ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION
PROJECT REPORT

PROJECT NO : 2

DUE DATE : 25.05.2023

GROUP NO : 51

GROUP MEMBERS:

150200917 : Erblina Nivokazi (Group Representative)

150220760 : Onur Yavri

150200919 : Abdullah Jafar Mansour Shamout

1 INTRODUCTION

In this project, we designed a hardwired control unit for the first project that we created. There are two types of instructions, address reference related and address reference unrelated, the instructions are 16 in total. The control unit that we created, connects to the inputs of the modules of the first project. Such inputs include selection, function selection, loading, ALU inputs etc. By creating a timing unit, we were able to make the components of the first project function correctly and synchronously.

1.1 Task Distribution

Fetching, decoding, writing the mem file, was done by Abdullah Jafar Mansour Shamout.

The designing of the Executions using Instructions with Address Reference were done by Erblina Nivokazi.

The designing of the Executions using Instructions without an Address Reference were done by Onur Yavri.

And the throughout the project debugging was done as whole group we also discussed each others parts and worked as a team on each part even though it was mainly single person focused.

2 PROJECT PARTS

2.1 General layout

For starters, before starting the project we created a flow chart diagram discussing how each operation will execute in regard to time and the decoded OPcode. The flow chart can be seen below.

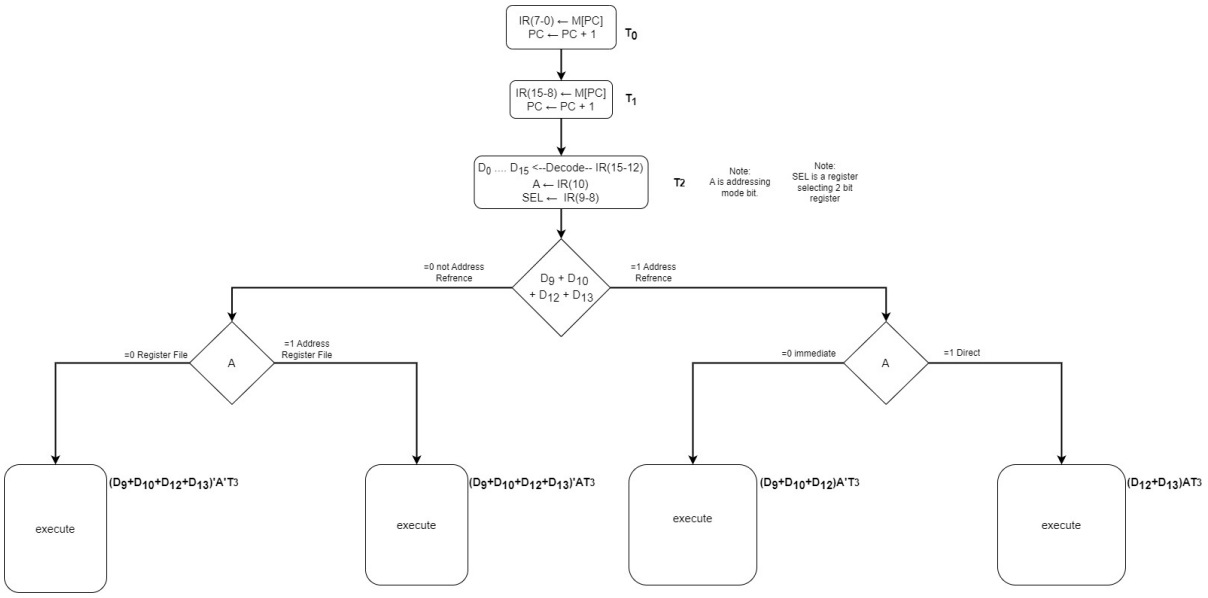


Figure 1: flow chart for the operations

Also to have our code function properly we need all the registers to start from 0 so we reset everything internally before doing anything.

