Experiment No: 1 8-bit Addition 04-05-2022

Aim: To write an Assembly level language program to perform 8 bit addition using 8085 microprocessor.

Program:

Address	Mnemonic	Comments:
8100	LDA 8500	Load 8500 add. in Accumulator.
8102	Mov B, A	move A Value in B register.
8104	LDA 8501	Load 8501 address value in B register
8107	ADD B	Add B register
8109	STA 8502	store count & Air B - register
810B	RST 1	breakpoint.

Input:

8500 = 24

8501 = 56

output:

8502 = 80

Result:

Thus, the program has been executed sucessfully using 8085 microprocessor.

04-05-2022

8-bit Subtraction

Exp No: 02

Aim: To design an Assembly level language program to implement 8-bit addition using 8085 microprocessors.

Program:

Address	Mnemonics
8100	LDA 8500
8103	MOV B, A
8104	LDA 8501
8107	SUB B
8108	STA 8502
810B	RST 1

Input:

8500 = 45

8501 = 13

output:

8502 = 32

Result:

Thus, the program has been executed Sucessfully with 8085 microprocessors.

8-bit Multiplication: 04-05-23

Exp No: 09

Aim: To design an Assembly level language program to implement 8-bit multiplication using 8085 Microprocessor

Program:

Address	Mnemonics
8100	LDA 8500
8103	MOV B, A
8104	LDA 8501
8107	MOV C, A
8108	CPI 00
810A	JZ LOOP
810D	XRA A
810E	LOOP1: ADD B
810F	DCR C
8110	JZ LOOP
8113	JMP LOOP1
8119	LOOP: STA 8502 RST 1

Input:

8500: 6

8501: 9

Output:

8502 = 54

Result: Thus, the program has been executed Successfully using 8085 Microprocessor.

04-05-2023

8-bit Division

Experiment No: 4

Aim: To write an assembly language program to implement 8-bit division using 8085 processor.

Program

Address	Mnemonics	Comments
8100	LDA 8501	Load 8501 Value to accumulator
8101	MOV B, A	Move Accumulator Value into Register B
8103	LDA 8500	Load 8500 Value to accumulator.
8104	MVI C, 00	Count for Quotient
8107	LOOP: CMP B	Check for A < B.
8109	JC :LOOP 1	If A < B then go to restore.
810A	LOOP 2: CMP B	Subs. Register B
810D	JC LOOP3	to accumulator.
810E	SUB B	Increment program
810J	INR C	Jump into 8109.
8113	JMP LOOP2	loop restore
B114	LOOP3: STA 8503	display result in 8503
8115	MOV A, C.	Move A to C
8117	LOOP1: STA 8502	Store 8502
8119	RST 1	Store the program

PRACTICE: 04-03-2023

Exp: 5

Aim: To design AL program for multiplication two 8-bit using 8085 processing. (4422)

Program:

Address	Mnemonics:
2000	MVI D, 00
2003	MVI A, 00
2004	LXI H, 4150
2005	MOV B, M
2006	INX H
2008	MOV C, M
2009	LOOP: ADD B
200A	JNC NEXT
200C	INR D
200D	NEXT: DCR C
200E	JNZ LOOP
200F	STA 4152
2010	MOV A, D
2013	STA 4153
2015.	HLT.

Input:

4150 = 8

4151 = 7

Output:

4152 = 56

Result:

Thus, the program has been executed successfully with 8085 processors.

Exp: 6:

Aim: To design Assemble language for division of 2 (8-bit No.) using 8085 Microprocessor.

Program:

Address	Mnemonics
4000	LXI H, 4150
4001	MOV B, M
4003	MVI C, 00
4005	INX H
4006	MOV A, M
4008	NEXT : CMP B
	JC LOOP
4009	SUB B
400B	INR C
400C	JMP NEXT
400E	LOOP : STA 4152
400F	MOV A, C
4011	STA 4153
4012	HLT.

