Relatório Prática 2 - Processador Multiciclo

Nomes: Abdul-Kevin Alexis e Pedro Santos Oliveira

1 - **Decisões Gerais de Projeto**

a) Uma vez que a memória para dados e para instruções é única, dividimos a memória do nosso projeto da seguinte maneira: posições de 0 a 9 são destinadas para dados e posições de 10 em diante são destinadas para instruções. Por esse motivo o PC começa com o valor 10, onde está localizada a primeira instrução na memória.

b) Como o registrador ADDR sempre recebe o valor salvo em PC no início de uma nova execução, optamos por atualizar o registrador ADDR junto com o registrador referente a PC, pois assim, não perdemos nenhum ciclo de clock para transferir o valor de PC para ADDR no início de uma nova execução, e a lógica se mantém coesa. Logo, sempre que uma instrução for finalizada e o sinal de incremento do PC receber 1, tanto o PC quanto o registrador ADDR terão seus valores atualizados para PC + 1. Para isso, foi implementada uma variável local no módulo do registrador ADDR que salva o valor de PC.

c) Instruções que acessam dados na memória (LD e MVI) retornam dados presentes na memória para a entrada DIN, e sempre que o DIN recebe algo ele faz o salvamento dos valores dos registradores nas suas respectivas variáveis, ou seja, pode acontecer de os valores dos registradores, que salvam o registrador de leitura e de destino, sejam alterados durante a execução da instrução. Para evitar que isso aconteça criamos dois sinais que dizem ao processador se a instrução executada é uma instrução que busca dados da memória ou não, esses sinais são os sinais *is_Load* e *is_mvi*, que evitam que ocorra alguma mudança nos sinais dos registradores durante a execução de um Load ou de um MVI, garantindo que os registradores de leitura e destino vão se manter corretos.

d) Time Steps: O Time Step 0 é utilizado apenas para resetar alguns sinais usados durante a execução das instruções, por exemplo o sinal *ALUop*, e para habilitar a escrita em ADDR, que sempre recebe o valor de PC no início da execução. O Time Step 1 é usado apenas para esperar a instrução ser buscada na memória e colocada em DIN. O Time Step 2 é usado para habilitar a escrita da instrução no registrador IR. Por fim, do Time Step 3 em diante ocorrem as execuções das instruções. Instruções que utilizam a ULA iniciam sua execução no Time Step 3 e finalizam no Time Step 5, instruções de "*Move*" que não utilizam a memória (MV e MVNZ) iniciam a execução no Time Step 3 e finalizam no Time Step 4, e por último, as instruções que acessam a memória iniciam sua execução no Time Step 3 e finalizam no Time Step 6, no caso de LD e SD, e no Time Step 7, no caso do MVI.

e) Em relação a instrução MVI, que move um imediato para um registrador, optamos por posicionar esse imediato na posição subsequente de memória em relação a posição em que está a instrução MVI, por exemplo, caso a instrução esteja na posição 10, o imediato estará na posição 11.

f) Em relação a instrução LD, optamos por criar uma variável local na unidade de controle chamada *r_dest*, essa variável tem como objetivo salvar o valor do registrador de destino da instrução LD, para garantir que o valor será salvo no devido registrador. Essa decisão foi tomada quando realizamos testes na FPGA e observamos que o valor do registrador de destino estava sendo perdido durante a execução da instrução.

2 - **DIN, Sinais da ULA e Sinais de Simulação**

a) Disposição dos bits de DIN

DIN possui ao todo 16 bits, dividimos esses 16 bits da seguinte forma:

DIN[15:10]	DIN[9:6]	DIN[5:3]	DIN[2:0]
Não foram utilizados	Opcode da instrução	Registrador X	Registrador Y

b) Sinais de operação da ULA

A ULA possui ao todo 6 operações, a seguir mostramos os sinais para cada operação:

Operação	Sinal da operação na ULA (*ALUop*)
ADD	000
SUB	001
OR	010
SLT	101
SLL	011
SLR	111

c) Para as simulações realizadas no projeto e expostas nesse relatório utilizamos como entrada um sinal de clock com as seguintes características: **duty value** = 50ns, **initial value** = 0, **clock period** = 50ns.

3 - **Instruções utilizadas para os testes**

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
ADD R1, R0	000000	0000	001 (R1)	000 (R0)	8	8
SUB R0, R1	000000	0001	000 (R0)	001 (R1)	65	41
MV R3, R1	000000	0110	011 (R3)	001 (R1)	409	199
MVI R0, #2	000000	0111	000 (R0)	000	448	1C0
OR R0, R1	000000	0010	000 (R0)	001 (R1)	129	81
SLT R0, R1	000000	0011	000 (R0)	001 (R1)	193	C1
SLL R1, R2	000000	0100	001 (R1)	010 (R2)	266	10A
SLR R0, R1	000000	0101	000 (R0)	001 (R1)	321	141
MVNZ R0, R1	000000	1010	000 (R0)	001 (R1)	641	281
MVNZ R0, R1	000000	1010	000 (R0)	001 (R1)	641	281
SD R0, R1	000000	1001	000 (R0)	001 (R1)	577	241
LD R2, R1	000000	1000	010 (R2)	001 (R1)	529	211

