DELHI TECHNOLOGICAL UNIVERSITY

Computer Architecture(CA) EC-208
PROJECT REPORT

TOPIC : MIPS-32 Pipeline Processor

SUBMITTED TO : Prof. Dinesh Kumar

SUBMITTED BY :

1. **Anshul (2K19/EC/022)**
2. **Abhay Lakhotra (2K19/EC/006)**

ABSTRACT

This project presents the design and implement a basic five stage pipelined processor MIPS-32 (Microprocessor without Interlocked Pipeline Stages). This processor architecture consists of blocks like memory unit, controlling unit, program counter, adder, sign expanded, multiplexers, data memory and ALU. Particular attention will be paid to the reduction of clock cycles for lower instruction latency.

INTRODUCTION

MIPS-32 is a popular *reduced instruction set architecture (RISC) processor*. It is a 32-bit processor (it can operate on 32 bits of data at a time). The MIPS processor structured depends on the RISC processor. It has 32 registers. The processor utilizes the 5 pipeline stages, which are Instruction Fetch (IF), Instruction Decoder (ID), Execute (EX), Memory Access (MEM) and Write Back (WB). The pipelined MIPS processor plays out every one of the stages in various clock cycles.

RISC Instruction Set will consist of fewer number of instructions, simpler instructions and large number of registers mostly general purpose registers. The result is that it's much easier to implement these kind of processors in hardware.

Instruction execution cycle is divide into 5 steps :

1) **IF** : Instruction Fetch
2) **ID** : Instruction Decode / Register Fetch
3) **EX** : Execution/ Effective Address Calculation
4) **MEM** : Memory Access/ Branch Completion
5) **WB** : Register Write-back

The ***Instruction Fetch stage*** is where a program counter will pull the next instruction from the correct location in program memory. In addition the program counter was updated with either the next instruction location sequentially, or the instruction location as determined by a branch.

- Here the instruction pointed to by *PC* is fetched from memory, and also the next value of *PC* is computed.
- Every MIPS32 instruction is of 32 bits.
- Every memory word is of 32 bits and has a unique address.
- For a branch instruction, new value of PC may be the target address. So, PC is not updated in this stage; new value is stored in a register *NPC*.

IF :- IR <- Mem [PC];

NPC <- PC + 1;

The ***Instruction Decode stage*** is where the control unit determines what values the control lines must be set to depending on the instruction. In addition, hazard detection is implemented in this stage, and all necessary values are fetched from the register banks.

- The instruction already fetched in *IR* is decoded.
- Decoding is done in parallel with reading the register operands rs and rt.
- In a similar way, the immediate data are sign-extended.

ID :- A <- Reg[rs];

B <- Reg[rt];

Imm <- (IR15)^16 ## IR15-0

Here, A and B are temporary registers which will be required later.

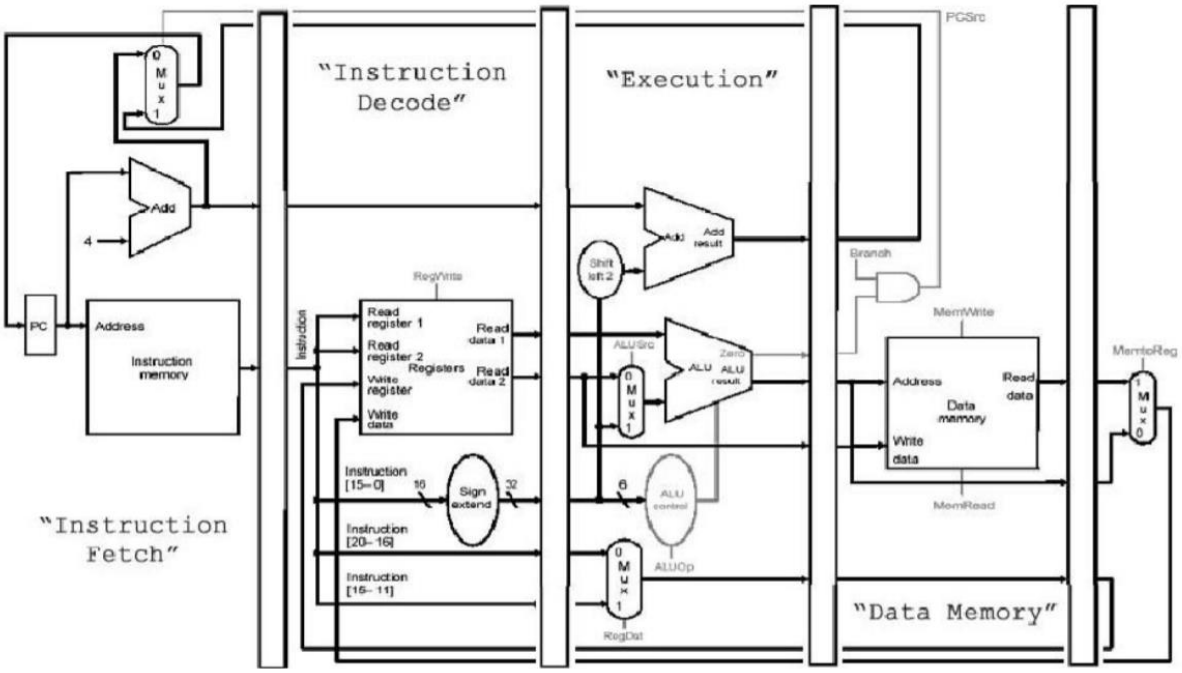
The ***Execute stage*** is where the instruction is actually sent to the ALU and executed. If necessary, branch locations are calculated in this stage as well.

