Lab 4: BSO Basic Computer Organization

CEG2136 Section B03 (B4O2)

Group 30

Facilitating TA : Surbhi
Course Professor: V.Groza

Date of Experiment: November, 28th 2023
Date of Submission: December, 5th 2023

Department of Computer Engineering
University of Ottawa

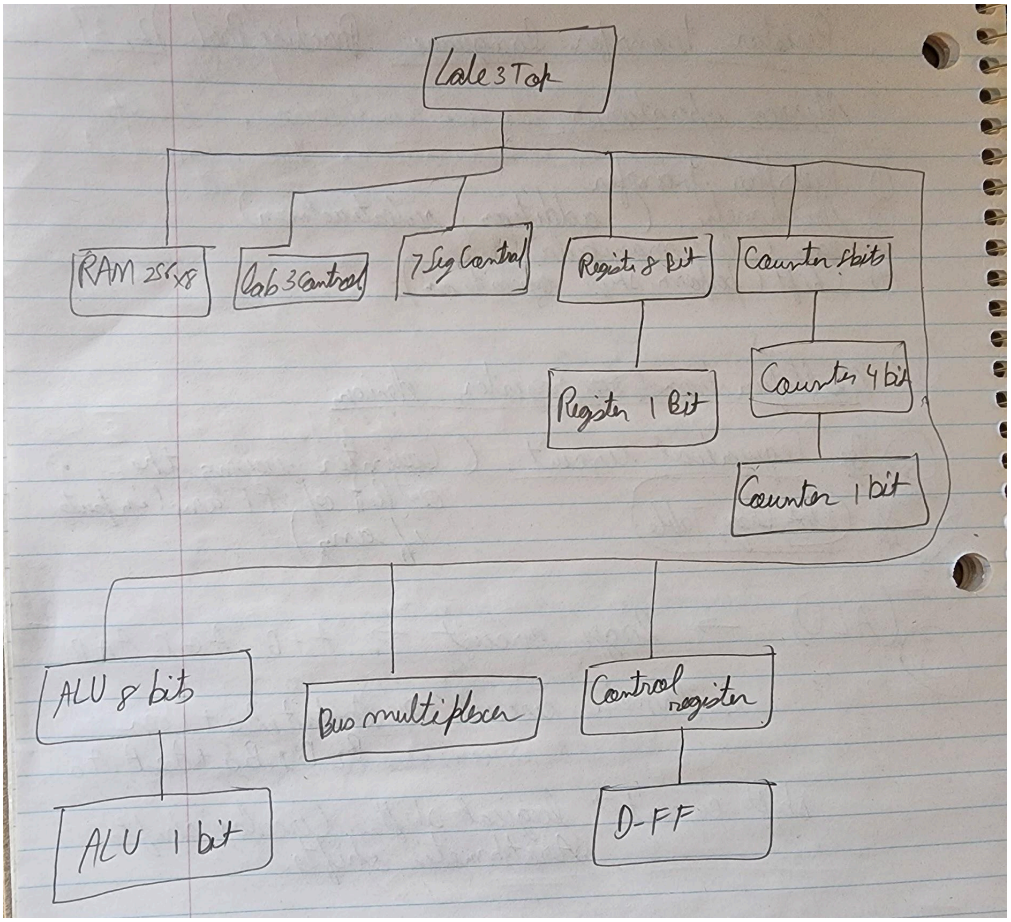
Lab Objectives:

- To assist students in understanding and analyzing the basic structures of computers and control.
- To understand how to utilize ‘opcodes’ and machine code to perform a program analysis and design.
- To initiate the students who are not familiar with the Altera Quartus II Design Software and the Altera FPGA based DE2-115 platform.
- To design, build and test an ALU (Arithmetic Logic Unit).
- To understand the operation of control signals and switching between micro operations.
- To compile, simulate, debug, and test their design.

Required Equipment:

- Quartus II Software
- Altera DE2-115 Board
- USB Blaster Cable

Preface/Prelab:

1. 

