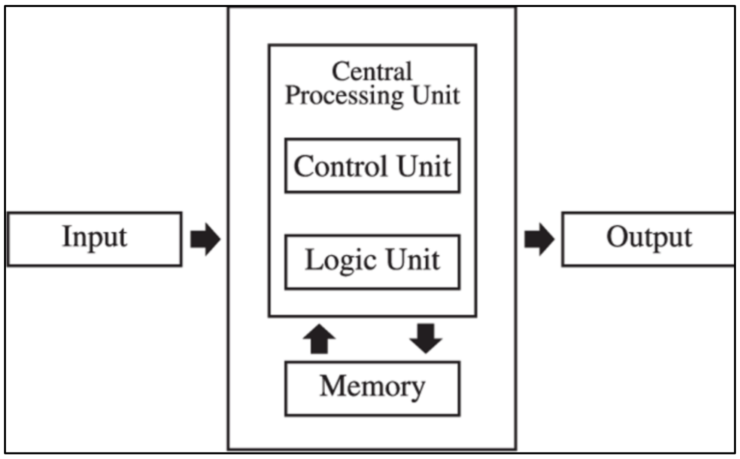
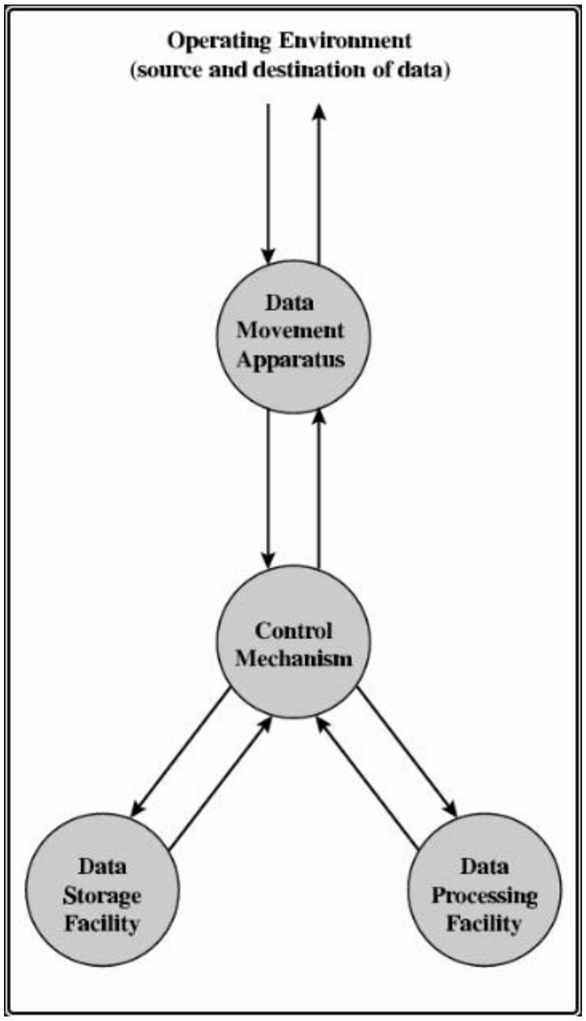


Fig. 2.1 General Purpose Von Neumann Architecture

2.2 Objectives:

To the furthest limit of carrying out a sensible CPU plan with thorough functional abilities, the four primary highlights of any computer system as enlisted in Fig. 2.2:

Fig. 2.2 Four operational attributes of ACE.



- Data Transfer and Communication
- Data Control Function
- Data Processing.
- Data Storage Facility

The objectives for ACE were achieved by incorporating various functionalities and ideas from the basic computer by redesigning the processor along with improved bus structure, register organization, arithmetic and logic unit operations and input output procedure. In order for a better data transmission the bus design is elaborated having many addressing mode. Main memory and processor registers have been logically defined for storage purpose and a complete architecture set of instructions to define data processing by the ALU.

2.3 Bus Network:

The bus structure in the basic computer required a total of n(n-1) number of wire connections due to which the over heads of the hardware (number of internal connections) were higher than what our improvised bus system/structure costs as this bus structure was designed in order to centralize most of the data transfer occurring in ACE processor by interlinking any source to any destination.

Furthermore, instead of using 'n' number of decoders for n-bit registers as used in the basic computer system architecture we only used 1 decoder whose output is then connected commonly in all 3 state gates. For example bit 0 from decoder output is connected to impedance input of the three state gate connected with register A. Then bit 1 from decoder output is connected to impedance input of the three state gates connected with register B. Then bit 2 from decoder output is connected to three state gates connected with register C and finally Then bit 3 from decoder output is connected to three state gates connected with register D.

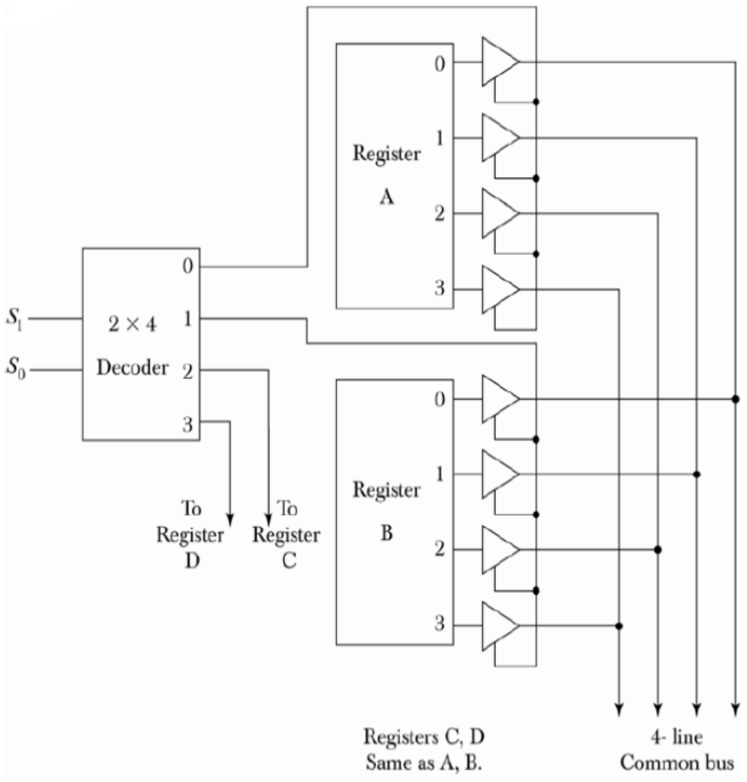
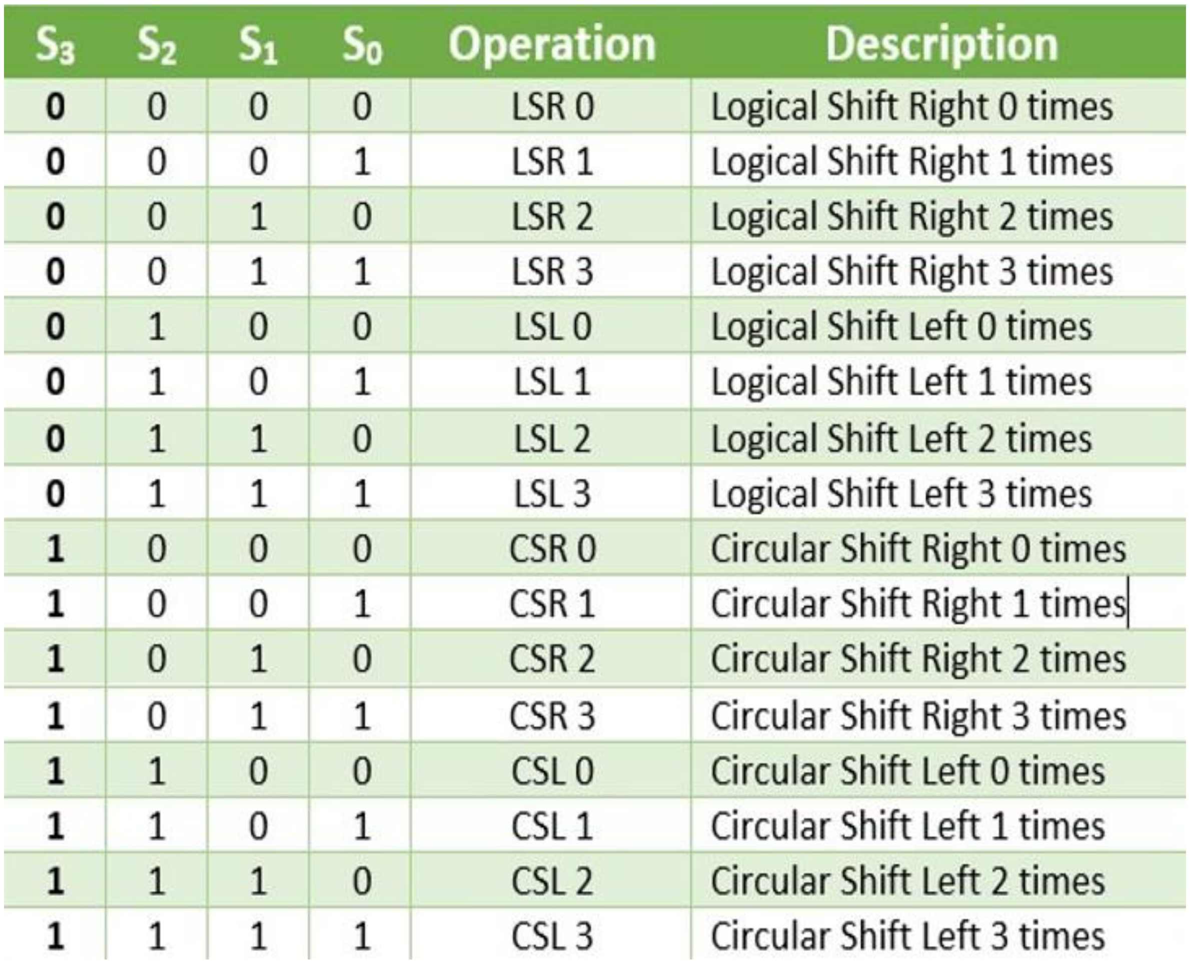


Fig. 2.3 Bus Structure in ACE using Decoder

2.4 Arithmetic, Logic & Shift Unit (ALU):

2.4.1 Shift Unit:

In ACE processor we have used specialized digital electronic circuit known as barrel shifter circuits which can perform up to three shifts in both directions left and right in just one clock cycle. Unlike basic computer which can only perform one shift in either direction in one clock cycle. This design is costly to implement but it is better than that in basic computer if performance is concerned.

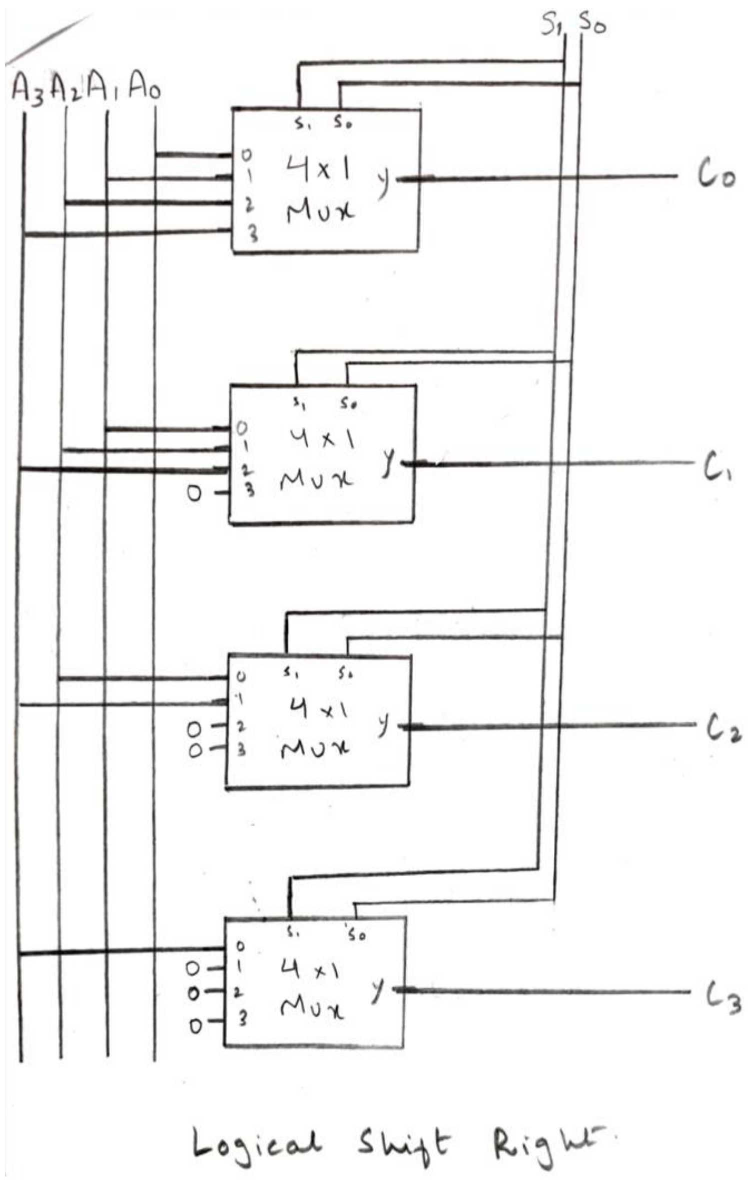


S_3	S_2	S_1	S_0	Operation	Description
0	0	0	0	LSR 0	Logical Shift Right 0 times
0	0	0	1	LSR 1	Logical Shift Right 1 times
0	0	1	0	LSR 2	Logical Shift Right 2 times
0	0	1	1	LSR 3	Logical Shift Right 3 times
0	1	0	0	LSL 0	Logical Shift Left 0 times
0	1	0	1	LSL 1	Logical Shift Left 1 times
0	1	1	0	LSL 2	Logical Shift Left 2 times
0	1	1	1	LSL 3	Logical Shift Left 3 times
1	0	0	0	CSR 0	Circular Shift Right 0 times
1	0	0	1	CSR 1	Circular Shift Right 1 times
1	0	1	0	CSR 2	Circular Shift Right 2 times
1	0	1	1	CSR 3	Circular Shift Right 3 times
1	1	0	0	CSL 0	Circular Shift Left 0 times
1	1	0	1	CSL 1	Circular Shift Left 1 times
1	1	1	0	CSL 2	Circular Shift Left 2 times
1	1	1	1	CSL 3	Circular Shift Left 3 times

Logical Shift Right:

In logical shift right operation we can shift bits from left to right up to three times just by control signals in one clock cycle and 0 will take place from the MSB side. The diagram 2.4.1.1 is based on 4 bits but this operation can take place for 16 bits in our design.

