# CPU Architecture

## LAB 3 - Digital System Design with VHDL


מגישים:

רועי שני 315995258

ינאי מייסון 206701591

# Top design layout

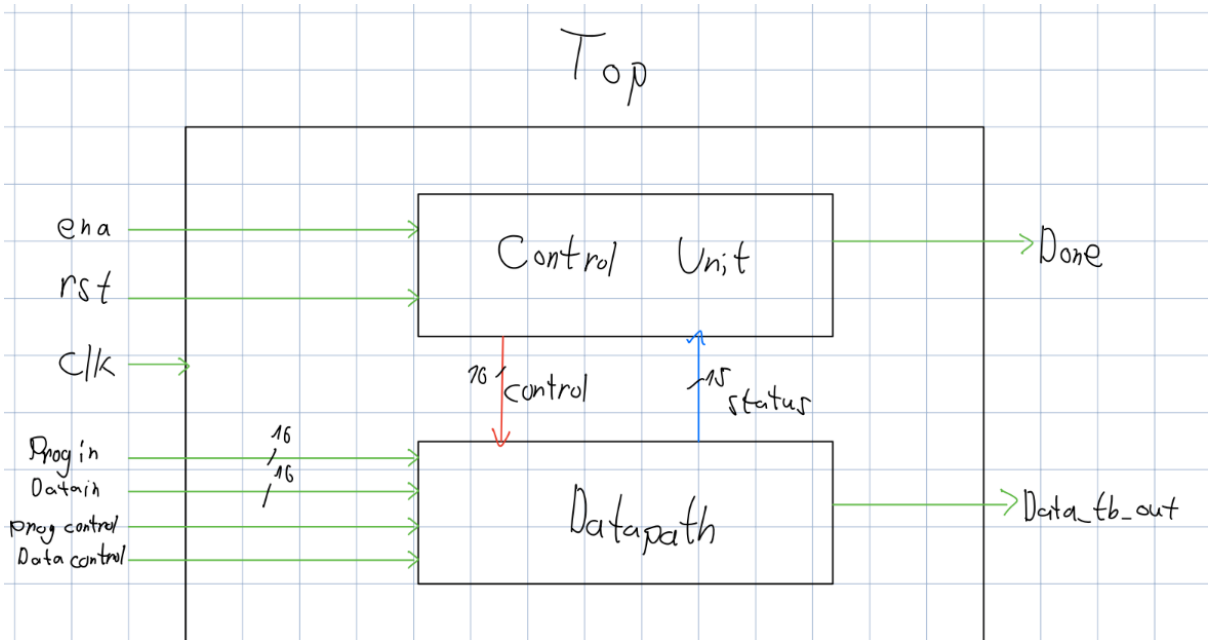


Fig 1 – Top design.

The top entity integrates both the Datapath and ControlUnit of the multi-cycle CPU. Its inputs and outputs are primarily used for control, memory interfacing, and testbench observation:

Inputs:

- clk, rst, ena: Standard clock, reset, and enable signals to control system timing and operation.
- tb_active: A testbench flag indicating when external memory access is active.
- DTCM_tb_addr_in, DTCM_tb_addr_out: Addresses used by the testbench to read from or write to the Data TCM (memory).
- DTCM_tb_in: Data input from the testbench to Data TCM.
- DTCM_tb_wr: Write enable signal from the testbench for writing to Data TCM.
- ITCM_tb_in, ITCM_tb_addr_in, ITCM_tb_wr: Instruction TCM inputs for loading instructions from the testbench.

Outputs:

- done: A signal asserted when the current instruction cycle is completed.
- DTCM_tb_out: Data output from the Data TCM back to the testbench.

