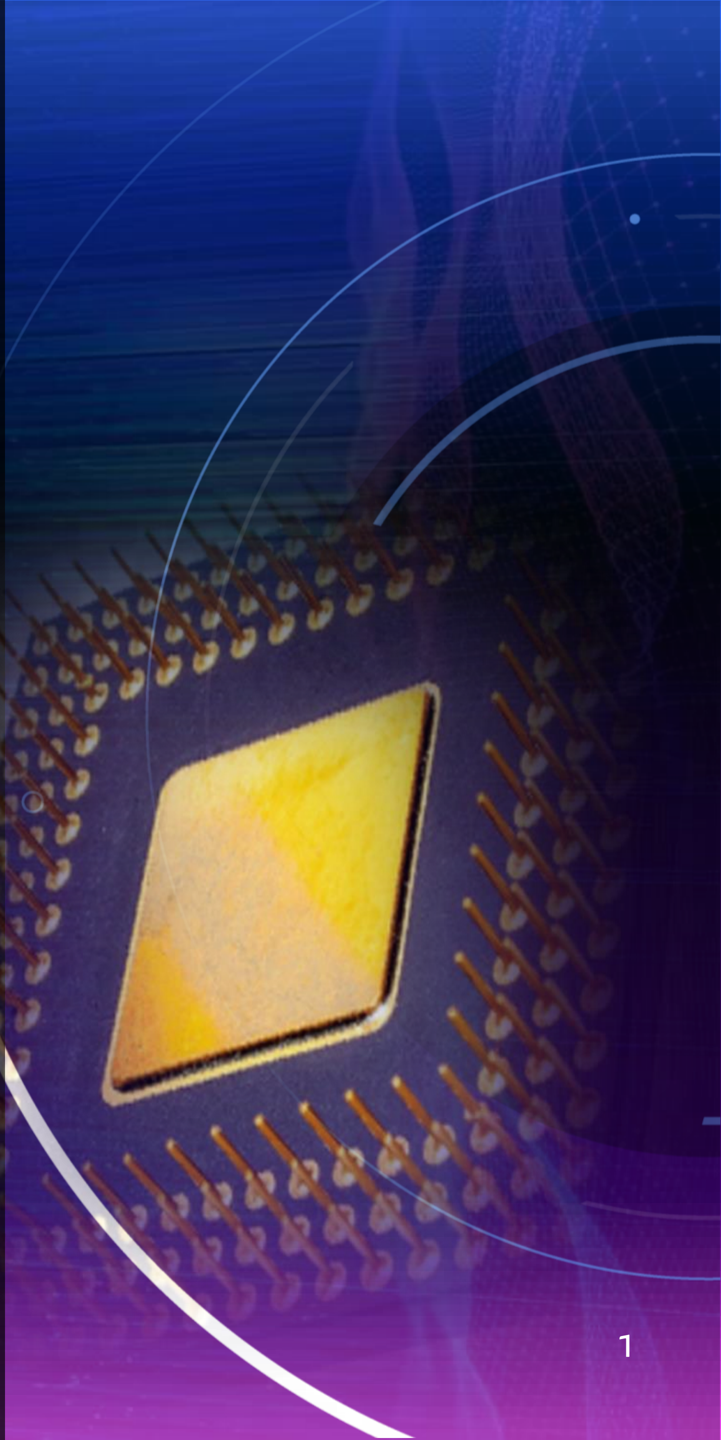
FBU CPU implementation

This Project is about creating and implementing a Fenerbahçe Central processing unit(CPU)

- You may ask what is a CPU??
- A CPU is the primary component of a computer that acts as its “control centre.” It is usually referred as the “central” or “main” processor, it is a complex set of electronic circuitry that runs the machine's operating system and apps.



GO NEXT

Overview

As described in the previous slide our CPU will almost do the same however Fenerbahce CPU will only contain three operations which are: Addition ,Subtraction ,Multiplication. To run those operations, we will obviously need other different instructions

Which our CPU will be able to run perfectly. The implementation of this CPU will be by the help simulator .

