O Computador Neander

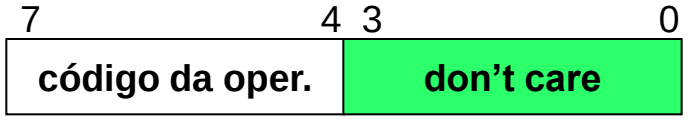
A Arquitetura: conjunto de instruções

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

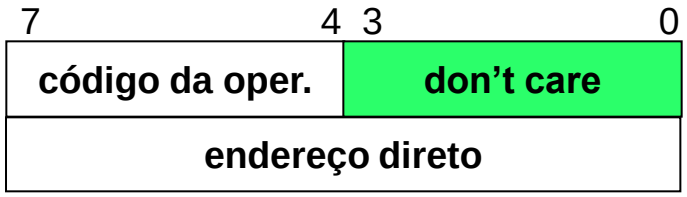
A Arquitetura: formato das instruções

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



A 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)**

O Computador Neander

A 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

- **A Arquitetura: o ciclo de busca (fetch)**

- **A Arquitetura: o ciclo de execução**

O Computador Neander

Arquitetura/Organização: transferências entre regs.

A fase de busca: é igual para todas as instruções

RI ← MEM(PC)

PC ← PC + 1

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

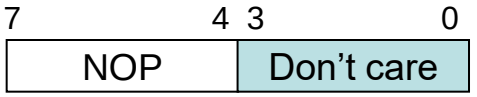
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução NOP (nenhuma operação)

Simbólico: NOP

RT: -



Passos no nível RT:

Busca:	**RI ← MEM(PC)**
	PC ← PC + 1
Execução:	**nenhuma operação**

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

O Computador Neander

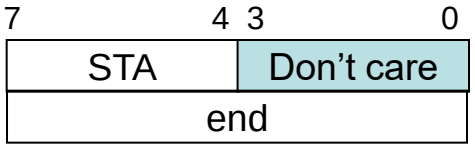
Arquitetura/Organização: transferências entre regs.

Instrução STA (armazena acumulador)

Simbólico: STA end

RT: MEM(end) ← AC

Passos no nível RT:



Busca:	**RI ← MEM(PC)**
	PC ← PC + 1
Execução:	**end ← MEM(PC)**
	PC ← PC + 1
	MEM(end) ← AC

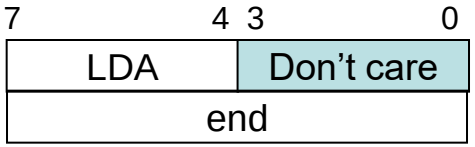
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução LDA (carrega acumulador)

Simbólico: LDA end

RT: AC ← MEM(end)



Passos no nível RT:

Busca:	RI ← MEM(PC)
	PC ← PC + 1
Execução:	end ← MEM(PC)
	PC ← PC + 1
	AC ← MEM(end); atualiza N e Z

O Computador Neander

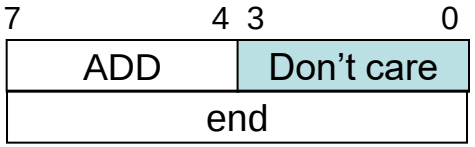
Arquitetura/Organização: transferências entre regs.

Instrução ADD (soma)

Simbólico: ADD end

RT: AC ← MEM(end) + AC

Passos no nível RT:



Busca:	RI ← MEM(PC)
	PC ← PC + 1
Execução:	end ← MEM(PC)
	PC ← PC + 1
	AC ← AC + MEM(end); atualiza N e Z

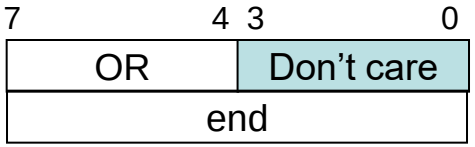
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução OR ("ou" lógico, bit a bit)

Simbólico: OR end

RT: AC ← MEM(end) OR AC



Passos no nível RT:

Busca:	RI ← MEM(PC)
	PC ← PC + 1
Execução:	end ← MEM(PC)
	PC ← PC + 1
	AC ← AC OR MEM(end); atualiza N e Z

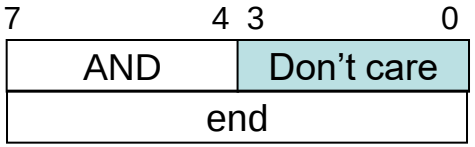
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução AND ("e" lógico, bit a bit)

Simbólico: AND end

RT: AC ← MEM(end) AND AC



Passos no nível RT:

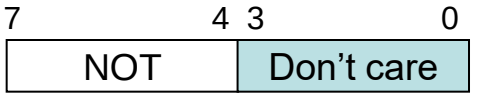
Busca:	RI ← MEM(PC)
	PC ← PC + 1
Execução:	end ← MEM(PC)
	PC ← PC + 1
	AC ← AC AND MEM(end); atualiza N e Z

Arquitetura/Organização: transferências entre regs.

