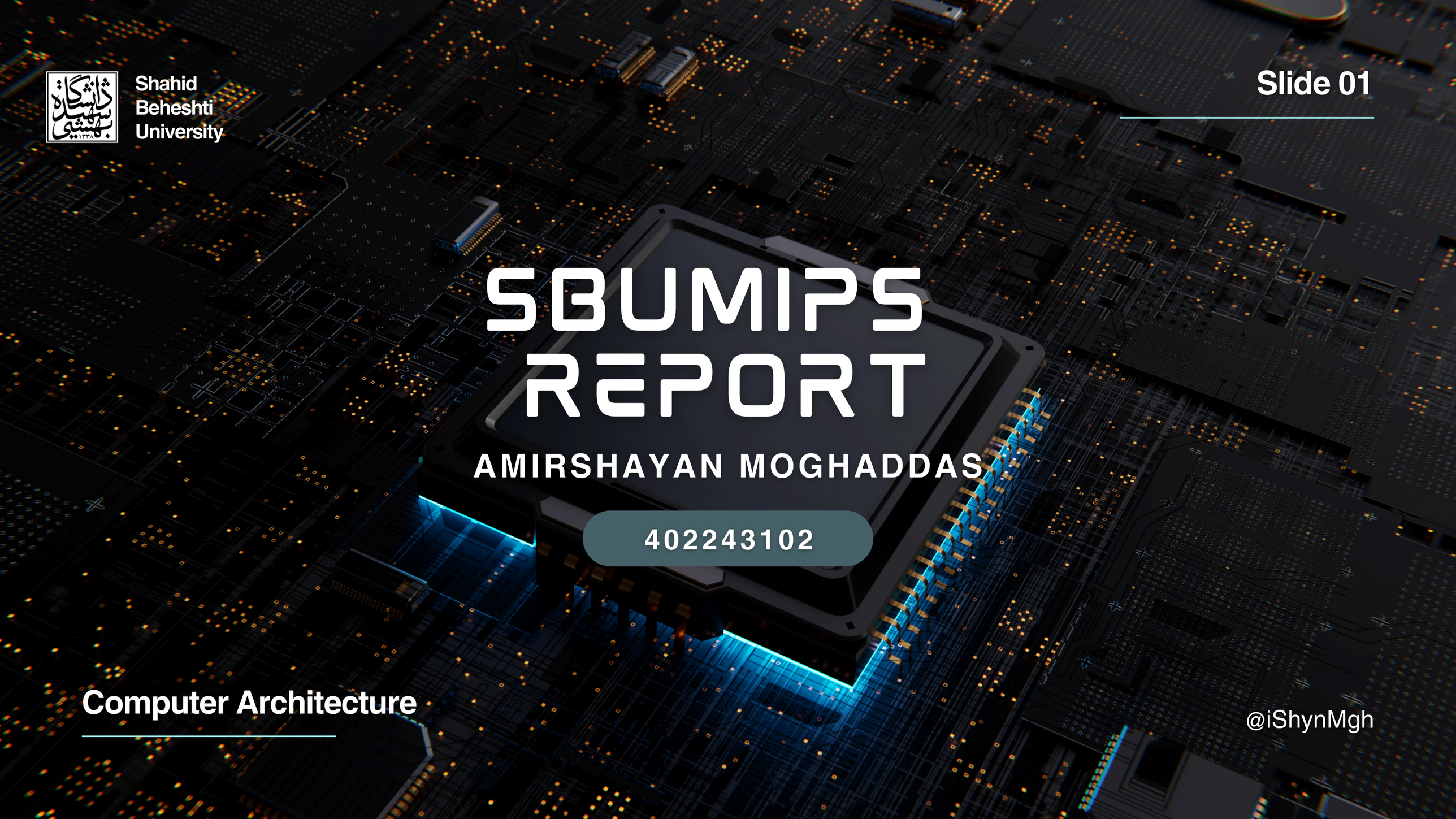

Shahid
Beheshti
University
Slide 01
SBUMIPS
REPORT
AMIRSHAYAN MOGHADDAS
402243102
Computer Architecture
@iShynMgh

5-STAGE PIPELINEDMIPS PROCESSOR

1. Introduction

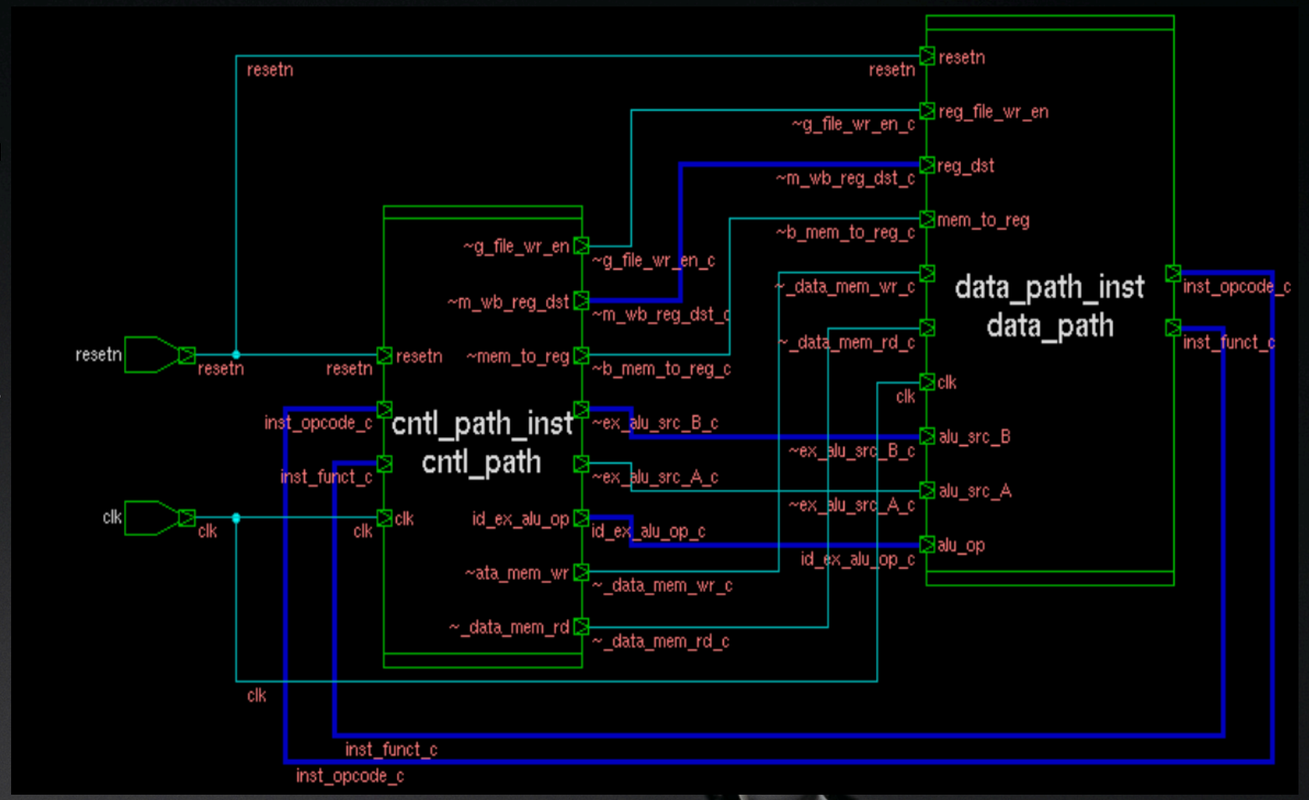
The goal of this project is to implement a basic MIPS processor using a traditional 5-stage pipeline architecture. The processor supports basic MIPS instructions including arithmetic operations, memory accesses (load and store), and branch instructions. The design objectives are to develop a datapath and control unit that allow correct execution of a MIPS program while handling pipeline hazards through forwarding and stalling techniques. This report describes the design, implementation, and simulation results of the 5-stage pipelined MIPS processor.

2. Processor Architecture Overview

The processor pipeline is divided into five stages:

1. Instruction Fetch (IF):
   - The processor uses a Program Counter (PC) to fetch the instruction from the instruction memory.
   - The instruction is stored in the Instruction Register.
2. Instruction Decode (ID):
   - The fetched instruction is decoded.
   - The Register File is accessed to read operands.
   - Immediate values are sign-extended to 32 bits.
   - The control unit generates control signals based on the opcode.
3. Execution (EX):
   - The Arithmetic Logic Unit (ALU) performs arithmetic or logical operations on the operands.
   - The branch target is calculated by adding the sign-extended immediate (shifted left by 2) to the PC.
4. Memory Access (MEM):
   - For load and store instructions, the data memory is accessed using the address computed in the EX stage.
   - For load instructions, data is read from memory; for store instructions, data is written.
5. Write-Back (WB):
   - The result from the ALU or the data memory is written back to the Register File.

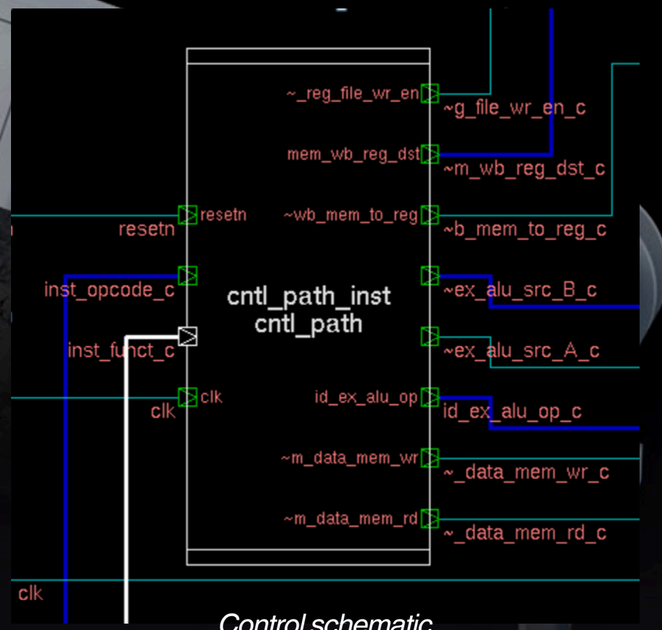
Between these stages, pipeline registers (IF/ID, ID/EX, EX/MEM, and MEM/WB) are inserted to hold data and control signals, ensuring smooth and synchronous propagation of instructions through the pipeline.



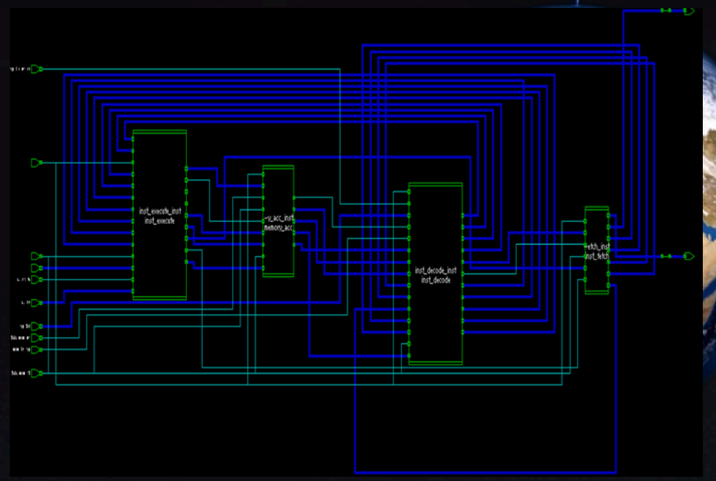
Top Module Schematic

Slide 03

- 3. Key Components and Data Path
- 3.1 Program Counter and Instruction Memory
- Program Counter (PC):
- The PC holds the address of the current instruction. At each clock cycle, the PC is updated (typically incremented by 4) to point to the next instruction.
- Instruction Memory:
- This is a read-only memory block that stores the MIPS instructions. The instruction at the PC address is fetched and stored in the Instruction Register in the IF stage.
- 3.2 Instruction Decode Stage
- Control Unit:
- Decodes the opcode and generates control signals such as RegWrite, ALUSrc, MemRead, MemWrite, Branch, and ALUOp.
- Register File:
- Consists of 32 registers. Two registers are read based on the instruction fields, and one register is written during the WB stage.
- Sign Extender:
- Extends 16-bit immediate values to 32 bits for arithmetic operations.
- 3.3 Execution Stage
- Arithmetic Logic Unit (ALU):
- Executes operations like addition, subtraction, bitwise AND, and OR. The ALU control unit (derived from the main control signals and the function field of R-type instructions) selects the operation.
- Branch Target Calculation:
- The sign-extended immediate is shifted left by two bits and added to the next sequential PC to calculate the branch target address.
- 3.4 Memory Access Stage
- Data Memory:
- Stores data used by the program. For a load instruction, the memory is read using the address computed by the ALU. For a store instruction, the data is written to the memory.
- 3.5 Write-Back Stage
- Write-Back Multiplexer:
- Selects whether the data to be written back comes from the ALU or from data memory. The selected value is then written into the Register File.
-



Control schematic



Datapath schematic

4. Pipeline Registers

To facilitate the flow of data between the pipeline stages, the following registers are used:

- IF/ID Register:
- Stores the PC and fetched instruction from the IF stage, forwarding them to the ID stage.
- ID/EX Register:
- Transfers control signals, register file outputs, and sign-extended immediate values from the ID stage to the EX stage.
- EX/MEM Register:
- Passes the ALU result, zero flag (for branches), and control signals (such as MemRead and MemWrite) from the EX stage to the MEM stage.
- MEM/WB Register:
- Holds the data from the data memory (for load instructions) or the ALU result (for R-type instructions) along with control signals from the MEM stage to the WB stage.

5. Hazard Handling

5.1 Data Hazards

Data hazards occur when an instruction depends on the result of a previous instruction that has not yet been written back.

- Forwarding/Bypassing:
- The design incorporates forwarding paths that supply the required operands directly from later pipeline stages (EX or MEM) to earlier stages (EX), reducing the need for stalls.
- Load-Use Hazards:
- Special handling (stall cycles) is inserted when a load instruction is immediately followed by an instruction that uses the loaded data.

5.2 Control Hazards

Control hazards arise from branch instructions because the target address might not be known until the EX stage.

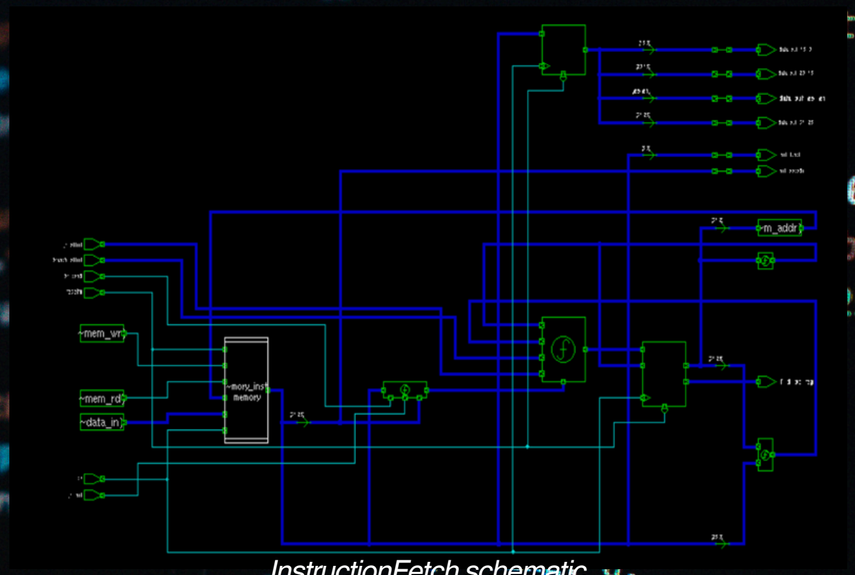
- Branch Handling:
- The processor may insert a branch delay slot (a nop) or use simple branch prediction to reduce stalls. In the 5-stage design, a single nop is inserted after the branch instruction to account for the delay.

5.3 Structural Hazards

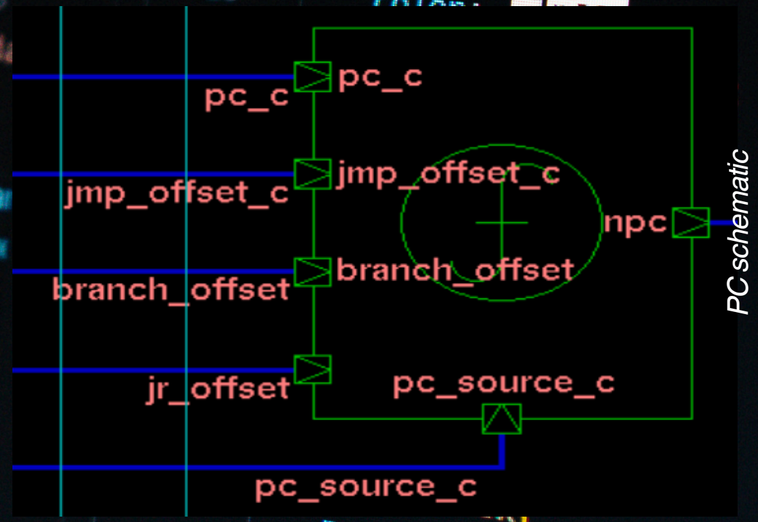
Structural hazards occur when hardware resources (e.g., memory or ALU) are required by more than one stage simultaneously.

- Resource Allocation:
- The design ensures that only one stage accesses the data memory or ALU at a time, eliminating structural hazards.

Slide 04

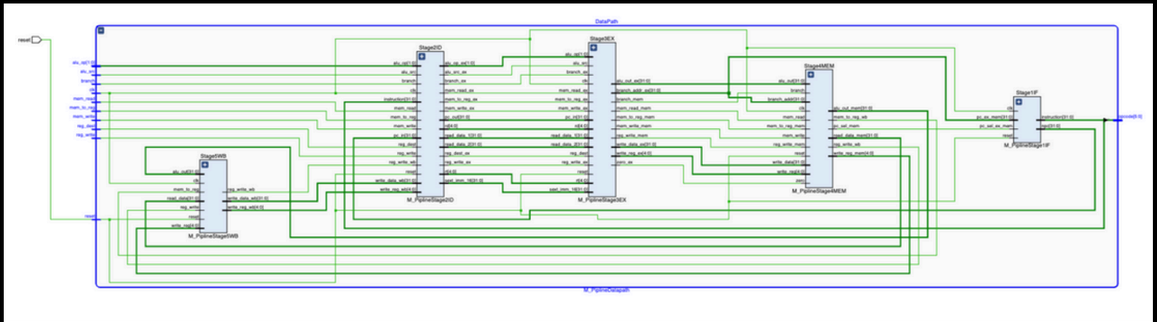


InstructionFetch schematic



PC schematic

synthesize,wave,output




```
Computed Fibonacci Terms (next 10 terms):
Term 3: 1
Term 4: 2
Term 5: 3
Term 6: 5
Term 7: 8
Term 8: 13
Term 9: 21
Term 10: 34
Term 11: 55
Term 12: 89
tb_MIPS_Processor.v:38: $finish called at 2000 (1s)
```

8-STAGE MIPS PROCESSOR

1. Introduction

The second part of the project extends the traditional 5-stage MIPS processor to an 8-stage pipeline. The primary motivation is to reduce the critical path delay in order to achieve a higher clock frequency (minimum 35 MHz) and improve instruction throughput. By breaking complex stages into more granular steps, the design reduces the overall combinational delay within each pipeline stage.

2. Design Objectives

The main objectives of the 8-stage pipeline extension were:

- Minimize Critical Path Delay: Divide long combinational paths into shorter stages.
- Increase Clock Speed: Achieve a target clock frequency of at least 35 MHz.
- Improve Throughput: Allow more instructions to be in flight simultaneously.
- Maintain Correctness: Ensure proper handling of data, control, and structural hazards by introducing additional pipeline registers and forwarding/bypassing logic.

3. Pipeline Modifications

In the 8-stage design, the original 5-stage pipeline is segmented further by introducing the following new stages:

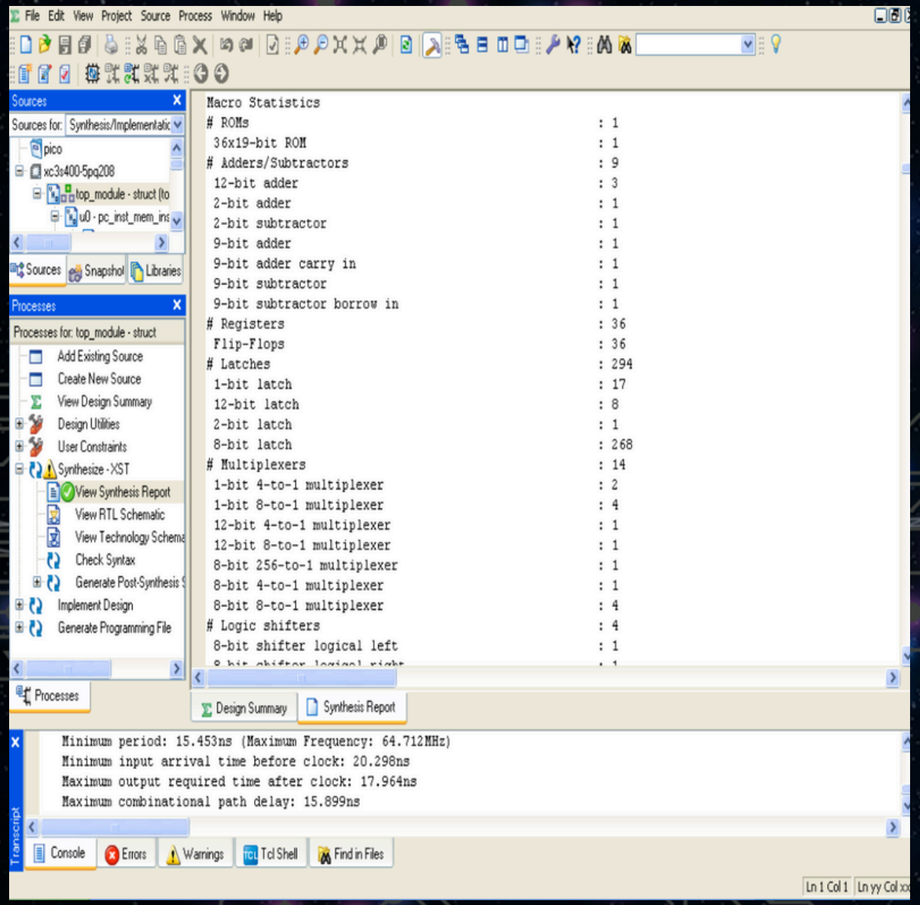
- PR (Register Read): Access the register file earlier than in the 5-stage design, reducing the load on the decode stage.
- ID (Instruction Decode): Decodes instructions without the burden of reading operands.
- EX (Execute): Performs simple arithmetic/logic operations.
- MR (Memory Read): Separates the memory read operation from the execution stage.
- MW (Memory Write): Isolates the memory write operation, reducing conflicts.
- WB (Write Back): Writes results back to the register file.
- FW (Forwarding): Dedicated stage to forward data for hazard resolution.

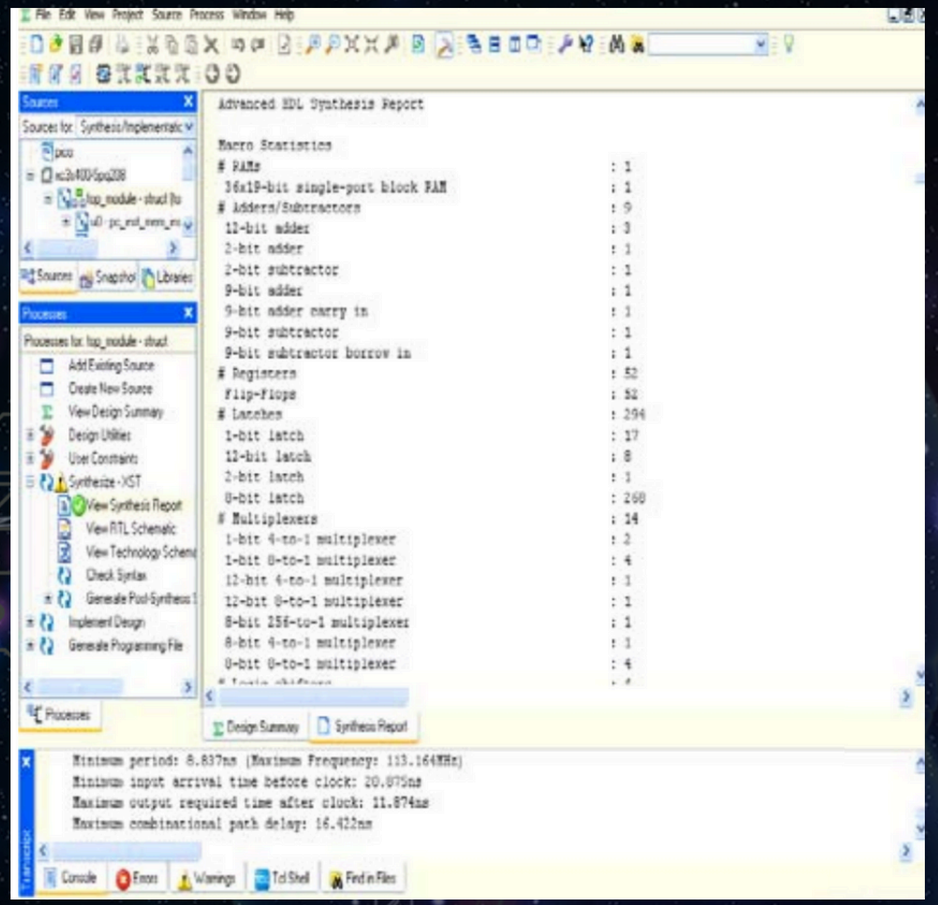
Additional pipeline registers (IF/PR, PR/ID, ID/EX, EX/MR, MR/MW, MW/WB, WB/FW) have been inserted between these stages to preserve data and control signals, ensuring smooth communication throughout the pipeline.