2. The bus multiplexer converts 7X1 output with 8 bits (0-7) and then the output register places the output on the databus one at a time.

3. In this specific case, the register reset will have to be sync, because it needs to pass through a DFF that is attached to a clock.

4. It depends on the priority of the load and reset. If priority levels are the same then we would have a result of undefined.

5. The address resistor is connected to the memory because we need address to access data and write data to the memory.

6. PC is implemented as a counter to sequentially fetch instructions from memory, Data resistor is implemented as a counter to support operations that involve iterating through a sequence of data. Accumulator is implemented as a counter to facilitate accumulation, which is common in many mathematical operations.

7. Reset has the highest priority as it is essential to clear and initialize the counters before they start any operation.
   Load has medium priority as it is used to load a specific value into the counter, preparing it for a new operation.
   Increment has least priority of all because once the counters are cleared or loaded, they can increment sequentially during normal operation.

8. NO. In this specific case the accumulator only will receive data from the Arithmetic Logic unit.

9.

Table:

Se2	**Sel**	**Se0**	**OPER.**
0	0	0	X + Y + CI
0	0	1	X + Y' + CI
0	1	0	X LS
0	1	1	X RS
1	0	0	X AND Y
1	0	1	X OR Y
1	1	0	Y
1	1	1	X

IN this case, all the SO are Arithmetic.

Prelab

Section 5.2

Memory (M(In, T))

1. memwrite : T9Y4 + T10Y6

CPU registers (R(In, T))

2. AR_Load : T0 + T2 + (T5+T6)IR6' + T7X2
3. PC_Load : T8Y5
4. PC_Inc : T2S' + T5IR6'S' + Y6DR'S'(T11+T12)
5. DR_Load : T8(Y0 + Y1 + Y2 + Y3 + Y6)
6. DR_Inc : T9Y6
7. IR_Load : T3
8. AC_Clear : (T5)(X1)(IR0)
9. AC_Load : T5X1(IR1 + IR2 + IR3) + T9(Y0 + Y1 + Y2 + Y3)
10. AC_Inc : T5.X1.IR4
11. OUTD_Load : T1

CPU ALU (ALU(In, T))

12. ALU_Sel2 : T9Y0 + T9Y3 + T5(X1)(IR1)
13. ALU_Sel1 : T5X1(IR2) + T5(X1)IR3 + T5(X1)(IR1) + T9Y3
14. ALU_Sel0 : T9Y2 + T5X1(IR3 +IR1)

Bus (data mux (Bus(In, T))

15. BusSel2 : T0 + T9Y4
16. BusSel1 : T10Y6 + T0 + T2 + T5
17. BusSel0 : T10Y6 + T9Y4 + T8Y5

Control Unit ((In, T))

18. SC_Clear : T5X1 + T9(Y0 + Y1 + Y2 + Y3 + Y4) + T8Y5 + T12Y6
19. Halt : T5.X1.IR5

Design - P1-6:

Below are screenshots of our completed schematic diagrams. These include the controller part with all of the control signals and gates added. The file was designed in spec to the table provided in the lab introduction manual.

Figures 1-3 Show the basic computer controller.

