RE : Ability to Read from the bus
WE : Ability to Write on the bus

Building the ALU

ADDER

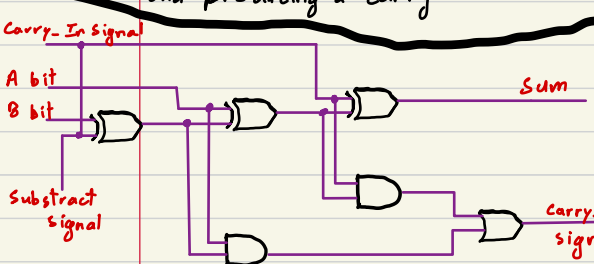
①

Addition Rule:

A + B	Sum	Carry
0 + 0	0	0
1 + 0	1	0
0 + 1	1	0
1 + 1	0	1

$$\begin{array}{r} {\scriptstyle 0\ 0\ 1} \\ 1001 \\ +\ 0101 \\ \hline 1110 \end{array} \Rightarrow \begin{array}{r} 9 \\ +\ 5 \\ \hline 14 \end{array}$$

For our ALU, we need to put together 8 adders, each other adding one bit from each register and producing a carry



③

A	B	Substract	Sum	C_out
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
1	0	1	1	0
1	1	1	0	1

Substract

②

Substracting is not as easy as adding.

$$\begin{array}{r} 1001 \\ -\ 0101 \\ \hline ?? \end{array} \neq \begin{array}{r} 9 \\ -\ 5 \\ \hline 4 \end{array}$$

Two's complement is the best method known to substract two binary numbers.

It consist of inversing of the bits of the binary to substract, and ADD the two data together. Then, we add 1 to the result:

$$\begin{array}{r} 1001 \\ -\ 0101 \\ \hline = \end{array} \Rightarrow \begin{array}{r} 1001 \\ +\ 1010 \\ \hline = 0011 \\ +\ \quad 1 \\ \hline 0100 \end{array} \Rightarrow \begin{array}{r} 9 \\ -\ 5 \\ \hline 4 \end{array}$$

Two's complement

But how to inverse the value of a bit ? By using XOR gates

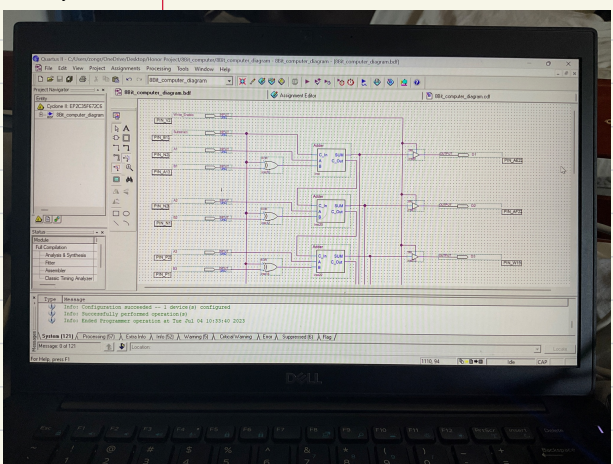
Substract_Signal	B	output
0	0	1
0	1	0
1	0	1
1	1	0

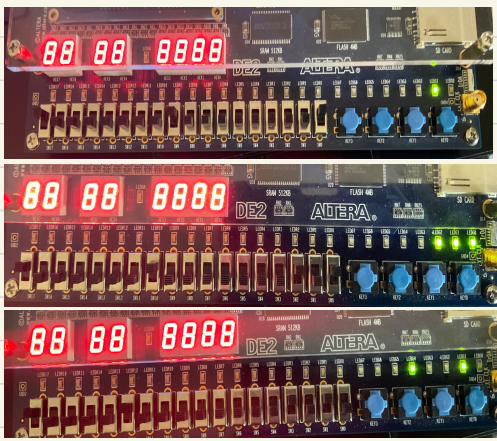
Inverted

then we feed the output to an adder

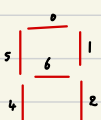
ALU Test:

$$\begin{array}{r} 1 \\ +\,1 \\ \hline 2 \end{array} \quad \begin{array}{r} 00000001 \\ 00000001 \\ \hline 00000010 \end{array} \qquad \begin{array}{r} 2 \\ +\,5 \\ \hline 7 \end{array} \quad \begin{array}{r} 00000010 \\ 00000101 \\ \hline 00000111 \end{array} \qquad \begin{array}{r} 8 \\ +\,10 \\ \hline 18 \end{array} \quad \begin{array}{r} 00001000 \\ 00001010 \\ \hline 00010010 \end{array}$$



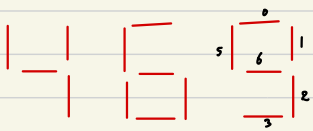


building a seven segment display decoder:



8 bits needs 3 displays, and each display should only display the number between 0-9.

