CS2310 : Final Assignment

Gajendra - 1

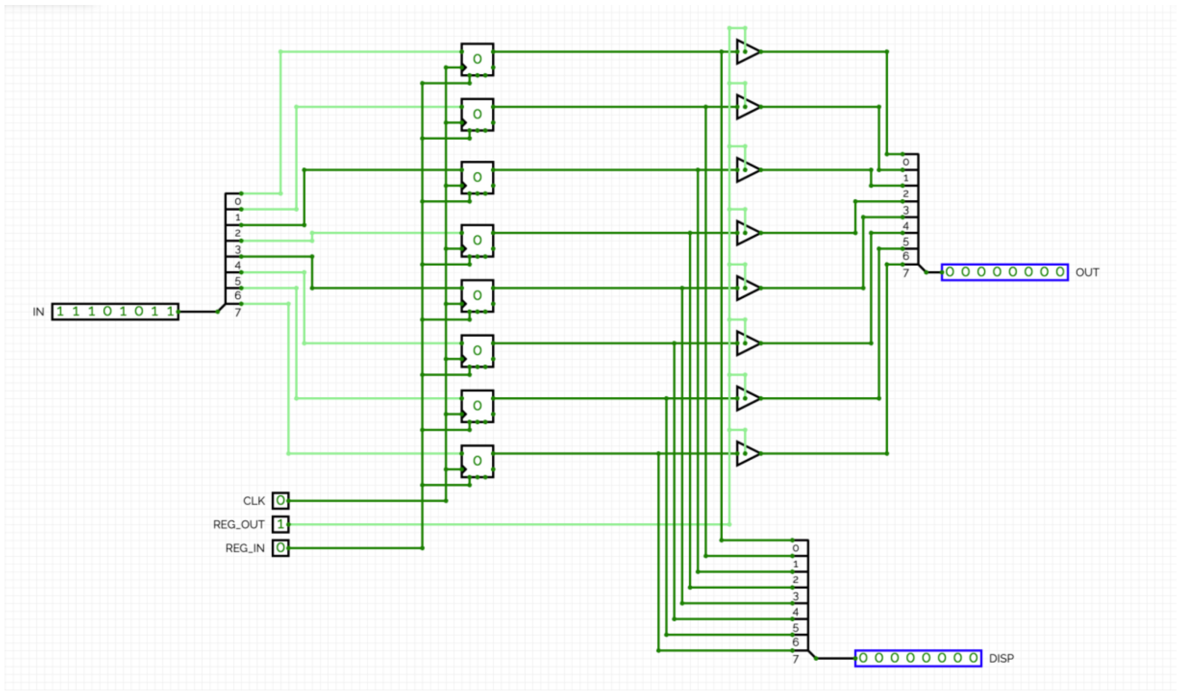
Group Number 35

CS22B020 && CS22B021

1. Overall architecture of CPU core (ARCH_GAJENDRA)

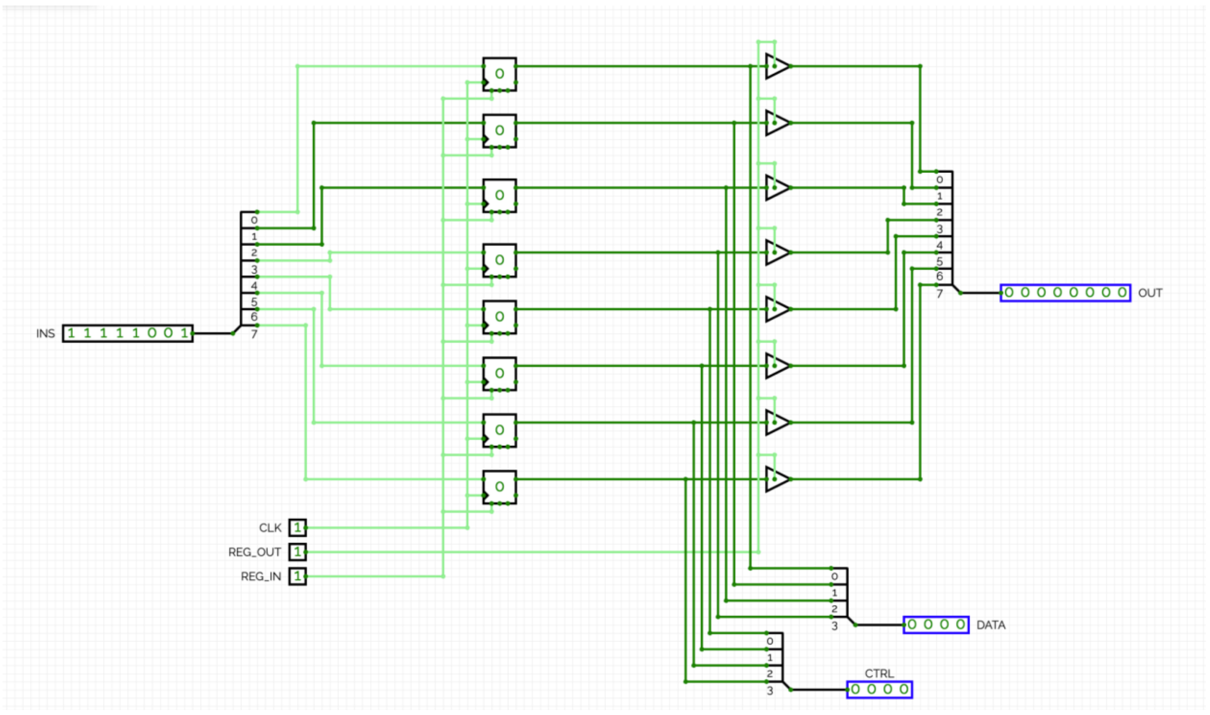
a) General CPU Register (reg_cpu_8):

