In this project a single cycle MIPS architecture needed to be extended in order to implement various instructions. The project was successfully completed with all instructions functioning properly.

The first group of instructions that were implemented are: add, sub, slt, andi, nor. In order for these instructions to be implemented the only thing that needed to be changed was the mips-control.v file. Similar to the already implemented addi and lui, the format of adding these new instructions was the same. The only difference between the instructions are the cases that correspond to the opcode and the function code and the control signals that tell the CPU what to do. Below is a table that shows the test used to verify the implementation.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PC | Instruction | opcode | func | Write Register Address | Write Register Data | Write Register Control |
| 0x00400000 | sll | 0x00 | 0x00 | $zero | 0x00000000 | no |
| 0x00400004 | lui | 0x0f | Don’t Care | $s2 | 0x10000000 | yes |
| 0x00400008 | addi | 0x08 | Don’t Care | $s0 | 0x00000005 | yes |
| 0x0040000c | addi | 0x08 | Don’t Care | $s1 | 0x00000000 | yes |
| 0x00400010 | add | 0x00 | 0x20 | $t0 | 0x10000000 | yes |
| 0x00400014 | sub | 0x00 | 0x22 | $t1 | 0xf0000005 | yes |
| 0x00400018 | slt | 0x00 | 0x2a | $t2 | 0x00000001 | yes |
| 0x0040001c | andi | 0x0c | Don’t Care | $t3 | 0x00000001 | yes |
| 0x00400020 | nor | 0x00 | 0x27 | $t4 | 0xffffffff | yes |

The second group of instructions that were implemented were lw and sw. To implement these two instructions the mips.v was heavily modified. Also, the mips-control.v file was also modified to handle the two new instructions. An instantiation of the Data Memory module had to be used and also a 32-bit 2-to-1 multiplexer. The Data Memory module took in as inputs the memory write control signal, memory read control signal, result of the ALU, and the second output of the register file module. The Data Memory output and the result of the ALU were fed into a multiplexer, called dataMemMux. It determined whether the ALU result or the data memory output needed to be sent to the write data input of the register file. The tables below show the test used to verify the implementation.

Register File Waveform Results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PC | Instruction | Immediate | Write Register Address | Write Register Data | Write Register Control |
| 0x04000000 | sll | 0 | $zero | 0x00000000 | no |
| 0x04000004 | lui | 0x1000 | $s2 | 0x10000000 | yes |
| 0x04000008 | sw | 0 | Don’t Care | Don’t Care | no |
| 0x0400000c | lw | 0 | $t1 | 0x00000000 | yes |

Data Memory Waveform Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PC | Instruction | Data Memory Address | Output Data Memory | Data Memory Write | Read Data Memory Control | Write Data Memory Control |
| 0x04000000 | sll | Don’t Care | Don’t Care | Don’t Care | no | no |
| 0x04000004 | lui | Don’t Care | Don’t Care | Don’t Care | no | no |
| 0x04000008 | sw | 0x10000000 | Don’t Care | 0x00000000 | no | yes |
| 0x0400000c | lw | 0x10000000 | 0x00000000 | Don’t Care | yes | no |

The third group of instructions that were implemented beq and bne. The mips-control file was manipulated to handle the two new instructions and give the proper control signals. Several things were modified in the mips.v file. A shift left 2 module was instantiated and the input of the module is the output of the sign extend module. A 32-bit adder was also added to calculate the potential branch destination and took in as input the program counter plus 4 and the output of the shift module. A multiplexer was added that determined whether the branch calculation or program counter plus 4 is sent to the program counter register. Table below shows the test used to verify the implementation. I also added an AND gate and a XOR gate. These two gates determined if the current instruction is beq or bne and the output of the result of these gates are used as the control signal for the multiplexer I added. Below is a table used to verify the implementation. Only the instructions that were executed were shown

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PC | Instruction | Zero Signal | Jump Control | Branch Control |
| 0x00400000 | sll | yes | no | no |
| 0x00400004 | lui | no | no | no |
| 0x00400008 | beq | yes | no | yes |
| 0x00400010 | addi | no | no | no |
| 0x00400014 | bne | yes | no | yes |
| 0x0040001c | addi | no | no | no |

The fourth group of instruction had the j, jr and jal instructions implemented. Like the previous groups of instructions the mips-control.v file was modified to handle these instructions and give appropriate output. For the jump instruction to function a shift left two module was added, it took the last 26 bits of the instruction as input. The output of the module then had the first four bit of PC plus 4 attached to the beginning of it. Another multiplexer was added to determine if the branch multiplexer output or jump address were to be sent to the program register. The jump register instruction needed to have another multiplexer attached to the jump multiplexer. This multiplexer determined whether the jump multiplexer output or the value in the rs register need to be sent to the program counter. An and gate that took in the jump control signal and a bit of instruction was used as a control signal for the multiplexer. Finally, for the jump and link instruction two multiplexers and an AND gate was added. The AND gate took in the jump control signal and a bit from the instruction. The output of the the AND gate was used to determine if the instruction is a jump and link instruction. Since the operation of a jump and link instruction is similar to that of a jump instruction, nothing was changed on the program counter side of the MIPS processor. The two multiplexers were used to determine if register number 31 needed to be written to and data that potentially needed to written to said register. Below is a table to verify the implementation. Only instructions that were run are shown.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PC | Instruction | Jump Control | Branch Control | Write Register Address | Write Register Data | Write Register Control |
| 0x00400000 | sll | no | no | $zero | 0x00000000 | no |
| 0x00400004 | addi | no | no | $ra | 0x0 | yes |
| 0x00400008 | j | yes | no | Don’t Care | Don’t Care | no |
| 0x00400014 | addi | no | no | $s1 | 0x1 | yes |
| 0x00400018 | jal | yes | no | $31 | 0x00400020 | yes |
| 0x00400028 | addi | no | no | $s2 | 0x2 | yes |
| 0x0040002c | jr | yes | no | Don’t Care | Don’t Care | no |
| 0x00400020 | addi | no | no | $t4 | 0x4 | yes |
| 0x00400024 | addi | no | no | $ra | 0x00400034 | yes |
| 0x00400028 | addi | no | no | $s2 | 0x2 | yes |
| 0x0040002c | jr | yes | no | Don’t Care | Don’t Care | no |