**Table of contents**

[**Prelab**](#_b73zuj2v08ww) **3**

[**Objectives**](#_einj9y9011ab) **10**

[**Equipment and components**](#_s64rqbsihtic) **11**

[**Algorithmic Solution**](#_36ncwrw5qk6o) **11**

[**Design Part**](#_1zb2jct6q92o) **12**

[**Discussion**](#_u2y82xxw8kx7) **20**

[**Conclusion**](#_rbzozmez0l1m) **21**

[**Group Member Contributions**](#_5ksblhtp0p0p) **21**

[**References**](#_6hp4dmhb0zjc) **21**

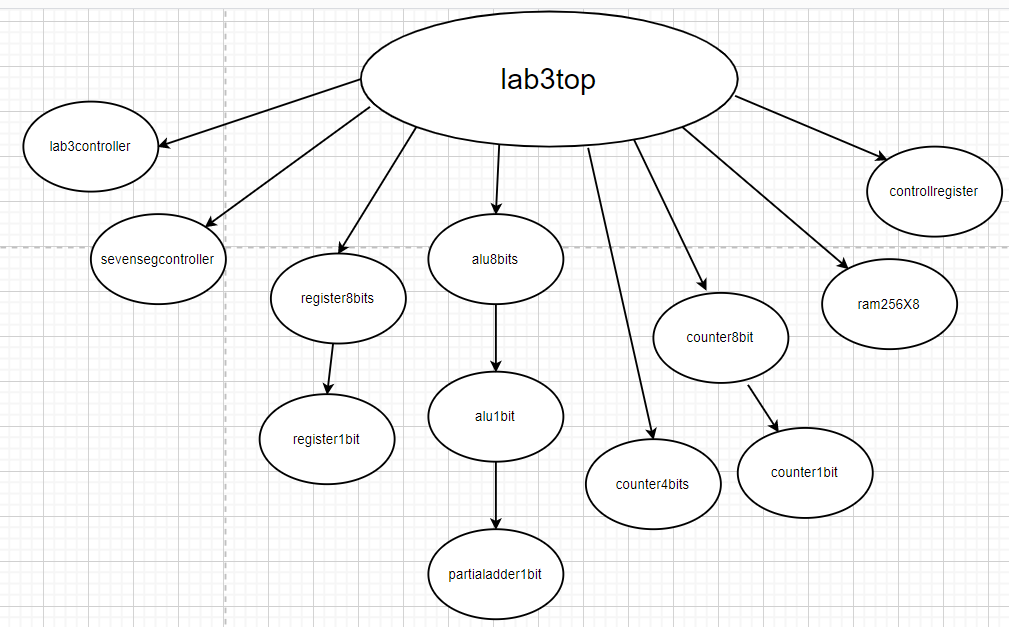
#### 

#### 

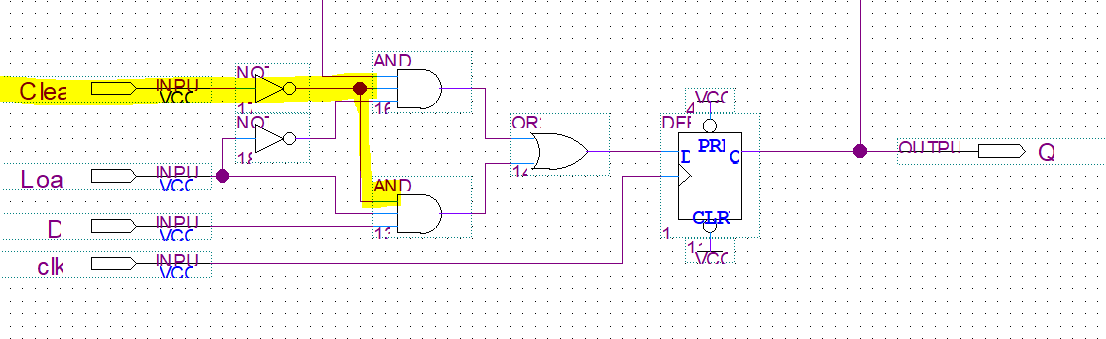
#### 

#### 

#### **Prelab**

****

1. The BUS multiplexer is an 8x1 MUX. Therefore it can allow only one register to place its output on the data bus at a time.
2. Synchronous. This is because register8bit.bdf is composed of register1bit.bdf which is composed of d flip flops in which they all share the same clock. When the clock is 1, both AND gates will be set to 0 meaning that all the register1bit.bdf’s will be cleared hence synchronous.
3. It will reset as the clear input seems to take precedence.



1. The address register is used for memory reference instructions. This requires storing data or fetching data from the memory. To know where to fetch or store, the address register is required. This is why the address register is connected directly to the memory.
2. The PC needs to be incremented to obtain the next instruction. The AC can also be incremented as seen in the register reference instruction INC. The DR can be incremented as seen in the memory reference instruction ISZ.
3. Increment has the lowest priority because it doesn’t require the other 2 counters to function and it’s input doesn’t influence the others. It only requires its own input and the current state of the JK flip flop. Clear has the highest priority because it is attached to the final or gates making it the final deciding factor on what both the out and increment outputs will be. The Load counter would have a lower priority because the In input determines what value will be sent to the or gates with the Clear input. Therefore from lowest to highest should be increment, load, clear.
4. No because AC does not have an input that is connected to the bus. This means that another clock cycle is needed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S2** | **S1** | **S0** | **Operation** | **Description** |
| 0 | 0 | 0 | AC + DR | Addition (Arithmetic) |
| 0 | 0 | 1 | AC + DR’ + 1 | Subtraction (Arithmetic) |
| 0 | 1 | 0 | Ashl AC | Arithmetic shift left AC |
| 0 | 1 | 1 | Ashlr AC | Arithmetic shift right AC |
| 1 | 0 | 0 | AC **∧** DR | Logic AND |
| 1 | 0 | 1 | AC **∨** DR | Logic OR |
| 1 | 1 | 0 | DR | Transfer DR (Logical) |
| 1 | 1 | 1 | AC’ | Complement AC (Logical) |



