Lab 4: BASIC COMPUTER ORGANIZATION

CEG2136

Summer Term 2022

By:

Alp Tatar (300241739)

Fahmi Sajid Ahmed (300250180)

Experiment 23-July-2022

Objectives

●Analyse and comprehend structure of a basic computer
●Design control unit

●Write simple programs in machine code

PreLab

5.1

Q1. Draw a diagram which shows the hierarchy of the files, with lab3top at the top. For the files lab3controller, ram256x8, and sevensegcontroller, you do not have to identify the subfiles.

A1.

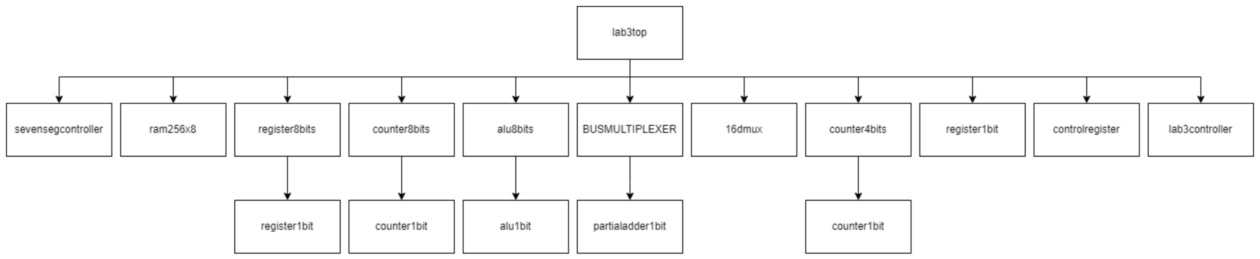


Figure 1 Hiearchy of the Lab Files

Q2. How can you check by analysing these files that only one register will place its output on the data bus at a time?

A2. The data in the bus comes from a multiplexer (which produces only 1 output from a selection of inputs), therefore making it impossible for it to have more than one register's output at a time.

Q3. Are the register reset (clear) signals synchronous or asynchronous? Explain your answer. Note that all the command signals are active at high (i.e. a "1" will reset a register to 0).

The register reset (clear) signal is always set to LOW (GND) therefore it must be asynchronous.

Q4. What happens if a load and a reset are simultaneously sent to a register? Why?

The outcome depends on the type of flip flop used in the register (as SR flip flops will produce a bad state). In the case of the D flip flop registers presented in this lab, when both LOAD and CLEAR are 1, the stored value becomes 0 regardless of the D input.

Q5. Why the address register is connected directly to the memory?

The address register is connected directly to the memory because the memory requires an address when it comes to fetching and storing data (as each word in the memory has its own unique address), which is provided only by the address register.

Q6. Why the Program Counter, the Data Register, and the Accumulator are implemented as Counters?

A6. The program counter is a counter because its stored value is incremented by one for every instruction address that it has successfully received and sent out, with each counter value holding a stored instruction address. Whenever the computer is reset, this value is reset to 0. Therefore, the Program Counter is required to be a counter in order to go from address to address (effectively tracking it).

The Data Register is implemented as a counter because it has a function where it has to increment its contents (ISZ), which is only possible when the Data Register is a counter.

The Accumulator is implemented as a counter because some of its functions require incrementing its contents (INC), which are some of the specialties of a counter.

Q7. Of all the three commands of the counters (reset, increment, and load), which one has the highest priority? Which one has the lowest priority? Explain your answer.

A7. “Clear” has the highest priority as, in the counter8bits BDF, it is the only input that is not put through any combinational logic before getting to the counter1bit BDF. In contrast, “Increment” and “Load” have to be put through AND gates, where “Load” only goes through when “Clear” is LOW, and where “Increment” goes through when both “Load” and “Clear” are LOW. Therefore, “Clear” has the highest priority.

“Increment” is the lowest priority as it requires both “Load” and “Clear” to be LOW in order for the “Increment” input to get to the counter1bit (according to the counter8bits BDF). The “Increment” input in the counter1bit BDF does nothing when it is LOW, therefore, “Increment” has the least priority.

Q8. Is it possible to read a value from memory directly to the accumulator? Explain your answer.

It is not possible to read a value directly to the accumulator from memory. This is because, in order for the value to reach the accumulator, it has to go through the 8-bit data register as well as the 8-bit ALU (as the accumulator is the only circuit not directly attached to the common bus). It should be noted that the purpose of the accumulator is to hold intermediate arithmetic data, which is only available after data has gone through an ALU.

Q9. Analyze the ALU and determine a truth table which describes the 8 operations which can be selected by the three control lines. Are the shift operations logical or arithmetic?

A9.

Table 1 Operations Using 3 Control Lines

S_2	S_1	S_0	**Operation**
0	0	0	Sum <= X + Y + C_In
0	0	1	Sum <= X + Y’ + C_In
0	1	0	Shift X left one bit
0	1	1	Shift X right one bit
1	0	0	Sum <= X $\wedge$ Y
1	0	1	Sum <= X V Y
1	1	0	Sum <= Y
1	1	1	Sum <= X’

The shift operations described by the ALU are arithmetic shifts. This is because, for the MUX of the MSB, the input of the right shift operation is AC_7 (aka the MSB of AC), which means that the MSB won't change. This follows the description of an arithmetic shift as the sign isn't changing (unlike logic shifts where the MSB should become 0 after a right shift).

5.2

Bus 0=$T_3Y_5+T_{10}Y_6+T_9Y_4$
Bus 1=$T_0+T_{10}Y_6+T_2+T_5$
Bus 2 =$T_0+T_9Y_4$

Or:

T_8Y_7:DR $\leftarrow$M[AR]
T_9Y_7:AC $\leftarrow$AC v DR

Tale 2 CPU ALU Conditions, Operations and ALU select

Control Conditions	Operations	ALU Select		
		ALU-Sel 2	ALU-Sel 1	ALU-Sel 0
T_9Y_1	AC+DR	0	0	0

T_9Y_2	AC+DR’+1	0	0	1
$T_5X_1IR_2$	Ashl AC	0	1	0
$T_5X_1IR_3$	Ashr AC	0	1	1
T_9Y_0	AC ∧ DR	1	0	0
T_9Y_7	AC V DR	1	0	1
T_9Y_3	DR	1	1	0
$T_5X_1IR_1$	AC’	1	1	1

<u>Bus</u>
BusSel0 = (T8)(Y5)+(T10)(Y6)+(T9)(Y4)
BusSel1 = (T10)(Y6)+T0+T5+(T5)(IR6)’
BusSel2 = (T9)(Y4)+T0

<u>ALU</u>
ALUSel0= (Y2)(T9)+((T5)(X1)(IR1+IR3)
ALUSel1= (IR1+IR2+IR3)(T5)(X1)+(T9)(Y3)
ALUSel2=(T5)(X1)(IR1)+(T9)(Y0+Y3)

<u>Memory</u>
MemWrite=(T9)(Y4)+(T10)(Y6)