2.2 Timing unit

Regarding how to make all of our operation execute synchronously and at the correct time, we need a time control unit. That is why we created a sequence counter that dictates when and what should function and how it should. However, to have different lines for each of the times we need to decode that value into separate lines that is why we used a decoder connected to it. Their schematics and simulations are as follows:

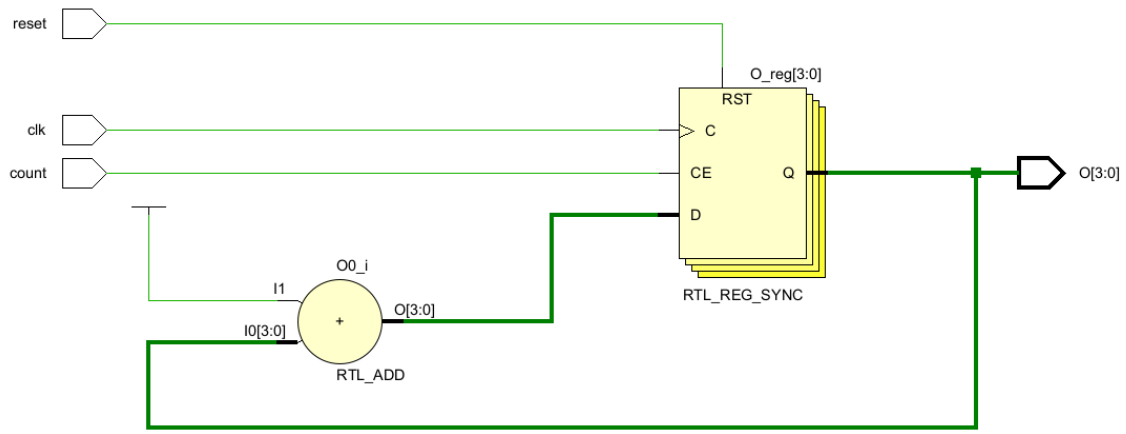


Figure 2: schematic for SC

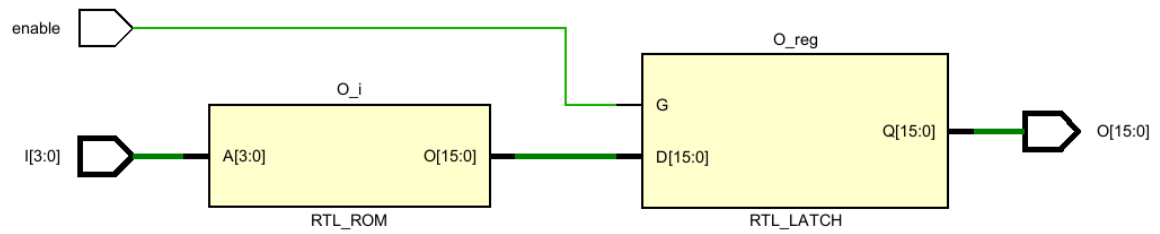


Figure 3: schematic for decoder used for SC

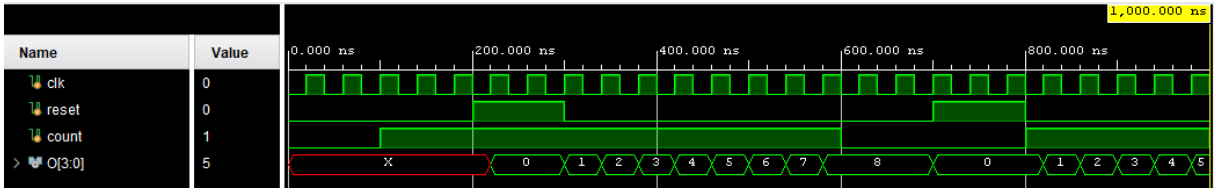


Figure 4: simulation for SC

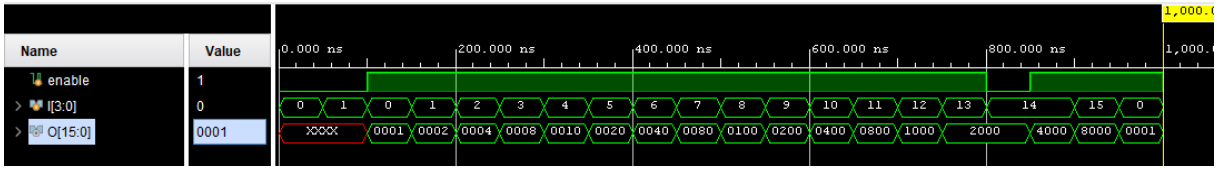


Figure 5: simulation for decoder used for SC

By connecting both of these components we were able to create T[15:0] which we used as a condition for our executions internally. The sequence counter is reset every time a function finishes executing to make the fetch and decode steps occur again.

