11 Data Path Main Components Design

11.1 Objectives

After completing this lab, you will:

- Design a 32x 32 bit register file
- Design a 32 bit arithmetic and logic unit (ALU)

11.2 Register File

The register file consists of 32 x 32-bit registers and has the following interface as shown in Figure 11.1:

- ✧ BusA and BusB: 32-bit output busses for reading 2 registers
- ✧ BusW: 32-bit input bus for writing a register when RegWrite is 1
- ✧ RA selects register to be read on BusA
- ✧ RB selects register to be read on BusB
- ✧ RW selects the register to be written

Thus, two registers are read and one register is written in a single cycle. Writing happens on the rising edge of the clock. During read operation, the register file bahaves as a combinational block and once the RA or RB have valid data, the content of the read register will appear on BusA or BusB after a certain access time.

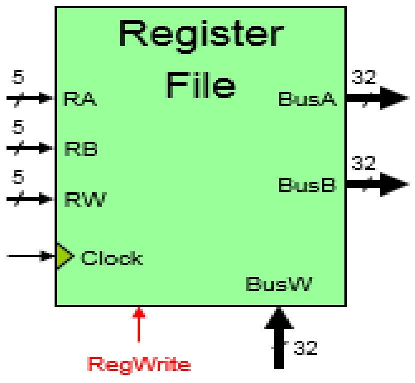

Figure 11.1: 32x32-bit register file interface.

The 32 x 32-bit register file design is given in Figure 11.2. It should be observed that register 0 is a constant. Each of BusA and BusB is connected to 32 tri-state buffers. Each tr-state buffer is connected to one of the 32 registers. The tri-state buffers enable signals are driven by the outputs of two 5x32 decoders, one with Ra input and the other with Rb input, to select which register puts its value on the corresponding bus. The enable signals of the 31 registers (register 1 to 31) are anded with the output of a 5x32 decoder and RegWrite signal. The 5x32 decoder input is connected to Rw to select which register should be written.

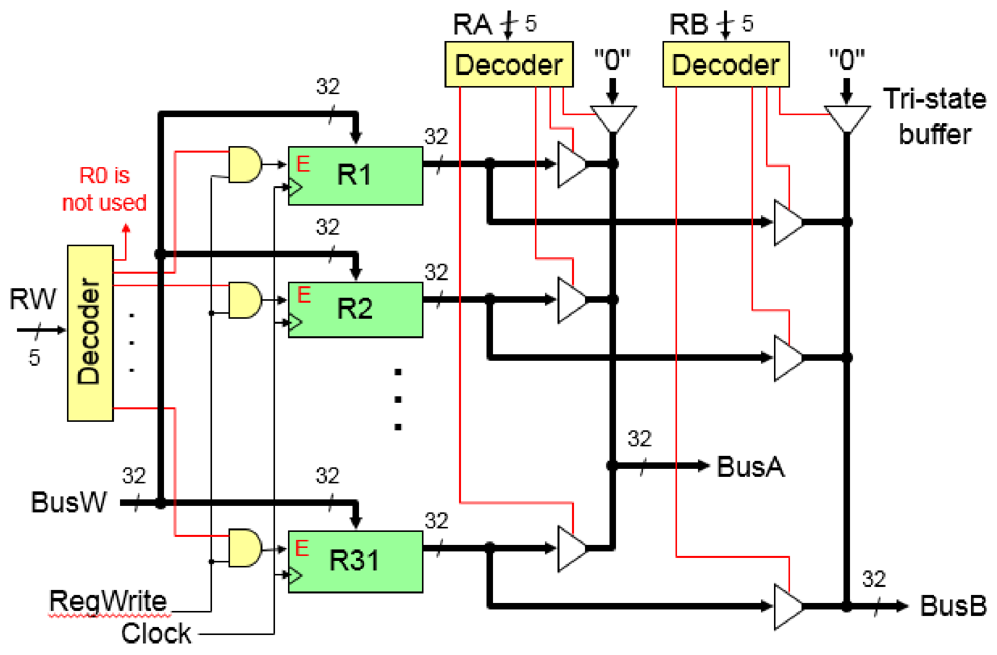

Figure 11.2: 32x32-bit register file design.

