ASSIGNMENT 5 REPORT

ES 204: Digital Systems
Indian Institute of Technology Gandhinagar

April 19, 2025
Devansh Lodha (23110091), Laksh Jain (23110185)

SIMPLE PROCESSOR

OVERVIEW

We have designed a simple processor. The processor has different parts, defined as modules. We have defined the following modules:

1] **processor.v** : This file has the processor module. It is the top module. All other modules are called within this module. It also has the implementation for the program counter or PC. It takes in clock and reset as inputs.

2] **instruction_memory.v** : This file has the instruction_memory module. We define the instruction memory separately from the processor memory. This memory stores the instruction set. We have initialized a 32 byte instruction memory. Also, after having defined the instructions, we fill the empty memory with HALT instruction. This is to prevent the program from running indefinitely if the PC jumps to random address values due to the conditional branch or return instruction.

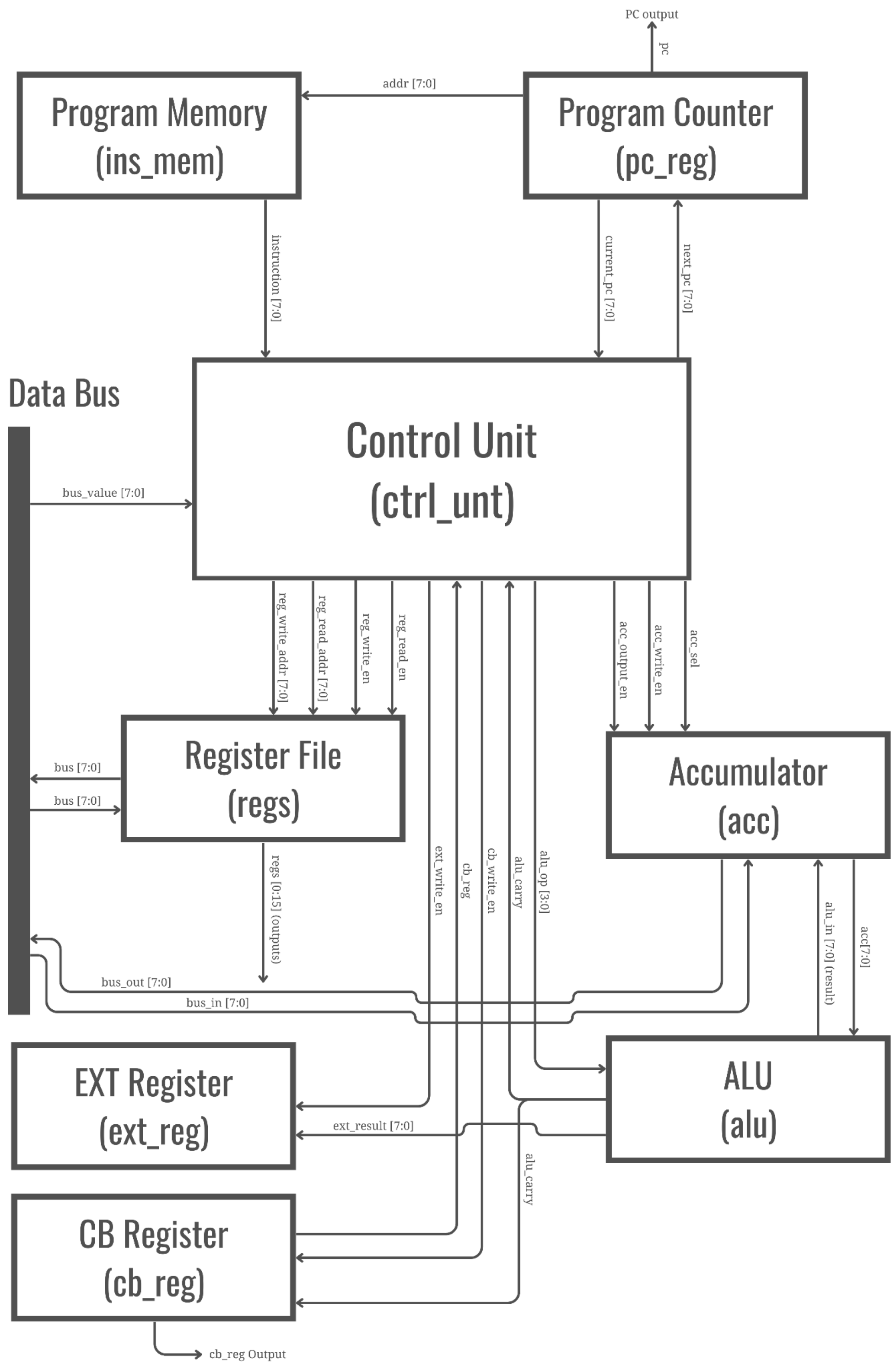
3] **register_file.v** : This file has the register file module. We have initialized the 16 8-bit registers. Also the registers are initialized with some value at time=0. This module takes in the clock, read/write enable signals, read/write register addresses and the bus. Based on the enable signals and the register address, further actions are performed.

4] **accumulator.v** : This has the accumulator module. Based on the select line (from data bus or the output from the ALU) and the write enable signal, the value of the accumulator changes.

5] **alu.v** : This has the implementation for the arithmetic logic unit module. Within this module, we have defined all the arithmetic/logic operations. It receives the instructions from the control unit, based on which, it performs the operation and returns the results.

