ISTANBUL TECHNICAL UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT REPORT

PROJECT NO: 4
Due Date: 14.07.2020
GROUP NO: G2

GROUP MEMBERS:

150180008 : Lal Verda Cakir
150170909 : Elaa Jamazi
150140009 : Serhat Vural
150180038 : Karahan Sezer

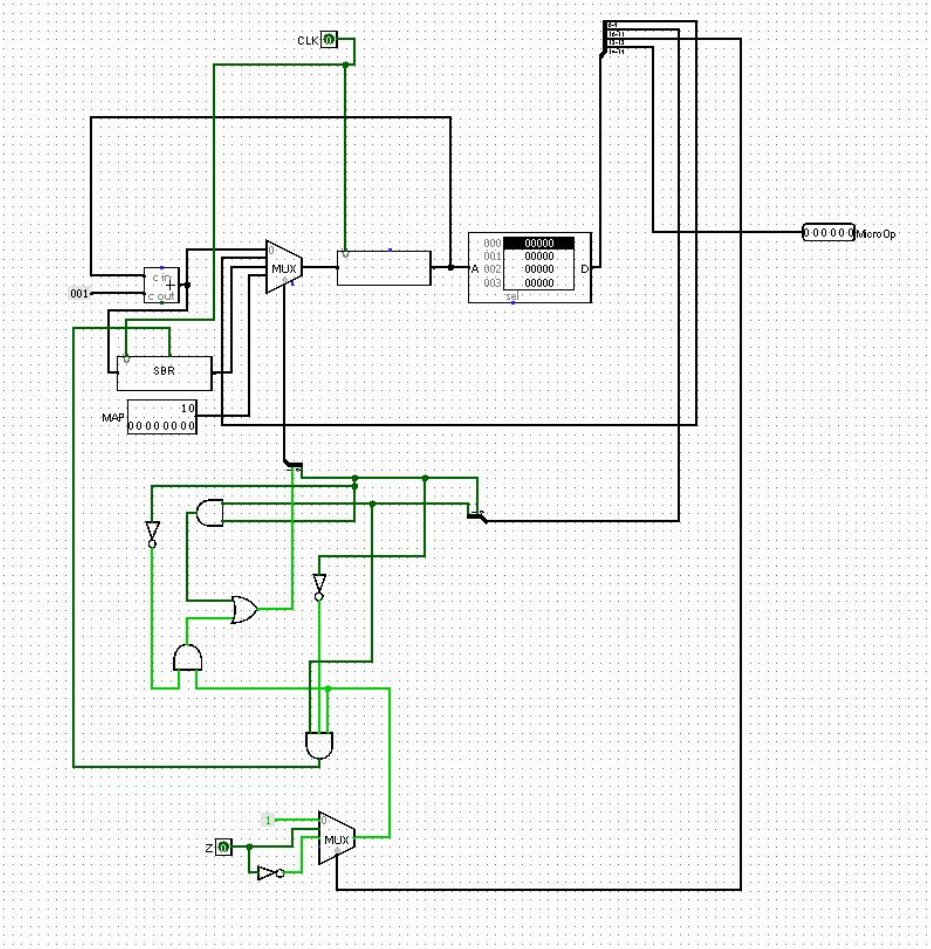
1. Introduction:

In project 4, we implemented micro-programmed version of the previous project. This time, we replaced the combinational Logic Circuits with micro-programs which is a program stored in memory. These micro-programs are a collection of microinstructions. This micro-instructions generates our control signals.t. Our microprogrammed control unit takes in 2 types of instruction. Depending on the instruction input, it performs the suitable Mapping and Branching to be able to select the correct micro-program to deliver the correct task provided by the instruction. We have a total of 58 micro-programs. Since the content of the main circuit is the same as other projects we will focus on explaining the functionalities of our Control Unit.

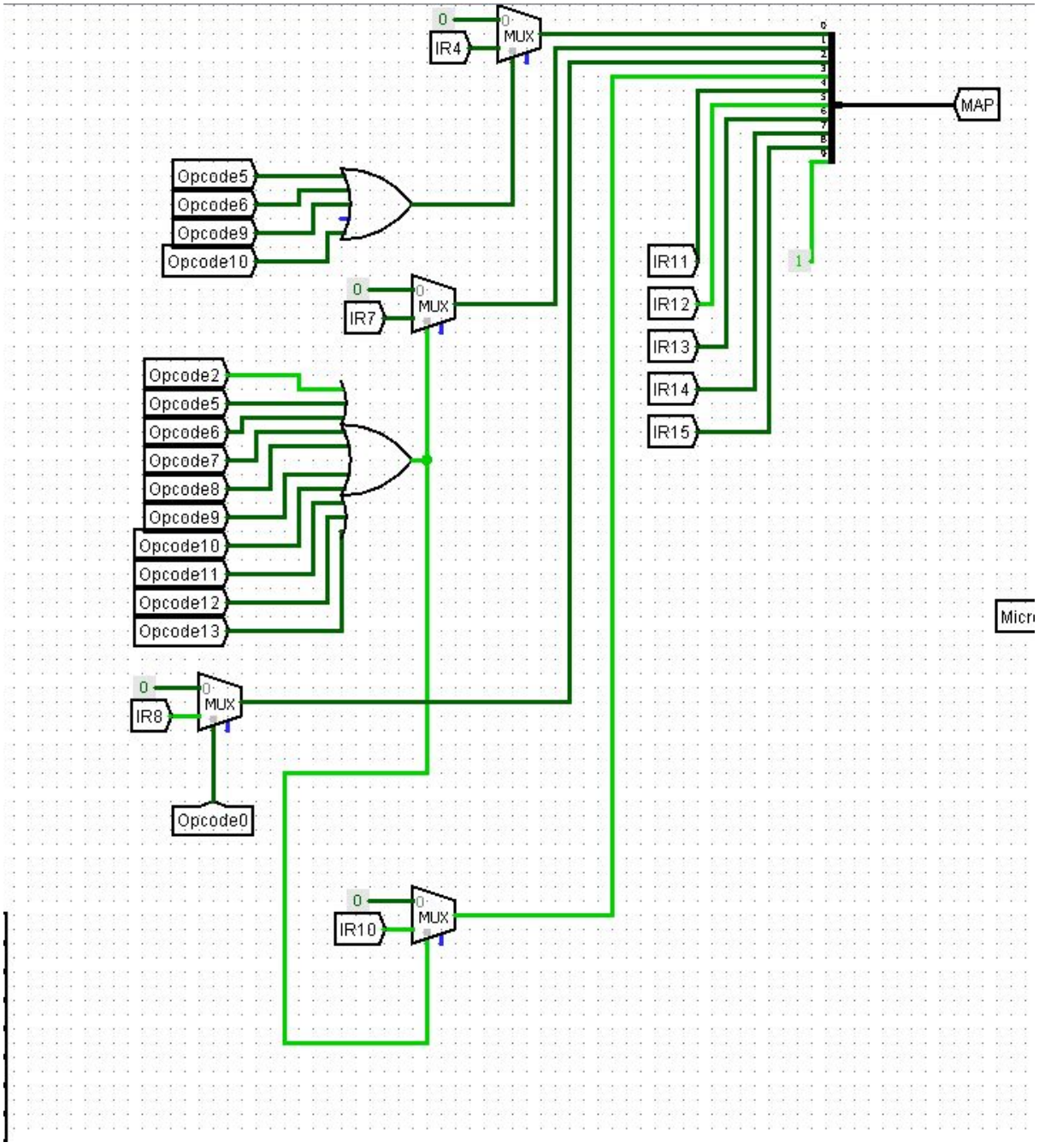
2. Project Parts:

2.1 Sequencer:

Our sequencer performs the operation necessary to present us with micro-instruction. This is what the content of the sequencer looks like. It increments the control address register, branches conditionally and unconditionally, performs and has a facility for subroutine call and return.



To implement the sequencer we used Multiplexers (condition and branch control) , an adder to increment the control address register and a ROM to function as control Storage. Afterwards, in the main circuit of our Control Unit, the outputs of the sequencer are processed through a decoder.
MAP comes outside of the sequencer. Our mapping logic located in the control unit.



In the lecture slides in the mapping there was buffer 0's but for this projects doing it like that would take twice as much of a rom we mapped in an efficient wat.

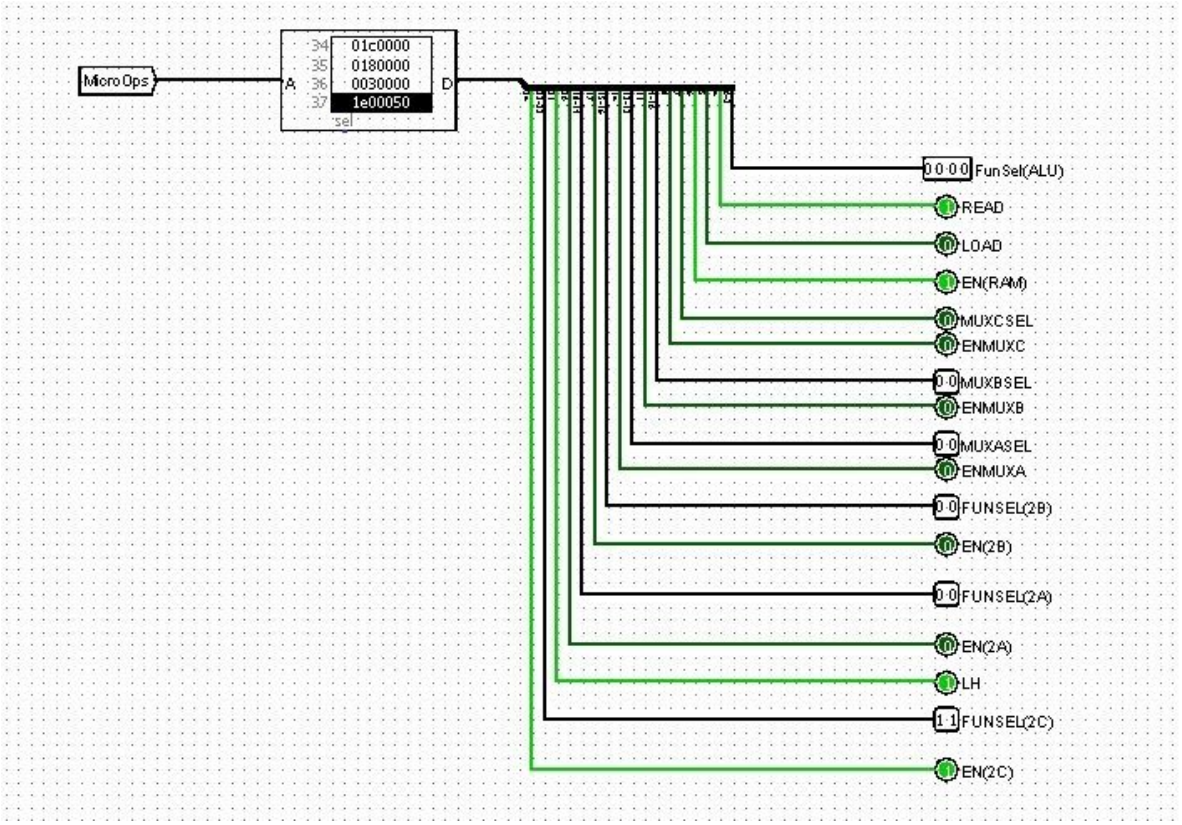
We placed out second half of the rom with this micro-programs because of this in the mapping we have 1 as a most significant bit. And then we have out opcode. So the operation we are going to execute can change regarding the value of IR10, IR8, IR7, and IR4 in

different opcodes. So If this situation effects our choice we are putting that IR(no) to the mapping logic.

Because of this logic not space between the microprograms are not the same but this is not an issue. Our instructions all fit there. For more info about mapping logic contact Lal Verda Çakır - cakirl18@itu.edu.tr

2.2 Constant control signals :

There are some fixed control signals during which our CU outputs are known already For example for a one microop some signals are constant but some of the signals values comes from the instruction or something else, so we used a rom to hold this constant control signals.