3 Fetching and decoding

Regarding fetching, since an instruction is 16 bits, however the memory can only save 8 bits at a time "a word is 8 bits", and the order is in little endian. Fetching an instruction from memory will take two cycles. We first fetch the Low part then the High part. Also at each time we increment the PC so that we get the information stored in the next place. Since all of the registers are reset at the start we start from position 0. Fetching the low part is T0 dependent while fetching the High part is T1 dependent. Regarding the controls they control, they both effect the modules in the same way, the only difference is whether the word taken from memory is saved in the high part or low part. The test simulations that I wrote can be seen below

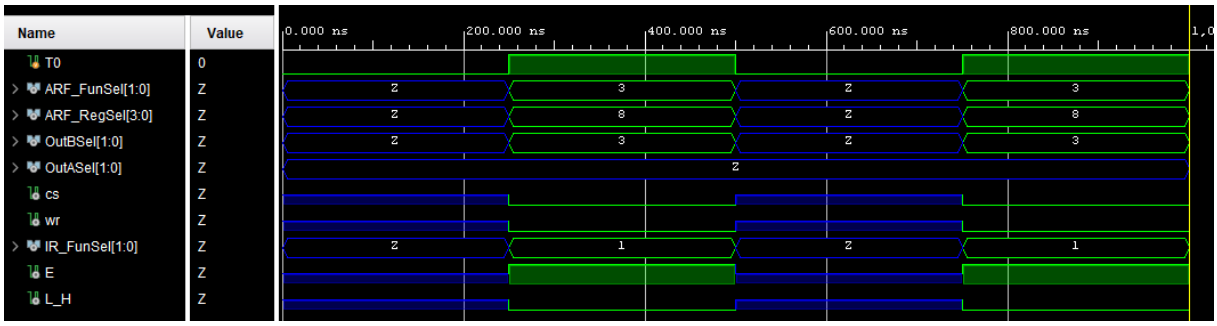


Figure 6: Simulation for fetch low

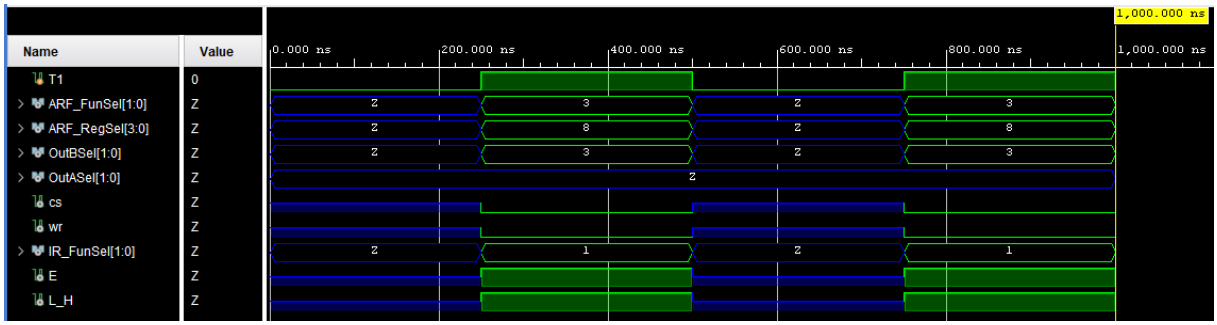


Figure 7: Simulation for fetch high

regarding their schematics, they are as follows:

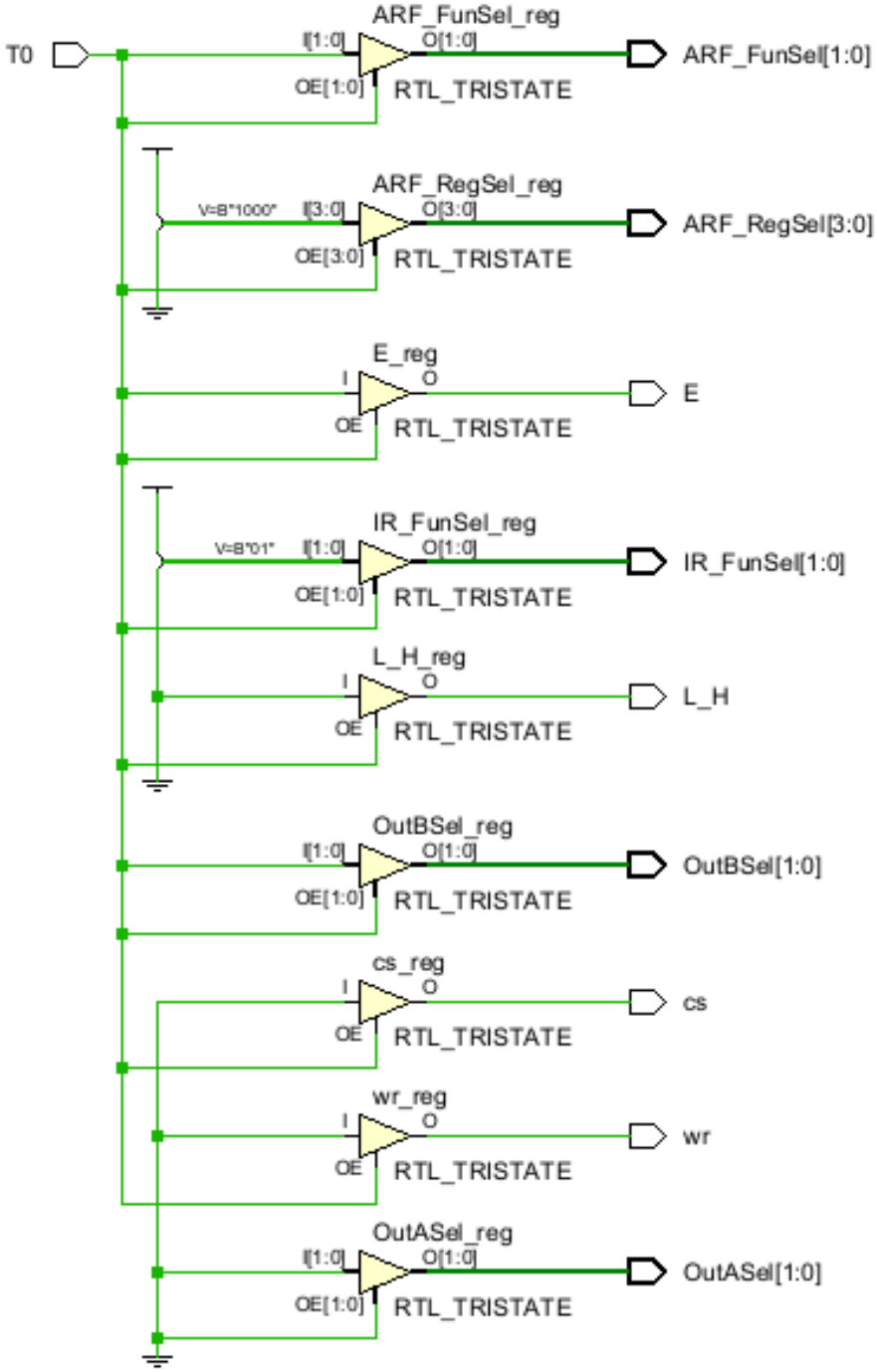


Figure 8: schematic for fetch low

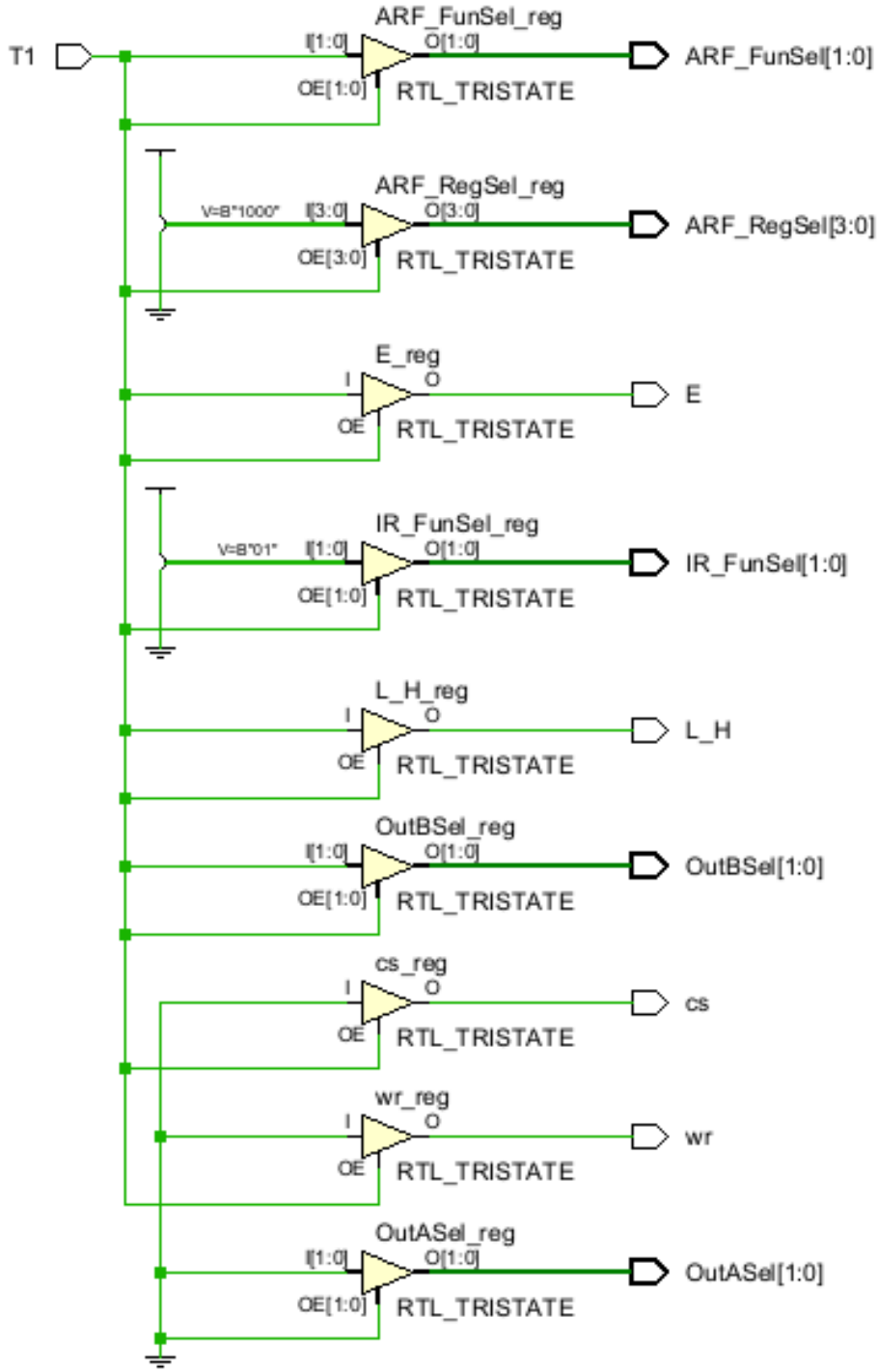


Figure 9: schematic for fetch high

as it can be seen in the simulations and as it can be understood from the schematics; it can be seen that there is a lot of high impedance cases. This is due to the fact that the module is not supposed to work unless it is during the correct timing given to it by the sequence counter. We used high impedance mode as the system might use that line for other executions at the same time. So it can be thought of allowing others to also effect that control part if they need to.

