Design of an Enhanced RISC-based Processor with Increased Memory Capacity and I/O Ports

Nathanael Adrian T. Cua
Department of Electronics and Computer Engineering
De La Salle University
2401 Taft Avenue, Manila, Philippines
nathanael_cua@dlsu.edu.ph

Louie T. Que
Department of Electronics and Computer Engineering
De La Salle University
2401 Taft Avenue, Manila, Philippines
louie_que_a@dlsu.edu.ph

Alvin Josh T. Valenciano
Department of Electronics and Computer Engineering
De La Salle University
2401 Taft Avenue, Manila, Philippines
alvin_valenciano@dlsu.edu.ph

Jana Johannes G. Valenzuela
Department of Electronics and Computer Engineering
De La Salle University
2401 Taft Avenue, Manila, Philippines
jana_valenzuela@dlsu.edu.ph

Abstract **- This project enhanced the RISC architecture, thus accomplishing the objective of enhancing performance and usability for the MIPS computer. It aimed at increasing the memory from one gigabyte to four gigabytes and adding four input/output ports. These enhancements were done and tested using some simulations on Vivado. Testing was performed for the maximum length of a program, capacity regarding integer values, and capacity regarding bits in communications, all of which proved that augmented memory and extra I/O ports had been integrated successfully. These changes made the RISC-based MIPS architecture much more efficient and versatile.**

Index Terms - RISC, computer architecture, increased memory, I/O ports, computer architecture

I. Introduction

A. *Introductory Information*

The Reduced Instruction Set Computer (RISC) is a computer architecture where each instruction performs a single, simple operation [1]. This type of architecture emphasizes the simplification of hardware by having an instruction set that is easy to decode. These types of instructions are usually easier to decode because they are uniform in size, which is usually one word, and execute in a single clock cycle. Due to RISC's reduced complexity, it makes use of pipelining to increase its performance. RISC has a large number of general-purpose registers, and simple addressing modes, thus making this type of architecture efficient.

The group chose to enhance the RISC architecture primarily due to its simplicity and efficiency. Unlike CISC which allows for complex tasks, RISC allows the straightforward and efficient implementation of the objective of the project. This characteristic aligns with the objective of the project, which is to improve the performance and usability of the MIPS architecture, through modifications in memory capacity and number of I/O ports.

To be more specific, the project aims to increase the MIPS's memory capacity from one gigabyte to four gigabytes in order to enhance its ability to handle larger sets of data and perform more complex operations. Modifying the memory to support this increase involves adjusting the addressing bits in the program memory and data memory modules. Additionally, the project aims to add four additional I/O ports to the MIPS's architecture. This can be done by modifying the control unit module to increase the number of ports. These ports improve the computer's versatility by allowing more external devices to be connected, in turn allowing better interaction and data exchange.

B. *Objectives*

The general objective of this was to improve the performance and usability of the MIPS architecture. This was done by successfully accomplishing the specific objectives:

- To increase the memory capacity of a RISC computer to four (4) gigabytes
- To add and/or implement four input/output ports to the RISC computer
- To simulate the results through Vivado.
- To test the proper implementation of the increase in memory and addition of ports using the test parameters

C. *Implemented Features and Test Parameters*

This project includes features such as memory capacity extension and the addition of more I/O ports, which were very important in enhancing the functionality and general performance of the MIPS architecture. A number of test parameters were therefore set up and tested to ensure these features conform to specifications.

Testing for increased memory involved the establishment of the maximum lengths of programs that could be run by the architecture. In this case, different program lengths and complexities were run to see how the system managed memory allocation and execution without any errors or degradation in performance.

Testing of the additional I/O ports tested the architecture with respect to data communication. The tests were meant to check the maximum value of integers the system can support and the maximum bits which could be successfully sent or received via the I/O ports. These tests would ensure that the new ports could support the necessary data types and speed of communication, and thus ensure the system could support larger and more complicated processes.

II. Computer Architecture Specifications and Application

A. *Application Area*

The group decided to focus on enhancing the performance and usability of a single-cyle 32-bit MIPS architecture through modifications in memory capacity and the addition of I/O ports. Hence, the application area of the project involves processing complex computations with additional I/O capabilities.

The design process began with a thorough analysis of the architecture, from its individual modules to its overall application, in order to identify the areas of code requiring modifications. The architecture was specified to include a 32-bit fixed-point multiplication and a floating-point unit (FPU) for handling complex arithmetic and floating-point operations. The components chosen were selected based on their ability to meet the performance requirements of the existing architecture. The 4GB memory provides ample storage for data, while the four I/O ports provide interaction with external devices.

The design was created using a hardware description language, Verilog. This was how the modifications in memory and the addition of I/O ports were implemented, as well as its integration with the multiplier and the FPU. Behavioral simulations were used to describe the functionality of the multiplier and the FPU's arithmetic processes. The structural design involved defining the interconnections between various modules, including how the CPU interfaces with memory and I/O ports. This also included specifying how the fixed-point multiplier and FPU interact with the CPU's operation systems.

Testbenches were developed to verify the functionality of each module. These testbenches included test cases for arithmetic operations, memory access, and I/O interactions. Simulations in Vivado were run to check the correctness of each component in isolation and in combination, wherein once individual modules were verified, integration testing was performed to validate that the whole CPU system, with the enhanced memory and I/O ports, works correctly together.