It stores a binary word using flip-flops and tri-states.



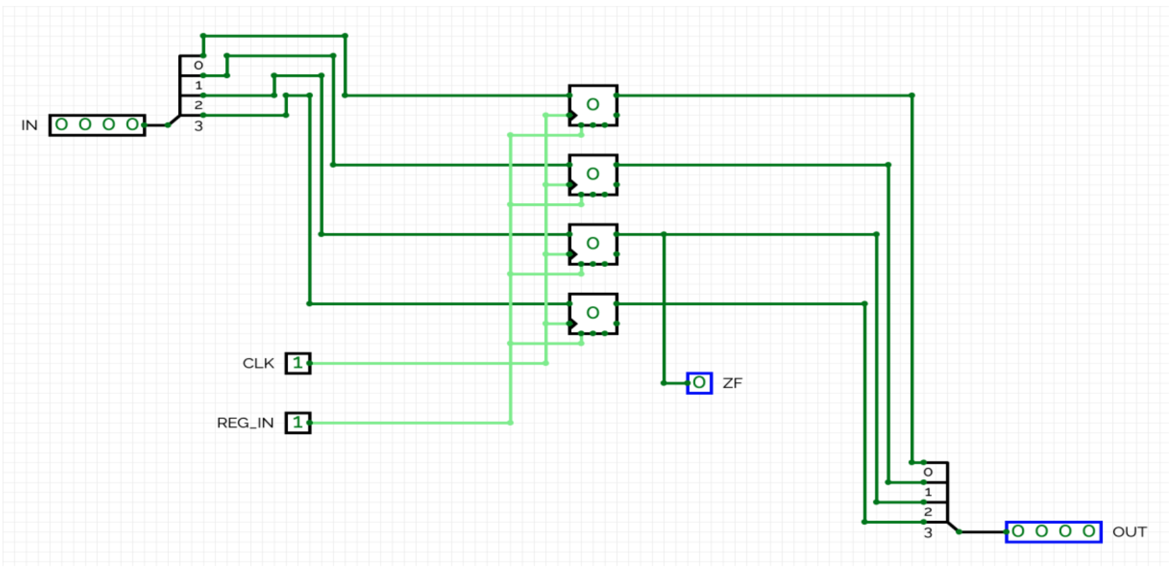
b) Instruction Register (reg_IR):

It is a part of the control unit that fetches the instruction from memory using read operation. It outputs the stored memory location contents to the common bus.



