

Universidade Federal do Rio Grande do Norte — UFRN

Instituto Metrópole Digital — IMD

**Disciplina:** Arquitetura de Computadores — DIM 0127

**Turma:** T01 24M34 (2019.1)

**Componentes:** Alexandre Alves Andrade

Francisco de Assis Campos Júnior

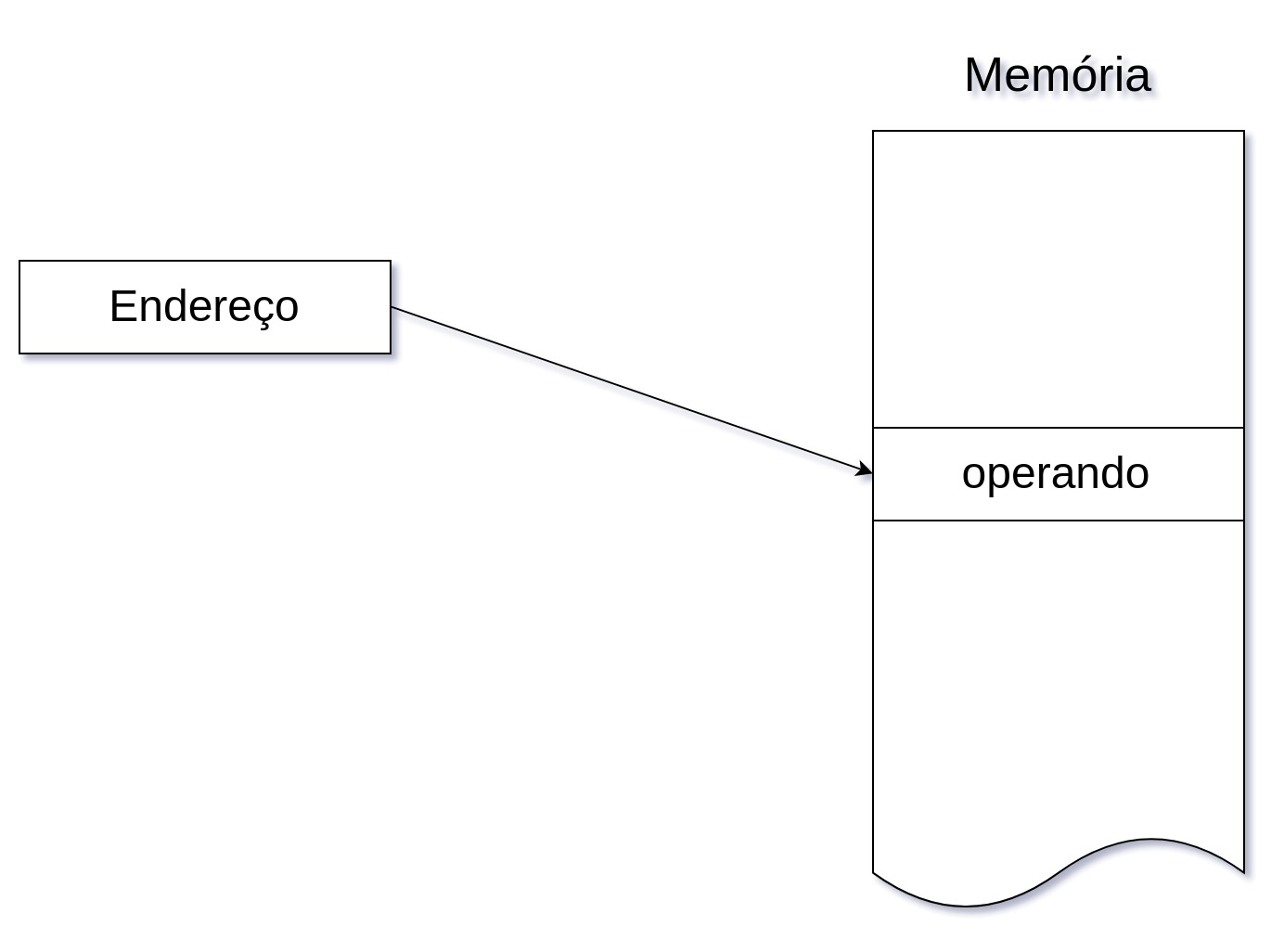
Kaio Henrique de Sousa

**Processador Neander**

Características:

