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.

RV32I	
Requerimentos Físicos:	Requerimentos Temporais:
ALMs: 4,004	Tpd: 24.839
Registradores: 3974	Th: 0.762; Tco: 15.052; Tsu: 2.793
Bits de Memória: 2,823,168	Máxima Frequência de Clock : 67.3Mhz
DSP: 12	Slack Setup: 9.237; Hold: 0.453

Figura 1. Tabela do RV32I

RV32IM	
Requerimentos Físicos:	Requerimentos Temporais:
ALMs: 6,800	Tpd: 27.353
Registradores: 3980	Th: 0.762 ; Tco: 15.107; Tsu: 3.846
Bits de Memória: 2,823,168	Máxima Frequência de Clock: 75.6Mhz
DSP: 24	Slack Setup: 10.053 ; Hold: 0.426

Figura 2. Tabela do RV32IM

RV32IMF	
Requerimentos Físicos:	Requerimentos Temporais:
ALMs: 9,691	Tpd: 29.361
Registradores: 6322	Th: 0.762; Tco: 16.459; Tsu: 3.147
Bits de Memória: 2,870,784	Máxima Frequência de Clock : 75.59Mhz
DSP: 30	Slack Setup: 10.052; Hold: 0.542

Figura 3. Tabela do RV32IMF

O que chama mais atenção são os requerimentos físicos das diferentes implementações do processador, pois as mudanças são mais expressivas. É possível observar que, beaseando-se no uniciclo RV32I, a adição das instruções de multiplicação e divisão (RV32IM) gera um aumento no número de requerimentos físicos, mantendo os bits de memória utilizados, porém adicionando 2796 ALMs, 6 registradores e chegando a dobrar no caso dos DSPs, o que mostra que necessitamos de mais hardware e área de chip; no caso da adição de instruções em ponto flutuante e das instruções de multiplicação (RV32IMF), temos um expressivo aumento nos requerimentos físicos, que, se comparado ao RV32IM, possui um aumento de 2891 ALMs, 2342 registradores, 47616 bits de memória e 6 DSPs, demontrando que é um hardware muito mais complexo do que o que temos no RV32I que necessita de mais área de chip.

Quanto aos requerimentos temporais, podemos ver que não houveram alterações tão expressivas, com exceção de uma mudança de clock de 67.3 MHz no RV32I para uma 75.6 MHz no RV32IM, que causa estranheza pois esperávamos que, com um hardware mais complexo, o período de clock seria maior.

2. Está no arquivo .qar

3.

3.1.

```
// CSR
input           iAlignException,          // store e load
input [31:0]    iPC,                      // Instrucao desalinhada ou fora do .text
input [31:0]    wAddress,                 // mem
output          oCSRInstruction,          // Instrucao CSR
output          oCSRException,            // Excecao CSR
//output [31:0]  oUstatus,                 // Excecao/Interrupção - status
output [31:0]   oUcause,                  // Causa da excecao
output [1:0]    oOrigExcecao,             // Controle Mux (origPC, utvect, uepc)
// break
output          oBreakInstr               // Instrução ebreak
```

Figura 4. Código Verilog das nova saídas e entradas do controle

Para a confecção do RV32IMF com tratamento de exceção foi necessário adicionar um banco de registradores chamado CSR (Control and Status Registers), com os registradores USTATUS, FFLAGS, FRM, FCSR, UI, UTVECT, USCRATCH, UEPC, UCAUSE, UTVAL e UIP, e adicionar um MUX (de sinal wCSSR = wCOrigExcecao) em cascata com o

MUX OrigPC no caminho de dados. Além disso, foi necessário fazer algumas mudanças no controle para que ele gerasse os sinais que vão para o CSR. Assim, foram adicionados e definidos os sinais e saída oUcause, oCSRInstruction (indica se a instrução executada é trabalhada pelo CSR), oCSRException (indica se está ocorrendo uma exceção), oOrigExcecao (define a saída do MUX: PC original, Utvect ou Uepc), oBreakInstr (define se haverá uma pausa), assim como os seus inputs iAlignExeption (indica se o store ou o load estão desalinhados), iPC (para verificar se o endereço da instrução está desalinhado ou é inválido) e iAddress (para verificar o endereço que sai da ULA em loads e stores) de forma que a CSR possa executar os procedimentos necessários para tratar a exceção. Normalmente, quando exceções acontecem, as escritas no banco de registradores em inteiro e no banco de registradores em ponto flutuante são desativadas, assim como a escrita e leitura da memória de dados.