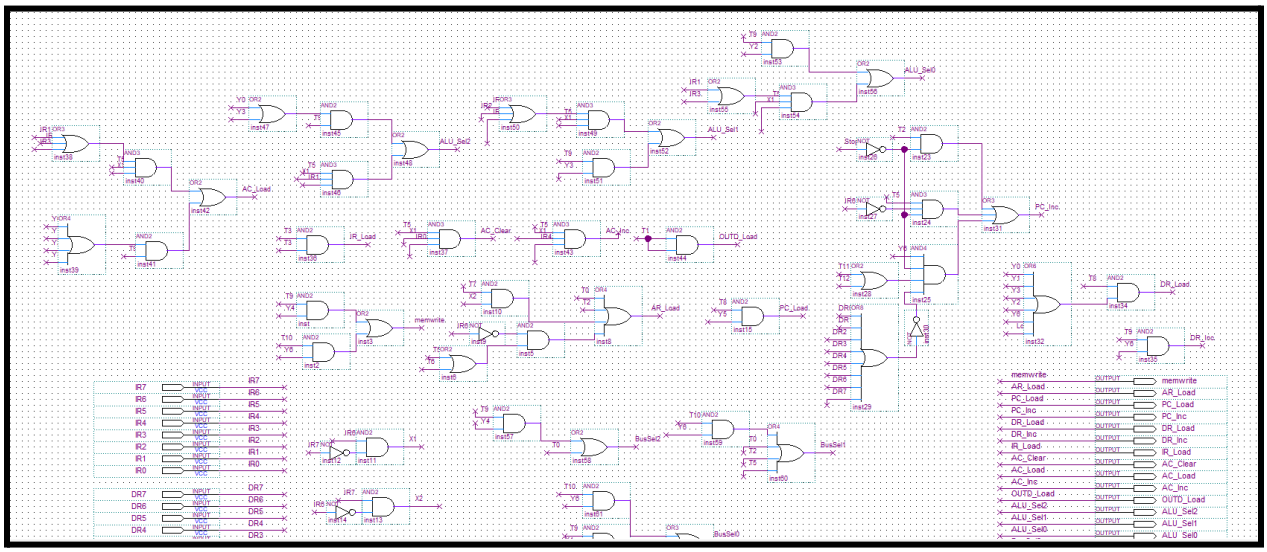


Figure 1

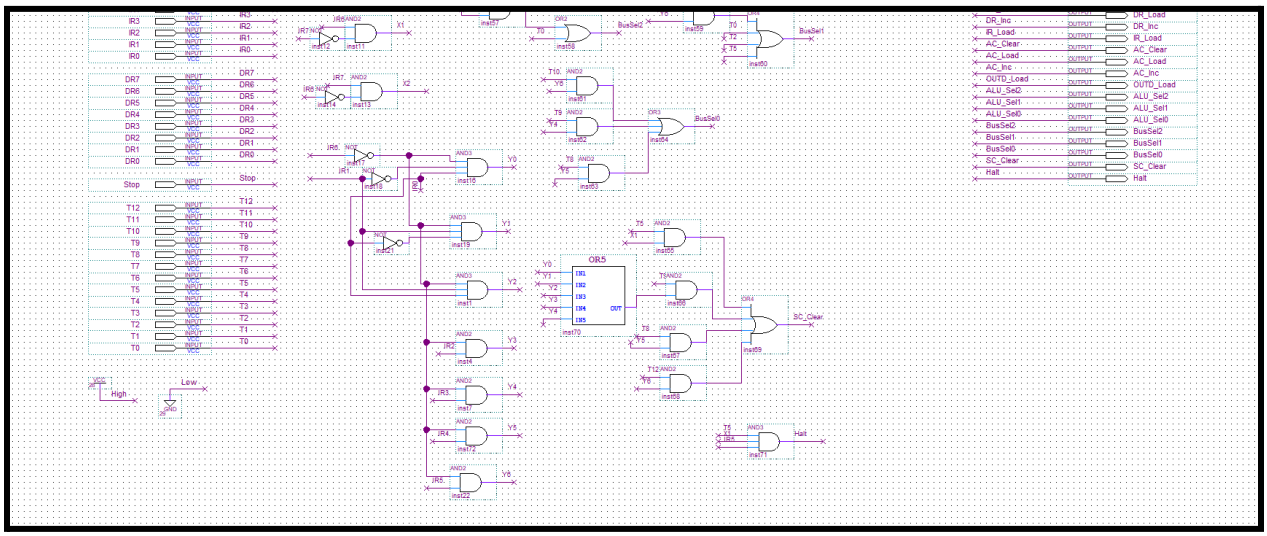


Figure 2 - Part 2 of the Circuit

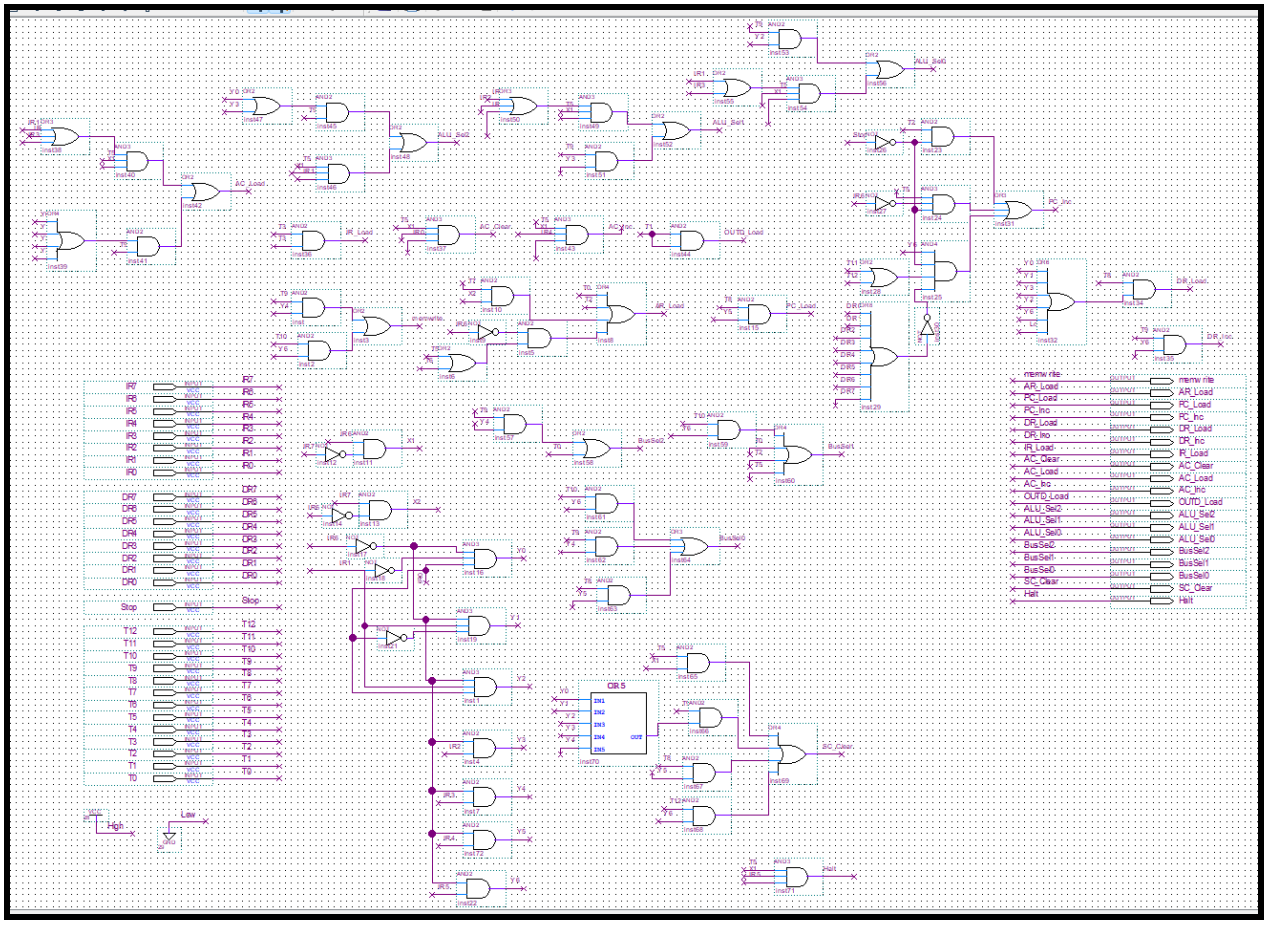


Figure 3 - Completed Block Diagram (TOP LEVEL ENTITY)

Analysis:

At this portion of the lab, we got our control unit from the table in the lab manual. We can now simulate and test if we could add numbers together. Functional simulations were run. Below is the simulation with A0 as DIP switches, this was later added to the altera board.