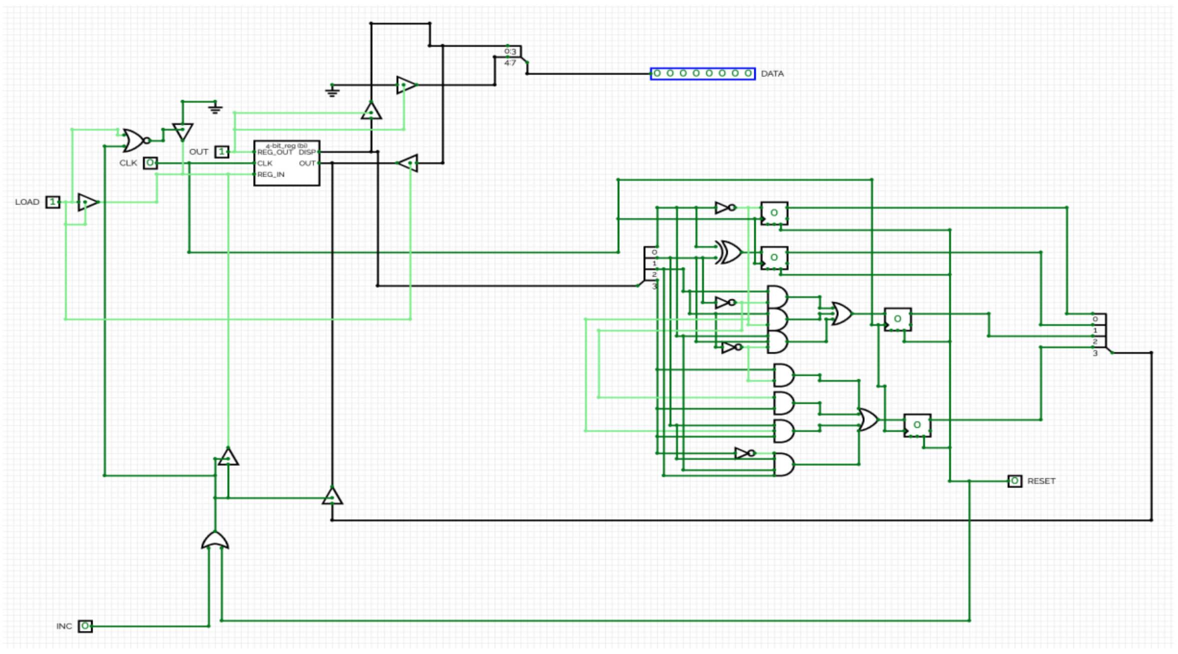
c) Memory Address Register (reg_MAR):

The address of the next instruction which is in the program counter is latched into this Memory address register (MAR). A bit later, MAR applies this 4-bit address to the ROM.



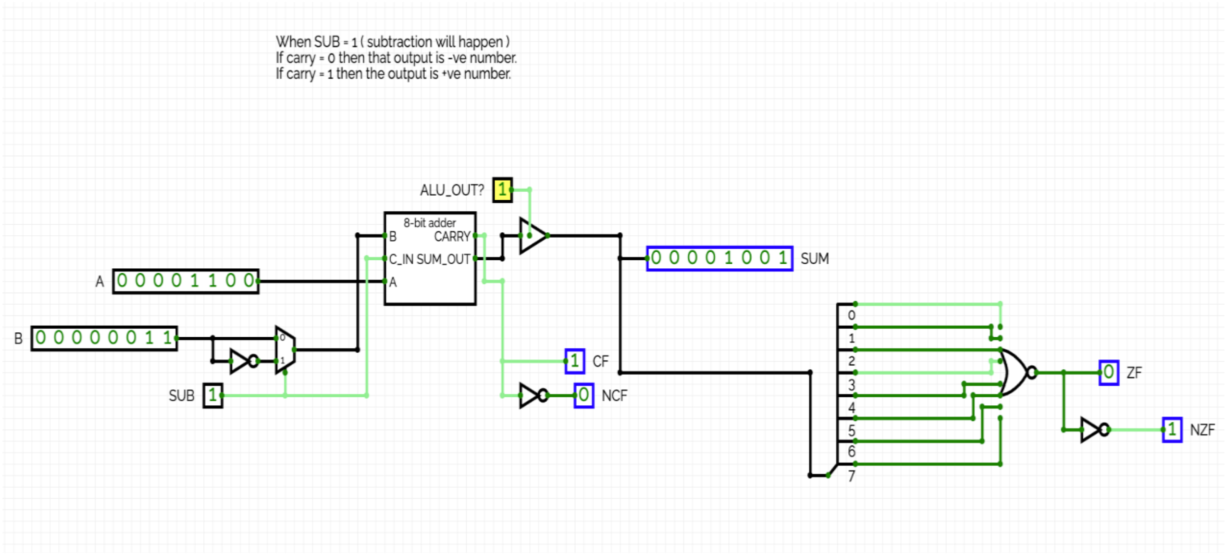
d) Program Counter (Counter_PC):