Exemple: 469 ⇒



Truth table of a 7-segment display → segments

Digit	Binary	0	1	2	3	4	5	6
0	0000	1	1	1	1	1	1	0
1	0001	0	1	1	0	0	0	0
2	0010	1	1	0	1	1	0	1
3	0011	1	1	1	1	0	0	1
4	0100	0	1	1	0	0	1	1
5	0101	1	0	1	1	0	1	1
6	0110	1	0	1	1	1	1	1
7	0111	1	1	1	0	0	0	0
8	1000	1	1	1	1	1	1	1
9	1001	1	1	1	0	1	1	1
A	1010	1	1	0	1	1	1	1
B	1011	0	0	0	1	1	1	1
C	1100	1	0	0	1	1	1	0
D	1101	0	1	1	1	1	0	1
E	1110	1	0	0	1	1	1	1
F	1111	1	0	0	0	1	1	1

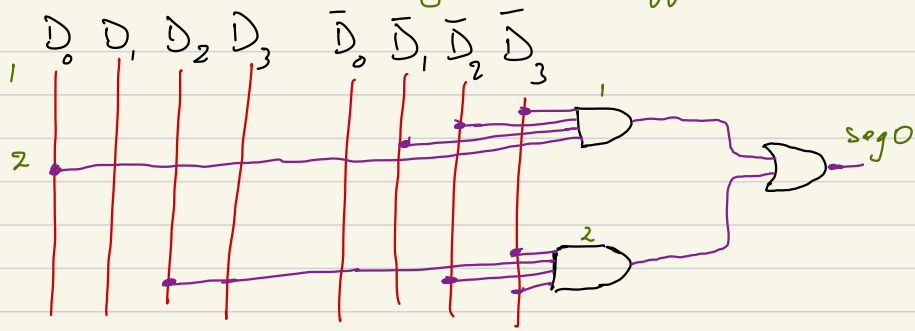
with the DE2,
apply "0" to a segment lights it up
apply "1" to turn it off (Common anode)

Digit	Binary	Seg0	Seg1	Seg2	Seg3	Seg4	Seg5	Seg6
0	0000	0	0	0	0	0	0	1
1	0001	1	0	0	1	1	1	1
2	0010	0	0	1	0	0	1	0
3	0011	0	0	0	0	1	1	0
4	0100	1	0	0	1	1	0	0
5	0101	0	1	0	0	1	0	0
6	0110	0	1	0	0	0	0	0
7	0111	0	0	0	1	1	1	1
8	1000	0	0	0	0	0	0	0
9	1001	0	0	0	0	1	0	0

Let's build a logic circuit to light up each segment based on the truth table:

in cases 1 and 2, seg0 is turned off

Digit	Binary	Seg0
0	0000	0
1	0001	1
2	0010	0
3	0011	0
4	0100	1
5	0101	0
6	0110	0
7	0111	0
8	1000	0
9	1001	0



So we make the circuit longer by using the same rule for the segments 1, 2, 3, 4, 5, and 6.

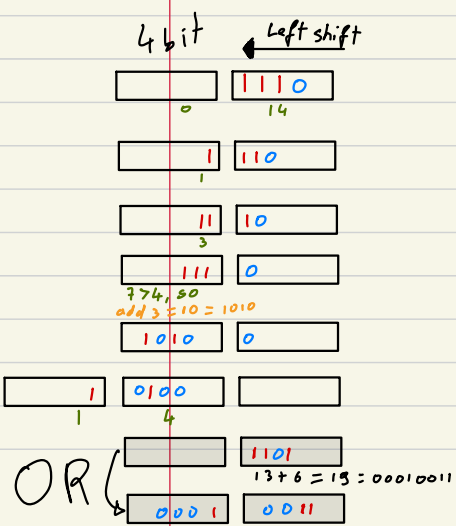
Now, we don't want our segments to display the letter ABCDEF. So we need to find a way to write the full 3 digit numbers on the 3 segments.

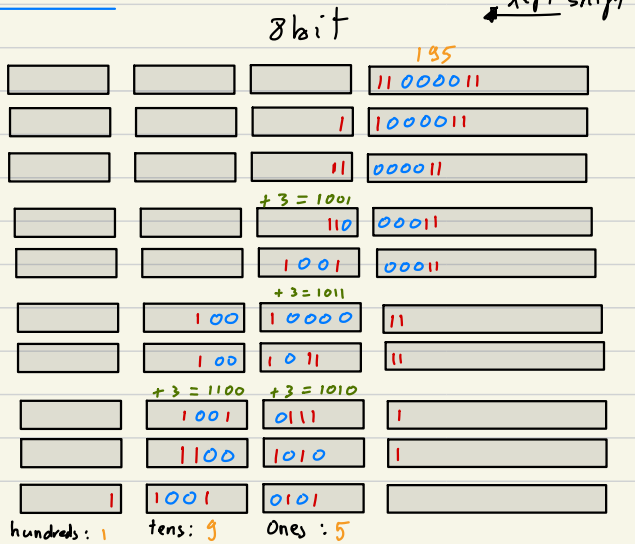
For that, I learned about the Double Dabble algorithm that takes an 8 bit value and break it down to many sets of 4-bit values.

