ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO : 3

DUE DATE : 03.06.2020

GROUP NO : G12

GROUP MEMBERS:

150180058 : Faruk Orak

150190739 : Yasin Abdülkadir Yokuş

150190719 : Oben Özgür

150190708 : Ramazan Yetişmiş

SPRING 2020

Contents

1 INTRODUCTION

In this project, we have implemented a basic computer by using parts of previous projects. In addition to Project 2 circuit, we have designed a hardwired control unit and combined each other. Control unit is designed according to the given instructions. The computer reads instructions from memory and does operations according to these instructions. Therefore, instructions should be written to memory. If it is necessary to explain how control unit works basically, it sets inputs of circuit's components according to the instruction. For example, to load result of ALU to PC, inputs of circuit should be like MuxBSEL = 11, RegSelC = 001, FunSelC = 01. Control unit sets these inputs when result of ALU should be loaded to PC according to the instruction. All instructions will be explained in detail following part of report.

2 PROJECT PARTS

2.1 Control Unit Structure

2.1.1 T Control Bits

Firstly we implemented T inputs.Our control unit consists of 5 T cycles which begin with T0 and up to T4. To perform this operation we used one 3-bit counter and one decoder(3x8). The logic is that we give the output of the counter into the decoder as input and finally the output of the decoder gives us the T control-bits[T0-¿T4].

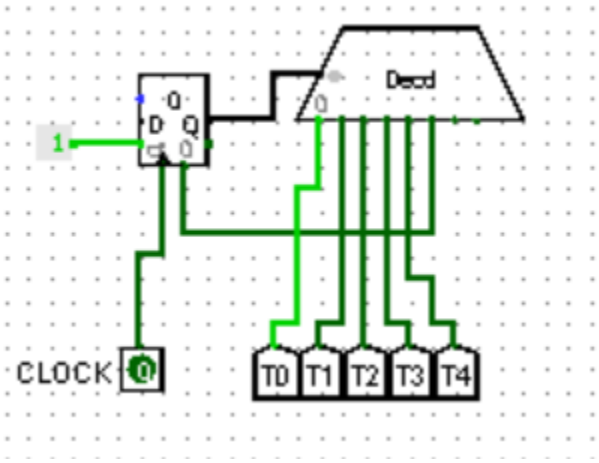

Figure 1: The T signal Circuit

2.1.2 D Control Bits

To create these D control-bits we took the most significant five-bits of the 16-bit instruction. After that, we used one decoder(5x32). So we named the output of the decoder from D0 to D18, each of these D signals corresponds to an operation in our operation table.

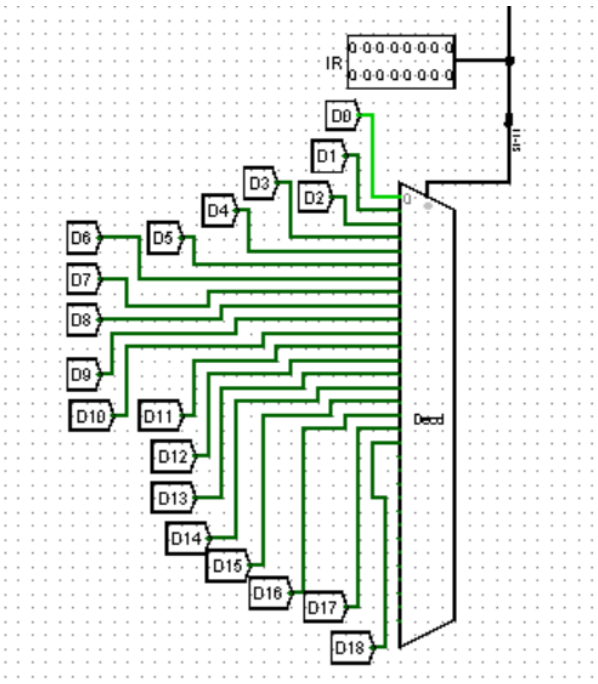

Figure 2: D control-input generator circuit

2.1.3 Register Decomposition

To find the corresponding Destination, Source1 and Source2 registers we simply decomposed the instructions corresponding-bits such that for destination part we took [8,9,10] bits and gave as input to a decoder(3x8). Then we grouped them as Register A and Register B such that A is for the Register that is R0 to R3. And for the B part that is PC, Adress, and Stack Registers. This grouping operation is simple we just used two OR gates. And for each register part (Destination, Source1, and Souce2) we used this grouping strategy iteratively. The only difference between them is the bits which we put the decoder as input. For Source1 we used [5,6,7] bits and finally for Source2 we used [2,3,4] bits. The first and second bits are do not count as anything.