To ensure our control unit was functioning properly, the display on the board should be 5F, if the inputs are A0 and FA, if the inputs are A1, this was accomplished when we tested our program. Since our hardware portion of the lab was done correctly, we could proceed to the software portion.

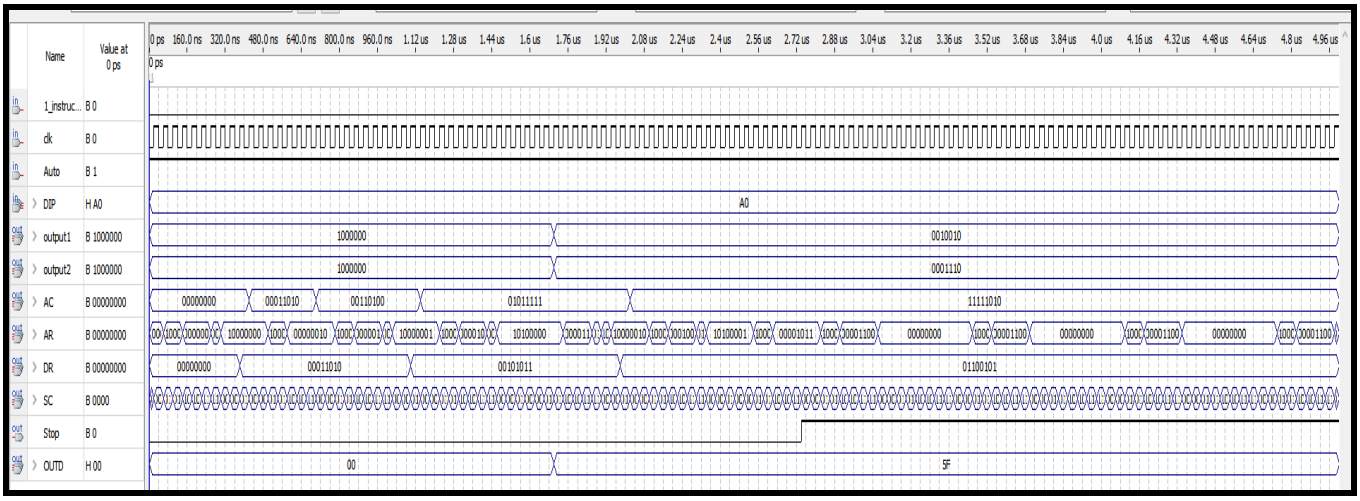


Figure 4- Simulation with A0 as DIP switches

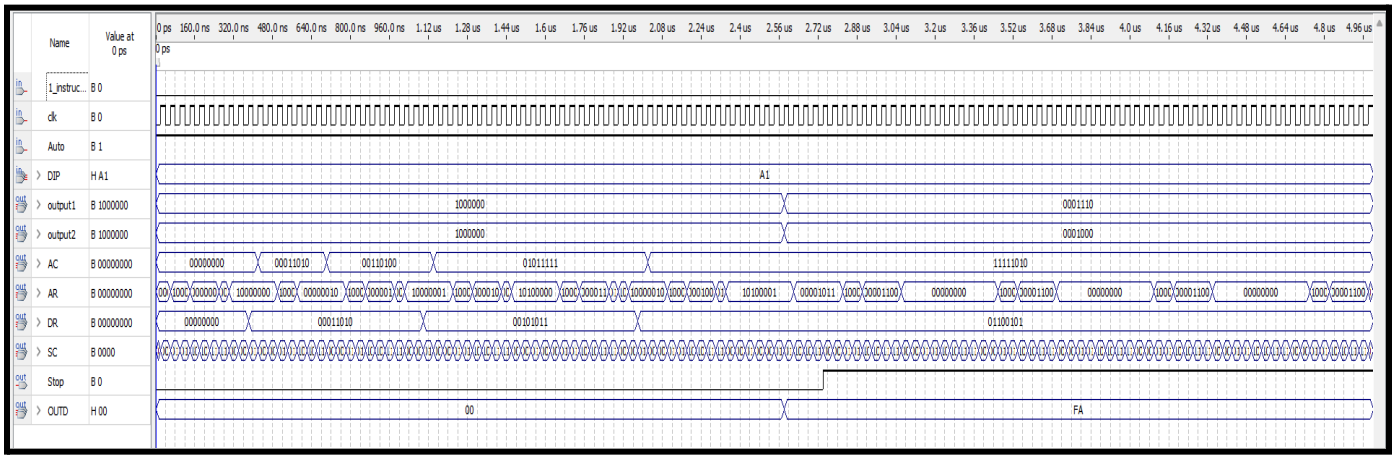


Figure 5 - Simulation with A1 DIP switches

Table 6: Pins Assignment

Pin Name	Pin Number	Component	Pin Name	Pin Number	Component
clk	PIN_Y2	50MHz clock	A1	PIN_G18	HEX0[0]
Auto	PIN_Y23	SW[17]	B1	PIN_F22	HEX0[1]
DIP7	PIN_AB26	SW[7]	C1	PIN_E17	HEX0[2]
DIP6	PIN_AD26	SW[6]	D1	PIN_L26	HEX0[3]
DIP5	PIN_AC26	SW[5]	E1	PIN_L25	HEX0[4]
DIP4	PIN_AB27	SW[4]	F1	PIN_J22	HEX0[5]
DIP3	PIN_AD27	SW[3]	G1	PIN_H22	HEX0[6]