<u>CPU</u>
ARLoad =(T0+T2+(T5)(IR6)’+(T6)(IR6)’+(T7)(X2)
PCload =(T8)(Y5)
PCinc=((T2)+(T5)(IR6)’)(Stop)’)+((DR7+DR6+DR5+DR4+DR3+DR2+DR1+DR0)’(Y6)(Stop)’(T11+T12))
DRload =(Y0+Y1+Y2+Y3+Y6)(T8)
DRinc =(T5)(Y6)
IRLoad = T3
ACclear =(T5)(X1)(IR0)
ACload =((IR1+IR2+IR3)(T5)(X1))+((T9)(Y0+Y1+Y2+Y3)
ACinc =(T5)(X1)(IR4)
OutLoad =T1

<u>Control Unit</u>
SCclear = (Y0+Y1+Y2+Y3+Y4)(T9)+(T8)(Y5)+(T12)(Y6)+(T5)(X1)
Halt =(T5)(X1)(IR5)

7.1 Prelab

A0 = Counter
A1 = X
A2 = Y
A3 = Z

2.
Pseudo code:
Counter is sent to the AC register from address A0H.
The content of the AC register is complemented, which means it now contains Counter'.
The content of the AC register is now stored back to address A0H in the memory.
The memory content at address A0H is now incremented, meaning it now contains Counter'+1.
Branches unconditionally to address 20H.
/////
Computer is halted, meaning the PC register is not incremented until the next instruction.
Via indirect addressing, data from the A1H address (which then contains the address 80H) in the memory is sent to the AC register.
Via indirect addressing, data from the A2H address in the memory (which contains the address 81H) is added to the data in the AC register.
The content of the AC register is sent to memory address A3H, which points to another address (82H) where the content of AC is stored.
Via indirect addressing, data from the A1H address (which then contains the address 80H) in the memory is sent to the AC register.
The content of the AC register is then incremented.
The content of the AC register is stored at address A2H, which contains another address (81H) where the data will be stored.
Branches unconditionally to address 05H.

The above process repeats until the Counter variable is 00H (which is 10 times).

So, essentially

80H: 01H
81H: 01H
82H: 02H
83H: 03H
84H: 05H
85H: 08H
86H: 0DH
87H: 15H
88H: 22H
…

3. This program calculates the Fibonacci sequence and stores them one after the other.

4. It is practical to use indirect addressing for this program as the pointers make it easy to move between addresses in order to calculate the next number in the Fibonacci sequence. This allows the function/subroutine to continue on seamlessly for as long as the counter permits.

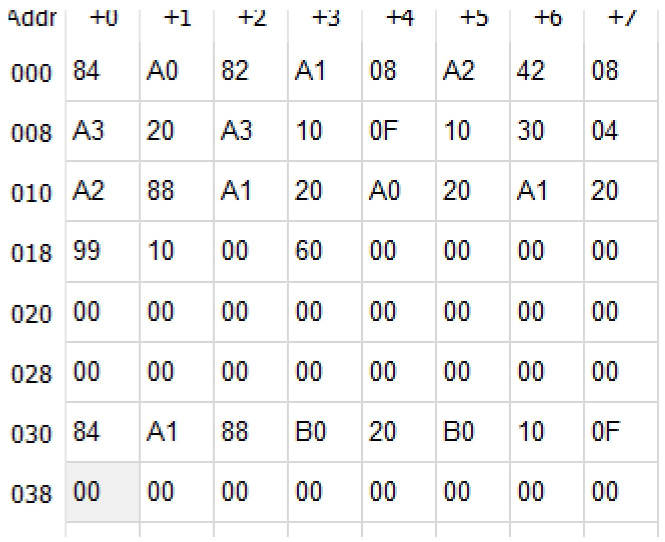
7.2

Memory addresses of operands (25 total)
80H: 21
81H: B5
82H: 37
83H: 08
84H: 5C
85H: 84
86H: A1
87H: 1D
88H: 72
89H: FF
8AH: F6
8BH: 43
8CH: 03
8DH: A9
8EH: D4
8FH: 19
90H: 31
91H: D9
92H: 47
93H: 82
94H: 14
95H: 52
96H: 07
97H: CA
98H: 04
99H: E7
A0H: 80 (address)
A1H: 81 (address)
A2H: holder of value
A3H: holder of value prime

00: LDA (indirect)
01: A0
02: ADD (indirect)
03: A1
04: STA (direct)

```
05: A2
06: CMA (works)
07: STA (direct)
08: A3
09: ISZ (direct)
0A: A3
0B: BUN (direct)
0C: 0F
0D: BUN (direct)
0E: 30
0F: LDA (direct)
10: A2
11: STA (indirect)
12: A1
13: ISZ (direct)
14: A0
15: ISZ (direct)
16: A1
17: ISZ (direct)
18: 99
19: BUN (direct)
1A: 00
1B: Halt

Subroutine
30: LDA (indirect)
31: A1
32: STA (indirect)
33: B0
34: ISZ (direct)
35: B0
36: BUN
37: 0F
```



Addr	+0	+1	+2	+3	+4	+5	+6	+7
000	84	A0	82	A1	08	A2	42	08
008	A3	20	A3	10	0F	10	30	04
010	A2	88	A1	20	A0	20	A1	20
018	99	10	00	60	00	00	00	00
020	00	00	00	00	00	00	00	00
028	00	00	00	00	00	00	00	00
030	84	A1	88	B0	20	B0	10	0F
038	00	00	00	00	00	00	00	00

Figure 2 Subroutine For Computer

Introduction

In this lab we are given an already implemented computer with a capacity of 256 words of 8 bits. The implementation of the basic computer will be shown in Section 4. The main purpose of this lab is to design the control unit, in order words to design the instruction decoder. We will test our implementation of the control unit with the Altera DE2-115. A more secondary purpose is to write a simple program in machine code and test using the basic computer we designed by completing the control unit.

Theory

Instruction Code

An instruction code in simple terms is a group of bits that instructs the computer to perform a specific operation. An instruction code must specify the operation register or memory address where the operand is located as well as the register or memory word where the result of the instruction is to be stored.

Control Unit

The control unit generates the timing and control signals for the operations of the computer and registers. It directs the operation of the processor which is in our case the accumulator. The control unit reads specific instruction from a specific address in the memory and executes it (generates the appropriate timing and control signals for the specific operation and enables appropriate registers for loading, clearing, or incrementing). The control unit also controls all input and output flow.

Computer Registers

The registers used in the implementation of the basic computer are the following:

- Address Register: It holds the memory address for memory. It communicates directly with memory.
- Program Counter: It holds the address of the instruction code. It normally goes through a counting sequence which allows the computer to read sequential instructions.

- Instruction Register: It holds the instruction code of the basic computer.
- Data Register: It holds a memory operand which is connected directly to the ALU and allows operations to be performed with the processor such as ADD, AND, OR.
- Accumulator: This is the processor register. Most of the micro-operation instructed include this register because this is the register the computer uses to manipulate data.
- Input & Output Register: These registers allow the computer to communicate with the user.

Commun Bus System
Since the basic computer includes many registers, a memory unit, and a control unit. One way or the other, paths must be provided to transfer information from one register to another, or from memory to a register. To transfer information more efficiently in a system like a basic computer that contains many registers we the common bus. A common bus is a system that transfers data between registers/components inside a computer. It allows very efficient data transfer.

