Lab 4 – Basic Computer Organization

CEG 2136 – Computer Architecture I

Fall 2022

University of Ottawa

Lab TAs: Reza & Yang

Jaiden Clee 300174453

Jasmin Cartier 300160723

Experiment Dates: 11/10/2022, 11/17/2022, 11/24/2022, 12/01/2022

Table of Contents

Introduction 2

Design 2

Simulation and Verification of Design 3

Discussion 3

Appendix 5

a. Figure 1a: Lab 3 controller circuit diagram
b. Figure 1b: Lab 3 controller circuit diagram
c. Figure 2: Lab 3 Controller fitter summary
d. Figure 3: Lab 3 Top fitter summary
e. Figure 4: Lab 3 Top waveform simulation where DIP points to A0
f. Figure 5: Lab 3 Top waveform simulation where DIP points to A1
g. Figure 6: Memory contents file
h. Figure 7: Addition software program waveform simulation
i. Figure 8: Memory contents file containing instructions for addition
j. Figure 9: Multiplication software program waveform simulation
k. Figure 10: Memory contents file containing instructions for the multiplication of 4 and 5

Introduction

The purpose of this lab was to implement and test a basic computer using Quartus II and the Altera DE2-115 board. After analyzing the given files for this lab, the control signal equations were derived and the control unit was designed. The control unit takes input from the Instruction Register, the Data Register, and the State Register, and decides which instruction needs to be executed. After testing the control unit, it was used to implement simple machine code programs. Opcodes were used to write an addition program and a multiplication program. These programs were written in a memory contents file which was used to perform waveform simulations on the basic computer. By the end of this lab, a fully functional basic computer was designed, tested, and used to display programs on the Altera board.

Design

For this lab all components of the computer were given except the Instruction Decoder. It was implemented in the file lab3controller which is shown in Figure 1. The controller takes as inputs the IR (Instruction Register) and the DR (Date Register); they are both 8 bit inputs. The controller also takes the SC (State Register) inputs that give the computer's present state between T0 and T12. Based on those inputs, nineteen functions determine the control signal that is essential for the proper functioning of the computer. Memwrite is used each time something has to be written in the memory. This function tells the register in charge of the memory to load what is on the DataBus into the address shown by the AR (Address Register). AR_Load, PC_Load, DR_Load , IR_Load, AC_Load, and OUTD_Load tell the component to load what is on the DataBus. PC_Inc, DR_Inc and AC_Inc increment the value of PC (Program Counter), the DR, and the AC (Accumulator) respectively. The Accumulator is a register used to temporarily hold the data while it is being manipulated. AC_Clear and SC_Clear are used to clear the information in AC or SC. The controller also uses a 3-bit selection (BusSel0..2) to select what the DataBus is going to show between the Memory, the AR, PC, DR, IR, AC or OUTA. The OUTA is used to send the DIP switch inputs to the DataBus. The DIP switches are set to provide an 8-bit signal that corresponds to a memory location. The contents of this memory location are then stored in OUTD (Out Data Register) and displayed on the 7 segments display. OUTA, OUTD, and the DIP switches are used in this lab for testing purposes. Finally, another 3-bit selection is used to control the ALU.

The functions described above are derived from the tables present in the lab instructions. The instructions also indicate when each function should take place in the Instruction Fetch Cycle of the basic computer. For example; the Halt function is activated when T5, IR6, IR5 are on and IR7 is off. In the implementation in Figure 1a in the bottom right corner you can see that NIR7 is used to be sure that IR7 is really off. NIR7 is simply the opposite of IR7 as you can see on the top left corner we use the same method for Stop and IR6. Minimal complications were encountered when deriving the implementation of each function. One problem occurred when we tried to simplify a logic expression unsuccessfully. To solve the problem we re-implement that function without simplification.