11.3 Arithmetic and Logic Unit (ALU) Design

The Arithmetic and Logic Unit (ALU) is the unit where most instructions are executed. It mainly performs arithmetic, logic and shift operations. The ALU will have four main blocks: Arithmetic, Comparator, Logic, and Shifter blocks as illustrated in Figure 11.3.

Arithmetic Block

The arithmetic block is composed of a 32-bit adder that can perform 32-bit addition and subtraction. Its internal implementation can be designed using a ripple carry adder composed of 32 full-adder blocks. The inputs A and B are two 32- bit integers and the output F is A+B or A-B. The arithmetic block has a control signal, ADD/SUB, to determine whether the operation to be performed is addition or subtraction. If this signal is 0, the adder will perform addition, otherwise it will perform subtraction. Note that subtraction is performed using 2's complement representation as A-B= A + B' + 1. B' is computed by the XOR gates in the arithmetic block. The adder also generates Carry-Out (Cout) and Overflow signal that can be used to test for correctness of the obtained result and for comparison purposes for unsigned and signed operations. Note that Overflow can be generated by XORing the last two carry-out signals (i.e. the carry-outs of bits 30 and 31).

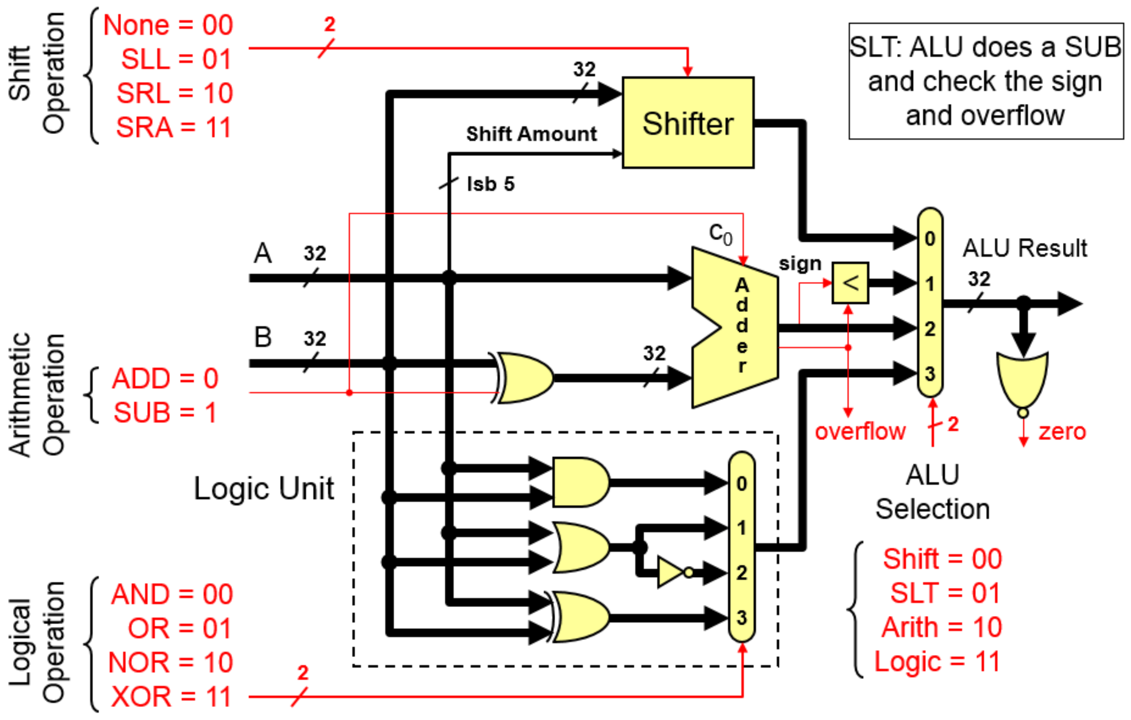

Figure 11.3: Arithmetic and Logic Unit (ALU) design.

