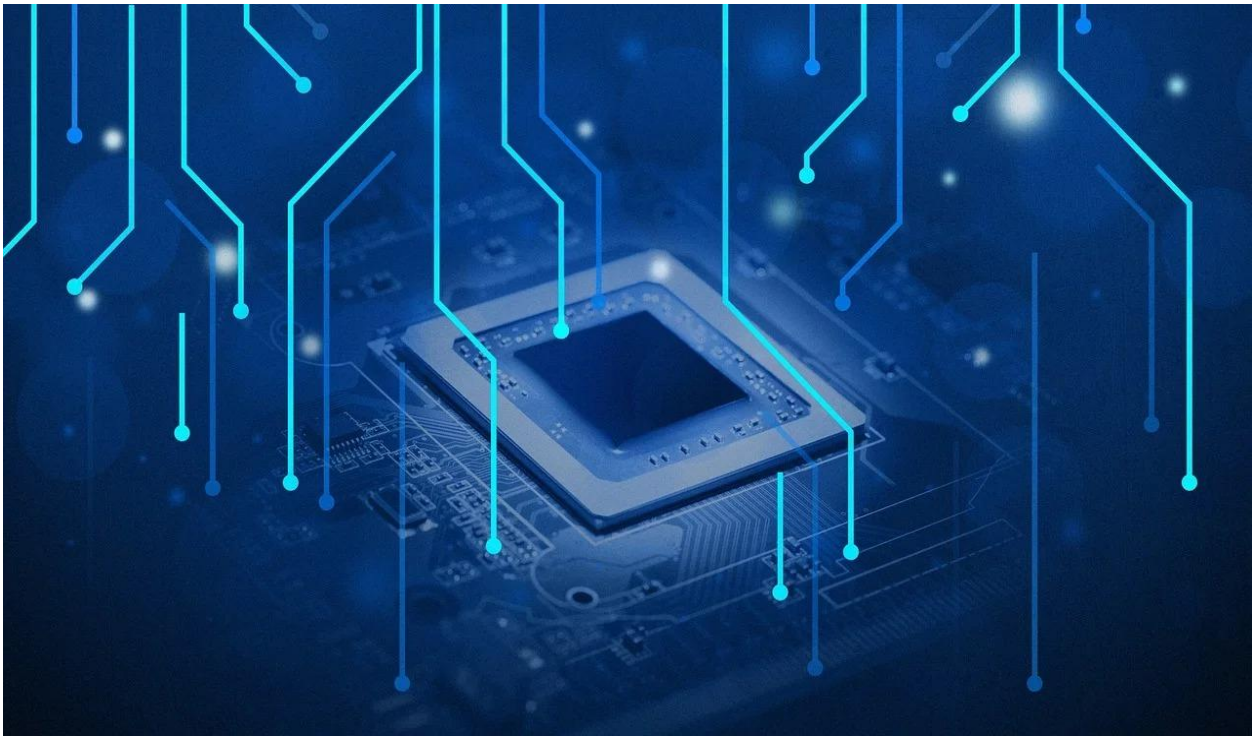
Digital Logic And Design

# FINAL REPORT (PHASE 4)

## Group Name: Abstraction



# Table Of Contents

## Group Members:

Lead: Muhammad Ibrahim Ayoubi 26269

Co-Lead: Ikhlas Ahmed Khan 27096

Abaan Noor 24911

Izma Khan 26926

Sameed Ahmad 26956

Itbaan Safwan 26197

Muhammad Hamza Zaman 27162

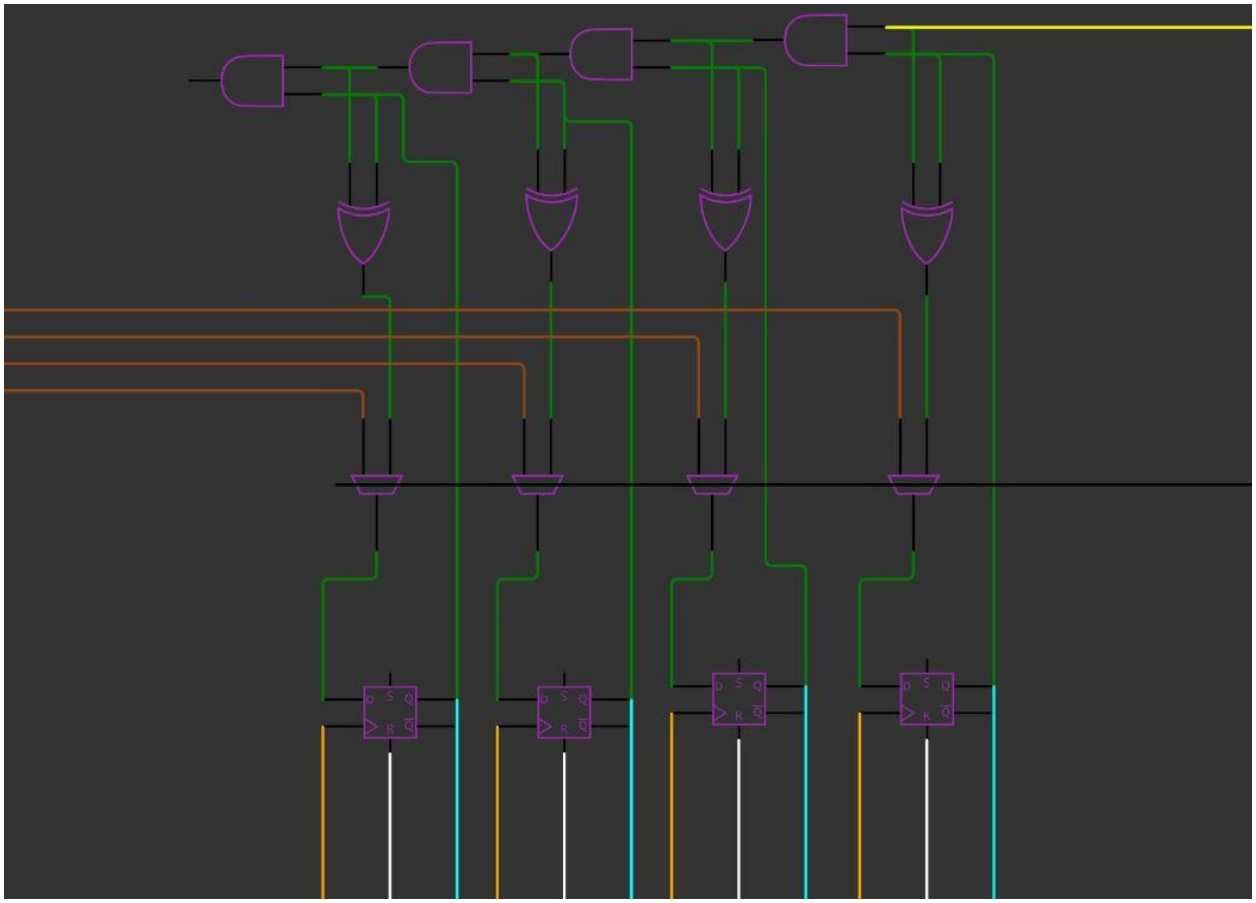
Muhammad Bilal Adnan 27151

# Objectives:

- To learn about sap-1 architecture and its different components.
- To learn how each individual component interfaces with one another to perform a set of instructions.
- To learn the integration/coding for the microcontroller, how it acts as our control unit in managing the inputs and outputs from its pins.
- Learn how data is stored, retrieved, and manipulated within the system.