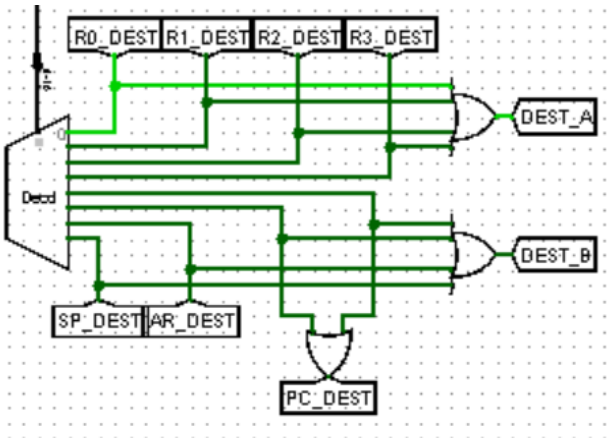

Figure 3: Decomposition of Destination Register without address reference

The upper Register decomposition part is for the instructions without address reference. Now the below part is with addressing mode. For this Regsel decomposition, we are only using The Part A register(R0, R1, R2, R3) so we took the instruction 9th and 10th bits and gave the decoder(2x4) as input and we named the outputs different from the instruction without addressing mode. Because each of the inputs is used as control signals to differ the operation whether it is with or without addressing mode.

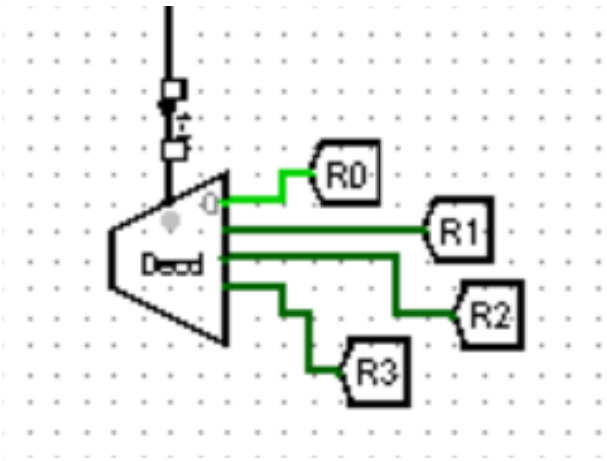

Figure 4: Register selection with an address reference

For the address reference part, we need 2 control-bits such that Direct/Immediate. Simply took the 8th bit of the 16-bits instruction and put it to the decoder if it is zero the IM signal is one otherwise Direct signal is one.

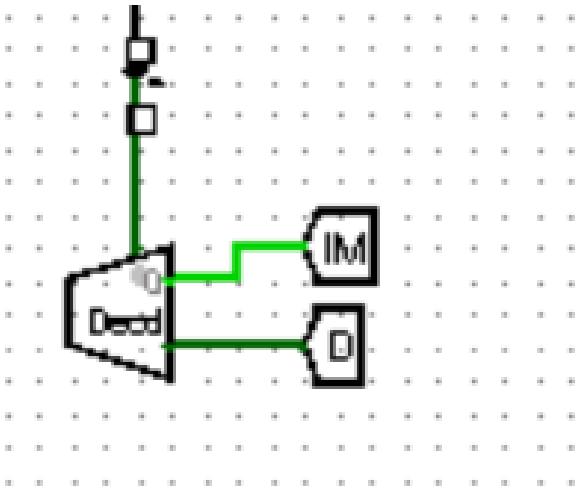

Figure 5: D/IM input bit generator circuit

Now after decomposition, we created some control inputs to simplify the control circuit.

2.1.4 D5 - D13 Control Bit

When we looked at the operation table we saw that N/A operations begin D5 and end D13 so we took all these D5-D13 input signals and put them into the OR gate, by this way we get rid of the redundant gate usage.

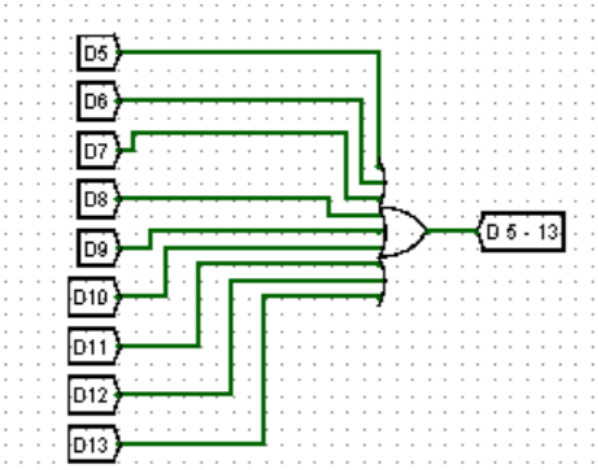

Figure 6: The D5-13 control input circuit

2.1.5 D5-13-6 Control Bit

This control input specifies that there is no D6 instruction. Because D6 operation corresponds to a substruction operation and in substruction operation is different because as first operand we use the Sourcereg2 and for the second operand we used SourceReg1(SRCREG2 - SRCREG1).But in ALU this operation is done the opposite(SRCREG1 – SRCREG2) way so we have to change the operands. This bit shows us when D6 operation occurs.

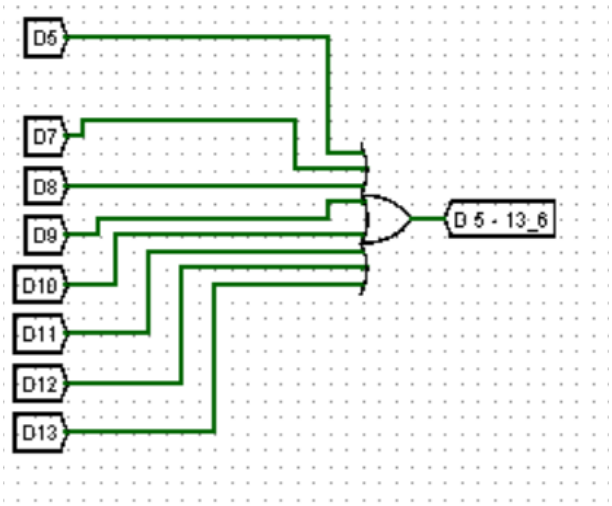

Figure 7: D5-13-6 control-bit generator circuit

2.1.6 Branch Control Bit

We realized that for the branch operation we can generate a simplification because they all make the operation in common T time input, IM control-bit, and Z control flag. The circuit simply works like this, if D14(BRA) occurs it writes pc to value no matter what. If D14 and D15 occurs it has to check the Z flag. So we came up with the below circuit which performs this operation.

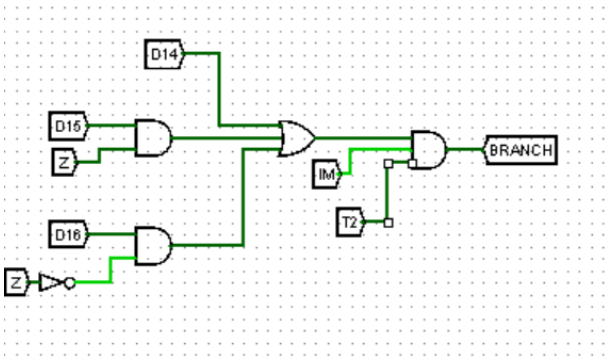

Figure 8: Branch Control-bit circuit

The last part of our control circuit is to take these inputs and generate the corresponding output bits for each part of the main circuit. All of these outputs are going to be explained later parts but for the main logic and visualization, now we are going to mention simply. To generate output we used an encoder, the logic is behind that is simple, now consider our 2-bits FUNSELC, to generate this control signal we used an encoder(1x2) if it is a load operation at the corresponding T time signal the 2nd gate of the encoder activated and as output 01 bits are given.