Comparator Block

This block is mainly used for comparing signed and unsigned numbers. For the MIPS CPU, we just need to compare if a number is less than another number for implementing the set on less than instructions (SLT, SLTU, SLTI, SLTIU). For unsigned comparison of two numbers A and B, we need to perform a subtraction operation, A- B, and then check whether we have a Carry-Out (Cout) or not. If Carry-Out=0, this means that there was a borrow when B is subtracted from A and thus A < B. However, if Carry-Out=1 then this implies that there was no borrow and hence A ≥ B.

Similarly, for comparing signed numbers A and B, we perform the subtraction operation A – B and then we check both the Sign of the result and the Overflow signal. The Sign of the result is the most significant bit of the result (i.e. but 31). If the Sign value is not equal to the Overflow value, then A < B, otherwise, A ≥ B. This can be done by XORing the Sign and the Overflow signals. If the result is 1, this means that A < B.

Logical Block

The logical block performs the logic operations, AND, OR, NOR, and XOR, for implementing the MIPS logic instructions. Functions in this block are performed using the basic logic gates. Logic gates can have up to 32 inputs in Logisim, and each input may have up to 32 bits. In our case, the gates are implemented as gates having 2 inputs, each having 32 bits. Figure 11.4 shows a model of a 2-input 8-bit AND gate in Logisim.

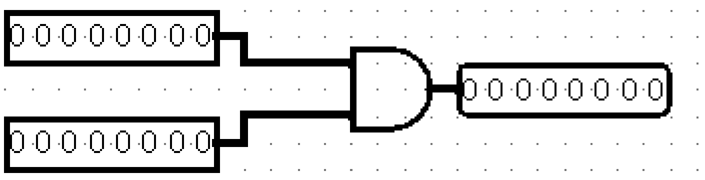

Figure 11.4: A 2-input 8-bit AND Block.

Shifter Block

The shifter block is used to implement MIPS shift instructions (SLL, SRL, SRA). A shift operation takes a binary number and shifts it left or right by a specified number of bits. There are two main kinds of shift operations: logical,and arithmetic.

- Logical shift: whenever bits are shifted to the left or right, 0's are injected.
- Arithmetic shift: when bits are shifted to the left, 0's are injected, however when bits are shifted to the right the sign bit (i.e., most significant bit) is injected.

The functionality of logical and arithmetic shit instructions is illustrated in Figure 11.5.

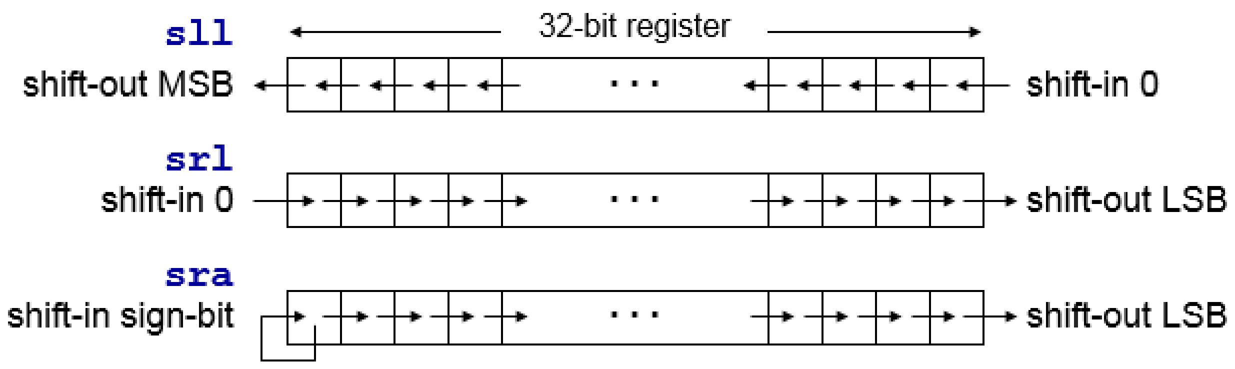

Figure 11.5: Functionality of logical and arithmetic shift instructions.

Logisim provides blocks for performing shift operations that can be used in the design of the Shifter block.

Multiplexor Block

A 32-bit 4x1 Multiplexor is used to select the output from either the Arithmetic block, the Comparator block, the Logical block or the Shifter block. This is done through a 2-bit ALU selection signal.