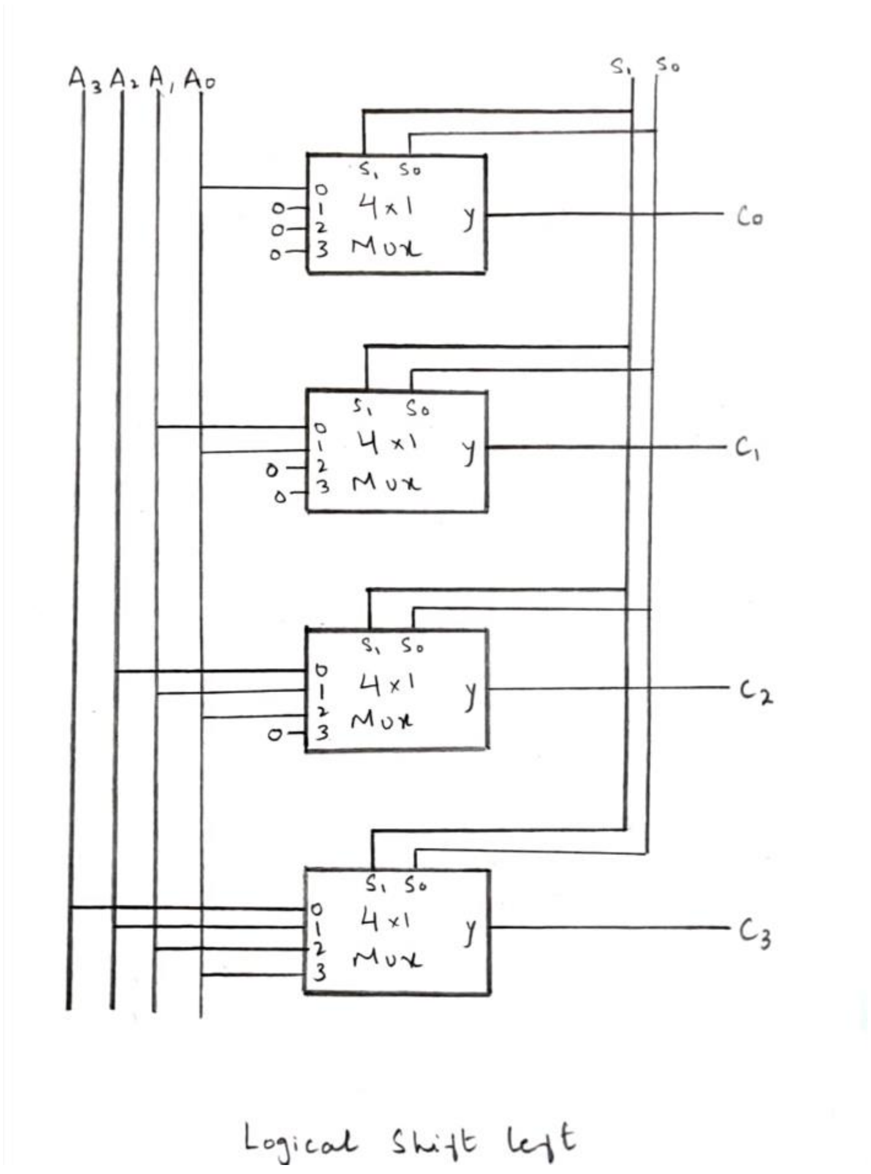
Fig. 2.4.1.1 Logical Shift Right



Logical Shift Left:

In logical left right operation we can shift bits from right to left up to three times just by control signals in one clock cycle and 0 will take place from the LSB side. The diagram 2.4.1.2 is based on 4 bits but this operation can take place for 16 bits in our design.

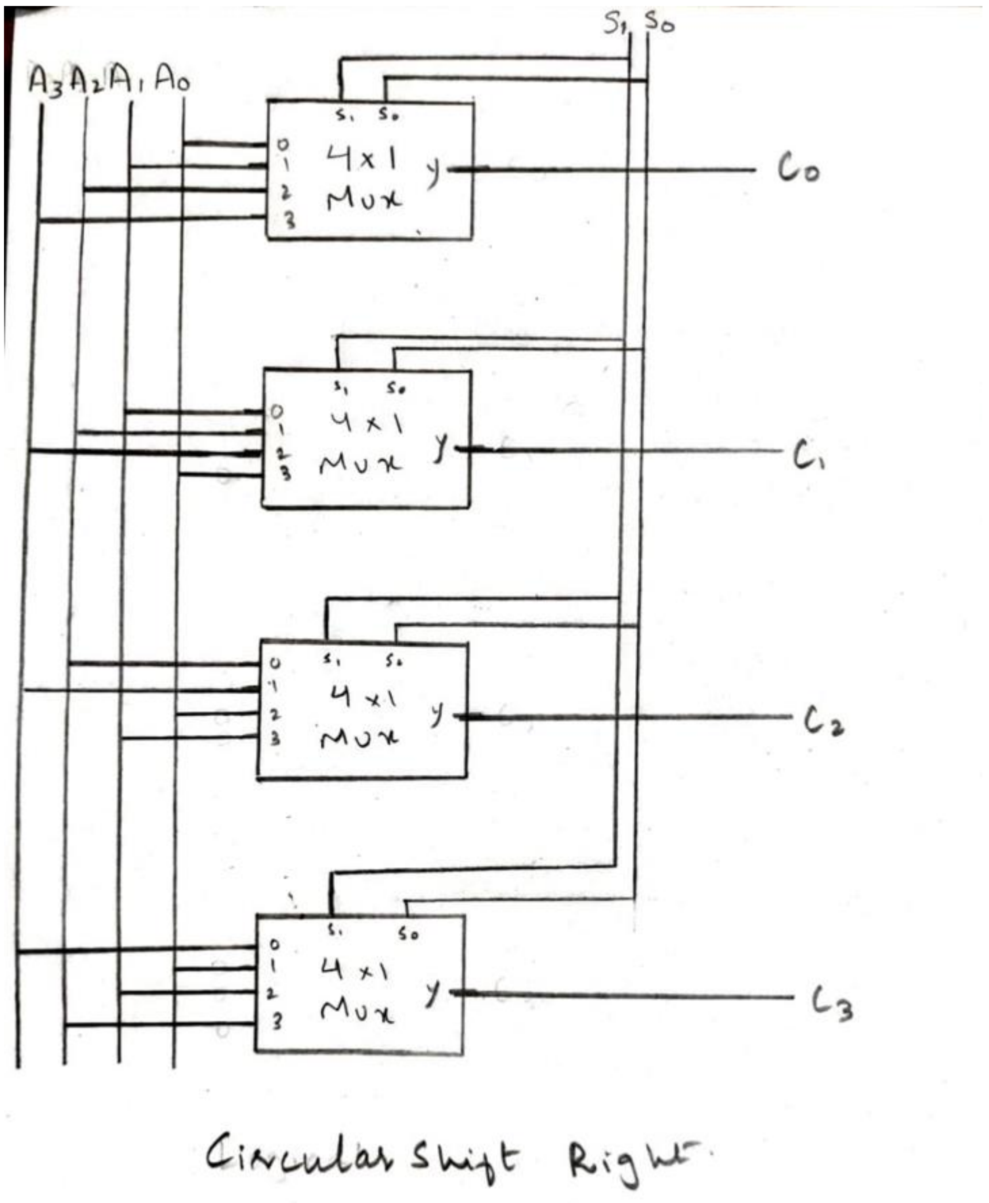
Fig. 2.4.1.2 Logical Shift Left



Circular Shift Right:

In circular shift right operation we can shift bits from left to right circularly bringing LSB to MSB and bringing all bits to next position. We can do this up to three times just by control signals in one clock cycle. The diagram 2.4.1.3 is based on 4 bits but this operation can take place for 16 bits in our design.

Fig. 2.4.1.3 Circular Shift Right



Circular Shift Left:

In circular shift left operation we can shift bits from right to left circularly bringing MSB to LSB and bringing all bits to next position. We can do this up to three times just by control signals in one clock cycle. The diagram 2.4.1.4 is based on 4 bits but this operation can take place for 16 bits in our design.

