Relatório - Laboratório 3 - Grupo 2

Eduardo Ferreira de Assis, 17/0102289
Thiaggo Ferreira Bispo de Souza, 17/0157024
Emanoel Johannes Cardim Lazaro, 17/0140997
Alexandre Souza Costa Oliveira, 17/0098168
Gabriel Pinheiro dos Santos, 170103579

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CiC 116394 - Organização e Arquitetura de Computadores - Turma A

Questionário

1.

1.1.

1.2.

1.3.

1.4.

Multiciclo - RV32I	
Requerimentos físicos:	Requerimentos temporais:
Número de ALMs: 4,114	Maior atraso tpd: 25.989
Número de Registradores: 4542	th: 0.762 tco: 12.284 tsu: 3.239
Quantidade de bits de memória: 2,823,168	Máxima frequência de Clock Utilizável: 92.53 MHz
Número de DSP: 12	Slack Setup: 11.263 Hold: 0.480
Multiciclo - RV32IM	
Requerimentos físicos:	Requerimentos temporais:
Número de ALMs: 6,904	Maior atraso tpd: 21.894
Número de Registradores: 4518	th: 0.762 tco: 16.424 tsu: 3.658
Quantidade de bits de memória: 2,823,168	Máxima frequência de Clock Utilizável: 90.07 MHz
Número de DSP: 24	Slack Setup: 11.115 Hold: 0.507
Multiciclo - RV32IMF	
Requerimentos físicos:	Requerimentos temporais:
Número de ALMs: 9,303	Maior atraso tpd: 23.804
Número de Registradores: 6920	th: 0.762 tco: 17.907 tsu: 2.846
Quantidade de bits de memória: 2,870,784	Máxima frequência de Clock Utilizável: 91.87 MHz
Número de DSP: 30	Slack Setup: 11.224 Hold: 0.441

Figura 1. Tabelas das diferentes implementações do processador RV32 Multiciclo

De acordo com os resultados, foi possível notar uma grande diferença nos requerimentos físicos a cada implementação analisada: é possível notar que da implementação RV32I para a implementação RV32IM houve um aumento significativo do número de ALMs e do número de DSPs com uma pequena redução no número de registradores (algo um tanto incomum), porém nenhuma alteração no número de bits de memória, o que mostra

que houve uma incrementação de hardware e que o RV32IM é uma implementação mais complexa, já que usa mais recursos. Já o RV32IMF é bem mais complexo, uma vez que aumentamos o número de ALMs mais uma vez, assim como o número de registradores e DSPs e também aumentamos o número de bits de memória.

No caso dos requisitos temporais chamam a atenção o tempo de tpd que variou durante as implementações de uma forma inesperada, o tco que aumentou de acordo com o nível de complexidade da implementação, o que era esperado, e o Clock que também variou de forma inesperada, aumentando de 90.07 MHz para 91.87 MHz se compararmos o RV32IM com o RV32IMF.

2. Está no arquivo .qar

3.

3.1. Para a confecção do multiciclo RV32IMF com detecção de exceções foram necessárias algumas mudanças no datapath, no CPU, no controle e nos parâmetros, além da criação de um novo circuito responsável pelos registradores de exceção, o mesmo usado no uniciclo.

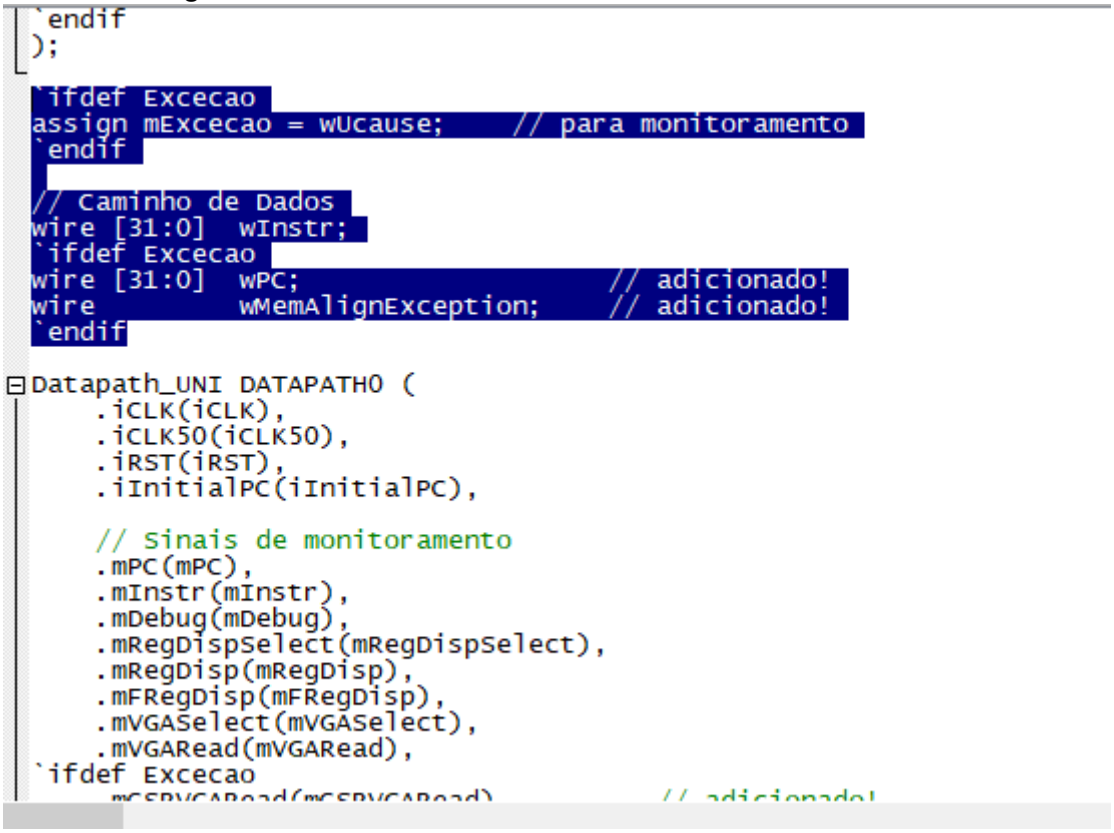
*CPU:

No cpu foram criados os fios necessários no datapath e no controle para o funcionamento do novo hardware.