**5.2**

**Control signals**

1. **Memwrite:** T9Y4 + T10Y6
2. **AR\_Load:** T0 + T2 + T5(IR6)’ + T6(IR6)’ + T7X2
3. **PC\_Load** = T8Y5
4. **PC\_Inc** = T2S’ + T5IR6’S’ + Y6S’(T11 + T12)(DR[7-0])’
5. **DR\_Load** = T8Y0 + T8Y1 + T8Y2 + T8Y3 + T8Y6
6. **DR\_INC** = T9Y6
7. **IR\_Load** = T3
8. **AC\_Clear** = T5X1IR0
9. **AC\_Load** = T5X1 (IR1 + IR2 + IR3) + T9(Y0 + Y1 + Y2 + Y3)
10. **AC\_Inc** = T5X1IR4
11. **OUTD\_Load** = T1
12. **ALU\_Sel2** = T9Y0 + T9Y3 + T5X1IR1
13. **ALU\_Sel1** = T5X1IR2 + T5X1IR3 + T5X1IR1 + T9Y3 + T10Y6
14. **ALU\_Sel0** = T9Y2 + T5X1IR3 + T5X1IR1
15. **BusSel2** = T0 + T9Y4
16. **BusSel1** = T2 + T5 + T10T6 + T0
17. **BusSel0** = T8T5 + T10T6 + T9T4
18. **SC\_Clear** = T5X1 + T9Y0 + T9Y1 + T9Y2 + T9Y3 + T9Y4 + T8Y5 + T12Y6
19. **Halt** =T5X1IR5

**7.1**

**1.**

00: load (LDA)

01: AC ← 0a: DR← M[a0], AC ← DR SC← 0 therefore AC contain 0a

02: complement AC (therefore 0a is 0000 1010 → (complemented) 1111 0101 F5) (AC ← AC’)

03: STA (store AC ): M[a0] ← AC : therefore M[a0] contains F5

04: variable a0 now contains F5 (therefore M[a0] ← F5)

05 and 06: increment memory a0 → a1 (ISZ → increment content of memory location

and and skip the following instruction if the increment number is 0)

DR← M[a0] (therefore DR← F5), DR← DR+1 (therefore DR← F6), M[a0] ← DR (therefore M[a0] ← F6) DR!= 0

07: PC ← AR (therefore PC ← a0 )

08: to address 20

20: LDA: DR← M[M[a1]] (therefore DR← M[80] (therefore DR← 01 ), AC← DR (therefore AC← 80)

21: The address of what to LDA

22: ADD: DR← M[M[a2]] (therefore DR← M[81] (therefore DR← 01)), AC← AC+DR

(therefore AC← 01 + 01 (therefore AC← 02))

24: STA( store AC indirectly )

25: M[M[a3]] ← AC (therefore M[M[a3]] ← 02 (therefore M[82] ← 02))

26 and 27: LDA: AC← M[a1] (therefore AC← 80)

28: INC: AC← 81

29 and 2a : M[a1] ← 81

2b: INC: AC ← 82

2c and 2d : STA: M[a2] ← 82

2e: AC← 83

2f and 30: M[a3] ← 83

31: BUN  
32: start loop again

**2. Write a simple pseudo code to describe the program. Give the following names to the variables stored at the memory addresses A0 - A3 (hex):**

Calling program:

LDA a0 % loads M[counter] = 0a

CMA % complement accumulator (0a → F5)

STA a0 % M[counter] ← F5

LOP:

05, ISZ a0 % increment M[counter] and skip if 0. 0a is the counter

BUN 20 % branch unconditionally to address 20

HLT % halt the program

LDA X I % load indirectly AC← M[M[X]]. AC now has the value 80

ADD Y I % add indirectly M[M[X]] and M[M[Y]]

STA Z I % M[M[Z]] ← AC

INC % increment AC by 1

STA Y % store M[Y] ← AC. a2 now has a new address

BUN 05 % branch unconditionally to address 05 to restart loop

Address table:

80, 01 % DATA ARE FOUND AT ADDRESSES 80 TO FF %

81, 01

Counter, 0a % loop counter, which will be done 10 times %

X, 80 % pointer to the first number to be added %

Y, 81 % pointer to the second number to be added %

Z, 82 % pointer to memory location where result will be stored %

|  |  |
| --- | --- |
| Address | Name of Variable |
| A0 | counter |
| A1 | X |
| A2 | Y |
| A3 | Z |

**3. What does this program calculate?**

This program adds the sum and the previous added number to create a new summation. The new summation follows the same process of adding itself and the previous added number.

General: M[M[a1]] + M[M[a2]] = M[M[a3]]

Loop0: M[80] + M[81] = M[82]

1 + 1 = 2

Loop1: M[81] + M[82] = M[83]

1+ 2 = 3

Loop 2: M[82] + M[83] = M[84]

2+ 3 = 5

Loop 3: M[83] + M[84] = M[85]

3 + 5 = 8

Loop 4: M[84] + M[85] = M[86]

5 + 8 = 13

Loop 5: M[85] + M[86] = M[87]

8 + 13 = 21

Loop 6: M[86] + M[87] = M[88]

13 + 21 = 34

Loop 7: M[87] + M[88] = M[89]

21+ 34 = 55

Loop 8: M[88] + M[89] = M[90]

34 + 55 = 89

Loop 9: M[89] + M[90] = M[91]

55 + 89 = 144

**4. Why is practical to use in this program memory - reference instructions with indirect addressing (in other words, instructions which use pointers)?**

By utilizing memory reference instruction (MRI) with indirect addressing, the content of memory addresses can be changed to point to another address. This allows us to refer to new or old addresses without having to directly point to it, since its location is stored in memory. This technique of using a memory of memory allows to perform looping by indirectly storing a set of data which are at new memory locations.

