Universidade de Brasília
Campus Universitário Darcy Ribeiro

Disciplina: Organização e Arquitetura de Computadores
Turma: A
Prof.: Marcus Vinicius Lamar

Laboratório 3
CPU RISC-V UNICICLO

Abdullah Zaiter - 15/0089392
Daniel Bauchspiess - 15/0078901
Danielle Almeida Lima - 14/0135740
José Reinaldo da Cunha - 14/0169148
Lucas dos Santos Schiavini - 14/0150749

Brasília, 4 de junho de 2018.

- **Questão 2**

a) Primeiramente, projetamos o nosso processador uniciclo compatível com a ISA RV32I tomando como base o caminho de dados explicitado em aula, o resultado encontra-se na Figura 1. As Tabelas 1 e 2 apresentam os sinais para nossos blocos de controle. A implementação do nosso processador encontra-se na pasta Grupo4/LAB3/.

Além dos sinais de controle padrão, OrigALU, EscreveReg, LeMem, EscreveMem, OpALU e MemparaReg, acrescentamos os sinais OpBJ, OrigPC e um bit a mais para o MemparaReg. A Tabela 3 apresenta a descrição das funções desses sinais de controle.

Ademais, acrescentamos um bloco de controle de transferência com o intuito de lidar com as instruções de branch, de jal e de jalr. Este bloco recebe o campo Funct3, o bit Zero da ULA e o OrigPC, desse modo quando OrigPC é 2'b00 a próxima instrução será determinada por PC+4, caso OrigPC seja 2'b01 a instrução é um branch e para isso precisamos do Funct3 para identificar se é beq, bne, blt, bge, bltu ou bgeu e também precisamos do resultado do bit Zero da ULA para verificar se a condição do branch é verdadeira ou falsa. Caso OrigPC seja 2'b10 a instrução é um jal e caso seja 2'b11 é jalr.

Acrescentamos um bit a mais para o sinal MemparaReg com o intuito de dar suporte para as instruções auipc, jal e jalr. Excluímos o bloco shift left 1, uma vez que essa operação já é feita no bloco de gerador de imediato.

Uma das dificuldades encontradas após a simulação do projeto foi a atualização do PC para PC+4, este fato se deu pois o sinal de clock principal do processador (CLK do topDE) não era encontrado nas simulações. Por isto, o PC nunca era atualizado. Para realizar as simulações decidimos que era melhor utilizar um clock fixo e escolhemos utilizar o clock de 25MHz que é retornado do módulo CLOCK_Interface. Trocamos, então, os clocks de entrada dos módulos CPU0, MEMDATA, MEMCODE e MEMORY (memory_interface) no arquivo de TopDE.

Outra dificuldade foi para a execução da instrução ADDI, pois em instruções do tipo I o imediato ocupa a posição do campo Funct7. Deste modo, poderia ocorrer a troca de operações dependendo do imediato do addi. Corrigimos isto ao colocarmos o ADD como operação do caso default quando o Funct3 é o de ADD/SUB no controle da ULA.

Verificamos também um erro ao executar a instrução BGE, o Rars ao realizar a montagem determina um Funct3 diferente do especificado na tabela de referência do RISC-V, desse modo, tal instrução não pode ser executada em nosso processador, por decisão da equipe deixamos o Funct3 como o especificado na tabela de referência.

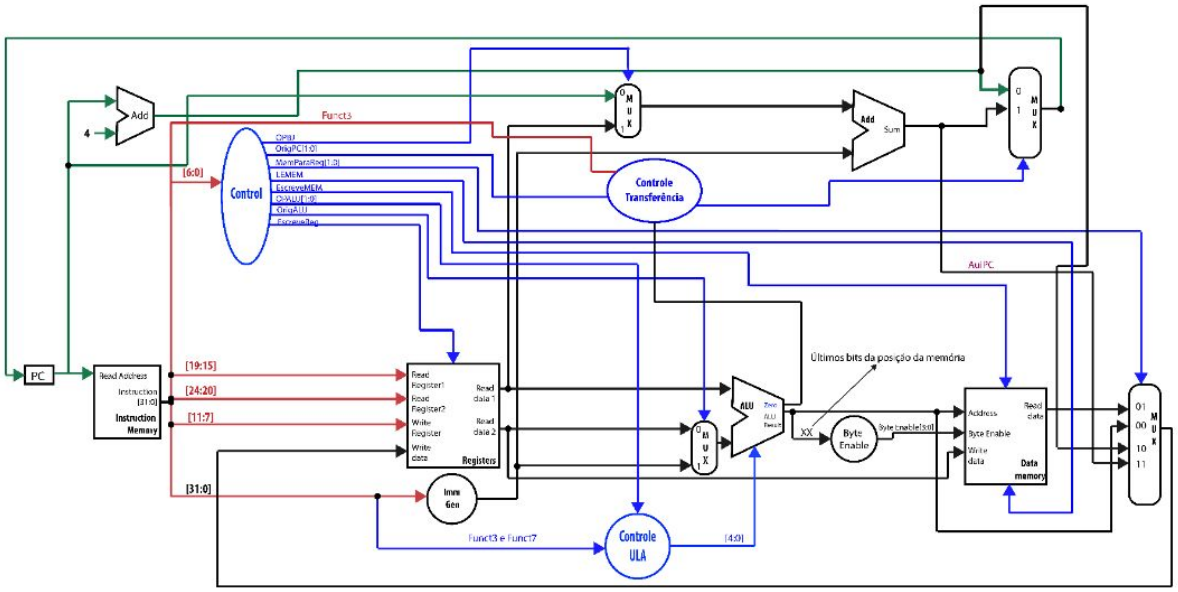


Figura 1 - Caminho de dados para as instruções listadas da ISA RV32I.

