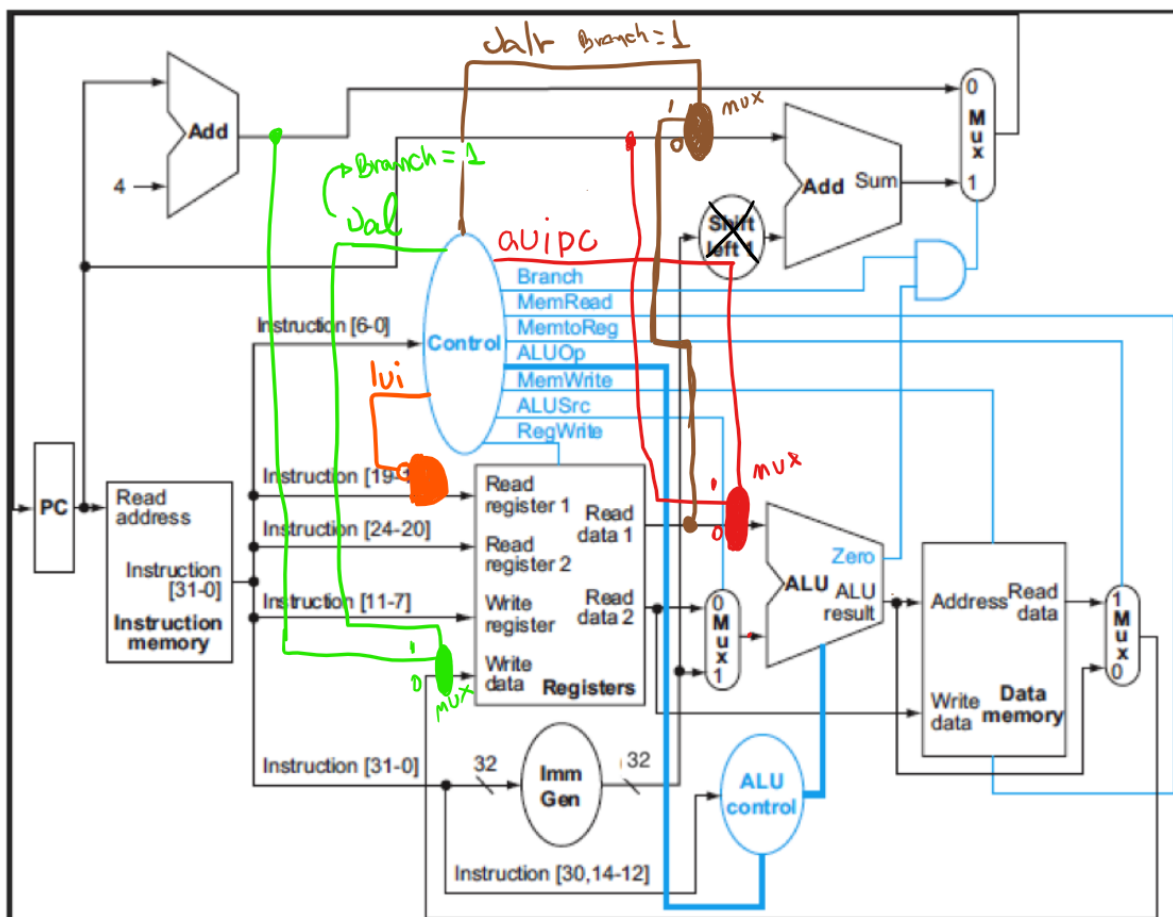


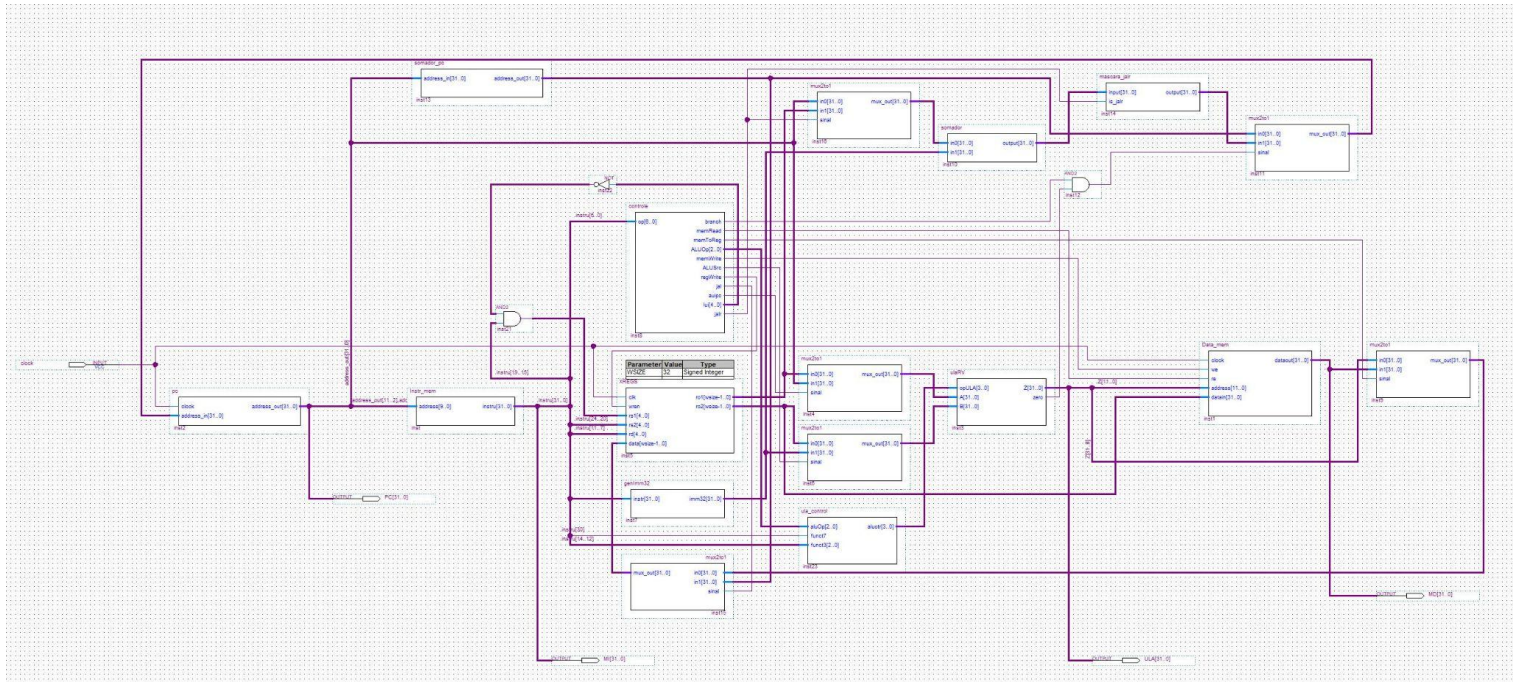
# Relatório Projeto Final OAC

Nicolas Meloni – 222001369

Adriano Ulrich Do Prado Wiedmann – 202014824

Neste projeto uma arquitetura RiscV, em sua versão uniciclo, foi implementada em linguagem VHDL. Para testes e desenvolvimento do projeto, foram utilizadas as plataformas Quartus II e ModelSim. Um diagrama esquemático do projeto foi criado, com todos os componentes programados em VHDL. O processador é capaz de executar 32 instruções assembly, necessitando apenas do código da área de texto, em hexadecimal, e do código da área de dados, também em base 16. Abaixo é possível ver um desenho do planejamento do projeto e o seu diagrama esquemático.





Algumas mudanças, em comparação à versão original, foram feitas. Uma das que valem ressaltar, é a que foi feita no controle da ULA e, conseqüentemente, no controle do processador: um bit a mais foi adicionado no campo ALUOp. Essa ideia foi levada em consideração pois foi percebida a necessidade de ter mais opções para realizar operações específicas com a ULA. Como, ao adicionar mais 1 bit, teríamos o dobro de possibilidades, usamos isso ao nosso favor:

- 000 – adição casual
- 001 – todas operações de BRANCH
- 010 – Lógico Aritméticas com tipo R
- 010 – Lógico Aritméticas com tipo I
- 100 – JAL
- 101 – JALR
- 110 – x
- 111 – x

As operações JAL e JALR foram implementadas na ULA para que o sinal zero pudesse ser ativado incondicionalmente caso uma dessas instruções fosse identificada. Isso podia ter sido feito utilizando apenas um único código ALUOp, porém, a fim de conseguir o mesmo resultado mostrado no exemplo do professor, essa foi a forma pensada para fazer com que a soma do jalr e o zero do jal aparecessem na saída da ULA.