Figura 2. Os fios com comentários foram alterados ou adicionados.

```
// Unidade de Controle
wire             wCOrigAULA;
wire             wCOrigBULA;
wire             wCRegWrite;
wire             wCMemWrite;
wire             wCMemRead;
wire [ 2:0]  wCMem2Reg;                 // alteraod
wire [ 2:0]  wCOrigPC;                  // alterado!
wire [ 4:0]  wCALUControl;
`ifdef RV32IMF
wire             wCFRegWrite;
wire [ 4:0]  wCFPALUControl;
wire             wCOrigAFPALU;
wire             wCFPALU2Reg;
wire             wCFWriteData;
wire             wCWrite2Mem;
wire             wCFPstart;
`endif

`ifdef Excecao
wire             wCSRInstruction;                           // Instrucao CSR
wire             wCSRException;                             // Excecao CSR
wire [3:0]   wUcause;                       // Causa da excecao
`endif
```

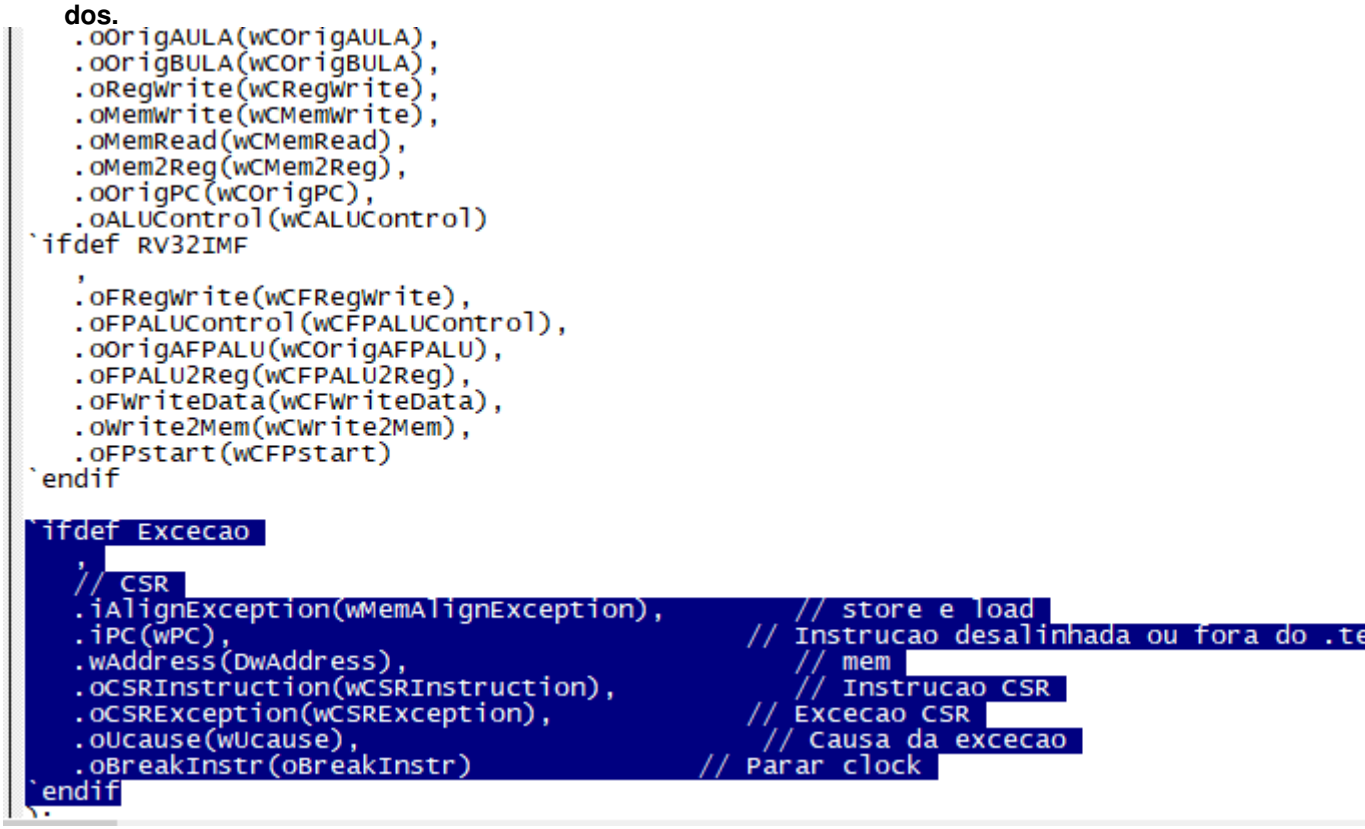
Figura 3. Os fios selecionados foram alterados ou adicionados.



```
`endif
);

`ifdef Excecao
assign mExcecao = wUcause;     // para monitoramento
`endif

// Caminho de Dados
wire [31:0]  wInstr;
`ifdef Excecao
wire [31:0]  wPC;                       // adicionado!
wire         wMemAlignException;        // adicionado!
`endif

Datapath_UNI DATAPATH0 (
    .iCLK(iCLK),
    .iCLK50(iCLK50),
    .iRST(iRST),
    .iInitialPC(iInitialPC),

    // Sinais de monitoramento
    .mPC(mPC),
    .mInstr(mInstr),
    .mDebug(mDebug),
    .mRegDispSelect(mRegDispSelect),
    .mRegDisp(mRegDisp),
    .mFRegDisp(mFRegDisp),
    .mVGASelect(mVGASelect),
    .mVGARead(mVGARead),
`ifdef Excecao
```

Para monitorar o funcionamento do multiciclo e facilitar o debug do processador, alguns fios de monitoramento foram adicionados, como o mExcecao.

Figura 4. Os fios com comentário ou selecionados foram alterados ou adicionados.



```
    .oOrigAULA(wCOrigAULA),
    .oOrigBULA(wCOrigBULA),
    .oRegWrite(wCRegWrite),
    .oMemWrite(wCMemWrite),
    .oMemRead(wCMemRead),
    .oMem2Reg(wCMem2Reg),
    .oOrigPC(wCOrigPC),
    .oALUControl(wCALUControl)
`ifdef RV32IMF
    ,
    .oFRegWrite(wCFRegWrite),
    .oFPALUControl(wCFPALUControl),
    .oOrigAFPALU(wCOrigAFPALU),
    .oFPALU2Reg(wCFPALU2Reg),
    .oFWriteData(wCFWriteData),
    .oWrite2Mem(wCWrite2Mem),
    .oFPstart(wCFPstart)
`endif

`ifdef Excecao
    ,
    // CSR
    .iAlignException(wMemAlignException),          // store e load
    .iPC(wPC),                                  // Instrucao desalinhada ou fora do .te
    .wAddress(DwAddress),                          // mem
    .oCSRInstruction(wCSRInstruction),             // Instrucao CSR
    .oCSRException(wCSRException),              // Excecao CSR
    .oUcause(wUcause),                           // Causa da excecao
    .oBreakInstr(oBreakInstr)                 // Parar clock
`endif
);
```

Nessa imagem nós ligamos os fios à caixa do CSR.

*Datapath:

No datapath, foram criados um registrador de 32 bits CSRout para guardar a saída do CSR para que possa ser usada em outras etapa e 4 fios de 32 bits Utvect (para receber utvect), Uepc (para receber uepc), CSRData (que liga a saida CSR ao CSROut) e wCSRVGARead para imprimir coisas na tela.

