RISC-V RV32I RTL DESIGN USING VERILOG HDL

FINAL PROJECT REPORT

UNDER GUIDELINES

Of

MAVEN SILICON

by

CHAKKA LAVISH VAMSI RAJA

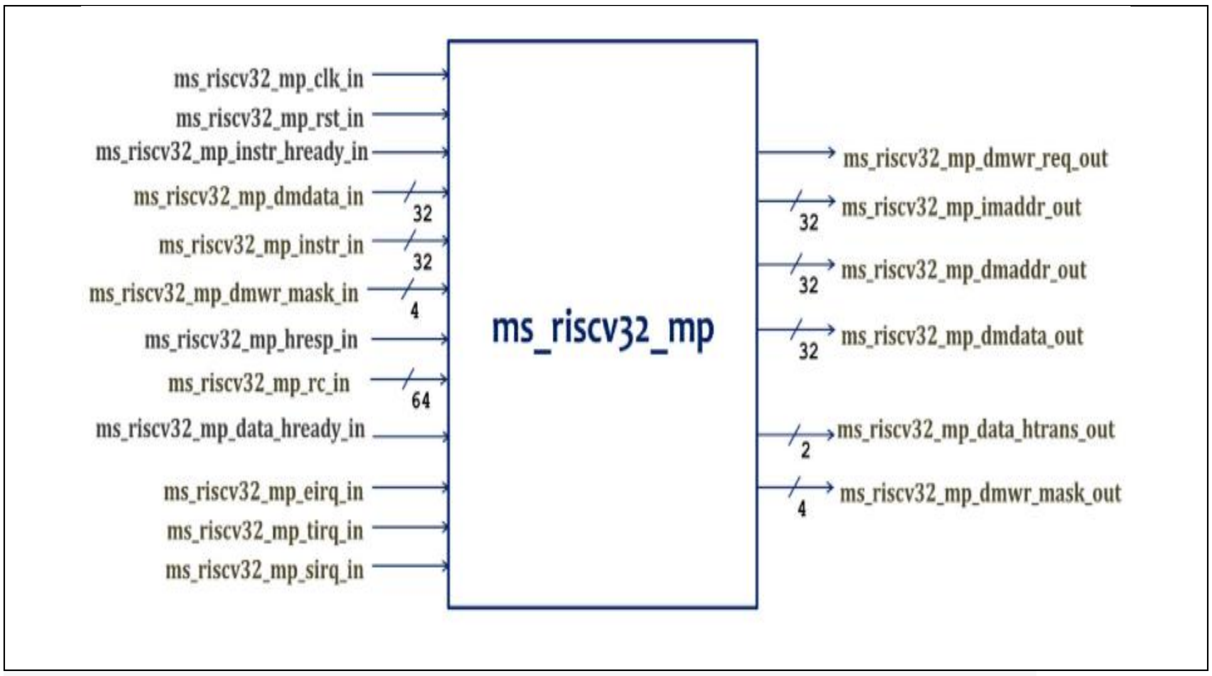
20BEV7004

VITAP

Project Overview:

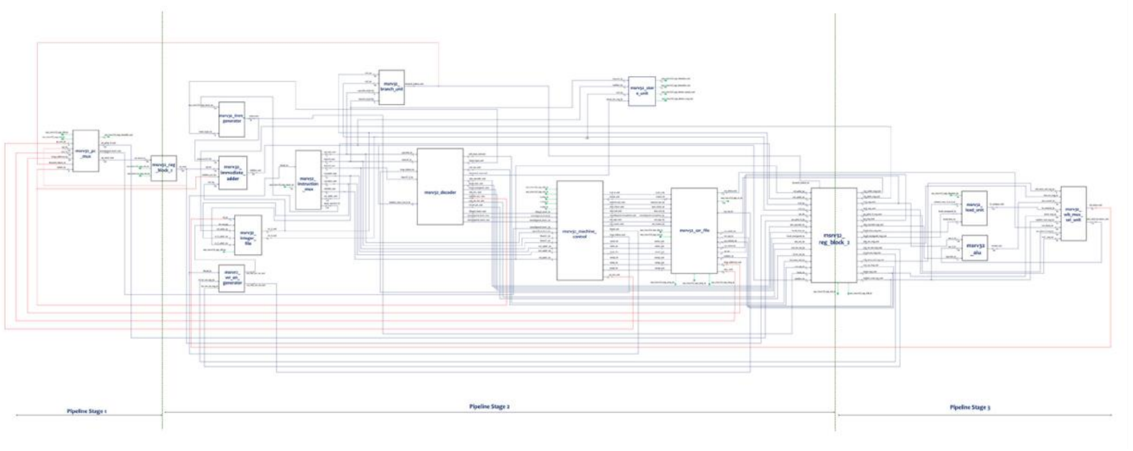
The RISC-V RV32I RTL Design project aims to design a 32-bit RISC-V processor using the RV32I instruction set architecture. The project involves the development of the processor's RTL (Register Transfer Level) design using Verilog HDL (Hardware Description Language) and the verification of its functionality through simulation.

Top Module Block Diagram and Functionality:



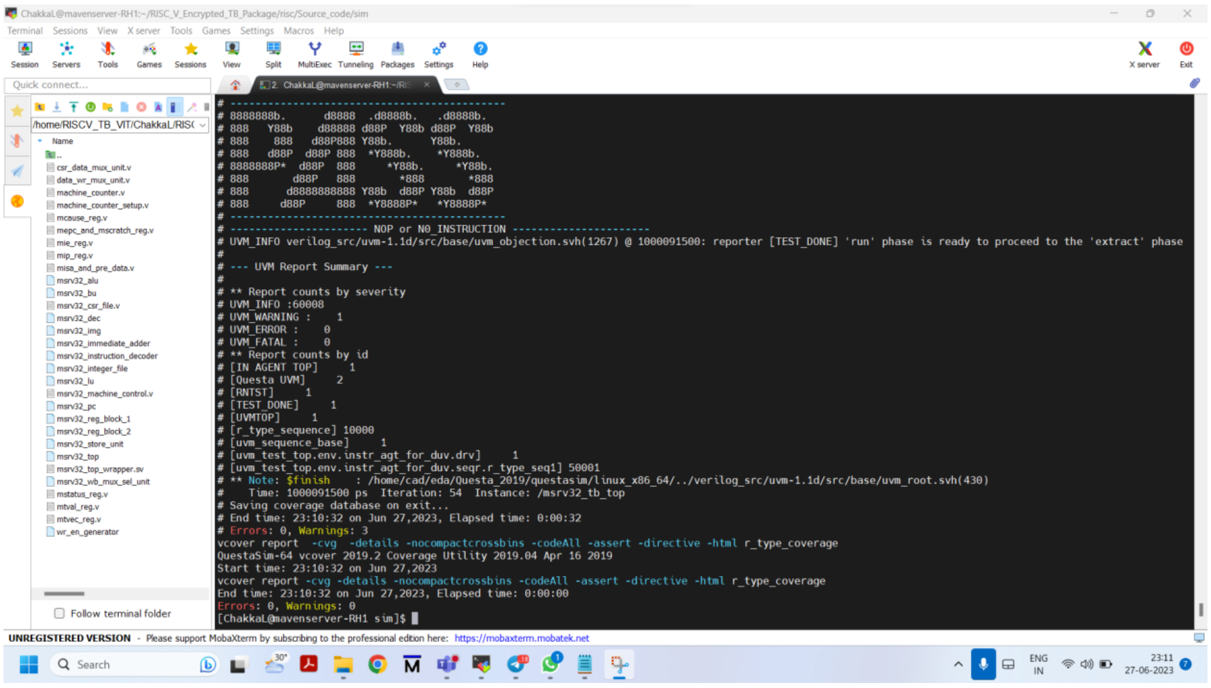
The top module in a digital design is responsible for coordinating the interactions between various submodules or functional units within the design. It serves as the main control unit and manages the flow of data and control signals throughout the design. The top module typically receives input signals, processes them through different submodules, and generates output signals accordingly. Its functionality can include tasks such as data routing, signal synchronization, clock management, and overall system control. Essentially, the top module acts as the central hub that connects and controls all the componentsof the design.

MICRO ARCHITECTURE OF RISC-V RV32I :



FINAL UVM VERIFICATION FOR RISC-V RV32I RTL

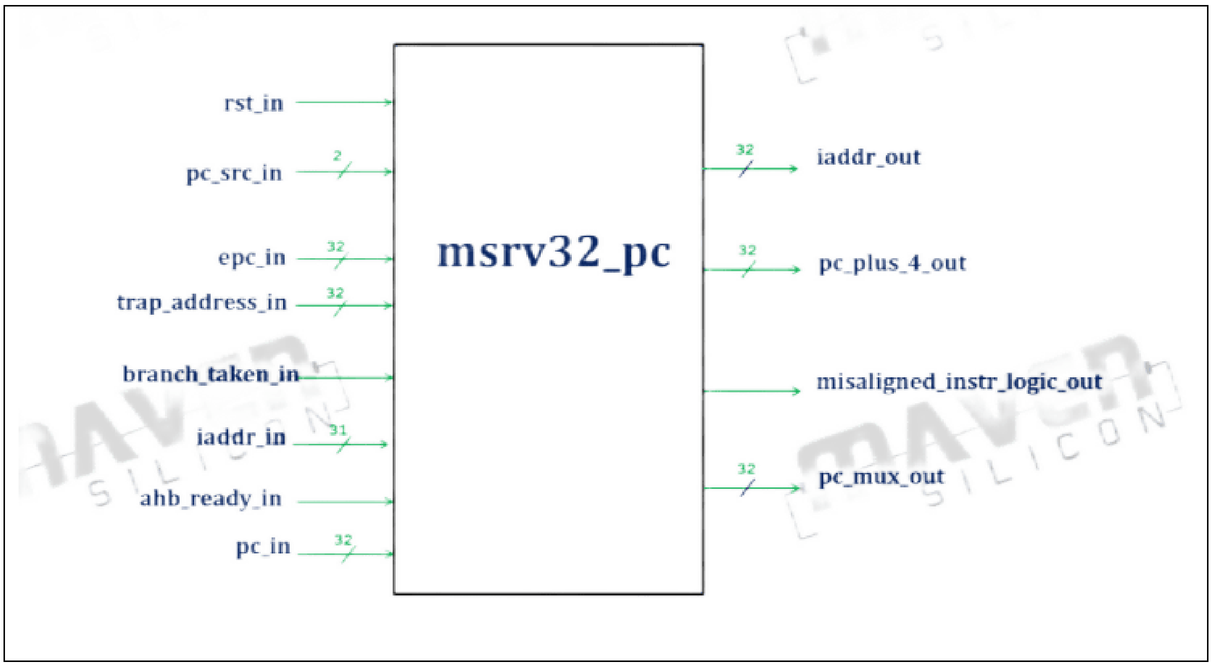
PASS



```
# -------------------------------------------
# 8888888b.      d8888  .d8888b.   .d8888b.
# 888   Y88b    d88888 d88P  Y88b d88P  Y88b
# 888    888   d88P888 Y88b.      Y88b.
# 888   d88P  d88P 888  *Y888b.    *Y888b.
# 8888888P*  d88P  888     *Y88b.     *Y88b.
# 888       d88P   888       *888       *888
# 888      d8888888888 Y88b  d88P Y88b  d88P
# 888     d88P     888  *Y8888P*   *Y8888P*
# -------------------------------------------
# ---------------------- NOP or N0_INSTRUCTION ----------------------
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 1000091500: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :60008
# UVM_WARNING :    1
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [IN AGENT TOP]     1
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [UVMTOP]     1
# [r_type_sequence] 10000
# [uvm_sequence_base]     1
# [uvm_test_top.env.instr_agt_for_duv.drv]     1
# [uvm_test_top.env.instr_agt_for_duv.seqr.r_type_seq1] 50001
# ** Note: $finish    : /home/cad/eda/Questa_2019/questasim/linux_x86_64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 1000091500 ps  Iteration: 54  Instance: /msrv32_tb_top
# Saving coverage database on exit...
# End time: 23:10:32 on Jun 27,2023, Elapsed time: 0:00:32
# Errors: 0, Warnings: 3
vcover report  -cvg  -details -nocompactcrossbins -codeAll -assert -directive -html r_type_coverage
QuestaSim-64 vcover 2019.2 Coverage Utility 2019.04 Apr 16 2019
Start time: 23:10:32 on Jun 27,2023
vcover report -cvg -details -nocompactcrossbins -codeAll -assert -directive -html r_type_coverage
End time: 23:10:32 on Jun 27,2023, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
[ChakkaL@mavenserver-RH1 sim]$ 
```