# Control Unit

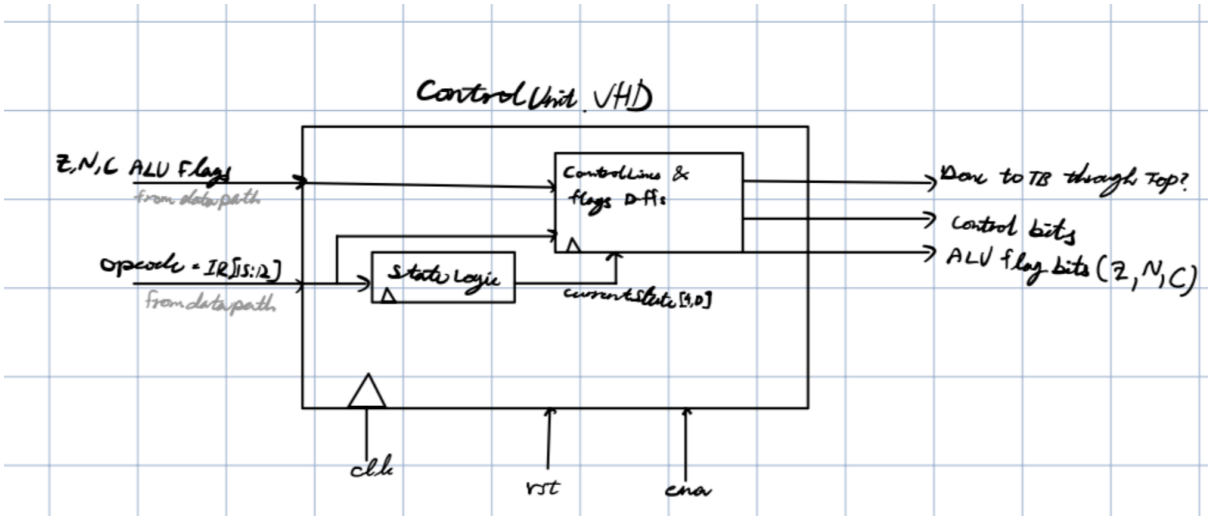


Fig 2 – Control unit design.

The ControlUnit is the top-level control module of a multi-cycle CPU. It connects two main components:
StateLogic: A finite state machine (FSM) generating control states.
ControlLines: A microcontroller that generates control signals based on the current state, opcode, and ALU flags.

**Inputs:**

- clk, rst, ena: Standard clock, reset, and enable inputs to control state progression and control signal generation.
- ALU_c, ALU_z, ALU_n: Status flags from the ALU indicating carry, zero, and negative conditions.
- opcode: 4-bit opcode fetched from the instruction register, used to determine operation type.

**Outputs:**

- done: Indicates when a multi-cycle instruction execution is completed.
- RF_addr_rd, RF_addr_wr: Addresses for reading from and writing to the register file.
- DTCM_wr: Enables write operation to Data TCM.
- DTCM_addr_sel, DTCM_addr_out, DTCM_addr_in: Address selection and routing control signals for the data memory.
- DTCM_out: Enables output from Data TCM to the bus.
- ALU_op: 3-bit signal specifying the ALU operation.
- Ain: Enables loading operand A into the ALU.
- RF_WregEn: Enables writing back to the register file.
- RF_out: Enables output from register file.
- IRin: Enables loading data into the instruction register.
- PCin: Enables updating the program counter.
- PCsel: Selects the source for the next PC value.
- Imm1_in, Imm2_in: Enables loading of immediate values.
- status_bits: 15-bit signal containing debug and status information, including ALU flags and current opcode/state.