It will send the address of the next instruction to be fetched and executed to the memory.



e) Arithmetic and Logic Unit (ALU):

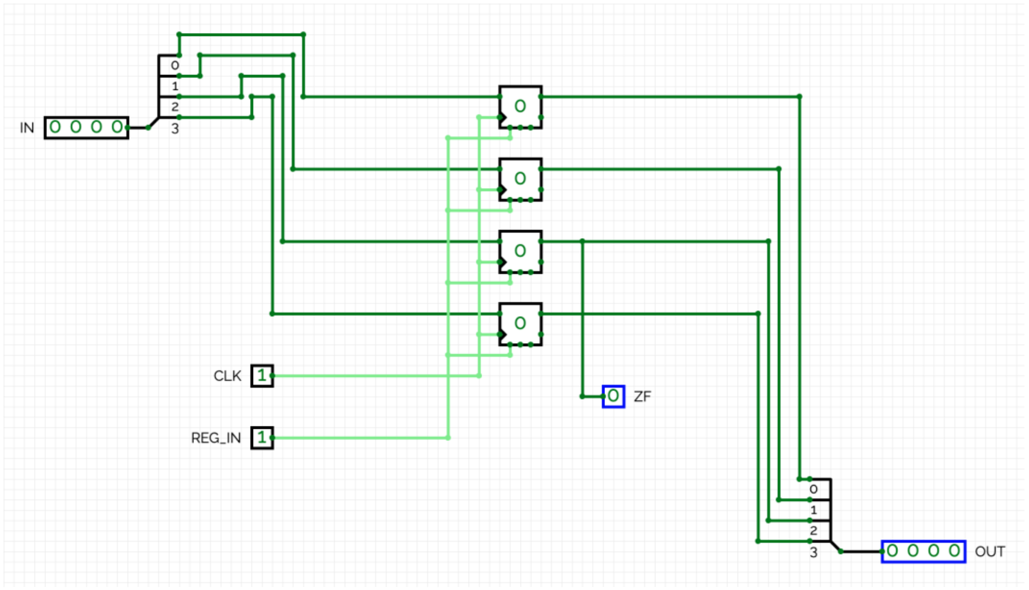
It represents the computational heart of the processor. This ALU performs addition/subtraction taking the data in Registers A and B as input arguments based on the given instruction.



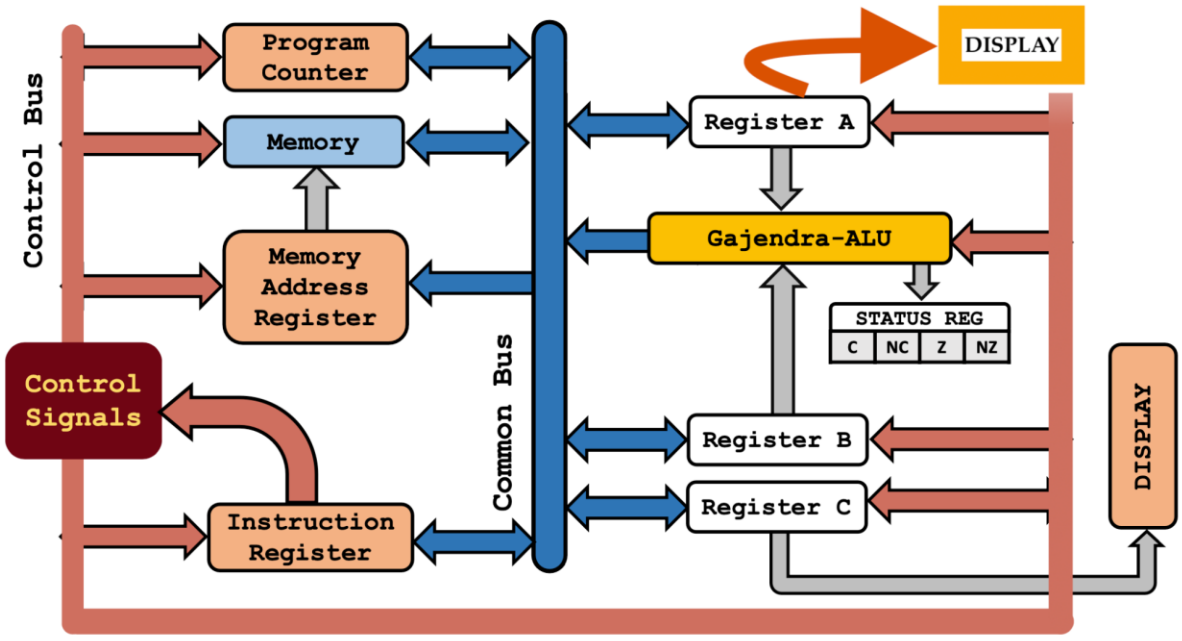
When SUB = 1 (subtraction will happen)
If carry = 0 then that output is -ve number.
If carry = 1 then the output is +ve number.
ALU_OUT? 1
8-bit adder
B
CARRY
C_IN SUM_OUT
A
A 0 0 0 0 1 1 0 0
B 0 0 0 0 0 0 1 1
SUB 1
0 0 0 0 1 0 0 1 SUM
1 CF
0 NCF
0
1
2
3
4
5
6
7
0 ZF
1 NZF