B. *Instruction Set Architecture*

The Instruction Set Architecture (ISA) for this RISC implementation includes a 32-bit single-cycle CPU enhanced with additional memory and I/O ports. The ISA supports standard MIPS instructions, as shown in Table 1.

Table 1
32-Bit MIPS ISA

Instruction Type	Instruction	Description
R	jr	jumps to the contents of $rd
R	sub	subtracts $rs by $rt, and stores the result in $rd
R	slt	sets $rd to 1 if $rs is less than rt
R	add	adds $rs by $rt, and stores the result in $rd
I	lw	loads a word to $rd
I	sw	stores a word to $rd
I	beq	branches if $rs and $rt are equal
I	addi	adds contents of $rs by immediate value, and stores the result in $rd
I	xori	performs XOR operation on $rs and $rt, and stores result in $rd
J	j	jumps to the $rs
J	jal	branches by an offset, and stores the current PC+4 to $ra
R	mult	multiplies $rs by $rt, and stores the result in $rd
R	mfhi	moves from HI to $rd
R	mflo	moves from LO to $rd
FR	add.s	adds $rs by $rt, and stores the result in $rd (FPU)
FR	sub.s	adds $rs by $rt, and stores the result in $rd (FPU)
I	lcw1	loads a word to $rd (FPU)
I	swc1	stores a word to $rd (FPU)

The ISA for this architecture includes addressing modes and instructions that utilize the large memory space of

4GB efficiently. Similar to other MIPS implementations, the memory is organized to support word alignment, where each word is 32- bits wide, which corresponds to 4 bytes. This alignment allows the simplification of the addressing and makes sure that memory accesses are consistent per module. Since the address bus of the CPU is capable of addressing up to 4GB, the architecture can support complex and data-intensive operations, which further allows the CPU to handle larger programs without frequent memory access delays.

The 32-bit MIPS architecture also involves the addition of four I/O ports–PORTA, PORTB, PORTC, and PORTD– which significantly expands the CPU's capability to interact with external devices and peripherals. Each port provides a bidirectional interface for data exchange between the CPU and external hardware, such as sensors, actuators, and other communication modules. For instance, PORTA could be used for general-purpose input/output operations wherein reading and writing data methods can be implemented.

The memory and I/O ports are integral to the CPU's functionality. The enhanced memory provides the necessary storage for programs and data, while the I/O ports enable communication with external devices. The interaction between the said components is important for achieving a cohesive system within the architecture. Data stored in memory can be manipulated by the CPU and then transmitted to or received from external devices through the I/O ports. The efficiency of these operations depends on the proper integration of memory management and I/O handling within the CPU architecture.

C. Computer Performance Testing

The evaluation of the enhanced RISC-based processor's performance focused on key parameters related to the increased memory capacity and additional I/O ports. Execution time (T_{exec}) was measured across various workloads, including arithmetic operations and control flow structures, using the formula:

$$T_{exec} = N_{instructions} \text{ x } CPI \text{ x } T_{clock} \quad (1)$$

where $N_{instructions}$ is the number of instructions executed, CPI is the average Cycles Per Instruction, and T_{clock} is the clock period. To optimize performance, minimizing T_{exec} is crucial. This involves reducing the number of instructions executed, minimizing CPI, and maximizing clock frequency.

Memory testing involved a comprehensive assessment of the system's ability to handle extensive data and program sizes. This included evaluating the maximum length of programs that could be executed without failure and determining the capacity for storing large quantities of integer or floating-point numbers. Such testing was crucial for verifying the successful implementation and functionality of the system's 4GB memory capacity. By pushing the system to its limits, the tests ensured that the memory could handle real-world applications effectively. This process was essential for validating the overall performance and robustness of the memory architecture, guaranteeing that it met the necessary specifications and user expectations.

For I/O ports, testing focused on the total number of bits available for communication, encompassing both data transfer rates and the system's ability to handle various data types through the newly added ports. This involved introducing diverse and unpredictable data sequences to assess the system's adaptability and robustness, capturing the intricacies of real-world usage. Additionally, the testing simulated bursty traffic patterns, where data transmission occurs in sudden, intense bursts followed by periods of low activity, to evaluate efficiency and stability under varying load conditions. This approach ensured that the I/O ports could sustain high performance levels during peak usage.

The testing process utilized simulations in Vivado to verify the functionality of each enhancement. Test cases were developed to exercise different aspects of the memory and I/O capabilities, ensuring that the increased capacity and additional ports were functioning as intended.

These performance measurements provided insights into how effectively the enhancements improved the RISC-based processor's capabilities, particularly in handling larger programs and facilitating more extensive data communication. The results of these tests demonstrated the successful implementation of the increased memory capacity and additional I/O ports, fulfilling the project's primary objectives.

D. Circuit Illustration

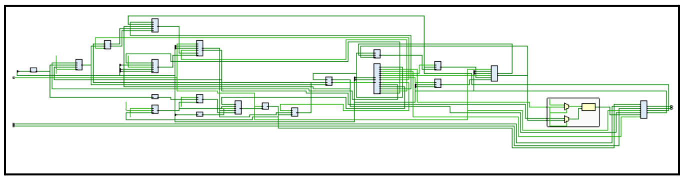


Fig. 1 Circuit diagram of enhanced RISC processor

The schematic of the enhanced processor, found in figure 1, shows the complexity of the modules for the computer set. While There are multiple modules found in in, the overall computer has a seamless flow that allows it to work with low latency and maximum output. The following figures describe each subsection of the enhanced architecture and the added capacity and ports.

