Final Project: MIPS

Andrés Antonio Bravo Orozco A01630783
Mariana González Bravo A01630948
Nicol Gómez Tisnado A00227180
Brenda Esquivel Pineda A01421610

Isaac Pérez Andrade
Computer Architecture

Thursday 4th of June 2020

1. Introduction

The MIPS processor is a RISC type of architecture, meaning that there are two instructions for accessing memory, a load instruction (to load data from the memory) and a store instruction (to write data to memory) [1]; also none of the other instructions can access memory directly. It's fundamental characteristics, such as the large number of registers, the number and the character of the instructions enable the MIPS architecture to deliver one of the highest performance per square millimeter for licensable IP cores, as well as high levels of power efficiency for today's SoC (System on a Chip) designs [2][3].

The purpose of this project was to create a custom microprocessor without Interlocked Pipeline Stage; we created three programs in Assembly Language in order to test our μP, these codes are meant to do the following: multiplication of two signed numbers, calculus of the first n-th Fibonacci numbers, and calculus of the first n-th element in a sum-of-square series.

2. Mips ISA design

Table of Instruction Encoding

Table of Instruction Encoding	
R-Type Instructions	**[31:26] - op → 6 bits** **[25:21] - rs → 5 bits** **[20:16] - rt → 5 bits** **[15:11] - rd → 5 bits** **[10: 6] - sa → 5 bits** **[5 : 0] - func → 6 bits**
I-Type Instructions	**[31:26] - op → 6 bits** **[25:21] - rs → 5 bits** **[20:16] - rt → 5 bits** **[15: 0] - immediate → 16 bits**
J-Type Instructions	**[31:26] - op → 6 bits** **[25: 0] - address → 26 bits**

Table 1. Instruction Encoding

Explanation of design

To achieve the final design of the mips we based ourselves in three different addressing modes R-type, I-Type and J-Type:

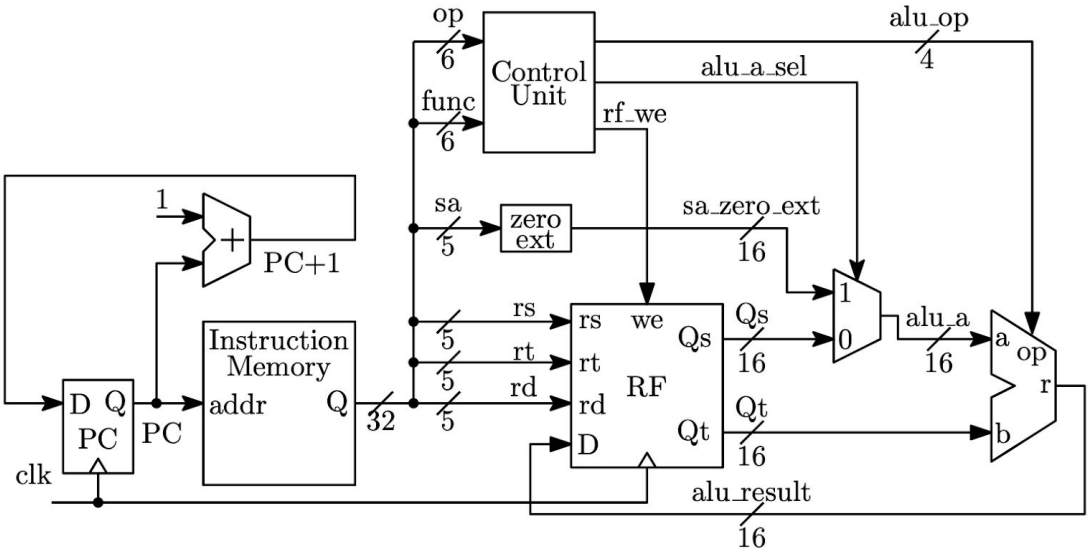

Fig. 1. R-Type instructions datapath schematic

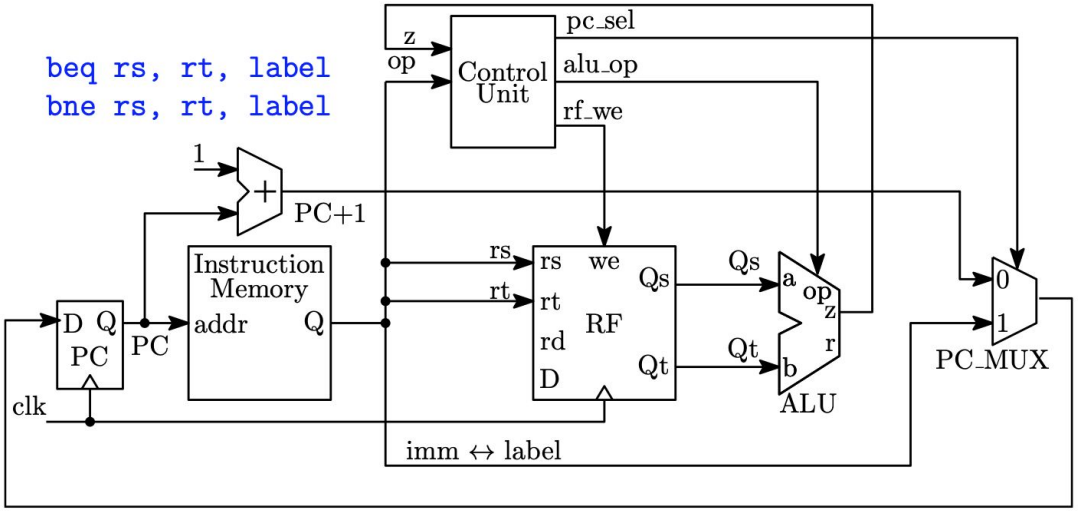

Fig. 2. I-Type instructions datapath schematic

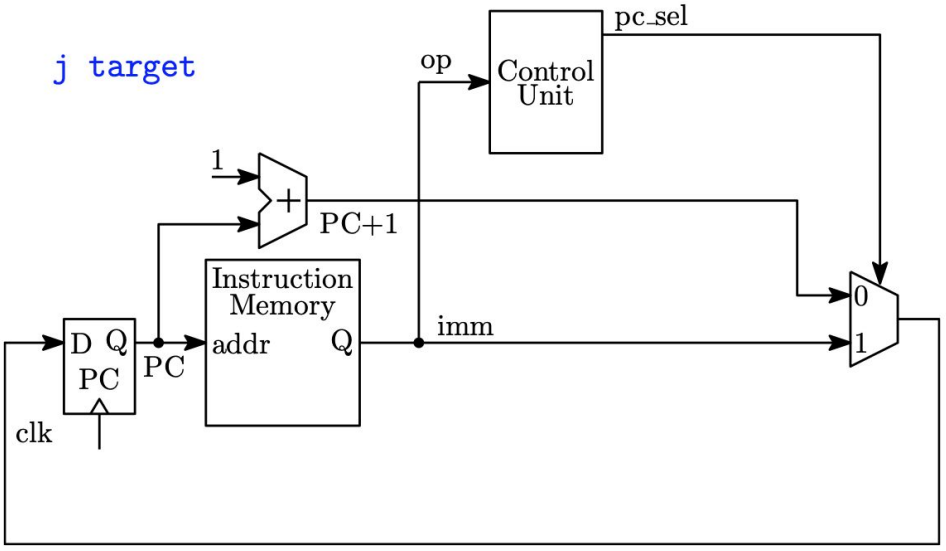

Fig. 3. J-Type instructions datapath schematic

We based our design in the R-type and then just added the necessary elements to work with each type of instructions. Since we first added the I-Type addressing mode the changes we did to our design were minimum, we added 5 muxes 2 to 1:

Mux 2 to 1	Inputs	Output	Selector
Mux 1 This mux was added so that shift operations work correctly	**-sa_zero_ext (output of sign extend)** **-Qs (output Qs of Register File)**	**-alu_a (input a of the alu)**	**-alu_a_sel (Output of the Control Unit)**
Mux 2 This mux was added so that all the operations that contain immediate values work correctly	**-Qt (output Qt of Register File)** **-immediate (signal directly gotten from the instruction)**	**-alu_b (input b of the alu)**	**-Qt_imm_sel (Output of the Control Unit)**
Mux 3 This mux was added so that rt and rd would be the same in the I-Type addressing mode	**-rt (signal directly gotten from the instruction)** **-rd (signal directly gotten from the instruction)**	**-rd2 (input to mux in J-type addressing mode)**	**-rd_sel (Output of the Control Unit)**
Mux 4 This mux was added so that we	**-pc1(output of the adder)** **-immediate (signal**	**-next_pc0 (input to mux in J-type addressing mode)**	**-Pc_sel (Output of the Control Unit)**

can give pc an immediate value	**directly gotten from the instruction)**		
Mux 5 This mux was added so we can add Data Memory values into the register file	**-Q (output Q of the DM)** **-alu_result (Output alu_result of the alu)**	**-ds (input to mux in J-type addressing mode)**	**-data_in_sel (Output of the Control Unit)**

Table 2. Mux table for R-Type with I-Type

After we added the I-Type addressing mode the changes to our design we then added 3 muxes 2 to 1 so that out R-Type - I-Type design could work along with our J-Type design:

Mux 2 to 1	Inputs	Output	Selector
Mux 1 This mux was added so that we can do the jal instruction work correctly	**-rd2 (output mux added in the I-type design)** **-5'b11111 (31 in decimal)**	**-rd (input rd to the Register File)**	**-jump_address_sel (Output of the Control Unit)**
Mux 2 This mux was added so that we can do the jal instruction work correctly	**-ds (output Qt of Register File)** **-pc1(output of the adder)**	**-D (input D of the Register File)**	**-jump_data_sel (Output of the Control Unit)**
Mux 3 This mux was added so that the instruction jr works correctly	**-next_pc0 (output from mux added in I-Type)** **-Qs (output Qs of Register File)**	**-next_pc (input D to Program Counter)**	**-jr_sel (Output of the Control Unit)**

Table 3. Mux table for R-Type, I-Type and J-Type

Finally after adding all of this muxes all of our 3 types of addressing modes finally work together and this design is the final design of out custom MIPS.

