Relatório Prática 3 - Tomasulo

Nome: Pedro Santos Oliveira

1 - Decisões Gerais de Projeto

a) No nosso projeto consideramos instruções de 16 bits divididas da seguinte maneira: Imediato[15:12], Opcode[11:9], RX[8:6], RY[5:3], RZ[2:0].

b) Em relação a Estação de Reserva optamos por utilizar uma Estação de Reserva que recebe tanto instruções de Soma e Subtração, como instruções de Load e Store, e que possui ao todo três lugares, ou seja, suporta até três instruções ao mesmo tempo.

c) Em relação a Memória de Instruções optamos por não utilizar a biblioteca LPM e construir a Memória de Instruções como sendo uma matriz de dimensões 5x16, ou seja, uma Memória de Instruções que suporta até 5 instruções de 16 bits.

d) Para controlar o despacho de instruções da Fila de Instruções para a Estação de Reserva criamos um sinal "*stall*", esse sinal é alterado pela Estação de Reserva quando todas as posições desta estiverem ocupadas. Além disso, esse sinal é passado como entrada para o módulo da Fila de Instruções e é analisado antes de realizar um possível despacho de instrução, garantindo assim, que instruções só serão despachadas caso a Estação de Reserva tenha espaço disponível.

e) Instruções de Load e Store utilizam apenas dois registradores, um para indicar o destino do dado, e outro para indicar um valor que será considerado para calcular o endereço de memória. Sendo assim, instruções de Load e Store não utilizarão os bits destinados a RZ, que são os 3 bits menos significativos dos 16 bits da instrução.

f) No caso das instruções de Load que possuem sempre um imediato, esse imediato sempre será salvo na coluna Vj.

g) Todas as colunas da Estação de Reserva, com exceção da coluna "Busy", serão inicializadas com todos os bits iguais a 1, que serão considerados valores inválidos.

h) A coluna "Address" da Estação de Reserva será inicializada com todos os bits iguais a 1 para garantir que a memória comece acessando a posição 31. A posição de memória 31 conterá um dado de 16 bits 1, que será considerado pelo programa um valor inválido, e será usado pelo CDB Arbiter para realizar checagens durante a execução.

i) A saída da ULA será iniciada com todos os bits iguais a 1, o que significa um valor inválido. Já a saída do CDB será iniciada com todos os bits iguais a 0, o que significa um valor inválido.

j) Em relação ao CDB Arbiter, caso ele receba dados provenientes de ambas as ULAs no mesmo ciclo, ele irá priorizar o dado proveniente da instrução que foi despachada primeiro.

2 - Sinais da ULA e Sinais de Simulação

Sinal do CDB:

cdb[15:15]	Habilita escrita em R0
cdb[14:14]	Habilita escrita em R1
cbd[13:13]	Habilita escrita em R2
cbd[12:11]	Posição da instrução na Estação de Reserva
cbd[10:10]	Indica qual ULA escreveu no CDB 1 - ULA de Soma e Subtração 0 - ULA de Load e Store
cbd[9:0]	Dado produzido pela ULA

Sinal das Instruções na memória de Instruções:

[15:12]	Imediato
[11:9]	Opcode da Instrução
[8:6]	Registrador de Destino (RX)
[5:3]	Operando 1 (RY)
[2:0]	Operando 2 (RZ)

Opcode das Instruções:

ADD	000
SUB	001
LD	010
SD	011

Sinal "*is_waiting_value*":

is_waiting_value[2:2]	**1 - R2 está esperando um novo valor** **0 - R2 não está esperando um novo valor**
is_waiting_value[1:1]	**1 - R1 está esperando um novo valor** **0 - R1 não está esperando um novo valor**
is_waiting_value[0:0]	**1 - R0 está esperando um novo valor** **0 - R0 não está esperando um novo valor**

Sinal de saída da ULA:

O sinal de saída da ULA é o sinal que é escrito no CDB, portanto, a configuração de bits desse sinal segue a mesma lógica da configuração de bits do sinal do CDB mostrado acima.

Sinais da Simulação:

Nas simulações apresentadas neste relatório aparecem alguns sinais indicados com o valor -1 em decimal. Esse valor indica que todos os bits daquele sinal são iguais a 1, e para esse trabalho consideramos que quando isso acontece o valor do sinal é um valor inválido.

3 - Formato dos Testes

Nos testes apresentados neste relatório seguiremos a seguinte linha de apresentação:

1. Apresentação das instruções executadas nos testes
2. Apresentação da Memória de Instruções no momento do teste
3. Apresentação das saídas da ULA para cada instrução executada
4. Apresentação dos valores iniciais e finais dos registradores usados na execução
5. Breve explicação da execução do teste
6. Print da simulação com explicações detalhadas

4 - Testes

a) Dependência de dados verdadeira ou RAW (Read After Write)

Instruções Executadas:

Instrução	Imediato[15:12]	Opcode[11:9]	RX[8:6]	RY[5:3]	RZ[2:0]
ADD R0, R1, R2	0000	000	000 (R0)	001 (R1)	010 (R2)
SUB R1, R0, R1	0000	001	001 (R1)	000 (R0)	001 (R1)

Memória de Instruções:

```
// Inicializando a Memória de Instruções
InstructionMemory[0] <= 16'b0000000000001010; // ADD R0, R1, R2
InstructionMemory[1] <= 16'b0000001001000001; // SUB R1, R0, R1
InstructionMemory[2] <= 16'b1111111111111111;
InstructionMemory[3] <= 16'b1111111111111111;
InstructionMemory[4] <= 16'b1111111111111111;
```

Para realizar esse teste utilizamos apenas duas instruções, portanto podemos desconsiderar as 3 entradas restantes da memória de instruções.