Figura 5. Os fios com comentários foram alterados ou adicionados.

```
`ifdef Excecao
// Banco de registradores de causa e status
reg  [31:0]    CSROut;
wire [31:0]    wUtvect;
wire [31:0]    wUepc;
wire [31:0]    wCSRData;
wire [31:0]    wCSRVGARead;
assign oMemAlignException = (wStoreException | wLoadException);          // desalinha

CSR CSR0 (
    .iCLK(iCLK),
    .iRST(iRST),

    .iCSRInstruction(iCSRInstruction),          // Caso instrção CSR
    .iCSRException(iCSRException),              // Caso de exceção
    .iInstr(IR),
    .iMemAddress(ALUOut),
    .iPC(PCBack),
    .iData(A),                                 // dado a ser escrito (reg saida ula)
    .oData(wCSRData),                           // saida CSR

    .iCAUSE(iUcause),
    .oUEPC(wUepc),
    .oUTVECT(wUtvect) ,

    // VGA
    .iRegDispSelect(),                        // seleção para display
    .oRegDisp(),                              // Reg display colocar fregdisp
    .iVGASelect(wVGASelect),                  // para mostrar Regs na tela
    .oVGARead(wCSRVGARead)                    // para mostrar Regs na tela colocar wfvga
);
```

Foi definido o fio wPC que recebe PCBack(PC da instrução em execução) no datapath e liga-se ao controle e alterados os fios wCMem2Reg e wCOrigPC para que possamos utilizar mais uma entrada em ambos os multiplexadores. Agora o Mem2Reg pode receber o dado gravado em CSROut para escrever no banco de registradores de acordo com as instruções CSRR e o OrigPC agora pode receber Utvect e Uepc para o tratamento de exceções e para a instrução uret.

Figura 6. Os fios com comentários foram alterados ou adicionados.

```
wire [31:0] wRegWrite;
always @(*)
    case(wCMem2Reg)
        3'b000:    wRegWrite <= ALUOut;
        3'b001:    wRegWrite <= PC;
        3'b010:    wRegWrite <= MDR;
`ifdef RV32IMF                                          //RV32IMF
        3'b011:    wRegWrite <= FPALUOut; // Uma entrada a mais no multiplexador de escrita no registrad
`endif
`ifdef Excecao
        3'b100:    wRegWrite <= CSROut;           // reg saida CSR
`endif
        default:   wRegWrite <= ZERO;
    endcase

wire [31:0] wiPC;
always @(*)
    case(wCOrigPC)
        3'b000:     wiPC <= wALUresult;                 // PC+4
      3'b001:     wiPC <= ALUOut;                        // Branches e jal
        3'b010:    wiPC <= wALUresult & ~(32'h1);    // jalr
`ifdef Excecao
        3'b011:    wiPC <= wUtvect;
        3'b100:    wiPC <= wUepc;
`endif
        default:       wiPC <= ZERO;
    endcase
```

Figura 7. Os fios com comentários foram alterados ou adicionados.

```
    output wire [31:0] wInstr,
`ifdef Excecao
    output wire [31:0] wPC,                 // adicionado!
`endif
    input  wire        wCEscreveIR,
    input  wire        wCEscrevePC,
    input  wire        wCEscrevePCCond,
    input  wire        wCEscrevePCBack,
    input  wire [ 1:0] wCOrigAULA,
    input  wire [ 1:0] wCOrigBULA,
    input  wire [ 2:0] wCMem2Reg,           // alterado!
    input  wire [ 2:0] wCOrigPC,            // alterado!
    input  wire        wCIouD,
    input  wire        wCRegWrite,
    input  wire        wCMemWrite,
    input  wire        wCMemRead,
    input  wire [ 4:0] wCALUControl,
`ifdef RV32IMF
    output wire        wFPALUReady,
    input  wire        wCFRegWrite,
    input  wire [ 4:0] wCFPALUControl,
    input  wire        wCOrigAFPALU,
    input  wire        wCFPALUStart,
    input  wire        wCFWriteData,
    input  wire        wCWrite2Mem,
`endif

     //  Barramento
    output wire        DwReadEnable, DwWriteEnable,
    output wire [ 3:0] DwByteEnable,
    output wire [31:0] DwAddress, DwWriteData
```

No barramento de dados foram definidos inputs e um output para o funcionamento da detecção de exceção. O CSRInstruction serve para dizer se estamos executando uma instrução, o CSRException serve para dizer se ocorreu uma exceção, o Ucause é o fio

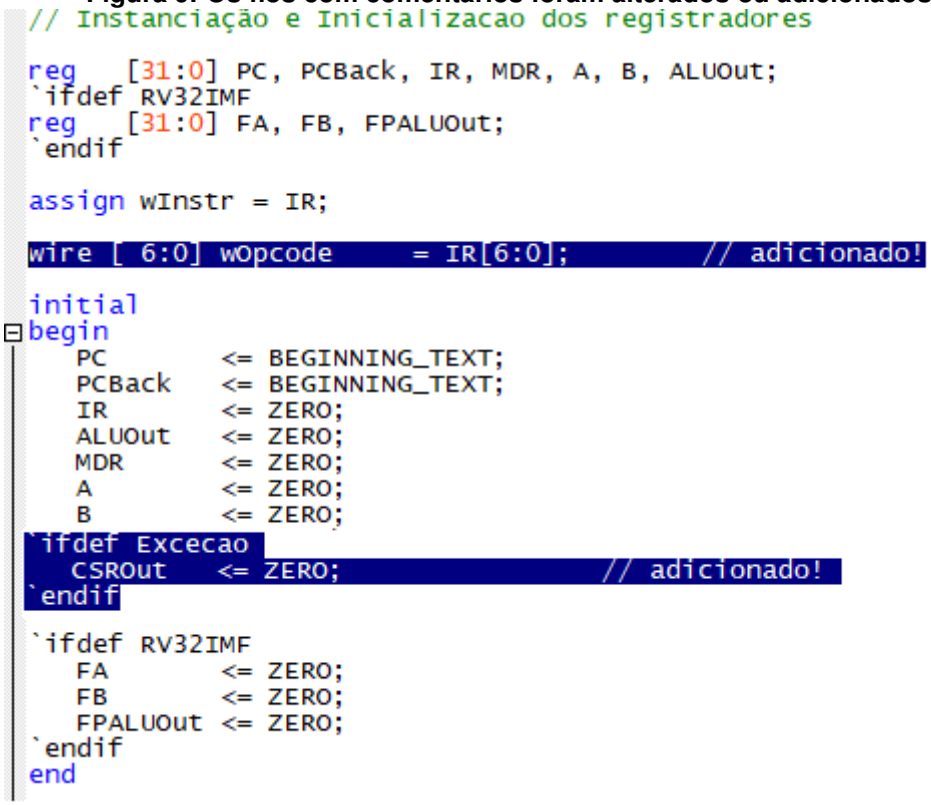
que vem do controle que marca o tipo de exceção ocorrida e o MemAlignException é o fio responsável por indicar se houve um desalinhamento ou segment fault do endereço do store ou do load.