https://youtu.be/hEDQpqhY2MA

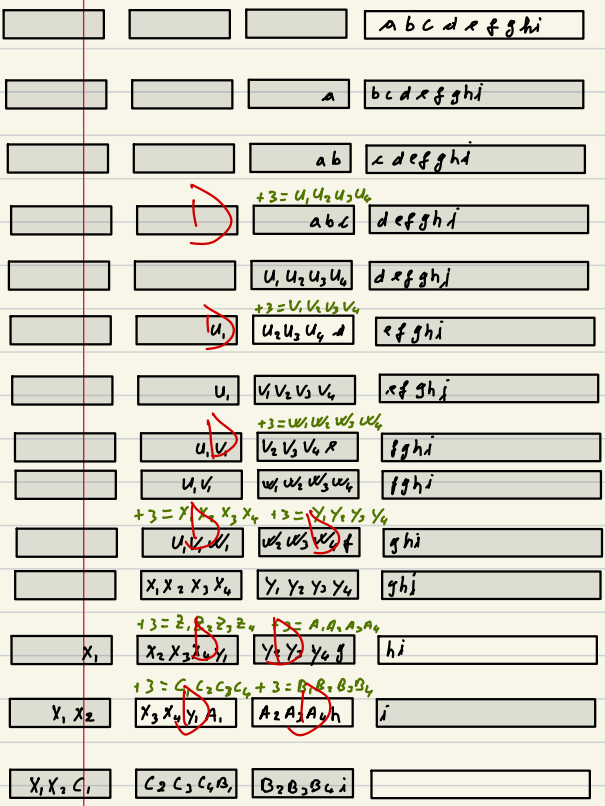
LINK TO VIDEO

Double Dabble



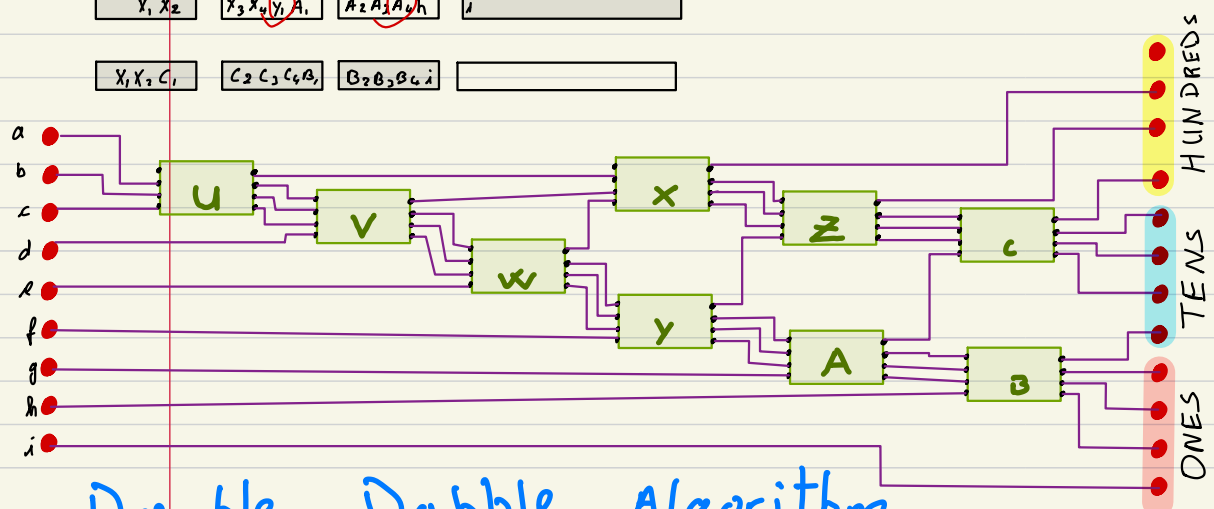


In General



Truth table

	Input	Output	
0	0000	0000	0
1	0001	0001	1
2	0010	0010	2
3	0011	0011	3
4	0100	0100	4
5	0101	1000	8
6	0110	1001	9
7	0111	1010	10
8	1000	1011	11
9	1001	1100	12

