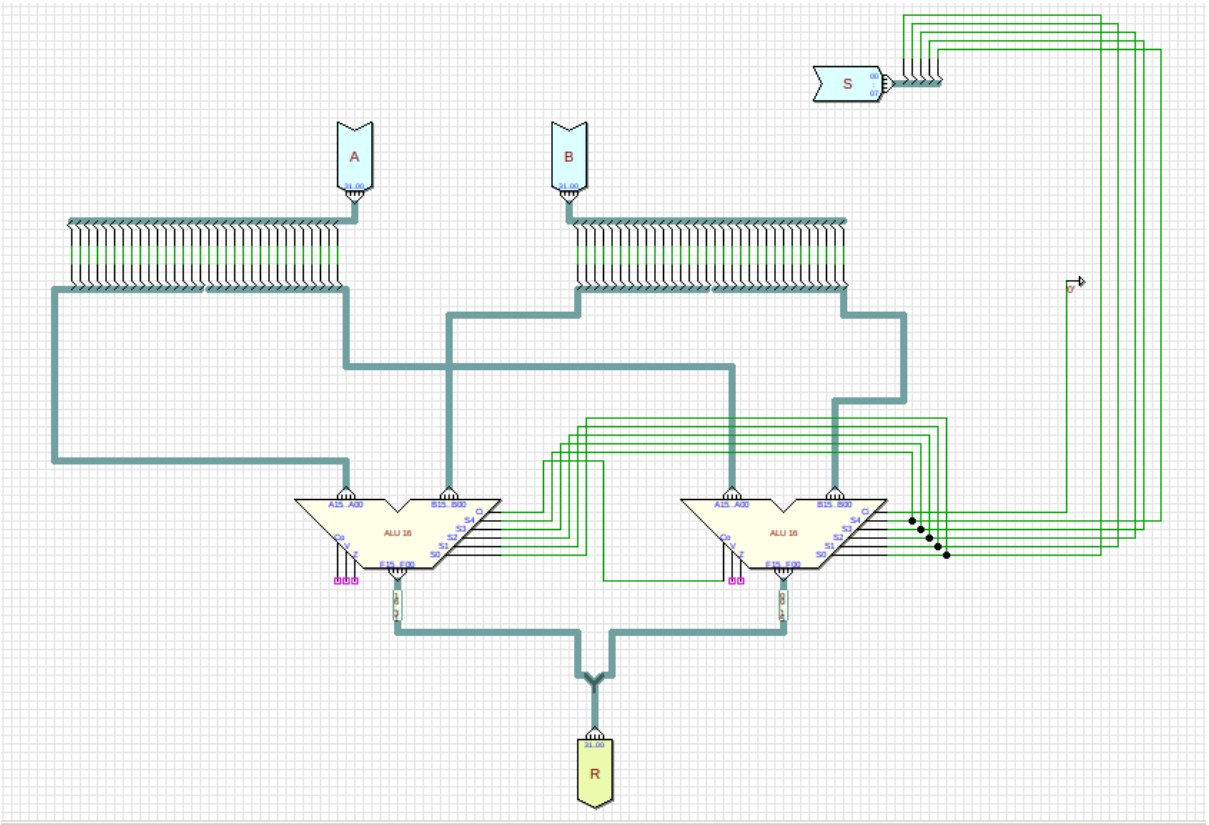
Grupo 12 - OAC - 2021.2
Laboratório 3
Eduardo Ferreira Marques Cavalcante - 202006368
Gabriel Mendes Ciriatico Guimarães - 202033202
Gustavo Lopes Dezan - 202033463

CPU μRISC-V UNICICLO

1.1) *Construa uma Unidade Lógico Aritmética (ULA), puramente combinacional de 32 bits, capaz de dar suporte a essas instruções. Defina o sinal de controle ALUCtrl de acordo com a tabela vista em aula.*

Para implementar a ULA de 32 bits, graças às limitações do software Deeds, foi necessário utilizar 2 ULAs de 16 bits. Essas 2 ULAs são conectadas pelo bit CI (Carry In) da ULA que recebe os bits mais significativos (MSB) ao bit CO (Carry On) da ULA de bits menos significativos (LSB). As entradas de dados, por sua vez, são divididas entre os 16 bits mais significativos de A e de B. Já as entradas de seleção S0…S4 são as mesmas para as duas ULAs.