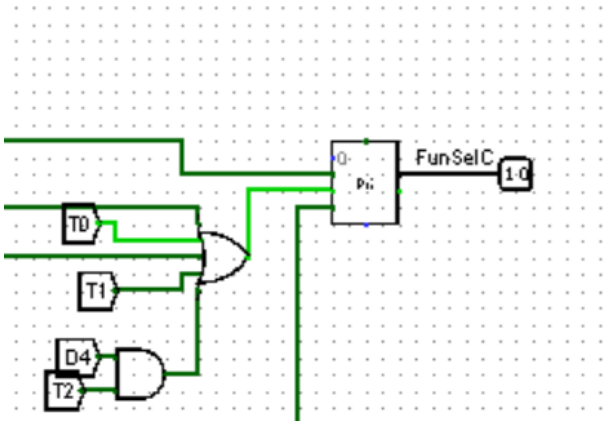

Figure 9: Example Output Generator Circut

2.2 D0

Rx ← Value should be carried out in this operation. Value may be two different inputs: It can be represented with Address part of instruction (IM) or can be M[AR] (D). Addressing Mode bit of instruction decides whether it is IM or D. T0 and T1 cycles are used to read instruction. Therefore, LD operation is carried out at T2 cycle and one clock cycle is sufficient for that operation. To set inputs of circuit properly, AND gates is used. That gates(Figure 10 and 11) means, if instruction is D0, time cycle is T2 and Addressing Mode is D or IM, then set the value according to the connected pin. In both (D and IM) cases, FunSelA should be 01. Therefore, These gates is connected to 01 pin of FunselA. In RegSelA input, these AND gate deciders need one more input which determines the target register. If Rx is R0 then these AND gates has one more input which is R0. After that, these AND gates is connected to the related pin of RegSelA. MuxASel should be 00, if Addressing mode is IM or it should be 01, if addressing mode is D. Hence, AND gate deciders is connected to MuxASel pins according to these informations. Finally, OutDSel should be 01, if Addressing Mode is D. Therefore, AND gate decider with D0, T2 and D inputs is connected to 01 pin of OutDSel.

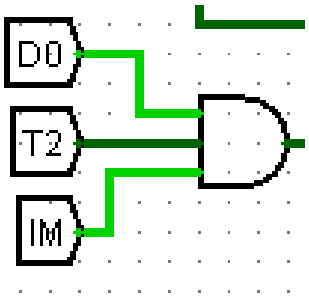

Figure 10: AND gate decider

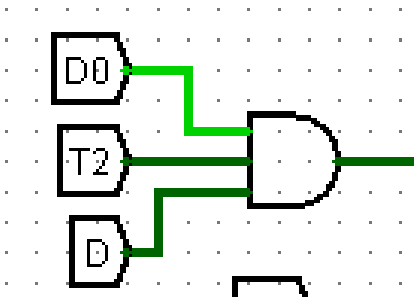

Figure 11: AND gate decider

2.3 D1

Value← Rx should be carried out in this operation. Value is M[AR] in this operation. That operation can be done just in one clock cycle, so T2 time cycle is sufficient. AND gate decider generally has three inputs such as D1, T2 and D. FunSelAlu should be 0001, MuxCSel should be 1, Store should be 1 and OutDSel should be 01. Therefore, AND gate deciders is connected to these informations. To determine Rx, one input is needed. R0, R1, R2 and R3 inputs should be added to AND deciders and should be connected to the OutASel's pins.

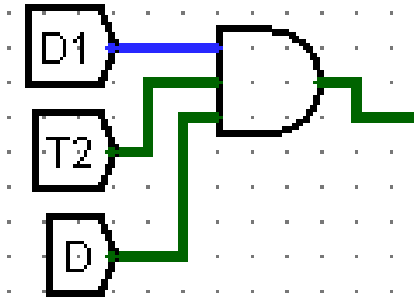

Figure 12: AND gate decider

2.4 D2

DESTREG← SRCREG1 should be carried out in this operation. D2 and T2 are fundamental inputs of AND gate deciders. Other inputs are connected according to the which one is destination and which one is source register. For example, if destination register is in part A, then AND gate decider has an input such as DEST-A and these deciders determines MuX's inputs. RegSelA can be 0001, 0010, 0100 and 1000 according to the destination register. R0-DEST, R1-DEST, R2-DEST and R3-DEST inputs determines the register which is loaded into.

FunSelAlu should be 0000 during this operation, so AND gate with D2 and T2 inputs are connected to the 0000 pin.

RegSelC should be set according to the which register is destination. AR-DEST, PC-DEST and SP-DEST inputs provides this decision mechanism.

If destination register is in the PartB, then FunSelC should be 01. DEST-B input provides this decision mechanism.

If source register1 is in the PartB, then OutCSel should be set according to which register is source. PC-REG1, AR-REG1 and SP-REG1 inputs provide this decision mechanism.

