Com Sci 152B Digital Design
Lab 1: ALU & Register File Implementation

Michael Hale	004-620-459
Matthew Nuesca	904-440-067
Shilin Patel	904-569-866
Bingxin Zhu	704-845-969

1 Overview

In this lab we implemented two key components found in modern processors: the arithmetic logic unit (ALU) and the register file. The purpose of the ALU is to perform a variety of operations on arbitrary 16-bit integers and write the result to an output port. On the other hand, the register file performs no computation and simply reads and writes to a series of internal 16-bit registers.

2 Design Requirements

2.1 ALU Requirements

The ALU shall perform a set of logical and arithmetic operations given the sixteen bit inputs `a` and `b` and the four bit op-code `op`. The ALU must write the output of the encoded operation to a sixteen bit output `out` and set two flags (`ovf`, `zero`) if the operation results in overflow or a zero output respectively.

The operations to be encoded are:

OP	**Description**	**Symbolically**
0000	Subtraction	$a - b$
0001	Addition	$a + b$
0010	Bitwise OR	$a \mid b$
0011	Bitwise AND	$a \ \& \ b$
0100	Decrement	$a - 1$
0101	Increment	$a + 1$
0110	Inverse	$\sim a$
1000	Logical left shift	$a << b$
1001	Set on less than or equal	$a \leq b$
1010	Logical right shift	$a >> b$
1100	Arithmetic left shift	$a <<< b$
1110	Arithmetic right shift	$a >>> b$

2.2 Register File Requirements

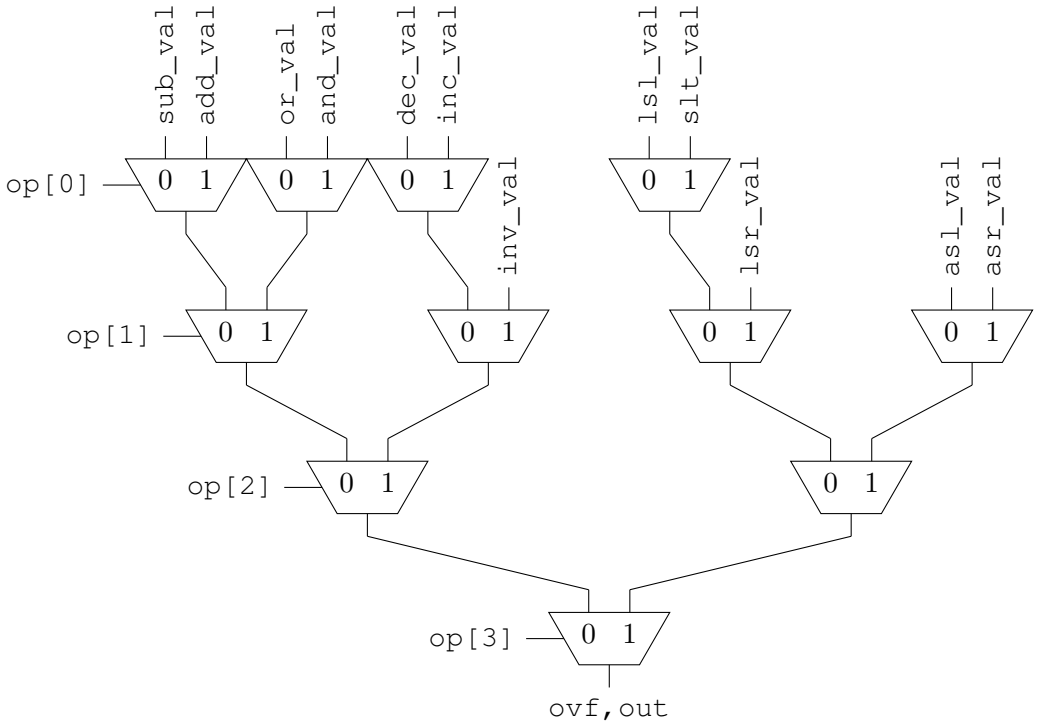
The register file shall contain 32 sixteen bit registers and must include two sixteen bit read ports and a write port (`bus_a, bus_b, write_bus`) with their corresponding five bit register select ports (`ra, rb, rw`). It must also accept a write-enable signal (`wren`), a clock signal (`clk`) and a reset signal (`rst`) which sets the contents of each register to zero. Additionally, if a register is being read and written at the same time, the read should reflect the updated value immediately.

3 Implementation

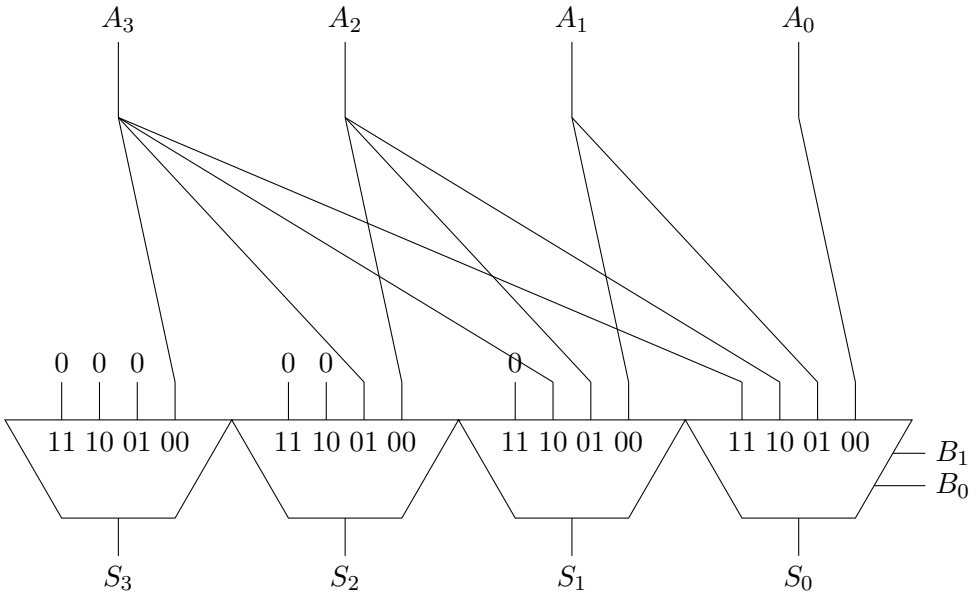
3.1 ALU Implementation

Our ALU implementation essentially partitions each instruction into a single Verilog module. In the top level ALU module, the inputs `a` and `b` fan out into each of these instruction modules which each independently generate an `ovf` and `result` signal concatenated into a single `out` signal. These signals are then fed into a multiplexer which uses the op-code `op` to determine the ALU output. The `zero` flag is generated by OR-ing each bit of the output and inverting the result.

Rather than designing a single multiplexer that would take 12 seventeen bit inputs and generate one seventeen bit output, we instead created a simple two-input multiplexer module. In our top level ALU module, we then instantiated arrays of these muxes of length 17 to handle the wide instruction outputs. To generalize the multiplexer to 12 inputs, we created a cascade of 2-input muxes where each stage was determined by a different bit of the op-code `op`.


sub_val
add_val
or_val
and_val
dec_val
inc_val
lsl_val
slt_val
inv_val
lsr_val
asl_val
asr_val
op[0]
op[1]
op[2]
op[3]
0 1
ovf,out

Shifter



Arithmetic Right Shifter