A visão geral da implementação da ULA de 32 bits pode ser vista abaixo:



O que define os sinais de controle S0..S4 dessa ULA de 32 bits vem da unidade de controle da ULA, feita com base na instrução lida pelo processador. O processador

implementado tem unidades de controle feitos no modelo não hierárquico, isto é, que utilizam os bits de Opcode, Funct3 e Funct7 para definir os sinais de controle.

Para definir esses sinais da controladora da ULA, primeiro foi necessário colocar em uma tabela as informações de cada operação que utilizava a ULA, bem como qual operação da ULA é necessária. Chegamos na tabela abaixo, com 12 das operações implementadas precisando usar a ULA:

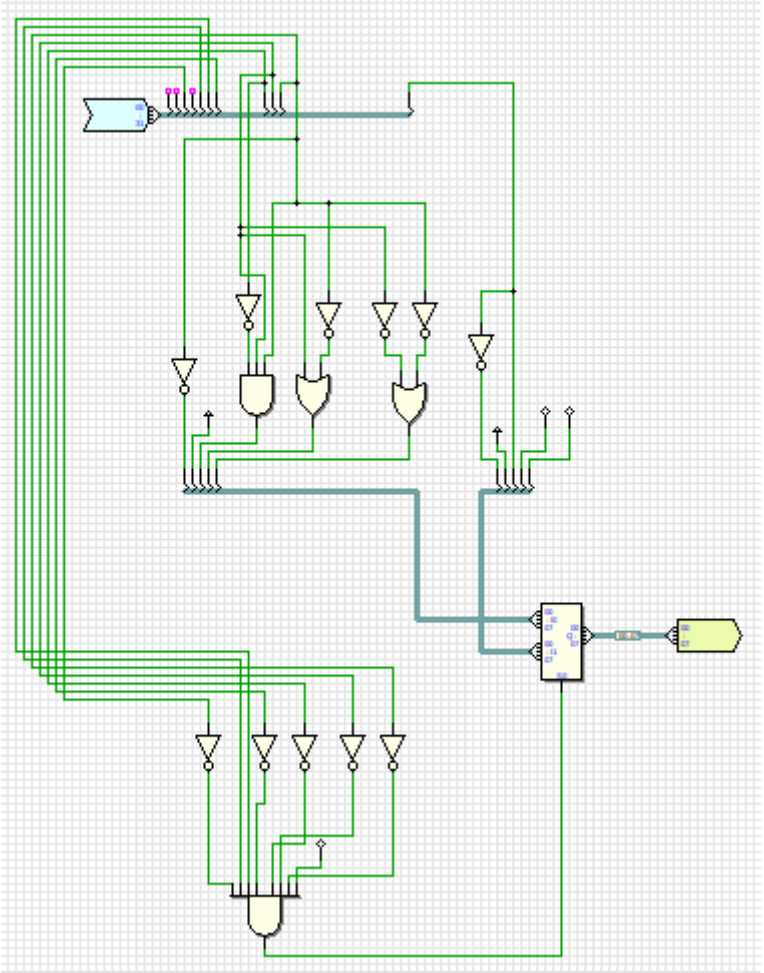
Mnemônico	**Operação**	**Opcode**	**Funct3**	**Funct7**
add	Adição	0110011	000	0000000
addi	Adição	0010011	000	
and	AND	0110011	111	0000000
andi	AND	0010011	111	
jalr	Adição	1100111	000	
lw	Adição	0000011	010	
or	OR	0110011	110	0000000
ori	OR	0010011	110	
sub	Subtração	0110011	000	0100000
sw	Adição	0100011	010	
xor	XOR	0110011	100	0000000
xori	XOR	0010011	100	

Utilizando apenas o Opcode, não é possűiel diferenciar todas essas operações, já que há Opcodes onde há diferentes operações para a ULA executar. Olhando Opcode, Funct3 e Funct7, porém, encontramos muitas entradas, complicando bastante o mapa de Karnaugh para ser feito.