f) Status Register (reg_status_8):

This is a set of binary flags which are useful to determine some aspects of the computation that are carried out in ALU. Zero flag tells us whether the value given out by ALU is zero or not.
ZF - Zero Flag, NZF - Not Zero Flag,
CF - Carry Flag, NCF - Not carry flag.



→ We used three general purpose registers namely A, B and C. Third register REG_C is used to swap. The display is connected to REG_A and REG_C. We can see the value in the corresponding registers and bus in the flag (names are given accordingly).



2. INSTRUCTION SET (IS)

ASSEMBLY	MACHINE CODE	
NOP	0000	0x0
LDA	0001	0x1
STA	0010	0x2
ADD	0011	0x3
SUB	0100	0x4
LDI	0101	0x5
JMP	0110	0x6
SWAP	0111	0x7
JNZ	1000	0x8
MOVAC	1001	0x9
MOVBA	1010	0xa
MOVCB	1011	0xb
MOVAB	1100	0xc
MOVCA	1101	0xd
MOVBC	1110	0xe
HLT	1111	0xf

- **NOP**

DESCRIPTION

No operation is performed.

Operation:

Only fetching.

Syntax:

NOP -

8-bit Opcode:

0000	-

- **LDA**

DESCRIPTION

Load the value at the given address to the accumulator (Register A).

Operation:

Reg_A ← Memory

Syntax:

LDA <address>

8-bit Opcode:

0001	-

- **STA**

DESCRIPTION

Storing the value in register A to the given address of Memory.

Operation:

Memory ← Reg_A

Syntax:

STA <address>

8-bit Opcode:

0010	**xxx**

- **ADD**

DESCRIPTION

Adding the value stored in the given address to the accumulator and finally storing it in the accumulator.

Operation:

Reg_A ← Reg_A + Reg_B

Syntax:

ADD <address>

8-bit Opcode:

0011	**xxxx**

- **SUB**

DESCRIPTION

Subtracting the value stored in the given address from the value in the accumulator and storing it in the accumulator.

Operation:

Reg_A ← Reg_A - Reg_B

Syntax:

SUB <address>

8-bit Opcode:

0100	**xxxx**

- **LDI**

DESCRIPTION

Load the given value to the accumulator.

Operation:

Reg_A ← <data>

Syntax:

LDI <data>

8-bit Opcode:

0101	**xxxx**

- **JMP**

DESCRIPTION

Jump to the instruction at the given address.

Operation:

PC ← <address>

Syntax:

JMP <address>

8-bit Opcode:

0110	**xxxx**

- **SWAP**

DESCRIPTION

Swapping the values stored in two registers.

Operation:

Reg_A ← Reg_C && Reg_C ← Reg_A

Syntax:

SWAP -

8-bit Opcode:

0111	**-**

- **JNZ**

DESCRIPTION

Jump to the instruction at the given address when the value in the accumulator is non-zero.

Operation:

PC ← <address>

Syntax:

JNZ <address>

8-bit Opcode:

1000	**xxxx**

- ***MOVAC***

DESCRIPTION

Move the value stored in Register A to Register C.

Operation:

Reg_C ← Reg_A

Syntax:

MOVAC -

8-bit Opcode:

1001	**-**

- ***MOVBA***

DESCRIPTION

Move the value stored in Register B to Register A.

Operation:

Reg_A ← Reg_B

Syntax:

MOVBA -

8-bit Opcode:

1010	**-**

- ***MOVCB***

DESCRIPTION

Move the value stored in Register C to Register B.

Operation:

Reg_B ← Reg_C

Syntax:

MOVCB -

8-bit Opcode:

1011	-

- ***MOVAB***

DESCRIPTION

Move the value stored in Register A to Register B.