Loop2:

05 and 06: ISZ: DR← M[a0] (therefore DR← F6, DR← F7, M[a0] ← F7)

07 and 08: BUN to address 20

20 and 21: LDA: AC← M[a1] (therefore AC← 81)

22: ADD: AC← AC+ M[a2]

**7.2 Program Design**

00: 04 LDA % LOP

01: fc (Dec 252) % Load the sum

02: 82 ADD I % Add the value to the sum

03: fd (Dec 253)

04: 08 STA

05: fc (Dec 252) % store the total sum

06: 84 LDA I % | (start block) load the last added value and store the last value in fd

07: fd (Dec 253) % |

08: 08 STA % |and store the last value in fb

09: fb (Dec 251) % | (end block)

10: 20 ISZ % increments the content of FF (thus M[FF] = A1 increments to A2)

11: fd (Dec 253)

12: 04 LDA % loading the total sum

13: fc (Dec 252)

14: 42 CMA %complement the sum

15: 04 LDA

16: fa (Dec 250) %loading to temporary

17: 20 ISZ %ISZ to see if the value is one

18: fa (Dec 250)

19: 10 LOP 00

20: 60 HLT

Variables:

fa: 0 % temp variable

fb: 0 % last number added

fc: 0 % sum

fd: A1 % address PTR to data

Note: a1 = Dec 161

a1: 21

a2: b5 (Dec 181)

a3: 37

a4: 08

a5: 5C (Dec 92)

a6: 84

a7: A1 (Dec 161)

a8:1D (Dec 29)

a9:72

aa: FF (Dec 255)

ab: F6 (Dec 246)

ac:43

ad: 03

ae: A9 (Dec 169)

af: D4 (Dec 212)

b0: 19

b1: 31

b2: D9 (217)

b3: 47

b4: 82

b5: 14

b6: 52

b7: 07

b8: CA (Dec 202)

b9: 04

#### **Objectives**

The Objective of this lab is to analyze the structure of a basic computer. The lab consists of 2 parts. The hardware portion consists of testing the construction of the control unit and the software part consists of writing and testing lower level languages. For the hardware part, we will use knowledge learned in chapter 5 to determine the logic equations for the control unit. For the software part, we will use our assembly language writing skills developed in chapter 6 to understand the code given and write our own assembly code.

#### **Equipment and components**

* Quartus II (web edition)
* Altera DE2-115 board
* USB-Blaster cable
* Power supply 12V/2A
* Probe
* Coaxial cable
* Wires
* Ribbon cable

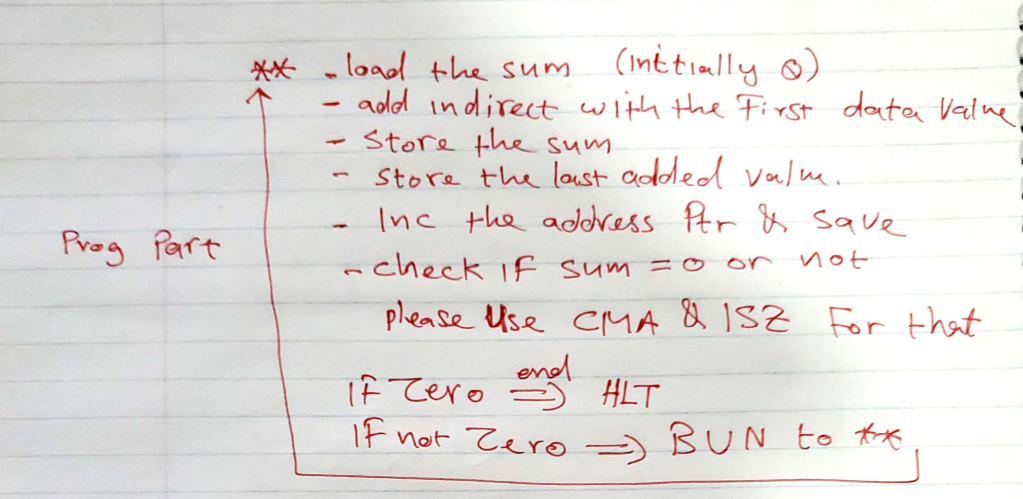
#### **Algorithmic Solution**

**Hardware:**

Analyzing tables 2, 3 and 4 in the lab manual, we created logic expressions for the control signal. These control signals are to be used to determine how the data flows in the CPU. For example registers can load data in, clear data, or transfer the data to the bus. Analyzing the tables, we figure out where each register is required to perform a specific task allowing us to make the boolean expressions. These expressions can then be transferred to quartus allowing us to create symbols and represent the CPU as a whole.

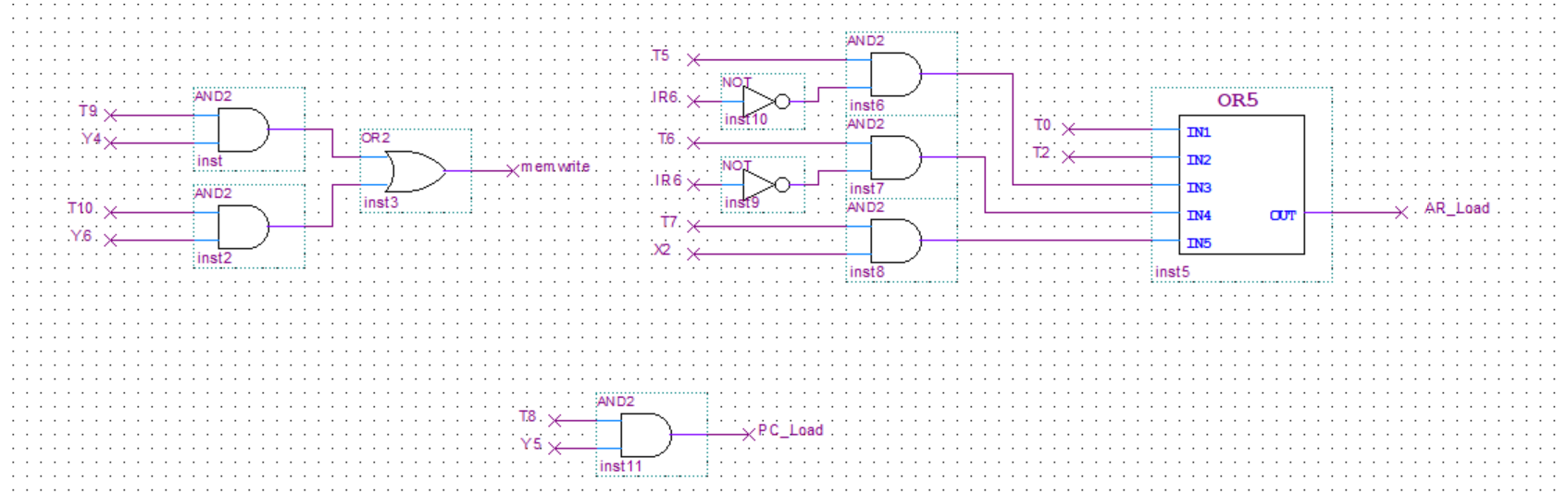
**Software:**

The logic/procedure for this part follows the one given in brightspace and shown below.

