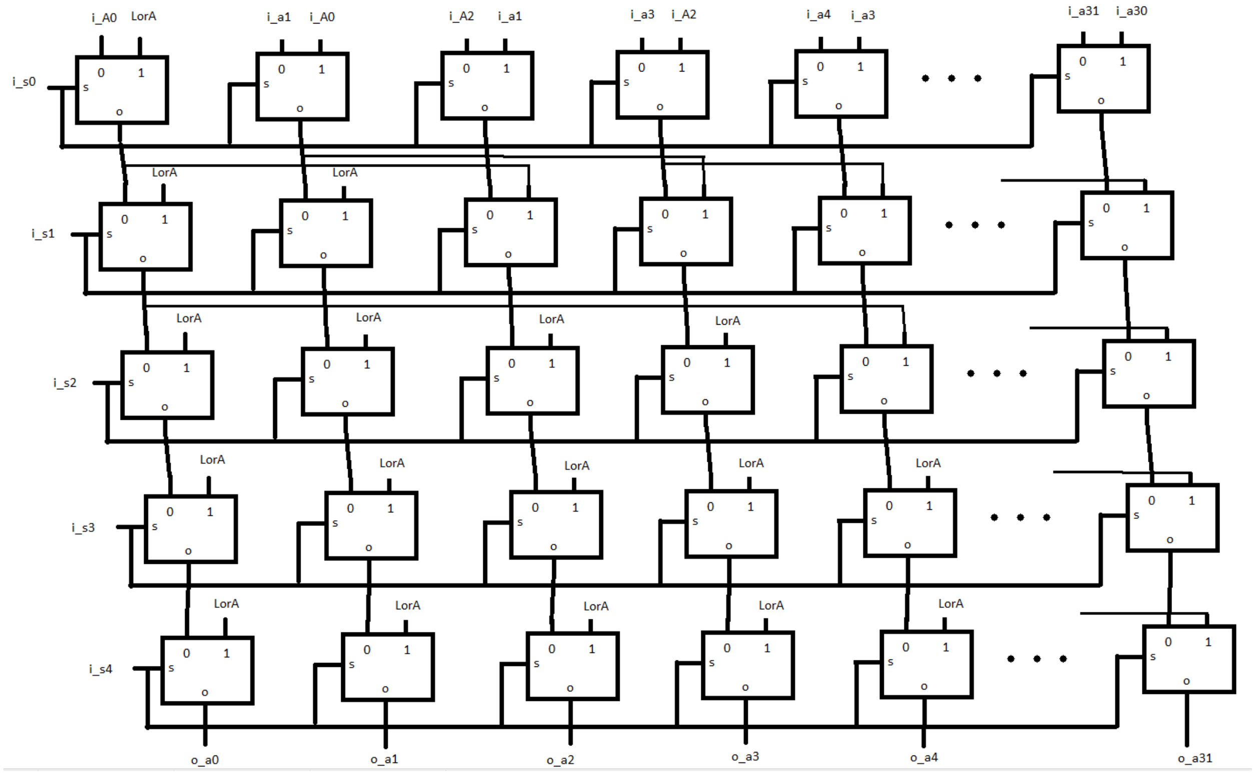
3) The MIPS ISA also contains several shifting instructions which can be implemented independently from the main ALU. For this part we will consider the barrel shifter design discussed in class.

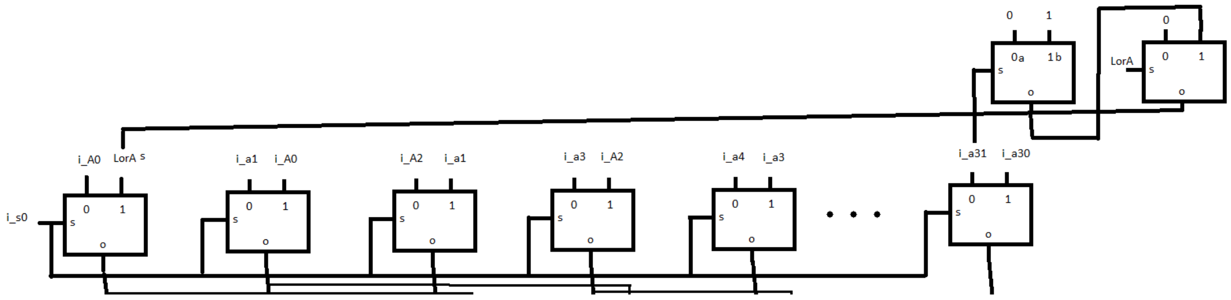
(a)Describe the difference between logical (srl) and arithmetic (sra) shifts. Why does MIPS not have a sla instruction?

