

# O Computador Neander

## Neander - Computador Hipotético Didático

The image displays the Neander computer simulation interface, which consists of three main windows: **Programa**, **Neander**, and **Dados**.

**Programa Window:** A table listing memory addresses (End.) and instructions (Mnemônico). The first instruction at address 0 is NOP. The base pointer (BP) is set to 255, and the register [0] contains the value 0.

P	End.	Dado	Mnemônico
→	0	0	NOP
	1	0	NOP
	2	0	NOP
	3	0	NOP
	4	0	NOP
	5	0	NOP
	6	0	NOP
	7	0	NOP
	8	0	NOP
	9	0	NOP
	10	0	NOP
	11	0	NOP
	12	0	NOP
	13	0	NOP
	14	0	NOP
	15	0	NOP
	16	0	NOP

**Neander Window:** The central control panel showing the state of the computer. It includes registers AC (Accumulator) and PC (Program Counter), both displaying 000. Status indicators N (Not) and Z (Zero) are shown. Execution statistics for accesses and instructions are displayed as 00000. The current instruction is NOP, with its register value also set to 0. A menu bar at the top includes Arquivo, Editar, Visualizar, Executar, and ?.

**Mnemônicos Table:**

Mnemonic	Value	End
NOP	00	
STA	16	end
LDA	32	end
ADD	48	end
OR	64	end
AND	80	end
NOT	96	
JMP	128	end
JN	144	end
JZ	160	end
HLT	240	

**Dados Window:** A table for data storage, showing addresses (End.) and data values (Dado). All values are currently 0.

End.	Dado
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0

**Footer:** The bottom right corner contains the Neander logo, version information (Versão 2.1, Julho '2002), authors (Raul Fernando Weber, Taicy Silva Weber), and the version of the operating system (Versão: Fabio Augusto Dal Castel, Win32).

# O Computador Neander

## ► Arquitetura: características gerais

- Largura de dados e endereços de 8 bits
- Dados representados em complemento de 2
- 1 acumulador 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)
- 1 modo de endereçamento: end. direto a memória

\* Esta pseudo-máquina foi criada em homenagem ao homem de Neanderthal, o antecessor do Homo Sapiens. :-)

\* Computadores e Simuladores didáticos criados a partir da série iniciada pelo Neander: Neander , Ahmes, Ramses e Cesar  
Simuladores Neander, Ahmes, Daedalus (Multi-Assembler Ahmes, Ramses e Cesar)

# O Computador Neander

## ► **Organização: alguns elementos necessários**

- **Um registrador de 8 bits para servir de acumulador**
- **Um registrador de 8 bits para o PC (registrador-contador)**
- **Dois flip-flops: um para o código de condição N e outro para Z**
- **Uma memória de 256 posições (endereços) x 8 bits**

# O Computador Neander

## Neander - Instruction Set

### ► Arquitetura: conjunto de instruções

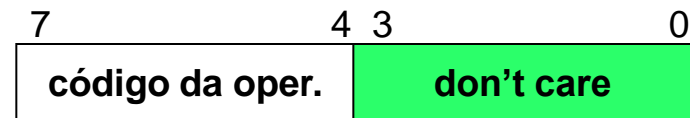
código	instrução	comentário
0000	NOP	Nenhuma operação
0001	STA end	$\text{MEM}(\text{end}) \leftarrow \text{AC}$
0010	LDA end	$\text{AC} \leftarrow \text{MEM}(\text{end})$
0011	ADD end	$\text{AC} \leftarrow \text{MEM}(\text{end}) + \text{AC}$
0100	OR end	$\text{AC} \leftarrow \text{MEM}(\text{end}) \text{ OR } \text{AC}$
0101	AND end	$\text{AC} \leftarrow \text{MEM}(\text{end}) \text{ AND } \text{AC}$
0110	NOT	$\text{AC} \leftarrow \text{NOT } \text{AC}$
1000	JMP end	$\text{PC} \leftarrow \text{end}$
1001	JN end	IF N=1 THEN $\text{PC} \leftarrow \text{end}$
1010	JZ end	IF Z=1 THEN $\text{PC} \leftarrow \text{end}$
1111	HLT	pára processamento

# O Computador Neander

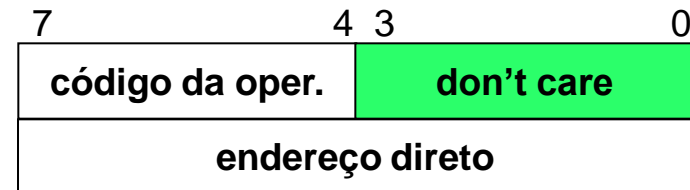
## ► Arquitetura: formato das instruções

As instruções do Neander possuem um ou dois bytes (ocupam uma ou duas posições de memória)

Instruções com um byte:  
NOP, NOT



Instruções com dois bytes:  
STA, LDA, ADD, OR, AND,  
JMP, JN, JZ

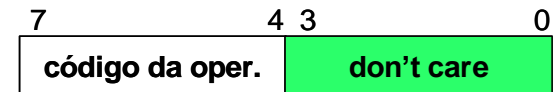


# O Computador Neander

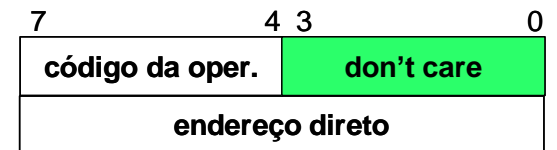
## ► Arquitetura: conjunto de instruções

código	instrução	comentário
0000	NOP	Nenhuma operação
0001	STA end	$\text{MEM}(\text{end}) \leftarrow \text{AC}$
0010	LDA end	$\text{AC} \leftarrow \text{MEM}(\text{end})$
0011	ADD end	$\text{AC} \leftarrow \text{MEM}(\text{end}) + \text{AC}$
0100	OR end	$\text{AC} \leftarrow \text{MEM}(\text{end}) \text{ OR } \text{AC}$
0101	AND end	$\text{AC} \leftarrow \text{MEM}(\text{end}) \text{ AND } \text{AC}$
0110	NOT	$\text{AC} \leftarrow \text{NOT } \text{AC}$
1000	JMP end	$\text{PC} \leftarrow \text{end}$
1001	JN end	IF N=1 THEN $\text{PC} \leftarrow \text{end}$
1010	JZ end	IF Z=1 THEN $\text{PC} \leftarrow \text{end}$
1111	HLT	pára processamento

Instruções com um byte:  
NOP, NOT



Instruções com dois bytes:  
STA, LDA, ADD, OR, AND,  
JMP, JN, JZ



### Mnemônicos

NOP	00	ADD	48 end	JMP	128 end
STA	16 end	OR	64 end	JN	144 end
LDA	32 end	AND	80 end	JZ	160 end
		NOT	96	HLT	240

# O Computador Neander

## ▶ Arquitetura: o ciclo de busca (fetch)

Busca  
instrução



Decodifica  
instrução



Executa/  
Busca operandos

## ▶ Arquitetura: o ciclo de execução

# O Computador Neander

## ► **Arquitetura/Organização:** transferências entre regs.

**A fase de busca:** é igual para todas as instruções

$RI \leftarrow \text{MEM}(\text{PC})$

$\text{PC} \leftarrow \text{PC} + 1$

- Novo elemento é necessário: o **registrador de instrução (RI)**
- $\text{MEM}(\text{PC})$  corresponde a um acesso à memória, usando o conteúdo do PC como fonte do endereço



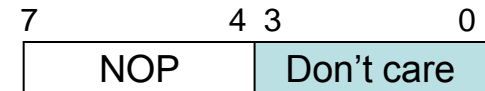
# O Computador Neander

► **Arquitetura/Organização:** transferências entre regs.