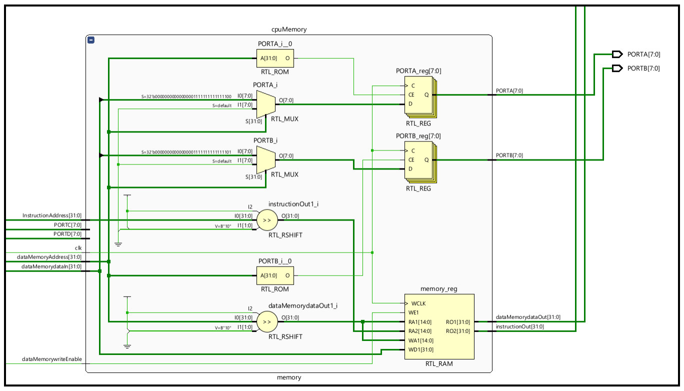


Fig. 2 CPU Memory Module

The CPU Memory Module schematic found in Figure 1.is responsible for managing botht he data and instruction memory for the CPU. It includes the parameters for the address width, memory depth, data width, and addresses for ports A, B, C, and D. The memory is implements as a 2 dimensional register array, where the read and write operations are perfromced based on the clock signal. If the ‘dataMemoryWriteEnable’ is high, the data at ‘datMemorydataIn’ is written to the memory address. The module is also responsible for outputting the current data and the instruction address continuously for other components to access. Additionally, it handles the specific addresses for ‘PORTA’ and ‘PORTB’, updating the values when the addresses are accessed.

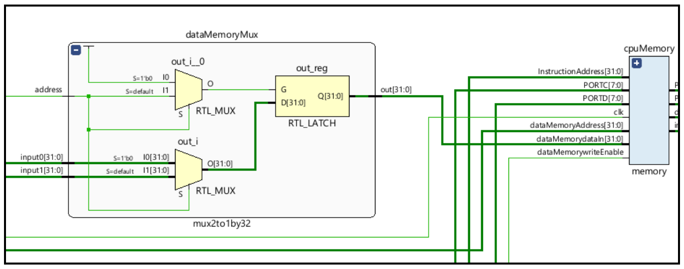


Fig. 3 Data Memory MUX Modules

The CPU takes input from the memory Multiplexer or multiplexer module for data as seen in figure 2. There are several mltiplexer modules used for the whole RISC architecture. The modules range from 8 by 1 32 bit to 4 to 1 by 5 bits. These multiplexers are used with varying input and output widths while route the input and outputs to respective locations. These multiplexer modules are fundamental components in digital design, enabling the selection of data paths based on control signals. They are commonly used in processors, communication systems, and various digital circuits to route signals efficiently and dynamically. By modularizing these components, the design becomes more manageable, reusable, and easier to understand. The use of parameterized widths and standard always blocks ensures that these modules are versatile and can be adapted to various data sizes and applications.

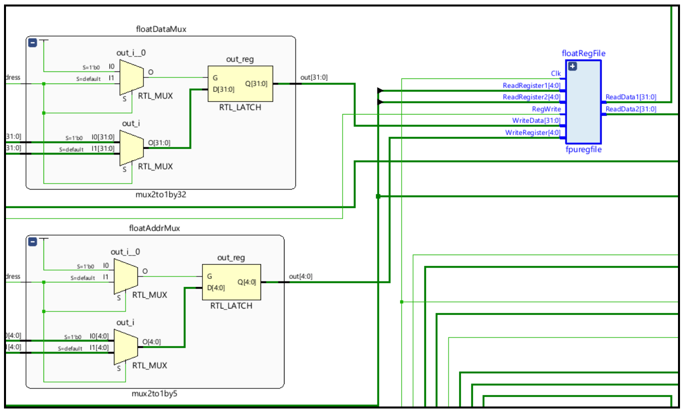


Fig 4. Float MUX and float Reg Modules

Figure 3 shows the schematic for the Multiplexer controllers for the float register file along with the input and outputs of the float register module. This module is responsible for storing data in a 32 bit wide array of registers. The module includes write data and read register inputs to determine which register to write various 32 bit wide data to. This module specifically handles storing float numbers for the user.