After deriving the control signal equations, we designed a program in machine code that would run on the basic computer. The goal of this program was to add together a series of hexadecimal numbers. When the current sum reaches zero, the program should display the last number added and then halt. The complete program is shown in Figure 8. In our program the current sum is stored in the memory location 20H and the last element added is stored in the memory location 22H. The program starts by loading the contents of the memory location where the sum is stored into the AC. At the start the sum is zero, so the first number to be loaded into the AC is zero. Then, we indirectly add the contents of memory location 20H to the AC. This allows you to stock a list of numbers somewhere in the computer and put the address of the first element of the list in the memory location 20H. By incrementing the address in memory location 20H you have access to the next element in the list; this is a direct incrementation. During the revision of the program this was changed from an indirect incrementation to a direct one to avoid mistakes because the purpose is to increment the address to have the next value of the list and not increment the value in the list. After storing the AC with the new sum, we complement the AC and store its complement in a memory location that we will call Check. AC then loads indirectly the value of memory location 20H and stores the value in a memory location that we will call LastElem. At this moment, the memory cell 20H can be incremented with the command ISZ; there is no fear of skipping the next line with this command here because the value in memory cell 20H will always stay positive and never reach zero. The same command is used after on the memory cell Check to see if the sum reaches zero. If it's the case when the complement of the sum was stored in Check it should have been 11111111 instead of 00000000 the value of sum. Notice that Check is then the 2's complement of -1, by calling ISZ it will increment it to zero and this command skip the next line when the value incremented reaches zero. The next line is used to send the program to the first instruction; if it is not executed the program continues and ends with the command Halt. The DIP is then used to show the value of the LastElem memory location.

Simulation and Verification of Implemented Design

To facilitate testing of our derived control signal equations, a machine code program was given which tested the instructions load, circular left shift, addition, store, subtraction, and halt. The program also used both direct and indirect addressing modes. After the given program was analyzed, it was written to the memory contents file seen in Figure 6. The breakdown of this program is as follows: First, the number stored in address 80H is loaded into the AC register. Second, the AC register contents are circularly shifted to the left. Next, the number stored in address 81H is added to the contents of the AC register. Then, the sum of this addition operation (currently being stored in the AC register) is stored in address A0H. Next, an indirect subtraction is performed where the number stored in 82H is subtracted from the contents of the AC register. Then, the difference of this subtraction operation (currently stored in the AC register) is stored in the address A1H. Lastly, the program is halted. After the completion of this program, the content of the memory location A0H is 5FH and the content of the memory location A1H is FAH.

To verify our implementation of the control signal equations, OUTA, OUTD and the DIP switches were used to help visualize data on a waveform simulation. To see the outputs of the given program, the DIP switches were set to the memory location where the program is going to store the final result. In this case the DIP switches were set to A0H and then set to A1H. The waveform simulations for both of these tests are shown in Figures 4 and 5. During the first simulation, the program wasn't triggering Stop output to 1 and the outputs were not the ones expected. After reviewing and modifying some of the logic functions in the Lab 3 controller file, Stop was triggered and OUTD was finally showing the correct results. The expected results were 5F and FA for the first and second simulation respectively.

For the bonus section, we wrote a machine code program that executed the multiplication of two unsigned 4-bit numbers. The program was written in a memory contents file and is shown in Figure 10. The program multiplies the number in memory location 21H with the number in location 22H and stores the result in location 20H. As shown on the waveform in Figure 9, the DIP switches were set to the memory location 20H to display the result of the multiplication operation. During verification, our program multiplied 4 and 5, and the OUTD output showed the expected result of 20.

Discussion

Firstly, the equations for the control signals generated by the control unit were derived. Equations for the signals controlling the memory, the CPU registers, the CPU ALU, the Bus, and the Control Unit were found and implemented in the Lab 3 controller design file (Figures 1 and 2). After implementing this circuit design, the basic computer was experimentally tested to make sure the control signal equations were correct.

During testing of the control signal equations, the 7-segment display was used to show the contents of a memory location specified by the DIP switches on the Altera board. Verification of the control signal equations was confirmed after completing two waveform simulations. As shown in Figures 4 and 5 respectively, when the DIP switches pointed to A0H the 7-segment display showed 5FH and when the DIP switches pointed to A1H the 7-segment display showed FAH. These were the expected results as described by the program above and therefore confirmed the control signal equations were correct.

After completing the verification of the derived control signal equations, the task was to write a program in machine code that could perform the addition of a given list of hexadecimal numbers. When the current sum of the addition became zero, the last number added would be stored in a memory location and displayed before the program halted. This program was written in a memory contents file and is shown in Figure 8. The program instructions start at the memory location 00H and the hexadecimal numbers to add are stored in memory locations 40H - 58H. To verify the program instructions, DIP switches were pointed to the memory address of the last number added and the contents of this address were shown on the 7-segment display. As expected, the waveform simulation in Figure 7 shows D9 was the last hexadecimal number added and this was stored in the memory address location 22H.