Figura 8. Os fios com comentários foram alterados ou adicionados.

```
    input  wire         wCOrigAFPALU,
    input  wire         wCFPALUStart,
    input  wire         wCFWriteData,
    input  wire         wCWrite2Mem,
`endif

     //  Barramento
    output wire         DwReadEnable, DwWriteEnable,
    output wire [ 3:0] DwByteEnable,
    output wire [31:0] DwAddress, DwWriteData,
    input  wire [31:0] DwReadData
`ifdef Excecao
    ,
    // CSR e exceção
    input wire         iCSRInstruction,
    input wire         iCSRException,
    input wire [3:0]   iUcause,

    output wire        oMemAlignException
`endif
);
```

Para que a memória detecte se estamos em uma operação de load ou store, criamos um fio de opcode que permite que a memória use o opcode da instrução para determinar quando o endereço recebido por ela deve ou não ser considerado um caso de exceção. Além disso, na inicialização, o CSROut recebe zero, porém a cada ciclo é atualizado com a saída do CSR.

Figura 9. Os fios com comentários foram alterados ou adicionados.

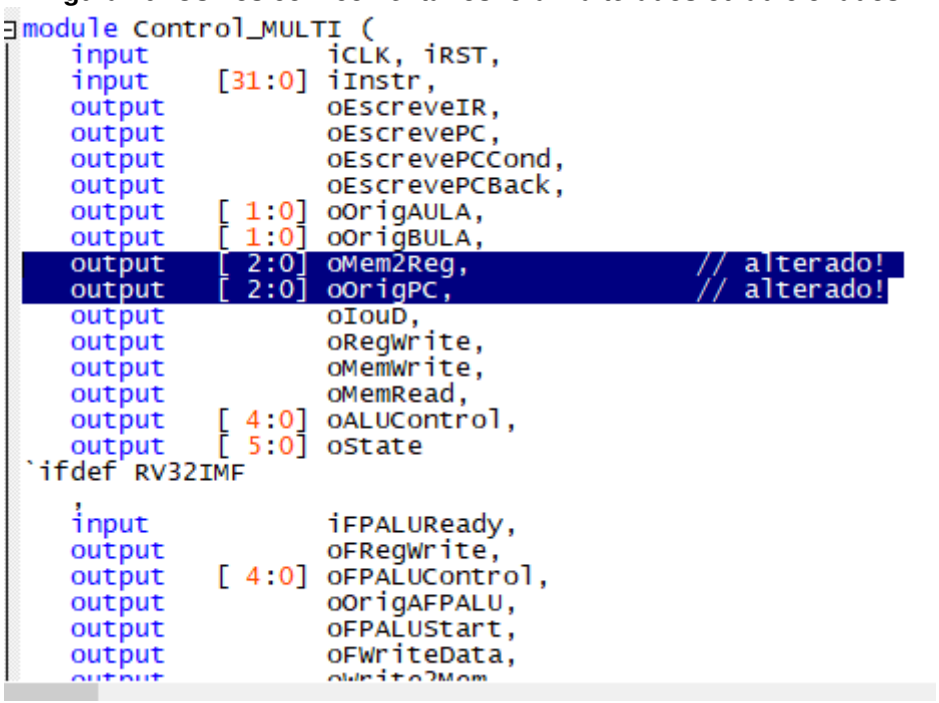


```
// Instanciação e Inicializacao dos registradores

reg   [31:0] PC, PCBack, IR, MDR, A, B, ALUOut;
`ifdef RV32IMF
reg   [31:0] FA, FB, FPALUOut;
`endif

assign wInstr = IR;

wire [ 6:0] wOpcode     = IR[6:0];        // adicionado!

initial
begin
   PC        <= BEGINNING_TEXT;
   PCBack    <= BEGINNING_TEXT;
   IR        <= ZERO;
   ALUOut    <= ZERO;
   MDR       <= ZERO;
   A         <= ZERO;
   B         <= ZERO;
`ifdef Excecao
   CSROut    <= ZERO;                  // adicionado!
`endif

`ifdef RV32IMF
   FA        <= ZERO;
   FB        <= ZERO;
   FPALUOut  <= ZERO;
`endif
end
```

*Controle

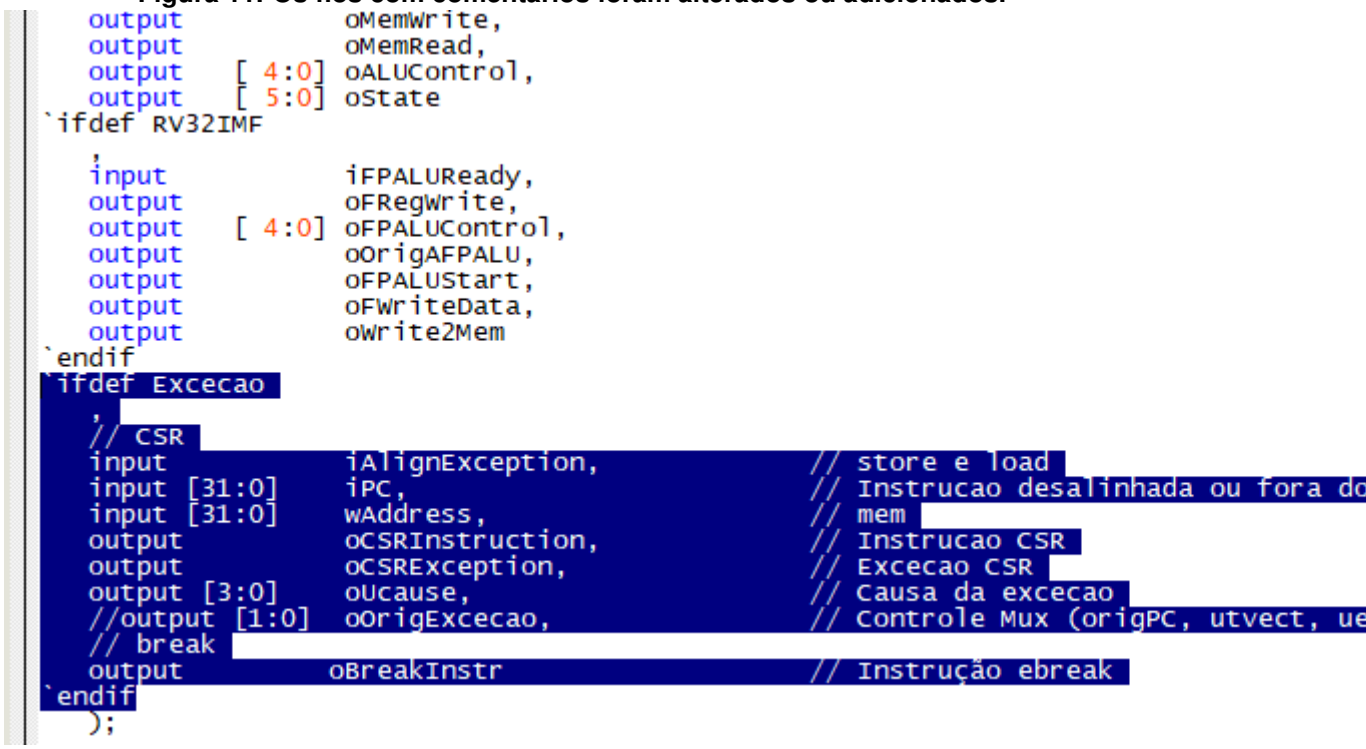
O controle também recebeu a alteração nos fios de controle dos muxes e o wPC. Ele agora possui os inputs AlignException (que contém uma flag que indica se houve um desalinhamento do endereço do load ou do store), PC (para definir se houve desalinhamento do endereço ou segment fault), Address (para verificar se o endereço que sai da ULA esta fora do segmento .data). Como outputs adicionamos CSRInstruction, CSRException, Ucause e BreakInstr (ligada a um fio conectado na break interface no TopDe). Esses outputs serão importantes para cada estado de exceção e das instruções implementadas.

