Aluno: João Victor Peluci e Ulisses Andrade Carvalho
Matrícula: 20213001488, 20223003341

CEFET-MG

Lab. AOC II

Prática III - Tomasulo

1 Introdução

O Algoritmo Tomasulo é utilizado para distribuição dinâmica de tarefas. Nele, utilizam-se despachos coordenados, execução e escrita nos registradores fora de ordem, a fim de otimizar o desempenho do processador e utilizar o máximo das unidades funcionais paralelamente.

2 Projeto

2.1 Unidades funcionais

2.1.1 Fila de Instruções

A fila de instruções foi implementada por meio de um vetor de 64 posicões instanciado junto de um Program Counter - PC. Cada instrução chamada na fila possui a seguinte disposição:

Opcode	Ri	Rj	Rk	Não utilizados
2 bits	3 bits	3 bits	3 bits	5 bits

Tabela 1: Instrução de tipo R

Opcode	Ri	Rk	Imediato
2 bits	3 bits	3 bits	8 bits

Tabela 2: Instrução de tipo I

Ao analisar a simulação, temos os sinais que indicam se a Estação de Reserva está cheia ou vazia. Se a ER estiver cheia, a instrução não é despachada.

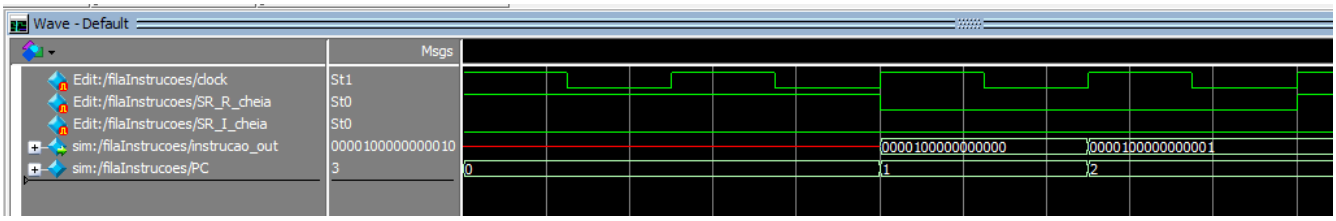

Figura 1: Simulação da Fila de Instruções

2.1.2 Banco de registradores

O banco de registradores é criado com 8 registradores, faz leitura de forma assíncrona e a escrita de forma síncrona por meio de um sinal que deve ser habilitado para atualizar o dado de um registrador.

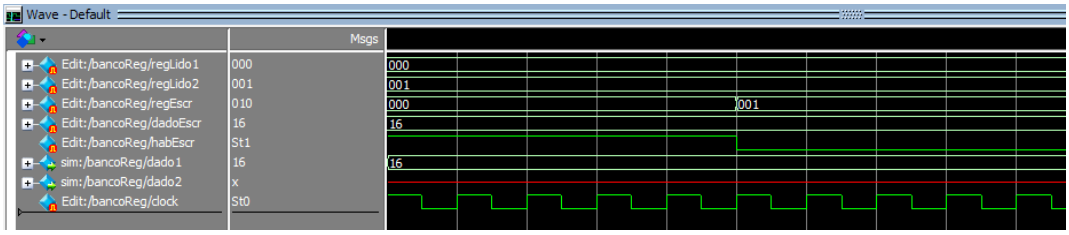

Figura 2: Simulação do Banco de Registradores

2.1.3 Unidade Funcional

A unidade funcional da implementação do Tomasulo dispõe de um sinal contador, que simula uma Unidade Funcional que, a cada 3 ciclos de clock, gera uma resposta de soma ou subtração, dependendo da operação de entrada. A UF também possui um sinal de controle usado para informar ao restante da implementação se o valor resultante já foi calculado.

