Curso de Engenharia de Computação

05434P – LABORATÓRIO DE ARQUITETURA DE COMPUTADORES

Projeto 2 - CPU Multi-ciclo
Prof. Dr. Ricardo Pannain

Equipe:

Henrique Sartori Siqueira	19240472
Jemis Dievas José Manhiça	19076272

1 - Descrição do projeto com a topologia da CPU

Este projeto foi realizado no sistema operacional Windows 10, com os softwares Quartus Prime Lite Edition 18.1 (testes e execução do código), Visio (desenho do datapath), Visual Studio Code e Github (edição e armazenamento do código), e Google Drive (confecção do relatório, tabela de estados e armazenamento dos demais arquivos).

Para este presente projeto foram registrados 64 endereços com 4 bits por endereço na memória, sendo que PC é atualizado em 4 posições, havendo 16 instruções executáveis com 16 bits cada uma. Com o datapath definido a seguir:

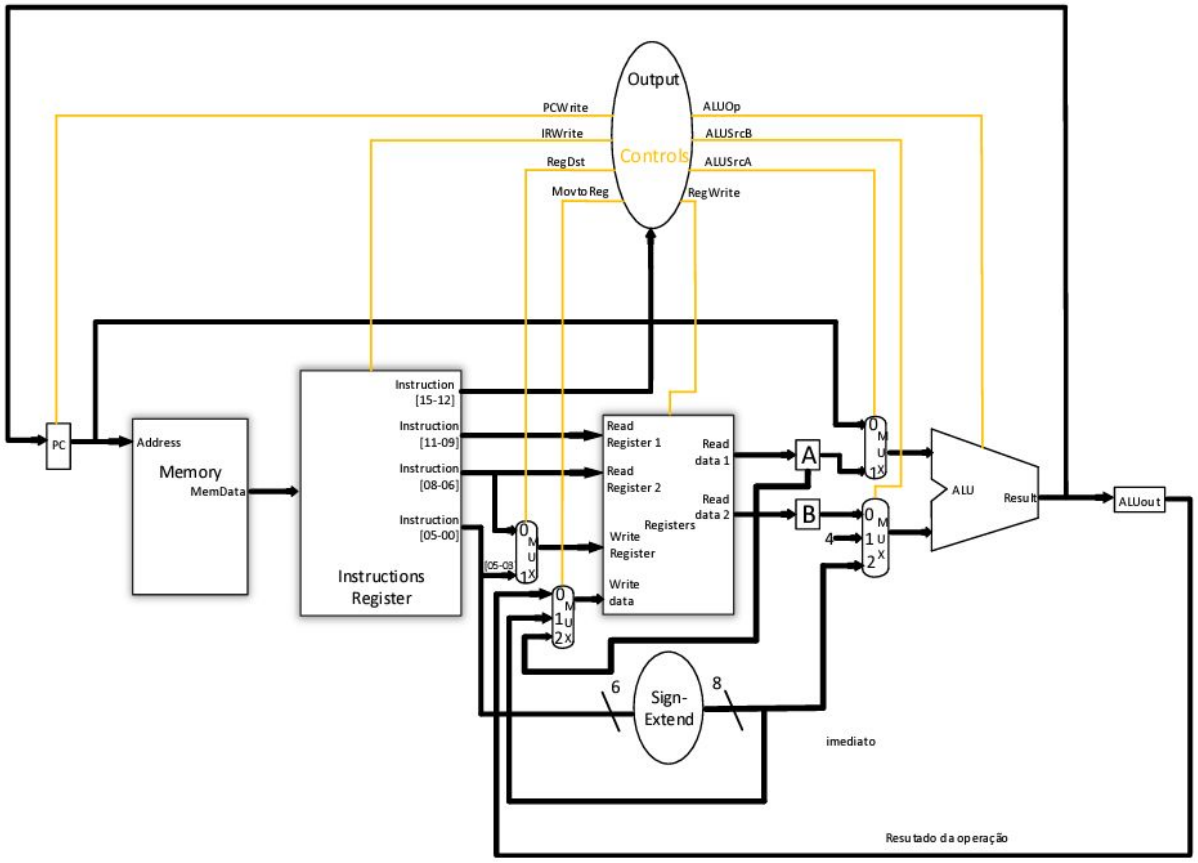

Imagem 1. Datapath.

Para a confecção em código deste datapath, foram feitas seis instâncias denotadas a seguir:

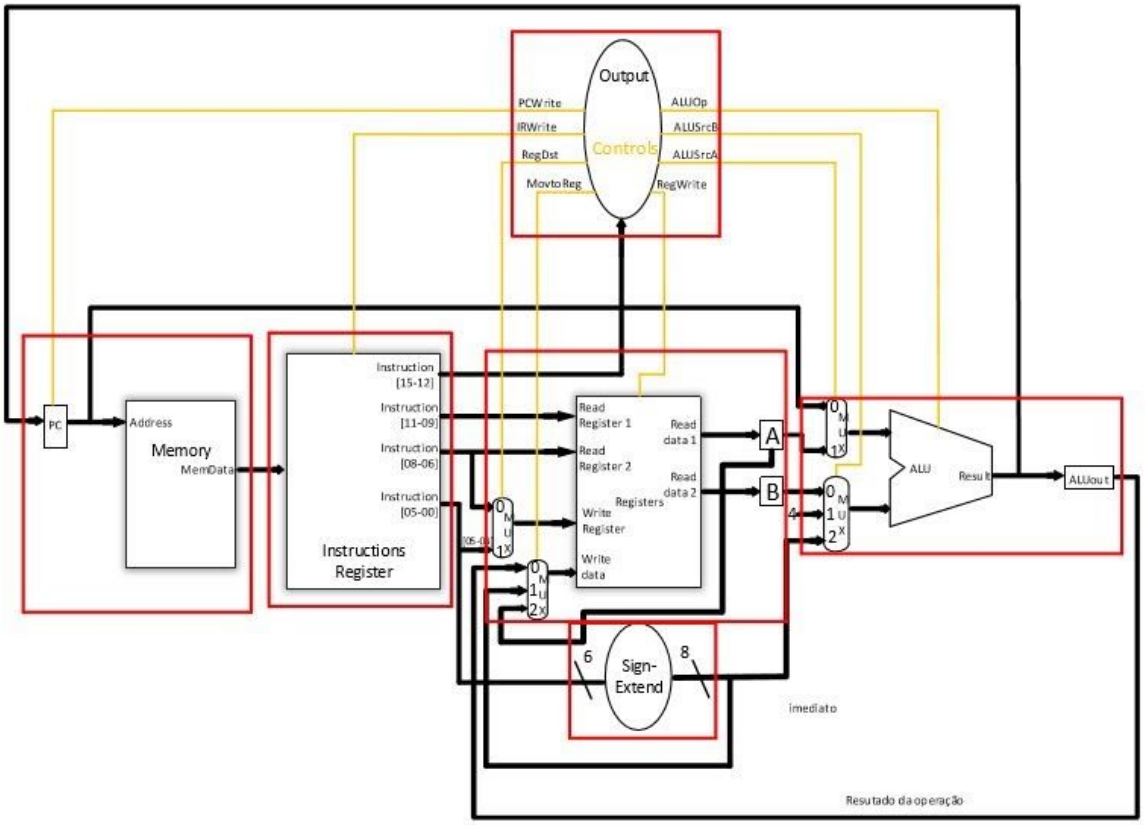

Imagem 2. Datapath com a separação de componentes.

Porém, para as instâncias do Instruction Register, Registers e ALU foram realizadas duas instâncias para cada uma, sendo que posteriormente as instâncias mais externas foram desconsideradas, totalizando sete arquivos em VHDL utilizáveis para a execução do programa.

2 - Especificação

2.1 - Registradores

Neste datapath, o banco de registradores possui oito registradores de 8 bits cada, com os respectivos endereços representados a seguir:

Registrador	Endereço
$0	000

$1	001
$2	010
$3	011
$4	100
$5	101
$6	110
$7	111

2.2 - Formato das instruções

As instruções possuem 16 bits, sendo divididas em dois formatos descritos a seguir:

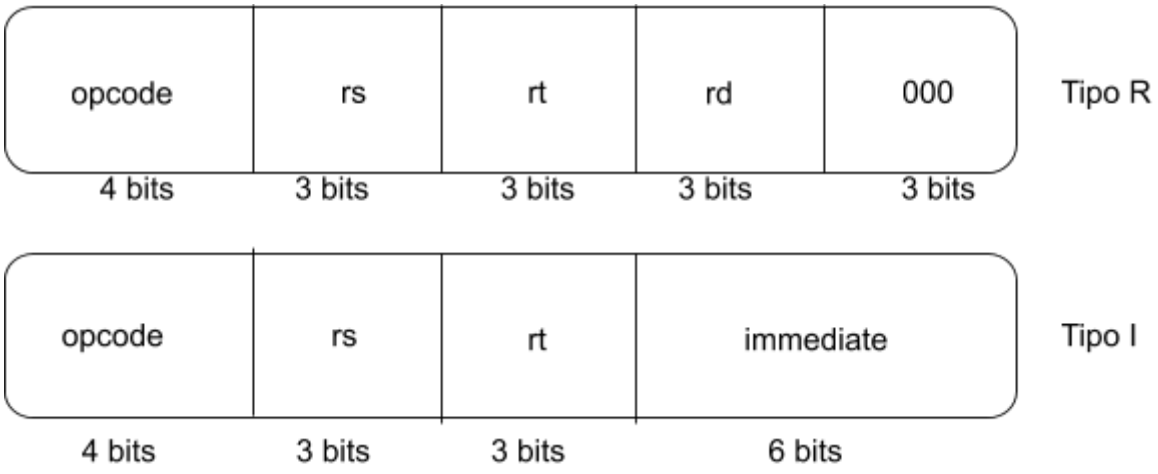

Imagem 3. Formato das instruções.

Opcode	Instrução	Significado	Descrição
0000	MOV Ri, Rj	Ri ← Rj	Move
0001	MOV Ri, Imediato	Ri ← Imediato	Move Imediato
0010	ADD Ri, Rj, Rk	Ri ← Rj + Rk	Adição

0011	ADDI Ri, Rj, Imediato	Ri ← Rj + Imediato	Adição Imediata
0100	SUB Ri, Rj, Rk	Ri ← Rj - Rk	Subtração
0101	SUBI Ri, Rj, Imediato	Ri ← Rj - Imediato	Subtração Imediata
0110	AND Ri, Rj, Rk	Ri ← Rj & Rk	And
0111	ANDI Ri, Rj, Imediato	Ri ← Rj & Imediato	And Imediato
1000	OR Ri, Rj, Rk	Ri ← Rj | Rk	Or
1001	ORI Ri, Rj, Imediato	Ri ← Rj | Imediato	Or Imediato

Instruções pares são do tipo R e instruções ímpares são do tipo I.

2.3 - Unidade de Controle: diagrama, tabela de estados, sinais e seus significados

Todas as instruções com exceção de movimentação de valores para um registrador são realizadas em quatro ciclos de clock, enquanto as instruções move são realizadas em três ciclos de clock.

Além disso, há oito sinais de controle que são responsáveis por gerenciar o controle do datapath a cada ciclo.

Sinal	Efeito quando ‘0’	Efeito quando ‘1’
PcWrite	Mantém valor de PC	Atualiza valor de PC
IRWrite	Mantém valor da instrução	Atualiza valor da instrução
RegDst	Salva resultado em rt (formato R)	Salva resultado em rd (formato I)
RegWrite	Não realiza escrita em registrador	Realiza escrita em registrador
ALUSrcA	Utiliza valor de PC para operação	Utiliza registrador para operação

Sinal	Efeito quando “00”	Efeito quando “01”	Efeito quando “10”	Efeito quando “11”
MovtoReg	Utiliza resultado da	Utiliza o valor do imediato	Utiliza o valor do registrador	NULL

	operação da ULA	estendido	rs	
ALUOp	Realiza AND	Realiza OR	Realiza adição	Realiza subtração
ALUScrB	Utiliza o valor do registrador rt	Utiliza o valor 4	Utiliza o valor do imediato estendido	NULL

Para a confecção da máquina de estados, foram utilizados 7 estados com o modelo de máquina de Mealy (tabela de estados em anexo).