Instruções	OrigALU	Mem2Reg	RegWrite	MemRead	MemWrite	ALUOp	OrigPC	OPBJ	oCStore
ADD	0	2b00	1	0	0	2b10	1b00	X	2b00
SUB	0	2b00	1	0	0	2b10	2b00	X	2b00
AND	0	2b00	1	0	0	2b10	2b00	X	2b00
OR	0	2b00	1	0	0	2b10	2b00	X	2b00
XOR	0	2b00	1	0	0	2b10	2b00	X	2b00
SLT	0	2b00	1	0	0	2b10	2b00	X	2b00
SLTU	0	2b00	1	0	0	2b10	2b00	X	2b00
SLL	0	2b00	1	0	0	2b10	2b00	X	2b00
SRL	0	2b00	1	0	0	2b10	2b00	X	2b00
SRA	0	2b00	1	0	0	2b10	2b00	X	2b00
ADDI	1	2b00	1	0	0	2b10	2b00	X	2b00
ANDI	1	2b00	1	0	0	2b10	2b00	X	2b00
ORI	1	2b00	1	0	0	2b10	2b00	X	2b00
XORI	1	2b00	1	0	0	2b10	2b00	X	2b00
SLTI	1	2b00	1	0	0	2b10	2b00	X	2b00
SLTIU	1	2b00	1	0	0	2b10	2b00	X	2b00
SLLI	1	2b00	1	0	0	2b10	2b00	X	2b00
SRLI	1	2b00	1	0	0	2b10	2b00	X	2b00
SRAI	1	2b00	1	0	0	2b10	2b00	X	2b00

AUIPC	X	2b11	1	0	0	X	2b00	0	2b00
LUI	1	2b00	1	0	0	2b11	2b00	0	2b00
BEQ	0	X	0	0	0	2b01	2b01	0	2b00
BNE	0	X	0	0	0	2b01	2b01	0	2b00
BGE	0	X	0	0	0	2b01	2b01	0	2b00
BGEU	0	X	0	0	0	2b01	2b01	0	2b00
BLT	0	X	0	0	0	2b01	2b01	0	2b00
BLTU	0	X	0	0	0	2b01	2b01	0	2b00
JAL	X	2b10	1	0	0	X	2b10	0	2b00
JALR	X	2b10	1	0	0	X	2b11	1	2b00
LB	1	2b01	1	1	0	2b00	2b00	0	2b00
LBU	1	2b01	1	1	0	2b00	2b00	0	2b00
LH	1	2b01	1	1	0	2b00	2b00	0	2b00
LHU	1	2b01	1	1	0	2b00	2b00	0	2b00
LW	1	2b01	1	1	0	2b00	2b00	0	2b00
SB	1	X	0	0	1	2b00	2b00	0	2b10
SH	1	X	0	0	1	2b00	2b00	0	2b10
SW	1	X	0	0	1	2b00	2b00	0	2b10
MUL	0	2b00	1	0	0	2b10	2b00	0	2b00
MULH	0	2b00	1	0	0	2b10	2b00	0	2b00
MULHU	0	2b00	1	0	0	2b10	2b00	0	2b00
MULHSU	0	2b00	1	0	0	2b10	2b00	0	2b00
DIV	0	2b00	1	0	0	2b10	2b00	0	2b00
DIVU	0	2b00	1	0	0	2b10	2b00	0	2b00
REM	0	2b00	1	0	0	2b10	2b00	0	2b00
REMU	0	2b00	1	0	0	2b10	2b00	0	2b00

Tabela 1 - Tabela verdade do bloco de controle.

Instrução	zero	OrigPC	oCtrlTransf
ADD	X	2b00	0
SUB	X	2b00	0
AND	X	2b00	0
OR	X	2b00	0
XOR	X	2b00	0
SLT	X	2b00	0
SLTU	X	2b00	0

SLL	X	2b00	0
SRL	X	2b00	0
SRA	X	2b00	0
ADDI	X	2b00	0
ANDI	X	2b00	0
ORI	X	2b00	0
XORI	X	2b00	0
SLTI	X	2b00	0
SLTIU	X	2b00	0
SLLI	X	2b00	0
SRLI	X	2b00	0
SRAI	X	2b00	0
AUIPC	X	2b00	0
LUI	X	2b00	0
BEQ	1	2b01	1
BNE	0	2b01	1
BGE	1	2b01	1
BGEU	1	2b01	1
BLT	0	2b01	1
BLTU	0	2b01	1
JAL	X	2b10	1
JALR	X	2b11	1
LB	X	2b00	0
LBU	X	2b00	0
LH	X	2b00	0
LHU	X	2b00	0
LW	X	2b00	0
SB	X	2b00	0
SH	X	2b00	0
SW	X	2b00	0

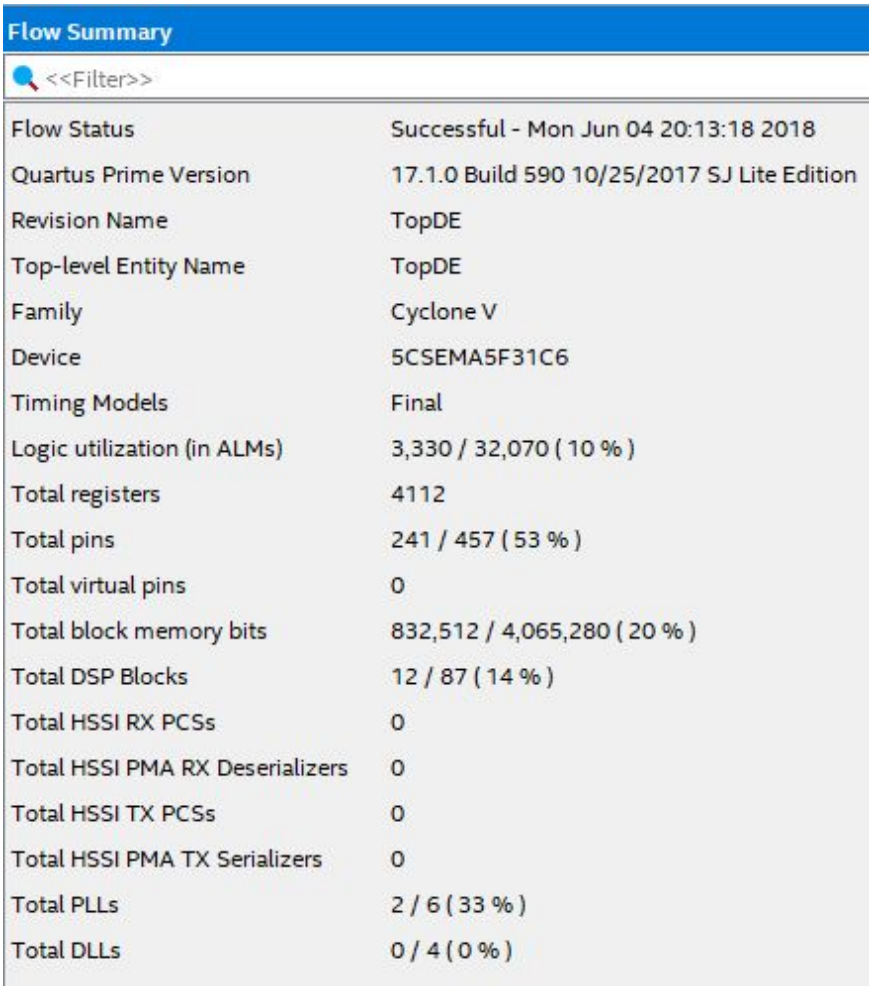
Tabela 2 - Tabela verdade do controle de transferência.

Sinal de Controle	Descrição
OpBJ	Determina se a instrução é um jalr
OrigPC	Determina se a instrução é um branch, jal ou jalr
MemparaReg	Determina o que vai ser escrito no registrador

EscreveMem	Determina se o dado vai ser escrito na memória
LeMem	Determina se o dado vai ser lido da memória
OpALU	Identifica a operação da ULA
OrigALU	Determina se o segundo argumento da ULA vem do gerador de imediatos ou do banco de registradores
EscreveReg	Determina se o dado vai ser escrito no banco de registradores

Tabela 3 - Descrição dos sinais de controle.

b) A Figura 2 apresenta os resultados encontrados para os requisitos físicos da implementação do nosso processador uniciclo compatível com a ISA RV32I.



Flow Summary

<<Filter>>

Flow Status	Successful - Mon Jun 04 20:13:18 2018
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	TopDE
Top-level Entity Name	TopDE
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	3,330 / 32,070 (10 %)
Total registers	4112
Total pins	241 / 457 (53 %)
Total virtual pins	0
Total block memory bits	832,512 / 4,065,280 (20 %)
Total DSP Blocks	12 / 87 (14 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	2 / 6 (33 %)
Total DLLs	0 / 4 (0 %)

Figura 2: Requisitos Físicos

- Número de Elementos Lógicos (ALMs): 3330/32070 (19%)
- Número de Registradores: 4112
- Quantidade de bits de memória: 832.512/4065280 (20%)

- Número de blocos DSP usados: 12/87 (21%)

c) Com o auxílio do TimeQuest e um clock de 50 MHz, encontramos:

- Caminho de maior atraso, tpd: 18,914 de SW[9] até VGA_B[7]
- Se algum requerimento não foi atendido: Não, todos foram atendidos
- th: 0,762
- tco: 5,883
- slack setup: 11,031
- slack hold: 0,439
- slack minimum pulse width: 15,284
- Frequência máxima utilizável: 88,72 MHz

d) A implementação do processador uniciclo da ISA RV32IM foi feita utilizando por base a ISA RV32I apresentada anteriormente, deve-se somente definir qual a ISA desejada no arquivo topDE. As alterações necessárias, para a adição das operações do módulo de multiplicação, foram nos blocos de controle da ULA, ULA e parâmetros. Como as instruções listadas possuem o mesmo opcode e o mesmo funct3 de algumas instruções listadas anteriormente, desse modo as diferenciamos pelo funct7, a Figura 3 mostra como poderia ser implementado para um dos casos.