The Structure of The Basic Computer

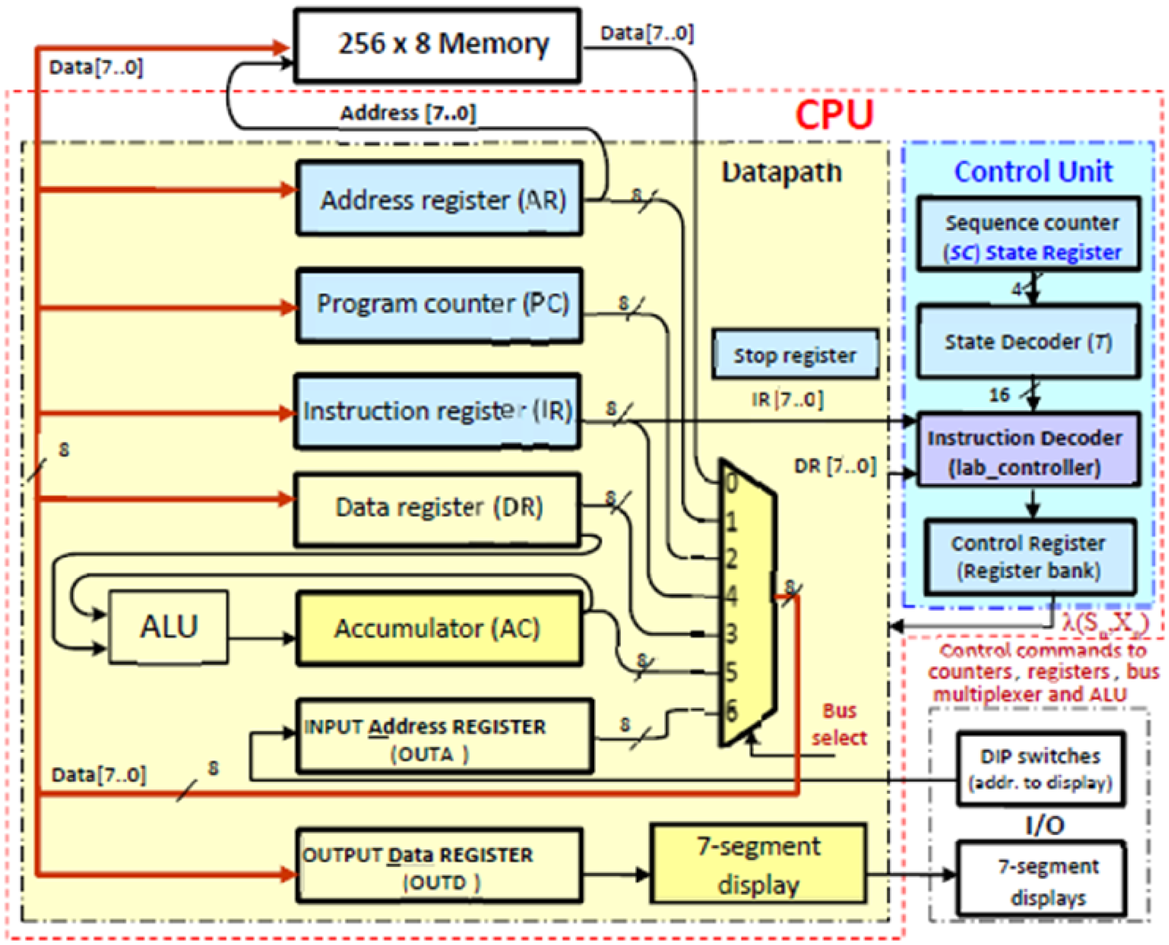

256 x 8 Memory
Data[7..0]
Data[7..0]
Address [7..0]
CPU
Datapath
Control Unit
Address register (AR)
Program counter (PC)
Instruction register (IR)
Data register (DR)
ALU
Accumulator (AC)
INPUT Address REGISTER (OUTA)
OUTPUT Data REGISTER (OUTD)
7-segment display
Stop register
IR [7..0]
DR [7..0]
Sequence counter (SC) State Register
State Decoder (T)
Instruction Decoder (lab_controller)
Control Register (Register bank)
4
16
8
λ(S_n,X_n)
Control commands to counters, registers, bus multiplexer and ALU
Bus select
0
1
2
4
3
5
6
Data[7..0]
DIP switches (addr. to display)
I/O
7-segment displays

Figure 3 Computer Block Diagram

Details Description of the control Function
In order to separate the types of instruction we have to look at the 2 most significant bits of the 8 bit instruction register. There’s three different types of instruction.For instance, the most significant bit indicates if in case of memory reference instruction it is direct or indirect (direct if IR7 = 0 and indirect if IR7 = 1).

X0=(IR7)’(IR6)’:Direct memory-reference instruction
X1=_(IR7)’(IR6):Register-Reference Instruction
X2=(IR7)(IR6)’:Indirect Memory-Reference Instruction

The memory-reference instructions contains 7 instructions which are differentiated by these control signals:

Y0=(IR6)’ (IR1)’ IR0:AND
Y1=(IR6)’(IR1)(IR0)’ :ADD
Y2=(IR6)’ (IR1)(IR0):SUB
Y3=(IR6)’(IR2):LDA
Y4=(IR6)’ IR3:STA
Y5= (IR6)’ IR4:BUN
Y6= (IR6)’ IR5:ISZ

The following tables will show the implementation of the basic computer

Table 3 Instruction Fetch Cycle (Common to all types of instruction)

State	Description	Notation RTL
T_0	Load the address register AR with the contents of OUTA	T_0: $AR \leftarrow OUTA$
T_1	Read memory location pointed to by AR to the *data output register* OUTD	T_1: $OUTD \leftarrow M[AR]$
T_2	▪ Load AR register with the ADDRESS of the *opcode* of the current instruction (PC) ▪ Increment PC (if the program is running, i.e., if Stop FF S = 0) to point to the address of the next byte to be read, which can be: o the next instruction, if the current instruction is a ***register-reference instruction*** o the 2nd byte of the current instruction, if it is a ***memory-reference instruction***.	T_2: $AR \leftarrow PC$ T_2S': $PC \leftarrow PC + 1$
T_3	Read the instruction's first byte (*opcode*) from memory location specified by AR to IR	T_3 : $IR \leftarrow M[AR]$
T_4	This state is a delay that allows the *opcode* to be decoded in the *Control Unit*.	(nothing)
T_5	If X_1, the byte in IR is a ***register - reference instruction*** and will be executed now	T_5X_1: execute instruction from Table 3 T_5X_1 :$SC \leftarrow 0$
	If X_0 or X_2 (*memory - reference instruction*), ▪ PC is copied to AR, i.e, the address of the instruction's 2nd byte goes from PC to AR => now AR contains the ADDRESS of o the *operand address* if *direct addressing*, or o the **address** *of the operand address* if *indirect addressing* ▪ Increment PC if the program is running (Stop FF S=0). Note that $(X_0 + X_2) = \overline{IR_6}$.	$T_5IR'_6 : AR \leftarrow PC$ $T_5IR'_6S' : PC \leftarrow PC + 1$
T_6	Read from memory location pointed to by AR to AR; the read byte is ▪ the *operand address*, if *direct addressing*, or ▪ the **address** *of the operand address, if indirect addressing*	$T_6IR'_6 : AR \leftarrow M[AR]$
T_7	If *indirect addressing*, read the *operand address* from memory location pointed to by AR	$T_7X_2 : AR \leftarrow M[AR]$
	If *direct addressing*, don't do anything, as the operand's address is already in AR since T_6	T_7X_0 : (nothing)
T_8 & after	Execute the ***memory - reference instruction*** as described in Table 4.	(see Table 4)