4 - **Testes**

a) Para fins de facilitar a testagem das instruções, os registradores R0, R1, R2 e G foram inicializados com valores diferentes de 0 durante a execução dos testes. Para cada teste está indicado em “Execução” os valores iniciais dos registradores utilizados naquele teste. No código final do projeto todos os registradores são inicializados com o valor 0 como foi pedido no trabalho.

b) O sinal *RNin* mostrado na simulação indica o registrador onde será realizada a escrita do dado, enquanto o sinal *RNout* indica de qual registrador o dado será lido.

c) Para cada teste separamos as explicações em 4 partes, sendo elas: **Instrução Executada**, **Memória RAM**, **Execução** e **Simulação**. Nas primeiras duas partes são mostradas a composição da instrução e da memória no momento da execução, já nas últimas duas partes é onde acontece a explicação da execução da instrução, assim como as explicações adicionais sobre a simulação.

4.1 - **ADD**

Instrução Executada:

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
ADD R1, R0	000000	0000	001 (R1)	000 (R0)	8	8

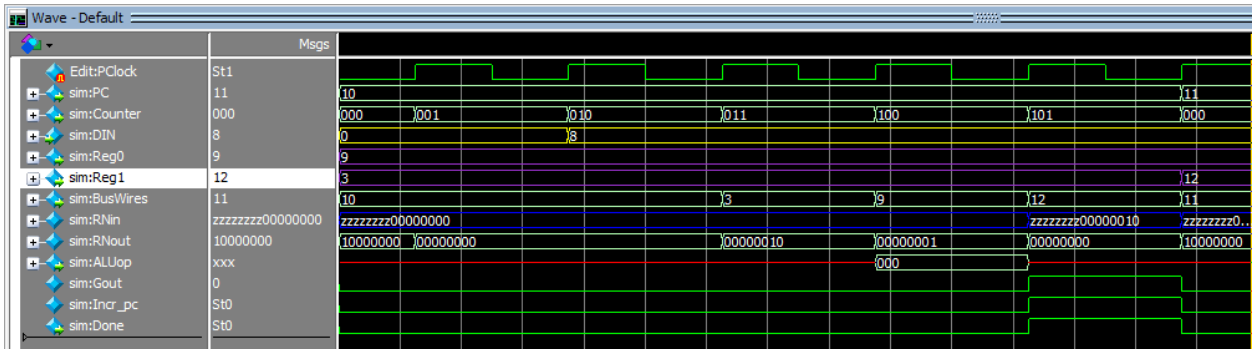
Memória RAM:

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

Essa foto ilustra a disposição da memória no momento do teste, indicando que o valor 8 está na posição 10, ou seja, o PC começa apontando para a instrução ADD R1, R0.

Execução: O teste em questão executa uma instrução de ADD envolvendo os registradores R1 e R0, ou seja, o correto funcionamento desse teste acarretará na soma dos valores presentes em R1 e R0, que são respectivamente 3 e 9, e acarretará também no salvamento da soma desses valores no registrador R1. Portanto, ao fim da execução R1 receberá o valor 12. (Inicialização dos Registradores: R0 = 9, R1 = 3)

Simulação:



As linhas Roxas indicam os registradores R0 e R1, enquanto a linha amarela indica o DIN, que recebe o valor 8 em decimal, valor esse referente a instrução ADD R1, R0 como mostrado acima.

4.2 - **SUB**

Instrução Executada:

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
SUB R0, R1	000000	0001	000 (R0)	001 (R1)	65	41

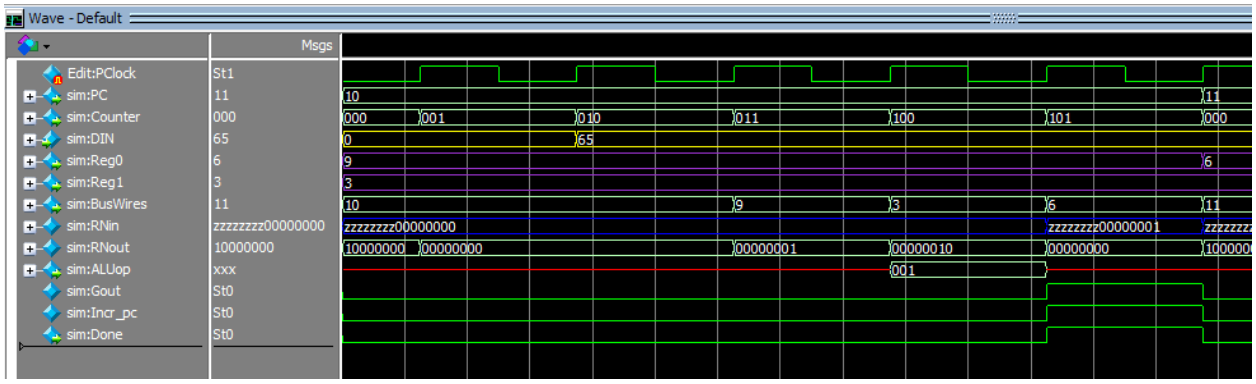
Memória RAM:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	5	7	8	4	0	0	0	0	.....
8	0	0	65	0	0	0	0	0	.A....
16	0	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....

