IITB CPU(The Journey)

Sarvadnya Desai, Ishaan Manhar, Harshraj Chaudhari, Akshat Taparia

November 2022

1 Overview

The project topic involved extensive pen paper design of a Finite state machine entity to process numerical data provided to it 16-bit registers. Operations on the data were primarily divided in 3 categories namely **Logical and Arithmetic** operations, **Data Access/Storage** operations and **Conditional/Unconditional Branch** operations.

2 Initial Approach Towards The Problem

We divided the task of state creation evenly in the following manner. Akshat was responsible for state creation of the Arithmetic operations namely ADD, ADI, ADC, ADZ. Ishaan designed the logical instructions named NDU, NDC, NDZ. Harshraj made the dataflow design for the Branch and Jumping Instructions. Sarvadnya made the states for the load and store operations SW, LW, LM, SM and the pen-paper design for the components ALU and the Programmer's Register.

Implementation was led by Ishaan who first modelled the components we would use like the Prgramer's Register, the Memory and the Muxes. Meanwhile **State minimization** and **optimization** was carried forth by Harshraj and assisted by Akshat for double checking. Most of the optimization was based on the fact that we could modify the inputs inside the ALU/ registers based on the OP Code give to the multiplexer select lines.

We implemented the design in two major entities namely **Datapath** and **FSM**. Datapath was the physical component architecture and the internal wiring of the CPU. We designed it such that it would include ALU, Temporary Registers, Programmer's Register, Signed Extensions and the Memory.

Finally, The entity labelled as FSM was a state machine which kept track of its next state based on conditional logic implemented by behavioural description. It kept track of all the control variables to the hardware architecture.

