ESE 345 Aaron Lin, Hang Chen

Project Report : Design Analysis

The goal of this project is to build a four-stage pipelined multimedia unit using VHDL. The stages of the pipeline are : Instruction Fetch, Instruction Decode, Execute, and Write Back. Each stage may consist of multiple smaller components. This report will discuss each stage in sections with diagrams and analysis. The first part will take a look at the block diagrams of each component of the pipeline and see how they're all connected. Analysis of the waveforms generated by each of these components will be examined next.

Instruction Fetch (IF) :

The Instruction Fetch stage consists of the Program Counter, Program Counter adder, and the Instruction Buffer. The Program Counter is a register that provides the address of the instruction to be fetched to the Instruction Buffer and the Program Counter adder. The Program Counter provides the address on each rising clock edge. The Program Counter adder takes the address and increments it by one, and passes it back to the Program Counter to use on the next rising clock edge. The Instruction Buffer takes the address and finds the instruction in its register that corresponds to that address, and outputs that instruction. The Instruction_Data and Instruction_Index is connected to the Instruction Buffer, and used to load the program at the beginning. The instruction gets passed to the IF/ID register.

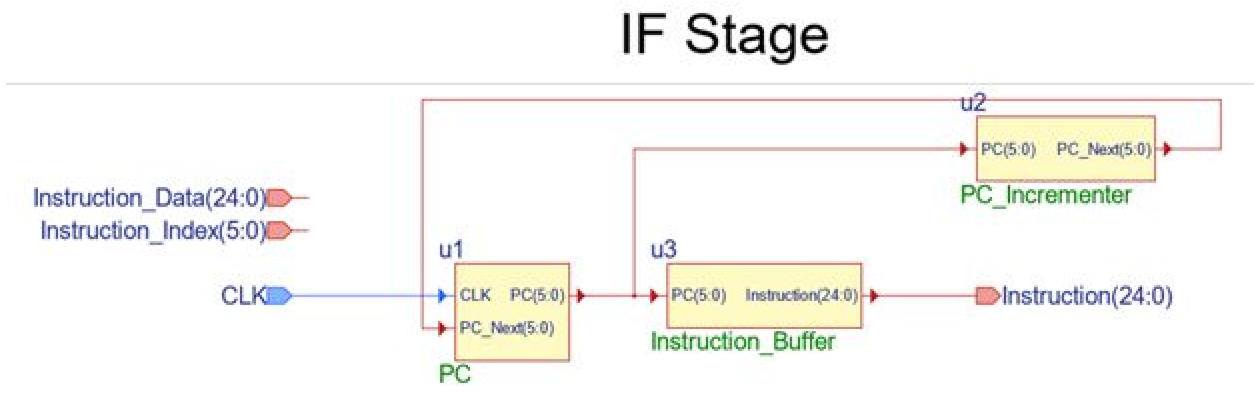


Figure 1 : Block Diagram for the IF Stage

<u>IF/ID Register</u> :

The IF/ID Register sits between the IF stage and the ID stage. It is simply a register that stores the value it is passed by the prior stage. In this case, the IF stage will be passing the instruction to this register. This register will simply be passing along what it is passed to the next stage, ID stage.