-Schematic

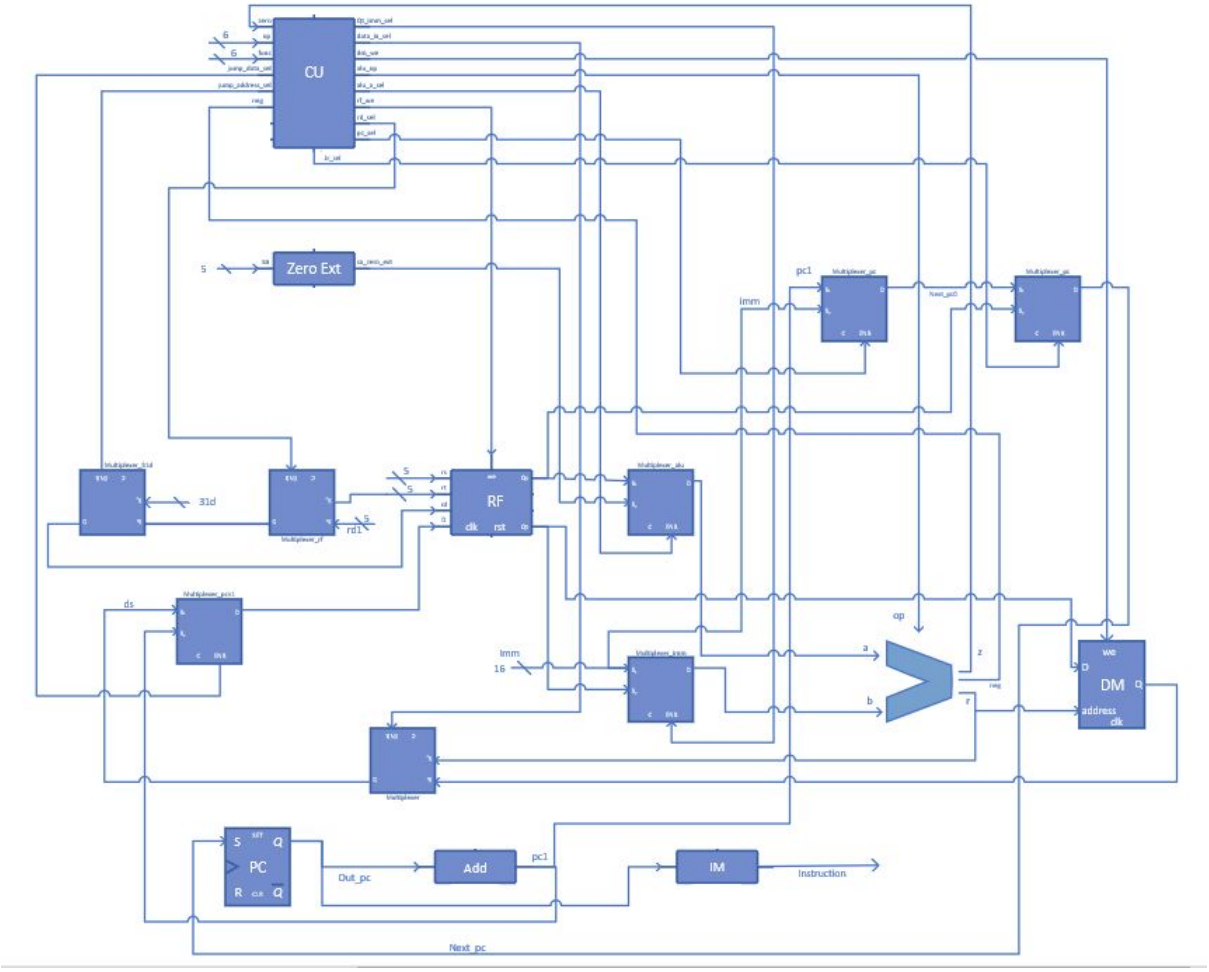

Fig. 4. MIPS design schematic

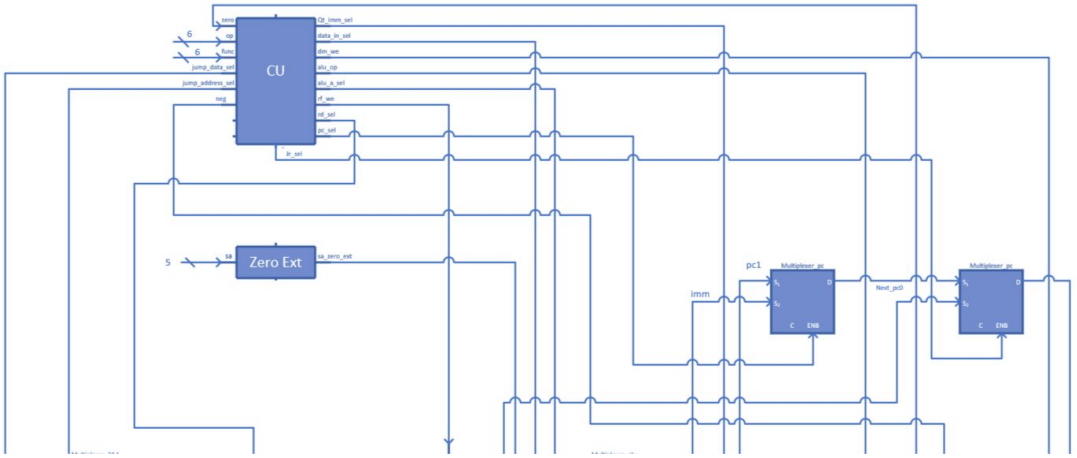

Fig. 4.1. MIPS design schematic closeup

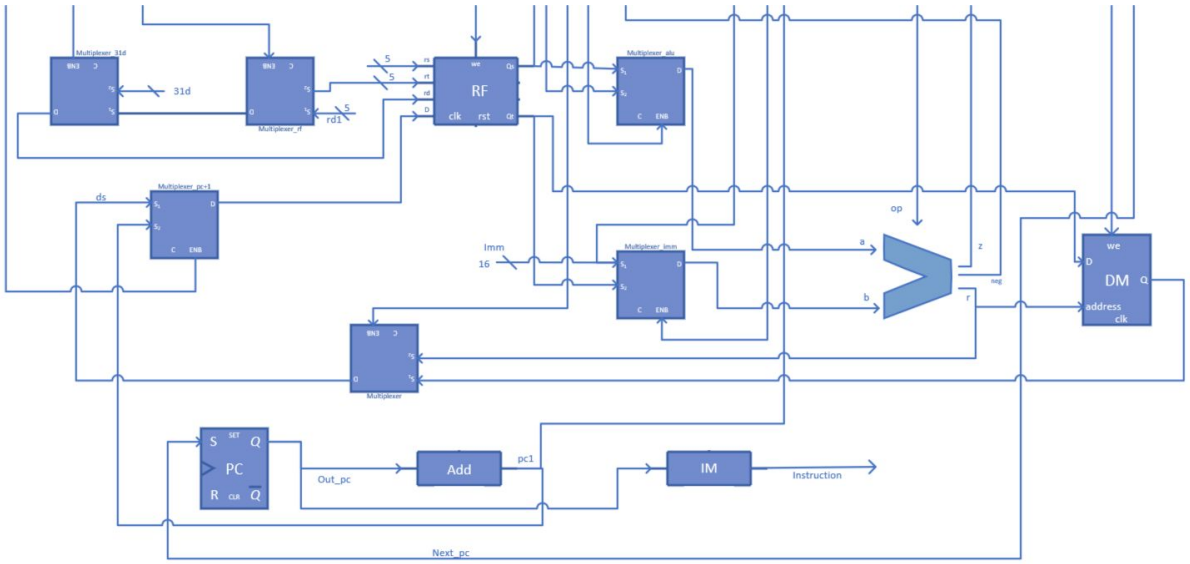

Fig. 4.2. MIPS design schematic closeup

3. Design Synthesis (RTL)

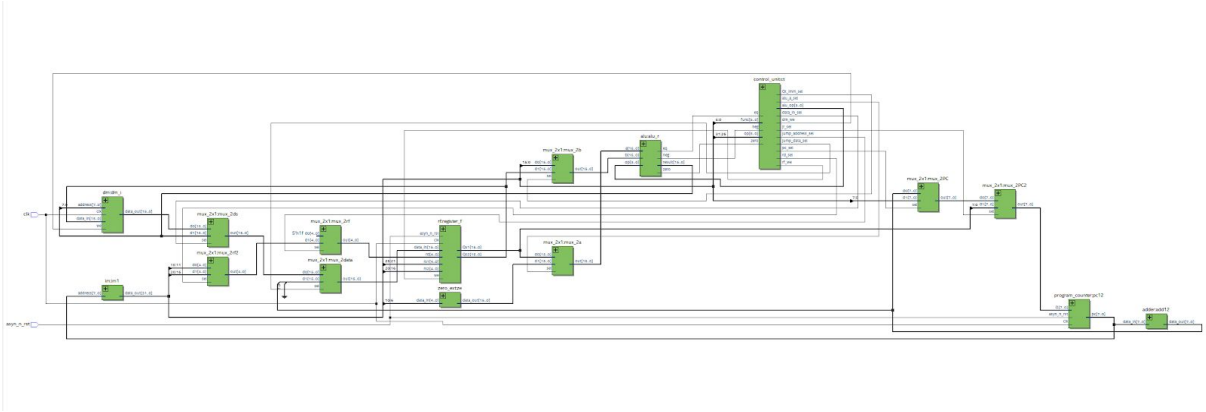

Fig. 5. RTL in Quartus

Test Programs

For this project, three different tasks were programed in order to test the correct functionality of the MIPS microprocessor and to familiarize ourselves with the way this custom MIPS operates.

Multiplication

The first program is a multiplication of two signed integers. These two operands are loaded into Mem[0] and Mem[1] of the microprocessor and the product of the two is loaded into Mem[2]. Since the data width of the register file and data memory is limited to 16 signed bits, the range of values able to be represented is from -32768 to 32767, meaning that the product of the two numbers cannot exceed this range. A table showing the explanation of each line of code for this program is given in Table 4.

Line	Binary Code	Meaning