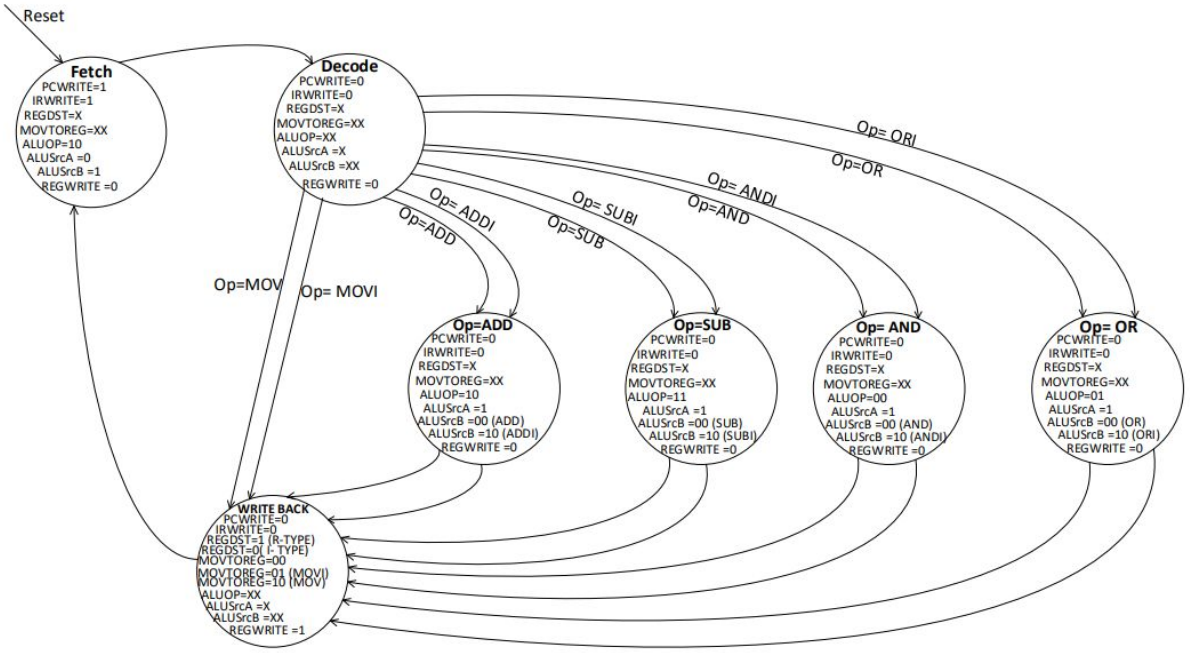

Imagem 4. Diagrama de estados.

Ciclo	Ação
1 (fetch)	Coleta a instrução na memória e atualiza valor de PC (PC = PC + 4)
2 (decode)	Atribui opcode para a máquina de estados e realiza a coleta dos valores dos registradores a serem utilizados
3 (execution / write)	Escreve o valor de volta ao registrador

	de destino (moves) ou realiza a operação na ULA para as demais instruções
4 (write back)	Escreve de volta o resultado da operação da ULA no banco de registradores

3 - Resultados

3.1 - Descrição dos testes realizados

Para a realização dos testes, foi utilizado um período de clock de 100 ns.

Testes para o registrador de instruções, em que primeiramente houve o teste para o tipo R e depois o teste para o tipo I de instruções:

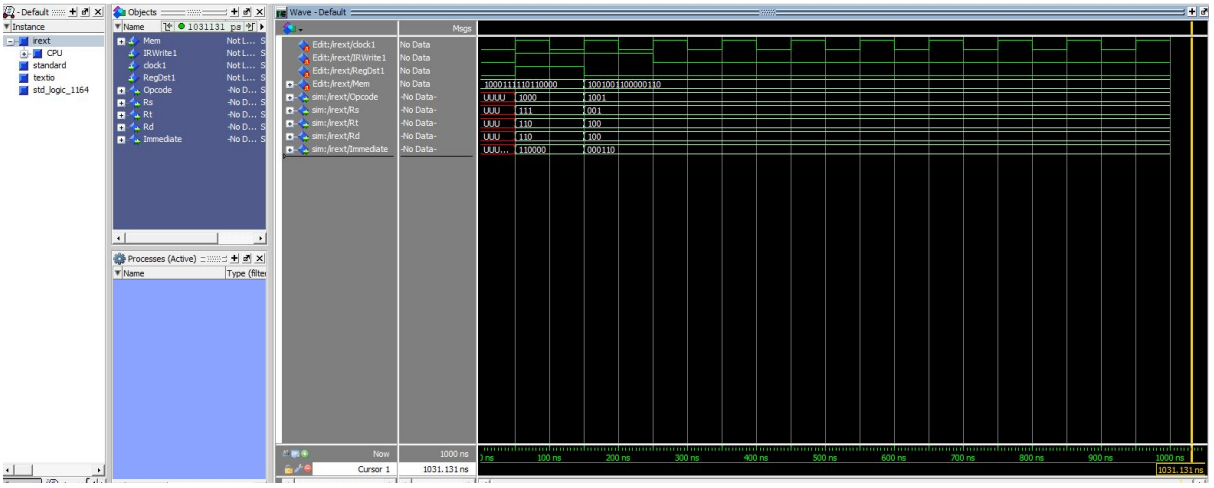

Imagem 5. Testes do registrador de instruções.

Testes para a memória, em que há o acesso a mesma na subida do clock e o envio dos dados na descida de clock:

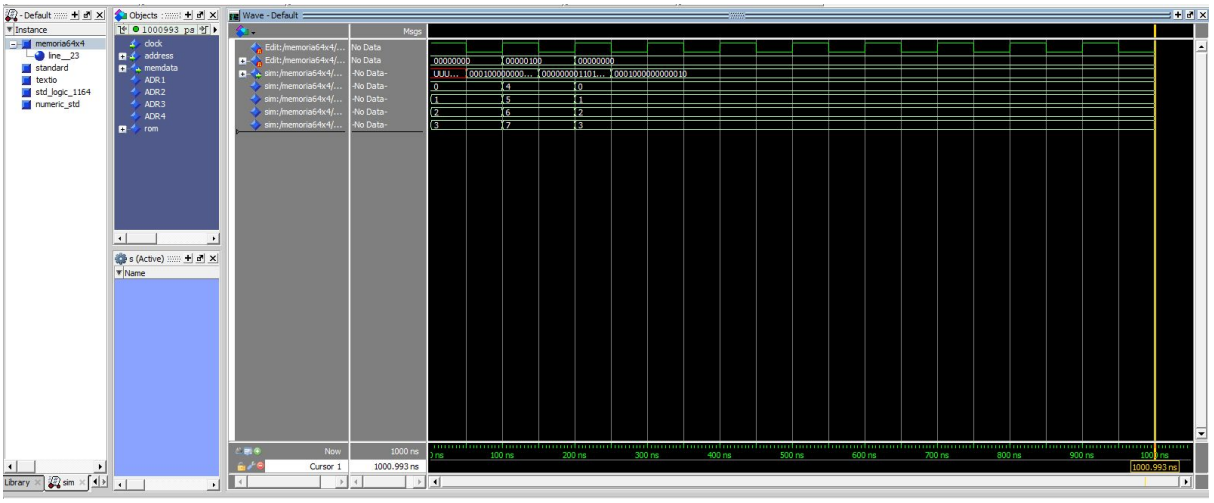

Imagem 6. Testes da memória.