O código VHDL do processador em si foi gerado a partir do diagrama feito, o que foi suficiente para que o código pudesse ser executado com uma testbench adequada. Foi escolhido fazer o bloco completo em diagrama esquemático pois, além de diminuir um pouco mais a abstração do problema, o desenvolvimento do projeto se torna mais agradável.

Os sinais de controle implementados são:

branch  
memRead  
memToReg  
ALUOp  
memWrite  
ALUSrc  
regWrite

E os sinais adicionados para o projeto:

jal - Para quando for instrução jal;  
auipc - Para quando for instrução auipc;  
lui - Para quando for instrução lui;  
jalr - Para quando for instrução jalr.

### **Memória de Instruções:**

Na Memória de Instruções é lido um arquivo code.txt com as instruções. Nesta memória é permitido apenas a leitura.

### **Memória de Dados:**

Na Memória de Dados é lido um arquivo data.txt com os dados. Nesta memória é possível ler e armazenar dados, que é controlado através dos sinais MemWrite e MemRead.

### **Simulação:**

Foi utilizado o arquivo uniciclo.asm disponibilizado pelo professor via Microsoft Teams, e assim tendo as instruções na Memória de Instruções e os dados na Memória de Dados. Para a realização do teste foi criado um testbench, chamado tb\_uniciclo.vhd, com um loop for de 0 a 40 para testar todas as instruções. A realização dos testes pode ser conferida nas imagens abaixo.

Obs: Foi utilizado a versão 2008 do vhd para realização dos testes.






#### **Instrução (ori t0, zero, 0xFF)**

/tb_uniciclo/dut/clock	0	
/tb_uniciclo/dut/MD	00000000	00000000
/tb_uniciclo/dut/MI	0FF06293	0FF06293
/tb_uniciclo/dut/P_C	00000000	00000000
/tb_uniciclo/dut/ULA	000000FF	000000FF


#### **Instrução (andi t0, t0, 0xF0)**

/tb_uniciclo/dut/clock	1	
/tb_uniciclo/dut/MD	00000000	00000000
/tb_uniciclo/dut/MI	0F02F293	0F02F293
/tb_uniciclo/dut/P_C	00000004	00000004
/tb_uniciclo/dut/ULA	000000F0	000000F0






### Instrução (lui s0, 2)

 /tb_unicido/dut/clock	1			
+  /tb_unicido/dut/MD	00000000	00000000		
+  /tb_unicido/dut/MI	00002437	00002437		
+  /tb_unicido/dut/P_C	00000008	00000008		
+  /tb_unicido/dut/ULA	00002000	00002000		






### Instrução (lw s1, 0(s0) e lw s2, 4(s0))

 /tb_unicido/dut/clock	1					
+  /tb_unicido/dut/MD	0000000F	0000000F		0000003F		
+  /tb_unicido/dut/MI	00042483	00042483		00442903		
+  /tb_unicido/dut/P_C	0000000C	0000000C		00000010		
+  /tb_unicido/dut/ULA	00002000	00002000		00002004		






### Instrução (add s3, s1, s2)

 /tb_unicido/dut/clock	1			
+  /tb_unicido/dut/MD	00000000	00000000		
+  /tb_unicido/dut/MI	012489B3	012489B3		
+  /tb_unicido/dut/P_C	00000014	00000014		
+  /tb_unicido/dut/ULA	0000004E	0000004E		






### Instrução (sw s3, 8(s0) e lw a0, 8(s0))

 /tb_unicido/dut/clock	1				
+  /tb_unicido/dut/MD	00000000	00000000		0000004E	
+  /tb_unicido/dut/MI	01342423	01342423		00842503	
+  /tb_unicido/dut/P_C	00000018	00000018		0000001C	
+  /tb_unicido/dut/ULA	00002008	00002008		00002008	


### Instrução (addi s4, zero, 0x7F0 e addi s5, zero, 0x0FF)

 /tb_unicido/dut/clock	1				
+  /tb_unicido/dut/MD	00000000	00000000			
+  /tb_unicido/dut/MI	7F000A13	7F000A13		0FF00A93	
+  /tb_unicido/dut/P_C	00000020	00000020		00000024	
+  /tb_unicido/dut/ULA	000007F0	000007F0		000000FF	