1	00010000000000010000000000000000	LW r1, 0(0)
2	00010000000000010000000000000001	LW r2, 1(0)
3	00101000000000110000000000000000	LI r3, 0
4	00010000000001000000000000000000	LW r4, 0(0)
5	00001100010000100000000000000001	SUBI r2, r2, 1
6	10001100010000000000000000001010	BNEG r0, r2, 10
7	00000000100000010010000000100000	ADD r4, r4, r1
8	00001000011000110000000000000001	ADDI r3, r3, 1
9	10000100010000110000000000000110	BNEQ r3, r2, 6
10	11000000000000000000000000001101	J 13
11	00000000100000010010000000100001	SUB r4, r4, r1
12	00001100011000110000000000000001	SUBI r3, r3, 1
13	10000100010000110000000000001010	BNEQ r3, r2, 10
14	00010100000001000000000000000010	SW r4, 2(0)
15	11000000000000000000000000001110	J 14

Table 4. Multiplication Test File

Screenshots of the simulation using Mem[0] = 217 and Mem[1] = 151 are given in Fig. 6, 7 and 8, the result of this operation should give the positive operational limit of 32767. A second simulation is given as well in Fig. 9, 10 and 11, using Mem[0] = 256 and Mem[1] = -128 giving as product the negative operational limit of -32768.

