UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

Isaque Barbosa Martins

João Guilherme Lopes Alves da Costa

Processador RISC com pipeline

NATAL, RN
2023

Isaque Barbosa Martins

João Guilherme Lopes Alves da Costa

Processador RISC com pipeline

Relatório técnico apresentado na disciplina Organização de Computadores (DIM0129) para a primeira unidade do semestre letivo 2023.1 do curso Bacharelado em Tecnologia da Informação pela Universidade Federal do Rio Grande do Norte.

Orientador: Profº. Dr. Marcio Eduardo Kreutz

NATAL, RN
2023

Sumário

1 DECISÕES DE PROJETO 4
1.1 Tamanho da palavra do processador 4
1.2 Formato da palavra de instrução 4
1.3 Modos de endereçamento de operandos 5
1.4 Tamanho do banco de registradores 5
1.5 Tamanho das memórias de instruções e de dados 5
1.6 Número e tipos de barramentos (ou canais dedicados) da parte operativa 5
1.7 Organização do pipeline 5

2 DIAGRAMAS 6

3 IMPLEMENTAÇÃO 7
3.1 Implementação do modelo da arquitetura em linguagem para descrição de hardware (VHDL ou SystemC) 7
3.2 Resultados de simulações da execução de instruções de pelo menos 3 algoritmos na arquitetura 7
3.2.1 Algoritmo 1 7
3.2.2 Algoritmo 2 10
3.2.3 Algoritmo 3 12
3.3 Relatório explicando e exemplificando a implementação da arquitetura e justificando as decisões de projeto acima elencadas. . . 14

4 CONCLUSÃO 15

1 Decisões de Projeto

Este presente relatório tem como objetivo explicitar a abordagem adotada para implementar um Processador RISC com Pipeline. Esse trabalho foi produzido durante o período 2023.1 para a disciplina de Organização de Computadores pela Universidade Federal do Rio Grande do Norte (UFRN), aplicando os conceitos aprendidos em sala para implementar o processador.

1.1 Tamanho da palavra do processador

O tamanho da palavra do processador adotada foi de 32 bits, para a padronização do projeto.

1.2 Formato da palavra de instrução

Por questões de padronização, construímos palavras de instrução de 32 Bits, onde tempos 3 tipos de palavras de instrução principais: Operações da ULA, Operações na Memória e Operações de Salto.

Para as Operações da ULA foi necessário o campo referente à Operação, um campo para a instrução a ser realizada pela ula, e 3 campos referentes ao endereço registrador fonte 1 e fonte 2 de onde virão os dados para operação, e o endereço do registrador destino.

Para as Operações de Memória foi necessário, também, o campo referente à Operação, e os campos de endereço fonte e destino.

Por fim, para as Operações de Salto foi necessário o campo referente à Operação, um campo de instrução utilizado nos Jumps que necessitam de comparação, e um campo de posição referente à posição para o qual o contador deve pular.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	OP			INSTRUÇÃO ULA			REGISTRADOR FONTE UM					REGISTRADOR FONTE 2					REGISTRADOR DESTINO																
AND	0	0	1	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X											
OR	0	0	1	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X											
XOR	0	0	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X											
NOT	0	0	1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X											
CMP	0	0	1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X											
ADD	0	0	1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X											
SUB	0	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X											
	OP			INSTRUÇÃO ULA			REGISTRADOR FONTE					REGISTRADOR DESTINO																					
LD	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X																
ST	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X																
	OP			INSTRUÇÃO ULA			POSIÇÃO																										
J	1	0	0	X	X	X	X	X	X	X	X																					
	OP			INSTRUÇÃO ULA			REGISTRADOR FONTE					POSIÇÃO																					
JN	1	0	1	1	0	1	X	X	X	X	X	X	X	X	X	X																
JZ	1	1	0	1	0	1	X	X	X	X	X	X	X	X	X	X																

