**Grupo 12 - OAC - 2021.2**
**Laboratório 4**
Eduardo Ferreira Marques Cavalcante - 202006368
Gabriel Mendes Ciriatico Guimarães - 202033202
Gustavo Lopes Dezan - 202033463

## CPU µRISC-V MULTICICLO

***1.1)*** *Construa o caminho de dados completo e identifique os sinais de controle. Defina o Diagrama de Estados do Bloco de Controle e projete um circuito que o implemente;*

Para montar o processador multiciclo, foi utilizado os componentes do processador uniciclo com pequenas alterações. A primeira coisa feita para a construção do caminho de dados foi adicionar registradores responsáveis por armazenar os dados de cada operação que pode mudar com mudança de ciclo.

Foram adicionados os registradores: R_A, que armazena o dado do registrador A do banco de registradores; R_B, do registrador B do banco de registradores; SaidaULA, registrador ligado à saída da ULA; PC_Back, que armazena o valor inicial de PC no estado 0; SaidaShifter, que armazena o valor de saída do shifter; e SaidaComp, ligado à saída do comparador.

A memória também precisou ser alterada, já que agora foi utilizado um bloco único de memória, tanto para a memória de instruções quanto para a de dados, como na Arquitetura de von Neumann. Dentro do bloco, no entanto, ainda foi utilizada uma memória ROM para a memória de instruções e uma memória RAM para a memória de dados. A saída desse bloco foi definida através de um MUX ligado a um comparador que checa se o endereço requisitado está no intervalo da memória de dados ou de instrução.

Na saída da memória, também foram ligados 2 registradores: R_Inst, que guarda a instrução; e R_Dados, que guarda o dado da memória de dados.

Outra mudança significativa no processador foi a eliminação do somador, sendo agora todas as operações feitas pela ULA, inclusive operações como PC + 4 e PC + Imm. Para lidar com isso, foi necessário adicionar um novo sinal de controle à unidade de controle, OP_ULA, que faz com que a ULA faça uma soma independente da entrada.

Novos MUX também foram adicionados ao processador para lidar com essas questões. A entrada A e B da ULA, por exemplo, passou a ser definida por MUX, o que também aumentou o tamanho da unidade de controle.

No fim, foram definidos 25 sinais de controle saindo da controladora geral. A tabela detalhada pode ser vista abaixo:

**Tabela 1.** Sinais de controle do processador multiciclo

| **Sinal** | **Valor** | **Explicação** | **Controla** |
|---|---|---|---|
| E_PC | 0 | Não habilita a escrita em PC | Permite que PC seja alterado ou permaneça no |

| | | | |
|---|---|---|---|
| | 1 | Habilita a escrita em PC | mesmo valor |
| E_PC_Back | 0 | Não habilita a escrita no registrador PC_Back | Controla a escrita no registrador PC_Back, que guarda o valor de PC |
| | 1 | Habilita a escrita no registrador PC_Back | |
| E_R_Inst | 0 | Não habilita a escrita no registrador R_Inst | Controla a escrita no registrador R_Inst, que guarda a instrução |
| | 1 | Habilita a escrita no registrador R_Inst | |
| E_R_Dados | 0 | Não habilita a escrita no registrador R_Dados | Controla a escrita no registrador R_Dados, que guarda o dado lido da memória |
| | 1 | Habilita a escrita no registrador R_Dados | |
| E_R_A | 0 | Não habilita a escrita no registrador R_A | Controla a escrita no registrador R_A, que guarda o valor RA do banco de registradores |
| | 1 | Habilita a escrita no registrador R_A | |
| E_R_B | 0 | Não habilita a escrita no registrador R_B | Controla a escrita no registrador R_A, que guarda o valor RB do banco de registradores |
| | 1 | Habilita a escrita no registrador R_B | |
| E_SaidaULA | 0 | Não habilita a escrita no registrador SaidaULA | Controla a escrita no registrador SaidaULA, que guarda o valor da operação feita na ULA |
| | 1 | Habilita a escrita no registrador SaidaULA | |
| E_SaidaShifter | 0 | Não habilita a escrita no registrador SaidaShifter | Controla a escrita no registrador SaidaULA, que guarda o valor da operação feita no |

| | | | |
|---|---|---|---|
| | 1 | Habilita a escrita no registrador SaidaShifter | shifter |
| E_SaidaComp | 0 | Não habilita a escrita no registrador SaidaComp | Controla a escrita no registrador SaidaComp, que guarda o resultado da comparação feita |
| | 1 | Habilita a escrita no registrador SaidaComp | |
| OP_ULA | 0 | Operação da ULA definida por OPCode, Funct3 e Funct7 | Permite forçar a ULA a fazer soma, independente da instrução dada |
| | 1 | Força ULA a fazer operação de soma | |
| OP_Brench | 0 | Operação dada não é de brench, não sendo possível fazer PC = PC + Imm | Define se operação de brench é permitida, podendo levar a PC = PC + Imm |
| | 1 | Operação é de brench, podendo ocorrer PC = PC + Imm (dependendo de outros fatores junto) | |
| WM | 0 | Não habilita a escrita na memória de dados | Controla a escrita na memória de dados |
| | 1 | Habilita a escrita na memória de dados | |
| RM | 0 | Não habilita a leitura da memória de dados | Controla a leitura da memória de dados |
| | 1 | Habilita a leitura da memória de dados | |
| WE | 0 | Não habilita a escrita em RD no banco de registradores | Controla a escrita no banco de registradores |

| | | | |
|---|---|---|---|
| | 1 | Habilita a escrita em RD no banco de registradores | |
| MUX_PC | 0 | PC = ULA | Define se PC é o que vem da ULA ou do registrador SaidaULA |
| | 1 | PC = SaidaULA | |
| MUX_Shifter | 0 | B = R_B | Define o valor B a ser usado no shifter |
| | 1 | B = Imm | |
| MUX_Comp | 0 | B = R_B | Define o valor B a ser comparado no comparador |
| | 1 | B = Imm | |
| MUX_MEM | 0 | Busca endereço na memória dado por PC | Memória de instruções é buscada pelo endereço dado na SaidaULA, enquanto a de instruções é dada por PC |
| | 1 | Busca endereço na memória dado por SaidaULA | |
| MUX_ULA_A | 00 | A = R_A | Define a entrada A da ULA |
| | 10 | A = PC_Back | |
| | 01 | A = PC | |
| MUX_ULA_B | 00 | B = R_B | Define a entrada B da ULA |
| | 10 | B = Imm | |
| | 01 | B = 4 | |
| MUX_D | 000 | RD = SaidaULA | Define o que é escrito no banco de registradores em RD |
| | 100 | RD = SaidaShifter | |
| | 010 | RD = SaidaComp | |
| | 110 | RD = Imm | |
| | 001 | RD = PC | |
| | 101 | RD = MDR | |

Com todos esses sinais definidos em mão, passou-se então para a elaboração do diagrama da máquina de estados. O processador multiciclo implementado tem 5 etapas: busca da instrução, decodificação e leitura dos registradores, execução e cálculo do endereço,

acesso à memória e conclusão do tipo-R, e conclusão lw. A primeira e segunda etapas são iguais para todos os comandos implementados, sendo representadas pelos estados 0 e 1.

O diagrama da máquina de estados pode ser visto detalhadamente no PDF e no arquivo de flowchart compactados junto ao projeto. Abaixo também é possível ter uma visão geral do diagrama:

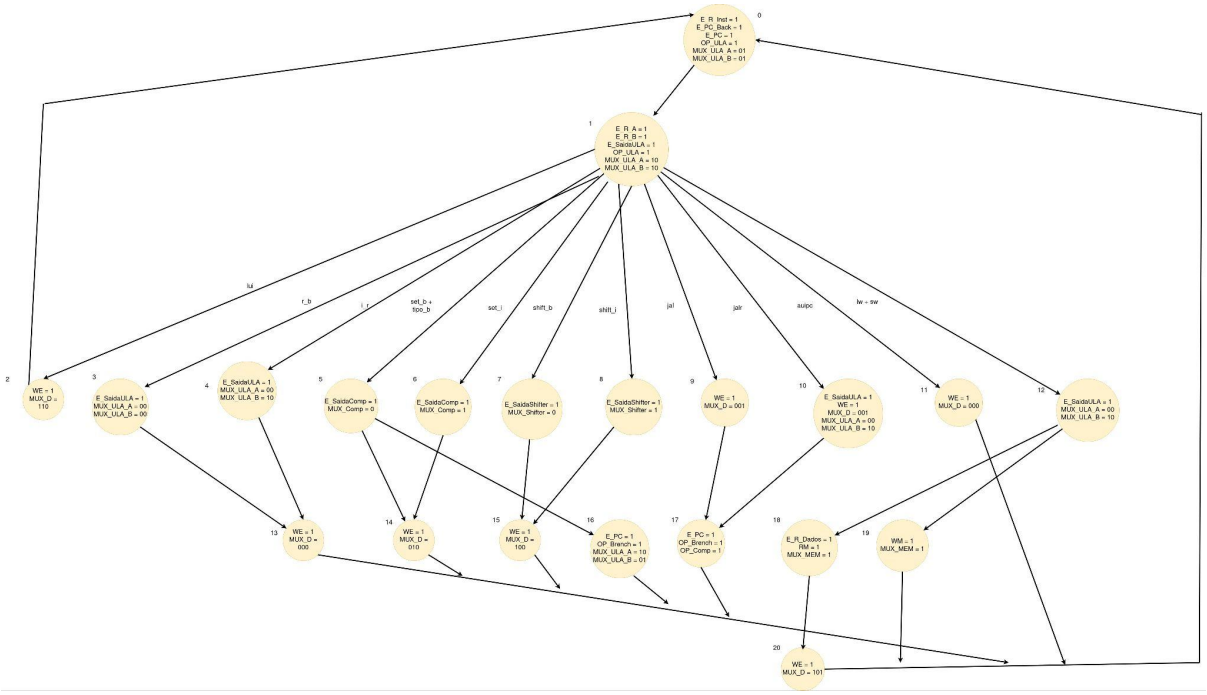



**Figura 1.** Visão geral do diagrama da máquina de estados implementada (visão detalhada pode ser encontrada nos arquivos compactados do projeto).

A tabela abaixo também permite compreender de uma forma mais simples cada estado de cada operação implementada. Observe que há um total de 21 estados na máquina feita:

**Tabela 2.** Estados por instrução da máquina de estados implementada.

| **Número do estado** | **Tipo** | **Comandos** | **Controle** | **Controle (simplificado)** | **Estado (detalhado)** |
|---|---|---|---|---|---|
| 0 | Busca da instrução | Todos | E_R_Inst = 1<br>E_PC_Back = 1<br>E_PC = 1<br>OP_ULA = 1<br>MUX_ULA_A = 01<br>MUX_ULA_B = 01 | ERI = 1<br>EPB = 1<br>EPC = 1<br>OPU = 1<br>MA1 = 1<br>MB1 = 1 | IR = Mem[PC]<br>PCback = PC<br>PC = PC + 4 |

| | | | | | |
|---|---|---|---|---|---|
| 1 | Decodificação, Leitura dos registradores | Todos | E_R_A = 1<br>E_R_B = 1<br>E_SaidaULA = 1<br>OP_ULA = 1<br>MUX_ULA_A = 10<br>MUX_ULA_B = 10 | ERA = 1<br>ERB = 1<br>ESU = 1<br>OPU = 1<br>MA0 = 1<br>MB0 = 1 | A = Reg[IR]<br>B = Reg[IR]<br>SaidaULA = PCBack + Imm |
| 2 | Execução, cálculo do endereço | lui | WE = 1<br>MUX_D = 110 | WE = 1<br>MD0 = 1<br>MD1 = 1 | Reg[IR] = Imm |
| 3 | Execução, cálculo do endereço | add, and, or, sub, xor | E_SaidaULA = 1<br>MUX_ULA_A = 00<br>MUX_ULA_B = 00 | ESU = 1 | SaidaULA = A op B |
| 4 | Execução, cálculo do endereço | addi, andi, ori, xori | E_SaidaULA = 1<br>MUX_ULA_A = 00<br>MUX_ULA_B = 10 | ESU = 1<br>MB0 = 1 | SaidaULA = A op Imm |
| 5 | Execução, cálculo do endereço | slt, sltu, beq, bge, bgeu, blt, bltu, bne | E_SaidaComp = 1<br>MUX_Comp = 0 | ESC = 1 | SaidaComp = A c B |
| 6 | Execução, cálculo do endereço | slti, sltiu | E_SaidaComp = 1<br>MUX_Comp = 1 | ESC = 1<br>MC = 1 | SaidaComp = A c Imm |
| 7 | Execução, cálculo do endereço | sll | E_SaidaShifter = 1<br>MUX_Shifter = 0 | ESS = 1 | SaidaShifter = A s B |
| 8 | Execução, cálculo do endereço | slli | E_SaidaShifter = 1<br>MUX_Shifter = 1 | ESS = 1<br>MS = 1 | SaidaShifter = A s Imm |

