ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION ASSIGNMENT REPORT

GROUP NO : 20

GROUP MEMBERS:

150180006 : DENİZ BÜYÜKTAŞ

150170070 : MUSTAFA EREN ŞENTÜRK

150170009 : SAMED CAN ÇOBANLI

150170090 : ENES ŞAŞMAZ

SPRING 2020

Contents

1 INTRODUCTION

The control unit (CU) is a component of a computer's central processing unit (CPU) that directs the operation of the processor. It tells the computer's memory, arithmetic and logic unit and input and output devices how to respond to the instructions that have been sent to the processor. It controls many execution units (i.e. ALU, data buffers and registers) contained within a CPU. In this project we use hard-wired control unit. Hard-wire control unit is implemented with the help of gates, flip flops, decoders etc. in the hardware. The inputs to control unit are the instruction register, flags, timing signals etc. This organization can be very complicated if we have to make the control unit large. This design was made using the parts in Project-2.

2 MATERIALS AND METHODS

In this assignment, we had to construct control unit of a basic computer that implement given instructions. We were given an instructions list:

OPCODE (HEX)	SYMB	ADDRESSING MODE	DESCRIPTION
0x00	LD	IM, D	Rx ← Value (Value is described in Table 3)
0x01	ST	D	Value ← Rx
0x02	MOV	N/A	DESTREG ← SRCREG1
0x03	PSH	N/A	M[SP] ← Rx, SP ← SP - 1
0x04	PUL	N/A	SP ← SP + 1, Rx ← M[SP]
0x05	ADD	N/A	DESTREG ← SRCREG1 + SRCREG2
0x06	SUB	N/A	DESTREG ← SRCREG2 - SRCREG1
0x07	DEC	N/A	DESTREG ← SRCREG1 - 1
0x08	INC	N/A	DESTREG ← SRCREG1 + 1
0x09	AND	N/A	DESTREG ← SRCREG1 AND SRCREG2
0x0A	OR	N/A	DESTREG ← SRCREG1 OR SRCREG2
0x0B	NOT	N/A	DESTREG ← NOT SRCREG1
0x0C	LSL	N/A	DESTREG ← LSL SRCREG1
0xOD	LSR	N/A	DESTREG ← LSR SRCREG1
0x0E	BRA	IM	PC ← Value
0x0F	BEQ	IM	IF Z=1 THEN PC ← Value
0x10	BNE	IM	IF Z=0 THEN PC ← Value
0x11	CALL	IM	M[SP] ← PC, SP ← SP – 1, PC ← Value
0x12	RET	N/A	SP ← SP + 1, PC ← M[SP]

Figure 1: Instructions Table

Before we start to examine functions to be implement, we add the circuit some necessary input organizers and some counters for clock signal. Firstly, we constructed a clock system by using a decoder and a counter:

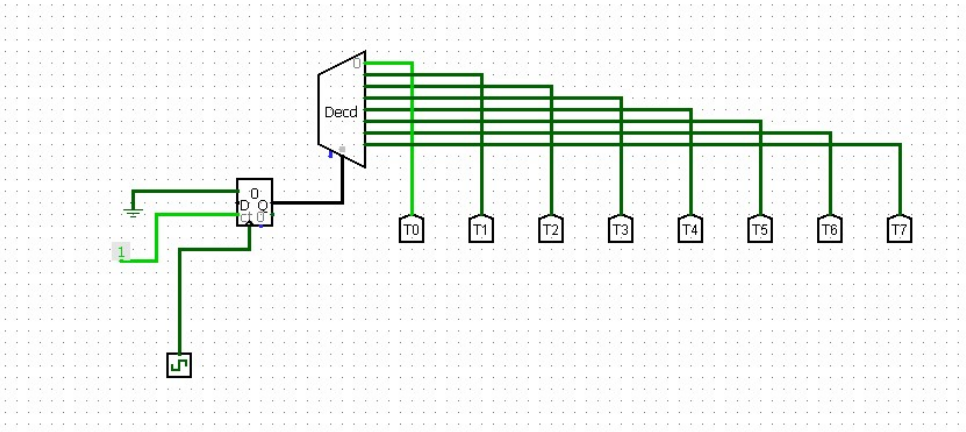


Figure 2: Clock system

Fetch operation was important for us to realise functions in our basic computer system. Therefore, we used first two clock for fetch operation. At the first clock signal for loading the most significant 8 bits of the Instruction register and the second clock T_0 for loading the least significant 8 bit:

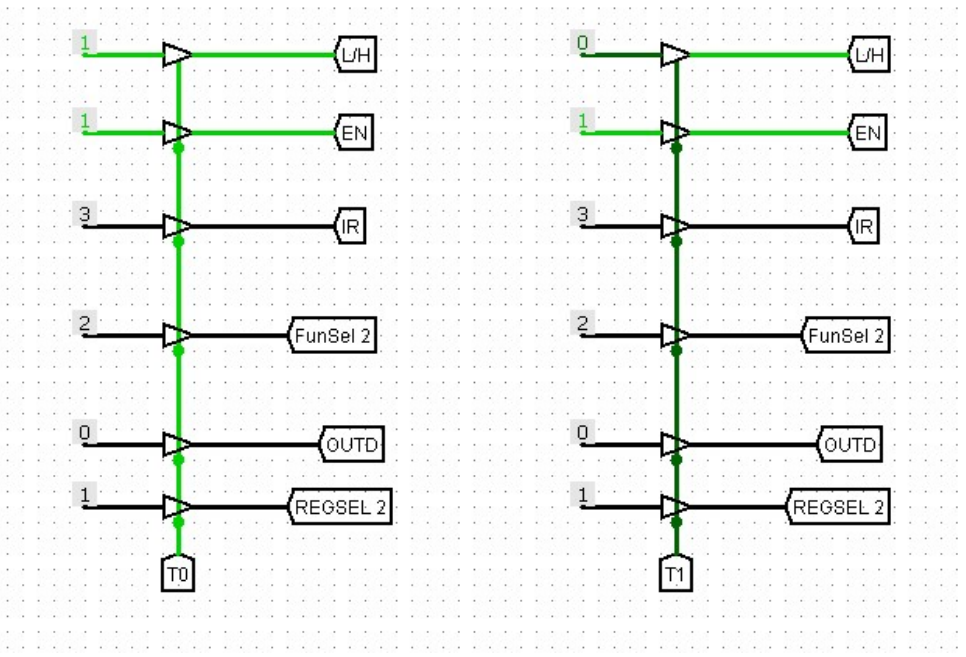


Figure 3: Fetch system

While this function was calling, we had to increment PC value to reach next instruction hence we load the Funsel value of AR as 10 and the Regsel of it as 001. When instruction register is loaded, we separated its bits for two different instruction format:

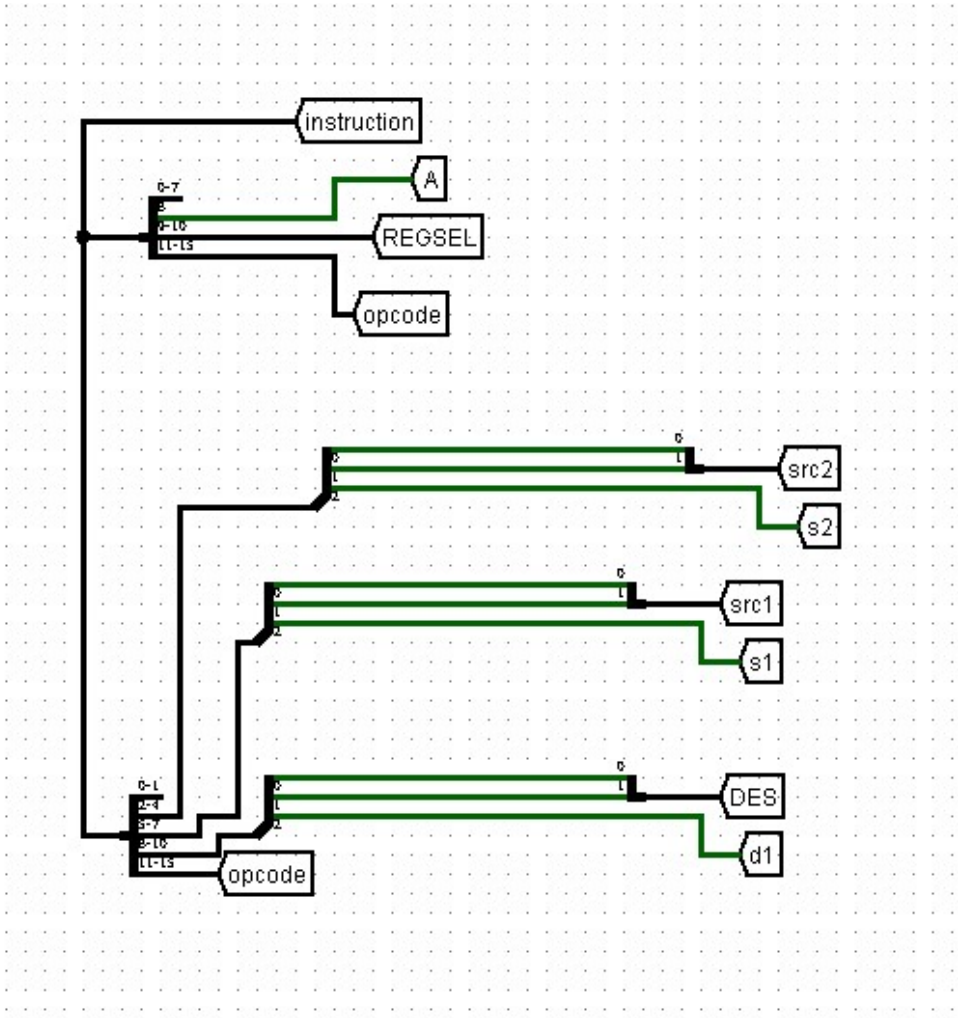


Figure 4: Instruction transfer to sub-inputs

And we had two different instruction format. First of them is instruction with address field:

(1) Instructions with address reference has the format shown in Figure 1:

- The OPCODE is a 5-bit field (See Table 1 for the definition).
- The REGSEL is a 2-bit field (See left side of Table 2 for the definition).
- The ADDRESSING MODE is a 1-bit field (See Table 3 for the definition).
- The ADDRESS is 8 bits

OPCODE	REGSEL	ADDRESSING MODE	ADDRESS

Figure 5: Instruction with address

2.1 Instructions with address reference

In this format of instruction. We were given a 16-bit instruction to fetch and decode. In this format, we had 4 separate region for determining different values. First of them is OPCODE that determines the function that we had to implement in the circuit. The second 2-bit area is the Rx that determines the register that we will apply the function. The next one bit is addressing mode and the least significant 8 bits of the instruction were constructing the address value. There were 9 functions to implement in this instruction format. They were:

2.1.1 0x00 LD Operation

In this function, we had to load the value to the required register. We were able to implement this function at a single clock signal. Firstly, this function could work in two different ways according to the addressing mode. If the addressing mode is 0, our value will be address field of the instruction register, other way; it will be M[AR]. Firstly we implemented the function when addressing mode is equals to 0. For transferring IR(0-7) to the register that located in register file, we made MUXA input 00:

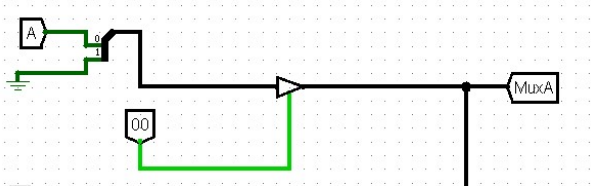


Figure 6: MUXA bus system 0x00 function part

After that for loading operation to register file, we had to make Funsel of the register file 01. For this operation, we used a bus Funsel bus system again as follows:

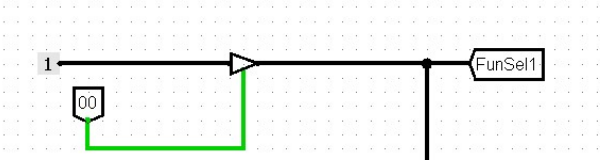


Figure 7: Funsel bus system 0x00 function part

And the last input that we had to modify for this function was Regsel of the register file. For implementing this part, we had to select the register that we will apply the load operation on it. If the Regsel of the instruction is 00, we had to load R0, in this case, we had to send Regsel of the RF 0001. This values were changing 0010 for R1, 0100 for R2 and 1000 for R3. For making this bus system we had implemented a Regsel bus system, then we used a multiplexer as follows:

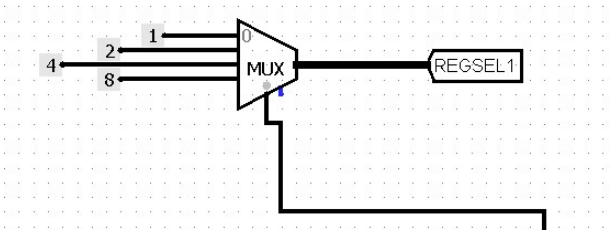


Figure 8: MUX for Regsel of RF

And we sent the Regsel value to the bus:

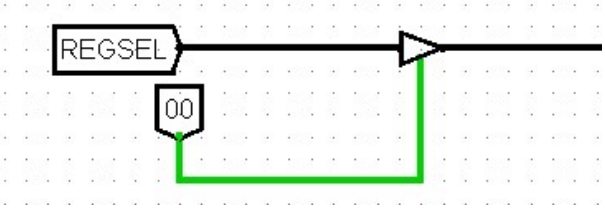


Figure 9: Regsel bus system 0x00 function part

And after that we moved to implement the same function when addressing mode is 1. For transferring the M[AR], we had to send memory the AR value. And for realising this, we had to select OutDSel as 01. Like RF Regsel input, we had implemented a MUX-bus system for OutCSel for choosing 01 value when the instruction area was 10. Our MUX is as follows:

And the value that we sent to bus:

For choosing the MemoryOut at A MUX, we had to send it 01. We implemented it by using this solution:

For the reason that we are applying load function to Register File as at the previous

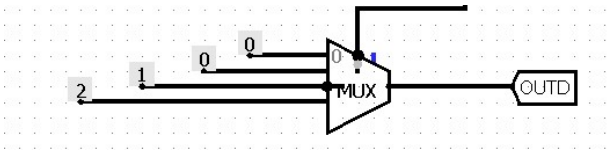


Figure 10: MUX for OutCSel of AR

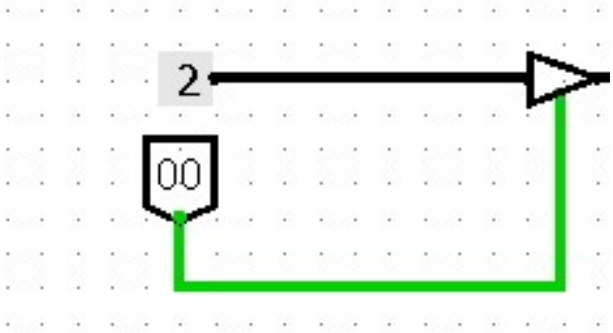


Figure 11: Bus for OutCSel of AR

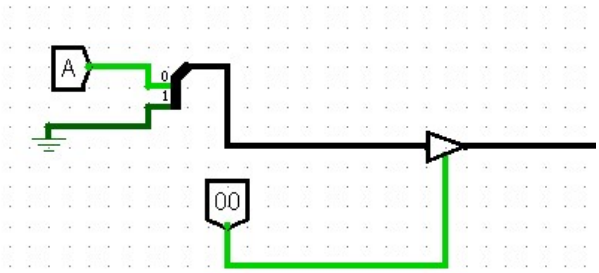


Figure 12: MUXA bus system 0x00 function part

one, the Funsel and the Regsel of our RF remained same with the LD operation when addressing mode is 0.(previous values).

2.1.2 0x01 ST operation

In this function, we had to load the value of the given register to the Value. Value was the M[AR], because our addressing mode is direct on the table. For implementing this function, first thing that we did was sending AR value to the memory. As at the previous function, we sent to the OutDSel 01 value:

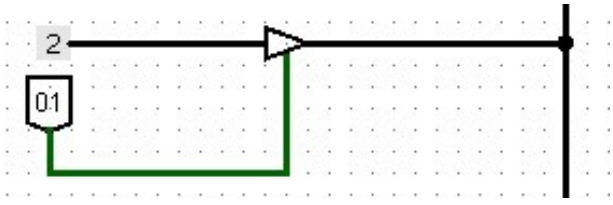


Figure 13: OutDSel bus system for the 0x01 function

After that, for selecting the value of the given register, we sent to the OutBSel the RegSel Value:

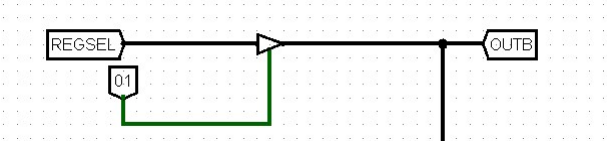


Figure 14: OutBSel bus system for the 0x01 function

After we have taken the B value, for transferring it to the memory. OutAlu should be the B input of the ALU. And for this to happen, Funsel value of the ALU should be 0001. We implemented it by using a bus system:

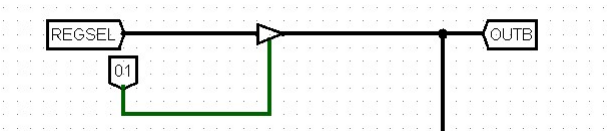


Figure 15: FunSelAlu bus system for the 0x01 function

2.1.3 0x03 PSH operation

In this operation, this time we had to take the value of the given register and load it to the M[SP]. While doing this operation, we had to decrement SP's value by 1 at every turn. First of all, we had to sent SP value to the memory, and for this we used the OutDSel bus system:

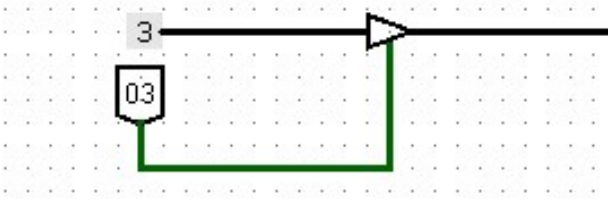


Figure 16: OutDSel bus system for the 0x03 function

For providing the decrement operation at SP, we had to make FunSel of Address Register as 11 which signifies the decrement by 1, and Regsel of this AR should choose the SP to apply the function:

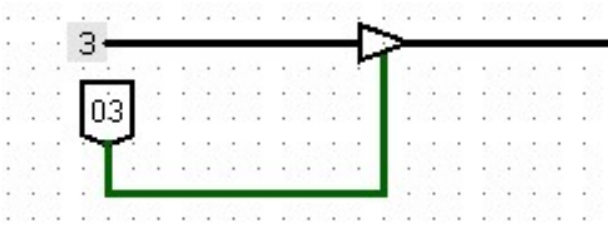


Figure 17: Regsel of AR bus system for the 0x03 function

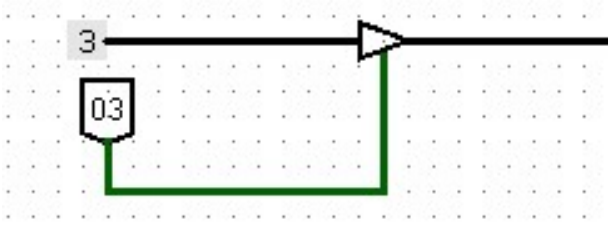


Figure 18: Funsel of AR bus system for the 0x03 function

And after we have implemented this part, we had to transfer the Rx that is the register which we will took the value from. As in the previous function, for realising this, we used the B input of the ALU. Our OutBSel:

2.1.4 0x04 PSH operation

In this function, we had to implement a reverse way from the previous function. We had to load the Rx value with M[SP] and after that, we had to increment the SP value

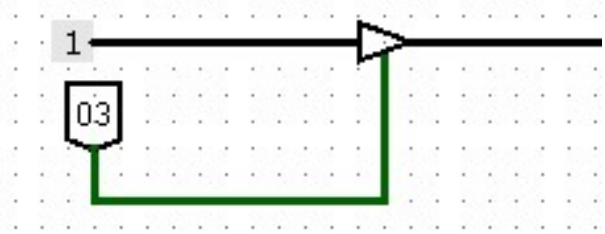


Figure 19: OutBSel bus system for the 0x03 function

After we have taken the B value, for transferring it to the memory. OutAlu should be the B input of the ALU. And for this to happen, Funsel value of the ALU should be 0001. We implemented it by using a bus system:

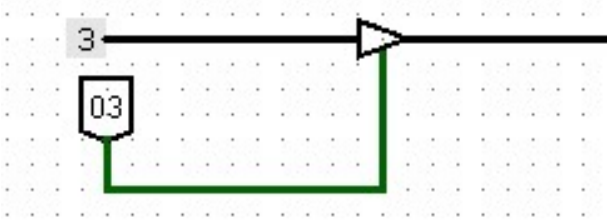


Figure 20: FunselAlu bus system for the 0x03 function

by 1. For transferring the OutMemory to the Register File, firstly we sent the OutDSel again SP value by doing this system:

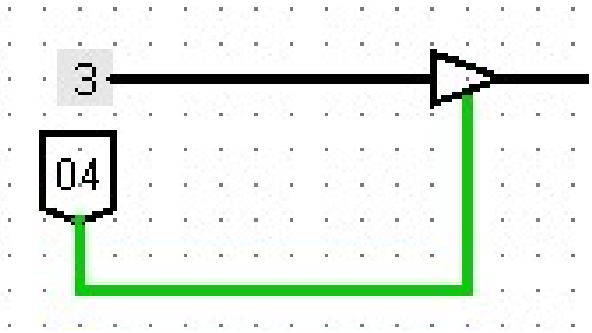


Figure 21: OutDSel bus system for the 0x04 function

After that, for incrementing operation, while Funsel should be 10, the Regsel of the AR should select the SP register. Our bus systems are as follows for implementing this:

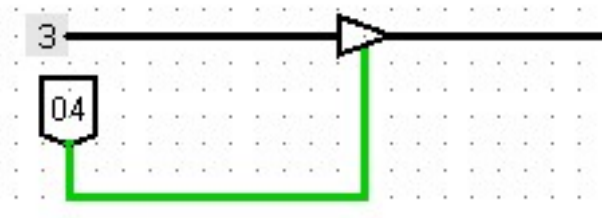


Figure 22: Regsel of AR bus system for the 0x04 function

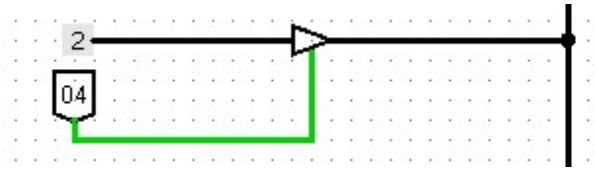


Figure 23: Funsel of AR bus system for the 0x04 function

After we have taken the output value of the memory, we should transfer it to Register file, and for this MuxASel should be 01:

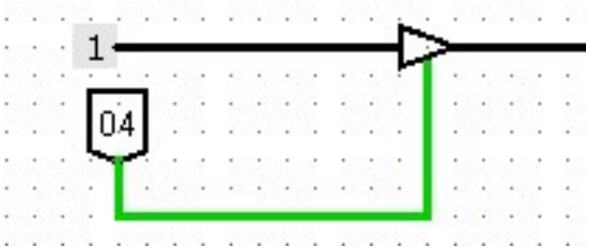


Figure 24: MuxASel bus system for the 0x04 function

And finally for implementing load operation to the register, we should send Regsel of the instruction to the Regsel of the RF and we had to send 01 value as the Funsel input of RF which signifies the load operation:

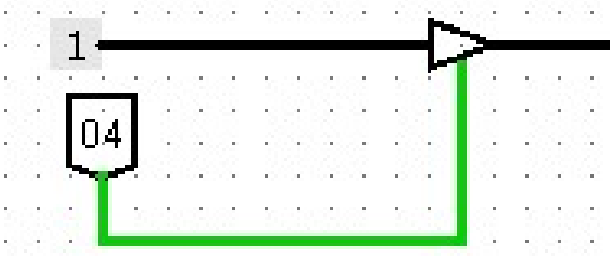


Figure 25: RegSel of RF bus system for the 0x04 function

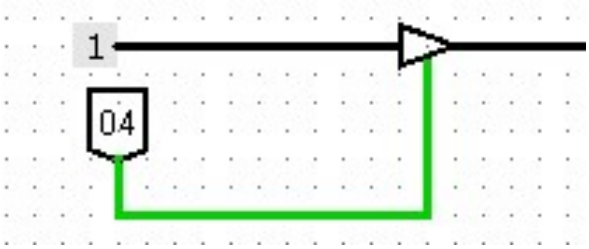


Figure 26: FunSel of RF bus system for the 0x04 function

2.1.5 0x0E BRA operation

In this Branch operation, we had to load address field of the instruction to the PC register. For realising this implementation. Firstly we had to send IR(0-7) value to the Address Register and for doing this, MuxBSel should be 01:

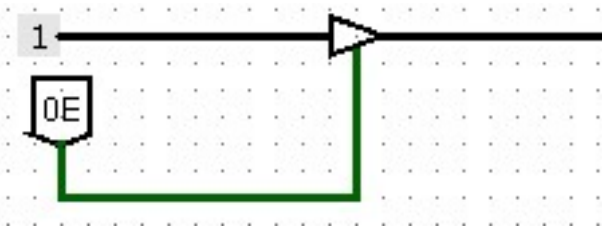


Figure 27: MuxBSel of RF bus system for the 0x05 function

And for loading to SP operation Funsel of RF should be 01 and Regsel of it should select the PC register:

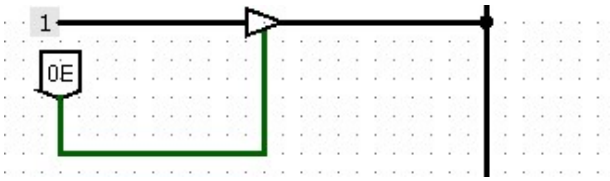


Figure 28: MuxBSel of RF bus system for the 0x05 function

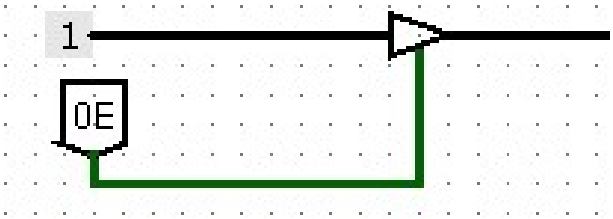


Figure 29: MuxBSel of RF bus system for the 0x05 function

2.1.6 0x0F BEQ operation

In this Branch operation we had to load address field of the instruction to the PC register like we did in 0X0E(BRA) part with a small change.Unlike the previous part, we needed to perform this operation if the value from the Z flag was 1. Firstly we designed the MuxBSel to send IR value to the address register. In this part we used an AND gate to consider Z flag value. So MuxB should be 01 and z should be 1 for this case:

After selecting the IR value from MuxBSel, to load this value to any register we should select FunSel as 01. After selecting FunSel value now we should select the PC register.

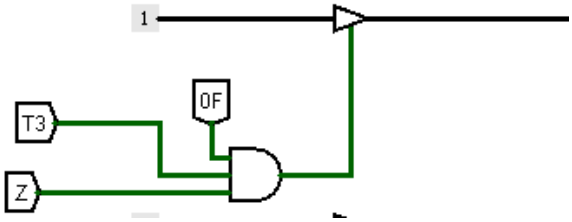


Figure 30: MuxBSel of RF bus system for the 0x0F function

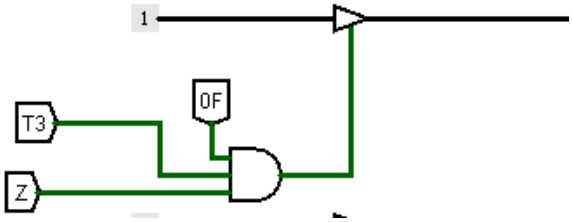


Figure 31: FunSel2 of RF bus system for the 0x0F function

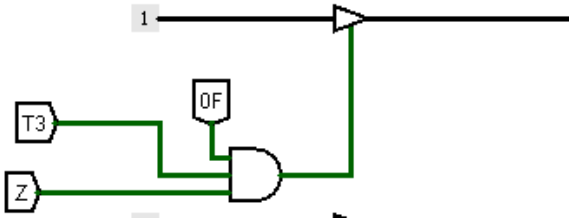


Figure 32: Regsel2 of RF bus system for the 0x0F function

2.1.7 0x10 BNE operation

In this Branch operation we did same operations with 0x0F branch with a small different. In this section, unlike the previous one, we took the opposite of the Z flag and did the same operations in the last section.

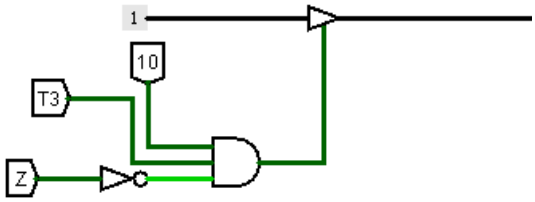


Figure 33: MuxBSel of RF bus system for the 0x10 function

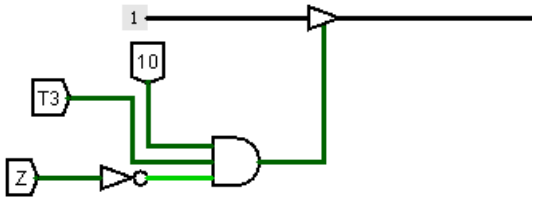


Figure 34: FunSel2 of RF bus system for the 0x10 function

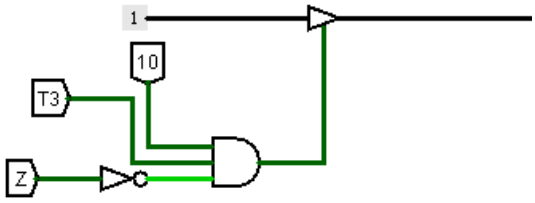


Figure 35: Regsel2 of RF bus system for the 0x10 function

2.1.8 0x11 CALL operation

In this function we had to load address field of the instruction to the PC register. In this part we did same operations like we did in 0x0E. After loading process, we chose the value from PC register with OutC and the value from SP register with OutD:

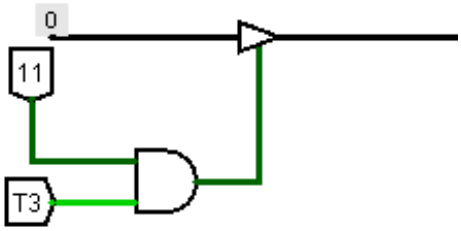


Figure 36: OutC of RF bus system for the 0x11 function

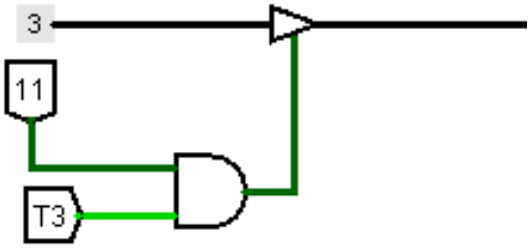


Figure 37: OutD of RF bus system for the 0x11 function

After OutC and OutD assignment we choose OutC value from MuxASel and MuxC-Sel.After this operations we select and pass the value of OutC from ALU with Fun-Sel(ALU)=0000.

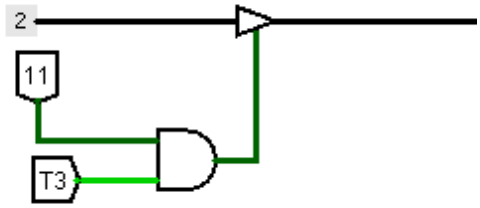


Figure 38: MuxASel of RF bus system for the 0x11 function

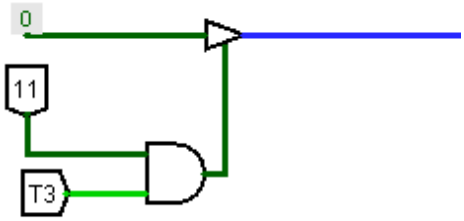


Figure 39: MuxCSel of RF bus system for the 0x11 function

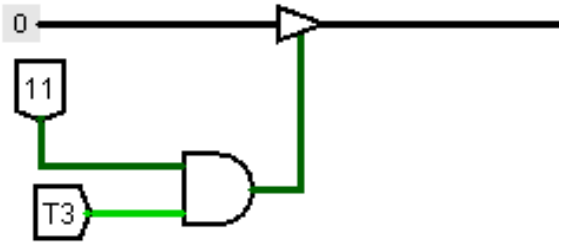


Figure 40: FunSel(ALU) of RF bus system for the 0x11 function

After first clock we applied second clock and decremented SP register. By doing that we change the address of memory. To do this operation we change the FunSel2 to 11(decrements operation) after decrementing SP we select RegSel2 as 100 to select OutD as SP by doing that we could select the address of the memory. After selecting the address we could assign the value to that address.

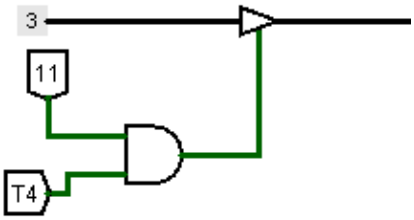


Figure 41: FunSel2 of RF bus system for the 0x11 function

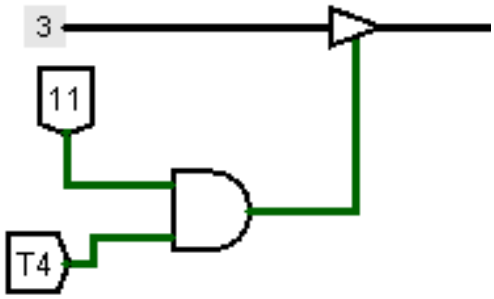


Figure 42: RegSel2 of RF bus system for the 0x11 function

2.1.9 0x12 RET operation

In this part we have assigned the value from the SP part of the memory to the PC register in first clock period. To take value from memory we choose MuxBSel as 10:

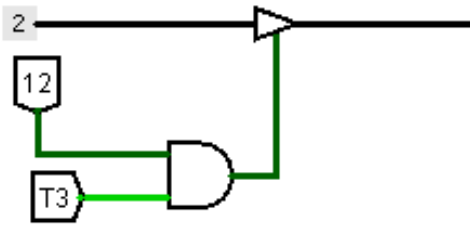


Figure 43: MuxBSel of RF bus system for the 0x12 function

After MuxBSel part we set FunSel2 as 01 to load the value and RegSel2 as 01 to select PC register.By doing so, it allowed us to assign the value from memory to the PC register.

After first clock operations we implement second clock operations to increment SP value by 1. To do this operations we set FunSel2 as 10(incerement) and RegSel2 as 11(100 SP register). With this assignments we achieved our goal.

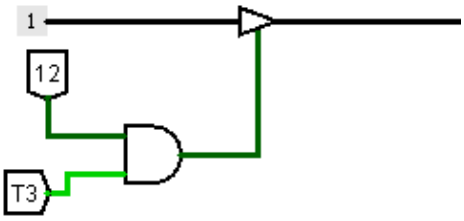


Figure 44: FunSel2 of RF bus system for the 0x12 function

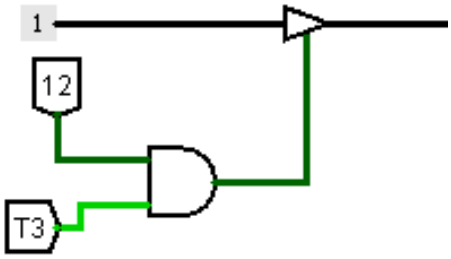


Figure 45: RegSel2 of RF bus system for the 0x12 function

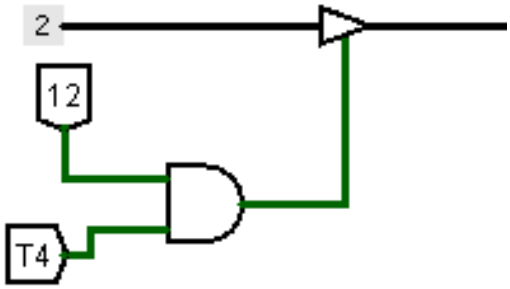


Figure 46: FunSel2 of RF bus system for the 0x12 function

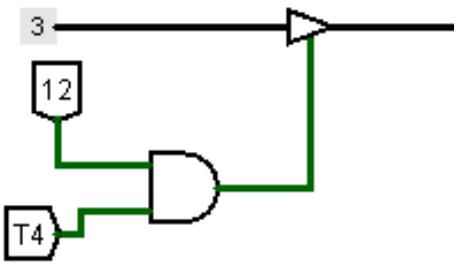
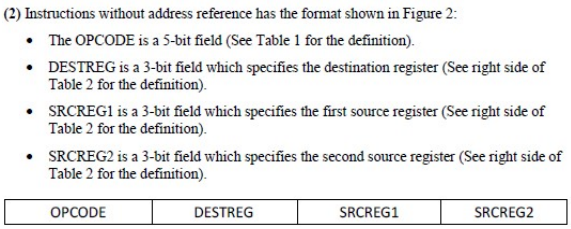


Figure 47: RegSel2 of RF bus system for the 0x12 function

2.2 Instructions without address

Format of this instructions is as follows:



(2) Instructions without address reference has the format shown in Figure 2:

- The OPCODE is a 5-bit field (See Table 1 for the definition).
- DESTREG is a 3-bit field which specifies the destination register (See right side of Table 2 for the definition).
- SRCREG1 is a 3-bit field which specifies the first source register (See right side of Table 2 for the definition).
- SRCREG2 is a 3-bit field which specifies the second source register (See right side of Table 2 for the definition).

OPCODE	DESTREG	SRCREG1	SRCREG2

Figure 48: Instructions without address

2.2.1 0x02 MOV OPERATION

In this operation we load the value of a register to another resister. For realising this operation our Funsel and Regsel values of the RF and AR were load opeartion with respect to destination registor at the regsel. We selected MuxA value according to source register and after that we sent the value of the source register to data way. Our MuxB select bits were 11 and our ALU was transferring the A input to the output. Our bus system photos are as follows:

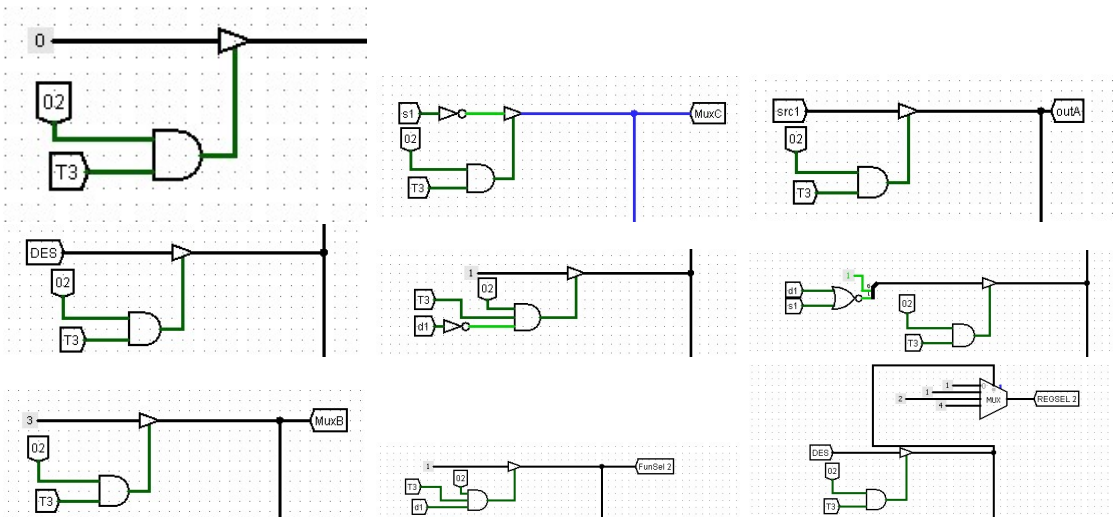


Figure 49: MOV operation bus systems

2.2.2 0x05 ADD operation

In this section, we collect the values we get from 2 different registers and write the value to the result register.

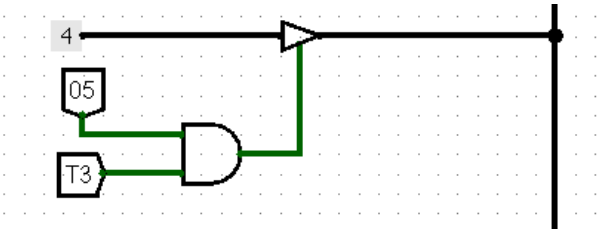


Figure 50: FunselALU for the 0x05 function

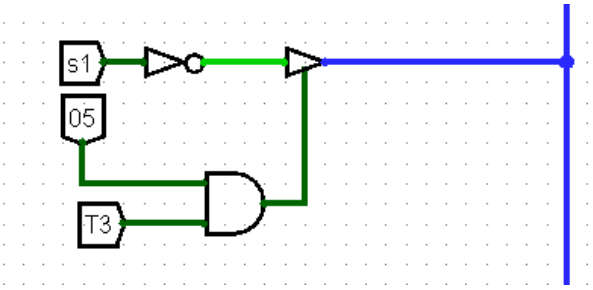


Figure 51: MuxCSel for the 0x05 function

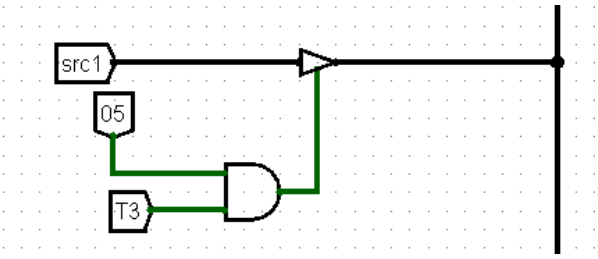


Figure 52: OutA for the 0x05 function

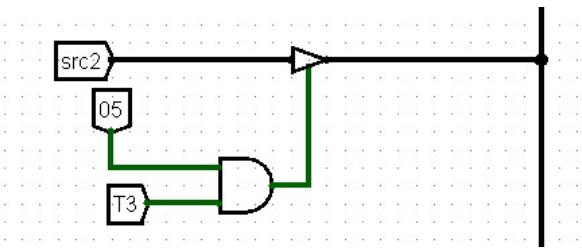


Figure 53: OutB for the 0x05 function

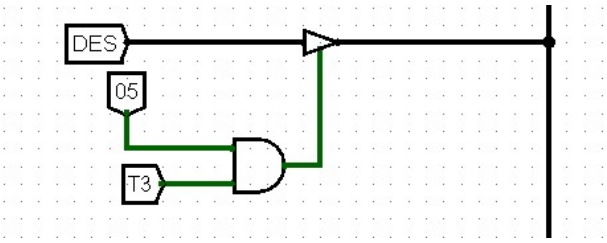


Figure 54: RegSel1 for the 0x05 function

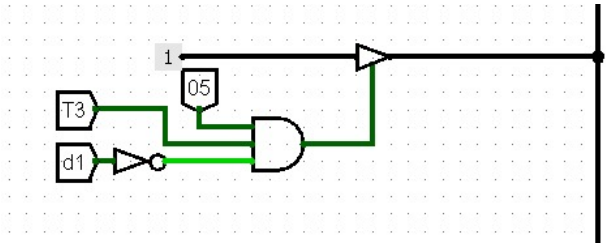


Figure 55: Funsel1 for the 0x05 function

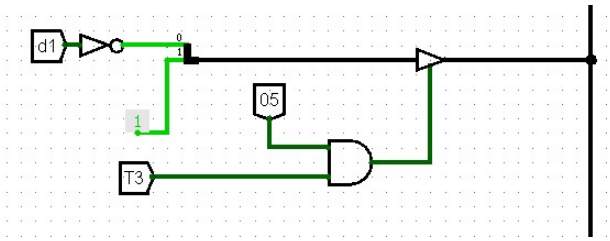


Figure 56: MuxA for the 0x05 function

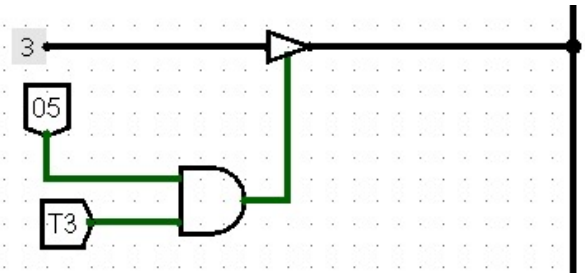


Figure 57: MuxB for the 0x05 function

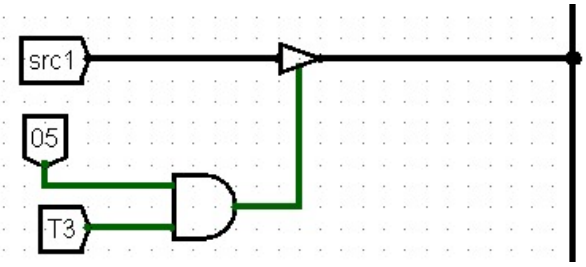


Figure 58: OutC for the 0x05 function

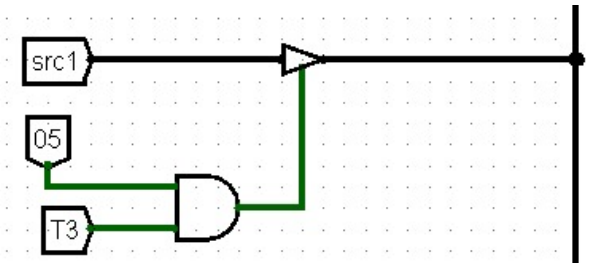


Figure 59: Funsel2 for the 0x05 function

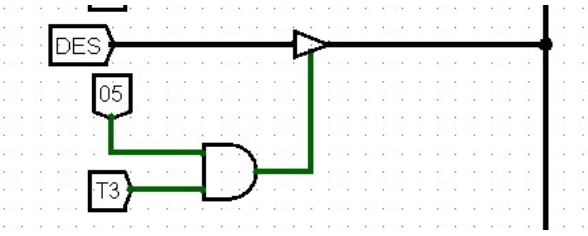


Figure 60: Regsel2 for the 0x05 function

2.2.3 0x06 SUB operation

In this section, we subtract the values we get from 2 different registers and write the value to the destination register.

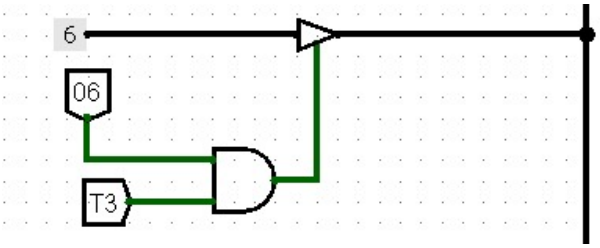


Figure 61: FunselALU for the 0x06 function

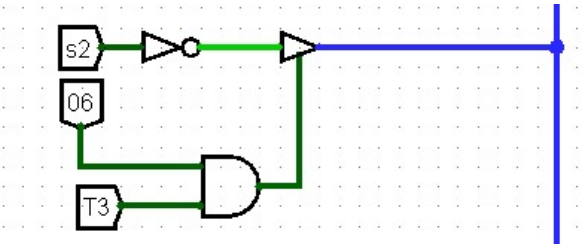


Figure 62: MuxCSel for the 0x06 function

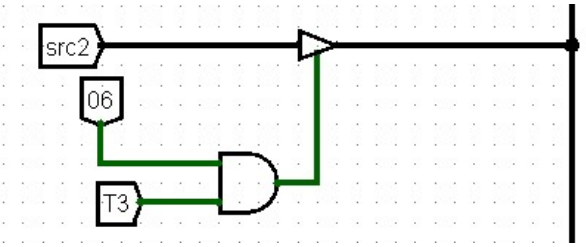


Figure 63: OutA for the 0x06 function

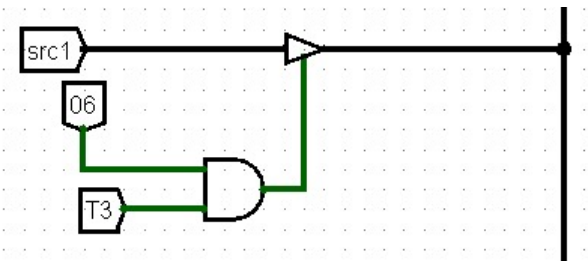


Figure 64: OutB for the 0x06 function

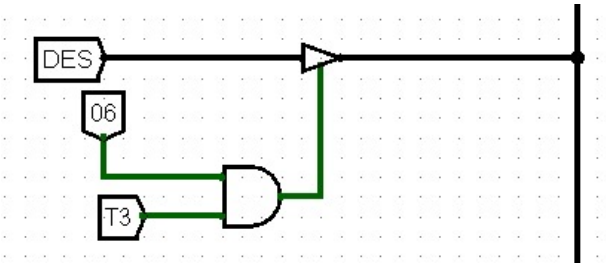


Figure 65: RegSel1 for the 0x06 function

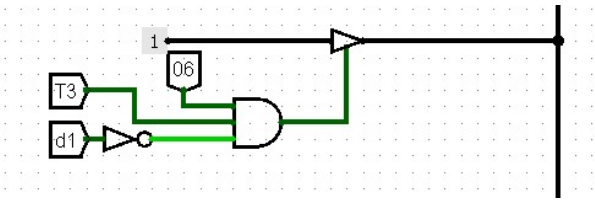


Figure 66: Funsel1 for the 0x06 function

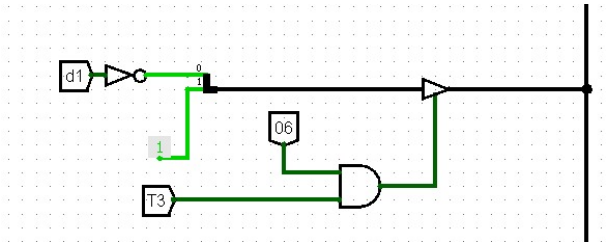


Figure 67: MuxA for the 0x06 function

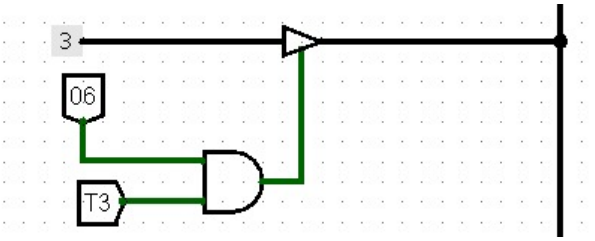


Figure 68: MuxB for the 0x06 function

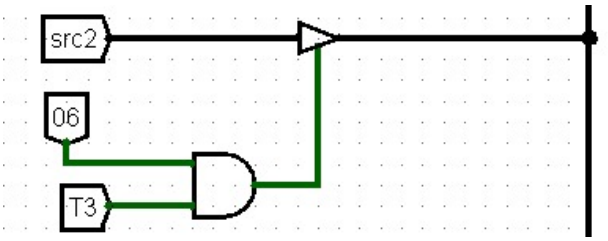


Figure 69: OutC for the 0x06 function

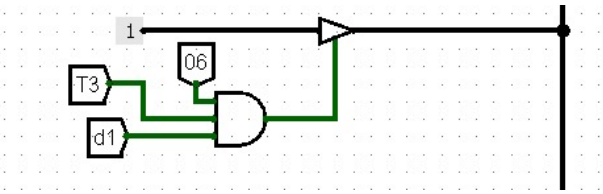


Figure 70: Funsel2 for the 0x06 function

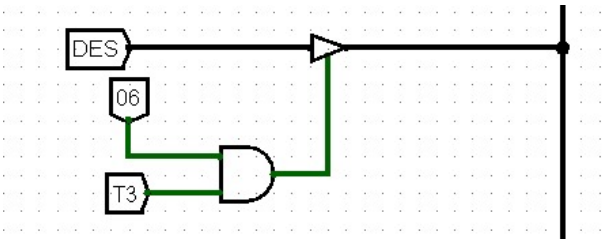


Figure 71: Regsel2 for the 0x06 function

2.2.4 0x07 DEC operation

It loads the value we get from the source register to the destination register by decreasing it.

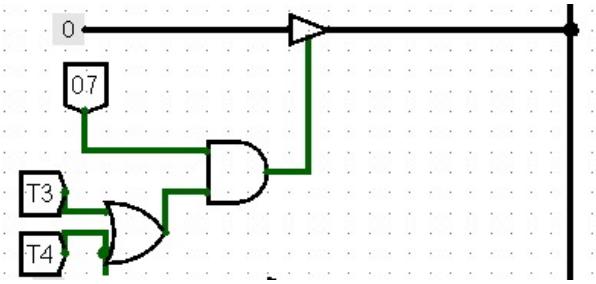


Figure 72: FunselALU for the 0x07 function

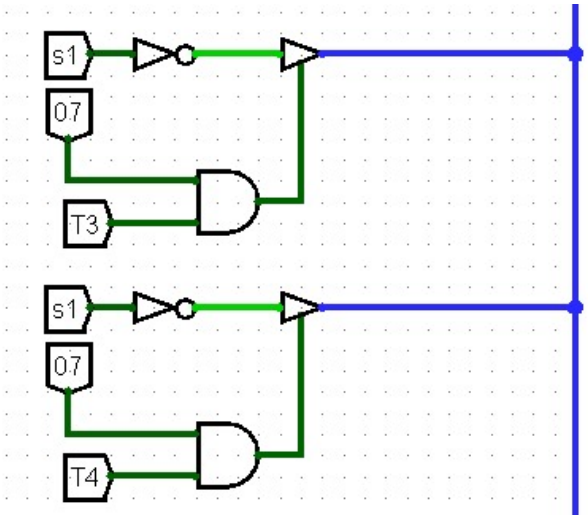


Figure 73: MuxCSel for the 0x07 function

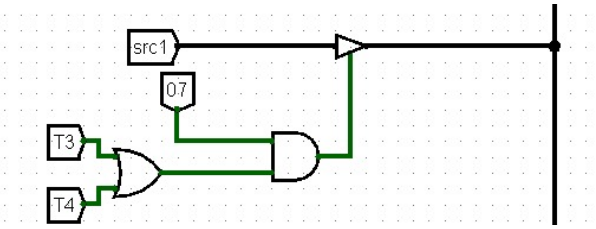


Figure 74: OutA for the 0x07 function

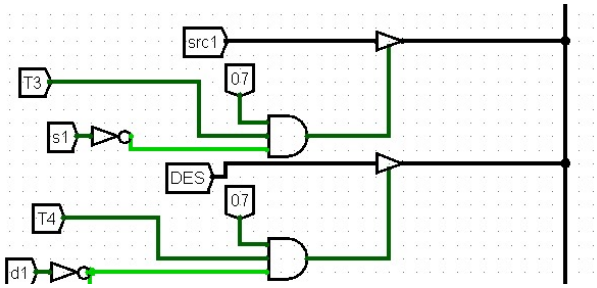


Figure 75: RegSel1 for the 0x07 function

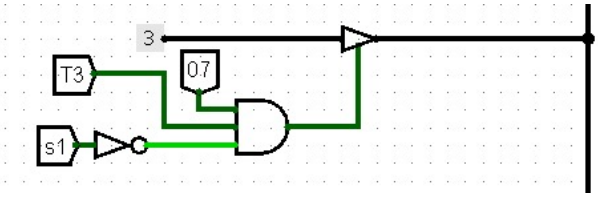


Figure 76: Funsel1 for the 0x07 function

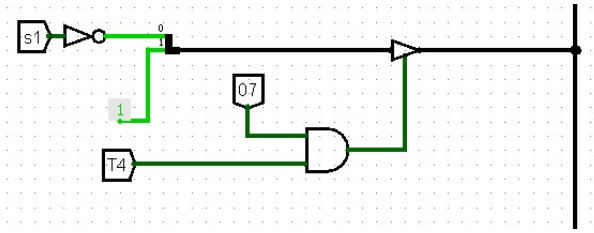


Figure 77: MuxA for the 0x07 function

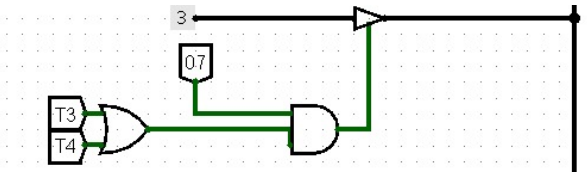


Figure 78: MuxB for the 0x07 function

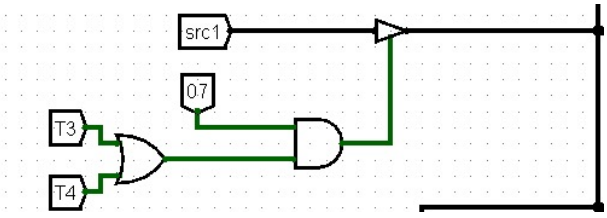


Figure 79: OutC for the 0x07 function

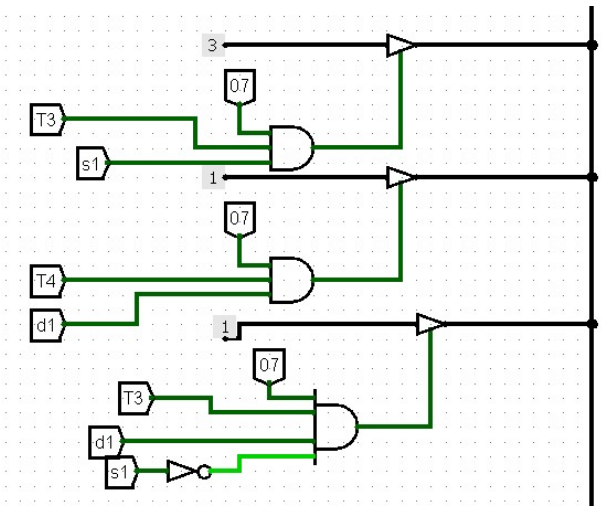


Figure 80: Funsel2 for the 0x07 function

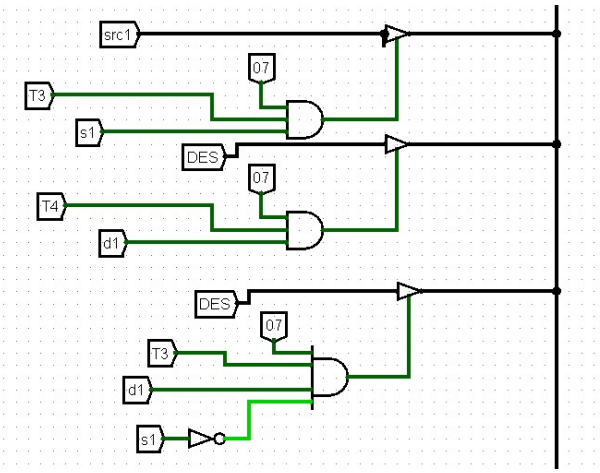


Figure 81: Regsel2 for the 0x07 function

2.2.5 0x08 INC operation

It loads the value we get from the source register to the destination register by increasing it.

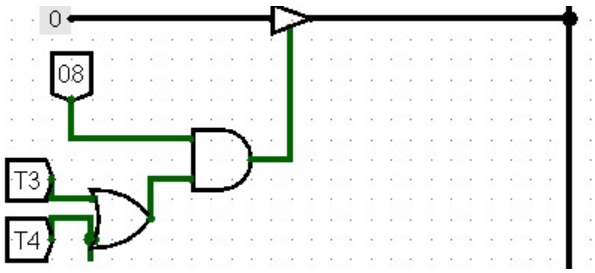


Figure 82: FunselALU for the 0x08 function

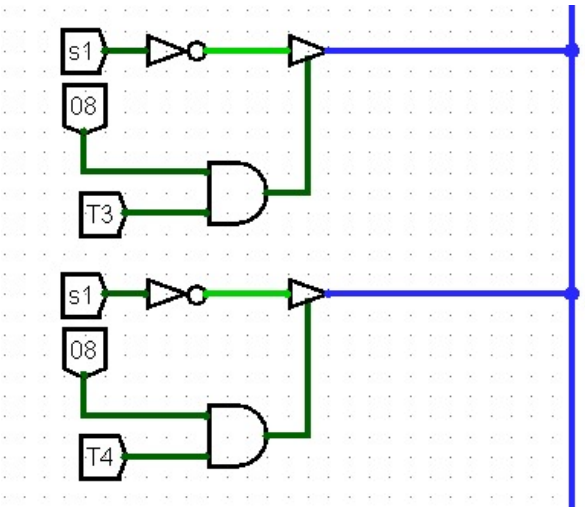


Figure 83: MuxCSel for the 0x08 function

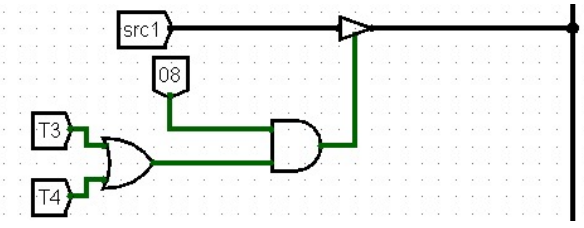


Figure 84: OutA for the 0x08 function

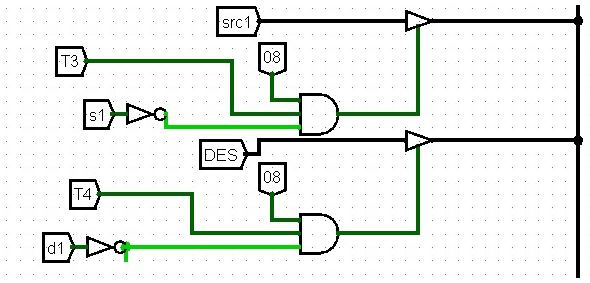


Figure 85: RegSel1 for the 0x08 function

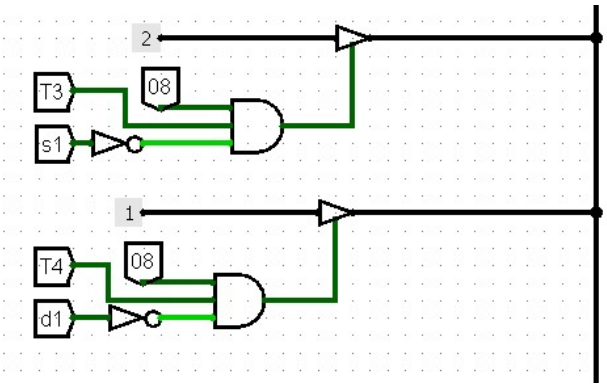


Figure 86: Funsel1 for the 0x08 function

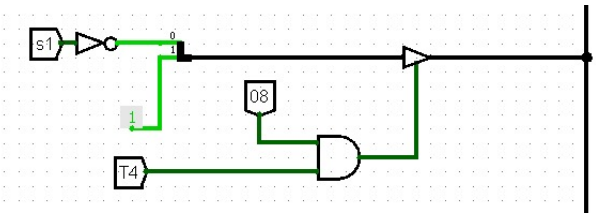


Figure 87: MuxA for the 0x08 function

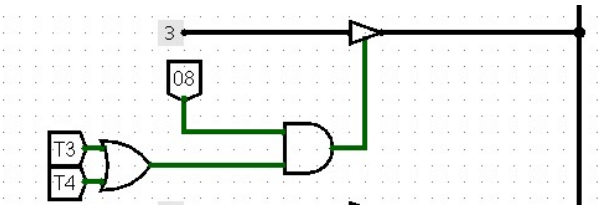


Figure 88: MuxB for the 0x08 function

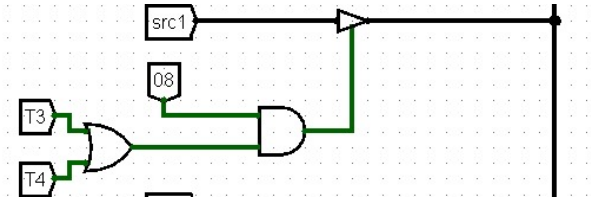


Figure 89: OutC for the 0x08 function

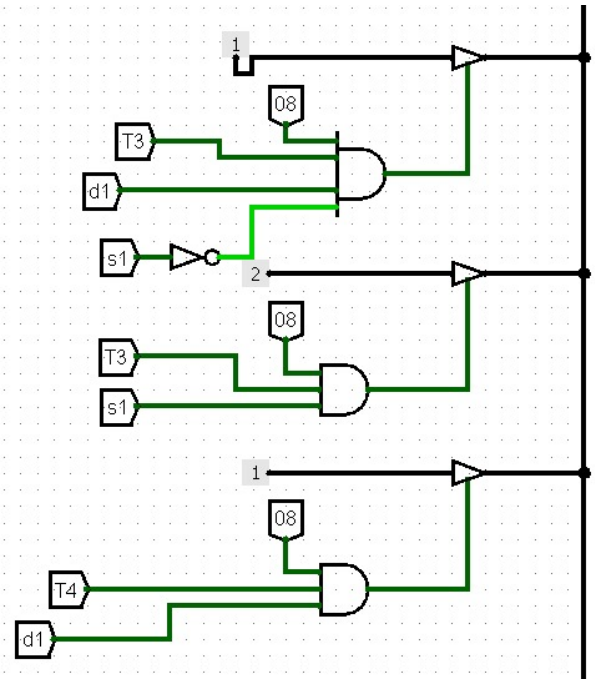


Figure 90: Funsel2 for the 0x08 function

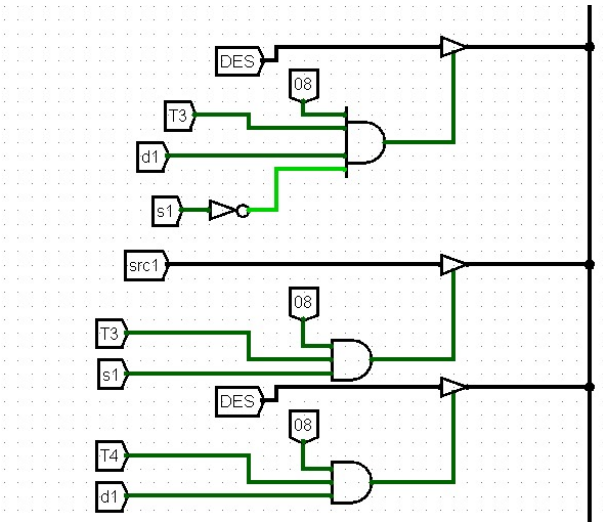


Figure 91: Regsel2 for the 0x08 function

2.2.6 0x09 AND operation

It processes the value it gets from two different source registers with AND gate and loads it to the destination register.