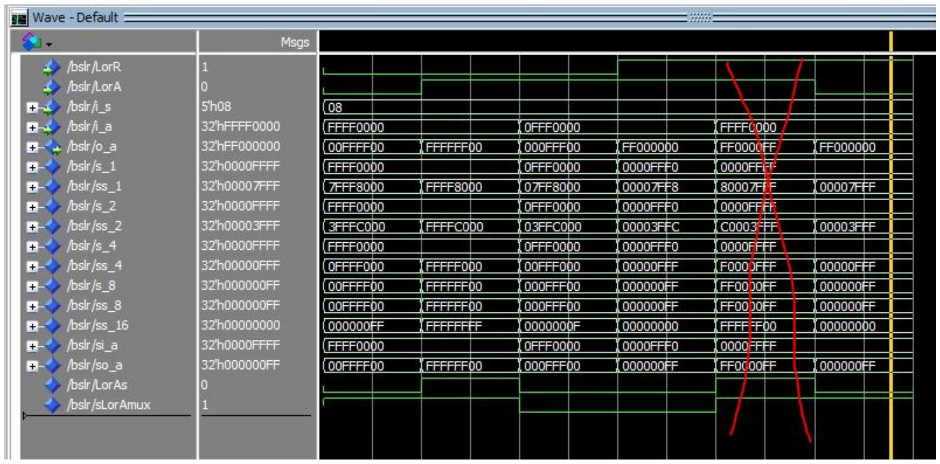
Srl is a logical operation used for unsigned numbers meaning we can pad the new shifted number. Sra is an arithmetic operation used for signed numbers because when shifting signed numbers right we need to pad with 1's instead of 0's. there is no need for a shift left arithmetic because the padding will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



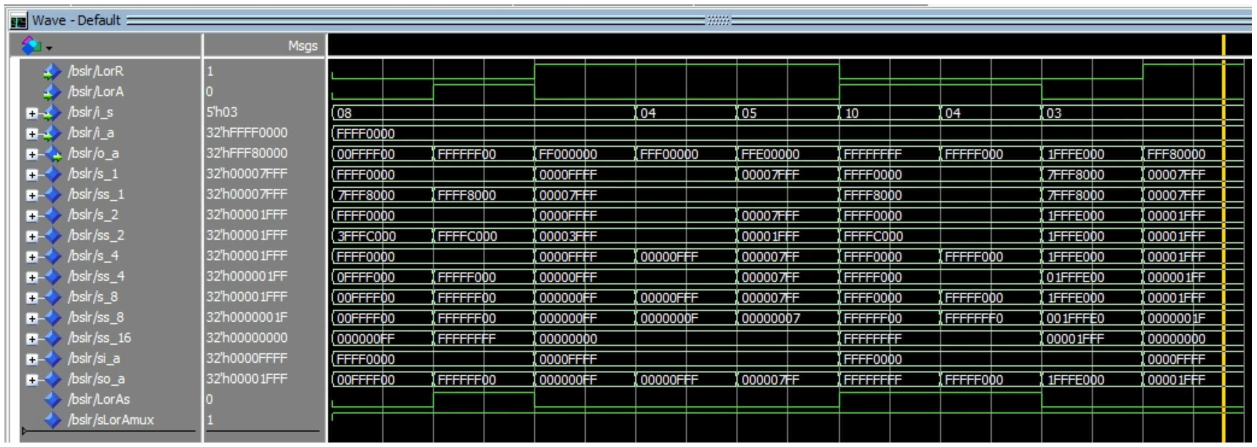


```
Wave - Default
                    Msgs
/bslr/LorR          1
/bslr/LorA          0
/bslr/i_s           5'h08          08
/bslr/i_a           32'hFFFF0000   FFFF0000                      0FFF0000                       FFFF0000
/bslr/o_a           32'hFF000000   00FFFF00  FFFFFF00  000FFF00  FF000000  FF0000FF  FF000000
/bslr/s_1           32'h0000FFFF   FFFF0000            0FFF0000  0000FFF0  0000FFFF
/bslr/ss_1          32'h00007FFF   7FFF8000  FFFF8000  07FF8000  00007FF8  80007FFF  00007FFF
/bslr/s_2           32'h0000FFFF   FFFF0000            0FFF0000  0000FFF0  0000FFFF
/bslr/ss_2          32'h00003FFF   3FFFC000  FFFFC000  03FFC000  00003FFC  C0003FFF  00003FFF
/bslr/s_4           32'h0000FFFF   FFFF0000            0FFF0000  0000FFF0  0000FFFF
/bslr/ss_4          32'h00000FFF   0FFFF000  FFFFF000  00FFF000  00000FFF  F0000FFF  00000FFF
/bslr/s_8           32'h000000FF   00FFFF00  FFFFFF00  000FFF00  000000FF  FF0000FF  000000FF
/bslr/ss_8          32'h000000FF   00FFFF00  FFFFFF00  000FFF00  000000FF  FF0000FF  000000FF
/bslr/ss_16         32'h00000000   000000FF  FFFFFFFF  0000000F  00000000  FFFFFF00  00000000
/bslr/si_a          32'h0000FFFF   FFFF0000            0FFF0000  0000FFF0  0000FFFF
/bslr/so_a          32'h000000FF   00FFFF00  FFFFFF00  000FFF00  000000FF  FF0000FF  000000FF
/bslr/LorAs         0
/bslr/sLorAmux      1
```