# Datapath

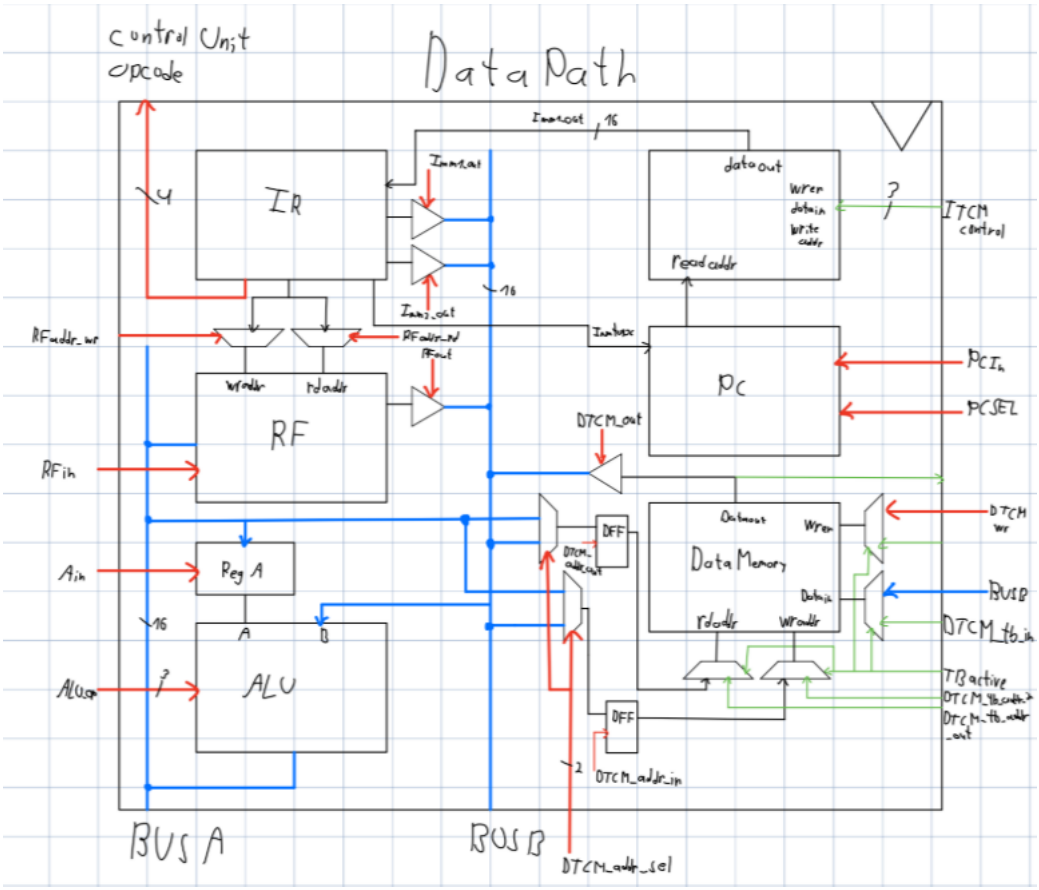


Fig 3 – Data path 2 bus design.

The Datapath is the central processing block of the CPU, coordinating the register file, ALU, instruction/data memory, and control-driven logic to execute instructions. It communicates with the Control Unit to receive control signals and with external testbenches for simulation and debugging.

Inputs:

**Clock & Reset:**

- clk: Main system clock.
- rst: Synchronous reset signal.
- ena: Enable signal to control data path activty.

**Control Signals from Control Unit:**

- DTCM_wr: Write enable for Data TCM (memory).
- DTCM_addr_sel: Selects the source for address muxing (bus A or B).
- DTCM_addr_out, DTCM_addr_in: Enable signals for loading memory address into read/write address registers.
- DTCM_out: Enables output from Data TCM.
- ALU_op: 3-bit ALU operation code (e.g., add, sub, etc.).
- Ain: Enable signal for storing ALU output in register A.
- RF_WregEn: Enables writing to the register file.
- RF_out: Enables reading from the register file to bus B.
- RF_addr_rd, RF_addr_wr: 2-bit read/write register addresses.
- IRin: Enable signal for loading the Instruction Register.
- PCin: Enable signal to load/update the Program Counter.

- PCsel: 2-bit selector for PC source (sequential, jump, etc.).
- Imm1_in, Imm2_in: Enable siginals for routing immediate operands to bus B.

**Testbench Inputs:**

- tb_active: When '1', testbench controls memory access (bypasses CPU).
- DTCM_tb_in: Testbench data input to Data TCM.
- DTCM_tb_wr: Write enable from testbench.
- DTCM_tb_addr_in, DTCM_tb_addr_out: Address lines for testbench memory access.
- ITCM_tb_in: Instruction data to be written into Instruction TCM.
- ITCM_tb_addr_in: Address for instruction memory.
- ITCM_tb_wr: Write enable for instruction memory from TB.