Operation:

Reg_B ← Reg_A

Syntax:

MOVAB -

8-bit Opcode:

1100	-

- ***MOVCA***

DESCRIPTION

Move the value stored in Register C to Register A.

Operation:

Reg_A ← Reg_C

Syntax:

MOVCA -

8-bit Opcode:

1101	-

- ***MOVBC***

DESCRIPTION

Move the value stored in Register B to Register C.

Operation:

Reg_C ← Reg_B

Syntax:

MOVBC -

8-bit Opcode:

1110	-

- ***HLT***

DESCRIPTION

Halt the program counter and ring counter.

Operation:

None

Syntax:

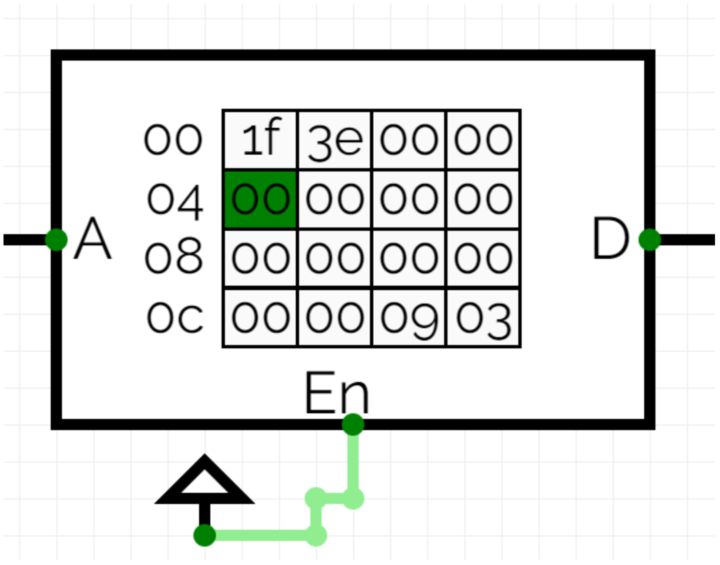
HLT -

8-bit Opcode:

1111	-

3. Few assembly programs implemented using our IS:

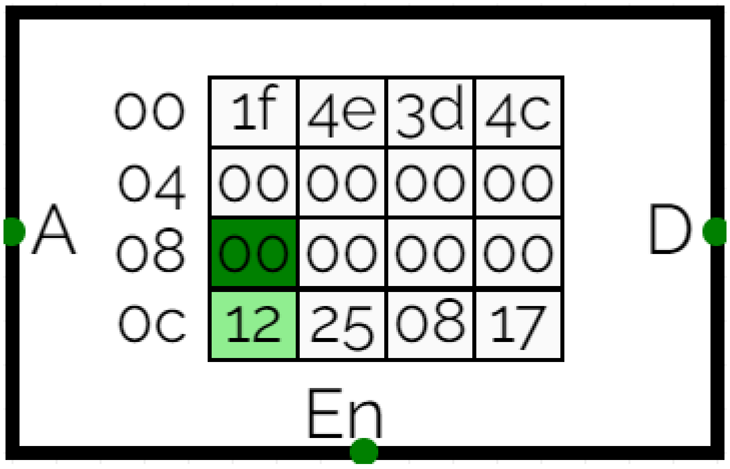
1. Adding two numbers and displaying the result:



Working:

- ➢ First PC points to 0x0 in ROM. 1f (LDA <Address = f>) loads the accumulator with the value contained in 0xf i.e., with 03 here.
- ➢ Next PC increments and points to 0x1 in ROM
  3e (ADD <Address = e>) adds the value at 0xe i.e., 09 here, to the value stored in the accumulator and stores the final value in the accumulator. Hence the final value stored in the accumulator is 0c.
- ➢ Next PC increments and points to 0x2. 00(NOP) No operation is performed.