Saída da ULA para ADD R0, R1, R2: 1000010000001011

Registrador de Destino	Posição da Instrução na Estação de Reserva	ULA que executa a instrução	Dado
100	00	1	0000001011

Saída da ULA para SUB R1, R2, R0: 0100110000000100

Registrador de Destino	Posição da Instrução na Estação de Reserva	ULA que executa a instrução	Dado
010	01	1	0000000100

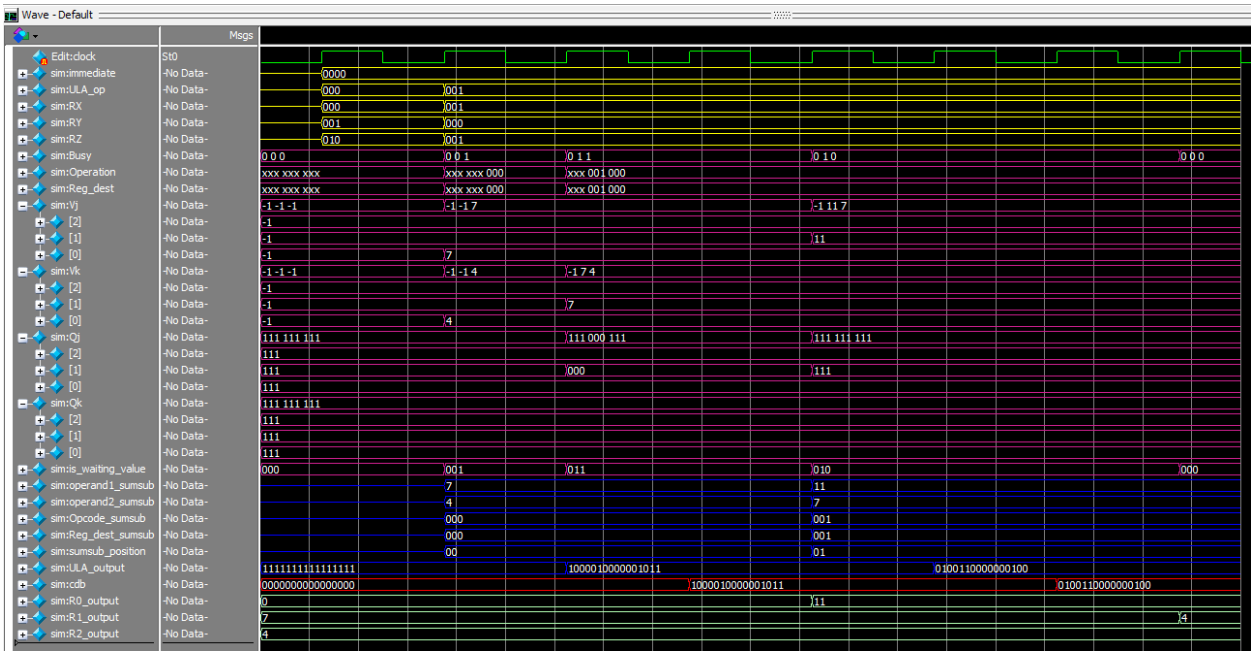
Inicialização dos Registradores: R0 = 0, R1 = 7, R2 = 4.

Valor Final dos Registradores: R0 = 11, R1 = 4, R2 = 4.

Execução: A instrução SUB R1, R0, R1 possui uma dependência verdadeira (RAW) em relação a instrução ADD R0, R1, R2. Sendo assim, a instrução ADD será executada primeiro e assim que for

finalizada, o novo valor de R0 será atualizado na Estação de Reserva, permitindo então que a instrução SUB possa ser executada. Vale a pena destacar que, dado o funcionamento correto do algoritmo, o novo valor de R0 será atualizado no Banco de Registradores e na Estação de Reserva no mesmo ciclo, ou seja, o CDB realiza o bypassing da informação para que a instrução SUB possa começar a ser executada mais rapidamente.

Print da Simulação:



Legenda de Cores:

Verde: Clock e Registradores
Amarelo: Sinais de Saída da Fila de Instruções
Rosa: Sinais da Estação de Reserva
Azul: Sinais de Entrada da ULA
Vermelho: CDB

Ciclo 1 (Primeiro Posedge): No primeiro ciclo podemos observar que ocorre o despacho da primeira instrução pela Fila de Instruções, se analisarmos a imagem podemos perceber que os sinais destacados de amarelo recebem os valores referentes a instrução ADD R0, R1, R2. Além disso, vale a pena destacar que o restante dos sinais estão com valores de inicialização durante o ciclo 1, ou seja, valores inválidos.

Ciclo 2 (Segundo Posedge): No segundo ciclo, analisando os sinais destacados de rosa, podemos perceber que a primeira instrução (ADD) é inserida na Estação de Reserva na linha 0, e como essa instrução já possui os dados dos dois operandos, as colunas Vj e Vk serão povoadas, enquanto as colunas Qj e Qk não serão utilizadas. Ao analisar a imagem podemos notar que na coluna Vj na linha 0 é adicionado o valor 7 (Primeiro Operando referente ao Registrador R1) e na coluna Vk na linha 0 é

adicionado o valor 4 (Segundo Operando referente ao registrador R2). Vale a pena destacar também que o Registrador R0 é salvo como registrador destino desta instrução, e podemos perceber isso analisando o sinal “is_waiting_value”, que recebe o bit 1 na posição 0, indicando que a partir de agora o R0 está esperando um novo valor. Além disso, nesse mesmo ciclo a instrução de ADD é também enviada para a ULA, e podemos perceber isso analisando os sinais destacados de azul. Por fim, a segunda instrução (SUB R1, R0, R1) também é despachada no segundo ciclo seguindo a mesma lógica de despacho apresentada anteriormente.