4 The Design of the Executions

In our design we have implemented if conditions to determine the operations getting called from the ALU System. As SREG1 was the most common used register type, we chose SREG1 to be always the A input of ALU in order to make things easier. Because A input of ALU had connections to both RF and ARF, it was easier to take the data inside the registers of register files into the ALU. We divided the operations into groups so that we could work on them easily. Group list includes operations with 2 operands, operations with 1 operand, increment/decrement. There are few cases we have discovered throughout the required actions to get the correct registers to correct places. There are 6 possible cases for operations with 2 operands (Actually 8 cases but getting both SREG1 and SREG2 from ARF is not the case for this design, according to what our Teaching Assistant has told us.); DSTREG can be from ARF or RF for an operation, and same goes for SREG1 and SREG2. As ALU determines the outputs of the operations, there's no difference between using any of the operations with 2 operands. ALU_FunSel changes according to the operation type in the ALU. Other than that, depending on the source registers and destination register, correct multiplexer selections, register activation selections and register output selections, and register function selections happens. Since there are a lot of cases, it is not added to this report to make it less verbose. If further information is needed, the code can be examined.
For any of the operations with 2 operands (0x00, 0x01, 0x03, 0x04), following RTL can be used:

$$D0\,T3 : ALU\_A \leftarrow SREG1(RF), ALU\_B \leftarrow SREG2(RF), DSTREG(RF) \leftarrow ALU\_Out$$

$$D0\,T3 : ALU\_A \leftarrow SREG1(ARF), RF\_T1 \leftarrow SREG2(RF), DSTREG(RF) \leftarrow ALU\_Out$$

$$D0\,T3 : ALU\_A \leftarrow SREG1(RF), ALU\_B \leftarrow SREG2(ARF), DSTREG(RF) \leftarrow ALU\_Out$$

$$D0\,T3 : ALU\_A \leftarrow SREG1(RF), ALU\_B \leftarrow SREG2(RF), DSTREG(ARF) \leftarrow ALU\_Out$$

$$D0\,T3 : ALU\_A \leftarrow SREG1(ARF), ALU\_B \leftarrow SREG2(RF), DSTREG(ARF) \leftarrow ALU\_Out$$

$$D0\,T3 : ALU\_A \leftarrow SREG1(RF), RF\_T1 \leftarrow SREG2(ARF), DSTREG(ARF) \leftarrow ALU\_Out$$

And the sequence counter reset happens for all of the operations at the same sequence.

$$D0\,T4 : SC \leftarrow 0$$

There are 4 different cases for operations with 1 operand and others are individuals. The operations with 1 operand are easy as the only thing we need to do is moving the register to the correct path using ALU. For any of the operations with 1 operand (0x02, 0x05, 0x06 and 0x0B), following RTL can be used:

$$D2\,T3 : ALU\_A \leftarrow SREG1(RF),\ DSTREG(RF) \leftarrow ALU\_OUT$$

$$D2\,T3 : ALU\_A \leftarrow SREG1(ARF),\ DSTREG(RF) \leftarrow ALU\_OUT$$

$$D2\,T3 : ALU\_A \leftarrow SREG1(RF),\ DSTREG(ARF) \leftarrow ALU\_OUT$$

$$D2\,T3 : ALU\_A \leftarrow SREG1(ARF),\ DSTREG(ARF) \leftarrow ALU\_OUT$$

And the sequence counter reset happens for all of the operations at the same sequence.

$$D2\,T4 : SC \leftarrow 0$$

In increment and decrement operations, we transfered the SREG1 to DSTREG and then since we didn't have any increment operation, we used one more cycle to increment the data inside the DSTREG. Storing 1 in a temporary register in RF was another solution but it was taking more clock cycles and since this was efficient, we chose this way. For increment (0x07) and decrement (0x08) operations, following RTL can be used:

$$D7\,T3 : ALU\_A \leftarrow SREG1(RF),\ DSTREG(RF) \leftarrow ALU\_OUT$$
$$D7\,T4 : DSTREG(RF) \leftarrow DSTREG(RF) + 1$$

$$D7\,T3 : ALU\_A \leftarrow SREG1(RF),\ DSTREG(ARF) \leftarrow ALU\_OUT$$
$$D7\,T4 : DSTREG(ARF) \leftarrow DSTREG(ARF) + 1$$

$$D7\,T3 : ALU\_A \leftarrow SREG1(ARF),\ DSTREG(RF) \leftarrow ALU\_OUT$$
$$D7\,T4 : DSTREG(RF) \leftarrow DSTREG(RF) + 1$$

$$D7\,T3 : ALU\_A \leftarrow SREG1(ARF),\ DSTREG(ARF) \leftarrow ALU\_OUT$$
$$D7\,T4 : DSTREG(ARF) \leftarrow DSTREG(ARF) + 1$$

And the sequence counter reset happens for all of the operations at the same sequence.

$$D7\,T5 : SC \leftarrow 0$$

For the executions that have an instruction with an address reference we used a slightly different approach. If statements were implemented for each input of the ALU System, the conditions check which bit of our decoded instruction register is enabled, it checks the

time sequence and for the instructions with address reference it also checks the addressing mode, if it is immediate or direct. For the execution of the BRA operation, or the 0x09 OPCODE operation, the value stored in the instruction register or the bits [7-0] of the register will be stored in the PC register. This operation takes one cycle to finish, and in the one cycle the inputs of the ALU System that need to be changed are MuxASel, which takes the value 01 so that the input of the Address Register File is the value of the instruction register, ARF_FunSel changes to 01 to enable load, and ARF_RegSel is set to 1000 to enable the PC register. BRA:

$$D9\,T3 : PC \leftarrow VALUE$$

$$D9\,T4 : SC \leftarrow 0$$

For the execution of the BNE operation, or the 0x0A OPCODE operation, everything is similar with the BRA operation except we have a condition for Z to be 0 for the execution to happen.

$$D10\,T3\,\bar{Z} : PC \leftarrow VALUE$$

$$D10\,T4 : SC \leftarrow 0$$

For the execution of the LD operation (OPCODE 0x0C), it is required to store a value on a selected register in RF, depending on the bit of the addressing mode this value could be the first 8 bits of the instruction register (Addressing Mode is equal to 0) or a value in memory, M[AR] (Addressing Mode is equal to 1). If the Addressing Mode is immediate (equal to 0) the operation will take one cycle and the inputs that will change are RF_Rsel (gets set to a register specified by the given 2 bit RSEL), MuxCSel (gets set to 10), and RF_FunSel (gets set to 01). If the Addressing Mode is 1 (direct) then the value being stored in the Rx is coming from the memory. This execution will take 1 cycle in which we read from the memory and store in RF.

$$D12\,T3\,IM : Rx \leftarrow VALUE\ Rx$$

$$D12\,T3\,D : Rx \leftarrow M[AR]$$

$$D12\,T4 : SC \leftarrow 0$$

For the execution of the ST operation (OPCODE 0x0D), it is required to store whatever value we have in a chosen RF register to memory in the address held by AR. This operation is done in one cycle and the inputs required in the ALU system are: RF_OutASel (set to 1 concatenated with the bits of RSEL in the instruction register), MuxCSel (set to 0), ALU_FunSel (set to 0000), Mem_WR (set to 1), Mem_CS (set to 0), and OutBSel (set to 00).

$$D13\,T3\,D : M[AR] \leftarrow Rx$$

$$D13\,T4\,:\;SC \leftarrow 0$$

In the execution of PUL operation (OPCODE 0X0E), it is required to increment the SP register and load the memory address of SP to a chosen RF R register. This operation is done in two cycles. In the first cycle, the SP register is incremented, to do this the inputs ARF_RegSel (0010) and ARF_FunSel (11) are changed. In the second cycle the value in address SP in Memory is saved in a selected RF R register. In this cycle the inputs of the ALU System that are changing are: Mem_CS (set to 0), Mem_WR (set to 0), ARF_OutBSel (set to 01), MuxASel (set to 01), RF_FunSel (set to 01), and RF_Rsel (set to a value selected by RSEL).