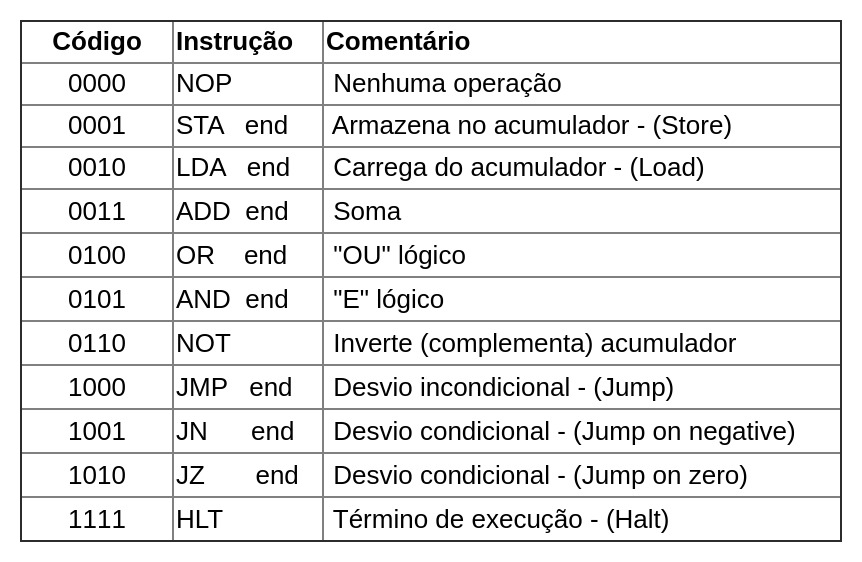
* Largura do dados e endereços de 8 bits
* Dados Representados em complementos de dois
* 1 acumalador de 8 bits (AC)
* 1 apontador de programa de 8 bits(PC)
* 1 registrador de estado com 2 códigos de condição: negativo(N) e zero (Z)

O processador Neander só possui um modo de endereçamento: O direto. Nesse modo, a palavra que segue o código da instrução contém, nas instruções de manipulação dedos, o endereço de memória do operando(figura 1.1).



*Figura 1.1 - Modo de endereçamento direto*

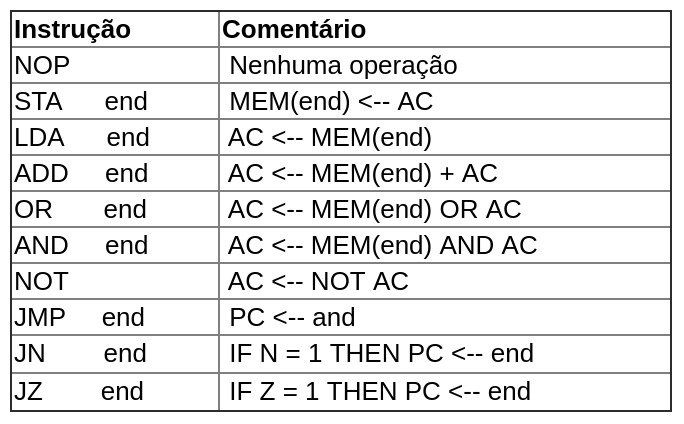
O conjunto de instruções de Neander compreende 11 instruções, codificadas através dos quatro bits mais significativos da palavra que contém o código da instrução(tabela 1.1):

**

*Tabela 1.1 - Conjuntos de instruções NEANDER*

Na tabela 1.1. *end* significa endereço direto. Nas instruções STA, LDA, ADD, OR e AND. *end* corresponde ao endereço de operando. Nas instruções JMP, JN e JZ, *end* corresponde ao endereço de desvio.

As ações efetuadas por cada uma das instruções da tabela 1.1 podem ser vistas na tabela 1.2:



*Tabela 1.2 - Ações executadas*

Na tabela 1.2:

AC é o acumulador;

MEM(end) significa conteúdo da posição end de memória;

N e Z são os códigos de condição;

E “<--” representa uma atribuição.

