# Duke 250/16 Processor Core Design

## 1 Overview

The processor design has followed the Duke 250/16 ISA specification and successfully implemented all of the following instructions.

| instruction | opcode | type | usage | operation |
|---|---|---|---|---|
| nand | 0000 | R | nand $rd, $rs, $rt | $rd = $rs NAND $rt |
| xor | 0001 | R | xor $rd, $rs, $rt | $rd = $rs XOR $rt |
| addi | 0010 | I | addi $rt, $rs, Imm | $rt=$rs+Imm |
| add | 0011 | R | add $rd, $rs, $rt | $rd=$rs+$rt |
| sub | 0100 | R | sub $rd, $rs, $rt | $rd=$rs-$rt |
| shra | 0101 | R | shra $rd, $rs, <shamt> | $rd = $rs shifted <shamt> to right (**arithmetic** shift that preserves sign); shamt is unsigned. Arithmetic shift means that the bits shifted in on the left are the same as the original most-significant bit. |
| shl | 0110 | R | slh $rd, $rs, <shamt> | $rd = $rs shifted <shamt> to left; shamt is unsigned. |
| bgt | 0111 | I | bgt $rs, $rt, Imm | if ($rs>$rt) then PC=PC+1+Imm |
| beqz | 1000 | R | beqz $rs, Imm | if ($rs==0) then PC=PC+1+Imm |
| lw | 1001 | I | lw $rt, Imm($rs) | $rt = Mem[$rs+Imm] |
| sw | 1010 | I | sw $rt, Imm($rs) | Mem[$rs+Imm] = $rt |
| j | 1011 | J | j L | PC = L (upper 4 bits same) |
| jr | 1100 | R | jr $rs | PC = $rs |
| jal | 1101 | J | jal L | $r7=PC+1; PC = L |
| output | 1110 | R | output $rs | print $rs on a TTY display |
| input | 1111 | R | input $rd | $rd = keyboard input |

Figure 1: Table of Instructions Implemented

Below is an overview of the main interface of the processor core:

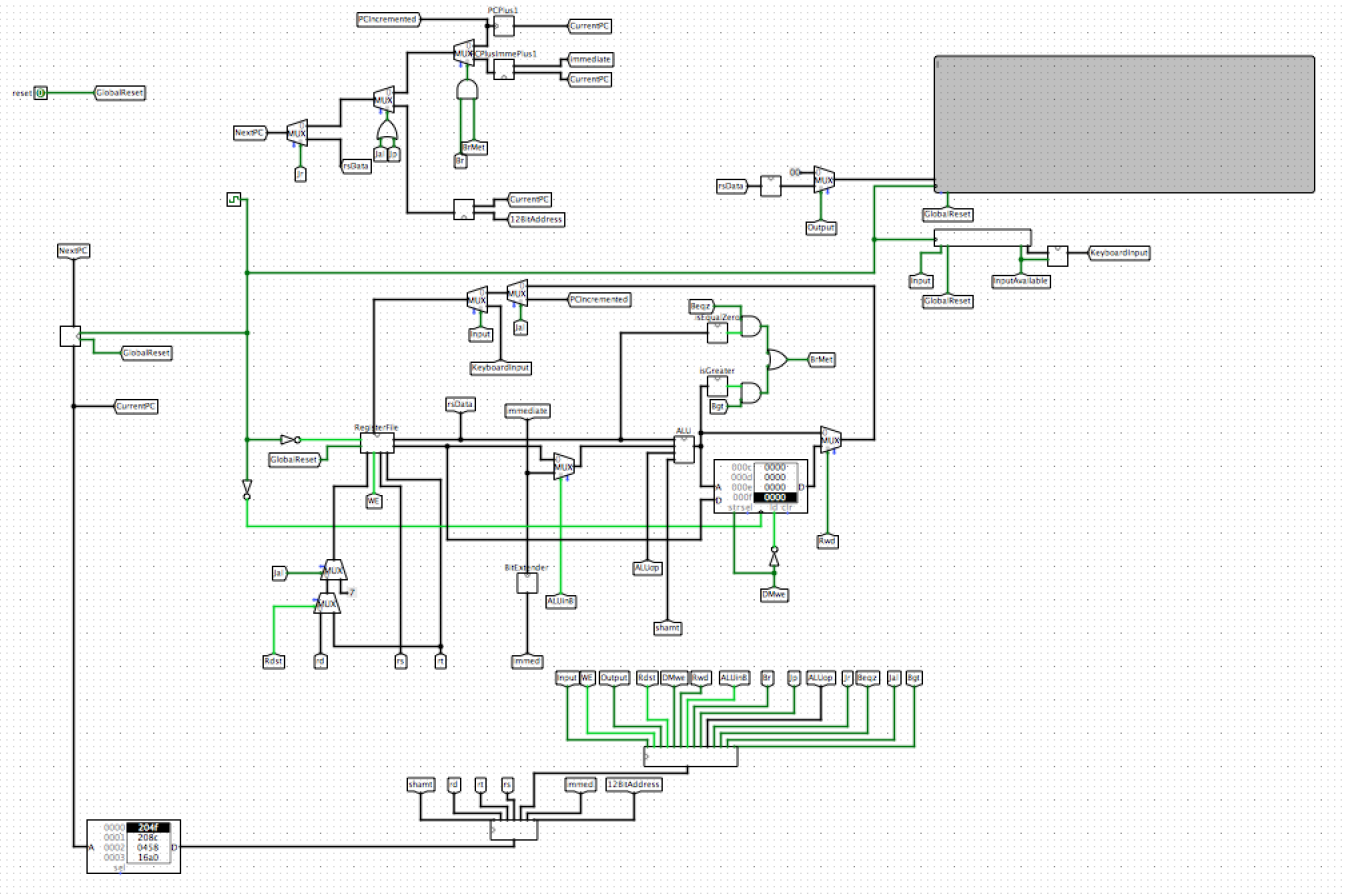


Figure 2: Integrated View of the Processor Core

The processor core has a few major components, the keyboard input, the tty display, the PC register, the instruction ROM, the data memory, the register file, the ALU and the control logic.