Essa foto ilustra a disposição da memória no momento do teste, indicando que o valor 65 está na posição 10, ou seja, o PC começa apontando para a instrução SUB R0, R1.

Execução: O teste em questão executa uma instrução de SUB envolvendo os registradores R0 e R1, ou seja, o correto funcionamento desse teste acarretará na subtração dos valores presentes em R0 e R1, que são respectivamente 9 e 3, e acarretará também no salvamento dessa subtração no registrador R0. Portanto, ao fim da execução R0 receberá o valor 6. (Inicialização dos Registradores: R0 = 9, R1 = 3)

Simulação:



As linhas Roxas indicam os registradores R0 e R1, enquanto a linha amarela indica o DIN, que recebe o valor 65 em decimal, valor esse referente a instrução SUB R0, R1 como mostrado acima.

4.3 - **MV**

Instrução Executada:

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
MV R3, R1	000000	0110	011 (R3)	001 (R1)	409	199

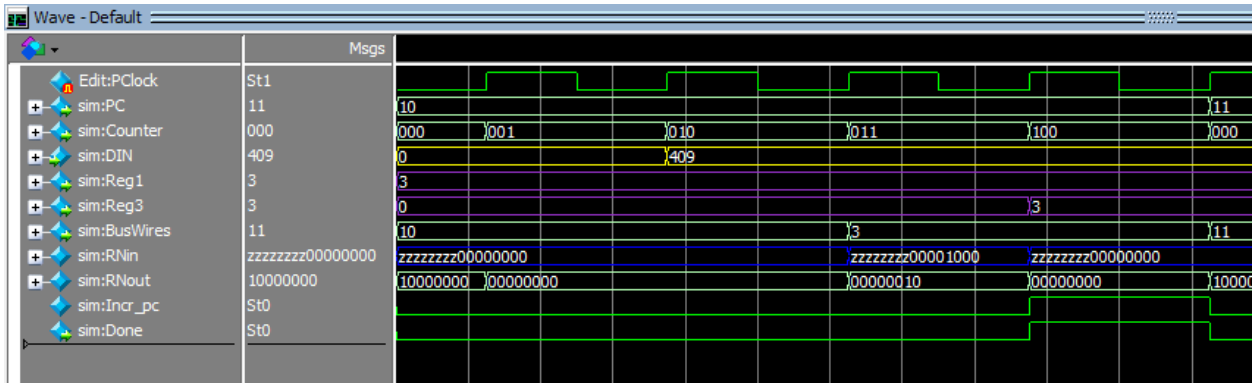
Memória RAM:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	5	7	8	4	0	0	0	0	.....
8	0	0	409	0	0	0	0	0	.....
16	0	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....

Essa foto ilustra a disposição da memória no momento do teste, indicando que o valor 409 está na posição 10, ou seja, o PC começa apontando para a instrução MV R3, R1.

Execução: O teste em questão executa uma instrução de MV envolvendo os registradores R3 e R1, ou seja, o correto funcionamento desse teste acarretará na movimentação do valor salvo no registrador R1, que contém o valor 3, para o registrador R3, que inicialmente contém o valor 0. Portanto, ao fim da execução R3 receberá o valor 3. (Inicialização dos Registradores: R1 = 3, R3 = 0)

Simulação:



As linhas Roxas indicam os registradores R1 e R3, enquanto a linha amarela indica o DIN, que recebe o valor 409 em decimal, valor esse referente a instrução MV R3, R1 como mostrado acima.

4.4 - **MVI**

Instrução Executada:

Instrução	DIN[15:10]	Opcode	RX	RY (Não Utilizado)	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
MVI R0, #2	000000	0111	000 (R0)	000	448	1C0

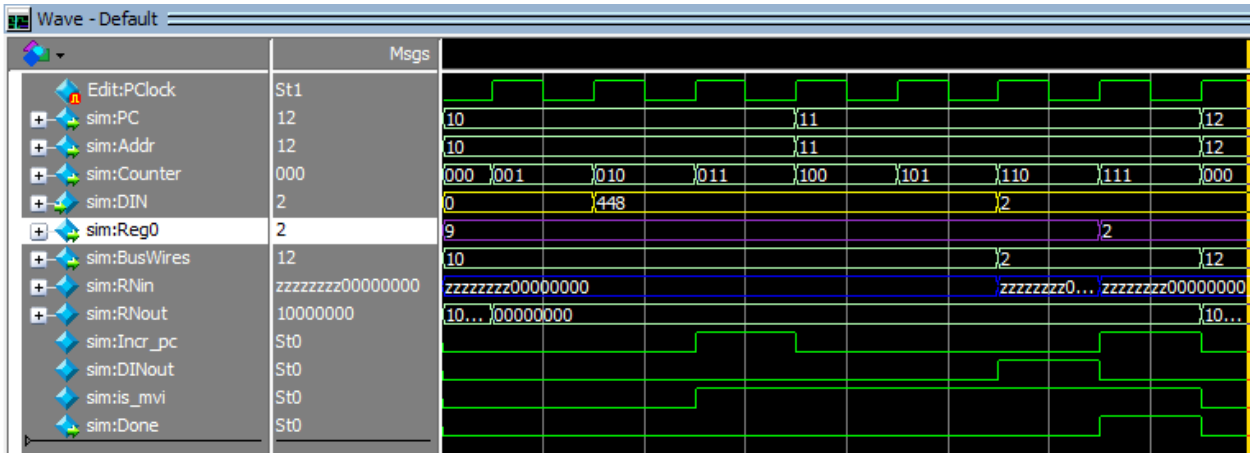
Memória RAM:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	5	7	8	4	0	0	0	0	.....
8	0	0	448	2	0	0	0	0	.....
16	0	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....