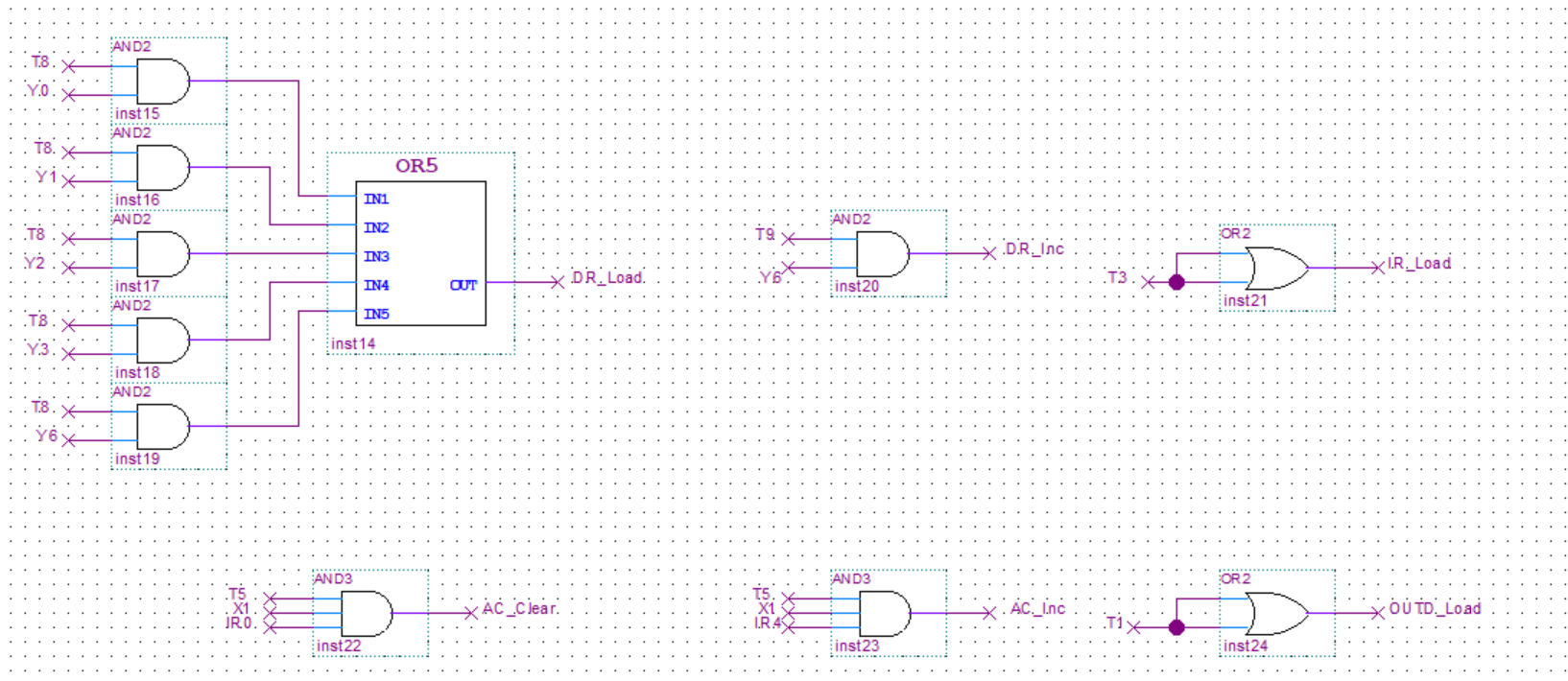


With this logic, we then used table 1 (Computer Instructions list) to write the machine code for the specific computer. Our machine code starts at memory address 00 and finishes at 20. We stored the 25 values for adding to the sum between addresses a1 to bb. Other variables required for the completion of this question such as the pointer, sum, memory address of the last value added etc. are stored in the memory addresses fa to fd.