| | | | | | |
|---|---|---|---|---|---|
| 9 | Execução, cálculo do endereço | jal | WE = 1<br>MUX_D = 001 | WE = 1<br>MD2 = 1 | Reg[IR] = PC |
| 10 | Execução, cálculo do endereço | jalr | E_SaidaULA = 1<br>WE = 1<br>MUX_D = 001<br>MUX_ULA_A = 00<br>MUX_ULA_B = 10 | ESU = 1<br>WE = 1<br>MD2 = 1<br>MB0 = 1 | SaidaULA = A + Imm<br>Reg[IR] = PC |
| 11 | Execução, cálculo do endereço | auipc | WE = 1<br>MUX_D = 000 | WE = 1 | Reg[IR] = SaidaULA |
| 12 | Execução, cálculo do endereço | lw, sw | E_SaidaULA = 1<br>MUX_ULA_A = 00<br>MUX_ULA_B = 10 | ESU = 1<br>MB0 = 1 | SaidaULA = A + Imm |
| 13 | Acesso à memória, conclsuão tipo-R | add, and, or, sub, xor, addi, andi, ori, xori | WE = 1<br>MUX_D = 000 | WE = 1 | Reg[IR] = SaidaULA |
| 14 | Acesso à memória, conclsuão tipo-R | slt, sltu, slti, sltiu | WE = 1<br>MUX_D = 010 | WE = 1<br>MD1 = 1 | Reg[IR] = SaidaComp |
| 15 | Acesso à memória, conclsuão tipo-R | sll, slli | WE = 1<br>MUX_D = 100 | WE = 1<br>MD0 = 1 | Reg[IR] = SaidaShifter |
| 16 | Acesso à memória, conclsuão tipo-R | beq, bge, bgeu, blt, bltu, bne | E_PC = 1<br>OP_Brench = 1<br>MUX_ULA_A = 10<br>MUX_ULA_B = 01 | E_PC = 1<br>OP_Brench = 1<br>OP_ULA = 1<br>MUX_ULA_A = 10<br>MUX_ULA_B = 01 | EPC = 1<br>OPB = 1<br>OPU = 1<br>MA0 = 1<br>MB1 = 1 |

| | | | | | |
|---|---|---|---|---|---|
| 17 | Acesso à memória, conclsuão tipo-R | jal, jalr | E_PC = 1<br>OP_Brench = 1<br>OP_Comp = 1 | EPC = 1<br>OPB = 1<br>OPC = 1 | PC = SaidaULA |
| 18 | Acesso à memória, conclsuão tipo-R | lw | E_R_Dados = 1<br>RM = 1<br>MUX_MEM = 1 | ERD = 1<br>RM = 1<br>MM = 1 | R_Dados = Mem[SaidaULA] |
| 19 | Acesso à memória, conclsuão tipo-R | sw | WM = 1<br>MUX_MEM = 1 | WM = 1<br>MM = 1 | Mem[SaidaULA] = B |
| 20 | Conclusão lw | lw | WE = 1<br>MUX_D = 101 | WE = 1<br>MD0 = 1<br>MD2 = 1 | Reg[IR] = R_Dados |

Para implementar essa máquina de estados, foi utilizada a PLA (*Programmable Logic Array*). Para isso, cada um dos 21 estados da máquina foi identificado através de portas AND, permitindo verificar a condição para passar ao próximo estado levando em conta o estado atual.

Como é uma Máquina de Mealy, apenas o estado atual não permite definir o próximo estado - é necessário levar em conta a entrada, que no caso é o OPCode, Funct3 e Funct7, dados pela instrução sendo executada. Perceba que é implementado um processador que não segue o modelo hierárquico, já que usa também os valores de Funct3 e Funct7 para definir os sinais de controle.

Os valores das entradas que definem, junto aos estados atuais, os próximos estados são:

tipo_b = beq, bge, bgeu, blt, bltu, bne = O2’*O3’*O4’*O5*O6
lui = O2*O5*O6'
jal = jal = O3*O4’*O5*O6
jalr = jalr = O2*O3'*O4'*O5*O6
auipc = auipc = O2*O3'*O4*O5'
sw = sw = O4'*O5*O6'
lw = lw = O4'*O5'*O6'

Além desses, foi também preciso diferenciar instruções do tipo-R que não fossemslt, sltu ou sll, bem como diferenciar instruções do tipo-I que não fossem sltiu, slti e slli, já que esses comandos vão para estados diferentes entre si.

Com isso, foram elaborados os seguintes circuitos combinacionais:

r = (O2'*O4*O5*O6')
r_minus_1 (slt + sltu) = T0'*T1*T2' + T0*T1*T2' = T1*T2'
r_minus_2 (sll) = T0*T1'*T2'
r_minus = r_minus_1 + r_minus_2

i (- sw e lw) = (O2'*O3'*O4*O5')
i_minus_1 (sltiu, slti) = T0*T1*T2' + T0'*T1*T2' = T1*T2' = r_minus_1
i_minus_2 (slli) = T0*T1'*T2' = r_minus_2
i_minus = i_minus_1 + i_minus_2 = r_minus

Veja que r_minus_1 = i_minus_1 e r_minus_2 = i_minus2. Com esses circuitos, podemos agora separar essas diferentes instruções:

r_b = add, and, or, sub, xor = r*r_minus'
i_r = addi, andi, ori, xori = i*r_minus'
set_b = slt, sltu = r*r_minus_1
set_i = slti, sltiu = i*r_minus_1
shift_b = sll = r*r_minus_2
shift_i = slli = i*r_minus_2

Com todos esses circuitos combinacionais implementados, foi possível passar para o mapeamento dos bits de estado. Como são 21 estados na máquina, foi necessário usar 5 bits, S0…S4. A primeira coisa feita foi mapear qual desses bits estava ativo em cada estado:

0 = 00000
1 = 10000 (S0)
2 = 01000 (S1)
3 = 11000 (S0, S1)
4 = 00100 (S2)
5 = 10100 (S0, S2)
6 = 01100 (S1, S2)
7 = 11100 (S0, S1, S2)
8 = 00010 (S3)
9 = 10010 (S0, S3)
10 = 01010 (S1, S3)
11 = 11010 (S0, S1, S3)
12 = 00110 (S2, S3)
13 = 10110 (S0, S2, S3)
14 = 01110 (S1, S2, S3)
15 = 11110 (S0, S1, S2, S3)
16 = 00001 (S4)
17 = 10001 (S0, S4)
18 = 01001 (S1, S4)

19 = 11001 (S0, S1, S4)
20 = 00101 (S2, S4)

Com essa informação, foi possível definir a passagem de cada estado:

0 -> 1: todos - lui (S0)

1 -> 2: todos (S1)

1 -> 3: r_b (S0, S1)
1 -> 4: i_r (S2)
1 -> 5: set_b + tipo_b (S0, S2)
1 -> 6: set_i (S1, S2)
1 -> 7: shift_b (S0, S1, S2)
1 -> 8: shift_i (S3)
1 -> 9: jal (S0, S3)
1 -> 10: jalr (S1, S3)
1 -> 11: auipc (S0, S1, S3)
1 -> 12: lw + sw (S2, S3)

3 -> 13: 3 (S0, S2, S3)
4 -> 13: 4 (S0, S2, S3)
5 -> 14: set_b (S1, S2, S3)
5 -> 16: tipo_b (S4)
6 -> 14: 6 (S1, S2, S3)
7 -> 15: 7 (S0, S1, S2, S3)
8 -> 15: 8 (S0, S1, S2, S3)
9 -> 17: 9 (S0, S4)
10 -> 17: 10 (S0, S4)
12 -> 18: lw (S1, S4)
12 -> 19: sw (S0, S1, S4)

19 -> 20: 19

Com essas informações, bastou ligar toda a PLA para definir os sinais de estado. Daí, cada sinal de estado S0…S4 pôde ser encontrado como:

- S0 vai para 1 quando: 0 ou (1 e r_b) ou (1 e (set_b + tipo_b)) ou (1 e shift_b) ou (1 e jal) ou (1 e auipc) ou (12 e sw) ou 4 ou 7 ou 8 ou 9 ou 10 ou 3;
- S1 vai para 1 quando: (1 e lui) ou (1 e r_b) ou (1 e set_i) ou (1 e shift_b) ou (1 e jalr) ou (1 e auipc) ou (5 e set_b) ou 6 ou 7 ou 8 ou 12;
- S2 vai para 1 quando: (1 e i_r) ou (1 e (set_b + tipo_b)) ou (1 e set_i) ou (1 e shift_b) ou (1 e (lw + sw)) ou (5 e set_b) ou 3 ou 4 ou 6 ou 7 ou 8 ou 18;
- S3 vai para 1 quando: (1 e shift_i) ou (1 e jal) ou (1 e auipc) ou (1 e (lw + sw)) ou (5 e set_b) ou 3 ou 4 ou 6 ou 7 ou 8 ou (1 e jalr);

- S4 vai para 1 quando: (5 e tipo_b) ou 9 ou 10 ou 12 ou 18.