Instrução NOT (complementa acumulador)

Simbólico: NOT

RT: AC ← NOT AC



Passos no nível RT:

Busca:	RI ← MEM(PC)
	PC ← PC + 1
Execução:	AC ← NOT(AC); atualiza N e Z

O Computador Neander

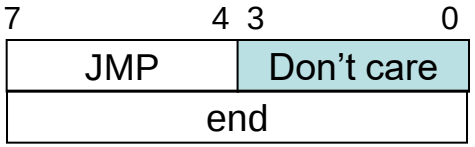
Arquitetura/Organização: transferências entre regs.

Instrução JMP (desvio incondicional - jump)

Simbólico: JMP end

RT: PC ← end

Passos no nível RT:



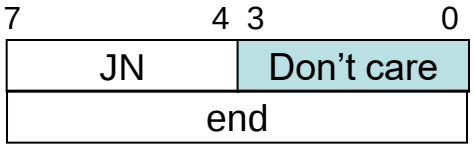
Busca:	**RI ← MEM(PC)**
	PC ← PC + 1
Execução:	**end ← MEM(PC)**
	PC ← end

Arquitetura/Organização: transferências entre regs.

Instrução JN (desvio condicional - jump on negative)

Simbólico: JN end

RT: IF N = 1 THEN PC ← end



Passos no nível RT:

Se N=1 (desvio ocorre)

Busca:	RI ← MEM(PC)
	PC ← PC + 1
Execução:	end ← MEM(PC)
	PC ← end

Se N=0 (desvio não ocorre)

Busca:	RI ← MEM(PC)
	PC ← PC + 1
Execução:	end ← MEM(PC)
	PC ← PC + 1

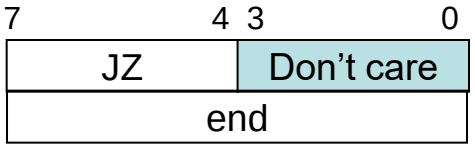
a rigor, desnecessário

Arquitetura/Organização: transferências entre regs.

Instrução JZ (desvio condicional - jump on zero)

Simbólico: JZ end

RT: IF Z = 1 THEN PC ← end



Passos no nível RT:

Se Z=1 (desvio ocorre)

Busca:	RI ← MEM(PC)
	PC ← PC + 1
Execução:	end ← MEM(PC)
	PC ← end

Se Z=0 (desvio não ocorre)

Busca:	RI ← MEM(PC)
	PC ← PC + 1
Execução:	end ← MEM(PC)
	PC ← PC + 1

a rigor, desnecessário

O Computador Neander

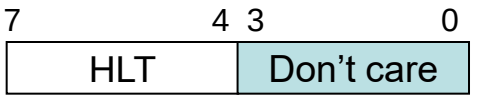
Arquitetura/Organização: transferências entre regs.

Instrução HLT (término de execução - halt)

Simbólico: HLT

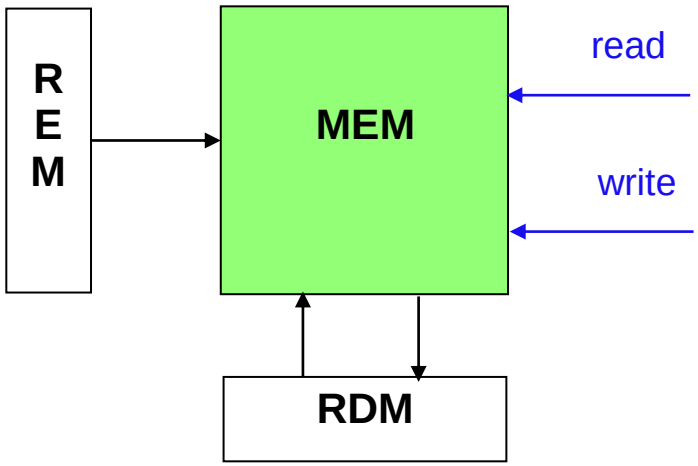
RT: --

Passos no nível RT:



Busca:	**RI ← MEM(PC)**
	PC ← PC + 1
Execução:	**parar o processamento**

Organização do Sistema de Memória



Arquitetura/Organização

Operações com a memória

x ← MEM(y) descreve uma leitura da memória, que é realizada pelos seguintes passos:

1. REM ← y	**copia y (que é um endereço) para o REM**
2. Read	**ativação de uma operação de leitura da memória**
3. x ← 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

MEM(y) ← x descreve uma escrita da memória, que é realizada pelos seguintes passos:

1. REM ← y	**copia y (que é um endereço) para o REM**
2. RDM ← 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**

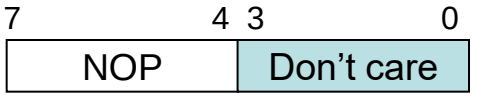
Então, detalhando mais as transferências entre registradores…

Arquitetura/Organização: transferências entre regs.