Ciclo 3 (Terceiro Posedge): No terceiro ciclo, o resultado da instrução de ADD é produzido pela ULA, e podemos perceber isso analisando o sinal “ULA_output” destacado de azul. Além disso, nesse mesmo ciclo, a instrução de SUB é adicionada na estação de reserva na linha 1, e vale a pena destacar que, como a instrução de SUB possui R0 como um de seus operandos, a coluna Qj na linha 1 será utilizada para indicar que esta instrução está esperando o novo valor de R0 (Dependência de dados verdadeira), logo, como a instrução de SUB não possui todos os operandos prontos, ela não será enviada para a ULA. Portanto, os sinais destacados de azul não recebem nenhum dado novo neste ciclo.

Ciclo 4 (Quarto Posedge): No quarto ciclo, o resultado produzido pela ULA referente a instrução de ADD é mandado para o CDB, podemos perceber isso analisando o sinal destacado de vermelho.

Ciclo 5 (Quinto Posedge): No quinto ciclo, o dado presente no CDB é enviado para a Estação de Reserva e para o Banco de Registradores. Analisando os sinais verdes na parte de baixo da imagem, percebemos que o registrador R0 recebe seu novo valor (11). Além disso, como o CDB faz o encaminhamento do dado também para a Estação de Reserva, podemos perceber que neste mesmo ciclo a coluna Qj na linha 1 é resetada e a coluna Vj na linha 1 recebe o valor 11, referente ao novo valor de R0. Sendo assim, a partir de agora a instrução de SUB fica pronta para ser enviada para a ULA, e se analisarmos os sinais destacados de azul conseguimos enxergar que neste mesmo ciclo esta instrução já é enviada para a ULA. Por fim, vale a pena destacar também que, ao atualizar o registrador R0 e escrever na Estação de Reserva a instrução de ADD é finalizada e é retirada da Estação, e podemos observar isso analisando o sinal de “Busy” destacado de rosa, que recebe o valor 010, indicando que a linha 0 foi liberada.

Ciclo 6 (Sexto Posedge): No sexto ciclo, a ULA produz a saída da instrução de SUB.

Ciclo 7 (Sétimo Posedge): No sétimo ciclo, o resultado produzido pela ULA referente a instrução de SUB é mandado para o CDB, podemos perceber isso analisando o sinal destacado de vermelho.

Ciclo 8 (Oitavo Posedge): No oitavo ciclo, o dado presente no CDB é enviado para a Estação de Reserva e para o Banco de Registradores. Sendo assim, é possível perceber o registrador R1 recebendo seu novo valor (4). Por fim, é possível perceber também que o sinal “Busy” recebe o valor 000, indicando que a instrução de SUB foi finalizada, e que a Estação de Reserva agora está vazia.

b e d) Dependência de saída (Write After Write) e Dependência/Hazard Estrutural (Unidade Funcional cheia)

Instruções Executadas:

Instrução	Imediato[15:12]	Opcode[11:9]	RX[8:6]	RY[5:3]	RZ[2:0]
SUB R0, R1, R2	0000	001	000 (R0)	001 (R1)	010 (R2)
ADD R0, R1, R2	0000	000	000 (R0)	001 (R1)	010 (R2)

Memória de Instruções:

```
// Inicializando a Memória de Instruções
InstructionMemory[0] <= 16'b0000001000001010; // SUB R0, R1, R2
InstructionMemory[1] <= 16'b0000000000001010; // ADD R0, R1, R2
InstructionMemory[2] <= 16'b1111111111111111;
InstructionMemory[3] <= 16'b1111111111111111;
InstructionMemory[4] <= 16'b1111111111111111;
```

Para realizar esse teste utilizamos apenas duas instruções, portanto podemos desconsiderar as 3 entradas restantes da memória de instruções.

Saída da ULA para SUB R0, R1, R2: 1000010000000101

Registrador de Destino	Posição da Instrução na Estação de Reserva	ULA que executa a instrução	Dado
100	00	1	0000000101

Saída da ULA para ADD R0, R1, R2: 1000110000001111

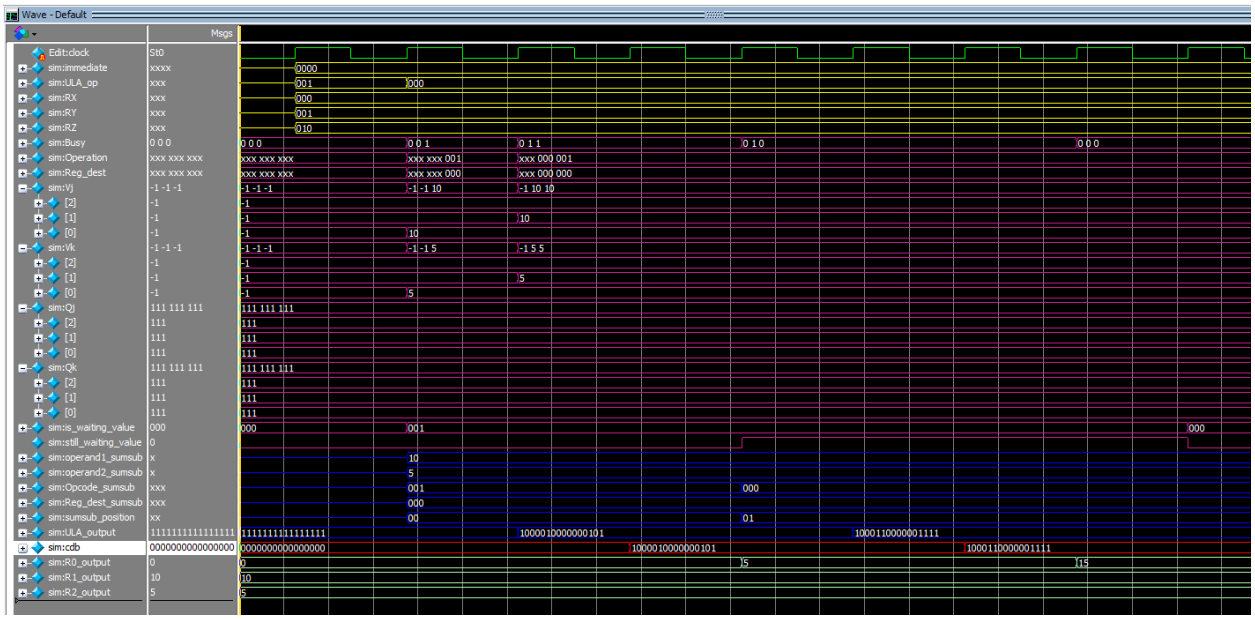
Registrador de Destino	Posição da Instrução na Estação de Reserva	ULA que executa a instrução	Dado
100	01	1	0000001111

Inicialização dos Registradores: R0 = 0, R1 = 10, R2 = 5.