MuxASel should be 10, if source register1 is in the B part. REG1-B input provides this decision mechanism.

MuxBSel should be 11, if destination register is in the B part, DEST-B input provides this decision mechanism.

OutASel should be set according to which register in Part A is source. R0-REG1, R1-REG1, R2-REG1 and R3-REG1 inputs provide this decision mechanism.

FunSelA should be 01, if destination register is in Part A. DEST-A input provides this decision mechanism.

If source register is in Part A then, MuxCSel should be 1. If source register is in Part B, then MuxCSel should be 0. REG1-A and REG1-B inputs provides this decision mechanism and they are connected according to these informations.

FunSelC should be should be 01 if destination register is in Part B. DEST-B input provides this decision mechanism and it is connected according to the this information.

2.5 D3

M[SP] ← Rx, SP ← SP -1 should be carried out in this operation. It can be operated in one clock cycle, so T2 is sufficient for this operation. D3 and T2 are fundamental inputs of AND gate deciders. FunSelAlu should be 0001, RegSelC should be 100, OutDSel should be 10, Store should be 1 and FunSelC should be 10 to determine M[SP] and decrease SP by one. To load value from Rx to M[SP], AND gate deciders should have one more input which determines Rx. If Rx is R0, then inputs of AND gate decider should be D3, T2 and R0 and should be connected to the OutBSel's 00 pin. Other Rx register decision mechanism is implemented according to these information and connected to the OutBSel pins.

2.6 D4

SP ← SP + 1, Rx ← M[SP] should be carried out in this operation. There should be two clock cycles to operate. Therefore, T2 and T3 cycles is used. In T2 cycle, SP should be incremented by one. For this reason, FunSelC should be 10 and RegSelC should be 100. Inputs of AND gate decider are D4 and T2. In T3 cycle, M[SP] value should be taken and should be loaded. Therefore, FunSelA should be 01, MuxASEl should be 01, OutDSel should be 01. To load value from M[SP] to Rx , AND gate deciders should have one more input which determines Rx. If Rx is R0, then inputs of AND gate decider should be D4, T3 and R0 and should be connected to the RegSelA's 0001 pin. Other Rx register decision mechanism is implemented according to these information and connected to the RegSelA pins.

2.7 D5 - D13

Operations from D5 to D13 are taken into one subsection because, they are all register referenced arithmetic operations which most of them are done the same way. Due to required design of the circuit from Project 2, ALU can only perform binary operations if there is at most one operand register from registers PC, AP, SP and if there is an operand from those the result can only be loaded to those registers due to MUXA. Input are fed into ALU using MUXA and MUXC but only directly except D6 because, D6 performs the operation in reverse order and ALU does not have the operation "B-A" inputs are reversed and operation subtraction operation command is given to the funsel of ALU. Then, result is loaded into destination register in the same clock cycle with help of MUXC.

Unary operations D7(DEC) and D8(INC) can be performed between any two registers because operations D7-D8 are performed in two cycles. In the first clock cycle of the op-

eration D7 and D8 SRCREG1 is loaded into DESTREG just like MOV operation, in the upcoming cycle decrements/increments using the function selection and register selection of the destination register.

In order to control MUXA an extra "Special Condition" tunnel is formed. If the destination register is one of the registers R(0-3) MUXASel and operation is binary (Operations D5-D13 but not 7 or 8) must have the input "11". So, the given combination is fed into the "11" input of MUXASel encoder. Combination for OUTASel is done the same way but for operation D6 operand number is reversed.

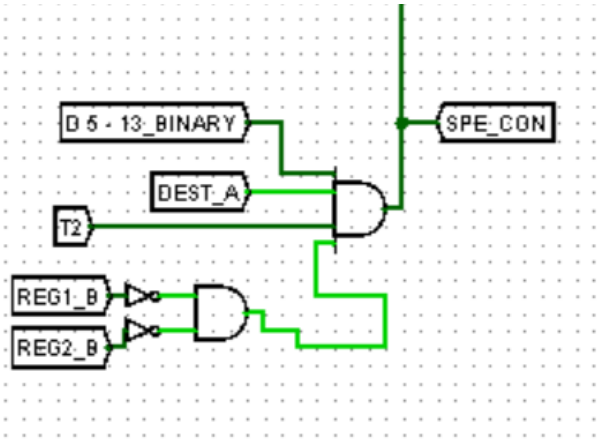

Figure 13: SPE-CON tunnel

If there is at least one input coming from register PC, AR, SP then MUXASel input must be "10". For that the given combination is fed into the "10" input of MUXASel encoder. Combination for OUTCSel is done the same way but for operation D6 operand number is reversed.