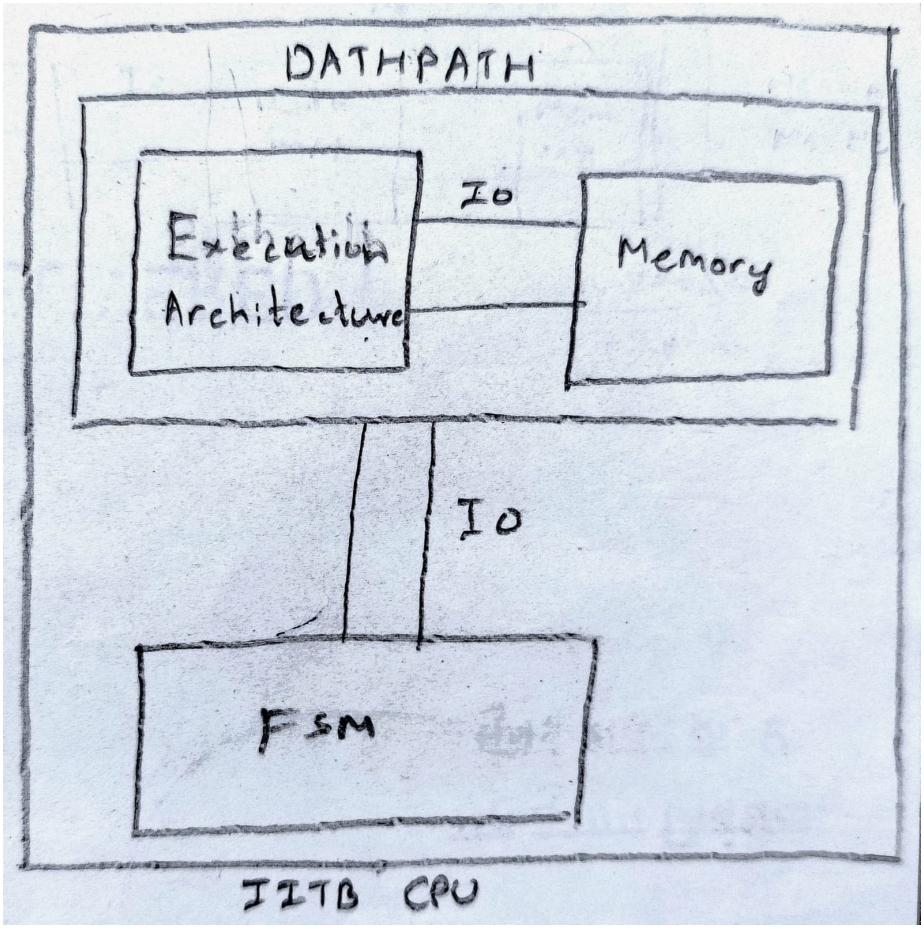

Figure 1: The division of entities inside the CPU

3 Optimization Stage

This was one of the most tedious tasks in the project. Numerous proof readings of each optimization stage were done to ensure that no instruction was misinterpret. State merging was carried forth based on methods taught in class and the use of conditionals on OP Codes in behavioural modeling.

The Strategy used here was to observe common outputs such that the inputs can be multiplexed inside based on FSM state logic statements under behavioural description. Next we thought about reordering the states to merge more and more states if possible. For example we shifted the PC update in the end so that instructions such as JAL,BEQ etc could be merged along with with the other instructions for some initial states.

4 Implementation

The implementation was carried out in two major parts. The first being the Dathpath and the second being FSM. The **Datapath** included all component instantiations and linking them with Multiplexers. The **FSM** consist of the control variables to modify the dataflow in the Dathpath constructed.

5 A Few Hurdles We Encountered

The first hurdle we encountered was designing the LM and SM operation in the FSM. We finally though a method to incorporate the bit value as control and an external counter to overcome the same. We moved forward from there to optimize this design until it became compact.

The second hurdle we encountered was the creation of the architecture and how to organize it. We referred the division in a programmable device and separately modelled the structural and behavioural entities and then finally interconnected them to design the final entity called 'IITB CPU'.

Lastly the code was debugged by Ishaan to resolved a few latch errors and syntax errors in the code. Finally we tried RTL simulation for the states using input to the memory in the memory.vhd file.

6 The Python Programmer for the CPU(Adding Personal Flavour)

Testing the device was getting tedious so we decided to add a python based programmer for the architecture. Here is how it works! You add an instruction as an input(similar to assembly), It generates the 16bit instruction. Lastly we added a cool feature where you can write an entire program. Simply parse the last instruction as 'No' and you are good to go!

7 Conclusion and Outcome

The following are the few milestones we achieved. Please refer the pictures below for some of the RTL and Netlist simulations we were able to achieve.

8 Programming the CPU for a Fun Activity

Let us program the CPU to generate a Fibonacci Series in the memory of the CPU. We shall include the machine code program and the RTL simulation for the same below.

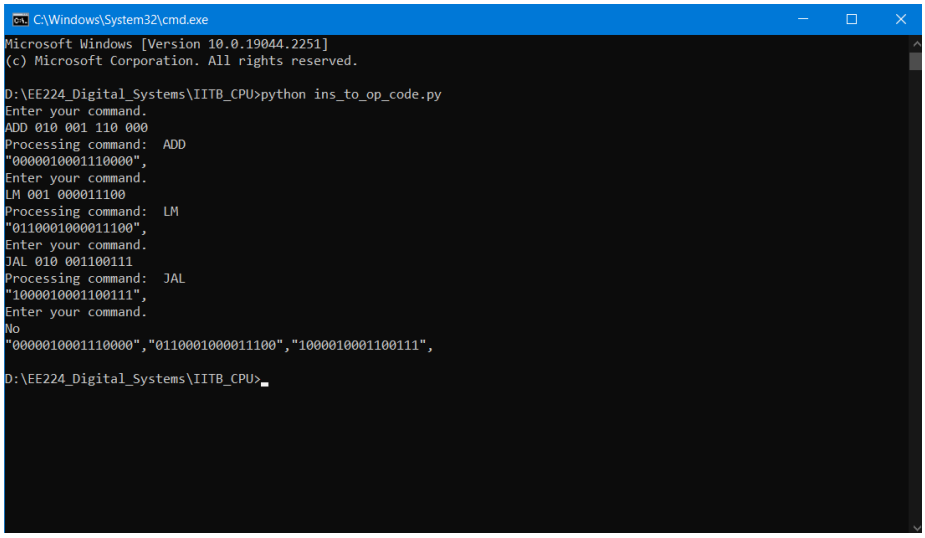

Figure 2: The execution of the program machine code generator made using python.