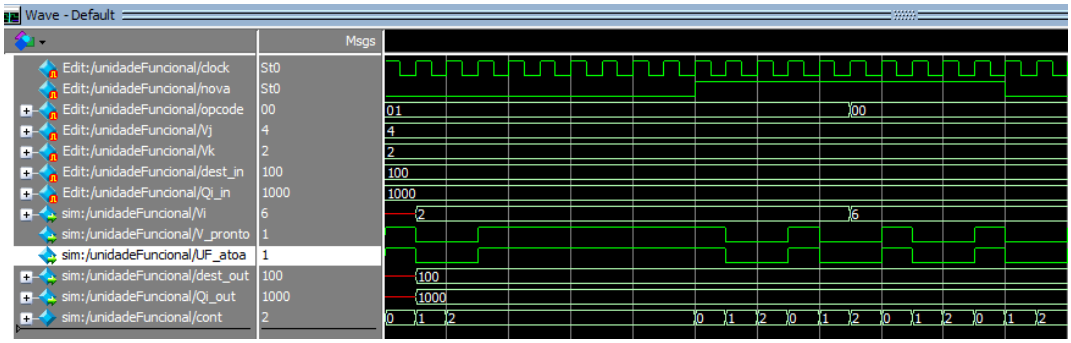

Figura 3: Simulação da Unidade Funcional

2.1.4 CDB/CDB Arbiter

O CDB funciona recebendo informações de adiantamento da memoria e da UF do tipo R e os organizando em uma fila de prioridade, de forma que apenas uma instrução é mandada para o CDB por ciclo de clock.

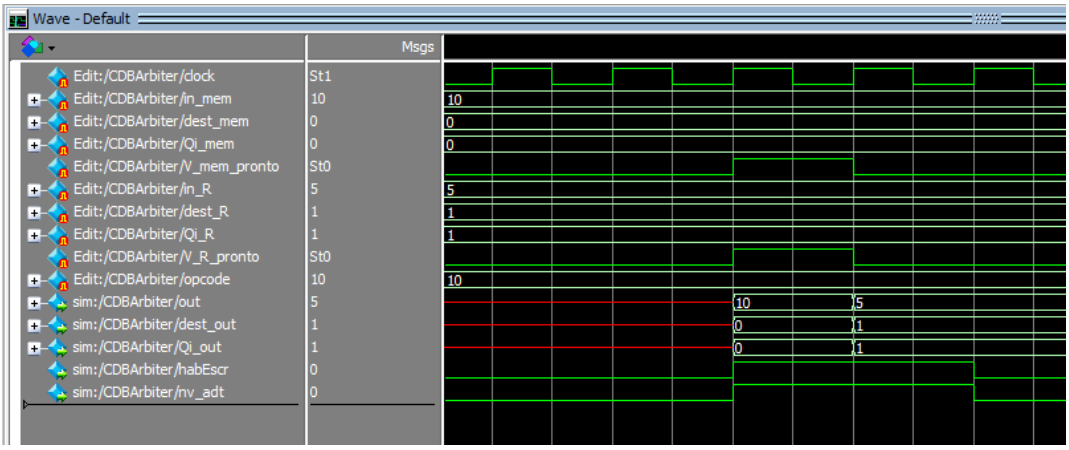

Figura 4: Simulação do CDB Arbiter

2.1.5 Memória de Dados

A memória de dados foi implementada usando a memória LPM e adiconando alguns sinais de controle que permitem informar a outros componentes quando uma leitura ou escrita foi finalizada.