Com cada estado definido, foi possível identificar os sinais de controle, que são definidos justamente pelo estado atual da máquina. Daí, foi possível identificar os circuitos que definem cada sinal de saída:

E_R_Inst = 0 (ERI)
E_PC_Back = 0 (EPB)
E_R_A = 1 (ERA)
E_R_B = 1 (ERB)
E_PC = 0 | 16 | 17 (EPC)
E_SaidaULA = 1 | 3 | 4 | 10 | 12 (ESU)
E_SaidaComp = 5 | 6 (ESC)
E_SaidaShifter = 7 | 8 (ESS)
E_R_Dados = 18 (ERD)

OP_Brench = 16 | 17 (OPB)
OP_ULA = 0 | 1 | 16 (OPU)
OP_Comp = 17 (OPC)
WE = 2 | 9 | 10 | 11 | 13 | 14 | 15 | 20
WM = 19
RM = 18

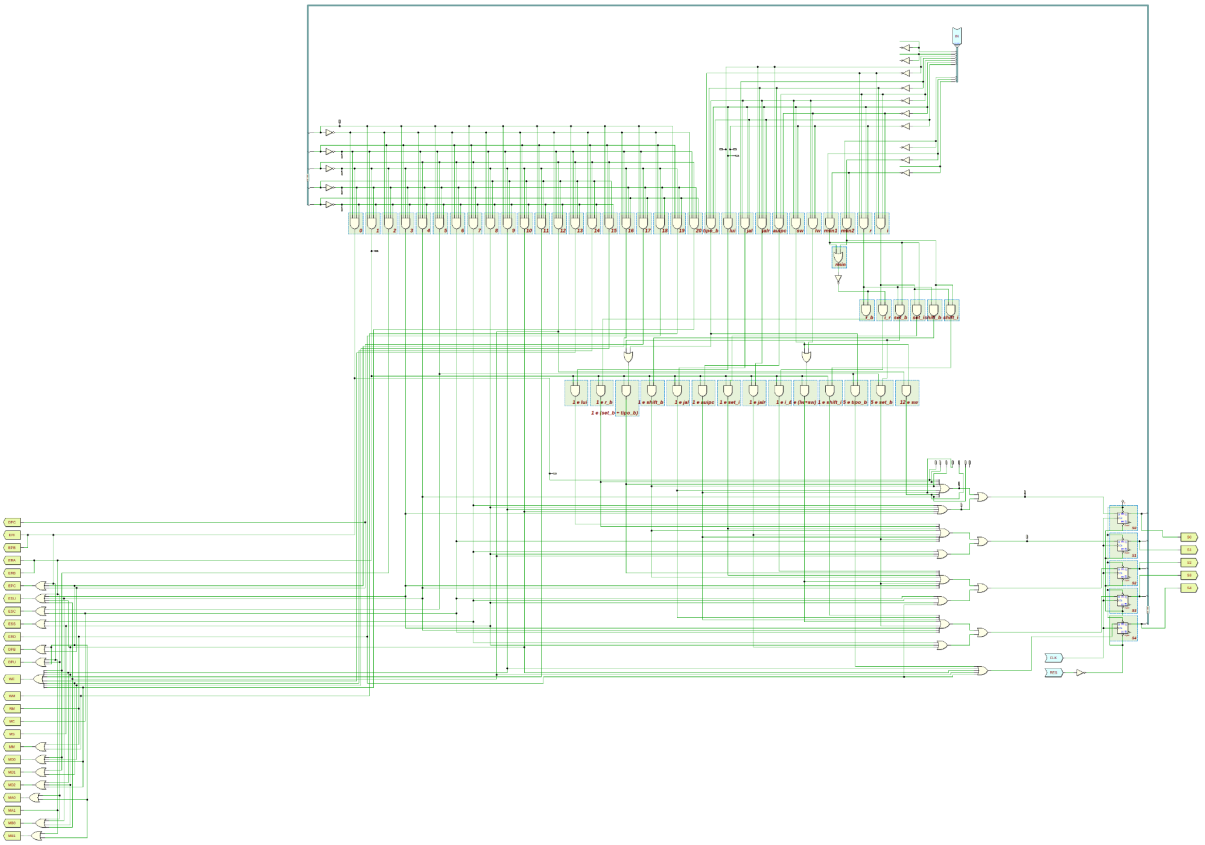
MUX_Comp = 6 (MC)
MUX_Shifter = 8 (MS)
MUX_MEM = 18 | 19 (MM)