## Instrução NOP (nenhuma operação)

**Simbólico:** NOP

**RT:** -



**Passos no nível RT:**

<b>Busca:</b>	$RI \leftarrow MEM(PC)$ $PC \leftarrow PC + 1$
<b>Execução:</b>	nenhuma operação

As transferências indicam quais caminhos de dados devem existir, mas não indicam os caminhos físicos reais entre os elementos (registradores e ULA)

Flags: N e Z não são afetados      Código Binário: 0000

Ciclos de Clock: ...

# O Computador Neander

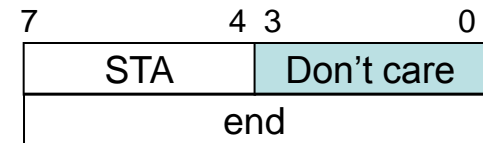
► **Arquitetura/Organização:** transferências entre regs.

## Instrução STA (armazena acumulador)

**Simbólico:** STA end

**RT:**  $\text{MEM}(\text{end}) \leftarrow \text{AC}$

**Passos no nível RT:**



**Busca:**  $\text{RI} \leftarrow \text{MEM}(\text{PC})$   
 $\text{PC} \leftarrow \text{PC} + 1$

**Execução:**  $\text{end} \leftarrow \text{MEM}(\text{PC})$   
 $\text{PC} \leftarrow \text{PC} + 1$   
 $\text{MEM}(\text{end}) \leftarrow \text{AC}$

# O Computador Neander

► **Arquitetura/Organização:** transferências entre regs.

## Instrução LDA (carrega acumulador)

**Simbólico:** LDA end

**RT:**  $AC \leftarrow \text{MEM}(\text{end})$

**Passos no nível RT:**

7	4	3	0
LDA			Don't care
end			

**Busca:**  $RI \leftarrow \text{MEM}(\text{PC})$

$\text{PC} \leftarrow \text{PC} + 1$

**Execução:**  $\text{end} \leftarrow \text{MEM}(\text{PC})$

$\text{PC} \leftarrow \text{PC} + 1$

$\text{AC} \leftarrow \text{MEM}(\text{end});$  atualiza N e Z

# O Computador Neander

## ► Arquitetura/Organização: transferências entre regs.

### Instrução ADD (soma)

**Simbólico:** ADD end

**RT:**  $AC \leftarrow MEM(end) + AC$

**Passos no nível RT:**

7	4	3	0
ADD			Don't care
end			

**Busca:**  $RI \leftarrow MEM(PC)$

$PC \leftarrow PC + 1$

**Execução:**  $end \leftarrow MEM(PC)$

$PC \leftarrow PC + 1$

$AC \leftarrow AC + MEM(end)$ ; atualiza N e Z

# O Computador Neander

## ► Arquitetura/Organização: transferências entre regs.

### Instrução OR (“ou” lógico, bit a bit)

**Simbólico:** OR end

**RT:**  $AC \leftarrow \text{MEM}(\text{end}) \text{ OR } AC$

**Passos no nível RT:**

7	4	3	0
OR			Don't care
end			

**Busca:**  $RI \leftarrow \text{MEM}(\text{PC})$

$\text{PC} \leftarrow \text{PC} + 1$

**Execução:**  $\text{end} \leftarrow \text{MEM}(\text{PC})$

$\text{PC} \leftarrow \text{PC} + 1$

$AC \leftarrow AC \text{ OR } \text{MEM}(\text{end}); \text{ atualiza N e Z}$

# O Computador Neander

## ► Arquitetura/Organização: transferências entre regs.

### Instrução AND (“e” lógico, bit a bit)

**Simbólico:** AND end

**RT:**  $AC \leftarrow \text{MEM}(\text{end}) \text{ AND } AC$

**Passos no nível RT:**

7	4	3	0
AND			Don't care
end			

**Busca:**  $RI \leftarrow \text{MEM}(\text{PC})$

$\text{PC} \leftarrow \text{PC} + 1$

**Execução:**  $\text{end} \leftarrow \text{MEM}(\text{PC})$

$\text{PC} \leftarrow \text{PC} + 1$

$AC \leftarrow AC \text{ AND } \text{MEM}(\text{end});$  atualiza N e Z

# O Computador Neander

► **Arquitetura/Organização:** transferências entre regs.

## Instrução NOT (complementa acumulador)

**Simbólico:** NOT

**RT:**  $AC \leftarrow \text{NOT } AC$

**Passos no nível RT:**

7	4	3	0
NOT			Don't care

**Busca:**  $RI \leftarrow \text{MEM}(\text{PC})$

$\text{PC} \leftarrow \text{PC} + 1$

**Execução:**  $AC \leftarrow \text{NOT}(AC)$ ; atualiza N e Z

# O Computador Neander

► **Arquitetura/Organização:** transferências entre regs.

## Instrução JMP (desvio incondicional - jump)

**Simbólico:** JMP end

**RT:**  $PC \leftarrow \text{end}$

**Passos no nível RT:**

7	4	3	0
JMP			Don't care
end			

**Busca:**  $RI \leftarrow \text{MEM}(PC)$

$PC \leftarrow PC + 1$

**Execução:**  $\text{end} \leftarrow \text{MEM}(PC)$

$PC \leftarrow \text{end}$



# O Computador Neander

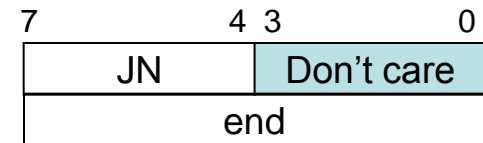
## ► Arquitetura/Organização: transferências entre regs.

### Instrução JN (desvio condicional - jump on negative)

**Simbólico:** JN end

**RT:** IF N = 1 THEN PC  $\leftarrow$  end

**Passos no nível RT:**



#### Se N=1 (desvio ocorre)

**Busca:** RI  $\leftarrow$  MEM(PC)  
PC  $\leftarrow$  PC + 1

**Execução:** end  $\leftarrow$  MEM(PC)  
PC  $\leftarrow$  end

#### Se N=0 (desvio não ocorre)

**Busca:** RI  $\leftarrow$  MEM(PC)  
PC  $\leftarrow$  PC + 1

**Execução:** end  $\leftarrow$  MEM(PC)  
PC  $\leftarrow$  PC + 1

a rigor, desnecessário

# O Computador Neander

## ► Arquitetura/Organização: transferências entre regs.

### Instrução JZ (desvio condicional - jump on zero)

**Simbólico:** JZ end

**RT:** IF Z = 1 THEN PC  $\leftarrow$  end

**Passos no nível RT:**

7	4	3	0
JZ			Don't care
end			

#### Se Z=1 (desvio ocorre)

**Busca:** RI  $\leftarrow$  MEM(PC)

PC  $\leftarrow$  PC + 1

**Execução:** end  $\leftarrow$  MEM(PC)

PC  $\leftarrow$  end

#### Se Z=0 (desvio não ocorre)

**Busca:** RI  $\leftarrow$  MEM(PC)

PC  $\leftarrow$  PC + 1

**Execução:** end  $\leftarrow$  MEM(PC)

PC  $\leftarrow$  PC + 1

a rigor, desnecessário

# O Computador Neander

► **Arquitetura/Organização:** transferências entre regs.