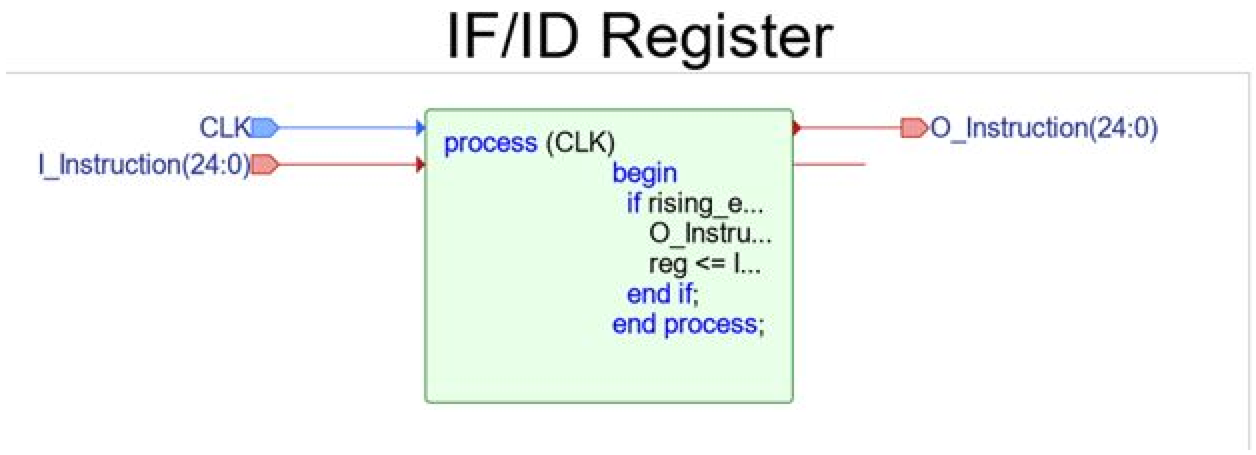


Figure 2 : Block Diagram for the IF/ID Register

<u>Instruction Decode</u> (ID) :

The Instruction Decode stage consists of the Register File, Control Unit, and a single 2 input multiplexor. The Control Unit takes the instruction and based on the msb 2 bits, determines if the instruction is a load, R4, or R3 instruction. If the instruction is determined to be a load instruction, then the write enable signal and multiplexor selector signal is asserted. The write enable signal implies that a register is to be written during the Write Back stage. The multiplexor selector signal is to determine if the register to read from in the register file should be rd or rs1. In this case, it is rd. If the instruction is determined to be an R4 instruction, only the write enable signal is asserted. If the instruction is determined to be an R3 instruction, then depending on the opcode of the instruction, the write enable signal will be asserted or cleared. Specifically, if the instruction is determined to be BCW, CLZ, ABSDB, MPYU, MSGN, POPCNTW, ROT, ROTW, or SHLHI, then the write enable signal is asserted. The Register File takes the instruction and the multiplexor output to determine which register to read from. It reads up to 3 registers at the same time, and writes to one if the write signal is asserted. If writing to a register is asserted, then the data to write and the register to write to comes from the Write Back stage.

The Write Enable signal, register to write to, register numbers being read from, the ALU opcode, and the data read from the 3 registers are passed to the ID/EX register.