**Outputs**:

- alu_c, alu_z, alu_n: ALU status flags indicating:
- Carry, Zero, and Negative results respectively — used by the Control Unit for conditional logic.
- opcode: The 4-bit operation code extracted from the instruction — used by the Control Unit to determine the operation to control.
- DTCM_tb_out: Data output from Data TCM, forwarded to the testbench for observation.

# Simulation results

## Top Testbench

The example code 1 file was stored in the following locations:
./datapath_code/ITCMinit.txt
./datapath_code/DTCMinit.txt
./datapath_code/DTCMcontent.txt
The code performs the following operation in C:

```
int arr[14]={20,11,2,23,14,35,6,7,48,39,10,11,12,13}
int res;

void main(){

        if((arr[5] & 31) >= (arr[4] & 31))
                res=0;
        else
                res=1;

        while(1);

}
```

This translates in assembly into the following:

```
data segment:
arr dc16 20,11,2,23,14,35,6,7,48,39,10,11,12,13
res ds16 1

code segment:
ld  r1,4(r0)
ld  r2,5(r0)
mov r3,31
mov r4,1
mov r5,res
and r1,r1,r3
and r2,r2,r3
sub r6,r2,r1
jc  2
add r6,r4,r0
jmp 1
add r6,r0,r0
st  r6,0(r5)
done
nop
jmp -2
```

The obtained DTCM, ITCM and output obtained are the following:

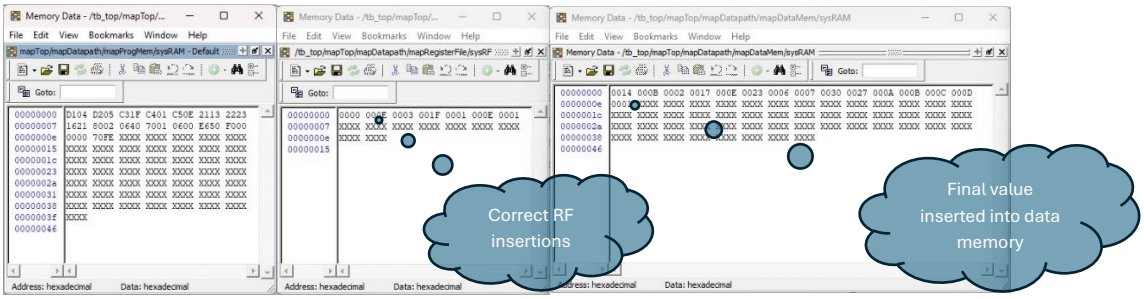


Fig 4 – ProgMem (left), RF (middle), DataMem (right).

As can be seen, the register files were updated to hold intermediate values as required by the program. The output file written from the data memory matches that expected from the output file provided, thereby indicating that the program indeed performed the desired calculations. This can also be seen by observing the data memory, were the correct values were indeed inserted into it.

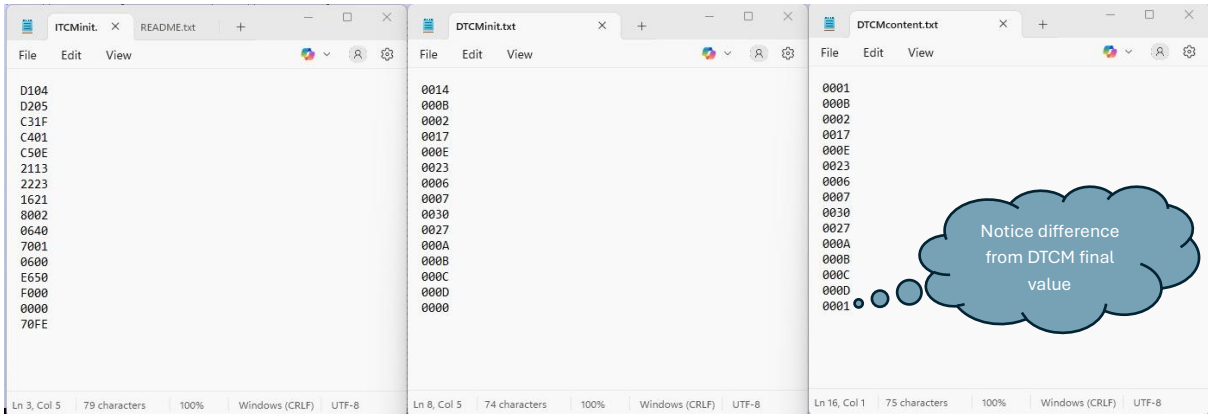


Fig 5 – output file as well as DTCM and ITCM files for code example 1.