Figura 10. Os fios com comentários foram alterados ou adicionados.



```
module Control_MULTI (
    input           iCLK, iRST,
    input    [31:0] iInstr,
    output          oEscreveIR,
    output          oEscrevePC,
    output          oEscrevePCCond,
    output          oEscrevePCBack,
    output   [ 1:0] oOrigAULA,
    output   [ 1:0] oOrigBULA,
    output   [ 2:0] oMem2Reg,              // alterado!
    output   [ 2:0] oOrigPC,               // alterado!
    output          oIouD,
    output          oRegWrite,
    output          oMemWrite,
    output          oMemRead,
    output   [ 4:0] oALUControl,
    output   [ 5:0] oState
`ifdef RV32IMF
    ,
    input           iFPALUReady,
    output          oFRegWrite,
    output   [ 4:0] oFPALUControl,
    output          oOrigAFPALU,
    output          oFPALUStart,
    output          oFWriteData,
    output          oWrite2Mem
```

Figura 11. Os fios com comentários foram alterados ou adicionados.



```
    output          oMemWrite,
    output          oMemRead,
    output   [ 4:0] oALUControl,
    output   [ 5:0] oState
`ifdef RV32IMF
    ,
    input           iFPALUReady,
    output          oFRegWrite,
    output   [ 4:0] oFPALUControl,
    output          oOrigAFPALU,
    output          oFPALUStart,
    output          oFWriteData,
    output          oWrite2Mem
`endif
`ifdef Excecao
    ,
    // CSR
    input           iAlignException,       // store e load
    input [31:0]    iPC,                   // Instrucao desalinhada ou fora do
    input [31:0]    wAddress,              // mem
    output          oCSRInstruction,       // Instrucao CSR
    output          oCSRException,         // Excecao CSR
    output [3:0]    oUcause,               // Causa da excecao
    //output [1:0]  oOrigExcecao,          // Controle Mux (origPC, utvect, ue
    // break
    output          oBreakInstr            // Instrução ebreak
`endif
    );
```

Abaixo, temos os estados de exceção e, em seguida os estados das instruções implementadas.

Figura 12. Os fios com comentários foram alterados ou adicionados.

```
    output            oMemWrite,
    output            oMemRead,
    output    [ 4:0] oALUControl,
    output    [ 5:0] oState
`ifdef RV32IMF
    ,
    input             iFPALUReady,
    output            oFRegWrite,
    output    [ 4:0] oFPALUControl,
    output            oOrigAFPALU,
    output            oFPALUStart,
    output            oFWriteData,
    output            oWrite2Mem
`endif
`ifdef Excecao
    ,
    // CSR
    input             iAlignException,          // store e load
    input [31:0]      iPC,                      // Instrucao desalinhada ou fora do
    input [31:0]      wAddress,                 // mem
    output            oCSRInstruction,          // Instrucao CSR
    output            oCSRException,            // Excecao CSR
    output [3:0]      oUcause,                  // Causa da excecao
    //output [1:0]    oOrigExcecao,             // Controle Mux (origPC, utvect, ue
    // break
    output          oBreakInstr                 // Instrução ebreak
`endif
    );
```

Figura 13. As linhas com comentários foram alterados ou adicionados.

```
      ST_ERRO:
        begin
`ifdef Excecao
            oUcause <= 4'd2;            // isntrução inválida
            oCSRInstruction <= 1'b0;
            oCSRException <= 1'b1;
            oBreakInstr <= 1'b0;
`endif
            oEscreveIR     <= 1'b0;
            oEscrevePCCond <= 1'b0;
            oEscrevePCBack <= 1'b0;
            oOrigAULA      <= 2'b00;
            oOrigBULA      <= 2'b00;
            oMem2Reg       <= 3'b000;
`ifdef Excecao
            oEscrevePC     <= 1'b1;
            oOrigPC        <= 3'b011;     // pc = utvect
`else
            oEscrevePC     <= 1'b0;
            oOrigPC        <= 3'b000;     // pc = pc+4
`endif
            oIouD          <= 1'b0;
            oRegWrite      <= 1'b0;
            oMemWrite      <= 1'b0;
            oMemRead       <= 1'b0;
            oALUControl    <= OPNULL;
`ifdef RV32IMF                                  //RV32IMF
            oFRegWrite     <= 1'b0;
            oFPALUControl  <= OPNULL;
            oOrigAFPALU    <= 1'b0;
            oFPALUStart    <= 1'b0;
```

Figura 14. As linhas com comentários foram alterados ou adicionados.

```
        ST_NOTALIGNL:
            begin

                oUcause <= 4'd4;                // Endereço do load desalin
                oCSRInstruction <= 1'b0;
                oCSRException <= 1'b1;
                oBreakInstr <= 1'b0;

                oEscreveIR      <= 1'b0;
                oEscrevePC      <= 1'b1;
                oEscrevePCCond  <= 1'b0;
                oEscrevePCBack  <= 1'b0;
                oOrigAULA       <= 2'b00;
                oOrigBULA       <= 2'b00;
                oMem2Reg        <= 3'b000;
                oOrigPC         <= 3'b011;      // pc = utvect
                oIouD           <= 1'b0;
                oRegWrite       <= 1'b0;
                oMemWrite       <= 1'b0;
                oMemRead        <= 1'b0;
                oALUControl     <= OPNULL;
`ifdef RV32IMF                                      //RV32IMF
                oFRegWrite      <= 1'b0;
                oFPALUControl   <= OPNULL;
                oOrigAFPALU     <= 1'b0;
                oFPALUStart     <= 1'b0;
                oFWriteData     <= 1'b0;
                oWrite2Mem      <= 1'b0;
`endif
                nx_state        <= ST_FETCH;
```