**Instrução HLT (término de execução - halt)**

**Simbólico:** HLT

**RT:** --

**Passos no nível RT:**

7	4	3	0
HLT			Don't care

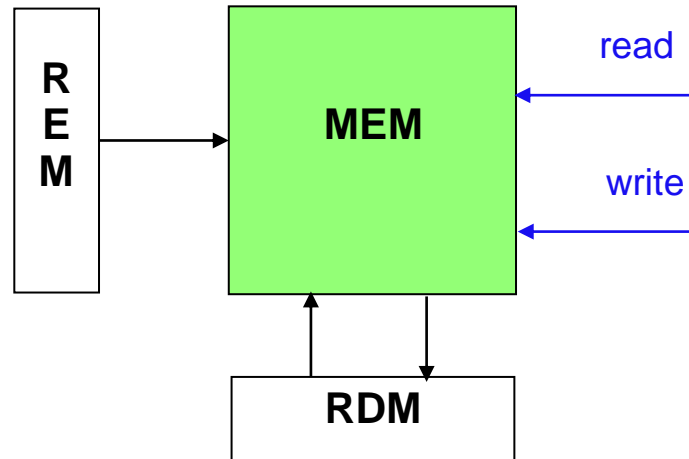
**Busca:**  $RI \leftarrow \text{MEM}(\text{PC})$

$\text{PC} \leftarrow \text{PC} + 1$

**Execução:** parar o processamento

# O Computador Neander

## ► Organização do Sistema de Memória



## ► Arquitetura/Organização

### Operações com a memória

$x \leftarrow \text{MEM}(y)$  descreve uma **leitura** da memória, que é realizada pelos seguintes passos:

1.  $\text{REM} \leftarrow y$       copia  $y$  (que é um endereço) para o REM
2. Read      ativação de uma operação de leitura da memória
3.  $x \leftarrow \text{RDM}$       copia o conteúdo de RDM para  $x$

- REM é o **registrador de endereços da memória**
- RDM é o **registrador de dados da memória**

## ► Arquitetura/Organização

### Operações com a memória

$\text{MEM}(y) \leftarrow x$  descreve uma **escrita** da memória, que é realizada pelos seguintes passos:

1.  $\text{REM} \leftarrow y$       copia  $y$  (que é um endereço) para o REM
2.  $\text{RDM} \leftarrow x$       copia  $x$  (que é um dado) para o RDM
3. write      ativação de uma operação de **escrita** na memória

## ► Arquitetura/Organização

### Operações com a memória

#### Observações (1)

- Após a leitura do PC, seu conteúdo deve ser incrementado, para apontar para a próxima posição
- O incremento do PC pode ser feito a qualquer instante após a transferência do PC para o REM
- O incremento do PC pode ser feito em paralelo com outras operações

## ► Arquitetura/Organização

### Operações com a memória

#### Observações (2)

- Um desvio condicional que não se realiza não necessita ler o valor do endereço de desvio
- Ou seja, basta incrementar o PC



## ► **Arquitetura/Organização**

**Então, detalhando mais as transferências entre registradores...**

# O Computador Neander

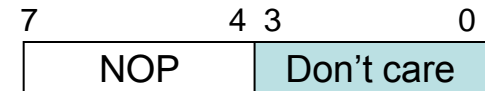
► **Arquitetura/Organização:** transferências entre regs.

## Instrução NOP (nenhuma operação)

**Simbólico:** NOP

**RT:** -

**Passos no nível RT:**



<b>Busca:</b>	REM $\leftarrow$ PC Read; PC $\leftarrow$ PC + 1 RI $\leftarrow$ RDM
<b>Execução:</b>	nenhuma operação

# O Computador Neander

## ► Arquitetura/Organização: transferências entre regs.

### Instrução STA (armazena acumulador)

**Simbólico:** STA end

**RT:**  $\text{MEM}(\text{end}) \leftarrow \text{AC}$

**Passos no nível RT:**

7	4	3	0
STA			Don't care
end			

**Busca:**

$\text{REM} \leftarrow \text{PC}$

Read;  $\text{PC} \leftarrow \text{PC} + 1$

$\text{RI} \leftarrow \text{RDM}$

**Execução:**

$\text{REM} \leftarrow \text{PC}$

Read;  $\text{PC} \leftarrow \text{PC} + 1$

$\text{REM} \leftarrow \text{RDM}$

$\text{RDM} \leftarrow \text{AC}$

Write

# O Computador Neander

## ► Arquitetura/Organização: transferências entre regs.

### Instrução LDA (carrega acumulador)

**Simbólico:** LDA end

**RT:**  $AC \leftarrow \text{MEM}(\text{end})$

**Passos no nível RT:**

7	4	3	0
LDA			Don't care
end			

**Busca:**

$REM \leftarrow PC$

Read;  $PC \leftarrow PC + 1$

$RI \leftarrow RDM$

**Execução:**

$REM \leftarrow PC$

Read;  $PC \leftarrow PC + 1$

$REM \leftarrow RDM$

Read

$AC \leftarrow RDM$ ; atualiza N e Z

# O Computador Neander

## ► Arquitetura/Organização: transferências entre regs.

### Instrução ADD (soma)

**Simbólico:** ADD end

**RT:**  $AC \leftarrow \text{MEM}(\text{end}) + AC$

**Passos no nível RT:**

7	4	3	0
ADD			Don't care
end			

**Busca:**

- REM  $\leftarrow$  PC
- Read; PC  $\leftarrow$  PC + 1
- RI  $\leftarrow$  RDM

**Execução:**

- REM  $\leftarrow$  PC
- Read; PC  $\leftarrow$  PC + 1
- REM  $\leftarrow$  RDM
- Read
- AC  $\leftarrow$  AC + RDM; atualiza N e Z

# O Computador Neander

## ► Arquitetura/Organização: transferências entre regs.

### Instrução OR (“ou” lógico, bit a bit)

**Simbólico:** OR end

**RT:**  $AC \leftarrow \text{MEM}(\text{end}) \text{ OR } AC$

**Passos no nível RT:**

7	4	3	0
OR			Don't care
end			

**Busca:**

- $REM \leftarrow PC$
- Read;  $PC \leftarrow PC + 1$
- $RI \leftarrow RDM$

**Execução:**

- $REM \leftarrow PC$
- Read;  $PC \leftarrow PC + 1$
- $REM \leftarrow RDM$
- Read
- $AC \leftarrow AC \text{ OR } RDM$ ; atualiza N e Z

# O Computador Neander

## ► Arquitetura/Organização: transferências entre regs.

### Instrução AND (“e” lógico, bit a bit)

**Simbólico:** AND end

**RT:**  $AC \leftarrow \text{MEM}(\text{end}) \text{ AND } AC$

**Passos no nível RT:**

7	4	3	0
AND			Don't care
end			

**Busca:**

- $REM \leftarrow PC$
- Read;  $PC \leftarrow PC + 1$
- $RI \leftarrow RDM$

**Execução:**

- $REM \leftarrow PC$
- Read;  $PC \leftarrow PC + 1$
- $REM \leftarrow RDM$
- Read
- $AC \leftarrow AC \text{ AND } RDM$ ; atualiza N e Z

# O Computador Neander

► **Arquitetura/Organização:** transferências entre regs.

## Instrução NOT (complementa acumulador)

**Simbólico:** NOT

**RT:**  $AC \leftarrow \text{NOT } AC$

**Passos no nível RT:**

7	4	3	0
NOT			Don't care

**Busca:**  $REM \leftarrow PC$   
Read;  $PC \leftarrow PC + 1$   
 $RI \leftarrow RDM$

**Execução:**  $AC \leftarrow \text{NOT}(AC)$ ; atualiza N e Z



# O Computador Neander

► **Arquitetura/Organização:** transferências entre regs.