Additionally, this is the stage where the forwarding unit will determine whether the output of the ALU or the memory unit should be forwarded to the ALU's inputs.

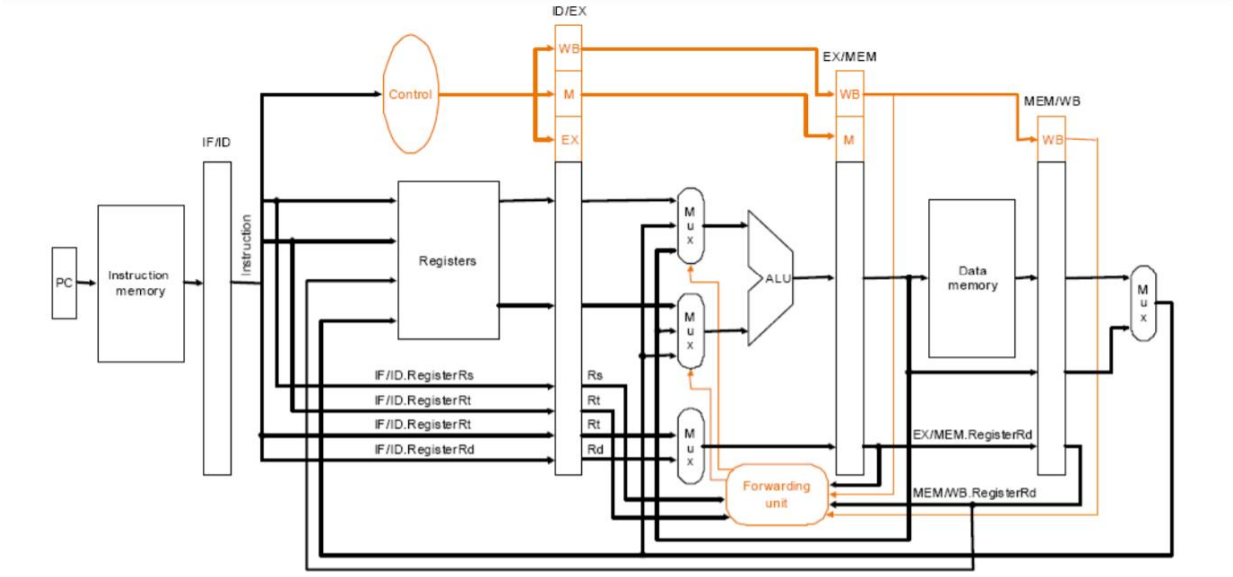
The **Memory Access stage** is where, if necessary, system memory is accessed for data. Also, if a write to data memory is required by the instruction it is done in this stage. In order to avoid additional complications it is assumed that a single read or write is accomplished within a single CPU clock cycle.

Finally, the ***Write Back stage*** is where any calculated values are written back to their proper registers. The write back to the register bank occurs during the first half of the cycle in order to avoid structural and data hazards if this was not the case.