6] **control_unit.v** : This has the control unit module. It receives the instructions and based on the instruction, it returns the alu operation, enable signals and the select lines

DIAGRAM



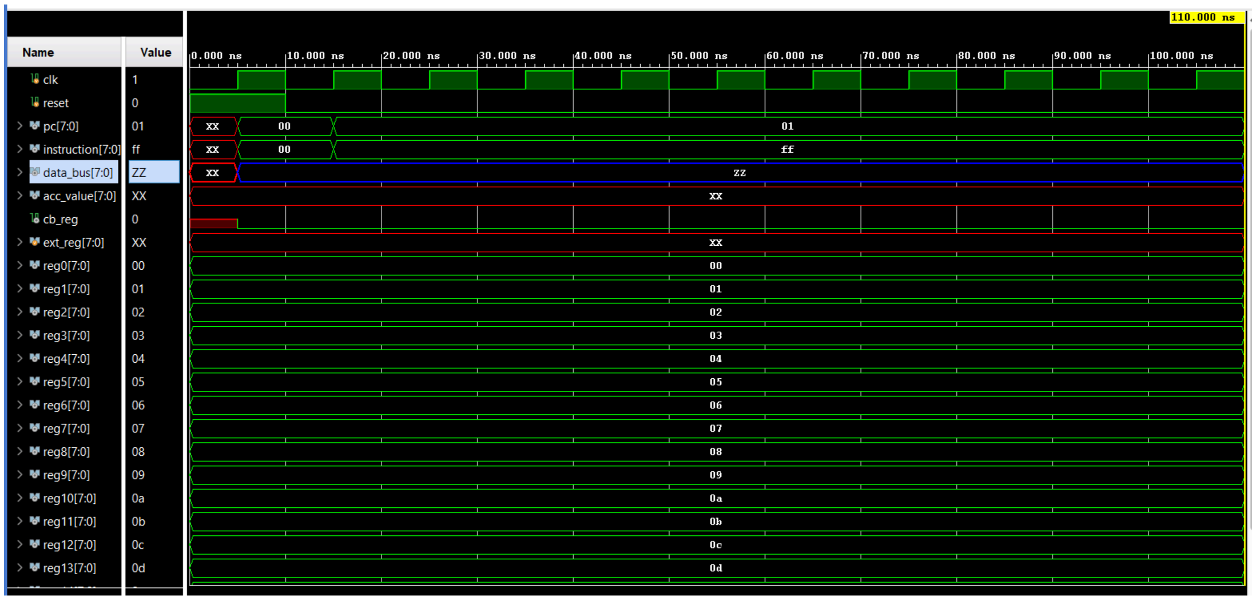
SIMULATION RESULTS

1] INDIVIDUAL OPERATIONS

A] NO OPERATION

```
mem[0] = 8'b0000_0000; // NOP

mem[1] = 8'b1111_1111; // HALT
```



B] ADD

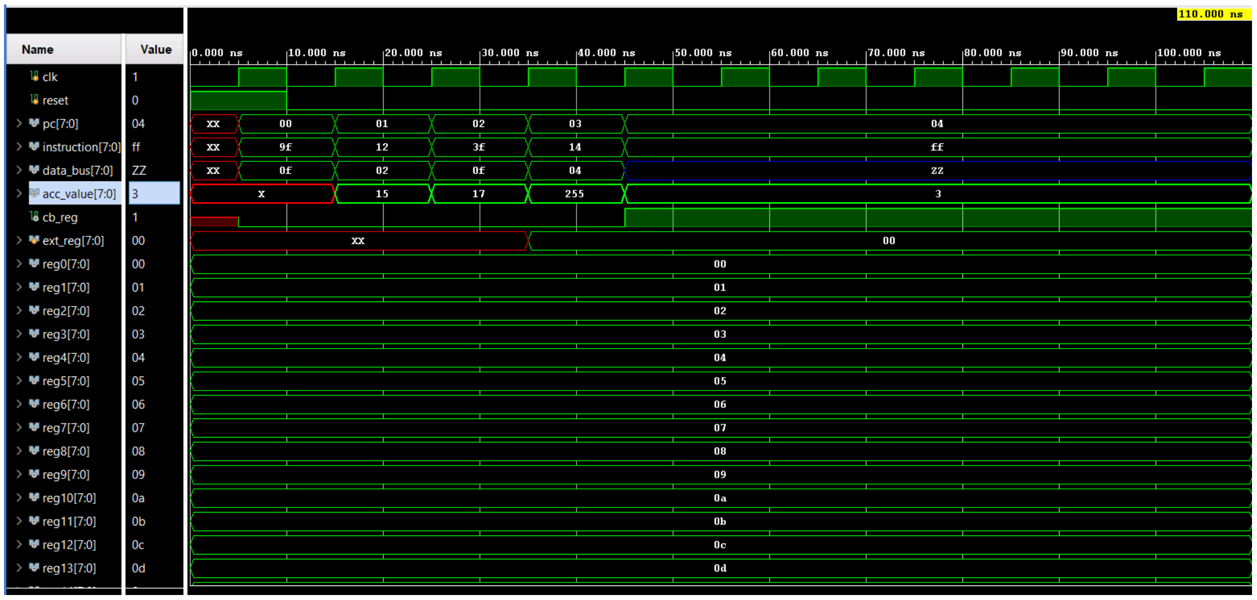
```
mem[0] = 8'b1001_1111; // MOV R15 -> ACC

mem[1] = 8'b0001_0010; // ADD R2 → ACC

mem[2] = 8'b0011_1111; // MUL R15 -> ACC

mem[3] = 8'b0001_0100; // ADD R4 → ACC(C/B=1)

mem[4] = 8'b1111_1111; // HALT
```