Input:

4150 : 16

4151 : 4

output:

8502 4

8503 : 0

Th Result:

The program has been executed Sucessfu

Exp I:

Aim: To design 8-bit addition of two numbers using 8085 processors.

Program:

Address	Mnemonics
8200	MVI C,00
8201	LDA 4150
8203	MOV B,A
8204	LDA 4151
8206	ADD B
8208	JNC LOOP
8209	INR C
820A	LOOP : STA 4152
820D	MOV A,C
820E	STA 4153
8210.	HLT.

Input:

4150 = 14

4151 = 23

output:

4153 = 27

Aim, Result:

Thus, the program has been executed successfully.

Exp: 8:

Aim: To design assembly level language for 2 8-bit subtraction using 8085 Microprocessor

Program:

```
MVI C,00
LDA 4150
MOV B,A
LDA 4151
SUB B
JNC Loop
CMA
INR A
INR C
Loop: STA 4152
MOV A,C
STA 4153
HLT.
```

Input:

4150 = 2

4151 = 6

output: 4

4152 = 4

Result:

Thus, The program has been executed Successfully using 8085 Microprocessor.

Exp: 5 16-bit Addition:

Aim: To write an Assembly language Program to implement 16-bit addition using 8086 processor.

Program: Starting Address : 2000

```
LDA 2080
MOV B,A
LDA 2052
ADD B
STA 2060
LDA 2051
MOV B,A
LDA 2052
ADC B
STA 2061
HLT.
```

Input:

2050 : 6
2051 : 3
2052 : 2
2053 : 8

output:

2060 : 8
2061 : 11

Result:

Thus the program has been executed successfully. using 8086 processor.

8

16-bit Subtraction:

Exp: 6:

Aim: To write an Assembly language program to implement 16-bit subtraction using 8086 processor.

Program:

```
LHLD 2050
XCHG
LHLD 2052
MVI C,00
MOV A,E
SUB,L
STA 2054
MOV A,D
SBB H
STA 2055
HLT.
```

Input:

2050: 6	2050 : 08
2051: 3	2051 40
2052: 2	2052 : 02
2053: 8	2053 : 10

output:

2060 : 8	2054 : 06
2061 : 11	2055 : 30

Result:

Thus, the program has been executed successfully.

16-bit Multiplication:

Exp:7:

Aim: To design a 16-bit Multiplication using assembly level language program using 8086 Microprocessor

Program:

```
LHLD 2001
XCHG
LHLD 2020
MOV C,M
MVI A,00H
LOOP: ADD D
DCR C
JNZ LOOP
MOV H,A
MOV B,L
MVI A,00H
LOOP: ADDE
DCR B
JNZ LOOP
MOV L,A
SHLD 2050.
HLT.
```

Input:

2001: 8

2002: 4

Output:

2010: 32

Result:

Thus, The program has been executed Successfully.

16-bit division:

Exp:8

Aim: To design a 16-bit division by Assembly level language using 8085 Microprocessor.

Program:

```
LHLD 2001
XCHG
LHLD 2005
MOV A,D
MOV B,H
MVI C,00H
LOOP: INR C
SUB B
JNZ LOOP
MOV H,L
MOV A,E
MOV B,L
MVI C,00H
LP: INR C
SUB B
JNZ LP
MOV L,L
SHLD 2050
HLT.
```

input:

2001 : 8

2002 : 4

output:

2004 : 02.

Result: Thus, the program has been successfully.

Factorial

Exp: 09

Aim: To design a factorial of a given number using assembly language in 8085 Microprocessor

Program:

```
LDA 2001
MOV B, A
MVI C, 01
MVI E, 01
LOOP: MOV D, C
MVI A, 00H
LP: ADDE
DCR D
JNZ LP
MOV E, A
INR C
DCR B
JNZ LOOP
MOV A, E
STA 2010
HLT.
```

input:

2001 : 5

output:

2010 : 120

im, the program has been executed sucu
with 8085 processors.