The last task of this lab was to write a program in machine code that could perform the multiplication of two unsigned 4 bit numbers. The result of this multiplication would be stored and shown on the 7-segment display. In this case, 4 and 5 were chosen as the two unsigned 4-bit numbers.This program was written in a memory contents file and is shown in Figure 10. The program instructions start at the memory locations 00H and the two unsigned 4-bit numbers to be multiplied are stored in the memory locations 21H and 22H respectively. The result of the multiplication operation was stored in the memory location 20H. The verification of this program was similar to the verification process of the addition program described above. As expected, the waveform simulation in Figure 9 shows the result of the multiplication is 20 and this was stored in the memory address location 20H.

In conclusion, this lab enhanced our understanding of the instruction fetch cycle in a basic computer. The knowledge gained from implementing and testing the Control Unit of this computer was used to further our understanding of machine code operations and how to write simple programs in memory content files. The process of designing, implementing, and testing the different components of a basic computer was fundamental knowledge that we will be able to build on in future studies.

Appendix

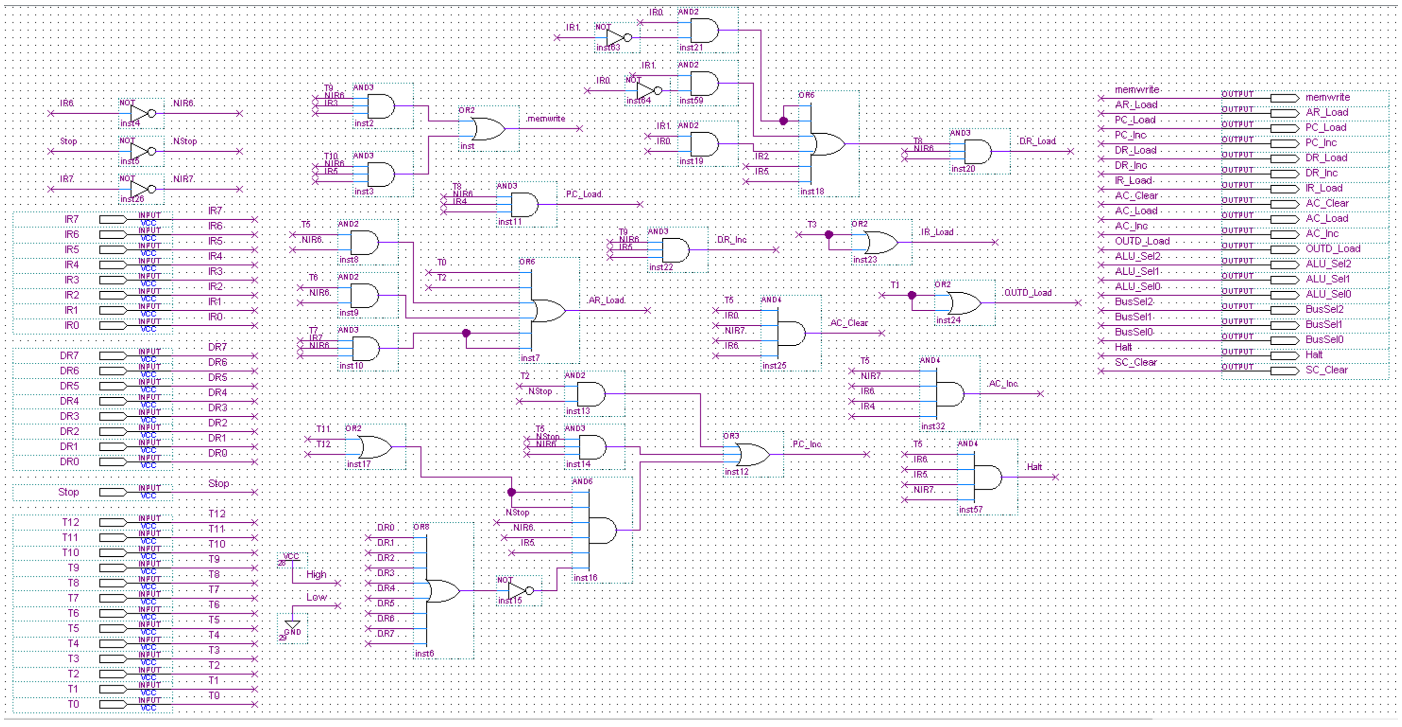


Figure 1a: Lab 3 controller circuit diagram (part 1)

