TASK 3 VHDL

Amit Nagar- Halevy and Tal Kapelnik
204210306 204117089

BGU
COMPUTER ENGEENIRING

Contents

Assignment description:

The aim of this laboratory is to design a simple MIPS compatible CPU. The CPU uses's a Single Cycle MIPS architecture and must be capable of performing instructions from MIPS instruction set. The design is executed on the Altera Board. The MIPS architecture is Harvard architecture in order to increase throughput and simplify the logic.
The architecture includes a MIPS ISA compatible CPU with data and program memory for hosting data and code. The block diagram of the architecture is given in Figure 1. The CPU have a standard MIPS register file. The top level and the MIPS core is structural. The design compiled and loaded to the Altera board for testing. A single clock (CLK) of 24MHZ is used in the design.

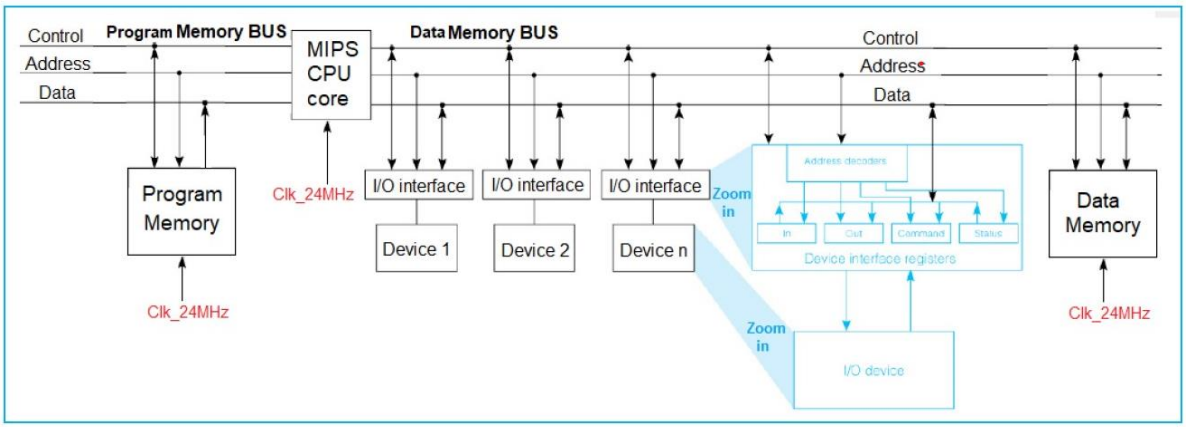
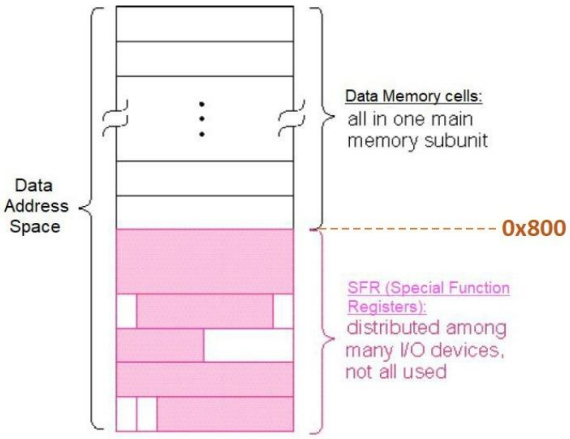


Figure 1 : System architecture



MEMORY Mapped I/O addresses:

PORT_LEDG[7-0] 0x800 - LSB byte (Output Mode)

PORT_LEDR[7-0] 0x804 - LSB byte (Output Mode)

PORT_HEX0[7-0] 0x808 - LSB byte (Output Mode)

PORT_HEX1[7-0] 0x80C - LSB byte (Output Mode)

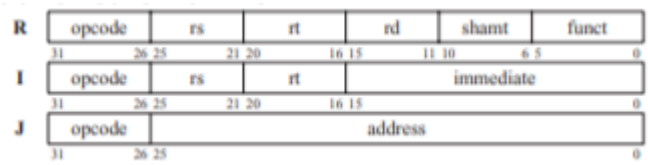
PORT_HEX2[7-0] 0x810 - LSB byte (Output Mode)

PORT_HEX3[7-0] 0x814 - LSB byte (Output Mode)

PORT_SW[7-0] 0x818 - LSB byte (Input Mode)

Supported Operation code formats:

Rtype Itype and Jtype as follow -



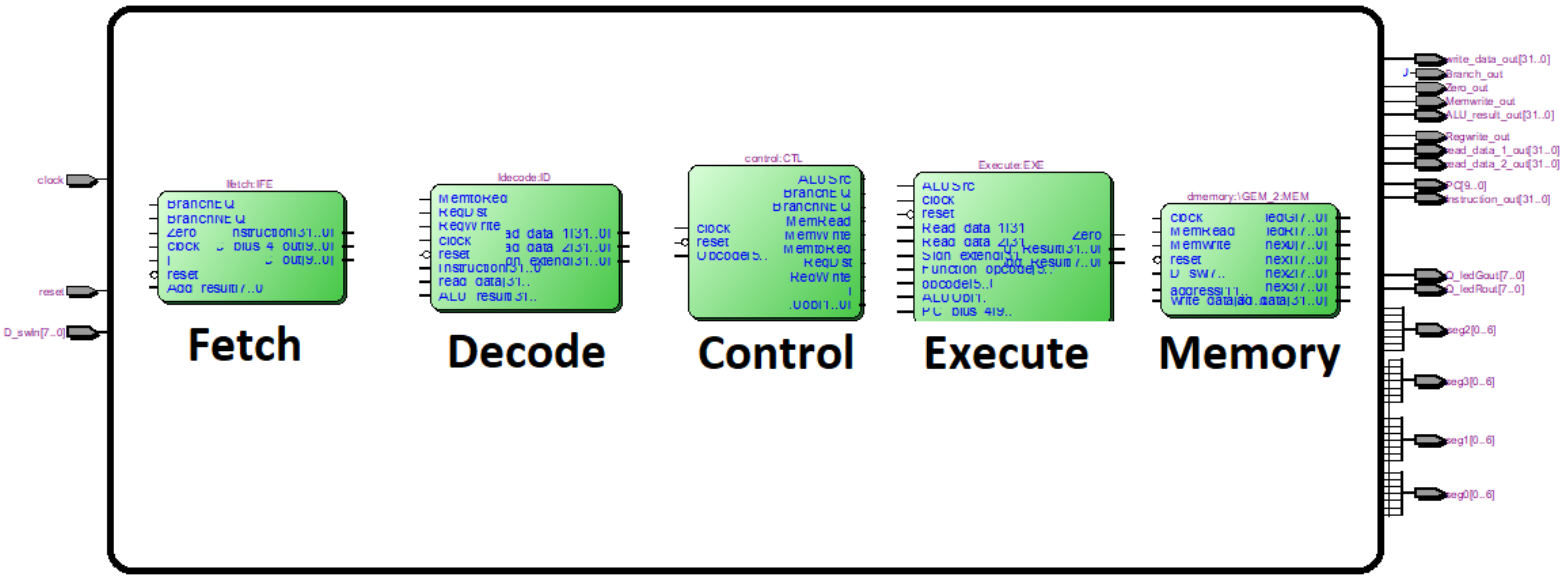
The supported set of opcodes in our version of single cycle Mips:

Or, ori, and, andi, xor, xori, add, addi, addu,

Lw, sw , slt, sll, srl , beq, bne, sub, j, jr, jal.

Blocks diagram:

Top entity - Mips:



Our single cycle Mips is divided to 5 stages:

Fetch – getting the next command

Decode- breaking the command to an instruction

Control – manage all the control lines

Execute- calculates data using the ALU and the control lines

Memory- reads and writes data to and from memory or I/O

Mips logic usage:

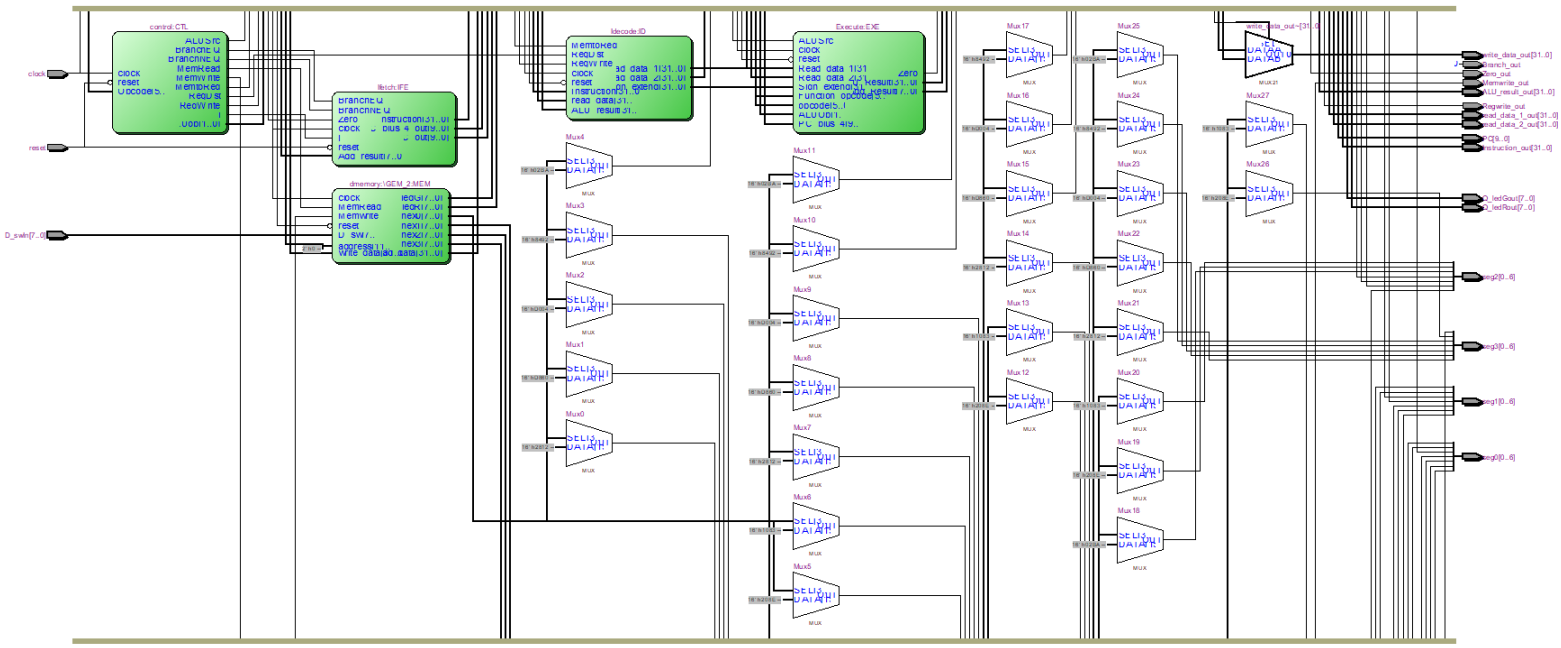
Flow Summary	
Flow Status	Successful - Wed Aug 19 13:49:25 2020
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	cputask3new
Top-level Entity Name	MIPS
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	3,396 / 18,752 (18 %)
Total combinational functions	2,762 / 18,752 (15 %)
Dedicated logic registers	1,779 / 18,752 (9 %)
Total registers	1779
Total pins	228 / 315 (72 %)
Total virtual pins	0
Total memory bits	82,432 / 239,616 (34 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

We can see that we have used 3,396 logic elements.