Instrução NOP (nenhuma operação)

Simbólico: NOP

RT: -



Passos no nível RT:

Busca:	**REM ← PC**
	Read; PC ← PC + 1
	RI ← RDM
Execução:	**nenhuma operação**

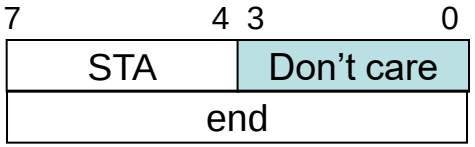
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução STA (armazena acumulador)

Simbólico: STA end

RT: MEM(end) ← AC



Passos no nível RT:

Busca:	REM ← PC
	Read; PC ← PC + 1
	RI ← RDM
Execução:	REM ← PC
	Read; PC ← PC + 1
	REM ← RDM
	RDM ← AC
	Write

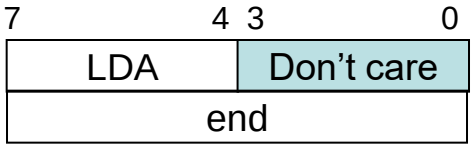
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução LDA (carrega acumulador)

Simbólico: LDA end

RT: AC ← MEM(end)



Passos no nível RT:

Busca:	REM ← PC
	Read; PC ← PC + 1
	RI ← RDM
Execução:	REM ← PC
	Read; PC ← PC + 1
	REM ← RDM
	Read
	AC ← RDM; atualiza N e Z

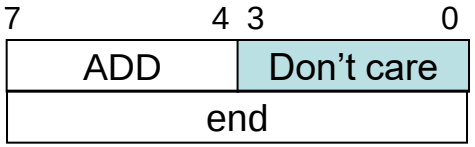
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução ADD (soma)

Simbólico: ADD end

RT: AC ← MEM(end) + AC



Passos no nível RT:

Busca:	REM ← PC
	Read; PC ← PC + 1
	RI ← RDM
Execução:	REM ← PC
	Read; PC ← PC + 1
	REM ← RDM
	Read
	AC ← AC + RDM; atualiza N e Z

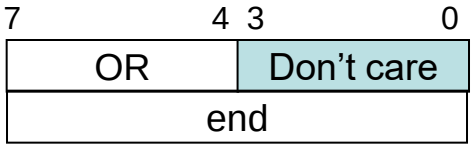
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução OR ("ou" lógico, bit a bit)

Simbólico: OR end

RT: AC ← MEM(end) OR AC



Passos no nível RT:

Busca:	REM ← PC
	Read; PC ← PC + 1
	RI ← RDM
Execução:	REM ← PC
	Read; PC ← PC + 1
	REM ← RDM
	Read
	AC ← AC OR RDM; atualiza N e Z

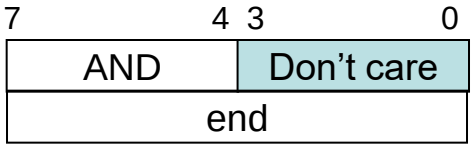
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução AND ("e" lógico, bit a bit)

Simbólico: AND end

RT: AC ← MEM(end) AND AC



Passos no nível RT:

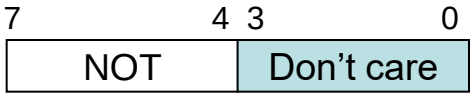
Busca:	REM ← PC
	Read; PC ← PC + 1
	RI ← RDM
Execução:	REM ← PC
	Read; PC ← PC + 1
	REM ← RDM
	Read
	AC ← AC AND RDM; atualiza N e Z

Arquitetura/Organização: transferências entre regs.

Instrução NOT (complementa acumulador)

Simbólico: NOT

RT: AC ← NOT AC



Passos no nível RT:

Busca:	REM ← PC
	Read; PC ← PC + 1
	RI ← RDM
Execução:	AC ← NOT(AC); atualiza N e Z

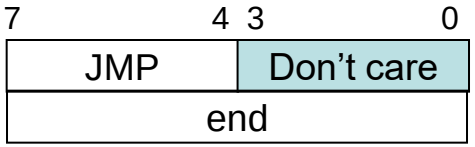
Arquitetura/Organização: transferências entre regs.

Instrução JMP (desvio incondicional - jump)

Simbólico: JMP end

RT: PC ← end

Passos no nível RT:



Busca:	REM ← PC
	Read; PC ← PC + 1
	RI ← RDM
Execução:	REM ← PC
	Read
	PC ← RDM

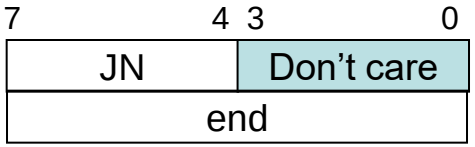
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução JN (desvio condicional - jump on negative)

Simbólico: JN end

RT: IF N = 1 THEN PC ← end



Passos no nível RT:

Se N=1 (desvio ocorre)