C] SUB

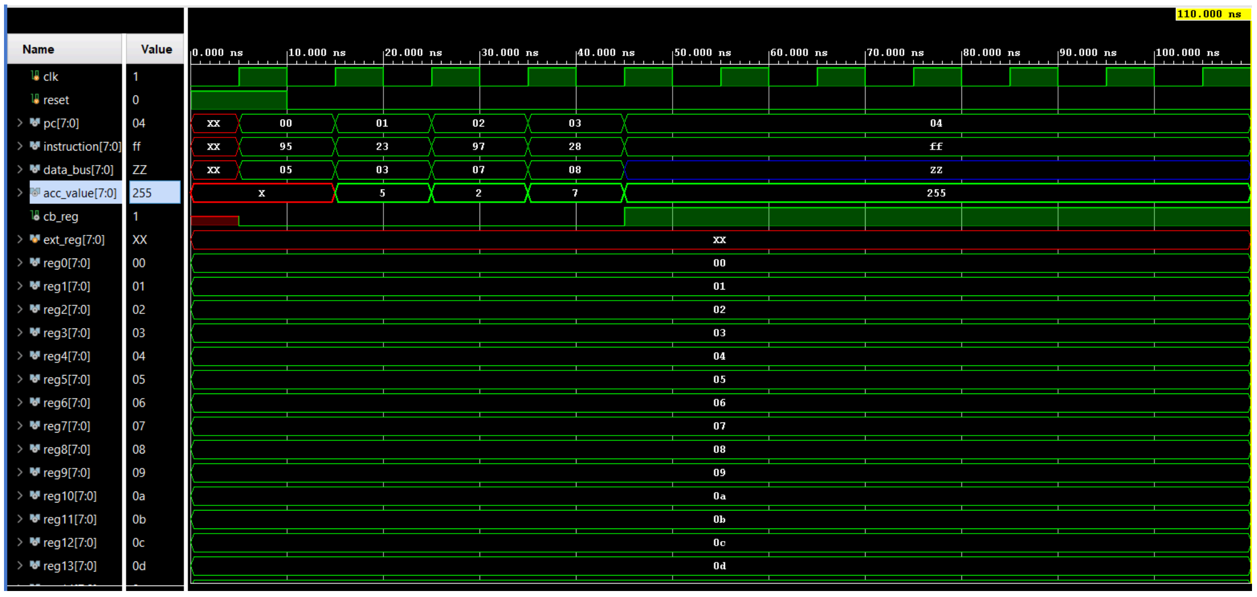
mem[0] = 8'b1001_0101; // MOV R5 -> ACC

mem[1] = 8'b0010_0011; // SUB R3 → ACC (C/B=0)

mem[2] = 8'b1001_0111; // MOV R7 -> ACC

mem[3] = 8'b0010_1000; // SUB R8 → ACC=255 (C/B=1)

mem[4] = 8'b1111_1111; // HALT

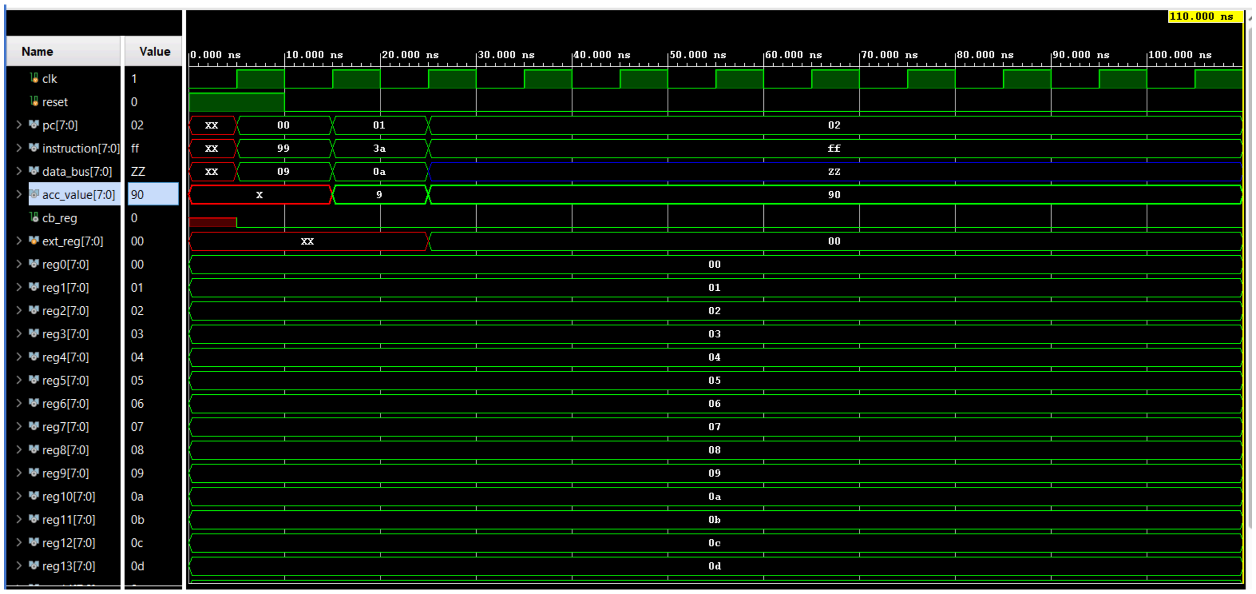


D] MUL

mem[0] = 8'b1001_1001; // MOV R9 -> ACC

mem[1] = 8'b0011_1010; // MUL R10 → ACC (EXT=0)

mem[2] = 8'b1111_1111; // HALT



Name	Value
clk	1
reset	0
pc[7:0]	02
instruction[7:0]	ff
data_bus[7:0]	ZZ
acc_value[7:0]	90
cb_reg	0
ext_reg[7:0]	00
reg0[7:0]	00
reg1[7:0]	01
reg2[7:0]	02
reg3[7:0]	03
reg4[7:0]	04
reg5[7:0]	05
reg6[7:0]	06
reg7[7:0]	07
reg8[7:0]	08
reg9[7:0]	09
reg10[7:0]	0a
reg11[7:0]	0b
reg12[7:0]	0c
reg13[7:0]	0d

E] LS/RS/CSL/CSR/ALS (SHIFTS)

mem[0] = 8'b1001_0101; // MOV R5→ ACC (C/B unaffected)