DIP2	PIN_AC27	SW[2]	A2	PIN_M24	HEX1[0]
DIP1	PIN_AC28	SW[1]	B2	PIN_Y22	HEX1[1]
DIP0	PIN_AB28	SW[0]	C2	PIN_W21	HEX1[2]
1 instruction	PIN_M23	KEY[0]	D2	PIN_W22	HEX1[3]
AR[0]	PIN_G19	LEDR[0]	E2	PIN_W25	HEX1[4]
AR[1]	PIN_F19	LEDR[1]	F2	PIN_U23	HEX1[5]
AR[2]	PIN_E19	LEDR[2]	G2	PIN_U24	HEX1[6]
AR[3]	PIN_F21	LEDR[3]	AC[0]	PIN_E21	LEDG[0]
AR[4]	PIN_F18	LEDR[4]	AC[1]	PIN_E22	LEDG[1]
AR[5]	PIN_E18	LEDR[5]	AC[2]	PIN_E25	LEDG[2]
AR[6]	PIN_J19	LEDR[6]	AC[3]	PIN_E24	LEDG[3]
AR[7]	PIN_H19	LEDR[7]	AC[4]	PIN_H21	LEDG[4]
DR[0]	PIN_J15	LEDR[10]	AC[5]	PIN_G20	LEDG[5]
DR[1]	PIN_H16	LEDR[11]	AC[6]	PIN_G22	LEDG[6]
DR[2]	PIN_J16	LEDR[12]	AC[7]	PIN_G21	LEDG[7]
DR[3]	PIN_H17	LEDR[13]	Stop	PIN_F17	LEDG[8]
DR[4]	PIN_F15	LEDR[14]			
DR[5]	PIN_G15	LEDR[15]			
DR[6]	PIN_G16	LEDR[16]			
DR[7]	PIN_H15	LEDR[17]			

Compile your project.