Fig. 2.4.1.4 Circular Shift Left

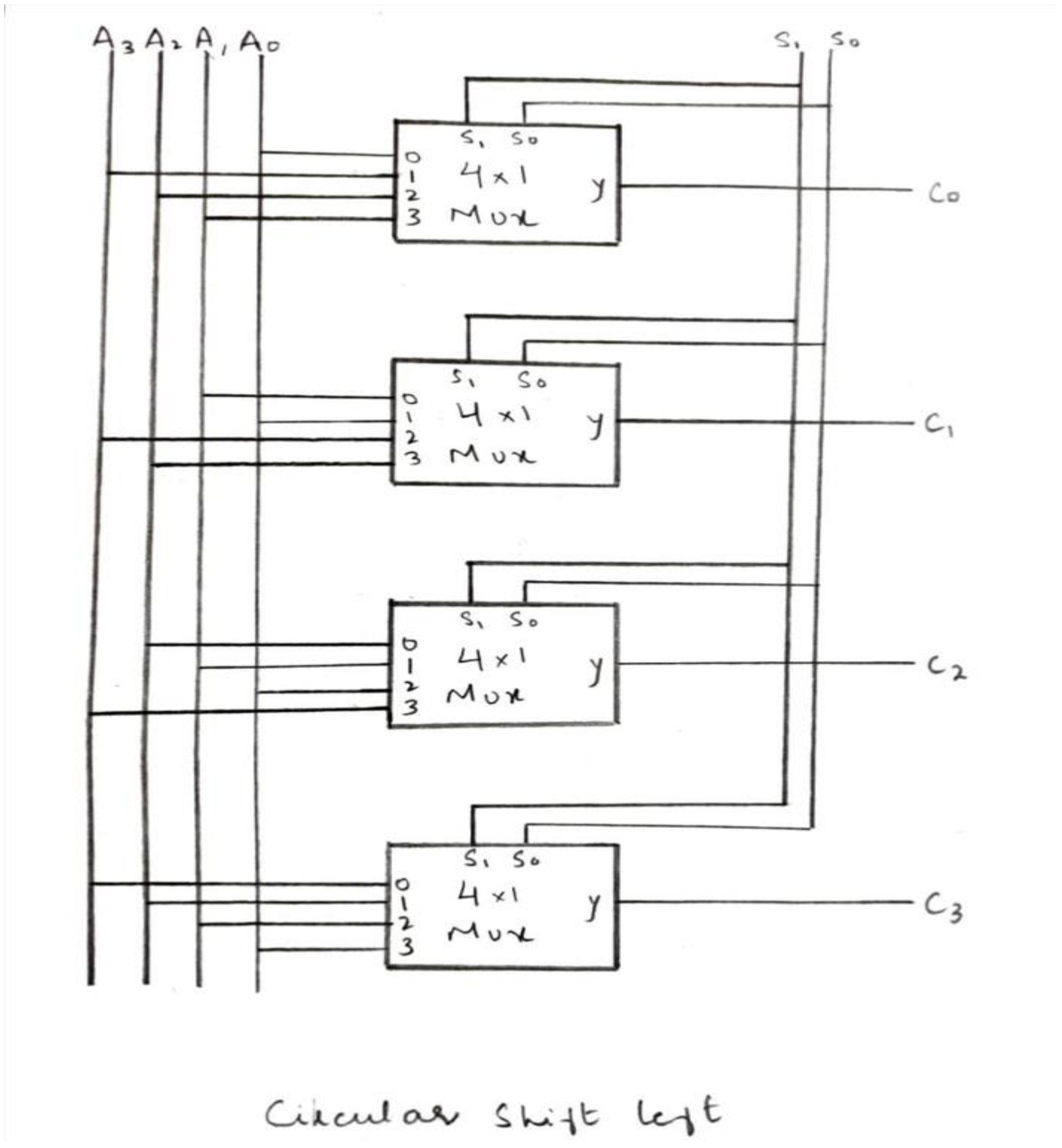
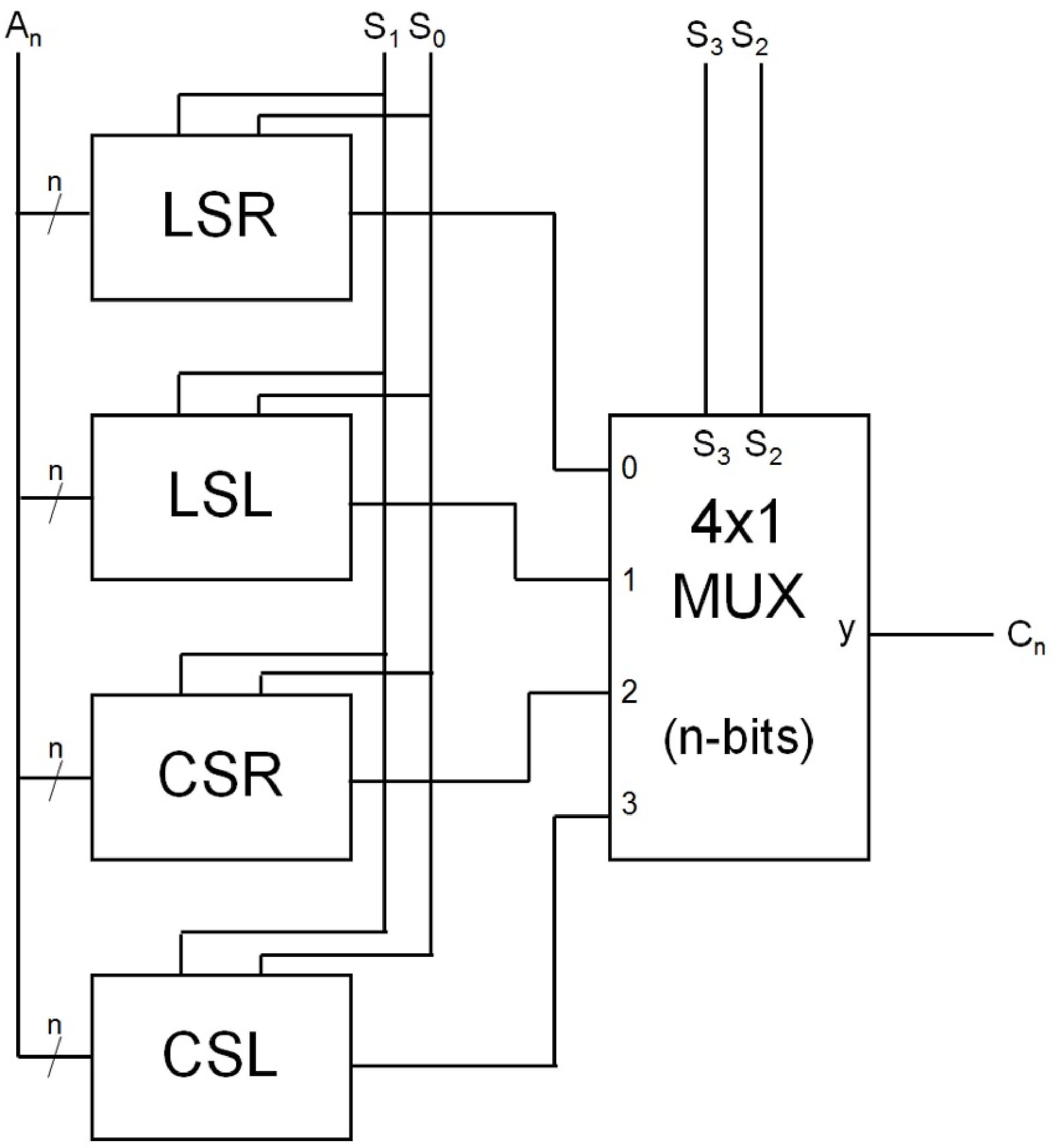
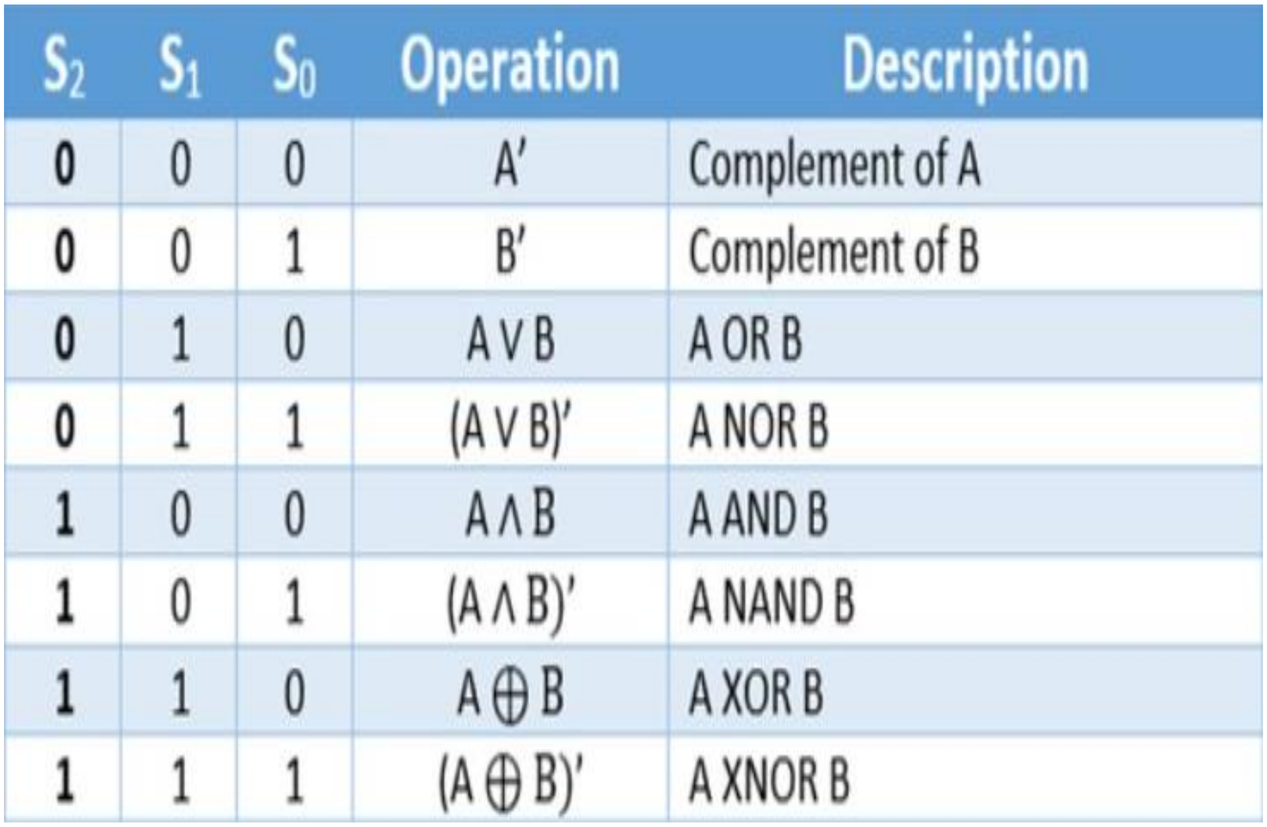


Fig. 2.4.1 Complete Shift Unit



2.4.2 Logic Unit:

In ACE processor the logic unit contains 8 operations with their respective functionalities. In basic computer we had only 4 basic operations. We have also written the microoperations for each of the functionalities we have introduced in our logic unit. The logic unit uses an 8x1 MUX with 3 select lines for implementation. The LU table of simple computer only contained 4 basic operations while the LU table of ACE consists of 8 basic operations hence increasing the functionalities of LU.



S_2	S_1	S_0	Operation	Description
0	0	0	A'	Complement of A
0	0	1	B'	Complement of B
0	1	0	$A \vee B$	A OR B
0	1	1	$(A \vee B)'$	A NOR B
1	0	0	$A \wedge B$	A AND B
1	0	1	$(A \wedge B)'$	A NAND B
1	1	0	$A \oplus B$	A XOR B
1	1	1	$(A \oplus B)'$	A XNOR B

2.4.2.1 LOGICAL UNIT OPERATIONS

Complement A:

Bit in A are inverted using a NOT gate.

Complement B:

Bit in B are inverted using a NOT gate.

A OR B:

Bit in A are OR'ed with bits in B using OR gate.

A NOR B:

Bit in A are NOR'ed with bits in B using OR gate first then inverting the bits by using NOT gate.

A AND B:

Bit in A are AND'ed with bits in B using AND gate.

A NAND B:

Bit in A are NAND'ed with bits in B using AND gate first then inverting the bits by using NOT gate.

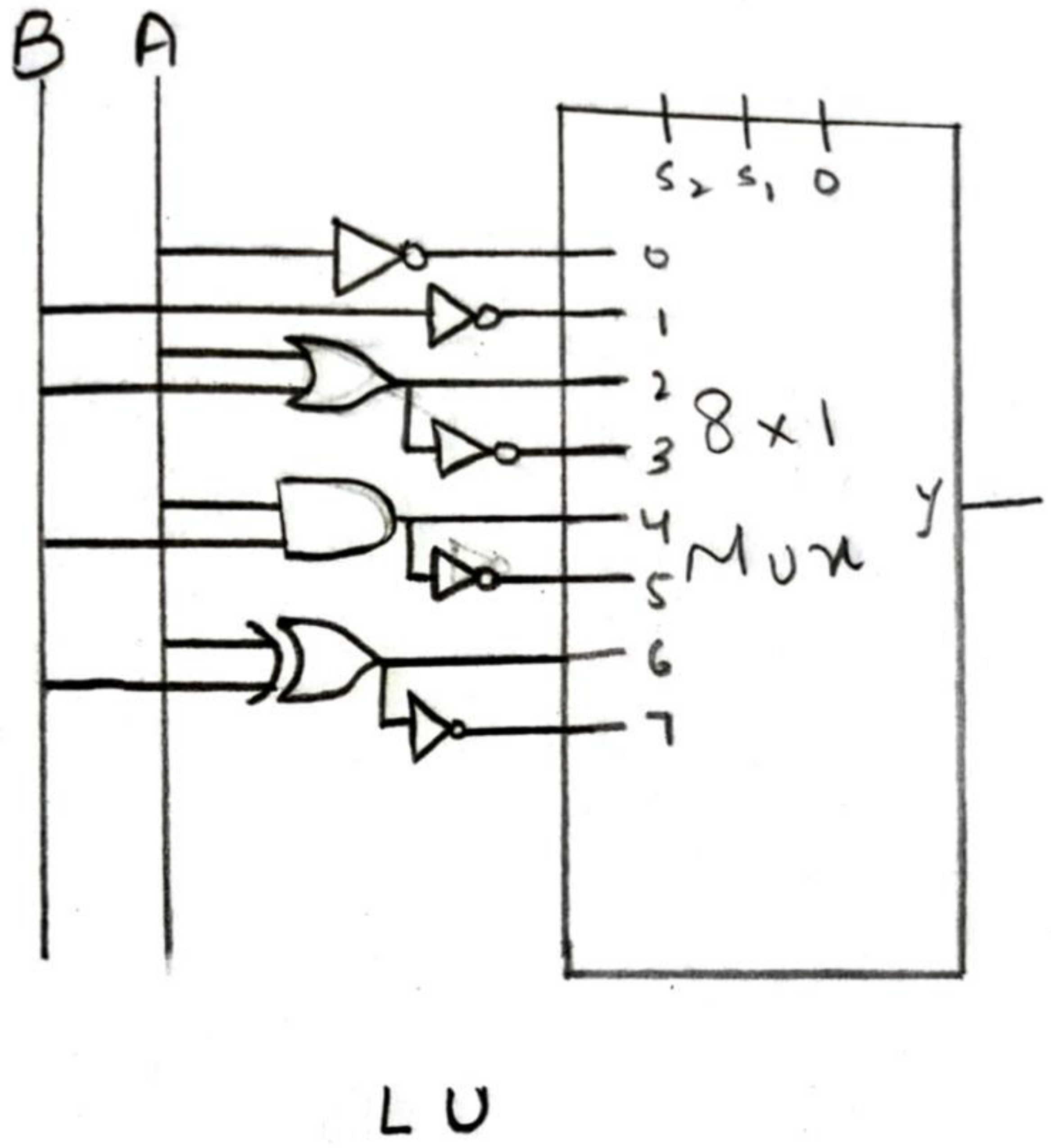
A XOR B:

Bit in A are XOR'ed with bits in B using XOR gate.

A XNOR B:

Bit in A are XNORED'ed with bits in B using XOR gate first then inverting the bits by using NOT gate.

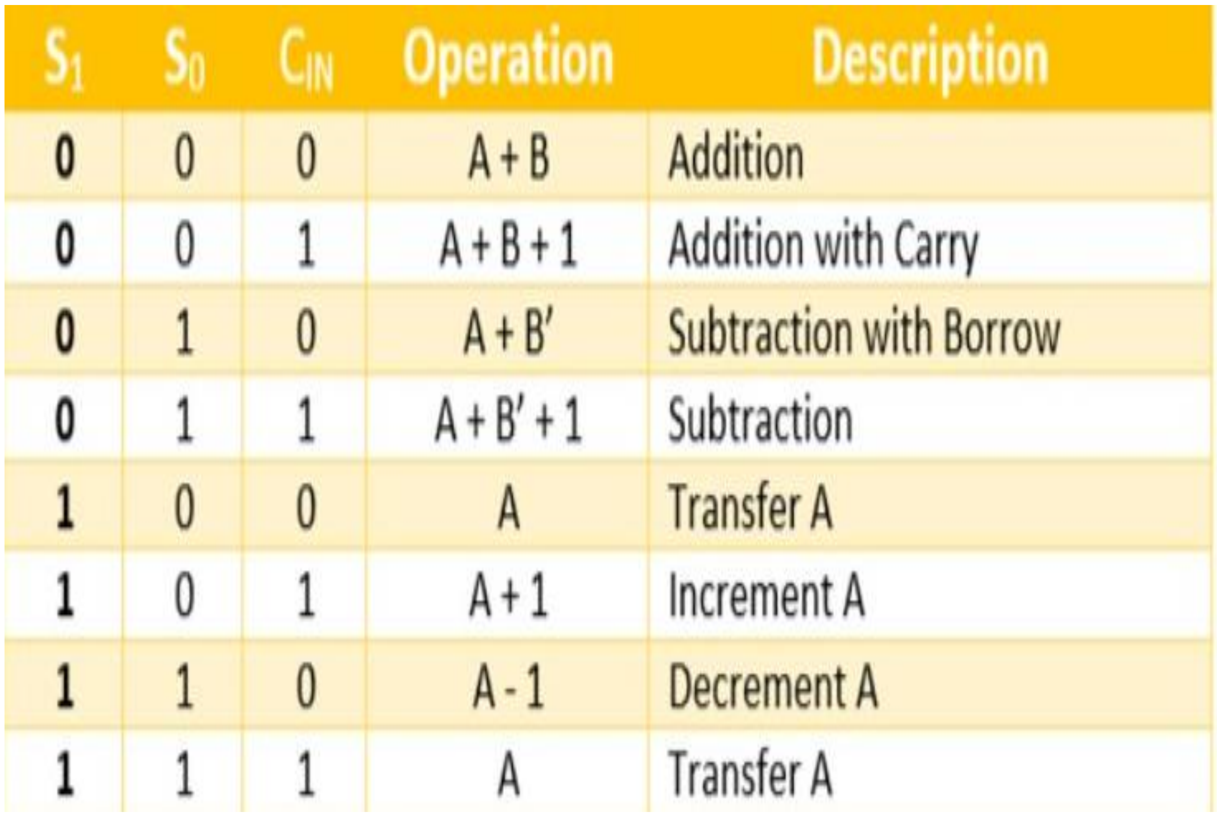
Fig. 2.4.2 Complete Logic Unit



2.4.3 Arithmetic Unit:

The arithmetic micro-operations are performed in the AU which are as follows, The ACE processor can perform all the arithmetic operations that the basic computer can perform.