Valor Final dos Registradores: R0 = 15, R1 = 10, R2 = 5.

Execução: Nesse teste será mostrado um caso de WAW (Write After Write), onde será executada uma instrução de SUB que escreve em R0, e logo em seguida uma instrução de ADD que escreve também em R0. O objetivo desse teste é mostrar que as instruções são mandadas para a ULA em ordem, o que garante que o valor final do registrador será o valor correto.

Print da Simulação:



Legenda de Cores:

Verde: Clock e Registradores
Amarelo: Sinais de Saída da Fila de Instruções
Rosa: Sinais da Estação de Reserva
Azul: Sinais de Entrada da ULA
Vermelho: CDB

Ciclo 1 (Primeiro Posedge): No primeiro ciclo podemos observar que ocorre o despacho da primeira instrução pela Fila de Instruções, se analisarmos a imagem podemos perceber que os sinais destacados de amarelo recebem os valores referentes a instrução SUB R0, R1, R2. Além disso, vale a pena destacar que o restante dos sinais estão com valores de inicialização durante o ciclo 1, ou seja, valores inválidos.

Ciclo 2 (Segundo Posedge): No segundo ciclo, analisando os sinais destacados de rosa, podemos perceber que a primeira instrução (SUB) é inserida na Estação de Reserva na linha 0, e como essa instrução já possui os dados dos dois operandos, as colunas Vj e Vk serão povoadas, enquanto as colunas Qj e Qk não serão utilizadas. Ao analisar a imagem podemos notar que na coluna Vj na linha 0 é adicionado o valor 10 (Primeiro Operando referente ao Registrador R1) e na coluna Vk na linha 0 é adicionado o valor 5 (Segundo Operando referente ao registrador R2). Vale a pena destacar também que o Registrador R0 é salvo como registrador destino desta instrução, e podemos perceber isso analisando o sinal “is_waiting_value”, que recebe o bit 1 na posição 0, indicando que a partir de agora o R0 está esperando um novo valor. Além disso, nesse mesmo ciclo a instrução de SUB é também enviada para a ULA, e podemos perceber isso analisando os sinais destacados de azul. Por fim, a segunda instrução (ADD R0, R1, R2) também é despachada no segundo ciclo seguindo a mesma lógica de despacho apresentada anteriormente.

Ciclo 3 (Terceiro Posedge): No terceiro ciclo podemos observar que a ULA produz a saída referente a instrução de SUB. Além disso, podemos observar também que a instrução de ADD é inserida na Estação de Reserva. É importante destacar que neste ciclo a instrução de ADD já está pronta para ser executada, uma vez que não existe dependência de dados verdadeira entre as instruções, porém, a instrução de dado não é enviada para a ULA pois esta está sendo usada pela instrução de SUB, ou seja, ocorre um hazard estrutural.

Ciclo 4 (Quarto Posedge): No quarto ciclo, o resultado produzido pela ULA referente a instrução de SUB é mandado para o CDB, podemos perceber isso analisando o sinal destacado de vermelho.

Ciclo 5 (Quinto Posedge): No quinto ciclo, o dado presente no CDB é enviado para a Estação de Reserva e para o Banco de Registradores. Analisando os sinais verdes na parte de baixo da imagem, percebemos que o registrador R0 recebe seu novo valor (5). Nesse mesmo ciclo, a instrução de SUB é finalizada e é retirada da Estação de Reserva, podemos perceber isso analisando o sinal de "Busy". Por fim, como a instrução de SUB foi finalizada, neste mesmo ciclo a instrução de ADD é enviada para a ULA, como indicado pelos sinais destacados de azul. O ADD só é enviado neste ciclo devido ao hazard estrutural que ocorre na unidade funcional.

Ciclo 6 (Sexto Posedge): No sexto ciclo a ULA produz o resultado da instrução ADD.

Ciclo 7 (Sétimo Posedge): No sétimo ciclo, o resultado produzido pela ULA referente a instrução de ADD é mandado para o CDB, podemos perceber isso analisando o sinal destacado de vermelho.

Ciclo 8 (Oitavo Posedge): No oitavo ciclo, o CDB atualiza o Banco de Registradores e a Estação de Reserva, podemos perceber o R0 recebendo seu novo valor (15). Por fim, a instrução de ADD é finalizada e retirada da Estação de Reserva. Vale a pena destacar que o registrador R0 finaliza a execução com o valor 15, como esperado.