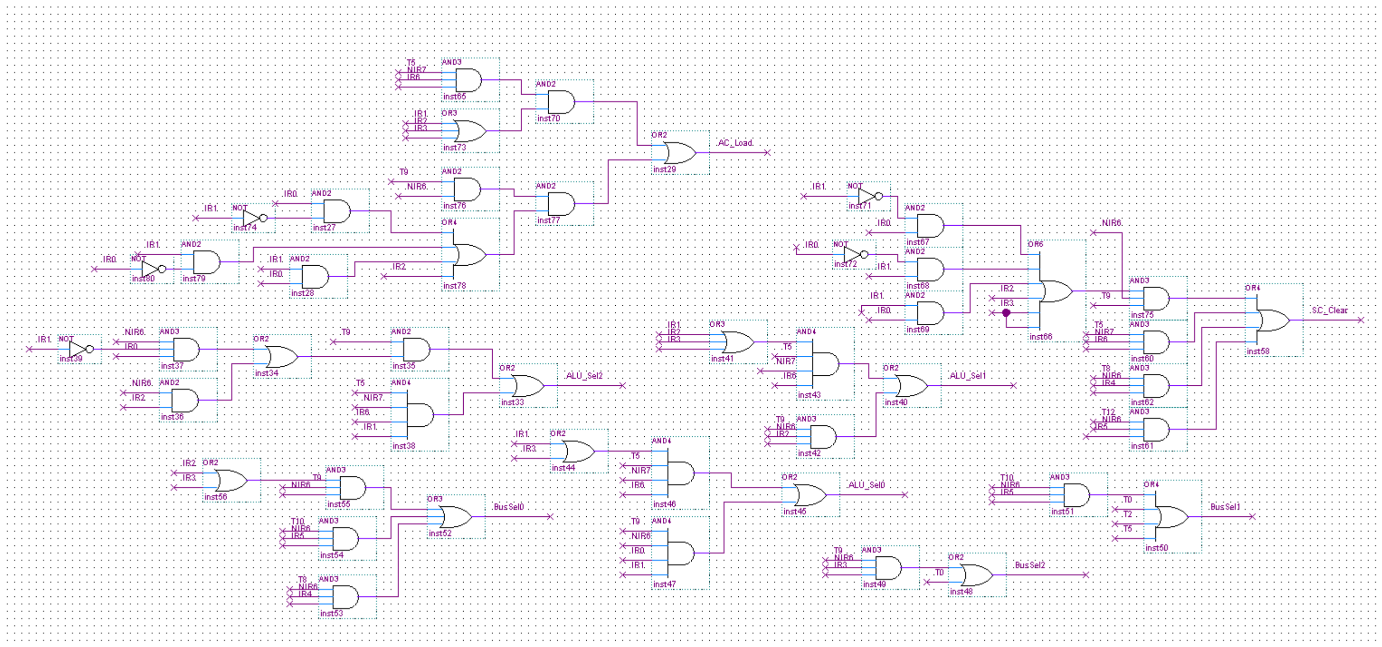
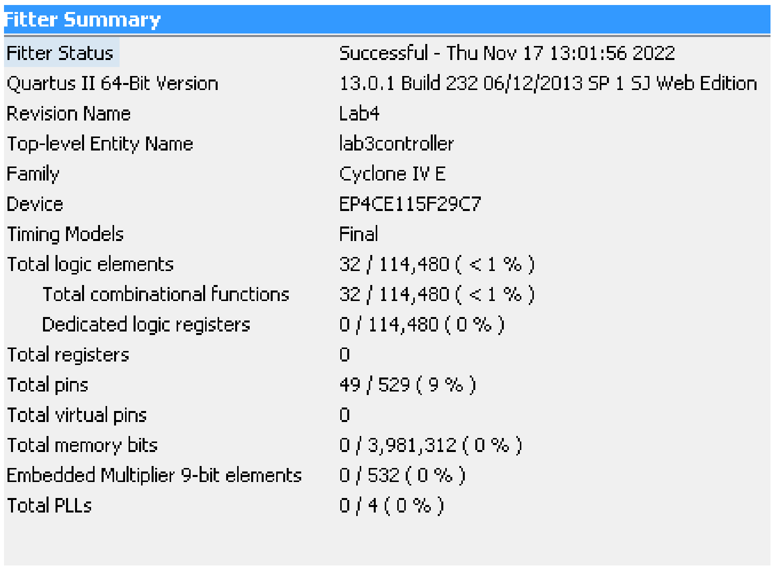