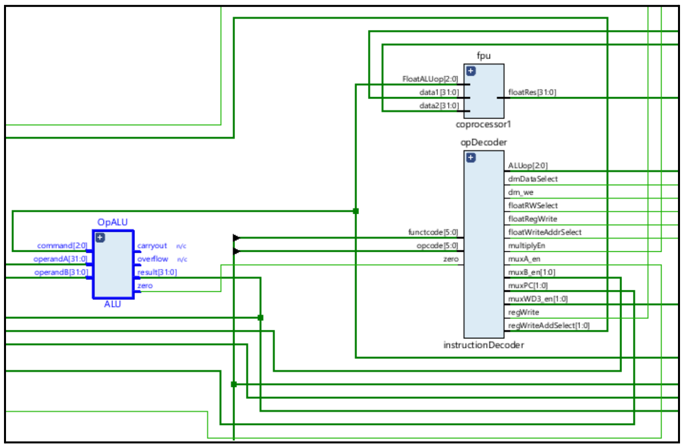


Fig 5. ALU, fpu and opcode Decoder Modules

Figure 5 shows the instruction decoder, copressor1, and ALU modules. The instructionDecoder module is central to interpreting the processor instructions. IT decodes opcodes and function codes to generate the correct control signals for the rest of the processor. This module supports a range of instructions, including R-type instructions (like ADD and SUB), load/store operations (LW and SW), branch instructions (BEQ and BNE), immediate operations (ADDI and XORI), and jump instructions (JUMP and JAL). It also handles floating-point instructions (FR, LWC1, SWC1) by setting up appropriate control signals for the floating-point unit. The coprocessor1 module deals with floating-point arithmetic operations. It integrates various components such as subtractors, shifters, sign extenders, and ALUs to perform operations on floating-point numbers. It adjusts significands and exponents based on the input data and operation type, performs arithmetic using the ALU, and normalizes the result.

Finally, the subtractor module is a simple yet essential component used within the coprocessor1 module. It calculates the difference between two input exponents

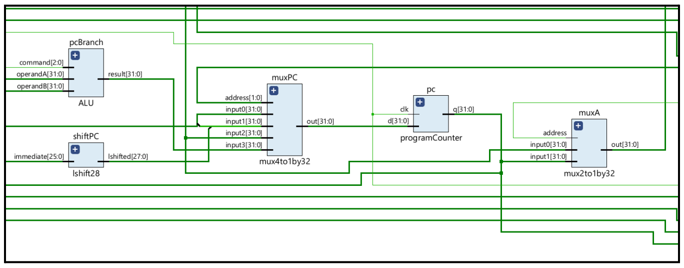


Fig 6. MUX and Program Counter Modules

Figure 6 shows the program counter along with the ALU program counter branch, shiftPC and a Multiplexer. The program counter is responsible for controlling and keeping track of which set of instructions to perform, it keeps a counter to move on to whichever instructions are pointed to. The ALU program counter branch handles conditional jumps based on computation results, while the shiftPC module adjusts the PC for branches and jumps by calculating address offsets. The Multiplexers selects between different input sources, such as the incremented address, branch targets, and jump addresses, to determine the next PC value. Together, these components manage instruction flow, handle control changes, and ensure the correct execution of programs.

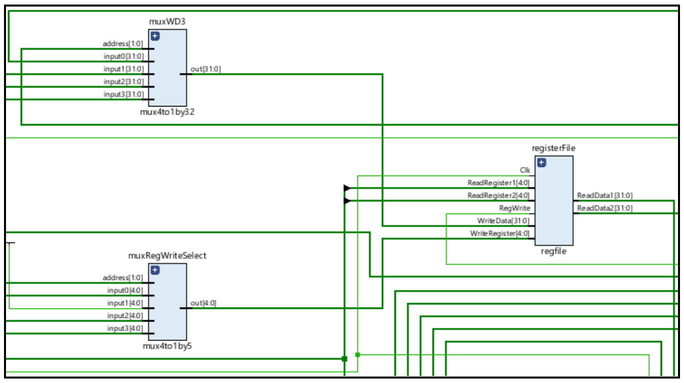


Fig 7. Main Register, MUX, and Write Modules

The schematic in figure 7. describes the RISC register file which handles a 32-bit wide register file with 32 registers. It features asynchronous read ports and a synchronous, positive edge-triggered write port. The regfile module uses a decoder1to32 to generate write enables for each register based on the WriteRegister address and the RegWrite signal. The module ensure that the registers' contents can be read and written correctly with the provided control signals and clock. The input for the register is two Multiplexers that select the write or WD3 functions of the register.

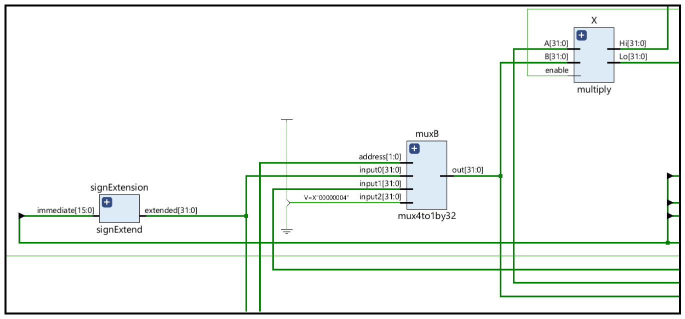


Fig 8. Sign Extension, MUX, and Multiply Modules

The sign extension module for the enhanced RISC processor is designed to extend a 16 bit value to 32 bit by preserving its sign. This is done by replicating the sign bit of immeidcate across the higher 16 bits of the 32 bit output. The multiply module performs a 32 bit mulitpociation of two inputs when the 'enable' signal is high. The result is split into two 32 bit outputs, the higher bit orders and the lower bit orders of the product.

III. Design Operation, Test Data and Statistical Analysis of Results

A. *Design Operation*

This project can be considered an improvement upon the single-cycle RISC-based processor initially given for this course's laboratory subject. The main goal for the completion of this project is the increase in memory capacity it would cause, as well as the addition of four I/O ports to the computer. This processor would also have an IEEE-754-compliant floating point unit (FPU) that would allow it to perform multiplication operations, floating point addition, and floating point subtraction of large or very small numbers. All the instructions within its instruction set will complete within a single cycle, allowing for it to be useful for simulation and visualization.