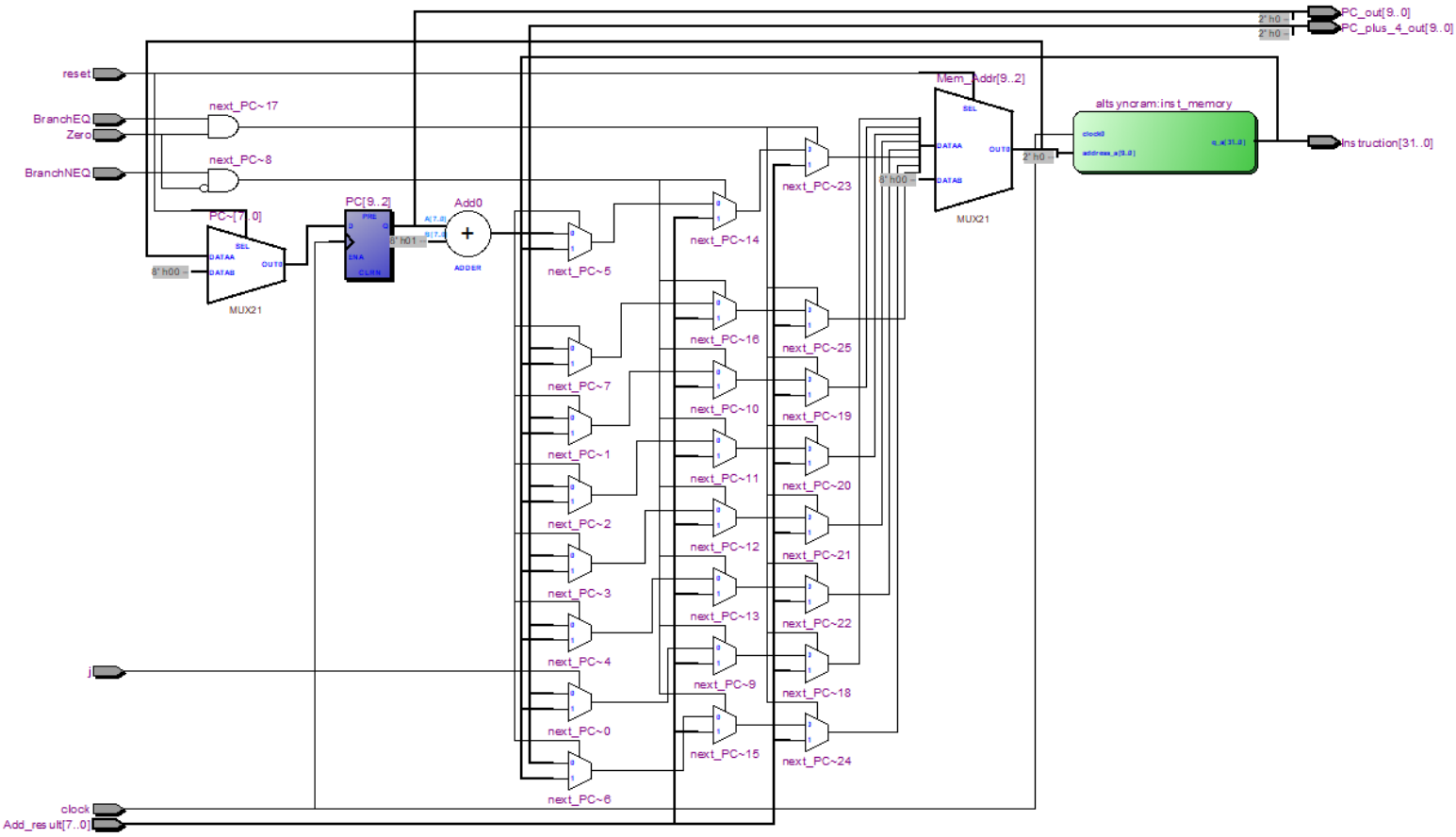
RTL View:

Now we **will** check each stage apart, starting from the top, Mips, and going down for each one of the stages from earlier.

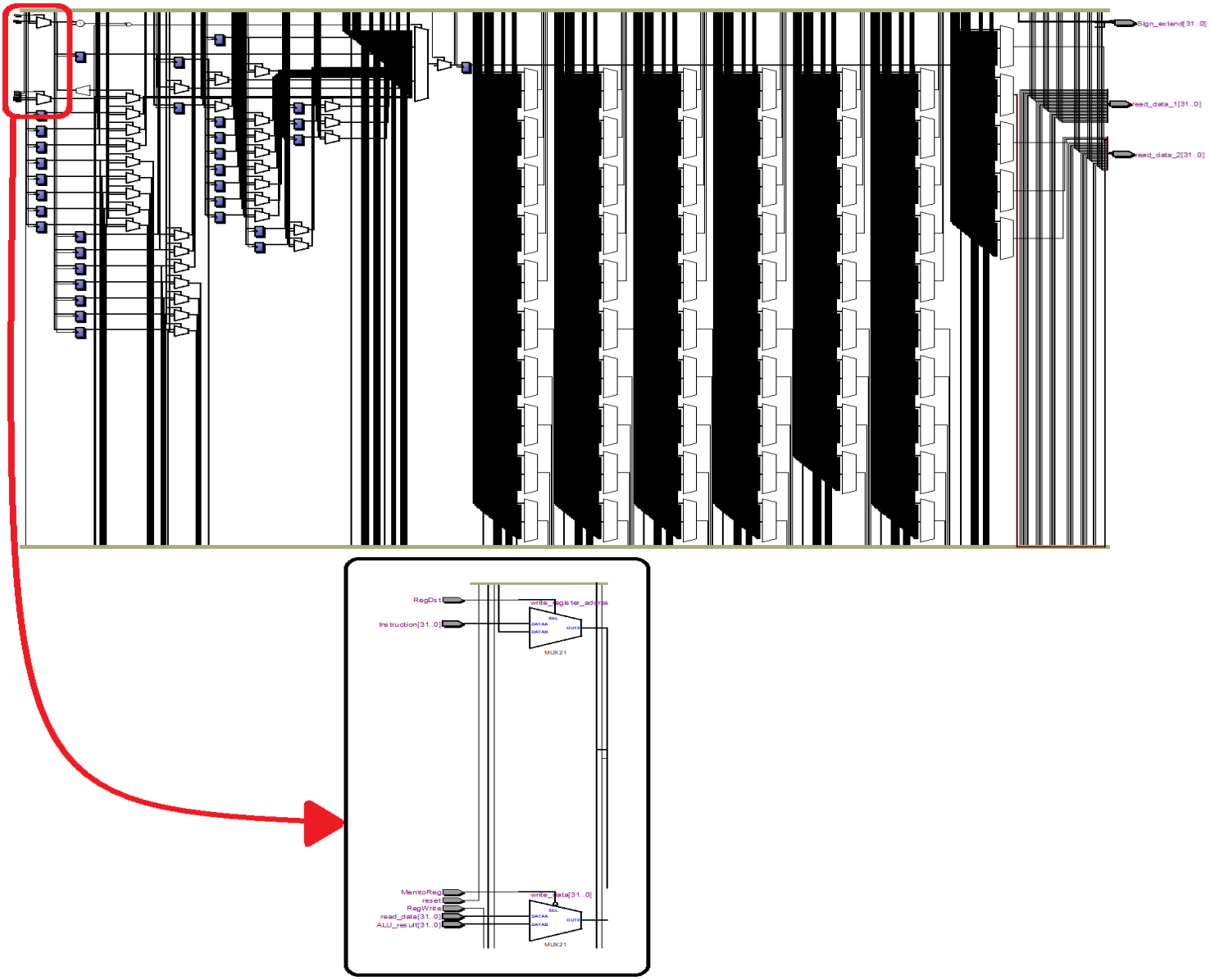
Mips:



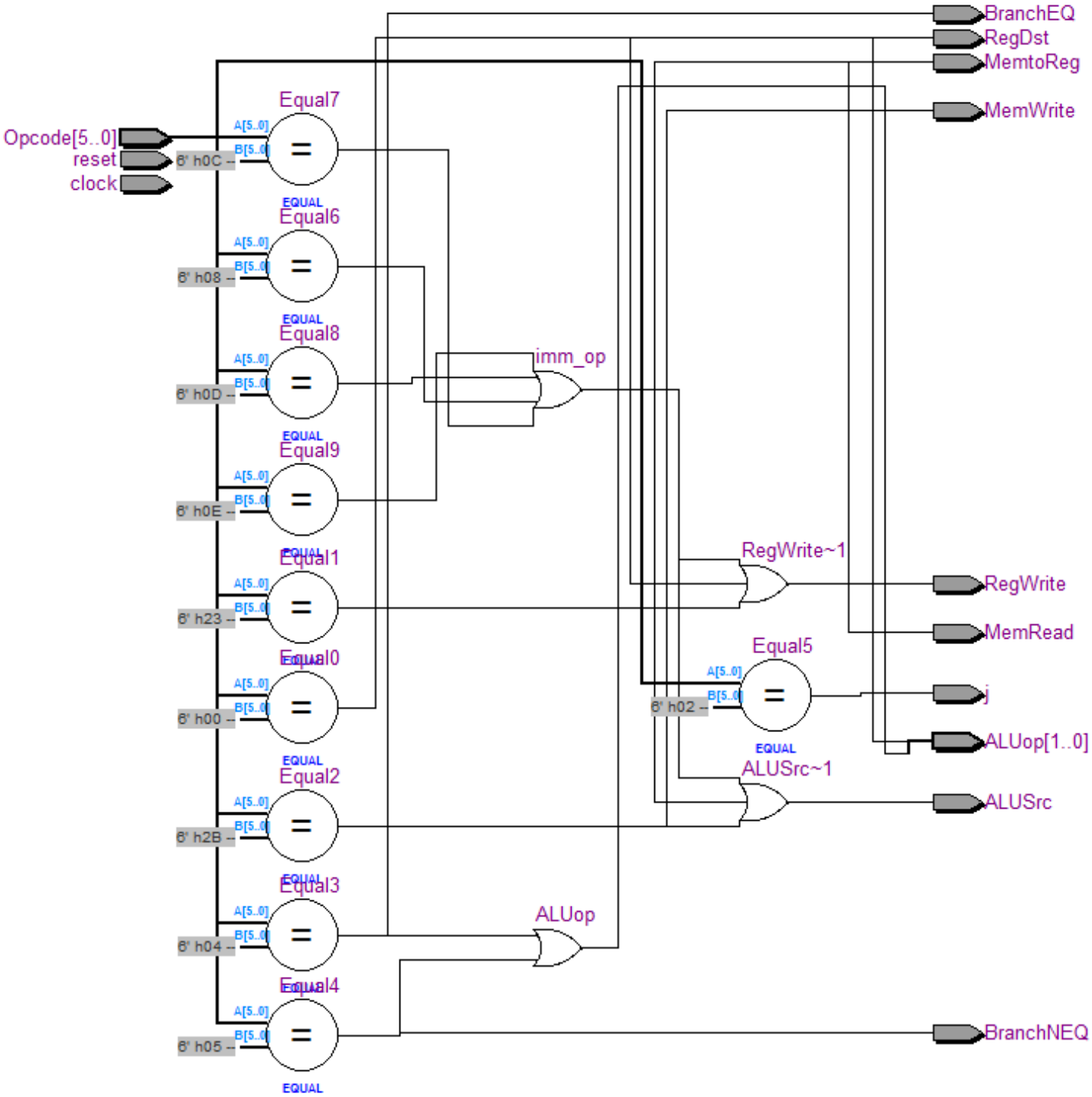
Fetch:


reset
BranchEQ
Zero
BranchNEQ
j
clock
Add_result[7..0]
next_PC~17
next_PC~8
PC~[7..0]
MUX21
PC[9..2]
Add0
ADDER
next_PC~5
next_PC~7
next_PC~1
next_PC~2
next_PC~3
next_PC~4
next_PC~0
next_PC~6
next_PC~14
next_PC~16
next_PC~10
next_PC~11
next_PC~12
next_PC~13
next_PC~9
next_PC~15
next_PC~23
next_PC~25
next_PC~19
next_PC~20
next_PC~21
next_PC~22
next_PC~18
next_PC~24
Mem_Addr[9..2]
MUX21
altsyncram:inst_memory
PC_out[9..0]
PC_plus_4_out[9..0]
Instruction[31..0]

Decode:

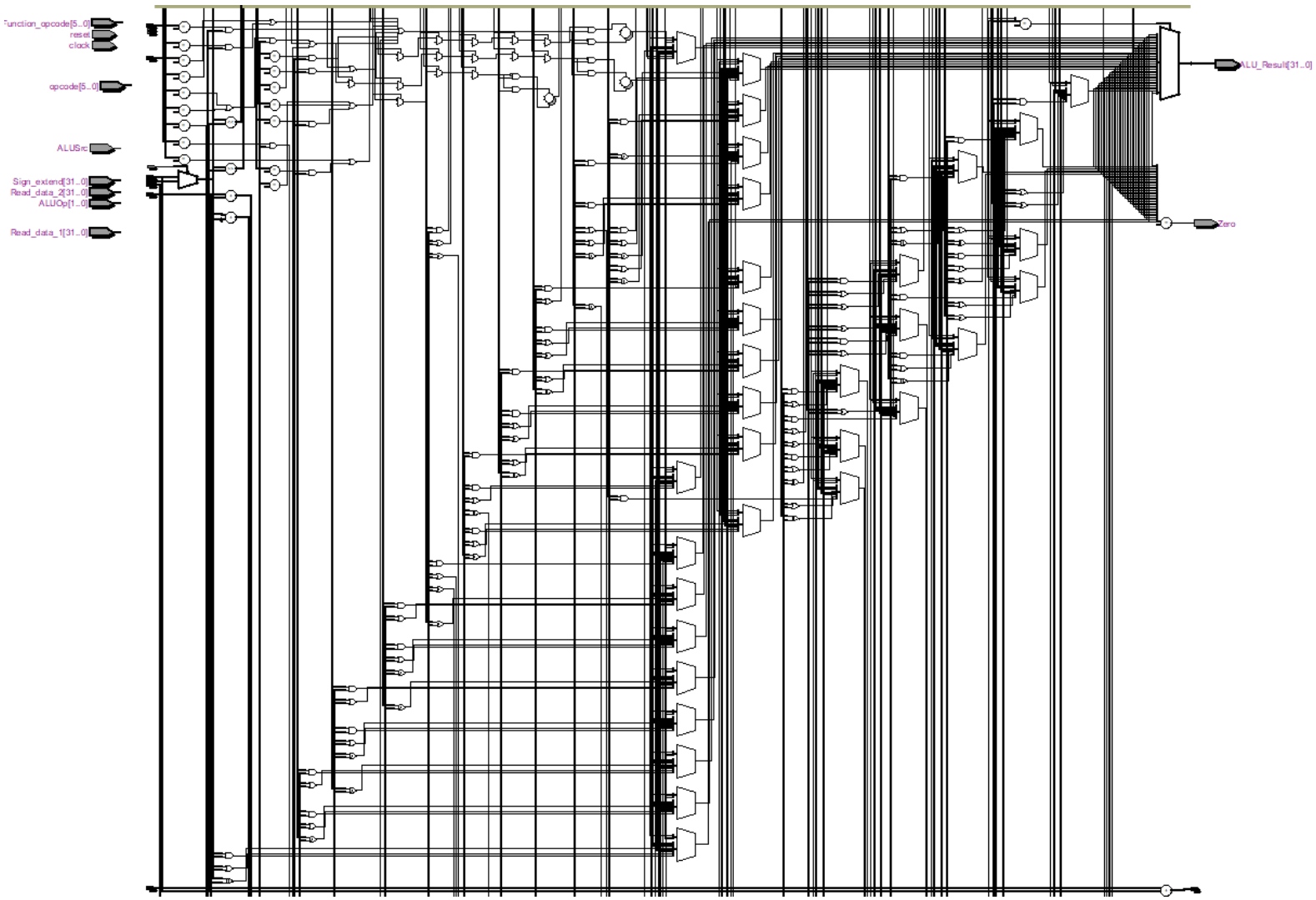

Sign_extend[31..0]
read_data_1[31..0]
read_data_2[31..0]
RegDst
Instruction[31..0]
write_register_address
SEL
DATAA
DATAB
OUT
MUX21
MemtoReg
reset
RegWrite
read_data[31..0]
ALU_result[31..0]
write_data[31..0]
SEL
DATAA
DATAB
OUT
MUX21

Control:



We added control lines for more commands support, like Branch or Jump.

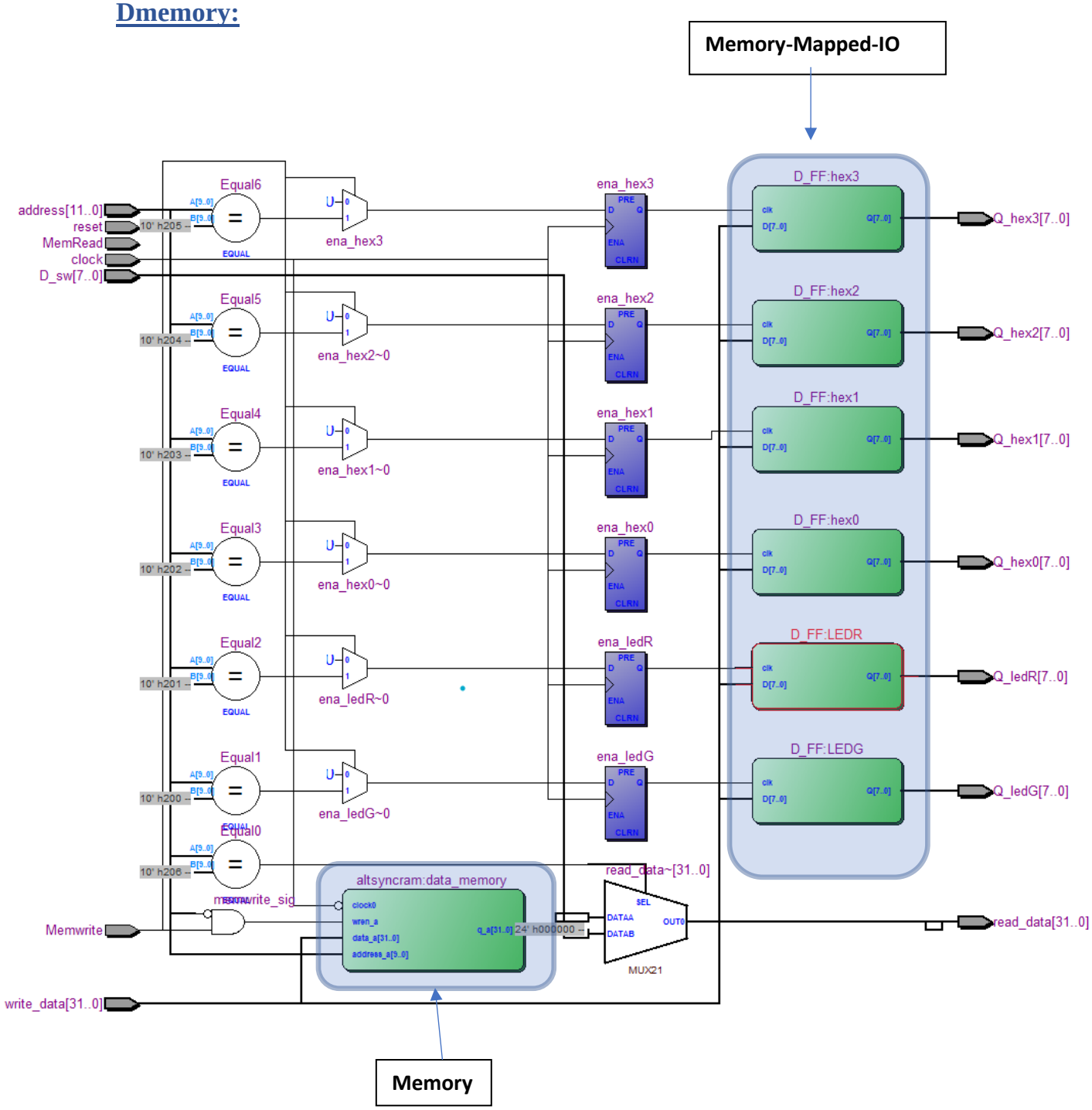
Execute:



Execute is responsible for doing all the arithmetic operations.

here we added the ALU support for AND,OR,SLL,SRL...

Dmemory:



In the memory component, we implemented all the Memory and all the Memory mapped I/O.

We can see the addition of the I/O flip flops here for each Memory mapped I/O that needed to be implemented.

Maximal operating clock:

we have found that the maximum frequency is

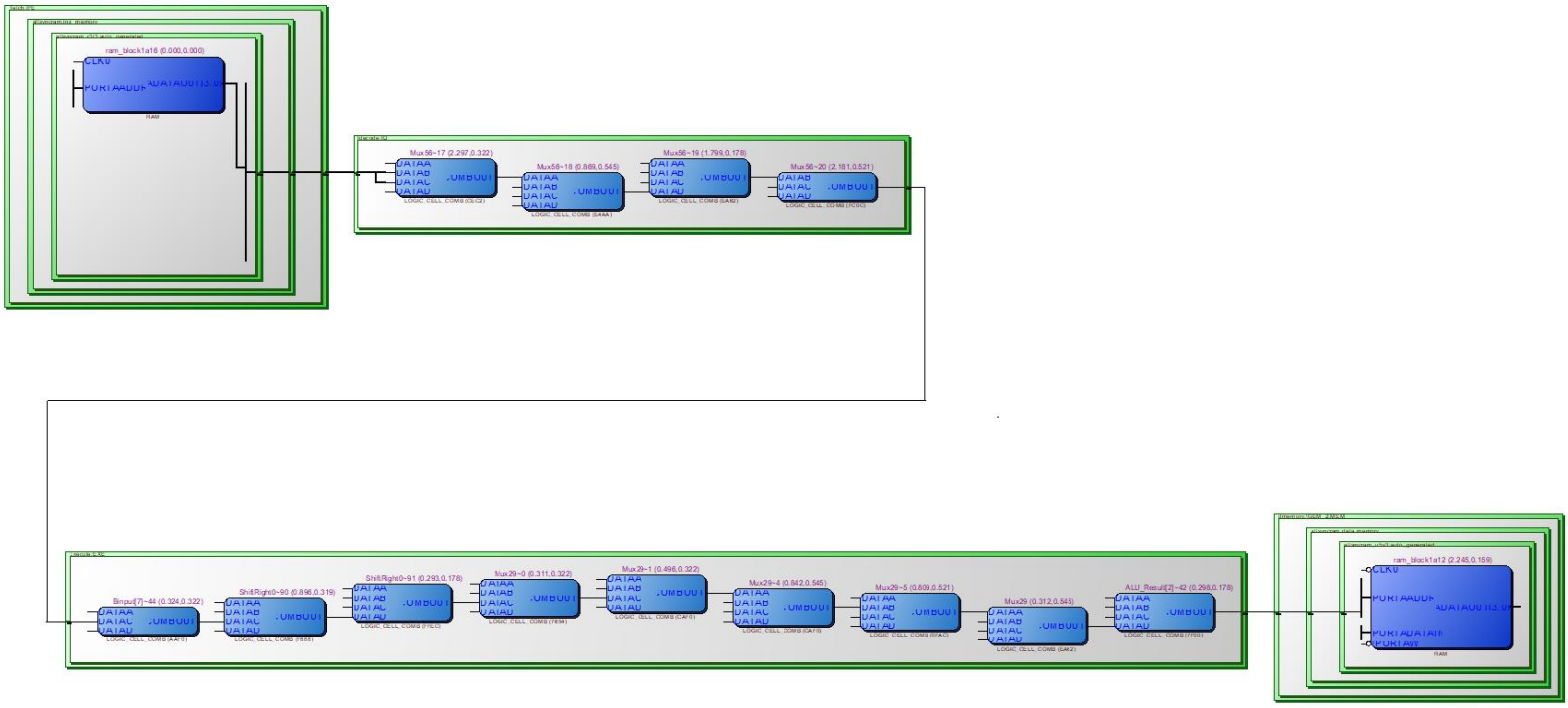
Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	23.7 MHz	23.7 MHz	clock	

So the Maximal operating clock is $\frac{1}{23.7Mhz} = 42.19ns$

Longest path:

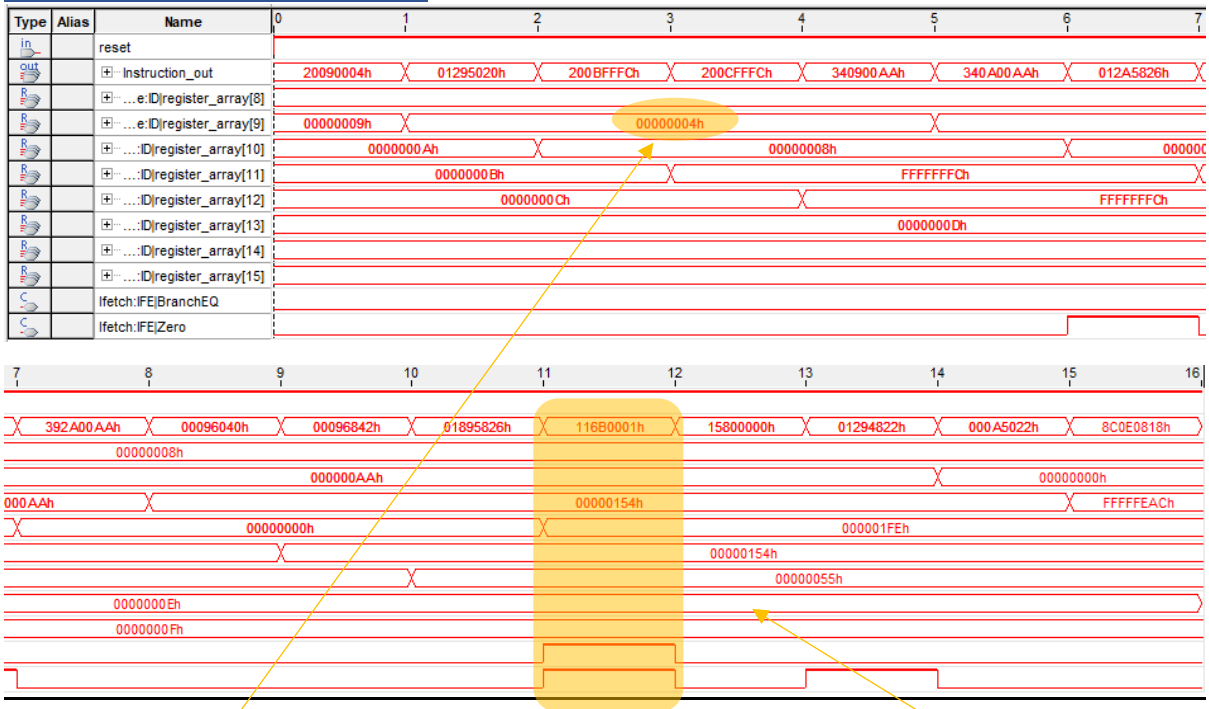
Memmory>decode>execute>memory

Will be longest on shift opcodes



Single-Tap HardWare Test:

Arithmetic Commands Test:



Addi $9,0,4

We are adding immediate 4 to register $9

Beq $11,$11, check1

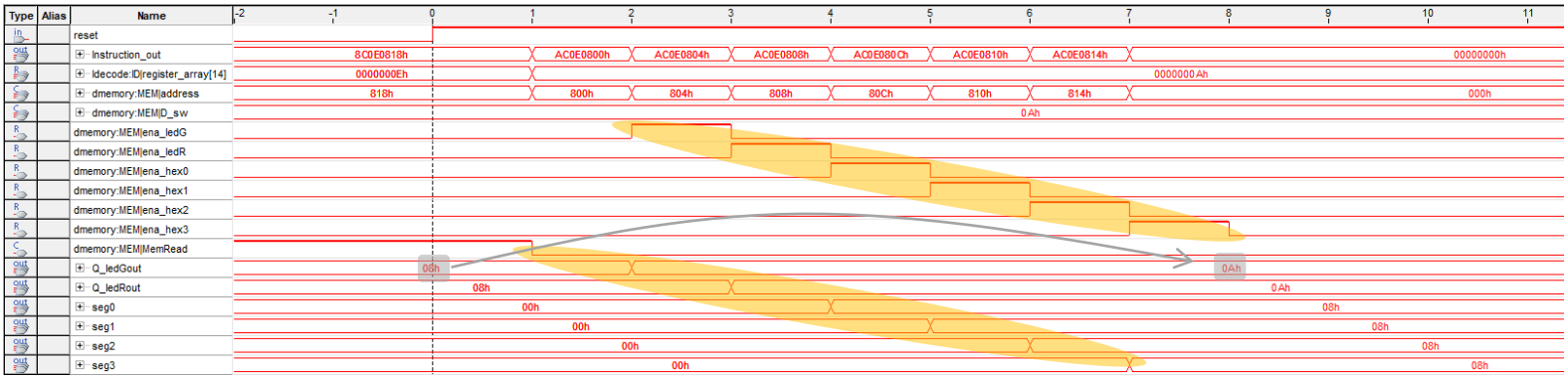
This command is always True, so we should jump to "check1", as we can see. We can also see that the "BranchEq" and "Zero" control line turned on

The code we are running:

Address	Code	Basic		
0x00003000	0x20090004	addi $9,$0,0x00000004	5:	addi $t1, $zero,4
0x00003004	0x01295020	add $10,$9,$9	6:	add $t2, $t1, $t1
0x00003008	0x200bfffc	addi $11,$0,0xfffff...	8:	add $t3, $zero, -4
0x0000300c	0x200cfffc	addi $12,$0,0xfffff...	9:	addi $t4, $zero, -4
0x00003010	0x340900aa	ori $9,$0,0x000000aa	11:	ori $t1, $zero,0xAA
0x00003014	0x340a00aa	ori $10,$0,0x000000aa	12:	or $t2, $zero,0xAA
0x00003018	0x012a5826	XOR $11,$9,$10	14:	XOR $t3, $t1,$t2
0x0000301c	0x392a00aa	XORi $10,$9,0x00000...	15:	XORi $t2, $t1,0xAA
0x00003020	0x00096040	sll $12,$9,0x00000001	17:	sll $t4,$t1, 1
0x00003024	0x00096842	srl $13,$9,0x00000001	18:	srl $t5,$t1, 1
0x00003028	0x01895826	XOR $11,$12,$9	19:	XOR $t3, $t4,$t1
0x0000302c	0x116b0001	beq $11,$11,0x00000...	21:	beq $t3, $t3, check1
0x00003030	0x08000c0c	j 0x00003030	23: loop:	j loop
0x00003034	0x15800000	bne $12,$0,0x00000000	25: check1:	bne $t4, $zero, check2
0x00003038	0x01294822	sub $9,$9,$9	27: check2:	sub $t1, $t1, $t1
0x0000303c	0x000a5022	sub $10,$0,$10	28:	sub $t2, $zero, $t2

