Laboratório de Arquitetura e Organização de Computadores II
Prática III

Nome do Aluno: Gabriel Alves Barbosa e Gabriel Luís Silva Pereira

Obs: Editores de texto reduzem a qualidade e nitidez das imagens. Assim, caso esteja difícil a leitura e interpretação de alguma imagem, há uma cópia de todas elas na pasta “imagens das simulações”, neste mesmo arquivo zip onde se encontra este relatório.

1 - Introdução

O algoritmo de Tomasulo é um dos algoritmos mais importantes quando se trata de algoritmos de hardware, tendo sido desenvolvido por Robert Tomasulo. Sua principal característica é permitir a execução de instruções fora de ordem, de modo a reduzir a quantidade de stalls do processador.

Neste trabalho, visamos implementar uma versão do algoritmo de Tomasulo, com as suas principais características, padrões e funcionalidades. Para isso, buscamos dividir o algoritmo em módulos individuais, de modo a permitir uma maior organização, além de permitir que o código pudesse ser testado de maneira mais eficiente.

Como mencionamos mais para frente, nosso projeto implementa as instruções de soma e subtração, e faz o controle de algumas hazards, de modo a permitir com que as instruções possam ser executadas de maneira mais adequada.

2 - Projeto

Module count:

```
1   module count(clk, saida); // Recebe um clock
2      input clk;
3      output reg [20:0]saida;
4      initial
5         saida = 0; //Seta a saída inicialmente como zero
6
7      always@(posedge clk) begin
8            saida = saida + 1; //Acrescemta 1 no valor que estava na saída
9      end
10  endmodule
```

Esse módulo é responsável por realizar o somatório unitário no sinal “saida” sempre que ocorrer uma borda de subida do clock (clk).

Module FilaInstrucoes:

```
module memoriaInstrucao(origem,saida);
   input [7:0]origem;
   output [11:0] saida;
   reg [11:0]MEM[127:0];   initial
   begin//Instruções a serem executadas
      MEM[0] = 12'b111111111111;  //Vai pular esta instrução (sempre pula a primeira)
      MEM[1] = 12'b001010000000; //R2 = R0 + R1 (2)
      MEM[2] = 12'b000100010001; //R4 = R2 - R0 (1)
      MEM[3] = 12'b101100011000; //R4 = R3 + R5 (3)
      MEM[4] = 12'b100101100000; //R5 = R4 + R4 (6)
      MEM[5] = 12'b111111111111;  //Vai pular esta instrução (sempre pula a primeira)
   end
   assign saida = MEM[origem];
endmodule
```

```
module PC(clk,reset,parada,pc);
  input clk,reset,parada;
  output reg [7:0]pc;
  initial
  begin
    pc <= 8'b00000000; //Valor incial de pc tem que ser zero
  end
  always @(posedge clk)
  begin
    if(reset) //Se reset for ativo, pc deve ser zerado novamente
      pc <= 8'b00000000;
    else if(!parada) //Se não estiver ocorrendo uma parada, pc deve ser acrescido em um
      pc <= pc + 1;
  end
endmodule
```

```
module FilaInstrucoes(clk,reset,inst,addFullControl);
//Modulo de controle das instrucoes a serem despachadas
   parameter ADD = 3'b000;
   parameter SUB = 3'b001;

   input [2:0]addFullControl;
   input clk,reset;
   output [11:0]inst;
   wire [7:0]PC;
   reg parada;

   wire [2:0]I;
   assign I = inst[2:0];
   //Sempre que o clock se alterar
   always @(clk) begin
      parada = 0;
      //Verificando se é uma soma ou subtração
      if(addFullControl == 3 && (I == ADD || I == SUB))begin
         parada = 1;
      end
   end
   PC pc(clk,reset,parada,PC);
   memoriaInstrucao memoriaInst(PC,inst);
endmodule
```

O módulo FilaInstrucoes, verifica primeiramente se a instrução recebida é do tipo ADD ou SUB e se a estação reserva de soma/subtração está totalmente preenchida (isso pode ser percebido quando o sinal “addFullControl” for igual a 3). Caso a estação esteja cheia, um sinal de “parada” é enviado para que não pegue a próxima instrução e caso contrário, a próxima instrução da “memoriaInstrucao” será reservada na estação.