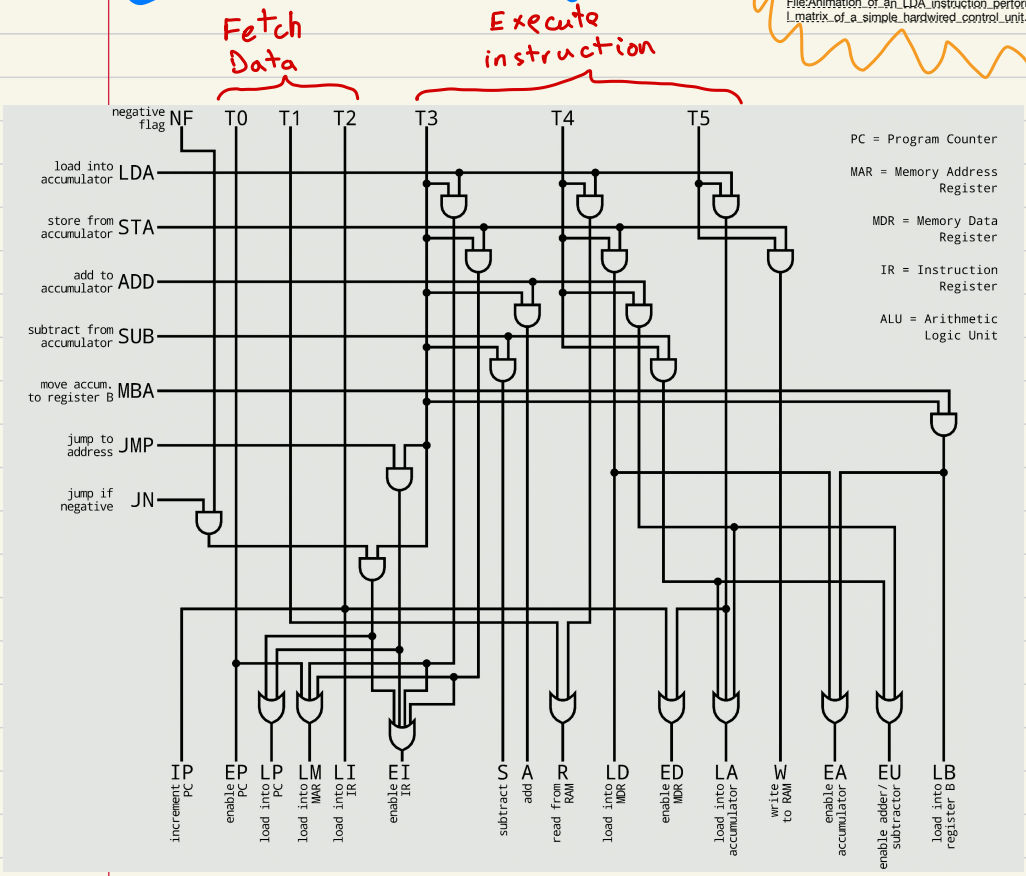


Double Dabble Algorithm for a 9 bit BCD Decoder

Control Unit

Refer to this animation for the load instruction:

https://commons.wikimedia.org/wiki/File:Animation_of_an_LDA_instruction_performed_by_the_control_matrix_of_a_simple_hardwired_control_unit.gif



LDA	0001	Load from RAM into Reg_A
ADD	0010	Add to value in A and save in Reg_A
SUB	0011	Substract to value in Reg_A and save in Reg A
OUT	0100	Output Result
HLT	0101	STOPS the clock

STEPS FOR LDA INSTRUCTION

T0	1st step :	PC_WE	MAR_RE	
T1	2nd step :	RAM_WE	IR_RE	PC_Enable
T2	3rd step :	IR_WE	MAR_RE	
T3	4th step :	RAM_WE	A_REG_RE	
T4	5th step :	Reset micro Counter		
T5				