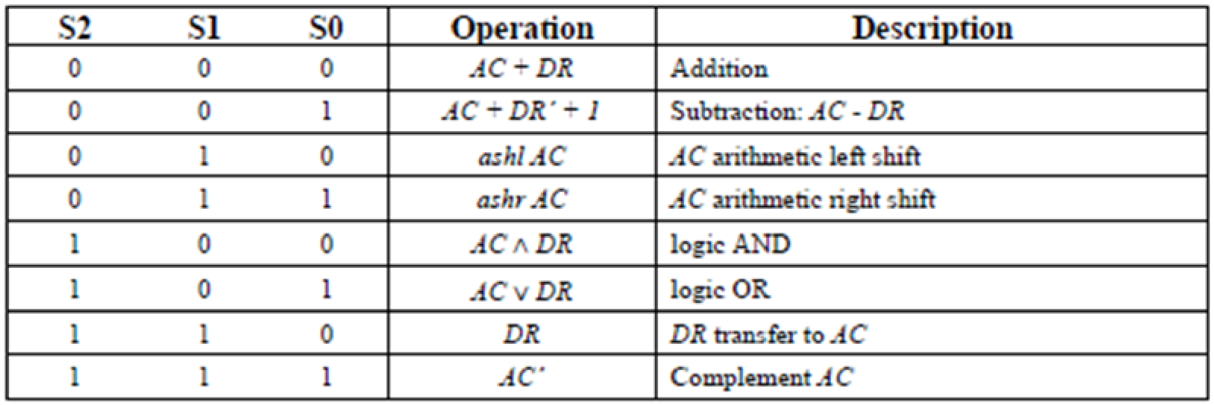
Table 4 Instruction Execution Cycle (Register-Reference Instructions)

Symbol	RTL Notation
CLA	$T_5X_1IR_0 : AC \leftarrow 0$
CMA	$T_5X_1IR_1 : AC \leftarrow \overline{AC}$
ASL	$T_5X_1IR_2 : AC \leftarrow \text{ashl } AC$
ASR	$T_5X_1IR_3 : AC \leftarrow \text{ashr } AC$
INC	$T_5X_1IR_4 : AC \leftarrow AC+1$
HLT	$T_5X_1IR_5 : S \leftarrow 1$

Table 5 - Instruction Execution Cycle (Memory-Reference Instructions)

Symbol	RTL Notation
AND	$T_8Y_0 : DR \leftarrow M[AR]$ $T_9Y_0 : AC \leftarrow AC \wedge DR,\ SC \leftarrow 0$
ADD	$T_8Y_1 : DR \leftarrow M[AR]$ $T_9Y_1 : AC \leftarrow AC + DR,\ SC \leftarrow 0$
SUB	$T_8Y_2 : DR \leftarrow M[AR]$ $T_9Y_2 : AC \leftarrow AC - DR,\ SC \leftarrow 0$
LDA	$T_8Y_3 : DR \leftarrow M[AR]$ $T_9Y_3 : AC \leftarrow DR,\ SC \leftarrow 0$
STA	T_8 : (cycle not allocated to allow the address bus to stabilize) $T_9Y_4 : M[AR] \leftarrow AC,\ SC \leftarrow 0$
BUN	$T_8Y_5 : PC \leftarrow AR,\ SC \leftarrow 0$
ISZ (assuming that the next instruction is a memory-reference instruction, stored at 2 memory location further down)	$T_8Y_6 : DR \leftarrow M[AR]$ $T_9Y_6 : DR \leftarrow DR+1$ $T_{10}Y_6 : M[AR] \leftarrow DR$ $T_{11}Y_6 : \text{si } (DR=0) \text{ alors } (\overline{S} : PC \leftarrow PC+1)$ $T_{12}Y_6 : \text{si } (DR=0) \text{ alors } (\overline{S} : PC \leftarrow PC+1),\ SC \leftarrow 0$

Table 6 ALU Operations



S2	S1	S0	Operation	Description
0	0	0	$AC + DR$	Addition
0	0	1	$AC + DR' + 1$	Subtraction: $AC - DR$
0	1	0	*ashl AC*	*AC* arithmetic left shift
0	1	1	*ashr AC*	*AC* arithmetic right shift
1	0	0	$AC \wedge DR$	logic AND
1	0	1	$AC \vee DR$	logic OR
1	1	0	*DR*	*DR* transfer to *AC*
1	1	1	AC'	Complement *AC*

Control Unit Equations

Bus

BusSel0 = (T8)(Y5)+(T10)(Y6)+(T9)(Y4)
BusSel1 = (T10)(Y6)+T0+T5+(T5)(IR6)’
BusSel2 = (T9)(Y4)+T0

ALU

ALUSel0= (Y2)(T9)+((T5)(X1)(IR1+IR3)
ALUSel1= (IR1+IR2+IR3)(T5)(X1)+(T9)(Y3)
ALUSel2=(T5)(X1)(IR1)+(T9)(Y0+Y3)

Memory

MemWrite=(T9)(Y4)+(T10)(Y6)

CPU

ARLoad =(T0+T2+(T5)(IR6)’+(T6)(IR6)’+(T7)(X2)
PCload =(T8)(Y5)
PCinc=((T2)+(T5)(IR6)’)(Stop)’)+((DR7+DR6+DR5+DR4+DR3+DR2+DR1+DR0)’(Y6)(Stop)’(T11+T12))
DRload =(Y0+Y1+Y2+Y3+Y6)(T8)
DRinc =(T5)(Y6)
IRLoad = T3
ACclear =(T5)(X1)(IR0)
ACload =((IR1+IR2+IR3)(T5)(X1))+((T9)(Y0+Y1+Y2+Y3)
ACinc =(T5)(X1)(IR4)
OutLoad =T1

Control Unit

SCclear = (Y0+Y1+Y2+Y3+Y4)(T9)+(T8)(Y5)+(T12)(Y6)+(T5)(X1)
Halt =(T5)(X1)(IR5)

Design/Implementation

Below are the different logic circuits of the solved design that can be found in the prelab. Below are the specific ones for each labeled section. To view them all at once please check lab3controller file attached in the zip file.