REMEMBER THE OPERATIONS ARE:

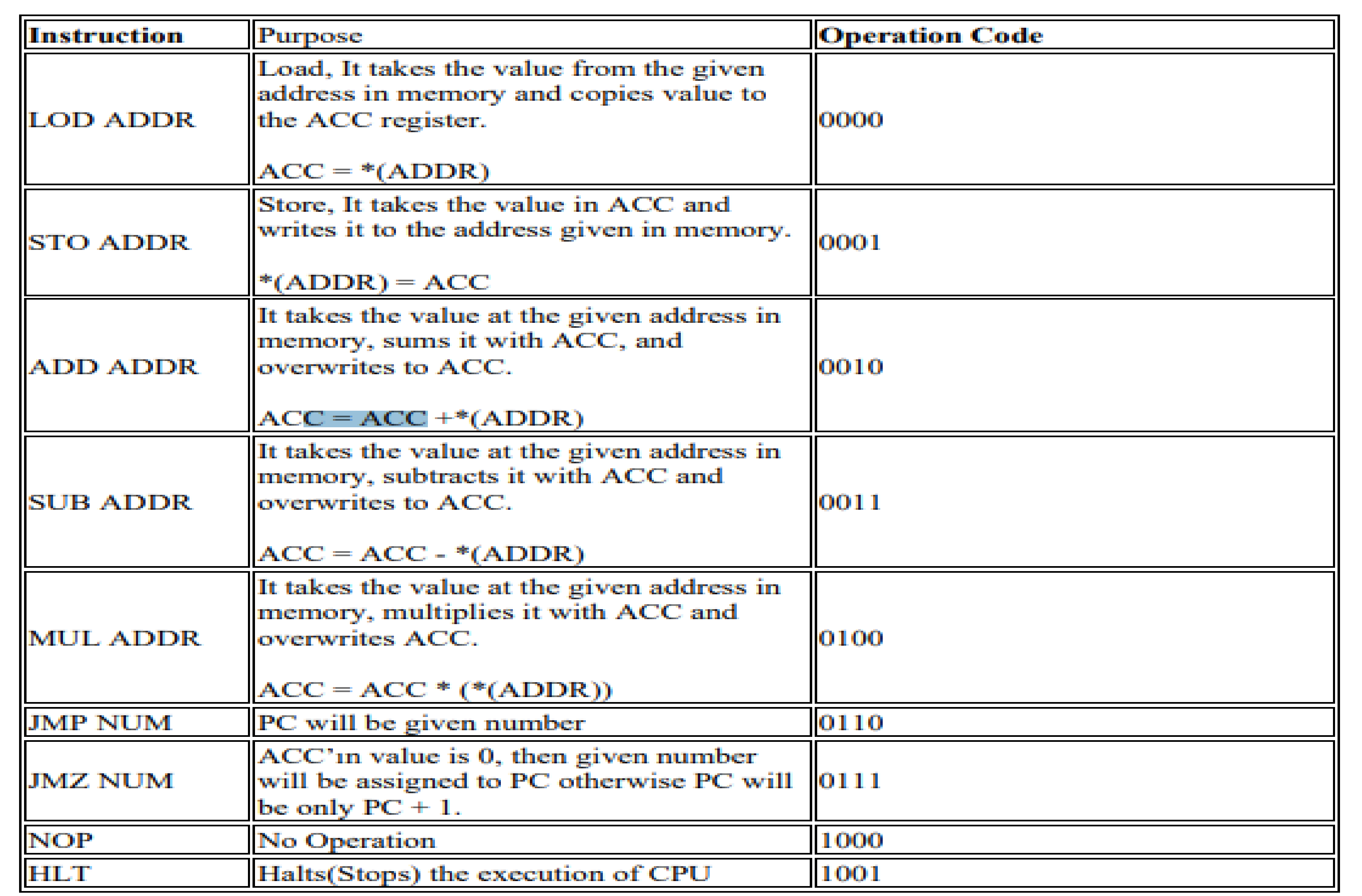
ADDITION

SUBSTRACTION

MULTIPLICATION

GO NEXT

OUR CPU instructions supported



Instruction	Purpose	Operation Code
LOD ADDR	Load, It takes the value from the given address in memory and copies value to the ACC register. ACC = *(ADDR)	0000
STO ADDR	Store, It takes the value in ACC and writes it to the address given in memory. *(ADDR) = ACC	0001
ADD ADDR	It takes the value at the given address in memory, sums it with ACC, and overwrites to ACC. ACC = ACC +*(ADDR)	0010
SUB ADDR	It takes the value at the given address in memory, subtracts it with ACC and overwrites to ACC. ACC = ACC - *(ADDR)	0011
MUL ADDR	It takes the value at the given address in memory, multiplies it with ACC and overwrites ACC. ACC = ACC * (*(ADDR))	0100
JMP NUM	PC will be given number	0110
JMZ NUM	ACC’ın value is 0, then given number will be assigned to PC otherwise PC will be only PC + 1.	0111
NOP	No Operation	1000
HLT	Halts(Stops) the execution of CPU	1001