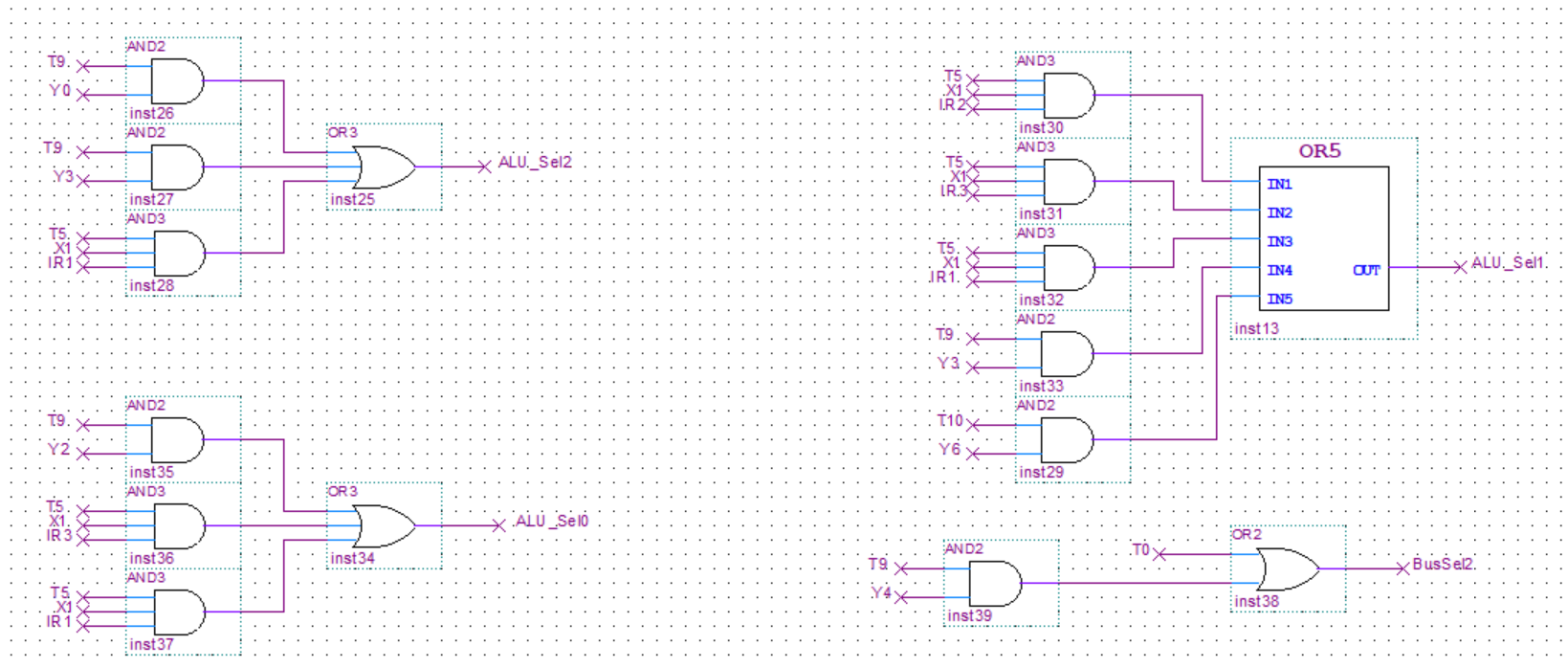
#### **Design Part**



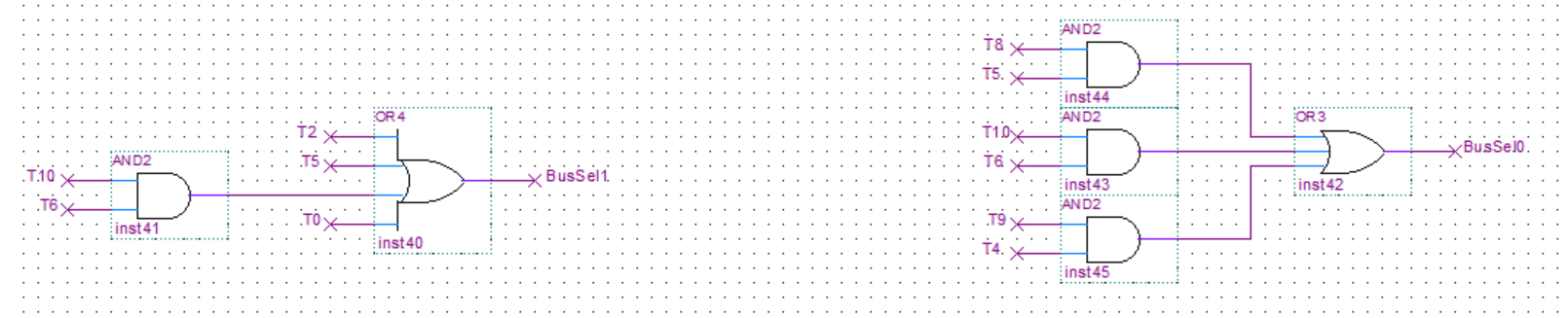
**Figure 1 Mem\_write, AR\_Load and PR\_Load**



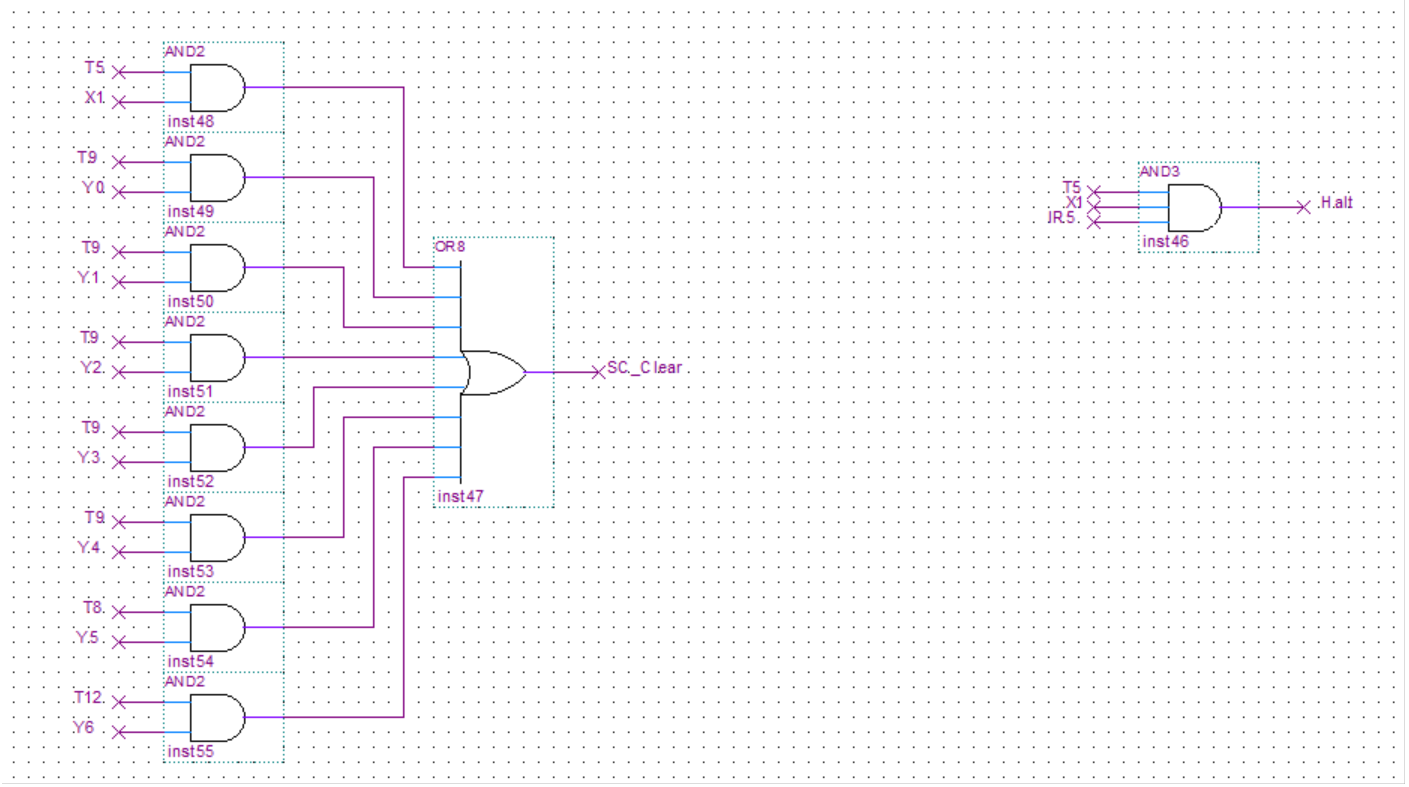
**Figure 2 DR\_Load, LR\_Load, and OUTD\_Load**