# Datapath Testbench

The following test bench was executed on the datapath to ensure correct operation of the component. It generates a clock signal, initializes control signals, and manually stimulates the datapath through a sequence of events that mimic instruction fetch and execution. During simulation, it writes a test instruction to the instruction memory (ITCM), triggers program counter updates, loads the instruction into the instruction register (IR), and toggles key control signals such as Ain and ALU_op to observe ALU behavior. It also simulates a write operation to data memory (DTCM).

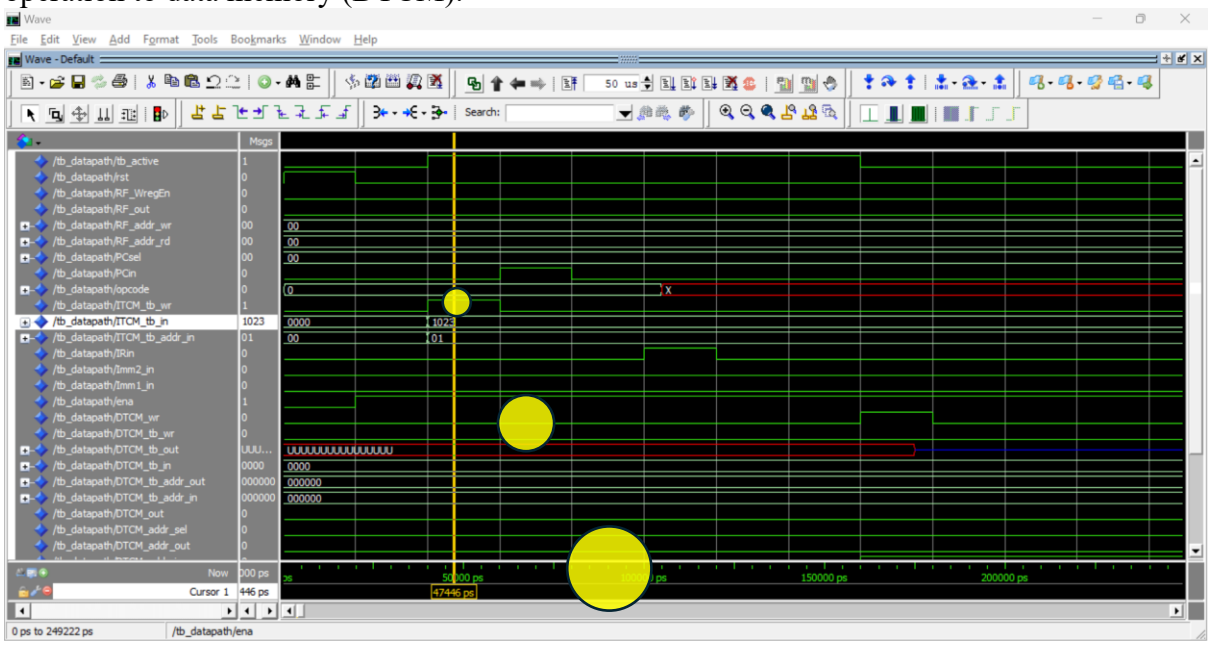


Fig 6 – wave simulation of datapath.vhd.

At 47,446 ps, the ena signal transitions from '0' to '1', start of active operation for the Datapath module. before, the system was at (rst = '1'), and the reset signal was set to '0' shortly before. The ena = '1' allows internal components such as the program counter (PC), instruction register (IR), and register file to begin responding to control signals and clk.

This point in time represents the transition from initialization to normal execution. It occurs just before the testbench begins writing an instruction to the program memory (ITCM_tb_wr = '1'), and shortly before the tb_active signal is set to '1'. thus, this event is the starting point for simulating instruction fetch and execution phases within the CPU datapath.