Simulator architecture

- This is an online simulator

- FPGA:(Field-Programmable gate array) is an integrated circuits that provide customers the ability to reconfigure the hardware to meet specific use case requirements after the manufacturing process

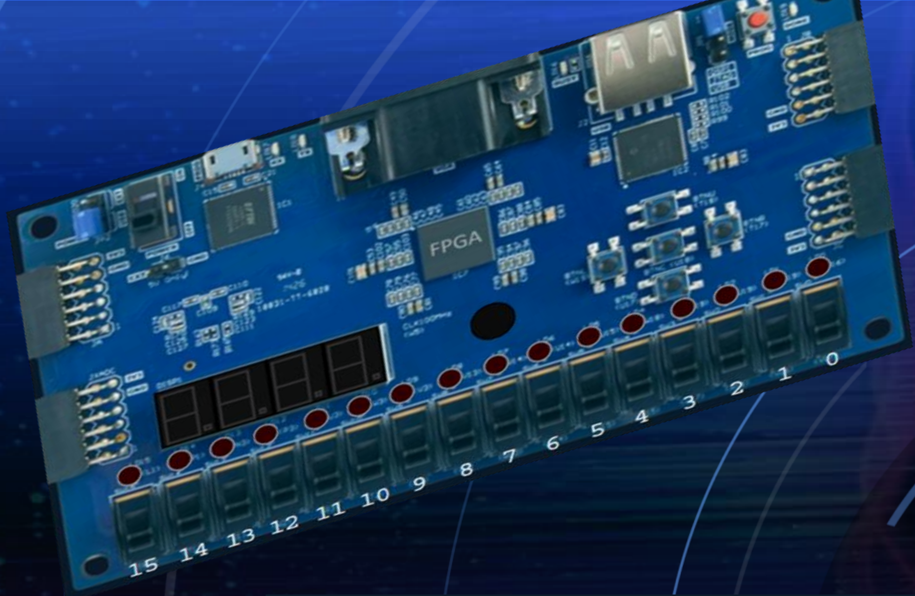
Our FPGA will be able to easily change its functionality after an instruction has been completed.

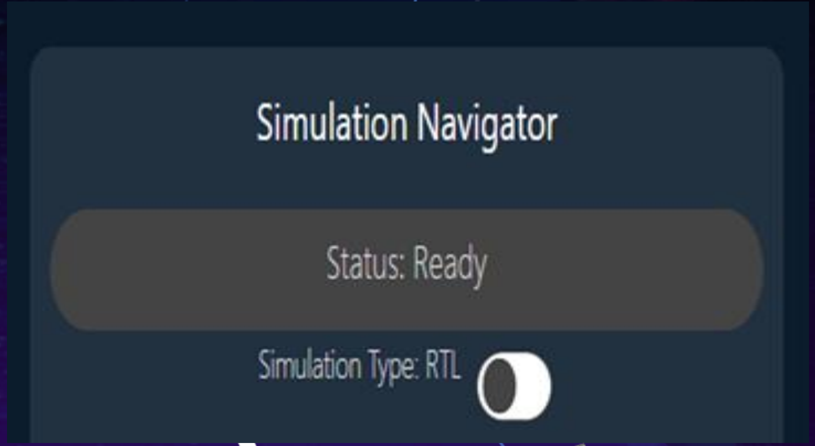
- Simulator Navigator: this part is basically used to run the code written in the coding shell

as you can see there are a red part where it is written: **Status Ready:** This shows us when the FGPA has completed the task

We also have **Simulation Type RTL:** This methos is recommended and it used to validate the correctness of digital IC designs.