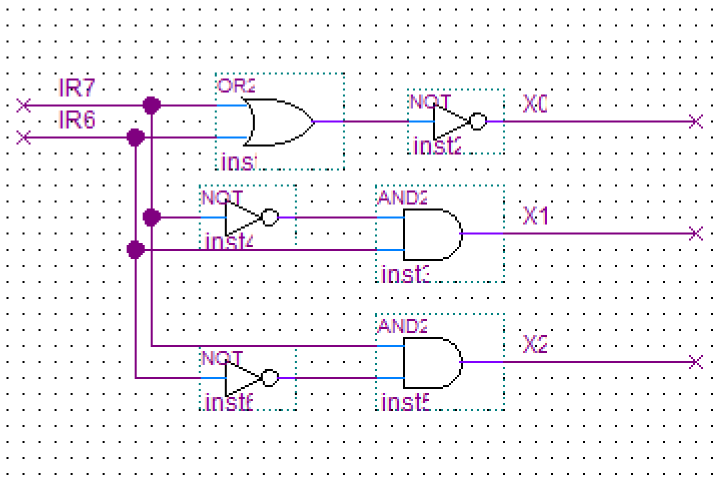


Figure 4 Logic Circuit of the X Inputs Gates

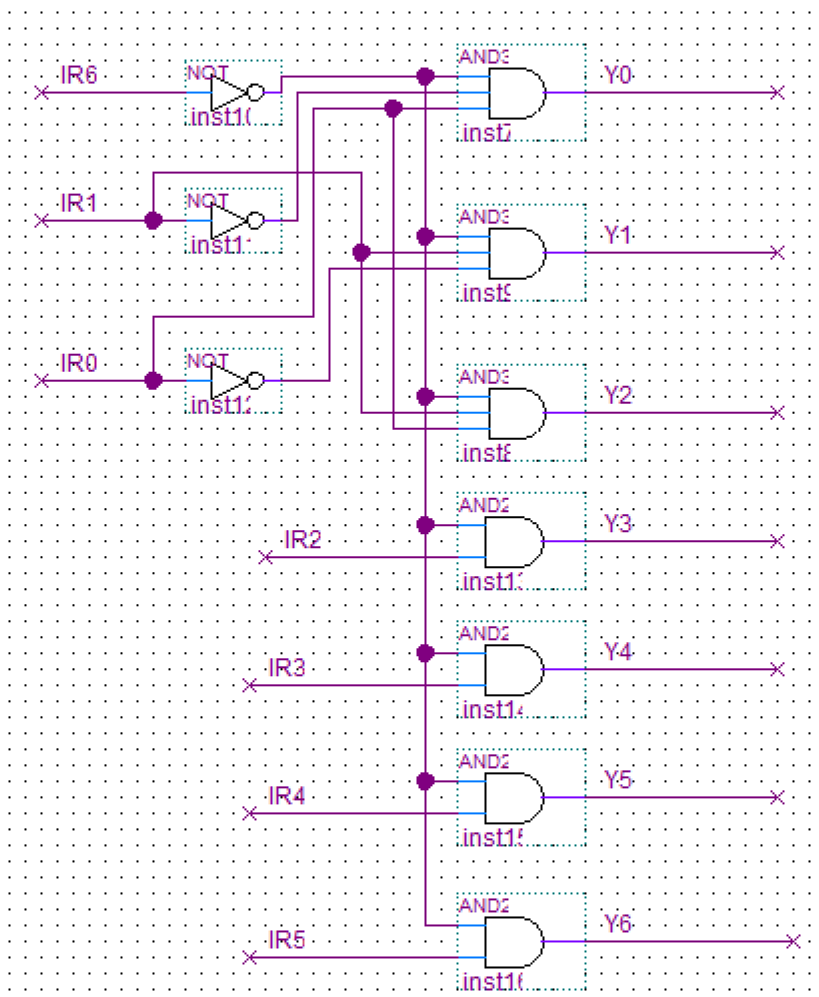


Figure 5 Logic Circuit of the Y Inputs Gates

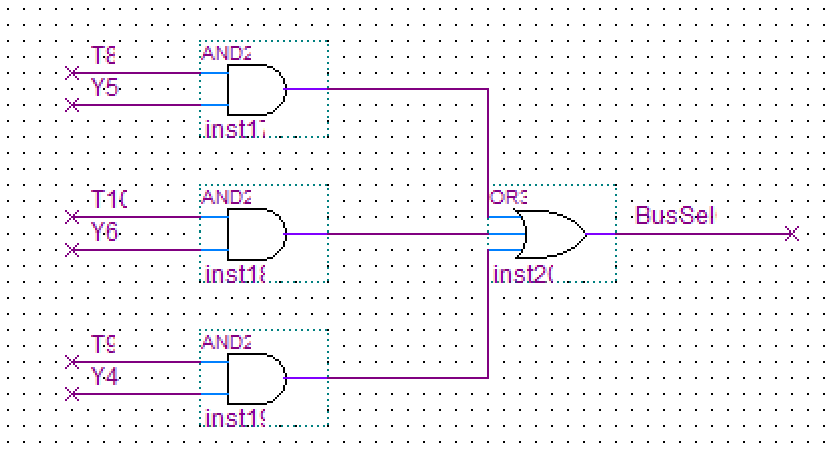


Figure 6 Logic Circuit of the Bus Selection 0

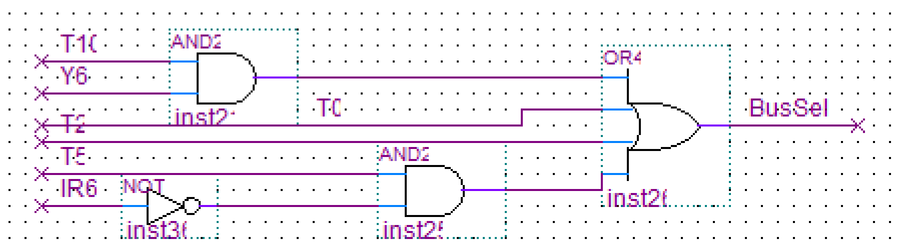


Figure 7 Logic Circuit of the Bus Selection 1

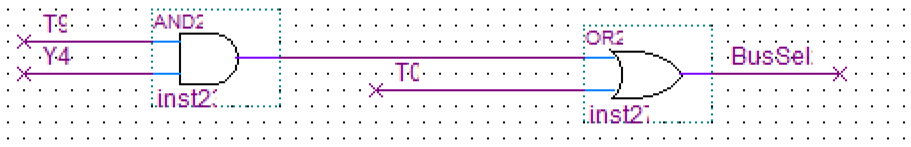


Figure 8 Logic Circuit of the Bus Selection 2

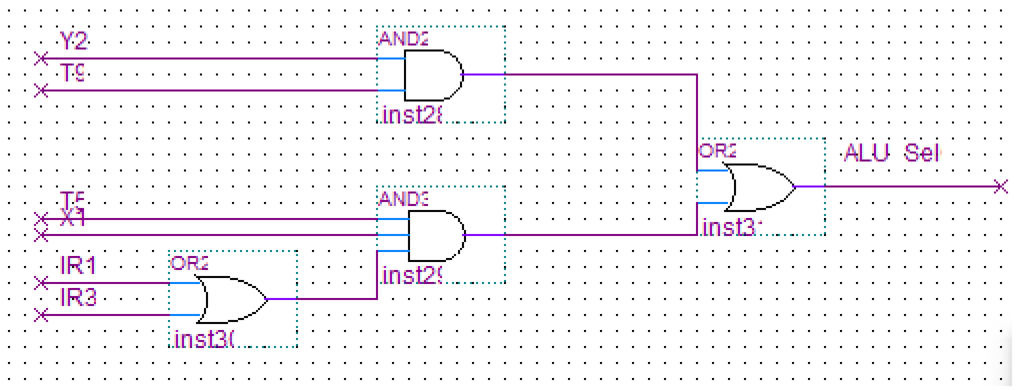


