COMPUTER ARCHITECTURE

LAB 2

FLUFFY THE GATEKEEPER

JULIAN CEIPEK
YUXIN GUAN
PHILIP Z LOH
SASHA SPROCH

EXECUTIVE SUMMARY

An Arithmetic Logic Unit (ALU) contains the logic to perform multiple instructions in parallel, and then uses a mux at the end to select the output of the desired function. We present an ALU design worthy of NI's semi-custom MIPS-like CPU core. It performs the instructions ADD, SUB, XOR, SLT, SLLV, SRAV, and SRLV in a reasonable amount of time (1560 ns max) in a small space (7197 gates, where a 7 input OR is counted as 6 gates). We provide test benches for all modules individually, but didn't have a cohesive testbench checking each of the instructions integrated in the ALU. Our ALU design proudly passed all our individual instruction tests. In general, we chose space efficiency over time efficiency so that we could fit into NI's small CPU.

SPEED VS SPATIAL EFFICIENCY

Our shifter and adder both demonstrate our space-time optimization. We use one shifter for both left and right shift, using only the addition of one mux on each end of the shifter. This increases our shifter time by 100 nanoseconds, but decreases our space by 96 2by1-muxes. (It would be 128 if a left arithmetic shift existed.) Our adder is a simple chain of adders, which, though slow, is as space efficient as we can imagine. Other adders we considered, such as the carry-lookahead adder, were faster, but larger.

Our MegaMuxOfDestiny demonstrates our worst space-time tradeoff. We used a simple behavioral verilog module representing a traditional 8-input 32-bit mux to save time (ours, not the ALU's) at the cost of wasting both time and space in the ALU. We could have designed this as a three input mux with bonus logic inside to differentiate ADD, SUB and SLT, which would be one input, and SLLV, SRAV and SRLV, which would be another (XOR gets its own wire). We did not consider it worth our time to produce the structural code to implement this efficiency.

Conveniently, NI decided that MUL was an unnecessary instruction and was consequently removed in favor of die-space.