O module PC é responsável por realizar o somatório unitário

Module registrador:

```
3   module registrador1(clk,InputData,DataControl,InputLabel,LabelControl,OutputLabel,OutputData);
4      input [8:0] InputData,InputLabel;
5      input DataControl,LabelControl,clk;
6      output reg[8:0] OutputLabel,OutputData;
7
8      initial begin
9         OutputLabel = 9'b111111111;
10        OutputData = 9'b000000001;
11     end
12     always@(posedge DataControl)begin
13        if(DataControl) begin
14           OutputData = InputData;
15        end
16     end
17     always@(posedge LabelControl)begin
18        if(LabelControl) begin
19              OutputLabel = InputLabel;
20        end
21     end
22
23  endmodule
```

Este módulo representa os nossos registradores. Nele, é o valor InputData, que representa o dado de entrada, InputLabel, que representa o nome do registrador (reg1, reg2, etc). Os valores DataControl e LabelControl verificam se pode ocorrer uma escrita nos valores OutputLabel e OutputData que receberão os valores InputLabel e InputData respectivamente.

Module SomSub:

```
1   module SubCounter(clk,RUN,reset,q);
2      input clk,RUN,reset;
3      output reg [2:0]q;
4      initial
5         q = 0;
6      always@(posedge clk) begin
7         if(q == 3 || reset)
8            q = 0;
9         else if(RUN)
10           q = q + 1;
11     end
12  endmodule
```

```
14  module SomSub(clk,RUN,RegX,RegY,RegZ,OpCode,Result,Done,XAddSub,LabelAddSub,EnderecoSaida,Label);
15     input [8:0]RegX,RegY,RegZ;
16     input [2:0]OpCode,XAddSub,LabelAddSub;
17     input clk,RUN;
18     output reg Done;
19     output [2:0]EnderecoSaida,Label;
20     output reg [8:0]Result;
21     wire [2:0]count;
22
23     assign EnderecoSaida = XAddSub;
24     assign Label = LabelAddSub;
25
26     always@(clk,RegX,RegY,RegZ,OpCode,RUN) begin
27        Done = 0;
28        if(count == 2 && OpCode == 3'b000) begin
29           Result = RegY + RegZ; //ADD
30           Done = 1;
31        end
32        else if(count == 2 && OpCode == 3'b001) begin
33           Result = RegY - RegZ; //SUB
34           Done = 1;
35        end
36     end
37     SubCounter c(clk,RUN,Done,count);
38  endmodule
```

O módulo SomSub é a unidade funcional que realiza as operações de soma e subtração. Primeiramente, ele verifica se houve uma alteração nos valores de clk, RegX, RegY, RegZ, OpCode ou Run. Havendo essa alteração, ele irá agora verificar se o opcode corresponde a uma dessas operações (add ou sub) e se count tem o valor de 2. Atendendo a estes requisitos, ele realizará a operação adequada, salvando o resultado em “Result” e deixará o valor de done ativo.