SUB MODULES BLOCK DIAGRAM WITH DESCRIPTION AND IT'S FUNCTIONALITY:

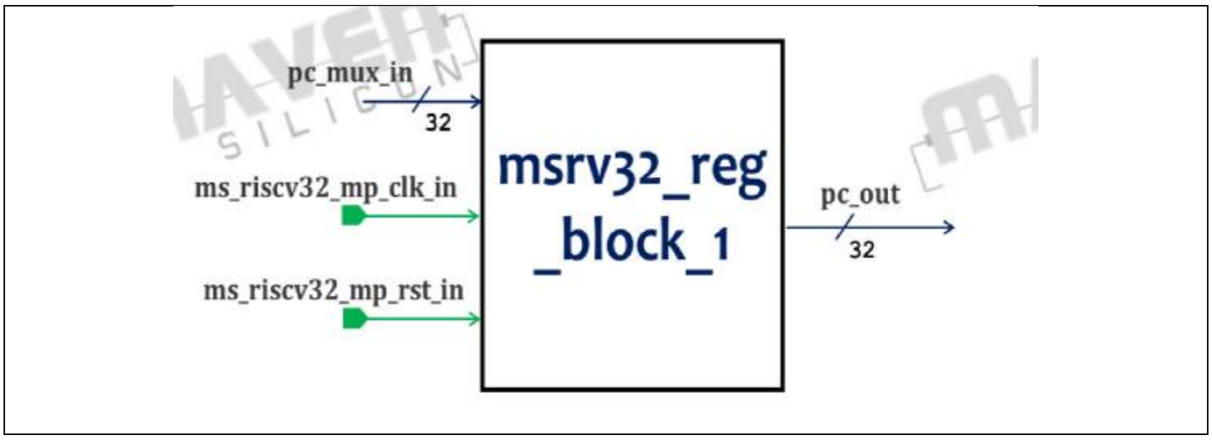
PC MUX:



Hold address of next instruction to be executed, it uses every input to the output

Boot_address to play with every input with its reset.

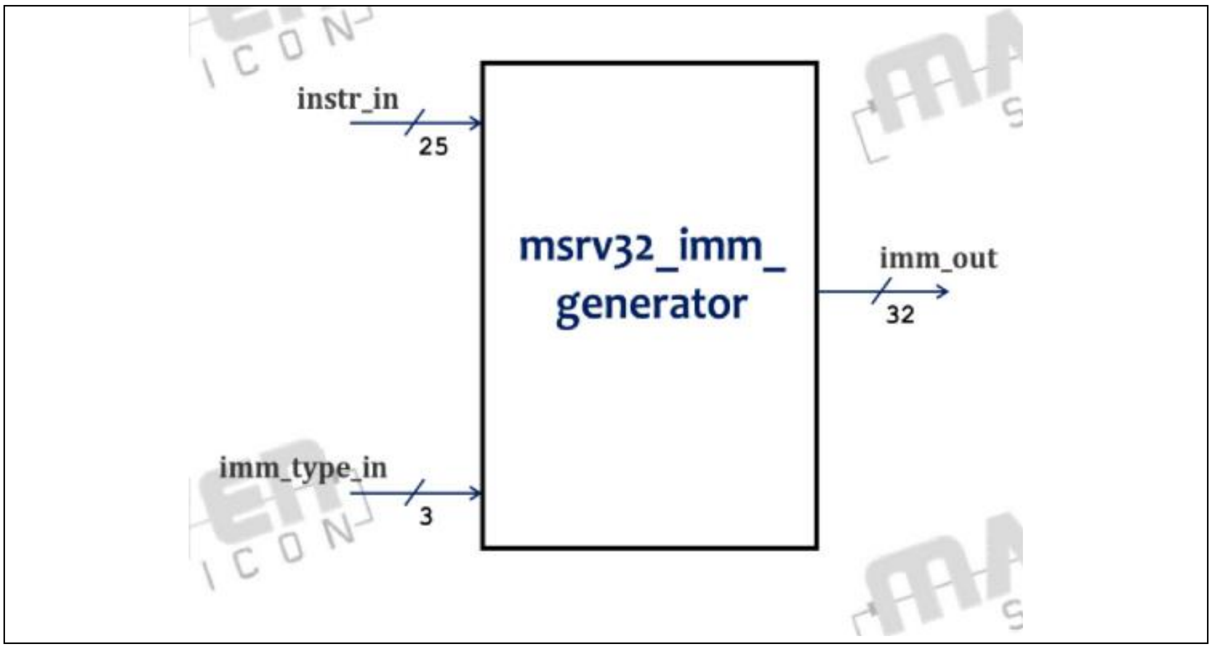
REG BLOCK 1



It register the pc_mux_in and produces the output at the posedge of clk if

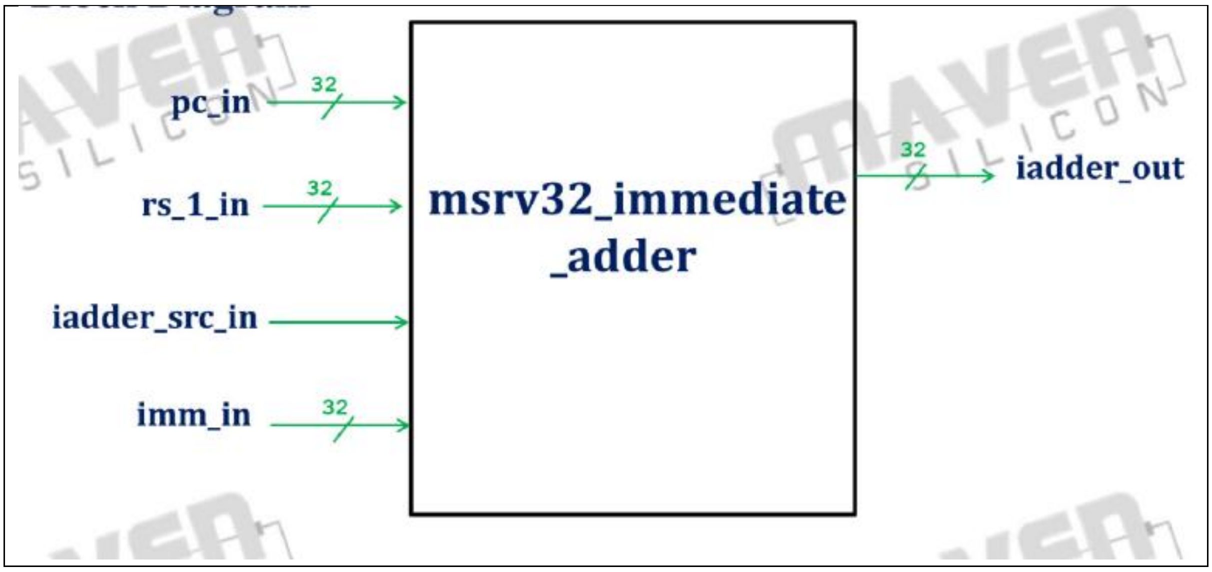
There is no reset.

IMMEDIATE GENERATOR



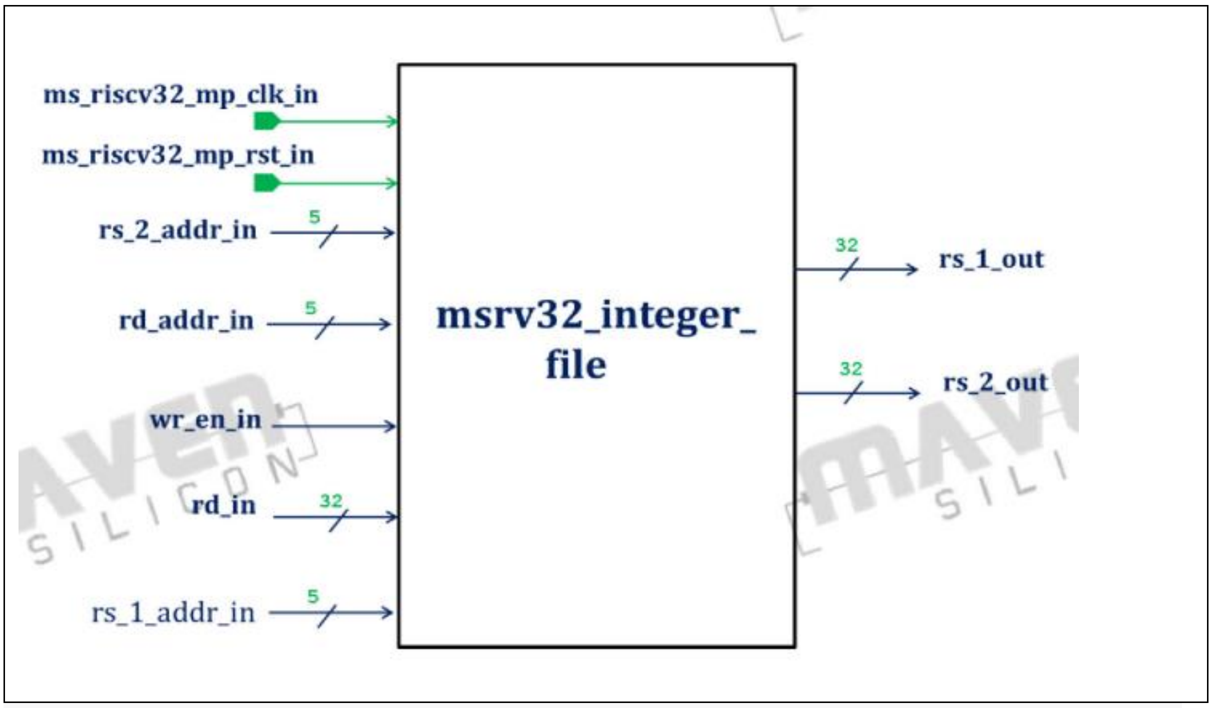
Rearranges the immediate bits continued in the instruction.

