Our ALU design was built using seperate subunits to handle different operations. The logic operations and, or, nor, and or are computed in Logic32. This takes in two 32-input buses, and returns the result of each logic operations in andbus, xorbus, orbus, and norbus. Inside Logice32, these operations are computed bit by bit in a for loop of length 32. These bit by bit logic operations can be found in myparts.cast, and are the basic building blocks of our ALU design.

For Addition and Subtraction, we implemented a AddSub32 block, which calls AddSub. AddSub takes in two 1-bit inputs, binvert, carryin, and returns out and carryout. Binvert is 1 when input b needs to be inverted for subtracting. Out is the sum of the addition or subtraction (depending on the Binvert bit) of a and b. The logic equations for out and carryout are on page 234 and 233 of "Computer Organization and Design" by Patterson and Hennessy. AddSub32 performs addition and subtraction for 32-bit buses by calling AddSub 32 times. AddSub32 takes in Buses a and b, and nodes Bnegate and carryintemp. It returns 32-bit out bus and a carryout node. Bnegate and carryintemp will both be 0 for addition, or will both be 1 for subtraction. For subtraction, b must be inverted. This is accomplished by inverting all the bit's of b, and then adding one. This is why AddBus32 will have a carryintemp of 1 when it is performing subtraction. The carryout of one call to AddSub will be the carrin to the next call of AddSub.

Operations slt and sltu use AddSub32. Slt and Sltu use subtraction to determine whether A is less than B. Since Slt uses 2-complement, a will be less than b if a minus b equals a negative number. This is accomplished by looking at the most significant bit of a minus b, which will be 1 if a is less than b. Sltu uses regular binary numbers. If a is less than b, then a-b will have a carryout from the last bit AddSub operation. This is why AddSub32 needs an carryout node(or the carryout node can be used for overflow detection).

Sll, srl, and sra are implemented using a logarithmic shifter. This type of shifter uses five 32-to-1 Mux's. The first 32-to-1 mux takes the b bus to be shifted, and a b bus that has already been shifted one bit (either to the left or right depending on the shift). If the least significant bit of the shift amount is 0, then the original b passes through the mux. If the least significant bit of the shift amount is 1, then the shifted bus passes through the mux. The second mux takes the output of the first mux, and a bus which contains the output of the first mux shifted 2 bits. The second least significant bit determines which bus to choose. The third mux takes the output of the second mux, and a bus which contains the output of the second bus shifted 4 times. THis continues in a similar manner throught the fourth and fifth muxes(which shift the bus by 8 and 16 respectively). ShiftLeft takes in the bus to be shifted and an array of the shift amount. It returns the shifted bus. Bus positions are padded with 0 if no bits are shifted in. ShiftRight can either perform slr or sra. To handle both operations, ShiftRight must also take in a node arith. This node is 1 if it is performing sra, and 0 if it is performing srl. This works the same way as shiftleft except that it shifts everything right(Duh), and pads the empty positions with either a 1 if it is performing sra, or 0 if it is performing slr.

ALU calls the 11 above operations, and decides which one to output. It takes in buses a and b, and ALUctrl, and outputs a 32-bit bus. ALUctrl is used to determine which operations to output. The first eleven bits pick the operation, and the last 5 determine the shift amount when shifting. ALU calls Logic32, AddSub32,slt, sltu, sra,srl, and sll. Since only 1 of the 11 ALUctrl bits will be 1, the correct output can be determined by anding each bit of an output bus from an operation with the node that controls

that operation. This will cause only one operation to be passed through. The results of these and's are combined through or's to produce one output. This output will contain only the output of the correct operation.

All of these modules and submodules are shown in the files ALu1.ps to Alu6.ps respectively. THe files use fairly standard notation. A thin line means a one bit wire connecting two components, and a thick line signifies either a bus or an array or nodes. WHen only one line from the bus is desired, the line becomes thin again. In order to not document large numbers of circuit elements, whenever a loop calling for many or gates is coded, only the first and last gates are included in the diagram, with three horizontal dots in between them to signify missing gates.

Our worst case delay through the ALU resulted when we added all ones to 1, or in hex 0xffffffff +0x00000001. This gave us a delay of 37.66 nanoseconds, as measured in irsim

For testing we used the file testalu.src, where we anaylzed all of the inputs and outputs to the alu and where we tried different output values and checked them with hand calculations. We tested three inputs for each function and tried to get a variance with the inputs.