Figure 9 Logic Circuit of the ALU Select 0

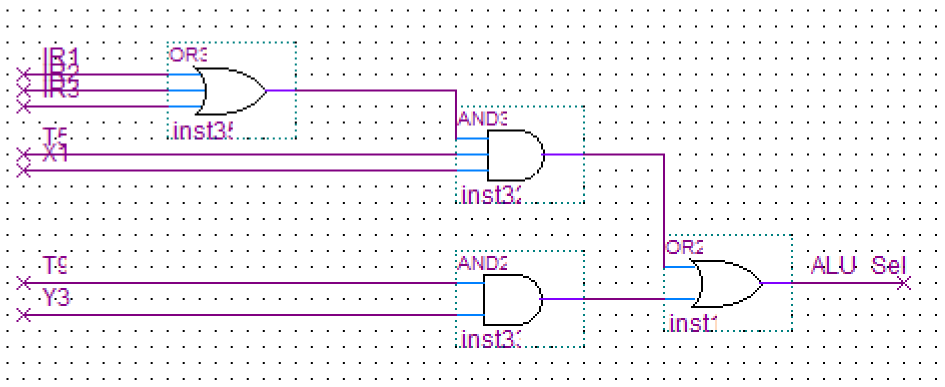


Figure 10 Logic Circuit of the ALU Select 1

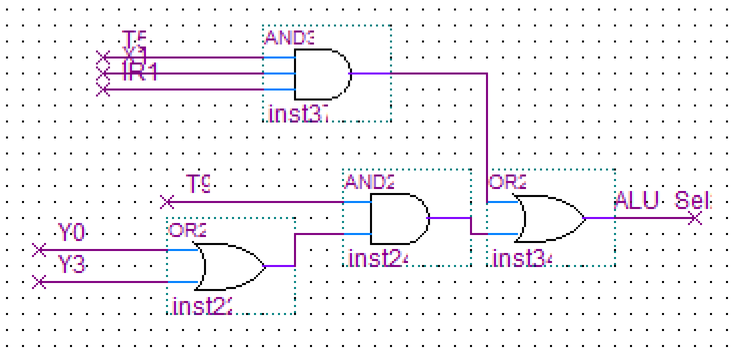


Figure 11 Logic Circuit of the ALU Select 2

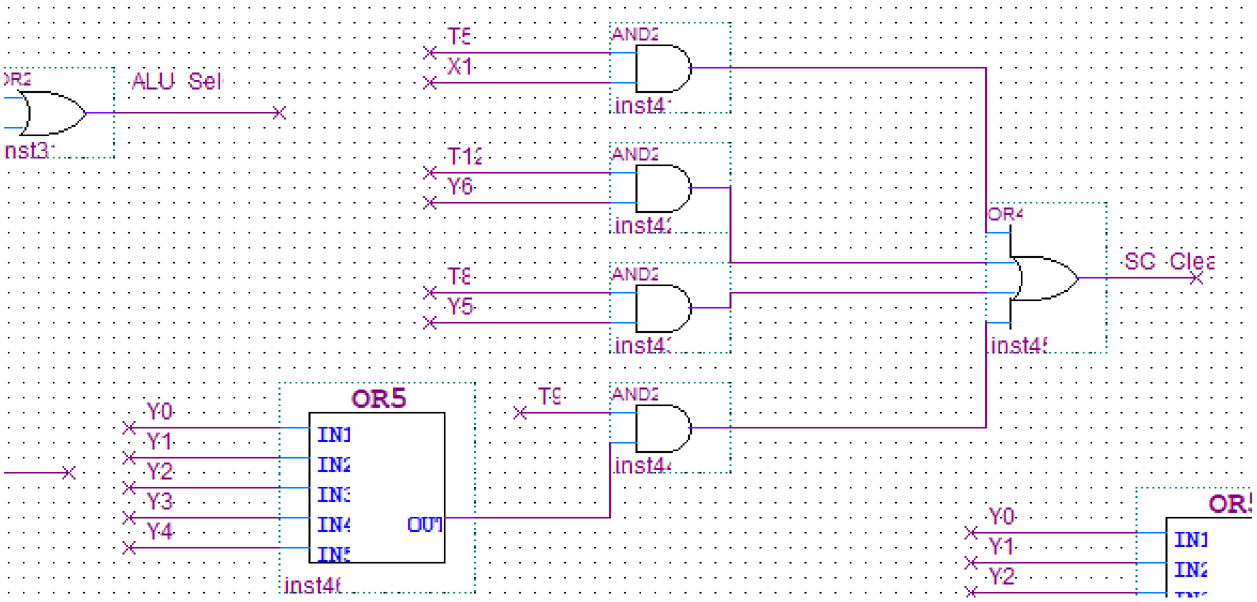


Figure 12 Logic Circuit of the SC Clear

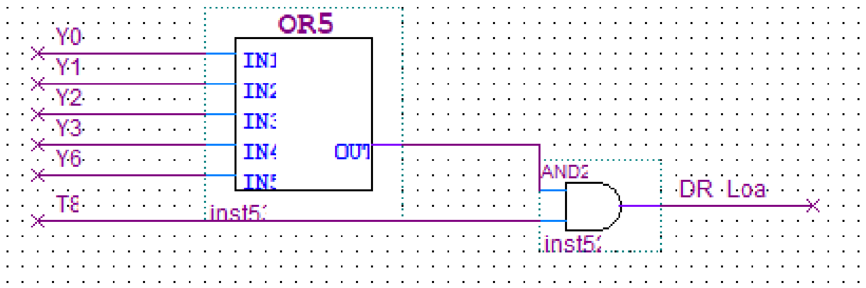


Figure 13 Logic Circuit of the DR Load

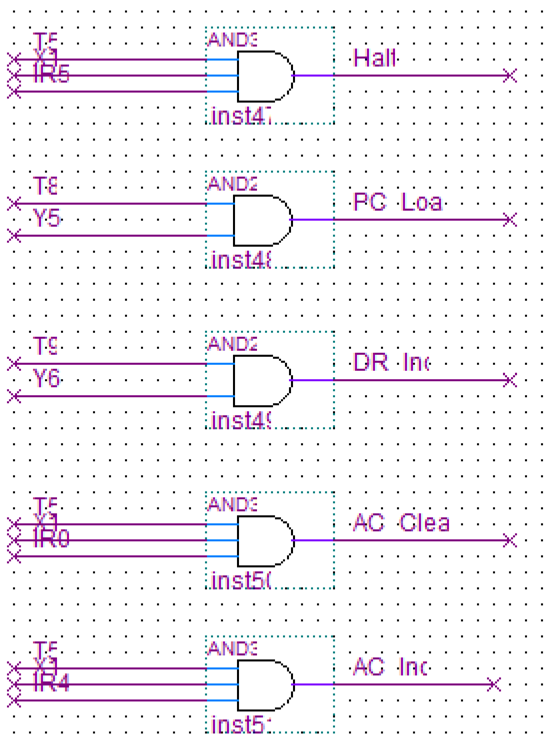


Figure 14 Logic Circuit of the Halt, PC Load, DR Increment, AC Clear, AC Increment