Module tomasulo:

```
1   module tomasulo(clk,regs_s0,regs_s1,regs_s2,regs_s3,regs_s4,regs_s5);
2         input clk;
3         integer i;
4         wire [20:0]Time;
5         reg breakLoop;
6         reg [20:0]minInputTime;
7         reg firstInst;
8         output reg [8:0]regs_s0,regs_s1,regs_s2,regs_s3,regs_s4,regs_s5;
9
10       //Variaveis para instruções novas
11        wire [11:0]InInst;
12        wire [2:0]InputZ,InputX,InputY,InputI;
13        assign InputZ = InInst[11:9];
14        assign InputX = InInst[8:6];
15        assign InputY = InInst[5:3];
16        assign InputI = InInst[2:0];
17
18       //Controle da estação
19        reg [2:0]AddSubFull;
```

```
18      //Controle da estação
19         reg [2:0]AddSubFull;
20
21      //Done de cada FU
22         wire DoneAddSub;
23      //Variaveis para FU de soma e subtração
24         reg Runaddsum;
25         reg [2:0]AddSubOp,InputX_AddSub,InLabelAddSub;
26         wire [2:0]AddressAddSubX,LabelAddSub;
27         reg [8:0]RegAddSubX,RegAddSubY,RegAddSubZ;
28         wire [8:0]ResultAddSub;
29
30
31      //Variaveis para controle do registrador
32         wire [8:0]Tags[6:0];
33         wire [8:0]Regs[6:0];
34         reg updateData[6:0];
35         reg updateLabel[6:0];
36         reg [8:0]newData, newLabel;
37
38      //Variaveis da estação de reserva
39         reg [0:0]Busy[5:0];
40         reg [8:0]ValueI[5:0];
41         reg [8:0]ValueJ[5:0];
42         reg [8:0]ValueK[5:0];
```

```
42         reg [8:0]ValueK[5:0];
43         reg [8:0]TagI[5:0];
44         reg [8:0]TagJ[5:0];
45         reg [8:0]TagK[5:0];
46         reg [11:0]Insts[5:0];
47         reg Exec[5:0];
48         reg [20:0]InputTime[5:0];
49
50         parameter SOM = 3'b000;
51         parameter SUB = 3'b001;
52
53         initial begin
54            InputTime[LabelAddSub]=21'b111111111111111111111;
55            AddSubFull = 0;
56            firstInst = 1;
57            Runaddsum = 0;
58            breakLoop = 0;
59            Insts[0] = InInst;
60            for(i = 0; i <= 6; i = i + 1) begin
61               updateLabel[i] = 0;
62               updateData[i] = 0;
63            end
64            for(i = 0; i <= 5; i = i + 1) begin
65               Busy[i] = 0;
66               Exec[i] = 0;
```

```
63          end
64          for(i = 0; i <= 5; i = i + 1) begin
65             Busy[i] = 0;
66             Exec[i] = 0;
67          end
68       end
69
70       always @(clk) begin
71          breakLoop = 0;
72          for(i = 0; i <= 6; i = i + 1) begin
73             updateLabel[i] = 0;
74             updateData[i] = 0;
75          end
76          @(posedge clk) begin
77
78          regs_s0=Regs[0];
79          regs_s1=Regs[1];
80          regs_s2=Regs[2];
81          regs_s3=Regs[3];
82          regs_s4=Regs[4];
83          regs_s5=Regs[5];
```

```
88       // Estação de reserva Soma/sub
89          if(AddSubFull != 3 && (InputI == SOM || InputI == SUB)) begin
90          //Passa  por toda a estação 0-3
91             for(i = 0; i <= 2 && breakLoop != 1; i = i + 1) begin
92                if(Busy[i] == 0) begin
93                      Busy[i] = 1;
94                      Exec[i] = 0;
95                      Insts[i] = InInst;
96                      InputTime[i] = Time;
97                      if(Tags[InputX] == 9'b111111111) begin
98                            ValueI[i] = Regs[InputX];
99                            TagI[i] = 9'b111111111;
100                     end
101                     else begin
102                           TagI[i] = Tags[InputX];
103                           ValueI[i] = 9'b111111111;
104                     end
105                     if(Tags[InputY] == 9'b111111111) begin
106                           ValueJ[i] = Regs[InputY];
107                           TagJ[i] = 9'b111111111;
108                     end
109                     else begin
110                           TagJ[i] = Tags[InputY];
111                           ValueJ[i] = 9'b111111111;
```

```
                    end
                    else begin
                          TagJ[i] = Tags[InputY];
                          ValueJ[i] = 9'bllllllllll;
                    end
                    if(Tags[InputZ] == 9'bllllllllll) begin