MUX_D[0] = 2 | 15 | 20 (MD0)
MUX_D[1] = 2 | 14 (MD1)
MUX_D[2] = 9 | 10 | 20 (MD2)

MUX_ULA_A[0] = 1 | 16 (MA0)
MUX_ULA_A[1] = 0 (MA1)
MUX_ULA_B[0] = 1 | 4 | 10 | 12 (MB0)
MUX_ULA_B[1] = 0 | 16 (MB1)

Abaixo, é possível ter uma visão geral da controladora geral implementada.

**Figura 2.** Visão geral da controladora implementada.



***1.2)*** *Além dos sinais monitorados na CPU Uniciclo, acrescente um display de 7 segmentos que monitore o estado atual do bloco de controle;*

O display foi facilmente implementado, bastando colocar na saída da controladora geral os sinais S0…S4, que mostram o estado atual da máquina.

***1.3)*** *Construa o processador µRISC-V, incluindo uma entrada de clock e um sinal de reset que deve resetar os valores dos registrados do BR e auxiliares, PC=0x00400000 e voltar ao Estado 0 do bloco de controle. Qual a máxima frequência utilizável no seu processador?*

A frequência máxima do processador implementado é de 8.33 MHz vendo o testBench e 6.25 MHz no teste2. Como o teste2.s tem mais uso do comando lw, que tem a etapa mais demorada do processador, é normal essa diferença. Assim, a frequência máxima geral do processador é de 6.25 MHz.

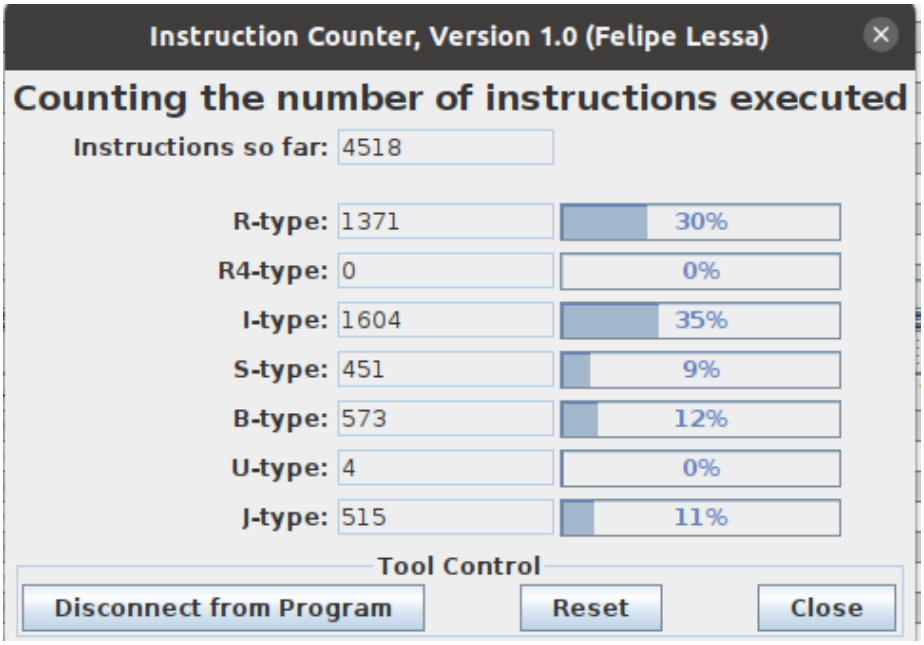
Com isso, foi possível aumentar a frequência máxima do processador do laboratório 3, que era uniciclo e de frequência máxima de 4.0 MHz. Também observe que a frequência máxima permitida na simulação temporal é bastante superior à frequência máxima possível na interface visual normal do Deeds, onde não é possível chegar nem a 100 Hz.

***1.4)*** *Execute o seu programa TestBench.s feito no Laboratório 3 que verifique se todas as instruções foram implementadas corretamente, e filme a sua execução.*

O vídeo rodando o programa TestBench pode ser visto aqui: https://youtu.be/9-x898EK-UI.