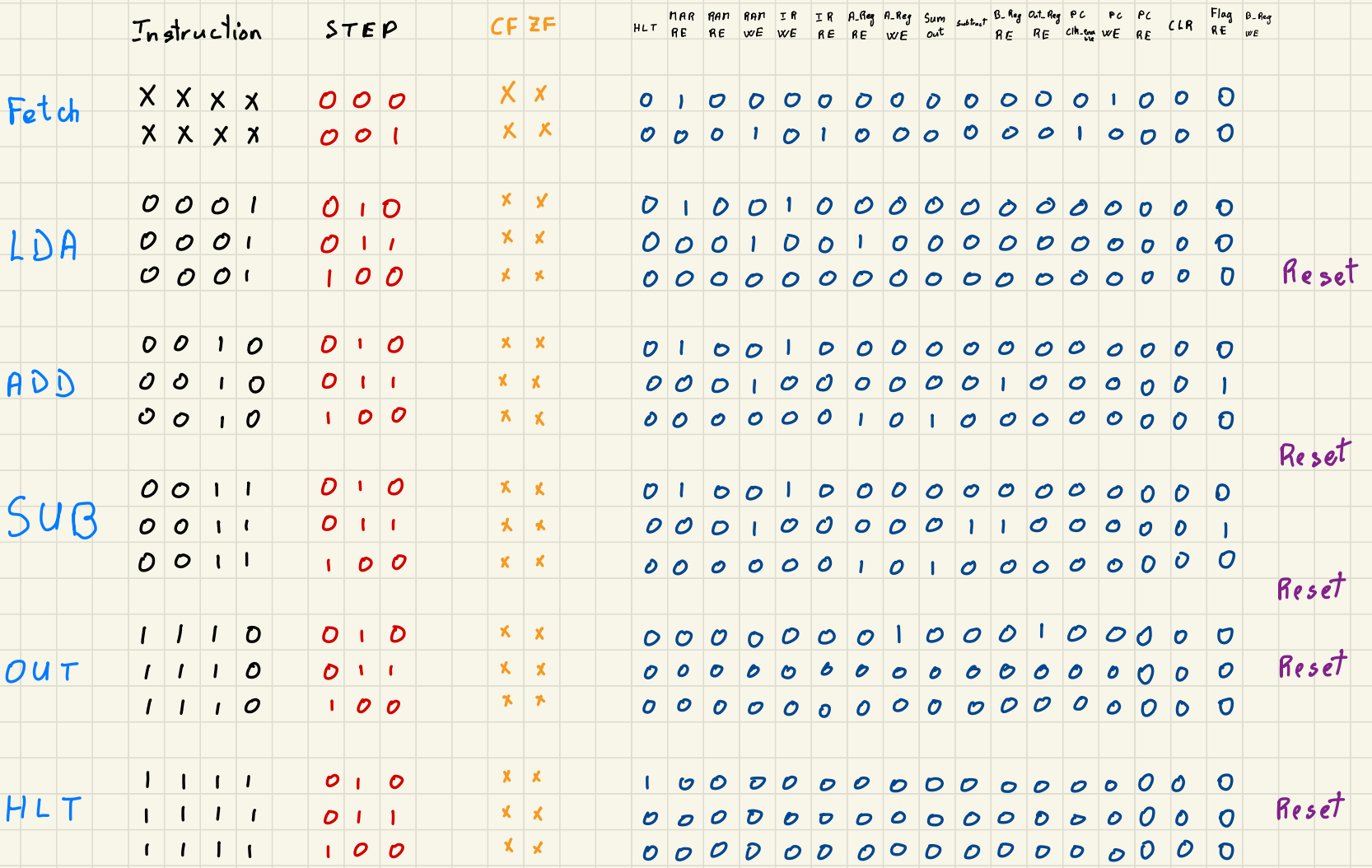
STEPS FOR ADD INSTRUCTION

T0	1st step :	PC_WE	MAR_RE	
T1	2nd step :	RAM_WE	IR_RE	PC_Enable
T2	3rd Step :	IR_WE	MAR_RE	
T3	4th step	RAM_WE	B_REG_RE	FLAG_RE
T4	5th step	Sum_Out	A_REG_RE	
T5	6th step	Reset micro Counter		

STEPS FOR OUT INSTRUCTION

T0	1st step :	PC_WE	MAR_RE	
T1	2nd step :	RAM_WE	IR_RE	PC_Enable
T2	3rd Step :	A_REG_WE	Out_REG_RE	
T3	4th step :	Reset micro Counter		
T4				
T5				

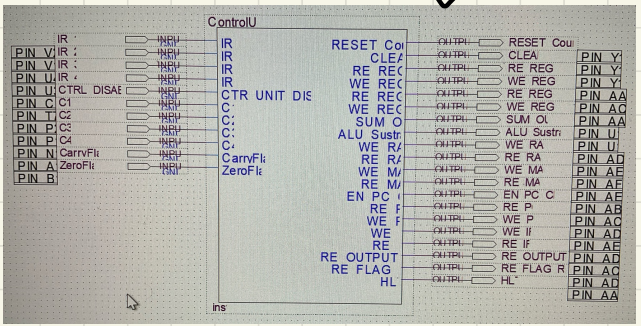
Fetch
" Same for all instruction"


Instruction
STEP
CF ZF
HLT
MAR RE
RAM RE
RAM WE
I R WE
I R RE
A_Reg RE
A_Reg WE
Sum out
Subtract
B_Reg RE
Out_Reg RE
PC clk_en
PC WE
PC RE
CLR
Flag RE
B_Reg WE
Fetch
LDA
ADD
SUB
OUT
HLT
Reset
Reset
Reset
Reset
Reset

	Instruction	STEP	CF	ZF	HLT	MAR RE	RAM RE	RAM WE	IR WE	IR RE	A_Reg RE	A_Reg WE	Sum out	subtract	B_Reg RE	Out_Reg RE	PC Clk_Enable	PC WE	PC RE	CLR	Flag RE	
STA	0 1 0 0	0 1 0	x	x	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
	0 1 0 0	0 1 1	x	x	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
	0 1 0 0	1 0 0			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset
LDI	0 1 0 1	0 1 0	x	x	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	
	0 1 0 1	0 1 1	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	0 1 0 1	1 0 0	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset
JMC	0 1 1 0	0 1 0	0	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	0 1 1 0	0 1 0	1	x	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	
	0 1 1 0	0 1 1	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	0 1 1 0	1 0 0	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset
JMZ	0 1 1 1	0 1 0	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	0 1 1 1	0 1 0	x	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	
	0 1 1 1	0 1 1	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	0 1 1 1	1 0 0	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