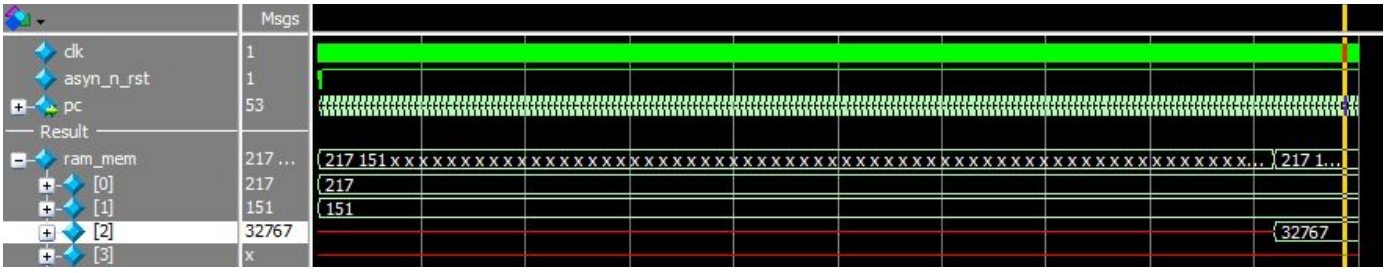

Fig. 6. Product of 217 and 151

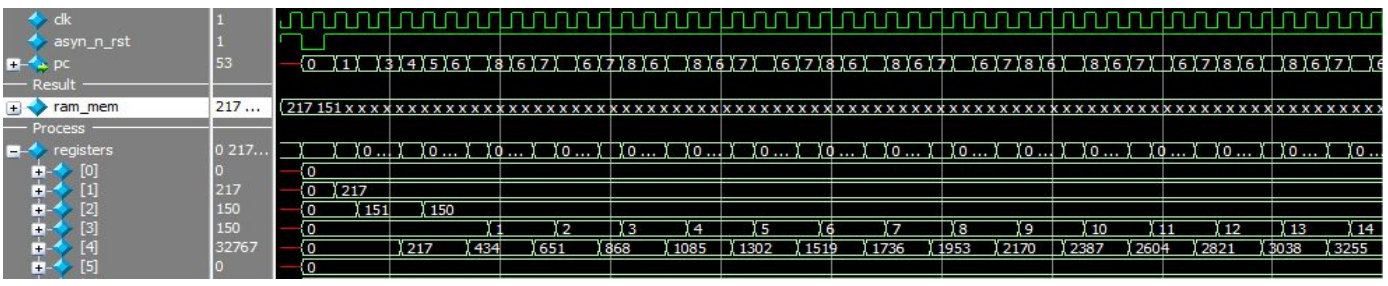

Fig. 7. First part of multiplication process

Fig. 8. Final part of multiplication process

Fig. 9. Product of 256 and -128

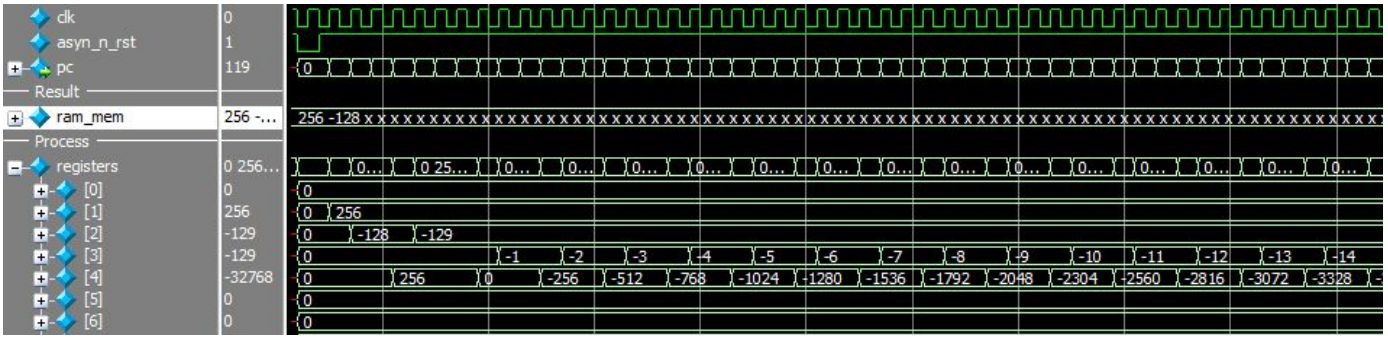

Fig. 10.First part of multiplication process

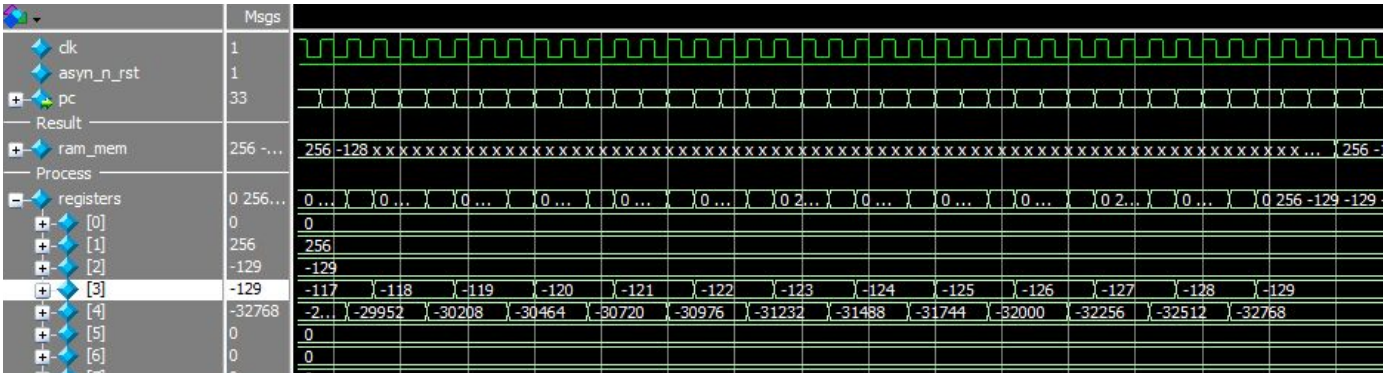

Fig. 11. Final part of multiplication process

Fibonacci sequence

The second program we made is the fibonacci sequence. This program should calculate the nth order Fibonacci sequence which is defined below. The value of n should be initialized in Mem[255]. The n + 1 elements of the sequence must be stored from Mem[0] to Mem[n].

$$F_n = F_{n-2} + F_{n-1}$$

Since we are considering 16 unsigned bits, the maximum value that can be represented is 65535, therefore the maximum order of the Fibonacci