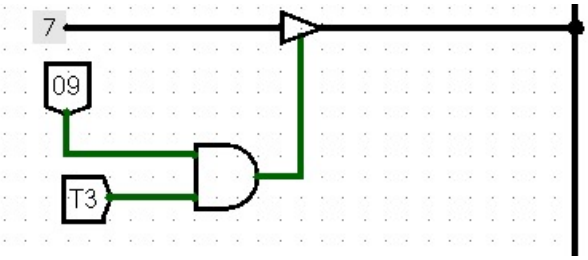


Figure 92: FunselALU for the 0x09 function

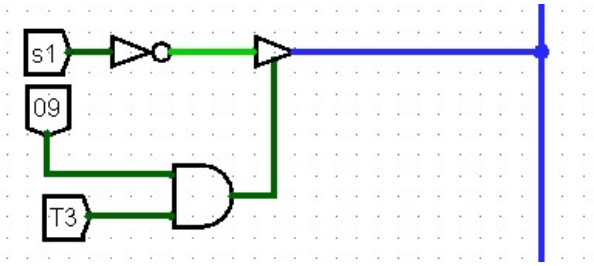


Figure 93: MuxCSel for the 0x09 function

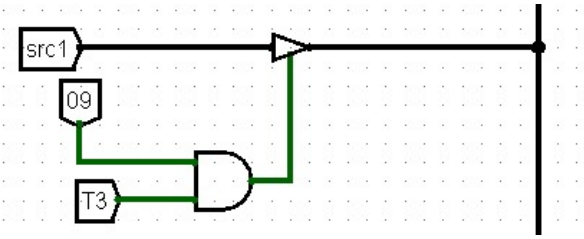


Figure 94: OutA for the 0x09 function

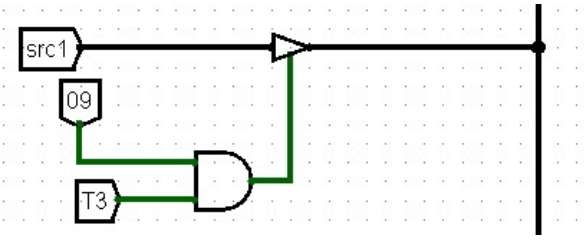


Figure 95: OutB for the 0x09 function

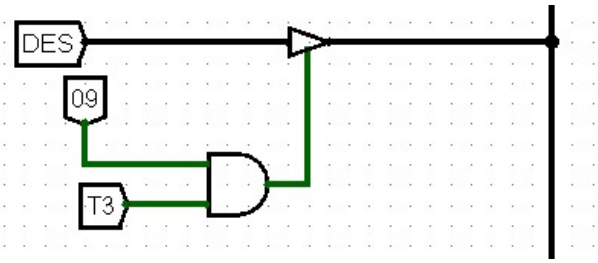


Figure 96: RegSel1 for the 0x09 function

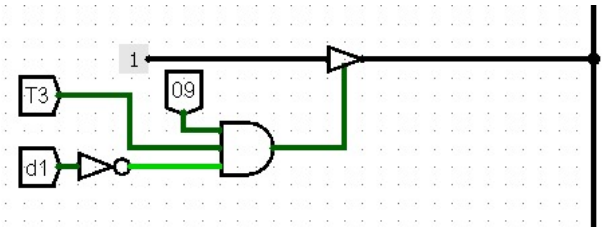


Figure 97: Funsel for the 0x09 function

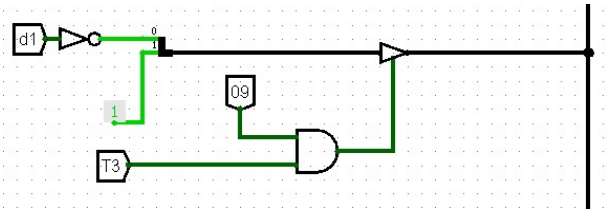


Figure 98: MuxA for the 0x09 function

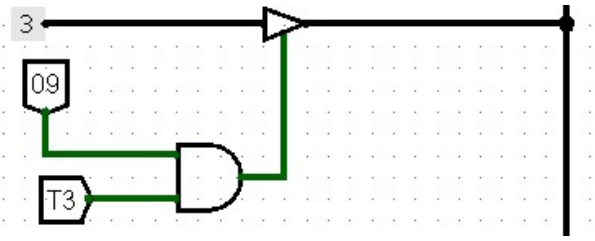


Figure 99: MuxB for the 0x09 function

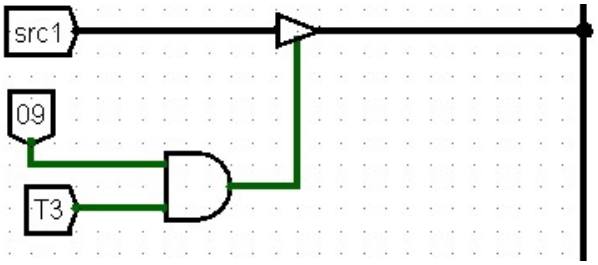


Figure 100: OutC for the 0x09 function

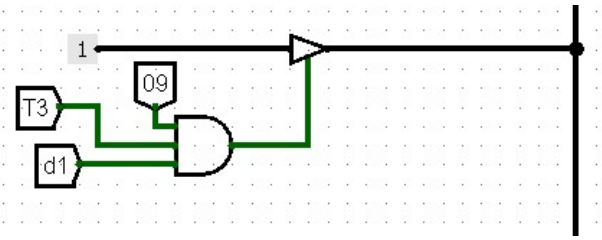


Figure 101: Funsel2 for the 0x09 function

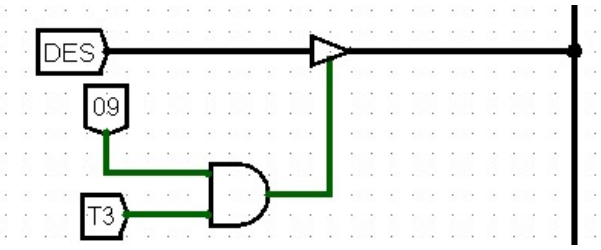


Figure 102: Regsel2 for the 0x09 function

2.2.7 0x0A OR operation

In this operation we had to connect two registers of register file into a or gate and after that we had to write its output to the destination register. For taking two values from register file, firstly we sent to OutASel and OutBSel src1 and src2. After that for applying or operation in ALU we have to send 1000 value as the Funsel input of the ALU. After that, we selected 11 in the MuxA and we selected 11 in the MuxB. While our funsel and regsel values choose the destination register and load operation. Our bus systems for these function were as follows:

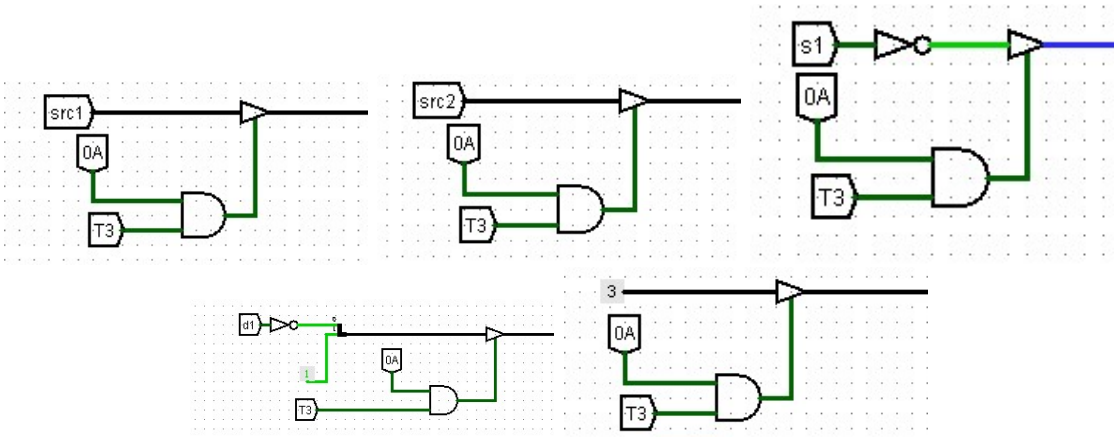


Figure 103: NOT operation bus systems

2.2.8 0x0B NOT OPERATION

In this part of the assignment, we had to take negation of one value and transfer it to another register. The first thing that we had done for realising this purpose, was selecting the source register according to src1 input. After that we implemented negation operation in ALU by sending its Funsel 0010 value which we can observe the negation of A input of the ALU at the output, after that we select the value of the MuxA with respect to most significant bit of the source register. And our funsel and regsel valueas are the same with the load operation at the previous function. Our bus systems are as follows for this function:

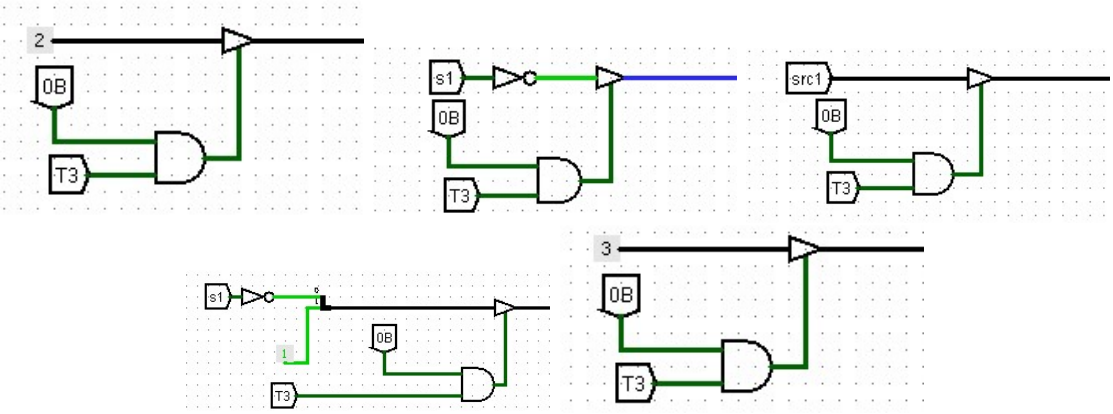


Figure 104: LSL operation bus systems

2.2.9 0X0C LSL OPERATION

In this part of the assignment, we had to take negation of one value and transfer it to another register. The first thing that we had done for realising this purpose, was selecting the source register according to src1 input. After that we implemented negation operation in ALU by sending its Funsel 1010 value which we can observe the LSL of A input of the ALU at the output, after that we select the value of the MuxA with respect to most significant bit of the source register. And our funsel and regsel valueas are the same with the load operation at the previous function. Our bus systems are as follows for this function:

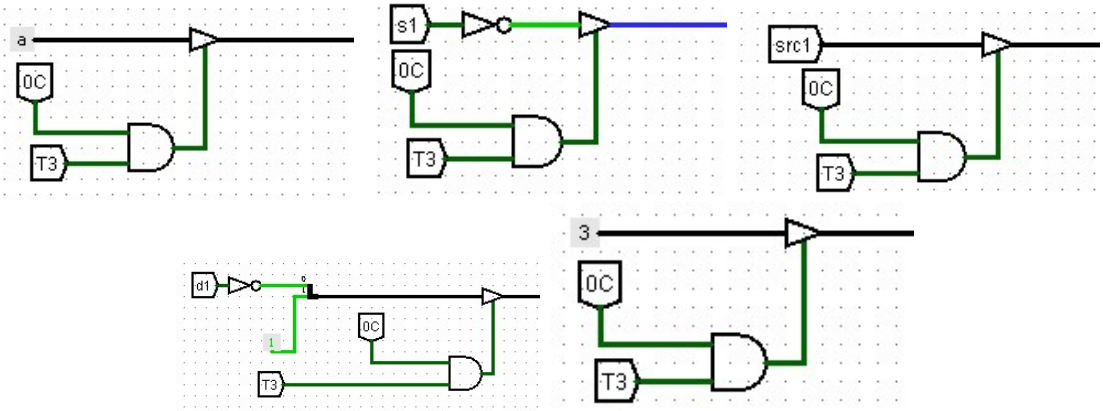


Figure 105: LSR operation bus systems

2.2.10 0X0C LSR OPERATION

In this part of the assignment, we had to take negation of one value and transfer it to another register. The first thing that we had done for realising this purpose, was selecting the source register according to src1 input. After that we implemented negation operation in ALU by sending its Funsel 1011 value which we can observe the LSR of A input of the ALU at the output, after that we select the value of the MuxA with respect to most significant bit of the source register. And our funsel and regsel valueas are the same with the load operation at the previous function. Our bus systems are as follows for this function:

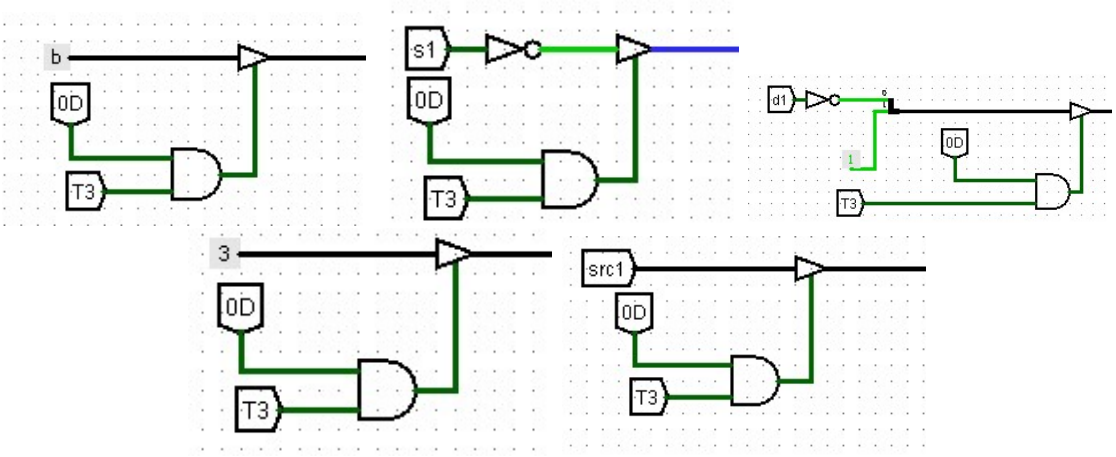


Figure 106: Caption

3 RESULTS

In this assignment, we observed a lot of result since we had to control every case. We will share some of them at the result section of the report. Firstly we want to share example program which is given at the last page of the assignment:

```
        ORG  0x20          # Write the program starting from the address 0x20
        LD R0 IM 0x05      # R0 is used for iteration number
        LD R1 IM 0x00      # R1 is used to store total
        LD R2 IM 0xA0
        MOV AR R2          # AR is used to track data adrress: starts from 0xA0
LABEL:  LD R2 D            # R2 <- M[AR] (AR = 0xA0 to 0xA4)
        INC AR AR          # AR <- AR + 1 (Next Data)
        ADD R1 R1 R2       # R1 <- R1 + R2 (Total = Total + M[AR])
        DEC R0 R0          # R0 <- R0 – 1 (Decrement Iteration Counter)
        BNE IM LABEL       # Go back to LABEL if Z=0 (Itertaion Counter > 0)
        INC AR AR          # AR <- AR + 1 (Total will be written to 0xA5)
        ST  R1 D           # M[AR] <- R1 (Store Total at 0xA5)
```

Figure 107: Example program

In this program, we had to sum up 5 values and load to a memory address. For realising this operation, we converted instructions to the binary and after that, we converted to hex and wrote to the memory, after that we run the program and we obtained expected results:

We saw that 01+02+03+04 = 0A was written at the address that we want to load.

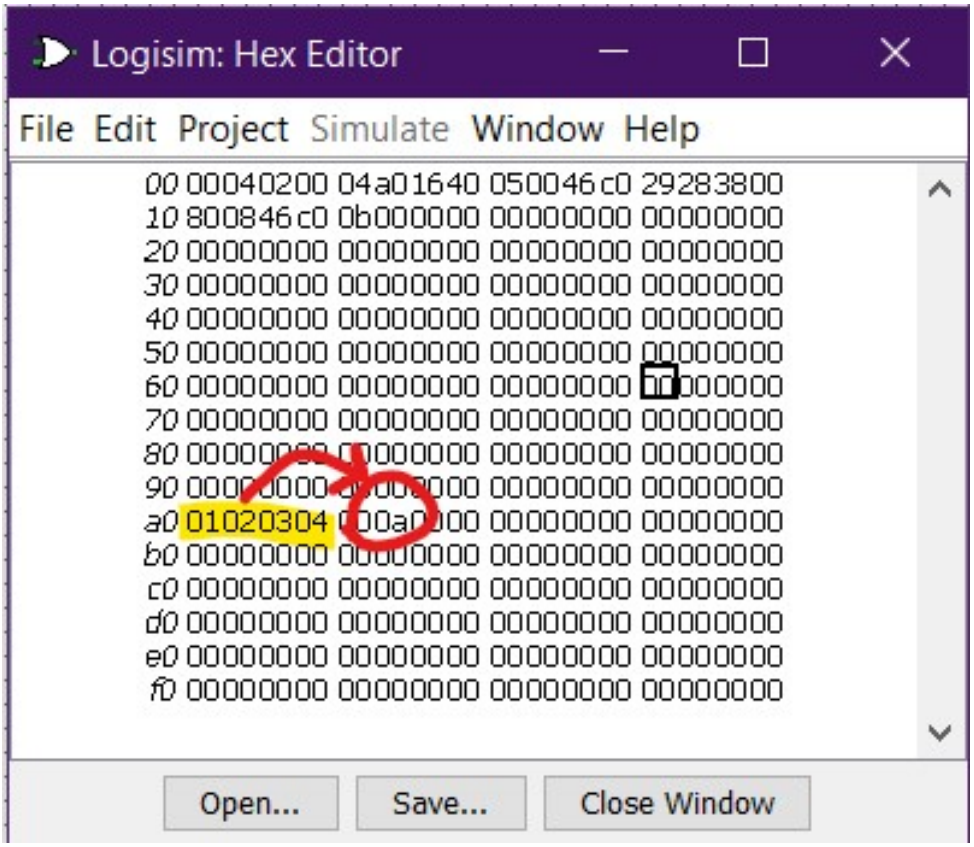


Figure 108: Example program- memory i/o

For another test-case; we load the memory with these instructions:

LD R0 IM 0X07

MOV SP R0

PUL R1

And we wrote to the memory the hex code of these instruction and some AA values to 07 and 08 addresses:

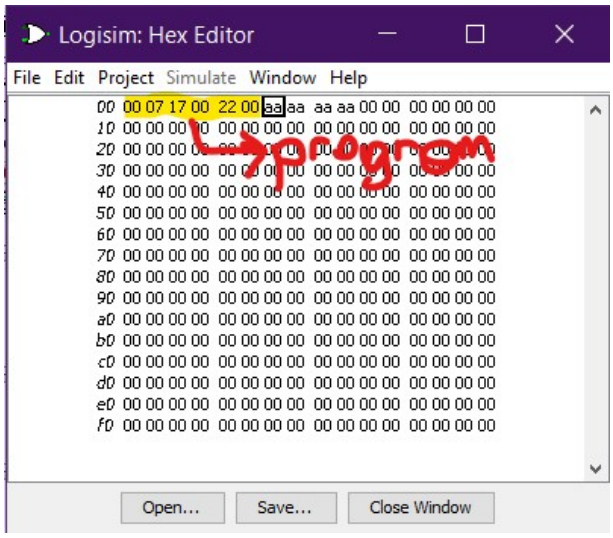


Figure 109: Example2 program- memory i/o

After we ran the program, we obtained that with first instruction R0 was loaded to 07:

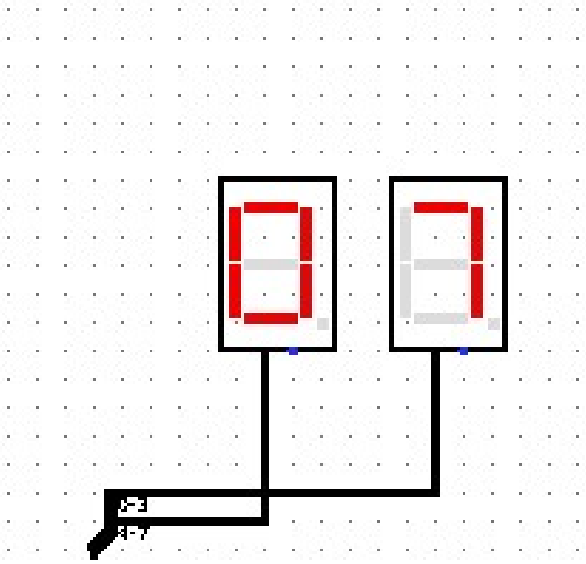


Figure 110: Value of R1

With the second instruction, SP is loaded with the value 07:

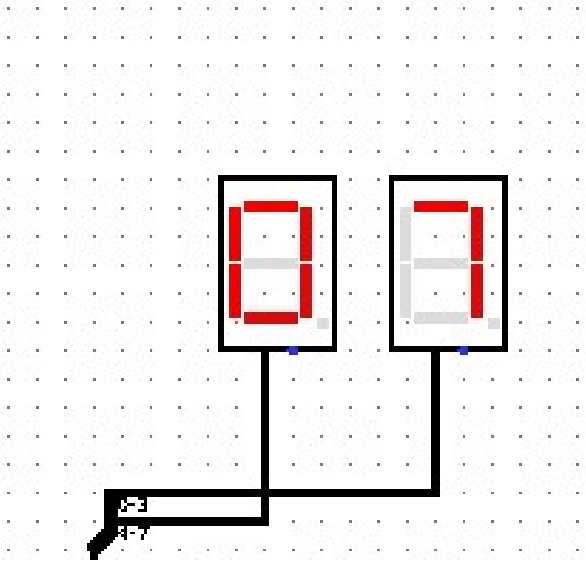


Figure 111: Value of SP

And the last instruction of the program make SP=08 and take the value of that address from memory and write it into R1.

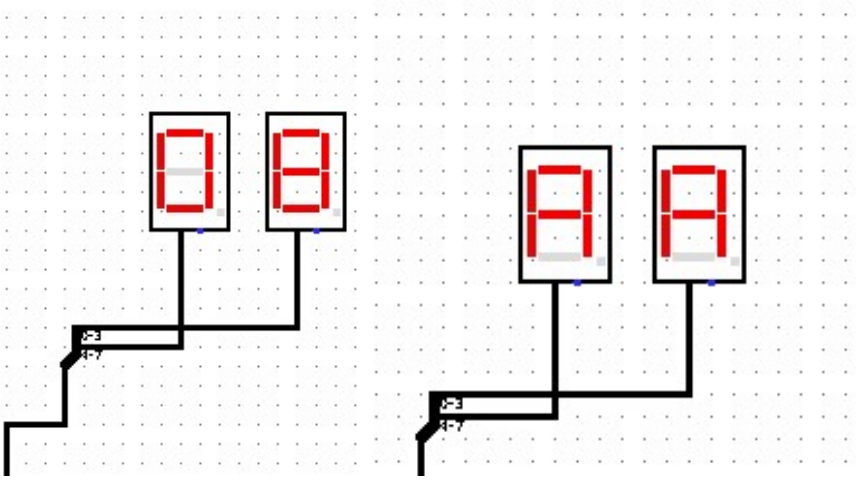


Figure 112: SP+1 and R1

4 DISCUSSION

It directs the operation of the other units by providing timing and control signals. Most computer resources are managed by the CU. It directs the flow of data between the CPU and the other devices. In modern computer designs, the control unit is typically an internal part of the CPU with its overall role and operation unchanged since its introduction. The control unit in this project can perform 19 different operations. What these are is described in detail in the previous section.

5 CONCLUSION

To summarize, we had practised the workflow of basic computer in this project. We learnt the implementations of fetch and decode operations. We worked with different instructions types. All of these works were a little bit hard but it was so funny to learn them and apply them. Actually, it was a long assignment and we had to think about it much, but finally when we see the system was running well and we were taking the expected values, it was worth it. It was well-prepared and instructive assignment we think.