                          ValueK[i] = Regs[InputZ];
                          TagK[i] = 9'bllllllllll;
                    end
                    else begin
                          TagK[i] = Tags[InputZ];
                          ValueK[i] = 9'bllllllllll;
                    end
                    newLabel = i;
                    updateLabel[InputX] = 1;
                    AddSubFull = AddSubFull + 1;
                    breakLoop = 1; // Variavel para parar de percorrer
                end
             end //end for
          end //end if
          breakLoop = 0;
          end //end Always
```

```
breakLoop = 0;
end //end Always

//Controle de hazard
//Se enquadrar nesse caso verifica-se õ tempo que a instrução está sendo rodada;

   if(DoneAddSub | (firstInst) | ~Runaddsum)begin
      minInputTime = 9'bllllllllll;
      if(DoneAddSub) begin
         Runaddsum = 0;
         newData = ResultAddSub;
         updateData[AddressAddSubX] = 1;
         Busy[LabelAddSub] = 0;
         if(!(AddressAddSubX == InputX) && (Tags[AddressAddSubX] == LabelAddSub))begin
            wait( updateLabel[0] || updateLabel[1]|| updateLabel[2]||
            updateLabel[3]|| updateLabel[4]||
            updateLabel[5]|| updateLabel[6] == 0) #1  updateLabel[AddressAddSubX] = 1;
            newLabel = 9'bllllllllll;
         end

         if(AddSubFull != 0 && DoneAddSub) begin
            AddSubFull = 0;// estava assim: AddSubFull = AddSubFull - 1;
         end
```

```
            if(AddSubFull != 0 && DoneAddSub) begin
               AddSubFull = 0;// estava assim: AddSubFull = AddSubFull - 1;
            end

            for(i = 0; i <= 5; i = i + 1) begin
               if(TagI[i] == LabelAddSub) begin
                  TagI[i] = 9'bllllllllll;
                  ValueI[i] = ResultAddSub;
               end
               if(TagJ[i] == LabelAddSub) begin
                  TagJ[i] = 9'bllllllllll;
                  ValueJ[i] = ResultAddSub;
               end
               if(TagK[i] == LabelAddSub) begin
                  TagK[i] = 9'bllllllllll;
                  ValueK[i] = ResultAddSub;
               end
            end
          end // endif if(DoneAddSub) begin - linha 327
```

```
167             for(i = 0; i <= 2; i = i + 1) begin
168                if(Exec[i] == 0 && ((ValueI[i] != 9'b111111111) && (ValueJ[i] != 9'b111111111))) begin
169                   if(minInputTime == 9'b111111111) begin
170                      minInputTime = i;
171                   end
172                   else if(InputTime[minInputTime] > InputTime[i])begin
173                      minInputTime = i;
174                   end
175                end
176             end
177             if(minInputTime != 9'b111111111) begin
178                firstInst = 0;
179                Exec[minInputTime] = 1;
180                AddSubOp = Insts[minInputTime][2:0];
181                RegAddSubY = ValueJ[minInputTime];
182                RegAddSubX = ValueI[minInputTime];
183                RegAddSubZ = ValueK[minInputTime];
184                InputX_AddSub = Insts[minInputTime][8:6];
185                InLabelAddSub = minInputTime;
186                Runaddsum = 1;
187             end
188          end // if if(DoneAddSub | (firstInst) | ~Runaddsum)begin - linha 325
189       end
```

```
191       registrador1 reg0(clk,newData,updateData[0],newLabel,updateLabel[0],Tags[0],Regs[0]);
192       registrador1 reg1(clk,newData,updateData[1],newLabel,updateLabel[1],Tags[1],Regs[1]);
193       registrador1 reg2(clk,newData,updateData[2],newLabel,updateLabel[2],Tags[2],Regs[2]);
194       registrador2 reg3(clk,newData,updateData[3],newLabel,updateLabel[3],Tags[3],Regs[3]);
195       registrador0 reg4(clk,newData,updateData[4],newLabel,updateLabel[4],Tags[4],Regs[4]);
196       registrador1 reg5(clk,newData,updateData[5],newLabel,updateLabel[5],Tags[5],Regs[5]);
197       count PC(clk,Time);
198       SomSub UF_SomSub(clk,Runaddsum,RegAddSubX,RegAddSubY,RegAddSubZ,AddSubOp,ResultAddSub,
199                        DoneAddSub,InputX_AddSub,InLabelAddSub,AddressAddSubX,LabelAddSub);
200       FilaInstrucoes instrucoes(clk,0,InInst,AddSubFull);
201
202    endmodule
```