A solução adotada foi dividir as operações em 2 grupos: comandos que se diferenciam quanto à operação da ULA com base em Funct3, e comandos que se diferenciam com base em Funct7. O que define cada um desses grupos é o Opcode e a Funct3, já que no segundo grupo estão apenas os comandos add e sub, enquanto no outro estão todos os comandos restantes.

Para cada um desses grupos foi montado um mapa de Karnaugh com 5 saídas (os sinais de controle da ULA). Para decidir qual a saída do controle da ULA, por sua vez, foi usado um MUX 2:1, cujo sinal de controle é decidido dependendo de qual grupo de comandos a instrução inserida é. Para esse bit do MUX, bastou inserir um circuito que checa se a instrução dada é add ou sub (através do Opcode e Funct3).

Abaixo, é possível ter uma visão geral dessa controladora da ULA:



No caso dos comandos definidos por Funct3, temos as saídas definidas por (Ti diz respeito aos bits de Funct3, onde i vai de 0 a 2):

S0 = T2’
S1 = 0
S2 = T0*T1*T2
S3 = T1 + T2’
S4 = T1’ + T2’

Já no caso de add e sub, definidos por Funct7, temos (Gi diz respeito aos bits de Funct7, onde i vai de 0 a 6):

S0 = G5’
S1 = 0
S2 = G5
S3 = 1
S4 = 1

Já o bit que controla o MUX 2:1 é definido pelo circuito (Oi diz respeito aos bits de Opcode, onde i vai de 0 a 6):

M = O2'*O4*O5*O6'*T0'*T1'*T2'

1.2) *Construa as memórias de Instruções (ROM 1Ki x 32 bits) e de Dados (RAM Síncrona 1Ki x 32 bits) que possuam barramentos de 32 bits e os sinais de controle EscreveMem e LeMem. Considere que a memória de dados começa no endereço 0x10010000 e a memória de programa em 0x00400000.*

Como no processador uniciclo implementado a memória de instruções (RAM) é separada da memória de dados (ROM), foram utilizadas duas memórias diferentes no processador. Foi implementado apenas um processo de checagem para verificar se o endereço requisitado estava no intervalo possível da memória de instruções ou da memória de dados.

O Deeds fornece apenas memórias de 16 bits, logo foi necessário utilizar uma memória para os bits mais significativos e uma para os menos significativos. A memória de dados também se difere da memória de instruções já que, no Deeds, não pode ser inicializada.

Além disso, na memória de instruções foi necessário implementar um processo de shift à direita em 2 bits, já que no RARS o endereçamento é feito de tal forma que um endereço de memória começa 4 bits antes da próxima, ocupando esse intervalo. No Deeds, porém, o endereçamento ocorre de uma posição em uma posição. Assim, numa movimentação padrão PC + 4, por exemplo, com o shifter temos PC + 1, se adequando à forma como o endereçamento de memória é feito no Deeds.

1.3) *Construa um banco de registradores de 32 registradores de 32 bits cada de acordo com os requerimentos vistos em aula. Defina o registrador sp com o valor default 0x100103FC (último endereço da memória de dados).*

O banco de registradores de 32 bits foi construído utilizando 64 registradores de 16 bits - 1 para os bits menos significativos (LSB) e outro para os mais significativos (MSB). O registrador 0 sempre tem o valor 0 registrado, enquanto o registrador sp tem o valor padrão 0x100103FC (último endereço da memória de dados). Quando o banco é resetado, todos os registradores têm o valor zerado (menos o sp, que volta ao valor padrão).