Para as funções implementadas foram adicionados aos Parâmetros do projeto os funct7 do EBREAK (0000000) e alterado o funct7 do URET para 0000000, os funct3 do EBREK, URET, CSRRW, CSRRS, CSRRC, CSRRWI, CSRRSI, CSRRCI e os RS2 do ECALL, URET e EBREAK. Para verificar o segmento MMIO, foi adicionado o parâmetro END_IODEVICES, que indica aonde termina o MMIO.

No CPU, foram adicionados os sinais de monitoramento mCSRVGARead (que mostra o que está escrito em um registrador selecionado pelo mVGAselect), mExcecao (que mostra Ucause) e oBreakInstr(sinal que vai para o break interface para parar o clock), todos como outputs. Foram criados os wires wCSRInstruction, wCSRException, wUcause e wCOrigExcecao que saem do controle. Foram criados também os fios wPC e wMemAlignException que se tornam iPC e iAlignException.

```
always @(*)
    // endereço de instrução desalinhado
    if (iPC[1] == 1'b1 || iPC[0] == 1'b1)
    begin
        oUcause <= 32'd0;
        oCSRInstruction <= 1'b0;
        oCSRException <= 1'b1;
        oOrigExcecao  <= 2'b01;      // pc = utvect
        oBreakInstr <= 1'b0;

        oOrigAULA   <= 1'b0;
        oOrigBULA   <= 1'b0;
        oRegWrite   <= 1'b0;
        oMemWrite   <= 1'b0;
        oMemRead    <= 1'b0;
        oALUControl <= OPNULL;
        oMem2Reg    <= 2'b00;
        oOrigPC     <= 2'b00;
`ifdef RV32IMF
        oFPALU2Reg    <= 1'b0;
        oFPALUControl <= OPNULL;
        oFRegWrite    <= 1'b0;
        oOrigAFPALU   <= 1'b0;
        oFWriteData   <= 1'b0;
        oWrite2Mem    <= 1'b0;
        oFPstart      <= 1'b0;
`endif
    end
```

Figura 5. Código Verilog para endereço de instrução desalinhado

```
    // endereço fora do .text
    else if ((iPC < BEGINNING_TEXT) || (iPC > END_TEXT))
    begin
        oUcause <= 32'd1;
        oCSRInstruction <= 1'b0;
        oCSRException <= 1'b1;
        oOrigExcecao  <= 2'b01;      // pc = utvect
        oBreakInstr <= 1'b0;

        oOrigAULA   <= 1'b0;
        oOrigBULA   <= 1'b0;
        oRegWrite   <= 1'b0;
        oMemWrite   <= 1'b0;
        oMemRead    <= 1'b0;
        oALUControl <= OPNULL;
        oMem2Reg    <= 2'b00;
        oOrigPC     <= 2'b00;
`ifdef RV32IMF
        oFPALU2Reg    <= 1'b0;
        oFPALUControl <= OPNULL;
        oFRegWrite    <= 1'b0;
        oOrigAFPALU   <= 1'b0;
        oFWriteData   <= 1'b0;
        oWrite2Mem    <= 1'b0;
        oFPstart      <= 1'b0;
`endif
    end
```

Figura 6. Código Verilog para endereço de instrução fora do segmento .text

Agora o controle verifica se o endereço de instrução não está desalinhado, observando se os dois últimos bits são diferentes de 0 e verifica se o endereço de instrução não está fora do segmento .text.

```
        default: // instrucao invalida
            begin
                oUcause <= 32'd2;

                oCSRInstruction <= 1'b0;
                oCSRException <= 1'b1;
                oOrigExcecao        <= 2'b01;       // pc = utvect
                oBreakInstr <= 1'b0;

                oOrigAULA   <= 1'b0;
                oOrigBULA   <= 1'b0;
                oRegWrite   <= 1'b0;
                oMemWrite   <= 1'b0;
                oMemRead    <= 1'b0;
                oALUControl <= OPNULL;
                oMem2Reg    <= 2'b00;
                oOrigPC     <= 2'b00;
`ifdef RV32IMF
                oFPALU2Reg    <= 1'b0;
                oFPALUControl <= OPNULL;
                oFRegWrite    <= 1'b0;
                oOrigAFPALU   <= 1'b0;
                oFWriteData   <= 1'b0;
                oWrite2Mem    <= 1'b0;
                oFPstart      <= 1'b0;
`endif
        end
```

Figura 7. Código Verilog para instrução inválida

Caso a instrução não seja reconhecida, seja pelo opcode, funct7, funct3 ou rs2, o controle define os sinais adicionados para tratar a exceção.

```
                // endereço desalinhado - load
                if (iAlignException == 1'b1)
                begin
                    oUcause         <= 32'd4;
                    oCSRException   <= 1'b1;
                    oOrigExcecao        <= 2'b01;       // pc = utvec
                    oRegWrite   <= 1'b0;
                    oMemWrite   <= 1'b0;
                    oMemRead    <= 1'b0;
`ifdef RV32IMF
                    oFRegWrite    <= 1'b0;
`endif
                end
```

