Arquitetura e Organização de Computadores
Turma C - 2015/02
Projeto MIPS Multiciclo

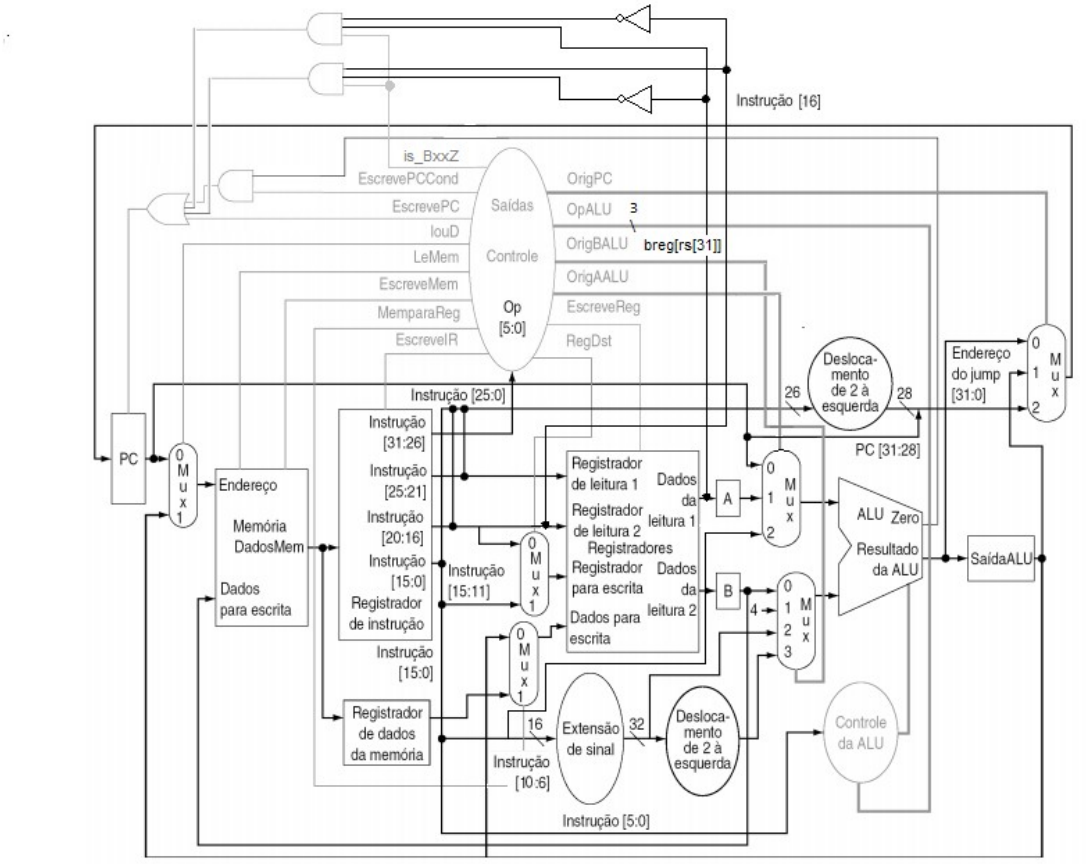
Alunos:
Amanda Lopes Dantas 13/0100391
Gabriel Martins de Miranda 13/0111350

1. Descrição do Projeto

O objetivo do projeto é o de implementar o processador MIPS Multiciclo numa FPGA, sendo necessário instanciar os módulos fornecidos no site da disciplina e adicionar as instruções ORI, ANDI, SLL, SRL, BGEZ, BLTZ e SLTI.

2. Descrição da implementação das instruções adicionais

2.1 Novo esquemático do processador



2.2 Descrição das novas instruções

→ **Tipo I**

ORI: ori rs, rt, imediato
breg[rt] = breg[rs] | 0x0000iiii; *extendido com zeros

ANDi: andi rs, rt, imediato
breg[rt] = breg[rs] & 0x0000iiii; *extendido com zeros

BGEZ: bgez rs, label //if breg[rs] >= 0 then branch
pc = pc + deslocamento*4;

BLTZ: bgez rs, label //if breg[rs] < 0 then branch
pc = pc + deslocamento*4;

SLTI: slti rt, rs, imediato //rt <- (rs < immediate)
breg[rt] = breg[rt] < sgn_ext(imediato);

→ **TIPO R**
SLL: sll rd, rt, shamt
breg[rd] = breg[rt] << shamt;

SRL: srl rd, rt, shamt
breg[rd] = breg[rt] >> shamt;