As decrement, increment, addition and subtraction by using full adders. Figure below display the adder-subtractor circuit diagram with its respective truth table. The full adder circuit is designed in such a way to be able to perform all four basic operations by a single circuit to decrease the number of hardware parts hence reducing overheads/built cost. The operations performed to corresponding inputs are displayed in the figure below.



S_1	S_0	C_{IN}	Operation	Description
0	0	0	$A+B$	Addition
0	0	1	$A+B+1$	Addition with Carry
0	1	0	$A+B'$	Subtraction with Borrow
0	1	1	$A+B'+1$	Subtraction
1	0	0	A	Transfer A
1	0	1	$A+1$	Increment A
1	1	0	$A-1$	Decrement A
1	1	1	A	Transfer A

2.4.3.1 ARITHMETIC UNIT OPERATIONS

Addition:

Bits of A and bits of B are added with carry in 0.

Addition with carry:

Bits of A and bits of B are added with carry in 1.

Subtract with borrow:

Bits of A are added with complemented bits of B. Hence performing subtraction with borrow .

Subtraction:

Bits of A are added with 2's complement of bits of B. Hence performing subtraction.

Transfer of A:

A is transferred as operation A+0000 is performed.

Increment A:

1 is added to bits of A.

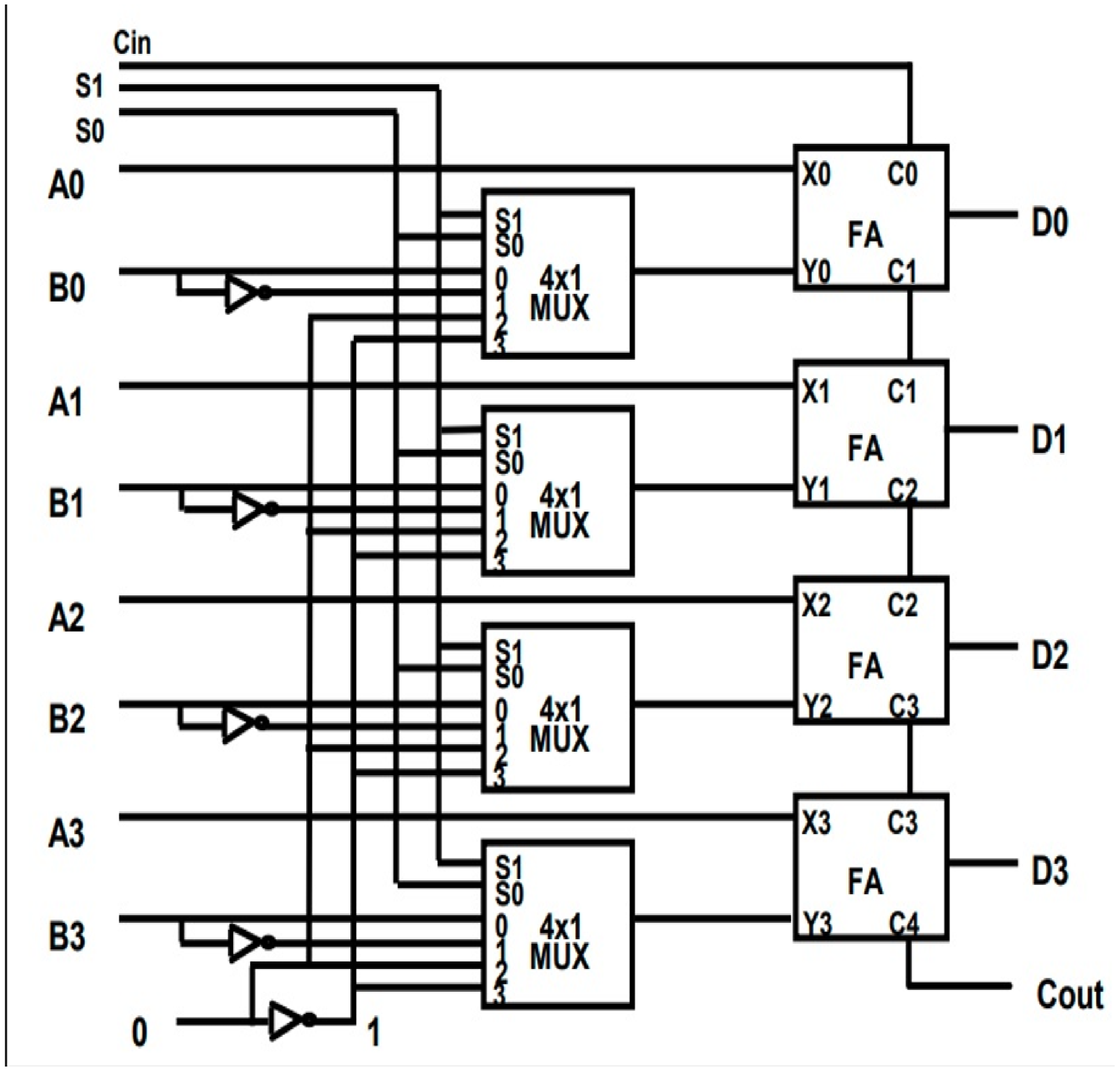
Increment B:

1 is added to bits of B.

Transfer of A:

A is transferred as operation A-1+1 is performed.

Fig. 2.4.3 Complete Arithmetic Unit



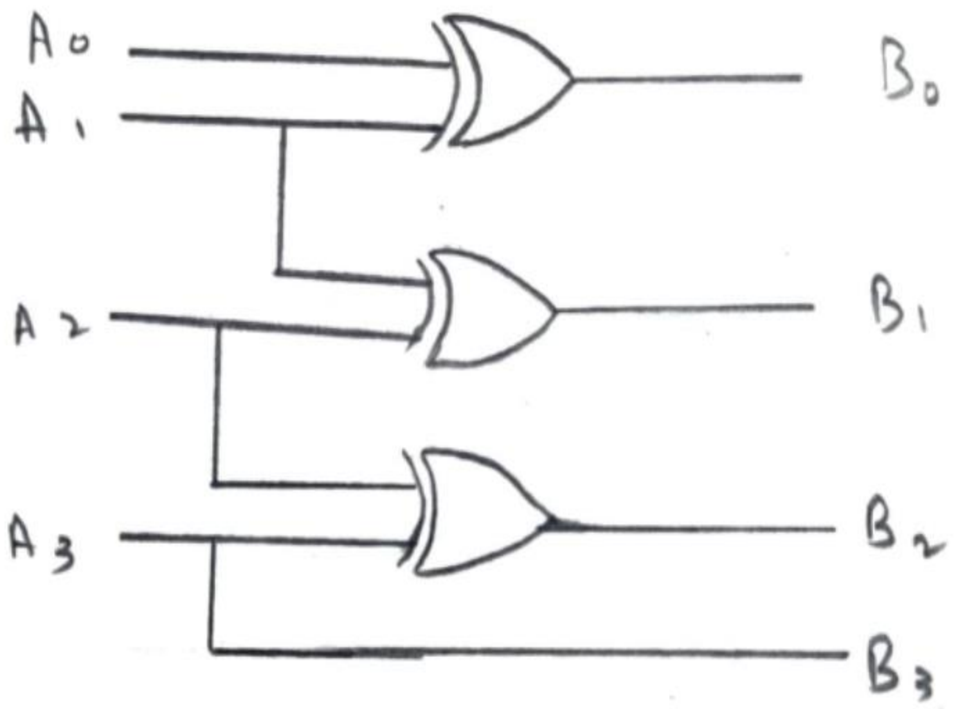
2.4.4 Binary-Gray Conversion Unit:

Binary to gray code and vice versa are performed in this unit. Gray code is the reflected binary code in which adjacent numbers have a single digit differing by 1. This unit is not present in basic computer however we have added it to our ACE.

Binary to gray:

Bits in binary are converted into gray code. Three XOR gates are used in it to convert into gray code bit by bit. $B_i = A_i$ XOR A_{i+1} and $B_n = A_n$ (where n is the last bit).

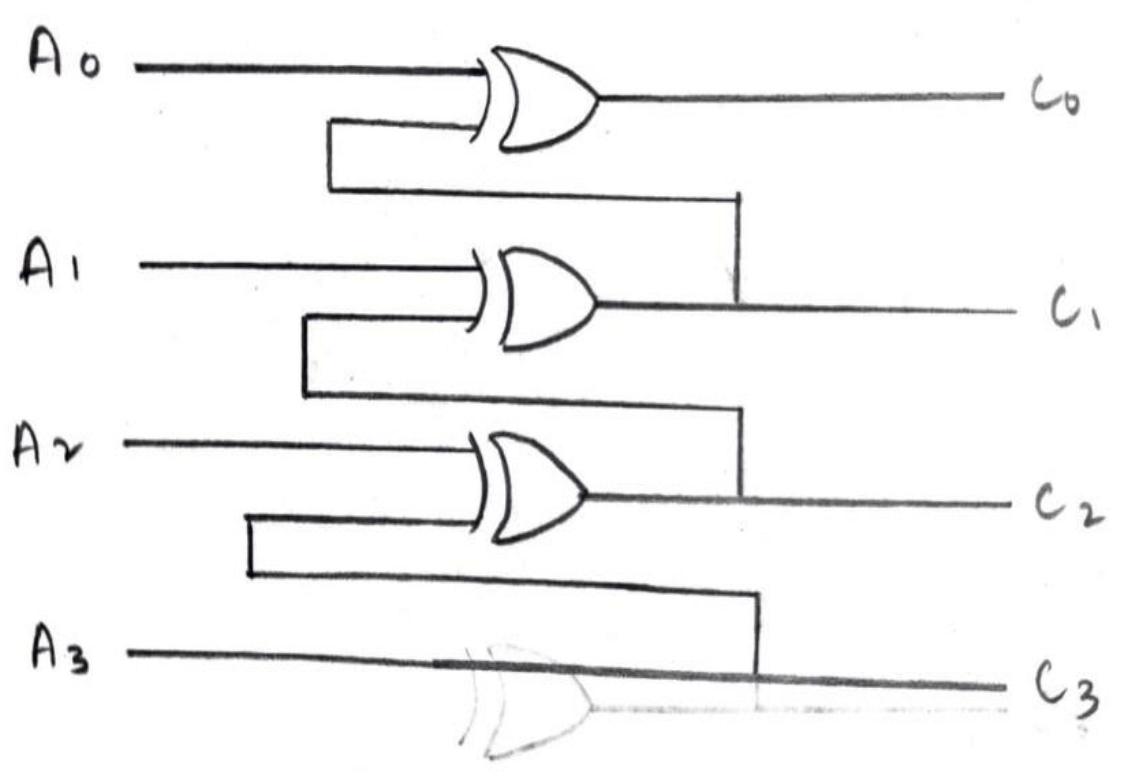
Fig. 2.4.4.1 Binary To Gray Code Circuit



Gray to binary:

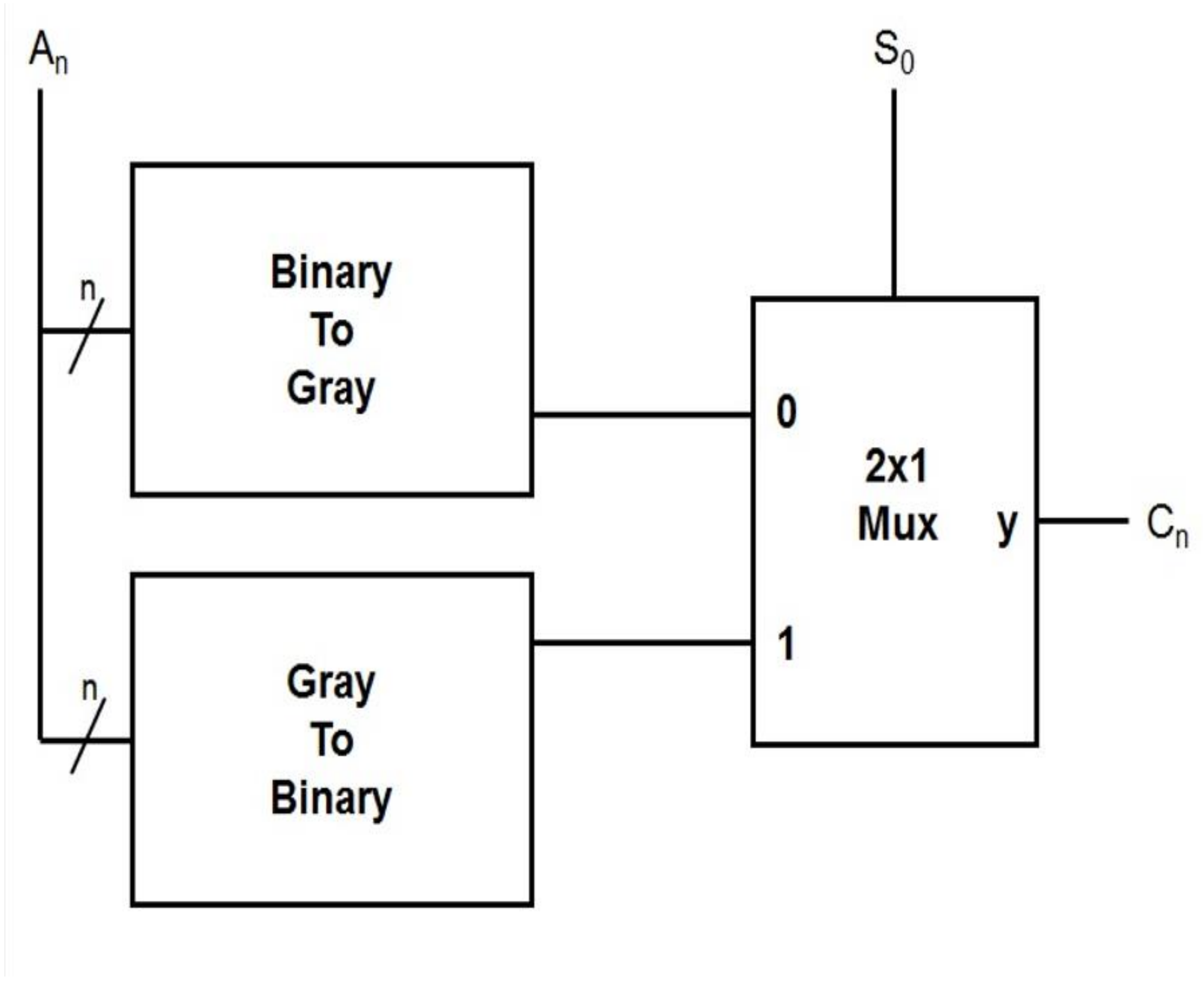
Bits in gray code are converted into binary. Three XOR gates are used in it to convert into binary code by bits. $C_i = A_i$ XOR C_{i+1} and $C_n = A_n$ (where n is the last bit).