Zero Flag Detector Block

The role of this block is to set the zero-flag bit to 1 whenever the output of any operation is equal to zero. This could easily be designed using a NOR gate at the output.

11.4 In-Lab Tasks

You are required to design a 32-bit MIPS-like processor with 31 general-purpose registers. The first building blocks of the CPU are the ALU and the register file.

1. **Task 1**:

   - Model the 32x32-bit register file given in Figure 11.2 as one single module in Logisim
   - Test the register file for correct operation by writing to and reading from different register combinations.

2. **Task 2**: **Arithmetic and Logical Unit (ALU) Design**

   - Design a 32-bit ALU to perform all the arithmetic, logic and shift operations required by your data path
   - Model the your designed 32-bit ALU in Logisim
   - Test the correct functionality of all operations implemented by the ALU.

11.5 Bonus Question

One possible implementation of the shifter known as the Barrel Shifter is given in Figure 11.6. This architecture has the advantage of performing a number of operations using the same hardware. You are required to design such shifter and adapt it to your design. You need then to use it instead of the shifter made up of available shifters in Logisim.

The shifter is implemented with multiplexers and wiring (splitters in our design), the shift operations can be: SLL, SRL, SRA, or ROR. The input data is extended to 63 bits according to Shift Op, and the 63 bits are shifted right according to $S_4S_3S_2S_1S_0$

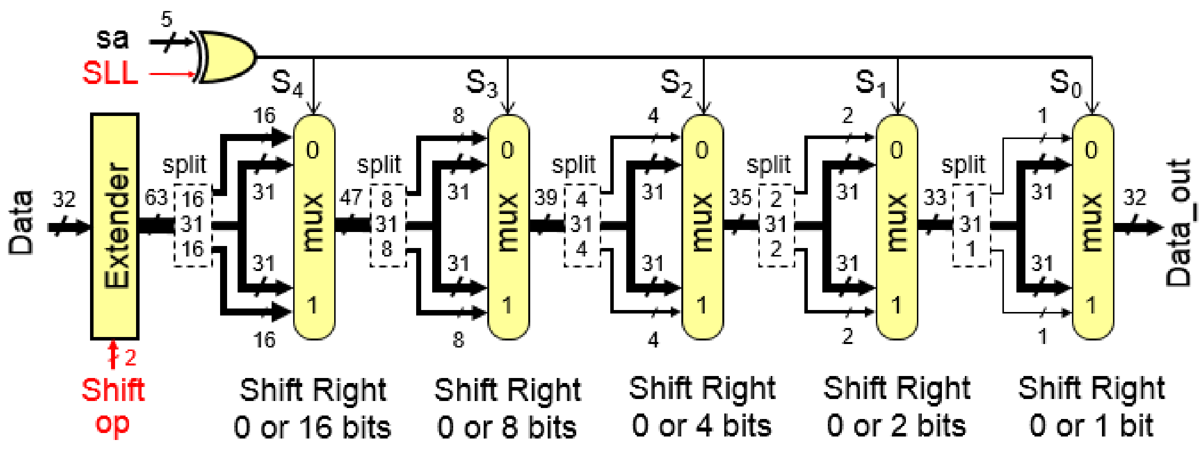

Figure 11.6: Barrel shifter.

The Input data is extended from 32 to 63 bits as follows:

- If shift op = SRL then ext_data[62:0] = $\{0^{31}$, Data[31:0]$\}$
- If shift op = SRA then ext_data[62:0] = $\{\text{Data}[31]^{31}$, Data[31:0]$\}$
- If shift op = ROR then ext_data[62:0] = {Data[30:0] , Data[31:0]}
- If shift op = SLL then ext_data[62:0] = $\{$Data[31:0] , $0^{31}\}$

- For SRL, the 32-bit input data is zero-extended to 63 bits
- For SRA, the 32-bit input data is sign-extended to 63 bits
- For ROR, 31-bit extension = lower 31 bits of data
- Then, shift right according to the shift amount

- As the extended data is shifted right, the upper bits will be: 0 (SRL), sign-bit (SRA), or lower bits of data (ROR)