GO NEXT

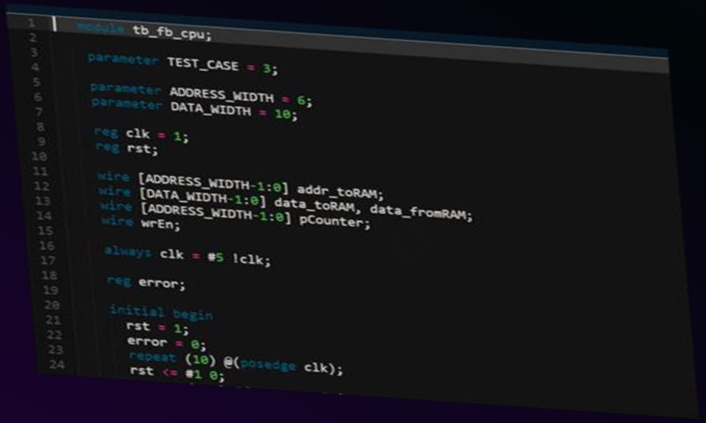



Simulation Navigator

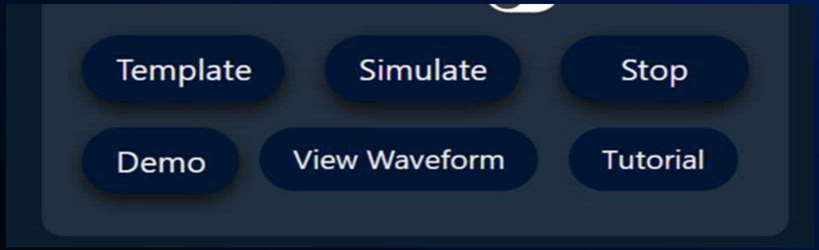
Status: Ready

Simulation Type: RTL

Simulator architecture



```
module tb_fb_cpu;

  parameter TEST_CASE = 3;

  parameter ADDRESS_WIDTH = 6;
  parameter DATA_WIDTH = 10;

  reg clk = 1;
  reg rst;

  wire [ADDRESS_WIDTH-1:0] addr_toRAM;
  wire [DATA_WIDTH-1:0] data_toRAM, data_fromRAM;
  wire [ADDRESS_WIDTH-1:0] pCounter;
  wire wrEn;

  always clk = #5 !clk;

  reg error;

  initial begin
    rst = 1;
    error = 0;
    repeat (10) @(posedge clk);
    rst <= #1 0;
```



❖ Shell Script: This is most useful for repetitive tasks that may be time consuming to execute by typing one line at a time. A few examples of applications shell scripts can be used for include: Automating the code compiling process. Running a program or creating a program environment.

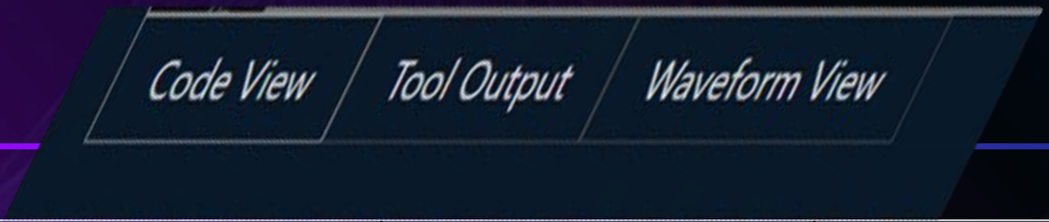
So, this part will be used to write our codes, every code that we will use ,will be written in this shell script

❖ As you can see we have the second picture having; Template, Simulate, Stop, Demo, View Waveform, Tutorial

- **Simulate**: basically means start running the code
- **Stop**: this means stop simulating
- **View waveform**: This will show us implicit signals such as; clocks, the enable chains, register update logic ,registers, and block diagram wires.
- **Template**: This is a form, Mold or pattern used as a guide to make something.

GO NEXT

Simulator architecture



❖ Code View: this will help us to view our code from the script shell a In case there are some changes that are needed to be done We will just click CODE VIEW	❖ Tool Output: This will help us to view the output of our code	❖ Waveform: This will show us implicit signals such as; clocks, the enable chains, register update logic ,registers, and block diagram wires.