2.3 Descrição das alterações fundamentais no código

→ **mips_pkg:** adicionados campos opcode (para o controle do MIPS) e funct (para o controle da ULA) das novas instruções.

```
18    -- Instrucoes do MIPs (OPCODES)                    **TODOS
19    constant iRTYPE      : std_logic_vector(5 downto 0) := "000000";
20    constant iLW         : std_logic_vector(5 downto 0) := "100011";
21    constant iSW         : std_logic_vector(5 downto 0) := "101011";
22    constant iJ          : std_logic_vector(5 downto 0) := "000010";
23    constant iBEQ        : std_logic_vector(5 downto 0) := "000100";
24    constant iBNE        : std_logic_vector(5 downto 0) := "000101";
25    ------------------------------------------------------------
26    constant iADDI       : std_logic_vector(5 downto 0) := "001000"; --I-TYPE
27    constant iORI        : std_logic_vector(5 downto 0) := "001101"; --I-TYPE
28    constant iANDI       : std_logic_vector(5 downto 0) := "001100"; --I-TYPE
29    constant iBxxZ       : std_logic_vector(5 downto 0) := "000001"; --I-TYPE
30    constant iSLTI       : std_logic_vector(5 downto 0) := "001010"; --I-TYPE
31
32    -- Campo funct         (FUNCT - para controle da ULA) **PARA R-TYPES
33    constant iADD        : std_logic_vector(5 downto 0) := "100000";
34    constant iSUB        : std_logic_vector(5 downto 0) := "100010";
35    constant iAND        : std_logic_vector(5 downto 0) := "100100";
36    constant iOR         : std_logic_vector(5 downto 0) := "100101";
37    constant iXOR        : std_logic_vector(5 downto 0) := "100110";
38    constant iNOR        : std_logic_vector(5 downto 0) := "100111";
39    constant iSLT        : std_logic_vector(5 downto 0) := "101010";
40    constant iSLL        : std_logic_vector(5 downto 0) := "000000"; --R-TYPE
41    constant iSRL        : std_logic_vector(5 downto 0) := "000010"; --R-TYPE
42    constant iSRA        : std_logic_vector(5 downto 0) := "000011";
```

→ **mips_control:** adicionado o controle do MIPS a ser tomado e o seguimento das etapas do processador.

```
134      case pstate is
135      when fetch_st =>  nstate <= decode_st;
136      when decode_st => case opcode is
137                        when iRTYPE => nstate <= rtype_ex_st;             --execute Rtype
138                        when iLW | iSW | iADDI | iORI | iANDI | iSLTI => nstate <= c_mem_add_st; --execute lw,sw,I
139                        when iBEQ | iBNE | iBxxZ => nstate <= branch_ex_st;--execute branch
140                        when iJ => nstate <= jump_ex_st;                  --execute jump
141                        when others => null;
142                        end case;
143      when c_mem_add_st => case opcode is
144                        when iLW => nstate <= readmem_st;                                  --mem_access load
145                        when iSW => nstate <= writemem_st;                                 --mem_access store
146                        when iADDI | iORI | iANDI | iSLTI =>  nstate <= arith_imm_st;       --writeback dos tipoI
147                        when others => null;
148                     end case;
149      when readmem_st   => nstate <= ldreg_st;                          --writeback load
150      when rtype_ex_st  => nstate <= writereg_st;                       --writeback dos tipoR
151      when others       => null;
```

→ **alu_ctr:** adicionado comportamento da ULA para as novas instruções.

```
21   alu_ctr <= -- Default (PC+4)
22               ULA_ADD when (op_alu="000") else
23               -- Type-R
24               ULA_AND when (op_alu="010" and funct=iAND) else    --AND
25               ULA_OR  when (op_alu="010" and funct=iOR)  else    --OR
26               ULA_XOR when (op_alu="010" and funct=iXOR) else    --XOR
27               ULA_SLL when (op_alu="010" and funct=iSLL) else    --SLL - shift left logical
28               ULA_SRA when (op_alu="010" and funct=iSRA) else    --SRA - shift right arith
29               ULA_SRL when (op_alu="010" and funct=iSRL) else    --SRL - shift right logical
30               ULA_ADD when (op_alu="010" and funct=iADD) else    --ADD
31               ULA_SUB when (op_alu="010" and funct=iSUB) else    --SUB
32               ULA_SLT when (op_alu="010" and funct=iSLT) else    --SLT
33               ULA_NOR when (op_alu="010" and funct=iNOR) else    --NOR
34               -- Type-I
35               ULA_SUB  when (op_alu="011") else                      --BEQ, BGEZ, BLTZ
36               ULA_OR   when (op_alu="100") else                      --ORI
37               ULA_AND  when (op_alu="101") else                      --ANDI
38               ULA_SLT  when (op_alu="110") else                      --SLTI
39               ULA_NOP;
```

3. Estratégia de verificação e teste do projeto

3.1 Código MIPS utilizado para os testes

Instrução | nº ciclos | endereço | saida_ULA

```
1   .data
2           aux: .word 0xFA00              #64000
3           aux2: .word 0x000002           #2
4   .text
5   LB0:    lw   $t2,aux                   #5 ciclos       - 0     - indefinido ('1' & alu_out(8 downto 2))
6           addi $t1,$t2,0x00CA            #4 ciclos       - 4     - 0x0000faca
7           ori  $t2,$t1,0x00000FF         #4 ciclos       - 8     - 0x0000faff
8           andi $t3,$t1,0x000000FF        #4 ciclos       - 12    - 0x000000ca
9           sll  $t3,$t3,2                 #4 ciclos       - 16    - 0x00000328
10          srl  $t2,$t2,2                 #4 ciclos       - 20    - 0x00003ebf
11          bgez $t3,LB                    #3 ciclos       - 24    - nada
12          add  $t2,$t1,$t2               #pulado         - 28    - pulado
13  LB:     lw   $t4,aux2                  #5 ciclos       - 32    - indefinido ('1' & alu_out(8 downto 2))
14          addi $t2,$t4, -6               #4 ciclos       - 36    - 0xfffffffc
15          bltz $t2,LB1                   #3 ciclos       - 40    - nada
16          addi $t2,$t2, 8                #pulado         - 44    - pulado
17  LB1:    slti $t3,$t2, 4                #4 ciclos       - 48    - 0x00000001
18          bgez $t2,LB0                   #3 ciclos       - 52    - nada
19          bltz $t1,LB1                   #3 ciclos       - 56    - nada
20          addi $t1,$t1,100               #4 ciclos       - 60    - 0x0000fb2e
21          j    LB                        #3 ciclos       - 64    - nada
22                                         #total = 10 + 32 + 15 = 57 ciclos = 15 instrucoes
```

