Comparch Lab 3

Henry Rachootin
Tobias Shapinsky

November 2, 2017

1 Architecture

diagram.png

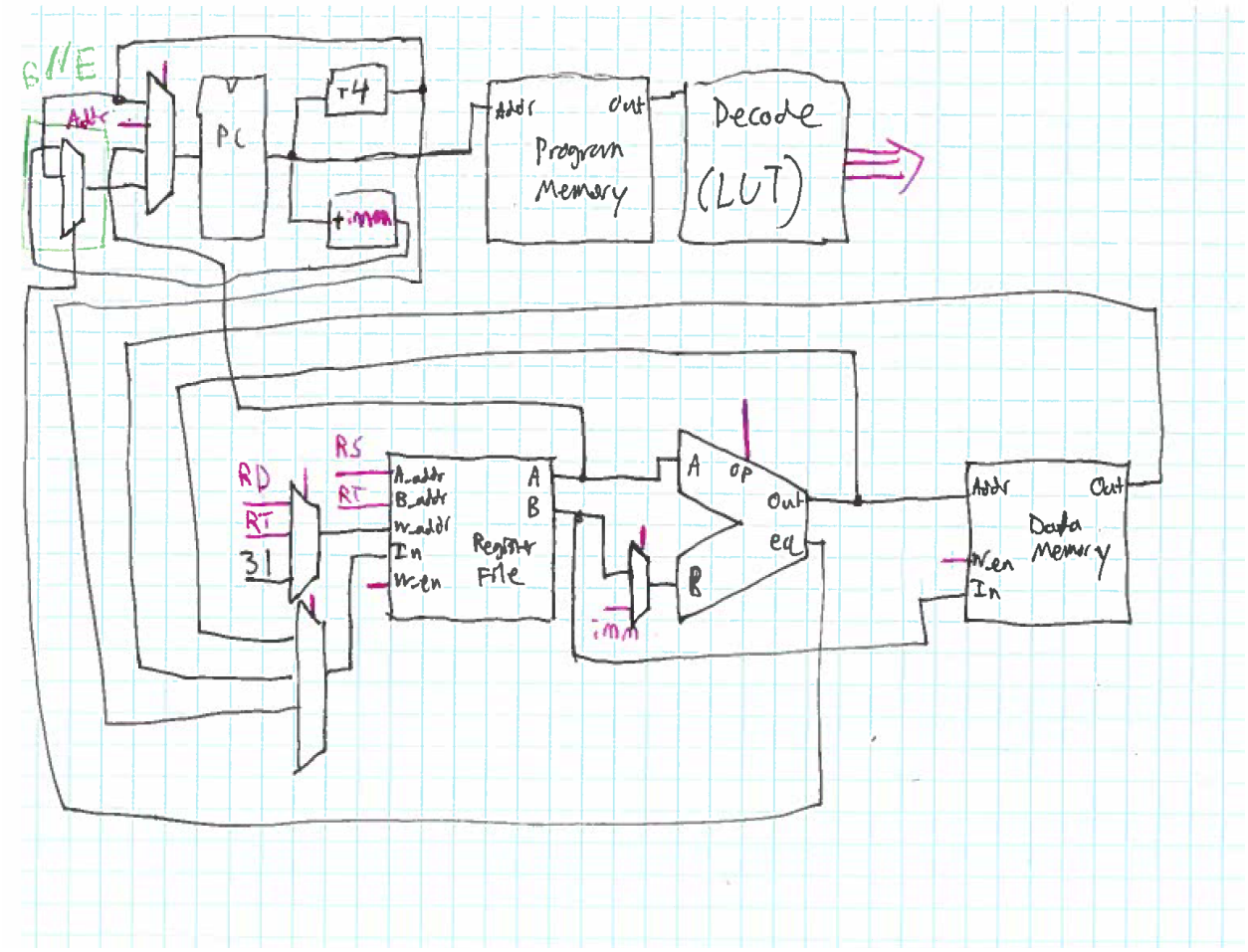

Figure 1: Block Diagram of our CPU

We designed a simple single cycle CPU. Each cycle a new value is loaded into the program counter and used to address the program memory. The output of the program memory goes straight into a giant lookup table, which controls each of the pink lines in Figure 1. Because it is a single cycle CPU, we were able to use the ALU to compute the address offsets for LW and SW instructions, saving slightly on board space.

1.1 Nonstandard RTL

Most of our implementation is right out of the specification, which the exception of our syscall implementation (described below) and our BNE implementation. For BNE, instead of PC becoming PC+4 or PC+(imm<<2), PC becomes PC+4 or PC+(imm<<2)+4. This was necessary because the compiled code from MARS seemed to think so.

2 Test Benches

2.1 Register File

We begin by checking to make sure that the zero register functions correctly by writing a non-zero value and reading to make sure that the register still reads back zero. The next test checks that reading and writing functions correctly by writing to a register and then reading back value to confirm that is had not changed. The final test is to confirm that the write enable input works correctly, so we simply pull it low and attempy to write to the register, if this doesn't work then the device passes the test.

2.2 Memory

The Memory test begins by loading a prepared memory file. After that the first three words are read and checked against the file to confirm that memory is being read correctly. After that there is a write-read test where the value 0xL337FADE is written to memory and then read back.

2.3 ALU

Testing the ALU was dead simple. The ALU supports ADD, SUB, XOR, and SLT operations. We tested ADD, SUB, and XOR by just giving it two operands to combine, and testing if the output was correct. For SLT we tested each combination of positive, negative, and zero operands in each case of them being less than, greater than, or equal to each other.

2.4 CPU

To test the CPU we implemented the syscall instruction to set a debug output of the CPU to the value of a register (selected by a debug input) and then pull an interrupt line high. On the test bench side, the interrupt handler compares the value of the debug output to the expected result. We then wrote a program which tested the functionality of each instruction, using syscall as an output. To test jumping and branching instructions, an instruction was added in the part of the program which shouldn't execute which would cause an incorrect value to be outputted by the syscall. This strategy was able to find a bug with our BNE implementation which caused it to always branch to the instruction before its target.

3 Performance

The following is synthesized from a Vivado synthesis report.

The main CPU module (which includes instruction fetch and program counter) unsurprisingly contains 21 muxes of bit widths varying from 1 to 32, and input numbers ranging from two to nine. The CPU also contains a single 14 bit register used to address memory. The CPU also has several low input Adders for PC and immediate operations.

The memory module (besides having 2^{14} addresses) uses two 14 bit adders. Besides that its footprint is fairly low.

The ALU has a single 32 bit, three input, adder which is used for addition and subtraction. In Addition it also ha a single 2 input 32-bit XOR and two four-input 32-bit muxes.

Due to the fact that our cpu is of the single cycle variety it will be more space efficient but strictly slower than any other non-esoteric CPU architecture.

4 Further Work

On top of making the CPU we also wrote a compiler from an esoteric programming language called Brainfuck to our instruction set. The idea being that people have made compilers from C to Brainfuck, so by implementing the limited functionality of Brainfuck we could gain the ability to write C code and have it compiled for our CPU.