Essa foto ilustra a disposição da memória no momento do teste, indicando que o valor 448 está na posição 10, e que o imediato 2 está na posição 11, ou seja, o PC começa apontando para a instrução MVI R0, #2.

Execução: O teste em questão executa uma instrução de MVI envolvendo o registrador R0 e o imediato #2, ou seja, o correto funcionamento desse teste acarretará na movimentação do valor imediato #2 para o registrador R0, que inicialmente contém o valor 9. Portanto, ao fim da execução R0 receberá o valor 2. (Inicialização dos Registradores: R0 = 9)

Simulação:



A linha Roxa indica o registrador R0, enquanto a linha amarela indica o DIN, que recebe o valor 448 em decimal, valor esse referente a instrução MVI R0, #2, e posteriormente recebe o valor 2, valor esse que é o imediato usado pela instrução. Além disso, vale a pena destacar o sinal de *Incr_pc*, que incrementa no meio da execução por conta de o imediato estar na posição subsequente da instrução MVI, e vale a pena destacar também o sinal *is_mvi*, que indica que a instrução é do tipo MVI e impede que a chegada do imediato em DIN altere o valor dos registradores durante a execução da instrução.

4.5 - **SLT**

Instrução Executada:

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
SLT R0, R1	000000	0011	000 (R0)	001 (R1)	193	C1

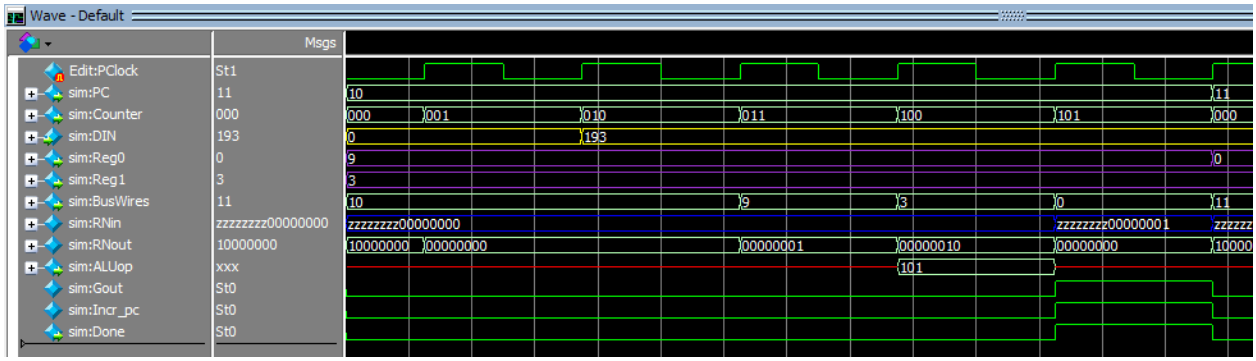
Memória RAM:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	5	7	8	4	0	0	0	0	.....
8	0	0	193	0	0	0	0	0	.....
16	0	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....

Essa foto ilustra a disposição da memória no momento do teste, indicando que o valor 193 está na posição 10, ou seja, o PC começa apontando para a instrução SLT R0, R1.

Execução: O teste em questão executa uma instrução de SLT envolvendo os registradores R0 e R1, ou seja, o correto funcionamento desse teste acarretará na checagem pela ULA se o valor contido em R0 é menor do que o valor contido em R1, caso isso seja verdade, o valor 1 é salve em R0, caso contrário o valor 0 é salvo em R0. Nesse caso, como inicialmente R0 contém 9 e R1 contém 3, ao final da execução R0 receberá o valor 0. (Inicialização dos Registradores: R0 = 9, R1 = 3)

Simulação:



As linhas Roxas indicam os registradores R0 e R1, enquanto a linha amarela indica o DIN, que recebe o valor 193 em decimal, valor esse referente a instrução SLT R0, R1 como mostrado acima. Além disso, vale a pena destacar o sinal *ALUop* que recebe o valor 101 referente a operação de comparação necessária para a execução da instrução SLT.