Entretanto por não termos visto a necessidade de utilizar todos bits nas instruções, segue abaixo a imagem com os bits significativos das instruções. Os bits não utilizados, então, são preenchidos sempre com 0.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11
	OP			INSTRUÇÃO ULA			REGISTRADOR FONTE UM					REGISTRADOR FONTE 2					REGISTRADOR DESTINO				
AND	0	0	1	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
OR	0	0	1	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
XOR	0	0	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
NOT	0	0	1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CMP	0	0	1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
ADD	0	0	1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SUB	0	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	OP			INSTRUÇÃO ULA			REGISTRADOR FONTE					REGISTRADOR DESTINO									
LD	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X					
ST	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X					
	OP			INSTRUÇÃO ULA			POSIÇÃO														
J	1	0	0	X	X	X	X	X	X	X	X										
	OP			INSTRUÇÃO ULA			REGISTRADOR FONTE					POSIÇÃO									
JN	1	0	1	1	0	1	X	X	X	X	X	X	X	X	X	X					
JZ	1	1	0	1	0	1	X	X	X	X	X	X	X	X	X	X					

1.3 Modos de endereçamento de operandos

Foi utilizado o modo de endereçamento direto para todas as operações do processador.

1.4 Tamanho do banco de registradores

Foi adotado o tamanho de 32 registros de 32 bits para o banco de Registradores.

1.5 Tamanho das memórias de instruções e de dados

Foi adotado o tamanho de 32 registros de 32 bits para as memórias de instruções e dados.

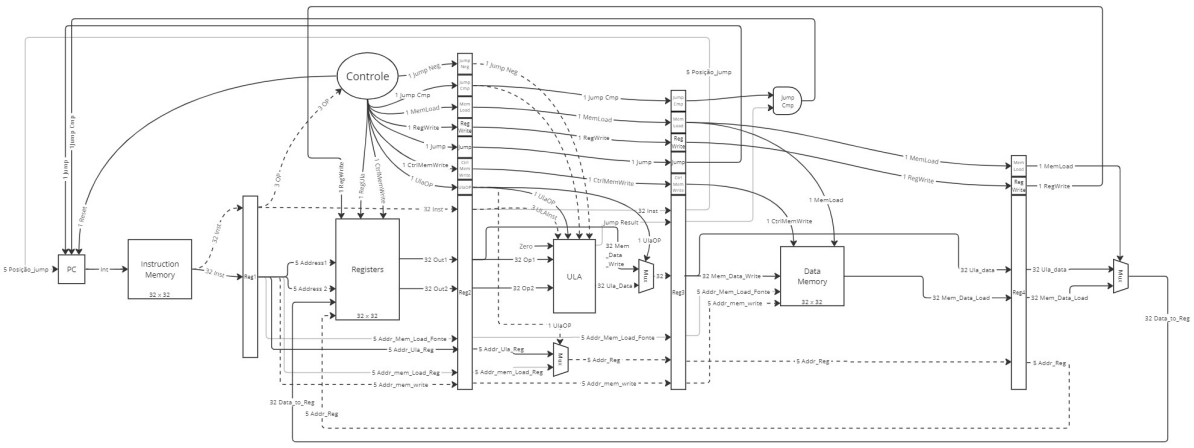
1.6 Número e tipos de barramentos (ou canais dedicados) da parte operativa

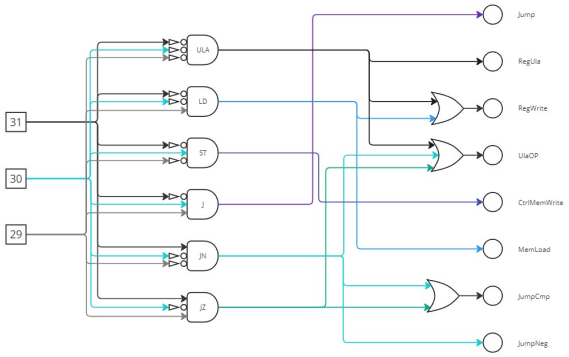
Foram utilizados 62 canais dedicados para fazer a conexão entre os componentes da parte operativa.