Para a seleção do registrador onde o dado vai ser escrito, foi usado um DECOD 5:32 (construído a partir de um DEC 2:4 e 4 DECOD 3:8), cuja saída é conectada à entrada de escrita dos registradores. Assim, apenas os registradores de RD têm a entrada E = 1. A seleção de RA e RB, por sua vez, é feita através de MUX 32:1 (construído através de 4 MUX 8:1 e um MUX 4:1), tanto para LSB quanto para MSB.

1.4) *Construa o caminho de dados completo e identifique os sinais de controle. Defina a tabela verdade do Bloco de Controle e projete um circuito que o implemente.*

O Bloco de Controle também foi feito utilizando o modelo não hierárquico, onde são usados os sinais de Opcode, Funct7 e Funct3. Assim como na controladora da ULA, foi feita

uma separação entre os diferentes tipos de comandos e o que os diferencia para reduzir a quantidade de entradas dos mapas de Karnaugh para as saídas. Assim, recebe 17 bits de entrada e retorna 9 bits de saída.

Os bits de saída controlam o caminho de dados junto com as outras controladoras implementadas (da ULA, do comparador e de branch). Uma explicação detalhada de cada bit de saída pode ser encontrada abaixo, junto com um esquema explicando cada valor possível:

Sinal	**Valor**	**Explicação**	**Controla**
R0…R2	000	Saída da memória de dados	O que é escrito em RD
R0…R2	100	Saída da ULA	
R0…R2	010	Saída do comparador	
R0…R2	110	Saída do shifter	
R0…R2	001	PC + 4	
R0…R2	101	Valor do imediato	
R0…R2	011	PC + imediato	
W	0	Não escreve no banco de registradores	Sinal WE do banco de registradores
W	1	Escreve no banco de registradores	
M	0	B é o valor de RB	Sinal de controle do MUX que define a entrada B da ULA, do shifter e do comparador
M	1	B é o imediato	
C	0	Não é branch ou jal	Define se PC pode mudar para PC + Imm (branch e jal)
C	1	É branch ou jal	
P	0	PC = PC +4 ou PC + Imm	Sinal de controle de MUX que define PC
P	1	PC = Saída da ULA (RS1 + Imm)	
WM	0	Não escreve na memória de dados	Sinal de escrita na memória de dados

WM	1	Escreve na memória de dados	
RM	0	Não lê da memória de dados	Sinal de leitura na memória de dados
RM	1	Lê da memória de dados	

Aqui, assim como na controladora da ULA, foi utilizado um MUX para diferenciar os tipos de comandos e, assim, reduzir as entradas nos mapas de Karnaugh a serem feitos. Foram identificados dois grupos de comandos: aqueles definidos por Funct3 e O5 (quinto bit do Opcode); e aqueles apenas por Opcode. A diferenciação entre esses dois grupos é feita com base no OPcode, em que no primeiro temos os comandos de Opcode 0010011 e 0110011 e no segundo, todo o restante.

A entrada de controle do MUX 2:1 que determina a saída da Unidade de Controle é definida pela equação:

S = O2 + O3 + O4' + O6

Já a tabela verdade do primeiro grupo, definido por Funct3 e O5 pode ser vista abaixo, junto com as equações que implementam cada saída da unidade:

Mn.	**O5**	**T0**	**T1**	**T2**		**R0**	**R1**	**R2**	**M**	**C**	**P**	**W**	**WM**	**RM**
addi	0	0	0	0		1	0	0	1	0	0	1	0	0
andi	0	1	1	1		1	0	0	1	0	0	1	0	0
ori	0	0	1	1		1	0	0	1	0	0	1	0	0
slli	0	1	0	0		1	1	0	1	0	0	1	0	0
slti	0	0	1	0		0	1	0	1	0	0	1	0	0
sltiu	0	1	1	0		0	1	0	1	0	0	1	0	0
xori	0	0	0	1		1	0	0	1	0	0	1	0	0
add	1	0	0	0		1	0	0	0	0	0	1	0	0
and	1	1	1	1		1	0	0	0	0	0	1	0	0
or	1	0	1	1		1	0	0	0	0	0	1	0	0
sll	1	1	0	0		1	1	0	0	0	0	1	0	0
slt	1	0	1	0		0	1	0	0	0	0	1	0	0