## 2 Register File

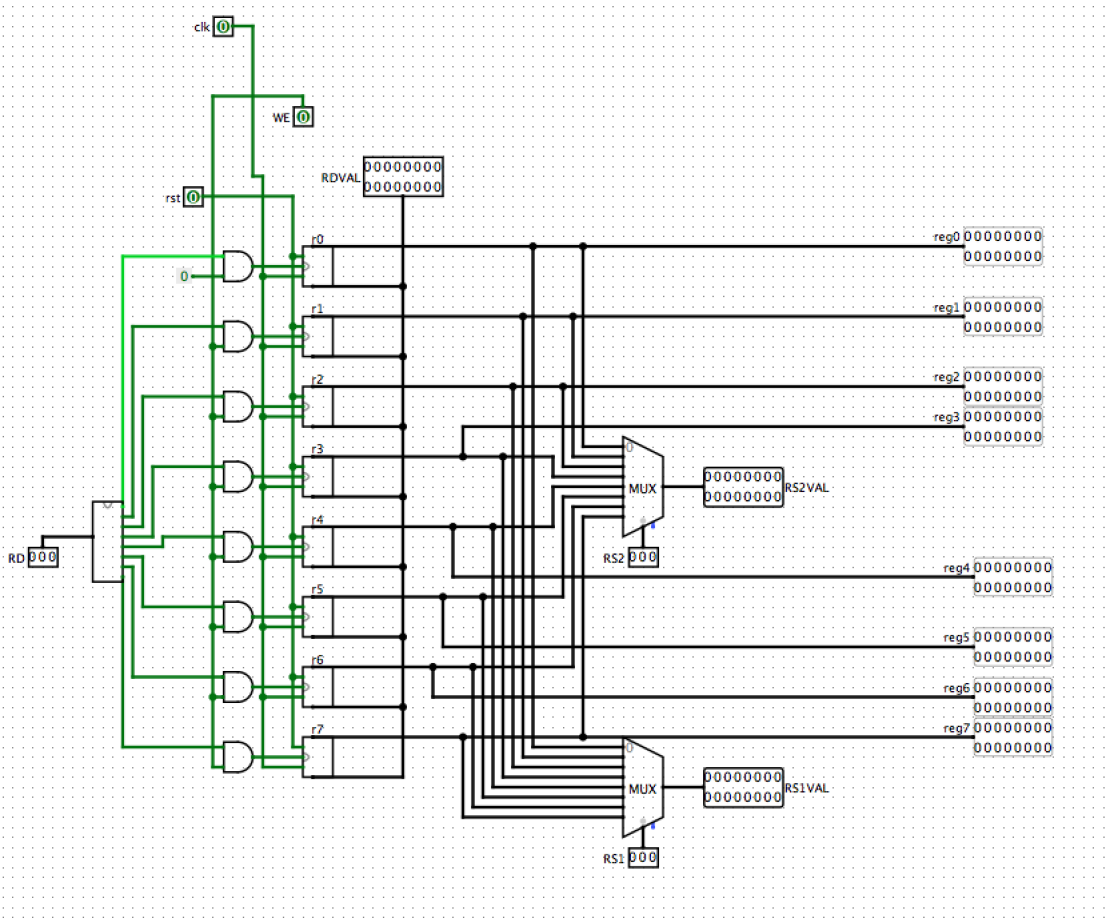


Figure 3: Register File

The register file consists of 7 general purpose registers and a zero register. The interface provides two parallel data reading ports which are labeled as RS1, RS1VAL and RS2, RS2VAL. A three-bit binary input to RS1 and RS2 are required to indicate the register to read and register data will be output through the corresponding output pin. The register file also provides a single write port, and RD input is used to designate the register to write the data input in RDVAL. On top of the IO interface, there are also two more inputs which are the clock pin and global reset pin.