IMMEDIATE ADDER



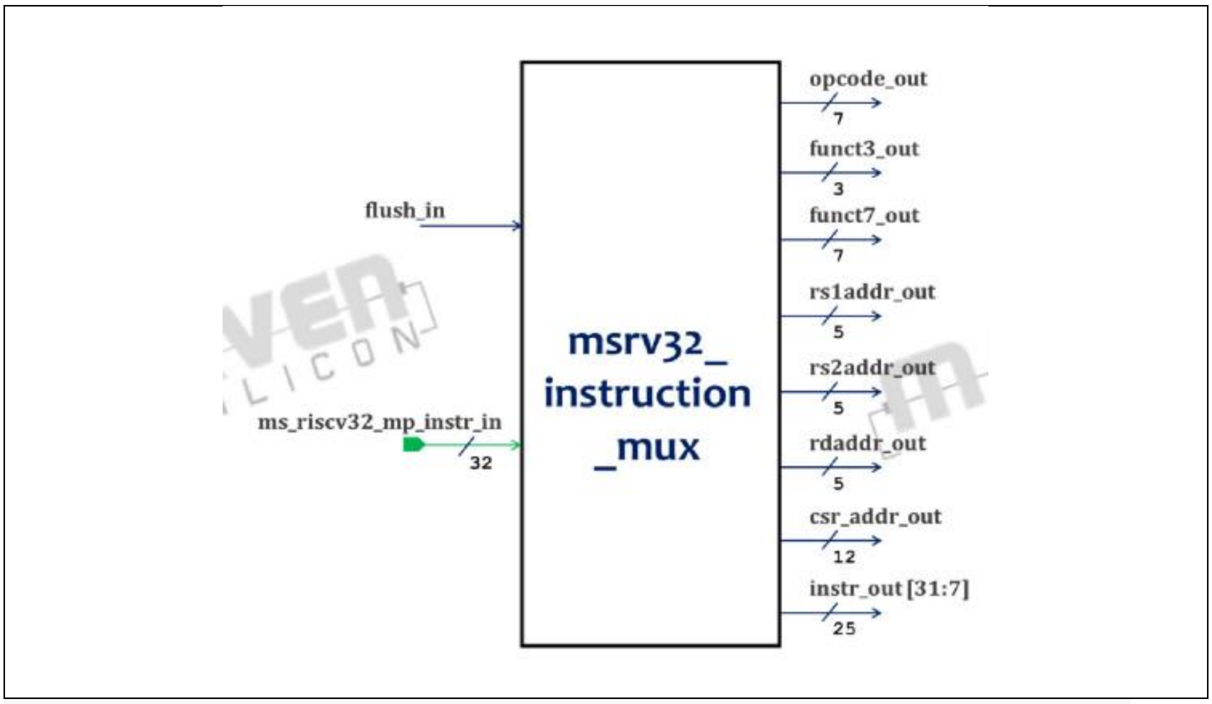
Addes the immediate bits.

INTEGER FILE WRITE ENABLE GENERATOR



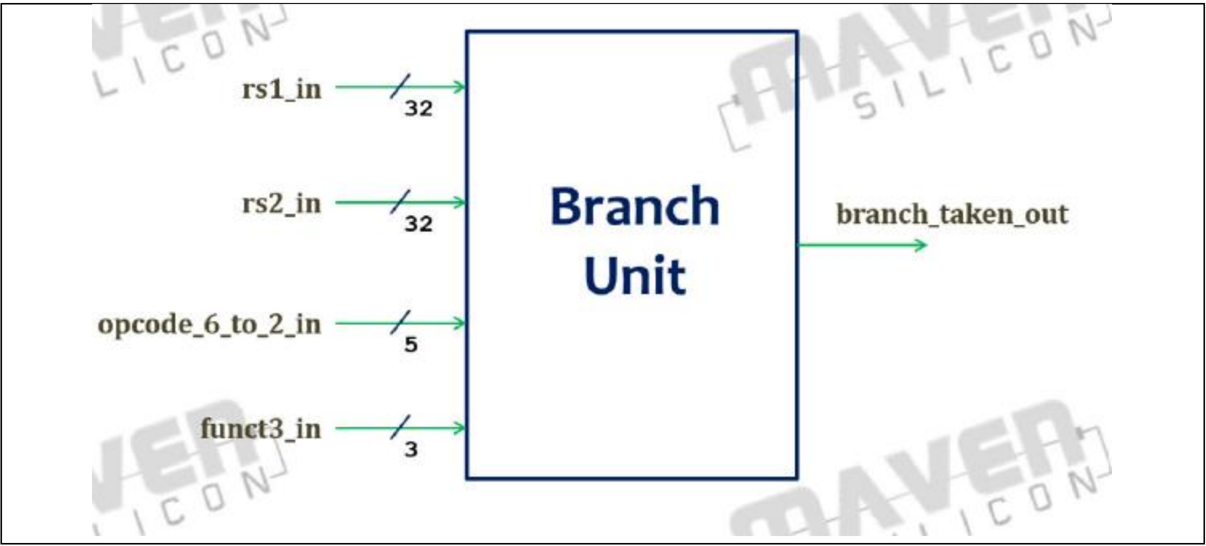
Integer file is a commonly used file format for storing data as plain text, where each line represents an integer value. It is typically used for input/output operations in programming, allowing for easy reading and writing of integer data from/to files.

INSTRUCTION MUX



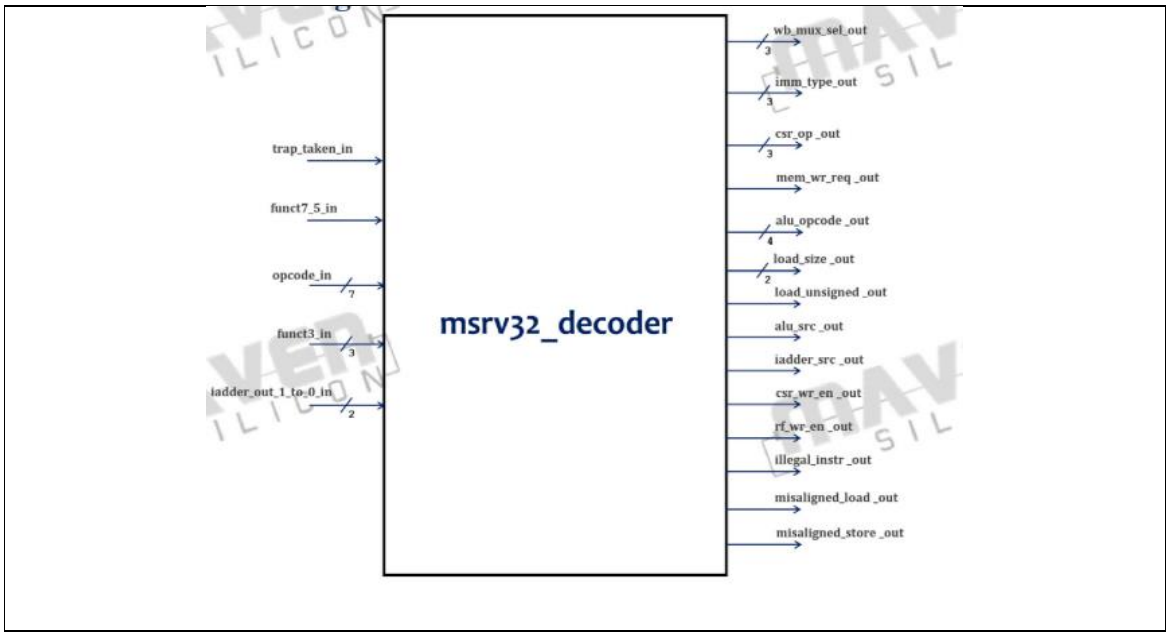
The instruction mux selects the appropriate instruction based on the control signals and opcode, enabling the processor to fetch the correct instruction for execution. It multiplexes between different instructions in the instruction memory based on the control signals and opcode inputs, providing the desired instruction to the subsequent stages of the processor for further processing and execution.

BRANCH UNIT



The branch unit in a processor is responsible for evaluating branch conditions and determining whether a branch instruction should be taken or not based on the specified condition, controlling the flow of program execution accordingly. It compares register values or other operands to determine if the branch condition is true, and signals the control unit to update the program counter (PC) to the target address if the condition is met.

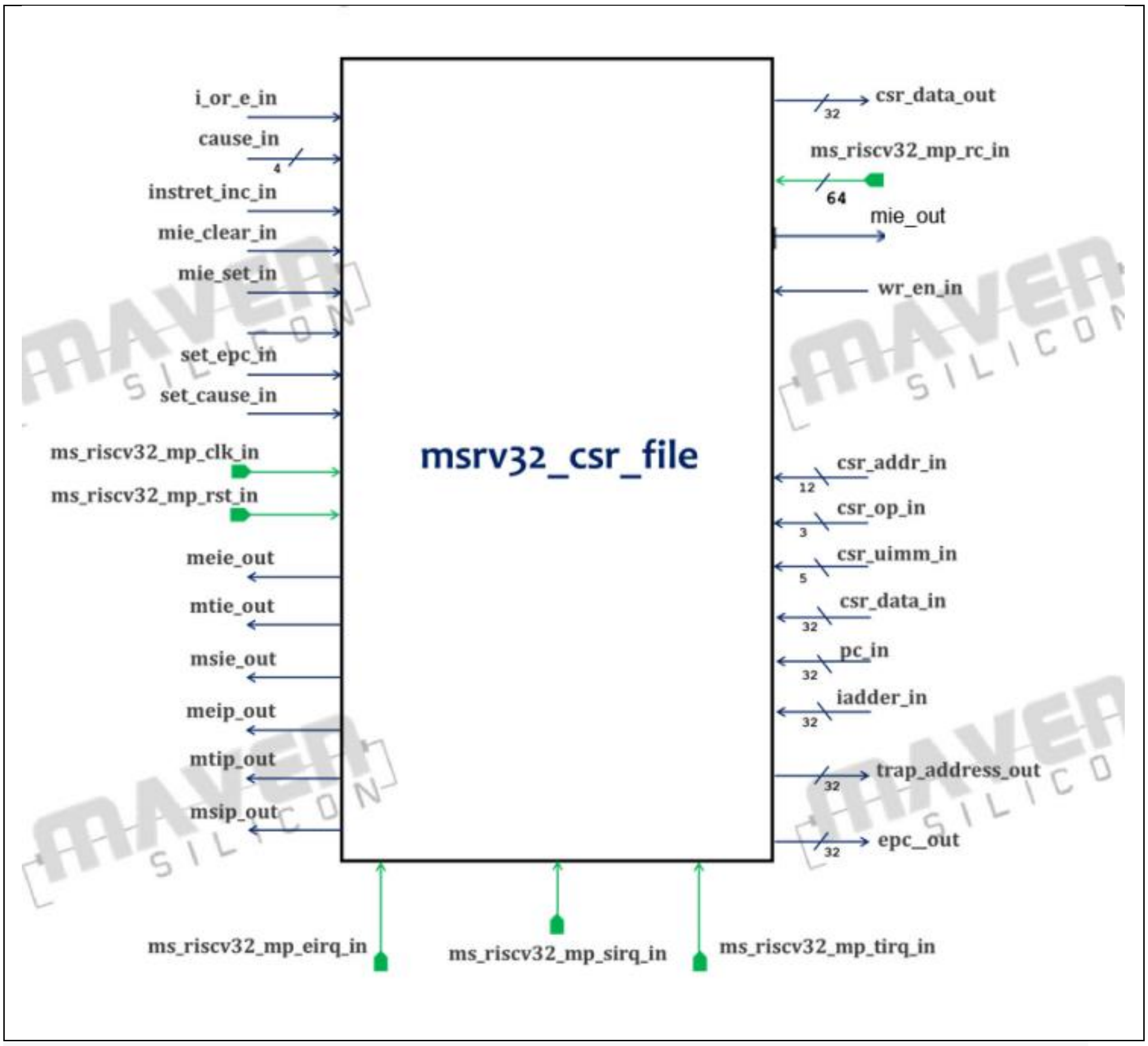
Decoder



Decoder Description: A decoder is a combinational logic circuit that takes an input code and activates a specific output line based on the input code, enabling a one-to-many mapping.