Testes para a componente de extensão de sinal, a mesma recebe como entrada um dado de 6 bits e estende o sinal deste dado para 8 bits:

Imagem 7. Testes do sinal estendido.

Os testes para o banco de registradores foram realizados através de um ciclo de escrito seguido por um ciclo de leitura:

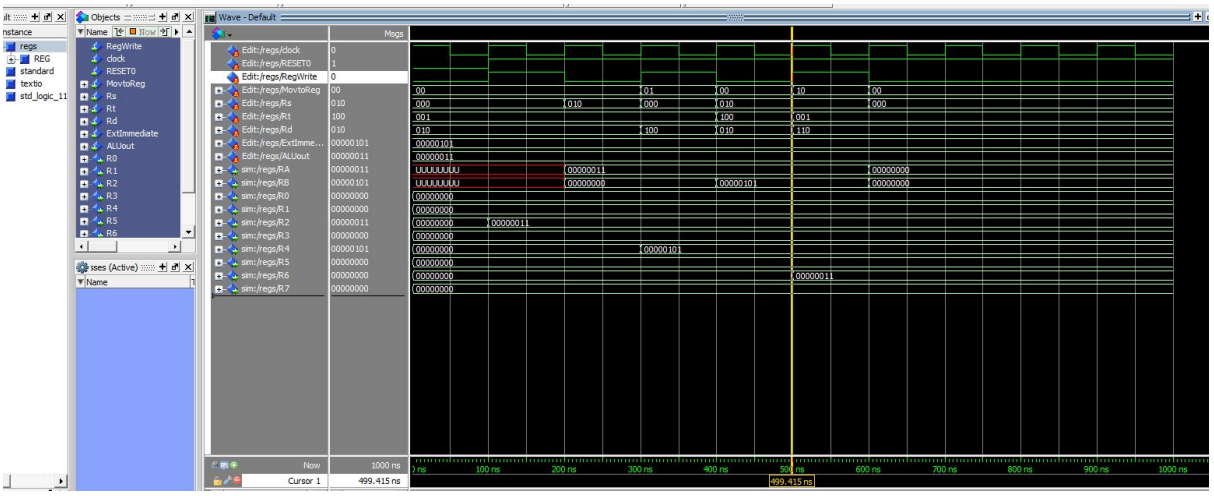

Imagem 8. Testes do banco de registradores.

Os testes da Unidade Lógica e Aritmética foram feitos através da execução de todas as operações, em que o resultado é escrito quando há a descida do clock:

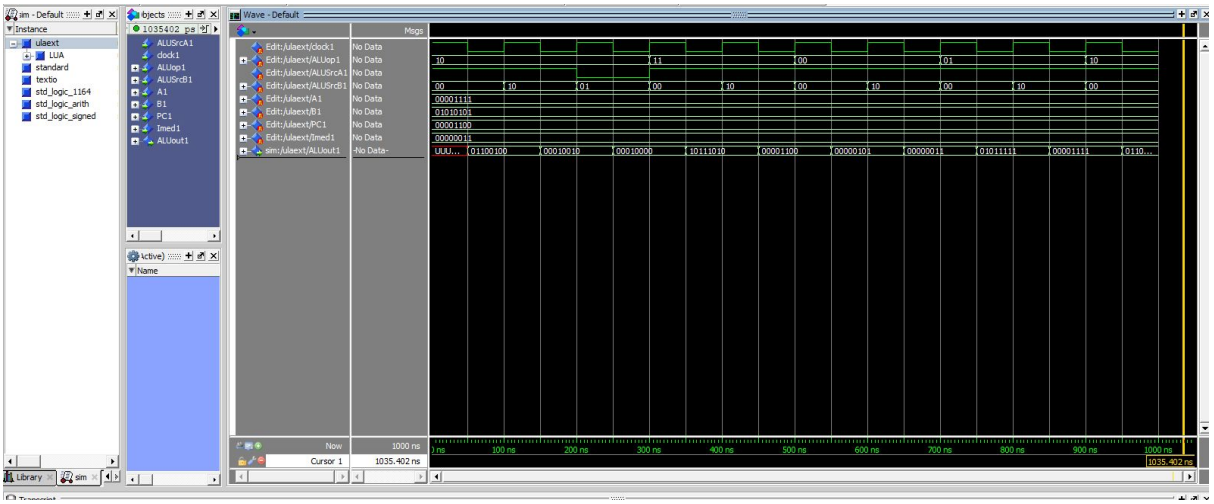

Imagem 9. Testes da unidade lógica e aritmética.

Os testes para a máquina de estados foram feitos a partir da execução de todas as instruções (0000 - 1001) em ordem:

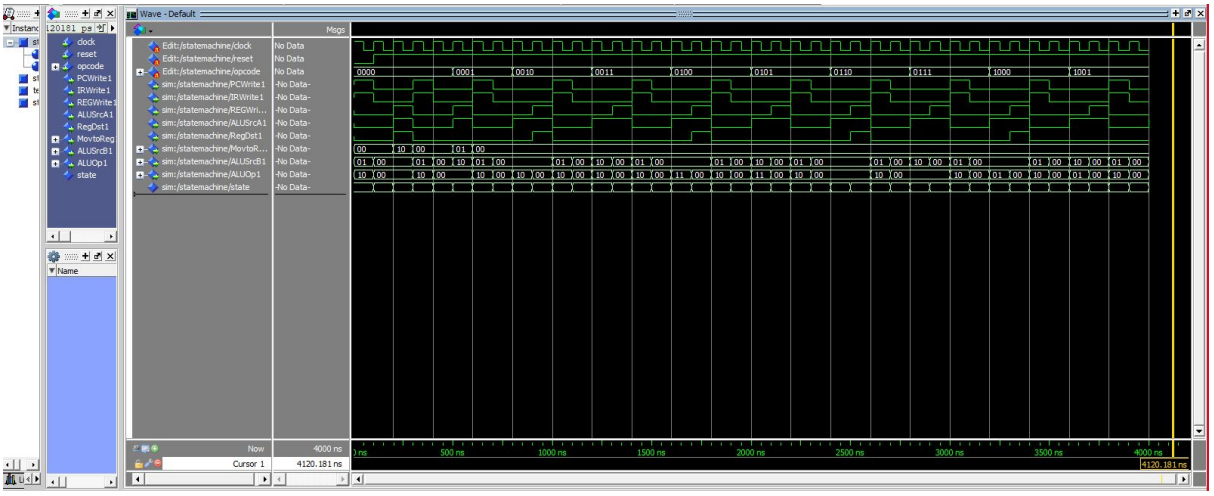

Imagem 10. Testes da máquina de estados.