1.7 Organização do pipeline

O pipeline foi organizado em 5 estágios: Contador e seleção de instrução, movimentação do banco de registradores, execução da ULA, movimentação da memória e o envio de dados ao banco de registradores.

2 Diagramas





	Jump	RegUla	RegWrite	UlaOP	CtrlMem Write	MemLoad	JumpCmp	JumpNeg
AND	0	1	1	1	0	0	0	0
OR	0	1	1	1	0	0	0	0
XOR	0	1	1	1	0	0	0	0
NOT	0	1	1	1	0	0	0	0
CMP	0	1	1	1	0	0	0	0
ADD	0	1	1	1	0	0	0	0
SUB	0	1	1	1	0	0	0	0
LD	0	0	1	0	0	1	0	0
ST	0	0	0	0	1	0	0	0
J	1	0	0	0	0	0	0	0
JN	0	0	0	1	0	0	1	1
JZ	0	0	0	1	0	0	1	0

3 Implementação

3.1 Implementação do modelo da arquitetura em linguagem para descrição de hardware (VHDL ou SystemC)

O processador foi implementado na linguagem SystemC, para isto, nós implementamos separadamente os componentes: PC, memória de instrução, registradores de pipeline, banco de registradores, ULA, memória de dados, controlador e os MUX de 5 e 32 bits.

Para interligar os componentes, foi implementado um arquivo que instância os sinais que representam os fios entre os componentes.

3.2 Resultados de simulações da execução de instruções de pelo menos 3 algoritmos na arquitetura

Foram executados 3 algoritmos e é possível ver as instruções de cada algoritmo, os registradores e a memória de dados após a execução e a forma de onda da execução de cada algoritmo.

3.2.1 Algoritmo 1

```
...
1   NOT 2 3 10
2   SUB 11 2 3
3   AND 12 4 5
4   OR 13 6 7
5   JN 10 9
6   ADD 14 8 15
7   LD 7 24
8   ST 5 17
9   ST 6 18
10  ST 8 20
11  ST 9 21
12  ST 10 22
13
```

Figura 1 – Instruções do Algoritmo 1

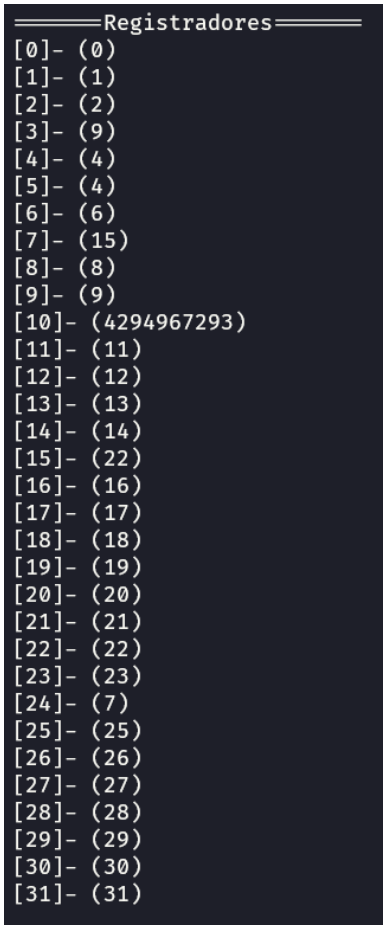


Figura 2 – Registradores após a execução do Algoritmo 1

```
=======Memória de Dados=======
[0]- (0)
[1]- (1)
[2]- (2)
[3]- (3)
[4]- (4)
[5]- (5)
[6]- (6)
[7]- (7)
[8]- (8)
[9]- (9)
[10]- (10)
[11]- (11)
[12]- (12)
[13]- (13)
[14]- (14)
[15]- (15)
[16]- (16)
[17]- (4)
[18]- (18)
[19]- (19)
[20]- (8)
[21]- (9)
[22]- (4294967293)
[23]- (23)
[24]- (24)
[25]- (25)
[26]- (26)
[27]- (27)
[28]- (28)
[29]- (29)
[30]- (30)
[31]- (31)
```