3.2 Código MIF associado

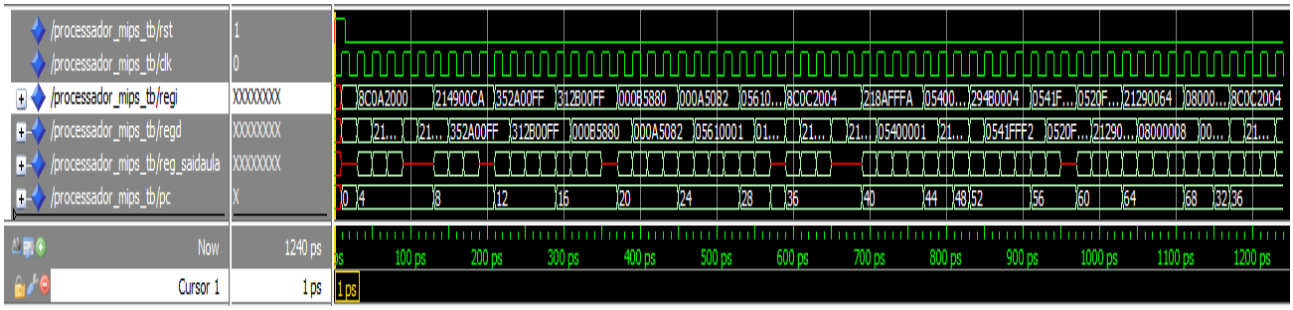
```
WIDTH=32;
DEPTH=256;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN
    000  :   8C0A2000;
    001  :   214900CA;
    002  :   352A00FF;
    003  :   312B00FF;
    004  :   000B5880;
    005  :   000A5082;
    006  :   05610001;
    007  :   012A5020;
    008  :   8C0C2004;
    009  :   218AFFFA;
    00A  :   05400001;
    00B  :   214A0008;
    00C  :   294B0004;
    00D  :   0541FFF2;
    00E  :   0520FFFD;
    00F  :   21290064;
    010  :   08000008;
    [011..07F]  :   00000000;
    080  :   0000FA00;
    081  :   00000002;
    082  :   00000005;
    083  :   00000007;
    084  :   00000009;
    [085..0FF]  :   00000000;
END;
```

4. Resultados da simulação

Para cada instrução foram avaliadas a corretude do registrador de instrução, registrador de dados, pc e saida do registrador da ULA, sendo que todos os resultados sairam como esperado.

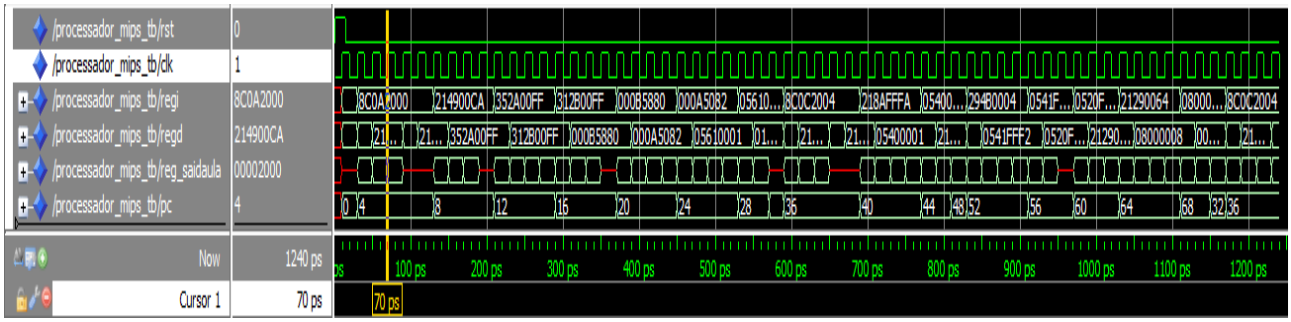
Tempo de simulação utilizado : 1240 ps.