Figura 8. Código Verilog para endereço de load desalinhado

Quando o endereço da saída da ULA está desalinhado em uma instrução load ou load de ponto flutuante (flw), os sinais acima são definidos e os outros sinais são mantidos como se a função estivesse sendo executada normalmente.

```
            // endereço fora .data - load e  fpload
            else if (((wAddress < BEGINNING_IODEVICES) || (wAddress > END_IODEVICES))
                    && ((wAddress < BEGINNING_DATA) || (wAddress > END_DATA)))
            begin
                oUcause         <= 32'd5;
                oCSRException   <= 1'b1;
                oOrigExcecao        <= 2'b01;       // pc = utvec

                oRegWrite   <= 1'b0;
                oMemWrite   <= 1'b0;
                oMemRead    <= 1'b0;
`ifdef RV32IMF
                oFRegWrite    <= 1'b0;
`endif
            end
```

Figura 9. Código Verilog para endereço de load fora do segmento .data

Para um load ou um load de ponto flutuante (flw), verificamos se o endereço que sai da ULA está dentro do segmento .data ou dentro do segmento MMIO. Caso o endereço não esteja, os sinais acima são definidos e os outros ficam como seriam definidos pela instrução em questão.

```
            begin
                // endereço desalinhado - store
                if (iAlignException == 1'b1)
                begin
                    oUcause         <= 32'd6;
                    oCSRException   <= 1'b1;
                    oOrigExcecao    <= 2'b01;       // pc = utvect

                    oRegWrite   <= 1'b0;
                    oMemWrite   <= 1'b0;
                    oMemRead    <= 1'b0;
`ifdef RV32IMF
                    oFRegWrite    <= 1'b0;
`endif
                end
```

Figura 10. Código Verilog para endereço de store desalinhado

Quando um store ou store em ponto flutuante (fsw) é realizado, verificamos se o endereço que sai da ULA está desalinhado; caso esteja, os sinais acima são definidos e os remanescentes são mantidos de acordo com a instrução.

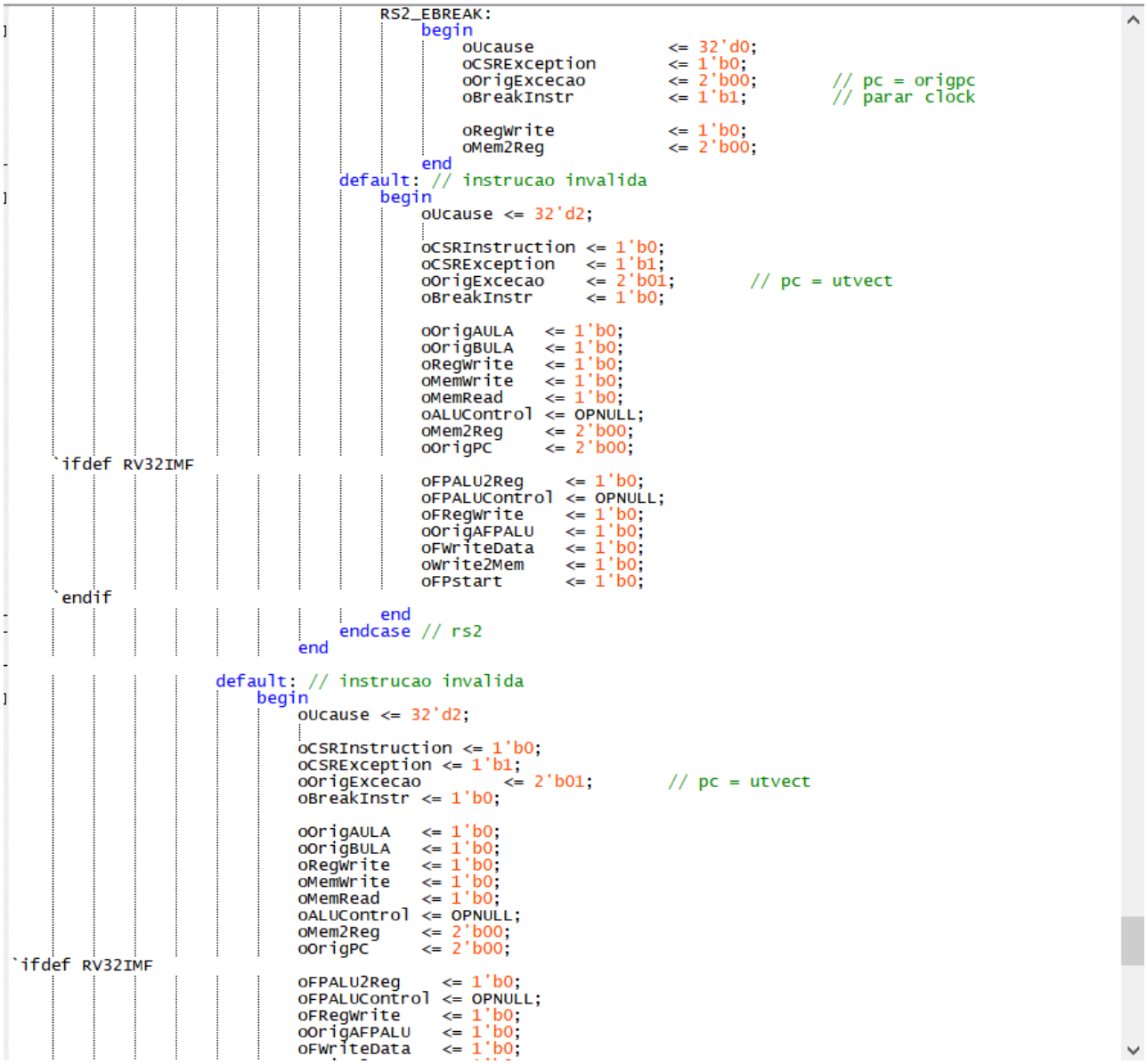
```
                // endereço fora .data - store
                else if ((wAddress < BEGINNING_IODEVICES) || (wAddress > END_IODEVICES))
                    && ((wAddress < BEGINNING_DATA) || (wAddress > END_DATA)))
                begin
                    oUcause         <= 32'd7;
                    oCSRException   <= 1'b1;
                    oOrigExcecao    <= 2'b01;       // pc = utvect

                    oRegWrite   <= 1'b0;
                    oMemWrite   <= 1'b0;
                    oMemRead    <= 1'b0;
`ifdef RV32IMF
                    oFRegWrite    <= 1'b0;
`endif
                end
```