Figura 3 – Memória de Dados após a execução do Algoritmo 1

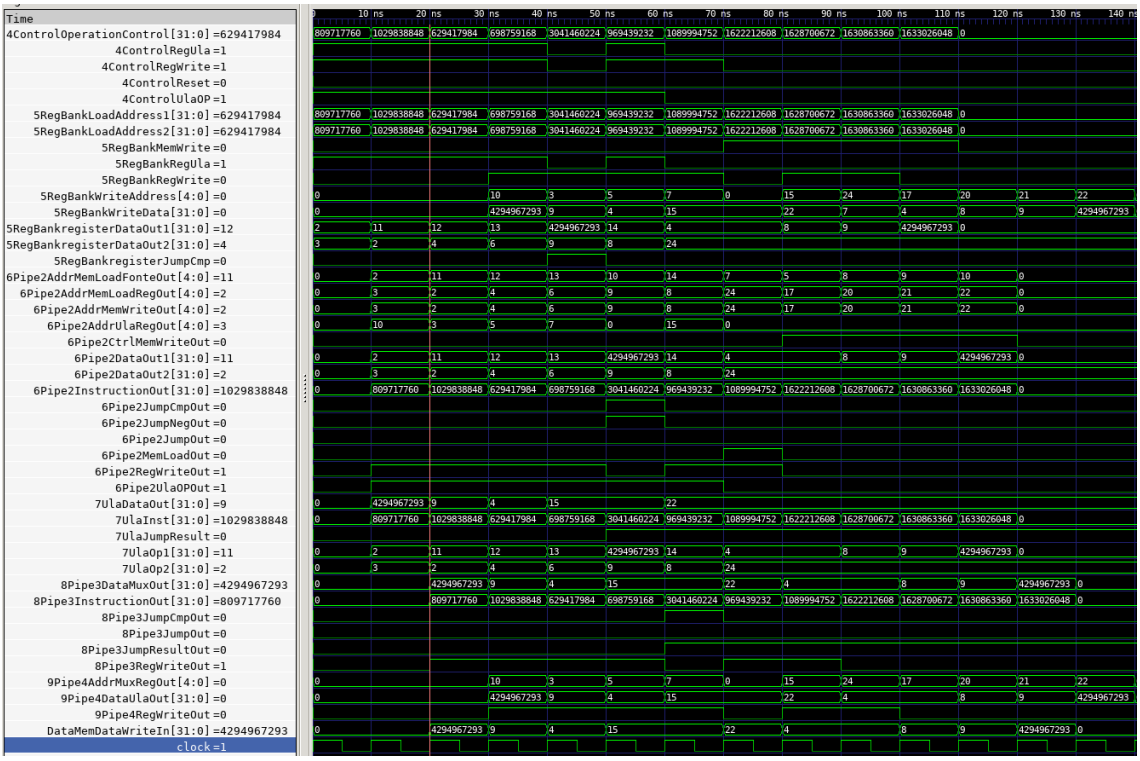


Figura 4 – Formas de onda do Algoritmo 1

3.2.2 Algoritmo 2

```
1  ADD 15 16 3
2  LD 27 1
3  LD 27 2
4  ADD 17 18 4
5  OR 1 3 5
6  JZ 0 11
7  CMP 27 1 29
8  OR 2 4 6
9  ST 3 10
10 ST 4 11
11 ST 3 12
12 ST 4 13
13
```

Figura 5 – Instruções do Algoritmo 2

```
=======Registradores=======
[0]- (0)
[1]- (27)
[2]- (27)
[3]- (31)
[4]- (35)
[5]- (31)
[6]- (59)
[7]- (7)
[8]- (8)
[9]- (9)
[10]- (10)
[11]- (11)
[12]- (12)
[13]- (13)
[14]- (14)
[15]- (15)
[16]- (16)
[17]- (17)
[18]- (18)
[19]- (19)
[20]- (20)
[21]- (21)
[22]- (22)
[23]- (23)
[24]- (24)
[25]- (25)
[26]- (26)
[27]- (27)
[28]- (28)
[29]- (1)
[30]- (30)
[31]- (31)
```