Extra: Nesse teste vale a pena destacar também o sinal "*is_waiting_value*" que permanece com o valor 1 na posição 0 durante a execução de ambas as instruções, isso se deve ao fato de ambas escreverem no registrador R0. É possível perceber que o sinal "*is_waiting_value*" na posição 0 só recebe o valor 0 no ciclo 9, quando ambas as instruções que escrevem em R0 saem da Estação de Reserva.

c) Antidependência ou WAR (Write After Read)

Instruções Executadas:

Instrução	Imediato[15:12]	Opcode[11:9]	RX[8:6]	RY[5:3]	RZ[2:0]
SUB R0, R1, R2	0000	001	000 (R0)	001 (R1)	010 (R2)
LD R1, 1(R2)	0001	010	001 (R1)	010 (R2)	…

Memória de Instruções:

```
// Inicializando a Memória de Instruções
InstructionMemory[0] <= 16'b0000001000001010; // SUB R0, R1, R2
InstructionMemory[1] <= 16'b0001010001010111; // LD R1, 1(R2)
InstructionMemory[2] <= 16'b1111111111111111;
InstructionMemory[3] <= 16'b1111111111111111;
InstructionMemory[4] <= 16'b1111111111111111;
```

Para realizar esse teste utilizamos apenas duas instruções, portanto podemos desconsiderar as 3 entradas restantes da memória de instruções.

Memória de Dados:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	1	2	3	5	0	0	2	0	.....
8	0	0	0	8	0	0	0	0	.....
16	7	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	65535	.....

Saída da ULA para SUB R0, R1, R2: 1000010000000101

Registrador de Destino	Posição da Instrução na Estação de Reserva	ULA que executa a instrução	Dado
100	00	1	0000000101

Saída da ULA para LD R1, 1(R2): 0100110000001111

Registrador de Destino	Posição da Instrução na Estação de Reserva	ULA que executa a instrução	Dado
010	01	0	0000000110

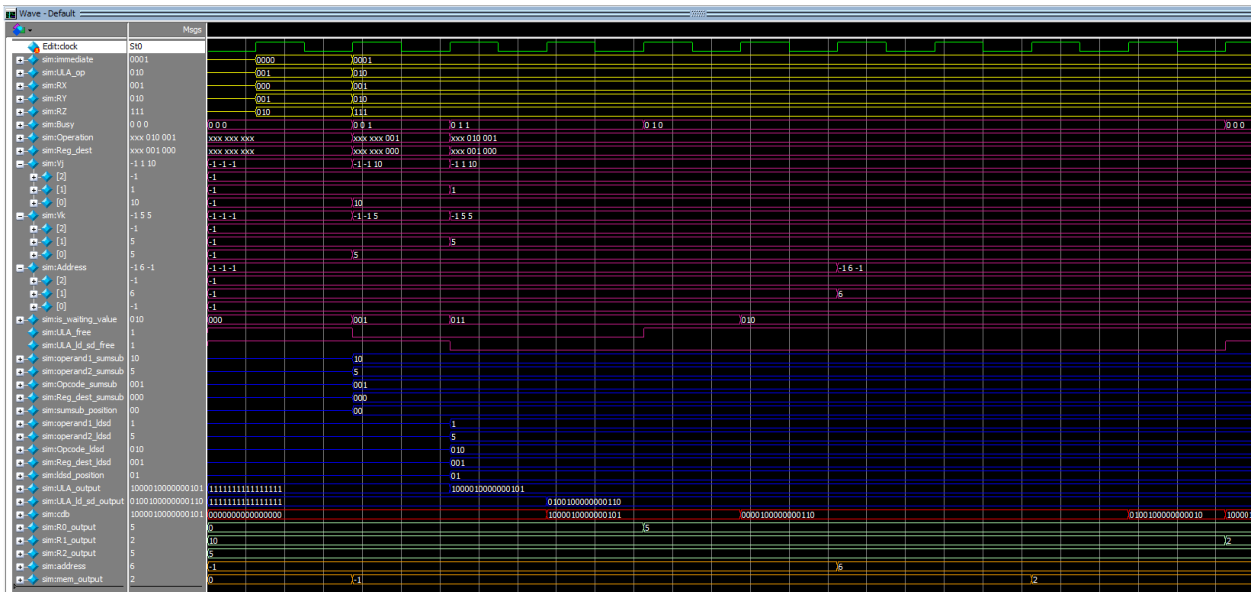
Inicialização dos Registradores: R0 = 0, R1 = 10, R2 = 5, MEM[6] = 2.

Valor Final dos Registradores: R0 = 5, R1 = 2, R2 = 5, MEM[6] = 2.

Execução: Nesse teste será realizado uma instrução de SUB que vai ler o valor presente no registrador R1, essa instrução irá salvar o valor 5 no registrador R0. Em seguida, será executada uma instrução de LD que vai escrever no registrador R1, mostrando assim um caso de WAR (Write After Read). Se observarmos a simulação poderemos observar que as instruções são executadas corretamente, e que os registradores vão finalizar a execução com os valores corretos. Além disso, poderemos observar também o

adiamento no CDB, uma vez que o CDB vai ter dois dados prontos para escrever no mesmo ciclo, porém vai priorizar o dado vindo da ULA de Soma e Subtração.

Print da Simulação:



Legenda de Cores:

Verde: Clock e Registradores
Amarelo: Sinais de Saída da Fila de Instruções
Rosa: Sinais da Estação de Reserva
Azul: Sinais de Entrada das ULAs
Vermelho: CDB
Laranja: Endereço acessado na memória e Valor retornado pela memória

Ciclo 1 (Primeiro Posedge): No primeiro ciclo podemos observar que ocorre o despacho da primeira instrução pela Fila de Instruções, se analisarmos a imagem podemos perceber que os sinais destacados de amarelo recebem os valores referentes a instrução SUB R0, R1, R2. Além disso, vale a pena destacar que o restante dos sinais estão com valores de inicialização durante o ciclo 1, ou seja, valores inválidos.