The main areas of improvement for the project are the memory modules of the CPU, where the addressing bits were increased and the ports were added. By increasing the bits it could handle to 32, the amount of memory this computer could handle was effectively increased to a total of 4096 MB or 4 GB. The ports are directly connected to the main CPU memory module due to its more simplified nature, allowing input or output to be directed to or form it instantly.

B. *Sorting Algorithm*

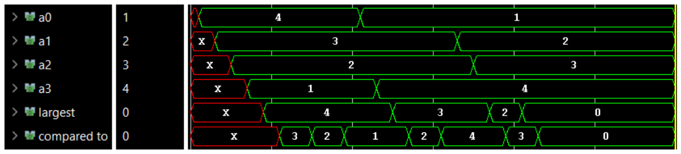


Fig 9. Bubble Sort Algorithm

In order to test the created architecture's ability to perform calculations and memory operations, a sample bubble sort was performed using the synthesized processor. The input array consisted of 4 numbers, stored in the registers a0 to a3. The array was 4, 3, 2, and 1. To perform a bubble sort, the computer goes through every adjacent pair of elements in an array and sees if the initial value is smaller than the next. Should that be the case, their positions are swapped, and the program goes on to the next pair. The sort will then continue until the end of the array, at which point it starts at the beginning once more but stops at the second-to-the-last element in its last run. It continues to do this until it can only compare one pair. The timing diagram shows this with the values of a0 to a3 changing with each successive clock cycle. The t0 and t1 registers have also had their wave names changed to largest and compared for clarity. Eventually, once the program is completed, the values of a0 to a3 are now 1, 2, 3, and 4 in that order.

C. Floating Point Addition and Subtraction

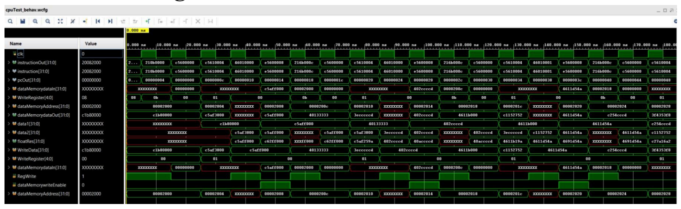


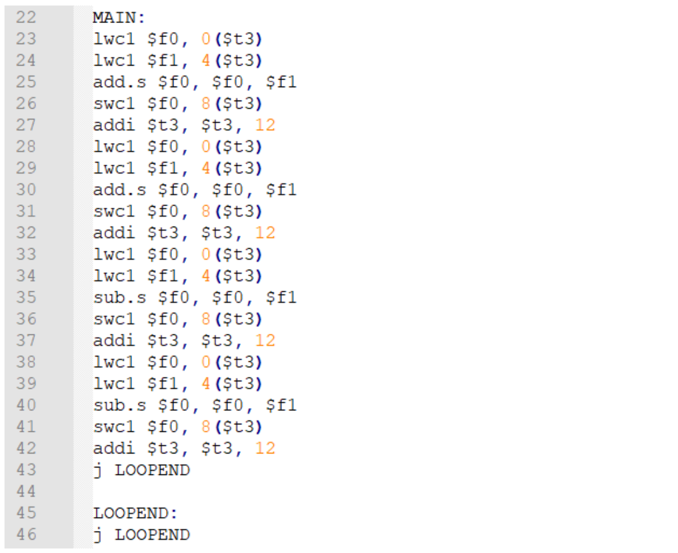
Fig 9. Addition and Subtraction of Floating Point Numbers

To showcase the ability of our processor to perform operations on floating point numbers, a simulation showing how it would perform these operations is shown above. At clock cycle 5, a floating point addition operation can be seen to be performed between the hexadecimal values c1b80000 and c5af3000. The timing diagram shows these waveforms to be labeled as data1 and data2, which are the inputs to the FPU as shown in the schematics of the previous section. The result of a floating point operation performed by the modules can be seen in the waveform labeled floatRes, which is the output of the FPU. The addition operation resulted in c5aff000, which is the correct answer to the equation.

The FPU created for the project was also able to perform floating point subtraction, specifically between numbers formatted in IEEE-754 (the same format used for floating point addition). This can be seen at clock cycle 17, where the values of the data1 and the data2 waveforms are 4611d54a and 4691d54a respectively. Once a floating point subtraction operation is performed on these numbers, it should result in a hex value of 3f4353f8. Looking at the value of floatRes at clock cycle 18, the operation seems to have been performed successfully.

Given that IEEE-754 allows for the encoding of values with an upper limit of 3.402823466 x 10^{38}, it can be used to perform operations that would be way out of reach if performed using integer addition and integer subtraction. The IEEE - 754 format also allows for negative values much lower than possible by even long integers, with a minimum of 1.175494351 x 10^{-38}. This allows the floating point unit to work with numbers much higher or much lower than what would normally be possible for a regular arithmetic logic unit.