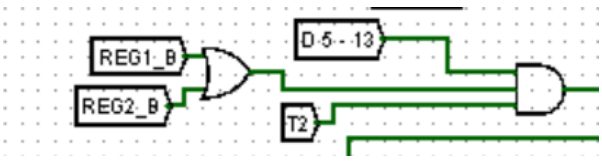

Figure 14: Given combination

After multiplexer arrangements, FunSelALU arrangements are made for the corresponding input and encoded known from project 2. For operation D7 and D8 FunSelALU is set to "0000" because DEC and INC is made by FunSel of the registers not ALU.

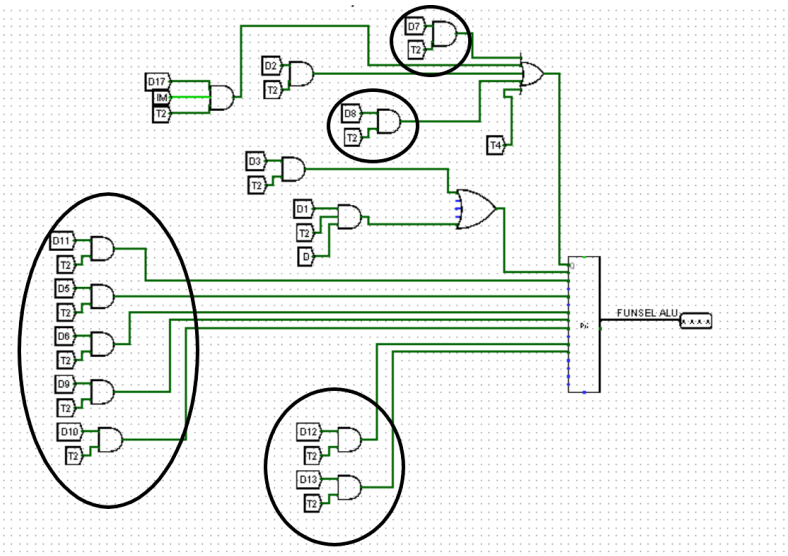

Figure 15: Given combination

After ALU performs the operation the result is loaded into destination with the corresponding combination of Funsel(A/C) RegSel(A/C). Consequently, cycle is done for binary operations (Operations D5-D13 but not 7 or 8). For operations D7 and D8 one more clock cycle is needed because for increment and decrement funsel of registers will be used. For that special purpose another "Special Condition" tunnel is formed.

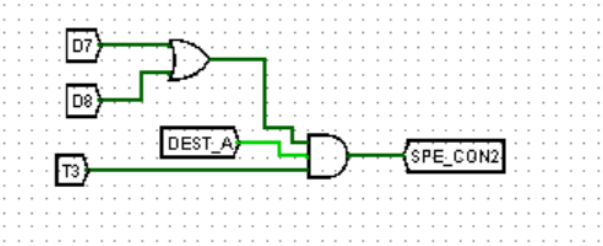

Figure 16: SPE-CON2 tunnel

With the help of that tunnel FunSels and RegSels are set to the right combination and destination register is incremented or decremented. As an example INC or DEC option for destination R0 is given. For FunSels of registers INC is "10" and DEC is "11".