Testes para escrita do valor de PC na descida do clock (havendo tempo para executar a soma na ULA), a primeira parte com um valor atualizado e segunda parte da onda com o valor de 64, fazendo com que PC reiniciasse o contador de endereços, pois atingiu o limite de instruções da memória:

Imagem 11. Testes do controlador de PC.

Ao realizar a junção de todas as componentes, foi constatado um erro em que o valor de PC mesmo que atribuído na declaração do mesmo (como buffer em entity ou como signal em arquitetura), a simulação não considerou tal valor de início para a variável, mas sim para o programa, fazendo com que o mesmo se tornasse nulo, como demonstrado a seguir e que o programa só executasse a primeira instrução:

```
SIGNAL MEMORY : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL ALURESULT,A,B,IMED8 : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL PC :  STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
SIGNAL IMED6 : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL OPcode : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL RT,RS,RD : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL MOVTOREG,ALUSRCB,ALUOP : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL PCWRITE,IRWRITE,REGDST,REGWRITE,ALUSRCA : STD_LOGIC;
```

Imagem 12. PC declarado como signal.

```
ENTITY CPU IS
   PORT(
            clock0, reset :IN   STD_LOGIC;
            R_0, R_1, R_2, R_3, R_4, R_5, R_6, R_7 : BUFFER STD_LOGIC_VECTOR(7 DOWNTO 0));
            PC : BUFFER STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000");
END CPU;
```

Imagem 13. PC declarado como buffer.

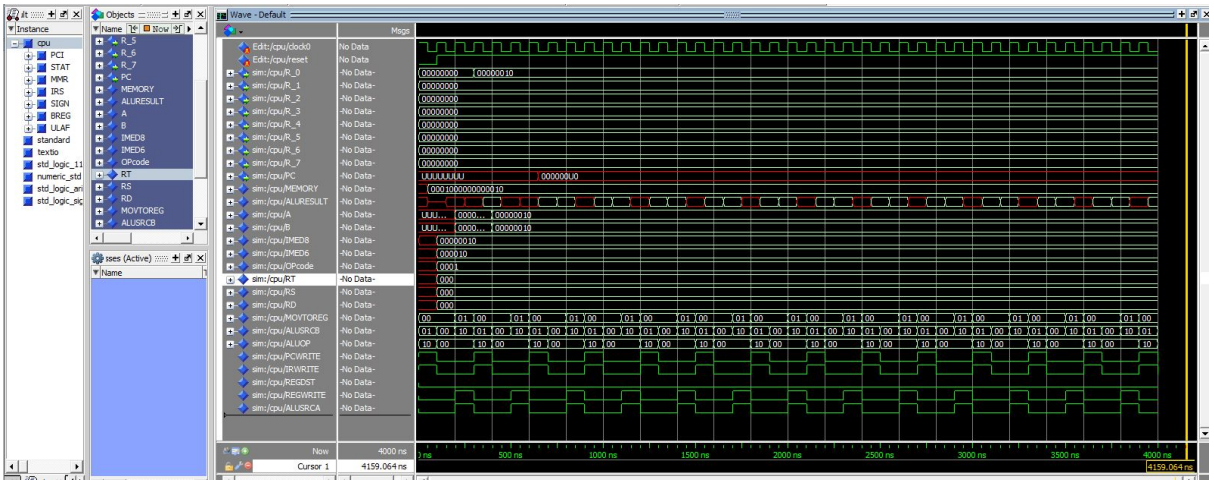

Imagem 14. Erro de PC declarado como buffer.

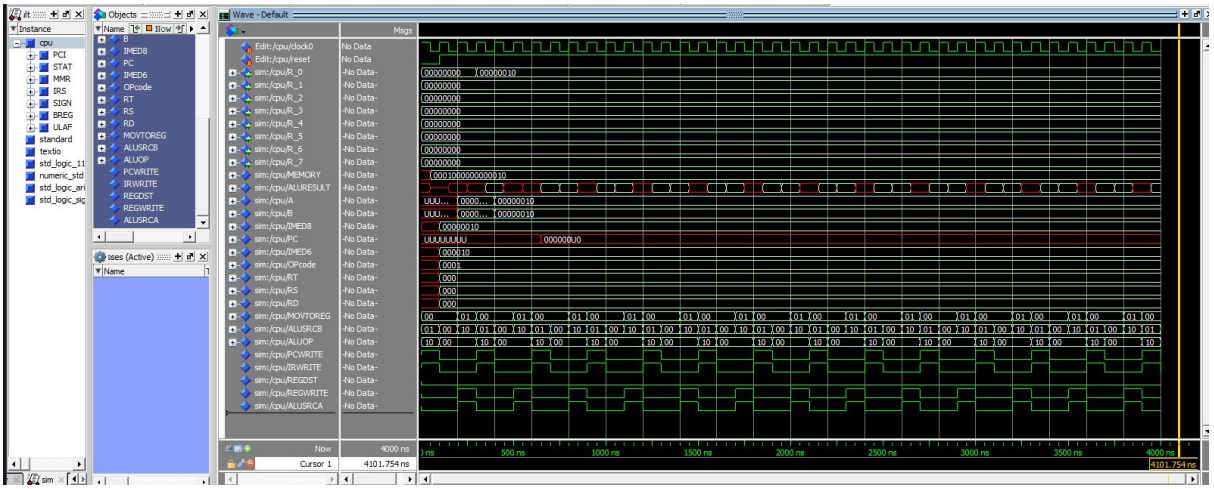

Imagem 15. Erro de PC declarado como signal.

Houve também a tentativa de atribuir um valor inicial a PC na componente de controle do mesmo (imagem 11), e também na máquina de estados, porém não é possível atribuir dois valores a uma variável de saída de uma componente. Para a resolução deste erro. Houve, então, a junção das componentes de memória e do controlador de PC, fazendo com que este registrador ficasse internamente na componente da memória e como um inteiro, fazendo com que haja a inutilização parcial de alguns sinais da ULA, isto é, os sinais ALUSrcA e ALUSrcB tendo como valores 0 e 01, respectivamente, não trariam efeito ao datapath, pois a atualização desta variável é feita diretamente na componente da memória.