Busca:	REM ← PC
	Read; PC ← PC + 1
	RI ← RDM
Execução:	REM ← PC
	Read
	PC ← RDM

Se N=0 (desvio não ocorre)

Busca:	REM ← PC
	Read; PC ← PC + 1
	RI ← RDM
Execução:	PC ← PC + 1

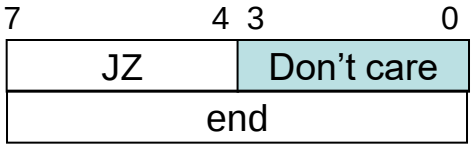
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução JZ (desvio condicional - jump on zero)

Simbólico: JZ end

RT: IF Z = 1 THEN PC ← end



Passos no nível RT:

Se Z=1 (desvio ocorre)

Busca:	REM ← PC
	Read; PC ← PC + 1
	RI ← RDM
Execução:	REM ← PC
	Read
	PC ← RDM

Se Z=0 (desvio não ocorre)

Busca:	REM ← PC
	Read; PC ← PC + 1
	RI ← RDM
Execução:	PC ← PC + 1

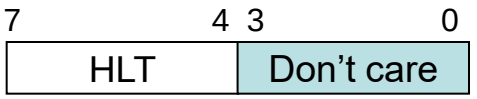
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução HLT (término de execução - halt)

Simbólico: HLT

RT: --



Passos no nível RT:

Busca:	**REM ← PC**
	Read; PC ← PC + 1
	RI ← RDM
Execução:	**parar o processamento**

O Computador Neander

A Arquitetura: conjunto de instruções

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

A Organização: transferências necessárias

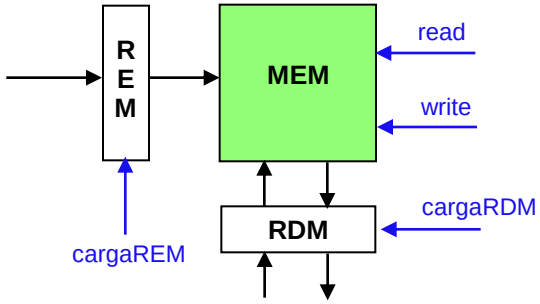
Analisando todas as descrições RT, a agrupando pelo registrador destino, tem-se:

RI ← RDM
RDM ← AC
Write
Read
AC ← RDM; atualiza N e Z
AC ← AC + RDM; atualiza N e Z
AC ← AC OR RDM; atualiza N e Z
AC ← AC AND RDM; atualiza N e Z
AC ← NOT(AC); atualiza N e Z
PC ← RDM
PC ← PC + 1
REM ← PC
REM ← RDM

A 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**

Organização do Sistema de Memória



Associados à Memória:

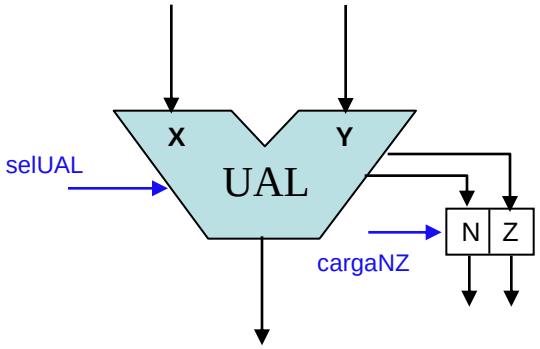
- **RDM (dados)**
- **REM (endereços)**
- **sinal de escrita (write)**
- **sinal de leitura (read)**