# Program Counter:

The Program Counter, as the name suggests, is responsible for counting and storing the address of the next instruction in the program sequence. It is a register that holds the memory address of the instruction to be fetched and executed next. The Program Counter is crucial for the sequential execution of instructions in a program. Now as you can see in the circuit below our 4 bit program counter has an enable signal coming directly from the controller itself and additionally a mux which helps us in the operation of jumping from one instruction to any other in the RAM. Additionally, The output of the PC is sent directly to the microcontroller to be readed and interpreted.



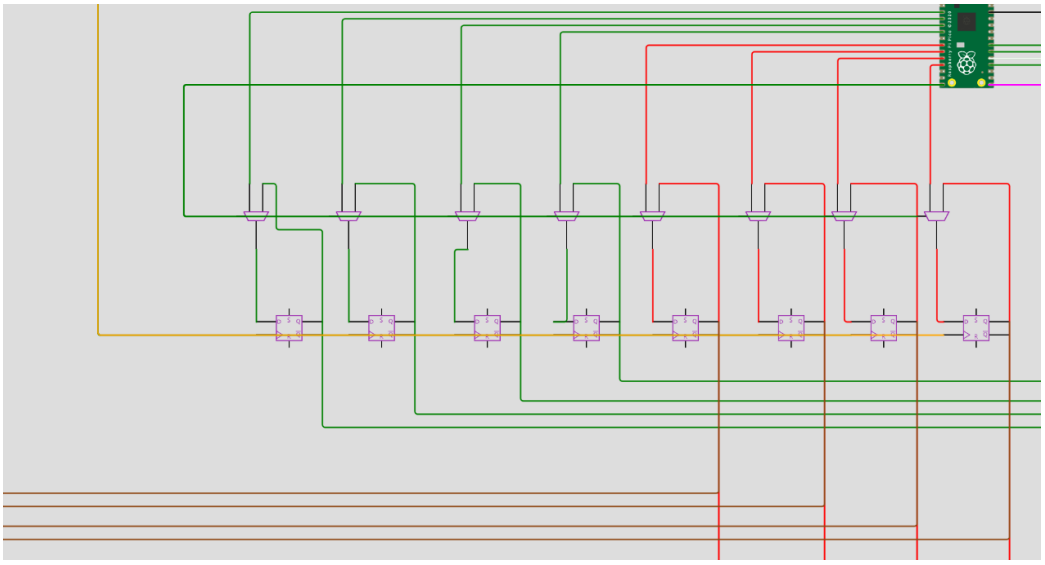
# Instruction- Register:

Next in line we have the Instruction register which upon receiving the counter value, gets the desired instruction from the controller. Note that this instruction is of 8 bits which is split into 2 parts by our IR; Number one the 4 bit opcode and Number two, the 4 bit data.

| Operation | Description | OPCODE |
|---|---|---|
| Load | Load data from controller to the accumulator | 0000 |
| Add | Add data from accumulator to value in from controller | 0001 |
| Subtract | Subtract data from accumulator to value in from controller | 0010 |
| Halt | Stop processing | 1111 |