$$D14\,T3\,:\;SP \leftarrow SP + 1$$

$$D14\,T4\,:\;Rx \leftarrow M[SP]$$

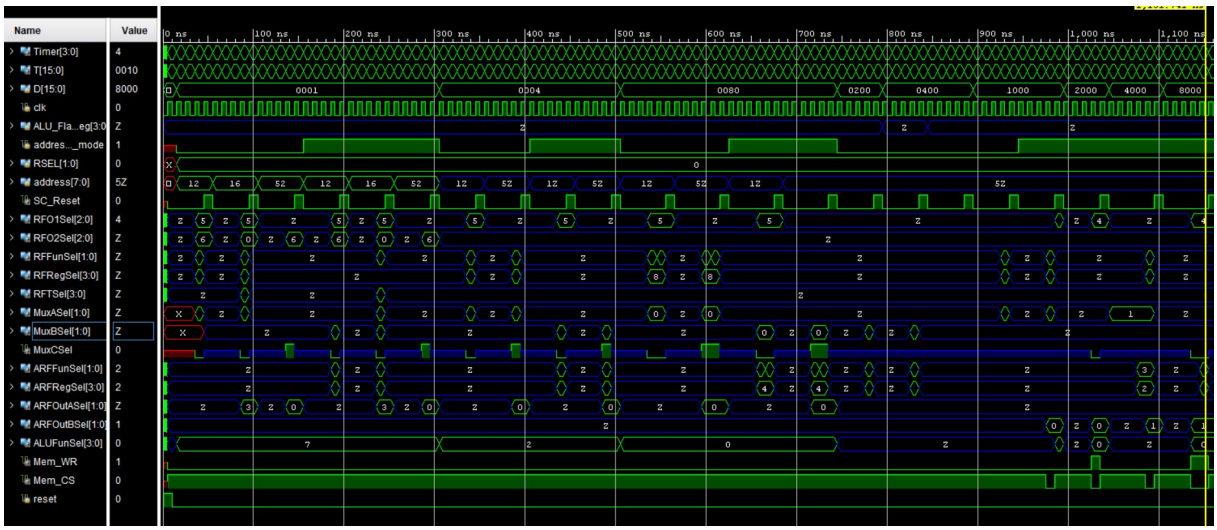
$$D14\,T5\,:\;SC \leftarrow 0$$

In the execution of PSH operation (OPCODE 0X0F), it is required to save a register from a chosen register in RF to a Memory address selected by register SP, afterwards the SP register is decremented. This operation takes two clock cycles. In T3 the registers enabled to finish M[SP]¡-Rx are: Mem_CS (set to 0), Mem_WR (set to 1), ARF_OutBSel (set to 01), RF_OutASel (set to a 3 bit value starting with 1 concatenated with the given RSEL in the instruction code), MuxCSel (set to 0), and ALU_FunSel (set to 0000). In T4 ARF_RegSel is set to 0010 and ARF_FunSel is set to 10, enabling SP to be decremented.

$$D15\,T3\,:\;M[SP] \leftarrow Rx$$

$$D15\,T4\,:\;SP \leftarrow SP - 1$$

$$D15\,T5\,:\;SC \leftarrow 0$$



Simulating all of the Executions

In the test written for given executions, they all work correctly to the given logic.

5 DISCUSSION

In this project, we have implemented a control unit in order to connect it to the ALU system we implemented before and let it run the system. Control unit decides which of the switches will be on and off according to the sequence counter in it and gives the switches as output to ALU system. Then, ALU system takes the outputs as its inputs and behaves according to the decisions of control unit. Control unit has the ability to choose which operation in the system is going to happen depending on the given instruction, which is taken from the memory. After figuring out what to do with operations, it was easier to implement. In first two cycles of sequence counter, fetch occurs. Then, one cycle is reserved for the decoding operation. After fetch and decode cycles, execution of the OPCODE starts and after execution ends, sequence counter is reset in the next cycle. Therefore, the system is able to start to get the next instruction from the memory.

6 CONCLUSION

While this project was very informative and taught us a lot, it was also very challenging and we had some trouble while coding and designing it. At first we designed a project that worked quite differently from the test bench given to us, but when the test bench was added to the class files we saw that we needed to change our entire approach, this was not very easy for us to do since the project is quite big and a lot of stuff needed to be handled. Our first approach worked well, in the first part the fetching and decoding was done in different modules, the executions were put together in one module and at last we put them all together in the control unit, doing this meant that we had multiple inputs and outputs for each module, after the launching of test bench we tried to change our approach and rearrange everything accordingly. Apart from the short time and the rearranging of the modules, this project was very interesting to work on.