The CPU included a hazard detection unit to determine when a stall cycle must be added. Due to data forwarding, this will only happen when a value is used immediately after being loaded from memory, or when a branch occurs.



MIPS pipeline processor



Forwarding Unit

Implementation

- The instruction fetch stage has multiple responsibilities in that it must properly update the CPU's program counter in the normal case as well as the branch instruction case. The instruction fetch stage is also responsible for reading the instruction memory and sending the current instruction to the next stage in the pipeline, or a stall if a branch has been detected in order to avoid incorrect execution. The instruction fetch stage is composed of three components: instruction memory, program counter, and the instruction address adder. The instruction memory also takes inputs from the outside world that allow the loading of instruction memory for later execution.

- The instruction memory unit was designed to model a small amount of cache and therefore was made to be accessed within a single CPU cycle.

The instruction memory was sized at 1k bits and could therefore at maximum contain 32 separate instructions.

- The Decode Stage is the stage of the CPU's pipeline where the fetched instruction is decoded, and values are fetched from the register bank. It is responsible for mapping the different sections of the instruction into their proper representations (based on R or I type instructions). The Decode stage consists of the Control unit, the Hazard Detection Unit, the Sign Extender, and the Register bank, and is responsible for connecting all of these components together. It splits the instruction into its various parts and feeds them to the corresponding components. Regisers Rs and Rt are fed to the register bank, the immediate section is fed to the sign extender, and the ALU opcode and function codes are sent to the control unit. The outputs of these corresponding components are then clocked and stored for the next stage.

- The hazard detection unit monitors output from the execute stage to determine hazard conditions. Hazards occur when we read a value that was just written from memory, as the value won't be available for forwarding until the end of the memory stage, and when we branch. The hazard detection unit will introduce a stall cycle by replacing the control lines with 0s, and disabling the program counter from updating. When a branch is detected the hazard detection unit will allow the PC to write, but will feed it the branch address instead of the next counted value. The sign extender is responsible for two functions. It takes the immediate value and sign extends it if the current instruction is a signed operation. It also has a shifted output for branches.

- The execute stage is responsible for taking the data and actually performing the specified operation on it. The execute stage consists of an ALU, a Determine Branch unit, and a Forwarding Unit. The execute stage connects these components together so that the ALU will process the data properly, given inputs chosen by the forwarding unit, and will notify the decode stage if a branch is indeed to be taken.

- The alu is responsible for performing the actual calculations specified by the instruction. It takes two 32 bit inputs and some control signals, and gives a single 32 bit output along with some information about the output – whether it is zero or negative. The forwarding unit is responsible for choosing what input is to be fed into the ALU. It takes the input from the decode stage, the value that the alu has fed to the Memory stage, and the value that the Alu has fed to the write back stage, as well as the register numbers corresponding to all of these, and determines if any conflicts exist. It will choose which of these values must be sent to the ALU.

- The Memory stage is responsible for taking the output of the alu and committing it to the proper memory location if the instruction is a store. The memory stage contains one component: the data_memory object. It connects the data memory to a register bank for the write back stage to read, and also forwards on information about the current write back register. This register's number and calculated value are fed back to the forwarding unit in the execute stage to allow it to determine which value to pass to the ALU.

- The writeback stage is responsible for writing the calculated value back to the proper register. It has input control lines that tell it whether this instruction writes back or not, and whether it writes back ALU output or Data memory output. It then chooses one of these outputs and feeds it to the register bank based on these control lines.