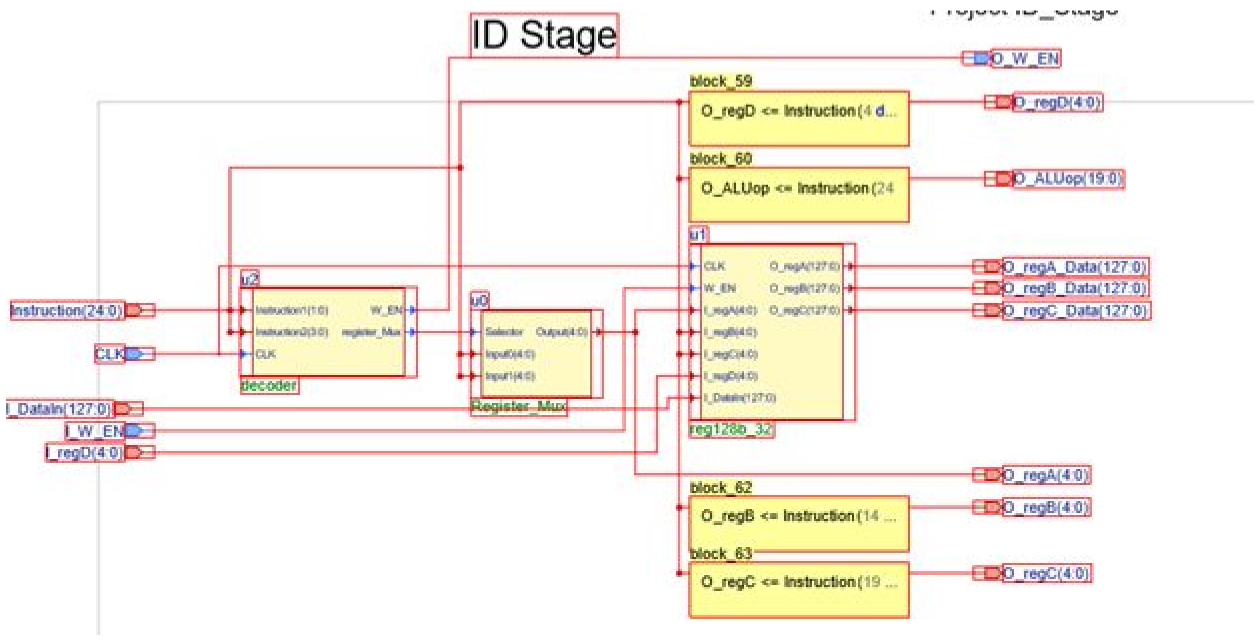


Figure 3 : Block Diagram for the ID stage

<u>ID/EX Register</u> :

The ID/EX Register sits between the ID stage and the EX stage. The ID stage will be passing to this register the Write Enable signal, register number to write to, register numbers being read from, ALU opcode, and the data from the 3 registers read from. This register will simply be passing along those information onto the EX stage. The signals Write Enable, and register to write to do not actually get used in the EX stage, rather they are passed straight to the EX/WB register. Likewise, the register numbers being read from do not get used in the EX stage either, but are passed straight to the Forwarding Unit.

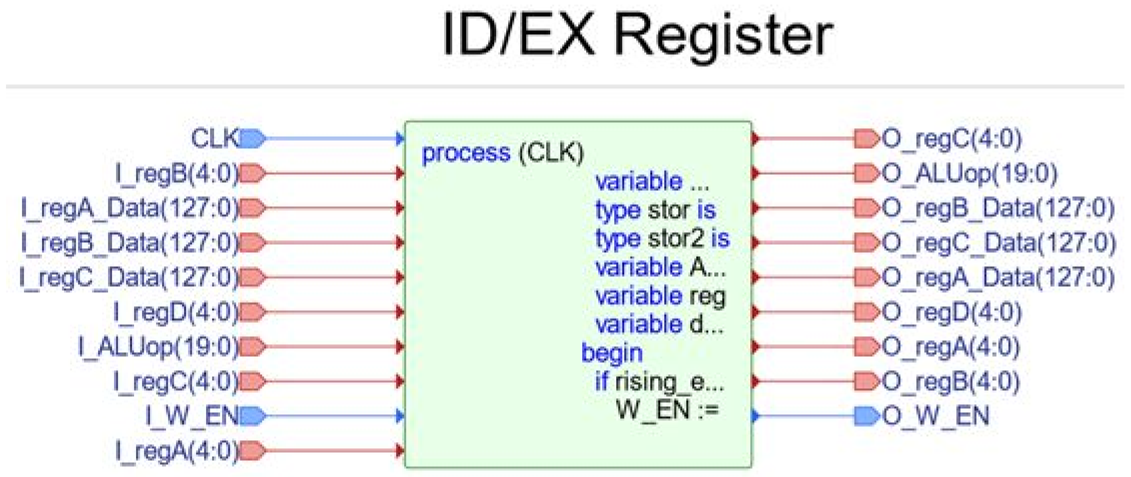


Figure 4 : Block Diagram for the ID/EX Register

<u>Execute</u> (EX) :

The Execute stage consists of the Forwarding Multiplexores and the ALU. The Forwarding Multiplexors determine whether the input values to the ALU should be the value read from the register file on the previous cycle, or the value calculated already and in the Write Back stage. The register numbers of each of the inputs to ALU is passed to the Forwarding Unit. The multiplexors' selector bits are determined by the Forwarding Unit which will be discussed later. The ALU executes a total of 29 different instructions based on the ALU opcode. The ALU calculated result is passed to the EX/WB register.