mem[1] = 8'b0000_0001; // SHL → ACC

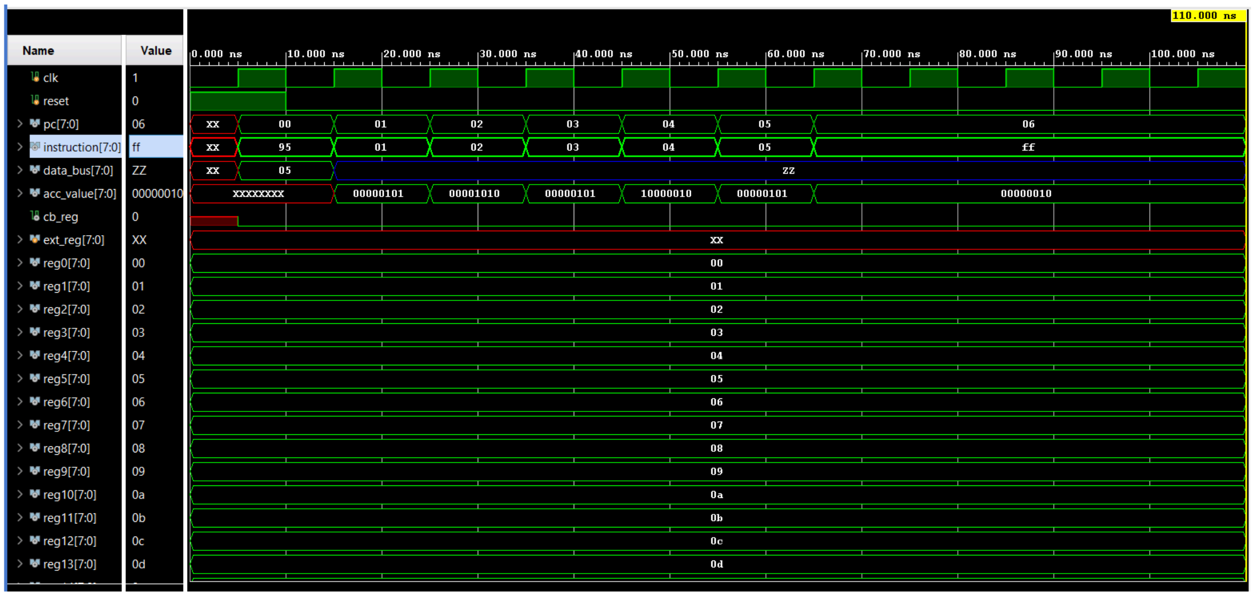
mem[2] = 8'b0000_0010; // SHR → ACC

mem[3] = 8'b0000_0011; // CRS → ACC

mem[4] = 8'b0000_0100; // CLS → ACC

mem[5] = 8'b0000_0101; // ASR → ACC

mem[6] = 8'b1111_1111; // HALT



F] AND/XOR/CMP

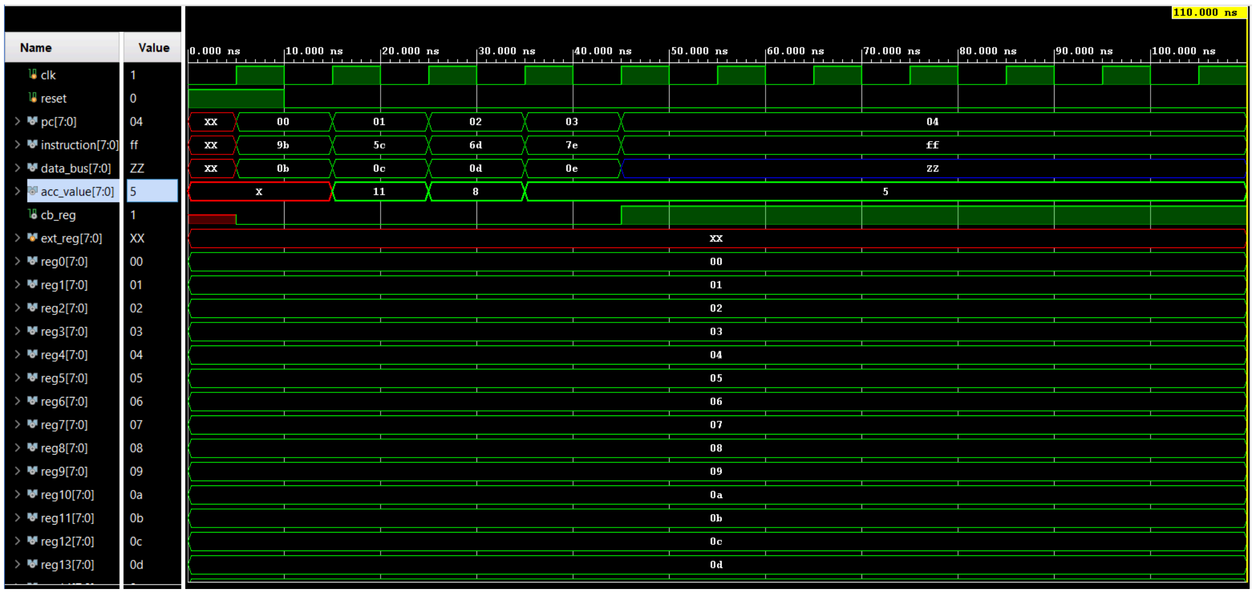
mem[0] = 8'b1001_1011; // MOV R11 -> ACC = 11

mem[1] = 8'b0101_1100; // AND R12 → ACC = 8

mem[2] = 8'b0110_1101; // XOR R13 → ACC = 5

mem[3] = 8'b0111_1110; // CMP R14 → C/B=1

mem[4] = 8'b1111_1111; // HALT



G] INC/DEC

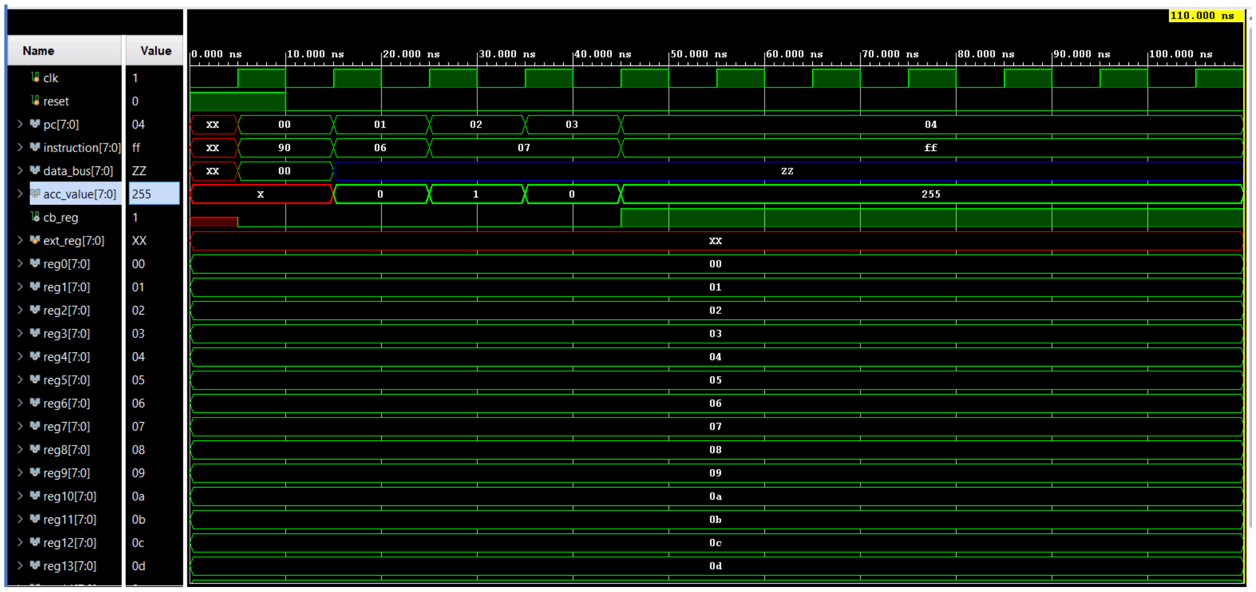
mem[0] = 8'b1001_0000; // MOV R0 -> ACC = 0