# 3 ALU

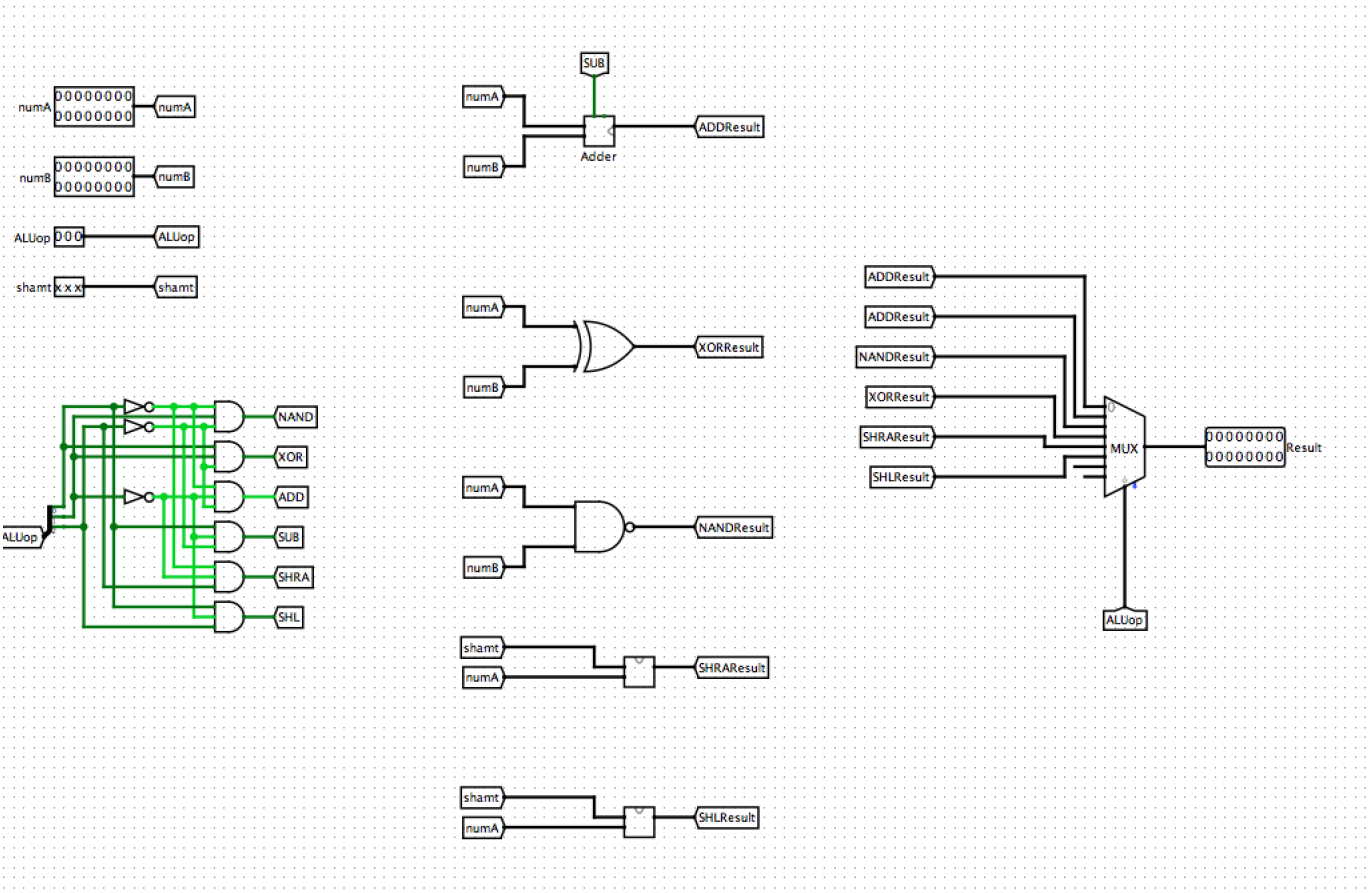


Figure 4: ALU Implementation

There are two data inputs to the ALU, denoted as numA and numB. These two numbers represent the operands. shamt is the special three-bit input to the ALU if the ALU is to perform binary shift. It denotes the number of bits to be shifted left or right. Finally it is the three-bit ALUop. ALUop is used to signify which arithmetic operation is required. The supported operations are NAND (010), XOR(011), ADD(000), SUB(001), SHRA(100), SHL(101).

# 4 PC Register

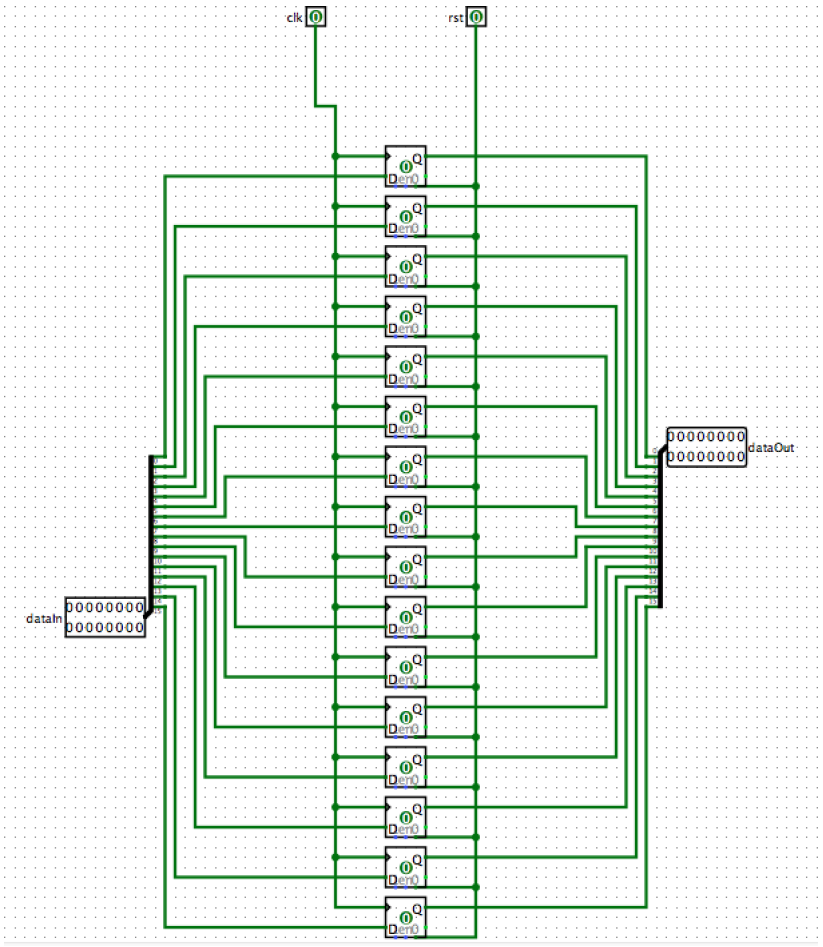


Figure 5: PC Register

Just like the general purpose registers, the PC register is implemented using 16 D flip-flops set to trigger at the rising edge. Other than the global reset and clock input, it also has dataIn input and dataOut output ports. dataOut outputs the current PC register value and dataIn gives the new PC value to be stored in the PC register.