Verilog code for Instruction fetch and Instruction decode cycle:

```
module pipe_MIPS32 (clk1, clk2);

input clk1, clk2;

reg [31:0] PC, IF_ID_IR, IF_ID_NPC;

reg [31:0] ID_EX_IR, ID_EX_NPC, ID_EX_A, ID_EX_B, ID_EX_Imm;

reg [2:0]  ID_EX_type, EX_MEM_type, MEM_WB_type;
```

```
reg [31:0] EX_MEM_IR, EX_MEM_ALUOut, EX_MEM_B;

reg       EX_MEM_cond;

reg [31:0] MEM_WB_IR, MEM_WB_ALUOut, MEM_WB_LMD;

reg [31:0] Reg [0:31];

reg [31:0] Mem [0:1023];

parameter ADD=6'b000000, SUB=6'b000001, AND=6'b000010, OR=6'b000011, SLT=6'b000100,
MUL=6'b000101, HLT=6'b111111,

        LW=6'b001000, SW=6'b001001, ADDI=6'b001010, SUBI=6'b001011, SLTI=6'b001100,
BNEQZ=6'b001101, BEQZ=6'b001110;

parameter RR_ALU=3'b000, RM_ALU=3'b001, LOAD=3'b010, STORE=3'b011, BRANCH=3'b100,
HALT=3'b101;

reg HALTED;

reg TAKEN_BRANCH;


always @(posedge clk1)        //IF

if (HALTED == 0)

begin

if (((EX_MEM_IR[31:26] == BEQZ) && (EX_MEM_cond == 1))