Fig. 2.4.4.2 Gray Code To Binary Circuit



One select line is used to select from BTG and GTB. Circuit Diagram below represents Binary-Gray Conversion Unit. Furthermore this unit will be the part of ALU of ACE.

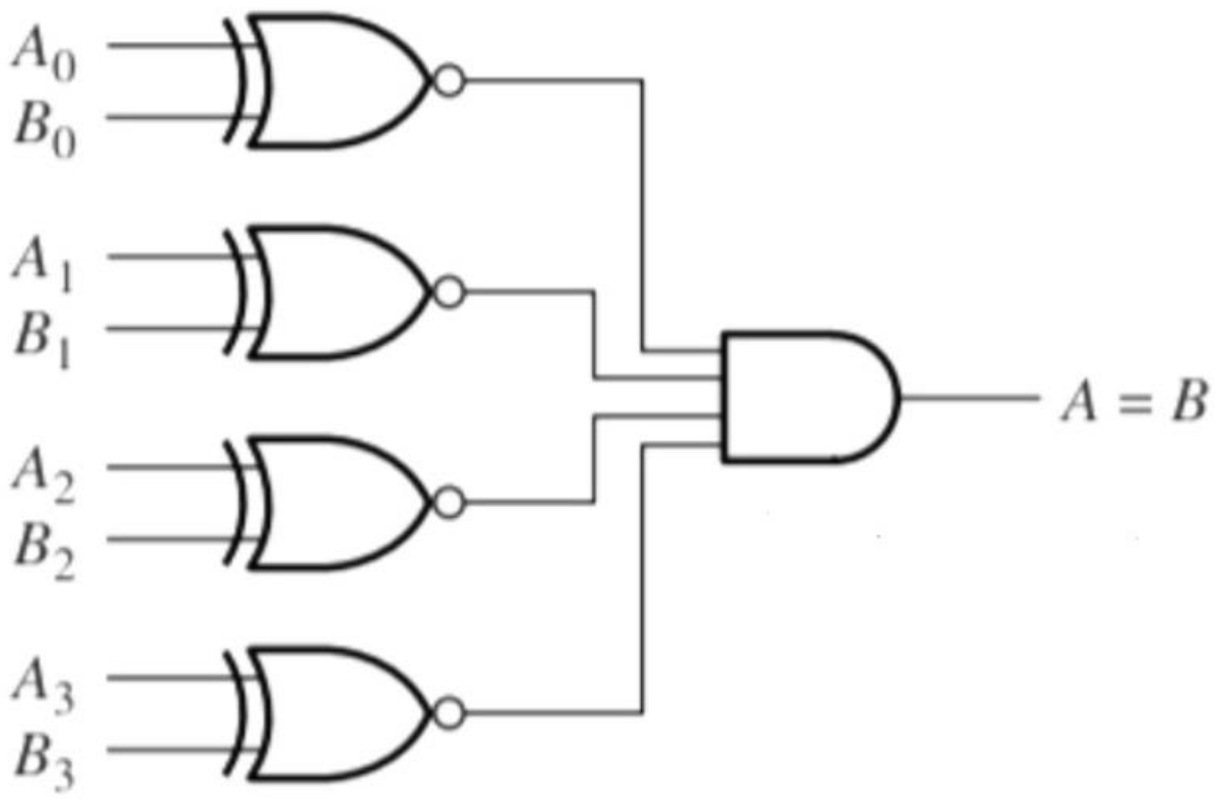
Fig. 2.4.4 Binary-Gray Conversion Unit



2.4.5 Equality Checker Unit:

In this unit we have designed a comparator which it checks either all of the 4 bits of A and B are equal or not. If bits of A and B are equal, then this circuit outputs 1 else output will be 0. 4 XNOR gates are used to check all of the 4 bits and AND is used to check either all 4 bits of A are equal to all 4 bits of B or not. The diagram 2.4.5 is based on 4 bits but this operation can take place for 16 bits in our design.

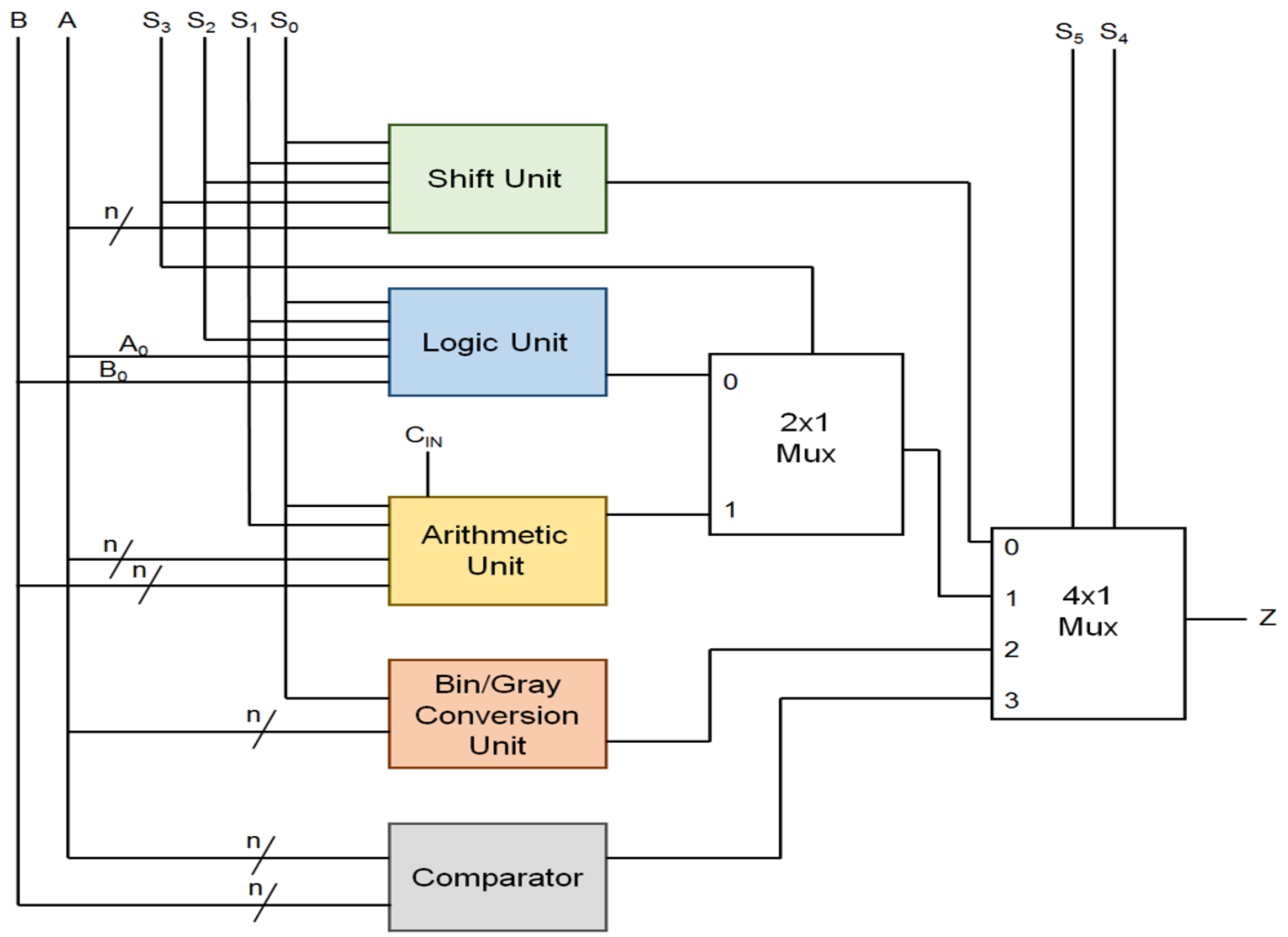
Fig. 2.4.5 Comparator



2.4.6 Design of complete Arithmetic, Logic (& Shift) Unit:

The ALU designed for ACE processor (specifically) can perform shift, logic, arithmetic, conversion operations and comparison on the inputs provided. All the circuits that are designed above are combined through multiplexers to form one unit (i.e. ALU). Total of 5 select lines and 1 input for C_{in} other than inputs A and B are used. Total of 5 select lines can direct 32 instructions but in ACE the design is in manner that 5 select lines can perform total of 35 instructions.

Fig. 2.4.6 Complete ALU



The table below is the combination of all possible instructions in ALU of ACE's processor.

S_5	S_4	S_3	S_2	S_1	S_0	C_{IN}	Operation	Description
0	0	0	0	0	0	x	LSR 0	Logical Shift Right 0 times
0	0	0	0	0	1	x	LSR 1	Logical Shift Right 1 times
0	0	0	0	1	0	x	LSR 2	Logical Shift Right 2 times
0	0	0	0	1	1	x	LSR 3	Logical Shift Right 3 times
0	0	0	1	0	0	x	LSL 0	Logical Shift Left 0 times
0	0	0	1	0	1	x	LSL 1	Logical Shift Left 1 times
0	0	0	1	1	0	x	LSL 2	Logical Shift Left 2 times
0	0	0	1	1	1	x	LSL 3	Logical Shift Left 3 times
0	0	1	0	0	0	x	CSR 0	Circular Shift Right 0 times
0	0	1	0	0	1	x	CSR 1	Circular Shift Right 1 times
0	0	1	0	1	0	x	CSR 2	Circular Shift Right 2 times
0	0	1	0	1	1	x	CSR 3	Circular Shift Right 3 times
0	0	1	1	0	0	x	CSL 0	Circular Shift Left 0 times
0	0	1	1	0	1	x	CSL 1	Circular Shift Left 1 times
0	0	1	1	1	0	x	CSL 2	Circular Shift Left 2 times
0	0	1	1	1	1	x	CSL 3	Circular Shift Left 3 times
0	1	0	0	0	0	x	A'	Complement of A
0	1	0	0	0	1	x	B'	Complement of B
0	1	0	0	1	0	x	$A \vee B$	A OR B
0	1	0	0	1	1	x	$(A \vee B)'$	A NOR B
0	1	0	1	0	0	x	$A \wedge B$	A AND B
0	1	0	1	0	1	x	$(A \wedge B)'$	A NAND B
0	1	0	1	1	0	x	$A \oplus B$	A XOR B
0	1	0	1	1	1	x	$(A \oplus B)'$	A XNOR B
0	1	1	x	0	0	0	A + B	Addition
0	1	1	x	0	0	1	A + B + 1	Addition with Carry
0	1	1	x	0	1	0	A + B'	Subtraction with Borrow
0	1	1	x	0	1	1	A + B' + 1	Subtraction
0	1	1	x	1	0	0	A	Transfer A
0	1	1	x	1	0	1	A + 1	Increment A
0	1	1	x	1	1	0	A - 1	Decrement A
0	1	1	x	1	1	1	A	Transfer A
1	0	x	x	x	0	x	BIN → GRAY	Binary to Gray Conversion of A
1	0	x	x	x	1	x	GRAY → BIN	Gray to Binary Conversion of A
1	1	x	x	x	x	x	A == B	Checks Equality b/w A and B