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)**

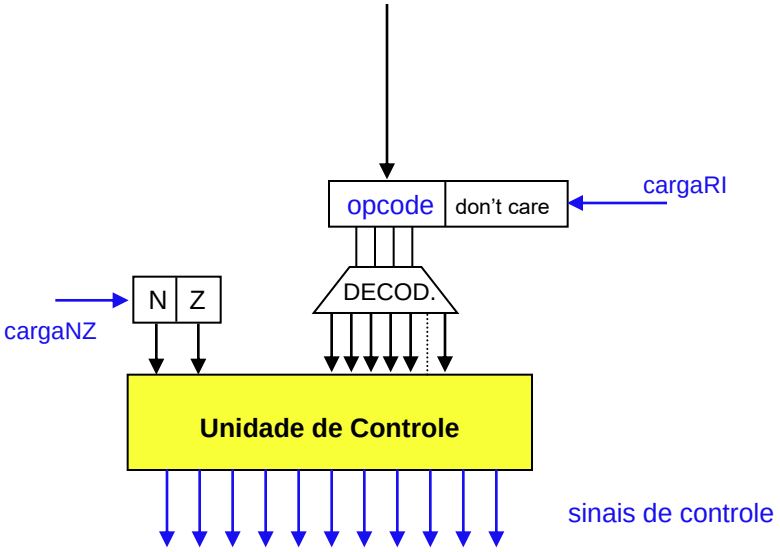


Flip-Flops devem ter sinal de carga

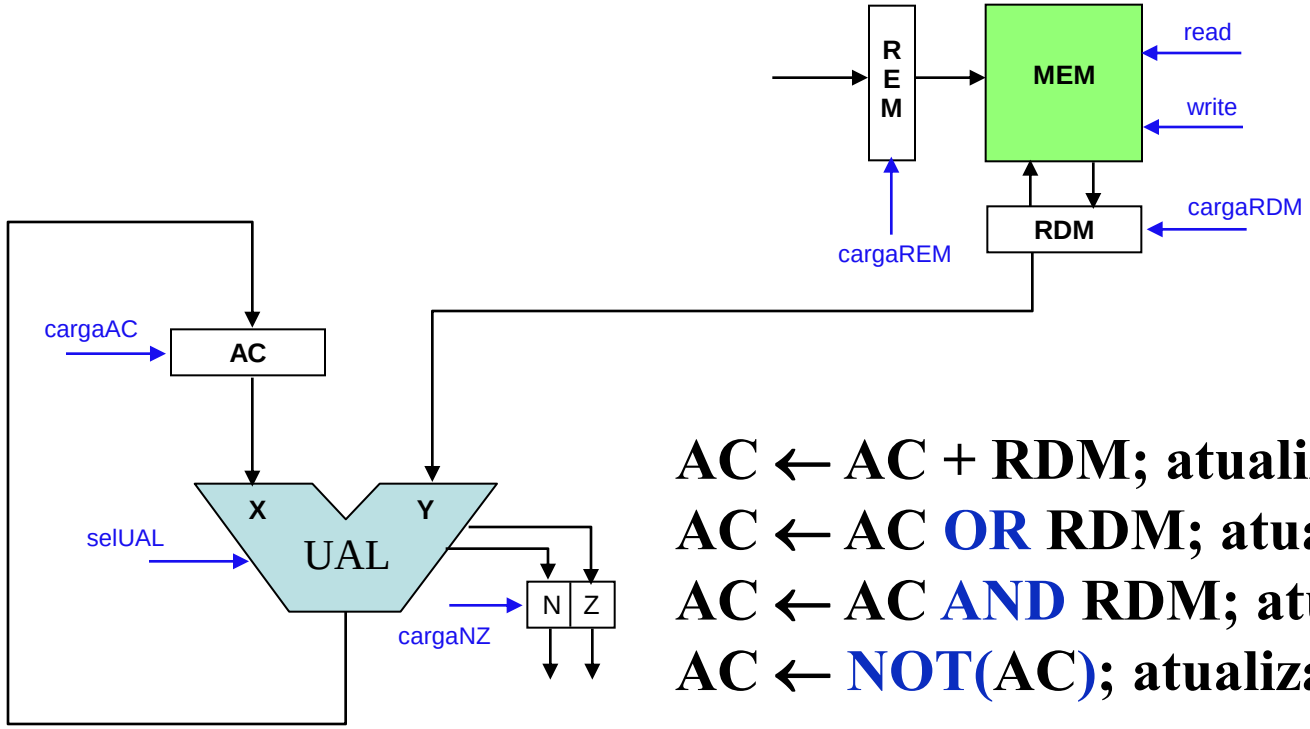
Organização do Registrador de Instrução

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**



Operações na UAL

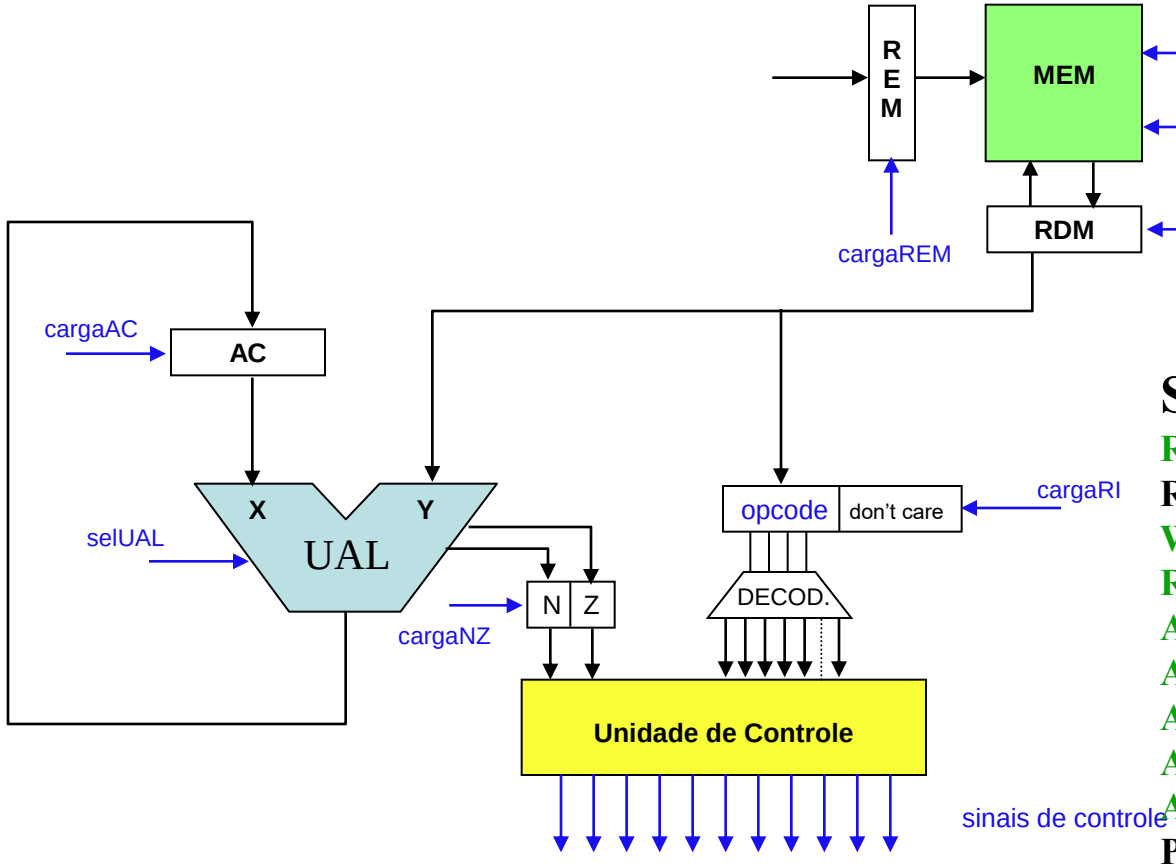


AC ← AC + RDM; atualiza N e Z
AC ← AC OR RDM; atualiza N e Z