Figure 6 - Pin Assignment

Design - P2-PreLab:

Part 1: analyze the code given

```
%  PROGRAM IS IN the RANGE Of ADDRESSES 00 TO 7F %
00: 04;    %  LDA (direct)  %
01: a0;    %  from address a0 %
02: 42;    %  CMA  %
03: 08;    %  STA (direct)  %
04: a0;    %  AC to address a0  %
05: 20;    %  ISZ (direct)  %
06: a0;    %  counter stored at a0  %
07: 10;    %  BUN (direct)  %
08: 20;    %  to address 20  %
09: 60;    %  HLT  %
20: 84;    %  LDA (indirect)  %
21: a1;    %  the number pointed to by the memory location a1  %
22: 82;    %  ADD (indirect)  %
23: a2;    %  the number pointed to by the memory location a2  %
24: 88;    %  STA (indirect)  %
25: a3;    %  to the memory location pointed to by a3  %
26: 04;    %  LDA (direct)  %
27: a1;    %  from the address a1  %
28: 50;    %  Inc  %
29: 08;    %  STA (direct)  %
2a: a1;    %  AC to the memory address a1  %
2b: 50;    %  Inc  %
2c: 08;    %  STA (direct)  %
2d: a2;    %  store AC to the memory address a2  %
2e: 50;    %  Inc  %
2f: 08;    %  STA (direct)  %
30: a3;    %  send AC to the memory address a3  %
31: 10;    %  BUN (direct)  %
32: 05;    %  to the memory address 05  %
80: 01;    %  DATA ARE FOUND AT ADDRESSES 80 TO FF  %
81: 01;
a0: 0a;    %  loop counter, which will be done 10 times  %
a1: 80;    %  pointer to the first number to be added  %
a2: 81;    %  pointer to the second number to be added  %
a3: 82;    %  pointer to memory location where result will be stored %
```

Figure 7 - Lab Report 4 Code

1. Pseudo code:

//counter is initialized with value 10.
Counter = Load from A0

while Counter != 0: //loops for 10 times
    X = Load from A1
    Y = Load from A2
    Z = X + Y
    Increment the values at A1, A2, and A3
    Counter--

Halt

2. What does the program calculate?

- The program appears to perform a loop that iteratively adds the values pointed to by X and Y and stores the result in the memory location pointed to by Z. This loop runs 10 times.

3.Why use memory-reference instructions with indirect addressing?

- Memory-reference instructions with indirect addressing (using pointers) provide flexibility and efficiency in accessing and manipulating data stored in memory. In this program, using pointers allows the program to operate on values at different memory locations without explicitly specifying the addresses, making the code more adaptable and easier to modify. It enables the program to perform operations on variables indirectly through pointers, which is essential for dynamic data manipulation and efficient memory utilization.

Section 7.2:

Design the program description below:

Write a program which adds consecutively each number of the following sequence of hexadecimal numbers: 21, B5, 37, 08, 5C, 84, A1, 1D, 72, FF, F6, 43, 03, A9, D4, 19, 31, D9, 47, 82, 14, 52, 07, CA, 04. When your current sum becomes equal to zero, your program should store into the memory the last number added, display this number, and eventually stop. Write the program in machine code in .mif file format, as shown in section 4.2.3

Software code:

```
% This program is in address zone of  00-A8%

00: 84  % LDA (indirect) %
01: A5  % Load Accumulator (LDA) indirectly from the memory location pointed to by A5 %
02: 08  % Store Accumulator (STA) to the direct address pointed by A6 %
03: A6  % Increment the value at the direct address pointed by A6 %
04: 04  % Load Accumulator (LDA) from the direct address pointed by A5 %
05: A5  % Load the value from the memory address A5 %
06: 50  % Increment (INC) the value at the direct address pointed by A5 %
```