O módulo Tomasulo interliga todos os componentes do projeto. Nele também está implementada a estação de reserva SomaSub, responsável por armazenar as instruções referentes a estas operações (adição e soma). A estação de reserva também é responsável por trabalhar o funcionamento dos rótulos (Tags). Neste módulo Tomasulo como um todo, temos os valores ValueI[ ], que armazena o conjunto de valores do registrador I. Assim, ele passa para a variável RegAddSubX o valor ValueI correspondente (tendo como controle de posição do vetor a variável “minInputTime”, ou seja, ValueI[min]). Essas operações que ocorrem dentro do módulo Tomasulo são essenciais para o correto funcionamento do projeto.

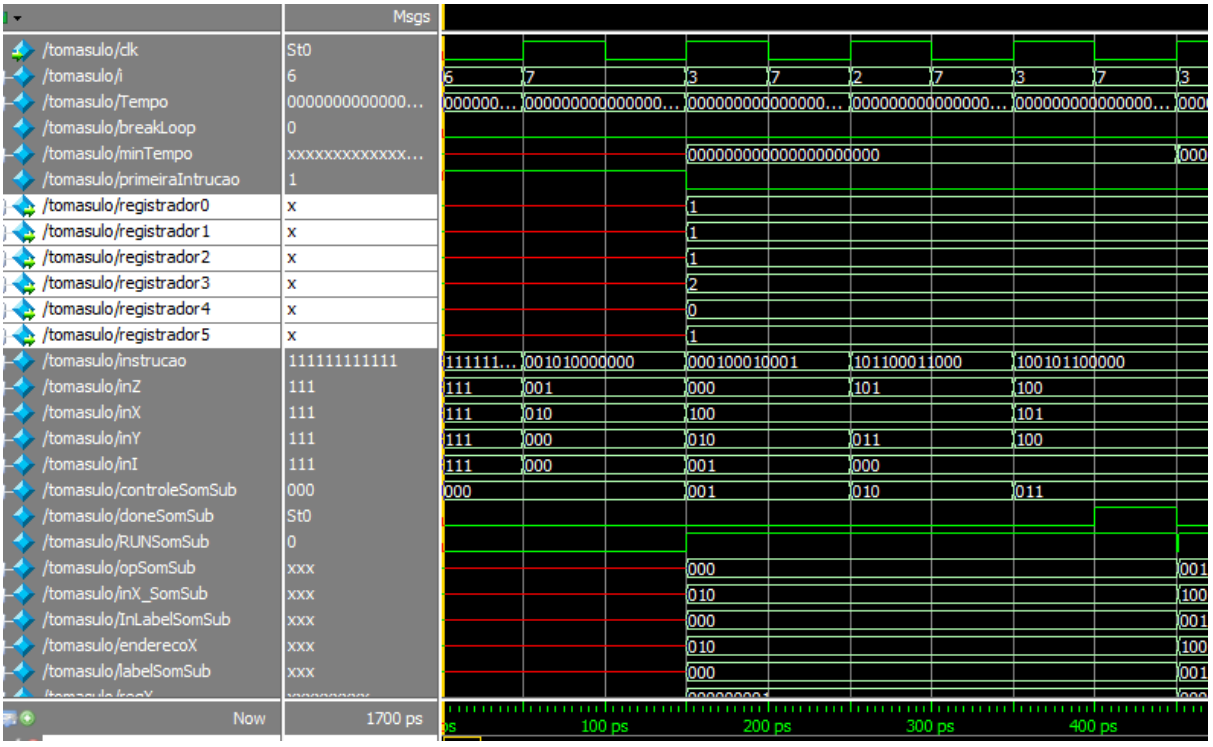
3 - Código de teste e explicação

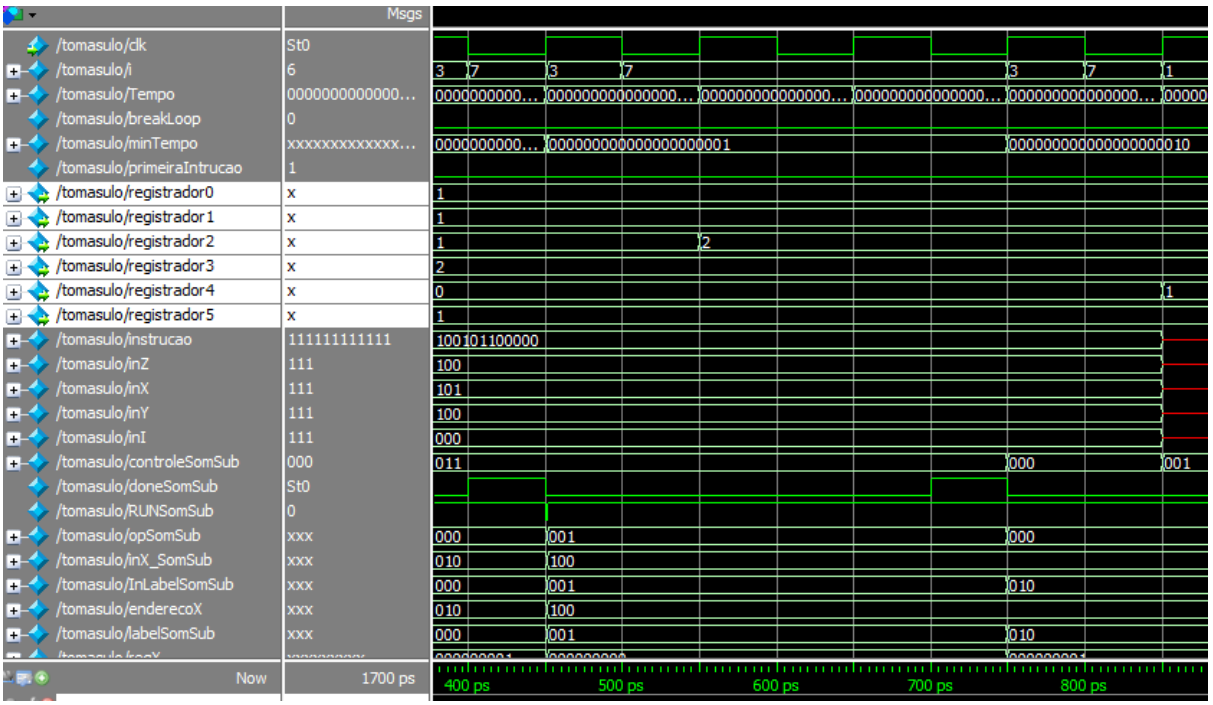
Tabela de instruções executadas:

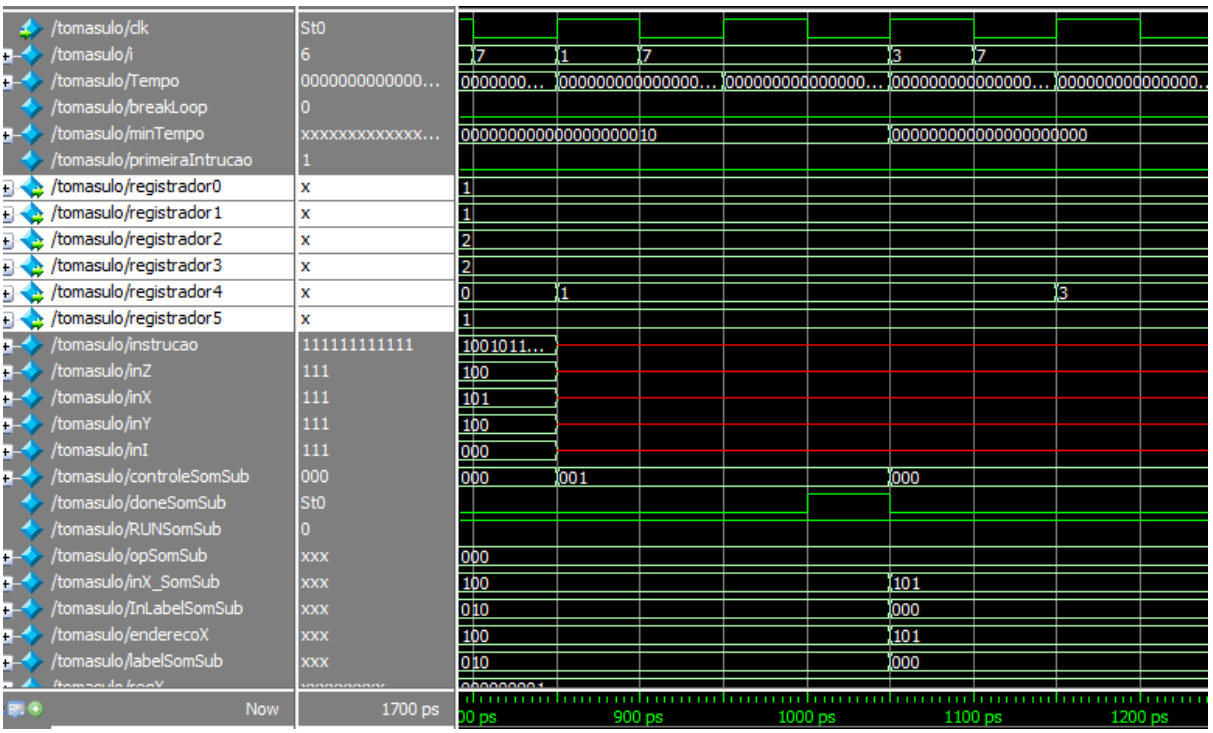
Como não conseguimos implementar as instruções de multiplicação no nosso Tomasulo, decidimos por simular somente somente as instruções de soma e subtração. Todas executaram de maneira adequada.