Swaping of two Numbers:

Exp: 10:

Aim: To Swaping of two Numbers using 8085 Microprocessors (Assembly language).

Program:

```
LDA 2001
MOV, B, A
LDA 2002
MOV C, A
STA 2003
MOV A, B
STA 2005
HLT.
```

Input:

2001: 2
2002: 7

Output:

2004 : 7
2005 : 2

Result:

Thus the program has been executed Successfully using 8085 processor.

1's and 2's complements:

Ex: 11.

Aim: To find the 8-bit Numbers of 1's and 2's complements using 8085 Microprocessor

Program:

```
LDA 3000
CMA
STA 3001
ADI 01
STA 3002
HLT.
```

Input:

3000: 01

3001: 02

~~3002: 02~~

~~3003: 01~~

output:

3010: 02

3011: 01

Result:

Thus, the program has been executed

Exp: 13

Aim: To write the assembly language program for AND operation

Program:

```
MVI  A,6        → 0110
MVI  B,4        → 0100
                  0100
ANA  B
STA  2500
HLT
```

Input:

06H

04H

Output:

2500 04H

Result: The program has been executed successfully using 8085 Microprocessor

Exp: 14

Aim: To find the assembly language program for OR operation.

Program:

```
MVI  A,8
MVI  B,7
ORA  B
STA  2500
HLT
```

Input:

08H

07H

Output:

2500 15H

Rotate Right Accumulator:

Exp: 15:

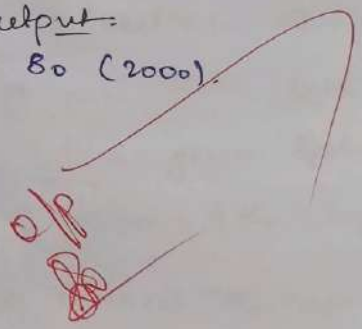
Aim: To write assembly language program for Rotate Right Accumulator.

Program:

```
MVI  A, 05
RRC
RRC
RRC
RRC
STA 2002
HLT.
```

output:

80 (2000)



Result:

Thus, the program has been executed Successfully.

Exp: 17 Two bit Half Adder using logism simulator.

Aim:

To design and implement the two bit half adder using logism.

Truth table:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

→ Insert 2 inputs into the canvas.

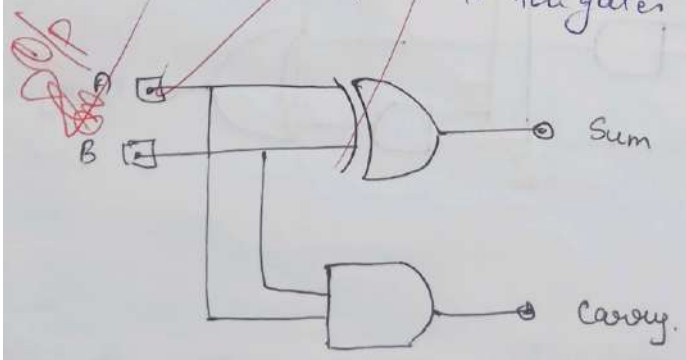
→ Label the inputs (A & B) by setting the attribute label in the attribute table.

→ Note that both inputs have now as .inside them green spots. These are the correct bit value of the input.

→ Connect the inputs to the XOR gate

→ Connect the inputs to the AND gate

→ Connect the output to the gates



Result:

The program has been executed successfully.

8

32-bit full adder using logism.