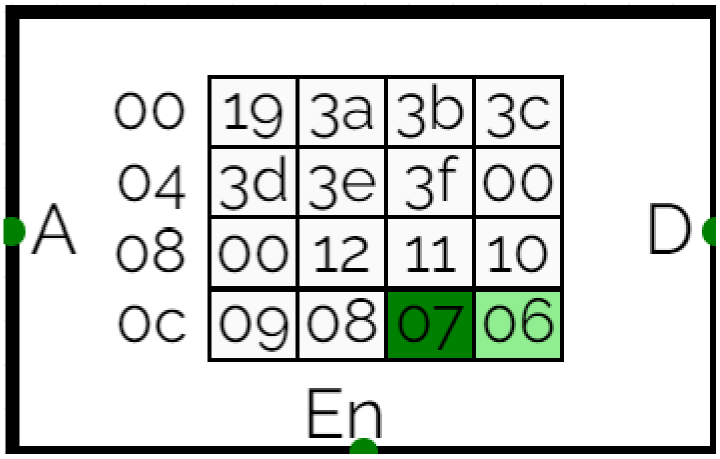
2. Adding and subtracting four numbers in some combination:



Working:

- ➢ First PC points to 0x0 in ROM. 1f (LDA <Address = f>) loads the accumulator with the value contained in 0xf i.e., with 17 here.
- ➢ Next PC increments and points to 0x1 in ROM
  4e (SUB <Address = e>) subtracts the value at 0xe i.e., 08 here, from the value stored in the accumulator and stores it in the accumulator. Hence the value stored in the accumulator is 09.
- ➢ Next PC increments and points to 0x2 in ROM.
  3d (ADD <Address = d>) adds the value at 0xd i.e., 25 here, to the value stored in the accumulator and stores it in the accumulator. Hence the value stored in the accumulator is 34.
- ➢ Next PC increments and points to 0x3 in ROM.
  4c (SUB <Address = c>) subtracts the value at 0xc i.e., 12 here, from the value stored in the accumulator and stores it in the accumulator. Hence the value stored in the accumulator is 22.
- ➢ Next PC increments and points to 0x4. 00(NOP) No operation is performed.

3. Adding numbers from a starting address to an ending address and displaying the result:



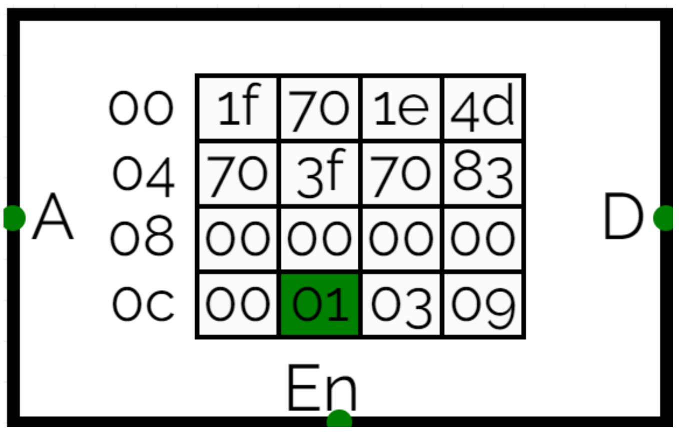
Working:

- ➢ First PC points to 0x0 in ROM. 19 (LDA <Address = 9>) loads the accumulator with the value contained in 0x9 i.e., with 12 here.
- ➢ Next PC increments and points to 0x1 in ROM

3a (ADD <Address = a>) adds the value at 0xa i.e., 11 here, to the value stored in the accumulator and stores it in the accumulator. Hence the value stored in the accumulator is 23.

- ➢ Next PC increments and points to 0x2 in ROM.
  3b (ADD <Address = b>) adds the value at 0xb i.e., 10 here, to the value stored in the accumulator and stores it in the accumulator. Hence the value stored in the accumulator is 33.
- ➢ Next PC increments and points to 0x3 in ROM.
  3c (ADD <Address = c>) adds the value at 0xc i.e., 09 here, to the value stored in the accumulator and stores it in the accumulator. Hence the value stored in the accumulator is 3c.
- ➢ Next PC increments and points to 0x4 in ROM
  3d (ADD <Address = d>) adds the value at 0xd i.e., 08 here, to the value stored in the accumulator and stores it in the accumulator. Hence the value stored in the accumulator is 44.
- ➢ Next PC increments and points to 0x5 in ROM.
  3e (ADD <Address = e>) adds the value at 0xe i.e., 07 here, to the value stored in the accumulator and stores it in the accumulator. Hence the value stored in the accumulator is 4b.
- ➢ Next PC increments and points to 0x6 in ROM.
  3f (ADD <Address = f>) adds the value at 0xf i.e., 06 here, to the value stored in the accumulator and stores it in the accumulator. Hence the value stored in the accumulator is 51.
- ➢ Next PC increments and points to 0x7. 00(NOP) No operation is performed.