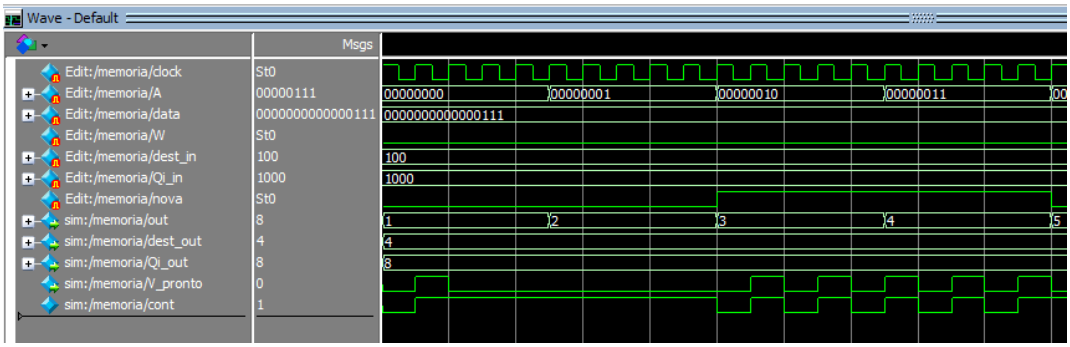

Figura 5: Simulação da Memória de Dados

2.1.6 Estação de Reserva

A Estação de Reserva possui 10 posições no total, as quais 5 são para instruções do tipo R e 5 para o tipo I. Os dois tipos de instruções podem ser armazenados em qualquer posição da Estação de Reserva, sendo diferenciados pelo seu opcode. O preenchimento é feito de acordo com a ordem de entrada em posições que não estão ocupadas.
Esse componente ainda gerência o encaminhamento das instruções as UF, a memória e ao CDB Arbiter, cada um feito a partir de certos sinais de controle e recebe e substitui os valores recebidos do CDB nos locais que estão o esperando.
As linhas da ER possuem as seguintes colunas:

Coluna	Função
Busy	Indica se a linha da ER está ocupada
Opcode	Indica a operação que será realizada
Vj	Armazena o valor do operando RX, se estiver pronto
Vk	Armazena o valor do operando RY, se estiver pronto
Qj	Armazena a label da ER que calculará o valor do registrador RX
Qk	Armazena a label da ER que calculará o valor do registrador RY
I	Guarda o imediato em operações do tipo I
A	Guarda o endereço calculado em operações do tipo I
Dest	Guarda o registrador destino da instrução
enc-UF	Indica o encaminhamento da UF
enc-mem	Indica o encaminhamento da Memoria

Tabela 3: Linhas da estação de reserva

A simulação do componente Estação de Reserva individualmente será omitida intencionalmente, pois será simulada com exemplos de dependências e hazards.

