NYU-6463-RV32I Processor Design

Fred Zhao -- YZ8751

Junqing Zhao -- JZ5954

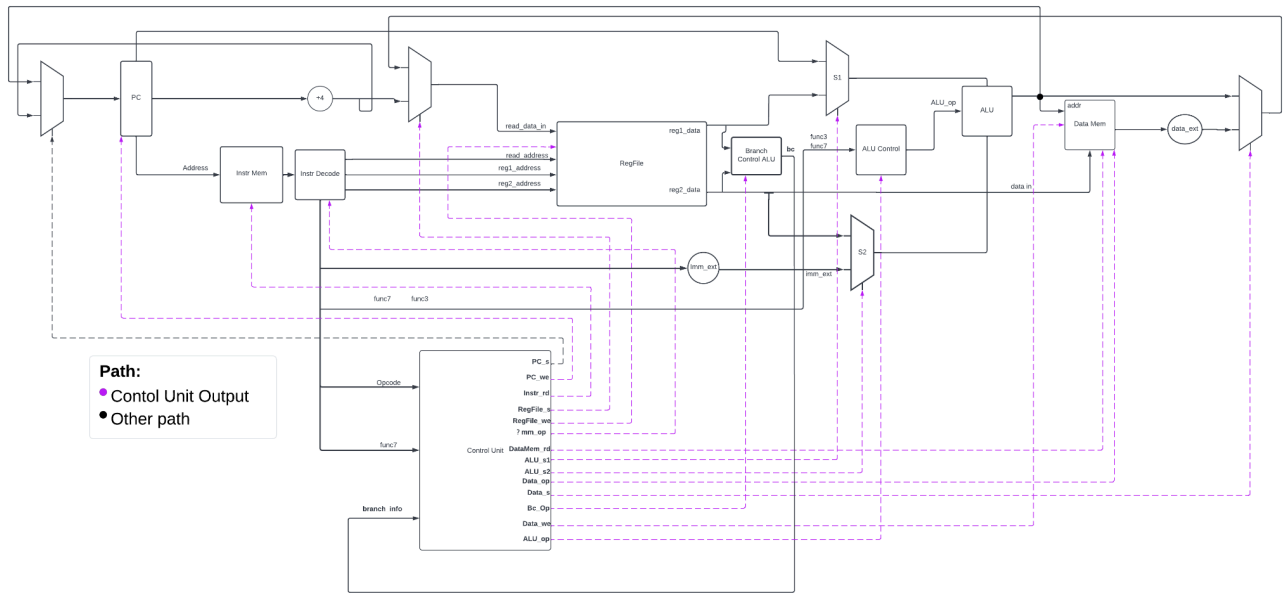
Rongze Li -- RL4670

NYU Tandon School of Engineering

12/17/2022

CONTENTS

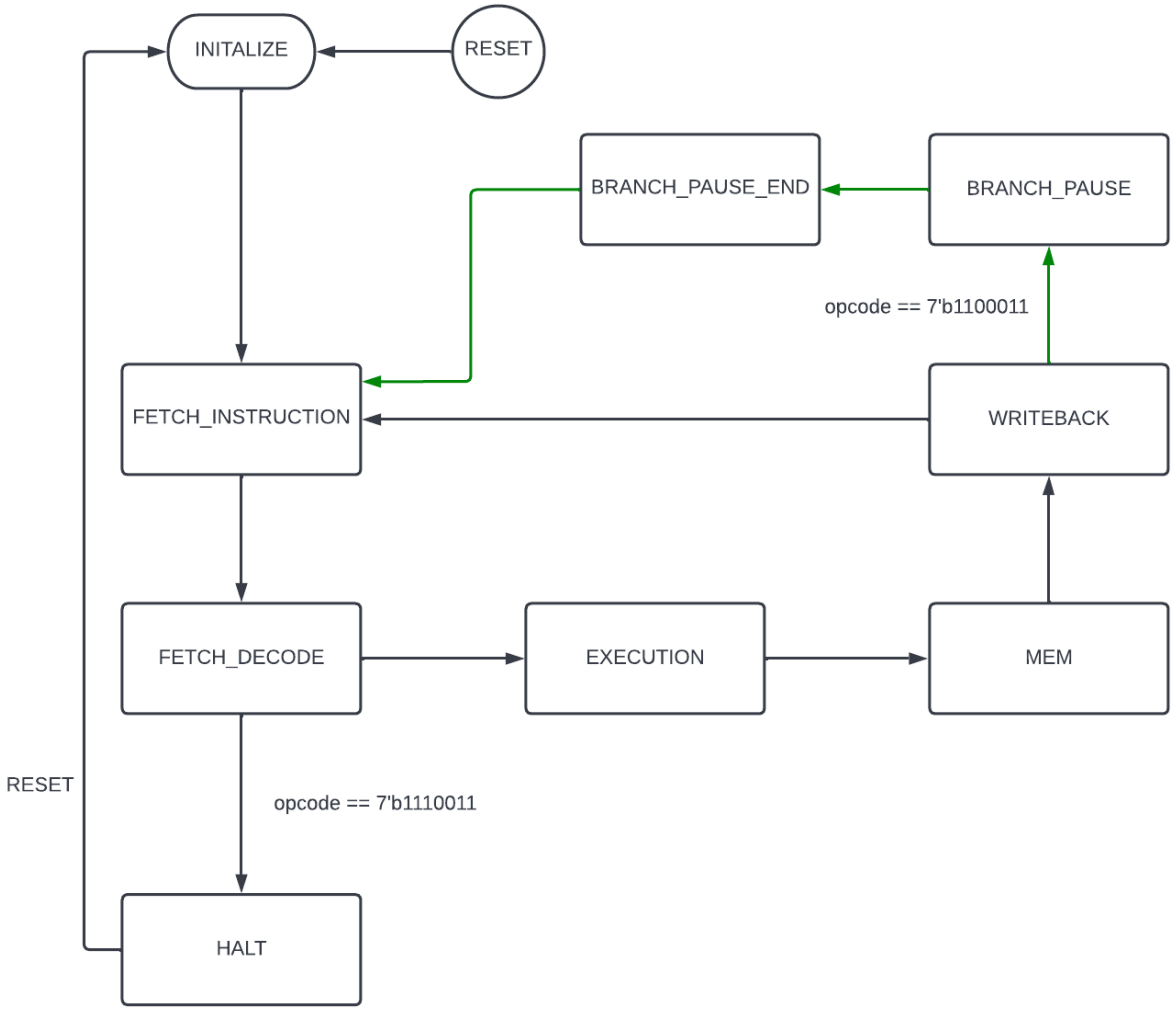
1. Complete Design Datapath



A Control Unit is implemented in the data flow for better illustration.

2. FSM Diagram for Multi-cycle Control

The control unit Finite State Machine is showed in the below figure, we can see that the main process is consisted of 5 stages, and there are some other stages that are action or operation code specific, which will be elaborated below.



The 5 common stages of the FSM is:

1. Fetch Instruction: at this stage, Program counter updates and generate new address for instruction and instruction memory retrieve the specific instruction

2. Fetch decode: at this stage the instruction decoder would decode the instruction and send it to different components, including control unit, ALU, and Register File.
3. Execution: at this stage the ALU should compute the results of the required calculations.
4. Mem: at this stage, if the instruction requires to write to data memory, the data memory will store the information that is calculated from the ALU and store it into data memory
5. Write back: at this stage, the data will be transmitted back to the register file and program counter so they can perform the required operations

There are some other stages for specific action of operation codes:

1. Halt: the processor enters halt stage when the decoder returns operation code indicating a halt. The halt stage can only be exit using reset.
2. Initialize: this is the stage the processor enters once the reset is hit.

However, even thought those stages perform perfectly in Vivado simulations, the branching operation takes longer time in real like in regards to the specific hardware we are using, so the branching operation will require two extra stages:

1. Branch pause: this stage simply pause for the branching operations to finished in real life.
2. Branch pause end: this indicates the end of the branch wait, and it writes back to the memory like the write back stage

3. Justification for the Correctness of Individual Components

a. ALU

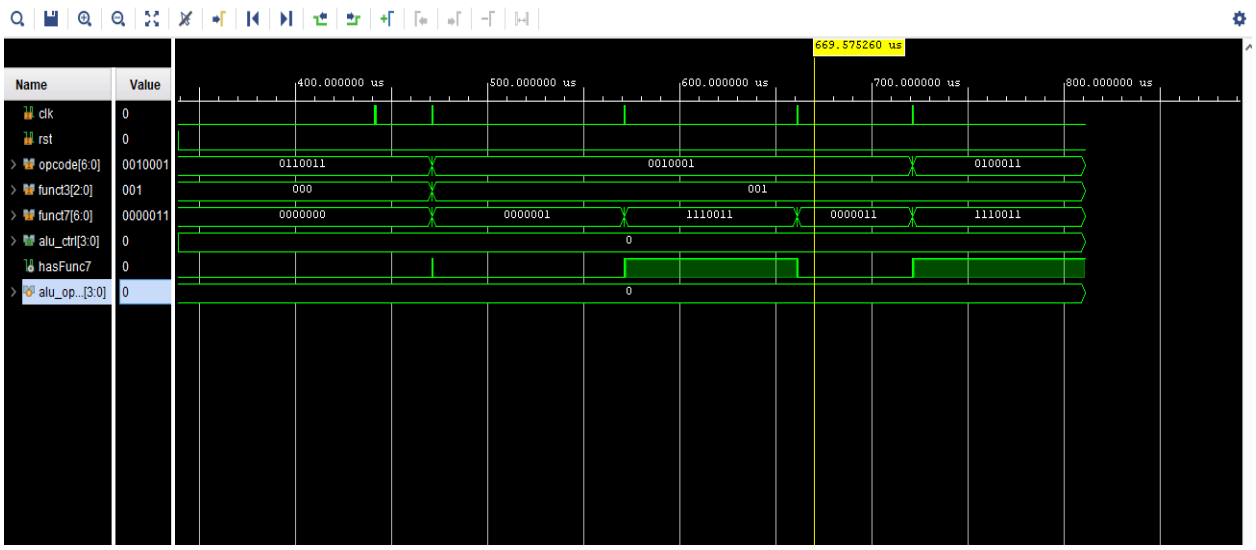
The ALU performs required calculation of the instructions. ALU responsible for following operations: Addition and subtraction of integers; Logical operations such as AND, OR, and XOR; Shift operations, which move the bits in a word left or right; Comparison operations, which determine if one number is greater than, less than, or equal to another; Conditional moves, which allow the processor to choose one of two values based on a condition. ALU components is unaware of the opcode and funct3 instead it operates on the operation code given by ALU control.

b. ALU Control

ALU control unit is responsible for selecting the specific operation that the ALU should perform on the input data. It does this by decoding the instruction that is passed to the processor and determining the appropriate operation to execute.