Figura 11. Código Verilog para endereço de store fora do segmento .data

Caso um store ou store em ponto flutuante (fsw) seja executado, também verificamos se o endereço que sai da ULA está fora do segmento .data e fora do segmento MMIO ou não. Se sim, os sinais acima são definidos dessa forma e os remanescentes permanecem da mesma forma como seriam definidos pela função em questão.

```
        OPC_CSR,
        OPC_URET,
        OPC_ECALL,
        OPC_EBREAK:
            begin
                //oUcause           <= 32'd0;
                //oCSRException     <= 1'b0;
                //oCSRInstruction   <= 1'b0;
                //oOrigExcecao      <= 2'b00;       // pc = origpc
                //oBreakInstr   <= 1'b0;

                oOrigAULA   <= 1'b0;
                oOrigBULA   <= 1'b0;
                //oRegWrite <= 1'b0;
                oMemWrite   <= 1'b0;
                oMemRead    <= 1'b0;
                oALUControl <= OPNULL;
                //oMem2Reg  <= 2'b00;
                oOrigPC     <= 2'b00;
`ifdef RV32IMF
                oFPALU2Reg    <= 1'b0;
                oFPALUControl <= OPNULL;
                oFRegWrite    <= 1'b0;
                oOrigAFPALU   <= 1'b0;
                oFWriteData   <= 1'b0;
                oWrite2Mem    <= 1'b0;
                oFPstart      <= 1'b0;
`endif
                case (Funct3)
                FUNCT3_ECALL,
                FUNCT3_EBREAK,
                FUNCT3_URET:
                    case (Funct7)
                        FUNCT7_ECALL,
                        FUNCT7_URET,
                        FUNCT7_EBREAK:
                            begin
                                oCSRInstruction     <= 1'b0;

                                case (Rs2)
                                    RS2_ECALL:
                                        begin
                                            oUcause         <= 32'd8;
                                            oCSRException       <= 1'b1;
                                            oOrigExcecao        <= 2'b01;       // pc = utvect
                                            oBreakInstr <= 1'b0;

                                            oRegWrite   <= 1'b0;
                                            oMem2Reg    <= 2'b00;
                                        end

                                    RS2_URET:
                                        begin
                                            oUcause         <= 32'd0;
                                            oCSRException       <= 1'b0;
                                            oOrigExcecao        <= 2'b10;       // pc = uepc
                                            oBreakInstr <= 1'b0;

                                            oRegWrite   <= 1'b0;
                                            oMem2Reg    <= 2'b00;
                                        end
```