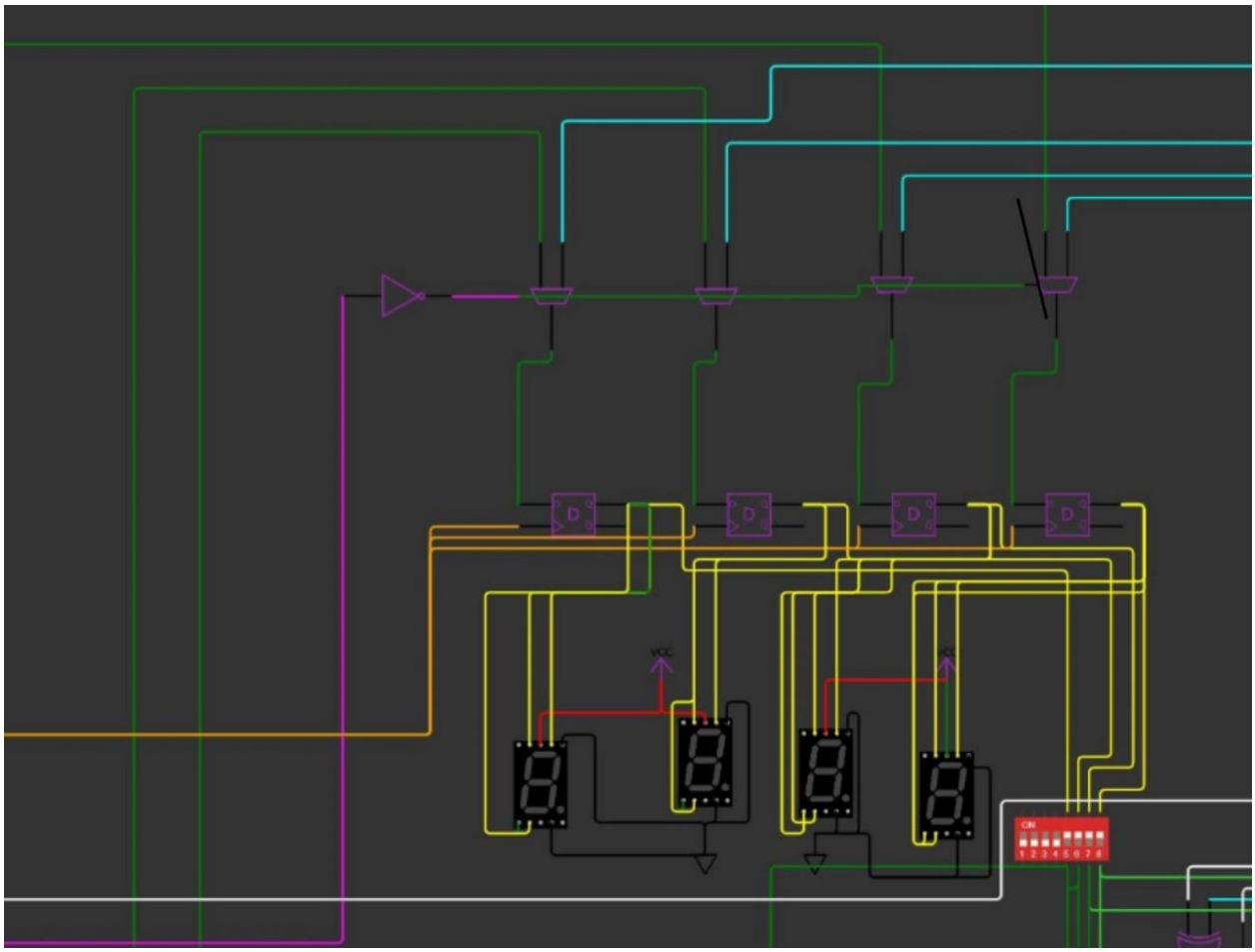
These are the following opcodes which we have used in our program;
**E.g INSTRUCTION: 0000 0010 means load data 0010 into the accumulator.**

The opcodes are sent directly back to our microcontroller to be recognized and to generate appropriate control signals for the remaining components of our sap-1.