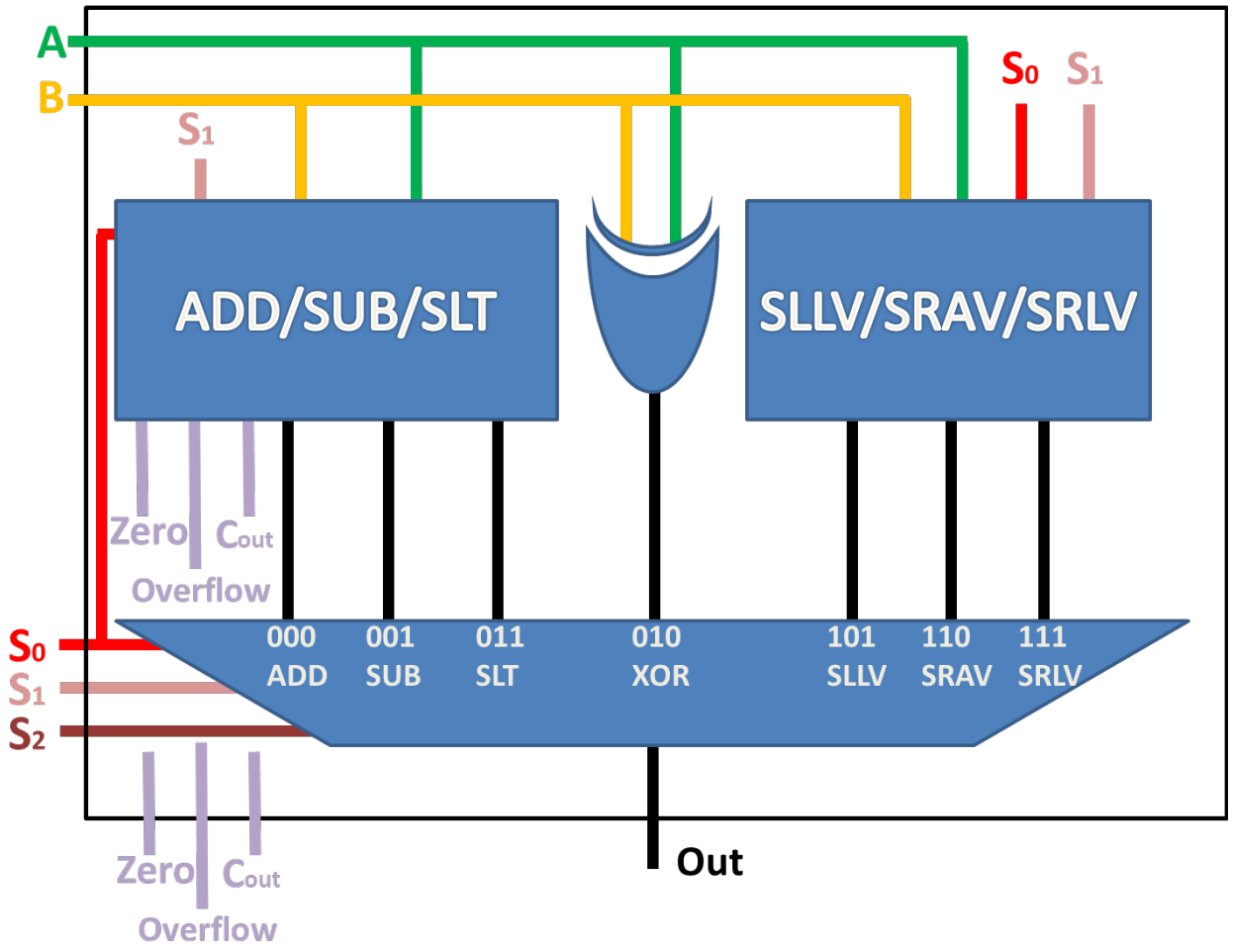
CORRECTNESS

At the module level, we have tests for AdderSubtractor, XOR_32, Shifter, and our mux32layers2by1 (32 layers of 2by1 muxes using the same select bit). All of the tests ran

successfully. We will conduct a fully integrated ALU test with all the instructions in future if we have more time. The Python script is generally complete, maybe with a few kinks that we'll find out if we have more time.

Design

Our ALU takes three inputs. A and B are a bus of 32 bits. They are either added, subtracted, compared, xor'd, or A is shifted by B. S is a control bus of 3 bits. It determines what function the ALU will perform. Out is the "main" output-- the solution to the addition, subtraction, shift, etc. We also provide three other outputs, Zero, which determines if out is zero, Overflow, which detects when out is too large to be stored in a 32 bit representation, and Cout, the final carry-out bit. The Add, XOR, and Shift blocks run in parallel before getting passed to the final mux, so our maximum time delay is the slowest of the three. Our slowest subsystem is the ADD/SUB/SLT block. In the worst case, it takes 1410 ns to produce a stable output, so when combined with the 150 ns final mux, our maximum run time becomes 1560 ns. The diagram below shows a high-level view of our ALU design.



Timing Analysis

Timing Diagram of worst-case propagation delays for 32-bit Adder

Gates	Propagation Delays
NOT	10
MUX 2 by 1 (active)	50
First bit: XOR AND_1 (active) (parallel with OR_1) OR_1 (active) (parallel with AND_1) AND_2 (active) OR_2 (active)	 30 20 **20** **20** **20**
2nd to 32nd Bit Carry chain:	(20+20)* 31
NOT	10
AND_slt_0	20
AND_slt_1 (parallel with AND_slt_0)	20
OR	20
MUX 2 by 1 (active)	50

Worst case Adder: 10+50+(20+20+20)+(20+20)*31+10+20+20+50 =1460

ALU Instructions	Worst-Case Propagation Delays
ADD*	1460
SUB*	1460
XOR	20
SLT*	1460
SLLV**	350 (from the 7 2-by-1 muxes in series)
SRAV**	350 (from the 7 2-by-1 muxes in series)
SRLV**	350 (from the 7 2-by-1 muxes in series)
* the same subblock is used	--
** the same subblock is used	--

Note: A 6 to 1 MUX is connected to the outputs of all the instructions, which has a timing delay at 150.

Test Cases

XOR
To test XOR, we ran through a few key cases. We alternated 0s and 1s for A and B in the four possible orientations: 1s and 0s line up, and they don't, beginning with 1 and beginning with 0. The outputs were all correct, and we feel these cases are comprehensive enough for a design test, so we believe we can assume XOR works as expected.

Shifter
To test the shifter, we ran through the various permutations of making the 0th, 1th, 7th, and 31th bit the odd one out (a 0 in a sea of 1s or a 1 in a sea of 0s). We repeated for all three types of shifts. Again, because all the tests were successful and the amount and variety of tests was fairly comprehensive, we consider the shifter to be working.

AdderSubtractor
To test the Adder/Subtractor, we tested various combinations of all 1s, all 0s, alternating bits, and creating an odd bit out. The test cases all work, but it had a few bugs until right up until the end that were eluding us. This held us up long enough to prevent successful integration testing.

MegaMux
MegaMux is what we lovingly call the 7 input mux at the bottom of our ALU overview diagram. The MegaMux takes in the outputs of our 7 instructions and, based on the 3-bit select bit, outputs the final 32-bit output of our ALU. To test our MegaMux, we performed a walking one across the seven inputs.

Conclusion

Our ALU design targets 7 instructions: ADD, SUB, XOR, SLT, SLLV, SRAV, and SRLV. All instructions take two 32-bit inputs. A 3-bit select bit chooses what instruction to output. A 7 to 1 MUX is connected to all the outputs of our individual ALU instructions. Depending on the 3-bit select bit, the MUX outputs a 32-bit output accordingly. Even though we had problems with the final integration testing, our overall design is sound, and our creative solution to creating multiple shifters despite only needing one demonstrates our grasp of and enthusiasm for the overall purpose of this lab.