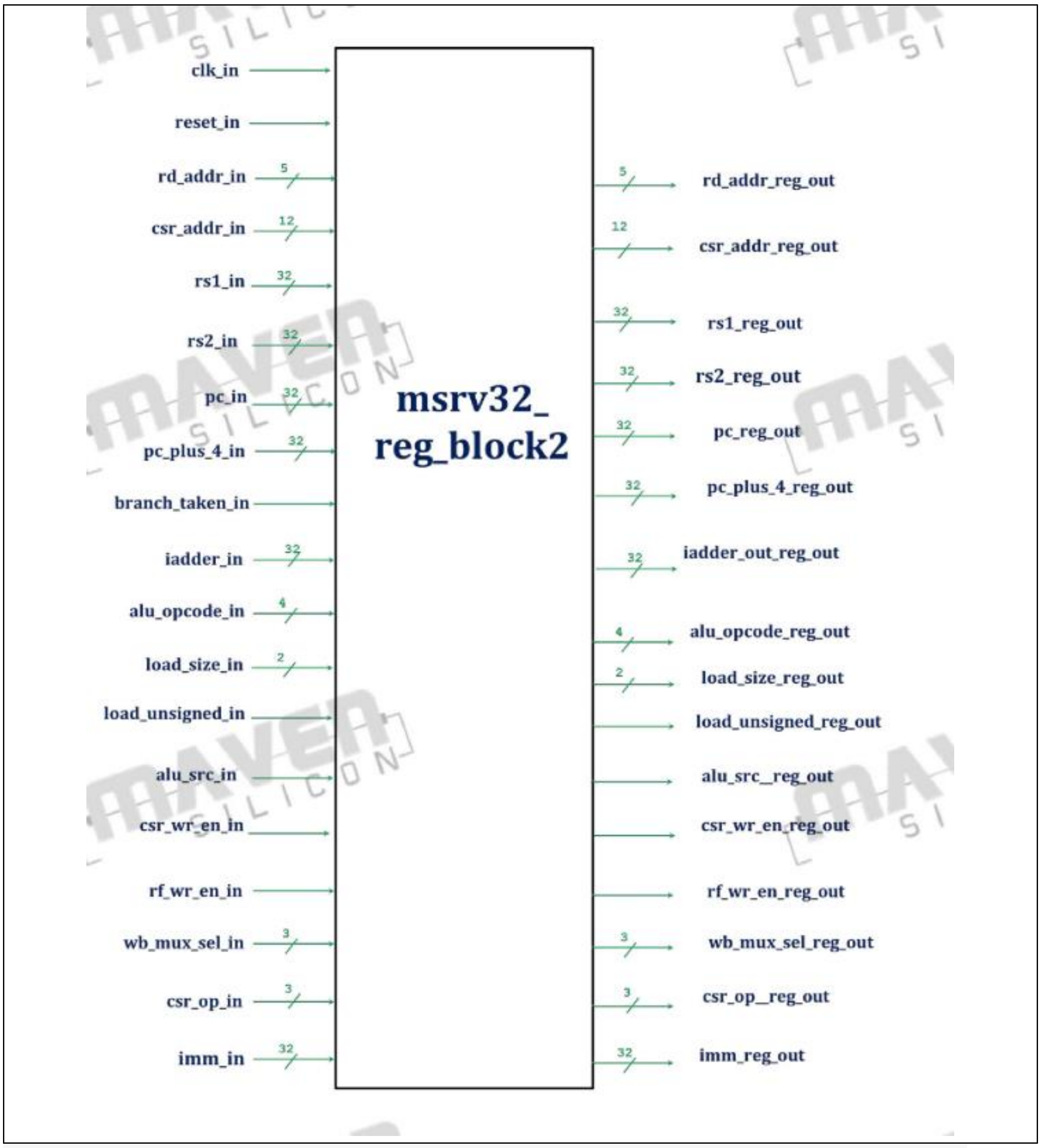
Decoder Functionality: The decoder decodes the input code and activates a single output line corresponding to the input code, allowing the selection or activation of a specific function or device among multiple options.

CSR FILE



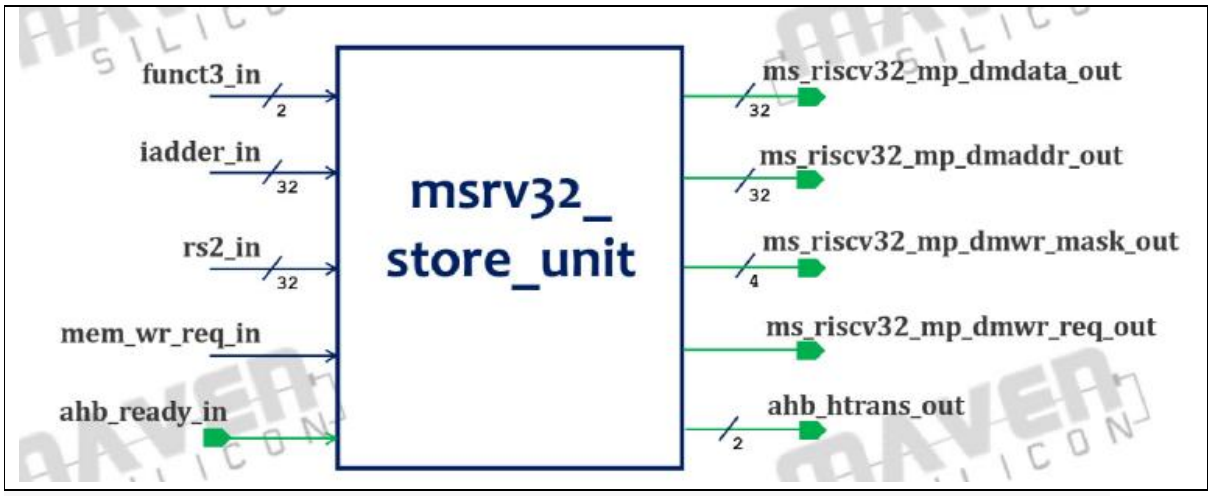
CSR (Control and Status Register) file is a special register file in a computer system that contains control and status information. It provides functionality for managing system-level operations and monitoring various aspects of the processor, such as exception handling, memory management, and privileged instructions.

REG BLOCK 2



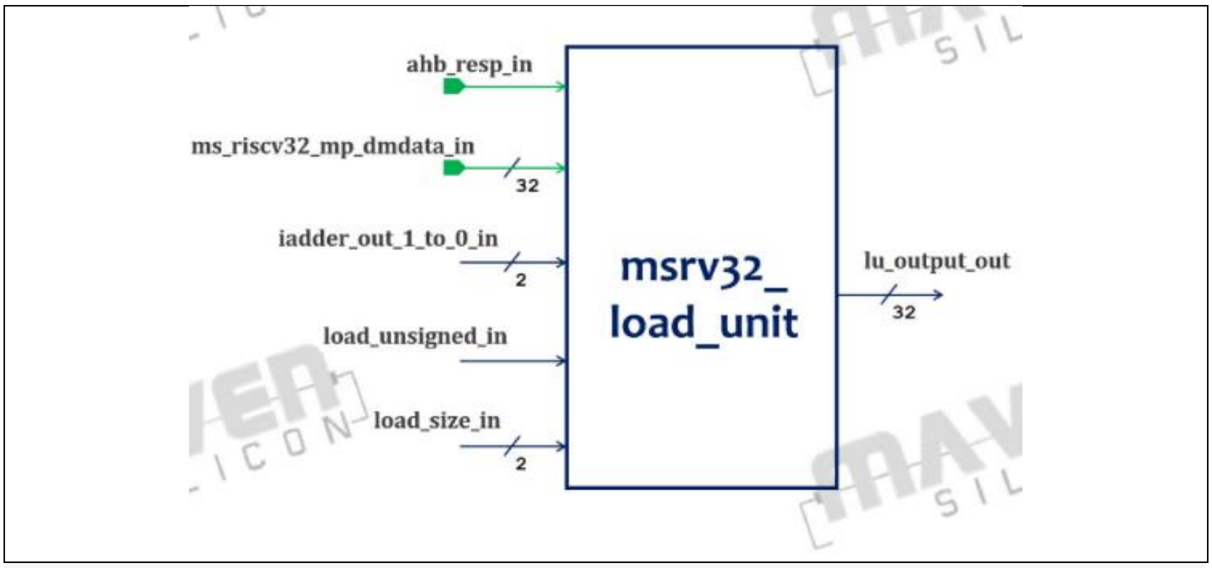
Themsrv32_reg_block_2 module is a register block in a processor design, responsible for storing and synchronizing register values. It uses clock edge-triggered flip-flops to update the register values based on the input signals during the rising edge of the clock signal.

STORE UNIT



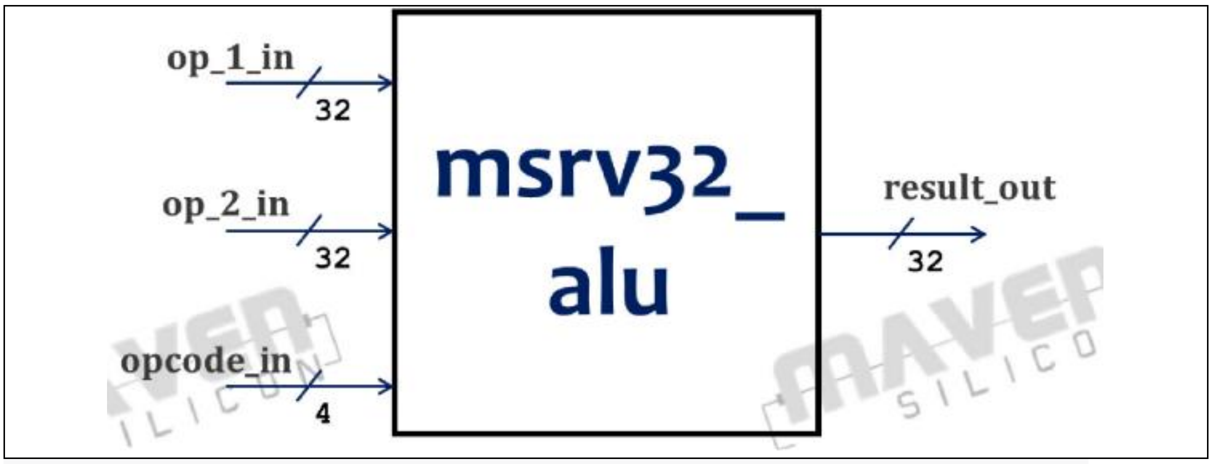
The store unit in a processor is responsible for handling store instructions, which store data from a register to memory. It calculates the memory address based on the instruction operands and data from the source register, and writes the data to the specified memory location.

LOAD UNIT



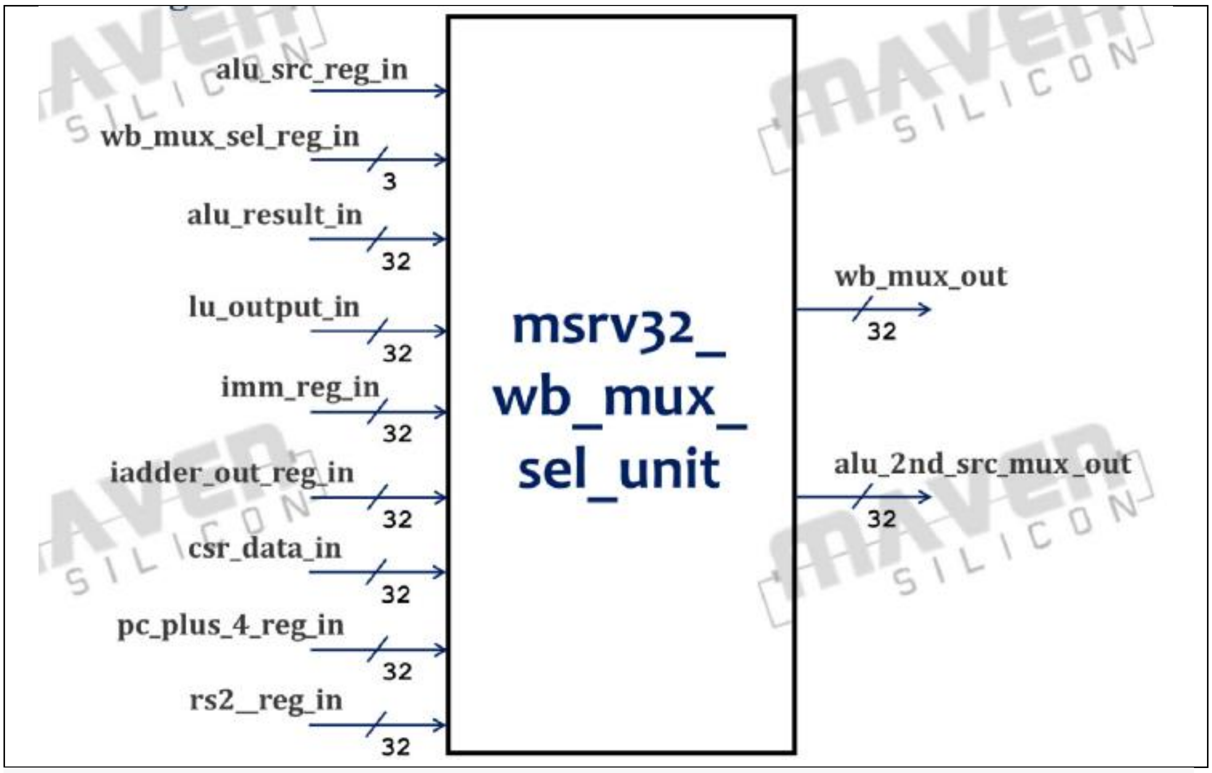
The load unit is responsible for handling memory load operations in a processor. It retrieves data from memory based on the specified memory address and forwards it to the appropriate destination register or pipeline stage for further processing or storage.

