Relatório Prática 2 - Processador Multiciclo

Nomes: Abdul-Kevin Alexis e Pedro Santos Oliveira

1 - **Decisões Gerais de Projeto**

a) A memória está dividida da seguinte maneira : Posição 0 a 9 são dados e posição 10 a 20 são instruções. Por esse motivo o PC começa com o valor 10, onde está localizada a primeira instrução na memória, e sendo assim, o Registrador ADDR também começa com a posição 10 uma vez que ele recebe o valor inicial de PC.

b) O Registrador ADDR sempre será atualizado junto com o PC, ou seja, sempre que uma instrução for finalizada e o sinal de incremento do PC receber 1, o Registrador ADDR também irá atualizar seu valor para PC + 1. Isso faz com que não seja necessário esperar 1 ciclo de clock no início da execução para que o valor de PC seja atribuído ao Registrador ADDR.

c) Foi criado um sinal is_Load que garante que os registradores XXX e YYY não podem ser atualizados durante uma instrução de Load, pois essa instrução é a única que escreve dados no DIN durante sua execução, podendo interferir no valor desses registradores. Portanto, esse sinal é setado para 0 quando a unidade de controle é inicializada e no Time Step 0, para garantir que ele só será 1 durante a execução de uma instrução de Load.

d) 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. No código final todos os registradores foram inicializados com o valor 0.

e) Explicar a organização do relatório (Fotos, Textos, …)

f) Explicar que os Time Steps 0 e 1 são apenas de leitura do pc e escrita da instrução em IR. Falar que cada instrução começa a execução apenas em T3.

g) Para a instrução MVI o imediato sempre estará na posição subsequente na memória em relação a posição da instrução, ou seja, se a instrução estiver na posição 10 o imediato estará na posição 11

h) Falar da disposição dos bits de DIN

i) Citar os sinais da ULA

j) Clock: duty value = 50ns, initial value = 0

2 - **Decisões Específicas de Projeto (Módulos)**

3 - **Formato das entradas e das instruções**

DIN[9:0] -> IIIIXXXYYY

	DIN[15:10]	**Opcode**	**XXX**	**YYY**	**IR Decimal**
ADD R1, R0	000000	0000	001 (R1)	000 (R0)	8
SUB R2, R1	000000	0001	010 (R2)	001 (R1)	81
MV R3, R2	000000	0110	011 (R3)	010 (R2)	410
MVI R0, #2	000000	0111	000 (R0)	YYY	448
OR R1, R2	000000	0010	001 (R1)	010 (R2)	138
SLT R0, R1	000000	0011	000 (R0)	001 (R1)	193
SLL R0, R1	000000	0100	000 (R0)	001 (R1)	257
SRL R0, R1	000000	0101	000 (R0)	001 (R1)	321
LD R0, R1	000000	1000	000 (R0)	001 (R1)	513
SD R0, R1	000000	1001	000 (R0)	001 (R1)	577
MVNZ R0, R1	000000	1010	000 (R0)	001 (R1)	641

4 - **Testes Unitários de cada Instrução**

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

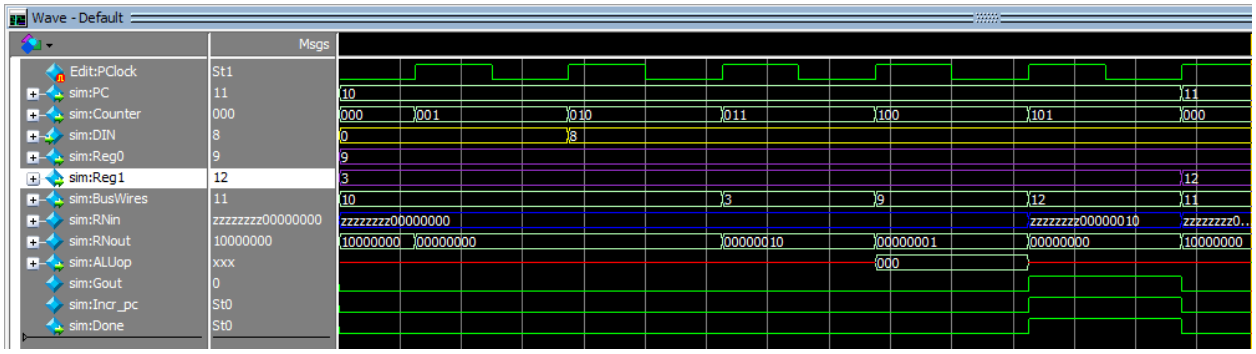
Memória RAM:

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

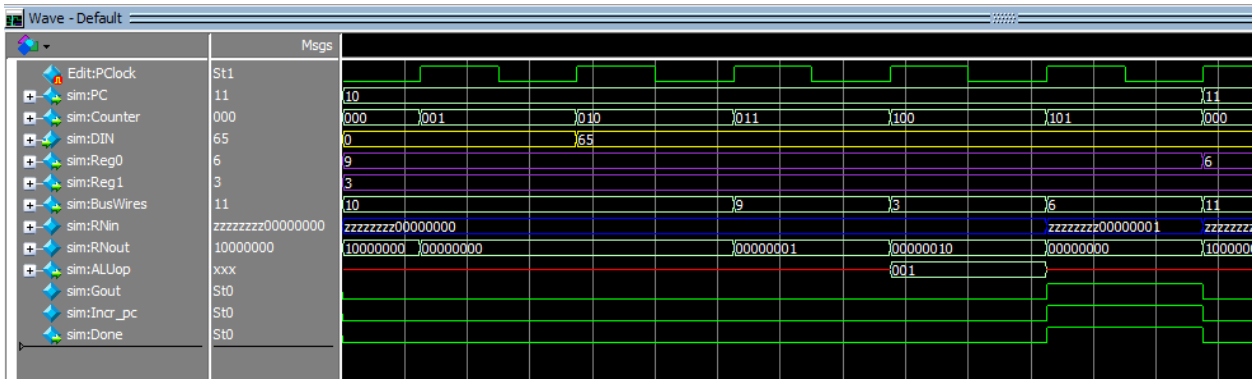
Memória RAM:

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

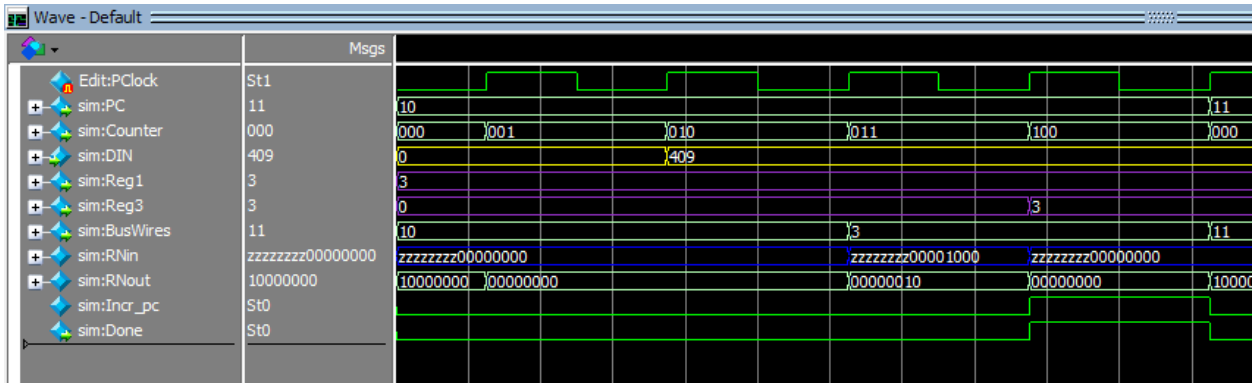
Memória RAM:

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

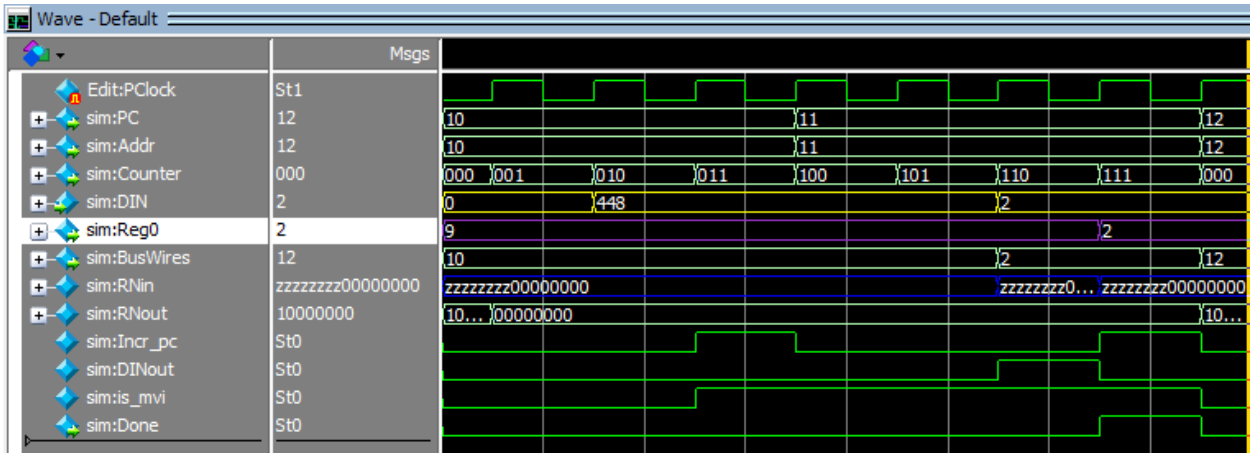
Memória RAM:

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

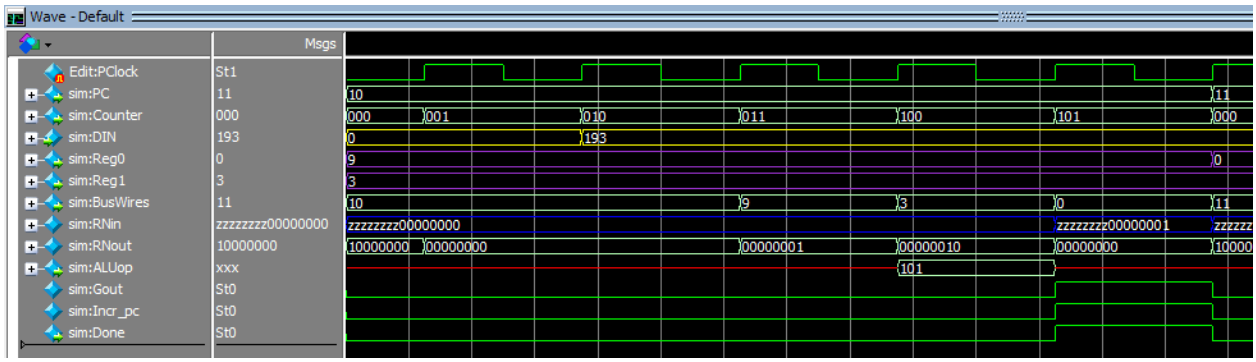
Memória RAM:

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

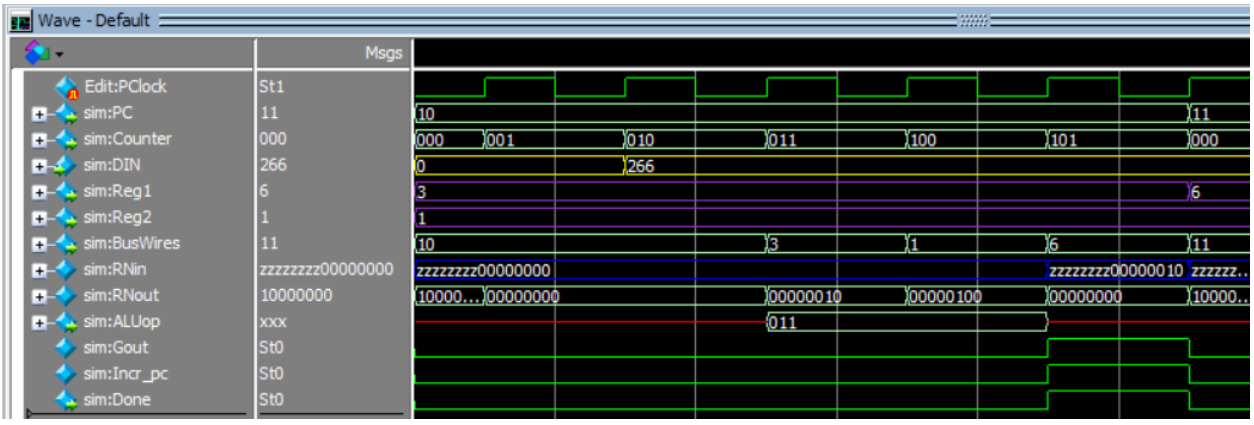
Memória RAM:

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)
SRL R0, R1	000000	0101	000 (R0)	001 (R1)	321	141

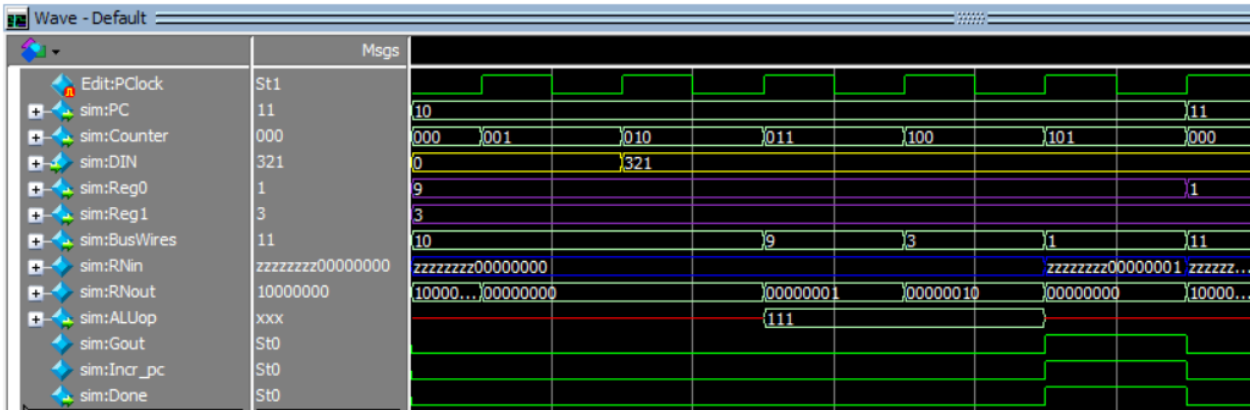
Memória RAM:

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

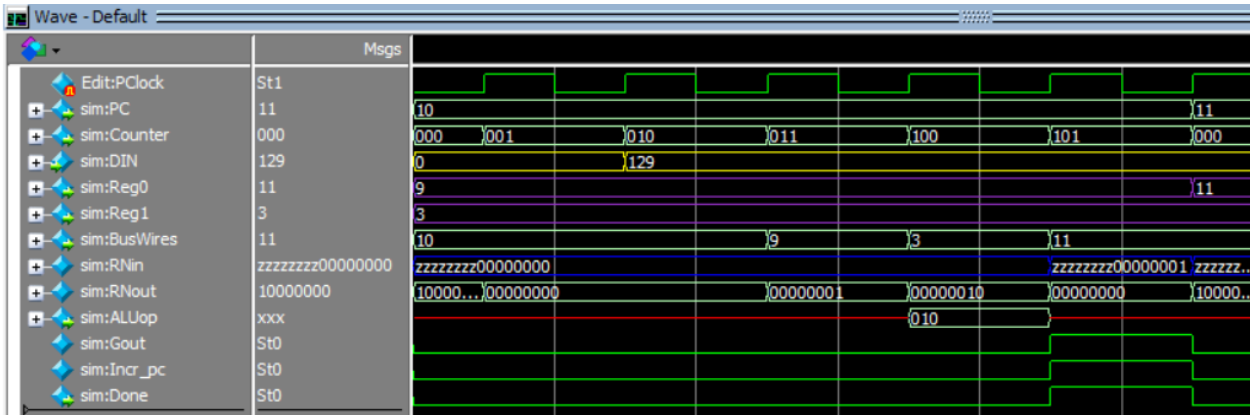
Memória RAM:

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

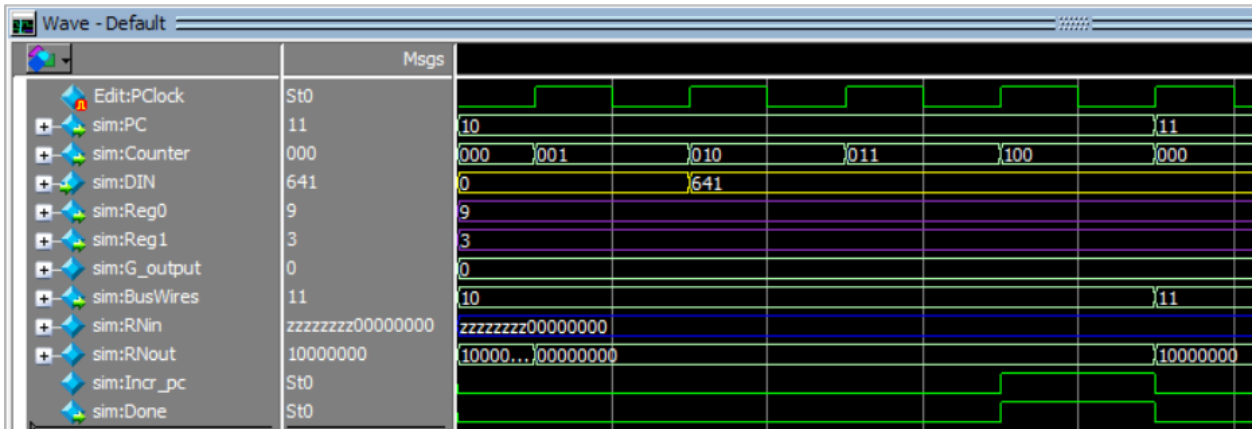
Memória RAM:

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:
Memória RAM:
Execução: (Inicialização dos Registradores: R0 = 9, R1 = 3, G = 1)
Simulação:

4.11 - **LD**

Instrução Executada:
Memória RAM:
Execução:
Simulação:

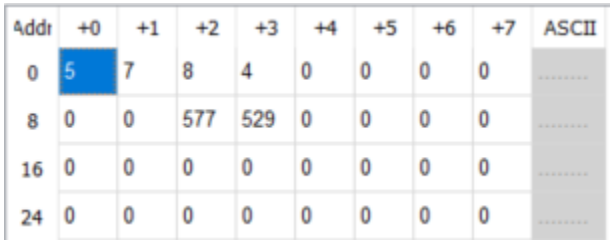
5 - **Testes de Instruções em sequência**

5.1 - **Teste SD seguido de 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

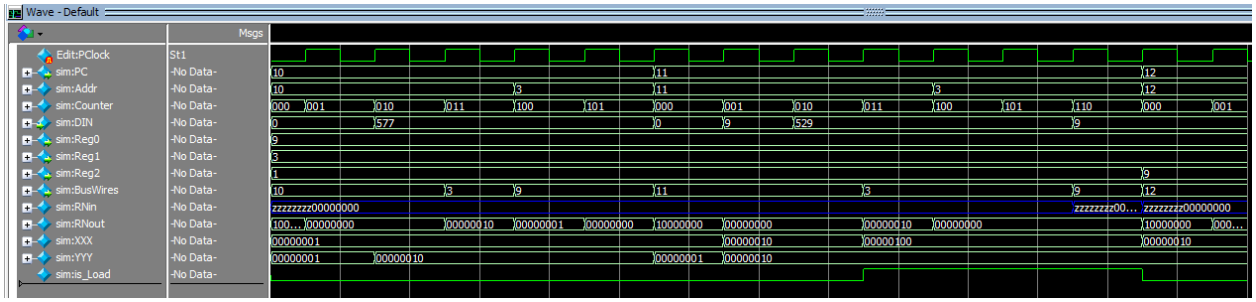
Memória RAM:



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	.....

Execução: O teste em questão deve executar primeiramente uma instrução de SD, que irá salvar o valor do registrador R0, o valor 9 em decimal, na posição de memória referente ao valor salvo 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:


Wave - Default
Msgs
Edit:PClock
sim:PC
sim:Addr
sim:Counter
sim:DIN
sim:Reg0
sim:Reg1
sim:Reg2
sim:BusWires
sim:RNin
sim:RNout
sim:XXX
sim:YYY
sim:is_Load
-No Data-
St1