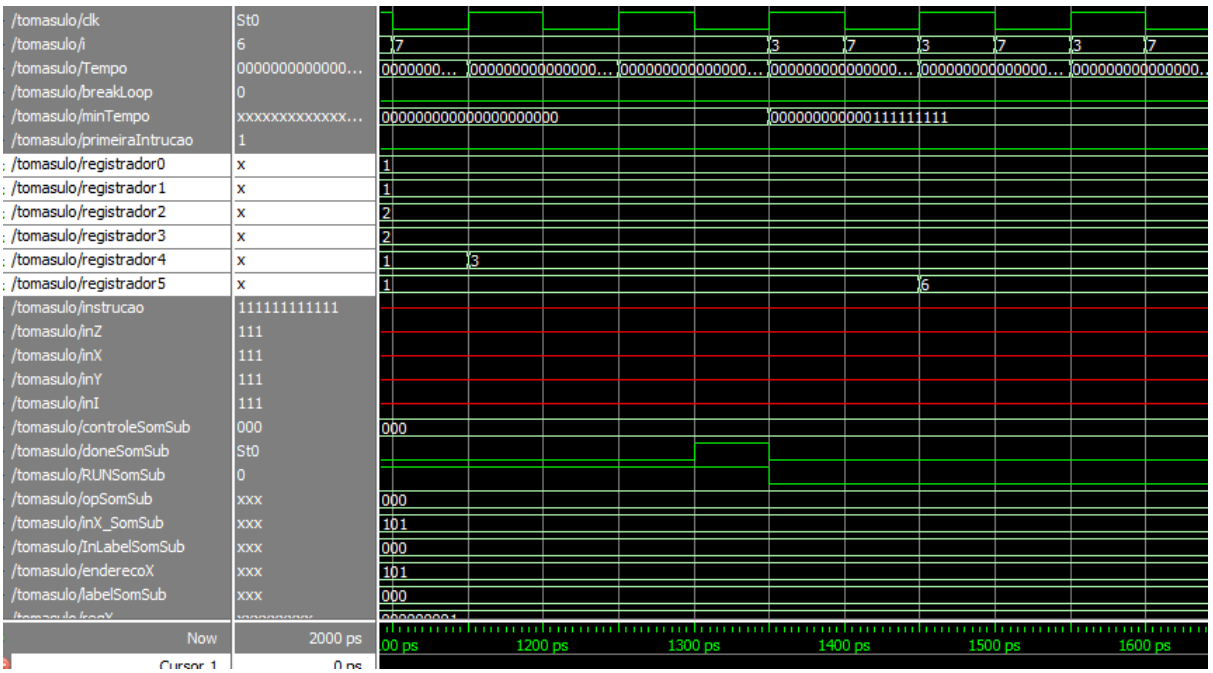
Instrução	R0	R1	R2	R3	R4	R5
Estado inicial	1	1	1	2	0	1
ADD R2, R0, R1	1	1	2	2	0	1
SUB R4, R2, R0	1	1	2	2	1	1
ADD R4, R3, R5	1	1	2	2	3	1
ADD R5, R4, R4	1	1	2	2	3	6