AC ← AC AND RDM; atualiza N e Z
AC ← NOT(AC); atualiza N e Z

Dúvida:
AC ← RDM; atualiza N e Z (via UAL)

O Computador Neander

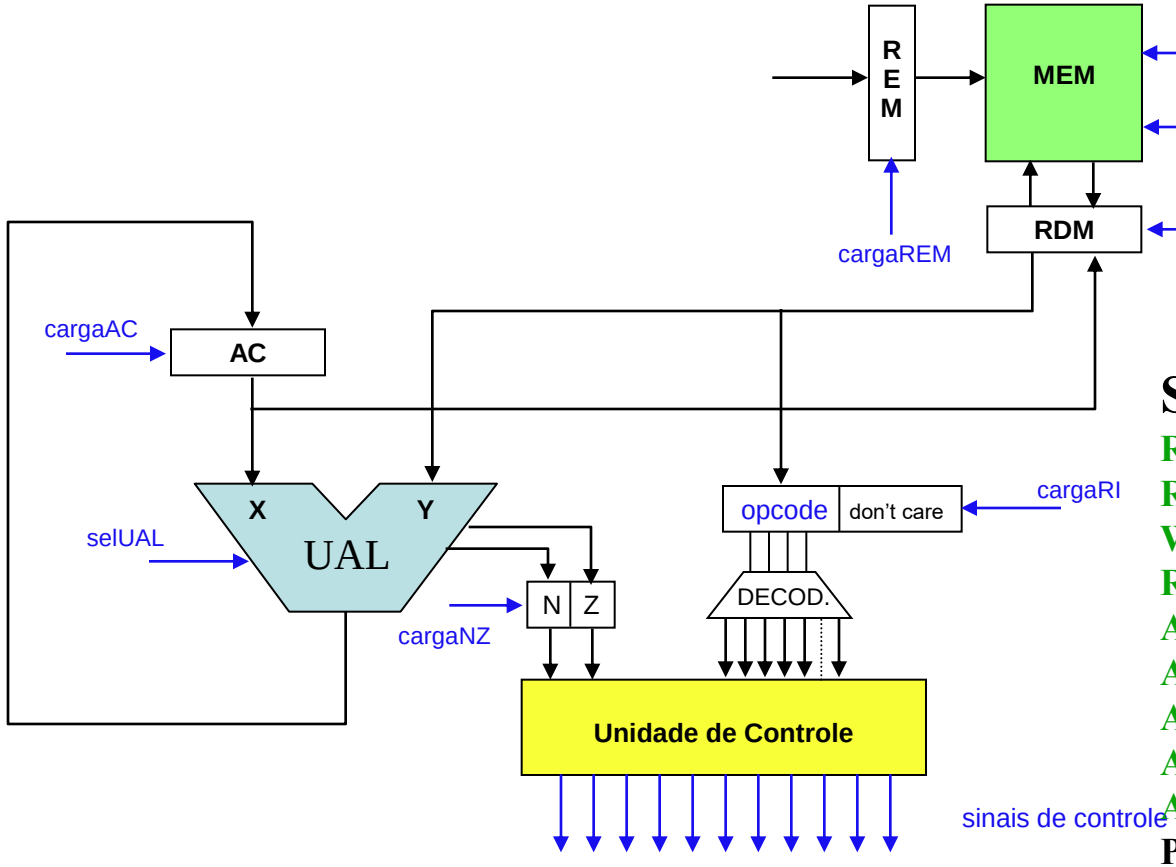
Situação até aqui



Sinais de Controle

RI ← RDM
RDM ← AC
Write
Read
AC ← RDM; atualiza N e Z
AC ← AC + RDM; atualiza N e Z
AC ← AC OR RDM; at. N e Z
AC ← AC AND RDM; at. N e Z
AC ← NOT(AC); atualiza N e Z
PC ← RDM
PC ← PC + 1
REM ← PC
REM ← RDM