Ciclo 2 (Segundo Posedge): No segundo ciclo, analisando os sinais destacados de rosa, podemos perceber que a primeira instrução (SUB) é inserida na Estação de Reserva na linha 0, e como essa instrução já possui os dados dos dois operandos, as colunas Vj e Vk serão povoadas, enquanto as colunas Qj e Qk não serão utilizadas. Ao analisar a imagem podemos notar que na coluna Vj na linha 0 é adicionado o valor 10 (Primeiro Operando referente ao Registrador R1) e na coluna Vk na linha 0 é adicionado o valor 5 (Segundo Operando referente ao registrador R2). Vale a pena destacar também que o Registrador R0 é salvo como registrador destino desta instrução, e podemos perceber isso analisando o sinal “is_waiting_value”, que recebe o bit 1 na posição 0, indicando que a partir de agora o R0 está

esperando um novo valor. Além disso, nesse mesmo ciclo a instrução de SUB é também enviada para a ULA, e podemos perceber isso analisando os sinais destacados de azul. Por fim, a segunda instrução (LD R1, 1 (R2)) também é despachada no segundo ciclo seguindo a mesma lógica de despacho apresentada anteriormente.

Ciclo 3 (Terceiro Posedge): No terceiro ciclo podemos observar que a ULA produz a saída referente a instrução de SUB. Além disso, podemos observar também que a instrução de LD é inserida na Estação de Reserva. Nesse mesmo ciclo a instrução de LD é despachada para a ULA de Load e Store, neste caso não ocorre hazard estrutural pois ambas as instruções utilizam ULAs diferentes.

Ciclo 4 (Quarto Posedge): No quarto ciclo, o resultado produzido pela ULA referente a instrução de SUB é mandado para o CDB, podemos perceber isso analisando o sinal destacado de vermelho. Além disso, a ULA de Load e Store produz neste ciclo também o resultado referente à instrução LD.

Ciclo 5 (Quinto Posedge): No quinto ciclo, o dado presente no CDB é enviado para a Estação de Reserva e para o Banco de Registradores. Analisando os sinais verdes na parte de baixo da imagem, percebemos que o registrador R0 recebe seu novo valor (5). Nesse mesmo ciclo, a instrução de SUB é finalizada e é retirada da Estação de Reserva, podemos perceber isso analisando o sinal de “Busy”.

Ciclo 6 (Sexto Posedge): No sexto ciclo o CDB recebe a saída gerada pela ULA de Load e Store.

Ciclo 7 (Sétimo Posedge): No sétimo ciclo a variável “address”, usada para salvar o endereço de memória a ser acessado, recebe seu novo valor (6), referente a soma do imediato com o valor salvo em R2. Além disso, quando o valor de “address” fica pronto, a coluna Address da Estação de Reserva é atualizada.

Ciclo 8 (Oitavo Posedge): No oitavo ciclo a memória está sendo acessada.

Ciclo 9, 10, 11: No nono ciclo o dado buscado na posição 6 de memória é retornado, se observarmos o arquivo mif podemos perceber que o dado correto foi retornado (2). No ciclo 10 o cdb é atualizado com o dado retornado da memória. No ciclo 11 o Banco de Registradores e a Estação de Reserva são atualizados e a instrução de LD é finalizada e retirada da Estação.

e) Conflito no CDB

Instruções Executadas:

Instrução	Imediato[15:12]	Opcode[11:9]	RX[8:6]	RY[5:3]	RZ[2:0]
ADD R0, R1, R2	0000	000	000 (R0)	001 (R1)	010 (R2)
ADD R1, R0, R2	0000	000	001 (R1)	000 (R0)	010 (R2)
LD R1, 1(R0)	0001	010	001 (R1)	000 (R0)	…

Memória de Instruções:

```
// Inicializando a Memória de Instruções
InstructionMemory[0] <= 16'b0000000000001010; // ADD R0, R1, R2
InstructionMemory[1] <= 16'b0000000001000010; // ADD R1, R0, R2
InstructionMemory[2] <= 16'b0001010001000111; // LD R1, 1(R0)
InstructionMemory[3] <= 16'b1111111111111111;
InstructionMemory[4] <= 16'b1111111111111111;
```

Para realizar esse teste utilizamos apenas três instruções, portanto podemos desconsiderar as 2 entradas restantes da memória de instruções.

Memória de Dados:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	1	2	3	5	0	0	2	0	.....
8	0	0	0	8	0	0	0	0	.....
16	7	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	65535	.....

Saída da ULA para ADD R0, R1, R2: 1000010000001111

Registrador de Destino	Posição da Instrução na Estação de Reserva	ULA que executa a instrução	Dado
100	00	1	0000001111

Saída da ULA para ADD R1, R0, R2: 0100110000010100

Registrador de Destino	Posição da Instrução na Estação de Reserva	ULA que executa a instrução	Dado
010	01	1	0000010100

Saída da ULA para LD R1, 1(R0): 0101000000000111

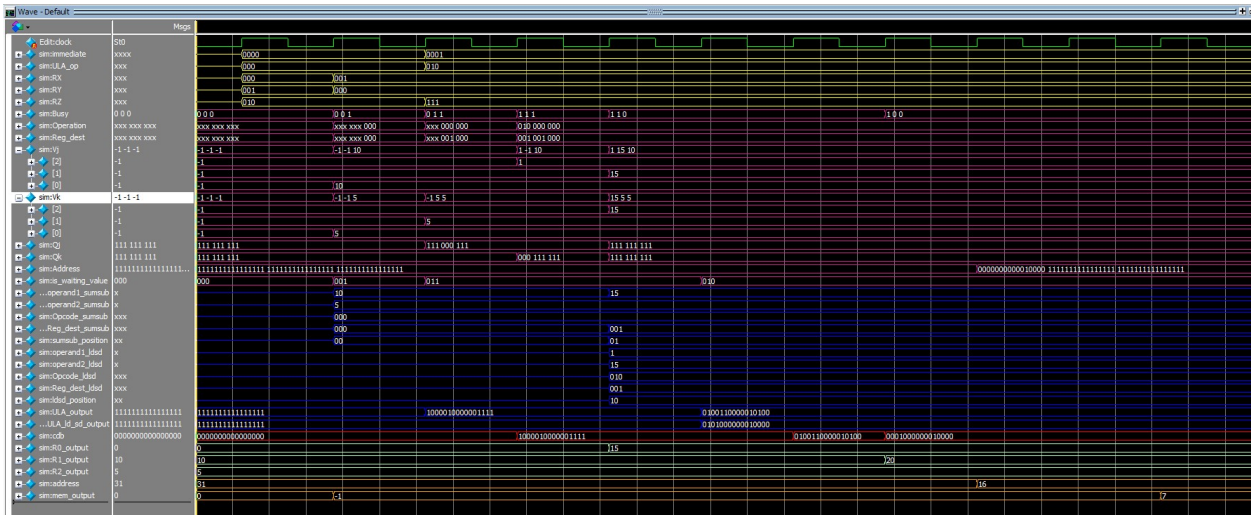
Registrador de Destino	Posição da Instrução na Estação de Reserva	ULA que executa a instrução	Dado
010	10	0	0000000111