A unidade lógica e aritmética do processador Neander fornece dois códigos de condição, que são usados pelas instruções JN e JZ. Veja a seguir:

N - (Negative) - sinal do resultado

true - resultado é negativo

false - resultado é positivo

Z - (Zero) - indica resultado igual a zero

true - resultado é igual a zero

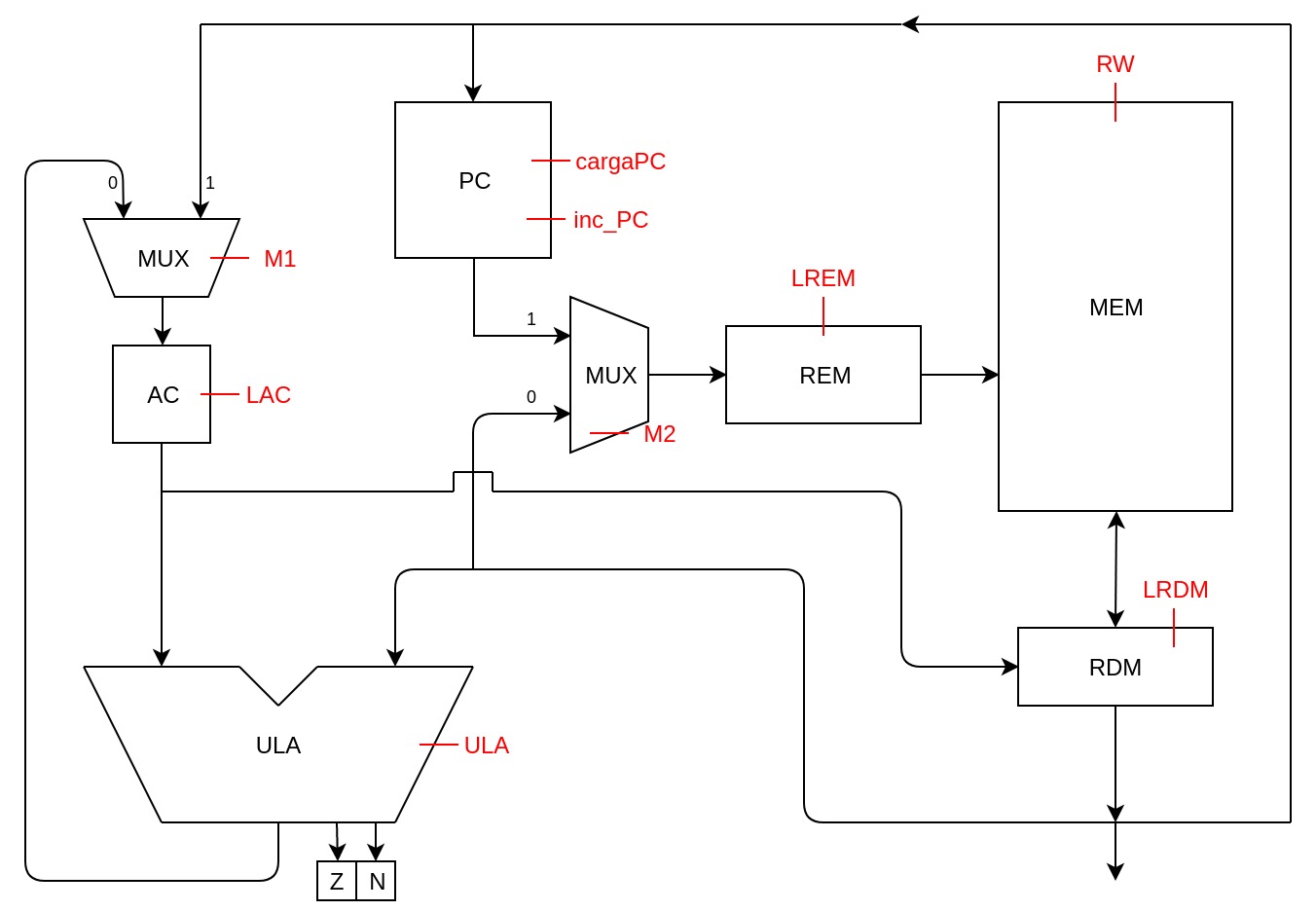
false - resultado é diferente de zero

Parte Operativa - PO

A parte operativa do processador utilizei dos seguintes componentes:

* MEM - Memória
* RDM - Registrador de dados na memória
* REM - Registrador de endereços na memória
* PC - Parte de controle(figura x.x)
* AC - Acumulador
* 2 MUX - Multiplexadores
* ULA
* Sinais de controle( Z e N )

Veja a tabela a seguir com a organização dos componentes mencionados (figura 1.2):