Figura 15. As linhas com comentários foram alterados ou adicionados.

```
`ifdef Excecao
      ST_NOTALIGNPC:                                        // =============
         begin

            oUcause <= 4'd0;                 // PC desalinhado
            oCSRInstruction <= 1'b0;
            oCSRException <= 1'b1;
            oBreakInstr <= 1'b0;

            oEscreveIR      <= 1'b0;
            oEscrevePC      <= 1'b1;
            oEscrevePCCond  <= 1'b0;
            oEscrevePCBack  <= 1'b0;
            oOrigAULA       <= 2'b00;
            oOrigBULA       <= 2'b00;
            oMem2Reg        <= 3'b000;
            oOrigPC         <= 3'b011;       // pc = utvect
            oIouD           <= 1'b0;
            oRegWrite       <= 1'b0;
            oMemWrite       <= 1'b0;
            oMemRead        <= 1'b0;
            oALUControl     <= OPNULL;
`ifdef RV32IMF                                  //RV32IMF
            oFRegWrite      <= 1'b0;
            oFPALUControl   <= OPNULL;
            oOrigAFPALU     <= 1'b0;
            oFPALUStart     <= 1'b0;
            oFWriteData     <= 1'b0;
            oWrite2Mem      <= 1'b0;
`endif
```

Figura 16. As linhas com comentários foram alterados ou adicionados.

```
        ST_NOTALIGNS:
            begin

                oUcause <= 4'd5;                // Endereço do store desalinhado
                oCSRInstruction <= 1'b0;
                oCSRException <= 1'b1;
                oBreakInstr <= 1'b0;

                oEscreveIR      <= 1'b0;
                oEscrevePC      <= 1'b1;
                oEscrevePCCond  <= 1'b0;
                oEscrevePCBack  <= 1'b0;
                oOrigAULA       <= 2'b00;
                oOrigBULA       <= 2'b00;
                oMem2Reg        <= 3'b000;
                oOrigPC         <= 3'b011;      // pc = utvect
                oIouD           <= 1'b0;
                oRegWrite       <= 1'b0;
                oMemWrite       <= 1'b0;
                oMemRead        <= 1'b0;
                oALUControl     <= OPNULL;
`ifdef RV32IMF                                      //RV32IMF
                oFRegWrite      <= 1'b0;
                oFPALUControl   <= OPNULL;
                oOrigAFPALU     <= 1'b0;
                oFPALUStart     <= 1'b0;
                oFWriteData     <= 1'b0;
                oWrite2Mem      <= 1'b0;
`endif
                nx_state        <= ST_FETCH;
```

Figura 17. As linhas com comentários foram alterados ou adicionados.

```
ST_SEGFAULTL:
   begin

      oUcause <= 4'd5;                // Endereço do load fora do .data
      oCSRInstruction <= 1'b0;
      oCSRException <= 1'b1;
      oBreakInstr <= 1'b0;

      oEscreveIR     <= 1'b0;
      oEscrevePC     <= 1'b1;
      oEscrevePCCond <= 1'b0;
      oEscrevePCBack <= 1'b0;
      oOrigAULA      <= 2'b00;
      oOrigBULA      <= 2'b00;
      oMem2Reg       <= 3'b000;
      oOrigPC        <= 3'b011;       // pc = utvect
      oIouD          <= 1'b0;
      oRegWrite      <= 1'b0;
      oMemWrite      <= 1'b0;
      oMemRead       <= 1'b0;
      oALUControl    <= OPNULL;
`ifdef RV32IMF                          //RV32IMF
      oFRegWrite     <= 1'b0;
      oFPALUControl  <= OPNULL;
      oOrigAFPALU    <= 1'b0;
      oFPALUStart    <= 1'b0;
      oFWriteData    <= 1'b0;
      oWrite2Mem     <= 1'b0;
`endif
```

Figura 18. As linhas com comentários foram alterados ou adicionados.

```
ST_SEGFAULTPC:
   begin

      oUcause <= 4'd1;                // PC fora do .text
      oCSRInstruction <= 1'b0;
      oCSRException <= 1'b1;
      oBreakInstr <= 1'b0;

      oEscreveIR     <= 1'b0;
      oEscrevePC     <= 1'b1;
      oEscrevePCCond <= 1'b0;
      oEscrevePCBack <= 1'b0;
      oOrigAULA      <= 2'b00;
      oOrigBULA      <= 2'b00;
      oMem2Reg       <= 3'b000;
      oOrigPC        <= 3'b011;       // pc = utvect
      oIouD          <= 1'b0;
      oRegWrite      <= 1'b0;
      oMemWrite      <= 1'b0;
      oMemRead       <= 1'b0;
      oALUControl    <= OPNULL;
`ifdef RV32IMF                        // RV32IMF
      oFRegWrite     <= 1'b0;
      oFPALUControl  <= OPNULL;
      oOrigAFPALU    <= 1'b0;
      oFPALUStart    <= 1'b0;
      oFWriteData    <= 1'b0;
      oWrite2Mem     <= 1'b0;
`endif
```

Figura 19. As linhas com comentários foram alterados ou adicionados.

```
    ST_SEGFAULTS:
       begin

           oUcause <= 4'd6;                // Endereço do store fora do .data
           oCSRInstruction <= 1'b0;
           oCSRException <= 1'b1;
           oBreakInstr <= 1'b0;

           oEscreveIR     <= 1'b0;
           oEscrevePC     <= 1'b1;
           oEscrevePCCond <= 1'b0;
           oEscrevePCBack <= 1'b0;
           oOrigAULA      <= 2'b00;
           oOrigBULA      <= 2'b00;
           oMem2Reg       <= 3'b000;
           oOrigPC        <= 3'b011;       // pc = utvect
           oIouD          <= 1'b0;
           oRegWrite      <= 1'b0;
           oMemWrite      <= 1'b0;
           oMemRead       <= 1'b0;
           oALUControl    <= OPNULL;
`ifdef RV32IMF                                      //RV32IMF
           oFRegWrite     <= 1'b0;
           oFPALUControl  <= OPNULL;
           oOrigAFPALU    <= 1'b0;
           oFPALUStart    <= 1'b0;
           oFWriteData    <= 1'b0;
           oWrite2Mem     <= 1'b0;
`endif
           nx_state       <= ST_FETCH;
       end
```

Figura 20. As linhas com comentários foram alterados ou adicionados.

```
`ifdef Excecao
// ========================+============== adicionado!
        ST_CSR: // instruções CSR, exceções (ecall) e ebreak
            begin
                //oUcause <= 4'd0;
                //oCSRInstruction <= 1'b0;
                //oCSRException <= 1'b0;
                //oBreakInstr   <= 1'b0;

                oEscreveIR      <= 1'b0;
                //oEscrevePC        <= 1'b0;
                oEscrevePCCond <= 1'b0;
                oEscrevePCBack <= 1'b0;
                oOrigAULA       <= 2'b00;
                oOrigBULA       <= 2'b00;
                //oMem2Reg      <= 3'b000;
                //oOrigPC           <= 3'b000;
                oIouD           <= 1'b0;
                //oRegWrite         <= 1'b0;
                oMemWrite       <= 1'b0;
                oMemRead        <= 1'b0;
                oALUControl     <= OPNULL;
`ifdef RV32IMF                                                          //RV32IMF
                oFRegWrite     <= 1'b0;
                oFPALUControl  <= OPNULL;
                oOrigAFPALU    <= 1'b0;
                oFPALUStart    <= 1'b0;
                oFWriteData    <= 1'b0;
                oWrite2Mem     <= 1'b0;
`endif
                //nx_state      <= ST_FETCH;

                case (Funct3)
                    FUNCT3_ECALL,
                    FUNCT3_EBREAK,
                    FUNCT3_URET:
                    begin
                        oCSRInstruction <= 1'b0;
                        //nx_state      <= ST_ECALLURET;

                        case (Rs2)

                            RS2_ECALL:
                                begin
                                    oUcause <= 4'd8;
```