# 5 Decoder

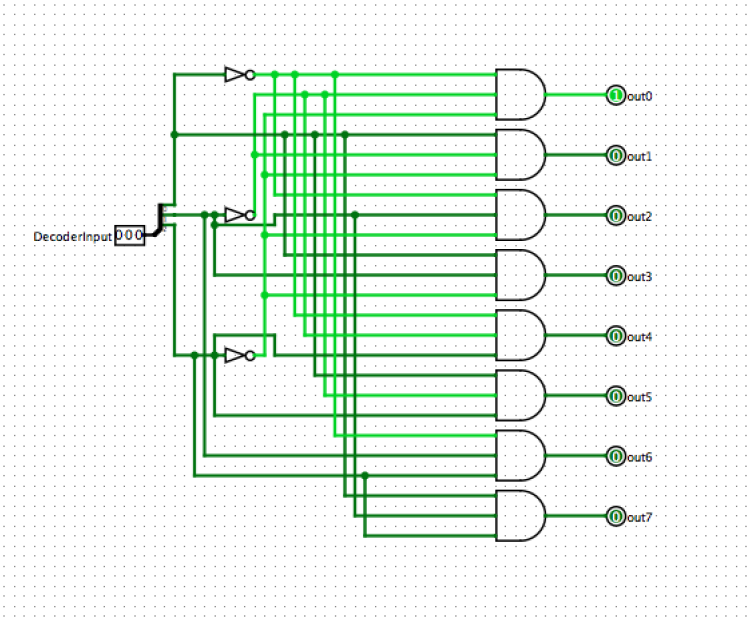


Figure 6: Decoder Implementation

It is just a decoder...nothing fancy about it.

# 6 Shifter

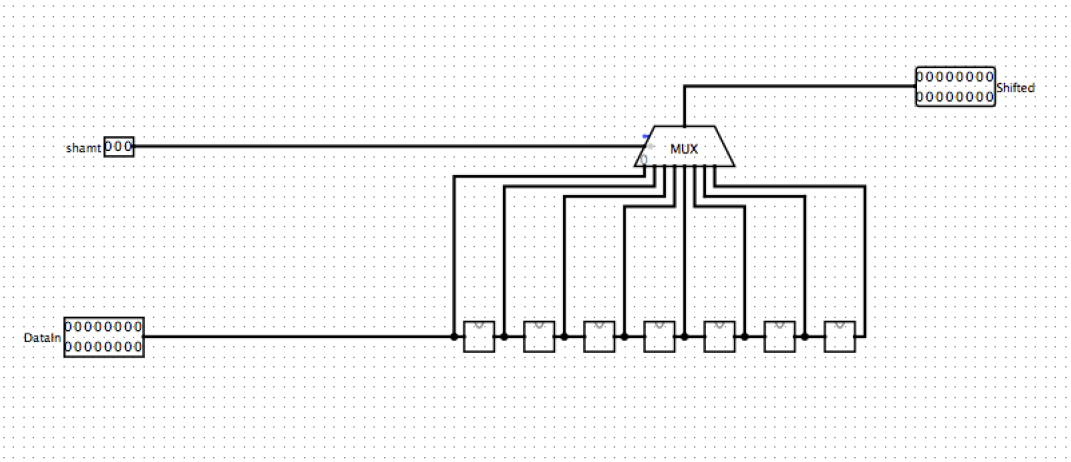


Figure 7: Shifter Implementation

Both the left and right shifters are implemented by connecting a series of 1-bit shifters, and use a multiplexer to choose the correct output. It has two inputs labeled as shamt and DataIn. shamt gives the number of bits shifted and DataIn inputs the number to be shifted.

# 7 Control

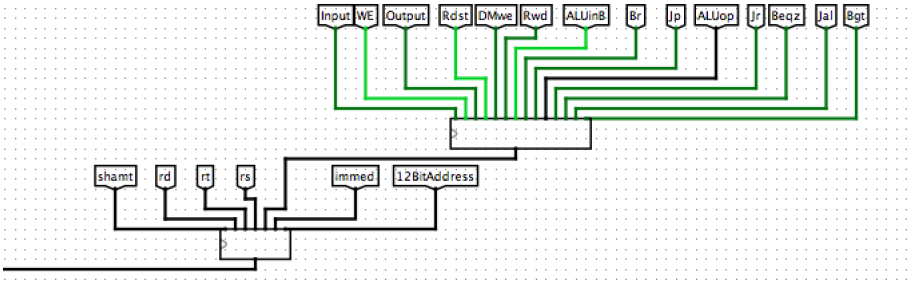


Figure 8: Control Implementation

The control logic is implemented using two layers. The first layer extracts the necessary bits such as Rs, Rt and opcode values. The second layer has a single input opcode, and translate the opcode into corresponding control signal outputs.