Figura 6 – Registradores após a execução do Algoritmo 2

```
=======Memória de Dados=======
[0]- (0)
[1]- (1)
[2]- (2)
[3]- (3)
[4]- (4)
[5]- (5)
[6]- (6)
[7]- (7)
[8]- (8)
[9]- (9)
[10]- (31)
[11]- (11)
[12]- (12)
[13]- (35)
[14]- (14)
[15]- (15)
[16]- (16)
[17]- (17)
[18]- (18)
[19]- (19)
[20]- (20)
[21]- (21)
[22]- (22)
[23]- (23)
[24]- (24)
[25]- (25)
[26]- (26)
[27]- (27)
[28]- (28)
[29]- (29)
[30]- (30)
[31]- (31)
```

Figura 7 – Memória de Dados após a execução do Algoritmo 2

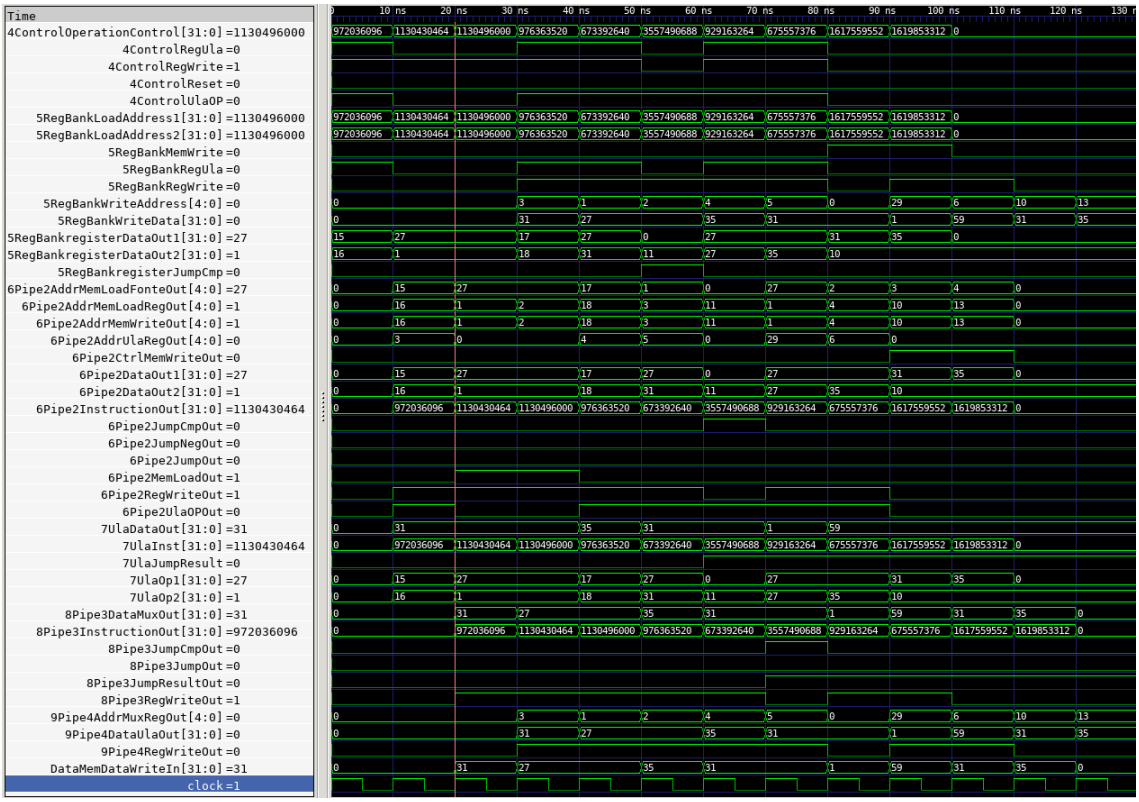


Figura 8 – Formas de onda do Algoritmo 2

3.2.3 Algoritmo 3

```
1  LD 15 1
2  LD 27 2
3  J 7
4  OR 1 2 4
5  ST 2 7
6  ST 3 8
7  ADD 1 3 5
8  ADD 2 3 6
9  ST 2 8
10 ST 3 9
11
```