ALU



ALU stands for Arithmetic Logic Unit, which is a digital circuit within a processor that performs arithmetic and logical operations on data. It takes input operands and an operation code, performs the specified operation, and produces the result as output, supporting operations such as addition, subtraction, multiplication, logical AND/OR, shifting, and comparison.

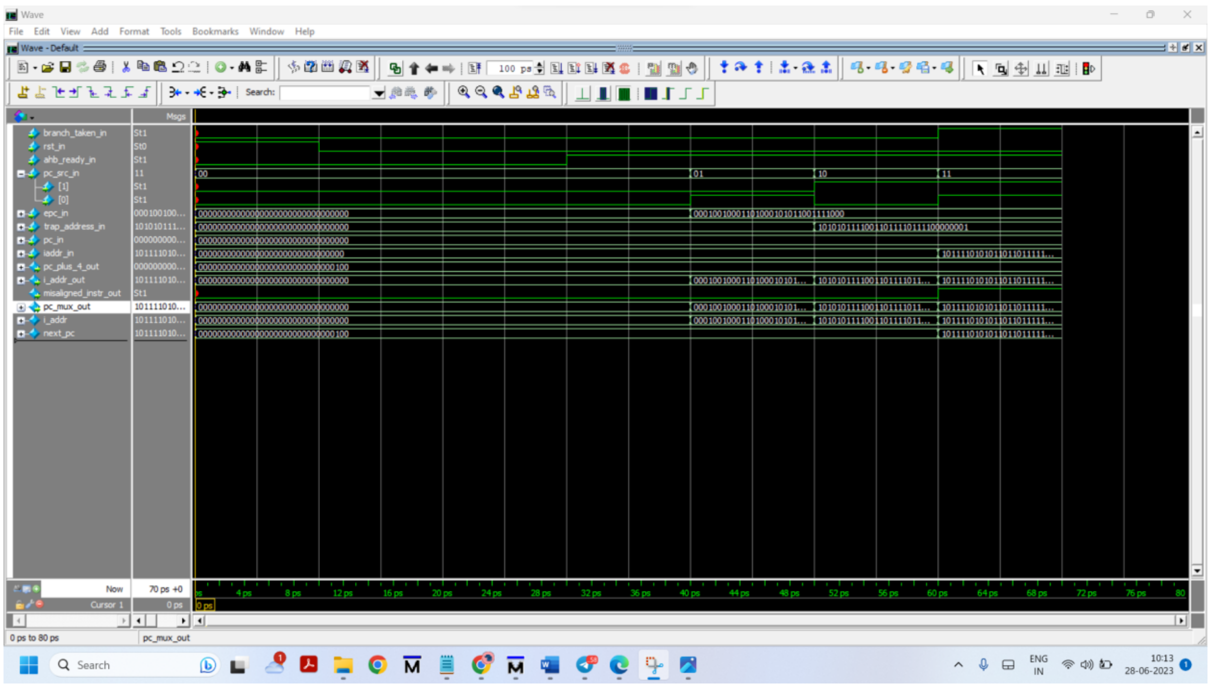
WB MUX SELECTION UNIT



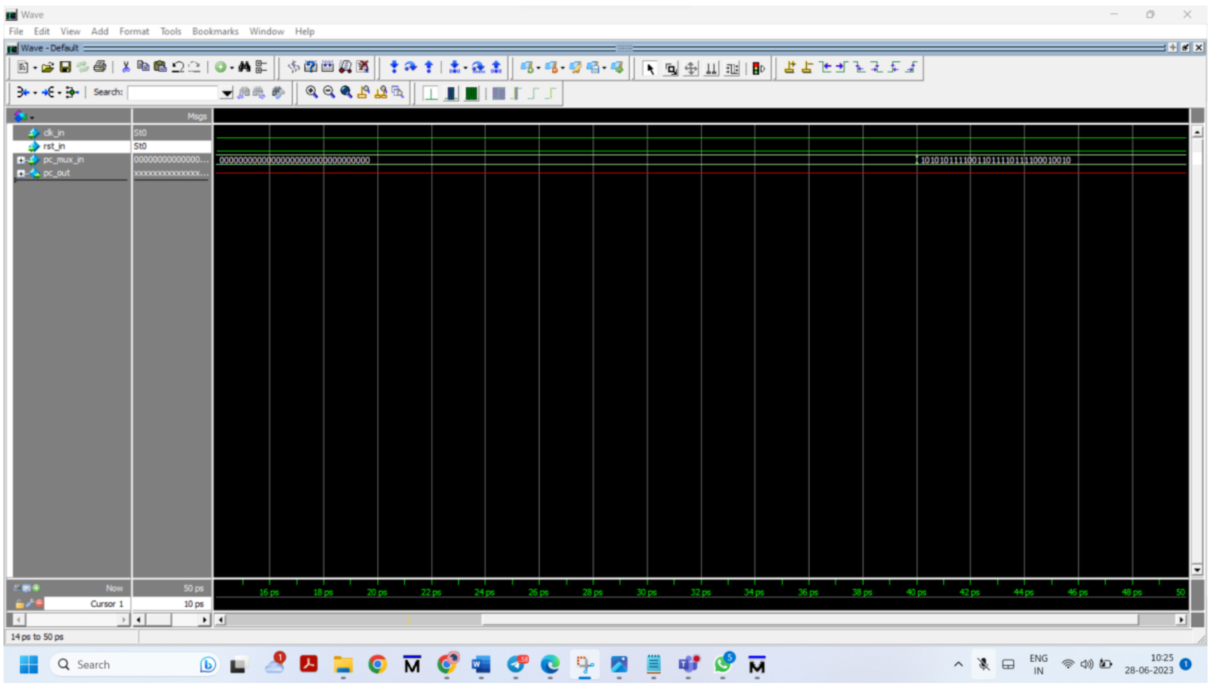
Its functionality is to select the appropriate data source for the write-back stage, typically choosing between the output of the ALU (Arithmetic Logic Unit) and the data read from memory, based on control signals such as the instruction opcode or operation type.

OUTPUT WAVEFORMS

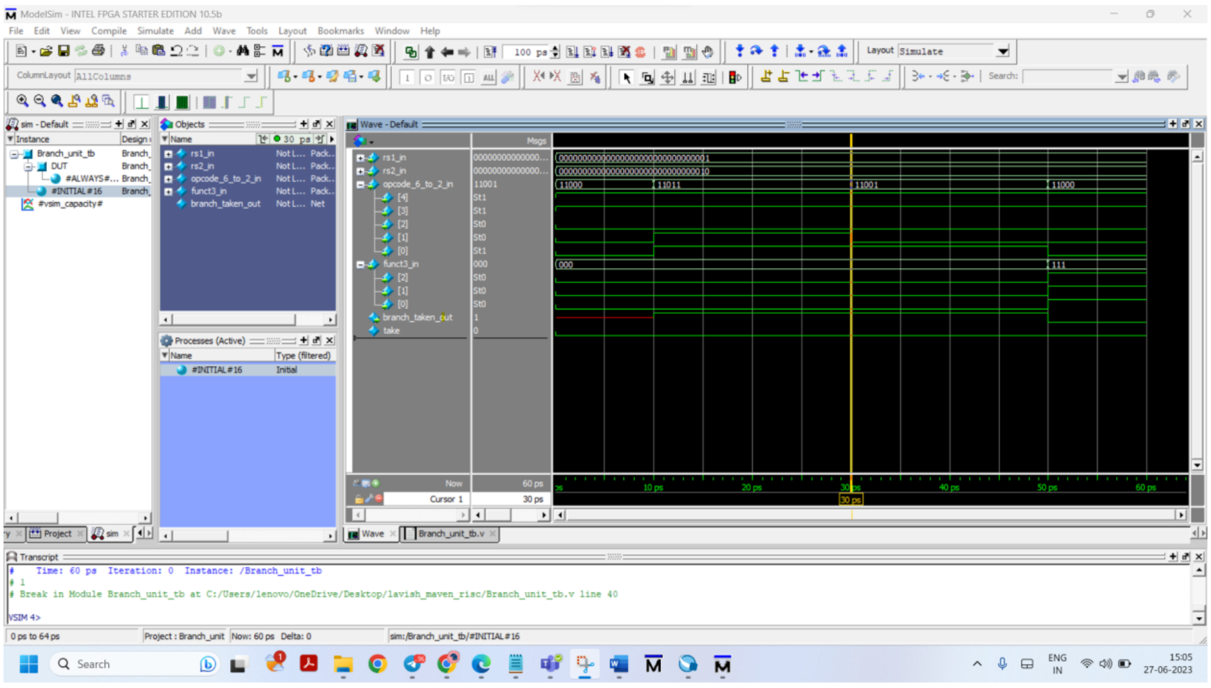
PC MUX:



REG_BLOCK_1:



BRANCH_UNIT:


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
sim - Default
Objects
Wave - Default
Processes (Active)
Transcript

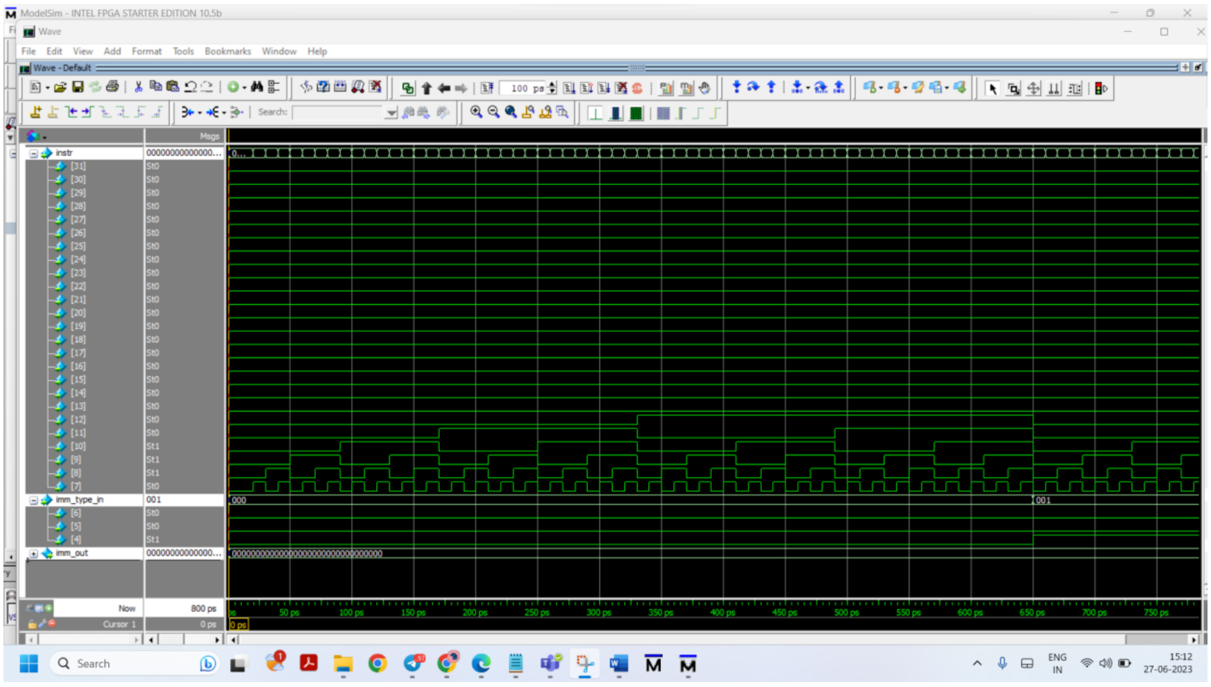
```
Instance          Design
Branch_unit_tb    Branch_
  DUT             Branch_
    #ALWAYS#...   Branch_
  #INITIAL#16     Branch_
#vsim_capacity#

Name               
rs1_in             Not L...  Pack...
rs2_in             Not L...  Pack...
opcode_6_to_2_in   Not L...  Pack...
funct3_in          Not L...  Pack...
branch_taken_out   Not L...  Net

Name           Type (filtered)
#INITIAL#16    Initial
```

```
                    Msgs
rs1_in              0000000000000...   00000000000000000000000000000001
rs2_in              0000000000000...   00000000000000000000000000000010
opcode_6_to_2_in    11001              11000  11011  11001  11000
  [4]               St1
  [3]               St1
  [2]               St0
  [1]               St0
  [0]               St1
funct3_in           000                000  111
  [2]               St0
  [1]               St0
  [0]               St0
branch_taken_out    1
take                0

Now         60 ps
Cursor 1    30 ps
```

```
# 	Time: 60 ps  Iteration: 0  Instance: /Branch_unit_tb
# 1
# Break in Module Branch_unit_tb at C:/Users/lenovo/OneDrive/Desktop/lavish_maven_risc/Branch_unit_tb.v line 40

VSIM 4>
```

```
0 ps to 64 ps    Project : Branch_unit    Now: 60 ps  Delta: 0    sim:/Branch_unit_tb/#INITIAL#16
```

