EECS 112L

Lab 3

Pipe Lined Processor

By:
Diego Torres
Student ID: 12602118

Date of Experimentation: 24th March 2018

Introduction: For this lab we were tasked to design a pipelined processor. This means that the execution of the instructions becomes layered. This is to increase the amount of instructions that can be processed in a given time. The time that it takes for a single instruction to be processed completely, remains unchanged or might become slower. However for large amounts of instructions the overall speed should increase by more than a factor of two.

Implementation:
The pipeline registers were created using the flopr module. There are a total of 4 pipeline register modules which contain varying amount of flopr modules inside of them. The four pipeline registers include the flopr_IFID, flopr_IDEXE, flopr_EXEMEM, and flopr_MEMWB. Signals that were required by other modules in later stages are written into these pipe line registers.
This an example of a **pipeline register**. Essentially a flopr for every input signal.

```
moduleflopr_IFID(
        input logic clk, reset,
        input logic [1:0]aluop, ....
        …….
        output logic [1:0]aluop_IDEXE,
        ……
);
flopr   (#2)    aluop_flpr(clk, reset, aluop, aluop_IDEXE);
endmodule
```

The code was segmented in a way that describes the physical placement or order of the execution stages of the pipeline like that seen in the single cycle.

The **Hazard Detection unit** utilized this code implementation:

```
    if((src1 == destR_IDEXE || src2 == destR_IDEXE) && MemRead_IDEXE)
        PCWrite = 1'b1;
        IFIDWrite = 1'b1;
        set_Zero = 1'b1;



    else
        PCWrite = 1'b0;
        IFIDWrite = 1'b0;
        set_Zero = 1'b0;
```

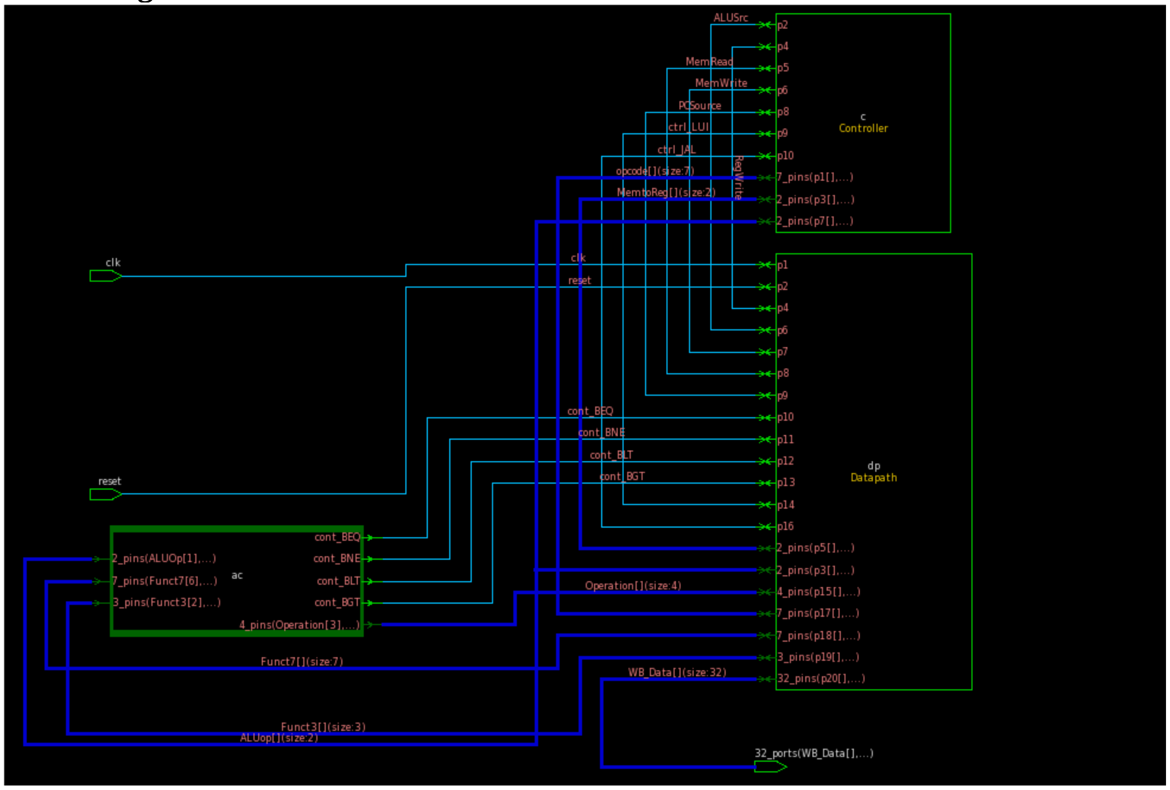
The **Forwarding unit** utilized this code implementation

```
assign sel_A =        ((dest_reg_EXEMEM == src1_reg_IDEXE) &&
                          (RegWrite_EXEMEM == 1'b'1)) ? 2'b01 :
                          ((dest_reg_MEMWB == src1_reg_IDEXE) &&
                          (MemtoReg_MEMWB == 2'b01)) ? 2'b10 : 00;
```

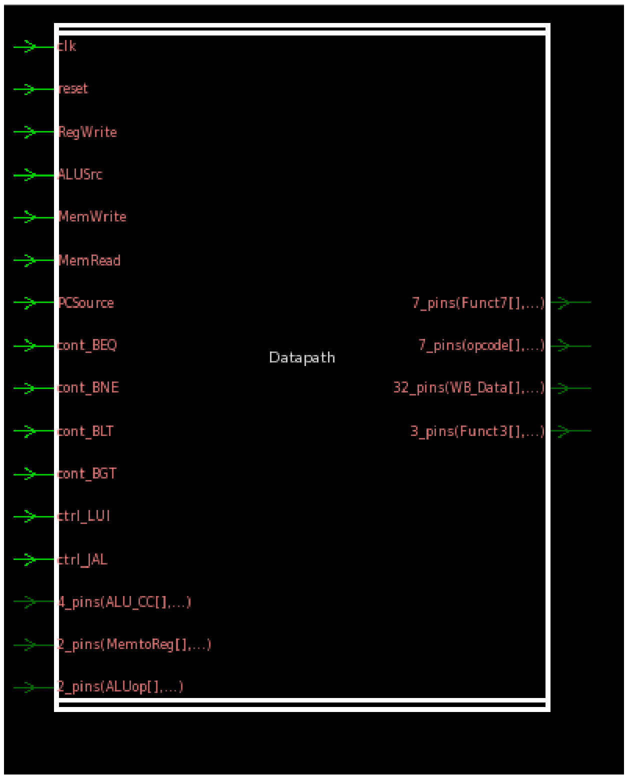
```
assign sel_B =          ((dest_reg_EXEMEM == src2_reg_IDEXE) &&
                               (RegWrite == 1'b'1)) ? 1'b01 :
                               ((dest_reg_MEMWB == src2_reg_IDEXE) &&
                               (MemtoReg_MEMWB == 2'b01)) ? 2'b10 : 00;
```

Different signals from each stage are combined together to find the value of selA and selB which are used to determine the value sent to the alu.

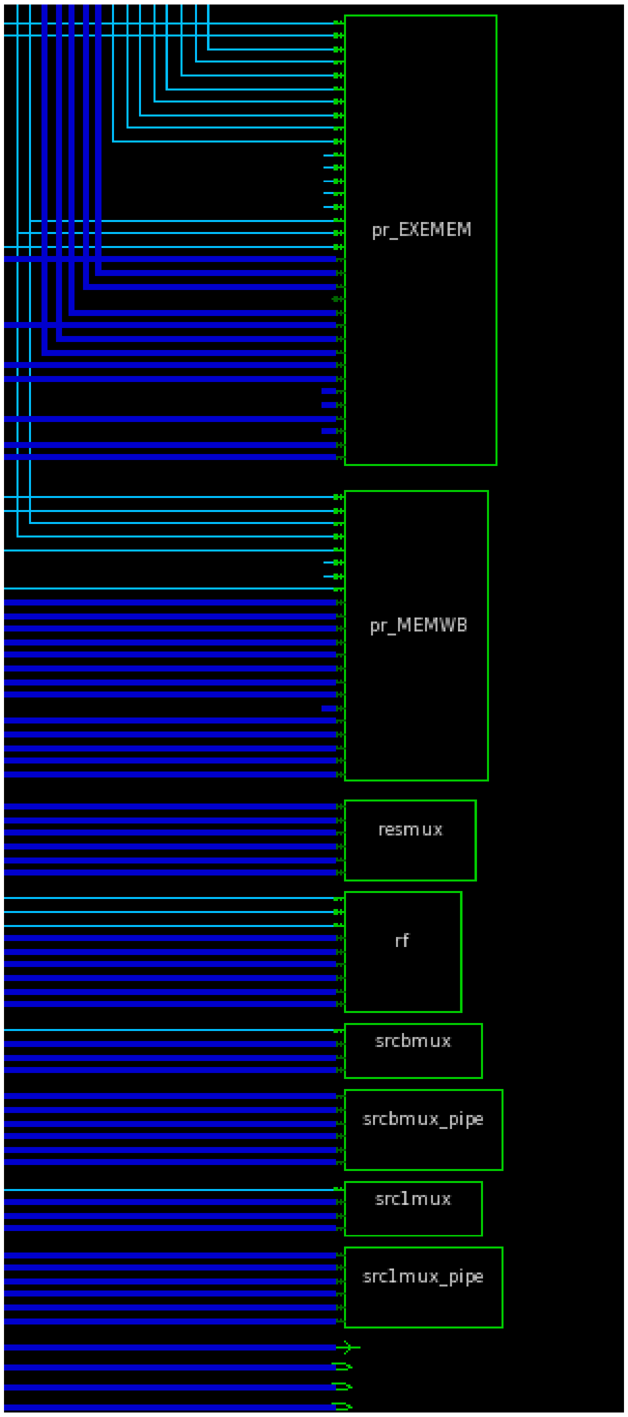
Block Diagram:



This is the schematic layout for the RISCV. The outputs and the inputs have not changed substantially. The only major change was the selection of the input from the stage. The input types were still the same, the opcode, funct3, funct4, and control signals like RegWrite, MemRead,MemtoReg, MemWrite, and so on. The names changed because each input had to be selected from a specific stage inside the pipeline.

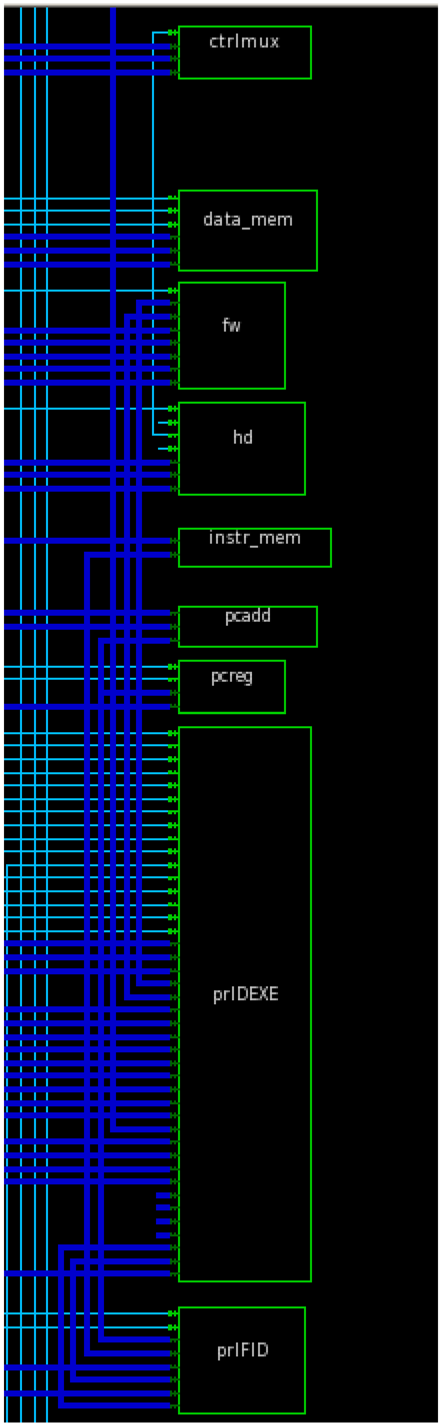


All of the major changes occurred inside the Datapath, within this module the instantiation of the additional Hazard Detection and Forwarding Unit are placed inside. The amount of the ports has remained unchanged from the original single cycle design.



In this overview the two of the pipeline register can be seen which are pr_EXEMEM and pr_MEMWB. As the name of the pipeline register suggests the pr_EXEMEM lies inbetween the execution stage and the memory stage, and the same applies for every other pipeline register. This order is with respect to the order of the execution for the stages and not the physical order from left to right of the data path. For instance the Write Back stage occurs on the left side but continues on the right hand side.

pr_EXEMEM contains the signals like z, lt, gt, pc_sel, alu_result, ctrl_JAL, dest_reg, and rd2. The first set of signals listed are to determine the value of the PC that will be written back depending on the type of instruction that is provided i.e. branching. In pr_MEMWB the signals that are stored are alu_result, RegWrite, and dest_reg. These signals are stored because they are needed when the values are written back to the register. Additional signals were included in the source code that were not used, but were included just incase there is further use.

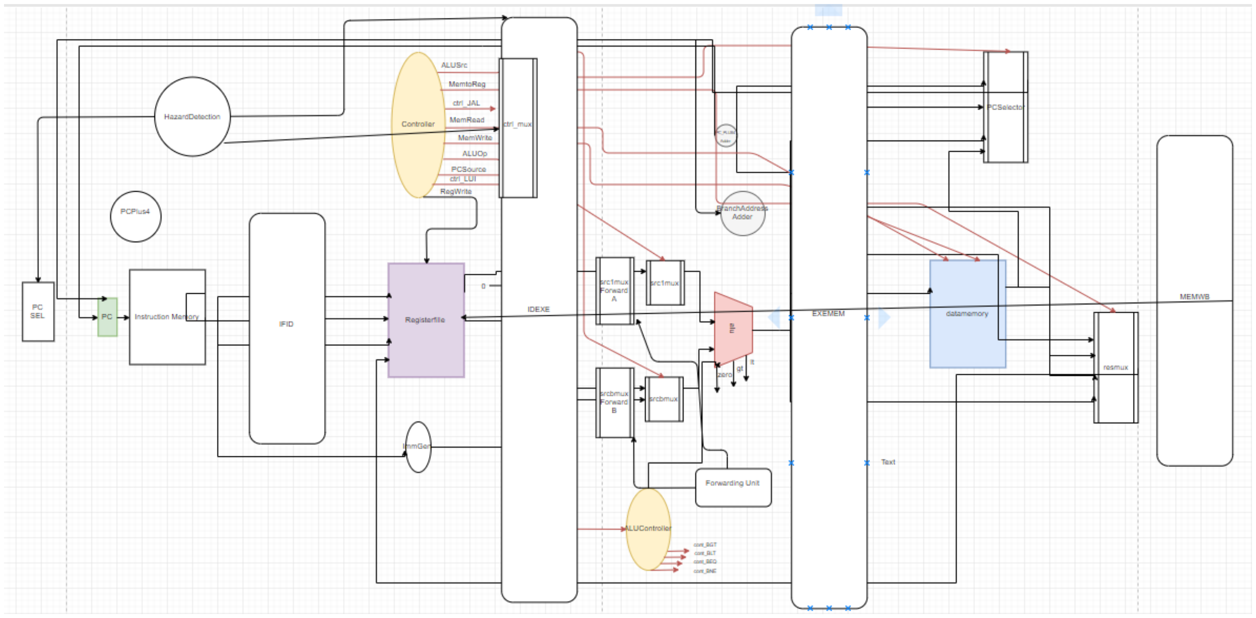


In this overview the pipeline registers shown are pr_IDEXE, and pr_IFID. For pr_IDEXE there exist the most unique signals which means the most floprs. All of the control signals from the Controller module are passed into this pipeline register like ctrl_LUI, ctrl_JAL, MemREAD,MemWrite, RegWrite and so on. Before they can be placed directly from the controller to the pipeline register they must be put through a multiplexor that determines whether, the control signals should be passed or not. This is for hazard detection and the implementation of a NOP. These signals are propagated through each stage. In the pr_IFID, the values stored are the PC, the instruction at that stage, and the next PC instruction.

The Hazard Detection block determines whether or not a stall is placed in the current cycle. This is going to be caused by a load instruction which reads the data from memory and that data is fed into the next input of the cycle. The output of the Hazard Detection includes a signal to increment the PC, flush the pipeline register, and flush the values of the control signals to include the NOP.

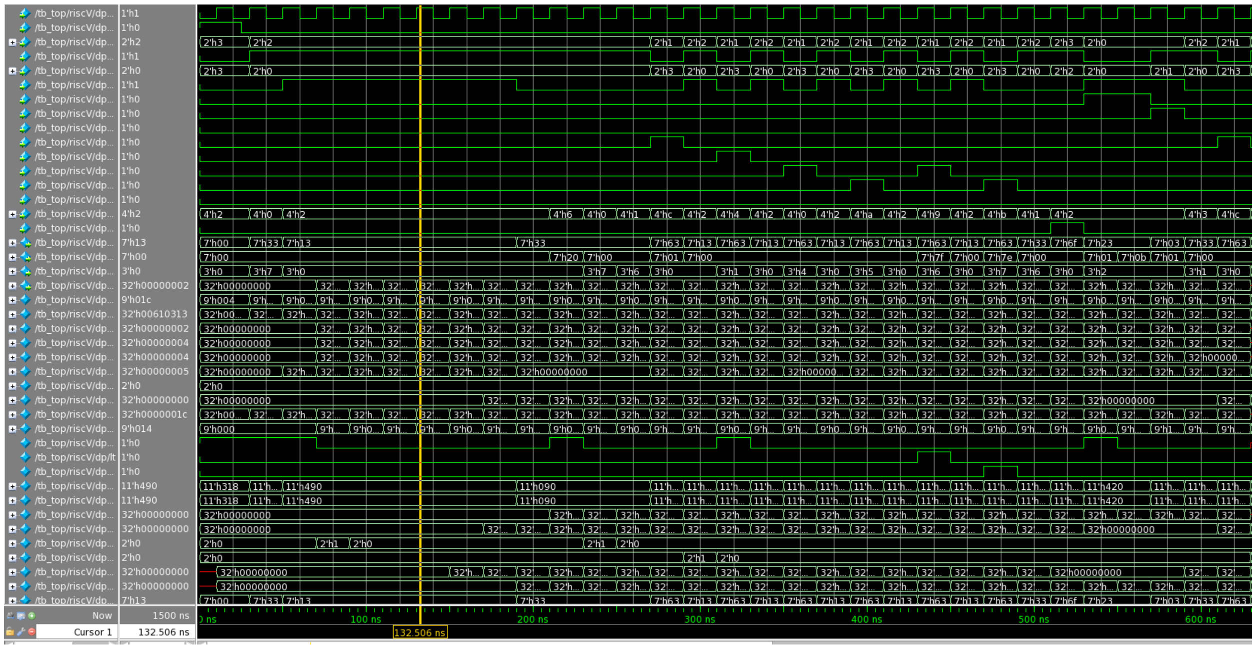
The forwarding unit used when there register writes and register reads sequentially. This can result in needing the alu result from the previous instruction and placing it in the new instruction. There is also an instance when the read result from the memory is needed in the alu is needed. The forwarding unit controlled by the source's address signals and the destination address signals. If they are similar and there is a write back to the register file then the outputs of the forwarding unit selA and selB are determined and choose the values that are inputted into th ALU.

More Generalized View



Waveform:

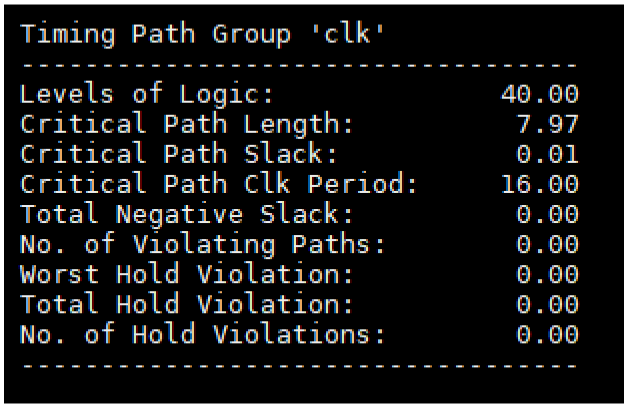
- Screenshot



-

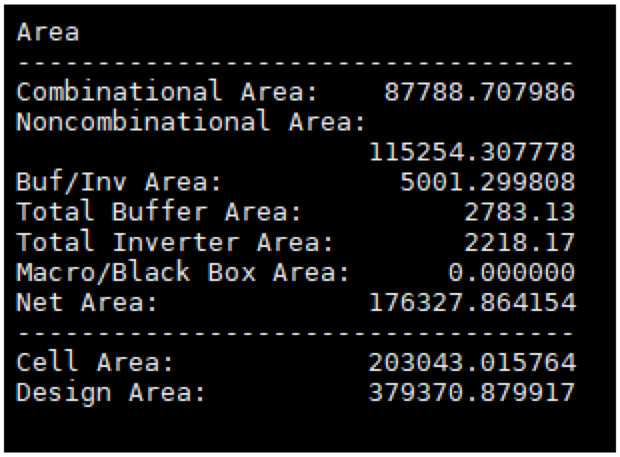
Synthesis Results:

- Critical Path Length
- Critical Path Slack



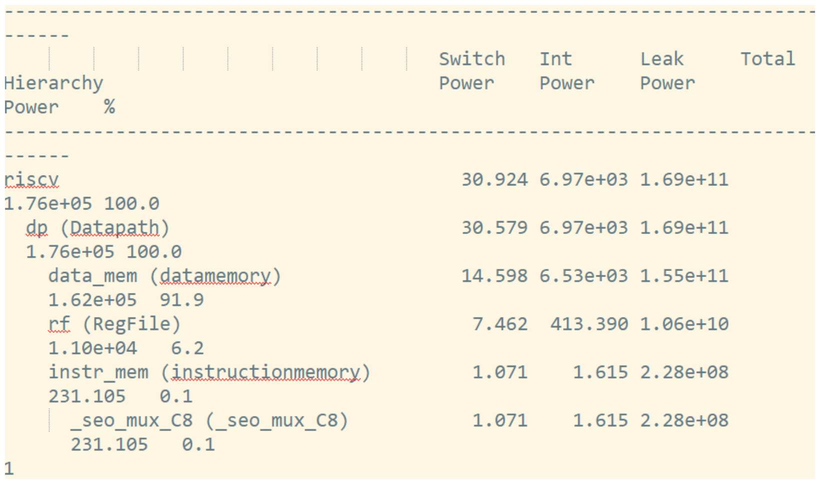
```
Timing Path Group 'clk'
-----------------------------------
Levels of Logic:                 40.00
Critical Path Length:             7.97
Critical Path Slack:              0.01
Critical Path Clk Period:        16.00
Total Negative Slack:             0.00
No. of Violating Paths:           0.00
Worst Hold Violation:             0.00
Total Hold Violation:             0.00
No. of Hold Violations:           0.00
-----------------------------------
```

- Area



```
Area
-----------------------------------
Combinational Area:     87788.707986
Noncombinational Area:
                       115254.307778
Buf/Inv Area:            5001.299808
Total Buffer Area:           2783.13
Total Inverter Area:         2218.17
Macro/Black Box Area:       0.000000
Net Area:              176327.864154
-----------------------------------
Cell Area:             203043.015764
Design Area:           379370.879917
```

- Synthesis Results of Pipeline vs Synthesis Results of Single Cycle



```
------------------------------------------------------------------------
------
                                       Switch   Int      Leak     Total
Hierarchy                              Power    Power    Power
Power    %
------------------------------------------------------------------------
------
riscv                                    30.924 6.97e+03 1.69e+11
1.76e+05 100.0
  dp (Datapath)                          30.579 6.97e+03 1.69e+11
  1.76e+05 100.0
    data_mem (datamemory)                14.598 6.53e+03 1.55e+11
    1.62e+05  91.9
    rf (RegFile)                          7.462  413.390 1.06e+10
    1.10e+04   6.2
    instr_mem (instructionmemory)         1.071    1.615 2.28e+08
    231.105   0.1
      _seo_mux_C8 (_seo_mux_C8)           1.071    1.615 2.28e+08
      231.105   0.1
1
```

The results of the synthesis from the pipeline differ from that of the single cycle. There is a performance improvement, when compared to before. Before the clock period was at the value of 20. In this implementation the value used for the clock period is 16. While there isn't a massive improvement overall with new pipelined design, with more refinement of controls and optimization of the code the difference should increase. The area also increased to include more of the pipelined registers as well as the additional Hazard Detection Unit as well as the forwarding unit. That is potential drawback of my implementation as there may not be enough additional throughput for the amount by which the area increased. The critical path length decreased substantially from 19 in the single cycle to 8 in the pipelined. This is much better for timing. The power reports differ from that of the single cycle. The amount of power that is leaked increases as a result of more components be introduced as well as decreasing the clock period and thus increasing the frequency at which the processor operates at. This will result in power being consumed overall.

Conclusion

This lab was very informative in the learning the techniques for implementing the way to handle pipelining vs single cycle. The process is much more demanding because more than one instruction is being handled at a time. This will lead to a much more complex Datapath involving much more signals and internal variables to store the values being stored for each stage.