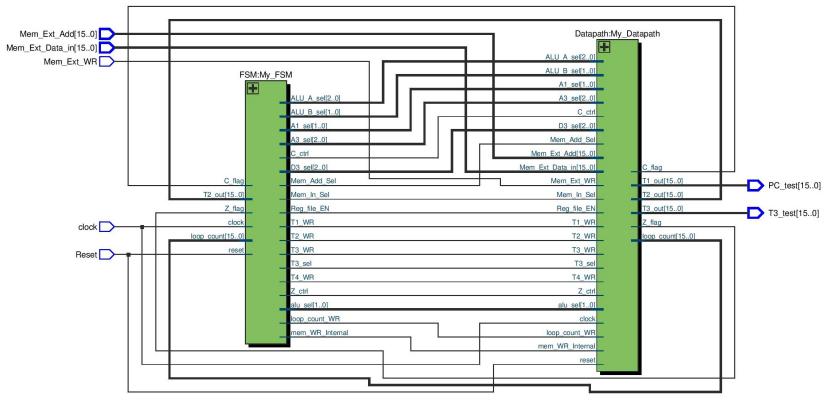

Figure 3: The RTL Netlist view of the CPU entity.

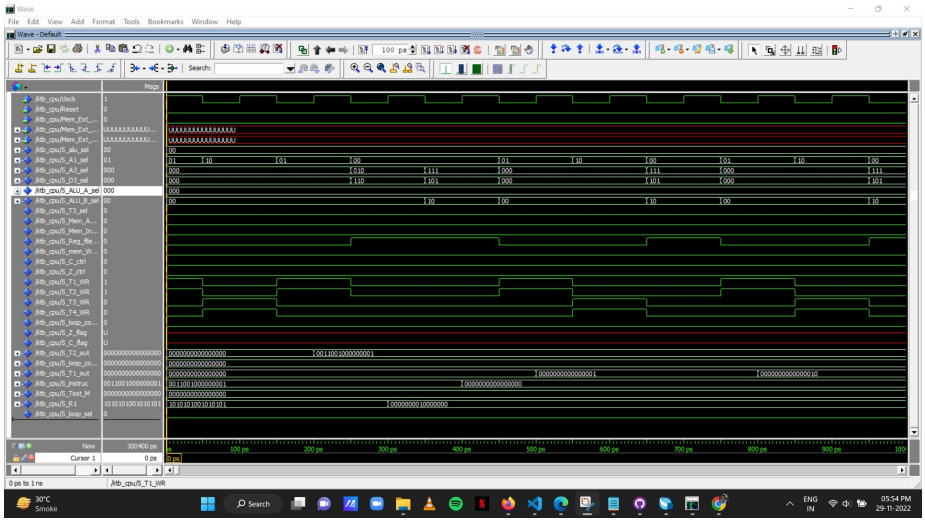

Figure 4: RTL simulation of the LHI instruction.

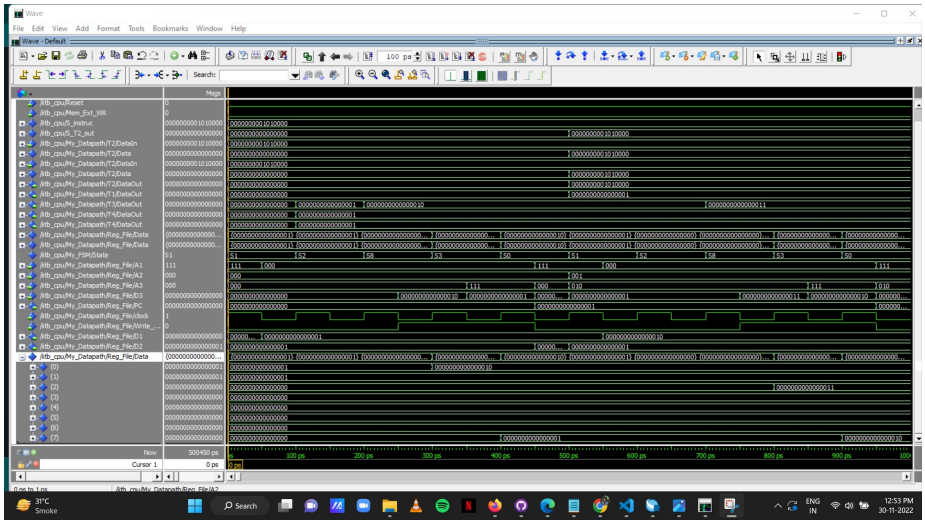

Figure 5: RTL simulation of the ADD instruction.