## Instrução JMP (desvio incondicional - jump)

**Simbólico:** JMP end

**RT:** PC  $\leftarrow$  end

**Passos no nível RT:**

7	4	3	0
JMP			Don't care
end			

**Busca:**

REM  $\leftarrow$  PC

Read; PC  $\leftarrow$  PC + 1

RI  $\leftarrow$  RDM

**Execução:**

REM  $\leftarrow$  PC

Read

PC  $\leftarrow$  RDM

# O Computador Neander

## ► Arquitetura/Organização: transferências entre regs.

### Instrução JN (desvio condicional - jump on negative)

**Simbólico:** JN end

**RT:** IF N = 1 THEN PC  $\leftarrow$  end

**Passos no nível RT:**

7	4	3	0
JN			Don't care
end			

#### Se N=1 (desvio ocorre)

**Busca:** REM  $\leftarrow$  PC  
Read; PC  $\leftarrow$  PC + 1  
RI  $\leftarrow$  RDM

**Execução:** REM  $\leftarrow$  PC  
Read  
PC  $\leftarrow$  RDM

#### Se N=0 (desvio não ocorre)

**Busca:** REM  $\leftarrow$  PC  
Read; PC  $\leftarrow$  PC + 1  
RI  $\leftarrow$  RDM

**Execução:** PC  $\leftarrow$  PC + 1

# O Computador Neander

## ► Arquitetura/Organização: transferências entre regs.

### Instrução JZ (desvio condicional - jump on zero)

**Simbólico:** JZ end

**RT:** IF Z = 1 THEN PC  $\leftarrow$  end

**Passos no nível RT:**

7	4	3	0
JZ			Don't care
end			

#### Se Z=1 (desvio ocorre)

**Busca:** REM  $\leftarrow$  PC  
Read; PC  $\leftarrow$  PC + 1  
RI  $\leftarrow$  RDM

**Execução:** REM  $\leftarrow$  PC  
Read  
PC  $\leftarrow$  RDM

#### Se Z=0 (desvio não ocorre)

**Busca:** REM  $\leftarrow$  PC  
Read; PC  $\leftarrow$  PC + 1  
RI  $\leftarrow$  RDM

**Execução:** PC  $\leftarrow$  PC + 1

# O Computador Neander

► **Arquitetura/Organização:** transferências entre regs.

**Instrução HLT (término de execução - halt)**

**Simbólico:** HLT

**RT:** --

**Passos no nível RT:**

7	4	3	0
HLT			Don't care

<b>Busca:</b>	REM $\leftarrow$ PC Read; PC $\leftarrow$ PC + 1 RI $\leftarrow$ RDM
<b>Execução:</b>	parar o processamento

# O Computador Neander

## Neander - Parte II: Arquitetura Interna

### ► Arquitetura: conjunto de instruções

código	instrução	comentário
0000	NOP	Nenhuma operação
0001	STA end	$\text{MEM}(\text{end}) \leftarrow \text{AC}$
0010	LDA end	$\text{AC} \leftarrow \text{MEM}(\text{end})$
0011	ADD end	$\text{AC} \leftarrow \text{MEM}(\text{end}) + \text{AC}$
0100	OR end	$\text{AC} \leftarrow \text{MEM}(\text{end}) \text{ OR } \text{AC}$
0101	AND end	$\text{AC} \leftarrow \text{MEM}(\text{end}) \text{ AND } \text{AC}$
0110	NOT	$\text{AC} \leftarrow \text{NOT } \text{AC}$
1000	JMP end	$\text{PC} \leftarrow \text{end}$
1001	JN end	IF N=1 THEN $\text{PC} \leftarrow \text{end}$
1010	JZ end	IF Z=1 THEN $\text{PC} \leftarrow \text{end}$
1111	HLT	pára processamento

# O Computador Neander

## ► Organização: transferências necessárias

Analizando todas as descrições RT, a agrupando pelo registrador destino, tem-se:

$RI \leftarrow RDM$

$RDM \leftarrow AC$

Write

Read

$AC \leftarrow RDM$ ; atualiza N e Z

$AC \leftarrow AC + RDM$ ; atualiza N e Z

$AC \leftarrow AC \text{ OR } RDM$ ; atualiza N e Z

$AC \leftarrow AC \text{ AND } RDM$ ; atualiza N e Z

$AC \leftarrow \text{NOT}(AC)$ ; atualiza N e Z

$PC \leftarrow RDM$

$PC \leftarrow PC + 1$

$REM \leftarrow PC$

$REM \leftarrow RDM$

# O Computador Neander

## ► Organização: registradores

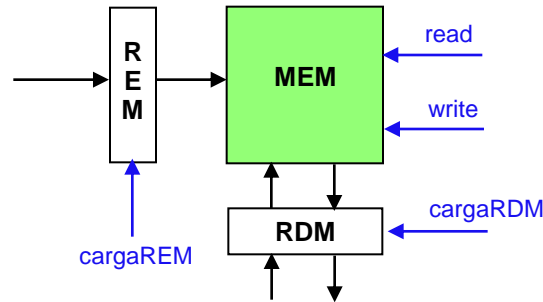
- **AC: um registrador de 8 bits**
- **PC: um registrador de 8 bits (registrador-contador)**
- **RI: um registrador de 4 bits (ou 8)**
- **RDM: um registrador de 8 bits (largura do dado)**
- **REM: um registrador de 8 bits (largura do endereço)**
- **N: um flip-flop para o código de condição N**
- **Z: um flip-flop para o código de condição Z**
- **Uma memória de 256 posições (endereços) x 8 bits**

# O Computador Neander

## ► Organização do Sistema de Memória

### Associados à Memória:

- RDM (dados)
- REM (endereços)
- sinal de escrita (write)
- sinal de leitura (read)



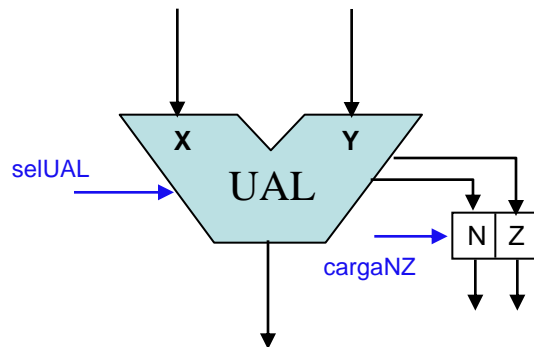
**Cada registrador é controlado por um sinal de carga**



## ► Organização da Unid. Aritmética e Lógica

**Associados à UAL:**

- 4 operações (ADD, AND, OR, NOT)
- sinal de controle (seleção)
- sinais de condição (N,Z)