mem[1] = 8'b0000_0110; // INC → ACC = 1

mem[2] = 8'b0000_0111; // DEC → ACC = 0

mem[3] = 8'b0000_0111; // DEC → ACC = 255 and C/B = 0 (updates)

mem[4] = 8'b1111_1111; // HALT



H] BR

mem[0] = 8'b1001_0000; // MOV R0 -> ACC = 0

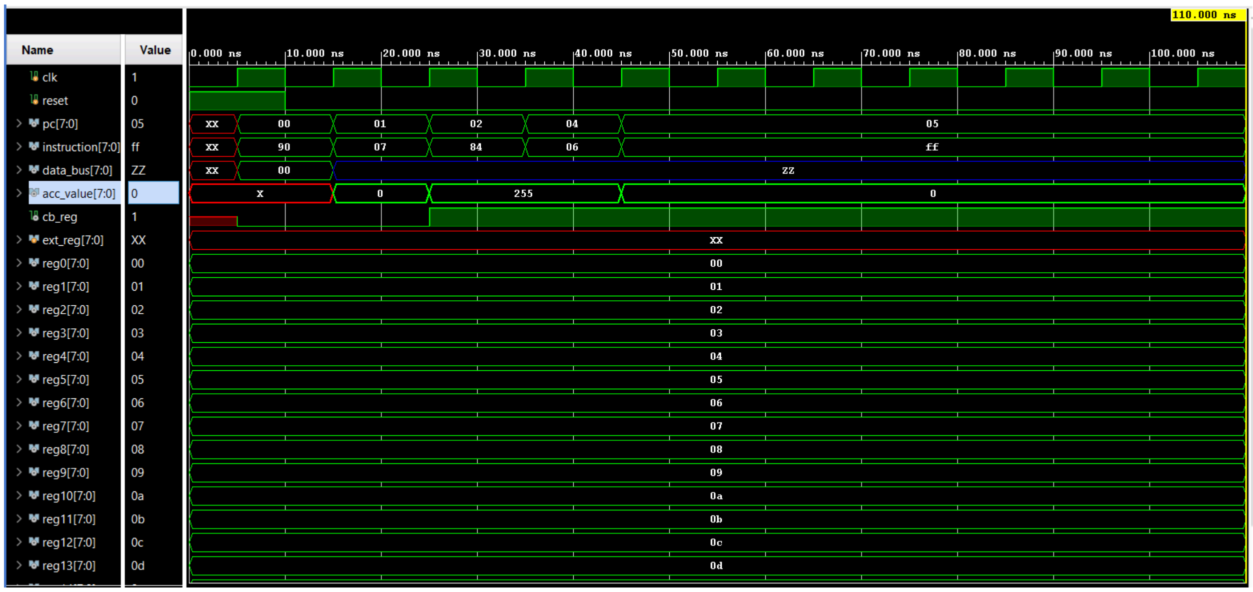
mem[1] = 8'b0000_0111; // DEC → ACC=255, C/B=1

mem[2] = 8'b1000_0100; // Branch to 4 if C/B=1

mem[3] = 8'b1111_1111; // HALT (Fallthrough)

mem[4] = 8'b0000_0110; // INC → ACC = 0 , C/B =1(skips halt in 2)

mem[5] = 8'b1111_1111; // HALT (Branch target)