***1.5)*** *Faça a simulação e forma de onda na maior frequência possível e filme a execução do programa Teste.s no seu processador. Qual a CPI média deste programa? Qual o tempo necessário à sua execução? Verifique se texec = I × CPI × T. Justifique.*

Para calcular a CPI média do programa teste2.s, podemos primeiro utilizar o contador de instruções do RARS para ver os diferentes tipos de instruções usadas nesse worbench. A partir dessa informação, é possível utilizar a máquina de estados e ver quantos ciclos cada instrução precisa para ser executada.




Primeiro, observe que o número de instruções executadas pelo processador, obviamente, bate com o número de instruções desse quadro do RARS. Ora, de cada um desses grupos, foram utilizados os seguintes comandos (entre parênteses, o número de ciclos para a instrução ser executada:

- Tipo-R: add (4);
- Tipo-I: addi (4), jalr (4), lw (5), slli (4);
- Tipo-S: sw (4);
- Tipo-B: beq (4), bge (4), blt (4);
- Tipo-U: auipc (3), lui (3);
- Tipo-J: jal (4).

Observe que no tipo-I temos um problema, já que lw precisa de 5 ciclos para ser executada e o restante, apenas 4. Como temos apenas a informação para o tipo-I,

aproximemos para o tipo-I para 4.5 ciclos necessário por comando. Com isso, temos que cada tipo utilizado no programa precisa de:

- Tipo-R: 4 ciclos;
- Tipo-I: 4.5 ciclos;
- Tipo-S: 4 ciclos;
- Tipo-B: 4 ciclos;
- Tipo-U: 3 ciclos;
- Tipo-J: 4 ciclos.

Para calcular a CPI média, basta fazer o cálculo:

CPI Média = Tipo-R*0.3 + tipo-I*0.35 + tipo-U*0 + tipo-S*0.09 + tipo-B*0.12 + tipo-J*0.11 = 4*0.3 + 4.5*0.35 + 3*0 + 4*0.09 + 4*0.12 + 4*0.11 = 4.055

Utilizando CPI = CPI Média e T = 1/(6.25*10^6), temos:

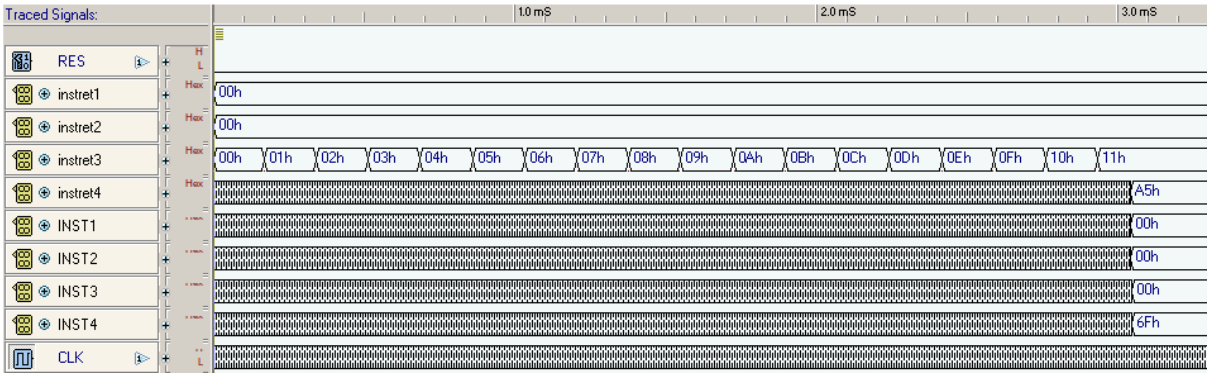
t_exec_esperado = 4518*4.055*1/(6.25*10^6) = 0.0029312784 = 2,9 milissegundos

Como o grupo não conseguiu implementar o timer, não foi possível observar através do CSR o tempo exato de execução do programa. No entanto, utilizando a própria ferramenta do Deeds de simulação temporal, é possível observar que o tempo de execução é de cerca de 3,05 milissegundos, um valor bem próximo dos 2,9 previstos.

Essa diferença pode ser explicada principalmente pela imprecisão no cálculo do CPI média graças ao tipo-I, onde lw precisa de 5 ciclos para ser executado e os outros comandos precisam de apenas 4.

Na figura abaixo, é possível ver uma captura de tela mostrando a execução do teste2.s no processador:

**Figura 2.** Simulação temporal do teste2.s no processador, utilizando a frequência máxima de 6.25 MHz.




O vídeo mostrando a execução do teste2.s no processador pode ser visto em: https://youtu.be/PCjOOH8yHeI.