series able to be calculated by our MIPS is 24, which gives the value of 46368. The 25 order Fibonacci series final value is 75025, which is above our limit. A table describing the program is shown in table 5. Screenshots of our simulation are given below.

Line	Binary Code	Meaning
1	00010000000000010000000011111111	LW r1, 255(0)
2	00001000001000010000000000000001	ADDI r1, r1, 1
3	00101000000001000000000000000001	LI r4, 1
4	00010100010000110000000000000000	SW r3, 0(r2)
5	00000000100000110010100000100000	ADD r5, r3, r4
6	00000000100000000001100000100000	ADD r3, r4, r0
7	00000000101000000010000000100000	ADD r4, r5, r0
8	00001000010000100000000000000001	ADDI r2, r2, 1
9	10000100001000100000000000000011	BNEQ r2, r1, 3

10	11000000000000000000000000001001	J 9

Table 5. Fibonacci test table

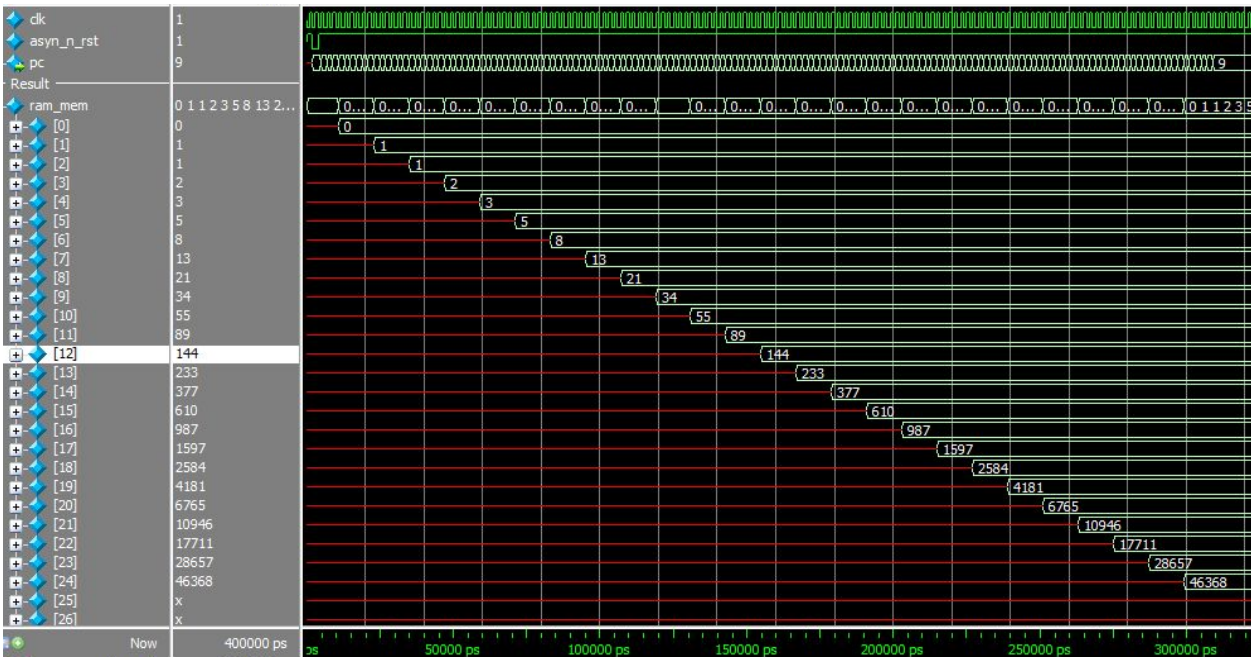

Fig. 12. 24th order Fibonacci series.