Figura 9 – Instruções do Algoritmo 3

```
=======Registradores=======
[0]- (0)
[1]- (15)
[2]- (27)
[3]- (3)
[4]- (15)
[5]- (5)
[6]- (30)
[7]- (7)
[8]- (8)
[9]- (9)
[10]- (10)
[11]- (11)
[12]- (12)
[13]- (13)
[14]- (14)
[15]- (15)
[16]- (16)
[17]- (17)
[18]- (18)
[19]- (19)
[20]- (20)
[21]- (21)
[22]- (22)
[23]- (23)
[24]- (24)
[25]- (25)
[26]- (26)
[27]- (27)
[28]- (28)
[29]- (29)
[30]- (30)
[31]- (31)
```

Figura 10 – Registradores após a execução do Algoritmo 3

```
=======Memória de Dados=======
[0]- (0)
[1]- (1)
[2]- (2)
[3]- (3)
[4]- (4)
[5]- (5)
[6]- (6)
[7]- (27)
[8]- (27)
[9]- (3)
[10]- (10)
[11]- (11)
[12]- (12)
[13]- (13)
[14]- (14)
[15]- (15)
[16]- (16)
[17]- (17)
[18]- (18)
[19]- (19)
[20]- (20)
[21]- (21)
[22]- (22)
[23]- (23)
[24]- (24)
[25]- (25)
[26]- (26)
[27]- (27)
[28]- (28)
[29]- (29)
[30]- (30)
[31]- (31)
```