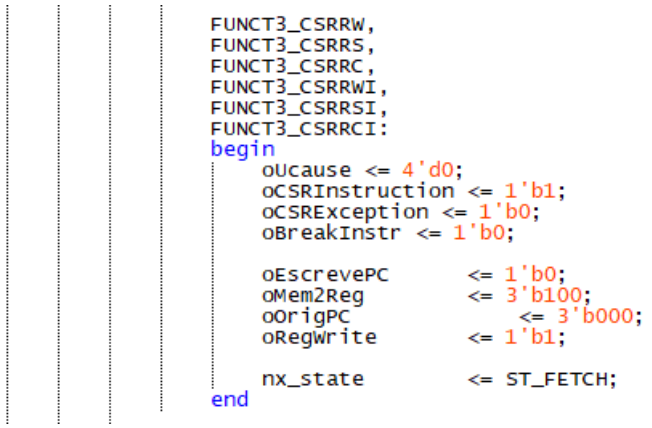
Figura 21. As linhas com comentários foram alterados ou adicionados.

```
            RS2_ECALL:
                begin
                    oUcause <= 4'd8;
                    oCSRException <= 1'b1;
                    oBreakInstr <= 1'b0;

                    oEscrevePC      <= 1'b1;
                    oMem2Reg        <= 3'b000;
                    oOrigPC             <= 3'b011;      // pc = utvect
                    oRegWrite       <= 1'b0;

                    nx_state        <= ST_FETCH;
                end
            RS2_URET:
                begin
                    oUcause <= 4'd0;
                    oCSRException <= 1'b0;
                    oBreakInstr <= 1'b0;

                    oEscrevePC      <= 1'b1;
                    oMem2Reg        <= 3'b000;
                    oOrigPC             <= 3'b100;      // pc = uepc
                    oRegWrite       <= 1'b0;

                    nx_state        <= ST_FETCH;
                end
            RS2_EBREAK:
                begin
                    oUcause <= 4'd0;
                    oCSRException <= 1'b0;
                    oBreakInstr <= 1'b1;

                    oEscrevePC      <= 1'b0;
                    oMem2Reg        <= 3'b000;
                    oOrigPC             <= 3'b000;
                    oRegWrite       <= 1'b0;

                    nx_state        <= ST_FETCH;
                end

            default:
                begin
                    oUcause <= 4'd0;
                    oCSRInstruction <= 1'b0;
                    oCSRException <= 1'b0;
```

Figura 22. As linhas com comentários foram alterados ou adicionados.

```
                default:
                    begin
                        oUcause <= 4'd0;
                        oCSRInstruction <= 1'b0;
                        oCSRException <= 1'b0;
                        oBreakInstr <= 1'b0;

                        oEscreveIR       <= 1'b0;
                        oEscrevePC       <= 1'b0;
                        oEscrevePCCond <= 1'b0;
                        oEscrevePCBack <= 1'b0;
                        oOrigAULA        <= 2'b00;
                        oOrigBULA        <= 2'b00;
                        oMem2Reg         <= 3'b000;
                        oOrigPC              <= 3'b000;
                        oIouD            <= 1'b0;
                        oRegWrite        <= 1'b0;
                        oMemWrite        <= 1'b0;
                        oMemRead         <= 1'b0;
                        oALUControl      <= OPNULL;
`ifdef RV32IMF
                        oFRegWrite     <= 1'b0;
                        oFPALUControl  <= OPNULL;
                        oOrigAFPALU    <= 1'b0;
                        oFPALUStart    <= 1'b0;
                        oFWriteData    <= 1'b0;
                        oWrite2Mem     <= 1'b0;
`endif
                        nx_state         <= ST_ERRO;
                    end
                endcase // rs2
            end

            FUNCT3_CSRRW,
            FUNCT3_CSRRS,
            FUNCT3_CSRRC,
            FUNCT3_CSRRWI,
            FUNCT3_CSRRSI,
            FUNCT3_CSRRCI:
            begin
                oUcause <= 4'd0;
                oCSRInstruction <= 1'b1;
                oCSRException <= 1'b0;
                oBreakInstr <= 1'b0;

                oEscrevePC       <= 1'b0;
```

Figura 23. As linhas com comentários foram alterados ou adicionados.



```
        FUNCT3_CSRRW,
        FUNCT3_CSRRS,
        FUNCT3_CSRRC,
        FUNCT3_CSRRWI,
        FUNCT3_CSRRSI,
        FUNCT3_CSRRCI:
        begin
            oUcause <= 4'd0;
            oCSRInstruction <= 1'b1;
            oCSRException <= 1'b0;
            oBreakInstr <= 1'b0;

            oEscrevePC      <= 1'b0;
            oMem2Reg        <= 3'b100;
            oOrigPC             <= 3'b000;
            oRegWrite       <= 1'b1;

            nx_state        <= ST_FETCH;
        end
```