07: 08 % Store Accumulator (STA) to the direct address pointed by A5 %
08: A5 % Load Accumulator (LDA) indirectly from the memory location pointed to by A5 %
09: 84 % Load the value from the memory location pointed to by A5 %
0A: A5 % ADD (indirect) the value from the memory location pointed to by A6 %
0B: 02 % Store Accumulator (STA) to the direct address pointed by A7 %
0C: A6 % Increment the value at the direct address pointed by A7 %
0D: 08 % Store Accumulator (STA) to the direct address pointed by A7 %
0E: A7 % Increment and Skip if Zero (ISZ) the value at the direct address pointed by A7 %
0F: 20 % Counter stored at A7 %
10: A7 % Branch Unconditionally (BUN) to the direct address 03 %
11: 10 % BUN to the direct address 03 %
12: 03 % Jump to the instruction at address 03 %
13: 84 % Load Accumulator (LDA) indirectly from the memory location pointed to by A5 %
14: A5 % Load the value from the memory address A5 %
15: 08 % Store Accumulator (STA) to the direct address pointed by A8 %
16: A8 % Store the value to address A8 %
17: 60 % Halt (HLT) %

%data starts from 81 to 100 in HEX (51 - 74)%
% data is in order 21, B5, 37, 08, 5C, 84, A1, 1D, 72, FF, F6, 43, 03, A9, D4, 19, 31, D9, 47, 82, 14, 52, 07, CA, 04.%

A8: 00 %Stores the last added number%

8. Simulation (Test the program)

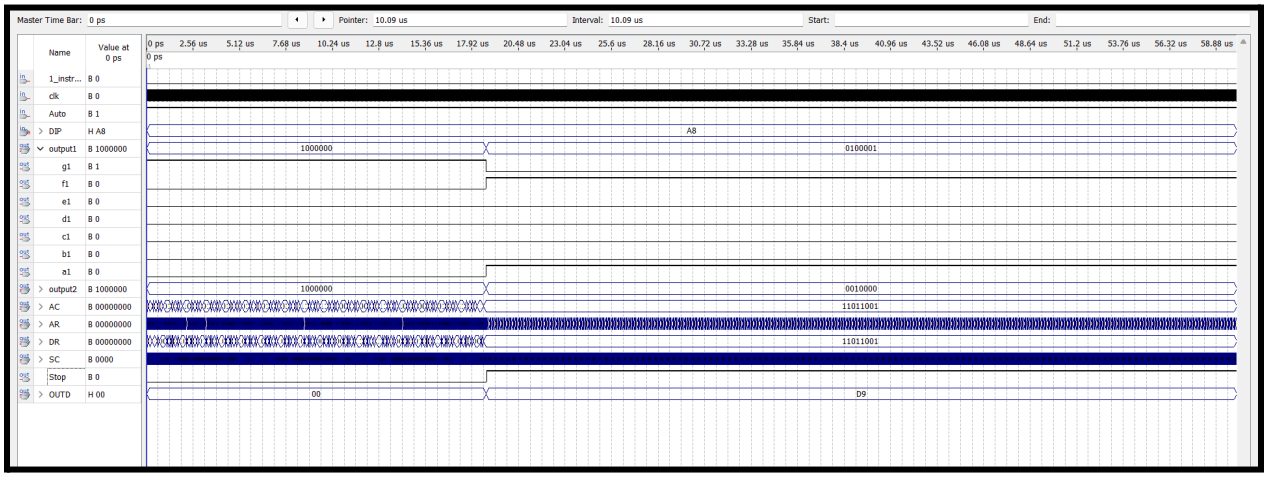


Figure 8 - Simulation of software code

Result: After performing the simulation by creating the waveform we got D9 as the last added number before the program halt due to the sum becoming zero.

Conclusion/Discussion:

In this lab, we were successful in understanding the basic operations and structure of a computer. We got to do hands-on work on designing a control unit from the table given to use in the lab manual. The control unit (hardware part) was verified in this lab by performing basic addition, we had switches in the program set to A0 which would then provide us with a result. The control unit gives us a basic computer to work with and to further use opcodes on creating simple programs. We understood how to analyze and simulate software code. As shown above, the end result is for the program to get tested and a result from the segment was D9, which is the answer.

We did run into multiple errors during the lab, however the teaching assistants assisted us in ensuring we understood the mistakes and gave us tips on moving forward which was very essential for us in having a successful lab result.