Computer Architecture

Single-cycle MIPS

Eduardo García Olmos A01351095
Erick Alberto Contreras Aguayo A01630105
José Luis Jiménez Arévalo A01633245
Luis Alfredo Aceves Astengo A01229441

October 25th, 2019

ISA

Our objective was to design and model a single-cycle MIPS with Interlocked Pipeline Stage, we had already a base model for this since we designed and modelled in a previous assignment, it supported R-Type and I-type instructions but without the Interlocked Pipeline Stage. This time we included J-Type instructions as well, we have R-type instructions that are useful to make operations and save data between registers, I-type instructions to read write instructions from the data memory and J-type instructions to make program counter moves.

The designed architecture works as follows, we designed a controller that can read instructions and send flags to our different modules in order to enable MUX modules, decoders and logic gates to select the desired path for our data, an instruction decoder that selects the operation that must be done and sends that information to our ALU, our ALU that is capable of doing a variety of operations and a register file of 32 directions that allows us to move data between instructions.

Design Flow

The design flow for our project was very straight forward, we started with a paper design from our las design exercise number two, from then we started thinking about the modules we would need to add or modify for the proper functionality of the J-type instructions. Once we identify the changes that needed to be made, we extended our encoding so it will handle the new instructions. Once we had a clear idea of the changes, we needed to make them in the code, we made them and where able to test them using the simulation.

As we were testing our instructions, we realize we still needed to add some logic for negative operands for the multiplication algorithm. We realize that with our inputs a and b our algorithm only worked if a was either positive or negative but it didn't worked with b having a signed behavior, but we were able to fix it easily by adding some extra logic steps.

Encoding

Table 1: Encoding of architecture.

Instruction type	**opcode**	**Instruction**	
		func	operation
R-type	000000	000000	ADD
		000001	SUB
		000010	AND
		000011	OR
		000100	XOR
		000101	SLL
		000110	SRL
		000111	SLA
		001000	SRA
I-type	000001	ADDI	
	000010	SUBI	
	000011	ANDI	
	000100	ORI	
	000101	XOR	
	000110	LUI	
	000111	LLI	
	001000	LWR	
	001001	SWR	
	001010	LWI	
	001011	SWI	
	001100	BEQ	
	001101	BNE	
J-type	001110	J	
	001111	JAL	
	010000	RET	

The instructions shown in table 1, are the ones implemented in our original architecture. However, in order to support a signed multiplication, some instructions had to be added:

- BNEG: Branch on negative. Treated as an I-type instruction with opcode 010001, it changes the program counter to an immediate value if a register's value is less than 0. A negative flag in the ALU was also implemented for this instruction to work. We used this instruction to handle the cases when a multiplication operand was negative.
- NOT: Not. Treated as an R-type instruction with func 001011, it performs a bitwise not to a register's value, and stores it in a register. We used this instruction to change a number's sign in two's complement format.

Collaboration

Table 2: Collaboration matrix.

Task	Student's name	Contribution (%)
Schematic of Microarchitecture	Eduardo	50
	Erick	50
	José Luis	0
	Luis Alfredo	0
Programming	Eduardo	25
	Erick	0
	José Luis	25
	Luis Alfredo	50
Testbench	Eduardo	0
	Erick	25
	José Luis	50
	Luis Alfredo	25
Overall	Eduardo	25
	Erick	25
	José Luis	25
	Luis Alfredo	25

RTL view

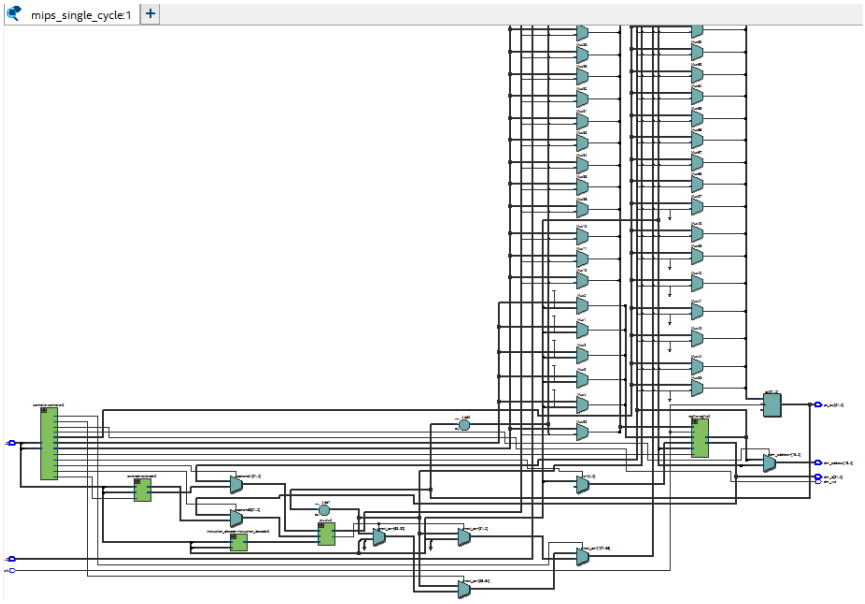

Figure 1: Design synthesis in Quartus.

mips_single_cycle