Todavia, mesmo com tais alterações, constatou-se um erro na execução das operações, em que não eram feitas as mesmas, como resolução deste problema, houve a conexão direta entre as instâncias mais internas das componentes, bem como a operação direta entre as variáveis de entrada (em que antes o valor dos operandos eram atribuídos a dois sinais para realizar a operação), tal procedimento foi executado com êxito, porém juntamente com o erro da ULA, também houve um erro no banco de registradores em que não havia a escrita do resultado da ULA no quarto ciclo das instruções que utilizam a ULA.

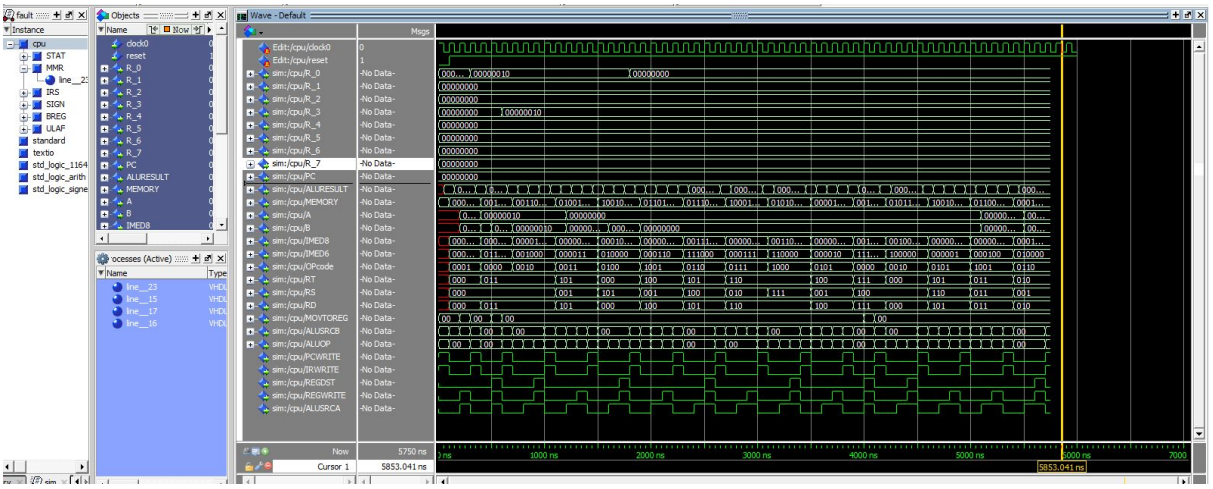

Imagem 16. Simulação com o erro de write back.

Com a constatação deste erro, foi feita uma análise dos sinais internos da componente do banco de registradores no momento de write back em uma operação de soma (ADD $1, $0, $3 em 600 ns - 1000 ns. Write-back em 900 ns - 1000 ns). Os sinais eram de fato como esperados, em que RegWrite estava com o sinal ativo, assim como o resultado da ULA também estava com o valor correto, entretanto, houve a percepção de um erro, em que o registrador destino estava sempre atribuído ao registrador alvo, devido à manipulação feita na componente IR, em que o RD já possuía o endereço, porém tal ação era totalmente imprecisa, pois

o sinal de RegDst sempre era 0, o que não poderia indicar com precisão o registrador que seria escrito o dado.

Após a correção dos erros, notou-se que a escrita do resultado de volta no banco de registradores estava ocorrendo apenas no final do ciclo, em que a ULA, na descida de clock realizaria uma operação *don't care*, fazendo com que a escrita no registrador fosse incongruente com o resultado original. Então, na máquina de estados houve a omissão de atribuição de sinal de operação para a ULA no estado de write back, fazendo com que os sinais de controle (ALUOp, ALUSrcA e ALUSrcB) ficassem com o mesmo valor do ciclo passado, realizando mais uma vez a operação para que o sinal permaneça com o valor correto.

Logo após a realização destas correções, foi novamente simulado e constatou-se que algumas operações estavam escrevendo o valor do resultado em diferentes registradores, ou seja, mesmo com o valor do endereço correto, código correto e intervalos de onda corretos, há a incongruência na escrita para o registrador 0. Depois de tal apontamento, foi realizado o teste individual, novamente, de todas as componentes, destacados a seguir:

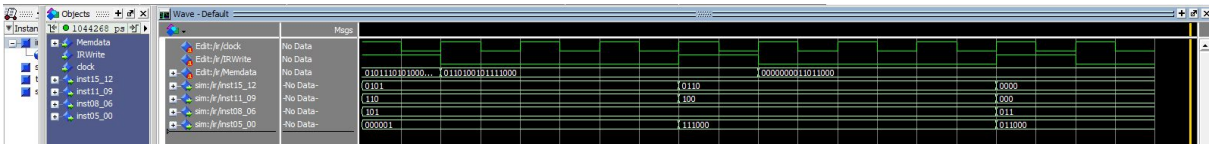

Imagem 17. Testes para o registrador de instruções.

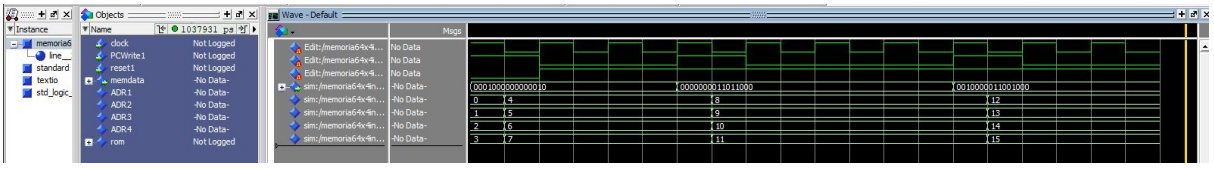

Imagem 18. Testes para a memória (e PC).