Testing software used

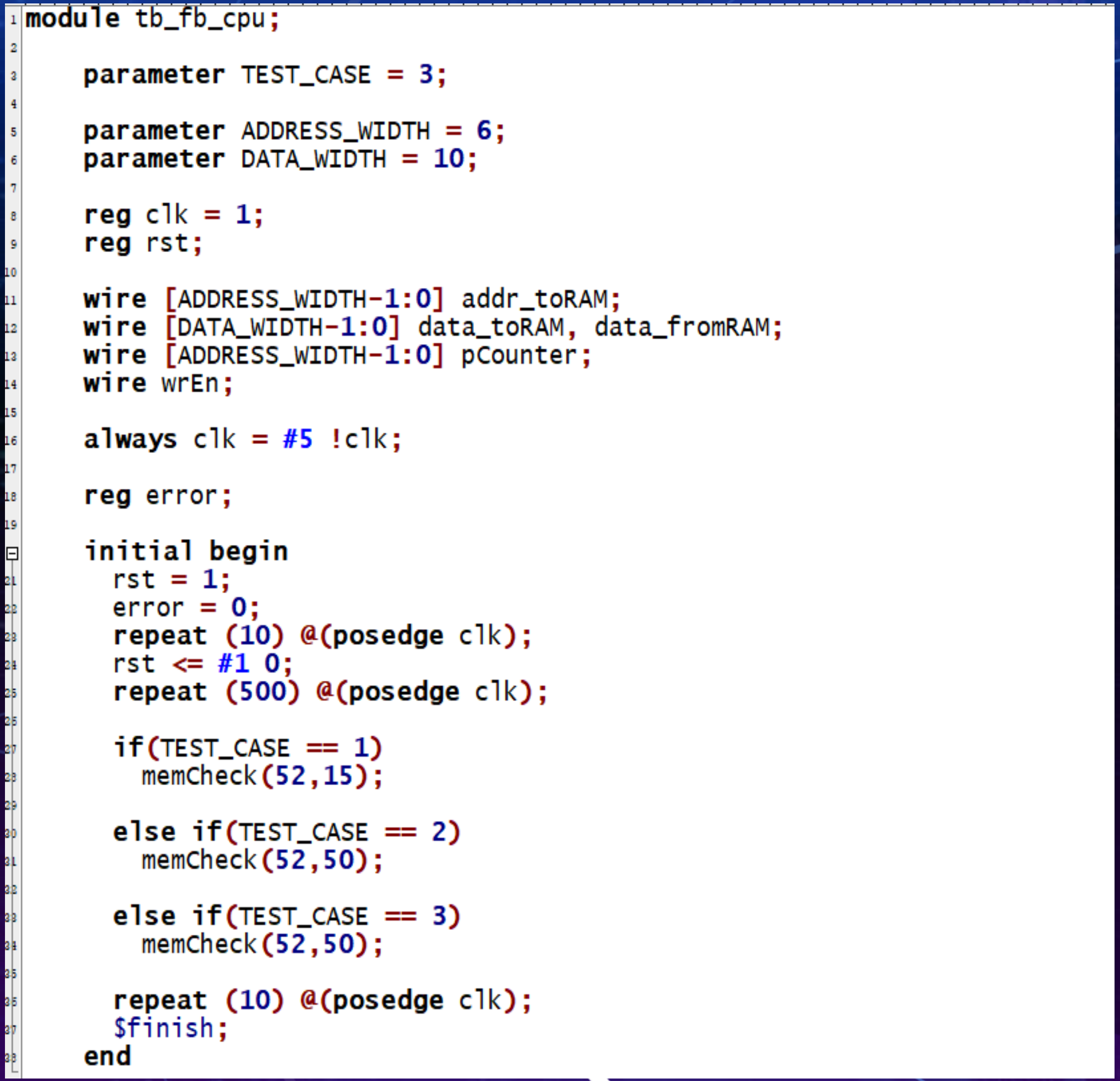
TO test our CPU ,we have used an online simulator that you can find on this link:

https://avionchip.com/virtualfpga//

This was designed by OUR FAVORITE TEACHER **Dr Vecdi Emre LEVENT**

In the previous slides we were able to describe shortly about this simulator

We used some lines of codes to test our CPU and we will share with you some.



```
module tb_fb_cpu;

   parameter TEST_CASE = 3;

   parameter ADDRESS_WIDTH = 6;
   parameter DATA_WIDTH = 10;

   reg clk = 1;
   reg rst;

   wire [ADDRESS_WIDTH-1:0] addr_toRAM;
   wire [DATA_WIDTH-1:0] data_toRAM, data_fromRAM;
   wire [ADDRESS_WIDTH-1:0] pCounter;
   wire wrEn;

   always clk = #5 !clk;

   reg error;

   initial begin
     rst = 1;
     error = 0;
     repeat (10) @(posedge clk);
     rst <= #1 0;
     repeat (500) @(posedge clk);

     if(TEST_CASE == 1)
       memCheck(52,15);

     else if(TEST_CASE == 2)
       memCheck(52,50);

     else if(TEST_CASE == 3)
       memCheck(52,50);

     repeat (10) @(posedge clk);
     $finish;
   end
```