4. A multiplication routine using repeated addition



00	1f	70	1e	4d
04	70	3f	70	83
08	00	00	00	00
0c	00	01	03	09

Working:

- ➢ First PC points to 0x0 in ROM. 1f (LDA <Address = f>) loads the accumulator with the value contained in 0xf i.e., with 09 here.
- ➢ Next PC increments and points to 0x1 in ROM. 70(SWAP) swaps values stored in Registers A and C. So final values stored in Register A is 00 and Register C is 09.
- ➢ Next PC increments and points to 0x2 in ROM. 1e (LDA <Address = e>) loads the accumulator with the value contained in 0xe i.e., with 03 here.
- ➢ Next PC increments and points to 0x3 in ROM. 4d (SUB <Address = d>) subtracts the value at address 0xd i.e., 01 from the value stored in the accumulator and stores back in the accumulator. So final value stored in the accumulator is 02.
- ➢ Next PC increments and points to 0x4 in ROM. 70(SWAP) swaps values stored in Registers A and C. So final values stored in Register A is 09 and Register C is 02.
- ➢ Next PC increments and points to 0x5 in ROM. 3f (ADD <Address = f>) adds the value at address 0xf i.e., 09 to the value stored in the accumulator and stores back in the accumulator. So final value stored in the accumulator is 12.
- ➢ Next PC increments and points to 0x6 in ROM. 70(SWAP) swaps values stored in Registers A and C. So final values stored in Register A is 02 and Register C is 12.
- ➢ Next PC increments and points to 0x7 in ROM. 83(JNZ <Address = 3>) stores the instruction at address 0x3 in PC. When value in the accumulator is non-zero, the PC points to the stored address i.e., 0x3 here. This acts as a loop.
- ➢ And finally, after loop stops, the final value i.e., multiplication of two numbers stored in 0xe and 0xf here, is finally stored in Register C.

4. Microinstructions and Controller Logic Design

OUR CONTROL WORD (16-BIT)

FLAG	PC	PC	PC	MEM	MEM	IR	IR	MAR	RA	RA	RB	RB	RC	RC	ALU
HLT	INC	O	L	IN	O	IN	O	IN	IN	O	IN	O	IN	O	O

- **NOP**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
0	T2
0	T3
0	T4

- **LDA**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<IR_OUT | 1<<MAR_IN	T2
1<<MEM_OUT | 1<<REG_A_IN	T3
0	T4

- **STA**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<IR_OUT | 1<<MAR_IN	T2
1<<RA_OUT | 1<<MEM_IN	T3
0	T4

- **ADD**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<IR_OUT | 1<<MAR_IN	T2
1<<MEM_OUT | 1<<REG_B_IN	T3
1<<ALU_OUT | 1<<REG_A_IN	T4

- **SUB**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<IR_OUT | 1<<MAR_IN	T2
1<<MEM_OUT | 1<<REG_B_IN	T3
1<<ALU_OUT | 1<<REG_A_IN	T4

- **LDI**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<IR_OUT | 1<<REG_A_IN	T2
0	T3
0	T4

- **JMP**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<IR_OUT | 1<<PC_LOAD	T2
0	T3
0	T4

- **SWAP**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<REG_A_OUT | 1<<REG_B_IN	T2
1<<REG_C_OUT | 1<<REG_A_IN	T3
1<<REG_B_OUT | 1<<REG_C_IN	T4

- **JNZ**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<IR_OUT | 1<<PC_LOAD	T2
0	T3
0	T4

- **MOVAC**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<REG_A_OUT | 1<<REG_C_IN	T2
0	T3
0	T4

- **MOVBA**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<REG_B_OUT | 1<<REG_A_IN	T2
0	T3
0	T4

- **MOVCB**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<REG_C_OUT | 1<<REG_B_IN	T2
0	T3
0	T4

- **MOVAB**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<REG_C_OUT | 1<<REG_B_IN	T2
0	T3
0	T4

- **MOVCA**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<REG_C_OUT | 1<<REG_B_IN	T2
0	T3
0	T4

- **MOVBC**

1<<PC_OUT | 1<<MAR_IN	T0
1<<PC_INC | 1<<MEM_OUT | 1<<IR_IN	T1
1<<REG_C_OUT | 1<<REG_B_IN	T2
0	T3
0	T4

- **HLT**

1<<MAR_IN	T0
1<<MEM_OUT | 1<<IR_IN	T1
1<<FLAG	T2
0	T3
0	T4