### Instrução (and s6, s5, s4)

 /tb_unicido/dut/clock	1			
+  /tb_unicido/dut/MD	00000000	00000000		
+  /tb_unicido/dut/MI	014AFB33	014AFB33		
+  /tb_unicido/dut/P_C	00000028	00000028		
+  /tb_unicido/dut/ULA	000000F0	000000F0		






### Instrução (or s7, s5, s4)

 /tb_unicido/dut/clock	1			
+  /tb_unicido/dut/MD	00000000	00000000		
+  /tb_unicido/dut/MI	014AEBB3	014AEBB3		
+  /tb_unicido/dut/P_C	0000002C	0000002C		
+  /tb_unicido/dut/ULA	000007FF	000007FF		






### Instrução (xor s8, s5, s4)

 /tb_unicido/dut/clock	1			
+  /tb_unicido/dut/MD	00000000	00000000		
+  /tb_unicido/dut/MI	014ACC33	014ACC33		
+  /tb_unicido/dut/P_C	00000030	00000030		
+  /tb_unicido/dut/ULA	0000070F	0000070F		






### Instrução (slli t1, s5, 4)

 /tb_unicido/dut/clock	1			
+  /tb_unicido/dut/MD	00000000	00000000		
+  /tb_unicido/dut/MI	004A9313	004A9313		
+  /tb_unicido/dut/P_C	00000034	00000034		
+  /tb_unicido/dut/ULA	00000FF0	00000FF0		






### Instrução (lui t2, 0xFF000)

 /tb_unicido/dut/clock	1			
+  /tb_unicido/dut/MD	00000000	00000000		
+  /tb_unicido/dut/MI	FF0003B7	FF0003B7		
+  /tb_unicido/dut/P_C	00000038	00000038		
+  /tb_unicido/dut/ULA	FF000000	FF000000		

### Instrução (srli t3, t2, 4)

 /tb_unico/dut/clock	1				
+  /tb_unico/dut/MD	00000000	00000000			
+  /tb_unico/dut/MI	0043DE13	0043DE13			
+  /tb_unico/dut/P_C	0000003C	0000003C			
+  /tb_unico/dut/ULA	0FF00000	0FF00000			






### Instrução (srai t4, t2, 4)

 /tb_unico/dut/clock	1				
+  /tb_unico/dut/MD	00000000	00000000			
+  /tb_unico/dut/MI	4043DE93	4043DE93			
+  /tb_unico/dut/P_C	00000040	00000040			
+  /tb_unico/dut/ULA	FFF00000	FFF00000			

### Instrução (slt s0, t0, t1 e slt s1, t1, t0)

 /tb_unico/dut/clock	1				
+  /tb_unico/dut/MD	00000000	00000000			
+  /tb_unico/dut/MI	0062A433	0062A433	005324B3		
+  /tb_unico/dut/P_C	00000044	00000044	00000048		
+  /tb_unico/dut/ULA	00000001	00000001	00000000		

### Instrução (sltu s3, zero, t0 e sltu s4, t0, zero)






 /tb_unico/dut/clock	1				
+  /tb_unico/dut/MD	00000000	00000000			
+  /tb_unico/dut/MI	005039B3	005039B3	0002BA33		
+  /tb_unico/dut/P_C	0000004C	0000004C	00000050		
+  /tb_unico/dut/ULA	00000001	00000001	00000000		

### Instrução (jal ra, testasub e salta para PC = 5C que realiza a instrução sub t3, t0, t1)






 /tb_unico/dut/clock	1				
+  /tb_unico/dut/MD	00000000	00000000			
+  /tb_unico/dut/MI	008000EF	008000EF	40628E33		
+  /tb_unico/dut/P_C	00000054	00000054	0000005C		
+  /tb_unico/dut/ULA	00000000	00000000	FFFFF100		



**Instrução (jalr x0, ra, 0 e salta para PC = 58)**

 /tb_unico/dut/clock	1						
+  /tb_unico/dut/MD	00000000	00000000					
+  /tb_unico/dut/MI	00008067	00008067			} 00C0006F		
+  /tb_unico/dut/P_C	00000060	00000060			} 00000058		
+  /tb_unico/dut/ULA	00000058	00000058			} 00000000		