Load immediate
LDI can only
load a value from
0 - 15 to the A_register

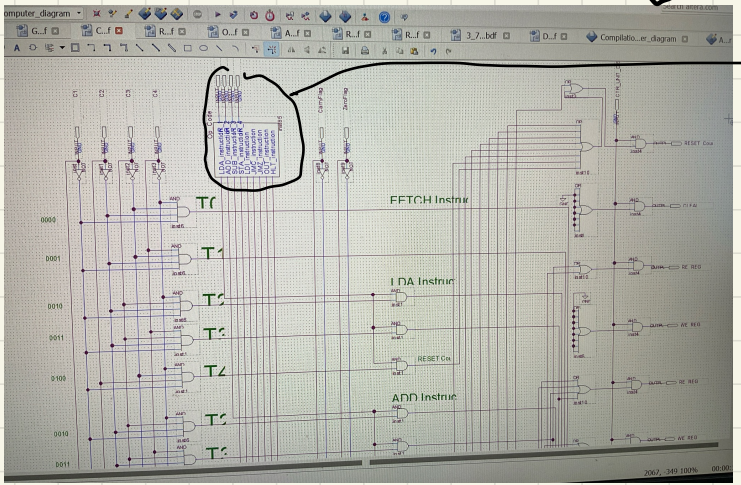
TESTING CONTROL UNIT

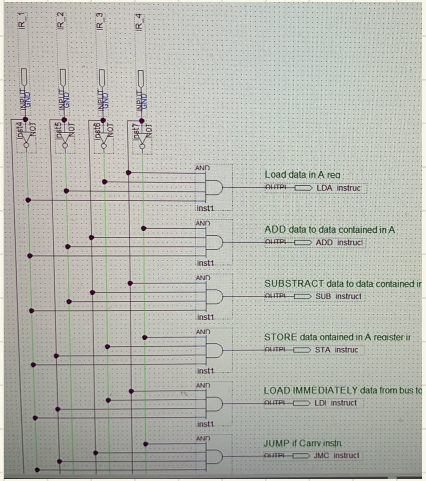


PIN ASSIGNMENT ON THE DE2 ALTERA Cyclone II

	Status	From	To	Value	Assignment Name
1	...		IR_1	PIN_V2	Location
2	...		IR_2	PIN_V1	Location
3	...		IR_3	PIN_U4	Location
4	...		IR_4	PIN_U3	Location
5	...		C1	PIN_T7	Location
6	...		C2	PIN_P2	Location
7	...		C3	PIN_P1	Location
8	...		C4	PIN_N1	Location
9	...		CarryFlag	PIN_A13	Location
10	...		ZeroFlag	PIN_B13	Location
11	...		CTRL_DISABLE	PIN_C13	Location
12	...		CLEAR	PIN_Y12	Location
13	...		EN_PC_CLK	PIN_AB21	Location
14	...		HLT	PIN_AA20	Location
15	...		RESET_Counter	PIN_Y18	Location
16	...		RE_FLAG_REG	PIN_AD22	Location
17	...		RE_IR	PIN_AD12	Location
18	...		WE_IR	PIN_AE12	Location
19	...		RE_MAR	PIN_AE13	Location
20	...		WE_MAR	PIN_AF13	Location
21	...		RE_RAM	PIN_AE15	Location
22	...		WE_RAM	PIN_AD15	Location
23	...		RE_REG_A	PIN_AC14	Location
24	...		WE_REG_A	PIN_AA13	Location
25	...		RE_REG_B	PIN_Y13	Location
26	...		WE_REG_B	PIN_AA14	Location
27	...		SUM_OUT	PIN_U17	Location
28	...		RE_OUTPUT_REG	PIN_AC22	Location
29	...		RE_PC	PIN_AC21	Location
30	...		WE_PC	PIN_AD21	Location
31	...		ALU_Sustract	PIN_U18	Location
32		<...>	<<new>>		<<new>>

INSIDE OF THE CONTROL



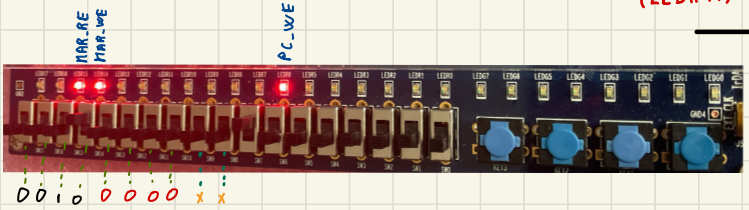


INSIDE OF THE INSTRUCTION DECODE

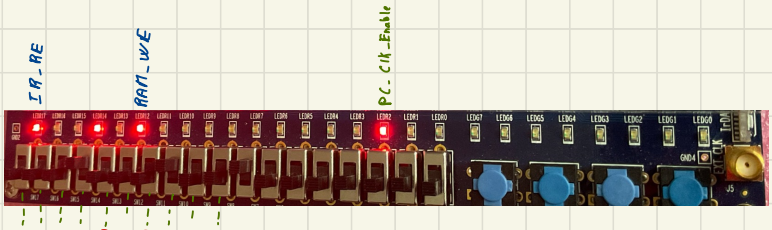
TESTING ADD Instruction

Note:
- O_Reg_WE (LEDR 8) — always 0 because feeds directly to the ALU and does not feed on the bus.
- MAR_WE (LEDR 14) — always 1 because it feeds directly to the MAR