Instrução	OrigALU	MemparaReg	EscreveReg	LeMem	EscreveMem	OpALU	OrigPC	OPBJ
MUL	0	2b00	1	0	0	2b10	2b00	0
MULH	0	2b00	1	0	0	2b10	2b00	0
MULHU	0	2b00	1	0	0	2b10	2b00	0
MULHSU	0	2b00	1	0	0	2b10	2b00	0
DIV	0	2b00	1	0	0	2b10	2b00	0
DIVU	0	2b00	1	0	0	2b10	2b00	0
REM	0	2b00	1	0	0	2b10	2b00	0
REMU	0	2b00	1	0	0	2b10	2b00	0

Tabela 4 - Tabela verdade do bloco de controle.

Instrução	zero	OrigPC	oCtrlTransf
MUL	X	2b00	0
MULH	X	2b00	0
MULHU	X	2b00	0
MULHSU	X	2b00	0
DIV	X	2b00	0

DIVU	X	2b00	0
REM	X	2b00	0
REMU	X	2b00	0

Tabela 5 - Tabela verdade do controle de transferência.

```
2'b10:
      case (iFunct3)
          F3_ADD:
              case (iFunct7)
                  F7_ADD:
                     oControlSignal  = OPADD;
                  F7_MUL:
                     oControlSignal  = OPMUL;
                  default:
                     oControlSignal  = 5'b00000;
              endcase
```

Figura 3 - Exemplo de um trecho de código do arquivo ALUControl.v para dar suporte a instrução mul.

e) A Figura 4 apresenta os requisitos físicos da implementação do processador RV32IM.

Flow Summary

<<Filter>>

Flow Status	Successful - Mon Jun 04 20:30:08 2018
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	TopDE
Top-level Entity Name	TopDE
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	6,047 / 32,070 (19 %)
Total registers	4161
Total pins	241 / 457 (53 %)
Total virtual pins	0
Total block memory bits	832,512 / 4,065,280 (20 %)
Total DSP Blocks	18 / 87 (21 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	2 / 6 (33 %)
Total DLLs	0 / 4 (0 %)

Figura 3: Requisitos.

- Número de Elementos Lógicos (ALMs): 6047
- Número de Registradores: 4161
- Quantidade de bits de memória: 832.512
- Número de blocos DSP usados: 18

f) Com o auxílio do TimeQuest e um clock de 50 MHz, encontramos:

- Caminho de maior atraso, tpd: 18,988 de SW[9] a VGA_B[6]
- Se algum requerimento não foi atendido: Não, todos foram atendidos
- th: 2,893
- tco: 5,883
- slack setup: 10,717
- slack hold: 0,139
- slack minimum pulse width: 15,270
- Frequência máxima utilizável: 84,04 MHz

Questão 3

De forma a analisar o processador, foi necessário rotear sinais do CLK para sinais de Clock25 em módulos no arquivo TopDE.v no diretório do projeto, assim como setar os sinais de CLKAutoFast e CLKSelectAuto em ON.

O comportamento do processador foi observado por meio dos sinais: Instrução, BR_Escrita, BR_Leitura1, BR_Leitura2, PC. Dessa forma foi possível verificar o código

hexadecimal das instruções, o valor de PC, e os dados lidos dos registradores ou imediatos, assim como o resultado das operações em BR_Escrita (valor de escrita no banco de registradores).

Em cada forma de onda, foi comparado o resultado com o esperado no Rars. Os arquivos .mif foram colocados de forma a facilitar a verificação de cada tipo de instrução.

- Instruções do tipo R:

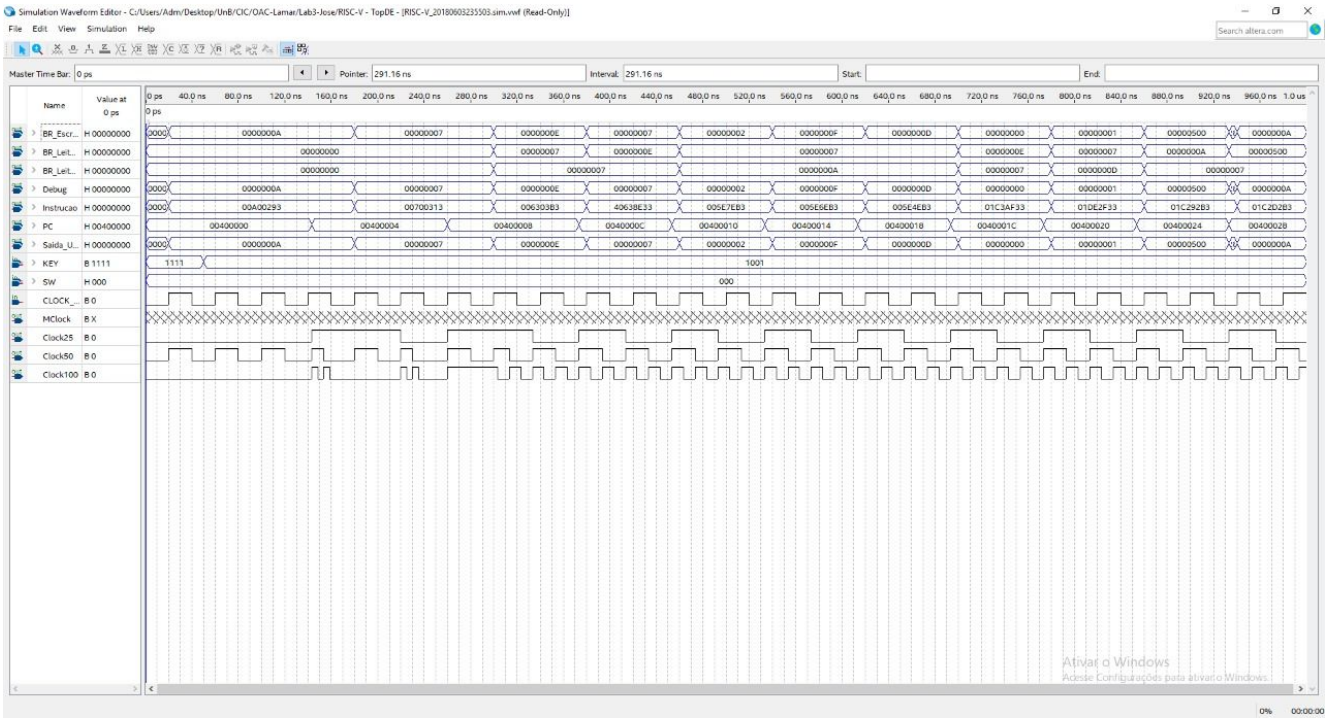


Figura 4: Resultado da simulação de instruções tipo R.

```
DEPTH = 4096;
WIDTH = 32;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT
BEGIN
00000000 : 00a00293;   % 5:     addi    t0, zero, 0xA       # t0 <- 0xA = 4'b1010 %
00000001 : 00700313;   % 6:     addi    t1, zero, 7     # t1 <- 7 %
00000002 : 006303b3;   % 7:     add t2, t1, t1      # t2 <- 14 %
00000003 : 40638e33;   % 8:     sub t3, t2, t1      # t3 <- 7 %
00000004 : 005e7eb3;   % 9:     and t4, t3, t0      # t4 <- 2 %
00000005 : 005e6eb3;   % 10:    or  t4, t3, t0      # t4 <- 0xF %
00000006 : 005e4eb3;   % 11:    xor t4, t3, t0      # t4 <- 0xD %
00000007 : 01c3af33;   % 12:    slt t5, t2, t3      # t5 <- 1'b0 %
00000008 : 01de2f33;   % 13:    slt t5, t3, t4      # t5 <- 1'b1 %
00000009 : 01c292b3;   % 15:    sll t0, t0, t3      # t0 <- t0<<t3 = 0x00000500 %
0000000a : 01c2d2b3;   % 16:    srl t0, t0, t3      # t0 <- 0xA %
END;
```

Figura 5 - Código para as instruções do tipo R.

- Instruções do tipo I:

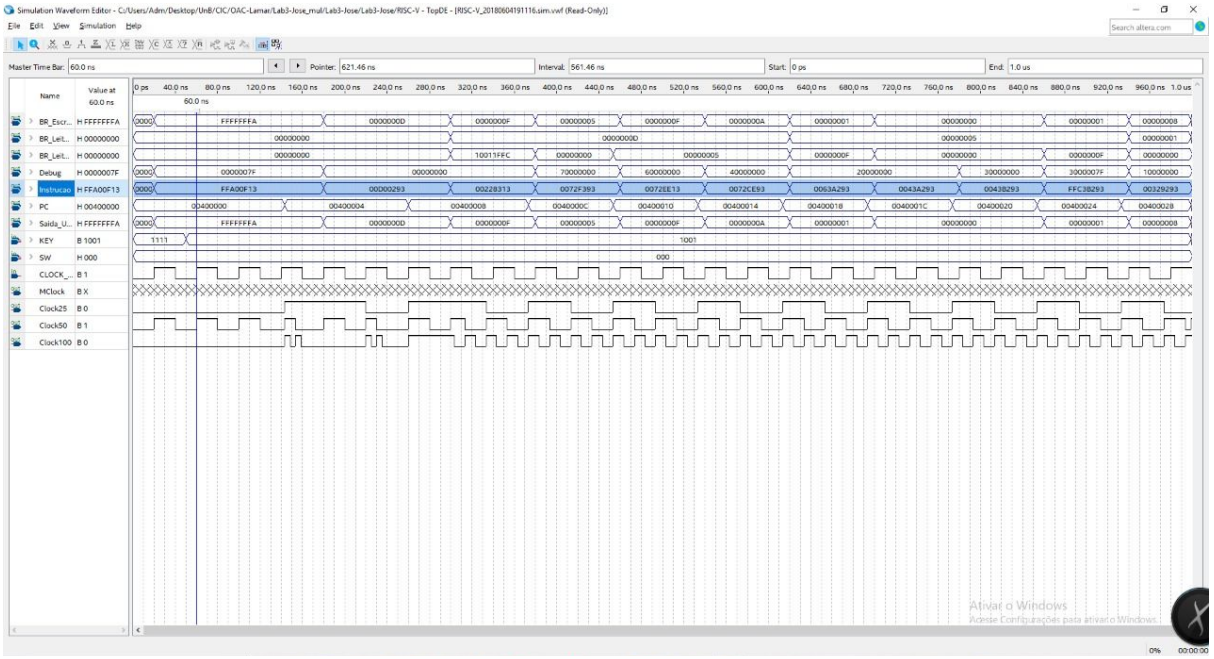


Figura 6 - Instruções do tipo I.

```
DEPTH = 4096;
WIDTH = 32;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT
BEGIN
00000000 : ffa00f13;   % 5:     addi    t5, zero, -6 %
00000001 : 00d00293;   % 6:     addi    t0, zero, 0xD   # t0 <- 0xD %
00000002 : 00228313;   % 7:     addi    t1, t0, 2   # t1 <- 0xF %
00000003 : 0072f393;   % 8:     andi    t2, t0, 0x7 # t2 <- 5 %
00000004 : 0072ee13;   % 9:     ori t3, t0, 0x7 # t1 <- 0xF %
00000005 : 0072ce93;   % 10:    xori    t4, t0, 0x7 # t4 <- 0xA %
00000006 : 0063a293;   % 11:    slti    t0, t2, 0x6 # t0 <- 1 %
00000007 : 0043a293;   % 12:    slti    t0, t2, 0x4 # t0 <- 0 %
00000008 : 0043b293;   % 13:    sltiu   t0, t2, 4   # t0 <- 0 %
00000009 : ffc3b293;   % 14:    sltiu   t0, t2, -4  # t0 <- 1 %
0000000a : 00329293;   % 15:    slli    t0, t0, 3   # t0 <- 0x8 %
0000000b : 0022d293;   % 16:    srli    t0, t0, 2   # t0 <- 0x2 %
0000000c : 401f5293;   % 17:    srai    t0, t5, 1   # t0 <- (-3) %
END;
```

Figura 7 - Código das instruções do tipo I.

- Instruções de JUMP:

Figura 8 - Instruções de jump.

```
1  DEPTH = 4096;
2  WIDTH = 32;
3  ADDRESS_RADIX = HEX;
4  DATA_RADIX = HEX;
5  CONTENT
6  BEGIN
7  00000000 : 018000ef;   % 2:     jal ra, pula_man %
8  00000001 : 01c0006f;   % 3:     jal zero, eh_tetra %
9  00000002 : 00a00293;   % 4:     addi    t0, zero, 10 %
10 00000003 : 005282b3;   % 5:     add t0, t0, t0 %
11 00000004 : 005282b3;   % 6:     add t0, t0, t0 %
12 00000005 : 005282b3;   % 7:     add t0, t0, t0 %
13 00000006 : 00008067;   % 8: pula_man:   jr  ra, 0 %
14 00000007 : 00f00293;   % 9:         addi    t0, zero, 15 %
15 END;
16
```

Figura 9 - Código das instruções de jump.

- Instruções MUL DIV, REM:

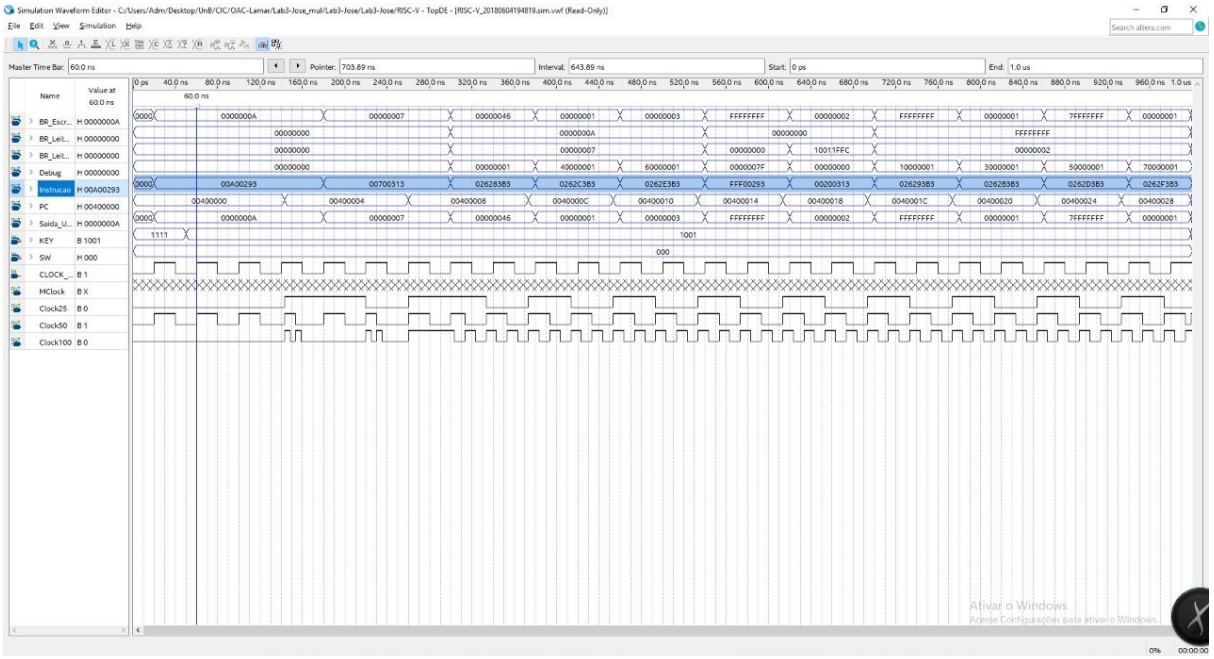


Figura 11 - Instruções de multiplicação, divisão e resto.

```
1   DEPTH = 4096;
2   WIDTH = 32;
3   ADDRESS_RADIX = HEX;
4   DATA_RADIX = HEX;
5   CONTENT
6   BEGIN
7   00000000 : 00a00293;   % 2:     li  t0, 10 %
8   00000001 : 00700313;   % 3:     li  t1, 7 %
9   00000002 : 026283b3;   % 4:     mul t2, t0, t1  # t2 <- 70 %
10  00000003 : 0262c3b3;   % 5:     div t2, t0, t1  # t2 <- 1 %
11  00000004 : 0262e3b3;   % 6:     rem t2, t0, t1  # t2 <- 3 %
12  00000005 : fff00293;   % 7:     li  t0, 0xFFFFFFFF %
13  00000006 : 00200313;   % 8:     li  t1, 0x00000002 %
14  00000007 : 026293b3;   % 9:     mulh    t2, t0, t1  # t2 <- 0xFFFFFFFF %
15  00000008 : 0262b3b3;   % 10:    mulhu   t2, t0, t1  # t2 <- 0x00000001 %
16  00000009 : 0262d3b3;   % 12:    divu    t2, t0, t1  # t2 <- 0x7FFFFFFF %
17  0000000a : 0262f3b3;   % 13:    remu    t2, t0, t1  # t2 <- 0x0x000001 %
18  END;
```

Figura 12 - Código das instruções de multiplicação, divisão e resto.

- Instruções BRANCH:

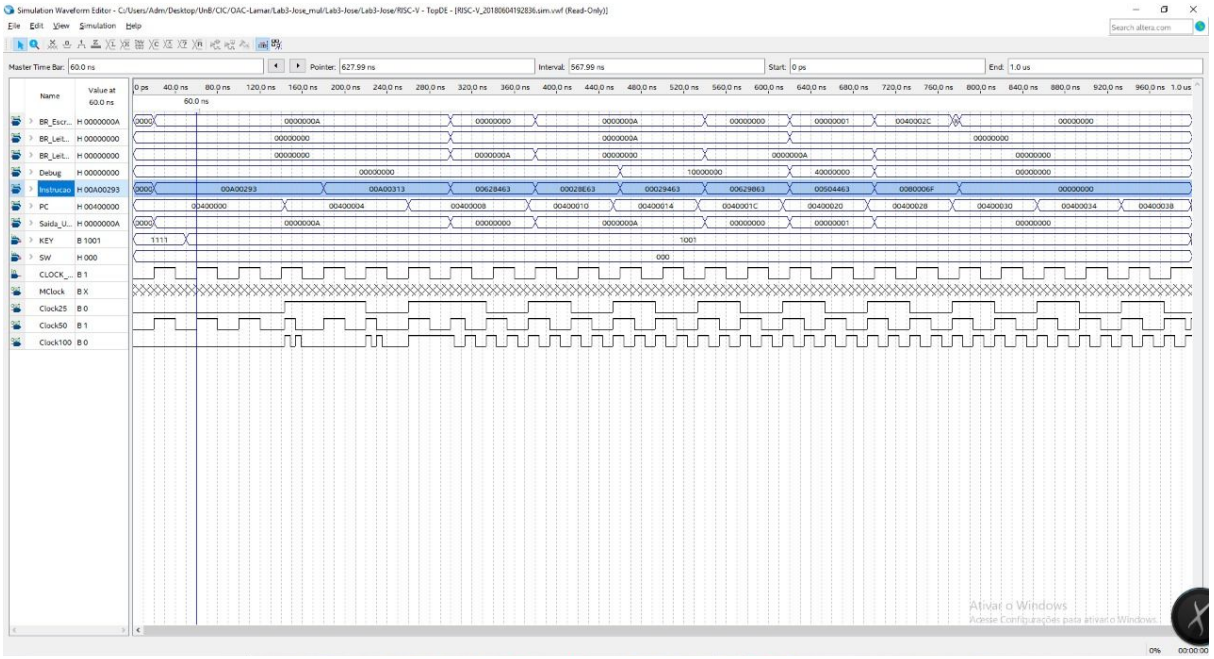


Figura 13: Instruções de Branch.

```
1   DEPTH = 4096;
2   WIDTH = 32;
3   ADDRESS_RADIX = HEX;
4   DATA_RADIX = HEX;
5   CONTENT
6   BEGIN
7   00000000 : 00a00293;   % 2: li  t0, 10 %
8   00000001 : 00a00313;   % 3: li  t1, 10 %
9   00000002 : 00628463;   % 4: beq t0, t1, pulo1 %
10  00000003 : 0200006f;   % 5: jal zero, ERRO %
11  00000004 : 00028e63;   % 6: pulo1:  beq t0, zero, ERRO %
12  00000005 : 00029463;   % 7: bne t0, zero, pulo2 %
13  00000006 : 0140006f;   % 8: jal zero, ERRO %
14  00000007 : 00629863;   % 9: pulo2:  bne t0, t1, ERRO %
15  00000008 : 00504463;   % 10: blt    zero, t0, pulo3 %
16  00000009 : 0080006f;   % 11: jal    zero, ERRO %
17  0000000a : 0080006f;   % 12: pulo3: jal zero, FIM %
18  0000000b : 005282b3;   % 14: ERRO:  add t0, t0, t0 %
19  END;
```

Figura 14 - Código das instruções de Branch.