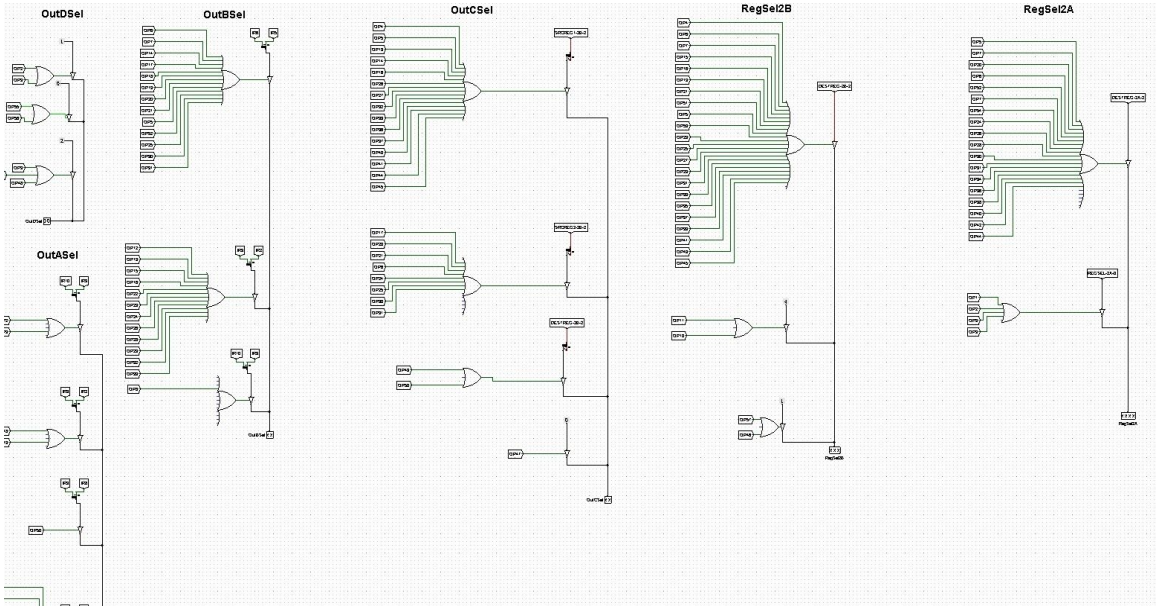


These tables show which control signals are fixed (EN2C, FUN2C, L/H, EN2A, FUN2A, EN2B, FUN2B, ENMUXA, MUXA, ENMUXB, MUXB, ENMUXC, MUXC, ENRAM, LOAD, READ, FUNALU), and their corresponding values for the designated micro operations(both codes of the microoperations and their meanings):

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
		ID	MICRO-OP NAME	EN2C	FUN2C	L/H	EN2A	FUN2A	EN2B	FUN2B	ENMUXA	MUXA	ENMUXB	MUXB	ENMUXC	MUXC	ENRAM	LOAD	READ	FUNALU		HEX
RX <-- IROUT(0-7)	000001	1	IRtRF	0	00	0	1	01	0	00	1	00	0	00	0	0	0	0	0	0000	00001010001000000000000	144000
RX <--M[AR]	000010	2	MARtRF	0	00	0	1	01	0	00	1	01	0	00	0	0	1	0	1	0000	00001010001010000001010000	145050
M[AR] <-- RX	000011	3	RFtMAR	0	00	0	0	00	0	00	0	00	0	00	1	1	1	1	0	0000	00000000000000001111000000	0001E0
ARF <-- ARF	000100	4	ARFtARF	0	00	0	0	00	1	01	0	00	0	00	0	0	0	0	0	0000	00000001010000000000000000	028000
ARF <-- RF	000101	5	RFtARF	0	00	0	1	01	0	00	1	10	0	00	0	0	0	0	0	0000	00001010001100000000000000	146000
RF <-- ARF	000110	6	ARFtRF	0	00	0	0	00	1	01	0	00	1	11	0	0	0	0	0	0001	00000001010001110000000001	028E01
RF <-- RF	000111	7	RFtRF	0	00	0	1	01	1	01	1	11	1	11	0	0	0	0	0	0001	00001011011111111000000001	16FE01
M[SP] <-- RX	001000	8	RFtMSP	0	00	0	0	00	0	00	0	00	0	00	0	0	1	1	0	0001	00000000000000000001100001	000061
RX <-- M[SP]	001001	9	MSPtRF	0	00	0	1	01	0	00	1	01	0	00	0	0	1	0	1	0000	00001010001010000001010000	145050
SP <-- SP-1	001010	10	DECSP	0	00	0	0	00	1	11	0	00	0	00	0	0	0	0	0	0000	00000001110000000000000000	038000
SP <-- SP+1	001011	11	INCSP	0	00	0	0	00	1	10	0	00	0	00	0	0	0	0	0	0000	00000001100000000000000000	030000
RF <-- RF+ RF	001100	12	RFpRFtRF	0	00	0	1	01	0	00	1	11	0	00	1	1	0	0	0	0100	00001010001110001100000100	147184
RF <-- RF+ ARF	001101	13	RFpARFtRF	0	00	0	1	01	0	00	1	11	0	00	0	0	0	0	0	0100	00001010001110000000000100	147004
RF <-- ARF+ RF	001110	14	ARFpRFtRF	0	00	0	1	01	0	00	1	11	0	00	0	0	0	0	0	0100	00001010001110000000000100	147004
ARF <-- RF+ RF	001111	15	RFpRFTtARF	0	00	0	0	00	1	01	0	00	1	11	1	1	0	0	0	0100	00000001010001111110000100	028F84
ARF <-- ARF+ RF	010000	16	ARFpRFtRF	0	00	0	0	00	1	01	0	00	1	11	0	0	0	0	0	0100	00000001010001110000000100	028E04
ARF <-- RF+ ARF	010001	17	RFpARFtARF	0	00	0	0	00	0	00	0	00	0	00	0	0	0	0	0	0100	00000000000000000000000100	000004
RF <-- RF - RF	010010	18	RFmRFtRF	0	00	0	1	01	0	00	1	11	0	00	1	1	0	0	0	0110	00001010001110001100000110	147186
ARF <-- RF - RF	010011	19	RFmRFtAF	0	00	0	0	00	1	01	0	00	0	00	1	1	0	0	0	0110	00000001010000001100000110	028186
RF <-- ARF -RF	010100	20	ARFmRFtRF	0	00	0	1	01	0	00	1	11	0	00	0	0	0	0	0	0110	00001010001110000000000110	147006
ARF <-- ARF - RF	010101	21	ARFmRFtARF	0	00	0	0	00	1	01	0	00	0	00	0	0	0	0	0	0110	00000001010000000000000110	028006
RF <-- RF AND RF	010110	22	RFandRFtRF	0	00	0	1	01	0	00	1	11	0	00	1	1	0	0	0	0111	00001010001110001100000111	147187
ARF <-- RF AND RF	010111	23	RFandRFtARF	0	00	0	0	00	1	01	0	00	1	11	1	1	0	0	0	0111	00000001010001111110000111	028F87
RF <-- ARF AND RF	011000	24	ARFandRFtRF	0	00	0	1	01	0	00	1	11	0	00	0	0	0	0	0	0111	00001010001110000000000111	147007
ARF <-- ARF AND ARF	011001	25	ARFandARFtARF	0	00	0	0	00	1	01	0	00	1	11	0	0	0	0	0	0111	00000001010001110000000111	0028E07
RF <--RF AND ARF	011010	26	RFandARFtRF	0	00	0	1	01	0	00	1	11	0	00	0	0	0	0	0	0111	00001010001110000000000111	0147007
ARF <-- RF AND ARF	011011	27	RFandARFtARF	0	00	0	0	00	1	01	0	00	1	11	0	0	0	0	0	0111	00000001010001110000000111	028E07
RF <-- RF OR RF	011100	28	RForRFtRF	0	00	0	1	01	0	00	1	11	0	00	1	1	0	0	0	1000	00001010001110001100001000	147188
ARF <-- RF OR RF	011101	29	RForRFtARF	0	00	0	0	00	1	01	0	00	1	11	1	1	0	0	0	1000	00000001010001111110001000	028F88
RF <-- RF OR ARF	011110	30	RForARFtRF	0	00	0	1	01	0	00	1	11	0	00	0	0	0	0	0	1000	00001010001110000000001000	147008
ARF <-- RF OR ARF	011111	31	RForARFtARF	0	00	0	0	00	1	01	0	00	1	11	0	0	0	0	0	1000	00000001010001110000001000	028E08
RF <-- ARF OR RF	100000	32	ARForRFtRF	0	00	0	1	01	0	00	1	11	0	00	0	0	0	0	0	1000	00001010001110000000001000	0147008
ARF <-- ARF OR RF	100001	33	ARForRFtARF	0	00	0	0	00	1	01	0	00	1	11	0	0	0	0	0	1000	00000001010001110000001000	0028E08
RF <-- NOT RF	100010	34	notRFtRF	0	00	0	1	01	0	00	1	11	0	00	1	1	0	0	0	0010	00001010001110001100000010	0147182
ARF <-- NOT RF	100011	35	notRFtARF	0	00	0	0	00	1	10	0	00	1	11	1	1	0	0	0	0010	00000001100000111110000010	0030F82
RF <-- NOT ARF	100100	36	notARFtRF	0	00	0	1	01	0	00	1	11	0	00	0	0	0	0	0	0010	00001010001110000000000010	0147002
ARF <-- NOT ARF	100101	37	notARFtARF	0	00	0	0	00	1	01	0	00	1	11	0	0	0	0	0	0010	00000001010001110000000010	0028E02
RF <-- LSL RF	100110	38	lslRFtRF	0	00	0	1	01	0	00	1	11	0	00	1	1	0	0	0	1010	00001010001110001100001010	014718A
ARF <-- LSL RF	100111	39	lslRFtARF	0	00	0	0	00	1	01	0	00	1	11	1	1	0	0	0	1010	00000001010001111110001010	0028F8A
RF <-- LSL ARF	101000	40	lslARFtRF	0	00	0	1	01	0	00	1	11	0	00	0	0	0	0	0	1010	00001010001110000000001010	014700A
ARF <-- LSL ARF	101001	41	lslARFtARF	0	00	0	0	00	1	01	0	00	1	11	0	0	0	0	0	1010	00000001010001110000001010	0028E0A
RF <-- LSR RF	101010	42	lsrRFtRF	0	00	0	0	00	1	01	1	11	0	00	1	1	0	0	0	1011	00000001011110001100001011	002F18B
ARF <-- LSR RF	101011	43	lsrRFtARF	0	00	0	0	00	1	01	0	00	1	11	0	0	0	0	0	1011	00000001010001110000001011	0028E0B
RF <-- LSR ARF	101100	44	lsrARFtRF	0	00	0	1	01	0	00	1	11	0	00	0	0	0	0	0	1011	00001010001110000000001011	014700B
ARF <-- LSR ARF	101101	45	lsrARFtARF	0	00	0	0	00	1	01	0	00	1	11	0	0	0	0	0	1011	00000001010001110000001011	0028E0B
PC <-- IROut(0,7)	101110	46	IRtPC	0	00	0	0	00	1	01	0	00	1	01	0	0	0	0	0	0000	00000001010001010000000000	0028A0C
M[SP] <-- PC	101111	47	PCtMSP	0	00	0	0	00	0	00	0	00	0	00	0	0	1	1	0	0000	00000000000000000001100000	0000060
PC <-- M[SP]	110000	48	MSPtPC	0	00	0	0	00	1	01	0	00	1	10	0	0	1	0	1	0000	00000001010001100010100000	0028C50
PASSALU ARF	110001	49	PASSALUARF	0	00	0	0	00	0	00	0	00	0	00	0	0	0	0	0	0000	00000000000000000000000000	0000000
PASSALU RF	110010	50	PASSALURF	0	00	0	0	00	0	00	0	00	0	00	1	1	0	0	0	0000	00000000000000001100000000	0000180
ARF <-- ARF – 1	110011	51	decARF	0	00	0	0	00	1	11	0	00	0	00	0	0	0	0	0	0000	00000001110000000000000000	0038000
RF <-- RF -1	110100	52	decRF	0	00	0	1	11	0	00	0	00	0	00	0	0	0	0	0	0000	00001110000000000000000000	01C0000
ARF <-- ARF +1	110101	53	incARF	0	00	0	0	00	1	10	0	00	0	00	0	0	0	0	0	0000	00000001100000000000000000	0030000
RF <-- RF + 1	110110	54	incRF	0	00	0	1	10	0	00	0	00	0	00	0	0	0	0	0	0000	00001100000000000000000000	0180000
IR(8,15) <-- M[PC]	110111	55	FETCHHIGH	1	11	1	0	00	0	00	0	00	0	00	0	1	0	1	0	0000	11110000000000000001010000	1E00050
IR(0,7) <-- M[PC]	111000	56	FETCHLOW	1	11	0	0	00	0	00	0	00	0	00	0	0	1	0	1	0000	11100000000000000001010000	1C00050
PC <-- PC+1	111001	57	incPC	0	00	0	0	00	1	10	0	00	0	00	0	0	0	0	0	0000	00000001100000000000000000	0030000

2.3 Other Control Signals:

As for the remaining control signals, we designed the appropriate circuitry as shown in the picture below. This way we supply the correct control signals depending on the micro instruction (OP1- 57) supplied from our decoder.



3. Results:

In this project we implemented a machine that can perform a set of operations, using the instructions that are given to it, after processing them through a microprogrammed Control Unit. Instructions are written by the user to RAM and the system reads these instructions and passes them to the control unit, then the control unit with its memory generates all the control signals required to execute the instruction set correctly.
The table below sums up all the operations performed by our system. It contains all the possible microinstructions our CU can perform and it also show the needed combinations of its parts. The explanation for the Microoperations' names are including in the previous part under the section 2.2.

For our condition for branching we have:

CD	Condition	Symbol
00 01 10	Always = 1 Z = 0 Z = 1	U Z0 Z1

Label	Micro ops	CD	BR	AD
FETCH	FETCHHIGH	U	JMP	NEXT
	INCPC	U	JMP	NEXT
	FETCHLOW	U	JMP	NEXT
	INCPC	U	JMP	NEXT
	NOP	U	MAP	
OPCODE0	IRtRF	U	JMP	FETCH
	MARtRF	U	JMP	FECTH
OPCODE1	RFtMAR	U	JMP	FETCH

OPCODE2				
IR7=0 IR10=0	RFtRF	U	JMP	FETCH
IR7=1 IR10=0	RFtARF	U	JMP	FETCH
IR7=0 IR10=1	ARFtRF	U	JMP	FETCH
IR7=1 IR10=1	ARFtARF	U	JMP	FETCH
OPCODE3	RFtMSP	U	JMP	NEXT
	DECSP	U	JMP	FETCH
OPCODE4	RFtMSP	U	JMP	NEXT
	DECSP	U	JMP	FETCH
OPCODE5				
IR10=0 IR7=0 IR4=0	RFpRFtRF	U	JMP	FETCH
IR10=0 IR7=0 IR4=1	RFpARFtRF	U	JMP	FETCH
IR10=0 IR7=1 IR4=0	ARFpRFtRF	U	JMP	FETCH
IR10=1 IR7=0 IR4=0	RFpRFtARF	U	JMP	FETCH
IR10=1 IR7=0 IR4=1	RFpARFtARF	U	JMP	FETCH
IR10=1 IR7=1 IR4=0	ARFpRFtRF	U	JMP	FETCH

OPCODE6				
IR10=0 IR7=0 IR4=0	RFmRFtRF	U	JMP	FETCH
	ARFmRFtRF	U	JMP	FETCH
IR10=0 IR7=0 IR4=1	RFmRFtARF	U	JMP	FETCH
IR10=1 IR7=0 IR4=0	ARFmRFtARF	U	JMP	FETCH
IR10=1 IR7=0 IR4=1				
OPCODE7				
R10=0 IR7=0	RFtRF	U	JMP	NEXT
	DECRF	U	JMP	PASSALURF
IR10=0 IR7=1	ARFtRF	U	JMP	NEXT
	DECRF	U	JMP	PASSALURF
IR10=1 IR7=0		U	JMP	NEXT
	RFtARF	U	JMP	PASSALUARF
IR10=1 IR7=1	DECARF	U	JMP	NEXT
	ARFtARF	U	JMP	PASSALUARF
	DECARF			

OPCODE8				
IR10=0 IR7=0	RFtRF	U	JMP	NEXT
	INCRF	U	JMP	PASSALURF
IR10=0 IR7=1		U	JMP	NEXT
	ARFtRF	U	JMP	PASSALURF
	INCRF	U	JMP	NEXT
IR10=1 IR7=0		U	JMP	PASSALUARF
	RFtARF	U	JMP	NEXT
	INCARF	U	JMP	PASSALUARF
IR10=1 IR7=1				
	ARFtARF			
	INCARF			

OPCODE9				
IR10=0 IR7=0 IR4 = 0	RFandRFtRF	U	JMP	FETCH
IR10=0 IR7=0 IR4 = 1	ARFandRFtRF	U	JMP	FETCH
	RFandARFtRF	U	JMP	FETCH
IR10=0 IR7=1 IR4 = 0	RFandRFtARF	U	JMP	FETCH
IR10=1 IR7=0 IR4 = 0	ARFandARFtARF	U	JMP	FETCH
IR10=1 IR7=0 IR4 = 1	RFandARFtARF	U	JMP	FETCH
IR10=1 IR7=1 IR4 = 0				

OPCODE10				
IR10=0 IR7=0 IR4 = 0	RForRFtRF	U	JMP	FETCH
IR10=0 IR7=0 IR4 = 1	RForARFtRF	U	JMP	FETCH
	ARForRFtRF	U	JMP	FETCH
IR10=0 IR7=1 IR4 = 0	RForRFtARF	U	JMP	FETCH
IR10=1 IR7=0 IR4 = 0	RForARFtARF	U	JMP	FETCH
IR10=1 IR7=0 IR4 = 1	ARForRFtARF	U	JMP	FETCH
IR10=1 IR7=1 IR4 = 0				
OPCODE11				
IR10=0 IR7=0	notRFtRF	U	JMP	FETCH
IR10=0 IR7=1	notARFtRF	U	JMP	FETCH
IR10=1 IR7=0	notRFtARF	U	JMP	FETCH
IR10=1 IR7=1		U	JMP	FETCH

	notARFtARF			
OPCODE12 IR10=0 IR7=0 IR10=0 IR7=1 IR10=1 IR7=0 IR10=1 IR7=1	lslRFtRF lslARFtRF lslRFtARF lslARFtARF	U U U U	JMP JMP JMP JMP	FETCH FETCH FETCH FETCH
OPCODE13 IR10=0 IR7=0 IR10=0 IR7=1 IR10=1 IR7=0 IR10=1 IR7=1	lsrRFtRF lsrARFtRF lsrRFtARF lsrARFtARF	U U U U	JMP JMP JMP JMP	FETCH FETCH FETCH FETCH
OPCODE14	IRtPC	U	JMP	FETCH

OPCODE15	NOP NOP IRtPC	Z0 U U	JMP JMP JMP	NEXT FETCH FETCH
OPCODE16	IRtPC NOP NOP	Z1 U U	JMP JMP JMP	NEXT FETCH FETCH
OPCODE17	PCtMSP DECSP IRtPC	U U U	JMP JMP JMP	NEXT FETCH FETCH
OPCODE18	INCPC MSPtPC	U U	JMP JMP	NEXT FETCH
PASSALURF	PASSALURF	U	JMP	FETCH
PASSALUARF	PASSALUARF	U	JMP	FETCH

4. Discussion:

Our design process began with dividing the whole process into smaller parts. After deconstructing the OPCODE's we figure out which micro instruction to use. Using the decoder that we created, we get the corresponding micro-operations. After deconstructing the micro-operations we made the circuit that produces the OUT selectors, REG selectors and other necessary outputs.
Here are the instructions break-down into clock signals

T0:

IR(8-15) ← M[PC] RegSel(2B): 000; OutDSel: 00; Read: 1; Load: 0; Enable(2C): 1; L/H:0; FunSel(2C):11;

T1:

PC ← PC + 1 FunSel(2B): 10; RegSel(2B):001; OutCSel: 00; Enable(2C): 0;

T2:

IR(0-7) ← M[PC] OutDSel: 00; RegSel(2B): 0000; Read: 1; Load: 0; Enable(2C): 1; L/H: 1; FunSel(2C): 11;

T3:

PC ← PC + 1 FunSel(2B): 10; RegSel(2B): 001; OurCSel: 00; Enable(2C): 0;

Opcode0: Rx ← Value

Note: Addressing mode IM, D

IR8 = 0

Update:

Opcode0T4: **Rx ← IROut(0,7)**

MuxASel = 00
FunSel(2A) = 01
RegSel(2A) = REGSEL-2A-0

IR8 = 1

Opcode0T4: **Rx ← M[AR]**

OutDSel = 01;
Read = 1;

Load = 0;

MuxASel = 01;
OutASel = IR(9,10);
FunSel(2A) = 01;
RegSel2A = REGSEL-2A-0;

Opcode1: Value ← Rx

Note: Addressing mode D

Opcode1T4: **M[AR] ← Rx**

RegSel(2A) = regsel-2a-0;
OutASel = IR(9,10);
MuxCSel = 1;
FunSel(ALU) = 0000;
OutDSel = 01;
Load = 1;
Read = 0;