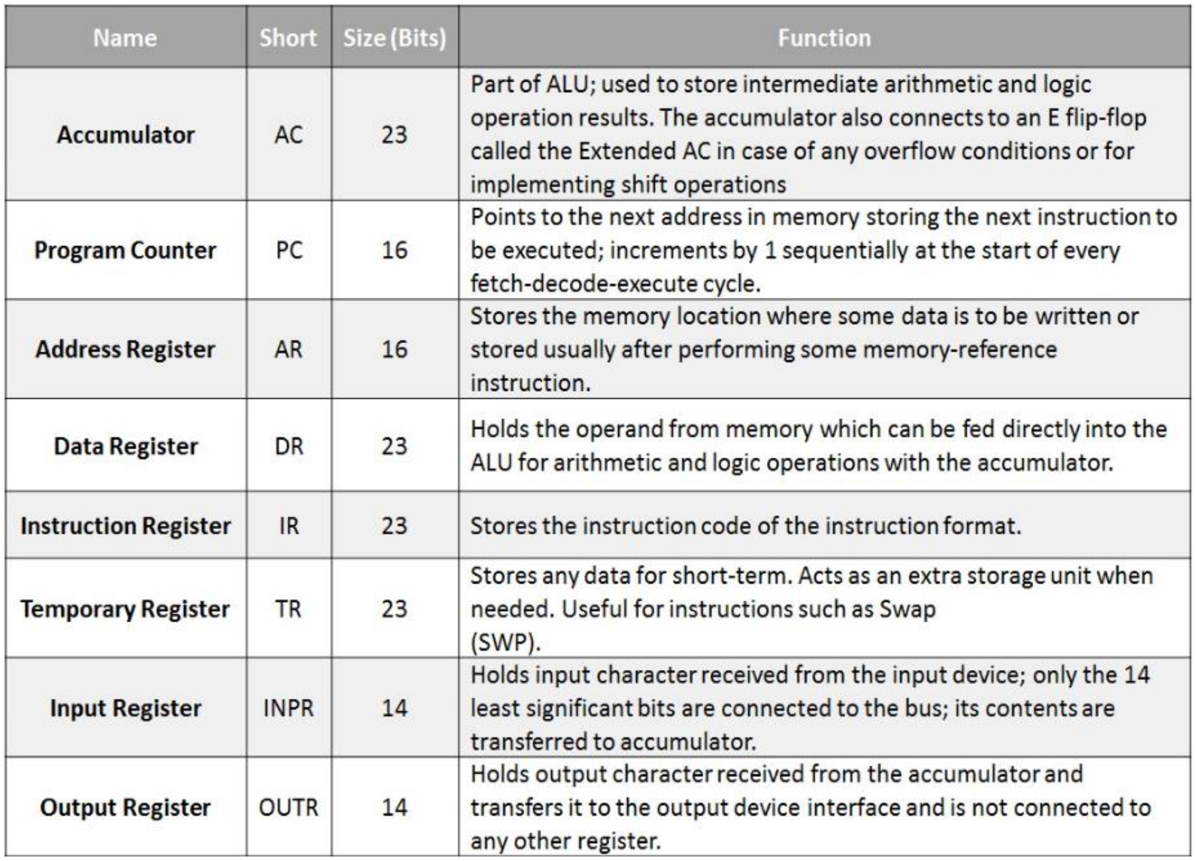
2.4.7 Main Memory Organization and Register Selection:

In ACE’s processor, for the storage of data and instructions, Memory Unit (MU) is included. It is one of the most important part and required mandatory for correct implementation of von-neuman architecture. Control Unit indexes the unidirectional bus presents in memory unit to access the specific words. Also the bi-directional bus is integrated in ACE’s processor to write and read. Furthermore, write and read pins are interconnected with storage unit to direct the exact instruction (given by Control Unit) to be executed.

The main memory of ACE consists of 23-bit words, where operands or addresses are saved in in bits 0 – 15. The length of data and address busses are 23 bits and 16 bits respectively. Storage Unit consists of total of 2^16 = 65535 unique memory cells which can store address or operand. Each unique memory cell can store up to 23 bits of word. Addresses are stored in Address Register which indicates that the particular address in Memory Unit. RTL representation for write operation is X ← M[AR], and for read operation is M[AR] ← X (where X can be any random register).

2.4.7.1 Register Organization:

The Table Below shows and describes functionalities of registers used in ACE. These register used to create a shared transfer system which will be managed and run by Control Unit to perform operations from Instruction Set Architecture.



Name	Short	Size (Bits)	Function
Accumulator	AC	23	Part of ALU; used to store intermediate arithmetic and logic operation results. The accumulator also connects to an E flip-flop called the Extended AC in case of any overflow conditions or for implementing shift operations
Program Counter	PC	16	Points to the next address in memory storing the next instruction to be executed; increments by 1 sequentially at the start of every fetch-decode-execute cycle.
Address Register	AR	16	Stores the memory location where some data is to be written or stored usually after performing some memory-reference instruction.
Data Register	DR	23	Holds the operand from memory which can be fed directly into the ALU for arithmetic and logic operations with the accumulator.
Instruction Register	IR	23	Stores the instruction code of the instruction format.
Temporary Register	TR	23	Stores any data for short-term. Acts as an extra storage unit when needed. Useful for instructions such as Swap (SWP).
Input Register	INPR	14	Holds input character received from the input device; only the 14 least significant bits are connected to the bus; its contents are transferred to accumulator.
Output Register	OUTR	14	Holds output character received from the accumulator and transfers it to the output device interface and is not connected to any other register.

Basic Computer was only compatible with ASCII Character Set. In ACE Processor the INPR and OUTR both have 14-bits so that it is similar with the Unicode Standard and can store 14-bit defined characters whereas the each register of ACE has pins of clear, clock, increment, and load. Sequence Counter (SC) feeds the clock into registers and Control Unit directs the three former pins. Outputs from DR, INPR, and AC are transferred to AC's ALU.

2.4.8 Instruction Format:

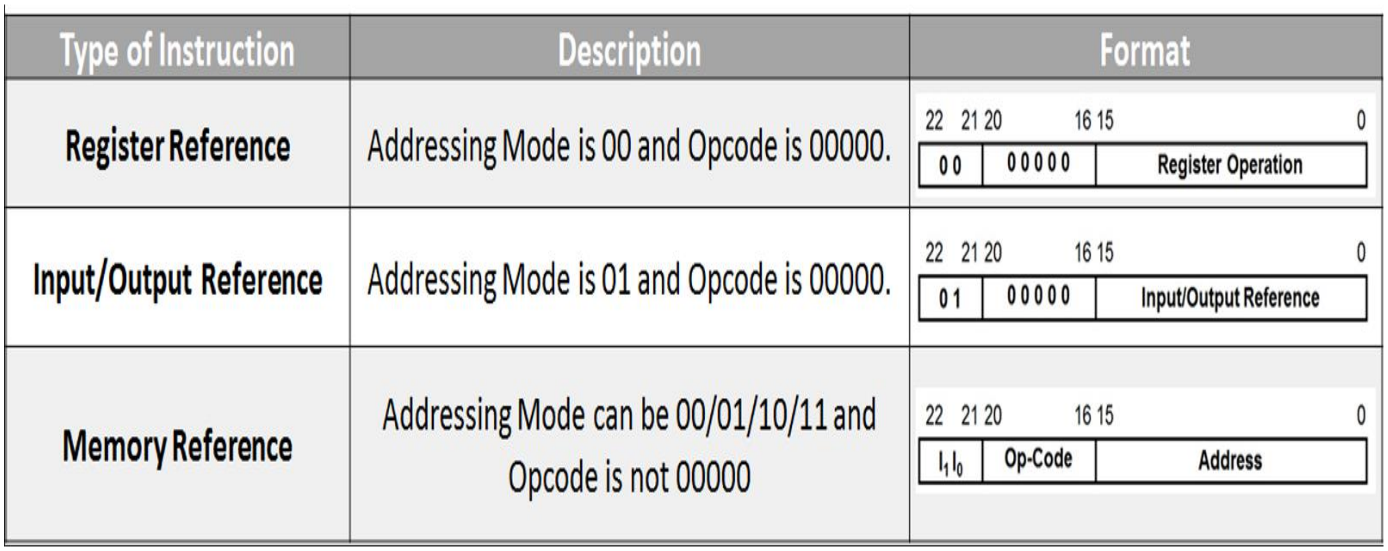
Total of 23 bits are in memory word. Bits 0-15 are for addresses or operands, Bits 16-20 are for operations code (op-code), and Bits 21-22 are for addressing mode.

The format of memory word is divided into three categories:

1. Register-Reference Instructions: These instructions perform operations directly on specific registers. There is no need to of indicating the operand or address from memory.

2. Input/Output-Reference Instructions: These instructions perform operations of input and output which may be required during execution of any program

3. Memory-Reference Instructions: These instructions perform operations on data and instruction in main memory.



Type of Instruction	Description	Format
Register Reference	Addressing Mode is 00 and Opcode is 00000.	22 21 20 16 15 0 00 | 00000 | Register Operation
Input/Output Reference	Addressing Mode is 01 and Opcode is 00000.	22 21 20 16 15 0 01 | 00000 | Input/Output Reference
Memory Reference	Addressing Mode can be 00/01/10/11 and Opcode is not 00000	22 21 20 16 15 0 $I_1 I_0$ | Op-Code | Address

Description:

- If op-code is 00000 then addressing lines will select either Register Reference or I/O Reference will execute.

- If op-code is not 00000 then Memory Reference will execute. Addressing Mode: 00 will do Direct Addressing, 01 will do Indirect Addressing, 10 will do Immediate Addressing, and 11 will do Relative Addressing.

- In direct addressing, operand is saved in that particular address while in indirect addressing; it holds the address of operand.

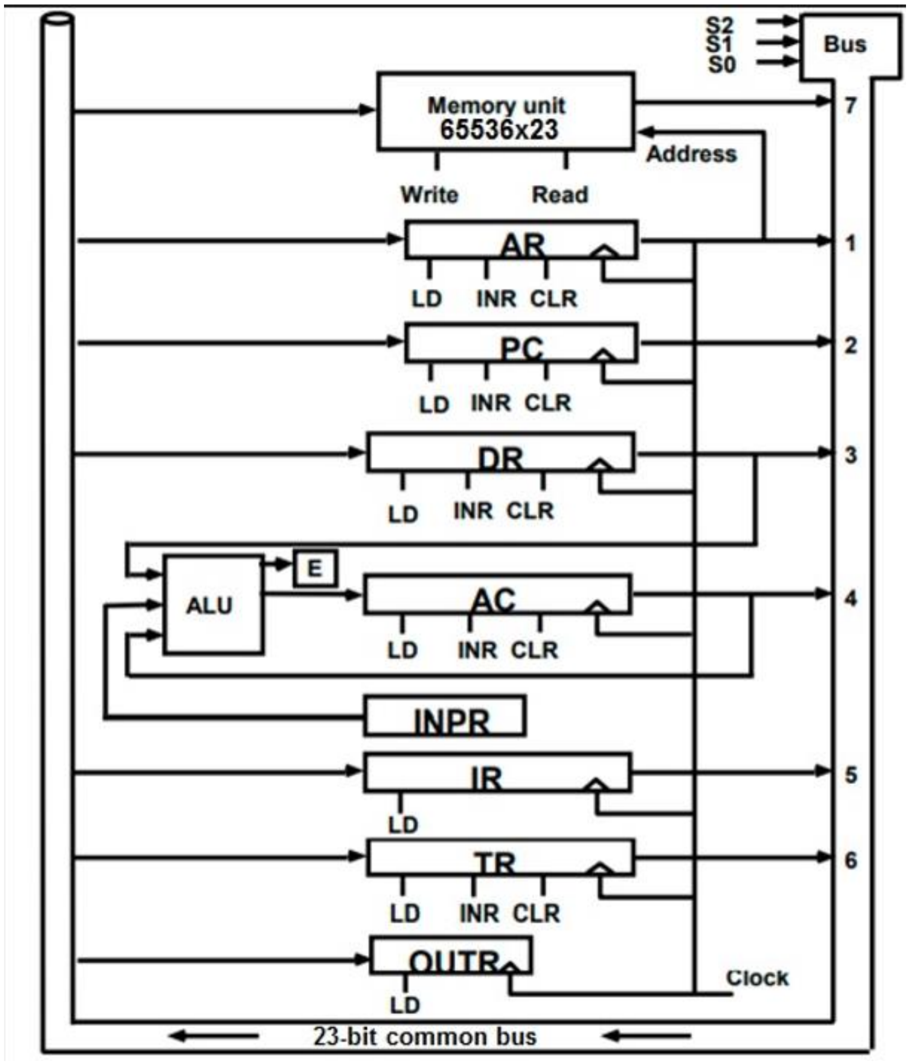
- Immediate addressing doesn’t hold address instead it holds operand to perform operation.

- In relative addressing it holds the number of addresses to jump to get the actual address of operand.

2.4.9 Shared Transfer System:

This section concisely specifies the organization of the processor registers described in the preceding sections. Loading and reading from and to the registers is performed by three select input lines and the registers are connected to a 23-bit common bus.

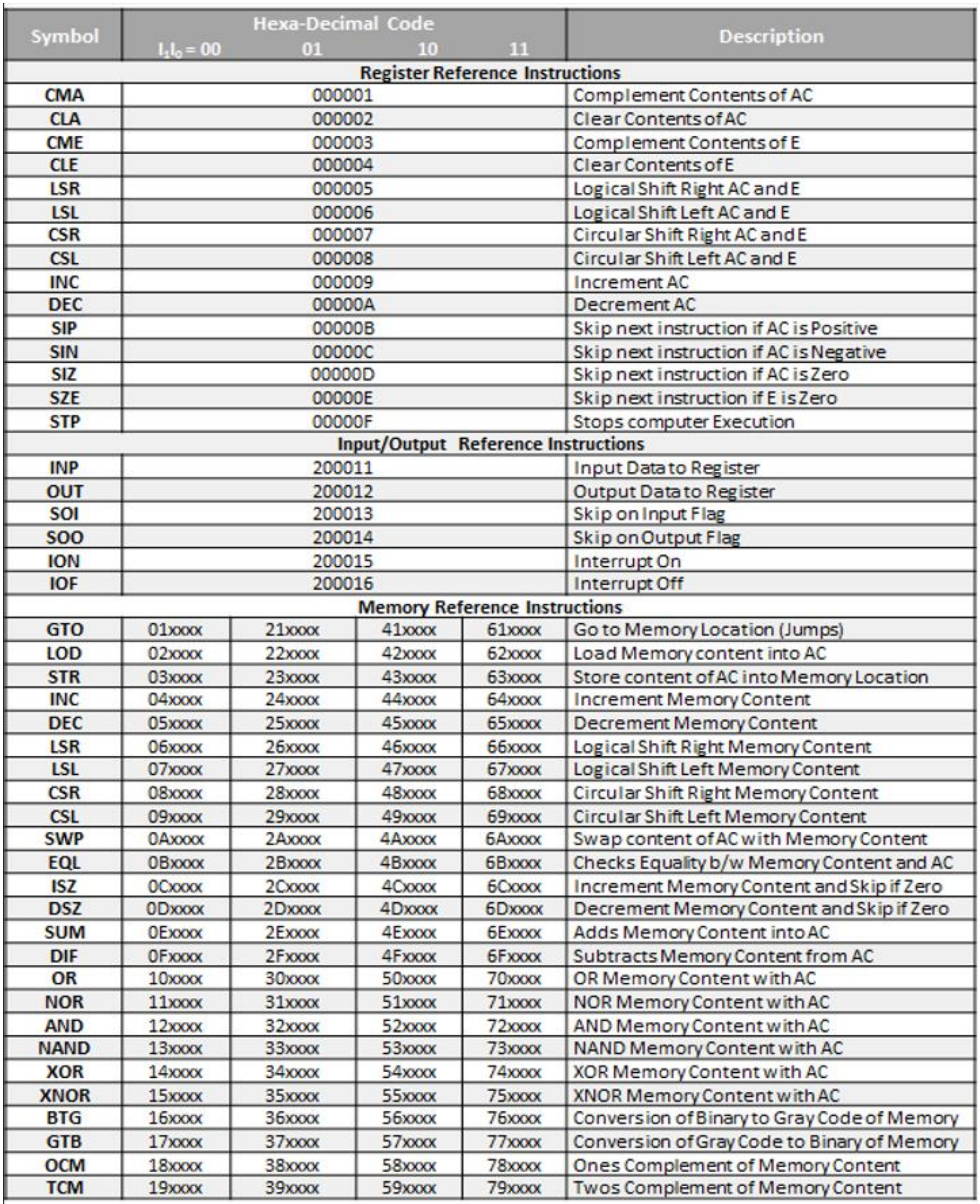
Fig. 2.4.9 Shared Transfer System



It is to be noted that in Shared Transfer System for reading and writing operations, AC is fed inputs by ALU, these inputs are the outputs of AC, DR, and INPR and system does not allow AC to connect directly with the bus. The usage of common bus saves cost of hardware which would have been used in direct connections of registers with another. Other main thing is that feeding of input into register is done externally and its value is directly passed to ALU, so for reading or writing, INPR does not connect to the bus. As the contents to outputted are shown directly on the external output interface, the OUTR does not connect with the Share Transfer System.