Hint of the code used

```
reg [2:0] state;

always @(posedge clk) begin

 case (state)

STATE_0 : begin

MAR <= PC;

   state <= STATE_1;

  end

STATE_1 : begin


   IR <= MDR;

   PC <= PC + 1;

   state <= STATE_2;

  end
```

```
STATE_2 : begin

    case(IR[15:12])
     ADD: begin
           ACC <= ACC + RAM[IR[11:0]];
         end
     SUB: begin
           ACC <= ACC - RAM[IR[11:0]];
         end
    endcase
    state <= STATE_3;
  end
```

The state machine performs different operations based on the opcode stored in the most significant bits of IR[15:12].

If the opcode is ADD, it adds the content of the Memory location specified by the lower 12 bits of IR to the Accumulator (ACC).

If the opcode is SUB, it subtracts the content of the Memory location specified by the lower 12 bits of IR from the Accumulator.

```
STATE_3: begin

   RAM[IR[11:0]] <= ACC;
   state <= STATE_0;
  end

  default:
   state <= STATE_0;

 endcase
End
```

GO NEXT

STATE_3

The content of the Accumulator is stored into the Memory location specified by the lower 12 bits of IR.

The state transitions back to STATE_0.

default (Continued):

If none of the specific cases match, it resets the state to STATE_0.

This code represents a finite state machine for a simple processor that fetches instructions from memory, executes them, and stores results back in memory. The specific operations performed depend on the opcode of the instruction.

The provided Verilog code represents a basic processor's state machine.

The riggered on the positive edge of a clock signal, the processor progresses through different states:

In STATE_0, it copies the Program Counter (PC)
to the Memory Address Register (MAR);
in STATE_1, it loads the Memory Data Register (MDR) to the Instruction Register (IR) and increments the Program Counter;

In STATE_2,
it interprets the instruction in IR and performs either an addition or subtraction operation on the Accumulator (ACC) using data from RAM.

In STATE_3, it writes the result back to RAM.
The state machine resets to _0 by default. This snippet reflects a basic fetch-decode-execute cycle, typical in a microprocessor is a STATE

Results

- The whole experiment is about creating and implementing a CPU in Fenerbahçe Which will only run with three operators Addition, Subtraction and Multiplication.
- We have our Simulator which is a software tool that enables candidates to write and test their code in a simulated environment. It has different sections which enables it to run some of the listed are The FPGA, Simulator Navigator, simulation type She'll script, code view, tool output, waveform and the likes of them By using all these, after writing the code, The Code should be able to run with only Addition, multiplication and Subtraction without an error.

PROJECT TEAM CV MEMBERS

- AISHA MUHSIN ABBA->220304149

Email: aisha.abba@stu.fbu.edu.tr

Phone number:+90-5354889546

Department :Computer Engineering

- LILIA VANIELLA DUHARIRE->220304138

Email: lilia.duharire@stu.fbu.edu.tr

Phone number:+90-5055613013

Department :Computer Engineering

- WAREN MOUDOUMI MOUSTAFA KONATE -> 220304142

Phone number : +90 5456591889

Email: waren.konate@stu.fbu.edu.tr

- TAIWO DAMILOLA SANUSI -> 220304143

Email :taiwo.sanusi@stu.fbu.edu.tr

- Olanrewaju Maleek Bello->220304139

Email:maleek.bello@stu.fbu.edu.tr
Phone number: +905397918322
Department:computer Engineering

- Aminata Kone-> 220304144

Email:aminata.kone@stu.fbu.edu.tr
Phone number:+905349854394
Department:computer Engineering