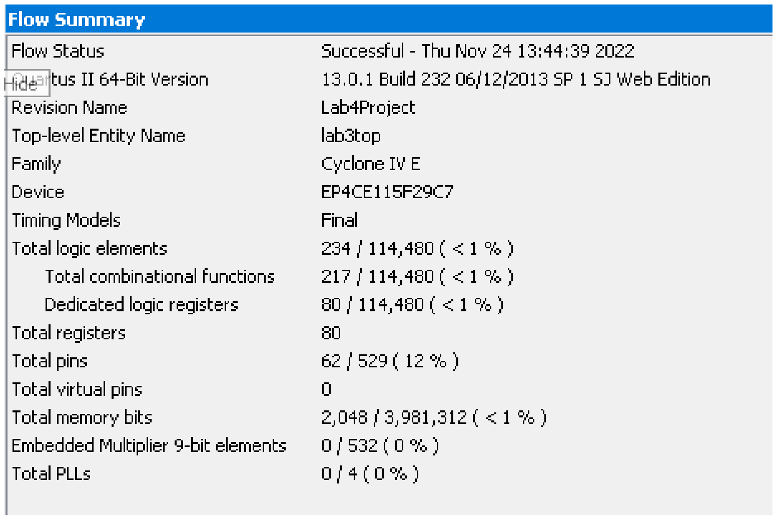
Figure 1b: Lab 3 controller circuit diagram (part 2)



Fitter Summary

Fitter Status	Successful - Thu Nov 17 13:01:56 2022
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Lab4
Top-level Entity Name	lab3controller
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	32 / 114,480 (< 1 %)
Total combinational functions	32 / 114,480 (< 1 %)
Dedicated logic registers	0 / 114,480 (0 %)
Total registers	0
Total pins	49 / 529 (9 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 2: Lab 3 Controller fitter summary



Flow Summary

Flow Status	Successful - Thu Nov 24 13:44:39 2022
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Lab4Project
Top-level Entity Name	lab3top
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	234 / 114,480 (< 1 %)
Total combinational functions	217 / 114,480 (< 1 %)
Dedicated logic registers	80 / 114,480 (< 1 %)
Total registers	80
Total pins	62 / 529 (12 %)
Total virtual pins	0
Total memory bits	2,048 / 3,981,312 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 3: Lab 3 Top fitter summary