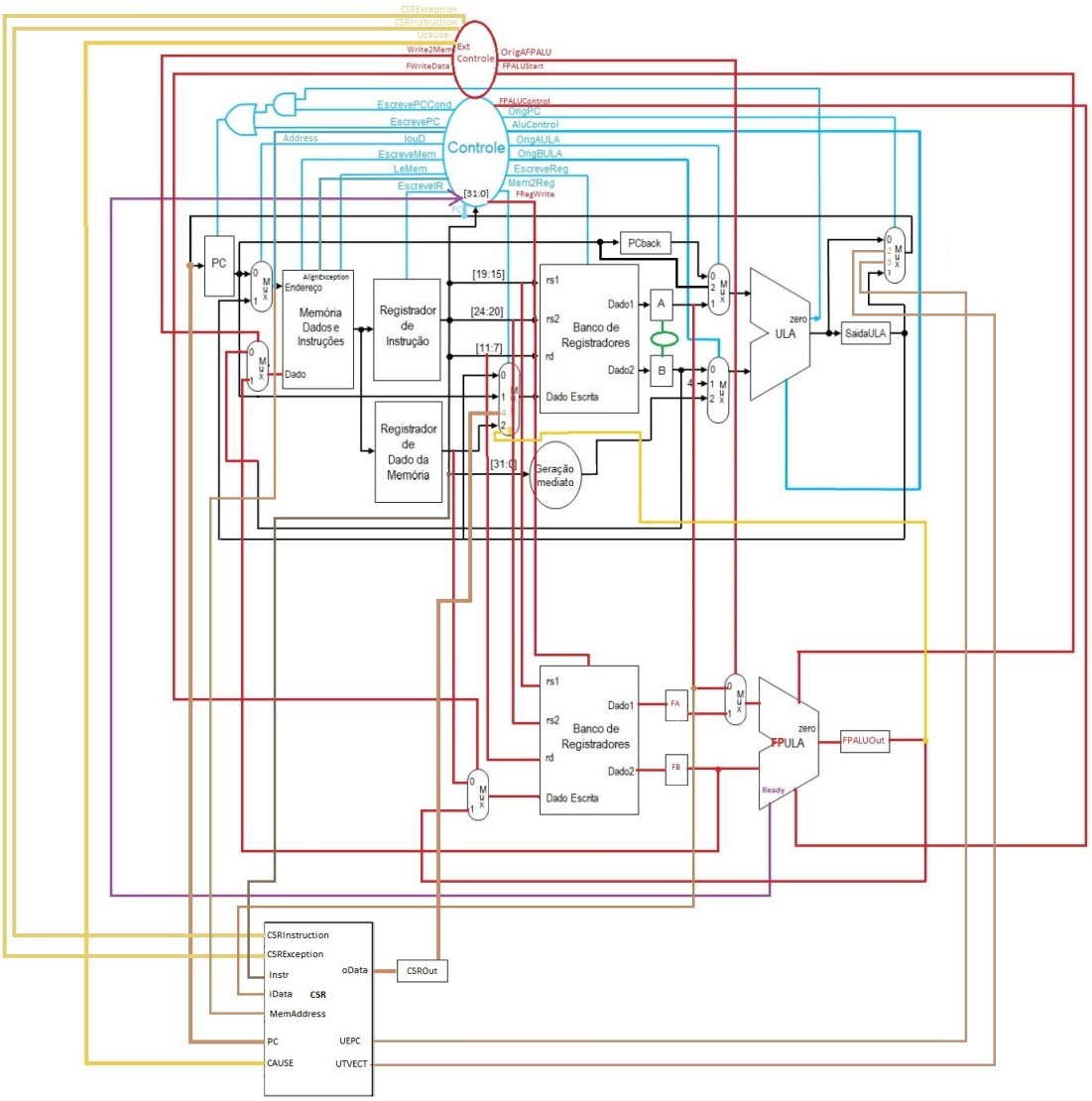
Assim, no controle, foram definidas, em cada etapa, as saídas de controle dos muxes, memória, registradores e ULAs de modo que as instruções e as exceções funcionassem de maneira correta.

*Parâmetros:

Nos parâmetros foram adicionados os códigos opcode, funct3, funct7 e rs2 de cada instrução, assim como uma macro para sabermos o final do .data e definir se um endereço está fora da faixa esperada.

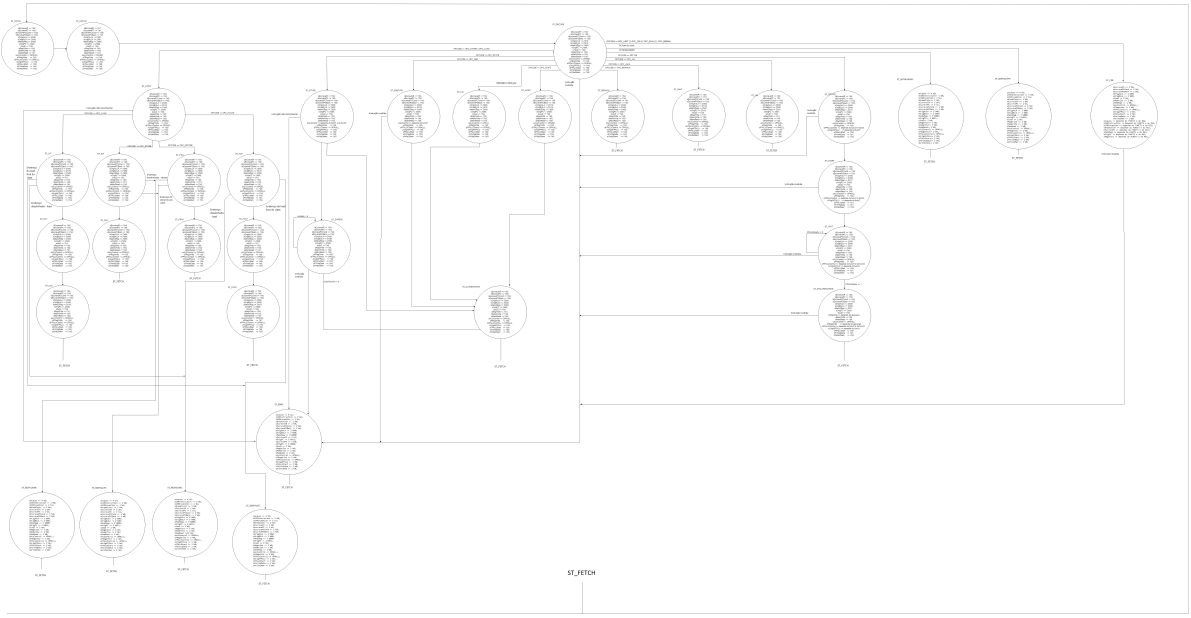
3.2.

Figura 24. Caminho de Dados da RV32IMF



3.3.

Figura 25. Máquina de estados.



3.4. As dificuldades que enfrentamos foram definir os estados das novas instruções de forma correta, respeitando o tempo necessário para que os dados estejam escritos corretamente nos registradores auxiliares de saída. Além de termos enfrentado bugs do QUARTUS que não permitiam que o projeto fosse 100% transferido para a placa DE1-SoC, foi necessário criar um ”fio solto”no Datapath para o processador passar 100% para a placa.

4. Link : `https://youtu.be/36J2HIvLJ_o`

De acordo com os resultados, foi possível observar que a implementação das instruções novas, assim como do tratamento de exceções foi feita de forma correta, uma vez que todas o processador é capaz de detectar os erros e pode executar, de maneira correta, todas as funções.

Não houve dificuldades de implementação, porém o acesso limitado ao laboratório (horário de uso coincidente com o horário das aulas, sobrando pouco tempo para realizar as tarefas) para realizar testes na DE1SoC prejudicou o andamento do trabalho.

5. Link: `https://youtu.be/cn9a5SOCLsIc`

Analisando os resultados do experimento, foi possível notar que o Ecall foi implementado com sucesso, permitindo que sejam feitas chamadas do sistema.

Não houve dificuldades de implementação, porém o acesso limitado ao laboratório (horário de uso coincidente com o horário das aulas, sobrando pouco tempo para realizar as tarefas) para realizar testes na DE1SoC prejudicou o andamento do trabalho.

6. Link: `https://youtu.be/EWluqEH3y4U`

O mais rápido foi o feito na DE1-SoC, pois na placa roda está rodando a um clock de 4.5 MHz em um circuito físico sem outros programas rodando paralelamente, diferentemente do RARS que é uma máquina java rodando em cima de um sistema operacional com

outras tarefas sendo executadas simultaneamente.

7. Link: `https://youtu.be/4u6pkf2ZCqM`