Figura 11 – Memória de Dados após a execução do Algoritmo 3

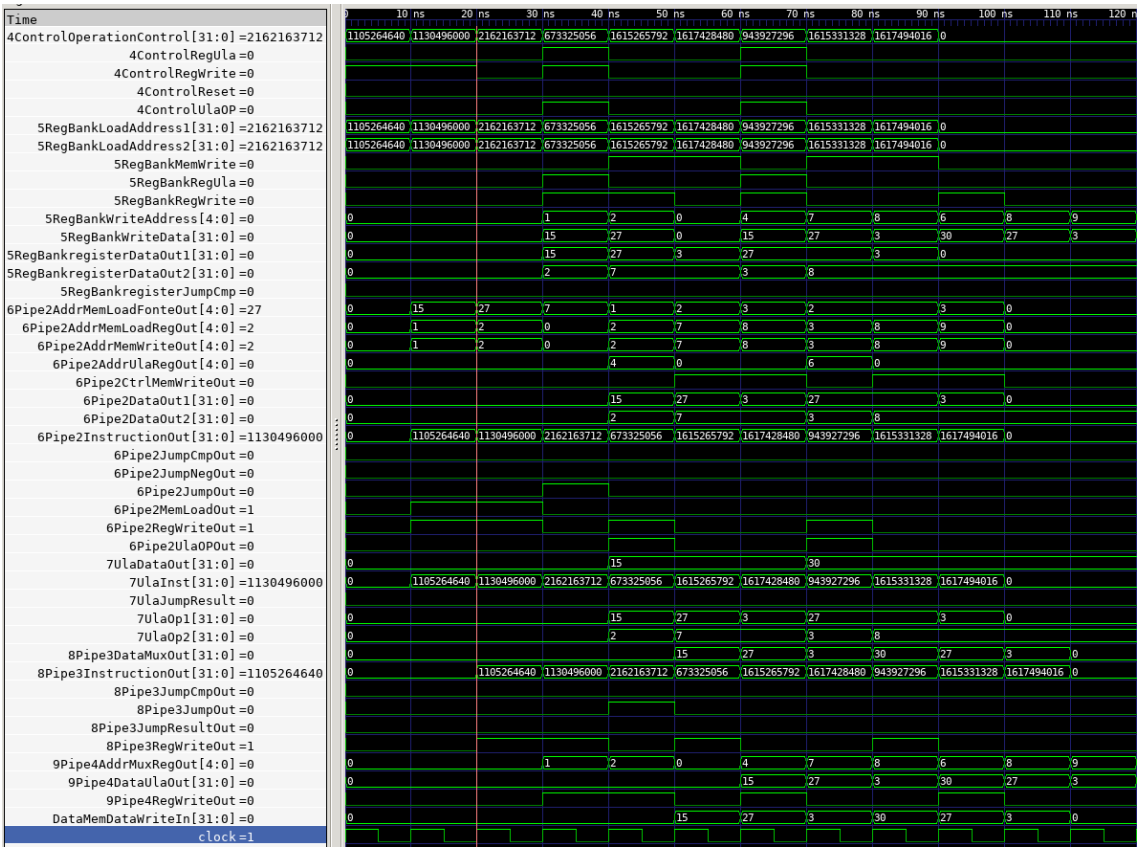


Figura 12 – Formas de onda do Algoritmo 3

3.3 Relatório explicando e exemplificando a implementação da arquitetura e justificando as decisões de projeto acima elencadas.

Para a implementação da arquitetura, primeiro foi projetado o diagrama operativo de como seria o processador, tomando como base o processador MIPS. Inicialmente, foi pensado em construir o processador com 4 fases de pipeline, entretanto, devido ao carregamento em registradores tanto da ula como da memória, optamos por 5 fases com uma fase separada para este carregamento.

Na memória de instruções há somente uma entrada que representa a posição da instrução que deve ser lida a seguir, e por fim, o contador do programa irá incrementar a cada clock a posição da instrução que deve ser lida, ou, quando há uma instrução de Jump, ela pulará para a posição que recebeu da instrução.

Para o banco de registradores, há a entrada de 3 sinais de controle que determinam se irá escrever dados no registrador, se devem sair dados para a ULA ou se devem sair dados para a escrita na memória, também há duas entradas de endereços apontando para os dados que devem sair.

Para a ULA, temos a entrada de 3 sinais de controle, um que ditará se a ula será usada ou não, e os outros dois para coordenar as operações das instruções Jump se negativo e Jump se zero. Foi decidido 2 sinais para o jump, pois um ditará se está ocorrendo o jump ou não, e o outro ditará qual comparação deve ser feita. O resultado de operações de instruções aritméticas saem da ULA em direção ao Mux, já o resultado do jump sai como um bit que informará se a comparação do dado recebido do registrador fonte atendeu à comparação do Jump requisitado.

Ainda no mesmo estágio do pipeline temos 2 Mux, um de 32 bits que coordenará os dados de escrita em memória ou do resultado da ULA, e um de 5 bits que coordenará o endereço do registrador destino de operações da ULA ou o endereço de carregamento de dados da memória para o registrador.

Na memória do programa decidimos por fazer 2 sinais de controle sendo um para leitura e outro para escrita para melhor controle sobre as entradas.

No mesmo estágio também é executada as instruções de Jump, no caso do jump absoluto é passado a posição para o qual deve pular e executa o jump, para os Jumps comparativos ainda existe uma verificação do tipo AND para verificar se o resultado da ULA foi verdadeira.

4 Conclusão

Concluímos que é possível implementar um processador RISC com 5 etapas de pipeline sem necessitar de tantos recursos, apenas utilizando a biblioteca SystemC de C++. No entretanto, a dificuldade se deu principalmente na resolução de conflitos do pipeline, o problema mais crítico ao se utilizar pipeline em um processador.