4.1 Formas de onda do modelsim

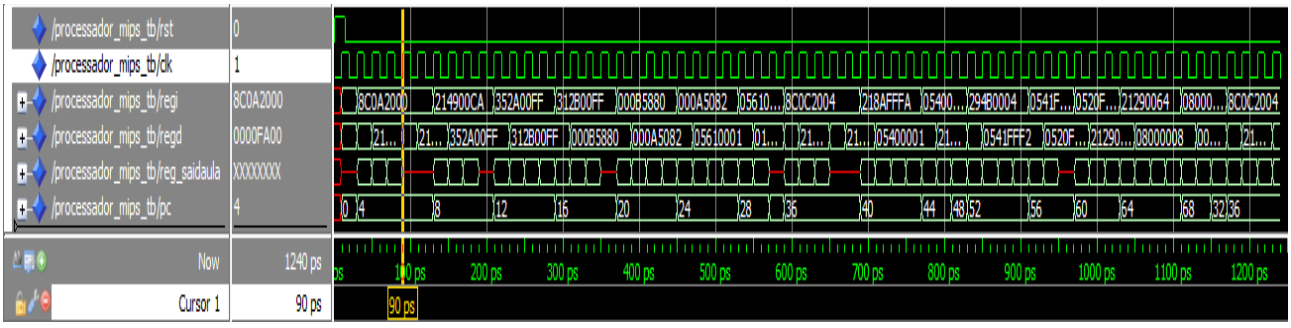


4.2 lw $t2, aux,
aux: .word 0xFA00

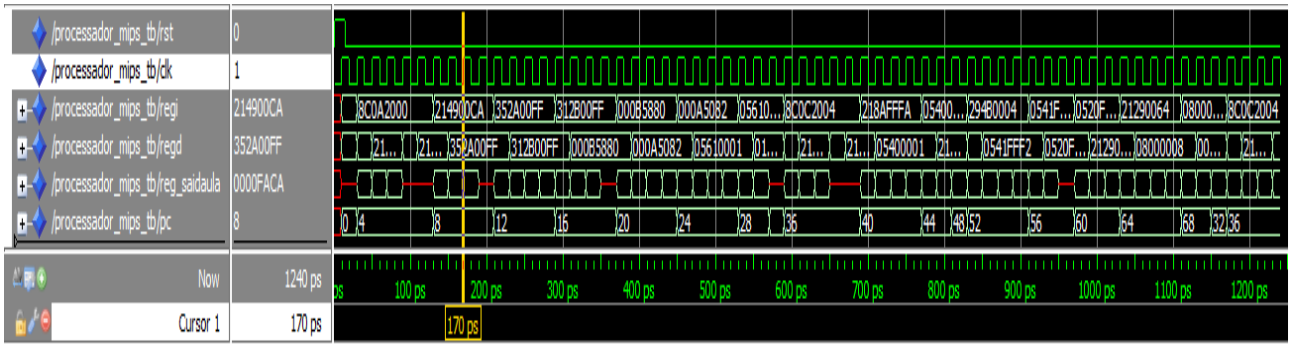
4.2.1 terceiro ciclo = definido endereço de acesso à memória de dados



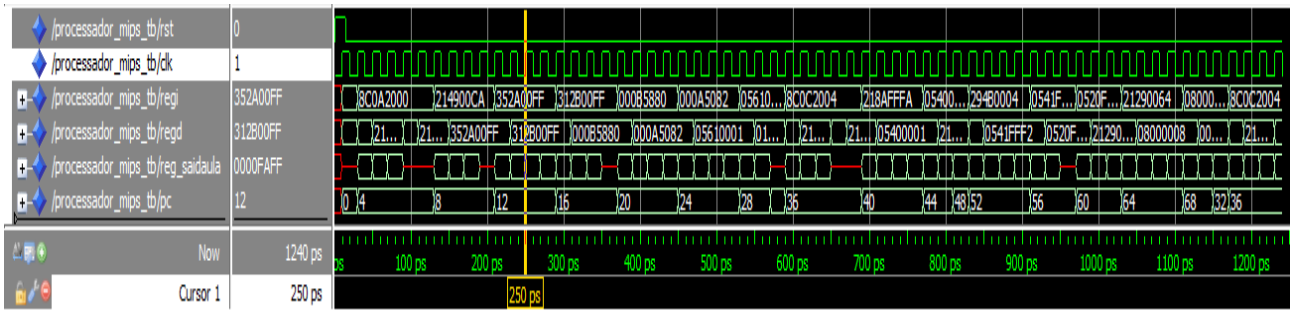
4.2.2 quarto ciclo = definido o valor no registrador de dados