T0

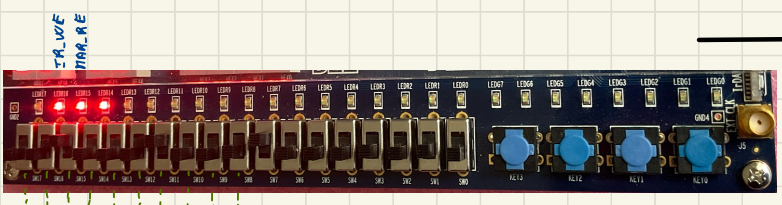


T1

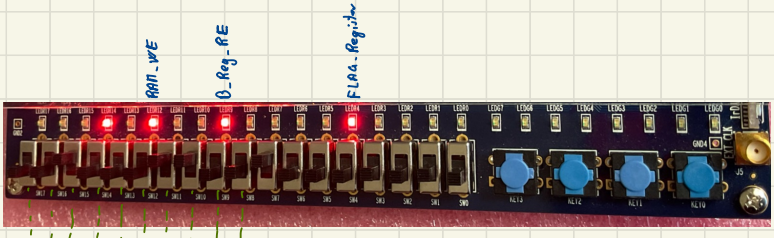


Fetch Instruction After Each Cycle

T2



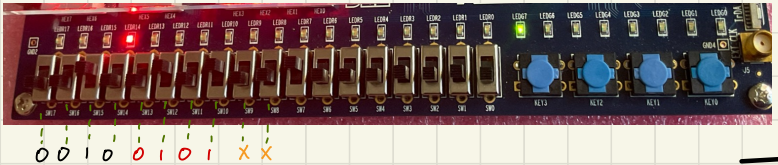
T3



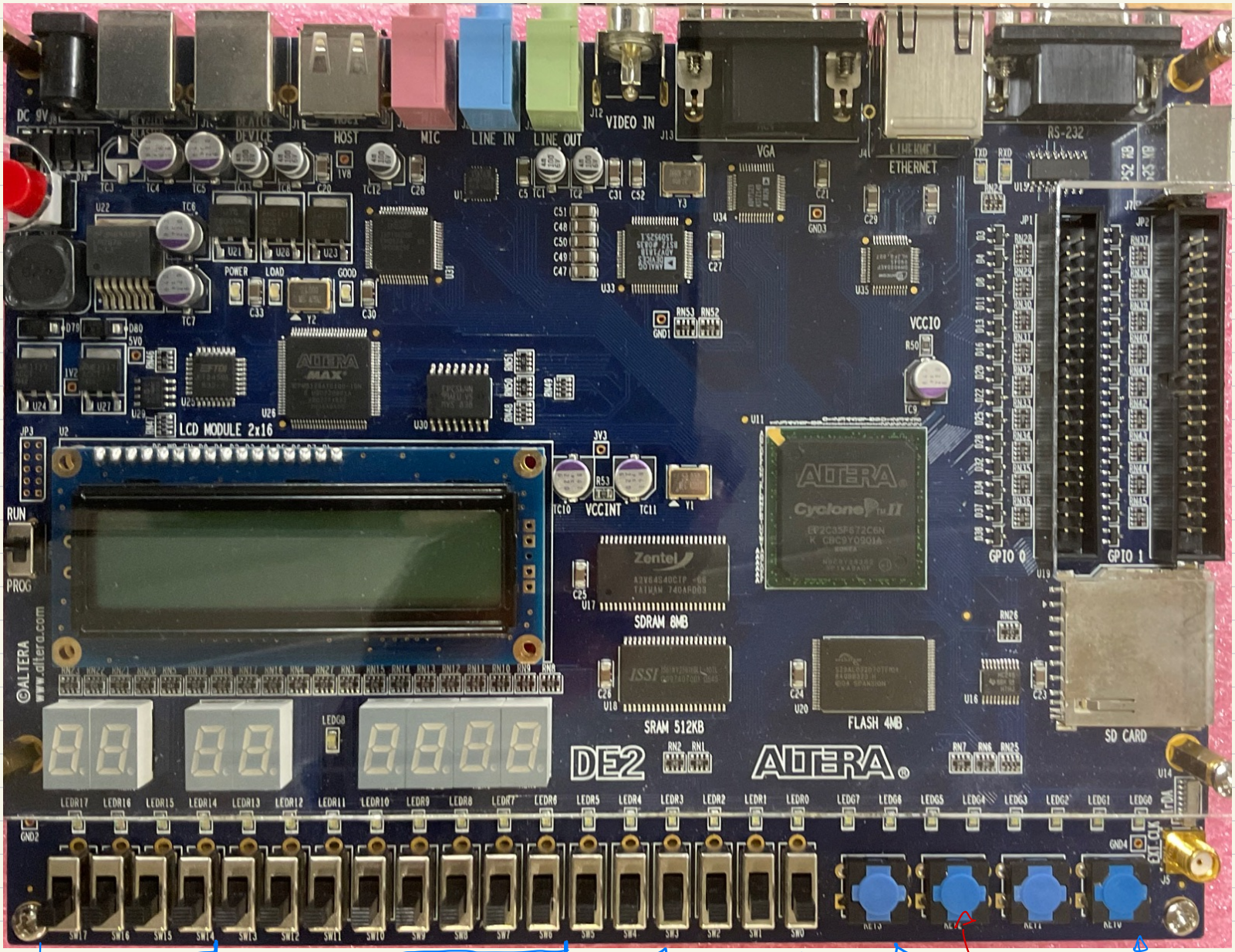
T4

STEPS FOR EXECUTING ADD INSTRUCTION