The ALU control unit receives input from the instruction decoder, which has already determined the type of instruction that is being executed. It then generates control signals that tell the ALU which operation to perform and how to interpret the input data.

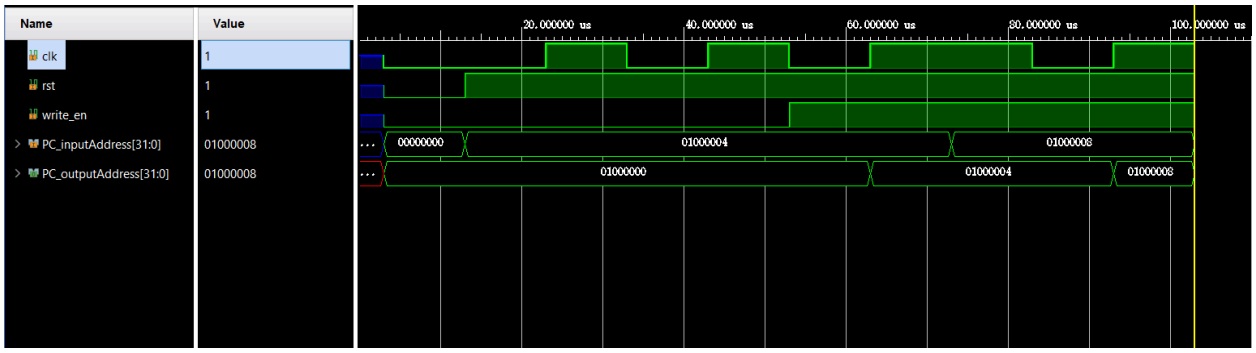
In this design, the ALU control unit takes in opcode, funct7, and funct3 and output operation code that can be recognzied by the ALU to perform required calculations. And there is a funct7 checker in ALU control unit.



c. Program Counter

Program counter outputs the next address for instruction memory to get the next instruction. The program counter only takes in an address and output it when it's enabled, it does not perform any calculation. On reset, the pc counter will be set to 0x01000000.

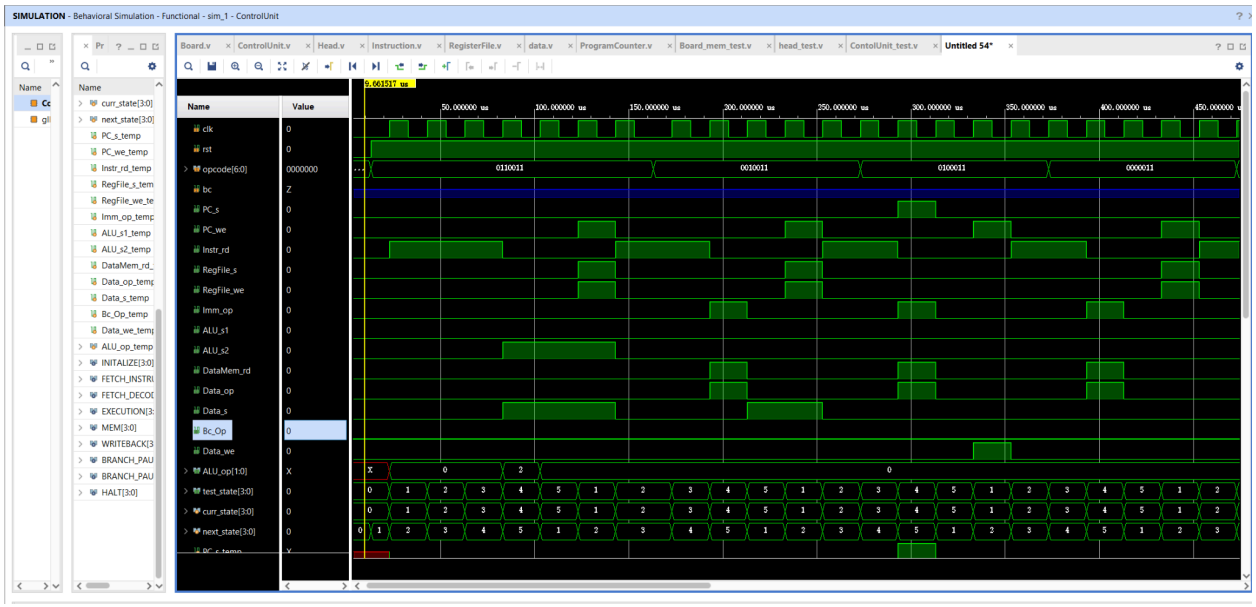
As we can see in the graph below. The simulation of the prorgam counter works as on reset it is set to the correct value and without write enable signal it will not write the input.