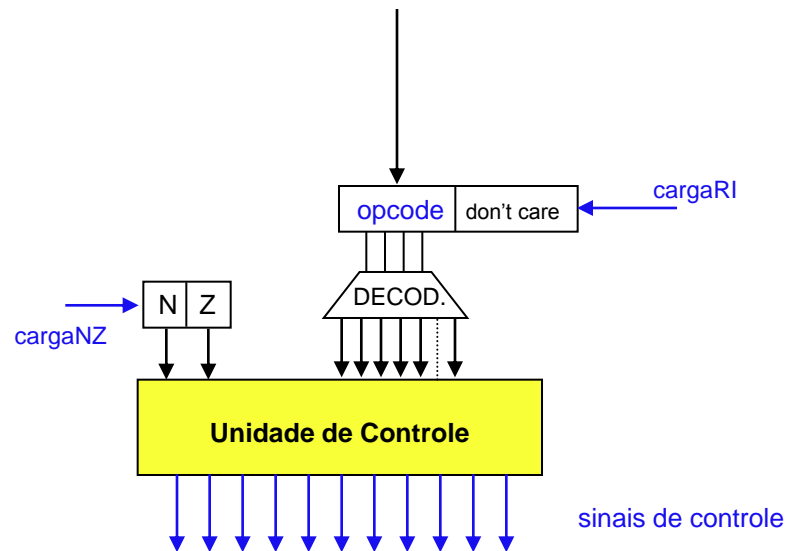
**Flip-Flops devem ter sinal de carga**

# O Computador Neander

## ► Organização do Registrador de Instrução

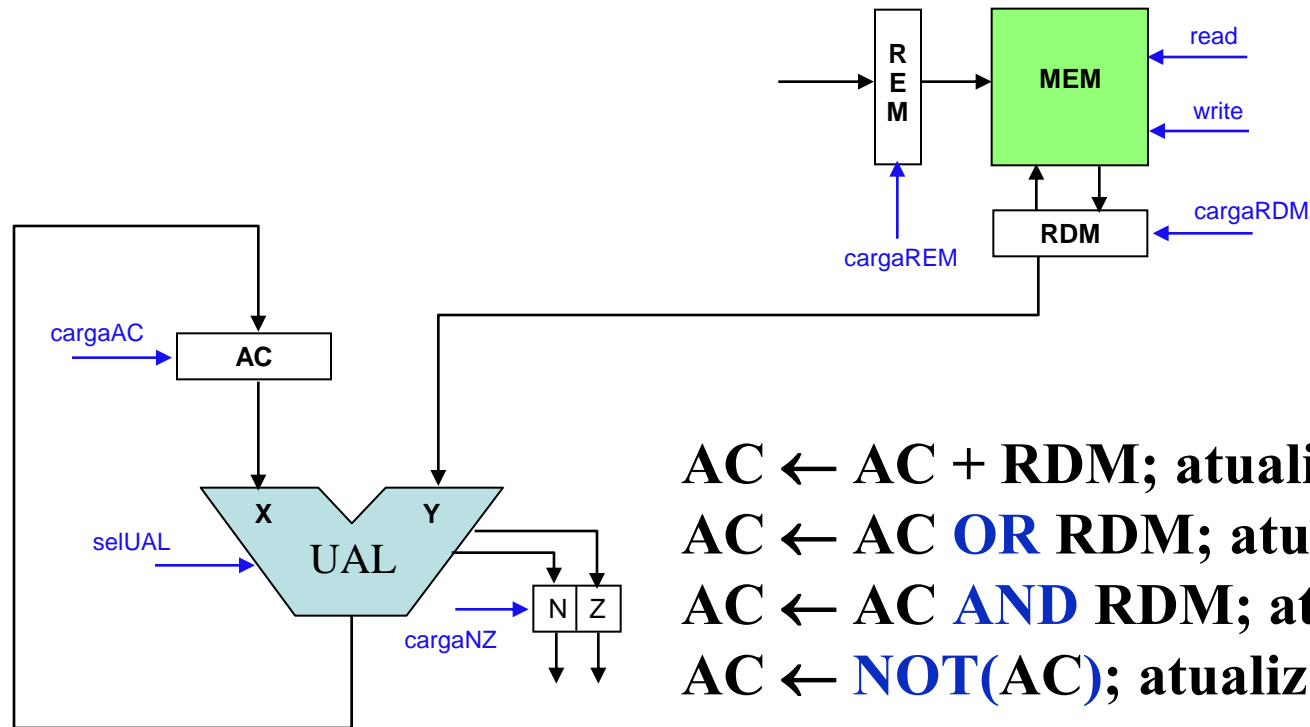
Associados ao Reg. de Instruções (4 ou 8 bits??):

- Decodificador (4 bits para 16 instruções)
- sinais de condição (N,Z) (para JN e JZ)
- registrador deve ter sinal de carga



# O Computador Neander

## ► Operações na UAL



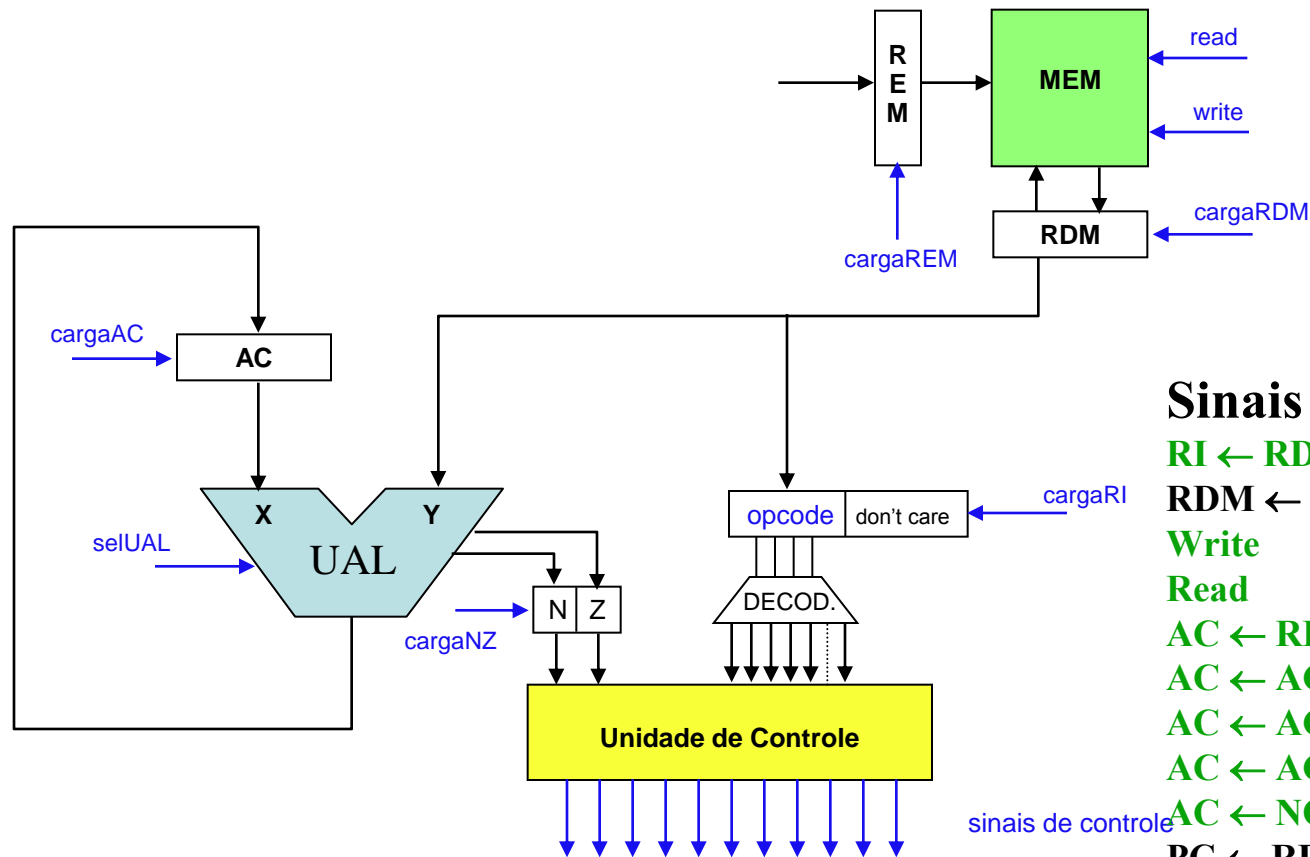
$AC \leftarrow AC + RDM$ ; atualiza N e Z  
 $AC \leftarrow AC \text{ OR } RDM$ ; atualiza N e Z  
 $AC \leftarrow AC \text{ AND } RDM$ ; atualiza N e Z  
 $AC \leftarrow \text{NOT}(AC)$ ; atualiza N e Z

Dúvida:

$AC \leftarrow RDM$ ; atualiza N e Z (via UAL)