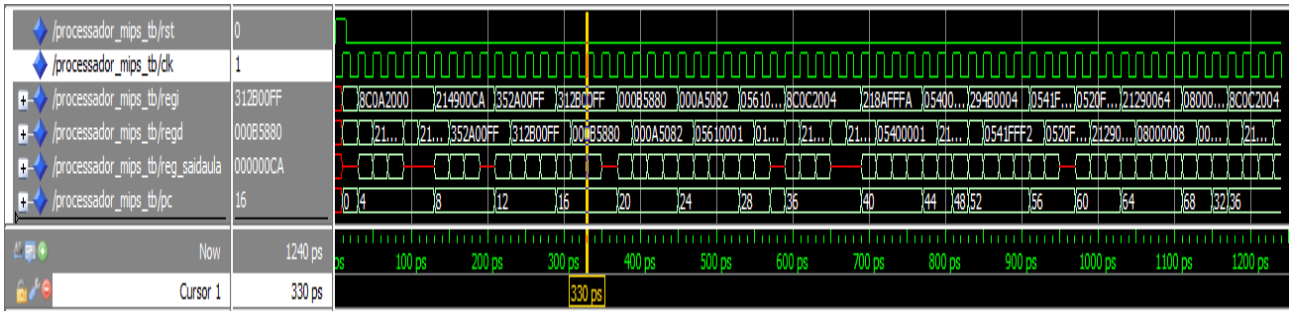
4.3 addi $t1,$t2,0x00CA



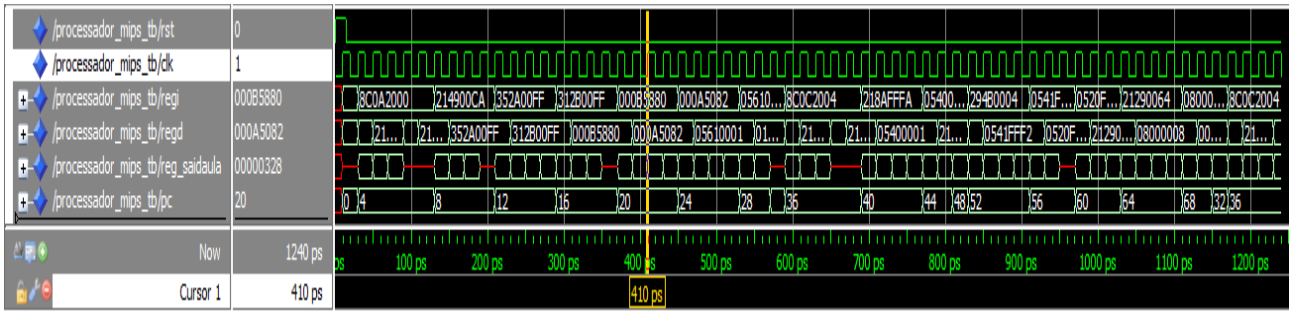
4.4 ori $t2,$t1,0x00000FF



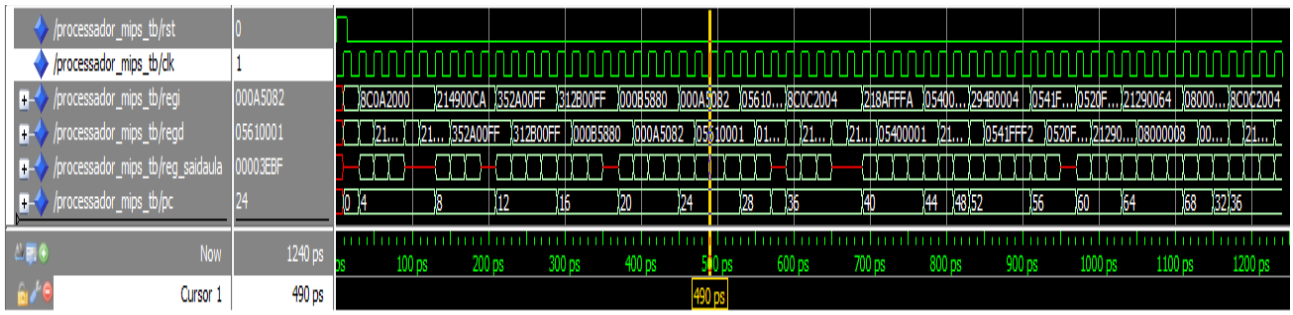
4.5 andi $t3,$t1,0x000000FF



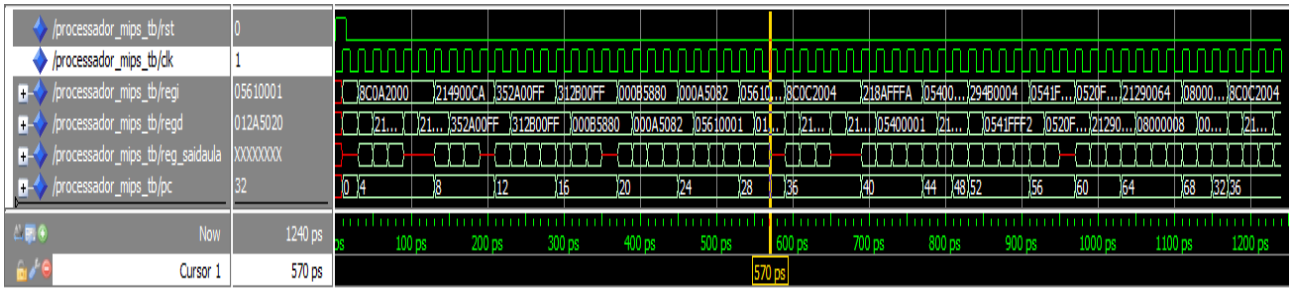
4.6 sll $t3,$t3,2



4.7 srl $t2,$t2,2

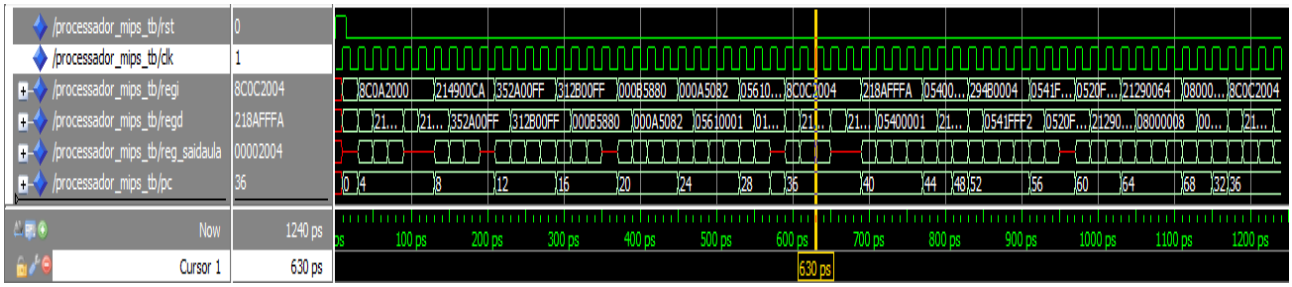