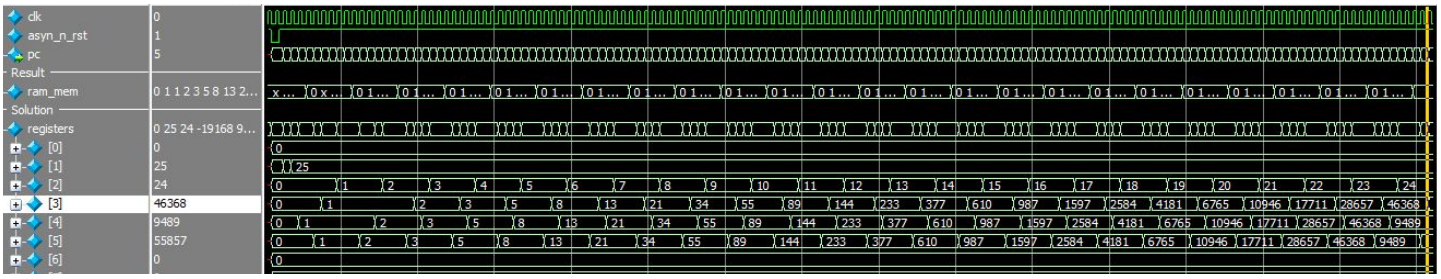

Fig. 13. Process of calculating 24th order Fibonacci series

Sum of Squares

The third program implemented in the custom MIPS is the sum of squares, This program should carry out the following series:

The n-th term in a sum of square series is defined as in Equation (1.2)

$$S_n = \sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \cdots + (n-2)^2 + (n-1)^2 + n^2 \tag{1.2}$$

Moreover, the n-th order sum of square sequence is defined as $\{S_1, S_2, \ldots, S_{n-2}, S_{n-1}, S_n\}$. For example, the 5-th order sum of square sequence is $\{S_1, S_2, S_3, S_4, S_5\} = \{1, 5, 14, 30, 55\}$.

Fig. 14. Sum of squares series

For this program, the value of n should be stored in Mem[0] and the subsequent results of the sum of squares series should be stored from Mem[1] to Mem[n]. As specified above, the upper limit of a 16 bit unsigned integer equals 65535. The series with n = 57 gives as result

63365, which is below the limit, but the series with n = 58 gives as result 66729 which is above the limit for our 16 bit unsigned integer. Therefore we can conclude that the maximum n value that can be carried out by our custom MIPS is 57. A table which describes the program is shown in Table 2. Screenshots of the simulation are given in Fig. 15, 16 and 17.

Line	Binary Code	Meaning
1	00010000000000010000000000000000	LW r1, 0(0)
2	00101000000000100000000000000000	LI r2, 0
3	00001000010000100000000000000001	ADDI r2, r2, 1
4	00101000000000110000000000000000	LI r3, 0
5	00101000000001000000000000000000	LI r4, 0
6	00000000100000100010000000100000	ADD r4, r4, r2
7	00001000011000110000000000000001	ADDI r3, r3, 1
8	10000100010000110000000000000101	BNEQ r3, r2, 5
9	00000000101001000010100000100000	ADD r5, r5, r4
10	00010100010001010000000000000000	SW r5, 0(r2)
11	10000100001000100000000000000010	BNEQ r2, r1, 2
12	11000000000000000000000000001011	J 11