**Instrução (jal x0, next e salta para PC = 64 que realiza a instrução addi t0, zero, -2)**

 /tb_unico/dut/clock	1						
+  /tb_unico/dut/MD	00000000	00000000					
+  /tb_unico/dut/MI	00C0006F	00C0006F			} FFE00293		
+  /tb_unico/dut/P_C	00000058	00000058			} 00000064		
+  /tb_unico/dut/ULA	00000000	00000000			} FFFFFFFE		

## Função BEQSIM que realiza teste da instrução BEQ:

beqsim:

```
addi t0, t0, 2          # 00000068 00228293 00000000* 00000000 * t0 = 0, 2
beq t0, zero, beqsim    # 0000006c fe028ee3 00000000 00000000 => 68, 70
```

PC = 68

/tb_unico/dut/dock	1			
+ /tb_unico/dut/MD	00000000	00000000		
+ /tb_unico/dut/MI	00228293	00228293		
+ /tb_unico/dut/P_C	00000068	00000068		
+ /tb_unico/dut/ULA	00000000	00000000		

PC = 6C

/tb_unico/dut/dock	1			
+ /tb_unico/dut/MD	00000000	00000000		
+ /tb_unico/dut/MI	FE028EE3	FE028EE3		
+ /tb_unico/dut/P_C	0000006C	0000006C		
+ /tb_unico/dut/ULA	00000000	00000000		

PC = 68

/tb_unico/dut/dock	1			
+ /tb_unico/dut/MD	00000000	00000000		
+ /tb_unico/dut/MI	00228293	00228293		
+ /tb_unico/dut/P_C	00000068	00000068		
+ /tb_unico/dut/ULA	00000002	00000002		

PC = 6C

/tb_unico/dut/dock	1			
+ /tb_unico/dut/MD	00000000	00000000		
+ /tb_unico/dut/MI	FE028EE3	FE028EE3		
+ /tb_unico/dut/P_C	0000006C	0000006C		
+ /tb_unico/dut/ULA	00000002	00000002		



## Função BNESIM que realiza teste da instrução BNE:

bn esim:

```
addi t0, t0, -1      # 00000070 fff28293 0000000* 00000000 * t0 = 1, 0
bne t0, zero, bn esim # 00000074 fe029ee3 00000000 00000000 => 70, 78
```

```
addi t0, zero, 1     # 00000078 00100293 00000001 xxxxxxxx
```

PC = 70

/tb_unicido/dut/clock	1			
+ /tb_unicido/dut/MD	00000000	00000000		
+ /tb_unicido/dut/MI	FFF28293	FFF28293		
+ /tb_unicido/dut/P_C	00000070	00000070		
+ /tb_unicido/dut/ULA	00000001	00000001		

PC = 74

/tb_unicido/dut/clock	1			
+ /tb_unicido/dut/MD	00000000	00000000		
+ /tb_unicido/dut/MI	FE029EE3	FE029EE3		
+ /tb_unicido/dut/P_C	00000074	00000074		
+ /tb_unicido/dut/ULA	00000000	00000000		

PC = 70

/tb_unicido/dut/clock	1			
+ /tb_unicido/dut/MD	00000000	00000000		
+ /tb_unicido/dut/MI	FFF28293	FFF28293		
+ /tb_unicido/dut/P_C	00000070	00000070		
+ /tb_unicido/dut/ULA	00000000	00000000		

PC = 74

/tb_unicido/dut/clock	1			
+ /tb_unicido/dut/MD	00000000	00000000		
+ /tb_unicido/dut/MI	FE029EE3	FE029EE3		
+ /tb_unicido/dut/P_C	00000074	00000074		
+ /tb_unicido/dut/ULA	00000001	00000001		

PC = 78

/tb_unicido/dut/clock	1			
+ /tb_unicido/dut/MD	00000000	00000000		
+ /tb_unicido/dut/MI	00100293	00100293		
+ /tb_unicido/dut/P_C	00000078	00000078		
+ /tb_unicido/dut/ULA	00000001	00000001		