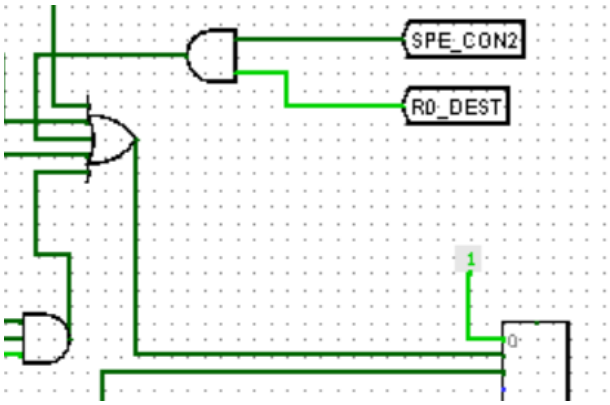

Figure 17: RegSel A combination to choose R0

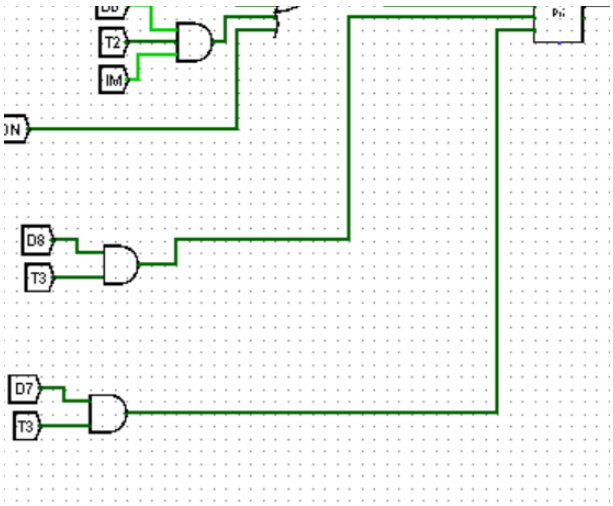

Figure 18: FunSel A combination to increment or decrement R0

2.8 D14 - D16

As described in Branch Control Bit section, we have realized that operations D14 - D16 have many similarities in terms of logic control so we came up with the design of the combinational logic circuit in Figure 8. D14 branches unconditionally whereas D15 branches if ZERO flag is raised and D16 branches if ZERO flag is not raised. Branch operation let us create small piece of programs such as for loops. If branching is succeded VALUE is directly loaded into PC by setting the multiplexers and registersels to right combination.

2.9 D17

We realized CALL operation in two clock cycle. At T2 and T3 clock signals we designed appropriate circuits in order to achieve this. !!!Because of excess number of nearly duplicate images there will be no example images more than 2, one for T2 and one for T3!!! T2: M[SP] ← PC, SP← SP-1
OutDSel 10, Select SP as the output of OutD
OutCSel 00, Select PC as the output of OutC
MuxASel 10, Let MuxAOut shows the PC
MuxCSel 00, send PC to the input A of the ALU
FunSelAlu 0000, OutALU → A
Store 01, Store Input to Memory
FunSelC 11, Selecting Decrement Operation on enabled register
RegSelC 100, Selecting SP

T3: PC ¡- Value
MuxBSel 00, Select IR(0,7)
FunSelC 01, Load Operation
RegSelC. 001, Select(enable) PC in order to load

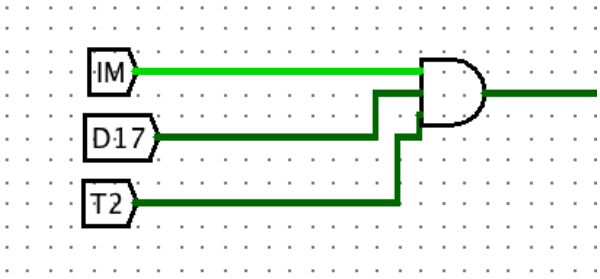

Figure 19: Combination at T2

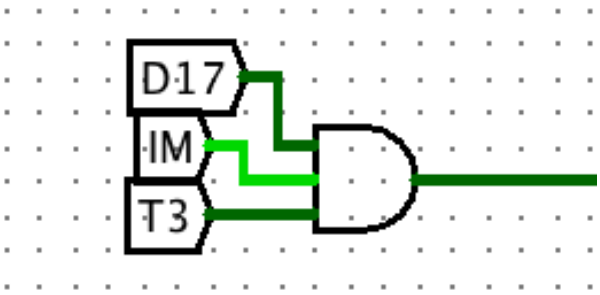

Figure 20: Combination at T3

2.10 D18

We realized RET operation in two clock cycle. At T2 and T3 clock signals we designed appropriate circuits in order to achieve this.