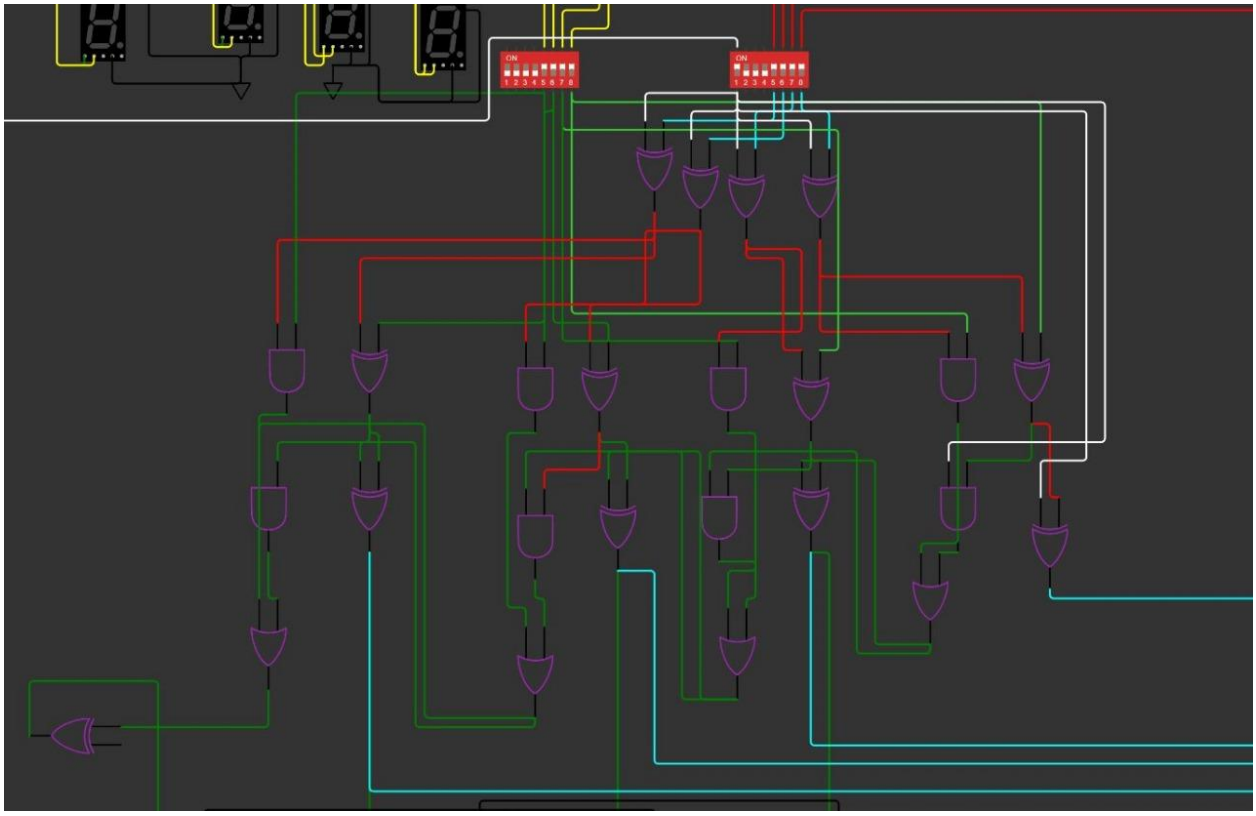
# Accumulator:

Now the accumulator is just a simple 4 bit register that is just responsible for storing the data either directly from the IR (LOAD OPERATION), or from the result calculated in the ALU(ADD/SUB OPERATION). It has a mux attached to its incoming flip flop inputs whose control signal determines whether to take input directly from IR or from the ALU result.



# ALU:

Our ALU is capable of doing 2 operations Addition or subtraction which is determined by the control signal (white signal), which if 0 performs addition and if 1 so subtraction. The first input of the ALU is always the output of the accumulator. The second input of the ALU gets activated on the addition/subtraction instructions which comes directly from the data bits from the instruction register through demuxes. The output of the ALU goes directly into the accumulator. Its circuit is as follows;



# WHAT WORKS?:

Now that's a really good question so for that we will be going through a quick demonstration for a single instruction execution cycle.