4.8 bgez $t3,LB
add $t2,$t1,$t2 #pulado

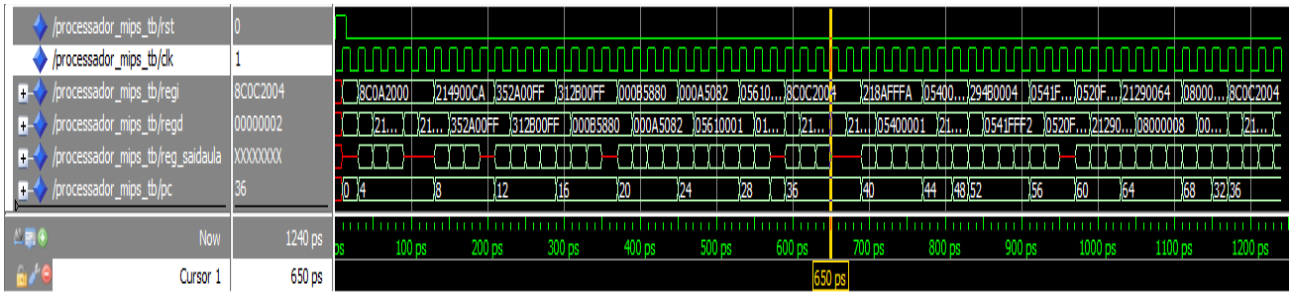


4.9 lw $t4,aux2
aux2: .word 0x000002

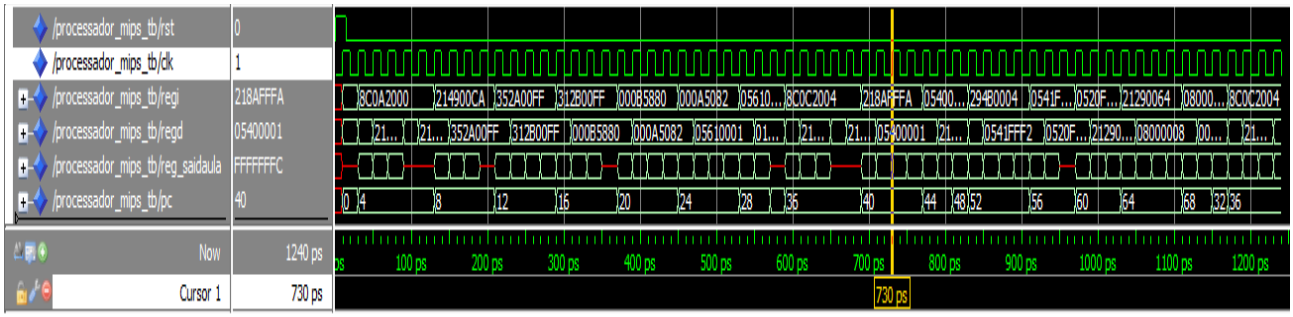
4.9.1 terceiro ciclo = definido endereço de acesso à memória de dados



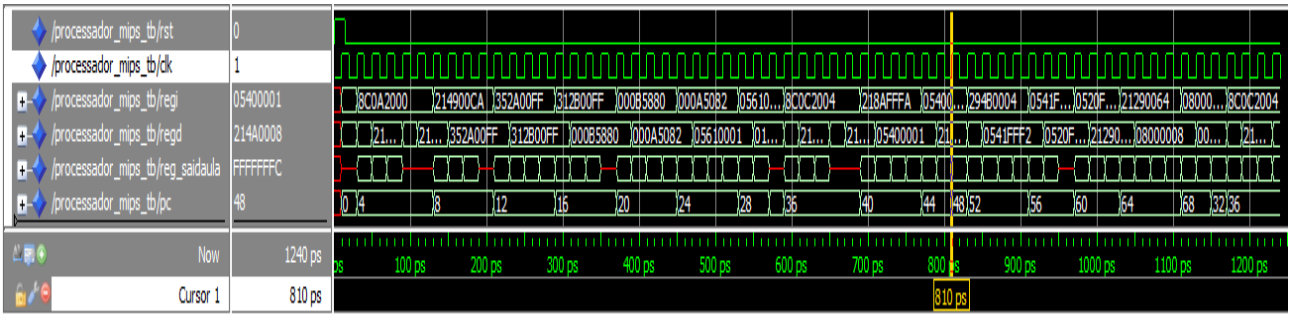
4.9.2 quarto ciclo = definido o valor no registrador de dados