T5



Reseting MicroINC counter to T0



RAM_ADDR

RAM_DATA

clear RAM

code RAM

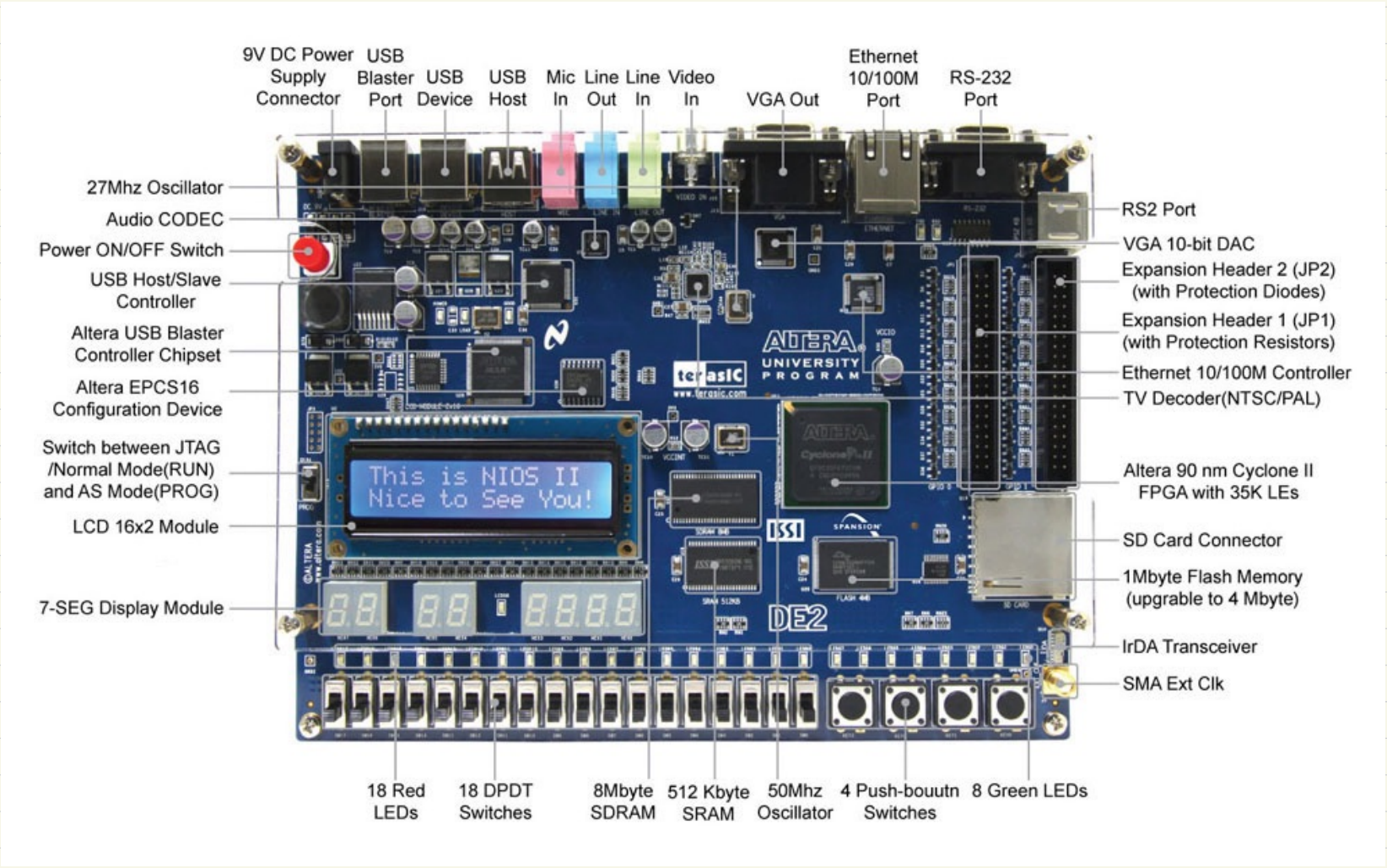
PROG (if up)

clk mode Switch (0 → ext_clk, 1 → manual clk)

RAM clk

CLR computer

manual clock



0 0 1 1 0 1 0 0

D_9 D_7 D_6 D_5 D_4 D_3 D_2 D_1