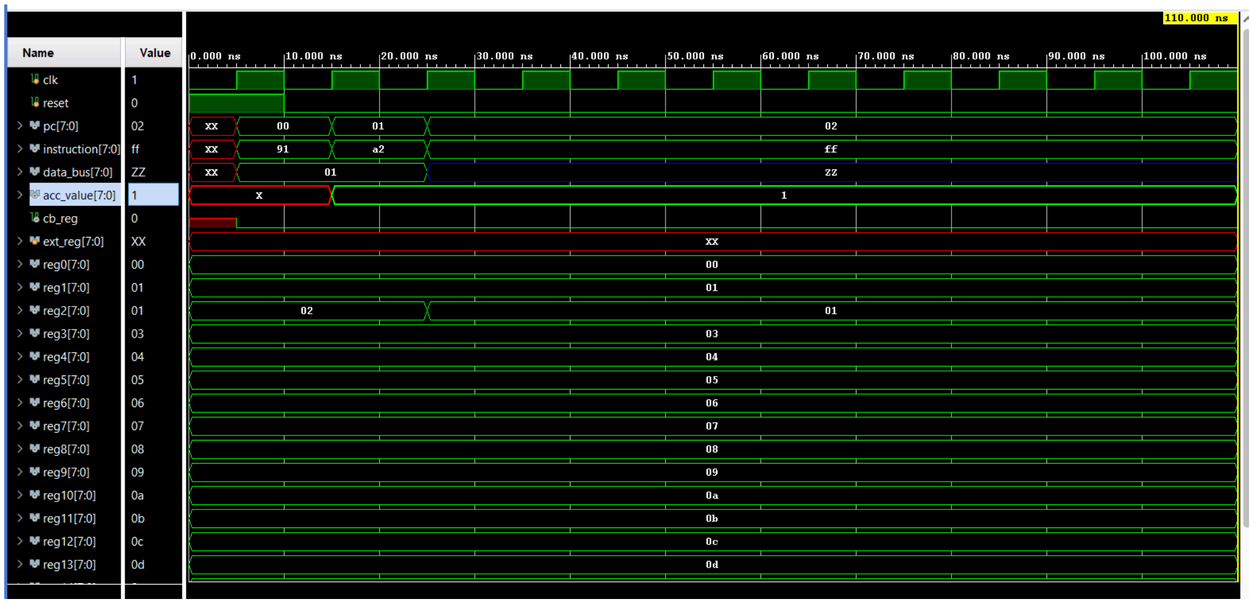


I] MOV

mem[0] = 8'b1001_0001; // MOV R1 -> ACC → ACC=1

mem[1] = 8'b1010_0010; // MOV ACC -> R2=2 → R2=1

mem[2] = 8'b1111_1111; // HALT



J] RETURN

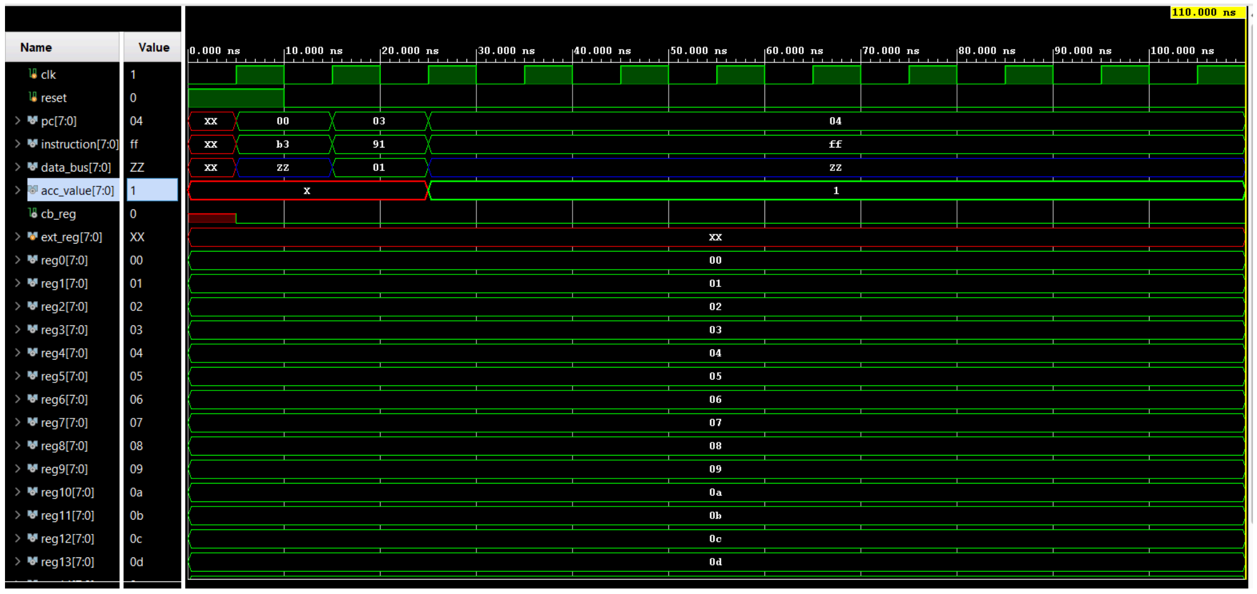
mem[0] = 8'b1011_0011; // RET → PC=3

mem[1] = 8'b1111_1111; // HALT (Unreachable)

mem[2] = 8'b1111_1111; // HALT (Unreachable)

mem[3] = 8'b1001_0001; // MOV R1 -> ACC → ACC=1

mem[4] = 8'b1111_1111; // HALT (Target)



2] COMBINED OPERATIONS

A] INSTRUCTION SET: (**GIVEN INSTRUCTION SET IN ASSIGNMENT**)