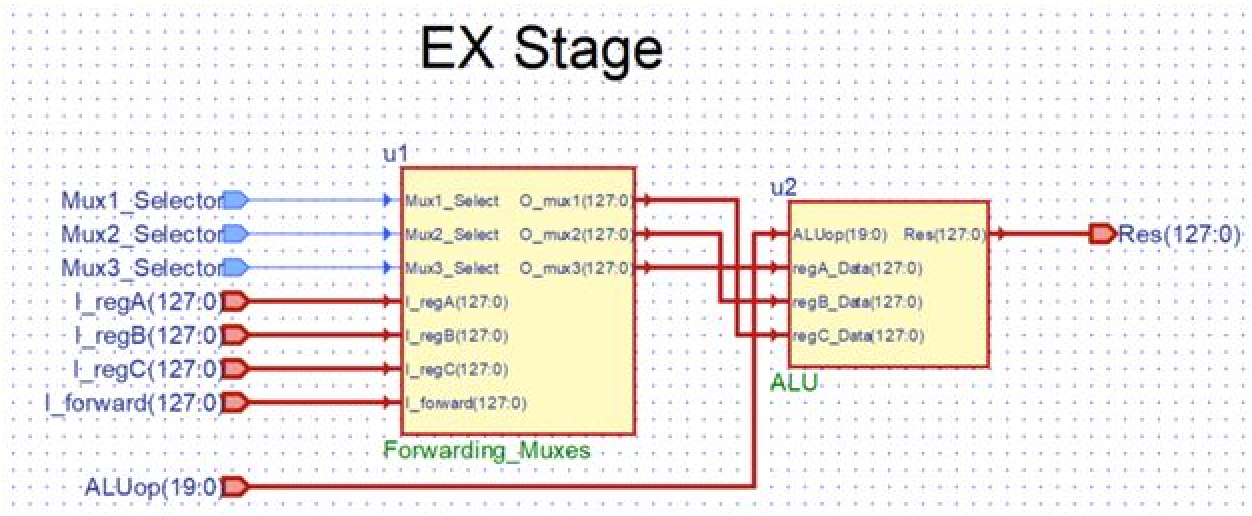


Figure 5: Block Diagram for the EX stage

EX/WB Register :

The EX/WB Register sits between the EX stage and the WB stage. The EX stage will be passing to this register the Write Enable signal, register number to write to, and the ALU calculated result. These signals are then passed onto the WB stage to be used.

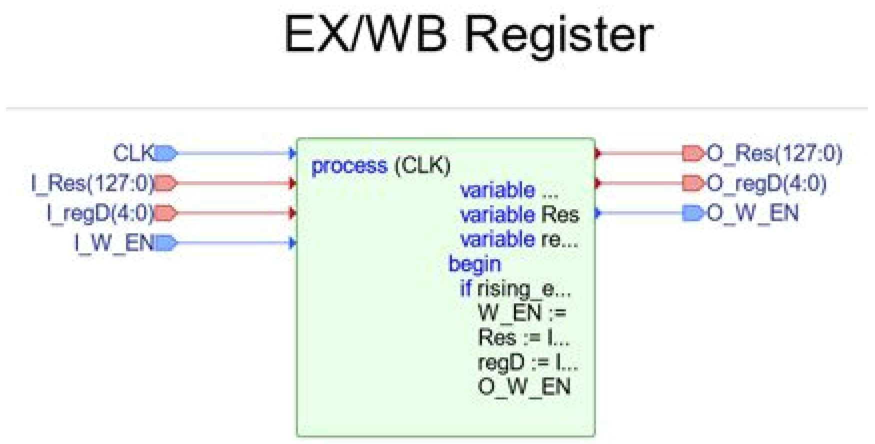


Figure 6 : Block Diagram for the EX/WB Register

Forwarding Unit :

The Forwarding Unit determines whether the data from the Write Back stage should be forwarded to the inputs of the ALU in the previous stage. It does this by first checking if the Write Back stage has a write enable signal asserted. If it doesn't, then no forwarding is need, and the forwarding multiplexor selector bits can be set to be 0. Otherwise, the register to be written to in the Write Back stage is compared to the registers that were read and whose data is to be used in the ALU inputs. If the register to be written to is the same register as any of the ALU inputs, then the corresponding ALU input forwarding multiplexor's selector bit is asserted so that the data from the Write Back stage can be used in the ALU input. If none of the registers match, then no forwarding would be needed.