# Control Unit Test Bench

Initially, the testbench resets the system, followed by enabling the unit (ena <= '1') to allow operation. The core of the simulation consists of a loop that systematically applies all possible 4-bit opcode values (0000 to 1111) to the ControlUnit, with each opcode held for several clock cycles to allow internal state transitions. During each opcode test, the ALU condition flags (ALU_c, ALU_z, ALU_n) are manipulated to simulate different execution scenarios and observe the control unit's corresponding output signals. This ensures that the control unit's finite state machine behaves correctly across instruction types and status conditions.

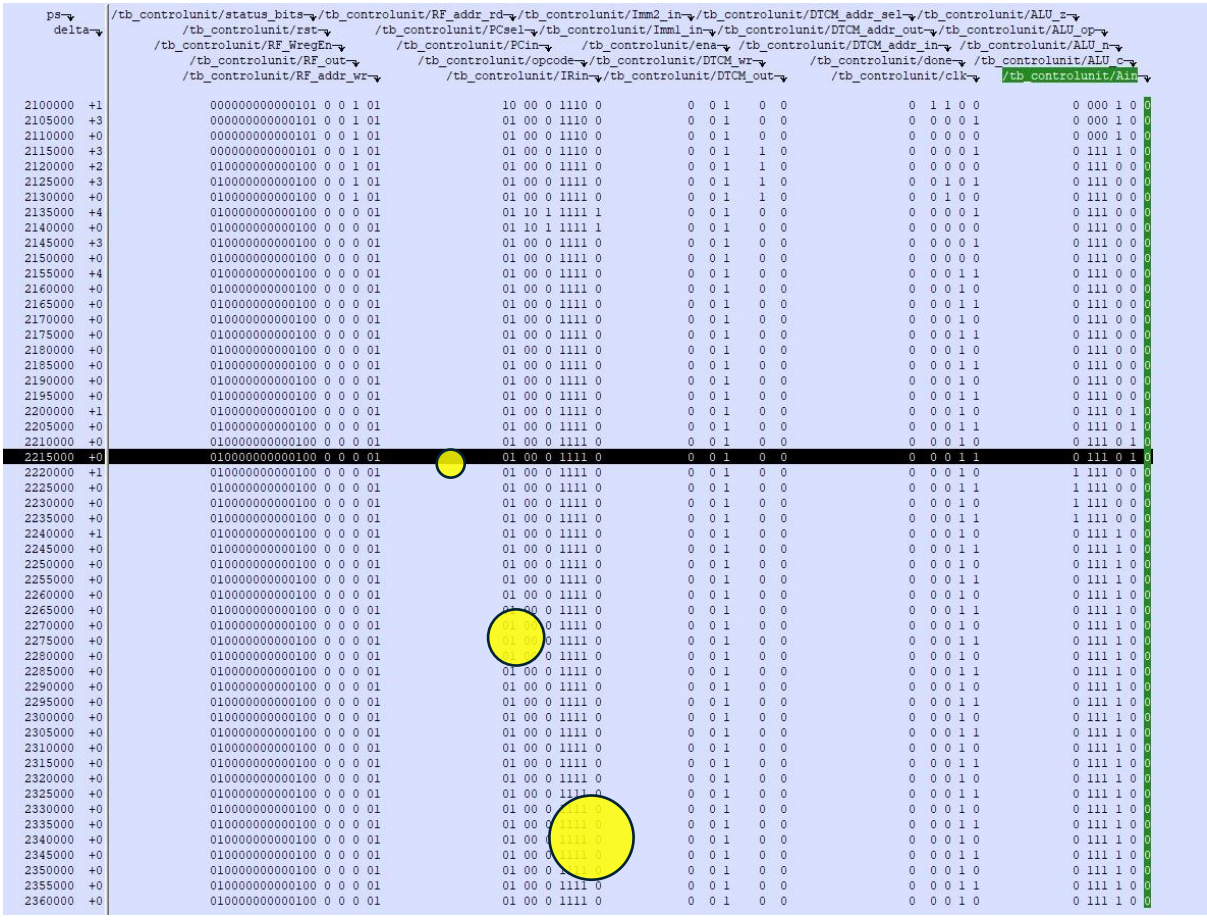


Fig 7 – list simulation of control unit file.