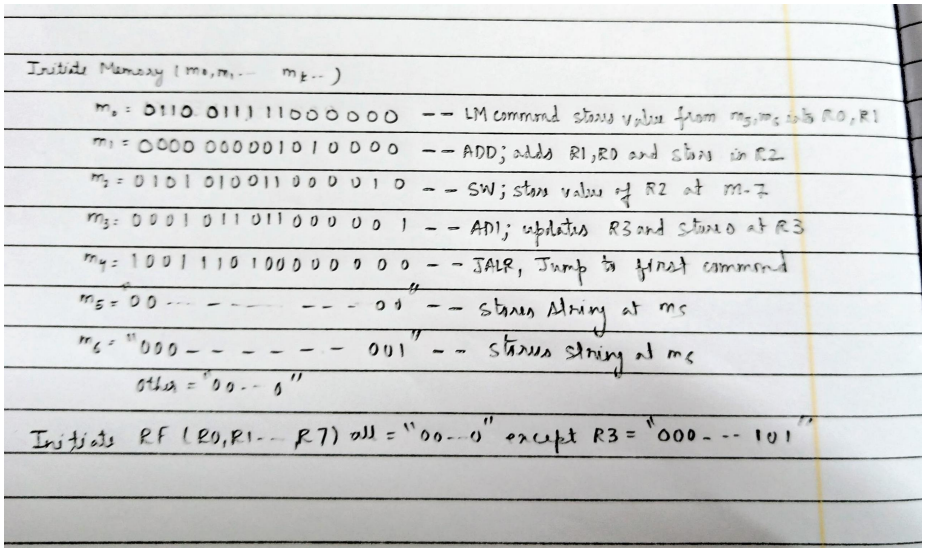

Figure 6: The program in machine code for the Fibonacci Counter

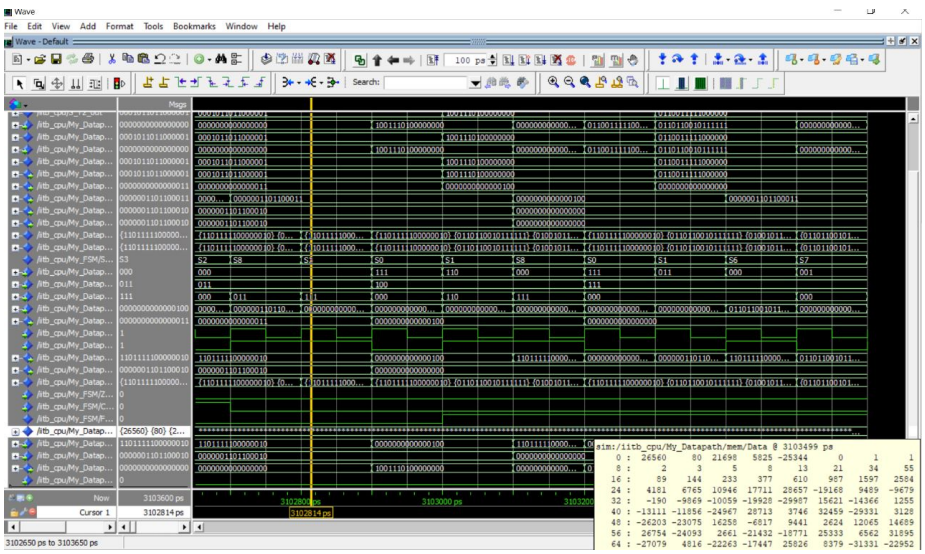

Figure 7: RTL simulation of the the Fibonacci program.