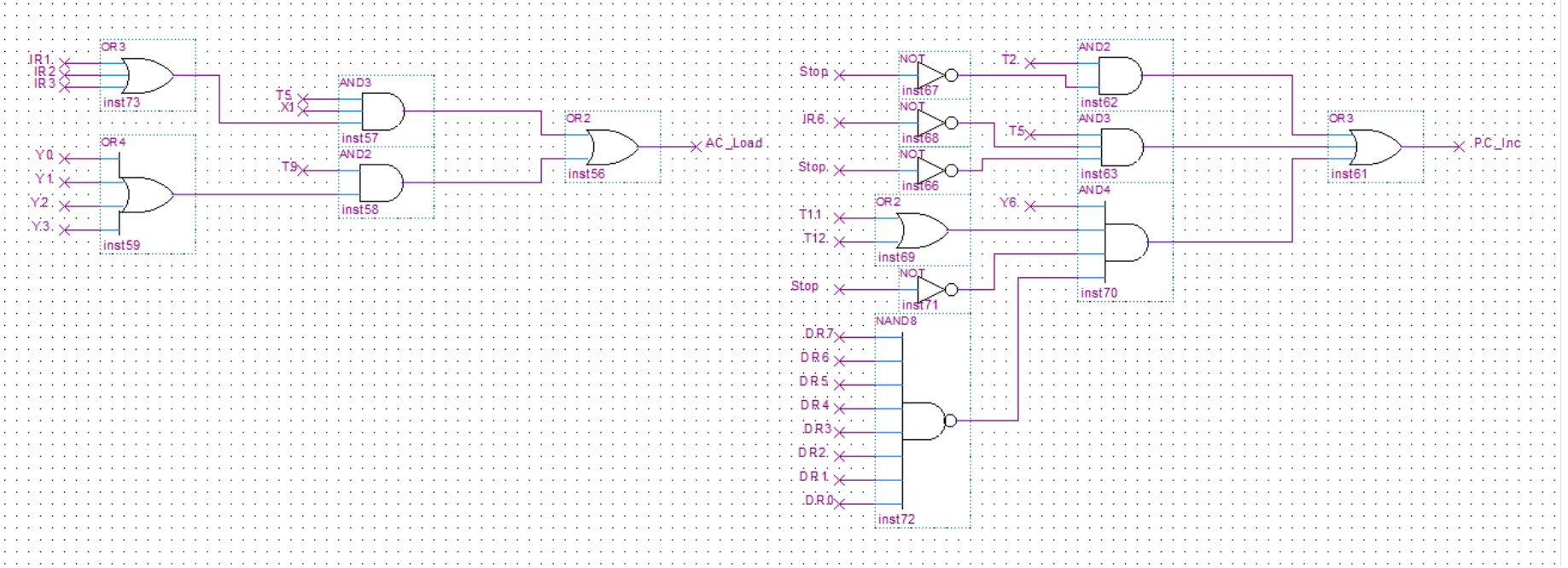
**Figure 3 ALU\_Sel2, ALU\_Sel1, ALU\_Sel0, BusSel2**