d. Control Unit

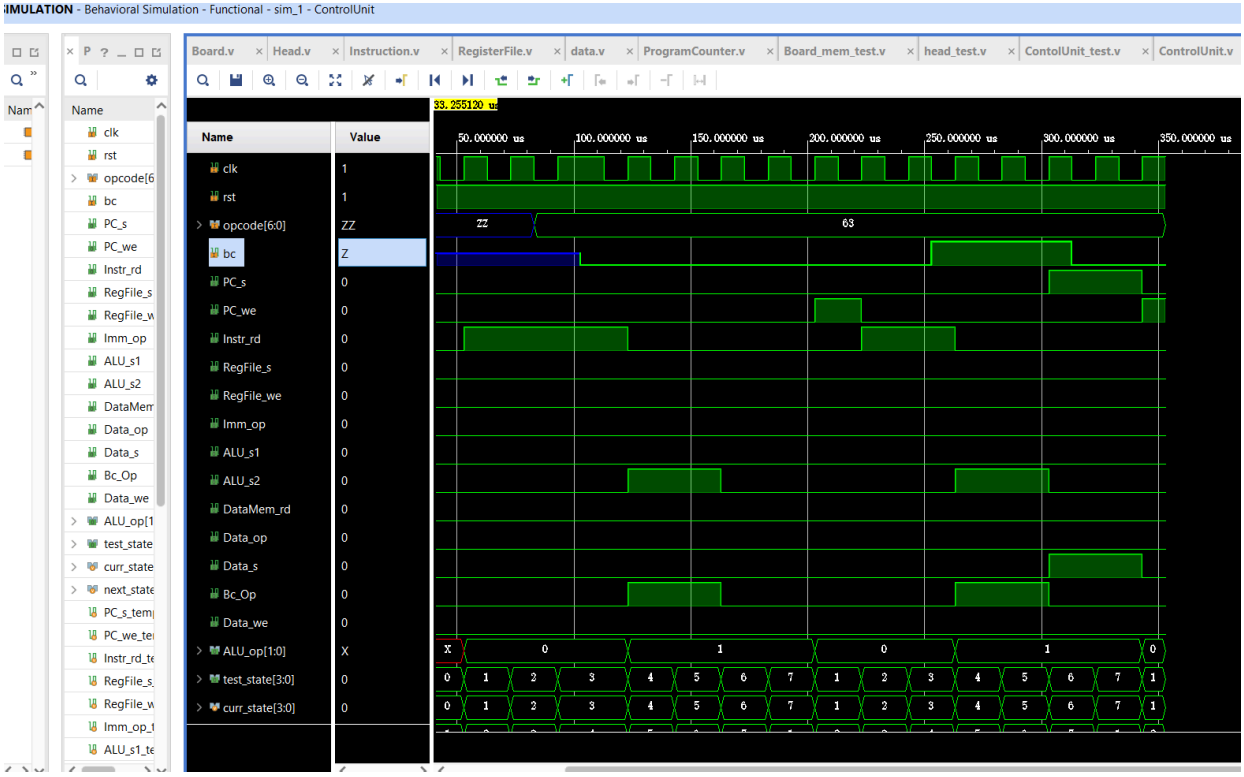
Control unit implement the finite state machine as described in the earlier section. This component control the state of the whole program and the on and off of different components as we can see in the Datapath in the earlier section.

Below is the testbench of the opcodes besides branch, they all take 5 clock cycles to be refreshed, the opcodes update after the instruction memory fetches new instruction at state 2.



The Branch opcode simulation is showed below, as we can see that the branch operation takes 7 cycles since the branch control unit may need one more extra cycle to process in real life, yet in simulation, without the extra state, the branch operation should work fine as well. We can see in the test bench that the control unit changes its

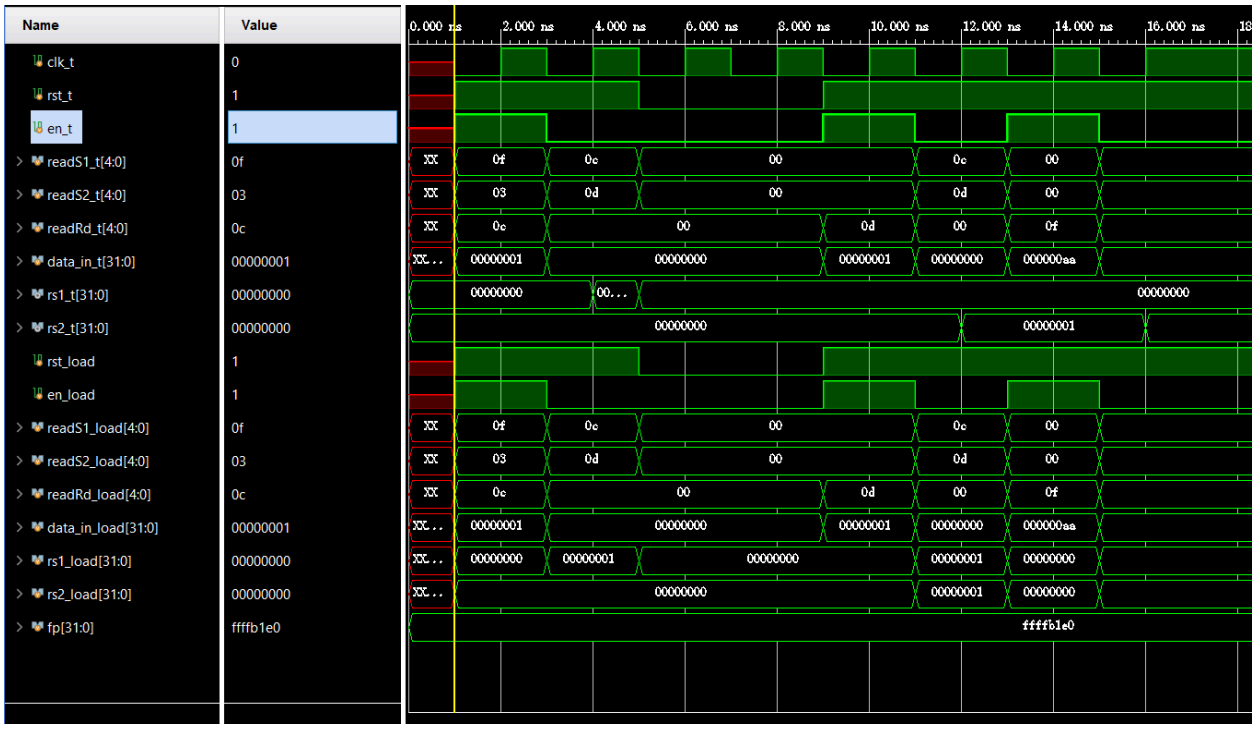
control signal based on the branch control unit output, which indicates whether the control unit should perform a branch operation or not.