Exp: 18:

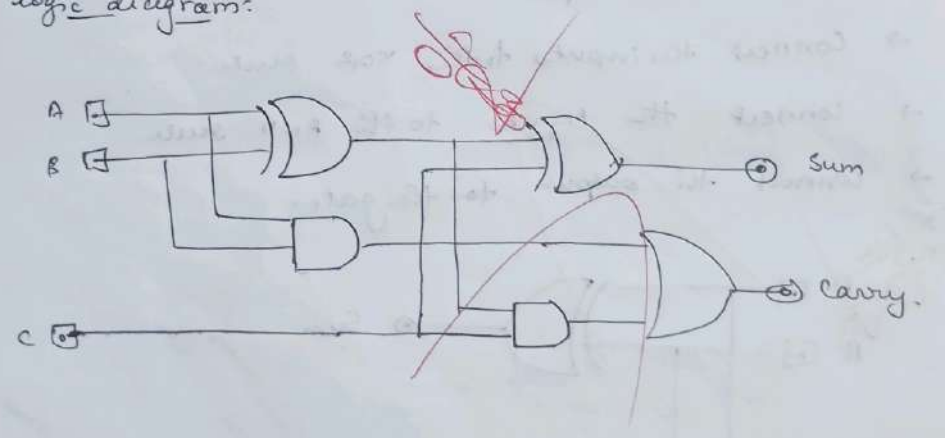
Aim:

To design and implement 32-bit full adder using logism simulator.

Truth table:

S.No	Input			output		Comment
	x	y	Cin	Cout	S	
0	0	0	0	0	0	$0+0+0 = 00_2$
1	0	0	1	0	1	$0+0+1 = 01_2$
2	0	1	0	0	1	$0+1+0 = 01_2$
3	0	1	1	1	0	$0+1+1 = 10_2$
4	1	0	0	0	1	$1+0+0 = 01_2$
5	1	0	1	1	0	$1+0+1 = 10_2$
6	1	1	0	1	0	$1+1+0 = 10_2$
7	1	1	1	1	1	$1+1+1 = 11_2$

logic diagram:



Result:

Thus, the designing of the 32-bit full adder using logism simulator has been executed.

Smallest No in an array:

Exp: 19

Aim: To design Smallest Number in an array using 8085 processor.

Program:

```
LXI H, 2050
MOV C,M
DCR C
INX H
MOV A,M
LOOP1: INX H
CMP M
JC LOOP
MOV A,M
LOOP: DCR C
JNZ LOOP1
STA 3050
HLT
```

input:

2050: 8
2051: 9
2052: 6
2053: 7

output:

3050: 6.

Results: Thus, The program has been executed Successfully.

SIMATS
School of Engineering

Compute

Sum of n Numbers

Exp: 21

Aim: To design a program assembly language for
of N Numbers.

Program:

```
LXI  H, 8000
MOV  C, M
MVI  A, 00
MOV  B, A
LOOP: ADD C
JNC  SKIP
INR  B
SKIP: DCR C
JNZ  LOOP
LXI  H, 8002
MOV  M, A
INX  H
MOV  M, B
HLT
```

Input:

8000: 1
8001: 2
8002: 3
8003: 4
8004: 5

← 8000:05.

Output:

8006: 15

Sum of n Numbers

Exp: 21

Aim: To design a program Assembly language for Sum of N Numbers.

Program:

```
LXI H, 8000
MOV C, M
MVI A, 00
MOV B, A
LOOP ADD C
JNC SKIP
INR B
SKIP : DCR C
JNZ LOOP
LXI H, 8007
MOV M, A
INX H
MOV M, B
HLT
```

input:

8000: 1 ← 8000:05.
8001: 2
8003: 3
8004: 4
8005: 5

output:

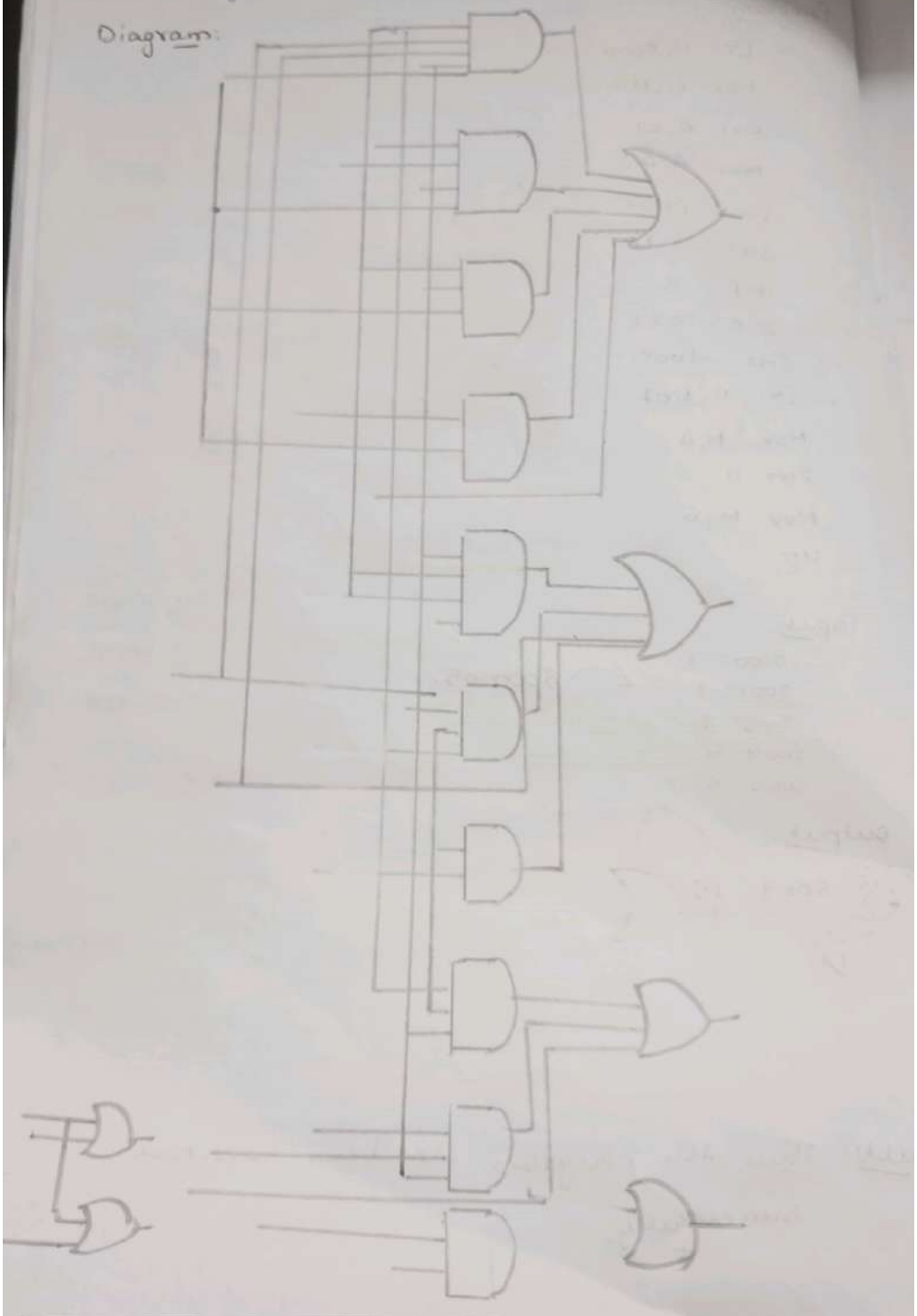
8007: 15

Result: Thus, the program has been executed Successfully.

Exp: 22 Carry look ahead adder:

Aim: To design carry look ahead adder using logisim.