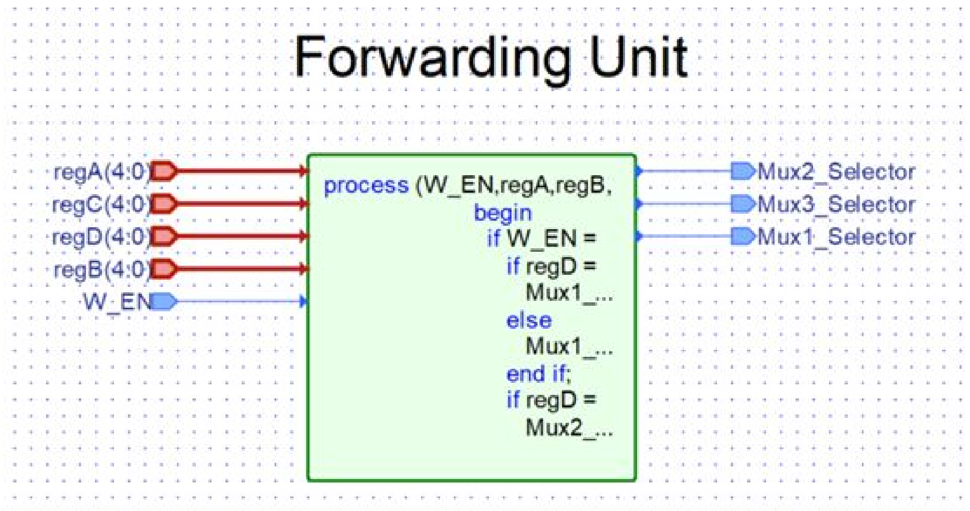


Figure 7 : Block Diagram for the Forwarding Unit

<u>**Top Level Design**</u> :

There is no actual special component in the Write Back stage that hasn't already been used in the previous stages, so the connection of the Write Back stage will be shown through the top level design of the pipeline. As can be seen, the Write Enable signal and the register to be written to are passed to the Forwarding Unit to use for determination of forwarding data or not. The data calculated from the ALU in the previous cycle and the register to be written to are also passed to ID stage, where the register file will accept them.