Figura 12. Código Verilog para as intruções implementadas



```
                    RS2_EBREAK:
                        begin
                            oUcause             <= 32'd0;
                            oCSRException       <= 1'b0;
                            oOrigExcecao        <= 2'b00;       // pc = origpc
                            oBreakInstr         <= 1'b1;        // parar clock

                            oRegWrite           <= 1'b0;
                            oMem2Reg            <= 2'b00;
                        end
                    default: // instrucao invalida
                        begin
                            oUcause <= 32'd2;

                            oCSRInstruction <= 1'b0;
                            oCSRException   <= 1'b1;
                            oOrigExcecao    <= 2'b01;       // pc = utvect
                            oBreakInstr     <= 1'b0;

                            oOrigAULA   <= 1'b0;
                            oOrigBULA   <= 1'b0;
                            oRegWrite   <= 1'b0;
                            oMemWrite   <= 1'b0;
                            oMemRead    <= 1'b0;
                            oALUControl <= OPNULL;
                            oMem2Reg    <= 2'b00;
                            oOrigPC     <= 2'b00;
    `ifdef RV32IMF
                            oFPALU2Reg    <= 1'b0;
                            oFPALUControl <= OPNULL;
                            oFRegWrite    <= 1'b0;
                            oOrigAFPALU   <= 1'b0;
                            oFWriteData   <= 1'b0;
                            oWrite2Mem    <= 1'b0;
                            oFPstart      <= 1'b0;
    `endif
                        end
                    endcase // rs2
                end

            default: // instrucao invalida
                begin
                    oUcause <= 32'd2;

                    oCSRInstruction <= 1'b0;
                    oCSRException <= 1'b1;
                    oOrigExcecao        <= 2'b01;       // pc = utvect
                    oBreakInstr <= 1'b0;

                    oOrigAULA   <= 1'b0;
                    oOrigBULA   <= 1'b0;
                    oRegWrite   <= 1'b0;
                    oMemWrite   <= 1'b0;
                    oMemRead    <= 1'b0;
                    oALUControl <= OPNULL;
                    oMem2Reg    <= 2'b00;
                    oOrigPC     <= 2'b00;
`ifdef RV32IMF
                    oFPALU2Reg    <= 1'b0;
                    oFPALUControl <= OPNULL;
                    oFRegWrite    <= 1'b0;
                    oOrigAFPALU   <= 1'b0;
                    oFWriteData   <= 1'b0;
```