4.6 - **SLL**

Instrução Executada:

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
SLL R1, R2	000000	0100	001 (R1)	010 (R2)	266	10A

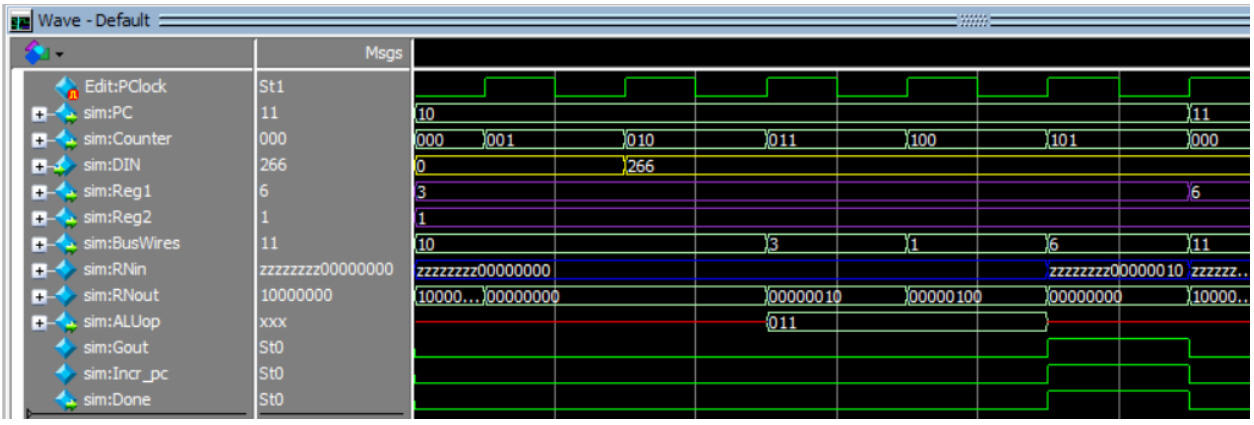
Memória RAM:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	5	7	8	4	0	0	0	0	.....
8	0	0	266	0	0	0	0	0	.....
16	0	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....

Essa foto ilustra a disposição da memória no momento do teste, indicando que o valor 266 está na posição 10, ou seja, o PC começa apontando para a instrução SLL R1, R2.

Execução: O teste em questão executa uma instrução de SLL envolvendo os registradores R1 e R2, ou seja, o correto funcionamento desse teste acarretará na operação de *Shift Logical Left* realizada pela ULA, que resultará em um *Shift Logical Left* do valor salvo em R1 pelo número de bits indicado pelo valor salvo em R2. Nesse caso, o valor contido em R1, que é inicialmente 3, sofrerá um *Shift Logical Left* de 1 bit, sendo que 1 é o valor salvo em R2, logo, isso resultará no salvamento do valor 6 em R1 (3 << 1 = 6). (Inicialização dos Registradores: R1 = 3, R2 = 1)

Simulação:



As linhas Roxas indicam os registradores R1 e R2, enquanto a linha amarela indica o DIN, que recebe o valor 266 em decimal, valor esse referente a instrução SLL R1, R2 como mostrado acima. Além disso,

vale a pena destacar o sinal *ALUop* que recebe o valor 011 referente a operação de *Shift Logical Left* realizada pela ULA.

4.7 - **SLR**

Instrução Executada:

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
SLR R0, R1	000000	0101	000 (R0)	001 (R1)	321	141

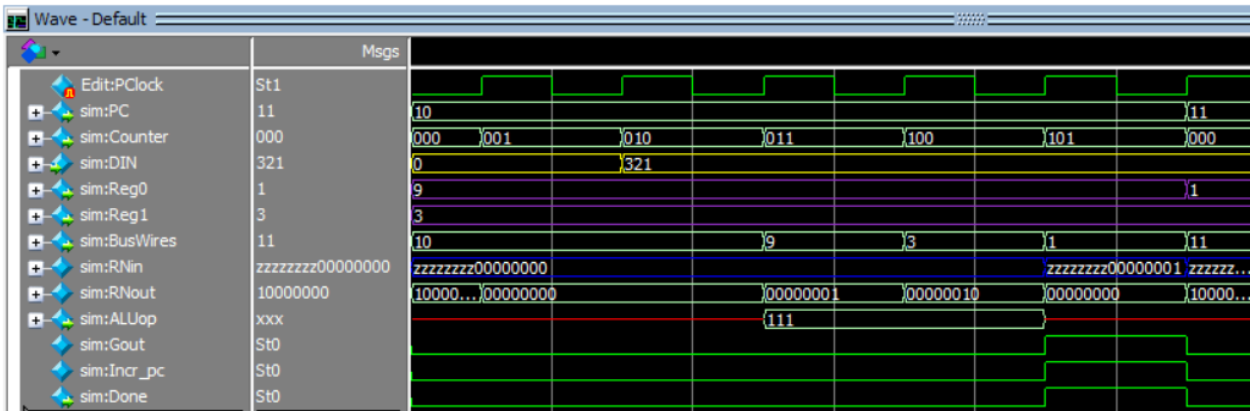
Memória RAM:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	5	7	8	4	0	0	0	0	.....
8	0	0	321	0	0	0	0	0	.....
16	0	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....

Essa foto ilustra a disposição da memória no momento do teste, indicando que o valor 321 está na posição 10, ou seja, o PC começa apontando para a instrução SRL R0, R1.