Table 6. Sum of squares code and meaning

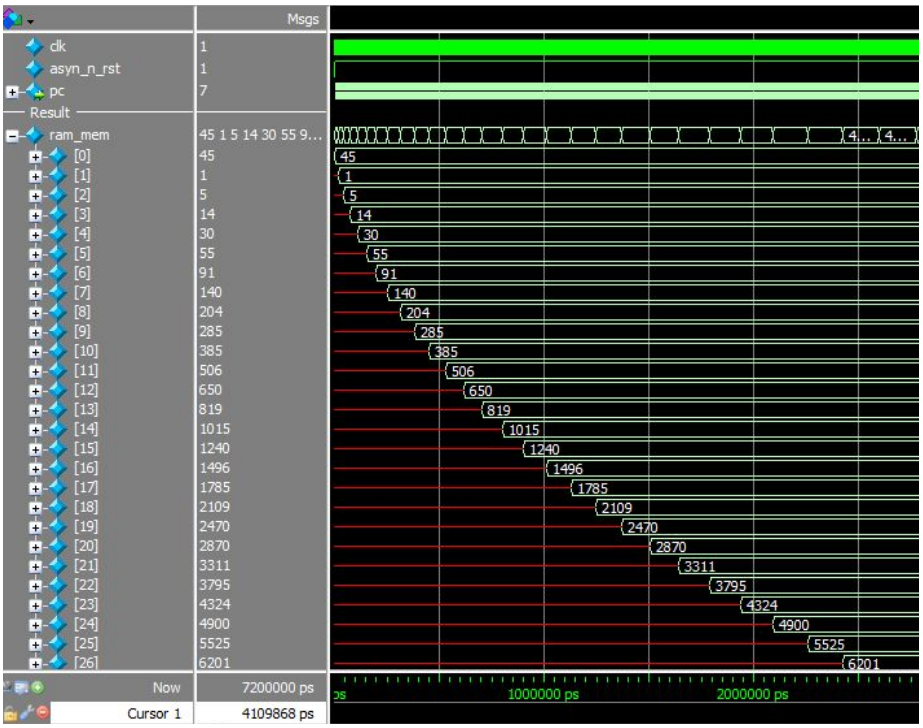

Fig. 15. Sum of squares series: 0 - 26

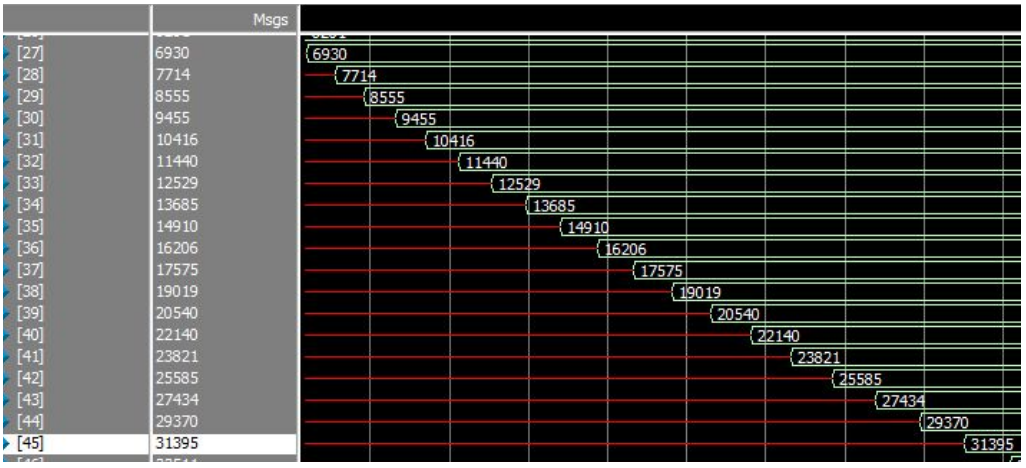

Fig. 16. Sum of squares series: 27 - 45

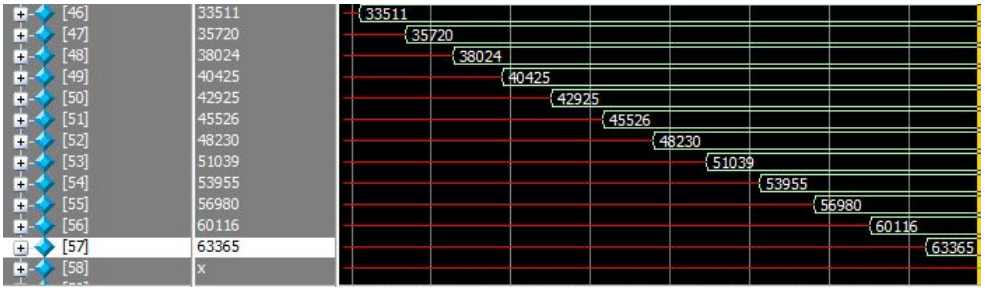

Fig. 17. Sum of squares series: 46 - 57

Improvements and future work

We consider that we can do few improvements in our MIPS for the future, such as implementing pipeline by dividing the design into different stages; we would have to delay some input and output signals into the flipflops so they arrive when needed, also add some pipeline control signals and controlling them with multiplexers , that would make our design to load instructions in parallel making our running time process shorter and faster [2]. Another thing we could improve, is the handling of interruptions. This could avoid future bugs or errors, when unexpected instructions are loaded.

Conclusions

With this project we acquire more knowledge regarding the mips architecture. We learned the differences between each type of instruction and how they get processed; with every stage of the project we understood how each addressing mode worked so we could design our own microprocessor. This also helped us to get more familiar with SystemVerilog programming and assembly language.

Overall, with this project, we gathered all of our knowledge from past courses, such as Digital Design, and this course to create something that is essential in our profession. We hope that in a future we can use this project as a base to innovate future technologies.

References

[1] **D.Harris , *Digital Design And Computer Architecture*, 1st ed. Amsterdam: Morgan Kaufmann, 2007. [E-book] Available: http://0-search.ebscohost.com.biblioteca-ils.tec.mx/login.aspx?direct=true&db=nlebk&AN=240231&lang=es&site=eds-live**

[2] **Crisp, J. *Introduction to Microprocessors and Microcontrollers*, Vol. 2nd ed. Amsterdam: Newnes, 2004. [E-book] Available: http://0-search.ebscohost.com.biblioteca-ils.tec.mx/login.aspx?direct=true&db=nlebk&AN=195647&lang=es&site=eds-live.**

[3] **Hennessy, John L., Krste Asanović, and David A. Patterson. *Computer Architecture : A Quantitative Approach*. Vol. 5th ed. Waltham, MA: Morgan Kaufmann. 2012 [E-book] Available: http://0-search.ebscohost.com.biblioteca-ils.tec.mx/login.aspx?direct=true&db=edsebk&AN=407995&lang=es&site=eds-live.**