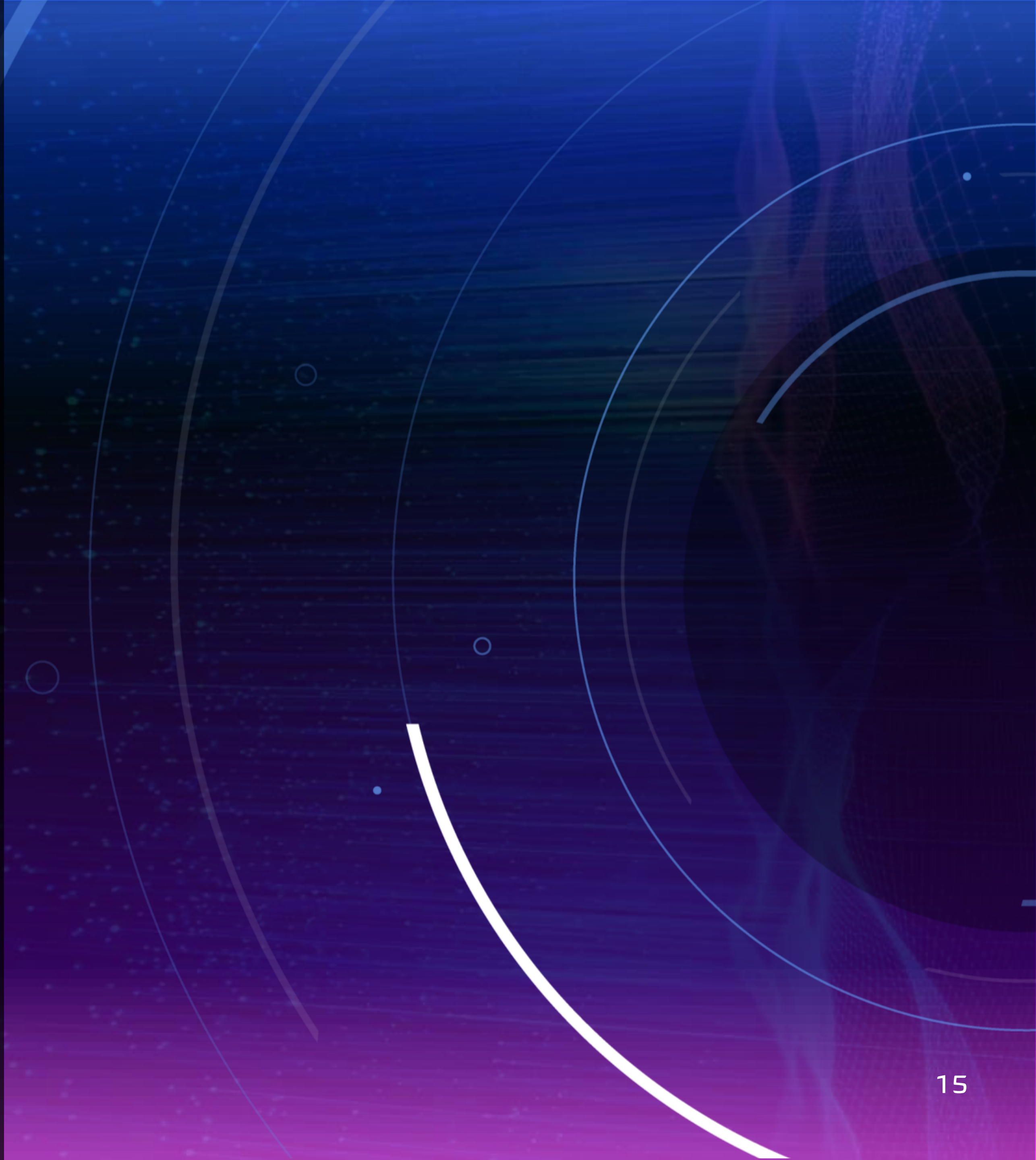
REFERENCES USED

- https://youtu.be/Z5JC9Ve1sfI?si=gUcnHcvYMOyhG_z7()
- https://www.youtube.com/watch?v=DYcLFHgVCn0
- https://www.youtube.com/watch?v=7d4ymjU9NqM
- https://www.oreilly.com/library/view/digital-logic-design/9780750645829/(Digital Logic Design, 4th Edition)

Please check out our YouTube channel, like and subscribe

https://youtu.be/Gp4W5Dg1oHU?si=GWpO0bKg4bRR3H4L

Thank you



See you☹**