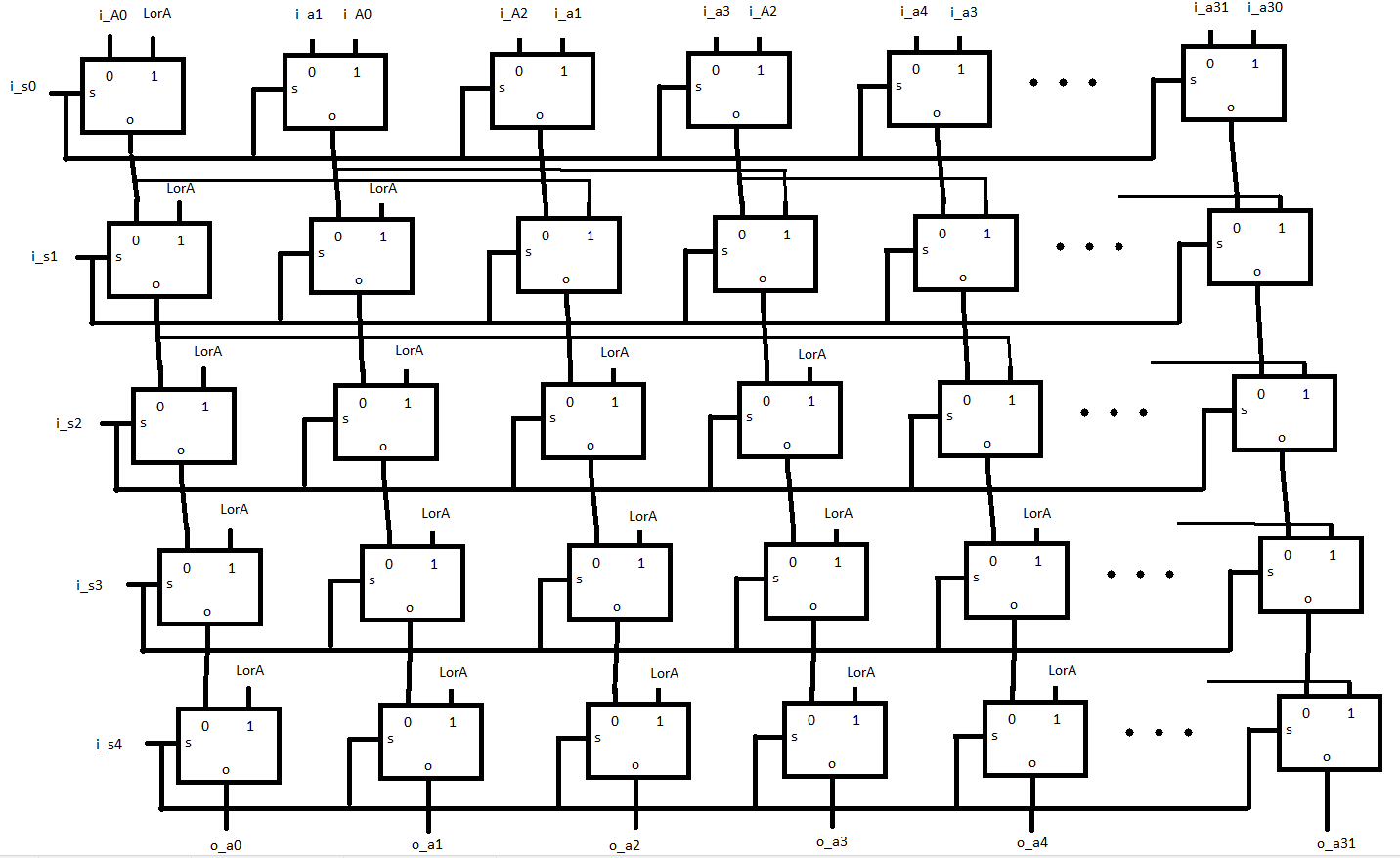
3) The MIPS ISA also contains several shifting instructions which can be implemented independently from the main ALU. For this part we will consider the barrel shifter design discussed in class.

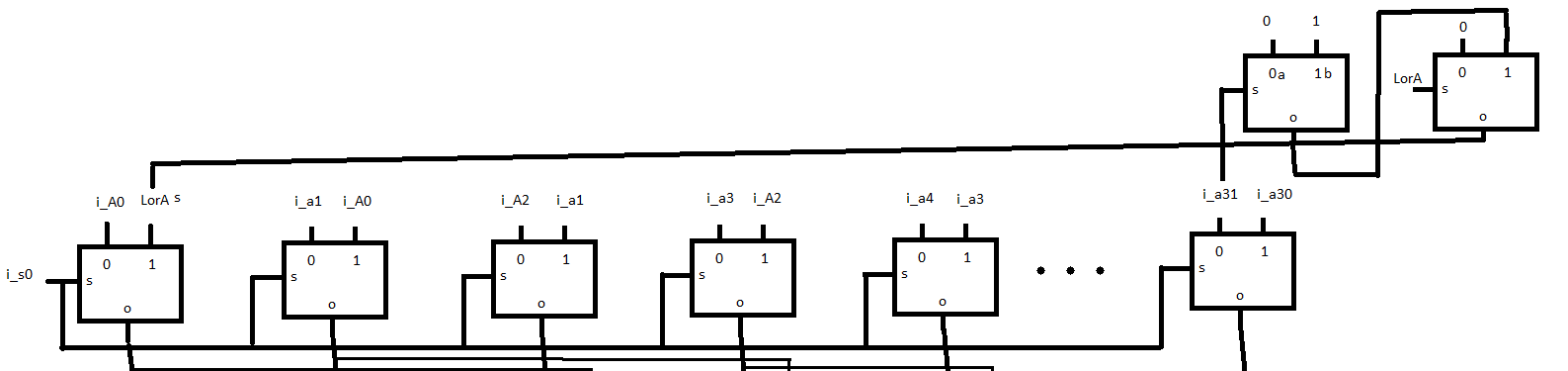
(a)Describe the difference between logical (srl) and arithmetic (sra) shifts. Why does MIPS not have a sla instruction?