The logic behind the test is shown in the assembly code below:



```
MAIN:
lwc1 $f0, 0($t3)
lwc1 $f1, 4($t3)
add.s $f0, $f0, $f1
swc1 $f0, 8($t3)
addi $t3, $t3, 12
lwc1 $f0, 0($t3)
lwc1 $f1, 4($t3)
add.s $f0, $f0, $f1
swc1 $f0, 8($t3)
addi $t3, $t3, 12
lwc1 $f0, 0($t3)
lwc1 $f1, 4($t3)
sub.s $f0, $f0, $f1
swc1 $f0, 8($t3)
addi $t3, $t3, 12
lwc1 $f0, 0($t3)
lwc1 $f1, 4($t3)
sub.s $f0, $f0, $f1
swc1 $f0, 8($t3)
addi $t3, $t3, 12
j LOOPEND

LOOPEND:
j LOOPEND
```

Fig 10. Assembly Program for Addition and Subtraction Test

D. Integer Multiplication and Division

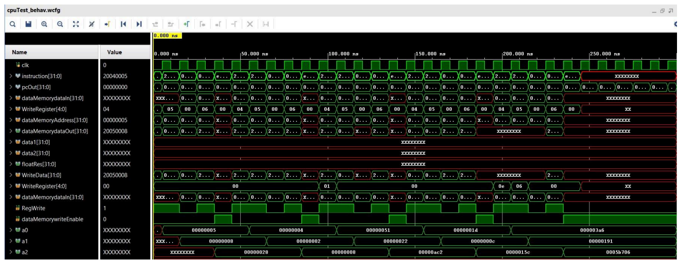


Fig 11. Multiplication and Division of Integers

Figure 11 shows the sample implementation of multiplying integers within the MIPS architecture. In analyzing the timing diagrams, the waveform signals clk, a0, a1, and a2 are of utmost importance. The a0, a1, and a2 registers, which are typically used for passing arguments to functions, were originally named registers 4, 5, and 6, respectively. The variable names were changed to easily compare the results of the waveforms to the assembly program.

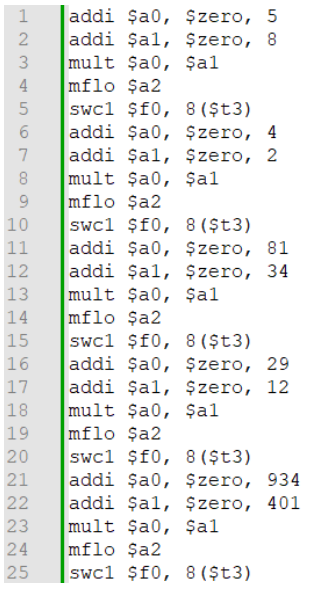
Each mult instruction takes four cycles to complete, and as shown in the figure, five multiplication tests were implemented, producing a total of 20 clock cycles. In the first four clock cycles, the hex values of 00000005 and 00000007 were loaded into the data register files, a0 and a1. The result after performing hex multiplication to the said values is

00000028, as shown in the a2 data register. The same process of loading two hex values, multiplying them, and storing the result in the specified register file is repeated at clock cycles 5, 10, 15, and 20. Additionally, to show the enhanced capabilities of the architecture's memory, the first two tests involved single-digit multiplication, the next two tests involved two digits, and so on.

Another important aspect of the multiplication instruction is the separation of the higher 32 bits (Hi) from the 32 lower bits (Lo). Since the test values only involved up to three hex digits, only the lower bits were utilized and stored in the a2 data register.

Furthermore, based on the implementation of the IEEE 754 format, the maximum allowed value of the integer is 2,147,483,647, and the minimum is -2,147,483,648.

The assembly program of the multiplication test is shown in Figure 12.



```
addi $a0, $zero, 5
addi $a1, $zero, 8
mult $a0, $a1
mflo $a2
swc1 $f0, 8($t3)
addi $a0, $zero, 4
addi $a1, $zero, 2
mult $a0, $a1
mflo $a2
swc1 $f0, 8($t3)
addi $a0, $zero, 81
addi $a1, $zero, 34
mult $a0, $a1
mflo $a2
swc1 $f0, 8($t3)
addi $a0, $zero, 29
addi $a1, $zero, 12
mult $a0, $a1
mflo $a2
swc1 $f0, 8($t3)
addi $a0, $zero, 934
addi $a1, $zero, 401
mult $a0, $a1
mflo $a2
swc1 $f0, 8($t3)
```