3. DEFINITION OF INSTRUCTION SET

The set of instructions implemented by ACE is shown in table below. For the sake of simplicity, instructions in the table are represented in Hexa-Decimal codes. For Hexa-Decimal characters for address bits, one for op-code (16-19) and one for op-code (20) and address mode (21-22). The total size of ACE’s instruction is 23 bits. The below instruction set holds instructions for: status, program sequencing control, input, output, shift, logic, and arithmetic. This gives ease to programmers of assembly language to execute their code. For the application of computer processing, the usage of subroutines is common. So within the instruction set of ACE, functionalities are hardwired to enable subroutine entry to and from the main program. Have a look at the table below.



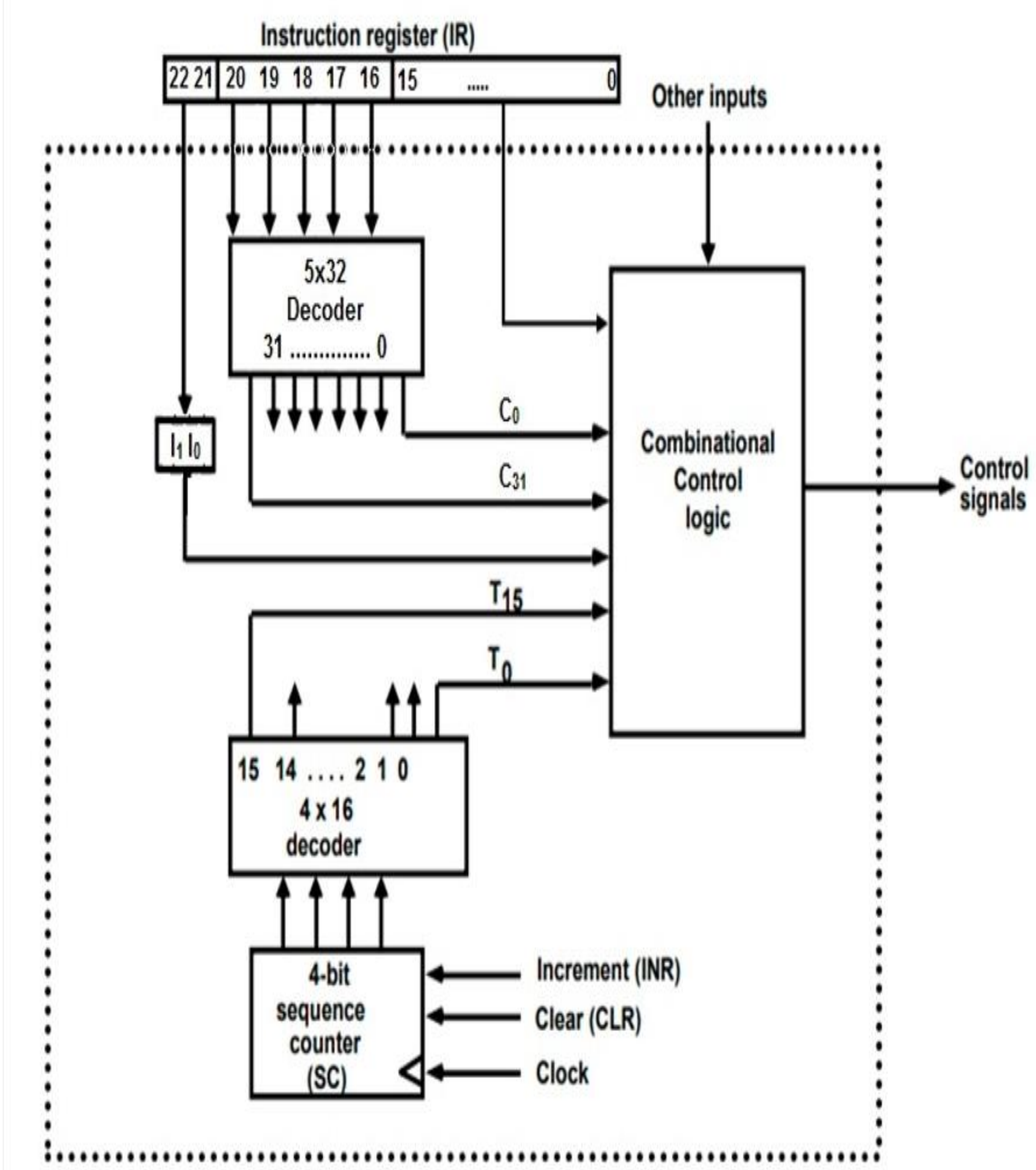
Symbol	Hexa-Decimal Code				Description
	I_1I_0 = 00	01	10	11	
Register Reference Instructions					
CMA	000001				Complement Contents of AC
CLA	000002				Clear Contents of AC
CME	000003				Complement Contents of E
CLE	000004				Clear Contents of E
LSR	000005				Logical Shift Right AC and E
LSL	000006				Logical Shift Left AC and E
CSR	000007				Circular Shift Right AC and E
CSL	000008				Circular Shift Left AC and E
INC	000009				Increment AC
DEC	00000A				Decrement AC
SIP	00000B				Skip next instruction if AC is Positive
SIN	00000C				Skip next instruction if AC is Negative
SIZ	00000D				Skip next instruction if AC is Zero
SZE	00000E				Skip next instruction if E is Zero
STP	00000F				Stops computer Execution
Input/Output Reference Instructions					
INP	200011				Input Data to Register
OUT	200012				Output Data to Register
SOI	200013				Skip on Input Flag
SOO	200014				Skip on Output Flag
ION	200015				Interrupt On
IOF	200016				Interrupt Off
Memory Reference Instructions					
GTO	01xxxx	21xxxx	41xxxx	61xxxx	Go to Memory Location (Jumps)
LOD	02xxxx	22xxxx	42xxxx	62xxxx	Load Memory content into AC
STR	03xxxx	23xxxx	43xxxx	63xxxx	Store content of AC into Memory Location
INC	04xxxx	24xxxx	44xxxx	64xxxx	Increment Memory Content
DEC	05xxxx	25xxxx	45xxxx	65xxxx	Decrement Memory Content
LSR	06xxxx	26xxxx	46xxxx	66xxxx	Logical Shift Right Memory Content
LSL	07xxxx	27xxxx	47xxxx	67xxxx	Logical Shift Left Memory Content
CSR	08xxxx	28xxxx	48xxxx	68xxxx	Circular Shift Right Memory Content
CSL	09xxxx	29xxxx	49xxxx	69xxxx	Circular Shift Left Memory Content
SWP	0Axxxx	2Axxxx	4Axxxx	6Axxxx	Swap content of AC with Memory Content
EQL	0Bxxxx	2Bxxxx	4Bxxxx	6Bxxxx	Checks Equality b/w Memory Content and AC
ISZ	0Cxxxx	2Cxxxx	4Cxxxx	6Cxxxx	Increment Memory Content and Skip if Zero
DSZ	0Dxxxx	2Dxxxx	4Dxxxx	6Dxxxx	Decrement Memory Content and Skip if Zero
SUM	0Exxxx	2Exxxx	4Exxxx	6Exxxx	Adds Memory Content into AC
DIF	0Fxxxx	2Fxxxx	4Fxxxx	6Fxxxx	Subtracts Memory Content from AC
OR	10xxxx	30xxxx	50xxxx	70xxxx	OR Memory Content with AC
NOR	11xxxx	31xxxx	51xxxx	71xxxx	NOR Memory Content with AC
AND	12xxxx	32xxxx	52xxxx	72xxxx	AND Memory Content with AC
NAND	13xxxx	33xxxx	53xxxx	73xxxx	NAND Memory Content with AC
XOR	14xxxx	34xxxx	54xxxx	74xxxx	XOR Memory Content with AC
XNOR	15xxxx	35xxxx	55xxxx	75xxxx	XNOR Memory Content with AC
BTG	16xxxx	36xxxx	56xxxx	76xxxx	Conversion of Binary to Gray Code of Memory
GTB	17xxxx	37xxxx	57xxxx	77xxxx	Conversion of Gray Code to Binary of Memory
OCM	18xxxx	38xxxx	58xxxx	78xxxx	Ones Complement of Memory Content
TCM	19xxxx	39xxxx	59xxxx	79xxxx	Twos Complement of Memory Content

3.1 Control Unit (CU):

Control Unit is one of the most important parts of CPU. CU fetches code of instructions from programs and supervises other units by providing timing signals and control signals. CU interprets instructions and controls sequential instruction execution. It sends and receives control signals from other computer devices and controls and regulates processor timings. Its tasks also include fetching, decoding, executing instructions and storing results.

CU includes of a 5x32 decoder which decodes the op-code and pass signals to combinational logic unit. The other decoder of 4x16 is used which decodes bits coming from sequence counter to timing signals. Combining the sequence counter and decoder both will work like ring counter which helps in sequential execution of instructions to be performed.

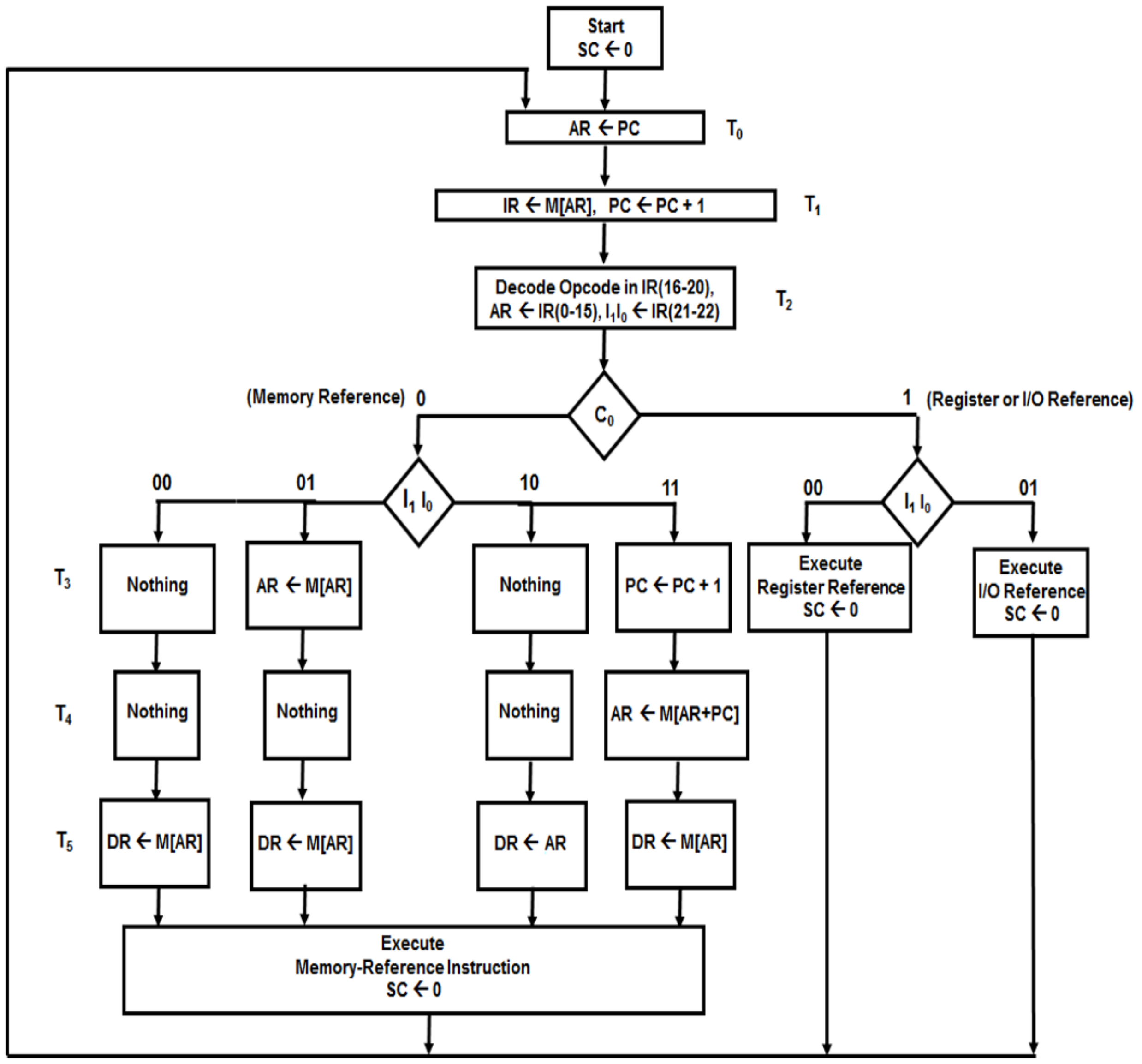
Fig. 3.1 Control Unit



3.2 Fetch-Decode Cycle:

The fetch and decode cycle are common for all the instructions after which the instruction gets executed. This cycle occurs at the start of every timing state. The flowchart below summarizes the fetch and decode cycle.