# O Computador Neander

## ► Situação até aqui



## Sinais de Controle

**RI**  $\leftarrow$  **RDM**

**RDM**  $\leftarrow$  **AC**

**Write**

**Read**

**AC**  $\leftarrow$  **RDM**; atualiza **N** e **Z**

**AC**  $\leftarrow$  **AC** + **RDM**; atualiza **N** e **Z**

**AC**  $\leftarrow$  **AC** OR **RDM**; at. **N** e **Z**

**AC**  $\leftarrow$  **AC** AND **RDM**; at. **N** e **Z**

**AC**  $\leftarrow$  NOT(**AC**); atualiza **N** e **Z**

**PC**  $\leftarrow$  **RDM**

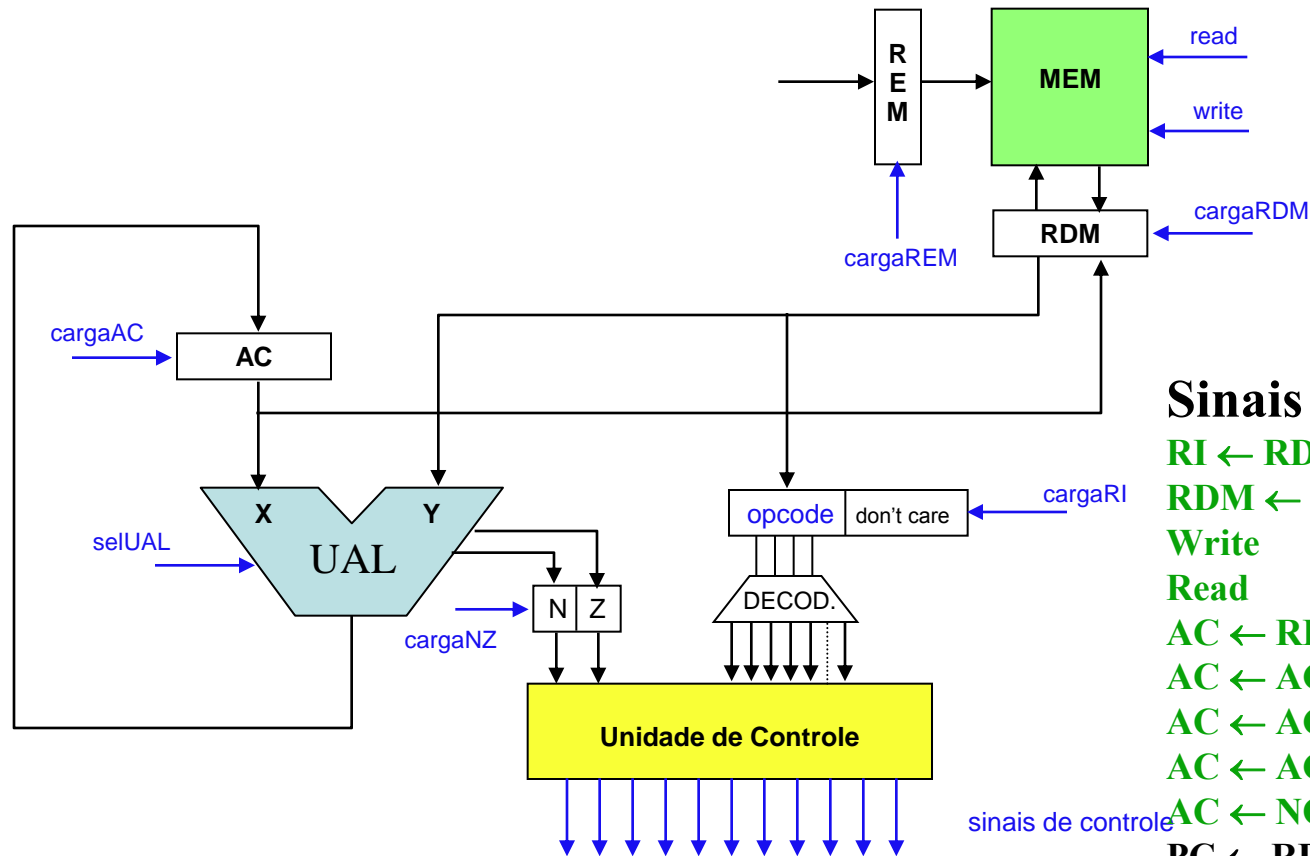
**PC**  $\leftarrow$  **PC** + 1

**REM**  $\leftarrow$  **PC**

**REM**  $\leftarrow$  **RDM**

# O Computador Neander

## ► Acrescentado Escrita do AC



### Sinais de Controle

$RI \leftarrow RDM$

$RDM \leftarrow AC$

Write

Read

$AC \leftarrow RDM$ ; atualiza N e Z

$AC \leftarrow AC + RDM$ ; atualiza N e Z

$AC \leftarrow AC \text{ OR } RDM$ ; at. N e Z

$AC \leftarrow AC \text{ AND } RDM$ ; at. N e Z

$AC \leftarrow \text{NOT}(AC)$ ; atualiza N e Z

$PC \leftarrow RDM$

$PC \leftarrow PC + 1$

$REM \leftarrow PC$

$REM \leftarrow RDM$

## ► **Acréscimo Program Counter (PC)**

**O incremento do PC pode ser feito:**

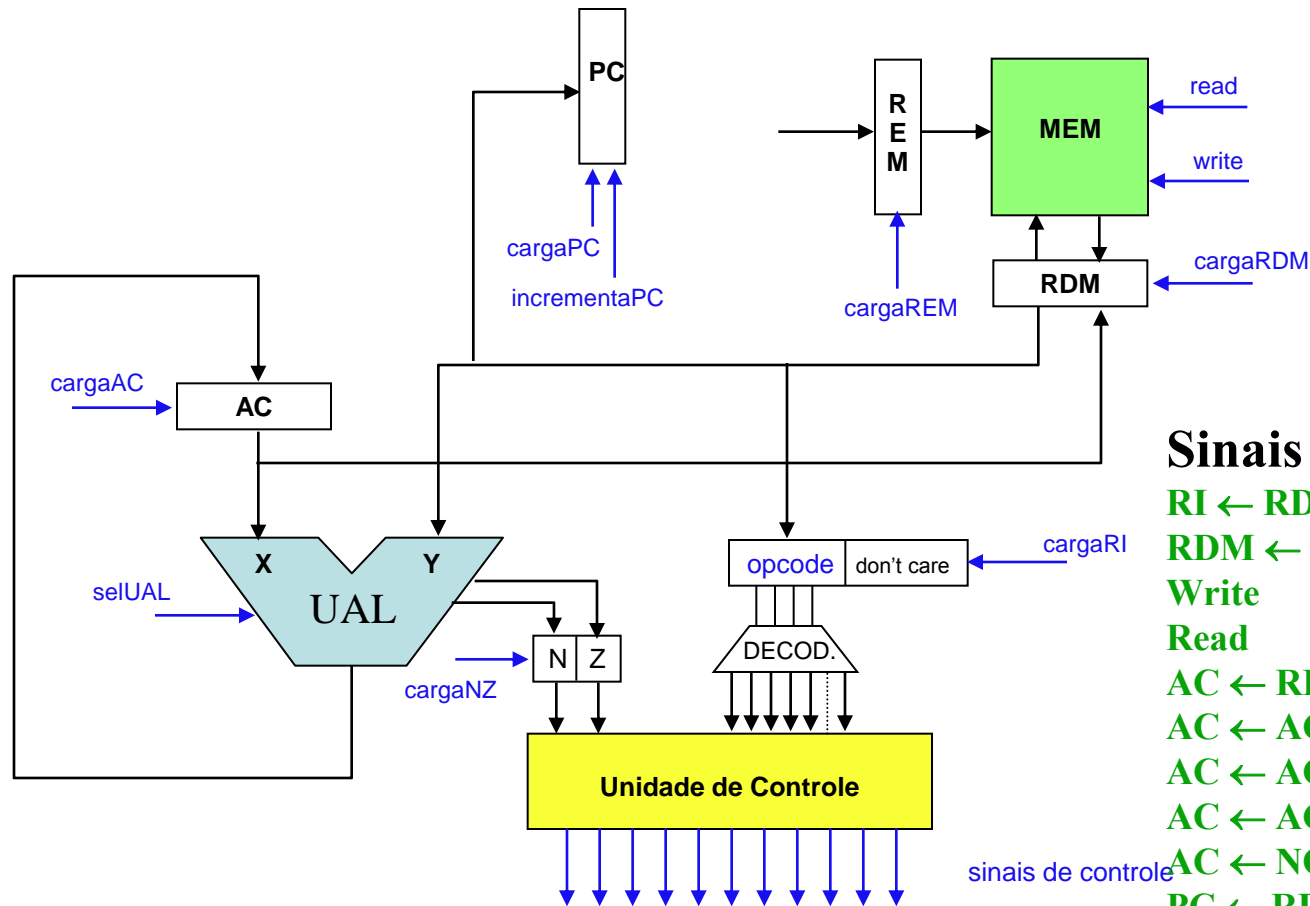
**Por meio de um somador  
dedicado**

**Usando a ULA**

**Por meio de um registrador-  
contador**

# O Computador Neander

## ► Acrescentado Program Counter (PC)



### Sinais de Controle

$RI \leftarrow RDM$

$RDM \leftarrow AC$

Write

Read

$AC \leftarrow RDM$ ; atualiza N e Z

$AC \leftarrow AC + RDM$ ; atualiza N e Z

$AC \leftarrow AC \text{ OR } RDM$ ; at. N e Z

$AC \leftarrow AC \text{ AND } RDM$ ; at. N e Z

$AC \leftarrow \text{NOT}(AC)$ ; atualiza N e Z

$PC \leftarrow RDM$

$PC \leftarrow PC + 1$

$REM \leftarrow PC$

$REM \leftarrow RDM$