e. Register File

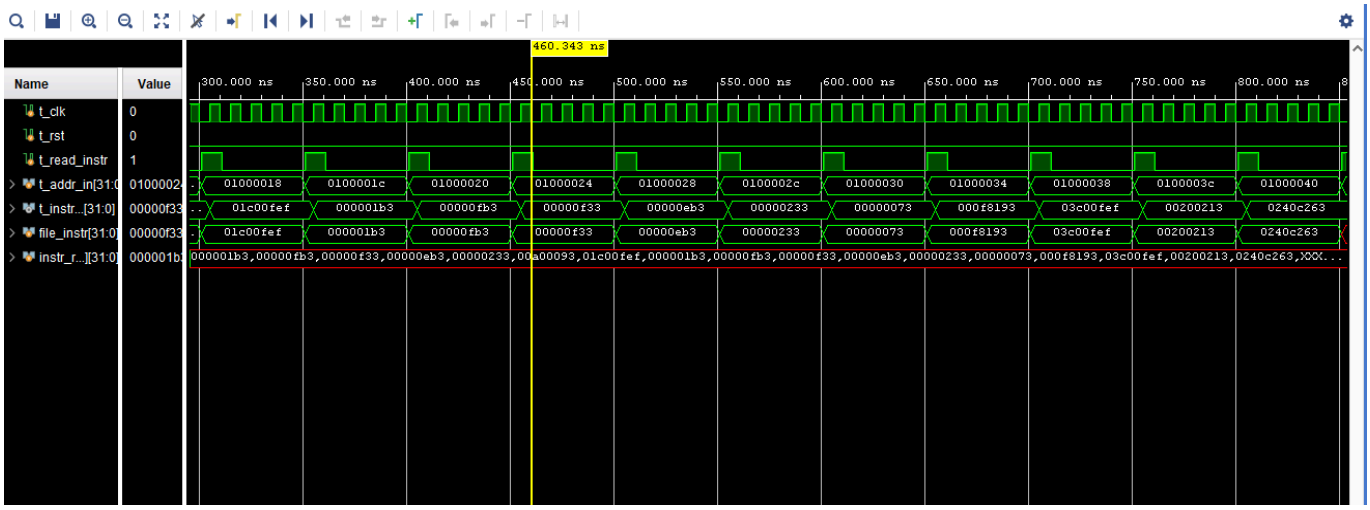
The register file is a central component that stores a set of high-speed data registers that can be quickly accessed by the CPU. These registers are used to store intermediate values and data during the execution of instructions.

As the simulation screenshot show below, the register file works well and it returns the correct values



f. Instruction Memory

The instruction memory is initialized to contain the program to be executed. The width of the instruction memory width is 4 bytes (32-bits). The maximum capacity of the instruction memory is 512 32-bits instructions. The instructions are read from memory file by get address from the program counter.



g. Data Memory

The data memory component is a read-access memory. Except few special read-only addresses, these addresses used for store certain constent as requirements: address 0x00100000, 0x00100004, and 0x00100008 will store N number as value; 0x00100010 used to implement read-only switches, 0x00100014 used to implement read-write LEDs:

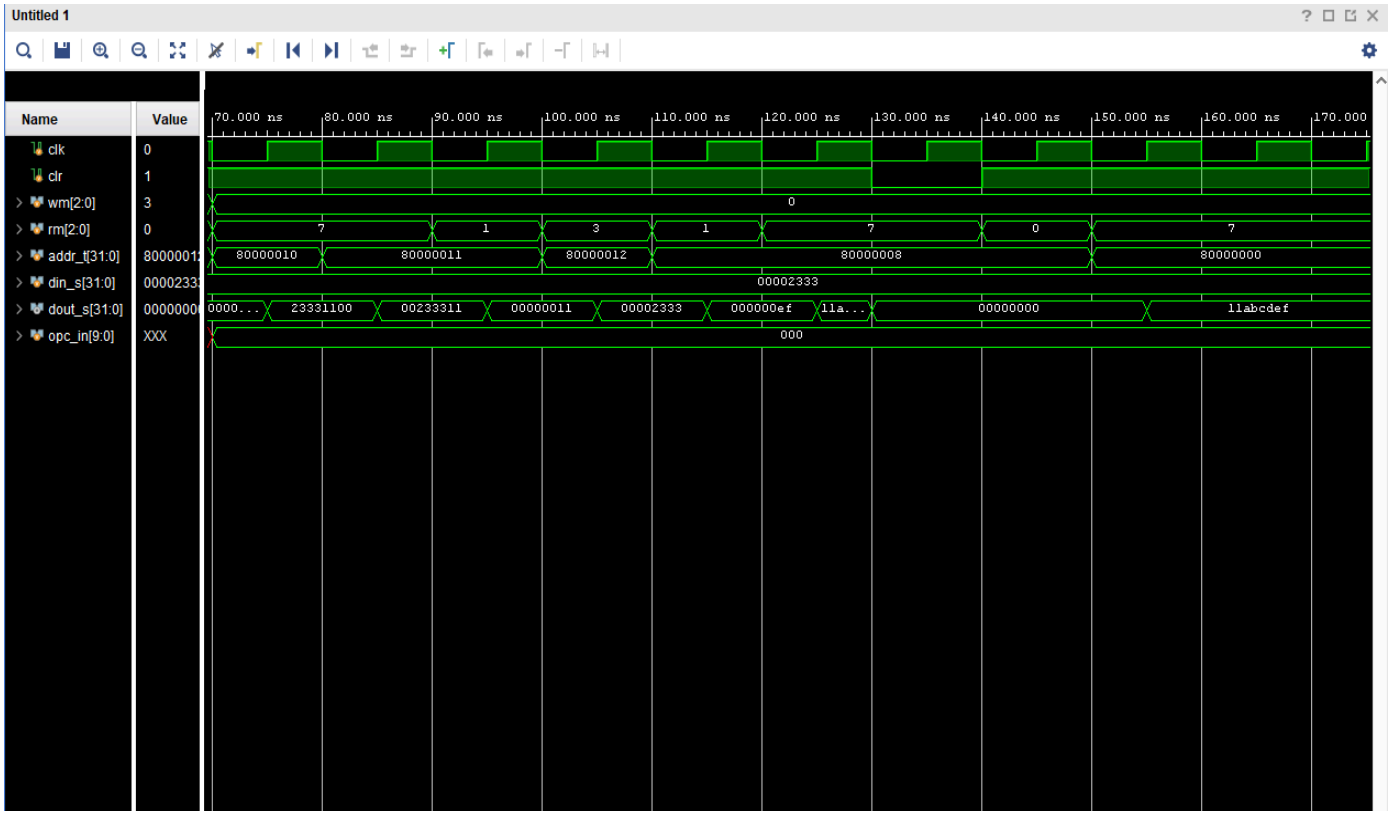
Address 0x00100000 contains 0x13000639

Address 0x00100000 contains 0x10923038

Address 0x00100000 contains 0x19039807

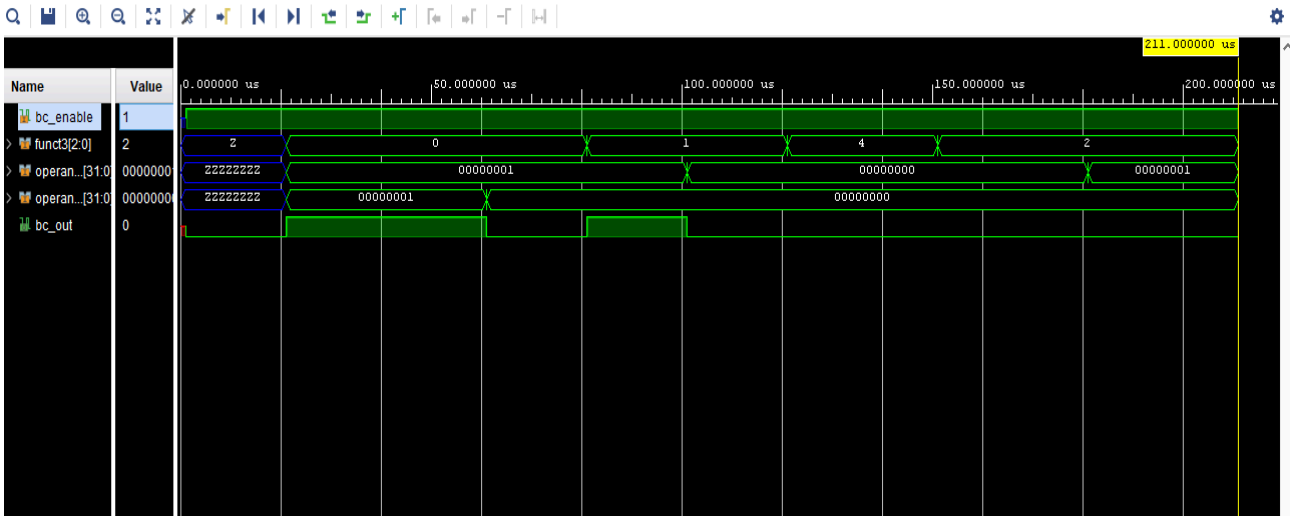
Address 0x00100000 contains 0x00000000

The extension function of data memory is implemented as an individual component. The maximum capacity of data memory is 1024 bits.



h. Branch Control

Branch control is the another component that does calculation but only when there is branch operation, it outputs a branch control signal that gets sent to the control unit and then the control unit will use that signal to adjust other signals sent to the other parts to perform a branch operation or continue to next command if it does not require branching.