T2 CYCLE: SP ← SP + 1

RegSelC 100 Selecting (enabling) SP

FunSelC 10, Selecting Increment Operation on enabled register

T3 CYCLE: PC ← M[SP]

MuxBSel 10 Select value loaded from address

RegSelC 001 Selecting PC

FunSelC 01 Send 01 to FunSel of PC

OutD 10 Display output at Output D

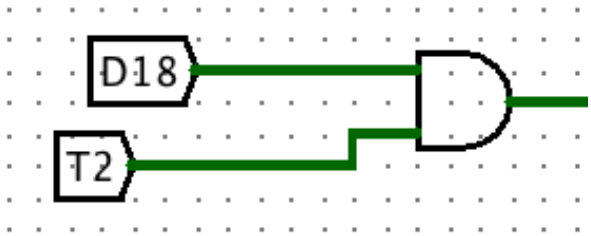

Figure 21: Combination at T2

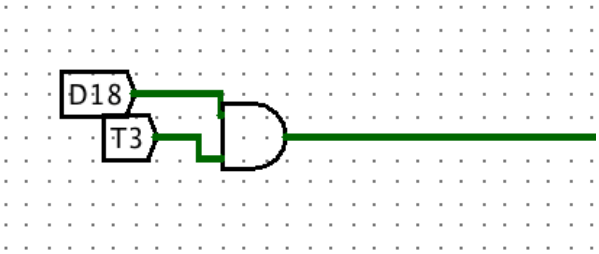

Figure 22: Combination at T3

3 DISCUSSION

In order to understand how processors work, we designed a simplified processor model in Logisim. M. Morris Mano introduces this simple processor model we designed as the BASIC COMPUTER. We practiced processor organization and the relationship of the RTL model to the higher-level computer processor. In order to design this project without any errors we had to have accomplish all the tasks wanted in previous experiments, which we did. In addition to previous parts, we added a Control Unit designed all the logic needed for a basic computer in this new part. One of the most important things we should understand in this part is Clock Cycles, because all operations we intended to make start with deciding how many clock cycles to use. Using a counter and decoder we get 5 T cycles, from T0 through T4.

Another important aspect of the control unit is designing a decoder which gives us operation codes from the Instruction Registers, so we can design more complex circuit parts in order to manipulate different parts of the previously designed projects as needed. We used a 5x32 decoder and chose 11-15 bits from IR to get 18 cables(tunnels) to use. To find the destinations needed for the instructions without addressing mode, we decomposed the needed bits from IR bits 8-10 and used 3x8 decoder. On the other hand, for the ones with addressing mode, using bits 9-10 and a 2x4 decoder we get the register we want among R0-R1-R2-R3. Also, we used IM and D signals as address reference which we realize through bit 8 and 1x2 decoder.

In operations we made some design decisions which give clear design in our circuit. In pperation part combined D5 – D13 ,which are N/A operations as Addressing Mode, into OR gate, so the circuit did become more clear. We also applied similar approach to D6 which is the subtraction operation. For Branch operation we realized a similar approach can be taken by using IM and T2 in an AND gate.

Due to some restrictions from project 2 (ALU), In operations from D5 to D13, register referenced arithmetic operations, we followed similar pattern. ALU performs operations and result is loaded into destination by combining Funsel and Regsel (A-C). This approach are which is explained in detail was necessary due to restrictions from previous projects. However, main combinational circuit designs are the same as other operations. Other operations with different addressing mode, IM or D, are realized similarly. For example in PSH, 0x03 operation we enable the register we wanted, apply the operation through correct combinational circuit logic and again select a register and load or do any other operations asked.

In addition to many test operations we tried during designing the Basic Computer, we also tested final project with the given Assembly code which is given below. Basic Computer we designed did every operation as expected and worked without any unex-

pected result. However, this result was not surprising at all. It was the result of many hours of Zoom meetings which interrupted in every 40 minutes because of free student accounts, many hours of discussions over implementations details and trying to decide on how to abstract different concepts so the Basic Computer would be more EFFICIENT. You heard me right, we just did not want this computer to become a living thing. We wanted this computer to become one of the most efficient and reliable BEAST among all basic computers ever created. To be honest, we can say that we have achieved this through extraordinary efforts.

4 CONCLUSION

In this project, we implemented a hardwired control-circuit. We learned that the control circuit operates all circuits like a maestro, and does this due to the corresponding operation. But to implement this circuit we should have all the knowledge about operation individually. Because we calculated the clock cycles and at that cycle which small operation should be done. But implementing this project took more time than we anticipated. Because some of the operations can not be performed in the given circuit design. So we tried to find different ways to perform them and to do that we tried lots of redundant implementations and not all of them suitable for the operations. So we solved this difficulty by doing other operations and waited for an answer from our instructor after we got the satisfying answer we implemented those operations in an updated way.