Acrescentado Escrita do AC



Sinais de Controle

RI ← RDM
RDM ← AC
Write
Read
AC ← RDM; atualiza N e Z
AC ← AC + RDM; atualiza N e Z
AC ← AC OR RDM; at. N e Z
AC ← AC AND RDM; at. N e Z
AC ← NOT(AC); atualiza N e Z
PC ← RDM
PC ← PC + 1
REM ← PC
REM ← RDM

Acrescentado Program Counter (PC)

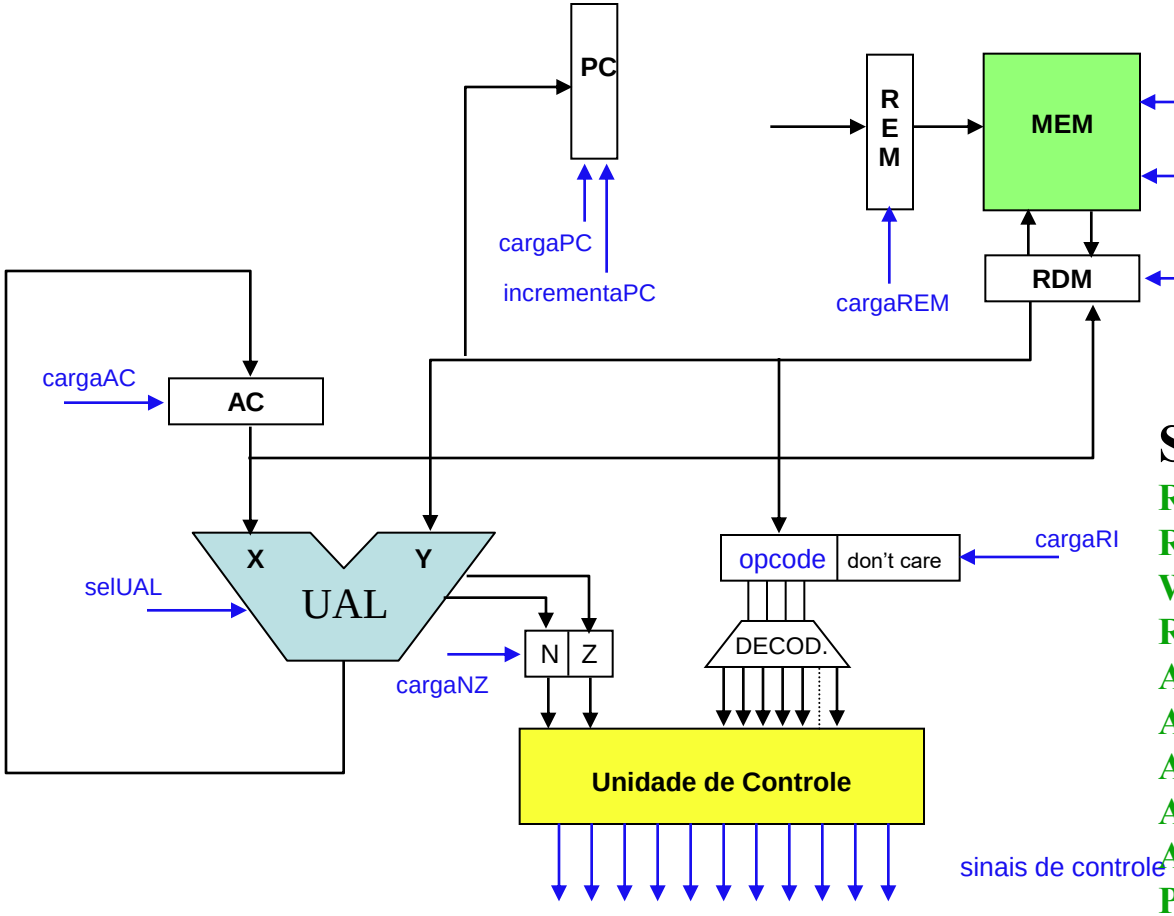
O incremento do PC pode ser feito:

Por meio de um somador dedicado

Usando a ULA

Por meio de um registrador-contador

Acrescentado Program Counter (PC)



Sinais de Controle

RI ← RDM
RDM ← AC
Write
Read
AC ← RDM; atualiza N e Z
AC ← AC + RDM; atualiza N e Z
AC ← AC OR RDM; at. N e Z
AC ← AC AND RDM; at. N e Z
AC ← NOT(AC); atualiza N e Z
PC ← RDM
PC ← PC + 1
REM ← PC
REM ← RDM