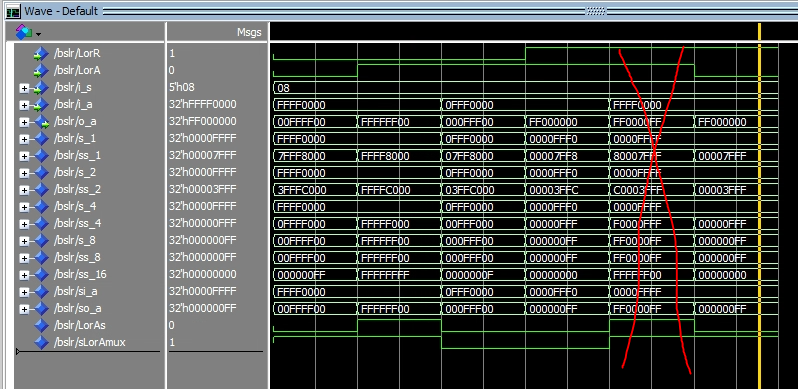
Srl is a logical operation used for unsigned numbers meaning we can pad the new shifted number. Sra is an arithmetic operation used for signed numbers because when shifting signed numbers right we need to pad with 1’s instead of 0’s. there is no need for a shift left arithmetic because the padding will be zero for both signed and unsigned numbers.

(b)Unfortunately, barrel shifters are not covered explicitly in the P&H textbook, but the following Java applet shows how an 8-bit barrel shifter can be created using cascaded 2:1 muxes (link). Implement a 32-bit right shifter (both arithmetic and logical) using structural VHDL. In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations.[There is a strong pattern to how the muxes are mapped in the conventional barrel shifter design. If you make sense of this pattern your code will be considerably simplified.]



I broke this problem up into 5 generate loops one for each bit of select line because each bit represents a shift by 1, 2, 4, 8, or 16 bits. As the data flows through each select bit set to 1 will apply a shift amount corresponding to the active bit. To implement the logical or arithmetic operations offered by the shifter I created a couple multiplexers to look at the most significant bit of the input and the logical of arithmetic input if both are 1 we know to append one’s on a right shift. There is no protection from trying a left arithmetic shift, it is down to the programmer to not allow this situation. (shown in wavewform below with a red X).

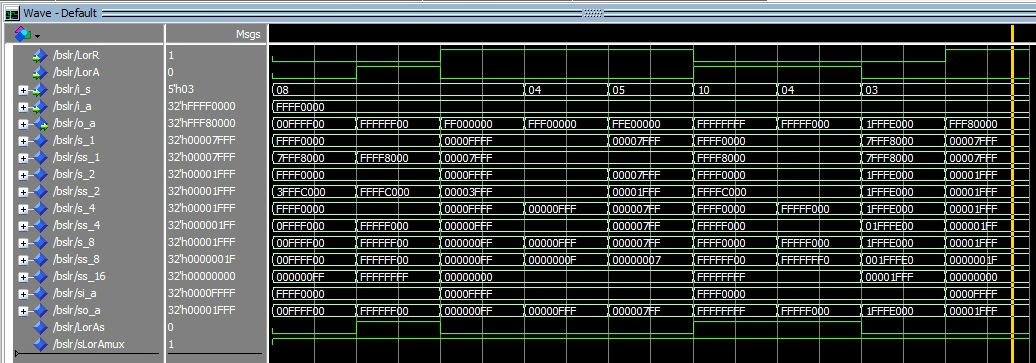




(c)In your writeup, explain how the right barrel shifter from part b) can be enhanced to also support left shifting operations. Integrate this functionality into your existing design. [Hint: it can be done by trivially modifying the input and output.An additional shifting component is not necessary.]

To do this we just need to invert the input before the shift and again invert back after the shift, this works because a backwards number shifted to the right is the same as a left shift. To do this I made a component called reverse that reverses the number if the left shift bit is set to 1. I added this component to the structure before and after the output.

(d)Use Modelsim to test your shifter designs thoroughly to make sure they are working as expected. Describe how the execution of the different shifting operations corresponds to the Modelsim waveforms in your writeup.



1. SRL by 8
   1. Testing a basic right shift
2. SRA by 8 bits
   1. Testing the arithmetic functionality
3. SLL by 8
   1. Testing basic left shift
4. SLL by 4 bits
   1. Testing basic left shift
5. SLL by 5
   1. Testing that each stage of the shift stacks correctly here we test stage 1 and 4
6. SRA by 10
   1. Testing combination of arithmetic and multistage shift
7. SRA by 4
   1. Testing of arithmetic shift
8. SRL by 3
   1. Basic right shift test
9. SLL by 3
   1. Basic left shift test