INTRODUCTION

The design and implementation of microprocessors have been at the forefront of computing advancements for decades. In recent years, the RISC-V instruction set architecture (ISA) has gained popularity due to its open-source nature and modular design. This project aims to design and implement a five-stage RISC-V RV32i microprocessor on an FPGA, specifically the MAX10 FPGA. The microprocessor solution will include 128 KiB of on-chip memory using M9K blocks, and programs will be streamed over the JTAG interface adn also an IO space(LED's and PUSH BUTTONS). The development of this microprocessor will provide a practical application of computer architecture principles, and will serve as a foundation for future research and experimentation in the field. In this report, we will discuss the design and implementation of the microprocessor, as well as the challenges and considerations that arose during the development process.

Theory Of Operation

PIPELINE STAGES:

1. Fetch: (IF)

In the Fetch stage, the processor fetches the instruction from memory. The instruction pointer (PC) is incremented to point to the next instruction in memory.

2. Decode: (ID)

In the Decode stage, the instruction is decoded and the necessary resources are allocated for executing the instruction. The operands are identified and loaded into the appropriate registers. Branch instructions are also executed in this stage, where the PC is updated to jump to the target address.

3. Execute: (ex)

In the Execute stage, the instruction is executed. The control signals are generated by using the forwarded instruction word, to determine which functional unit to use for execution. This involves performing arithmetic or logical operations on the operands or accessing memory to load or store data.

4. Memory: (mem)

In the Memory stage, data is read from or written to memory. This stage is used for load/store instructions that access memory. The data read from memory is stored in a temporary register and the data to be written to memory is sent to the memory unit.

5. Write back: (wb)

In the Writeback stage, the results of the instruction execution are written back to the register file. The data is stored in the destination register specified by the instruction.

Overall, the 5-stage pipeline of RISC-V architecture improves the performance of the processor by overlapping the execution of multiple instructions. By breaking the instruction execution into separate stages, each stage can be optimized to perform its specific task efficiently. This allows the processor to handle more instructions per clock cycle, resulting in faster execution times.

Handling Hazards:

This design handles all kinds of hazards such as data hazards and memory hazards .

Data Hazards:

Data Hazards are handled by forwarding the data and it's corresponding registers from ex ,mem and wb to the ID stage .The ID stage compares the current decoding register aganist all the forwarded registeers ,if there is any match between the current register and forwarded register then the respective forwarded data is given to ex block for execution.By this we solved the data hazard .The below code snippet explains it very well.

```
//data Hazard detection
//For Rs1
always @(*)
begin
    if((regif_rs1_reg == df_ex_reg) && df_ex_enable && (regif_rs1_reg !
        rs1_data_out_buff   <=  df_ex_data  ;
    else if((regif_rs1_reg == df_mem_reg) && df_mem_enable && (regif_rs
        rs1_data_out_buff   <=  df_mem_data ;
    else if((regif_rs1_reg == df_wb_reg) && df_wb_enable && (regif_rs1_
        rs1_data_out_buff   <=  df_wb_data  ;
    else
        rs1_data_out_buff   <=  regif_rs1_data              ;
end
//For Rs2
always @(*)
begin
    if((regif_rs2_reg == df_ex_reg) && df_ex_enable && (regif_rs2_reg !
        rs2_data_out_buff   <=  df_ex_data  ;
    else if((regif_rs2_reg == df_mem_reg) && df_mem_enable && (regif_rs
        rs2_data_out_buff   <=  df_mem_data ;
```

Memory Hazard:

Memory Hazard caused due to the data dependency between the load instruction followed by any ex block related instruction or branch instruction .This hazard is solved by stalling the pipeline. In my design the branch decoding and target address calculation is done in ID stage .So, if load is followed by arithmetic instruction the **one stall** is introduced in the pipeline or if the load is followed by branch instruction then **two stalls** is introduced.

```
always @(posedge clk)
begin
    if(jump_enable_rsg_del || lw_stall)
    begin
        iw_out      <=  32'h00000013    ;
        wb_reg_out  <=  5'd0                ;
    end
    else
    begin
        iw_out      <=  iw_in               ;
        wb_reg_out  <=  iw_in[11:7]     ;
```

Introducing stall is nothing but adding NOP instruction in between the instructions.

State of the Project

At present, the processor is tested by compiling C code (LED_toggle.c,stop_go.c) using riscv_gcc compiler and it is working .This processor works at 10 MHz clock rate and the future scope of this project is to increasing the clock frequency and adding branch prediction support .

RESULTS

Signal Tap Analyzer Pictures: