EECE 321 Computer Organization

Project Report

Expansion of the Archer RV32I Processor to Support the "M" ISA Extension

Team 12 members:

Elias El-Khoury

Hadi Elham

Submitted to: Prof. Mazen Al. Saghir

Date: April 30, 2024

Table of Contents

1. Introduction

This Computer Organization course project focuses on the expansion of a single-cycle RV32I processor core, known as Archer, to support the "M" ISA extensions which include integer multiplication and division instructions. This enhancement is essential for broadening the processor's capabilities, allowing it to handle more complex arithmetic operations which are vital in many computational tasks.

The primary goal of this project is to design and integrate a dedicated hardware unit for multiplication and division into the existing RV32I Archer processor. This involves modifying the processor's data path and adding necessary control logic to manage the new operations efficiently. The project also entails rigorous simulation and testing using the Xilinx Vivado simulator to ensure the functionality and reliability of the expanded processor.

Through this project, students are expected to gain practical skills in digital design and processor architecture, deepening their understanding of hardware implementation aspects of computing systems. The project not only challenges the students to apply their theoretical knowledge but also prepares them for real-world engineering problems in the field of computer organization and architecture.

2. Project Objectives

Here are the detailed objectives for the EECE 321 project on expanding the RV32I Archer processor to include "M" ISA extensions:

1. ***Integration of "M" ISA Extensions:*** To expand the Archer processor's capabilities by incorporating the integer multiplication and division instructions specified in the "M" ISA extension. This involves modifying the existing single-cycle data path and integrating additional hardware units necessary for these operations.

2. ***Design of Multiply/Divide Unit:*** To design and implement a separate multiply/divide unit within the processor architecture. This unit should be capable of executing complex arithmetic operations independently of the main Arithmetic Logic Unit (ALU), focusing on optimizing performance and accuracy.

3. ***Simulation and Validation:*** To use the Xilinx Vivado simulator for the thorough testing and validation of the new hardware units and the complete processor datapath. This includes running simulations to verify the correct implementation of the new instructions and ensuring that they function as expected under various conditions.

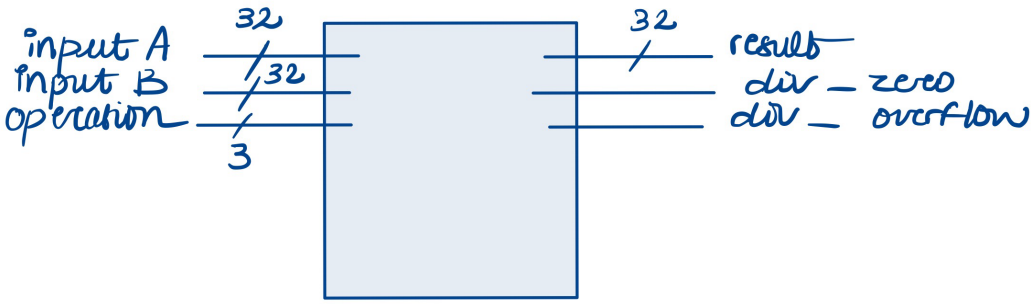
4. ***Functional and Performance Testing***: To develop and execute a comprehensive set of test cases that include scenarios for division by zero and overflow conditions. The tests should confirm that the processor handles these situations correctly and that the standard arithmetic operations are performed without errors.

5. ***Documentation and Reporting:*** To create detailed documentation of the design process, including block diagrams of the modified datapath and the new multiply/divide unit. The report should also contain descriptions of the testing procedures and the results, along with waveforms and other simulation outputs to illustrate the processor's functionality.

6. ***Practical Learning and Skill Enhancement:*** To provide students with practical experience in the field of processor design, emphasizing the application of theoretical concepts in real-world scenarios. This project aims to enhance students' problem-solving skills and deepen their understanding of computer architecture through hands-on design and implementation activities.

3. Block Diagram of the Multiply/Divide Unit

Summarized View:



4. New Block Descriptions and Modifications

New Blocks:

1) MULDIV_ALU which corresponds to ***mul_div.v*** and module instantiation ***muldiv***.
2) ALU_MULDIV_MUX_2TO1 which corresponds to ***alu_muldiv_mux2to1.v*** and module instantiation ***mux_alu_and_muldiv.***

MULDIV_ALU Description:

Inputs: - InputA and InputB directly ported from the Register File, each of 32-bit values

- Operation: 3-bit control signal that determines which of the following mul_div instructions to execute:

MUL:	3'b000	
MULH:	3'b001	
MULHSU:	3'b010	
MULHU:	3'b011	(MAPPING)
DIV:	3'b100	
DIVU:	3'b101	
REM:	3'b110	
REMU:	3'b111	

Outputs: - Result : 32 bit-value result of the computation of the selected mul_div instruction

Flags (outputs): - Div_zero : 1-bit value Flag set to 1'b1 in case of division by zero, and 1b'0 otherwise. This flag is configured to 0 when (inputB==0)

-Div_overflow: 1-bit value flag set to 1b'1 in case of division overflow and 1b'0 otherwise. This flag is configured when (inputA==-2^31)&&(inputB==-1)

Block Functionality:

The block uses three intermediate values (wires) to compute the result. It sets **intermediate1** to the multiplication of signed inputs A and B, sets **intermediate2** to their unsigned multiplication, and sets **intermediate3** to the multiplication of signed input A and unsigned input B. After this computation, the result of a MUL instruction is set by the lower 32 bits of intermidiate1, the result of MULH instruction is set by the upper 32 bits of intermediate1, and the result of a MULHSU instruction to the is set by the upper 32 bits of intermediate3. The result of a DIV instruction is set to -1 if input B is 0, input A in case of division overflow, and the signed division of the two signed inputs otherwise. The result of a DIVU instruction follows the same procedure for a division by zero case, and is set to the unsigned division of the two unsigned inputs otherwise. The result of a REM instruction is set to input A if input B was 0, 0 in case of division overflow, and the signed modulo of the two signed inputs otherwise. REMU result I also set to input A in case input B was 0, and the unsigned modulo of the two unsigned inputs otherwise. Then the MultDiv 3-bit value (operation) acts as a command or selector to choose what instruction to execute according to the earlier mapping.

ALU_MULDIV_MUX_2TO1 Description:

Inputs: - ALU Output ported directly from the main ALU that is used for the basic integer set instructions execution.

- MULDIV Output ported directly from our MULDIV_ALU (new block) that is used to execute the additional instructions corresponding to the M extension.
- SEL connected to the new control signal Mext described later.

Outputs: - MUX Output which corresponds to the output of the main ALU in case of basic I set instructions, and the output of our new MULDIV ALU otherwise.

Block Functionality:

This block chooses between the output of both ALUs, it ports the output of the main ALU in case Mext==1'b0 which corresponds to a basic I set instruction, and ports the output of the MULDIV ALU when Mext==1b'1 which corresponds to a MULDIV instruction.

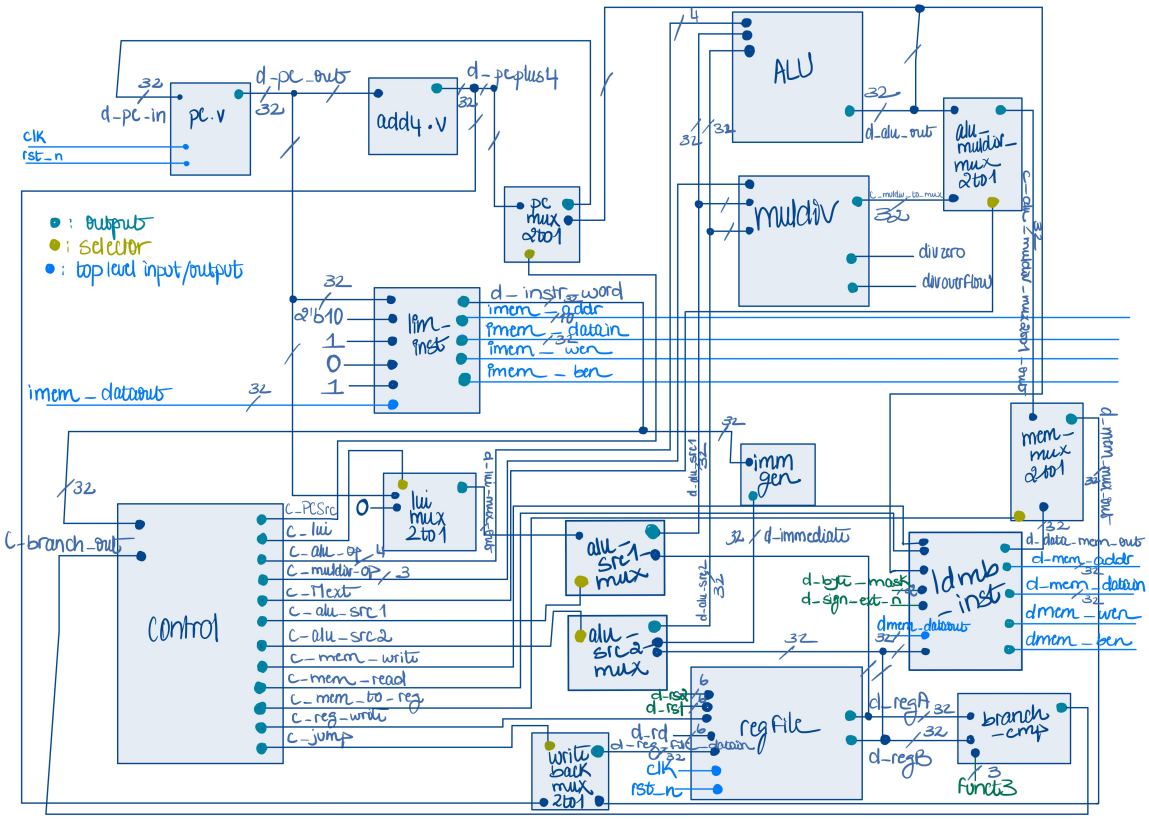
Modifications:

-Addition of a new control signal called Mext which is set when the processor needs to execute a MULDIV instruction and cleared otherwise, based on the least significant bit of the field function 7.

-Addition of a new 3-bit value operational control signal that acts as a selector between MULDIV instructions, based on the bits of the field function 3.

5. Detailed Datapath Representation

After successfully implementing the circuit, we display the following datapath for the designed RISC-VM Instruction Set:



Note: The signals *d_byte_mask, d_sign_ext_n, funct3, d_rd, d_rs1,* and *d_rs2* are derived from *d_instr_word*. To keep the block diagram looking clean, these signals are highlighted in green and correspond to:

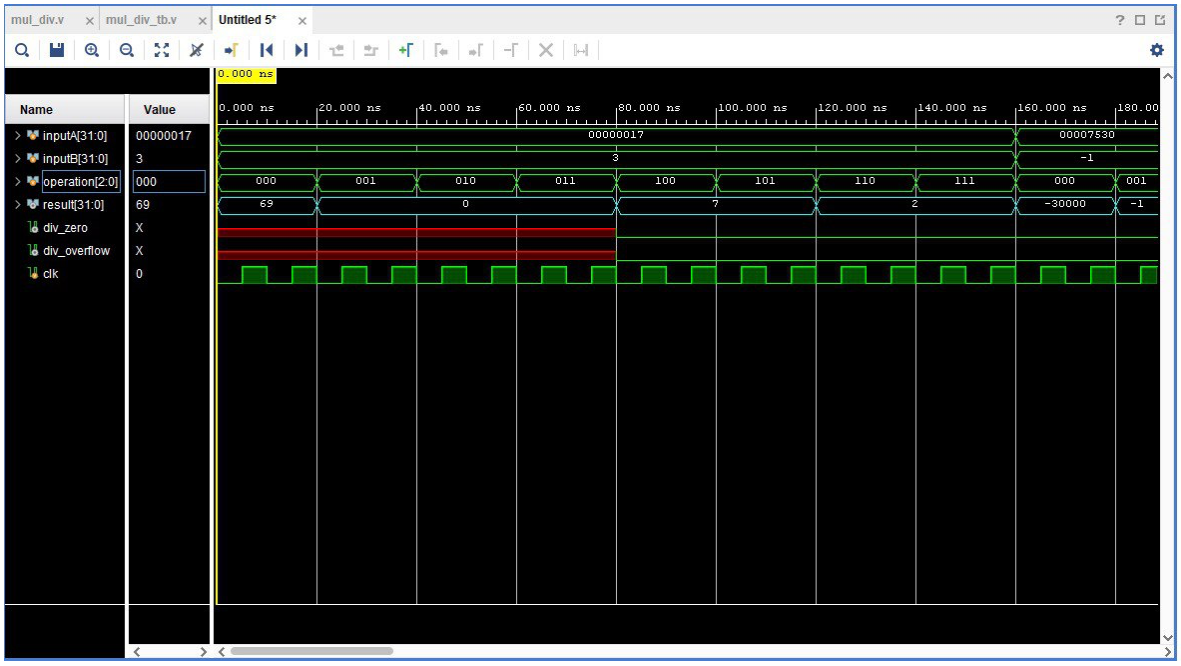
- *d_funct3 = d_instr_word[14:12]*

- *d_sign_ext_n = d_funct3[2]*
- *d_byte_mask = d_funct3[1:0]*
- *d_rs1 = d_instr_word[19:15]*
- *d_rs2 = d_instr_word[24:20]*
- *d_rd = d_instr_word[11:7]*

6. Assembly Code Implementation and Vivado Simulations

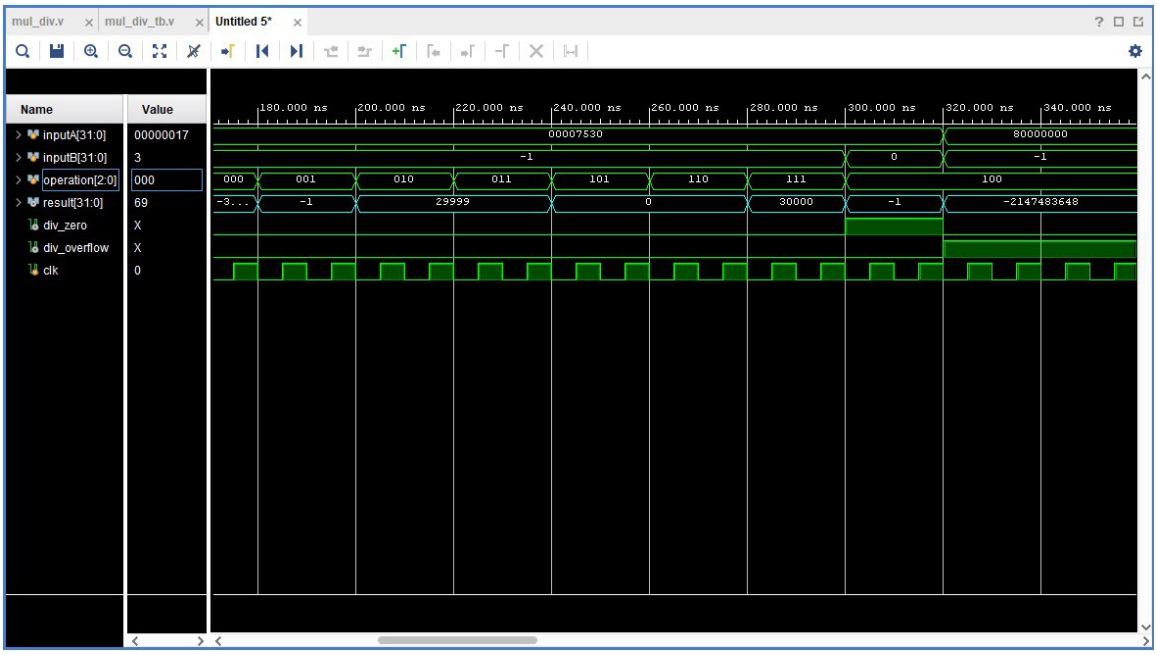
Attached with the project submission are the rom.v and assembly.s files corresponding to the applied testing and the implemented simulations.

Testing MULTDIV block alone:



After testing the block, we can verify that when command 000 is selected, corresponding to the MUL instruction, **inputA** was multiplied with **inputB** and the result was stored in **result** (0x017 x 0x03 = 23 x 3 = 69). When **operation** was 001,010,011 which respectively correspond to MULH, MULHSU, MULHU, **result** was set to 0 (verified). Moreover, setting **operation** to 100, or 101 yielded in **result**=7, and setting **operation** to 110, or 111 yielded in **result**=2. These results also verify the division output when operation was set to 100 corresponding to DIV and 101 corresponding to DIVU instructions (23 ÷ 3 = 7 (integer division) and 23 % 3 = 2 (remainder)). The following

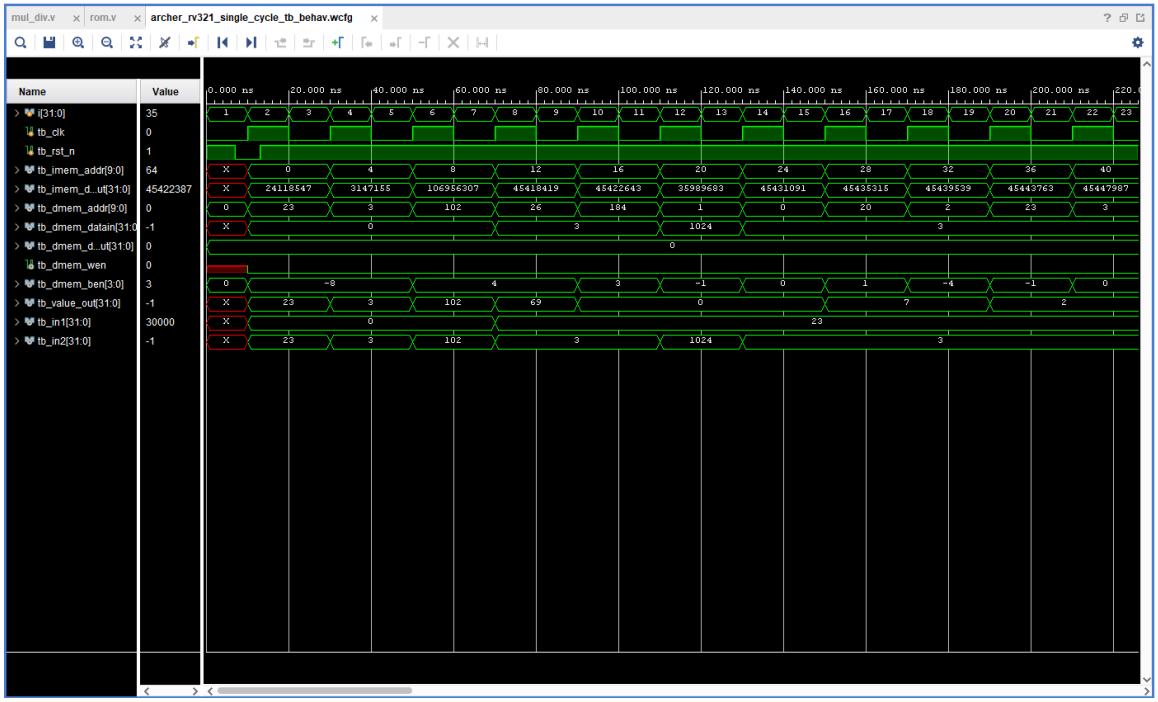
images also show the block functionality for further verification.



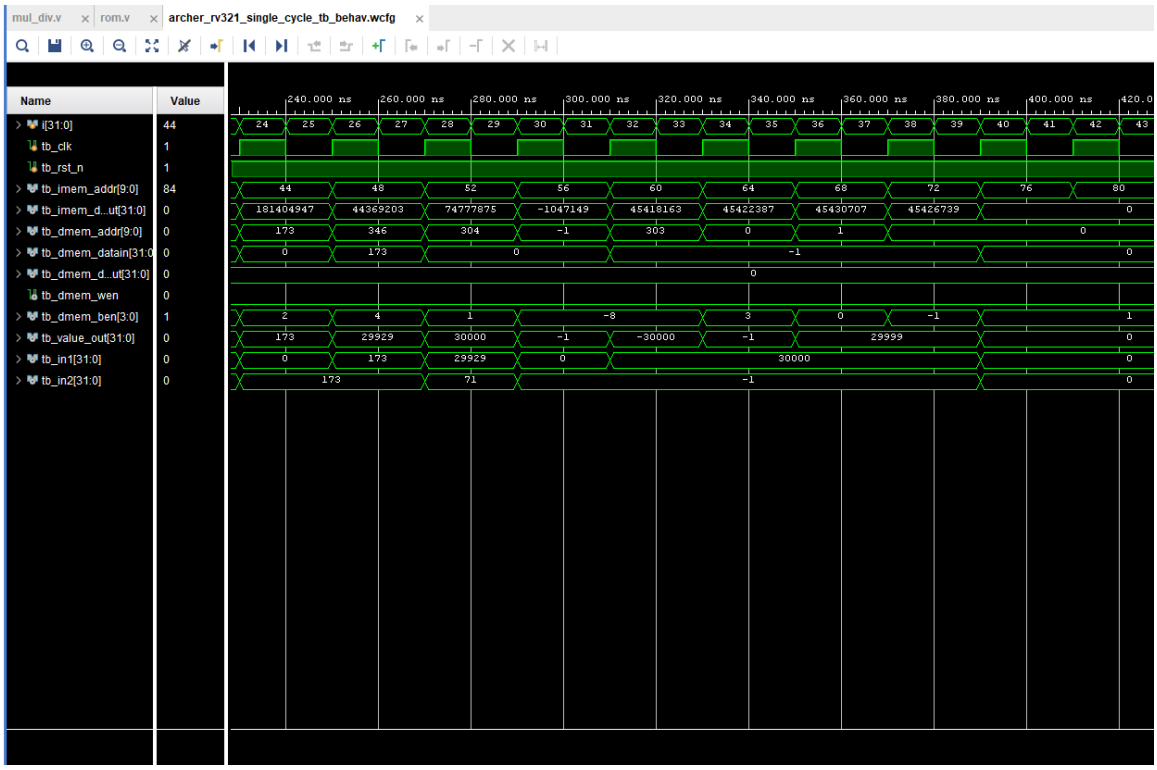
Now we verify through simulations the validity of the interconnection of this block with the whole processor circuit to fully construct our RISCVM Instruction set architecture.

Our inputs in the registers x10, x11, x12 are all fed into **inputB** and this is because the **addi** instructions in our assembly code all use **regB** to save the immediate value (ex: 23,3 and 102). Moreover, we can see that multiplication, division and rem instructions hold true for all our tests.

The following images are Vivado simulations that validate our implementation.

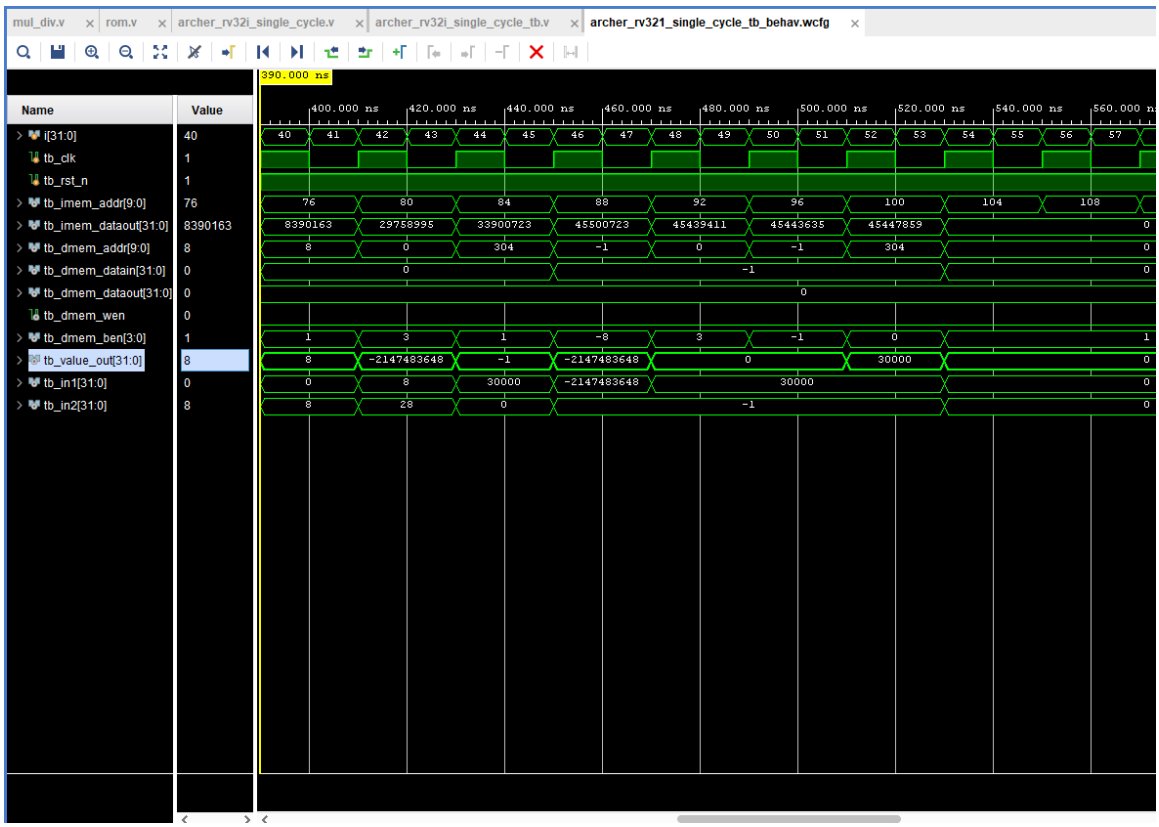


When we began implementing the assembly code into the ROM file as machine code generated from the Venus simulator, we observed the output **tb_value_out** in the waveforms, which depended on the inputs **tb_in1** and **tb_in2**. Initially, we initialized three registers, x10, x11, and x12, with the values 23, 3, and 102 respectively. Then, we executed the mul x15, x10, x11 operation, expecting to store the value 69 in register x15. Like when testing the mul_div block alone, we also tested the mulh, mulhu, and mulhsu operations, expecting a result of 0 since the result fits within 32 bits. Following that, we attempted to divide 23 by 3, yielding a quotient of 7 and a remainder of 2. It's worth noting that we employed both signed and unsigned division and remainder instructions, but they produced identical results because we were already working with unsigned inputs.



We proceeded with testing as before, this time incorporating signed numbers, which are crucial for computing the higher part of the multiplications. To begin, we initialized registers x10 and x11 with the values 30000 and -1 respectively. Since initializing 30000 directly in assembly isn't possible, we had to calculate it indirectly using the equation (173)**2 + 71 = 30000.

Now, we can compute the different instructions: we started by multiplying 30000 by -1, resulting in the value -30000. Then, we extracted the higher part of the result, which involved a simple sign extension, giving us the value -1. We also tested mulhu and mulhsu, expecting the same output since in mulhsu, we interpret a positive number as signed, which is equivalent to treating it as unsigned. Consequently, when taking the higher part of the unsigned multiplication, we get the value 29999.



To test for the overflow condition, we needed to create a specific combination of inputs. Therefore, we began by setting the register x12 to the value 4'b1000. Then, we decided to shift it 28 bits to the left and store it back in register x12.

To test the division by zero case, we executed "div x17, x10, x0," resulting in the output becoming -1. Similarly, when we attempted to divide the register x12 by -1, we obtained the same answer; this signifies an overflow condition, where the output matches inputA.

When we perform unsigned division of 30000 by -1, it's logical to get a quotient of 0. Likewise, in the regular remainder operation, dividing 30000 by -1 yields a remainder of 0.

Lastly, performing the unsigned remainder of 30000 divided by -1, which is essentially 30000 divided by 4294967295, results in a remainder of 30000.

7. Conclusion

In summary, the extension of the Archer processor with the "M" ISA extensions significantly enhances the performance features of Archer, especially in relation to integer multiplication and division operations. Testing with the Xilinx Vivado simulator and EDA playground led to a successful integration of a dedicated multiply/divide unit into the Archer's datapath. Further, this unit was tested with very rigorous conditions,

such as divide by zero and division overflow.
Our project not only extends the processor instruction set but also paves the way to solid ground for further improvements and optimizations. These deliverables, such as the modified Verilog source files, the new datapath block diagrams, and the comprehensive testing documentation, are full evidence of team collaboration and technical ability. This endeavor not only improved our understanding of digital design and processor architecture but also prepared us to tackle more complex system-on-chip designs in the future.

8. Team Contributions

	Elias Khoury	Hadi Elham	Joe Azar
Verilog and Syntax	X	X	X
Testing and Simulation	X		X
Assembly Coding and ROM Configuration	X	X	
Technical Report		X	
Block Descriptions	X	X	
File Modifications	X	X	
Datapath and Design	X		
Understanding Given Archer Processor File	X		

Note: We have collaborated with Joe Azar from Team 1 to join forces due to the tight deadline.