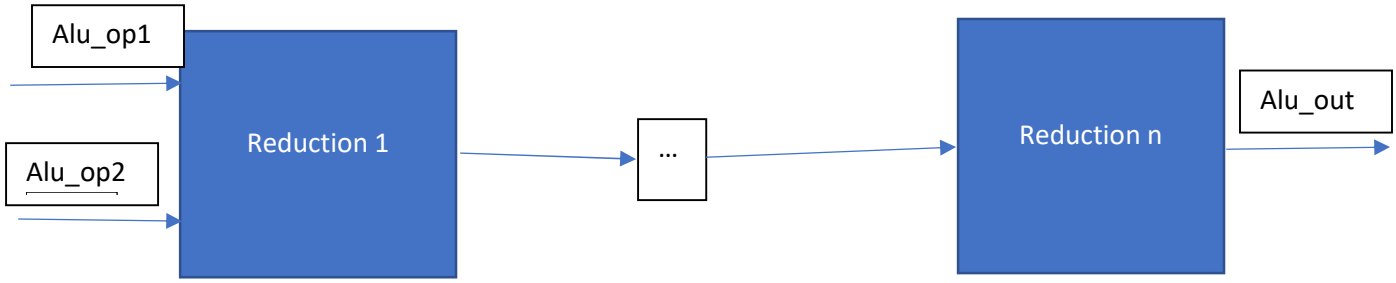
Multiplier.

A multiplier and a divider have been added to the design. They sits in the EXE stage of the pipeline. This multiplier is an advanced multiplier, as it is based on a Wallace tree method.

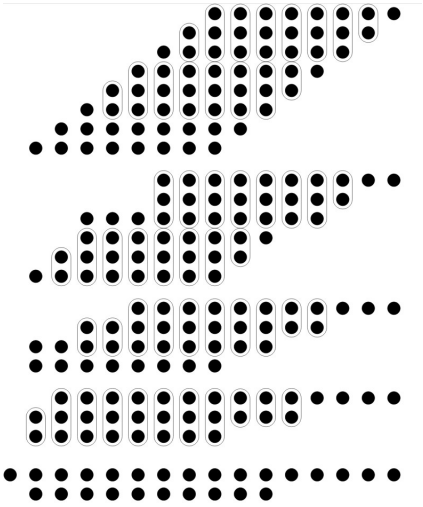
Wallace Tree multiplication is a reduction multiplication technique. You multiply bitwise, reduce the number of products, and add wires until completion.

The time the Wallace tree takes to complete is very long, as it involves a series of reduction layers. The inputs are fed into one layer, and propagated through to the end.



The code for a Wallace multiplier is very long (2000+ lines) and it had to be generated using a python script. The entire multiplication seems to need about 6 cycles to complete.

This image from Wikipedia illustrates the reduction method used in Wallace Tree Multiplication



Wikipedia contributors. (2021, January 9). Wallace tree. In *Wikipedia, The Free Encyclopedia*. Retrieved 04:49, April 29, 2021, from https://en.wikipedia.org/w/index.php?title=Wallace_tree&oldid=999294309

The module connections for the multiplier is provided below

```
module wallace_mul(
    input [31:0] a, b,
    input [1:0] mulop,
    output logic [63:0] f
);
```

And the corresponding connections

```
wallace_mul MULTIPLIER(
      .a(forwardmux1_out),
      .b(forwardmux2_out),
      .mulop(reg_in.funct3[1:0]),
      .f(mul_out)
);
```

Note that the multiplier has not yet been split into cycles, so there is no clock or reset.

The module connections for the divider

```
module divider_unsigned
(
      input logic clk,
      // divopals are self-explainatory
      input logic start,
      input logic [31:0] a,
      input logic [31:0] b,
      output logic [31:0] q,
      output logic [31:0] r,
      output logic done
);
```

And the corresponding connections

```
divider DIVIDER(
      .clk,
      .start(div_start),
      .a(forwardmux1_out),
      .b(forwardmux2_out),
      .divop(reg_in.funct3[1:0]),
      .done(div_done),
      .f(div_out)
);
```

Testing and Performance:

As far as performance goes, the thing to consider is how many cycles a multiplication or division takes. From a quick analysis, it seems the multiplication takes somewhere between 6 to 8 cycles to complete. This seems decently fast.

Testing was done with running a test assembly program as well as running a custom test bench. The results obtained from the multiplier were the same as what was expected. An example testbench is provided below. This testbench tests both signed and unsigned multiplication. Note that the numbers are tweaked to test different values.

```
module testbench;
`timescale 1ns/10ps


logic [31:0] a, b;
logic [1:0] mulop;
logic [63:0] j, k, f, curr_product;


wallace w (.*);

initial begin
        assign curr_product = w.curr_product;
        for(j = 0; j < 32'hFFFFFFFF; j = j+25418677) begin
                for(k = j%27382991; k < 32'hFFFFFFFF; k = k+27382991) begin
                        for(int i = 1; i < 4; i++) begin
                                mulop = i;
                                a = j[31:0];
                                b = k[31:0];
                                #1
                                // */
                                if(i == 1) begin
                                    if ($signed(f) != $signed(a) * $signed(b)) begin
                                        $error("Product Error! %d * %d != %d", $signed(a), $signed(b), $signed(f));
                                        $finish;
                                    end
                                    else $display("%d * %d = %d", $signed(a), $signed(b), $signed(f));
                                end
                                if(i == 2) begin
                                    if ($signed(f) != $signed(a) * $unsigned(b)) begin
                                        $error("Product Error! %d * %d != %d", $signed(a), $unsigned(b), $signed(f));
                                        $finish;
                                    end
                                    else $display("%d * %d = %d", $signed(a), $unsigned(b), $signed(f));
                                end
                                if(i == 3) begin
                                    if ($unsigned(f) != $unsigned(a) * $unsigned(b)) begin
                                        $error("Product Error! %d * %d != %d", $unsigned(a), $unsigned(b), $unsigned(f));
                                        $finish;
                                    end
                                    else $display("%d * %d = %d", $unsigned(a), $unsigned(b), $unsigned(f));
                                end
                                // */
                        end
                end
        end
        $finish;

end



endmodule
```