Opcode2: DESTREG ← SRCREG1

IR7 = 1, IR10 = 1

Opcode2T4: **ARF ← ARF**

OutCSel = SRCREG1-2B-2[2][1]
MUXCSEL = 0
FUNSEL(ALU) = 0000
MUXBSEL=11
FUNSEL(2B) = 01
REGSEL(2B) = DESTREG-2B-2

IR7 = 1, IR10 = 0

Opcode2T4: **ARF ← RF**

OutCSel = SRCREG1-2B-2[2][1]
MuxASel=10
FunSel(2A) = 01
RegSel(2A) = DESTREG-2A-2

IR7 = 0, IR10 = 1

Opcode2T4: **RF ← ARF**

OutBSel=SRCREG1[1][0];
FunSel(ALU) = 0001;

MuxBSel = 11;
FunSel(2B)=01;
RegSel(2B) = DESTREG-2B-2;

IR7 = 0, IR10 = 0

Opcode2T4: **RF ← RF**

OutBSel=IR[5,6];
FunSel(ALU) = 0001;
MuxBSel = 11;
FunSel(2B)=01;
RegSel(2B) = DESTREG-2B-2;
MuxASel=11;
FunSel(2A)=01;
RegSel(2A) = DESTREG-2A-2;

Opcode3: M[SP] ← Rx, SP ← SP - 1

Opcode4T4: **M[SP] ← Rx**

Load = 1;
Read = 0;
OutBSel = IR[9,10];
FunSel(ALU)= 0001;

Opcode4T5: **SP ← SP-1**

FunSel(2B) = 11;
RegSel(2B) = 10;

Opcode4: SP ← SP + 1, Rx ← M[SP]

Opcode4T4: **SP ← SP + 1**

FunSel(2B) = 10;
RegSel(2B) = 100

Opcode4T5: **Rx ← M[SP]**

OutDSel = 10;
Read = 1;
Load = 0;
MuxASel = 01;
FunSel(2A) = 01;
RegSel(2A) = REGSEL-2A-0

Opcode5: DESTREG ← SRCREG1 + SRCREG2

rewind?

IR10 = 0, IR7=0, IR4 = 0

Opcode5T4: **RF ← RF + RF**

OutBSel = IR(2,3)
OutASel = IR(5,6)
MuxCSel = 1;
FunSel(ALU) = 0100
MuxASel = 11;
RegSel = DESTREG-2A-2
FunSel (2A)= 01

IR10 = 0, IR7=0, IR4 = 1

Opcode5T4: **RF ← RF + ARF**

OutCSel = SRCREG1-2B-2[2][1]
MuxCSel = 0;
OutBSel = IR(2,3);
FunSel(ALU) = 0100
MuxASel = 11;
RegSel = DESTREG-2A-2
FunSel (2A)= 01

IR10 = 0, IR7=1, IR4 = 0

Opcode5T4: **RF ← ARF + RF**

OutCSel = SRCREG1-2B-2[2][1]
MuxCSel = 0;
OutBSel = IR(5,6);
FunSel(ALU) = 0100
MuxASel = 11;
RegSel = DESTREG-2A-2
FunSel (2A)= 01

IR10 = 1, IR7=0, IR4 = 0

Opcode5T4: **ARF ← RF + RF**

OutBSel = IR(2,3)
OutASel = IR(5,6)
MuxCSel = 1;
FunSel(ALU) = 0100
MuxBSel = 11
FunSel(2B) = 01
RegSel(2B) = DESTREG-2B-2

IR10 = 1, IR7=1, IR4 = 0

Opcode5T4: **ARF ← ARF + RF**

OutCSel = SRCREG1-2B-2[2][1]

MuxCSel = 0

OutBSel = IR(2,3)

FunSel(ALU) = 0100

MuxBSel = 11
FunSel(2B) = 01
RegSel(2B) = DESTREG-2B-2

IR10 = 1, IR7=0, IR4 = 1

Opcode5T4: **ARF ← RF + ARF**

OutBSel = IR(5,6)

OutCSel = SRCREG2-2B-2(1,2)

MuxCSel = 0

FunSelALU = 0100
RegSel(2B) = DESTREG-2B-2

Opcode6: DESTREG ← SRCREG2 - SRCREG1

IR10 = 0, IR7=0, IR4 = 0

Opcode5T4: **RF ← RF - RF**

OutASel = IR(2,3);
MuxCSel = 1
OutBSel = IR(5,6)
FunSel(ALU) = 0110;
MuxASel = 11;
FunSel(2A) = 01;
RegSel = DESTREG-2A-2

IR10 = 1, IR7=0, IR4 = 0

Opcode5T4: **ARF ← RF - RF**

OutASel = IR(2,3)

MuxCSel = 1
OutBSel = IR(5,6)
FunSel(ALU) = 0110;
MuxBSel = 11
FunSel(2B) = 01;
RegSel(2B) = DESTREG-2B-2

IR10 = 0, IR7=0, IR4 = 1

Opcode5T4: **RF ← ARF – RF**

OutBSel = IR(5,6)
OutCSel) SRCREG2-2B-2(1,2)
MuxCSel = 0;
FunSel(ALU) = 0110
MuxASel = 11
FunSel(2A) = 01
RegSel(2A) = DESTREG-2A-2

IR10 = 1, IR7=0, IR4 = 1

Opcode5T4: **ARF ← ARF - RF**

OutBSel = IR(5,6)
OutCSel) SRCREG2-2B-2(1,2)
MuxCSel = 0;
FunSel(ALU) = 0110
MuxBSel = 11;
FunSel(2B) = 01;
RegSel(2B) = DESTREG-2B-2

Opcode7: DESTREG ← SRCREG1 - 1

IR10 = 1, IR7 = 1

Opcode7T4: **ARF ← ARF**

OutCSel = SRCREG1-2B-2(1,2)
MuxASel = 10
MuxCSel = 0
FunSel(ALU) = 0000
MuxBSel=11
FUNSEL(2B) = 01
REGSEL(2B) = DESTREG-2B-2

Opcode7T5: **ARF ← ARF - 1**

RegSel(2B) = DESTREG-2B-2
FunSel(2B) = 11

Opcode7T6: **PASSALU ARF**

OutCSel = DESTREG-2B-2(1,2)
MuxCSel = 0;

FunSel(ALU) = 0000;

IR10 = 0, IR7 = 1:

Opcode7T4: **RF ← ARF**

OutCSel = SRCREG-2B-2[2][1]
MuxASel=10
FunSel(2A) = 01
RegSel(2A) = DESTREG-2A-2

Opcode7T5: **RF ← RF - 1**

REGSEL = DESTREG-2A-2
FUNSEL(2A) = 11

Opcode7T6: **PASS ALU RF**

OutASel = IR(8,9)
MuxCSel = 1;
FunSel(ALU) = 0000;

IR10 = 1, IR7 = 0:

Opcode7T4: **ARF ← RF**

OutBSel= IR(5,6)
FunSel(ALU) = 0001
MUXBSEL = 11
FUNSEL(2B)=01
REGSEL(2B) = DESTREG-2B-2

Opcode7T5: **ARF ← ARF - 1**

RegSel(2B) = DESTREG-2B-2
FunSel(2B) = 11

Opcode7T6: **PASSALU ARF**

OutCSel = DESTREG-2B-2(1,2)
MuxCSel = 0;
FunSel(ALU) = 0000;

IR10 = 0, IR7 = 0:

Opcode7T4: **RF ← RF**

OutBSel= IR(5,6)
FunSel(ALU) = 0001
MuxASel=11
FunSel(2a)=01
RegSel(2A) = DESTREG-2A-2

Opcode7T5: **RF ← RF - 1**

RegSel(2A) = DESTREG-2A-2
FunSel(2A) = 11

Opcode7T6: **PASSALU RF**

OutASel = IR(8,9)

MuxCSel = 1;

FunSel(ALU) = 0000;

Opcode8: DESTREG ← SRCREG1 + 1

IR10 = 1, IR7 = 1

Opcode8T4: **ARF ← ARF**

OutCSel = SRCREG1-2B-2[2][1]

MuxCSel = 0

FunSel(ALU) = 0000

MuxBSel=11

FUNSEL(2B) = 01

REGSEL(2B) = DESTREG-2B-2

Opcode8T5: **ARF ← ARF + 1**

RegSel(2B) = DESTREG-2B-2

FunSel(2B) = 10

Opcode8T6: **PASSALU ARF**

OutCSel = DESTREG-2B-2(1,2)

MuxCSel = 0;

FunSel(ALU) = 0000;

IR10 = 0, IR7 = 1:

Opcode8T4: **RF ← ARF**

OutCSel = SRCREG1-2B-2[2][1]

MuxASel=10

FunSel(2A) = 01

RegSel(2A) = DESTREG-2A-2

Opcode8T5: **RF ← RF + 1**

REGSEL = DESTREG-2A-2

FUNSEL(2A) = 10

Opcode8T6: **PASSALU RF**

OutCSel = DESTREG-2B-2(1,2)

MuxCSel = 0;

FunSel(ALU) = 0000;

IR10 = 1, IR7 = 0:

Opcode8T4: **ARF ← RF**

OutBSel=SRCREG1[1][0]

FunSel(ALU) = 0001
MUXBSEL = 11
FUNSEL(2B)=01
REGSEL(2B) = DESTREG-2B-2

Opcode8T5: **ARF ← ARF + 1**
RegSel(2B) = DESTREG-2B-2
FunSel(2B) = 10

Opcode8T6: **PASSALU ARF**
OutCSel = DESTREG-2B-2(1,2)
MuxCSel = 0;
FunSel(ALU) = 0000;

IR10 = 0, IR7 = 0:

Opcode8T4: **RF ← RF**
OutBSel= IR(5,6)
FunSel(ALU) = 0001
MuxASel=11
FunSel(2a)=01
RegSel(2A) = DESTREG-2A-2

Opcode8T5: **RF ← RF + 1**
RegSel(2A) = DESTREG-2A-2
FunSel(2A) = 10

Opcode8T6: **PASSALU RF**
OutASel = IR(8,9)
MuxCSel = 1;
FunSel(ALU) = 0000;

Opcode9: DESTREG ← SRCREG1 AND SRCREG2

IR4=0,IR7=0,IR10=0

Opcode9T4: **RF ← RF AND RF**
OutASel = IR(5,6)
MuxCSel = 1;
OutBSel = IR(2,3)
FunSel(ALU) = 0111;
MuxASel = 11;
FunSel(2a) = 01;
RegSel(2a) = DESTREG-2A-2

IR4 = 0, IR7=0, IR10 = 1

Opcode9T4: **ARF ← RF AND RF**

OutASel = IR(5,6)
MuxCSel = 1;
OutBSel = IR(2,3)
FunSel(ALU) = 0111;
MuxBSel = 11;
FunSel(2B) = 01;
RegSel(2B) = DESTREG-2B-2

IR4 = 1, IR7=0, IR10 = 0:

Opcode9T4: **RF ← ARF AND RF**

OutBSel = IR(5,6)
OutCSel = SRCREG2-2B-2(2,1)
MuxCSel = 0;
FunSel(ALU) = 0111
MuxASel = 11;
FunSel(2a) = 01;
RegSel(2A) = DESTREG-2A-2

IR4 = 1, IR7=0, IR10 = 1:

Opcode9T4: **ARF ← ARF AND ARF**

OutBSel = IR(5,6)
OutCSel = SRCREG2-2B-2(2,1)
MuxCSel = 0;
FunSel(ALU) = 0111
MuxBSel = 11;
FunSel(2B) = 01;
RegSel(2B) = DESTREG-2B-2

IR4 = 0, IR7=1, IR10 = 0:

Opcode9T4: **RF ← RF AND ARF**

OutBSel = IR(2,3)
OutCSel = SRCREG1-2B-2(1,2)
MuxCSel = 0;
FunSel(ALU) = 0111
MuxASel = 11;
FunSel(2A) = 01;
RegSel(2a) = DESTREG-2A-2;

IR4 = 0, IR7=1, IR10 = 1:

Opcode9T4: **ARF ← RF AND ARF**

OutBSel = IR(2,3)
OutCSel = SRCREG1-2B-2(1,2)
MuxCSel = 0;
FunSel(ALU) = 0111;
MuxBSel:11

FunSel(2B):01
RegSel(2B):DESTREG-2B-2

Opcode10: DESTREG ← SRCREG1 OR SRCREG2

IR4=0,IR7=0,IR10=0

Opcode10T4: **RF ← RF OR RF**
OutASel = IR(5,6)
MuxCSel = 1;
OutBSel = IR(2,3)
FunSel(ALU) = 1000;
MuxASel = 11;
FunSel(2a) = 01;
RegSel(2a) = DESTREG-2A-2

IR4 = 0, IR7=0, IR10 = 1
Opcode10T4: **ARF ← RF OR RF**
OutASel = IR[5,6];
MuxCSel = 1;
OutBSel = IR[2,3]
FunSel(ALU) = 1000;
MuxBSel = 11;
FunSel(2B) = 01;
RegSel(2B) = DESTREG-2B-2

IR4 = 1, IR7=0, IR10 = 0:
Opcode10T4: **RF ← RF OR ARF**
OutBSel = IR(5,6)
OutCSel = SRCREG2-2B-2(2,1)
MuxCSel = 0;
FunSel(ALU) = 1000
MuxASel = 11;
FunSel(2a) = 01;
RegSel(2A) = DESTREG-2A-2