# O Computador Neander

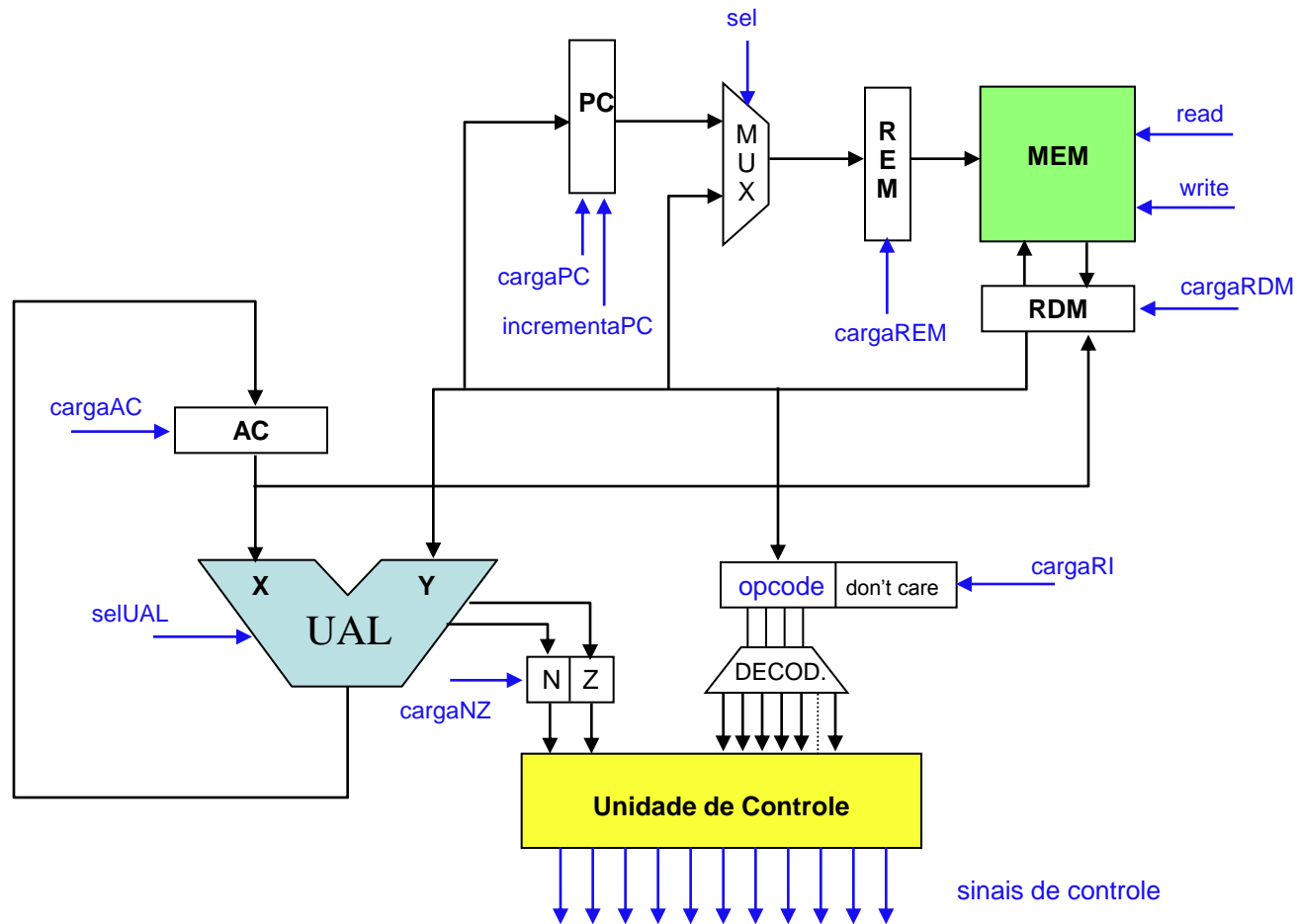
## ► Valores para o REM

- Existem duas transferências para o REM
$$\text{REM} \leftarrow \text{PC}$$
$$\text{REM} \leftarrow \text{RDM}$$
- O único registrador que recebe dados de duas fontes é o REM
- Para solucionar este conflito usa-se um multiplexador



# O Computador Neander

## ► Organização final



# O Computador Neander

## ► A Organização: sinais de controle para cada transferência

Transferência	Sinais de controle
$REM \leftarrow PC$	$sel=0$ , cargaREM
$PC \leftarrow PC + 1$	incrementaPC
$RI \leftarrow RDM$	cargaRI
$REM \leftarrow RDM$	$sel=1$ , cargaREM
$RDM \leftarrow AC$	cargaRDM
$AC \leftarrow RDM$ ; atualiza N e Z	$selUAL(Y)$ , cargaAC, cargaNZ
$AC \leftarrow AC + RDM$ ; atualiza N e Z	$selUAL(ADD)$ , cargaAC, cargaNZ
$AC \leftarrow AC \text{ AND } RDM$ ; atualiza N e Z	$selUAL(AND)$ , cargaAC, cargaNZ
$AC \leftarrow AC \text{ OR } RDM$ ; atualiza N e Z	$selUAL(OR)$ , cargaAC, cargaNZ
$AC \leftarrow \text{NOT}(AC)$ ; atualiza N e Z	$selUAL(NOT)$ , cargaAC, cargaNZ
$PC \leftarrow RDM$	cargaPC

# O Computador Neander

## Temporização dos sinais de controle (parte 1)

tempo	STA	LDA	ADD	OR	AND	NOT
t0	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM
t1	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC
t2	carga RI	carga RI	carga RI	carga RI	carga RI	carga RI
t3	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	UAL(NOT), carga AC, carga NZ, goto t0
t4	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	
t5	sel=1, carga REM	sel=1, carga REM	sel=1, carga REM	sel=1, carga REM	sel=1, carga REM	
t6	carga RDM	Read	Read	Read	Read	
t7	Write, goto t0	UAL(Y), carga AC, carga NZ, goto t0	UAL(ADD), carga AC, carga NZ, goto t0	UAL(OR), carga AC, carga NZ, goto t0	UAL(AND, carga AC, carga NZ, goto t0	