4. Implementation Details

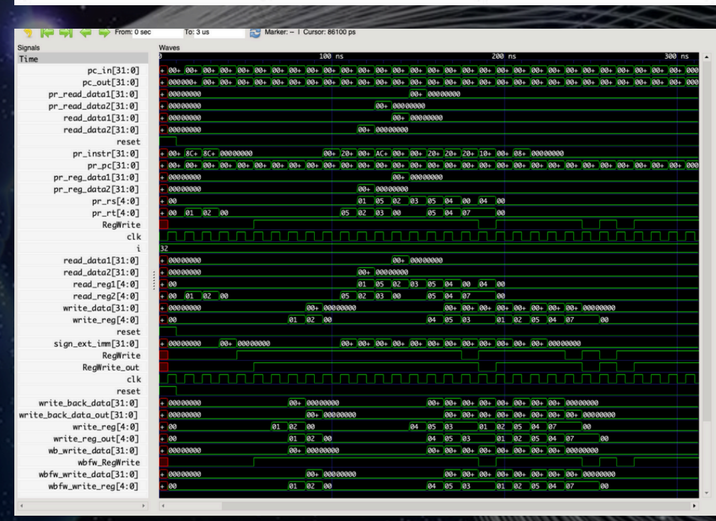
- Datapath Modifications:
- The original datapath was re-schematized to show the new pipeline stages. Critical components (ALU, register file, data memory, and control unit) were preserved; however, the data and control signals are now passed via additional pipeline registers.
- Control Signal Generation:
- The control unit was updated to generate the appropriate signals for each of the new stages. Multiplexers and forwarding paths were adjusted to resolve data hazards, particularly for load-use and branch instructions.
- Hazard Handling:
- The design employs forwarding (bypassing) paths and insertion of stall cycles (nops) where necessary. The separation of memory read and write operations into two distinct stages (MR and MW) ensures that structural hazards are minimized and that there is no conflict when accessing memory.

8stage vs 5stage





Minimum Delay	15.543 ns	8.837 ns
Maximum i/p Frequency	64.712 MHz	113.164 MHz
36 X 19 bits RAM	1	1
Adders/Subtractors	9	9
Registers	36	52
Latches	294	294
Multiplexers	14	14
Logic Shifters	4	4
XORs	9	9



SUPERSCALAR MIPS PROCESSOR

1. Introduction

The third part of the project extends the previous 8-stage pipelined MIPS processor to support superscalar execution. In a superscalar processor, multiple instructions are issued, decoded, and executed in parallel during a single cycle. This extension is aimed at increasing instruction throughput (higher IPC) by leveraging parallelism, while managing hazards that arise from issuing multiple instructions concurrently.

2. Design Objectives

The key objectives for the superscalar extension were to:

- Issue and Execute Multiple Instructions Per Cycle: Allow the processor to fetch and decode two instructions in parallel (dual-issue).
- Increase Instruction Throughput: Improve overall performance by executing independent instructions concurrently.
- Hazard and Resource Conflict Management: Introduce mechanisms for dependency checking (to detect data hazards) and resource scheduling, so that instructions that conflict are either stalled or converted to no-ops.
- Maintain Correctness: Ensure that the processor still computes the correct results (e.g., the Fibonacci sequence) despite parallel instruction issuance.

3. Key Modifications

3.1 Dual-Issue Front-End

- Instruction Memory Modification:
- The instruction memory was modified to output two instructions per fetch. The program is arranged in pairs so that one lane (lane1) performs the main computation (e.g., the Fibonacci calculation), while the other lane (lane2) is either used for auxiliary operations or forced to a no-operation (nop) when dependencies exist.
- Pipeline Registers for Dual-Issue:
- New pipeline registers (IF/PR_Dual and PR/ID_Dual) were introduced to carry a pair of instructions (and the corresponding PC value) simultaneously through the front end. The ID_Dual module decodes both instructions and includes a simple dependency checker that converts lane2's instruction to a nop if it depends on lane1.

3.2 Execution and Back-End

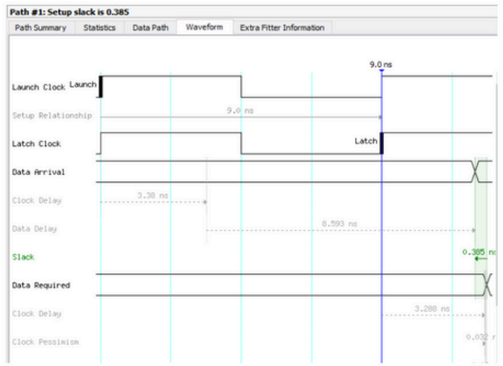
- Parallel Execution Units:
- Two ALU units are instantiated in the EX stage to support parallel execution. However, in our toy design only lane1 is actively used to compute the Fibonacci term, while lane2 is forced to nop when necessary.
- Shared Memory and Write-Back:
- The memory access and write-back stages are shared among the issued instructions. A store pointer is maintained to store computed results sequentially in Data_Memory. Additional logic ensures that only the correct computed result is stored, avoiding structural hazards.

3.3 Hazard and Dependency Management

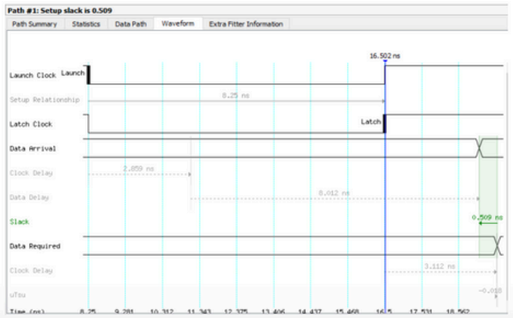
- Data Hazard Detection:
- The ID_Dual module implements a basic dependency checking mechanism. If the second instruction (lane2) depends on the result produced by the first instruction (lane1), lane2 is converted into a nop. This helps in avoiding data hazards that can compromise the correctness of parallel execution.
- Structural Hazard Handling:
- The design ensures that shared resources (such as the data memory and write-back unit) are not accessed simultaneously by conflicting instructions. In our design, careful scheduling and inter-stage registers maintain the integrity of the datapath.

SCHEMATIC

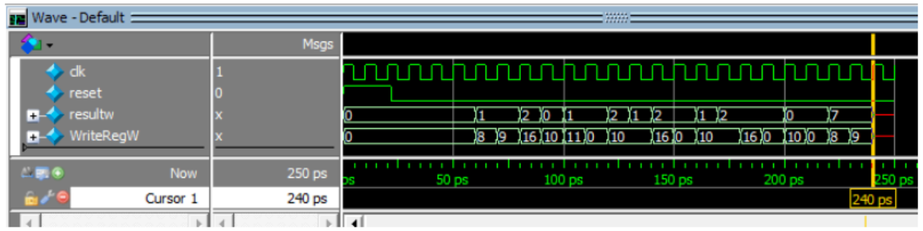
Original Pipeline Processor:



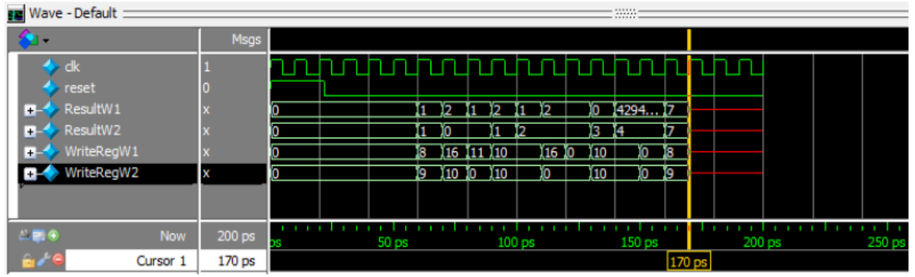
Superscalar Pipeline Processor:



Original Pipeline Processor Simulation:



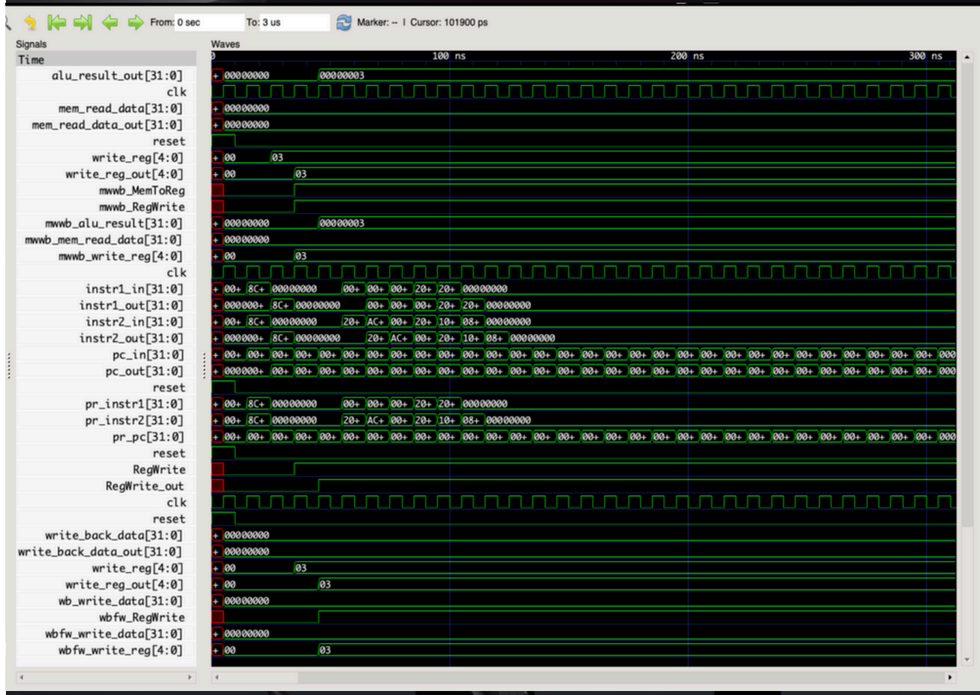
Superscalar Pipeline Processor Simulation:



Total Run Time (Original) = 240 ps

Total Run Time (Superscalar) = 170 ps

Superscalar is <u>1.41x faster</u> than the original pipeline processor



BRANCH PREDICTION INTEGRATION IN THE SUPERSCALAR MIPS PROCESSOR

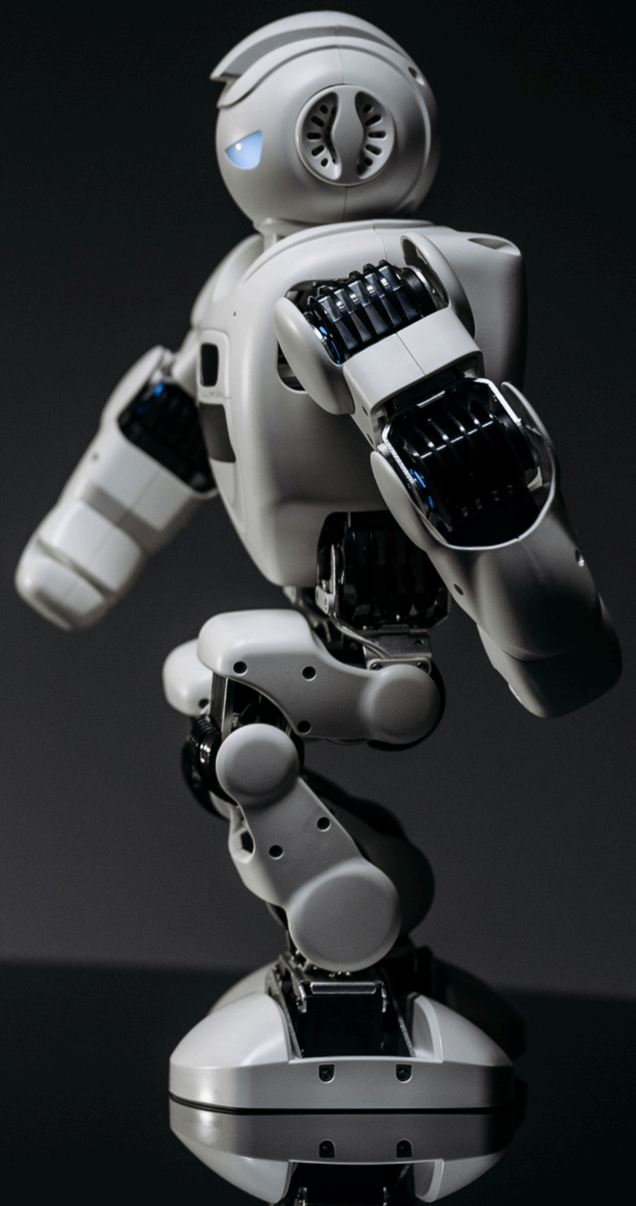
1. Introduction

The final part of the project focused on minimizing control hazards in a deeply pipelined, superscalar MIPS processor by incorporating a branch prediction mechanism. Branch mispredictions can severely degrade performance in high-clock-frequency pipelines. To address this, an advanced branch predictor was implemented and integrated into the design to predict branch outcomes early, thereby reducing stall cycles and improving overall throughput.

2. Design Objectives

The primary goals for the branch predictor integration were to:

- Reduce Control Hazards: Predict branch outcomes (taken or not taken) to avoid pipeline flushes due to mispredictions.
- Enhance Pipeline Efficiency: Improve instruction throughput by minimizing the number of cycles lost on branch mispredictions.
- Support Multiple Schemes: Implement several prediction strategies, including:
  - Global Adaptive Scheme (GAs)
  - Gshare (using XOR indexing of the PC with a global history register)
  - Gshare/Bimodal Hybrid (combining the strengths of two predictors)
  - Always Taken and Always Not Taken as baseline strategies
- Integrate with Superscalar Execution: Ensure the predictor works with a dual-issue pipeline, predicting branch outcomes for the front end so that multiple instructions can be issued per cycle without frequent pipeline flushes.



3. Implementation

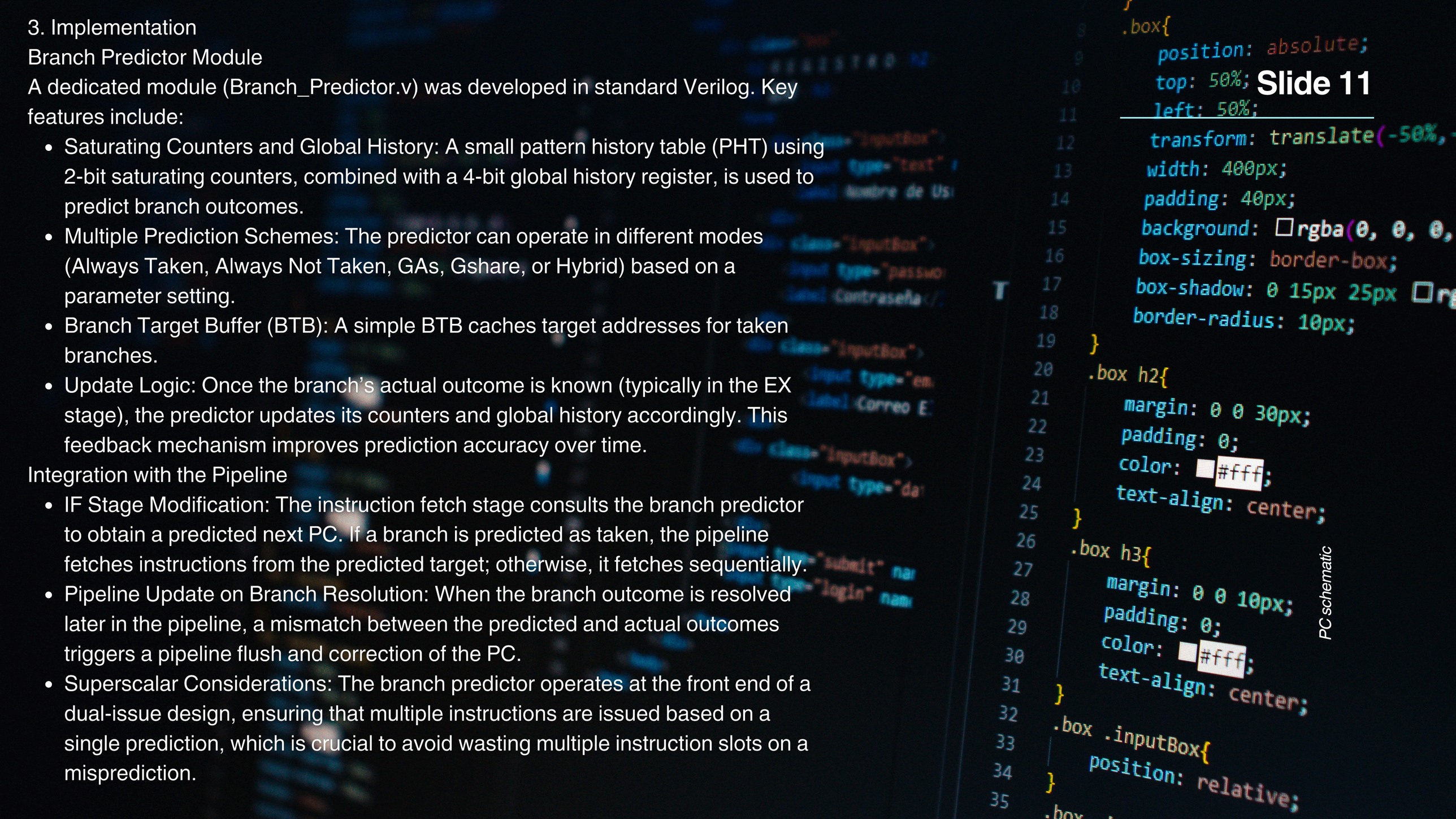
Branch Predictor Module

A dedicated module (Branch_Predictor.v) was developed in standard Verilog. Key features include:

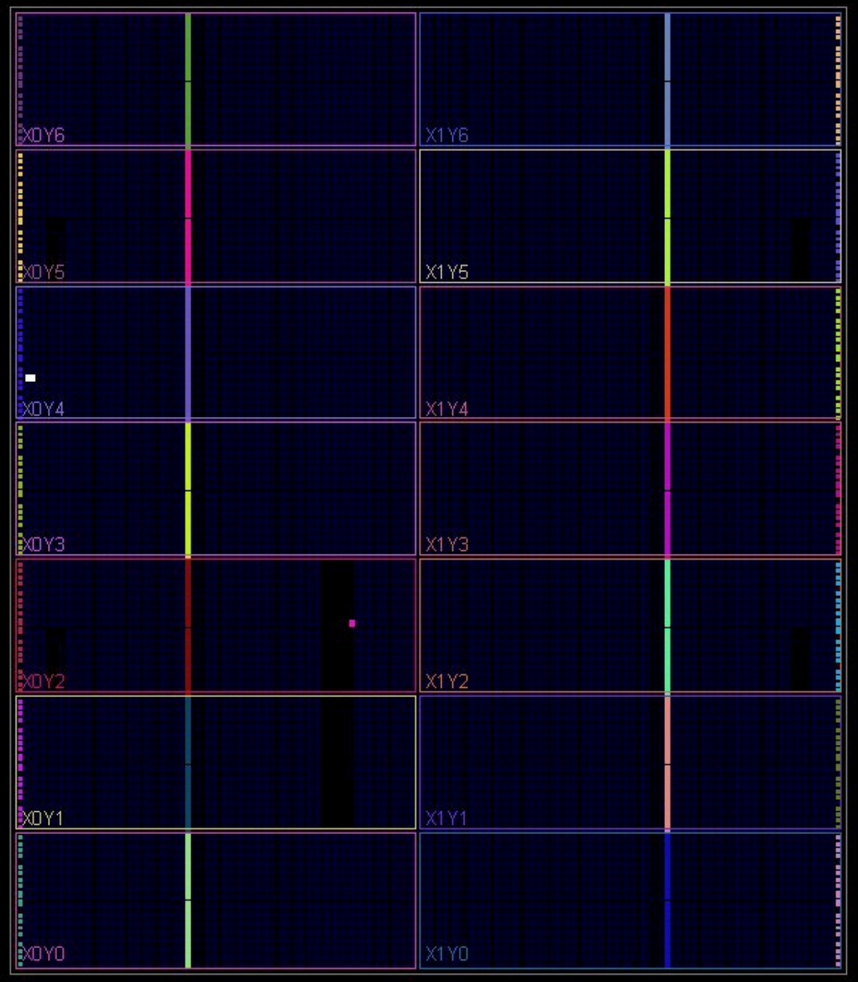
- Saturating Counters and Global History: A small pattern history table (PHT) using 2-bit saturating counters, combined with a 4-bit global history register, is used to predict branch outcomes.
- Multiple Prediction Schemes: The predictor can operate in different modes (Always Taken, Always Not Taken, GAs, Gshare, or Hybrid) based on a parameter setting.
- Branch Target Buffer (BTB): A simple BTB caches target addresses for taken branches.
- Update Logic: Once the branch's actual outcome is known (typically in the EX stage), the predictor updates its counters and global history accordingly. This feedback mechanism improves prediction accuracy over time.

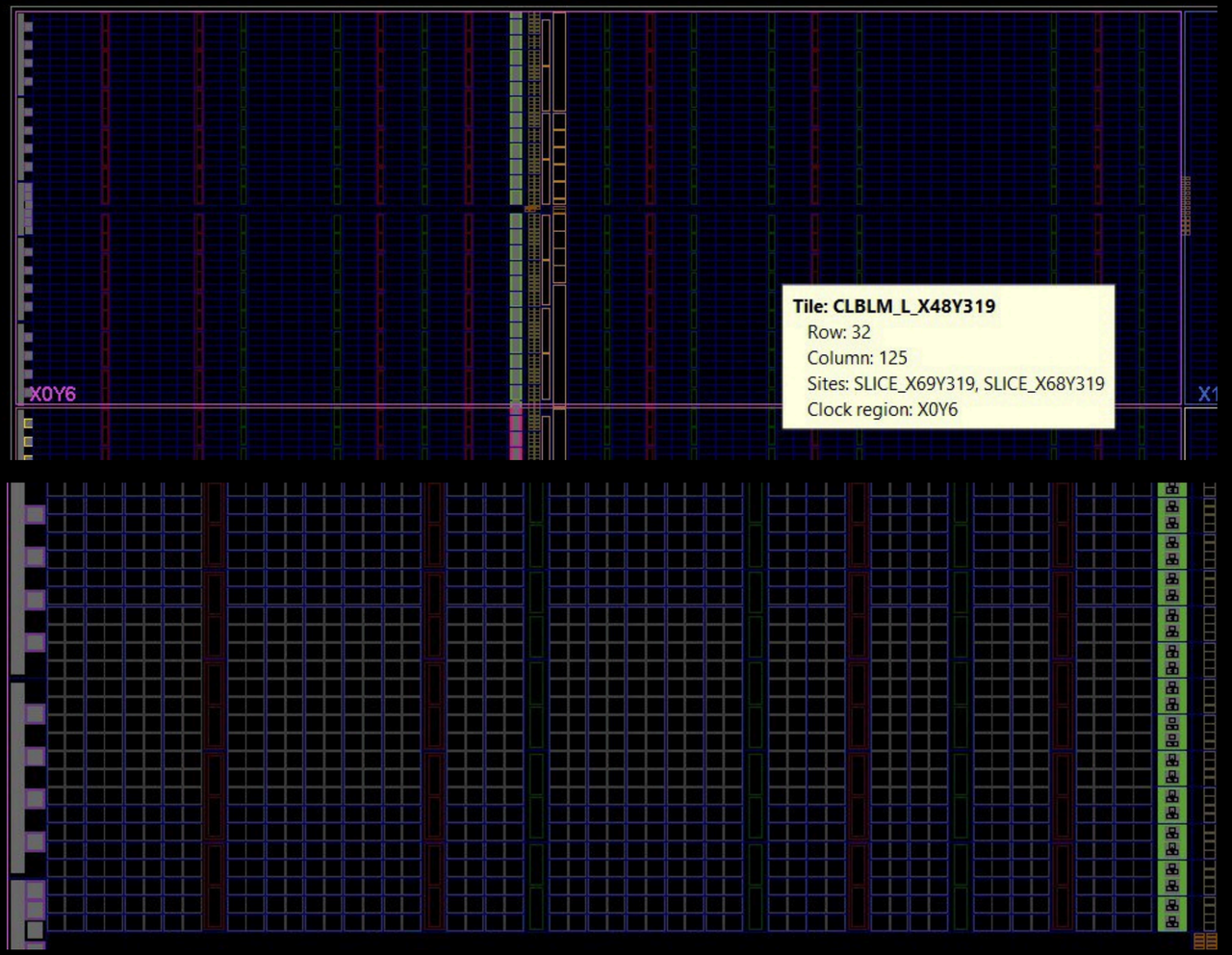
Integration with the Pipeline

- IF Stage Modification: The instruction fetch stage consults the branch predictor to obtain a predicted next PC. If a branch is predicted as taken, the pipeline fetches instructions from the predicted target; otherwise, it fetches sequentially.
- Pipeline Update on Branch Resolution: When the branch outcome is resolved later in the pipeline, a mismatch between the predicted and actual outcomes triggers a pipeline flush and correction of the PC.
- Superscalar Considerations: The branch predictor operates at the front end of a dual-issue design, ensuring that multiple instructions are issued based on a single prediction, which is crucial to avoid wasting multiple instruction slots on a misprediction.

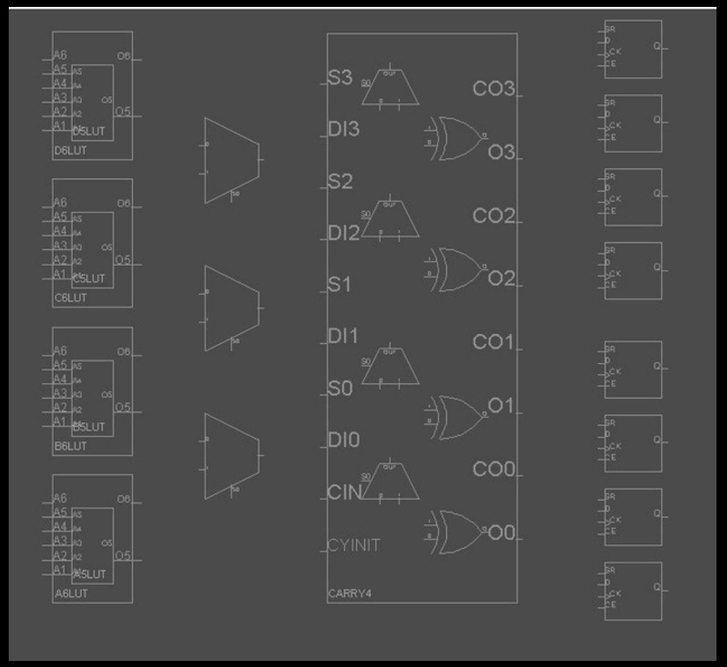


synthesize,wave,output





synthesize,wave,output




```
Computed Fibonacci Terms (next 10 terms)
Term 3: 1
Term 4: 2
Term 5: 3
Term 6: 5
Term 7: 8
Term 8: 13
Term 9: 21
Term 10: 34
Term 11: 55
Term 12: 89
```

THANK YOU

FOR YOUR ATTENTION

amirshayan.mgh7@gmail.com

@iShynMgh