(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.



```
Wave - Default
                    Msgs
/bslr/LorR          1
/bslr/LorA          0
/bslr/i_s           5'h03          08                            04        05        10        04        03
/bslr/i_a           32'hFFFF0000   FFFF0000
/bslr/o_a           32'hFFF80000   00FFFF00  FFFFFF00  FF000000  FFF00000  FFE00000  FFFFFFFF  FFFFF000  1FFFE000  FFF80000
/bslr/s_1           32'h00007FFF   FFFF0000            0000FFFF            00007FFF  FFFF0000            7FFF8000  00007FFF
/bslr/ss_1          32'h00007FFF   7FFF8000  FFFF8000  00007FFF                      FFFF8000            7FFF8000  00007FFF
/bslr/s_2           32'h00001FFF   FFFF0000            0000FFFF            00007FFF  FFFF0000            1FFFE000  00001FFF
/bslr/ss_2          32'h00001FFF   3FFFC000  FFFFC000  00003FFF            00001FFF  FFFFC000            1FFFE000  00001FFF
/bslr/s_4           32'h00001FFF   FFFF0000            0000FFFF  00000FFF  000007FF  FFFF0000  FFFFF000  1FFFE000  00001FFF
/bslr/ss_4          32'h000001FF   0FFFF000  FFFFF000  00000FFF            000007FF  FFFFF000            01FFFE00  000001FF
/bslr/s_8           32'h00001FFF   00FFFF00  FFFFFF00  000000FF  00000FFF  000007FF  FFFF0000  FFFFF000  1FFFE000  00001FFF
/bslr/ss_8          32'h0000001F   00FFFF00  FFFFFF00  000000FF  0000000F  00000007  FFFFFF00  FFFFFFF0  001FFFE0  0000001F
/bslr/ss_16         32'h00000000   000000FF  FFFFFFFF  00000000                      FFFFFFFF            00001FFF  00000000
/bslr/si_a          32'h0000FFFF   FFFF0000            0000FFFF                      FFFF0000                      0000FFFF
/bslr/so_a          32'h00001FFF   00FFFF00  FFFFFF00  000000FF  00000FFF  000007FF  FFFFFFFF  FFFFF000  1FFFE000  00001FFF
/bslr/LorAs         0
/bslr/sLorAmux      1
```

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