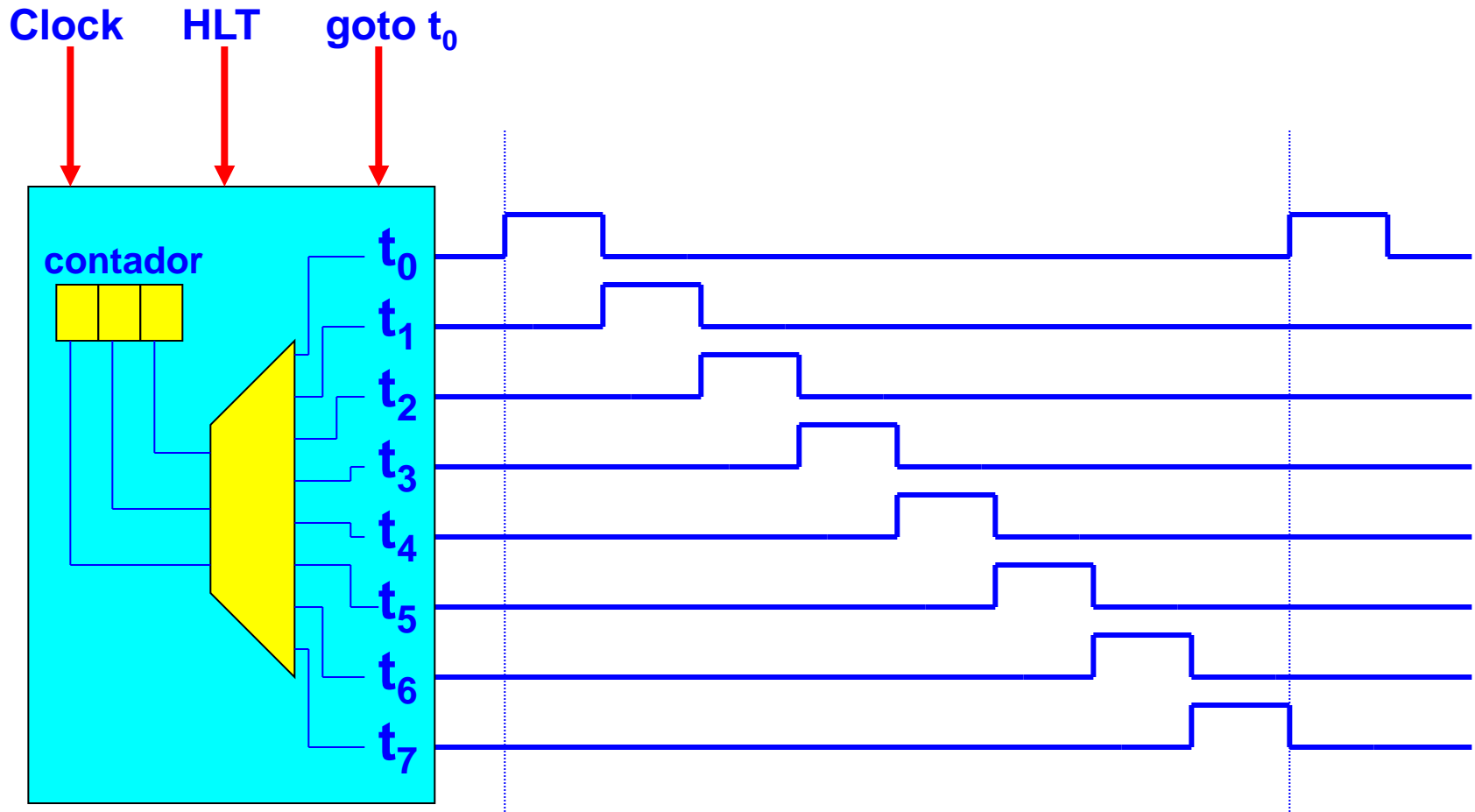
# O Computador Neander

## Temporização dos sinais de controle (parte 2)

tempo	<b>JMP</b>	<b>JN, N=1</b>	<b>JN, N=0</b>	<b>JZ, Z=1</b>	<b>JZ, Z=0</b>	<b>NOP</b>	<b>HLT</b>
t0	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM
t1	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC
t2	carga RI	carga RI	carga RI	carga RI	carga RI	carga RI	carga RI
t3	sel=0, carga REM	sel=0, carga REM	incrementa PC, goto t0	sel=0, carga REM	incrementa PC, goto t0	goto t0	Halt
t4	Read	Read		Read			
t5	carga PC, goto t0	carga PC, goto t0		carga PC, goto t0			
t6							
t7							

# O Computador Neander

## Gerador dos sinais de temporização



# O Computador Neander

## Expressões booleanas dos sinais de controle

**carga REM** =  $t_0 + t_3.(STA+LDA+ADD+OR+AND+JMP+JN.N+JZ.Z) + t_5.(STA+LDA+ADD+OR+AND)$

**incrementa PC** =  $t_1 + t_4.(STA+LDA+ADD+OR+AND) + t_3.(JN.N' + JZ.Z')$

**carga RI** =  $t_2$

**sel** =  $t_5.(STA+LDA+ADD+OR+AND)$

**carga RDM** =  $t_6.STA$

**Read** =  $t_1 + t_4.(STA+LDA+ADD+OR+AND+JMP+JN.N+JZ.Z) + t_6.(LDA+ADD+OR+AND)$

**Write** =  $t_7.STA$

**UAL(Y)** =  $t_7.LDA$

**UAL(ADD)** =  $t_7.ADD$

**UAL(OR)** =  $t_7.OR$

**UAL(AND)** =  $t_7.AND$

**UAL(NOT)** =  $t_3.NOT$

**carga AC** =  $t_7.(LDA+ADD+OR+AND) + t_3.NOT$

**carga NZ** =  $t_7.(LDA+ADD+OR+AND) + t_3.NOT = \text{carga AC}$

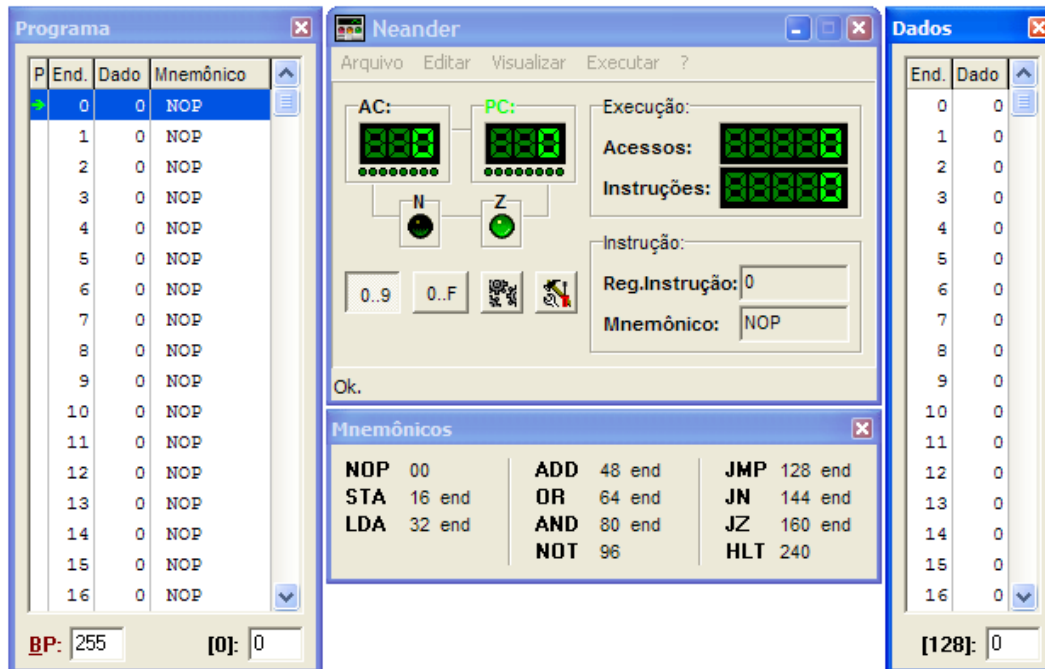
**carga PC** =  $t_5.(JMP+JN.N+JZ.Z)$

**goto t0** =  $t_7.(STA+LDA+ADD+OR+AND) + t_3.(NOP+NOT+JN.N'+JZ.Z') + t_5.(JMP+JN.N+JZ.Z)$

# O Computador Neander

## Próximos passos...

- Implementação de um Simulador
- Programação em Códigos Neander
- Simulação



## E por que não...

- Implementação do Neander em Hardware!
- > Hardware Reconfigurável FPGA :-)