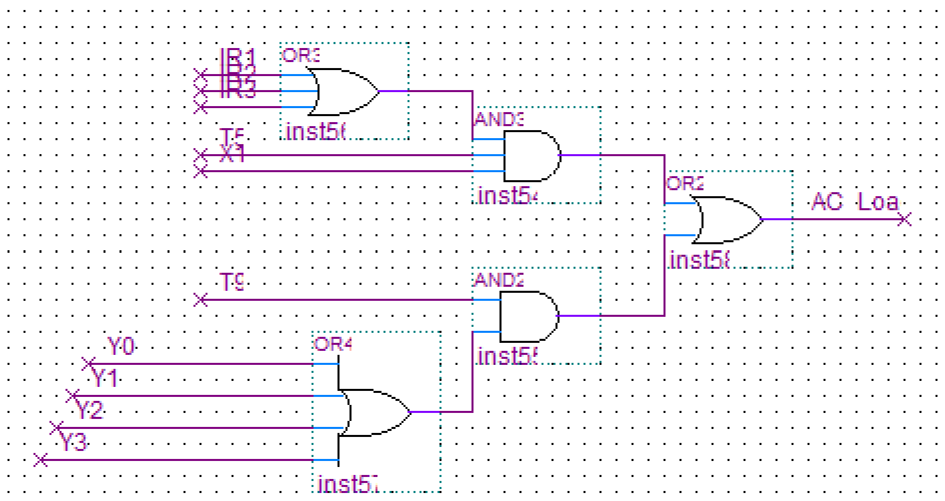


Figure 15 Logic Circuit of the AC load

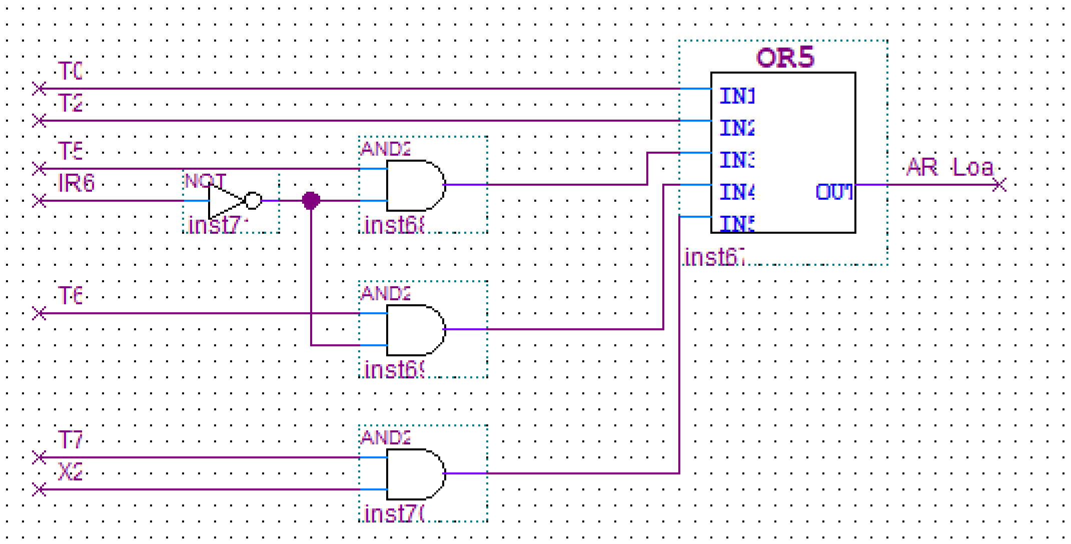


Figure 16 Logic Circuit of the AR load

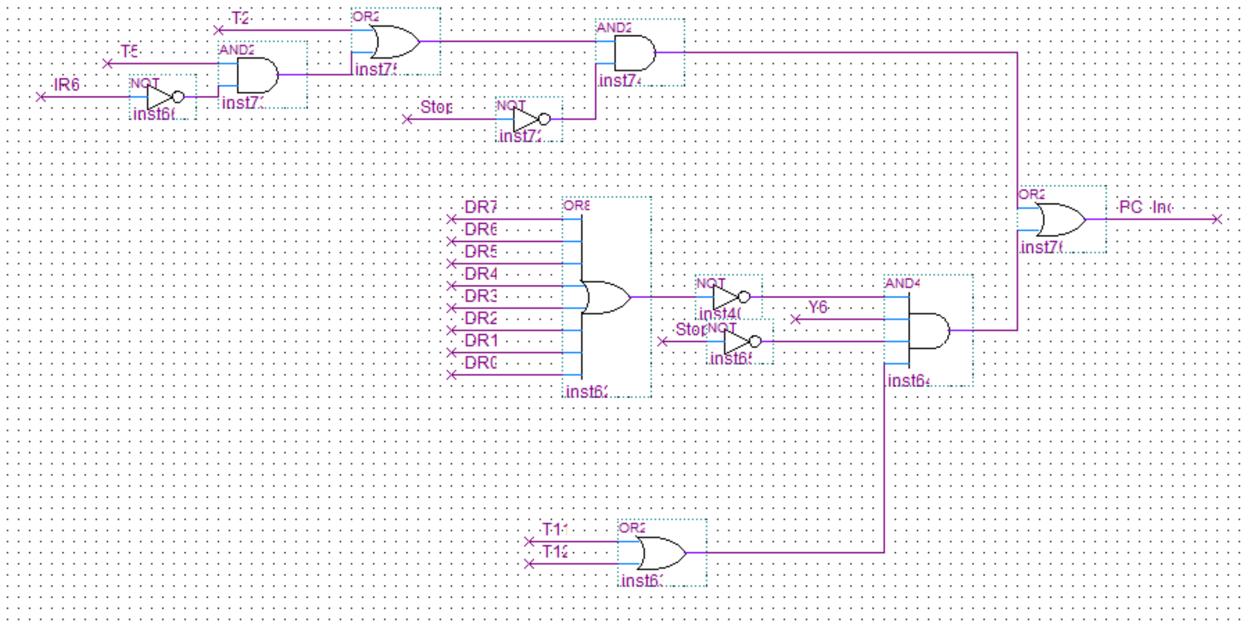


Figure 17 Logic Circuit of the PC Increment

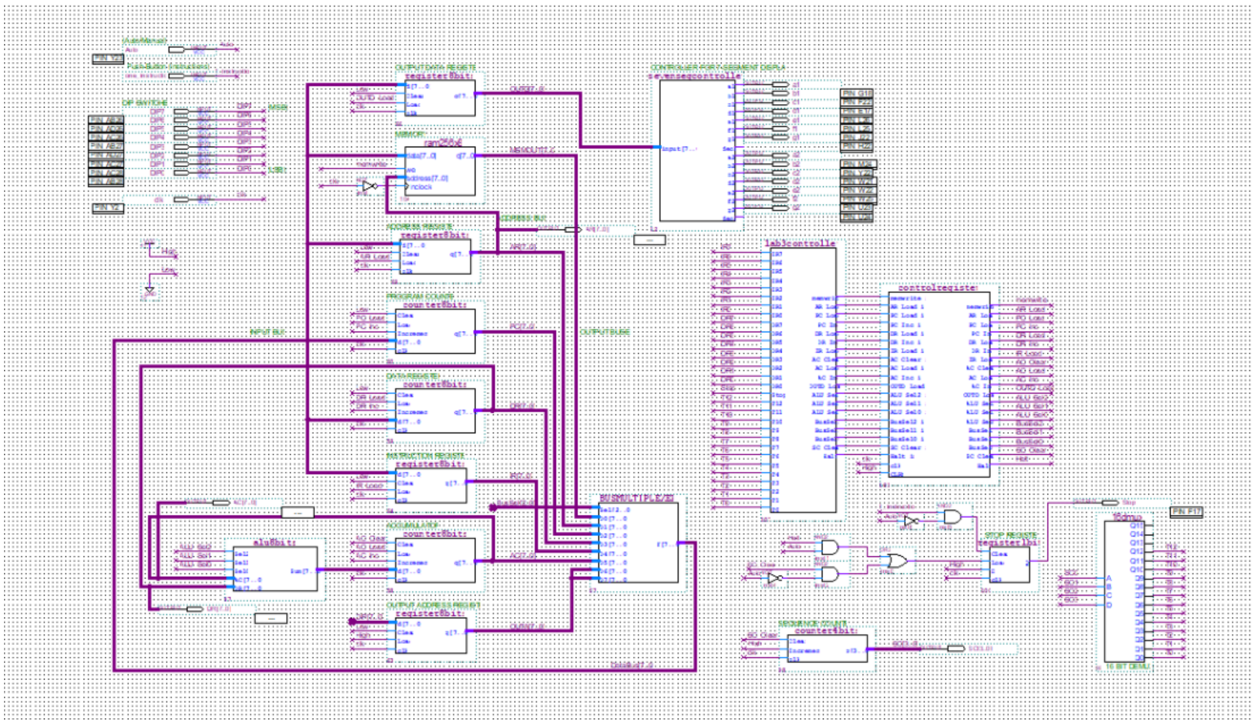
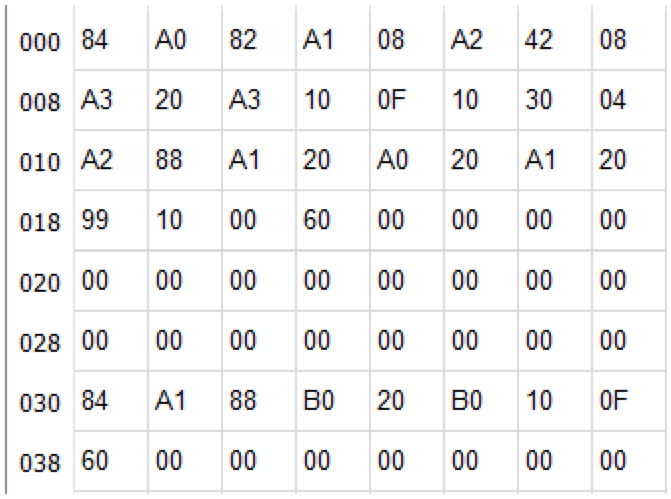


Figure 18 Logic Circuit of the Lab3Top.bdf File



000	84	A0	82	A1	08	A2	42	08
008	A3	20	A3	10	0F	10	30	04
010	A2	88	A1	20	A0	20	A1	20
018	99	10	00	60	00	00	00	00
020	00	00	00	00	00	00	00	00
028	00	00	00	00	00	00	00	00
030	84	A1	88	B0	20	B0	10	0F
038	60	00	00	00	00	00	00	00

Figure 19 Main Program and Subroutine

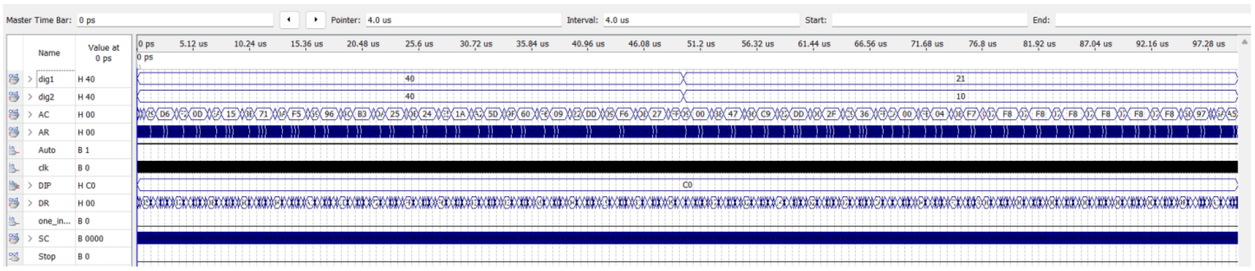
080	21	B5	37	08	5C	84	A1	1D
088	72	FF	F6	43	03	A9	D4	19
090	31	D9	47	82	14	52	07	CA
098	04	DB	00	00	00	00	00	00
0a0	80	81	00	00	00	00	00	00
0a8	00	00	00	00	00	00	00	00
0b0	C0	00	00	00	00	00	00	00

Figure 20 Operands, Counter and Pointers

Simulation/Verification

Both simulations were able to demonstrate the necessary values of D9 and CA, Demonstrating a successful version of the was achieved. The pictures below demonstrate the successful waveform.

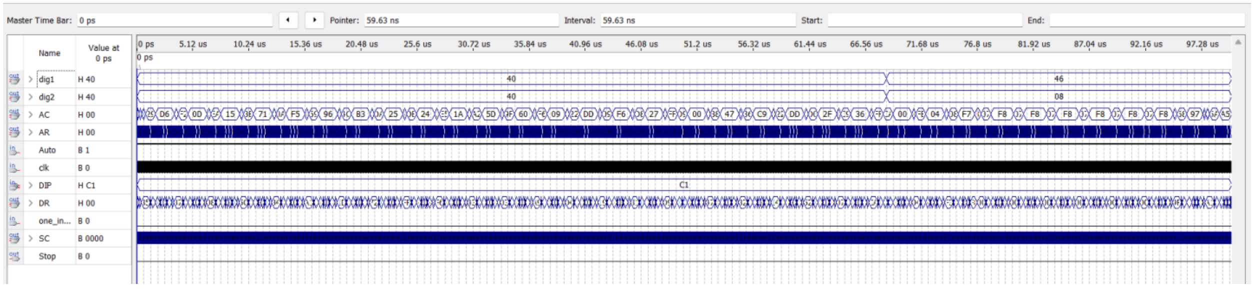
Note* If images are hard to see please compile and check on the added Zip file in submission.



Master Time Bar: 0 ps Pointer: 4.0 us Interval: 4.0 us Start: End:

Name	Value at 0 ps
dig1	H 40
dig2	H 40
AC	H 00
AR	H 00
Auto	B 1
clk	B 0
DIP	H C0
DR	H 00