1. Suppose your pc increments and now points to instruction 5 (0101)
2. This address goes into the microcontroller and it fetches the 5th instruction from its pre-coded RAM.
3. Once it fetches that instruction, it sends that 8 bit instruction into the IR which receives and breaks it into 2 halves of 4 bits.....The first part mainly being the opcode, while the second half consisting of the data.
4. The opcode is sent back to the controller for interpretation, Suppose if the instruction is of LOAD (0000), then a control signal is generated in 2 parts where the 1st part of it allows the data to pass from the demux from the IR towards the accumulator, while the 2nd part allows the select line of the accumulator mux to enable data being loaded into the accumulator.
5. Whereas in the ADD (0001) OR SUB (0010) instruction, the control signal allows the data coming out of the IR from the demux to pass on towards the 2nd input of ALU. (**Note** that the 1st input of ALU is always from the accumulator) while also this same control signal also directs the mux of the accumulator to load data from the result calculated in ALU.
6. Finally the output of the current value in the accumulator is always displayed in four 7 segment LEDS in binary form.
7. Consequently for a jump instruction the pc updates its values from the IR data and then that same instruction is executed.
8. For the halt instruction, the program halts at its current position.

So basically in our SAP-1 we are able to perform these tasks in each of these components;

**Program Counter (PC):**

- Successful sequencing of instructions in the program.
- Accurate reception and processing of instruction addresses.

**Instruction Register (IR):**

- Efficient reception and storage of instructions.
- Accurate decoding and execution of instructions.

**ALU:**

- Precise execution of ADD/SUB instructions.
- Integration with the system for cohesive computation.
- Efficient sending and receiving of data in 4 bits.
- If larger bit subtracted by smaller bit then normal answer in output
- If smaller bit subtracted by larger bit then 2's complement answer in output

**Accumulator:**

- Controlled modification and manipulation of its state.
- Effective coordination with other components.
- Storing and displaying immediate results in the 7 segment LEDs.

# BASIC WORKING FUNCTIONALITIES:

Similar to ours, these enduring commands are a constant presence in SAP-1.

- **LOAD** OPERATION
- **ADD** OPERATION
- **HALT** OPERATION

- # INNOVATIONS:

  ★ Firstly We have added the functionality of **JUMP counter** in our sap-1 which simply means that the data within a respective instruction, now becomes the new address for the counter.

  ★ Secondly, for the ALU we have added the functionality of **SUBTRACTION**.

  ★ Additionally we have also added 4 LEDs of **7 segment** to **display** the output of the **accumulator**.

# WHAT WE TRIED OUR BEST BUT COULDN'T DO 😪:

Being really honest and straightforward we did everything which we mentioned as part of our inclusions, from making individual components to combining them and integrating them to perform perfectly. A minor issue which occurred during our halt operation is that if we halt the program at any state so if the preceding state is a load operation then the data last loaded is preserved but **if that preceding operation is an add/sub instruction then that output does not get preserved.**

Apart from this we have checked all the other operations to be working completely fine without any glitches or so.

# Closing Remarks:

In our SAP-1 project, we learned how the Program Counter, Instruction Register, Accumulator, and Arithmetic Logic Unit work together in a coordinated dance. Troubleshooting real issues sharpened our problem-solving skills, and exploring trade-offs deepened our understanding of design choices. It wasn't just about circuits; it was about teamwork, practical problem-solving, and the thoughtful decisions that make a system tick.

Thank you.

# User Manual:

Under this section you will be completely guided how to run our SAP and perform its basic operations that we just discussed in the previous sections.