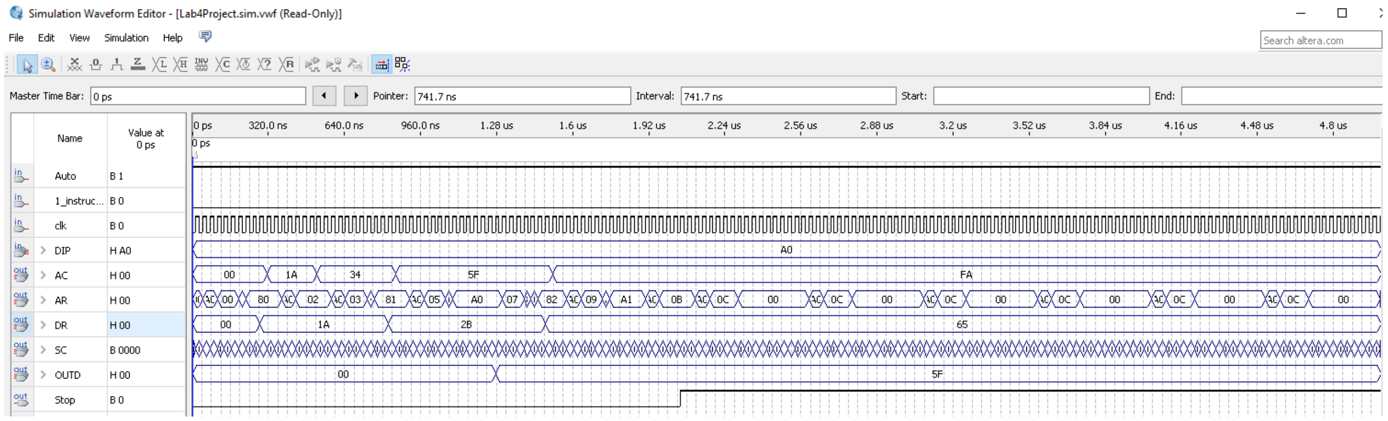


Figure 4: Lab 3 Top waveform simulation where DIP points to A0

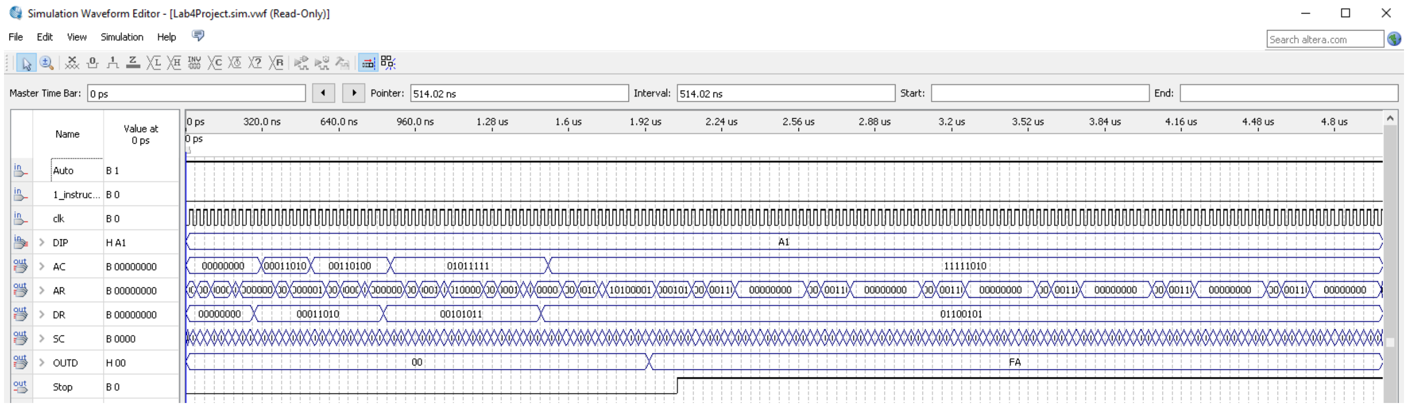
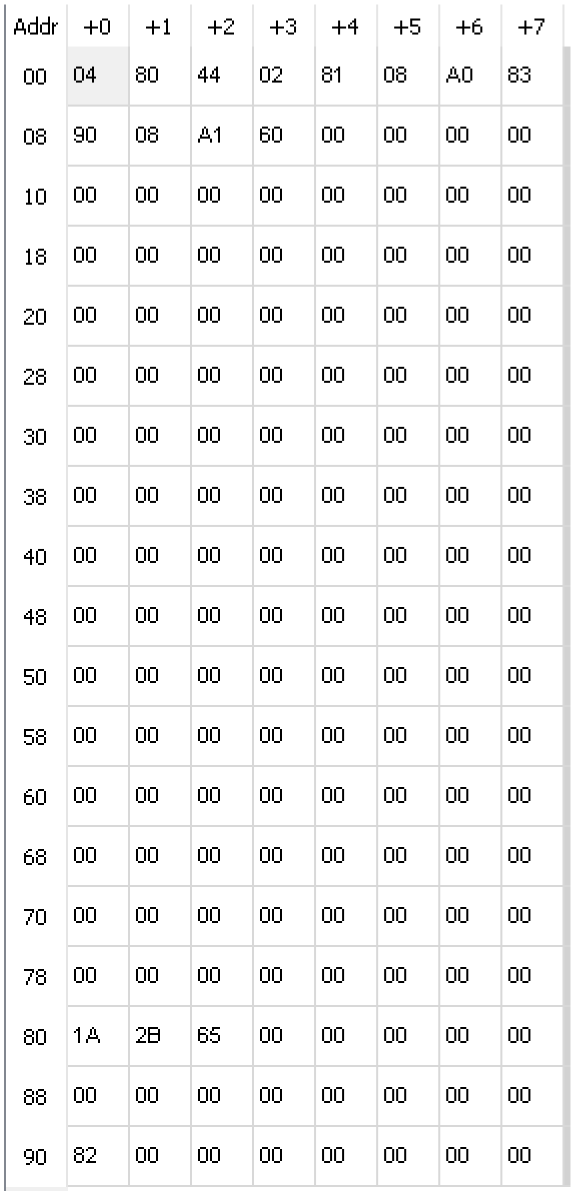


