## Design of the ALU

|  |  |  |
| --- | --- | --- |
| **opcode name** | **opcode** | **opext** |
| ADD | 0000 | 0101 |
| ADDI | 0101 | XXXX |
| ADDU | 0000 | 0110 |
| ADDUI | 0110 | XXXX |
| ADDC | 0000 | 0111 |
| ADDCU | 1010 | 0101 |
| ADDCUI | 1010 | 0110 |
| ADDCI | 0111 | xxxx |
| SUB | 0000 | 1001 |
| SUBI | 1001 | xxxx |
| CMP | 0000 | 1011 |
| CMPI | 1011 | xxxx |
| CMPU/I | 1010 | 0010 |
| AND | 0000 | 0001 |
| OR | 0000 | 0010 |
| XOR | 0000 | 0011 |
| NOT | 1010 | 0011 |
| LSH | 1000 | 0100 |
| LSHI | 1000 | xxxx |
| RSH | 0000 | 1110 |
| RSHI | 1110 | xxxx |
| ALSH | 1010 | 0001 |
| ARSH | 1010 | 0100 |
| nop | 0000 | 0000 |
| MOV | 0000 | 1101 |
| MOVI | 1101 | xxxx |

The design and implementation of the ALU closely followed the guidelines and suggestions from course material. All operations which were requested to be implemented in the ALU lab handout were successfully implemented, tested, and simulated.

The ALU operations that were implemented are shown in Table 1. This includes addition, subtraction, comparison, bit logical operations, shift, and move instructions. The opcodes assigned to each instruction were taken from the ISA handout. If the ISA handout did not include the particular instruction an opcode was selected which would not conflict with any opcodes currently assigned to the other instructions.

All ALU instructions that were planned were implemented. No plans have been made to implement multiplication or division. All additional instructions which will be implemented for the design of the computer are non-ALU instructions.

The ALU takes as input two 16 bit vectors A and B, a 4-bit opcode and a 4-bit opcode extension to describe what operation will be performed by the ALU. Also, a carry bit is taken as input from the processor status register. The carry input flag is only used for the add with carry instructions.

The ALU has two separate outputs. A 16 bit vector S, which is the result of performing the desired operation on the A and B inputs. A status flag is also emitted which signals certain conditions as a byproduct of performing the desired operation.

Table - ALU instructions with opcode

The status flags contain only 5 bits. The purpose of each bit is described in Table 2. The meaning of each bit has came from the CR16A Programmer's Reference Manual. All ADD and SUB instructions have the potential to set the flag bit. Only CMP instructions will set the low flag, zero bit, and negative bit values of the status flags. This was done for consistency with the CR16A Programmer's Reference Manual.

The code for the ALU consisted of a series of casex statements, one for each possible operation implemented. The body of the statement set the appropriate registers as described in the CR16A Programmer's Reference Manual. Many operations performed identical operations, such as ADD and ADDI, as well as every CMP instruction. This was done because of the inability to implement fall though in the Verilog language.

|  |  |  |
| --- | --- | --- |
| **CLFZN[4:0]** | **description** | **summary** |
| CLFZN[4] | carry | set when a carry is generated on an ADD instruction |
| CLFZN[3] | low flag | set in CMP instructions when (unsigned)B<(unsigned)A |
| CLFZN[2] | flag bit | signals overflow in ADD and SUB instructions |
| CLFZN[1] | zero bit | set in CMP instruction when A==B |
| CLFZN[0] | negative bit | set in CMP instructions when (signed)B < (signed)A |

Table 2 - Processor Status Register

The synthesis report for the ALU module showed that the delay for this circuit is 14.951ns, with 12 levels of logic. The path of the longest delay in the circuit is from opcode<3> to S<6>. Also, the report indicated 261 of 9312 4-input LUTs were being utilized for this design.

## Testing of the ALU

The ALU was tested using a sequence of coded tests. Each instruction that needed to be tested was fed predetermined input and the output was tested with the predetermined desired result of the operation. In addition several random values were generated as input to the ALU and the output was verified to be correct.

Also, the Post-Synthesis Simulation Model was compared with the Pre-Synthesis Model. A large number of random inputs were generated and fed into both models. The output of both were compared for equality with each other. No differences were found in this comparison. This showed that the synthesized model performed as expected from the description of the ALU provided.

## Future Plans for the ALU

The ALU is a basic unit of the Computer system being developed. All operations which were initially planned for the ALU were implemented. Some operations, however, will require some additional operations to be performed by another component before or after the ALU.

The difference between adding two values stored in registers and adding a register value to an immediate value is not noticeable to the ALU. When this is encountered the control logic will place the value of the register on one input to the ALU and do a sign extend on the immediate value and place that on the other input to the ALU. After this happens, the ALU will compute the result of the operation. This functionality will be implemented when the control logic is designed.

In the next section the design of the Register file will be explained. After the register file has been properly connected to the ALU many operations of the computer will be operational.