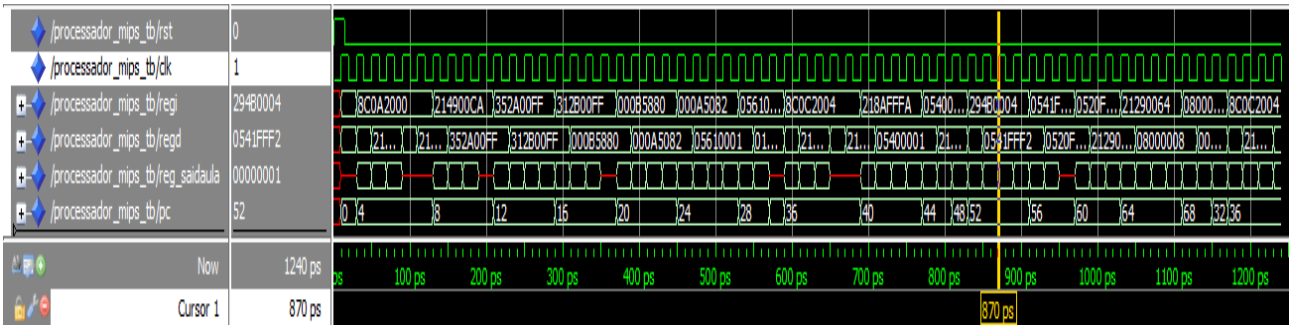
4.10 addi $t2,$t4, -6



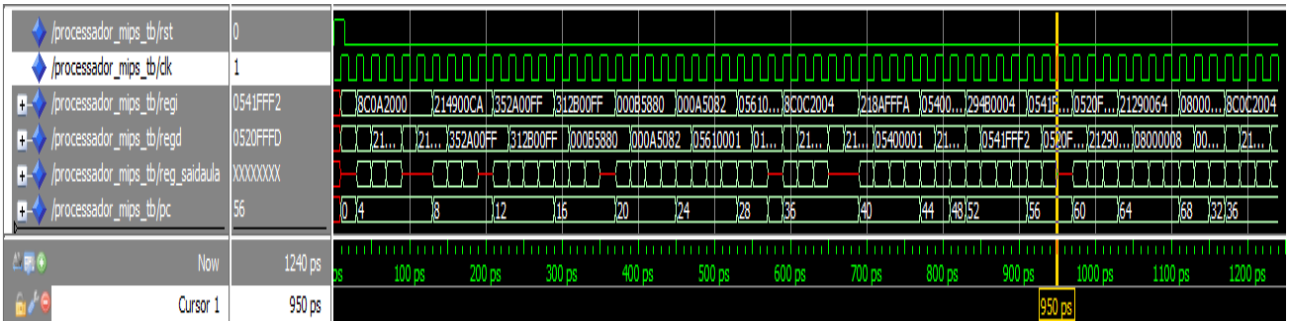
4.11 bltz $t2,LB1
addi $t2,$t2, 8 #pulado