Figure 5: Lab 3 Top waveform simulation where DIP points to A1



Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	04	80	44	02	81	08	A0	83
08	90	08	A1	60	00	00	00	00
10	00	00	00	00	00	00	00	00
18	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00
28	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00
38	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00
48	00	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00	00
58	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00
68	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00
78	00	00	00	00	00	00	00	00
80	1A	2B	65	00	00	00	00	00
88	00	00	00	00	00	00	00	00
90	82	00	00	00	00	00	00	00

Figure 6: Memory contents file used for waveform simulations in Figures 4 & 5

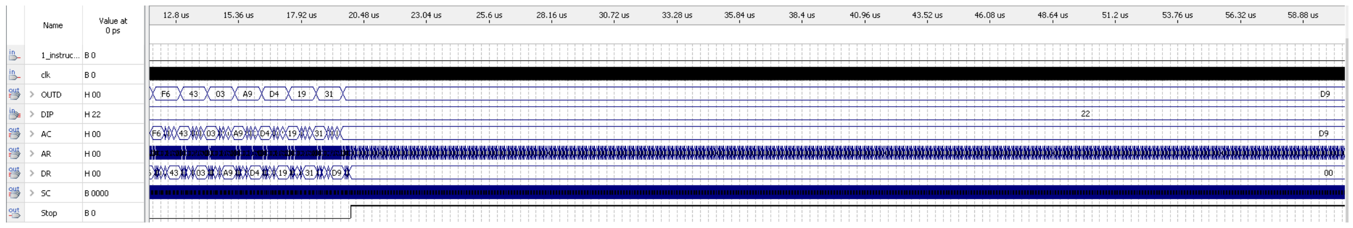
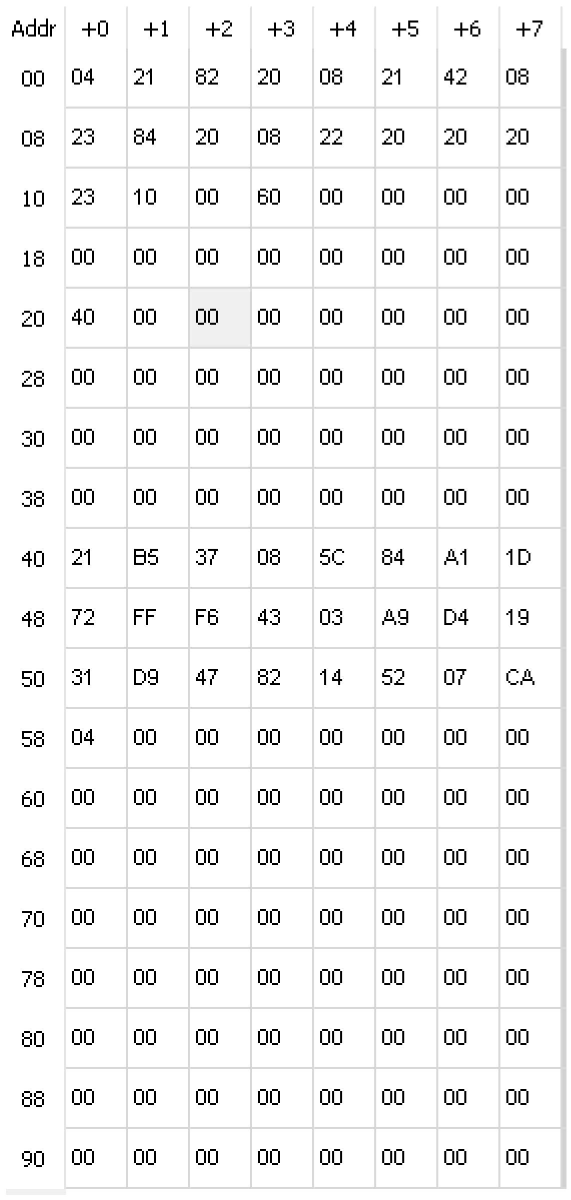
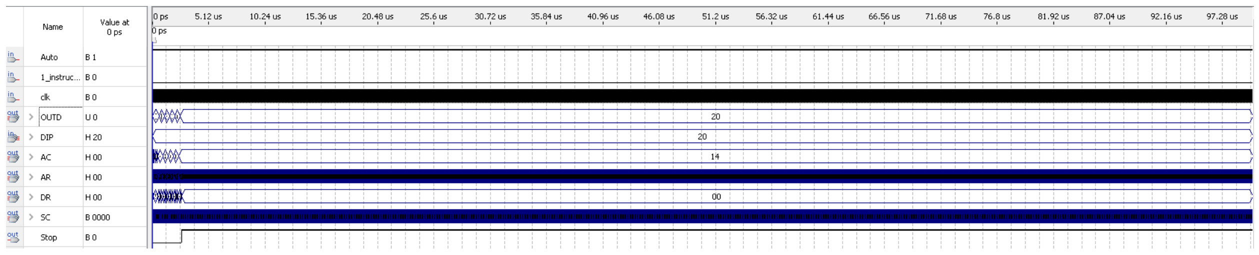