Fig. 3.1 Fetch-Decode Cycle flowchart



3.3 Input, Output and Interrupt:

Computer has input and output flags to let the processor know if user wants to output the data or take input. If input flag is 1 (FGI=1) then processor stores data into input register and then stores it in accumulator and when output flag is 1 (FGO=1) then data from output register gets copied to output register. It all depends what user wants to implement. Basic computer also has an interrupt enable function (IEN). When this flag is active, all background processing immediately stops and then the task is performed. Processor keeps a check on interrupt flag before fetch and decode instructions if the IEN is active or not.

Fig. 3.3.1 Output flag flowchart

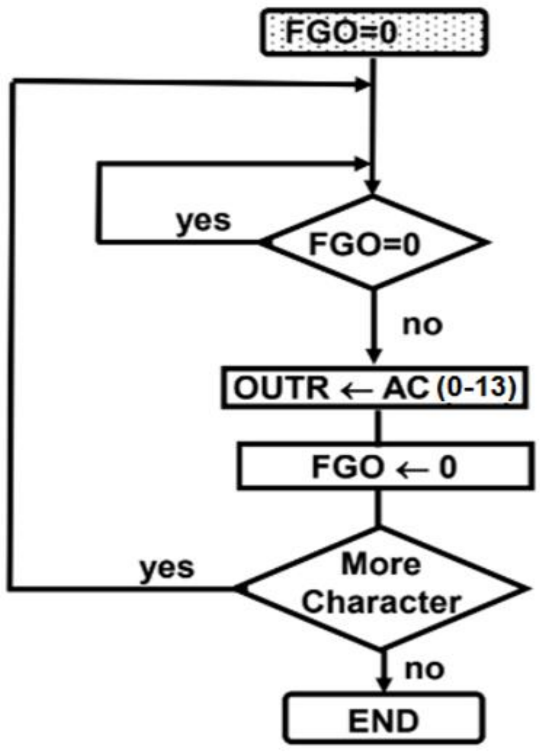


Fig. 3.3.2 Input flag flowchart

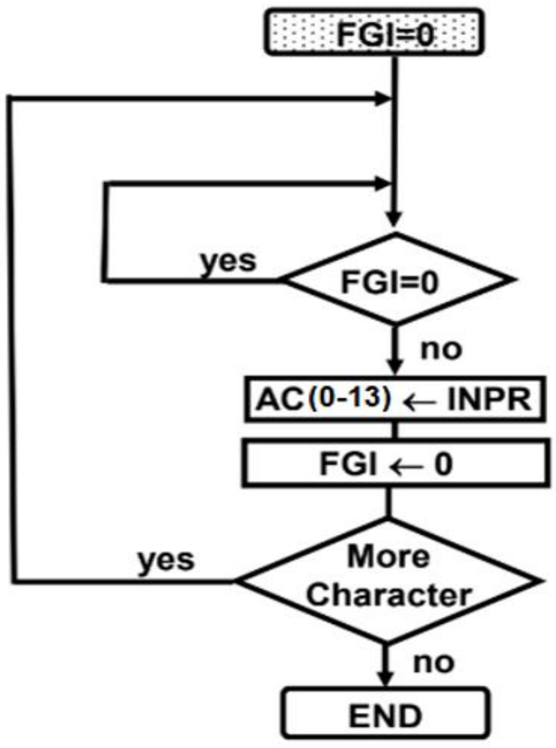
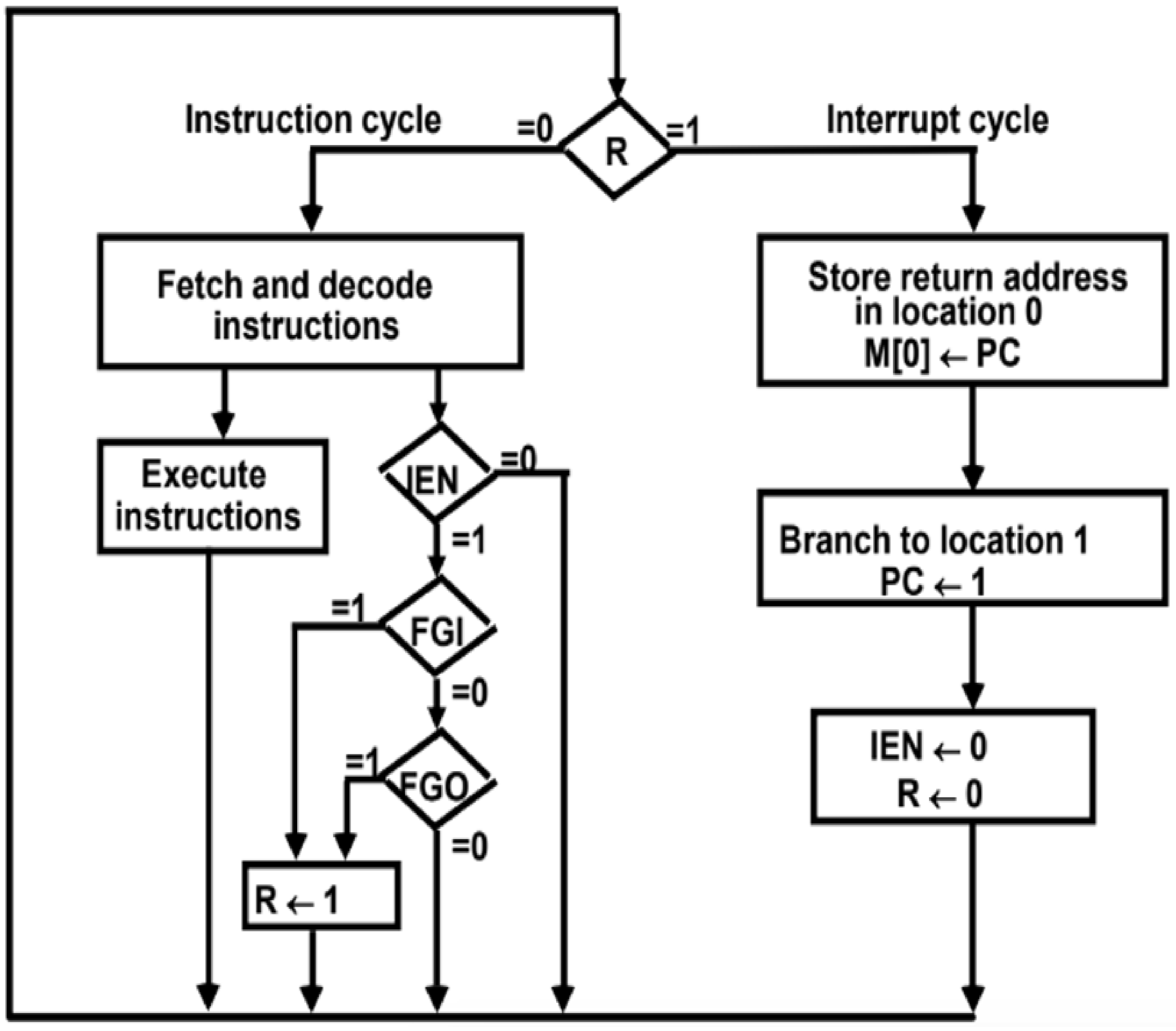


Fig. 3.3.3 Enable flag flowchart



4. COMPLETE COMPUTER DESCRIPTION

The complete computer description with set of instructions, its definition and implementations along with the micro-operations are given in the table below:

Fetch-Decode Cycle	
Fetch	
R'T_0	AR ← PC
R'T_1	IR ← M[AR], PC ← PC + 1
Decode	
R'T_2	D_0, …, D_{31} ← Decode IR(16-20), AR ← IR(0-15), I ← IR(21-22)
Direct Addressing	
R'T_5	DR ← M[AR]
Indirect Addressing	
R'T_3	AR ← M[AR]
R'T_5	DR ← M[AR]
Immediate Addressing	
R'T_5	DR ← AR
Relative Addressing	
R'T_3	PC ← PC + 1
R'T_4	AR ← M[AR + PC]
R'T_5	DR ← M[AR]
Interrupt	
T_0' T_1' T_2' (IEN)(FGI + FGO)	R ← 1
RT_0	AR ← 0, TR ← PC
RT_1	M[AR] ← TR, PC ← 0
RT_2	PC ← PC + 1, IEN ← 0, SC ← 0

Execution Cycles		
Register Reference Instructions		
$C_0 I_1' I_0' T_3$ = n, Bi		
CMA	nB_1	AC ← AC'
CLA	nB_2	AC ← 0
CME	nB_3	E ← E'
CLE	nB_4	E ← 0
LSR	nB_5	AC(0), AC ← lsr AC, AC(22) ← 0
LSL	nB_6	E ← AC(22), AC ← lsl AC, AC(0) ← 0
CSR	nB_7	E ← AC(0), AC ← lsr AC, AC(22) ← E
CSL	nB_8	E ← AC(22), AC ← lsl AC, AC(0) ← E
INC	nB_9	AC ← AC + 1
DEC	nB_A	AC ← AC + all 1's (AC ← AC – 1)
SIP	nB_B	If (AC(22) = 0), then PC ← PC + 1
SIN	nB_C	If (AC(22) = 1), then PC ← PC + 1
SIZ	nB_D	If (AC = 0), then PC ← PC + 1
SZE	nB_E	If (E = 0), then PC ← PC + 1
STP	nB_F	SC ← 0

Execution Cycles		
Input/Output Reference Instructions $C_0 I_1' I_0 T_3 = m$, Bi		
INP	mB_1	AC(0-13) ← INPR, FGI ← 0
OUT	mB_2	OUTR ← AC(0-13), FGO ← 0
SOI	mB_3	If(FGI = 1), then PC ← PC + 1
SOO	mB_4	If(FGO = 1), then PC ← PC + 1
ION	mB_5	IEN ← 1
IOF	mB_6	IEN ← 0

Execution Cycles		
Input/Output Reference Instructions		
GTO	$C_1 T_6$	AR ← PC, SC ← 0
LOD	$C_2 T_6$	AC ← DR
STR	$C_3 T_6$	M[AR] ← AC
INC	$C_4 T_6$	AC ← DR
	$C_4 T_7$	AC ← AC + 1, SC ← 0
DEC	$C_5 T_6$	AC ← DR
	$C_5 T_7$	AC ← AC - 1, SC ← 0
LSR	$C_6 T_6$	AC ← DR
	$C_6 T_7$	AC ← lsr AC, SC ← 0
LSL	$C_7 T_6$	AC ← DR
	$C_7 T_7$	AC ← lsl AC, SC ← 0
CSR	$C_8 T_6$	AC ← DR
	$C_8 T_7$	AC ← csr AC, SC ← 0
CSL	$C_9 T_6$	AC ← DR
	$C_9 T_7$	AC ← csl AC, SC ← 0
SWP	$C_A T_6$	M[AR] ← AC
	$C_A T_7$	AC ← DR, SC ← 0
EQL	$C_B T_7$	AC ← (AC == DR), SC ← 0
ISZ	$C_C T_6$	AC ← DR
	$C_C T_7$	AC ← AC + 1
	$C_C T_8$	if (AC == 0), then PC ← PC + 1, SC ← 0
DSZ	$C_D T_6$	AC ← DR
	$C_D T_7$	AC ← AC – 1
	$C_D T_8$	if (AC == 0), then PC ← PC + 1, SC ← 0
SUM	$C_E T_6$	AC ← AC + DR, E ← C_{out}, SC ← 0
DIF	$C_F T_6$	AC ← AC – DR, E ← C_{out}, SC ← 0
OR	$C_{10} T_6$	AC ← AC ∨ DR, SC ← 0
NOR	$C_{11} T_6$	AC ← AC ∨ DR
	$C_{11} T_7$	AC ← AC', SC ← 0
AND	$C_{12} T_6$	AC ← AC ∧ DR, SC ← 0
NAND	$C_{13} T_6$	AC ← AC ∧ DR
	$C_{13} T_7$	AC ← AC', SC ← 0
XOR	$C_{14} T_6$	AC ← AC ⊕ DR, SC ← 0
XNOR	$C_{15} T_6$	AC ← AC ⊕ DR
	$C_{15} T_7$	AC ← AC', SC ← 0
BTG	$C_{16} T_6$	AC ← DR
	$C_{16} T_7$	AC ← btg AC, SC ← 0
GTB	$C_{17} T_6$	AC ← DR
	$C_{17} T_7$	AC ← gtb AC, SC ← 0
OCM	$C_{18} T_6$	AC ← DR
	$C_{18} T_7$	AC ← AC', SC ← 0
TCM	$C_{19} T_6$	AC ← DR
	$C_{19} T_7$	AC ← AC'
	$C_{19} T_8$	AC ← AC + 1, SC ← 0

5. SIMULATOR (C++)

A visual representation of our ACE processor’s organization, a simulator was constructed by our team members to give an idea of how our processor works and is different from the basic computer. We implemented our simulator on C++ and the detailing of simulator is present in separate handbook.

BIBLIOGRAPHY

[1] Thomas L.Floyd, Digital Fundamentals, 11th Edition, Pearson, July 2014, ISBN: 9780132737968.

[2] Microsoft Visio, 2013.

[3] Paint, 2013.

[4] Morris Mano, Computer System Architecture, 3rd Edition, ISBN: 9788120308558.

[5] Albert P. Malvino & Jerald A. Brown, Digital Computer Electronics, 3rd Edition, McGraw-Hill, 1993,
ISBN: 9780028005942.