Execução: O teste em questão executa uma instrução de SRL envolvendo os registradores R0 e R1, ou seja, o correto funcionamento desse teste acarretará na operação de *Shift Logical Right* realizada pela ULA, que resultará em um *Shift Logical Right* do valor salvo em R0 pelo número de bits indicado pelo valor salvo em R1. Nesse caso, o valor contido em R0, que é inicialmente 9, sofrerá um *Shift Logical Right* de 3 bits, sendo que 3 é o valor salvo em R1, logo, isso resultará no salvamento do valor 1 em R0 (9 >> 3 = 1). (Inicialização dos Registradores: R0 = 9, R1 = 3)

Simulação:



As linhas Roxas indicam os registradores R0 e R1, enquanto a linha amarela indica o DIN, que recebe o valor 321 em decimal, valor esse referente a instrução SRL R0, R1 como mostrado acima. Além disso, vale a pena destacar o sinal *ALUop* que recebe o valor 111 referente a operação de *Shift Logical Right* realizada pela ULA.

4.8 - **OR**

Instrução Executada:

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
OR R0, R1	000000	0010	000 (R0)	001 (R1)	129	81

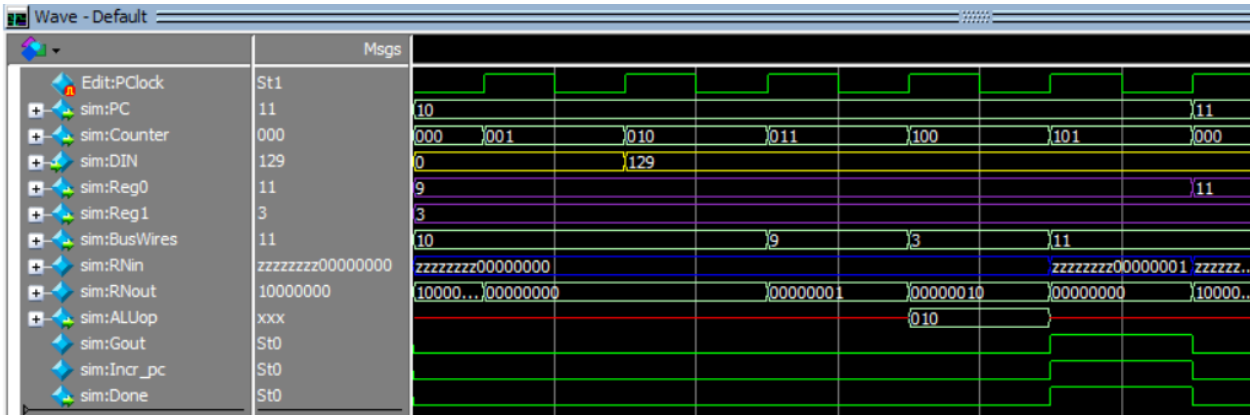
Memória RAM:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	5	7	8	4	0	0	0	0	.....
8	0	0	129	0	0	0	0	0	.....
16	0	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....

Essa foto ilustra a disposição da memória no momento do teste, indicando que o valor 129 está na posição 10, ou seja, o PC começa apontando para a instrução OR R0, R1.

Execução: O teste em questão executa uma instrução de OR envolvendo os registradores R0 e R1, ou seja, o correto funcionamento desse teste acarretará na operação lógica de *OR* realizada pela ULA, que terá seu resultado salvo em R0. Nesse caso, o valor contido em R0, que é inicialmente 9, será usado para uma operação de *OR* com o valor de R1, que inicialmente é 3, resultando no valor 11, que será salvo em R0. (Inicialização dos Registradores: R0 = 9, R1 = 3)

Simulação:



As linhas Roxas indicam os registradores R0 e R1, enquanto a linha amarela indica o DIN, que recebe o valor 129 em decimal, valor esse referente a instrução OR R0, R1 como mostrado acima. Além disso, vale a pena destacar o sinal *ALUop* que recebe o valor 010 referente a operação de *OR* realizada pela ULA.

4.9 - **MVNZ (Usando G = 0)**

Instrução Executada:

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
MVNZ R0, R1	000000	1010	000 (R0)	001 (R1)	641	281

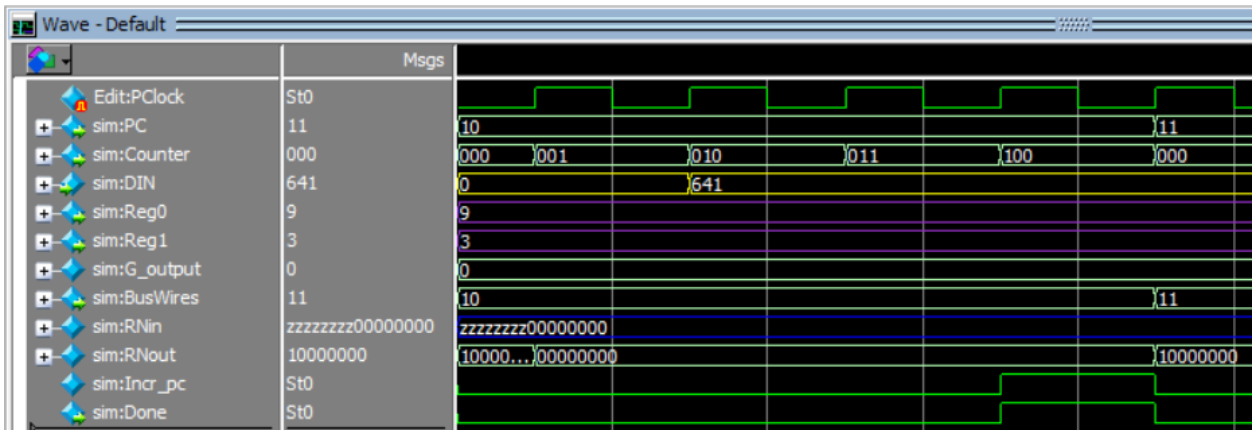
Memória RAM:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	5	7	8	4	0	0	0	0	.....
8	0	0	641	0	0	0	0	0	.....
16	0	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....