*Figura 1.2 - Parte Operativa(PO) do processador Neander*

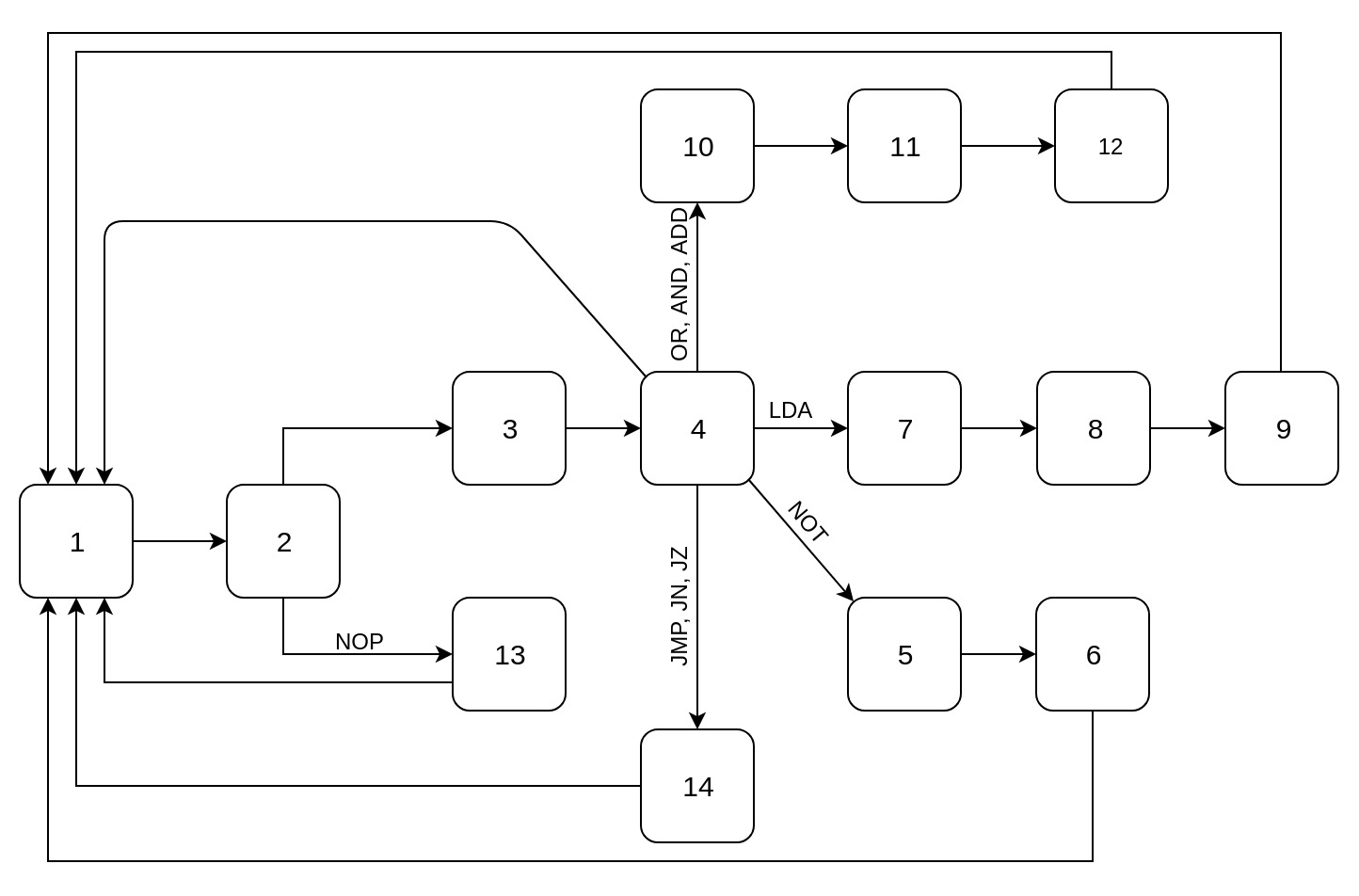
Na figura 1.2:

* M1 e M2 escolha da saída de seus respectivos multiplexadores
* LAC - Load ACumulador
* ULA - Controla a operação da ULA
* LRDM - Load Registro de dados da/para memória
* LREM - Load Registro de endereços para memória
* cargaPC - recebe endereço para o *Jump*
* inc\_PC - incrementa em 1 a parte de controle
* RW - Read(Ler) ou Write(Escrever) da/na memória

A memória recebe ligações da RDM e REM, a RDM Possui ligação direta com a ULA, e para uma MUX que tem conexão com o AC, assim é possível fazer uma operação em poucos estados. A ULA recebe dados do AC e também manda dados para a mesma, possuindo também, ligação com os sinais de controle Z(zero) e N(Negative). A PC pode receber endereços da RDM para um salto nas instruções da memória, como também possui o sinal de controle *inc\_PC* que incrementa em 1 o estado da PC, e está ligada a um segundo MUX que redireciona para a REM ligada a Memória.