Figure 7: Addition software program waveform simulation



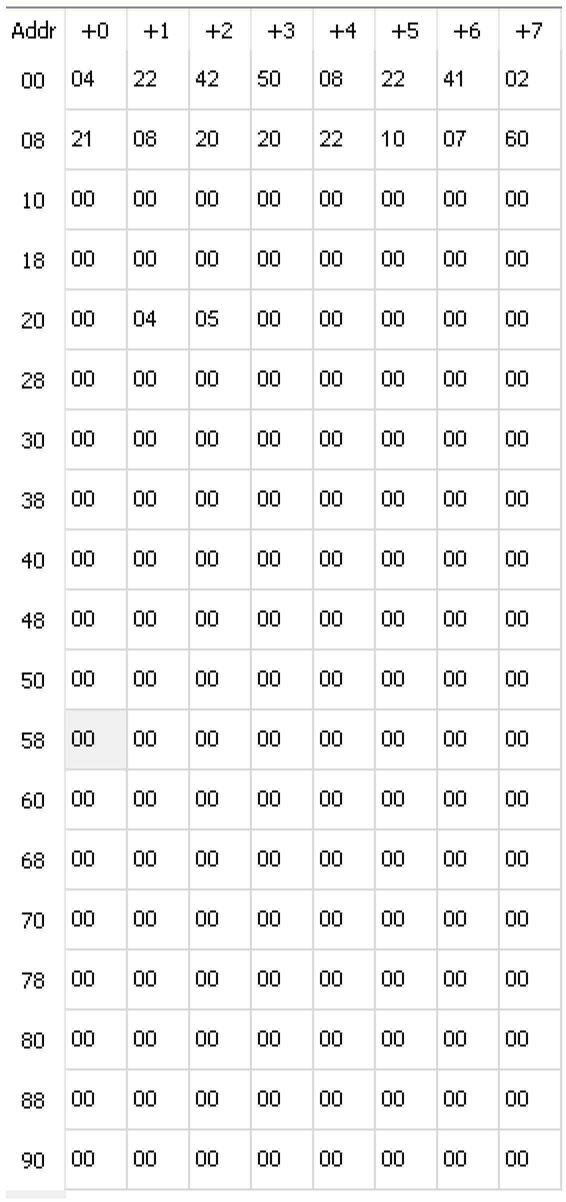
Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	04	21	82	20	08	21	42	08
08	23	84	20	08	22	20	20	20
10	23	10	00	60	00	00	00	00
18	00	00	00	00	00	00	00	00
20	40	00	00	00	00	00	00	00
28	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00
38	00	00	00	00	00	00	00	00
40	21	B5	37	08	5C	84	A1	1D
48	72	FF	F6	43	03	A9	D4	19
50	31	D9	47	82	14	52	07	CA
58	04	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00
68	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00
78	00	00	00	00	00	00	00	00
80	00	00	00	00	00	00	00	00
88	00	00	00	00	00	00	00	00
90	00	00	00	00	00	00	00	00

Figure 8: Memory contents file containing instructions for addition



Name	Value at 0 ps
Auto	B 1
1_instruc...	B 0
clk	B 0
OUTD	U 0
DIP	H 20
AC	H 00
AR	H 00
DR	H 00
SC	B 0000
Stop	B 0

Figure 9: Multiplication software program waveform simulation



Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	04	22	42	50	08	22	41	02
08	21	08	20	20	22	10	07	60
10	00	00	00	00	00	00	00	00
18	00	00	00	00	00	00	00	00
20	00	04	05	00	00	00	00	00
28	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00
38	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00
48	00	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00	00
58	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00
68	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00
78	00	00	00	00	00	00	00	00
80	00	00	00	00	00	00	00	00
88	00	00	00	00	00	00	00	00
90	00	00	00	00	00	00	00	00

Figure 10: Memory contents file containing instructions for the multiplication of 4 and 5