Fig 12. Assembly Program for Multiplication and Division

E. Architecture Enhancement Tests

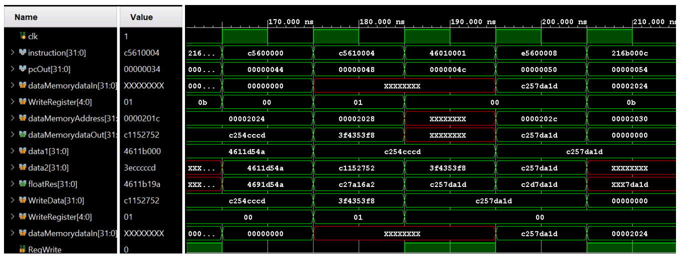

Name
Value
170.000 ns
180.000 ns
190.000 ns
200.000 ns
210.000 ns
clk 1
instruction[31:0] c5610004
pcOut[31:0] 00000034
dataMemorydataIn[31:0] XXXXXXXX
WriteRegister[4:0] 01
dataMemoryAddress[31: 0000201c
dataMemorydataOut[31: c1152752
data1[31:0] 4611b000
data2[31:0] 3eccccd
floatRes[31:0] 4611b19a
WriteData[31:0] c1152752
WriteRegister[4:0] 01
dataMemorydataIn[31:0] XXXXXXXX
RegWrite 0
216... c5600000 c5610004 46010001 e5600008 216b000c
000... 00000044 00000048 0000004c 00000050 00000054
000... 00000000 XXXXXXXX c257da1d 00002024
0b 00 01 00 0b
00002024 00002028 XXXXXXXX 0000202c 00002030
c254cccd 3f4353f8 XXXXXXXX c257da1d 00000000
4611d54a c254cccd c257da1d
XXX... 4611d54a c1152752 3f4353f8 c257da1d XXXXXXXX
XXX... 4691d54a c27a16a2 c257da1d c2d7da1d XXX7da1d
c254cccd 3f4353f8 c257da1d 00000000
00 01 00
000... 00000000 XXXXXXXX c257da1d 00002024

Fig 13. Timing Diagram for Memory Addresses

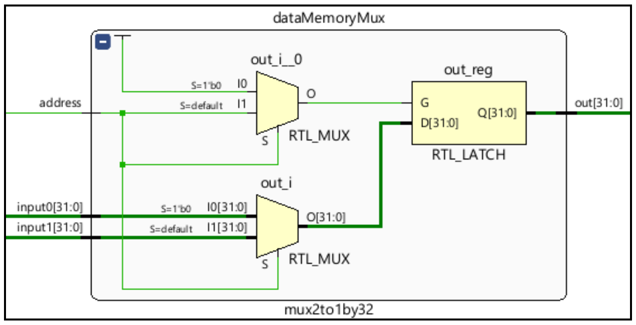


Fig 14. Data Memory Multiplexer

As shown in the timing diagram and the schematic diagram above, the memory addressing of the designed processor has effectively been increased to 32 bits. From the timing diagram, both the dataMemoryAddress and the dataMemorydataOut waveforms consist of 8 hex digits, which convert to 32 binary bits. There are also numerous values in the simulation with all the digits populated, indicating that it is usable up to its 32-bit limit. To obtain the total memory capacity of this implementation, the power of 2 raised to the number of bits must be obtained, which will be equal to 4294967296, 4 GB.

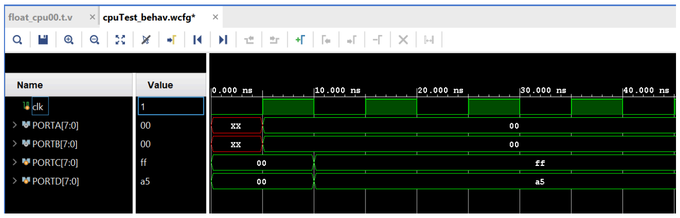


Fig 15. I/O Ports Timing Simulation

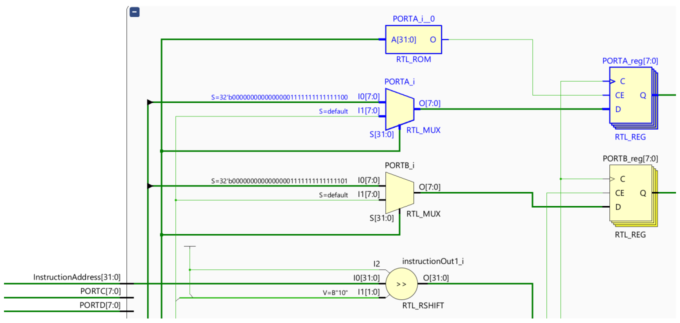


Fig 16. I/O Ports Schematic Diagram

The next architectural improvement was the addition of input and output ports to the program. As shown in the timing diagram (Figure 15), the ports added to the architecture were changed from uninitialized to 0 and from 0 to either ff or a5. These input ports were 8 bits each, but 4 were added to the architecture to allow for a 32-bit input/output when combined. Each bort contained the input/output wire from the other parts of the processor, along with the wire that leads to outside the processor. They also had their respective control registers for dictating whether they would be used as input or output vehicles, which would depend on what the instruction being used is.

F. Statistical Analysis

TABLE 2
MEMORY INCREASES AS MORE BITS ARE ADDED

Number of Bits	Maximum Memory (2^n)	IEEE-754 numbers which can be stored
2	4 bytes	1 number
4	16 bytes	4 numbers
8	256 bytes	64 numbers
16	65536 bytes (65+ kb)	2048 numbers
24	16777216 bytes (16+ mb)	5242888 numbers
32	4294967296 (4 gb)	1073741824 numbers

As shown in the table above, the limits of how much memory can be allocated are kept in check by how many bits a processor can use for addressing. Although the initial limitations were simply how much hardware could physically store, the software side of computers also carried some limitations with regard to memory allocation. To illustrate how increasing the number of addressing bits increases the data a processor can store, Table 2 likens each common bit number to how many IEEE-754 numbers it can store, with the amount each number of bits can store jumping exponentially as more bits are added. Once the number of bits reaches 32, which is how much this implementation can store, a million IEEE-754 numbers can be kept in the memory of the computer.