Diagram:



Exp 23 Descending order:

Aim: To design an Assembly language for Descending order

```
LOOP: LXI H, 3500
MVI D, 00
MVI C, 05
LOOP1: MOV A, M
INX H
CMP M
INC LOOP2
MOV B, M
MOV M, A
DCX H
MOV M, B
INX H
MVI D, 01
LOOP2: DCR C
JNZ LOOP1
MOV A, D
RRC
JC LOOP
HLT
```

input:

3500: 2
3501: 6
3502: 5
3503: 8
3504: 1

output:

3500: 8
3501: 6
3502: 5
3503: 2
3504: 1

Result:

Thus, the program has been executed successfully

Exp 24 Ascending order

Aim: To design an Assembly language for Ascending order

```
LOOP : LXI H, 3500
MVI  D,00
MVI  C,05
LOOP1: MOV A,M
INX H
CMP H
JC LOOP2
MOV B,M
MOV M,A
DCX H
MOV M,B
INX H
MOVI D,01
LOOP2: DCR C
JNZ LOOP1
MOV A,D
RRC
JC LOOP
HLT
```

input:

3500 : 1
3501 : 2
3502 : 5
3503 : 8
3504 : 9

output:

3500 : 2
3501 : 1
3502 : 5
3503 : 8
3504 : 9

esult. Thus, The program execution has ning GNUsim8085 successfully.

CPU Performance:

Exp: 29:

Aim: To design a program for CPU performance

Algorithm:

1) Get the input from user.
2) Declare the variable to store the performance factor.
3) if the input not equal to 1; break.
4) check whether the given input is valid or
5) To create and design the required duired input.
6) stop the execution.

Sample input:

Enter the no. of processor : 3
Enter the cycles per instruction : 1.5
Enter the ASI clockrate : 3
Enter the cycle per instruction B : 1
Enter clock rate B : 2.2
Enter the cycle per instruction C : 2
Enter the clock rate C : 1

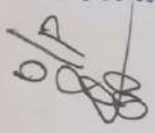
Sample output:

CPU time for A = 500.000

CPU time for B = 454.545

CPU time for C = 2000.000.

The lowest execution time is: B = 454.545.



Result:

Thus, The program has been executed succesfu

4 stage pipeline

Exp: 20

Aim: To design c program for 4- stage pipelining.

Algorithm:

1) Get the input from user.
2) Declare four stages as required.
3) Stage as choice to choice 1. Instruction.
4) Fetch decode write execution Menu is displayed.
5) Process is done in a cycle of 4-stage.
6) In this stage, ALU operations are performed.
7) Display the schedule pipelined value.
8) Stop the execution.

Sample input:

Enter 1st value: 2
Enter 2nd value: 4
Enter the option:
1. Add. 2. Sub. 3. Div. 4. Multiplication
Enter the choice: 4.

Sample output:

Performing Multiplication operation
Cycle value is: 3
Enter the No. instruction: 5

Performance Measure is: 2.5.

Result:

Thus, The 4-stage pipeline c program executed Successfully.

2-stage pipeline:

Exp-31 python

Aim: To design & program for 2-stage pipeline.

Algorithm:

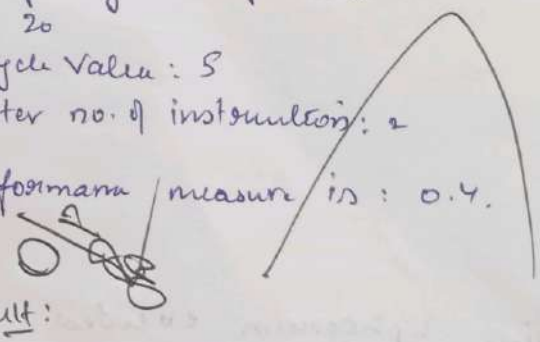
1.y Get the input values from the users.
2.y stage as Instruction execution.
3.y Fetch the data from the data table.
4.y Store in Separate Memory Address.
5.y Execute the Fetch data from the store location
6.y Perform the operation as per needs.
7.y Stop the execution.

Sample input:

Enter 1st value: 4
Enter 2nd value: 5
Enter the option:
1. Addition
2. Substraction
3. Multiplication
4. Division
Enter the choice

Sample output:

Enter the choice: 3
Performing Multiplication
20
Cycle value: 5
Enter no. of instruction: 2
Performance measure is: 0.4.



Result:

Thus, the python program for 2-stage pipeline as executed sucessfully.