IR4 = 1, IR7=0, IR10 = 1:
Opcode10T4: **ARF ← RF OR ARF**
OutBSel = SRCREG1(0,1)
OutCSel = SRCREG2-2B-2(2,1)
MuxCSel = 0;
FunSel(ALU) = 1000
MuxBSel = 11;

FunSel(2B) = 01;
RegSel(2B) = DESTREG-2B-2

IR4 = 0, IR7=1, IR10 = 0:

Opcode10T4: **RF ← ARF OR RF**

OutBSel = SRCREG2(0,1);
OutCSel = SRCREG1-2B-2(1,2)
MuxCSel = 0;
FunSel(ALU) = 1000
MuxASel = 11;
FunSel(2A) = 01;
RegSel(2a) = DESTREG-2A-2;

IR4 = 0, IR7=1, IR10 = 1:

Opcode10T4: **ARF ← ARF OR RF**

OutBSel = SRCREG2(0,1);
OutCSel = SRCREG1-2B-2(1,2)
MuxCSel = 0;
FunSel(ALU) = 1000;
MuxBSel:11
FunSel(2B):01
RegSel(2B):DESTREG-2B-2

Opcode11: DESTREG ← NOT SRCREG1

IR7=0,IR10=0

Opcode11T4: **RF ← NOT RF**

OutASel = IR(5,6)
MuxCSel = 1;
FunSel(ALU) = 0010;
MuxASel = 11;
RegSel(2A) = DESTREG-2A-2;
FunSel(2A) = 01

IR7=0,IR10=1

Opcode11T4: **ARF ← NOT RF**

OutASel = IR(5,6)
MuxCSel = 1;
FunSel(ALU) = 0010;
MuxBSel = 11;
RegSel(2B) = DESTREG-2B-2;
FunSel(2B) = 10

IR7=1,IR10=0

Opcode11T4: **RF ← NOT ARF**

OutCSel = SRCREG1-2B-2
MuxCSel = 0;
FunSel(ALU) = 0010;
MuxASel = 11;
FunSel(2A) = 01;
RegSel(2A) = DESTREG-2A-2

IR7=1,IR10=1

Opcode11T4: **ARF ← NOT ARF**

OutCSel = SRCREG1-2B-2
MuxCSel = 0;
FunSel(ALU) = 0010;
MuxBSel= 11;
FunSel(2B) = 01;
RegSel(2B) = DESTREG-2B-2

Opcode12: DESTREG ← LSL SRCREG1

IR7=0,IR10=0

Opcode12T4: **RF ← LSL RF**

OutASel = IR(5,6)
MuxCSel = 1;
FunSel(ALU) = 1010;
MuxASel = 11;
FunSel(2A) = 01;
RegSel(2A) = DESTREG-2A-2

IR7=0,IR10=1

Opcode12T4: **ARF ← LSL RF**

OutASel = IR(5,6)
MuxCSel = 1;
FunSel(ALU) = 1010;
MuxBSel= 11;
FunSel(2B) = 01;
RegSel(2B) = DESTREG-2B-2

IR7=1,IR10=0

Opcode12T4: **RF ← LSL ARF**

OutCSel = SRCREG1-2B-2;
MuxCSel = 0;
FunSel(ALU) = 1010;
MuxASel = 11;
FunSel(2A) = 01;
RegSel(2A) = DESTREG-2A-2

IR7=1,IR10=1

Opcode12T4: **ARF ← LSL ARF**

OutCSel = SRCREG1-2B-2;
MuxCSel = 0;
FunSel(ALU) = 1010;
MuxBSel = 11;
FunSel(2B) = 01;
RegSel(2B) = DESTREG-2B-2

Opcode13: DESTREG ← LSR SRCREG1

IR7=0,IR10=0

Opcode13T4: **RF ← LSR RF**

OutASel = IR(5,6)
MuxCSel = 1;
FunSel(ALU) = 1011;
MuxASel = 11;
FunSel(2A) = 01;
RegSel(2A) = DESTREG-2A-2

IR7=0,IR10=1

Opcode13T4: **ARF ← LSR RF**

OutASel = IR(5,6)
MuxCSel = 1;
FunSel(ALU) = 1011;
MuxBSel= 11;
FunSel(2B) = 01;
RegSel(2B) = DESTREG-2B-2

IR7=1,IR10=0

Opcode13T4: **RF ← LSR ARF**

OutCSel = SRCREG1-2B-2;
MuxCSel = 0;
FunSel(ALU) = 1011;
MuxASel = 11;
FunSel(2A) = 01;
RegSel(2A) = DESTREG-2A-2

IR7=1,IR10=1

Opcode13T4: **ARF ← LSR ARF**

OutCSel = SRCREG1-2B-2;
MuxCSel = 0;
FunSel(ALU) = 1011;
MuxBSel = 11;
FunSel(2B) = 01;
RegSel(2B) = DESTREG-2B-2

Opcode14: PC ← Value

Note: Addressing mode IM

Opcode14T4: **PC ← IROut(0,7)**

MuxBSel = 01;
RegSel(2B) = 001;
OutDSel =
FunSel = 01;

Opcode15: IF Z = 1 THEN PC ← Value

Note: Addressing mode IM

Z = 1:

Opcode15T4: **PC ← IROut(0,7)**

MuxBSel = 01;
RegSel(2B) = 001;
FunSel(2B)= 01;

Opcode16: IF Z = 0 THEN PC ← Value

Note: Addressing mode IM

Z = 0:

Opcode16T4: **PC ← IROut(0,7)**

MuxBSel = 01;
RegSel(2B) = 001;
FunSel(2B)= 01;

Opcode17: M[SP] ← PC, SP ← SP-1, PC ← Value

Note: Addressing mode IM

Opcode17T4: **M[SP] ← PC**

Read = 0;
Load = 1;
OutDSel = 10;
OutCSel = 00;
MuxCSel = 0;
FunSel(ALU) = 0000;

Opcode17T5: **SP ← SP-1**

RegSel(2B) = 100;
FunSel(2B) = 11;

Opcode17T6: **PC ← IROut(0,7)**

MuxBSel = 01;

RegSel(2B) = 01;

FunSel(2B) = 01;

Opcode18: SP ← SP+1, PC ← M[SP]

Opcode18T4: **SP ← SP+1**

RegSel(2B) = 100;

FunSel(2B) = 10;

Opcode18T5: **PC ← M[SP]**

OutDSel = 10;
Read = 1;
Load = 0
MuxBSel = 10;
RegSel(2B) = 001;
FunSel(2B) = 01;

5. Conclusion

In conclusion, during this project we gain technical knowledge as well as some soft skills. Due to COVID-19 pandemic we were forced to work online, this was a very challenging process but We believe that our abilities to do this have improved. We understood further how microprogrammed Control units work and how to implement them. It was a bit fard at first to wrap our heads around this concept but we eventually figured it out and implemented the necessary microprograms.