Figura 13. Código Verilog para as intruções implementadas

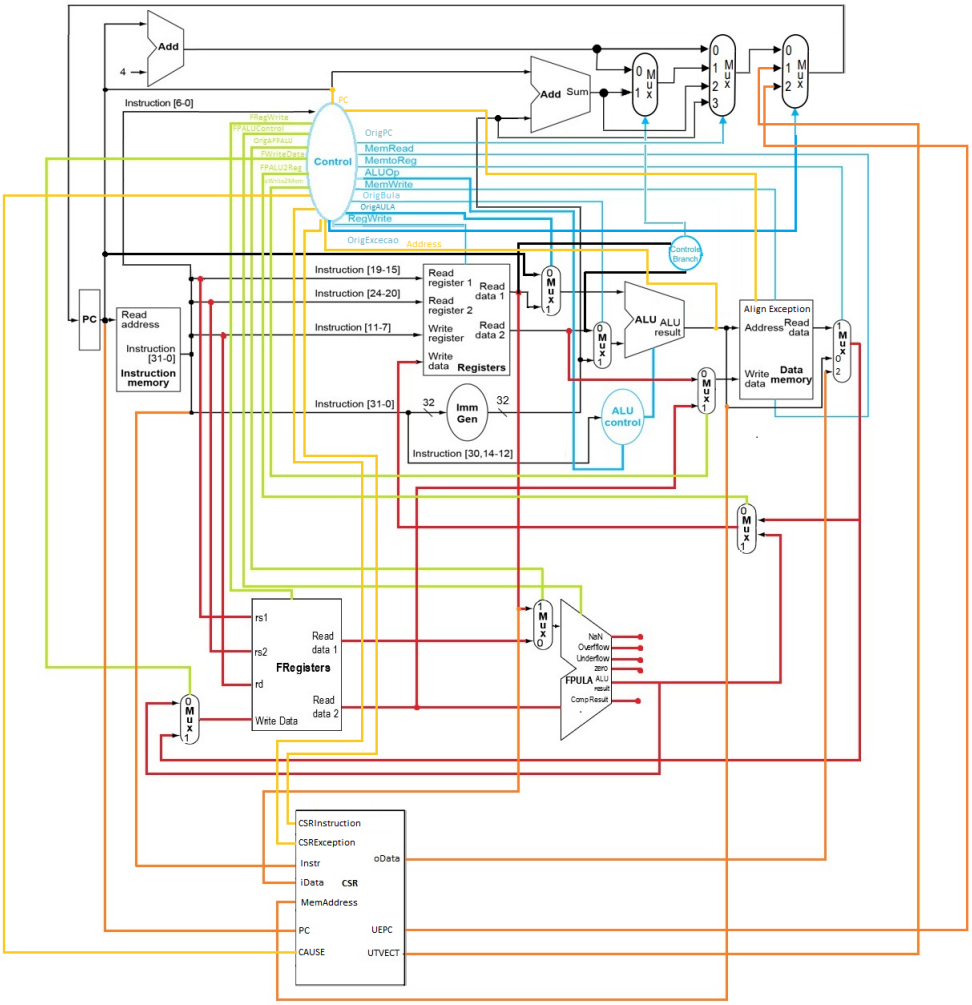
```
                    oWrite2Mem    <= 1'b0;
                    oFPstart      <= 1'b0;
`endif
                end
            endcase // funct7

        FUNCT3_CSRRW,
        FUNCT3_CSRRS,
        FUNCT3_CSRRC,
        FUNCT3_CSRRWI,
        FUNCT3_CSRRSI,
        FUNCT3_CSRRCI:
            begin
                oUcause         <= 32'd0;
                oCSRInstruction <= 1'b1;
                oCSRException   <= 1'b0;
                oOrigExcecao    <= 2'b00;       // pc = origpc
                oBreakInstr <= 1'b0;

                oRegWrite   <= 1'b1;
                oMem2Reg    <= 2'b11;           // reg = CSRData
            end

        default: // instrucao invalida
            begin
                oUcause <= 32'd2;

                oCSRInstruction <= 1'b0;
                oCSRException <= 1'b1;
                oOrigExcecao        <= 2'b01;       // pc = utvect
                oBreakInstr <= 1'b0;

                oOrigAULA   <= 1'b0;
                oOrigBULA   <= 1'b0;
                oRegWrite   <= 1'b0;
                oMemWrite   <= 1'b0;
                oMemRead    <= 1'b0;
                oALUControl <= OPNULL;
                oMem2Reg    <= 2'b00;
                oOrigPC     <= 2'b00;
`ifdef RV32IMF
                oFPALU2Reg    <= 1'b0;
                oFPALUControl <= OPNULL;
                oFRegWrite    <= 1'b0;
                oOrigAFPALU   <= 1'b0;
                oFWriteData   <= 1'b0;
                oWrite2Mem    <= 1'b0;
                oFPstart      <= 1'b0;
`endif
            end
        endcase // funct3
    end // opc_ecall
```

Figura 14. Código Verilog para as intruções implementadas

No caso das instruções que foram adicionadas (csrrw, csrrs, csrrc, csrrwi, csrrsi, csrrci, uret, ecall e ebreak), as instruções foram feitas da maneira acima, com as outputs oOrigAula, oOrigBula, oMemWrite, oMemRead, oAluControl, oOrigPc, oFPALU2Reg, oFPALUControl, oFRegWrite, oOrigAFPALU, oFWriteData, oWrite2Mem, oFPstart sendo definidas igualmente para elas, enquanto que as outputs oUcause, oCSRException, oOrigExcecao, oBreakInstr, oRegWrite e oMem2Reg são definidas para cada instrução com mais especificidade de acordo com o Funct3, Funct7 e RS2 de cada função.

3.2.

Figura 15. Caminho de Dados da RV32IMF



3.3.

Para confeccionar a tabela verdade do controle, classificamos as tabelas em tabelas de instruções antigas (cor vermelha), tabela de instruções novas (cor azul) e tabela de exceções, que inclui o ecall já que além de uma instrução é considerado uma exceção (cor laranja).

Figura 16. Tabela do controle - instruções

Instrução	oALUControl	oOrigAULA	oOrigBULA	oRegWrite	oMemWrite	oMemRead	oMem2Reg	oOrigPC	oWrite2Mem	oFPALU2Reg	oFPALUControl	oFRegWrite	oOrigAFPALU	oFWriteData	oFPstart	CSRInstruction	CSRException	Ucause	OrigExcecao	BreakInstr
load	OPADD	0	1	1	0	1	10	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
addi	OPADD	0	1	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
slli	OPSLL	0	1	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
slti	OPSLT	0	1	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
sltiu	OPSLTU	0	1	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
xori	OPXOR	0	1	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
srli	OPSRL	0	1	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
srai	OPSRA	0	1	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
ori	OPOR	0	1	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
andi	OPAND	0	1	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
auipc	OPADD	1	1	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
store	OPADD	0	1	0	1	0	00	00	0	0	OPNULL	0	X	X	0	0	0	0	00	0
add	OPADD	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
sub	OPSUB	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
sll	OPSLL	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
slt	OPSLT	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
sltu	OPSLTU	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
xor	OPXOR	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
srl	OPSRL	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0
sra	OPSRA	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	00	0