A execução das instruções acima foi apresentada durante a apresentação da prática. Para fins de avaliação, segue imagens das simulações sendo executadas:




Msgs
/tomasulo/clk
/tomasulo/i
/tomasulo/Tempo
/tomasulo/breakLoop
/tomasulo/minTempo
/tomasulo/primeiraIntrucao
/tomasulo/registrador0
/tomasulo/registrador1
/tomasulo/registrador2
/tomasulo/registrador3
/tomasulo/registrador4
/tomasulo/registrador5
/tomasulo/instrucao
/tomasulo/inZ
/tomasulo/inX
/tomasulo/inY
/tomasulo/inI
/tomasulo/controleSomSub
/tomasulo/doneSomSub
/tomasulo/RUNSomSub
/tomasulo/opSomSub
/tomasulo/inX_SomSub
/tomasulo/InLabelSomSub
/tomasulo/enderecoX
/tomasulo/labelSomSub
Now
1700 ps
400 ps
500 ps
600 ps
700 ps
800 ps


/tomasulo/clk
/tomasulo/i
/tomasulo/Tempo
/tomasulo/breakLoop
/tomasulo/minTempo
/tomasulo/primeiraIntrucao
/tomasulo/registrador0
/tomasulo/registrador1
/tomasulo/registrador2
/tomasulo/registrador3
/tomasulo/registrador4
/tomasulo/registrador5
/tomasulo/instrucao
/tomasulo/inZ
/tomasulo/inX
/tomasulo/inY
/tomasulo/inI
/tomasulo/controleSomSub
/tomasulo/doneSomSub
/tomasulo/RUNSomSub
/tomasulo/opSomSub
/tomasulo/inX_SomSub
/tomasulo/InLabelSomSub
/tomasulo/enderecoX
/tomasulo/labelSomSub
Now
1700 ps
900 ps
1000 ps
1100 ps
1200 ps



Podemos perceber que todas as instruções executaram seus respectivos papéis, atribuindo os valores de resultado ao registradores corretos. As instruções executaram na mesma ordem em que foram alocadas no módulo que simula a memória de instrução.

4 - Dificuldades encontrada

Ao longo da prática algumas dificuldade tiveram que ser enfrentadas, dentre as quais gostaríamos de ressaltar:

- Tempo: levando em consideração que a entrega da Prática II ocorreu no dia 05 de março, tivemos somente 14 dias para realizar a Prática III. Embora tenhamos iniciado a mesma desde o primeiro dia, e trabalhado nela mais do que uma hora por dia (havendo dias em que trabalhamos 5 horas seguidas com ela), o tempo foi insuficiente para que pudéssemos finalizar todas as implementações necessárias ao Tomasulo.
- Falta de um material de base: Seria de grande ajuda caso fosse fornecido um modelo de base (reduzido) que servisse para nortear a construção do nosso Tomasulo e, assim, aprenderíamos mais com a prática.

5 - Sugestões de melhorias da prática

Além dos tópicos mencionados no tópico 4, Dificuldades Encontradas, gostaríamos de propor algumas melhorias para a prática:

- A prática poderia ser subdividida em várias entregas individuais, cada uma referente a uma “parte” do Tomasulo, distribuídas ao longo do tempo. Cada entrega poderia ser avaliada individualmente. Após cada entrega, seria disponibilizado um código gabarito, referente a entrega que acabou de ocorrer. Assim, para as próximas entregas os alunos poderiam utilizar o seu código como base. Assim, evitaria que erros cometidos nos módulos básicos comprometam todo o projeto.
- A disponibilização de enunciados mais claros e auto-explicativos, não só para a prática do Tomasulo, mas como para as outras práticas também.

6 - Comentários adicionais

Embora tenhamos ressaltado dos pontos de dificuldade e dois pontos em que a prática poderia ser melhorada, gostaríamos também de ressaltar dois pontos positivos:

- A disponibilidade da Professora em tirar dúvidas, tanto na aula de dúvidas da disciplina de laboratório quanto na disciplina teórica.
- O fato da prática ser realizada em dupla, o que permite com que surjam mais ideias de como resolver o problema.