mem[0] = 8'b1001_0001; // MOV R1 -> ACC

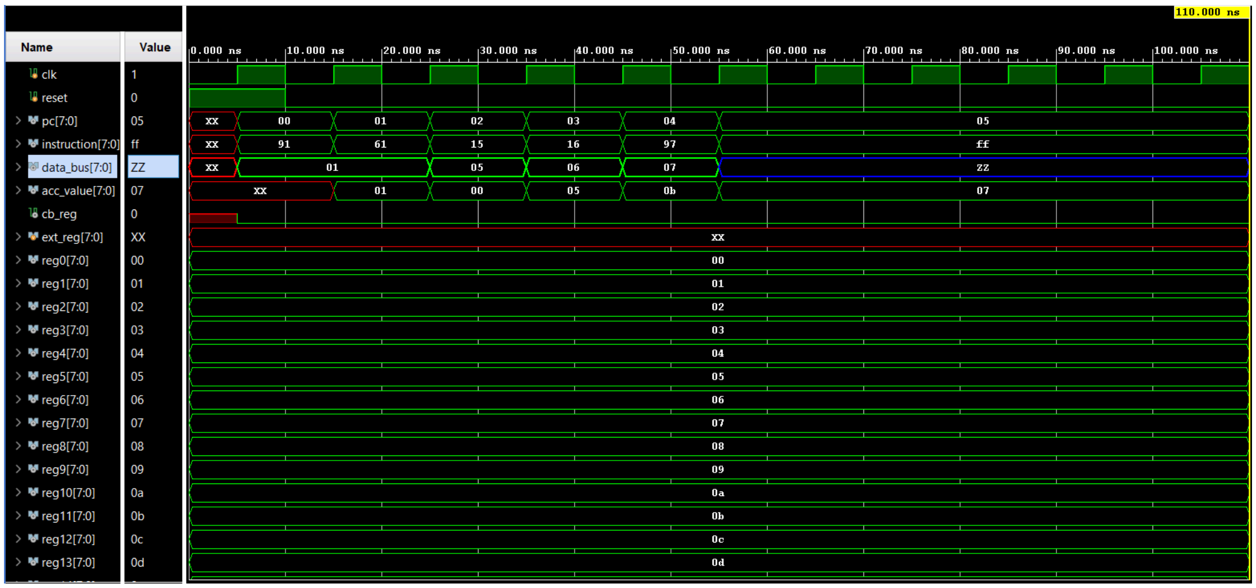
mem[1] = 8'b0110_0001; // XRA R1

mem[2] = 8'b0001_0101; // ADD R5

mem[3] = 8'b0001_0110; // ADD R6

mem[4] = 8'b1001_0111; // MOV R7 -> ACC

mem[5] = 8'b1111_1111; // HALT

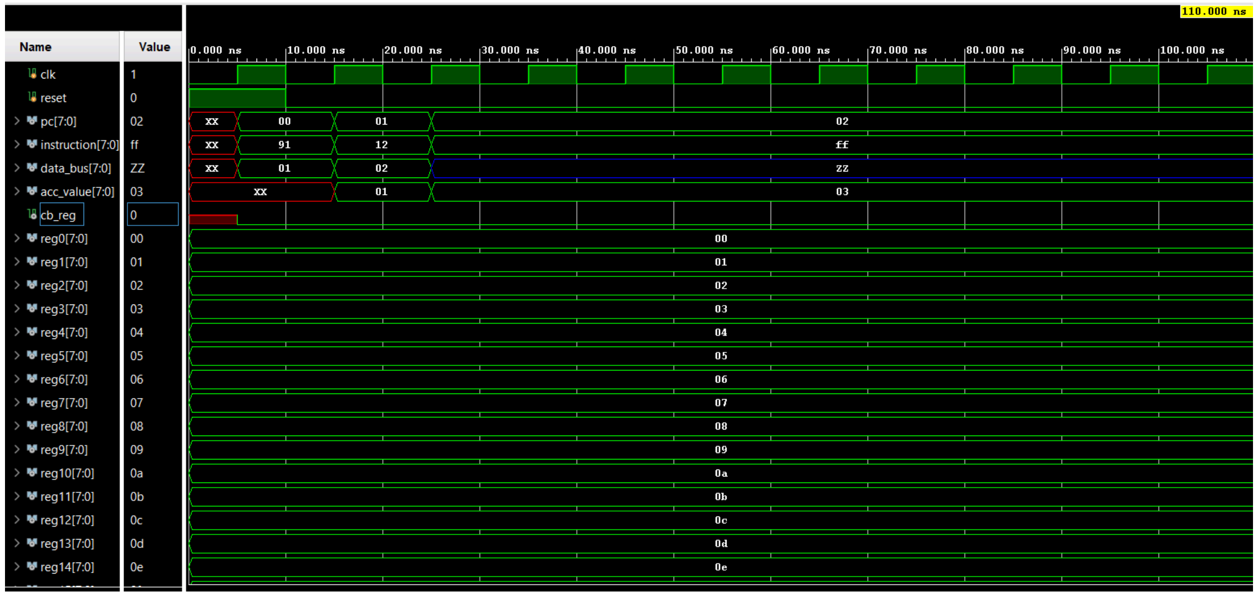


B] INSTRUCTION SET:

mem[0] = 8'b1001_0001; // MOV R1 to ACC

mem[1] = 8'b0001_0010; // ADD R2

mem[2] = 8'b1111_1111; // HALT



C] INSTRUCTION SET:

mem[0] = 8'b1001_0010; // MOV R2 -> ACC

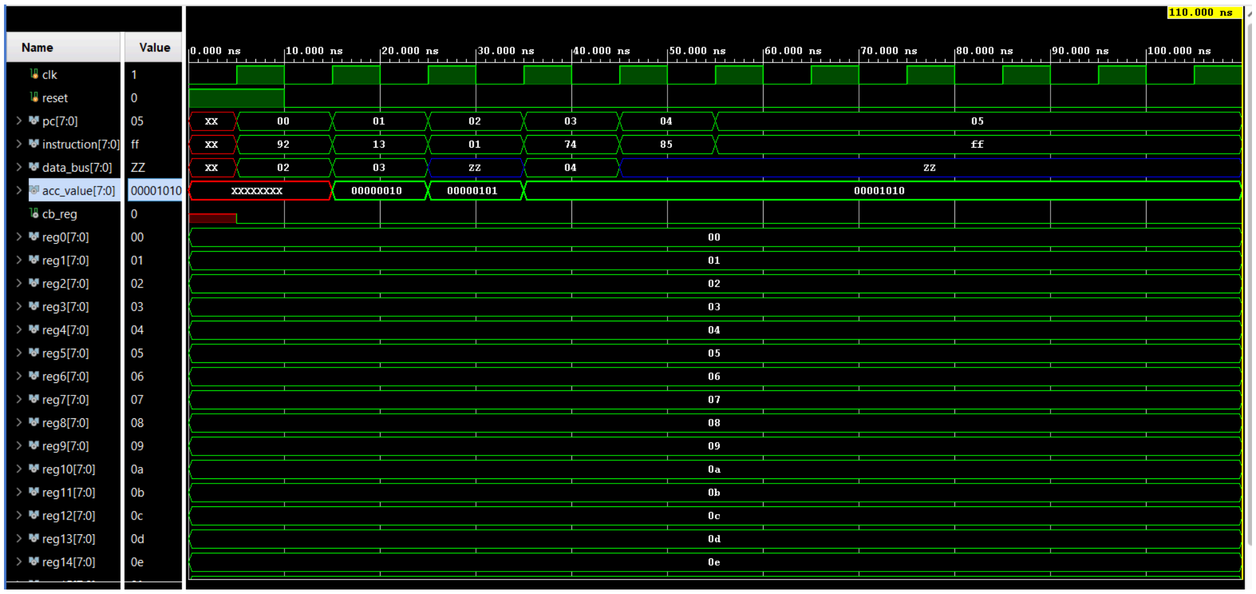
mem[1] = 8'b0001_0011; // ADD R3

mem[2] = 8'b0000_0001; // SHL ACC

mem[3] = 8'b0111_0100; // CMP R4

mem[4] = 8'b1000_0101; // Branch to 5 if C/B=1

mem[5] = 8'b1111_1111; // HALT



D] INSTRUCTION SET:

mem[0] = 8'b1001_1010; // MOV R10 -> ACC

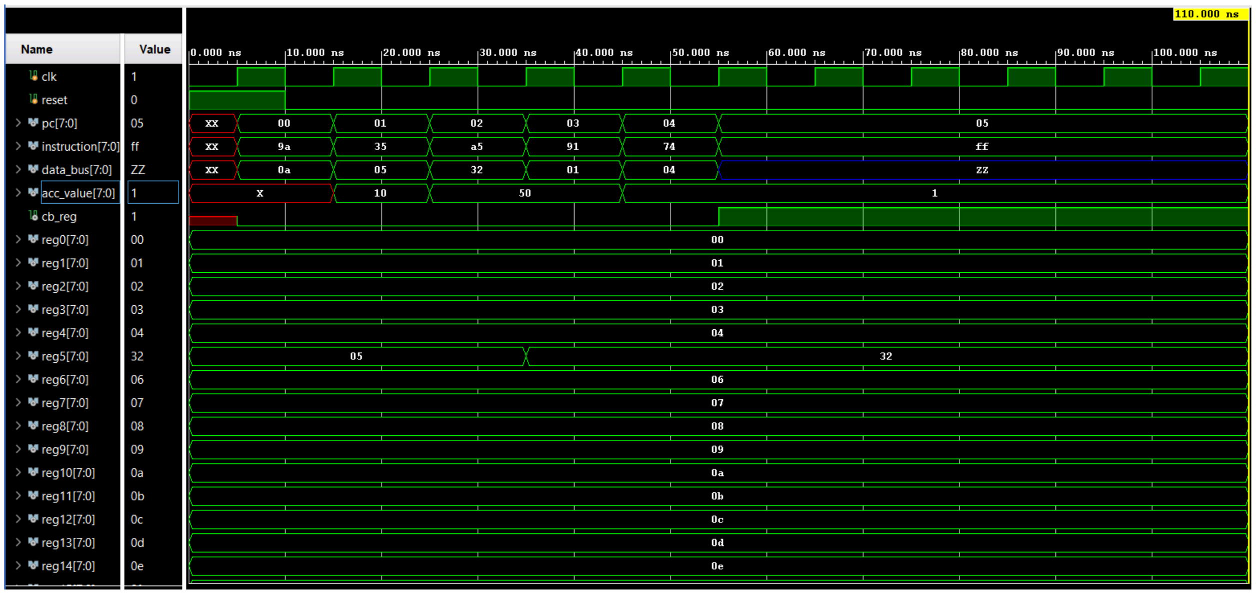
mem[1] = 8'b0011_0101; // MULTIPLY R5

mem[2] = 8'b1010_0101; // MOV ACC -> R5

mem[3] = 8'b1001_0001; // MOV R1 -> ACC

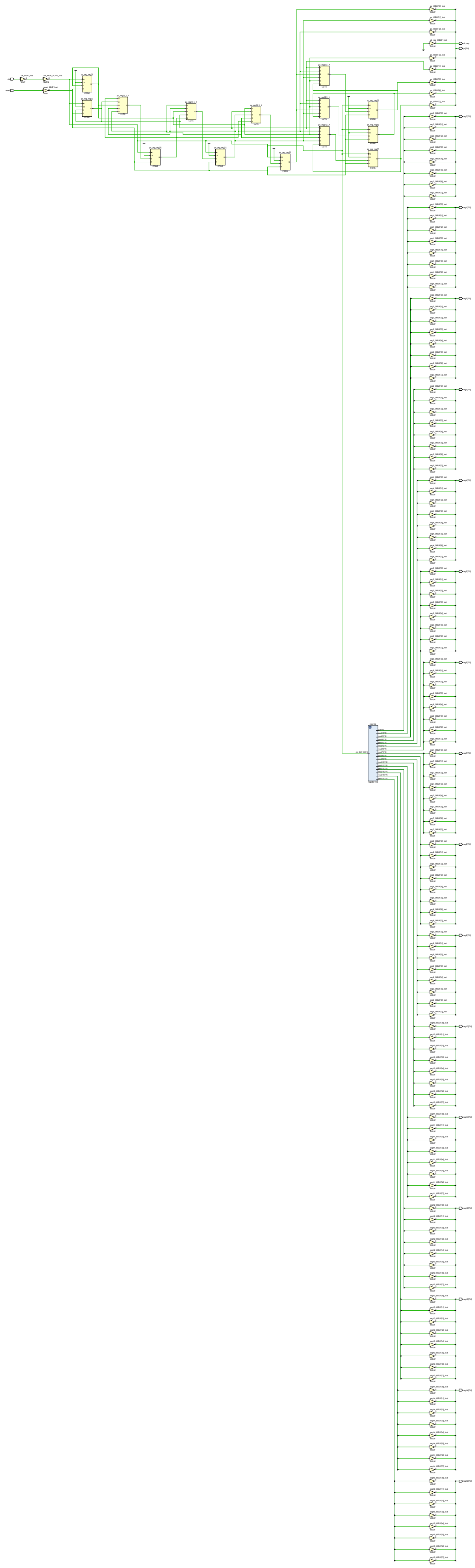
mem[4] = 8'b0111_0100; // COMPARE WITH R4

mem[5] = 8'b1111_1111; // HALT



Name	Value
clk	1
reset	0
pc[7:0]	05
instruction[7:0]	ff
data_bus[7:0]	ZZ
acc_value[7:0]	1
cb_reg	1
reg0[7:0]	00
reg1[7:0]	01
reg2[7:0]	02
reg3[7:0]	03
reg4[7:0]	04
reg5[7:0]	32
reg6[7:0]	06
reg7[7:0]	07
reg8[7:0]	08
reg9[7:0]	09
reg10[7:0]	0a
reg11[7:0]	0b
reg12[7:0]	0c
reg13[7:0]	0d
reg14[7:0]	0e

SCHEMATIC DIAGRAM GENERATED USING VERILOG


clk
reset
clk_IBUF_inst
IBUF
clk_IBUF_BUFG_inst
BUFG
reset_IBUF_inst
pc_reg_reg[3]
pc_reg_reg[4]
FDRE
pc_reg[2]_i_1
pc_reg[1]_i_1
pc_reg[0]_i_1
LUT5
pc_reg_reg[2]
pc_reg_reg[1]
pc_reg_reg[0]
pc_reg[5]_i_1
pc_reg[6]_i_1
pc_reg[7]_i_1
LUT6
pc_reg_reg[5]
pc_reg_reg[6]
pc_reg_reg[7]
pc_OBUF[0]_inst
pc_OBUF[1]_inst
pc_OBUF[2]_inst
db_reg_OBUF_inst
OBUF
db_reg
pc[7:0]
pc_OBUF[3]_inst
pc_OBUF[4]_inst
pc_OBUF[5]_inst
pc_OBUF[6]_inst
pc_OBUF[7]_inst
reg0_OBUF[0]_inst
reg0[7:0]
reg1_OBUF[0]_inst
reg1[7:0]