Inicialização dos Registradores: R0 = 0, R1 = 10, R2 = 5, MEM[16] = 7.

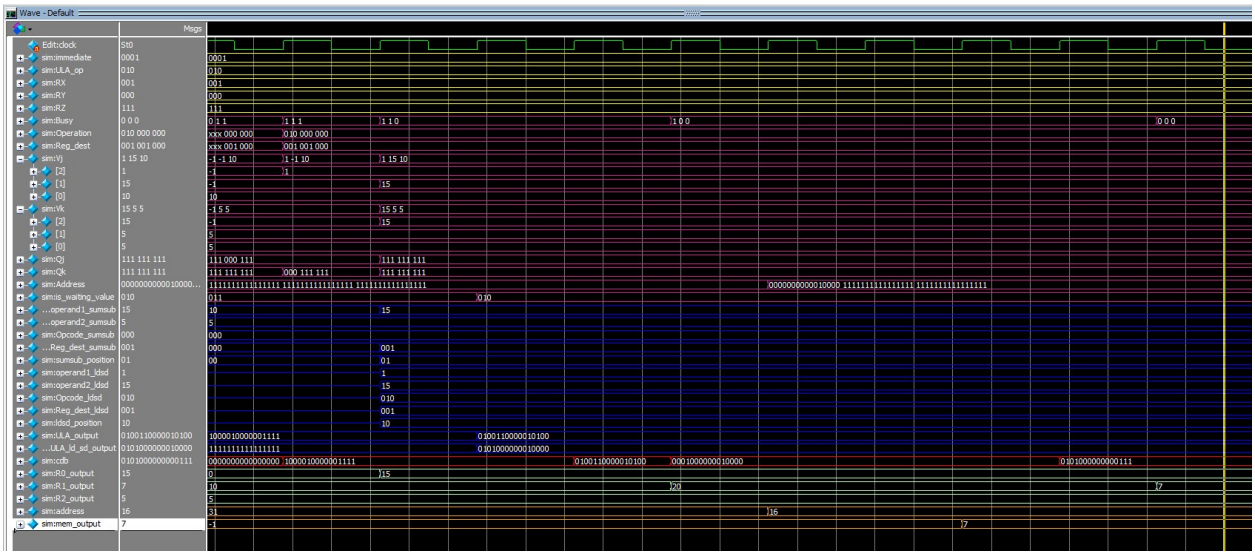
Valor Final dos Registradores: R0 = 15, R1 = 7, R2 = 5, MEM[16] = 7.

Execução: Nesse teste será demonstrado o atraso no CDB. Será executada inicialmente uma instrução de ADD que produz um novo valor para R0. Em seguida, serão despachadas duas instruções, ADD e LD, que tem dependência verdadeira com a primeira instrução, mas que utilizam ULAs diferentes para serem executadas. O objetivo deste teste é mostrar que quando as duas ULAs produzem resultado no mesmo ciclo, o CDB escolhe sempre o resultado proveniente da instrução que foi despachada primeiro, ou seja, o CDB realiza o adiamento.

Print da Simulação (Parte 1):



Print da Simulação (Parte 2):



Legenda de Cores:

Verde: Clock e Registradores
Amarelo: Sinais de Saída da Fila de Instruções
Rosa: Sinais da Estação de Reserva
Azul: Sinais de Entrada das ULAs
Vermelho: CDB
Laranja: Endereço acessado na memória e Valor retornado pela memória

Ciclo 1, 2, 3, 4: Nos quatro primeiros ciclos podemos observar o despacho do primeiro ADD, seu salvamento na Estação de Reserva, o envio para a ULA, a produção do resultado da ULA e o envio desse resultado para o CDB.

Ciclo 5: No ciclo cinco, o CDB atualiza a Estação de Reserva e o Banco de Registradores com os valores referentes à primeira instrução de ADD. Sendo assim, neste ciclo, as duas próximas instruções que estão na estação de reserva podem ser executadas. Analisando os sinais destacados de azul podemos perceber que tanto a instrução de ADD quanto a instrução de LD são despachadas para as respectivas ULAs neste mesmo ciclo.

Ciclo 6: No ciclo seis podemos observar que ambas as ULAs produzem resultados para as respectivas instruções.

Ciclo 7: No ciclo sete podemos então observar que o CDB recebe o dado referente a ULA de Soma e Subtração, pois a instrução de ADD foi despachada primeiro para a Estação de Reserva. Sendo assim, o CDB realiza o adiamento do recebimento do dado referente a ULA de Load e Store, que só irá acontecer no ciclo 8, quando a instrução de ADD for finalizada.

Ciclo 8 ao 13: Nos ciclos restantes podemos observar a execução completa da instrução de LD, e podemos perceber que o registrador R1 finaliza a execução com o valor correto (7).