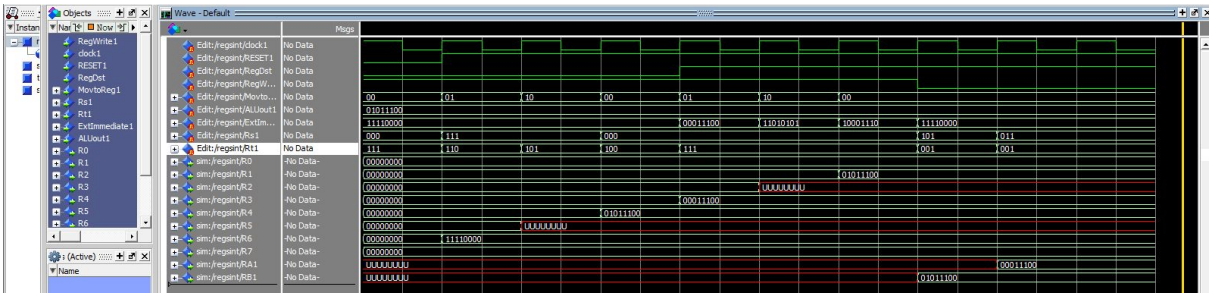

Imagem 19. Testes para o banco de registradores.

Nota-se que o banco de registradores funcionou perfeitamente com todos os tipos de escrita em todos os registradores, o que não aconteceu na junção de todos os componentes em algumas partes da execução.

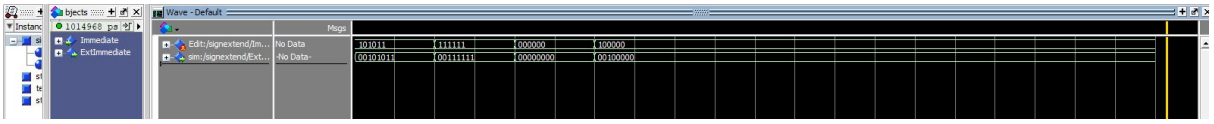

Imagem 20. Teste da componente de extensão de sinal.

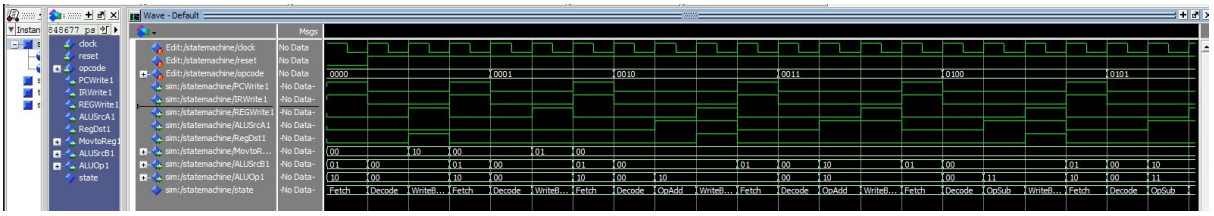

Imagem 21. Testes para a máquina de estados, primeira parte.

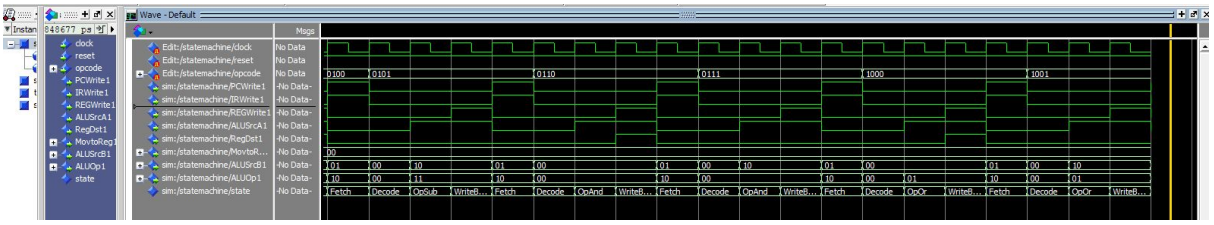

Imagem 22. Testes para a máquina de estados, segunda parte.

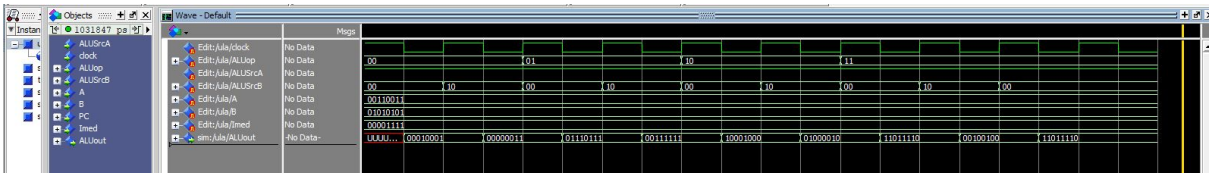

Imagem 23. Testes para a unidade lógica e aritmética.

O teste da CPU juntando todas as componentes, resultou na seguinte execução, em que permaneceu a escrita do resultado em um mesmo registrador, mesmo que o valor de endereço de escrita fosse diferente.

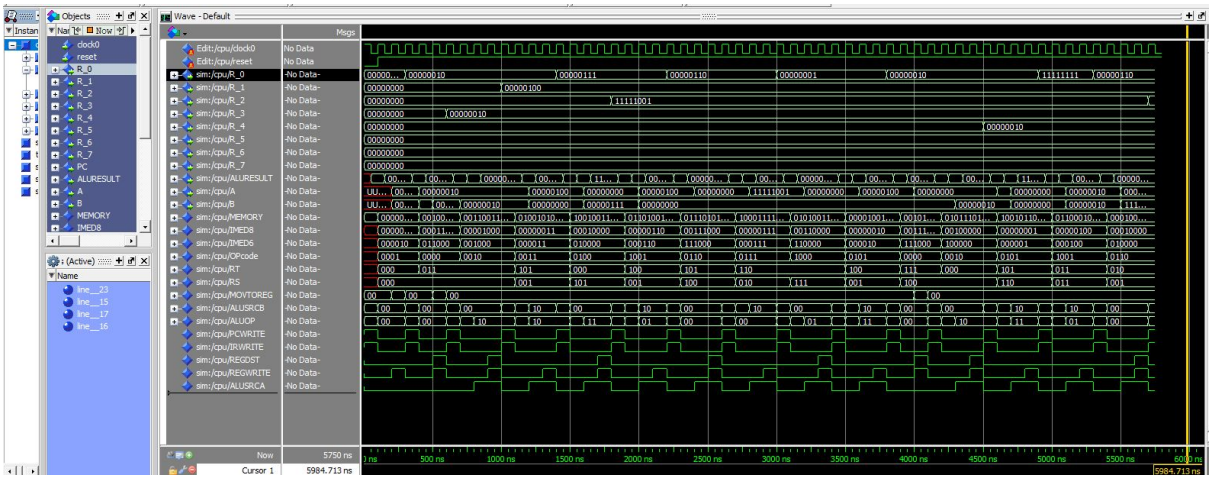

Imagem 24. Testes para a junção de todas as componentes (CPU).