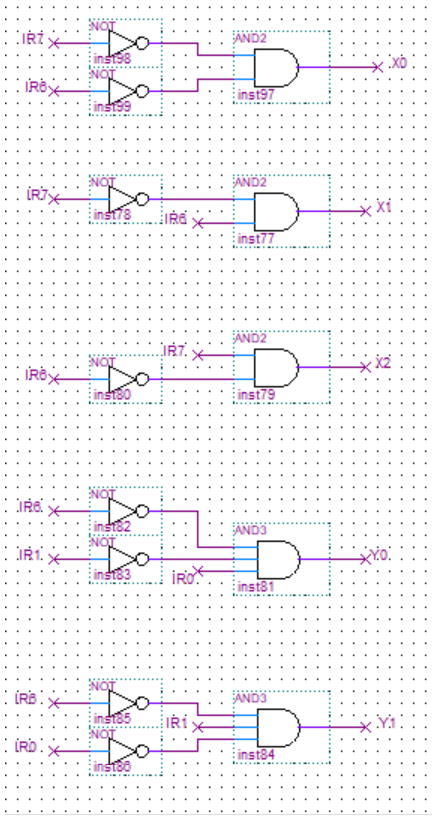
**Figure 4 BusSel1, BusSel0**



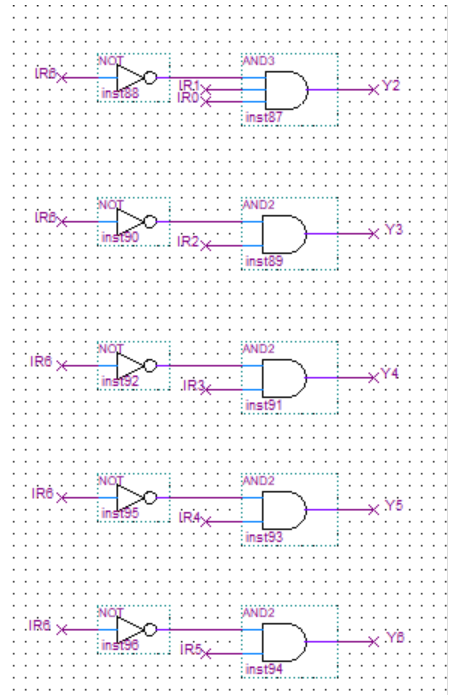
**Figure 5SC\_Clear, Halt**



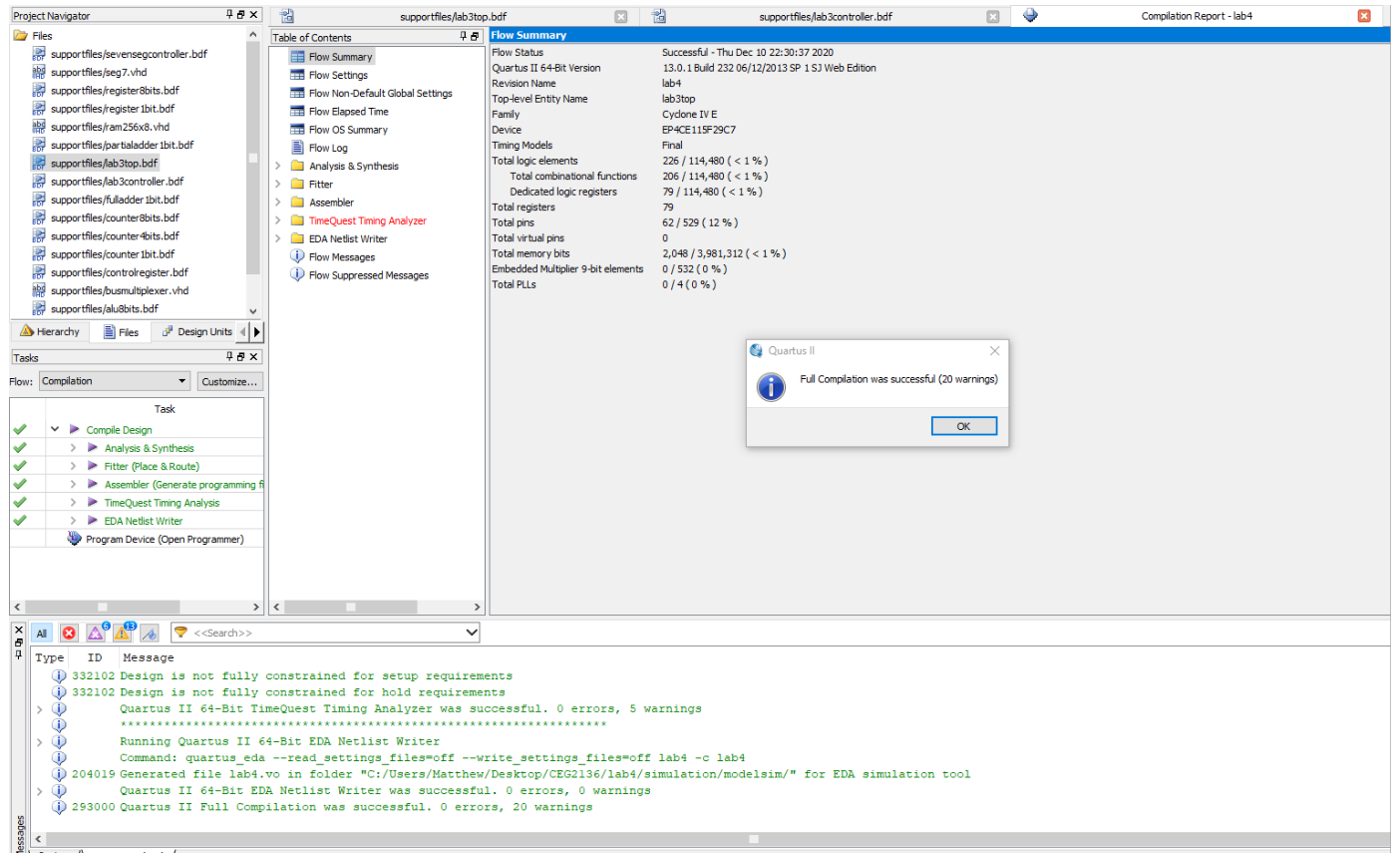
**Figure 6 AC\_Load, PC\_Inc**



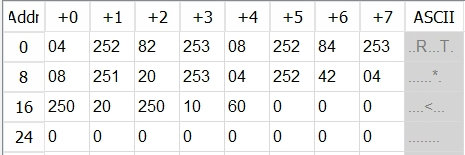
**Figure 7 X0 - X2, Y0, Y1**

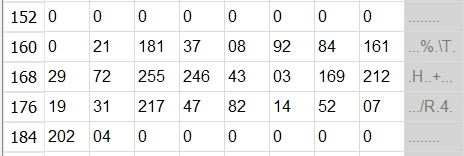


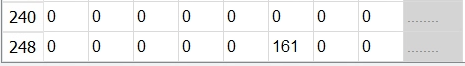
**Figure 8 Y2 to Y6**



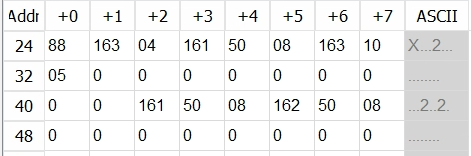
**Figure 9 Successful compilation**

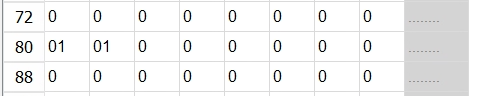


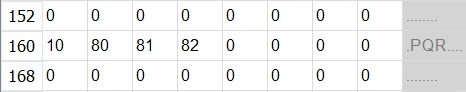




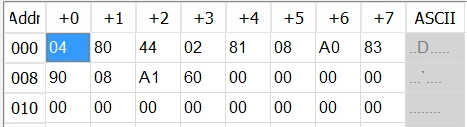
**Figure 10 memory contents of 7.2 in decimal form**