## Função BLTADD que realiza teste da instrução BLT:

bltadd:

```
addi t0, t0, -1      # 0000007c fff28293 00000000 xxxxxxxx
blt  t0, zero, blton  # 00000080 0002c463 xxxxxxxx xxxxxxxx => 84, 88
j    bltadd           # 00000084 ff9ff06f xxxxxxxx xxxxxxxx => 7c
```

### PC = 7C

/tb_unico/dut/dock	1			
/tb_unico/dut/MD	00000000	00000000		
/tb_unico/dut/MI	FFF28293	FFF28293		
/tb_unico/dut/P_C	0000007C	0000007C		
/tb_unico/dut/JLA	00000000	00000000		

### PC = 80

/tb_unico/dut/dock	1			
/tb_unico/dut/MD	00000000	00000000		
/tb_unico/dut/MI	0002C463	0002C463		
/tb_unico/dut/P_C	00000080	00000080		
/tb_unico/dut/JLA	00000001	00000001		

### PC = 84

/tb_unico/dut/dock	1			
/tb_unico/dut/MD	00000000	00000000		
/tb_unico/dut/MI	FF9FF06F	FF9FF06F		
/tb_unico/dut/P_C	00000084	00000084		
/tb_unico/dut/JLA	00000000	00000000		

### PC = 7C

/tb_unico/dut/dock	1			
/tb_unico/dut/MD	00000000	00000000		
/tb_unico/dut/MI	FFF28293	FFF28293		
/tb_unico/dut/P_C	0000007C	0000007C		
/tb_unico/dut/JLA	FFFFFFFF	FFFFFFFF		

### PC = 80 e pula para PC = 88

/tb_unico/dut/dock	1				
/tb_unico/dut/MD	00000000	00000000			
/tb_unico/dut/MI	0002C463	0002C463	0002D663		
/tb_unico/dut/P_C	00000080	00000080	00000088		
/tb_unico/dut/JLA	00000000	00000000	00000001		

## Função BLTON que realiza teste da instrução BGE:

blton:

```

bge t0, zero, end    # 00000088 0002d663 xxxxxxxx xxxxxxxx => 8c, 94
addi t0, t0, 1        # 0000008c 00128293 00000000 xxxxxxxx
j blton               # 00000090 ff9ff06f xxxxxxxx xxxxxxxx => 88

```

**PC = 88**

/tb_unico/dut/clock	1			
/tb_unico/dut/MD	00000000	00000000		
/tb_unico/dut/MI	0002D663	0002D663		
/tb_unico/dut/P_C	00000088	00000088		
/tb_unico/dut/ULA	00000001	00000001		

**PC = 8C**

/tb_unico/dut/clock	1			
/tb_unico/dut/MD	00000000	00000000		
/tb_unico/dut/MI	00128293	00128293		
/tb_unico/dut/P_C	0000008C	0000008C		
/tb_unico/dut/ULA	00000000	00000000		

**PC = 90**

/tb_unico/dut/clock	1			
/tb_unico/dut/MD	00000000	00000000		
/tb_unico/dut/MI	FF9FF06F	FF9FF06F		
/tb_unico/dut/P_C	00000090	00000090		
/tb_unico/dut/ULA	00000000	00000000		

**PC = 88 e pula para PC = 94**

/tb_unico/dut/clock	1			
/tb_unico/dut/MD	00000000	00000000		
/tb_unico/dut/MI	0002D663	0002D663	0000A317	
/tb_unico/dut/P_C	00000088	00000088	00000094	
/tb_unico/dut/ULA	00000000	00000000	0000A094	

### Instrução (auipc t1, 10)

/tb_unico/dut/clock	1				
/tb_unico/dut/MD	00000000	00000000			
/tb_unico/dut/MI	0000A317	0000A317			
/tb_unico/dut/P_C	00000094	00000094			
/tb_unico/dut/ULA	0000A094	0000A094			

### Instrução (addi t0, x0, 10 e addi t1, x0, 20 e add t2, t0, t1)

/tb_unico/dut/clock	1								
/tb_unico/dut/MD	00000000	00000000							
/tb_unico/dut/MI	00A00293	00A00293		01400313			00628383		
/tb_unico/dut/P_C	00000098	00000098		0000009C			000000A0		
/tb_unico/dut/ULA	0000000A	0000000A		00000014			0000001E		