Essa foto ilustra a disposição da memória no momento do teste, indicando que o valor 641 está na posição 10, ou seja, o PC começa apontando para a instrução MVNZ R0, R1.

Execução: O teste em questão executa uma instrução de MVNZ envolvendo os registradores R0 e R1, ou seja, o correto funcionamento desse teste acarretará na movimentação do valor de R1 para R0 caso o valor do registrador G seja diferente de 0. Nesse caso, como G possui o valor 0, o registrador R0 não terá seu valor alterado. (Inicialização dos Registradores: R0 = 9, R1 = 3, G = 0)

Simulação:



As linhas Roxas indicam os registradores R0 e R1, enquanto a linha amarela indica o DIN, que recebe o valor 641 em decimal, valor esse referente a instrução MVNZ R0, R1 como mostrado acima. Além disso, vale a pena destacar o sinal *G_output* que mostra qual o valor presente no registrador G no momento em que a instrução é executada.

4.10 - **MVNZ (Usando G = 1)**

Instrução Executada:

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
MVNZ R0, R1	000000	1010	000 (R0)	001 (R1)	641	281

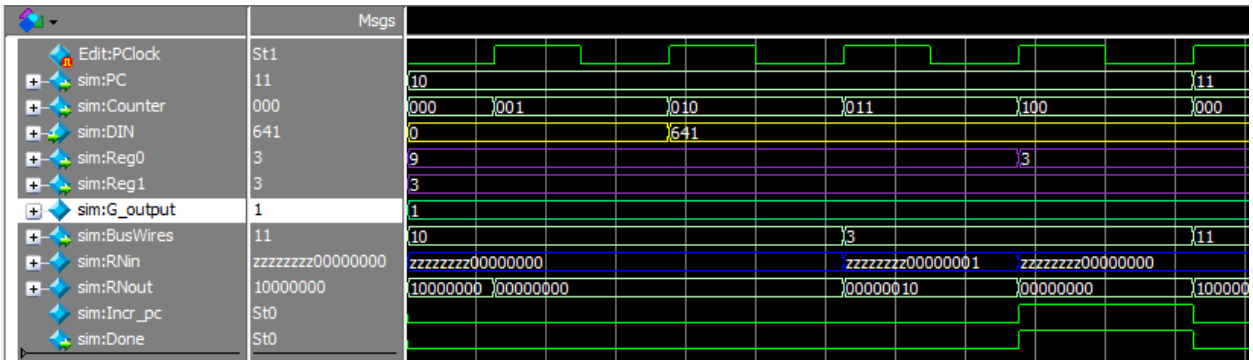
Memória RAM:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	5	7	8	4	0	0	0	0	.....
8	0	0	641	0	0	0	0	0	.....
16	0	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....

Essa foto ilustra a disposição da memória no momento do teste, indicando que o valor 641 está na posição 10, ou seja, o PC começa apontando para a instrução MVNZ R0, R1.

Execução: O teste em questão executa uma instrução de MVNZ envolvendo os registradores R0 e R1, ou seja, o correto funcionamento desse teste acarretará na movimentação do valor de R1 para R0 caso o valor do registrador G seja diferente de 0. Nesse caso, como G possui o valor 1, o registrador R0 terá seu valor alterado para o valor contido em R1, ou seja, R0 receberá 3.
(Inicialização dos Registradores: R0 = 9, R1 = 3, G = 1)

Simulação:



As linhas Roxas indicam os registradores R0 e R1, enquanto a linha amarela indica o DIN, que recebe o valor 641 em decimal, valor esse referente a instrução MVNZ R0, R1 como mostrado acima. Além disso, vale a pena destacar o sinal *G_output* que mostra qual o valor presente no registrador G no momento em que a instrução é executada.

4.11 - **SD e LD**

Instruções Executadas:

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)	DIN[15:0] (Hex)
SD R0, R1	000000	1001	000 (R0)	001 (R1)	577	241
LD R2, R1	000000	1000	010 (R2)	001 (R1)	529	211