4. Future Optimization

There are few ways to optimize the performance in the future, since the team is limited by the due date. On a high-level perspective, there are a few ways we can optimize the overall performance of the CPU

1. Use a high-performance implementation of the RV32I, such as using a high-speed pipeline or implementing advanced branch prediction techniques.
2. Use efficient algorithms and data structures: Choosing the right algorithms and data structures can significantly impact the performance of the program.
3. Use compiler optimizations: Compilers can often apply various optimization techniques to the program to improve its performance.
4. Use multithreading and parallelism: If the workload is highly parallelizable, multiple cores or threads techniques can be used to improve performance by leveraging.
5. Use power management techniques: Reducing the power consumption of the processor can improve its overall performance by allowing it to run at higher clock speeds or by extending its battery life.

In regards to the current implementation, there is a few other things we can do to optimize it

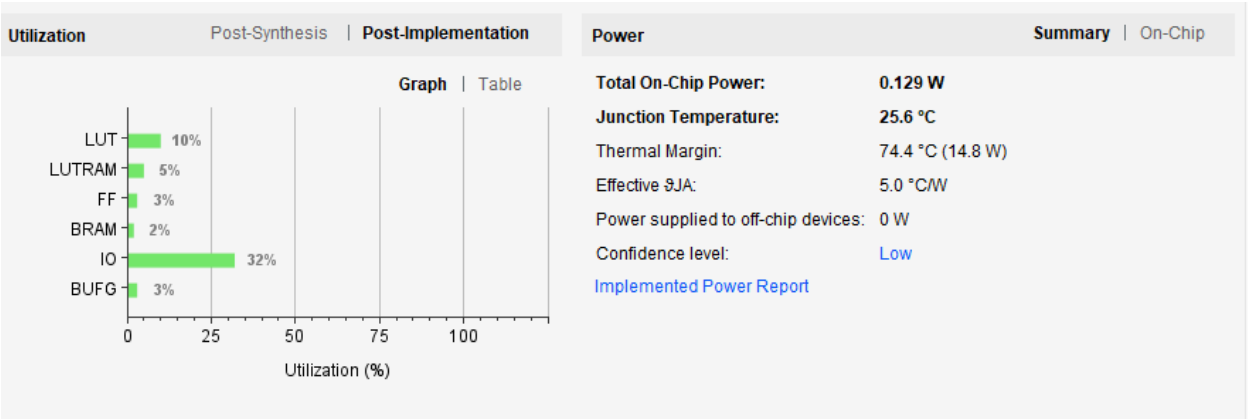
1. The branch operation does not has to be more cycles from the other operations when it runs on the actual hardware, since the ALU component and the branch control unit will get the same information at the same time. One way to opitimize

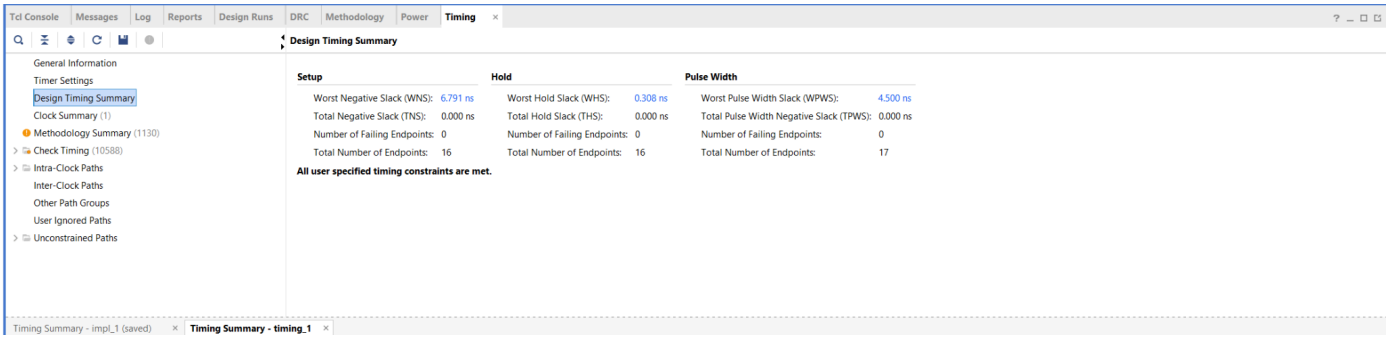
it is to try to implement the alu control and branch control together, so the cycles will be the same for each operation.

2. There are some extra stages for certain operations in the finite state machine, we can optimize it by making the state machine more opcode specific, which means some operations can be really fast instead of taking the same time as the other operation.

5. Performance and Area Analysis of the Design

Synthesis results and report from Vivado:





Area Analysis

Based on the datapath, there are 5 multipliers used. Each individual component can be operated independently, so they could be counted as slices. Totally, there are 7 slices in the processor. 3 registers are implemented separately in instruction memory, data memory, and register file.