sltu	1	1	1	0		0	1	0	0	0	0	1	0	0
sub	1	0	0	0		1	0	0	0	0	0	1	0	0
xor	1	0	0	1		1	0	0	0	0	0	1	0	0

R0 = T1' + T2
R1 = T1*T2' + T0*T2'
R2 = 0
W = 1
M = O5'
C = 0
P = 0

A tabela verdade e as equações do segundo grupo de comandos podem ser vistas abaixo:

Mn.	**O2**	**O3**	**O4**	**O5**	**O6**		**R0**	**R1**	**R2**	**M**	**C**	**P**	**W**	**WM**	**RM**
auipc	1	0	1	0	0		0	0	1	X	0	0	1	0	0
beq	0	0	0	1	1		X	X	X	0	1	0	0	0	0
bge	0	0	0	1	1		X	X	X	X	1	0	0	0	0
bgeu	0	0	0	1	1		X	X	X	X	1	0	0	0	0
blt	0	0	0	1	1		X	X	X	X	1	0	0	0	0
bltu	0	0	0	1	1		X	X	X	X	1	0	0	0	0
bne	0	0	0	1	1		X	X	X	X	1	0	0	0	0
jal	1	1	0	1	1		0	0	1	X	1	0	1	0	0
jalr	1	0	0	1	1		0	0	1	1	0	1	1	0	0
lui	1	0	1	1	0		1	0	1	X	0	0	1	0	0
lw	0	0	0	0	0		0	0	0	1	0	0	1	0	1
sw	0	0	0	1	0		X	X	X	1	0	0	0	1	0

R0 = O2'*O5 + O4*O5
R1 = O2'*O5 + O4*O5'
R2 = O4 + O5
W = O2 + O5'
M = O2 + O6'

C = O3 + O2'*O6
P = O2*O3'*O4'

Note que os sinais WM e RM não são definidos nessas equações. Esses sinais são definidos fora do MUX 2:1. WM só ocorre quando é o comando sw, logo ele foi implementado como um circuito que identifica se o comando dado é sw. RM, por sua vez, é o sinal contrário a WM. Apesar da leitura da ROM ocorrer com o comando lw, o dado que vem da memória de dados é escrito no banco de registradores apenas quando R0…R2 é definido por lw.

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

A frequência máxima utilizável no processador desenvolvido é de 4.0 MHz. Essa frequência pode ser vista apenas na simulação em forma de onda. Fora dali, o Deeds permite que o processador rode sem erro apenas em 15 Hz (desconhecemos a causa disso, provavelmente ligada a limitações da interface do software).

1.6) *Adicione ao seu processador os registradores do CSR cycle, time e instret, apenas como registradores de monitoramento (não acessíveis ao processador). Dica: Criem um controle do clock desses contadores por uma condição Ex.: Instrução = FIM: j FIM 0x0000006f.*

Como é um processador uniciclo, utilizamos o mesmo registrador para o cycle e instret, já que uma instrução é executada a cada ciclo. O registrador time, por sua vez, não pôde ser implementado com sucesso, já que tivemos dificuldade de entender o que era para colocar ali. Para os cálculos de tempo, utilizamos as informações do próprio Deeds no diagrama temporal.

1.7) *Implemente um circuito onde vc escolhe 2 registradores (por 5 + 5 chaves) e seus valores sejam apresentados em 2 conjuntos de 8 displays de 7 segmentos. Mostre também o registrador PC, a instrução lida da memória e os registradores cycle, timer e instret em displays;*

A implementação dessa escolha de registradores é feita dentro do banco de registradores através de MUX que usam as entradas de seleção desses registradores (no processador, chamados de RX e RY).