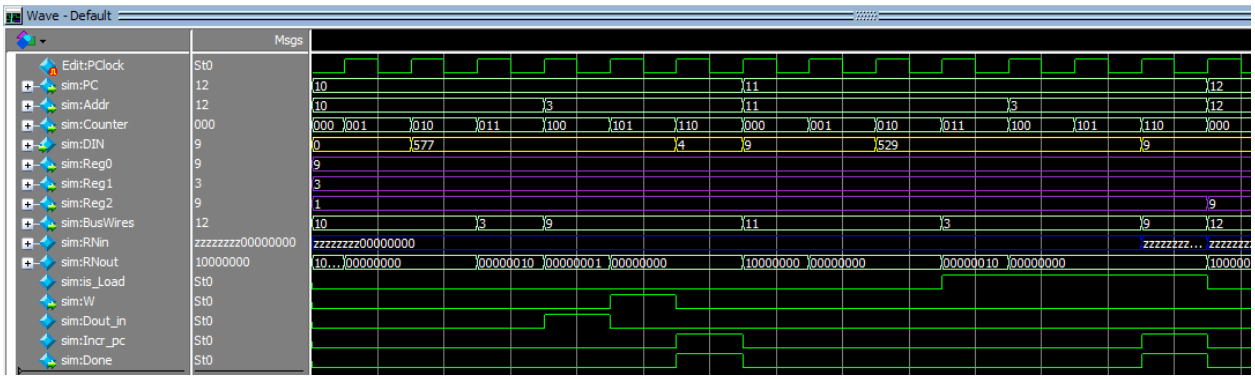
Memória RAM:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	5	7	8	4	0	0	0	0	.....
8	0	0	577	529	0	0	0	0	.....
16	0	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....

Essa foto ilustra a disposição da memória no momento do teste, indicando que o valor 577 está na posição 10, ou seja, o PC começa apontando para a instrução SD R0, R1, e indicando que o valor 529 está na posição 11, ou seja, logo após o SD será executada a instrução LD R2, R1.

Execução: O teste em questão deve executar primeiramente uma instrução de SD, que irá salvar o valor do registrador R0, o valor 9, na posição de memória referente ao valor presente no registrador R1, que é o valor 3, ou seja, uma correta execução acarretaria no salvamento do valor 9 na posição 3 da memória. Segundamente, o teste executa uma instrução de LD que acessa a posição de memória referente ao valor salvo no registrador R1, que é a posição 3 da memória, e salva o valor presente nessa posição de memória no registrador R2, ou seja, uma correta execução acarretaria no salvamento do valor 9 no registrador R2. (Inicialização dos Registradores: R0 = 9, R1 = 3, R2 = 1)

Simulação:



As linhas Roxas indicam os registradores R0, R1 e R2, enquanto a linha amarela indica o DIN, que recebe o valor 577 e depois 529, ambos em decimal, valor esses referentes às instruções SD R0, R1 e LD R2, R1 como mostrado acima. Além disso, vale a pena destacar o sinal *W*, que indica quando a escrita na memória é habilitada e o sinal *Addr* que altera seu valor durante a execução mostrando que a memória está sendo acessada. OBS: 1 - Os valores 4 e 9 que aparecem no DIN durante a execução aparecem pois o *Addr* está com o valor 3 e retorna, primeiro o número 4, valor inicial de MEM[3], e em seguida o valor 9, valor salvo pelo SD em MEM[3], e só depois que o PC é incrementado é que DIN recebe o valor 529 referente a instrução de LD. 2 - Esses valores acabam aparecendo pela demora que ocorre para a instrução acessar e escrever na memória.

5 - **Teste Loop**

Instruções Executadas:

Instrução	DIN[15:10]	Opcode	RX	RY	DIN[15:0] (Decimal)
MVI R2, #1	000000	0111	010	000	464
MVI R4, #10	000000	0111	100	000	480
MV R3, R7	000000	0110	011	111	415
SUB R4, R2	000000	0001	100	010	98
MVNZ R7, R3	000000	1010	111	011	699

Memória RAM:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	1	2	3	4	0	0	0	0	.....
8	0	0	464	1	480	10	415	98	.....b
16	699	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....

Execução:

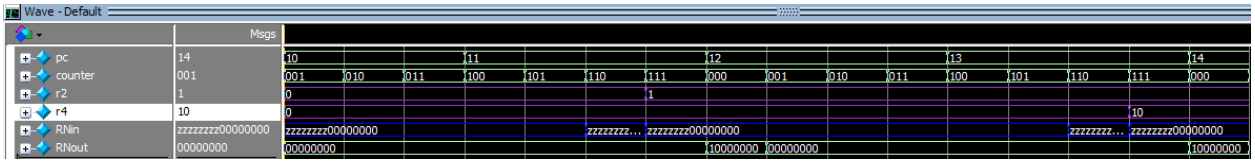
Instrução	Resultado Final
MVI R2, #1	R2 = 1
MVI R4, #10	R4 = 10
MV R3, R7	R3 = 14
SUB R4, R2	R4 = 9
MVNZ R7, R3	R7 = 14

Para a execução do Loop todos os registradores foram inicializados com valor 0.

Simulação:

Durante a apresentação na FPGA o Loop funcionou normalmente até a última instrução, como mostrado em sala, não conseguimos entender o porquê a atribuição de um novo valor para PC não foi executada corretamente, uma vez que o módulo do registrador PC é praticamente igual ao dos demais registradores. Durante a simulação do Loop, aparentemente os sinais XXX e YYY não receberam os valores corretos em diversos momentos, não conseguimos entender o porque disso, sendo que as entradas estavam todas iguais as que foram apresentadas em sala. Sendo assim, o Loop não teve um funcionamento tão adequado quanto devia, portanto, colocamos abaixo a execução de algumas das instruções presentes no Loop só para ilustrar o funcionamento isolado delas.

MVI R2, #1 e MVI R4, #10



MV R3, R7