TABLE 3
I/O PINS AND COMMON CONNECTORS

Number of Pins	Connectors That Fit the Minimum	Connector Pin Counts
8	7	USB Type - A, (4-Pin), USB Type - B (4-Pin), mini USB Type A (5-Pin), mini USB Type B (5-Pin), micro USB Type A (5-Pin) micro USB Type B (5-Pin) RJ45 (8-Pin)
16	4	USB 3.0 Type - A, (9-Pin), USB 3.0 Type - B (9-Pin), micro USB 3.0 Type B (10-Pin) VGA (15-Pin)
24	6	HDMI Type A (19-Pin), Mini-HDMI Type C (19-Pin), Micro-HDMI Type D (19-Pin), DisplayPort (20-Pin), Mini DiplayPort (20-Pin), USB–C (24-Pin)
32	0	N/A

The table above shows the various common cable connectors used today, as obtained from [2] and [3]. Some of the connectors in the reference were omitted due to them being archaic (FireWire) or having had minimal adoption (DVI). Although the table above suggests that 8 pins can be enough to support most modern connectors for input and output, modern computers typically need more than one USB Type-A connector in conjunction with possible display connectors. To show this, the chart below shows how many USB Type-A ports or other popular port configurations each number of pins could support.

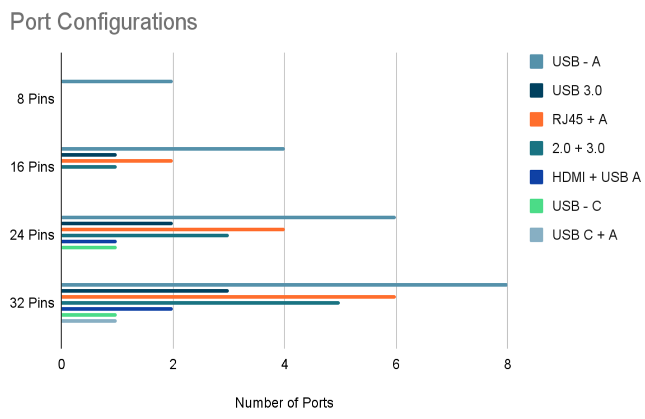


Fig 17. Possible Port Configurations for Each Number of Pins

As shown in the chart above, there is a visibly large difference between the number of ports each number of pins can support for the popular configurations listed. The first configuration is simply USB Type-A ports, which an 8-pin processor can connect to two of, with two more ports being possible for each 8-fold increase in pins. Two Type-A ports seem to be the limit of what an 8-pin processor can likely do. For example, for the HDMI + USB-A configuration, which was popular for laptops in the mid-2010s, even a 16-pin processor would be unable to support it, with a 24-pin one being able to at least support one more USB-A port, while a 32-pin could support at least 2 more USB-A ports. For the current most popular connector USB-C, only a 24-pin and 32-pin processor would be able to support it, with a 32-pin processor even being able to support another USB-A port.

TABLE 4
TWO-WAY ANNOVA STATISTICAL ANALYSIS

Source of Variation	SS	df	MS	F	P-value	F crit
Rows	75.81	6	12.63	24.49	4.53E-06	2.996
Columns	23.14	2	11.57	22.43	8.83E-05	3.885
Error	6.19	12	0.52			
Total	105.1	20				

The last table above shows the statistical testing between the number of ports each number of pins could support. A Two-Way ANOVA test was used as the difference between multiple of groups was being tested for. Looking at the obtained p-values, and comparing them to the test's level of significance (0.05), it can be said that there is a significant difference between the number of ports each pin amount can support. It can then be assumed that adding memory and more ports to a processor implementation improves its usability and possible real-world performance.

IV. CONCLUSION AND FUTURE INVOLVEMENTS

The single-cycle 32-bit MIPS architecture was successfully modified to increase memory and I/O ports. Memory was successfully increased from one gigabyte to four gigabytes. Through modifying the control and memory units, the addition of the four I/O ports were done. The requirement of an IEEE-754-compliant floating point unit to perform more complex arithmetic operations was added. Through the simulation, it was shown that the increase in memory was successfully done. To test the capability of the computer, some arithmetic operations such as floating point addition, floating point subtraction, multiplication, and division were done and it was able to carry out operations correctly.

Future involvements in this project could work on modifying the instruction set architecture such that the system would be more efficient on performing complicated operations. Additionally, to further improve the capability of the system it is suggested that the IEEE-754-compliant floating point unit to support more operations and wider precision levels. Furthermore, improvements done in this project could also be applied to other computer architectures such as CISC or ARM. Not only can the system be translated into those other architectures but it is also suggested that they may be integrated with one another.

REFERENCES

[1] GeeksforGeeks, "RISC and CISC in Computer Organization," GeeksforGeeks, Jul. 23, 2018. https://www.geeksforgeeks.org/computer-organization-risc-and-cisc/ (accessed Aug. 07, 2024).

[2] [. Commons, "File:Cable connector reference Chart.pdf — Wikimedia commons, the free media repository," 2024. https://commons.wikimedia.org/w/index.php?title=File:Cable%3Csub%3EC%3C/sub%3Eonnector%3Csub%3ER%3C/sub%3Eeference%3Csub%3EC%3C/sub%3Ehart.pdf&oldid=845113366

[3] "Understanding USB-C Pinout Requirements," *Ultra Librarian*, Jun. 14, 2023. https://www.ultralibrarian.com/2023/06/22/understanding-usb-c-pinout-requirements-ulc