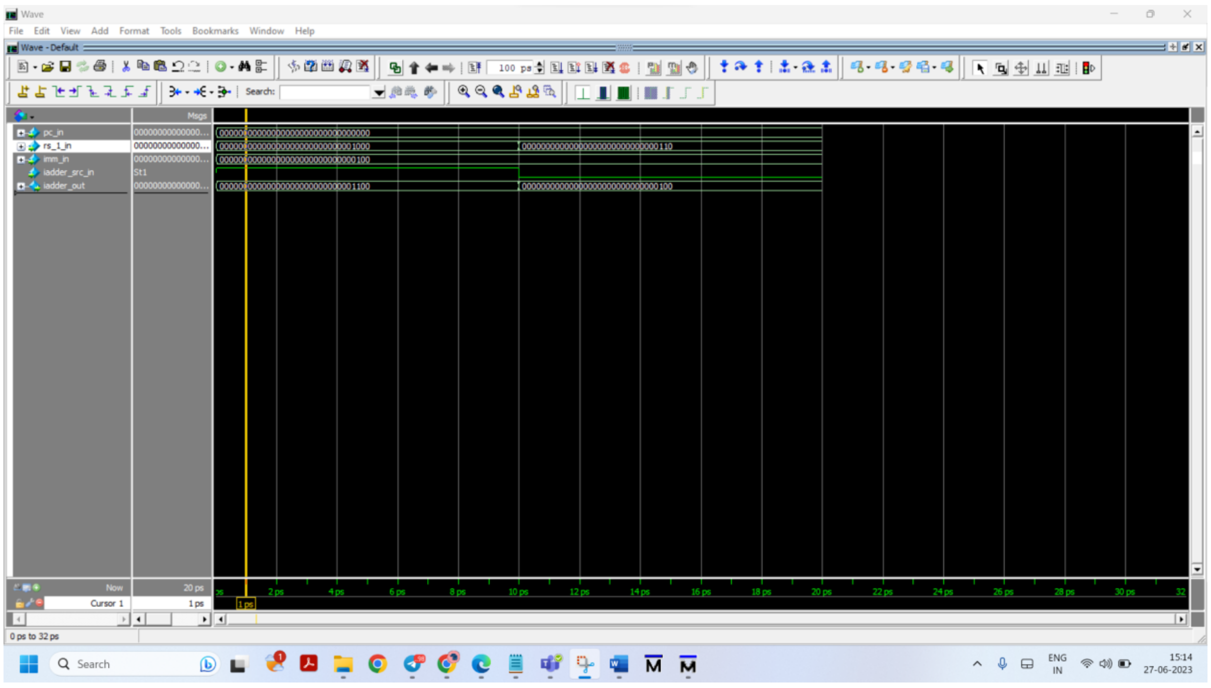
IMM_GENERATOR:


ModelSim - INTEL FPGA STARTER EDITION 10.5b
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:

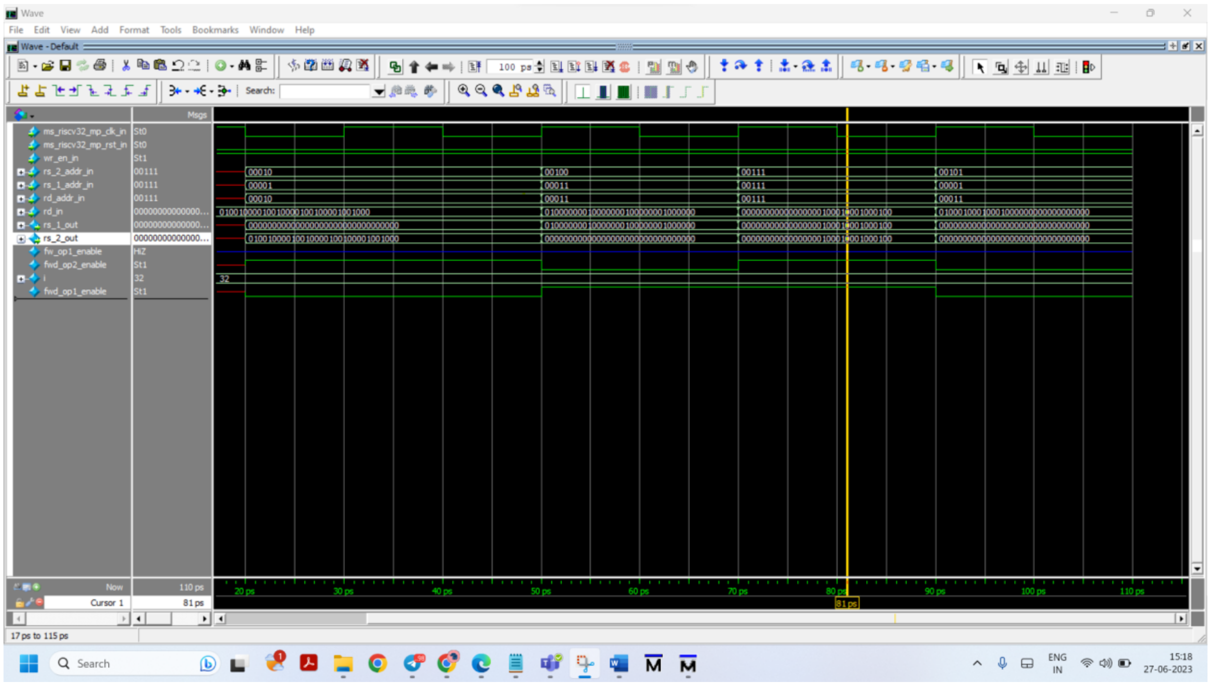
```
                 Msgs
instr            0000000000000...
  [31]           St0
  [30]           St0
  [29]           St0
  [28]           St0
  [27]           St0
  [26]           St0
  [25]           St0
  [24]           St0
  [23]           St0
  [22]           St0
  [21]           St0
  [20]           St0
  [19]           St0
  [18]           St0
  [17]           St0
  [16]           St0
  [15]           St0
  [14]           St0
  [13]           St0
  [12]           St0
  [11]           St0
  [10]           St1
  [9]            St1
  [8]            St1
  [7]            St0
imm_type_in      001                000  001
  [6]            St0
  [5]            St0
  [4]            St1
imm_out          0000000000000...   00000000000000000000000000000000

Now         800 ps
Cursor 1    0 ps
```

Search
ENG IN
15:12
27-06-2023

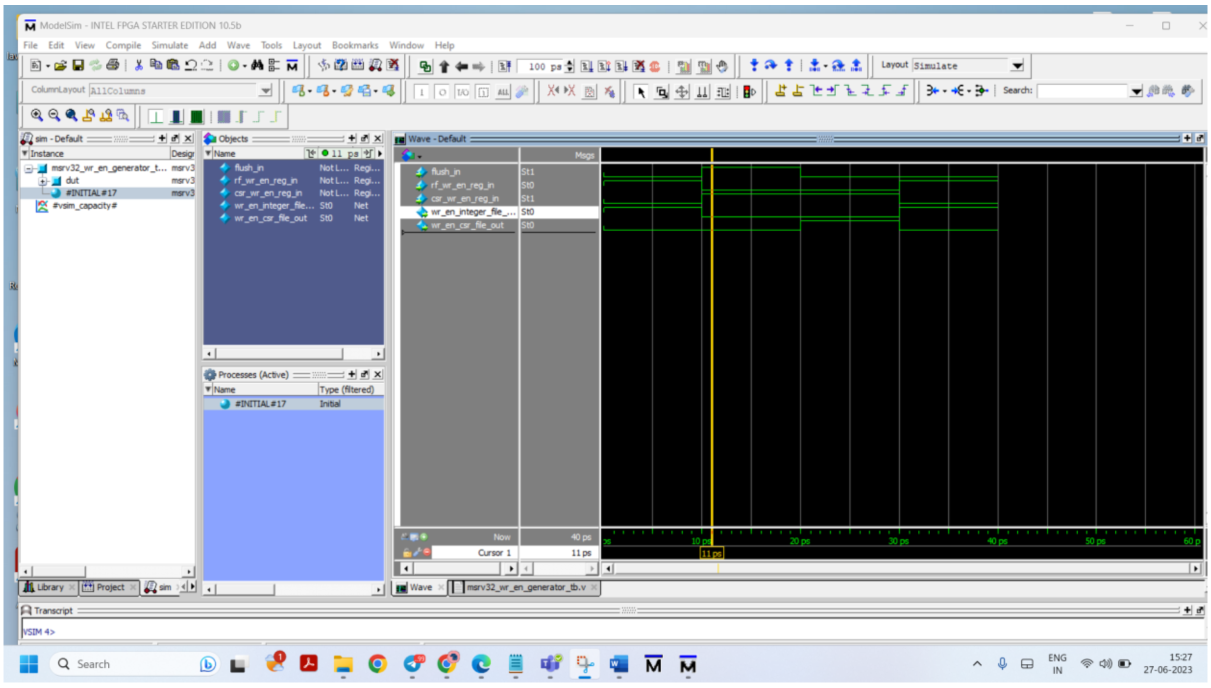
IMM_ADDER:



INTEGER_FILE:

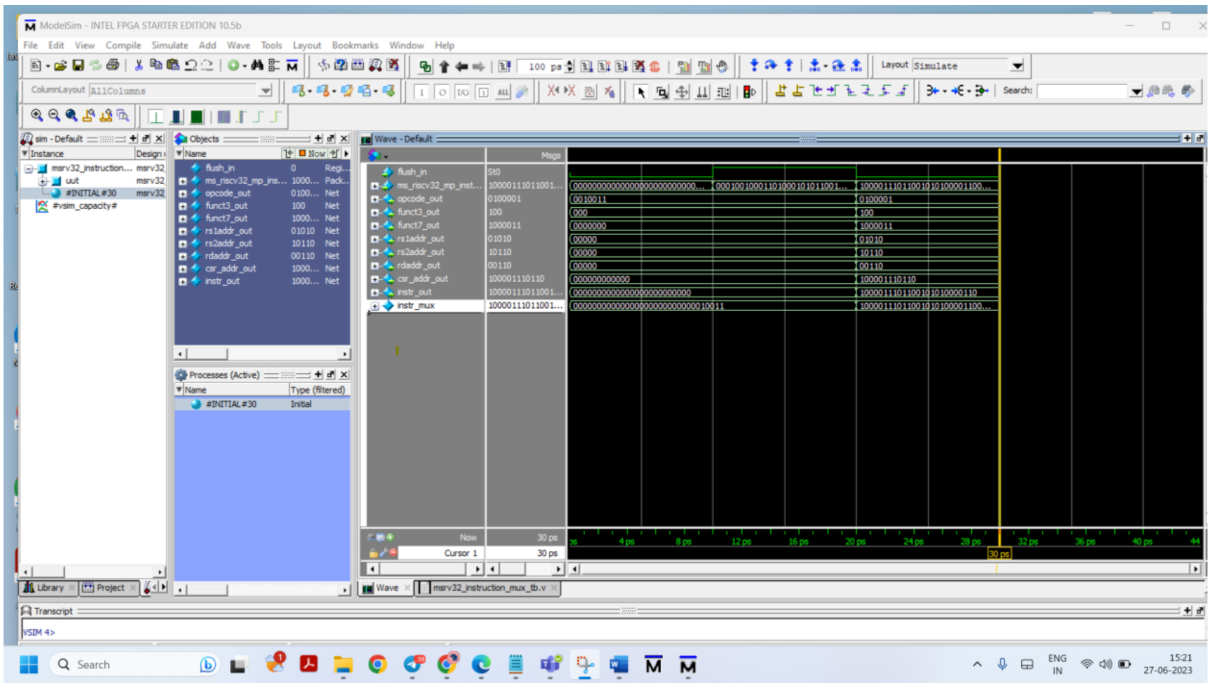


WR_EN_GENERATOR:


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
sim - Default
Objects
Wave - Default
Processes (Active)
Transcript
Library
Project
Wave
msrv32_wr_en_generator_tb.v

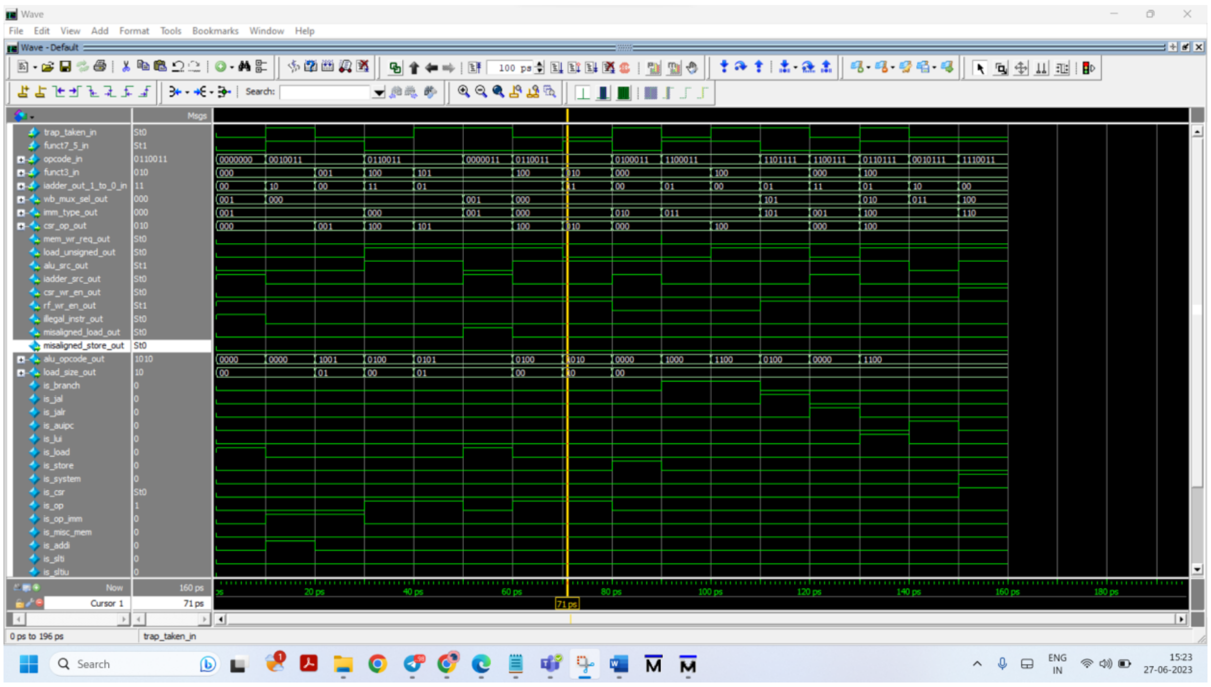
```
Instance                         Design
msrv32_wr_en_generator_t...      msrv3
  dut                            msrv3
  #INITIAL#17                    msrv3
#vsim_capacity#

Objects (11 ps)
flush_in                 Not L...  Regi...
rf_wr_en_reg_in          Not L...  Regi...
csr_wr_en_reg_in         Not L...  Regi...
wr_en_integer_file...    St0       Net
wr_en_csr_file_out       St0       Net

Wave - Default                   Msgs
flush_in                         St1
rf_wr_en_reg_in                  St0
csr_wr_en_reg_in                 St1
wr_en_integer_file_...           St0
wr_en_csr_file_out               St0

Now        40 ps
Cursor 1   11 ps

Processes (Active)
Name             Type (filtered)
#INITIAL#17      Initial

VSIM 4>
```

INSTRUCTION_MUX:


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
sim - Default
Objects
Wave - Default
Processes (Active)
Transcript
Library
Project
Wave
msrv32_instruction_mux_tb.v

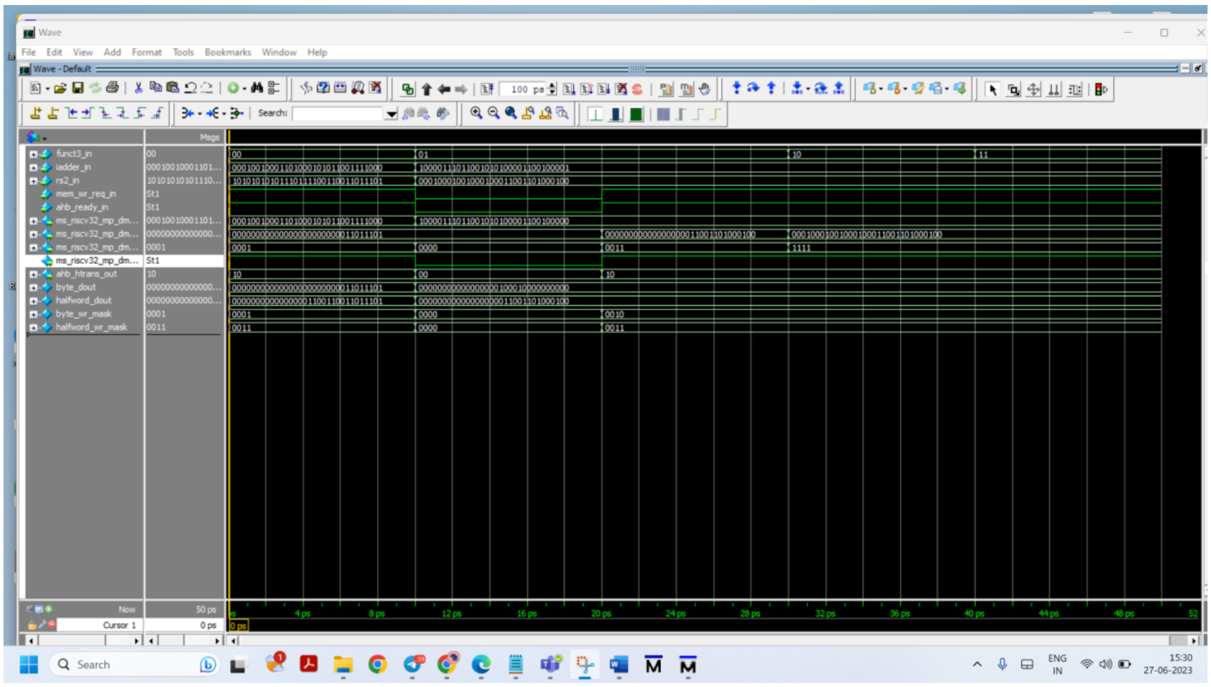
```
Instance                         Design
msrv32_instruction...            msrv32
  dut                            msrv32
  #INITIAL#30                    msrv32
#vsim_capacity#

Objects (Now)
flush_in                 0         Regi...
ms_riscv32_mp_ins...     1000...   Pack...
opcode_out               0100...   Net
funct3_out               100       Net
funct7_out               1000...   Net
rs1addr_out              01010     Net
rs2addr_out              10110     Net
rdaddr_out               00110     Net
csr_addr_out             1000...   Net
instr_out                1000...   Net

Wave - Default           Msgs
flush_in                 St0
ms_riscv32_mp_inst...    1000011101100 1...  | 0000000000000000000000000... | 0001001000110100010 1011001... | 10000111011001010100001100...
opcode_out               0100001             | 0010011                                                       | 0100001
funct3_out               100                 | 000                                                           | 100
funct7_out               1000011             | 0000000                                                       | 1000011
rs1addr_out              01010               | 00000                                                         | 01010
rs2addr_out              10110               | 00000                                                         | 10110
rdaddr_out               00110               | 00000                                                         | 00110
csr_addr_out             100001110110        | 000000000000                                                  | 100001110110
instr_out                10000111011001...   | 0000000000000000000000000                                     | 1000011101100101010000110
instr_mux                10000111011001...   | 000000000000000000000000000010011                             | 10000111011001010100001100...

Now        30 ps
Cursor 1   30 ps

Processes (Active)
Name             Type (filtered)
#INITIAL#30      Initial

VSIM 4>
```

DECODER:



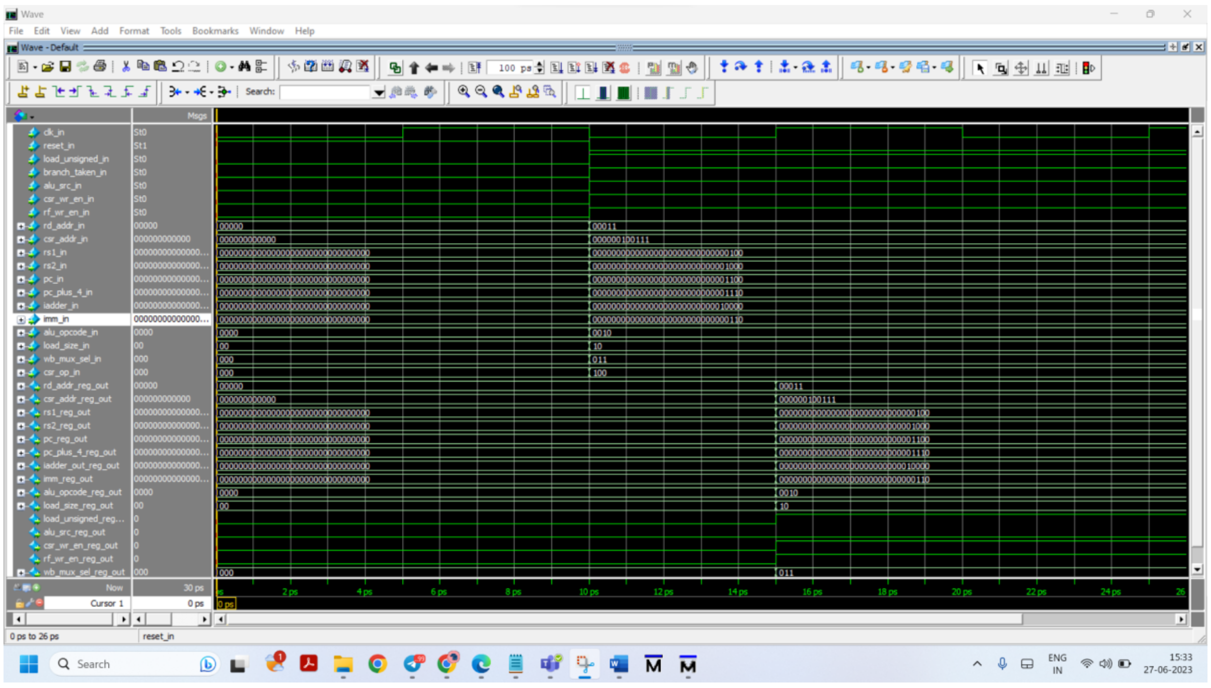
```
trap_taken_in            St0
funct7_5_in              St1
opcode_in                0110011
funct3_in                010
iadder_out_1_to_0_in     11
wb_mux_sel_out           000
imm_type_out             000
csr_op_out               010
mem_wr_req_out           St0
load_unsigned_out        St0
alu_src_out              St1
iadder_src_out           St0
csr_wr_en_out            St0
rf_wr_en_out             St1
illegal_instr_out        St0
misaligned_load_out      St0
misaligned_store_out     St0
alu_opcode_out           1010
load_size_out            10
is_branch                0
is_jal                   0
is_jalr                  0
is_auipc                 0
is_lui                   0
is_load                  0
is_store                 0
is_system                0
is_csr                   St0
is_op                    1
is_op_imm                0
is_misc_mem              0
is_addi                  0
is_slti                  0
is_sltiu                 0
Now                      160 ps
Cursor 1                 71 ps
0 ps to 196 ps           trap_taken_in
```