2.2 Esquemático

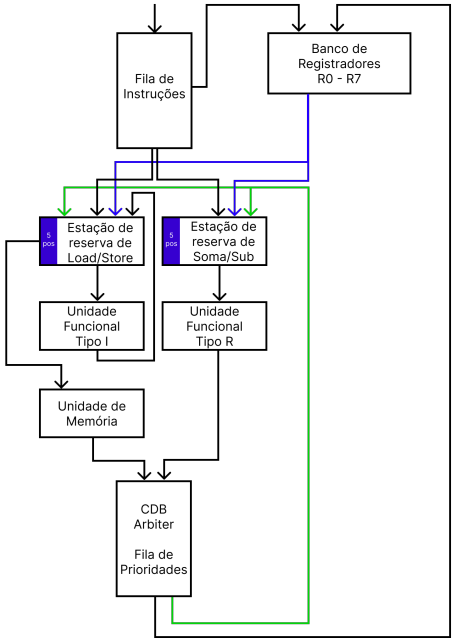

Figura 6: Esquemático do Projeto

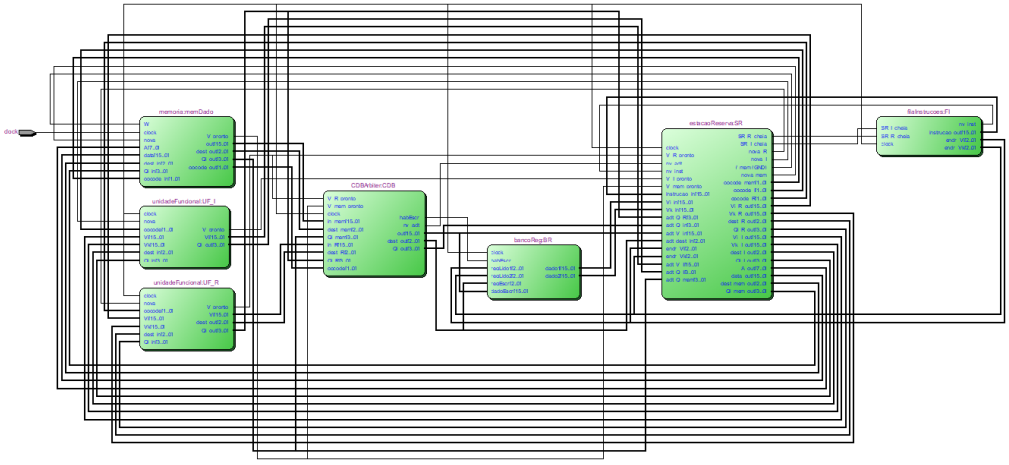

Figura 7: Esquemático do Projeto em Verilog HDL

3 Simulação

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

A dependência de dados verdadeira acontece quando um registrador cujo dado ainda está sendo escrito é usado por uma instrução subsequente.

```
1       LD  R0, 4 (R7)
2       LD  R1, 9 (R7)
3       ADD R2, R2, R0
```

No código acima, há uma dependência de dados entre as instruções 3 e 1. Sendo assim, todas as instruções são colocadas na estação de reserva, mas a terceira instrução só é executada após o cálculo do valor que será carregado em R0.
Exemplo:

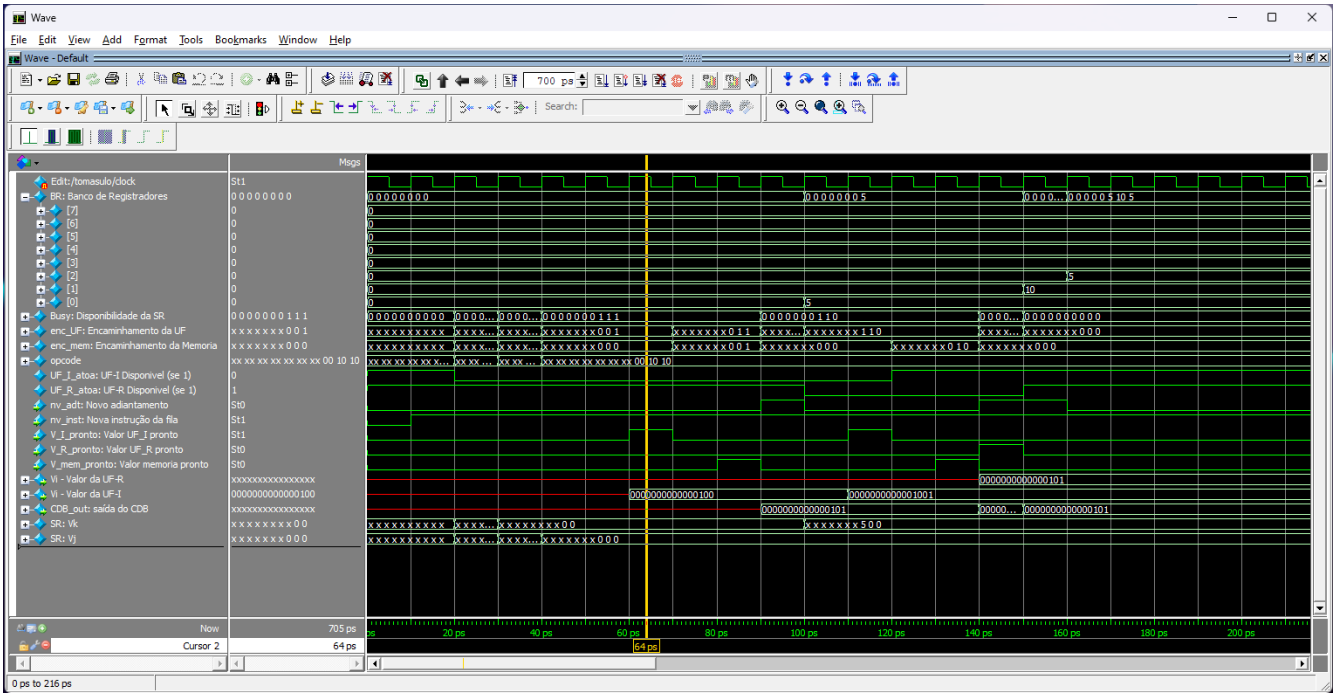

Figura 8: Simulação de RAW

Para interpretar os dados acima:
A simulação acima mostra a estação de reserva sendo preenchida (verificar os busy sendo atualizados e posteriormente sendo liberados). Por outra perspectiva, acompanhando os valores do Banco de Registradores, vemos que o primeiro dado a ser carregado é da primeira instrução, sem dependências. Depois, a segunda instrução espera a liberação da UF-I para executar. Vemos também que no momento em que o dado 5 é escrito no registrador R0, a UF-R torna ocupada, pois a instrução de soma já estava na estação de reserva aguardando o dado ficar pronto para ser executada. Contando os três ciclos de clock, a escrita da operação de soma só ocorre após a liberação do CDB pela instrução 2.

3.2 Dependência de saída ou WAW (Write After Write)

A dependência de saída ou WAW (Write After Write) acontece quando duas ou mais instruções estão escrevendo no mesmo registrador.

```
1    LD R0, 4 (R7)
2    LD R1, 4 (R7)
3    LD R2, 4 (R7)
4    ADD R2, R0, R0
5    SUB R2, R0, R0
```

Como podemos ver acima, o registrador R2 é escrito nas instruções 3, 4 e 5.
Exemplo: Para interpretar os dados acima:

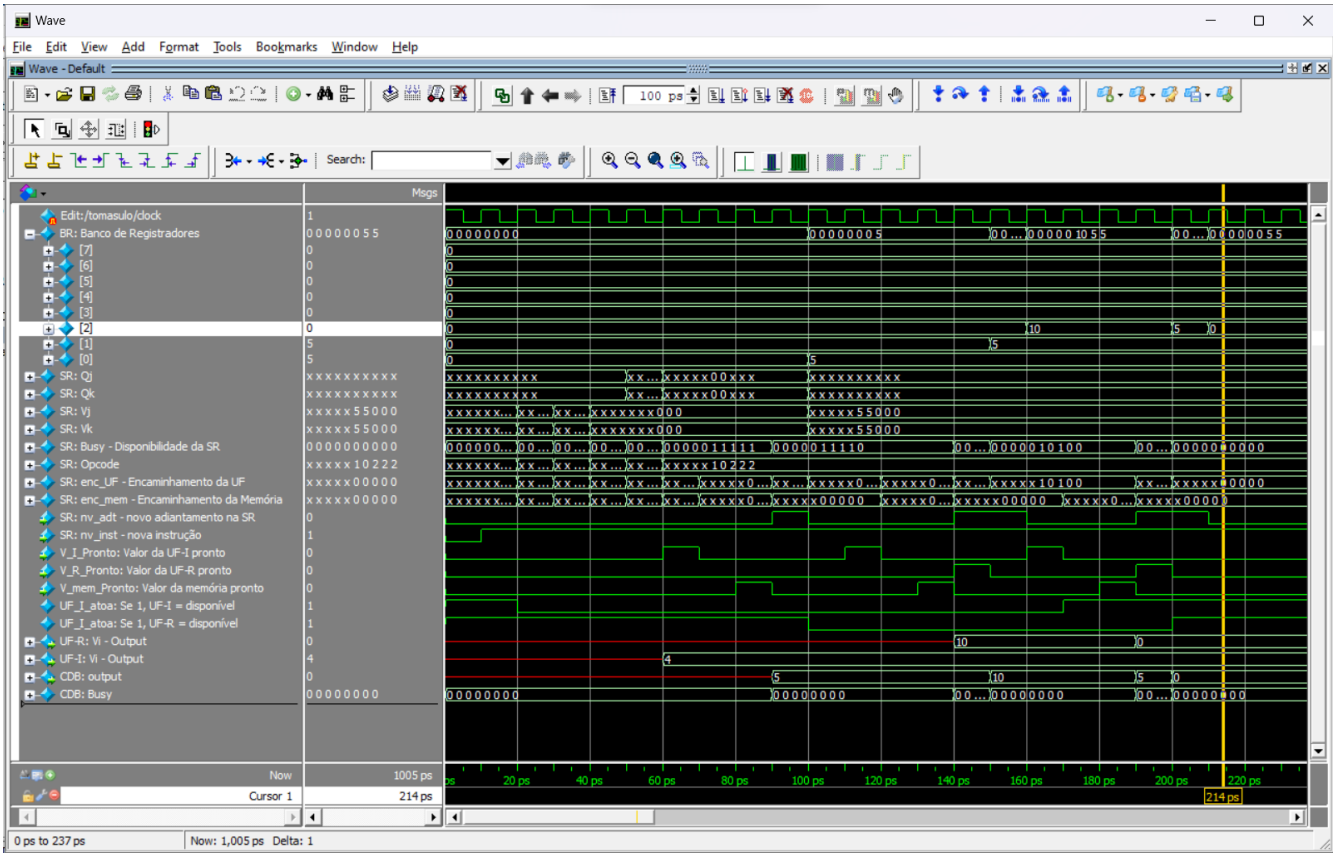

Figura 9: Simulação de WAW

Vemos que a implementação feita do algoritmo Tomasulo contorna a dependência de saída por meio da renomeação dos registradores e posteriomente pelo adiamento da escrita feito pelo CDB. A implementação de Tomasulo feita controla a escrita no banco de registradores por meio de uma fila de prioridades. Quando o CDB está ocupado, a fila adiciona um novo elemento, assim que está liberado, o primeiro item da fila é escrito.

3.3 Antidependência ou WAR (Write After Read)

A antidependência ou WAR (Write After Read) acontece quando uma instrução está escrevendo em um registrador que está sendo lido por uma instrução anterior. O Tomasulo resolve essa antidependência anotando os valores dos registradores na estação de reserva assim que ocorre o despacho.

```
1    LD R0, 0 (R7) -> 1
2    LD R1, 1 (R7) -> 2
3    LD R2, 4 (R7) -> 3
4    ADD R3, R0, R1 -> 3
5    SUB R0, R2, R2 -> 0
```

As instruções 1, 2 e 3 não possuem dependências, a não ser hazards estruturais. A instrução 5 escreve em um registrador que está sendo lido na instrução 4.
Exemplo:

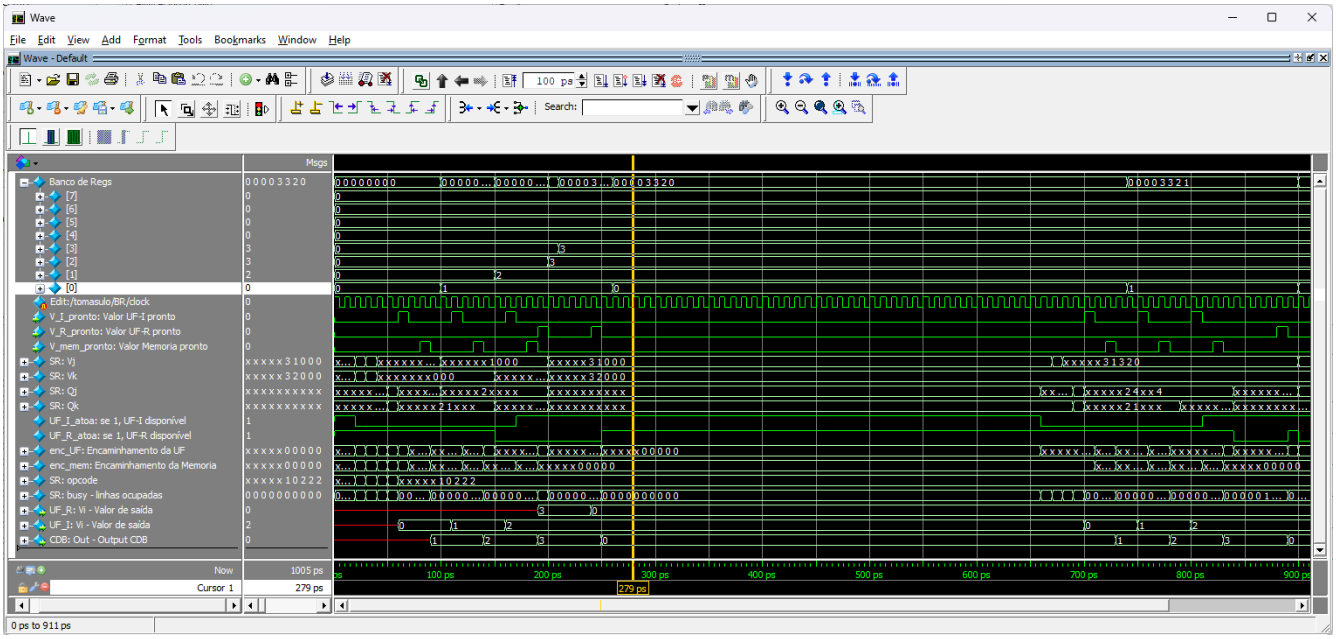

Figura 10: Simulação de WAR

Observando os valores de Vj e Vk nessa simulação, visualiza-se que eles guardam o valor dos registradores no momento em que a instrução foi despachada, permitindo, assim que ocorra o despacho das instruções seguintes sem afetar o resultado. Para interpretar a simulação acima, basta visualizar que Vj e Vk são indexados como vetores que armazenam os dados

3.4 Dependência/hazard estrutural (STALL - unidade funcional cheia)

A dependência estrutural acontece quando duas ou mais instruções precisam usar a mesma unidade funcional ao mesmo tempo.

```
1    LD R0, 0 (R7) -> 1
2    LD R1, 1 (R7) -> 2
3    LD R2, 4 (R7) -> 3
4    ADD R3, R0, R1 -> 3
5    SUB R0, R2, R2 -> 0
```

Como só temos uma unidade funcional para as operações de soma, subtração e cálculo de endereço, instruções que compartilham a mesma unidade funcional devem aguardar o término da instrução anterior para usar a unidade funcional.
Exemplo:

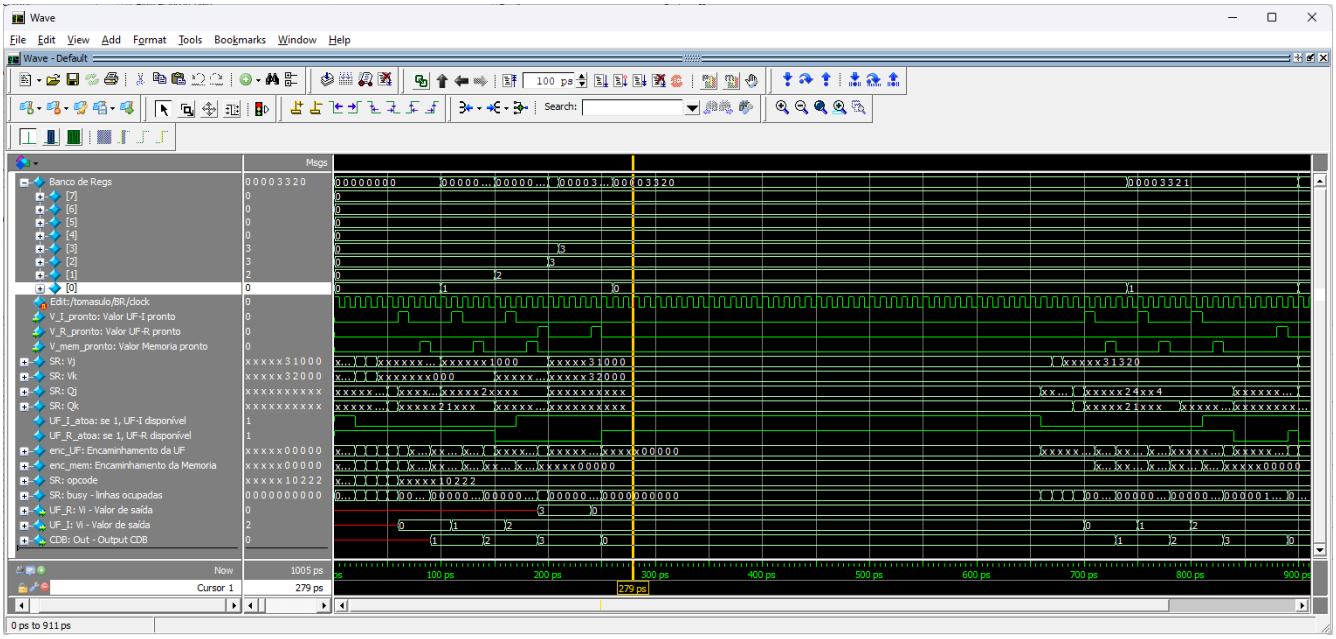

Figura 11: Simulação de Hazard Estrutural

Na simulação acima, vemos que o sinal UF-I disponível como 0 no início da simulação. Isso acontece porque, sempre que a UF deveria ser dada como disponível, havia uma instrução aguardando para utilizá-la, demonstrando que há eficiência na execução. Nos ciclos seguintes, como as instruções utilizam a UF-R, é possível ver que, dessa vez, essa se encontra ocupada.

3.5 Conflito no CDB

O CDB (Control Data Bus) é um barramento que transporta os dados entre as unidades funcionais. Um conflito no CDB acontece quando duas ou mais instruções precisam colocar um dado no CDB ao mesmo tempo. Para isso, usamos o CDB Arbiter. A simulação de conflito no CDB é mais facilmente compreendida observando a simulação da seção 2.1.4, na qual é possível ver que os sinais de controle *V-mem-pronto* e *V-R-Pronto* estão ativos, mas a saída *out* recebe um dos valores de cada vez.

3.6 Adiantamento na estação de reserva

O adiantamento na estação de reserva é uma técnica que permite que informações sejam disponibilizadas antes do dado ficar pronto no Banco de Registradores. Na simulação 3.2, pode-se visualizar que ocorre um adiantamento na ER quando o dado que é utilizado pela instrução 4 é disponibilizado pela instrução 1 (vide simulação no momento imediatamente anterior à escrita do valor 5 no registrador R0), pode-se verificar o funcionamento desse adiantamento pois o sinal de *nv-adt* fica ativado e no ciclo seguinte a UF-R já se torna ocupada - sinal de *UF-R-atoa* = 0 - indicando que a operação começa a ser executada.

3.7 Renomeação de registradores

A renomeação de registradores é uma técnica que permite que registradores sejam renomeados para que duas ou mais instruções possam usar o mesmo registrador sem causar dependências. Na simulação 3.2, ocorre uma renomeação de registradores, pois as instruções 4 e 5 dependem do dado que está sendo escrito na instrução 1. Essa renomeação de registradores ocorre por meio do uso das colunas Qj e Qk na estação de reserva, que recebe os valores 0 nas posições 4 e 5 do vetor Qj e Qk. Este valor 0 indica que as instruções 4 e 5 estão aguardando o resultado que será gerado no índice 0 da estação de reserva.

4 Dificuldades, sugestões e comentários

4.1 Sugestões:

Poderia ser utilizada a ferramenta de apresentação em vídeo para substituir a apresentação presencial e o relatório.