  ((EX_MUM_IR[31:26] == BNEQZ) && (EX_MEM_cond == 0)))

begin

IF_ID_IR <= #2 Mem[EX_MEM_ALUOut];

TAKEN_BRANCH <= #2 1'b1;

IF_ID_NPC <= #2 EX_MEM_ALUOut + 1;

PC <= #2 EX_MEM_ALUOut +1;

end

else

begin

IF_ID_IR <= #2 Mem[PC];
```

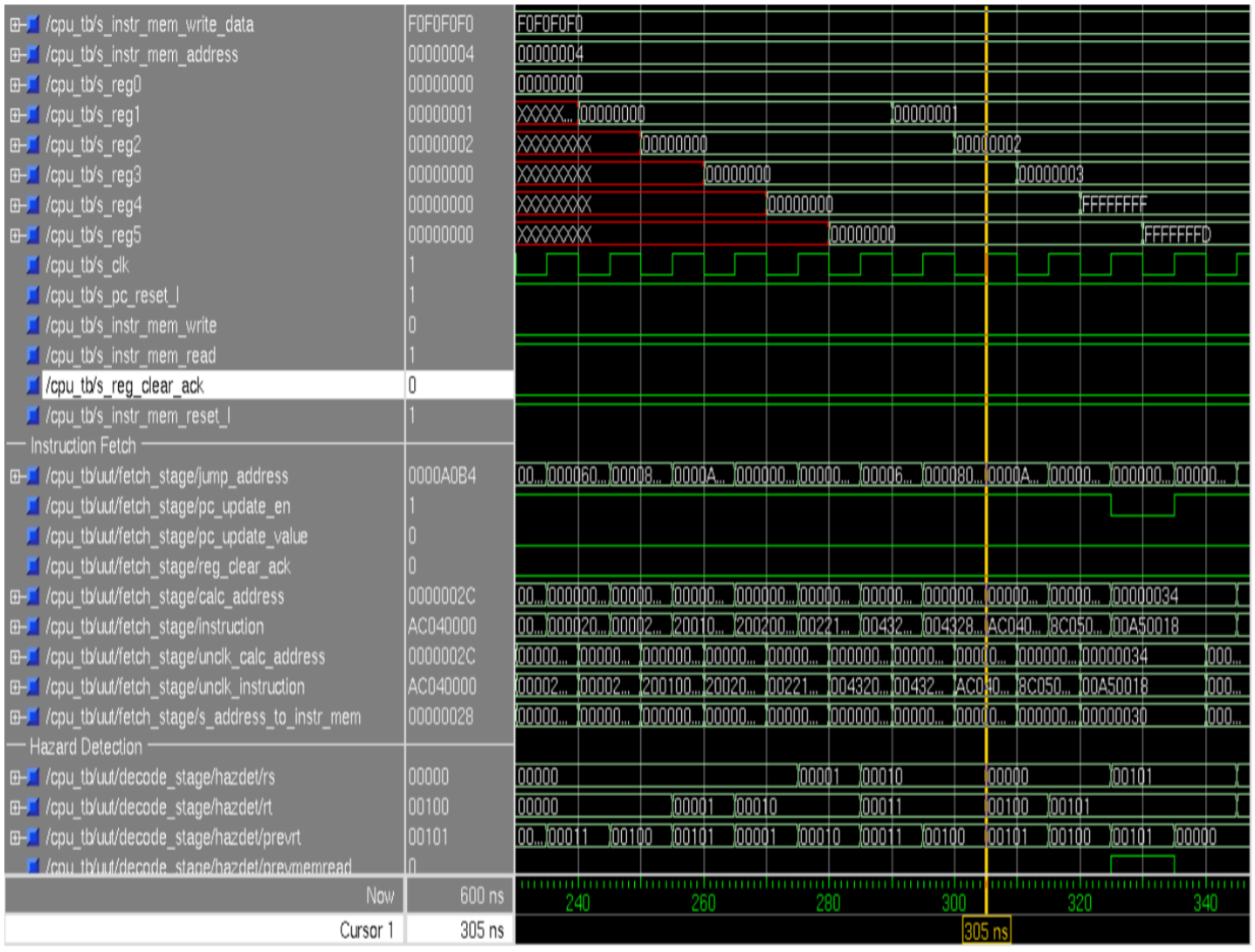
```
IF_ID_NPC <= #2 PC + 1;
PC <= #2 PC + 1;
end
end

always @(posedge clk2)       //ID
 if (HALTED == 0)
begin
 if (IF_ID_IR[25;21] == 5'b00000 ) ID_EX_A <= 0;
else ID_EX_A <= #2 Reg[IF_ID_IR[25:21]];

if (IF_ID_IR[20:16] == 5'b00000) ID_EX_B <= 0;
else ID_EX_B <= #2 Reg[IF_ID_IR[20:16]];

ID_EX_NPC <= #2 IF_ID_NPC;
ID_EX_IR <= #2 IF_ID_IR;
ID_EX_Imm <= #2 ((16(IF_ID_IR[15])), (IF_ID_IR[15:0]));

case (IF_ID_IR[31:26])
ADD, SUB, AND, OR, SLT, MUL : ID_EX_type <= #2 RR_ALU;
ADDI, SUBI, SLTI :  #2 RM_ALU;
LW : #2 LOAD;
SW : #2 STORE;
BEQZ, BNEQZ : #2 BRANCH;
HLT : #2 HALT;
default : #2 HALT;
```