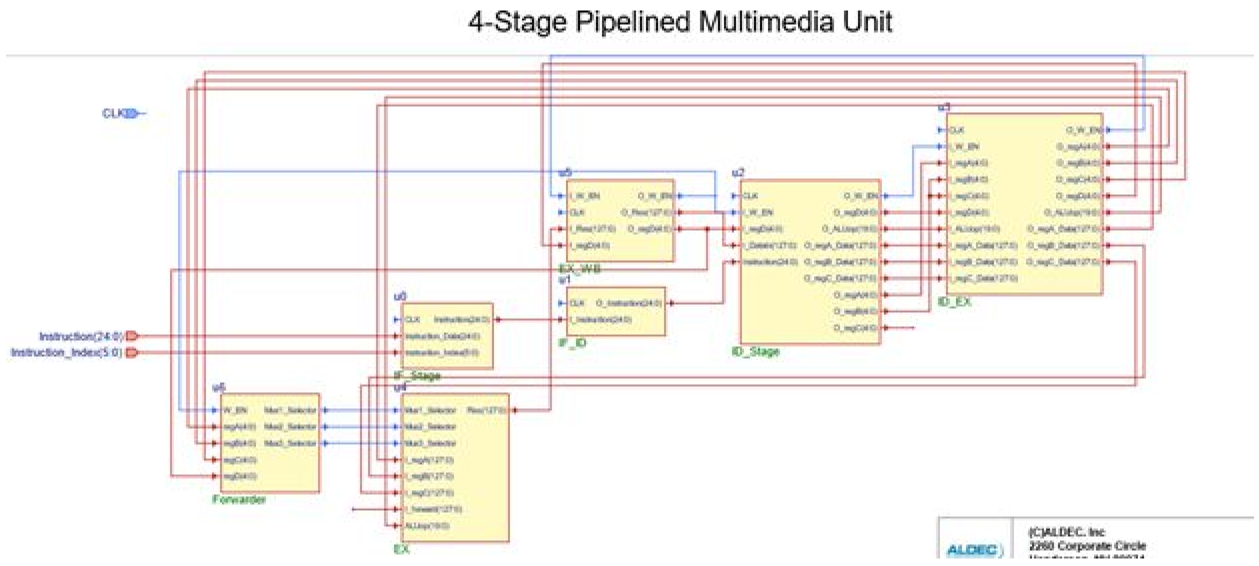
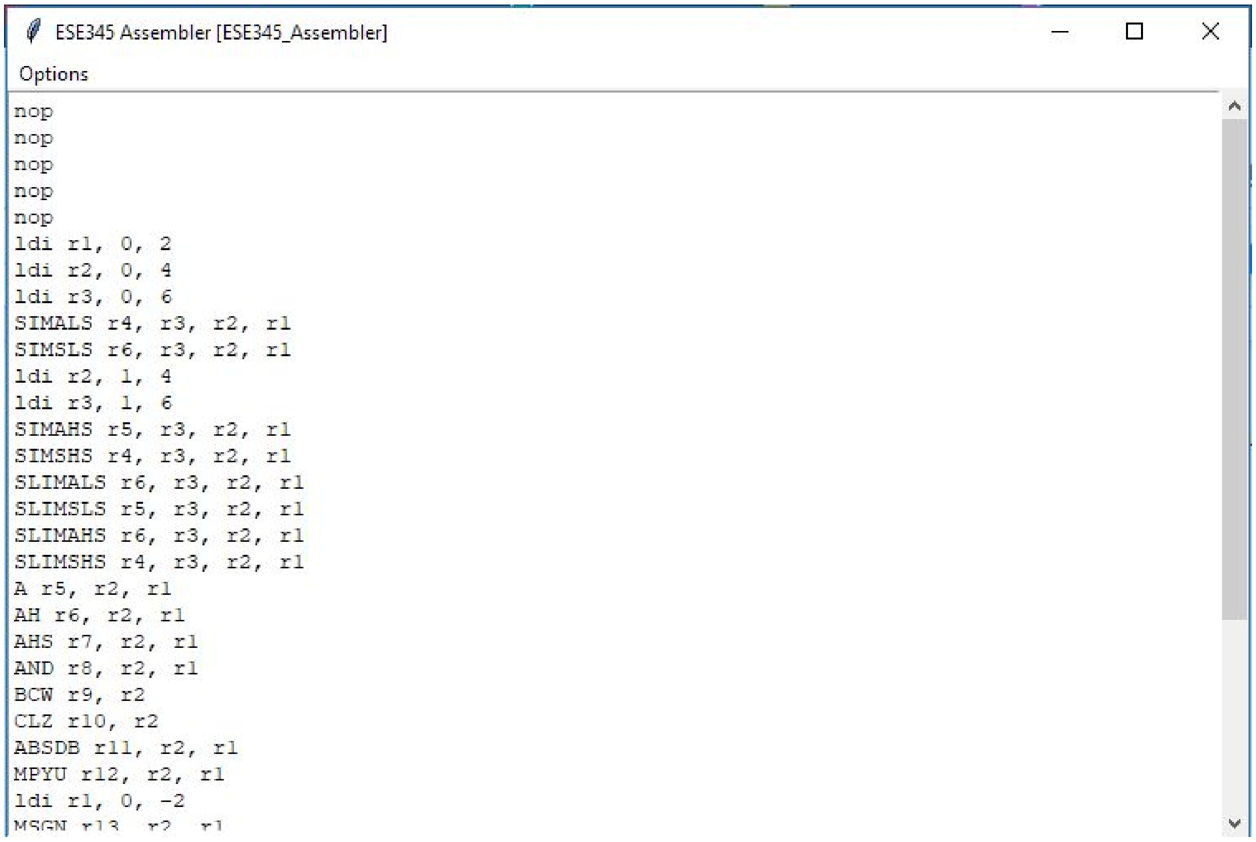


Figure 8 : Top level block diagram for the 4-Stage Pipelined Multimedia Unit

Assembler:

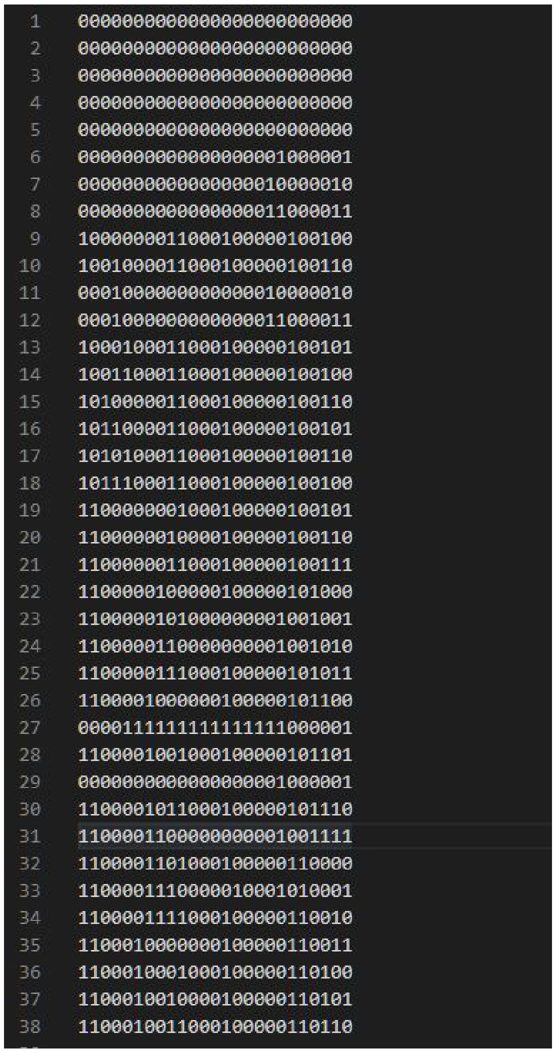
Assembler (written in Python) is responsible for converting the instructions into a binary format text file with filename as “ESE345 Assembler.txt”. This file can be read into the instruction buffer in the Instruction Fetch stage. All the don’t care(X) values and invalid instructions would be converted into 0 in binary.

The assembler we have follows a few rules about spacing and each register field needs to start with character ‘r’. For example, load instructions require syntax: *ldi rd, index, imm* . Notice the spacing between each element and comma. The assembler would not work if user type syntax such as: *ldi rd,index,imm.* The user can type the instructions in the text area and click the option menu to compile as shown in figure 9.



```
nop
nop
nop
nop
nop
ldi r1, 0, 2
ldi r2, 0, 4
ldi r3, 0, 6
SIMALS r4, r3, r2, r1
SIMSLS r6, r3, r2, r1
ldi r2, 1, 4
ldi r3, 1, 6
SIMAHS r5, r3, r2, r1
SIMSHS r4, r3, r2, r1
SLIMALS r6, r3, r2, r1
SLIMSLS r5, r3, r2, r1
SLIMAHS r6, r3, r2, r1
SLIMSHS r4, r3, r2, r1
A r5, r2, r1
AH r6, r2, r1
AHS r7, r2, r1
AND r8, r2, r1
BCW r9, r2
CLZ r10, r2
ABSDB r11, r2, r1
MPYU r12, r2, r1
ldi r1, 0, -2
MSGN r13, r2, r1
```

Figure 9: User Interface of Assembler with test instructions typed



```
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000001000001
000000000000000010000010
000000000000000011000011
100000001100010000100100
100100001100010000100110
000100000000000010000010
000100000000000011000011
100010001100010000100101
100110001100010000100100
101000001100010000100110
101100001100010000100101
101010001100010000100110
101110001100010000100100
110000000100010000100101
110000001000010000100110
110000001100010000100111
110000010000010000101000
110000010100000001001001
110000011000000001001010
110000011100010000101011
110000100000010000101100
000011111111111111000001
110000100100010000101101
000000000000000001000001
110000101100010000101110
110000110000000001001111
110000110100010000110000
110000111000001000101000 1
110000111100010000110010
110001000000010000110011
110001000100010000110100
110001001000010000110101
110001001100010000110110
```

Figure 10: Binary format of the test instructions