- Instruções AUIPC, LUI, LOAD, STORE:

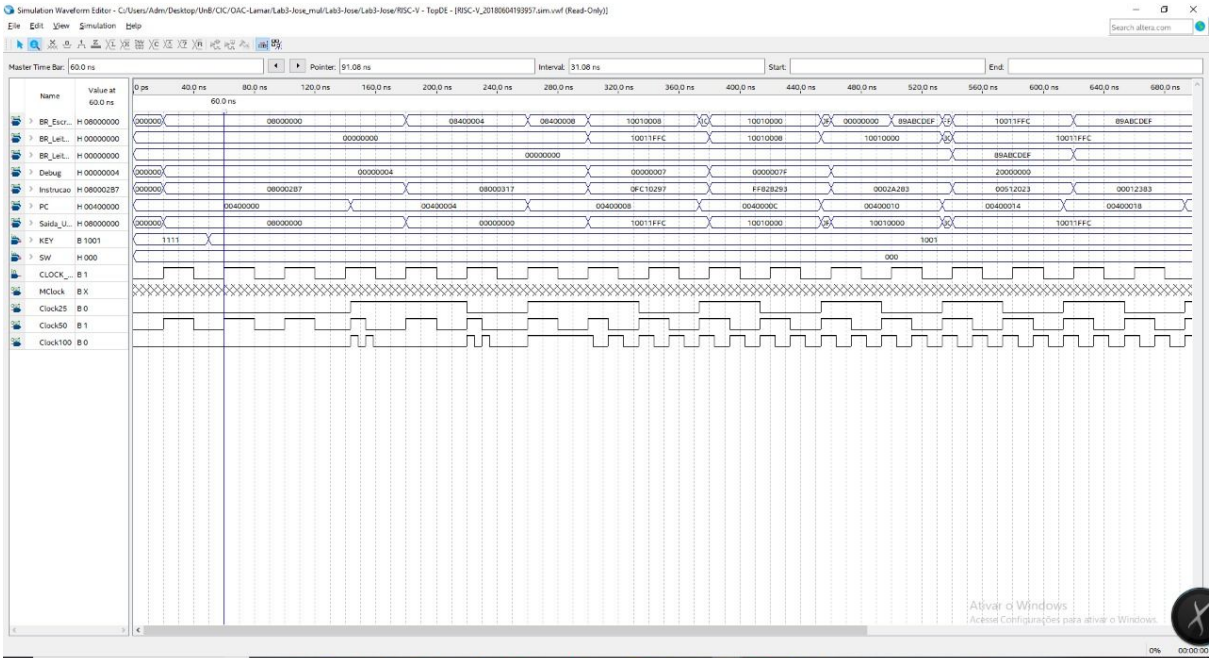


Figura 15 - Instruções de Auipc, Lui, Load e Store.

```
DEPTH = 4096;
WIDTH = 32;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT
BEGIN
00000000 : 080002b7;   % 4:     lui t0, 0x8000  # t0 <- 0x08000000 %
00000001 : 08000317;   % 5:     auipc   t1, 0x8000  # t1 <- 0x84000004 %
00000002 : 0fc10297;   % 7:     la  t0, const %
00000003 : ff828293;   % 7:  %
00000004 : 0002a283;   % 8:     lw  t0, 0(t0) %
00000005 : 00512023;   % 10:    sw  t0, 0(sp) %
00000006 : 00012383;   % 11:    lw  t2, 0(sp) %
END;

```

Figura 16 - Código das instruções de Branch .Text

Como pôde ser visto, nos códigos acima, do lado de cada linha há o número da linha a ser executada, seguida do código de máquina da instrução, e a instrução em seguida. Observamos no campo instrução em cada ciclo do clock de 25MHz o que está sendo executado. Todos os códigos .mif estão no diretório (Grupo4/Lab3/Arquivos_mif), no entanto não colocamos os .data neste relatório devido ao seu tamanho.

Foi analisada cada instrução nas diversas formas de onda, e obtivemos formas de onda coerentes com o resultado esperado. Por fim, a partir das análises com os testbenchs e as formas de onda, verificamos que o nosso processador dá suporte para todas as instruções listadas no roteiro do laboratório, exceto lb, lh, sb e sh.

Dentro do diretório contendo os arquivos .mif há também um arquivo para o testbench de grande parte das instruções da ISA RV32IM com uma rotina de ERRO onde o endereço da função que der erro vai ser armazenada no registrador t0. Este testbench não foi realizado pois tivemos dificuldades em alterar o tempo de simulação do waveform no quartus. Os arquivos para a simulação do testbench com rotina de ERRO são: Testbench_data.mif e Testbench_text.mif.