At simulation time 2215000 ps, the control unit responds to the opcode 1111 and the active ALU negative flag (ALU_n = '1') by asserting the control signal Ain. This indicates that the control unit has correctly decoded the instruction and evaluated the relevant status flags, subsequently enabling the input to the A register. This behavior verifies that the control unit's finite state machine (FSM) transitions into the appropriate state for handling the specific instruction under the given condition. The assertion of Ain makes tha ALU pass the singal from bus B to Bus A and store it in register A.

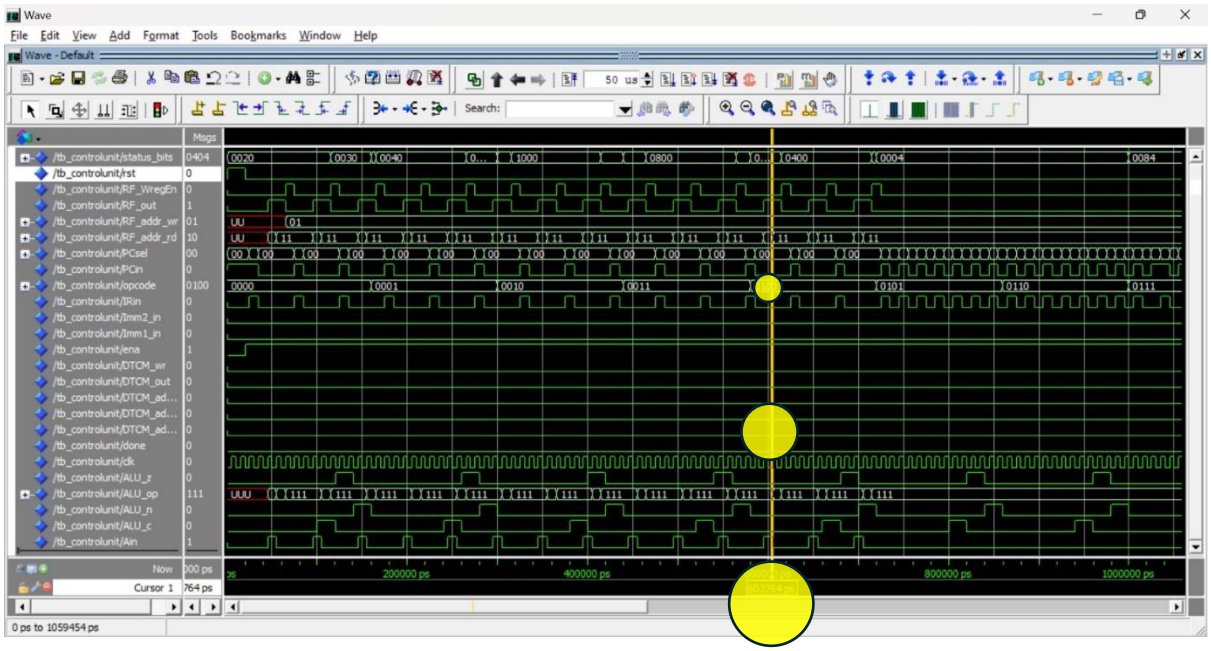


Fig 8 – wave simulation of ControlUnit.vhd file.

At simulation time 603764 ps, the control unit is executing the instruction associated with the opcode 0111. At this point, the ALU_op is correctly set to 111, indicating the selection of the corresponding ALU operation. Simultaneously, several control signals are asserted: IRin, Imm1_in, and Ain, signifying that the instruction is being loaded into the instruction register, an immediate operand is being latched, and the ALU A input is being enabled, respectively. The immediate is thus being passed from bus B into Bus A and register A. this is usually a step that perceeds alu calculations with two operands with the given 2 bus system. Additionally, the done signal remains low, which confirms that the instruction execution is still in progress.