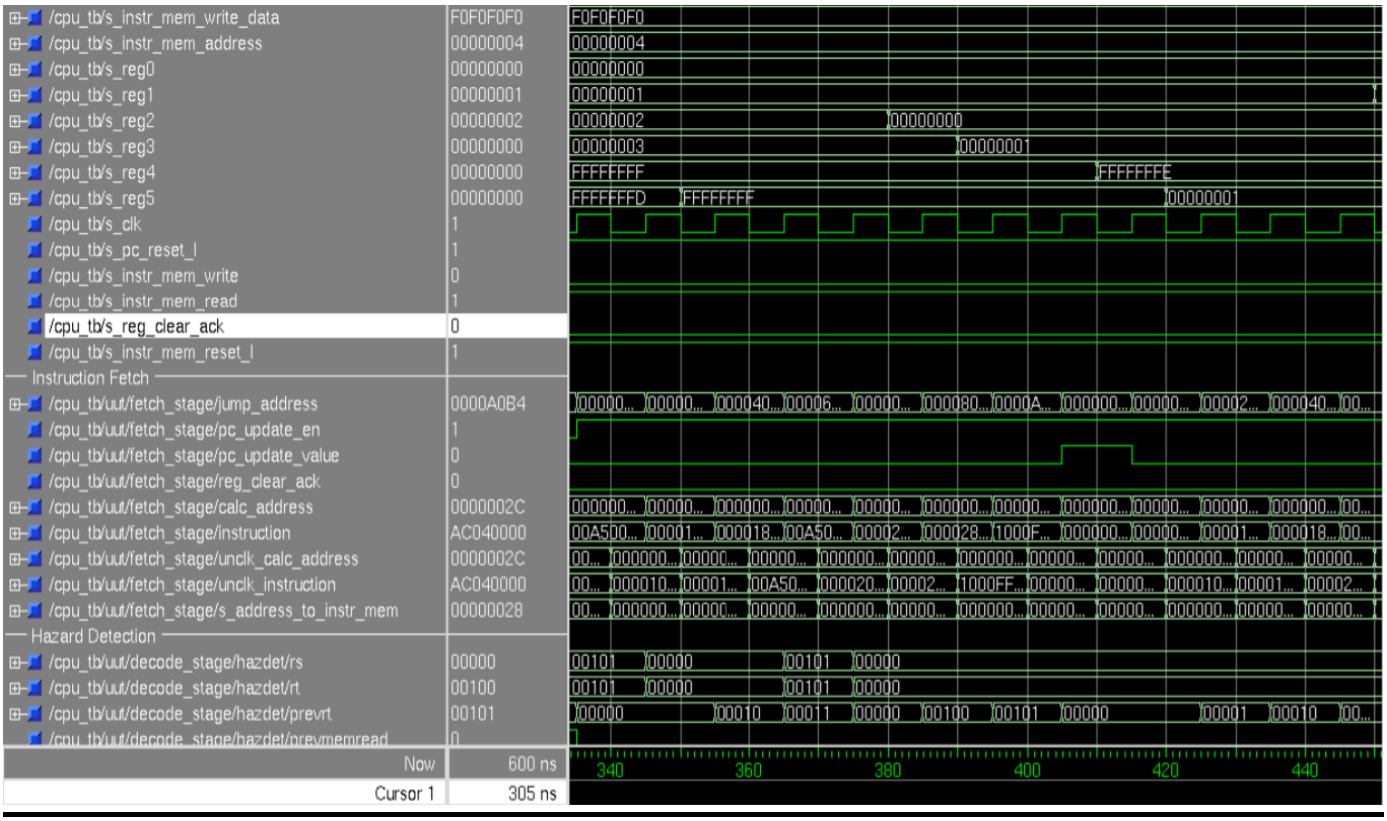
```
endcase
end
```

SIMULATION RESULTS

For simulation, a number of instructions were fed and the outputs of registers 0 through 5 were monitored. The instructions that were tested included register based and immediate adds, subtracts (both signed and unsigned), multiplication (signed and unsigned), reading and writing data memory, and a loop that would force the CPU to jump back to the start of instruction memory and execute those same instructions again. The different adds were important because each enables different parts including the data forwarding unit, multiple registers and different functions within the ALU itself.

Simulation Waveforms


/cpu_tb/s_instr_mem_write_data
/cpu_tb/s_instr_mem_address
/cpu_tb/s_reg0
/cpu_tb/s_reg1
/cpu_tb/s_reg2
/cpu_tb/s_reg3
/cpu_tb/s_reg4
/cpu_tb/s_reg5
/cpu_tb/s_clk
/cpu_tb/s_pc_reset_l
/cpu_tb/s_instr_mem_write
/cpu_tb/s_instr_mem_read
/cpu_tb/s_reg_clear_ack
/cpu_tb/s_instr_mem_reset_l
Instruction Fetch
/cpu_tb/uut/fetch_stage/jump_address
/cpu_tb/uut/fetch_stage/pc_update_en
/cpu_tb/uut/fetch_stage/pc_update_value
/cpu_tb/uut/fetch_stage/reg_clear_ack
/cpu_tb/uut/fetch_stage/calc_address
/cpu_tb/uut/fetch_stage/instruction
/cpu_tb/uut/fetch_stage/unclk_calc_address
/cpu_tb/uut/fetch_stage/unclk_instruction
/cpu_tb/uut/fetch_stage/s_address_to_instr_mem
Hazard Detection
/cpu_tb/uut/decode_stage/hazdet/rs
/cpu_tb/uut/decode_stage/hazdet/rt
/cpu_tb/uut/decode_stage/hazdet/prevrt
/cpu_tb/uut/decode_stage/hazdet/prevmemread
Now
Cursor 1
240
260
280
300
320
340
305 ns



Conclusion

MIPS processor is a widely used RISC processor in industry. In this project, we have successfully designed and synthesized a basic model of pipelined MIPS processor. The design has been modeled in Verilog.

References

1. https://www.rroij.com/open-access/design-of-bit-risc-cpu-based-on-mips-20-24.pdf

2. https://www.researchgate.net/publication/312581426_VHDL_Design_and_Simulation_of_a_32_Bit_MIPS_RISC_Processor
3. https://www.youtube.com/watch?v=NCrlyaXMAn8&list=PLJ5C_6qdAvBELELTSPgzYkQg3HgclQh-5
4. https://www.youtube.com/watch?v=8xS1D8xYUoE&t=202s
5. https://en.wikipedia.org/wiki/MIPS_architecture