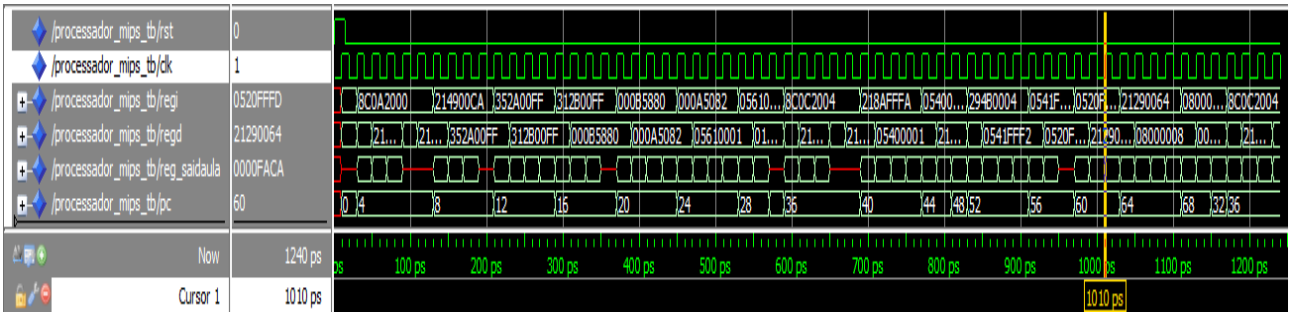
4.12 slti $t3,$t2, 4



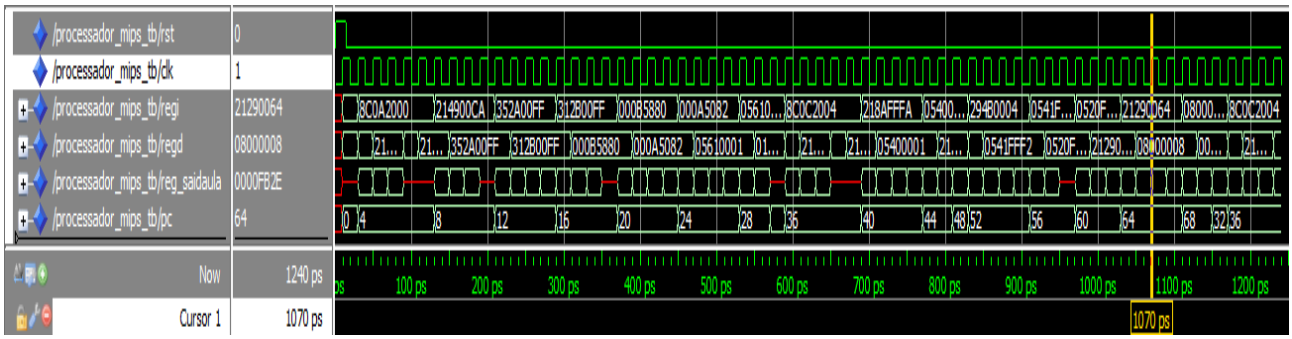
4.13 bgez $t2,LB0



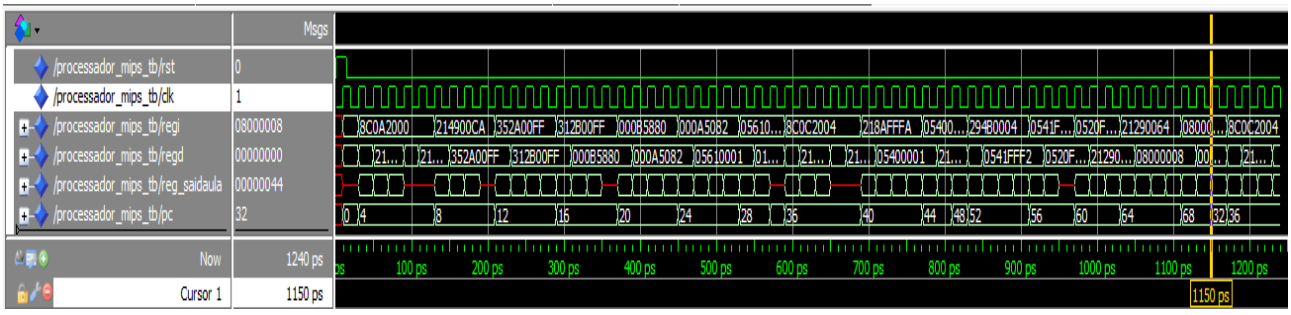
4.14 bltz $t1,LB1



4.15 addi $t1,$t1,100



4.16 j LB



5. Resultados da síntese (antes e depois)

5.1 antes

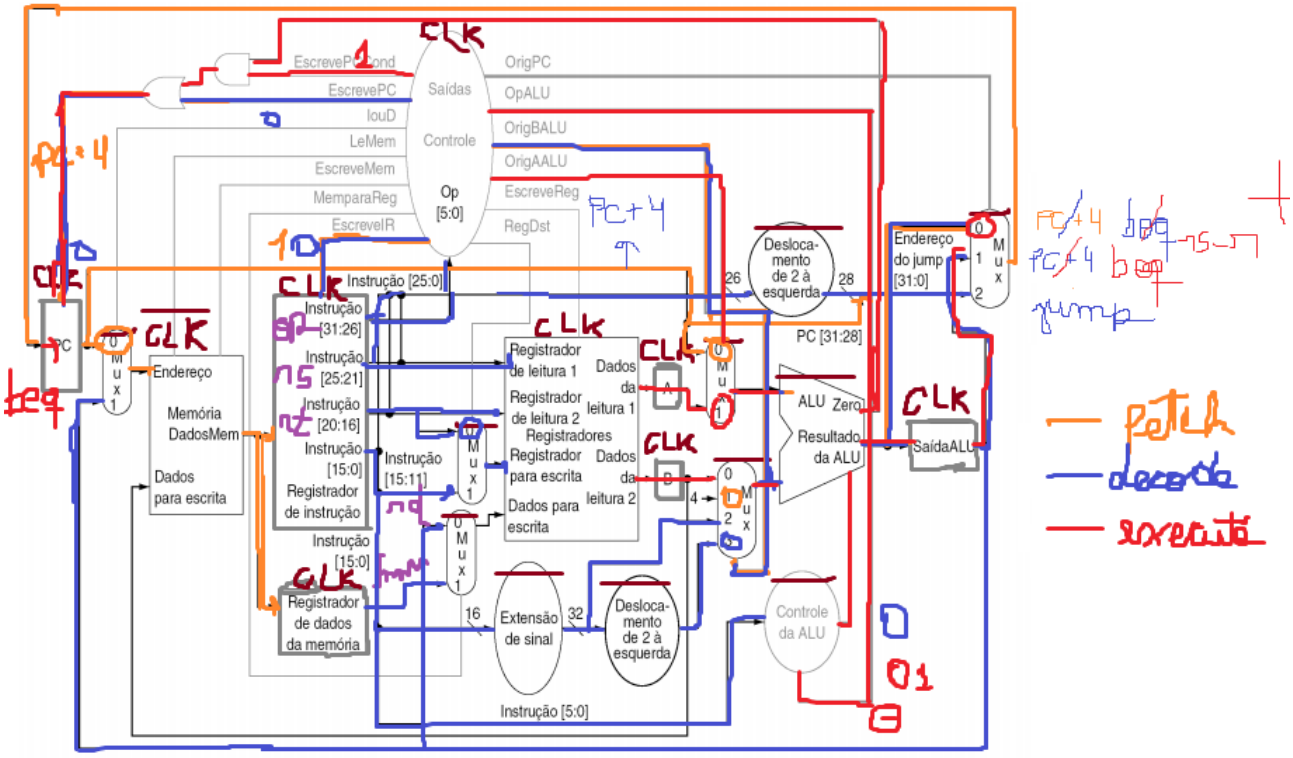
Flow Summary	
Flow Status	Successful - Tue Dec 01 22:30:39 2015
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	mips_multi
Top-level Entity Name	mips_multi
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	587
Total combinational functions	553
Dedicated logic registers	241
Total registers	241
Total pins	37
Total virtual pins	0
Total memory bits	10,240
Embedded Multiplier 9-bit elements	0
Total PLLs	0

5.2 depois

Flow Summary	
Flow Status	Successful - Tue Dec 15 12:21:47 2015
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	processador_mips
Top-level Entity Name	processador_mips
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	1,077
Total combinational functions	1,043
Dedicated logic registers	241
Total registers	241
Total pins	130
Total virtual pins	0
Total memory bits	10,240
Embedded Multiplier 9-bit elements	0
Total PLLs	0

6. Problemas e dificuldades encontradas no projeto.

6.1 Entendimento do fluxo das instruções = necessidade de teste de mesa.



6.2 Dificuldades de implantar instruções beq, bne = operação da ULA de subtração com resultados inesperados.