Figura 17. Tabela do controle - instruções

Instrução	oALUControl	oOrigAULA	oOrigBULA	oRegWrite	oMemWrite	oMemRead	oMem2Reg	oOrigPC	oWrite2Mem	oFPALU2Reg	oFPALUControl	oFRegWrite	oOrigAFPALU	oFWriteData	oFPstart	CSRInstruction	CSRException	Ucause	OrigExcecao	BreakInstr
or	OPOR	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
and	OPAND	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
mul	OPMUL	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
mulh	OPMULH	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
mulhsu	OPMULHSU	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
mulhu	OPMULHU	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
div	OPDIV	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
divu	OPDIVU	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
rem	OPREM	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
remu	OPREMU	0	0	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
lui	OPLUI	0	1	1	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
branch	OPADD	0	0	0	0	0	00	01	X	0	OPNULL	0	X	X	0	0	0	0	0	0
jalr	OPADD	0	0	1	0	0	01	11	X	0	OPNULL	0	X	X	0	0	0	0	0	0
jal	OPADD	0	0	1	0	0	01	10	X	0	OPNULL	0	X	X	0	0	0	0	0	0
nop	OPNULL	0	0	0	0	0	00	00	X	0	OPNULL	0	X	X	0	0	0	0	0	0
fadd.s	OPNULL	X	X	0	0	0	XX	00	X	X	FOPADD	1	1	0	1	0	0	0	0	0
fsub.s	OPNULL	X	X	0	0	0	XX	00	X	X	FOPSUB	1	1	0	1	0	0	0	0	0
fmul.s	OPNULL	X	X	0	0	0	XX	00	X	X	FOPMUL	1	1	0	1	0	0	0	0	0
fdiv.s	OPNULL	X	X	0	0	0	XX	00	X	X	FOPDIV	1	1	0	1	0	0	0	0	0
fsqrt.s	OPNULL	X	X	0	0	0	XX	00	X	X	FOPSQRT	1	1	0	1	0	0	0	0	0

Figura 18. Tabela do controle - instruções

Instrução	oALUControl	oOrigAULA	oOrigBULA	oRegWrite	oMemWrite	oMemRead	oMem2Reg	oOrigPC	oWrite2Mem	oFPALU2Reg	oFPALUControl	oFRegWrite	oOrigAFPALU	oFWriteData	oFPstart	CSRInstruction	CSRException	Ucause	OrigExcecao	BreakInstr
fcvt.s.w	OPNULL	X	X	0	0	0	XX	00	X	X	FOPCVTSW	1	0	0	1	0	0	0	0	0
fcvt.s.wu	OPNULL	X	X	0	0	0	XX	00	X	X	FOPCVTSWU	1	0	0	1	0	0	0	0	0
fcvt.w.s	OPNULL	X	X	1	0	0	XX	00	X	1	FOPCVTWS	0	1	0	1	0	0	0	0	0
fcvt.wu.s	OPNULL	X	X	1	0	0	XX	00	X	1	FOPCVTWUS	0	1	0	1	0	0	0	0	0
fmv.s.x	OPNULL	X	X	0	0	0	XX	00	X	X	FOPMV	1	0	0	1	0	0	0	0	0
fmv.x.s	OPNULL	X	X	1	0	0	XX	00	X	1	FOPMV	0	1	0	1	0	0	0	0	0
feq.s	OPNULL	X	X	1	0	0	XX	00	X	1	FOPCEQ	0	1	0	1	0	0	0	0	0
fle.s	OPNULL	X	X	1	0	0	XX	00	X	1	FOPCLE	0	1	0	1	0	0	0	0	0
flt.s	OPNULL	X	X	1	0	0	XX	00	X	1	FOPCLT	0	1	0	1	0	0	0	0	0
fsgnj	OPNULL	X	X	0	0	0	XX	00	X	X	FOPSGNJ	1	1	0	1	0	0	0	0	0
fsgnjn	OPNULL	X	X	0	0	0	XX	00	X	X	FOPSGNJN	1	1	0	1	0	0	0	0	0
fsgnjx	OPNULL	X	X	0	0	0	XX	00	X	X	FOPSGNJX	1	1	0	1	0	0	0	0	0
fmax	OPNULL	X	X	0	0	0	XX	00	X	X	FOPMAX	1	1	0	1	0	0	0	0	0
fmin	OPNULL	X	X	0	0	0	XX	00	X	X	FOPMIN	1	1	0	1	0	0	0	0	0
flw	OPADD	0	1	0	0	1	10	00	X	X	OPNULL	1	X	1	0	0	0	0	0	0
fsw	OPADD	0	1	0	1	0	XX	00	1	X	OPNULL	0	X	X	0	0	0	0	0	0

Nas imagens acima temos as tabelas das instruções com todos os sinais definidos pelo controle.