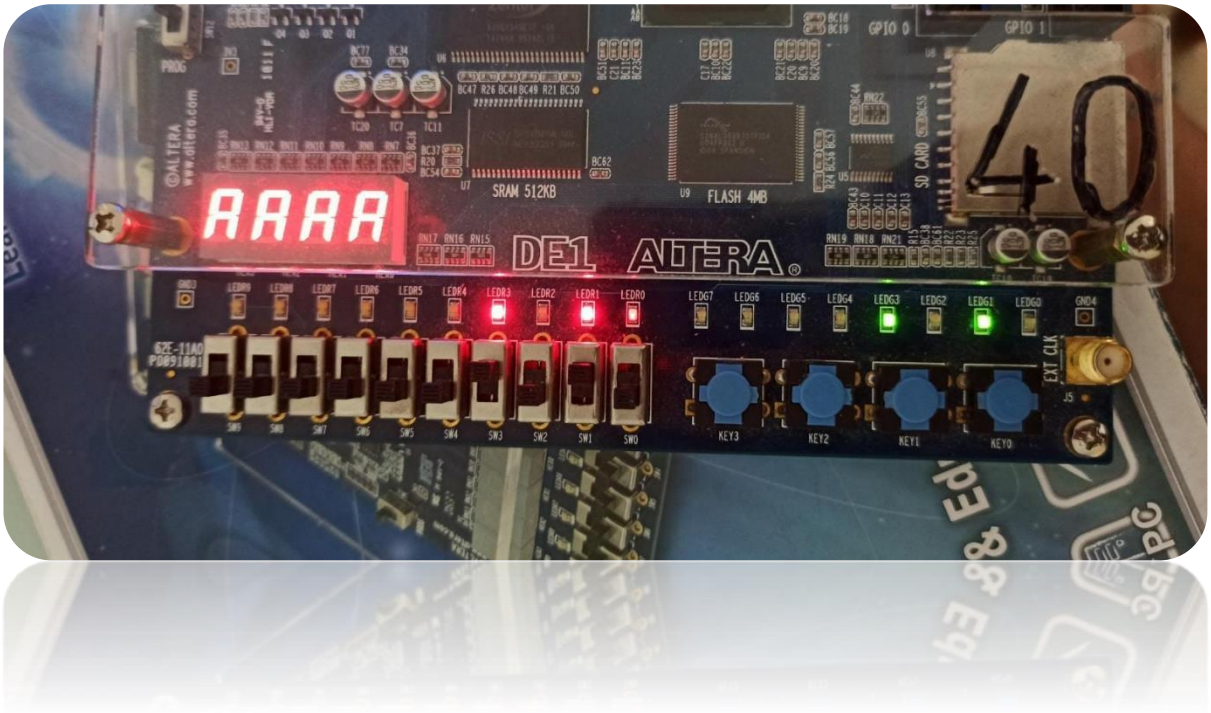
I/O Test:

we tested here the I/O, we read the data from the switches and copied it to the leds and hexes.



In this example the switches were set to 0x0A.

We can see that the enable control to each I/O is set and the value changes to 0x0A



The code we are running:

Address	Code	Basic			
0x00003000	0x8c0e0818	lw $14,0x00000818($0)	6:	lw	$t6,0x818
0x00003004	0xac0e0800	sw $14,0x00000800($0)	7:	sw	$t6,0x800
0x00003008	0xac0e0804	sw $14,0x00000804($0)	8:	sw	$t6,0x804
0x0000300c	0xac0e0808	sw $14,0x00000808($0)	9:	sw	$t6,0x808
0x00003010	0xac0e080c	sw $14,0x0000080c($0)	10:	sw	$t6,0x80C
0x00003014	0xac0e0810	sw $14,0x00000810($0)	11:	sw	$t6,0x810
0x00003018	0xac0e0814	sw $14,0x00000814($0)	12:	sw	$t6,0x814

Sorting array code:

We used bubble sort, it can be done simpler. we implemented it a bit more complicated then it needed to be with verity of commands, only for checking our Mips with different commands.

we chose the number 0x7A1200 for the 1 sec loop as so:

we have 3 cycles in the "loopt6" loop, our clock is 24MHz:

$$3 * B * \frac{1}{24MHz} = 1sec$$

$$B = 0x7A1200$$

```
1  .data
2          array: .word 16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1
3          B: .word 0x7A1200
4          C: .word 64
5  .text
6                                  la $t8,B            # set s1 also to be the address of the array for display on the screen
7                                  lw $t9,0($t8)
8                                  la $t8,C            # set s1 also to be the address of the array for display on the screen
9                                  lw $s3,0($t8)
10                                 move $s0,$zero#la   $s0,array          # set s0 to base address of array , also to be the address of the array for
11                                 move $t0,$zero              # set t0 = 0 (i =0) index of array
12
13 #loop- Sorting loop using good old bubble sort
14 loop:                           beq  $t0,$s3,finish        # if i = 16 then stop
15                                 beq  $t0,$0,INC
16                                 add  $t7,$s0,$t0        # t7 points to the address reltive to the index
17                                 lw   $t1,0($t7)
18                                 lw   $t2,-4($t7)
19
20                                 addi $t1, $t1,1
21                                 slt $t5, $t2,$t1
22
23                                 beq  $t5, $zero, swap         # if array[i]<array[i-1] then swap
24
25 #INC - moving to the next cell in the array
26 INC:                       addi $t0,$t0,4
27                                 j    loop
28
29 #Swap - Swaps two cells in the array at array[$t7] <-> array[$t7-4]
30 swap:                           addi $t1, $t1, -1   #to return $t2 to its original value( because of line 16 slt command)
31                                 sw   $t2,0($t7)
32                                 sw   $t1,-4($t7)
33                                 sub  $t0,$t0,4         # i = i -1
34                                 j    loop
35
36
37 #finish- after the array is sorted, display on green leds at mem(800)
38 finish:                         la $s2,array                  # set s1 also to be the address of the array for display on the screen
39                                 lw $t0,0($s2)
40                                 sw $t0,0x800
41                                  j one_sec_timer
42
43 #show- place the next cell in the array on leds until the end and repeat
44 show:                      beq  $s2,$s3,finish        # if t1 = 64 then startover
45                            addi  $s2,$s2,4          # going to the next number in the array
46                            lw $t0,0($s2)
47                            sw $t0,0x800
48
49  #one_sec_timer- 24M/3 commands  for 1 sec in 24mhz clock  ($t9 =  0x7A1200)
50 one_sec_timer:             move $t6,$zero
51                            move $t3,$zero
52 loopt6:                    beq $t6,$t9,show
53                            addi $t6,$t6,1
54                            j loopt6
```