Overview

My project is a MIPS processor implemented in Vivado 2023.2 using VHDL, that includes the basic stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). Each phase of the MIPS is represented by a specific VHDL component:

1. Instruction Fetch (IF): IF.vhd

This stage fetches the next instruction from the instruction memory. It uses the program counter (PC) to index the instruction memory (ROM), increments the PC, and handles branches and jumps.

2. Instruction Decode (ID): ID.vhd

This stage decodes the fetched instruction, reads registers, and sign-extends immediate values if necessary. It also determines the control signals required for the execution phase.

3. Execute (EX): EX.vhd

This stage performs arithmetic or logic operations based on the decoded instruction. It calculates addresses for branch instructions and performs comparisons for conditional branches.

4. Memory Access (MEM): MEM.vhd

This stage handles memory read and write operations. It uses the ALU result to access data memory and handles data forwarding to the next stage if necessary.

5. Write Back (WB): Handled within the ID and MEM components.

This stage writes the results back to the register file. It either writes the result of an ALU operation or data loaded from memory.

The project has 5 other additional files:

1. test_env.vhd - The top level component which puts all the components together.
2. MPG.vhd - Generates enable signals for single pulse operations based on the input buttons.
3. SSD.vhd - Displays the output on a seven-segment display.
4. MainControl.vhd - Generates control signals based on the opcode of the instruction. These signals are RegDst, ExtOp, ALUSrc, Branch, Jump, MemWrite, MemtoReg, RegWrite, BrNE, and ALUOp.
5. testbench.vhd - A simulation file used to verify the functionality of the MIPS processor by running a series of test cases that simulate the operations the processor would perform during execution. It includes checks that verify the outputs of the MIPS processor match the expected results. More specifically, it is checking register values, memory contents, or specific signal states at certain times. This helps identify any errors in the processor's design before it is synthesized into actual hardware.

In the IFetch file, there is an example execution of a program:

1. Initialize R1 with 10: *addi $1, $0, 10*
2. Initialize R2 with 15: *addi $2, $0, 15*

3. Store R2 - R1 in R3: *sub $3, $2, $1*
4. Store R2 + R1 in R4: *add $4, $2, $1*
5. Store value of R3 at memory address 6 + R1: *sw $3, 6($1)*
6. Load from address R2 + 1 to R2: *lw $2, 1($2)*
7. Shift right R1 by 1: *srl $1, $1, 1*
8. Branch if R1 == R2: *beq $1, $2, 1*
9. Jump to instruction 4: *j 4*
10. Store value of R1 at address R2 + 2: *sw $1, 2($1)*
11. Subtract R2 from R4 and store in R4: *sub $4, $4, $2*
12. Shift left R1 by 1: *sll $1, $1, 1*
13. Branch if R1 == R4: *beq $1, $4, 1*
14. Jump to instruction 11: *j 11*
15. Store value of R1 at address 5: *sw $1, 5($0)*

I made a table with the control signals to have a clear reference for understanding how each instruction affects the control signals:

Instruction	**Opcode** *Instr(15-13)*	**RegDst**	**ExtOp**	**ALUSrc**	**Branch**	**BrNE**	**Jump**	**MemWrite**	**MemtoReg**	**RegWrite**	**ALUOp (1:0)**	**func** *Instr(2-0)*	**ALUCtrl (2:0)**
add	000	1	x	0	0	0	0	0	0	1	00(R)	000	000(+)
Sub	000	1	x	0	0	0	0	0	0	1	00(R)	001	001(-)
SLL	000	1	x	0	0	0	0	0	0	1	00(R)	010	101(<<l)
Srl	000	1	x	0	0	0	0	0	0	1	00(R)	011	110(>>l)
and	000	1	X	0	0	0	0	0	0	1	00(R)	100	010(&)
Or	000	1	X	0	0	0	0	0	0	1	00(R)	101	011(|)
Xor	000	1	x	0	0	0	0	0	0	1	00(R)	110	100(^)
Sra	000	1	X	0	0	0	0	0	0	1	00®	111	111(>>a)
Addi	001	0	1	1	0	0	0	0	0	1	01(+)	X	000(+)
Lw	010	0	1	1	0	0	0	0	1	1	01(+)	X	000(+)
Sw	011	0	1	1	0	0	0	1	X	0	01(+)	X	000(+)
Beq	100	X	1	0	1	0	0	0	X	0	10(-)	X	001(-)
Ori	101	0	0	1	0	0	0	0	0	1	11(|)	X	011(|)
Bne	110	X	1	0	0	1	0	0	X	0	10(-)	X	001(-)
j	111	x	x	x	X	0	1	0	x	0	xx	x	Xxx