Parte de Controle - PC

Veja abaixo a parte de controle do Neander, que controla os estados para a execução das instruções, antes vistas(tabela 1.1), existentes no processador(figura 1.3).



*Figura 1.3 - Parte de controle(PC) do processador Neander*

Na figura 1.3. Temos (1) como o primeiro estado, onde começa qualquer que seja a instrução a ser executada, o estado (2) há duas ramificações, (3) e (13), (13) é executado o NOP e o (3) segue com o (4) que possui cinco ramificações, são elas com suas respectivas instruções: (5)(6) cuidará do NOT, (7)(8)(9) para a execução do LDA, (10)(11)(12) para o OR, AND e ADD pois possuem o mesmo conjunto de micro-instrução, apenas com alteração na ULA, (14) JMP, JN e JZ segue também com o mesmo princípio de micro-instruções idênticas apenas com alteração na ULA. E todas elas voltam para o início do programa, o estado (1), esperando uma nova instrução.

Para a execução de cada instrução é necessário que haja sinais de controle, micro-instruções, para cada componente do processador nos diferentes estados formando uma instrução completa, abaixo segue cada microinstrução em seu respectivo estado:

**Função de Saída(FS)**

(1) M1(0); LREM; RW(0); inc\_PC

(2) LRDM

(3) M1(0); LREM; RW(0); inc\_PC

(4) LRDM

STA

(5) M1(1); LREM; LRDM

(6) RW(1)

LDA

(7) M1(1); LREM; RW(0)

(8) LRDM

(9) M2(1); AC

ADD, OR, AND

(10) M1(1); LREM; RW(0)

(11) LRDM; ULA(ADD || OR || AND)

(12) M2(0); AC

NOT

(13) ULA(NOT); M2(0); AC

JMP, JN, JZ

(14) cargaPC

**Processador Ramsés**

O presente projeto consiste na implementação de um simulador do processador Ramsés, o qual será divido em parte operativa, parte de controle, diagrama de estados, função de saída e implementação na linguagem de programação C++ (segue em anexo os códigos fonte). As divisões citadas acima serão detalhadas a seguir.

**Parte operativa (PO):**

O processador tem em sua parte operativa (PO) como componentes dois registradores de uso geral (RA, RB), um registrador de índice (RX), uma ULA, um somador, três multiplexadores de três entradas, um multiplexador de duas entradas, o contador de programa (PC), o registrador de endereço de memória (REM), o registrador de dados de memória (RDM), o registrador de instrução (RI), flags de controle (N, Z, C) e a Unidade de Controle que manda os sinais de controle que são:

1. LA: Carrega no RA
2. LB: Carrega no RB
3. LX: Carrega no RX
4. OP: seleciona operação da ULA
5. MUX1(-): entradas 0, 1 ou 2
6. MUX2(-): entradas 0, 1 ou 2
7. MUX3(-): entradas 0, 1 ou 2
8. MUX4(-): entradas 0 ou 1
9. L+: seleciona somador
10. Inc-PC: incrementa 1 na PC
11. LPC: Carrega na PC
12. LREM: Carrega na REM
13. LRDM: Carrega na RDM
14. Read: lê dado da MEM
15. Write: escreve dado na MEM
16. LRI: Carrega registrador de instrução
17. LN: Carrega na flag N
18. LZ: Carrega na flag Z
19. LC: Carrega na flag C

A ULA tem como operações ADD, AND, OR, NEG, NOT, SUB, SHR e tem como entrada um dos registradores e um dado da memória.

Os registradores e a memória têm largura de oito bits e os dados são representados em complemento de dois.