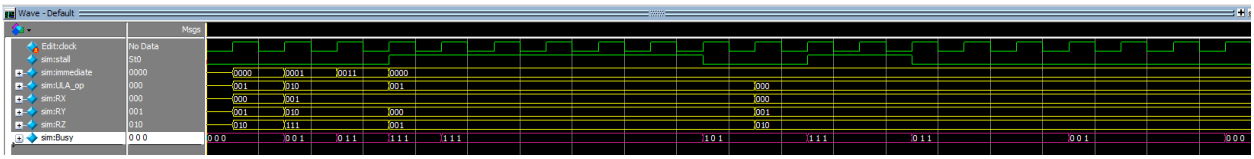
f) Adiamento na estação de reserva

Memória de Instruções:

```
// Inicializando a Memória de Instruções
InstructionMemory[0] <= 16'b0000001000001010; // SUB R0, R1, R2
InstructionMemory[1] <= 16'b0001010001010111; // LD R1, 1(R2)
InstructionMemory[2] <= 16'b0011010001010111; // LD R1, 3(R2)
InstructionMemory[3] <= 16'b0000001001000001; // SUB R1, R0, R1
InstructionMemory[4] <= 16'b0000000000001010; // ADD R0, R1, R2
```

Execução: Neste caso de teste não importa quais instruções foram executadas, o objetivo aqui é apenas demonstrar que quando a Estação de Reserva fica cheia, a Fila de Instruções para de despachar instruções novas. Para isso, adicionamos 5 instruções diferentes na Memória de Instruções para realizar o teste. No fim do teste fica claro que está ocorrendo corretamente o adiamento na Estação de Reserva.

Print da Simulação:



Legenda de Cores:

Verde: Clock e Sinal “Stall”
Amarelo: Sinais de Saída da Fila de Instruções
Rosa: Coluna “Busy” da Estação de Reserva

Ciclo 1, 2, 3: Nos três primeiros ciclos são despachadas as três primeiras instruções da Fila de Instruções. Podemos perceber isso analisando os sinais destacados de amarelo.

Ciclo 4 e 5: No ciclo quatro, analisando o sinal de “Busy”, podemos perceber que a Estação de Reserva está cheia, sendo assim, neste mesmo ciclo o sinal de “stall” recebe o valor 1. O sinal de “stall” é responsável por avisar a Fila de Instruções que a Estação de Reserva está cheia, para que as instruções parem de ser despachadas. É importante observar, que ainda no ciclo quatro, a quarta instrução da Fila de Instruções é despachada, pois o sinal de “stall” ainda não é 1, porém ela não é inserida na Estação, ela só será inserida no ciclo cinco, quando a primeira instrução sair da Estação. Podemos perceber esse comportamento analisando o sinal “Busy”.

Ciclo 10, 11, 12: No ciclo dez, uma das instruções presentes na Estação é finalizada e retirada, sendo assim, o sinal de “stall ” recebe o valor 0. Logo, no ciclo onze a quinta instrução da Fila de Instruções é despachada, e no ciclo doze ela é inserida na Estação de Reserva.

Destaque: Destacar que no ciclo 5 a primeira instrução finaliza sua execução e é retirada da Estação de Reserva (Semelhante ao que acontece no teste B), porém como a instrução seguinte já foi despachada pela fila de instruções, ela é adicionada na Estação de Reserva no mesmo ciclo em que a primeira instrução é retirada, por isso notamos que a coluna “Busy” se mantém com todos os bits iguais a 1. Já a quinta e última instrução só é despachada quando surge mais um espaço na Estação de Reserva.

5 - Extras

Por fim, foram adicionados dentro dos módulos “InstructionQueue” e “BankOfRegisters”, comentários com todos os valores de cada teste apresentado neste relatório. Isso foi feito para facilitar a execução de cada teste se necessário. Além disso, foi adicionado nesses comentários um teste extra (Teste G), que tem como objetivo apenas demonstrar o funcionamento da instrução de Store. Seguem os prints dos comentários.

Módulo da Instruction Queue:

```
// Teste A
// ADD R0, R1, R2 -> 0000000000001010
// SUB R1, R0, R1 -> 0000001001000001

// Teste B e D
// SUB R0, R1, R2 -> 0000001000001010
// ADD R0, R1, R2 -> 0000000000001010

// Teste C
// SUB R0, R1, R2 -> 0000001000001010
// LD R1, 1(R2)   -> 0001010001010111

// Teste E
// ADD R0, R1, R2 -> 0000000000001010
// ADD R1, R0, R2 -> 0000000001000010
// LD R1, 1(R0)   -> 0001010001000111

// Teste F
// SUB R0, R1, R2 -> 0000001000001010
// LD R1, 1(R2)   -> 0001010001010111
// LD R1, 3(R2)   -> 0011010001010111
// SUB R1, R0, R1 -> 0000001001000001
// ADD R0, R1, R2 -> 0000000000001010

// Teste G
// SD R2, 1(R1) -> 0001011010001111
// LD R0, 1(R1) -> 0001010000001111
```

Módulo do Bank Of Registers:

```
// Teste A
// Início -> R0 = 0, R1 = 7, R2 = 4
// Fim    -> R0 = 11, R1 = 4, R2 = 4

// Teste B e D
// Início -> R0 = 0, R1 = 10, R2 = 5
// Fim    -> R0 = 15, R1 = 10, R2 = 5

// Teste C
// Início -> R0 = 0, R1 = 10, R2 = 5, MEM[6] = 2
// Fim    -> R0 = 5, R1 = 2, R2 = 5, MEM[6] = 2

// Teste E
// Início -> R0 = 0, R1 = 10, R2 = 5, MEM[16] = 7
// Fim    -> R0 = 15, R1 = 7, R2 = 5, MEM[16] = 7

// Teste G
// Início -> R0 = 0, R1 = 10, R2 = 5, MEM[11] = 8
// Fim    -> R0 = 5, R1 = 10, R2 = 5, MEM[11] = 5
```