- So when you open our SAP from the github link or the link gaven; https://wokwi.com/projects/385196534766239745
- You will see the left most side with the code including a pre hard-coded RAM;

```
#RAM predefined instructions
# LOAD 0001 D3 D2 D1 D0 THE 4 BITS OF DATA TO BE LOADED into the acc
# ADD 0001 D3 D2 D1 D0 THE 4 BITS OF DATA TO BE ADDED from val in acc
# SUB 0010 D3 D2 D1 D0 THE 4 BITS OF DATA TO BE SUBTRACTED from val in acc
# JUMP 1010 D3 D2 D1 D0 THE 4 BITS OF NEW PC ADDRESS RANGE (0-15) IN BINARY
# HALT 1111 D3 D2 D1 D0 THE 4 BITS OF NEW PC ADDRESS RANGE (0-15) IN BINARY
```

```
ram = [
    [0, 0, 0, 0, 1, 1, 1, 0],  # i=0 load 14
    [0, 0, 0, 1, 0, 0, 0, 1],  # i=1 # add 1
    [0, 0, 0, 0, 0, 1, 1, 0],  # i=2 load 6
    [1, 1, 1, 1, 0, 0, 0, 1],  # i=3 HALT
    [0, 0, 0, 0, 0, 0, 1, 0],  # i=4
    [0, 0, 0, 0, 0, 0, 1, 1],  # i=5
    [0, 0, 0, 0, 0, 0, 0, 0],  # i=6
    [0, 0, 0, 0, 0, 0, 0, 0],  # i=7
    [0, 0, 0, 0, 0, 0, 0, 0],  # i=8
    [0, 0, 0, 0, 0, 0, 0, 0],  # i=9
    [0, 0, 0, 0, 0, 0, 0, 0],  # i=10
    [0, 0, 0, 0, 0, 0, 0, 0],  # i=11
    [0, 0, 0, 0, 0, 0, 1, 1],  # i=12
    [0, 0, 0, 0, 0, 1, 0, 0],  # i=13
    [0, 0, 0, 0, 0, 0, 1, 1],  # i=14 #load 3
    [0, 0, 0, 1, 0, 0, 1, 0]   # i=15 #add 2 ans 5 0101
```

- So the idea is that when you run the program the instructions will run in a sequence starting from i=0. So if you want to add a 1st instruction so you replace your instruction at the line i=0, similarly to add a 2nd instruction add at line i=1 etc.
- Now you certainly know all about our opcodes so suppose you want to load 3 as 1st instruction you will replace the line (i=0; 1st instruction) with **0,0,0,0,0,0,1,1** where the **blue part indicates load(0000)**, while the **red part is the value 3 in binary form (0010)**.
- Following this, if you want to perform the add operation (e.g add 3), then you can simply write an instruction, **0,0,0,1,0,0,1,1** where the **green part indicates add(0001)**, while the **red part is the value 3 in binary form (0011)**.
- Next if you want to perform the Subtract operation(e.g subtract 2), then you can simply write an instruction, **0,0,1,0,0,0,1,0** where the **pink part indicates sub(0010)**, while the **red part is the value 2 in binary form (0010)**.

  NOTE: FOR ADDING OR SUBTRACTING 2 VALUES YOU WILL FIRST NEED TO LOAD THE FIRST VALUE AND THEN PERFORM INSTRUCTION FOR ADD/SUB, E.G

  To perform **2 + 3**: first load 2 **0,0,0,0,0,0,1,0**, then add 3 **0,0,0,1,0,0,1,1** then you will have a **result of 5 (0101)** in the accumulator.

  To **5 - 3**: first load 5 **0,0,0,0,0,1,0,1** then sub 3 **0,0,1,0,0,0,1,1** then you will have a **result of 2 (0010)** in the accumulator.

- Consequently if you want to perform the JUMP instruction to jump from one pc address to another(e.g jump to address 14 [1110] ) then you can write an instruction **1,0,1,0,1,1,1,0** where the **orange part indicates jump(0010)**, while the **blue part is the new value for the PC (14 1110) which will point to a specific instruction in the ram**. For example the instruction, **1,0,1,0,1,1,1,0** is the 2nd instruction **which states to jump to the 14th(1110) instruction in the ram** hence our sap will run the first 2 instructions and then jump to the 14th instruction ignoring all the instructions in between.
- Lastly, if u want the program to **HALT** write instruction **1,1,1,1,D3,D2,D1,D0,** (**NOTE**: The values of d3 d2 d1 d0 are ignored hence they could be anything for this instruction)