one_in...	B 0
SC	B 0000
Stop	B 0

Figure 21 Waveform Showing the Storage of the Operand Leading to a 00 Sum



Master Time Bar: 0 ps Pointer: 59.63 ns Interval: 59.63 ns Start: End:

Name	Value at 0 ps
dig1	H 40
dig2	H 40
AC	H 00
AR	H 00
Auto	B 1
clk	B 0
DIP	H C1
DR	H 00
one_in...	B 0
SC	B 0000
Stop	B 0

Figure 22 Waveform, Showing the Storage of the Second (and Final) Operand Leading to a 00 Sum

Discussions/Conclusion

The goal of this lab was to analyze and understand the structure of a basic computer. Through our understanding, we derive the equations for the control unit and design the latter. Another goal was to write a program using machine code language. We manage to complete our objectives as we started to understand a more developed nature of the structure of a basic computer. We did derive the equations and design the control unit with the proper outputs. During our design of the control unit, we used symbols (bdf files) to simplify our design of the control unit. In the beginning, there was an error with the Bus select with the selection of the inputs. This was quickly sorted out and managed with the virtual connections of inputs and outputs to help organize the view much better. Finally, the software and hardware parts were fully functional and were able to get the required outputs. In conclusion, this lab was successful in achieving all the points of the objectives and created a successfully functioning computer.