3.2 - Resultados e discussões

Para a execução e simulação da CPU, foi criado um programa com 16 instruções que estão presentes na memória descritas em binário, sendo elas:

programa.txt

```
1   MOV $0, 2          // 0001 0000 0000 0010
2   MOV $3, $0         // 0000 0000 1101 1000
3   ADD $1, $0, $3     // 0010 0000 1100 1000
4   ADDI $5, $1, 3     // 0011 0011 0100 0011
5   SUB $2, $5, $0     // 0100 1010 0001 0000
6   ORI $4, $1, 6      // 1001 0011 0000 0110
7   AND $7, $4, $5     // 0110 1001 0111 1000
8   ANDI $6, $2, 7     // 0111 0101 1000 0111
9   OR $6, $7, $6      // 1000 1111 1011 0000
10  SUBI $4, $1, 2     // 0101 0011 0000 0010
11  MOV $7, $4         // 0000 1001 1111 1000
12  ADD $4, $4, $0     // 0010 1000 0010 0000
13  SUBI $5, $6, 1     // 0101 1101 0100 0001
14  ORI $3, $3, 4      // 1001 0110 1100 0100
15  AND $2, $1, $2     // 0110 0010 1001 0000
16  MOV $0, 7          // 0001 0000 0000 0111
```

Imagem 25. Programa para a execução da CPU.

Ajustando no formato binário, os números em vermelho seriam relacionados ao opcode, os números em verde com o registrador RS, os números em azul com o registrador RT, os números em roxo com o registrador RD, os números em amarelo

com o imediato e os números em preto seriam somente para completar o formato da instrução de 16 bits.

0001	0000	0000	0010	0000	0000	1101	1000
0010	0000	1100	1000	0011	0011	0100	0011
0100	1010	0001	0000	1001	0011	0000	0110
0110	1001	0111	1000	0111	0101	1000	0111
1000	1111	1011	0000	0101	0011	0000	0010
0000	1001	1111	1000	0010	1000	0010	0000
0101	1101	0100	0001	1001	0110	1100	0100
0110	0010	1001	0000	0001	0000	0000	0111

Devido aos problemas anteriormente citados, com PC principalmente, que posteriormente foi atribuído como um inteiro juntamente com a memória, houve uma mudança de implementação do datapath inicialmente proposto (imagem 1), em que PC e a memória se uniram, fazendo com que a entrada ‘0’ do multiplexador de ALUSrcA ficasse inutilizada, assim como a entrada “01” do multiplexador de ALUSrcB.

No entanto, um dos erros que impossibilitou a execução de algumas instruções foi o de escrita do resultado no banco de registradores, em que houve tal escrita, mas em um registrador que não condiz com o endereço especificado, em que checando o código, executando a componente separadamente e analisando os sinais internos tudo estava completamente como previsto.

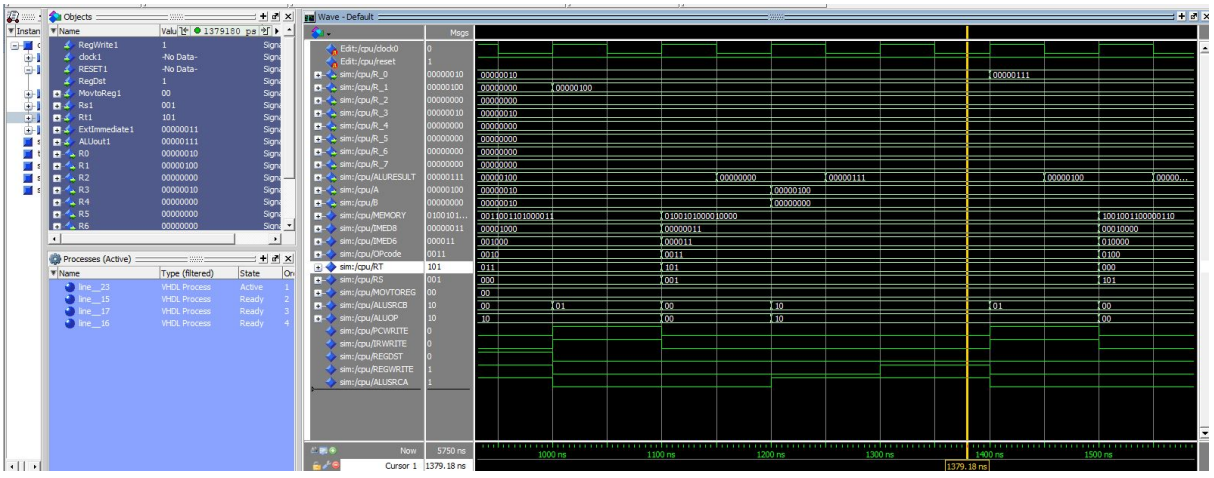

Imagem 26. Demonstração do erro de escrita, com os sinais internos do banco na janela azul.

Pode-se notar que a instrução que antecede 1000 ns foi executada com êxito, bem como sua escrita no correto registrador. Já a instrução pertencente ao tempo de 1000 ns - 1400 ns obteve sua escrita em um registrador incorreto.

A execução de tal escrita, em algumas instruções, foi realizada em um registrador inesperado. Em que houve tentativas de correção com o aumento do período de clock, e mudança de escrita para a descida de clock, para conferências de tempo, porém não alcançando o objetivo desejado.

O desenvolvimento deste presente projeto nos deu a noção da dificuldade que é planejar, projetar e realizar a simulação de um hardware que está presente em praticamente todos os componentes eletrônicos.

4 - Bibliografia

http://people.sabanciuniv.edu/erkays/el310/MemoryModels.pdf

https://stackoverflow.com/questions/9018087/shift-a-std-logic-vector-of-n-bit-to-right-or-left

https://www.nandland.com/vhdl/examples/example-shifts.html#:~:text=Create%20shift%20registers%20in%20your,to%20create%20a%20Shift%20Register.

https://stackoverflow.com/questions/26683335/vhdl-code-to-convert-5-bit-vector-to-integer

https://stackoverflow.com/questions/27164411/vhdl-program-counter-using-signals-and-previously-made-components

https://stackoverflow.com/questions/47471415/net-which-fans-out-cannot-be-assigned-more-than-one-value