Figura 19. Tabela do controle - instruções CSRR, ebrea e uret

Instrução	oALUControl	oOrigAULA	oOrigBULA	oRegWrite	oMemWrite	oMemRead	oMem2Reg	oOrigPC	oWrite2Mem	oFPALU2Reg	oFPALUControl	oFRegWrite	oOrigAFPALU	oFWriteData	oFPstart	CSRInstruction	CSRException	Ucause	OrigExcecao	BreakInstr
csrrw	OPNULL	0	0	1	0	0	00	00	0	0	OPNULL	0	0	0	0	1	0	0	00	0
csrrs	OPNULL	0	0	1	0	0	00	00	0	0	OPNULL	0	0	0	0	1	0	0	00	0
csrrc	OPNULL	0	0	1	0	0	00	00	0	0	OPNULL	0	0	0	0	1	0	0	00	0
csrrwi	OPNULL	0	0	1	0	0	00	00	0	0	OPNULL	0	0	0	0	1	0	0	00	0
csrrsi	OPNULL	0	0	1	0	0	00	00	0	0	OPNULL	0	0	0	0	1	0	0	00	0
csrrci	OPNULL	0	0	1	0	0	00	00	0	0	OPNULL	0	0	0	0	1	0	0	00	0
uret	OPNULL	0	0	0	0	0	00	00	0	0	OPNULL	0	0	0	0	0	0	0	10	0
ebreak	OPNULL	0	0	0	0	0	00	00	0	0	OPNULL	0	0	0	0	0	0	0	00	1

Figura 20. Tabela do controle - Exceções

Exceção	oALUControl	oOrigAULA	oOrigBULA	oRegWrite	oMemWrite	oMemRead	oMem2Reg	oOrigPC	oWrite2Mem	oFPALU2Reg	oFPALUControl	oFRegWrite	oOrigAFPALU	oFWriteData	oFPstart	CSRInstruction	CSRException	Ucause	OrigExcecao	BreakInstr
Endereço da instrução desalinhado	OPNULL	0	0	0	0	0	00	00	0	0	OPNULL	0	0	0	0	0	1	0	01	0
Endereço da instrução fora do .text	OPNULL	0	0	0	0	0	00	00	0	0	OPNULL	0	0	0	0	0	1	1	01	0
Instrução inválida	OPNULL	0	0	0	0	0	00	00	0	0	OPNULL	0	0	0	0	0	1	2	01	0
Endereço desalinhado - Load e fpload	OPADD	0	1	0	0	0	10	00	0	0	OPNULL	0	0	0	0	0	1	4	01	0
Endereço fora do .data - Load e fpload	OPADD	0	1	0	0	0	10	0	0	0	OPNULL	0	0	0	0	0	1	5	01	0
Endereço desalinhado - Store e fpstore	OPADD	0	1	0	0	0	00	00	0	0	OPNULL	0	0	0	0	0	1	6	01	0
Endereço fora do .data - store e fpstore	OPADD	0	1	0	0	0	00	00	0	0	OPNULL	0	0	0	0	0	1	7	01	0

A tabela acima possui o ecall pois além de instrução é considerado uma exceção.

3.4. As dificuldades que enfrentamos foram: decidir se deveríamos ou não criar uma unidade de controle e execução das instruções que deveríamos implementar, pois, optando por criar essa unidade estaríamos criando um circuito que iria alterar os sinais de controle, assim como o Controle e, após implementar dessa forma e encarar diversos problemas de loop combinacionais optamos por criar um banco de registradores (CSR - Constrol and Status Registers) e deixar com que o Controle detectasse as exceções e instruções de acordo com os sinais de entrada que iria receber; outra dificuldade que tivemos foi com a instrução store byte, pois essa gerava uma exceção sempre que a executávamos, porém resolvemos esse problema corrigindo

4. Link : `https://youtu.be/oGIRmjKQ1Ko`

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 implentação, 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/WiuGnnBHDrw`

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 implentação, 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/BdlYALJPPP4`

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/tI2RNkoaKWw`