STORE_UNIT:

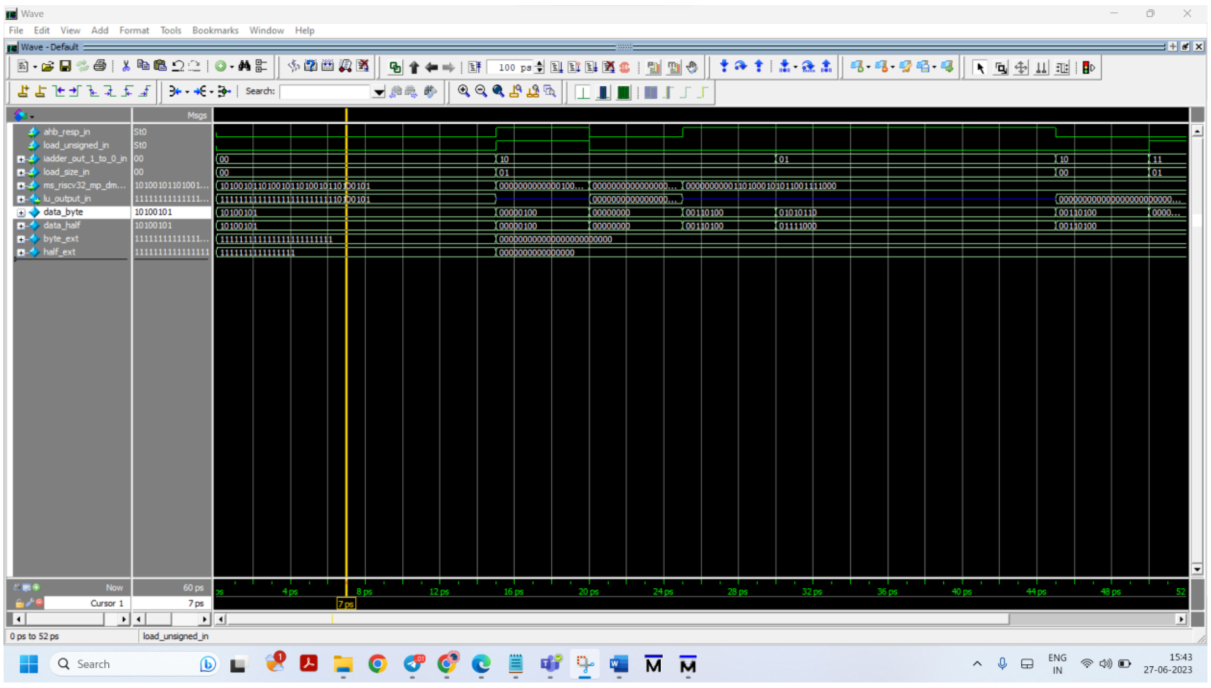


```
funct3_in                00
iadder_in                000100100011 01...
rs2_in                   1010101010 1110...
mem_wr_req_in            St1
ahb_ready_in             St1
ms_riscv32_mp_dm...      000100100011 01...
ms_riscv32_mp_dm...      000000000000000...
ms_riscv32_mp_dm...      0001
ms_riscv32_mp_dm...      St1
ahb_htrans_out           10
byte_dout                000000000000000...
halfword_dout            000000000000000...
byte_wr_mask             0001
halfword_wr_mask         0011
Now                      50 ps
Cursor 1                 0 ps
```

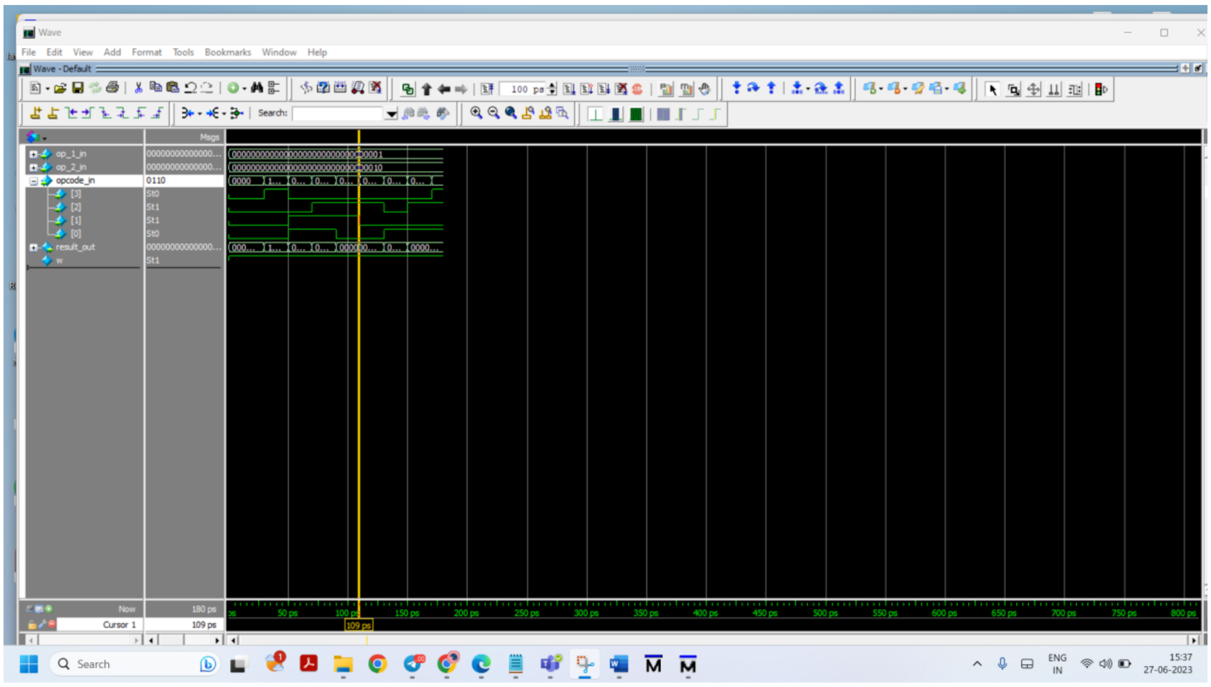
REG_BLOCK_2:



LOAD_UNIT:

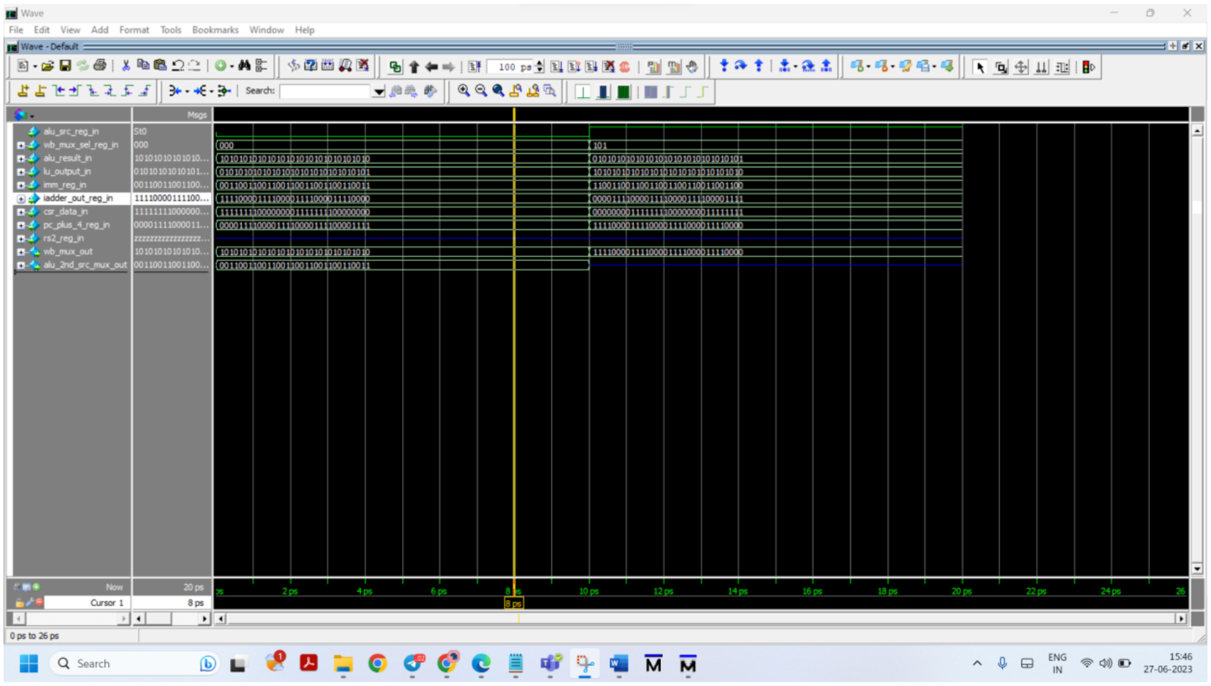


ALU:



```
op_1_in       000000000000000...   00000000000000000000000000000001
op_2_in       000000000000000...   00000000000000000000000000000010
opcode_in     0110                 0000  1...  0...  0...  0...  0...  0...  0...
  [3]         St0
  [2]         St1
  [1]         St1
  [0]         St0
result_out    000000000000000...   000...  1...  0...  0...  000000...  0...  0000...
w             St1
```

WB_MUX_SEL_UNIT:



```
alu_src_reg_in        St0
wb_mux_sel_reg_in     000                  000                                   101
alu_result_in         10101010101010...    10101010101010101010101010101010      01010101010101010101010101010101
lu_output_in          01010101010101...    01010101010101010101010101010101      10101010101010101010101010101010
imm_reg_in            00110011001100...    00110011001100110011001100110011      11001100110011001100110011001100
iadder_out_reg_in     11110000111100...    11110000111100001111000011110000      00001111000011110000111100001111
csr_data_in           11111111000000...    11111111000000001111111100000000      00000000111111110000000011111111
pc_plus_4_reg_in      00001111000011...    00001111000011110000111100001111      11110000111100001111000011110000
rs2_reg_in            zzzzzzzzzzzzzzzz...
wb_mux_out            10101010101010...    10101010101010101010101010101010      11110000111100001111000011110000
alu_2nd_src_mux_out   00110011001100...    00110011001100110011001100110011
```

THANK YOU