6. Description of Complex Assembly Test Cases

Fibonacci sequence:

0100193 // addi R3 R0 1
00200213 // addi R4 R0 1
Finish initialization

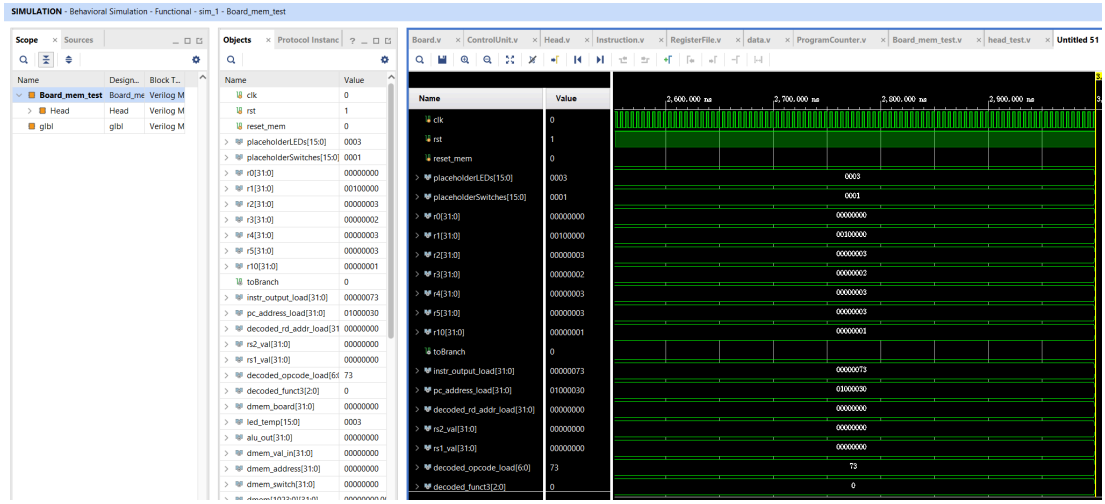
004182B3 // R5 = R4 + R3 // add R5 R4 R3 —
00020193 // R3 = R4 // addi R3 R4 0 —
00028213 // R4 = R5 // addi R4 R5 0 —
40A10133 // R2 = R2 - R10 // sub R2 R2 R10

FE2018E3 // Bne R2 R0 -16

00202A23 // sw x2 value to leds
00000073 // halt, stop running when done

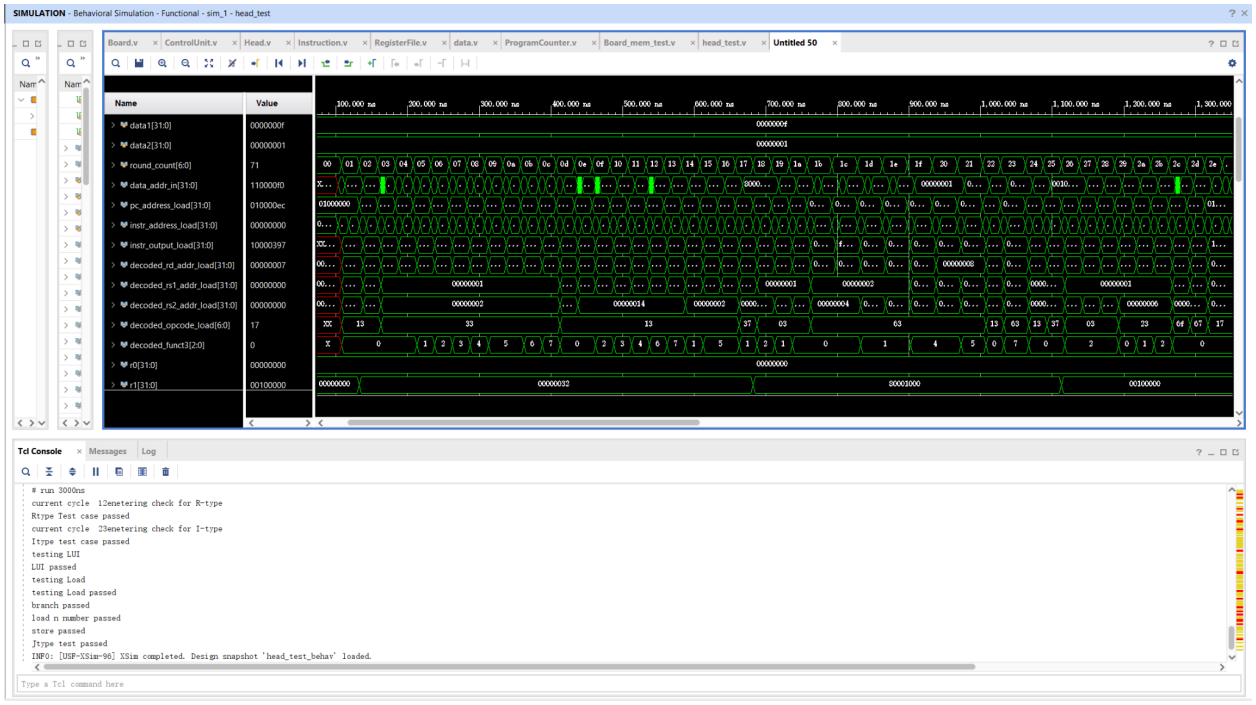
Below is the high level testbench for the Fibonacci sequence:

In the testbench we can see when the switch value is one, the output LEDs will be 3, which indicates the correct answer. However, this testbench is only to test if the Fibonacci sequence is working correctly based on the loop number. The testbench described below is the comprehensive test of the whole processor.



Besides the above, In the **instruction.mem**, we have tested all the assembly commands in the spec. You can run the memory file with the test file called **head_test.v.** Below is its testbench results, we can see the TCL console indicates that all tests have passed for all types of instructions.

To run this testbench, instruction memory needs to be **instruction.mem** and data memory should be **initialize.txt**, which is a completely empty data memory



=

7. Video of Implementation on Basys3 (XC7A35T-ICPG236C) FPGA Board

https://drive.google.com/file/d/1BSrVYlrTVl39aDF7wH7RBJufbzFHXSUd/view?usp=share_link