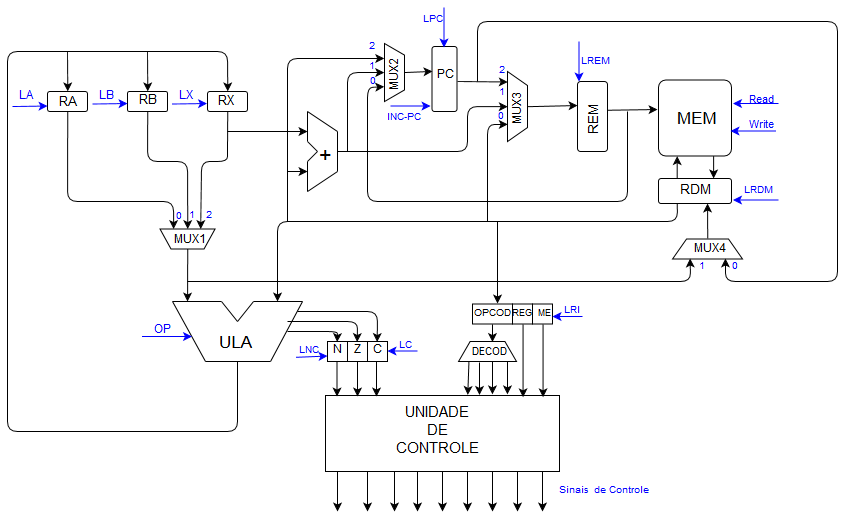


Figura 1.1 - Parte Operativa

**Instruções:**

As instruções do Ramsés são formadas por dois bytes, no qual o primeiro byte contém o código da instrução (4 bits mais significativos), uma referência ao processador (bits 3 e 2), e seleção do modo de endereçamento (bits 1 e 0). No segundo bytes há um endereço de memória ou um dado imediato.O conjunto de instruções do Ramsés é composto por 16 instruções, codificadas através dos quatro bits mais significativos do byte de instrução, Conforme mostrado na tabela 1.1:

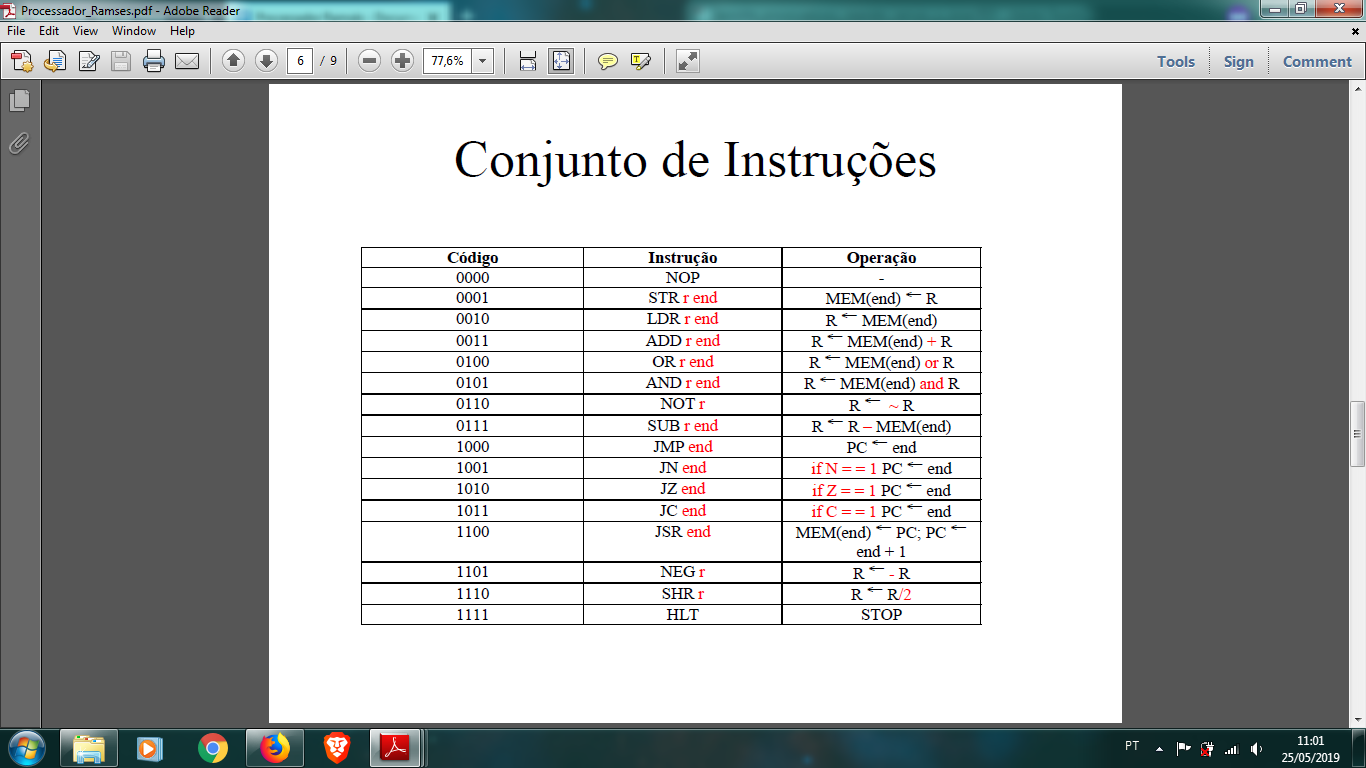


Tabela 1.1 - conjunto de instruções

**Diagrama de Estados:**

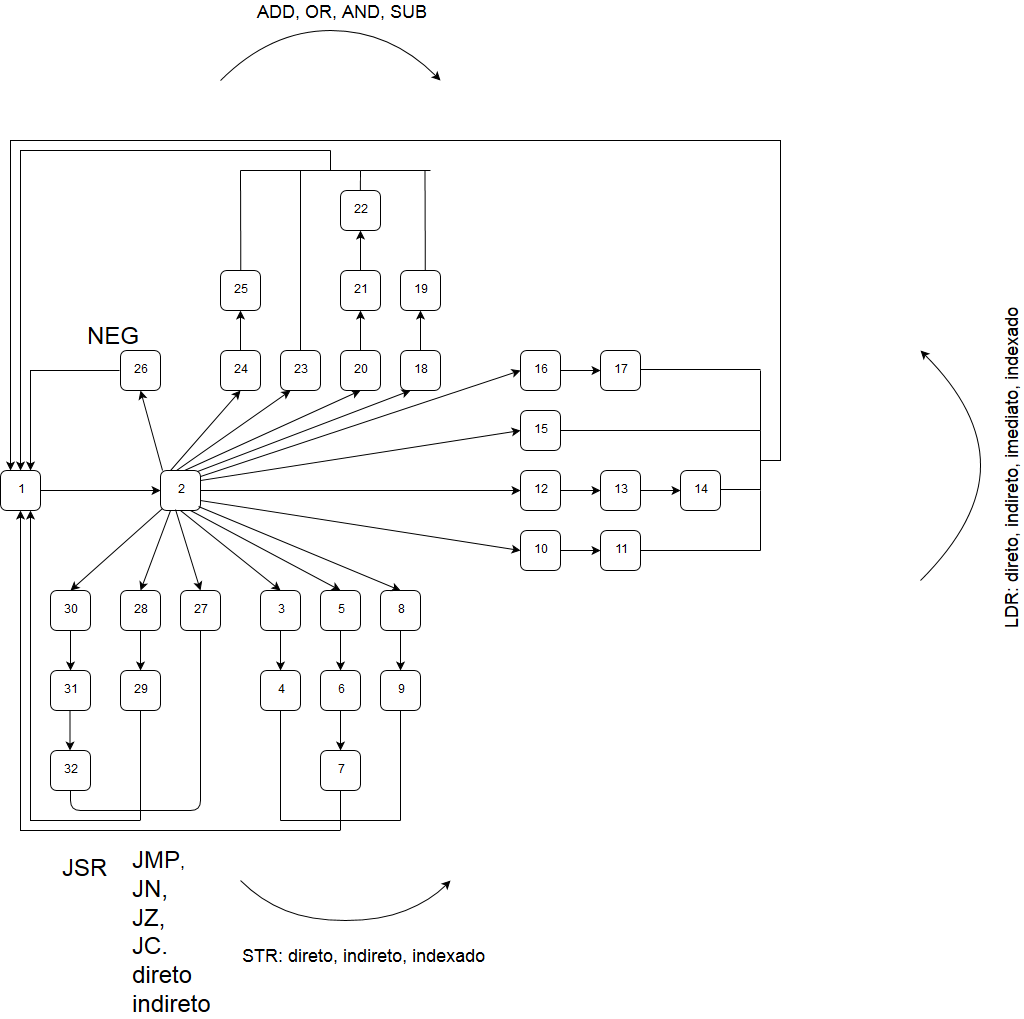


Figura 1.2 - Diagrama de Estados

**Função de Saída (FS):**

A seguir é exemplificado como os componentes internos executam cada instrução do conjunto de instruções do processador seguindo a Figura 1.2.