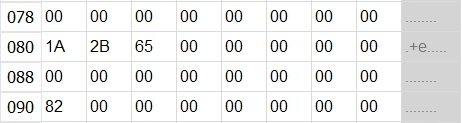
****

****

****

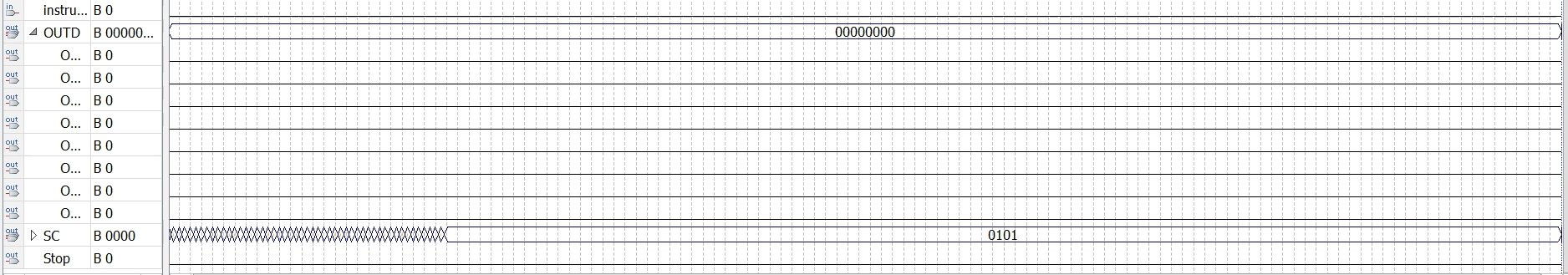
**Figure 11 memory contents of 7.1 in decimal form**

****

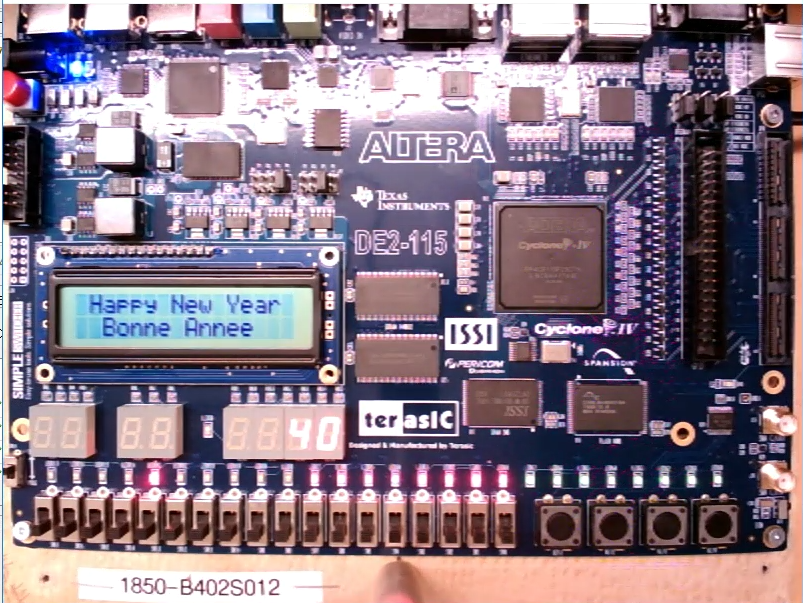
****

**Figure 12 memory contents of example 4.3 in hexadecimal form**

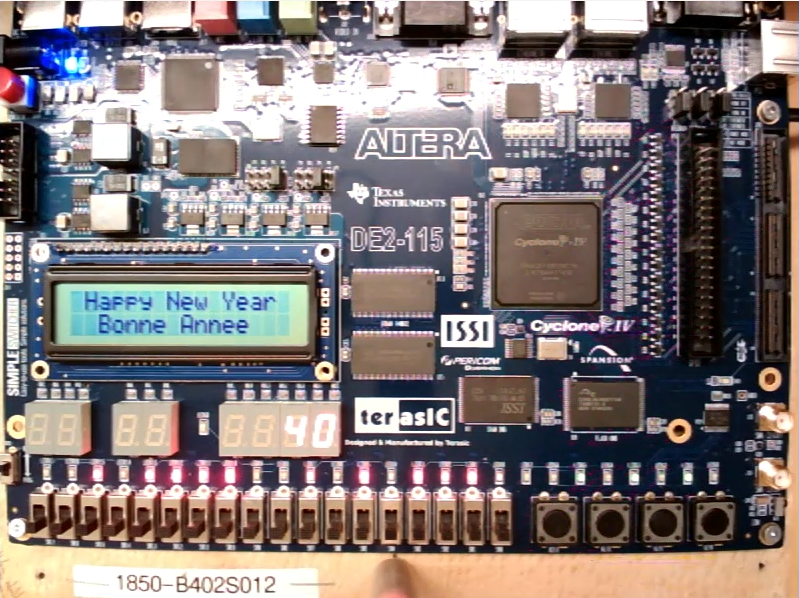




**Figure 13 7.2 waveform**



**Figure 14 Altera board for 7.2**

****

**Figure 15 Altera board for 7.1**

#### **Discussion**

Despite creating the correct mif files and circuits, we were unable to successfully generate the waveforms. However, we were able to understand the functionings of a basic computer and how each component was constructed. Based upon the tables provided in the lab manual, we were able to to derive and implement the boolean equations. This allowed the instruction cycles to operate which also includes the memory reference instructions and register reference instructions. However, the tested results did not provide the expected answers; thus, we concluded that our problem must be reflected in the schematics we designed based on the derived equations. Although we briefly checked over the schematics for the source of error, we were unsuccessful in finding the problem. We later found out that our MIF files may be incorrect due to the change in hex and decimal settings; however, this was still not the issue. This eventually led to an incorrect waveform that changed its AR value but not the OUTD value which is displayed on the altera board. The waveforms did not make sense because it is supposed to display when the sum would be 0 but this did not reflect the result. Due to these circumstances our lab was unsuccessful.

#### **Conclusion**

In this lab we created a control unit. Despite our lab being unsuccessful, we learned how to write machine code and create circuits to control the loading, incrementing and clearing of registers.

#### **References**

“DE2-115 User Manual,” Terasic Technologies Inc.,http://www.terasic.com.tw/cgibin/page/archive.pl?Language=English&CategoryNo=165&No=502&PartNo=4 , 2013

U Ottawa Brightspace TA Machine code logic 2020