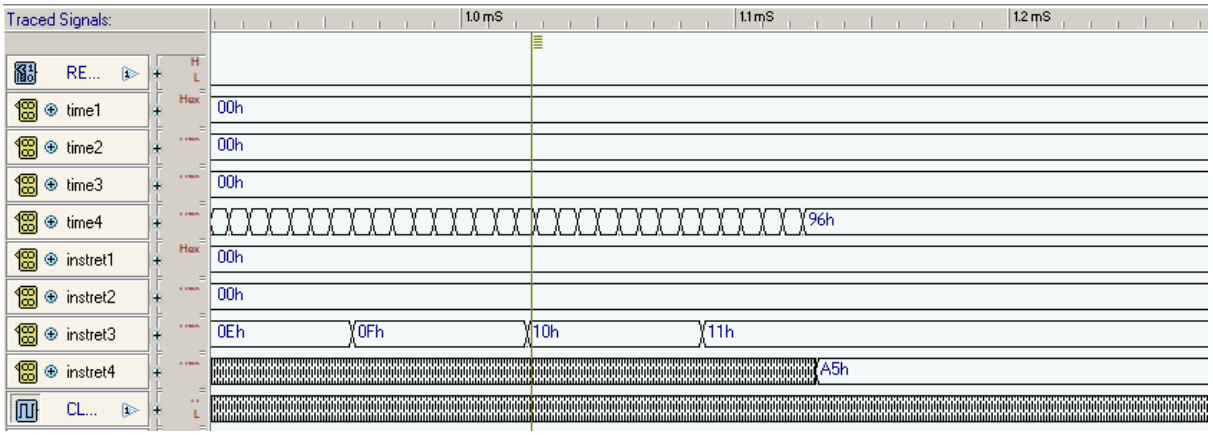
1.8) *Escreva um programa TestBench.s que verifique se todas as instruções foram implementadas corretamente, e filme a sua execução.*

O programa TestBench foi programado para testar os 27 comandos implementados no processador. O vídeo da sua execução pode ser visto em: https://youtu.be/f-UETBH2xJ8.

1.9) *Faça a simulação e forma de onda na maior frequência possível e filme a execução do programa (com uma frequência menor) Teste2.s no seu processador. Verifique se texec = I × CPI × T justifique.*

Como é um processador uniciclo, cada instrução tem um CPI (Ciclo Por Instrução) de 1. Fazendo a simulação em forma de onda e usando os registradores do CSR, é possível obter as informações do número de instruções. O período T, por sua vez, é o inverso da frequência máxima do processador. No caso, os testes mostraram que a frequência máxima do processador implementado é de 4.0 MHz.

Observe abaixo a captura de tela da simulação temporal no Deeds do programa teste2.s no processador implementado:



Primeiro, observe que o registrador time do CSR não está configurado corretamente. O grupo não soube implementá-lo da forma correta, o que resultou na saída 0x96 segundos, que em decimal equivale a 150 segundos. No entanto, a informação correta sobre o tempo de execução pode ser vista no Deeds, onde é possível ver que o registrador instret para em cerca de 1,13mS.

O registrador insret, por sua vez, foi implementado com sucesso. Ele para de ser incrementado no valor 0x11A5, isto é, em 4517 instruções. Daí, temos I = 4517 e t (real) = 1,13mS.

Fazendo o cálculo do tempo esperado, temos:

t (esperado) = 1*(1/(4*10^6))*4517 = 0.00112925

Veja que o tempo esperado é de aproximadamente 1,13mS, ou seja, bate aproximadamente com o valor esperado. Esse é um resultado esperado, já que trata-se de um processador uniciclo, onde o valor do CPI é exato.

O vídeo da execução do programa teste2.s pode ser visto em: https://youtu.be/-tRd03ItUWA.