Busca

1.MUX1(2); LREM; Read; LRDM; Inc-PC; LRI;

2.MUX3(2); LREM; Read; LRDM; Inc-PC;

STR - Direto

3.MUX3(0); LREM; MUX1( - ); MUX4(1);

4.LRDM; Write;

STR - Indireto

5.MUX3(0); LREM; Read; LRDM;

6.MUX3(0); LREM; MUX1( - ); MUX4(1);

7.LRDM; Write;

STR - Indexado

8.L+; MUX3(1); LREM; MUX1(-);

9.MUX4(1); LRDM; Write;

LDR - Direto

10.MUX3(0); LREM; Read; LRDM;

11.ULA(NOP); L(N || Z); LR(A || B || X);

LDR - Indireto

12.MUX3(0); LREM; Read; LRDM;

13.MUX3(0); LREM; Read; LRDM;

14.ULA(NOP); L(N || Z); LR(A || B || X);

LDR - Imediato

15.ULA(NOP); L(N || Z); LR(A || B || X);

LDR - Indexado

16.L+; MUX3(0); LREM; Read; LRDM;

17.ULA(NOP); L(N || Z); LR(A || B || X);

ADD; OR; AND; SUB - Direto

18.MUX3(0); LREM; Read; LRDM;

19.MUX1(0 || 1 || 2); ULA(ADD || OR || AND || SUB); L(N || Z || C); LR(A || B || X);

ADD; OR; AND; SUB - Indireto

20.MUX3(0); LREM; Read; LRDM;

21.MUX3(0); LREM; Read; LRDM;

22.MUX1(0 || 1 || 2); ULA(ADD || OR || AND || SUB); L(N || Z || C); LR(A || B || X);

ADD; OR; AND; SUB - Imediato

23.MUX1(0 || 1 || 2); ULA(ADD || OR || AND || SUB); L(N || Z || C); LR(A || B || X);

ADD; OR; AND; SUB - Indexado

24.L+; MUX3(1); LREM; Read; LRDM;

25.MUX1(0 || 1 || 2); ULA(ADD || OR || AND || SUB); L(N || Z || C); LR(A || B || X);

NOT; NEG; SHR - Direto

26. MUX1(0 || 1 || 2); ULA(NOT || NEG || SHR); L(N || Z); LR(A || B || X);

JMP; JN; JZ; JC - Direto

27. MUX2(2); LPC;

JMP; JN; JZ; JC - Indireto

28. MUX3(0); LREM; Read; LRDM;

29. MUX2(2); LPC;

JSR

30. MUX4(0); LRDM;

31. LPC; MUX3(2); LREM;

32. Write; Inc-PC;

**Análise geral:**

1 - Somar duas variáveis:

Tanto o Neander com o Ramsés apresentaram o mesmo tamanho de programa com 2 instruções para fazer a soma mesma dificuldade, ambos precisaram de 15 ciclos de relógio para executar o programa.

2 - Subtrair duas variáveis:

Enquanto o Neander o tamanho do programa é de 4 instruções o Ramsés precisou de apenas duas. Para implementar a subtração no Neander, o programador precisa saber que não há tal instrução pronta, desse modo, é preciso fazer a negação da variável e somar um para obter o valor negativo, pois os números estão sendo representados em complemento de dois, porém o Ramsés possui a operação de subtração em seu conjunto de instruções.

Devido a maior dificuldade de implementação o Neander precisou de 25 ciclos de relógio, enquanto o Ramsés fez o algoritmo em 15 ciclos de relógio.

3 - Chamar o programa 2 como sub-rotina por um programa principal:

Este algoritmo não é programável no Neander. Logo foi feito apenas no Ramsés e para implementar programador deve saber como a instrução JSR funciona.

A JSR é uma instrução de sub-rotina, que altera o contador de programa para outra posição e nessa posição é gravado o valor anterior do contador (endereço de memória), após executar uma sub-rotina o programador deve chamar a JSR indireta, no endereço de memória do início da sub-rotina e assim o programa retorna para uma instrução após a chamada da JSR.

O programa principal de tem 4 instruções e ao todo tem 7 instruções, que executam em um tempo de 46 clocks totais.