Valores para o REM

- **Existem duas transferências para o REM**

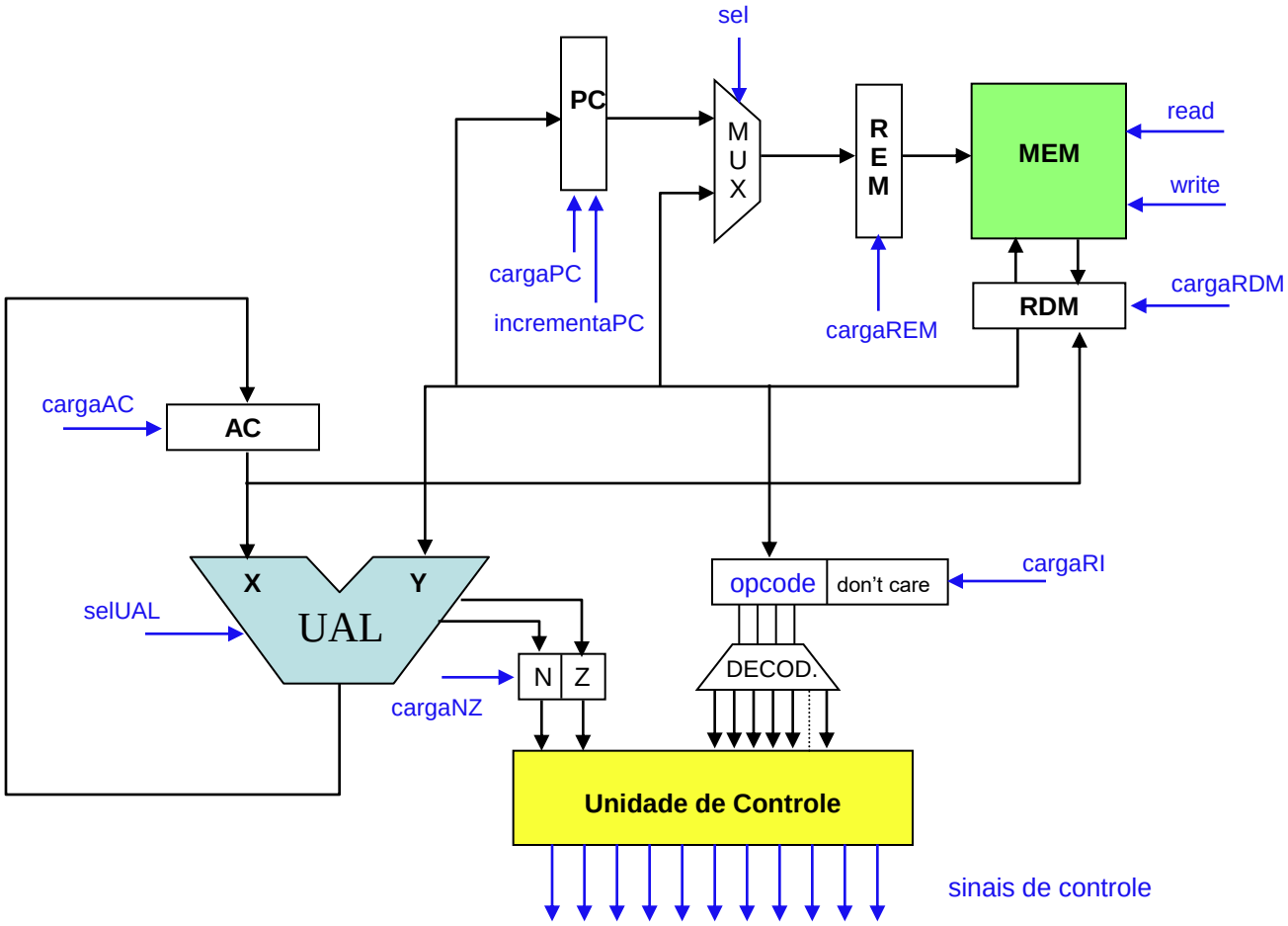
  REM ← PC

  REM ← 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 ← PC	sel=0, cargaREM
PC ← PC + 1	incrementaPC
RI ← RDM	cargaRI
REM ← RDM	sel=1, cargaREM
RDM ← AC	cargaRDM
AC ← RDM; atualiza N e Z	selUAL(Y), cargaAC, cargaNZ
AC ← AC + RDM; atualiza N e Z	selUAL(ADD), cargaAC, cargaNZ
AC ← AC AND RDM; atualiza N e Z	selUAL(AND), cargaAC, cargaNZ
AC ← AC OR RDM; atualiza N e Z	selUAL(OR), cargaAC, cargaNZ
AC ← NOT(AC); atualiza N e Z	selUAL(NOT), cargaAC, cargaNZ
PC ← 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	

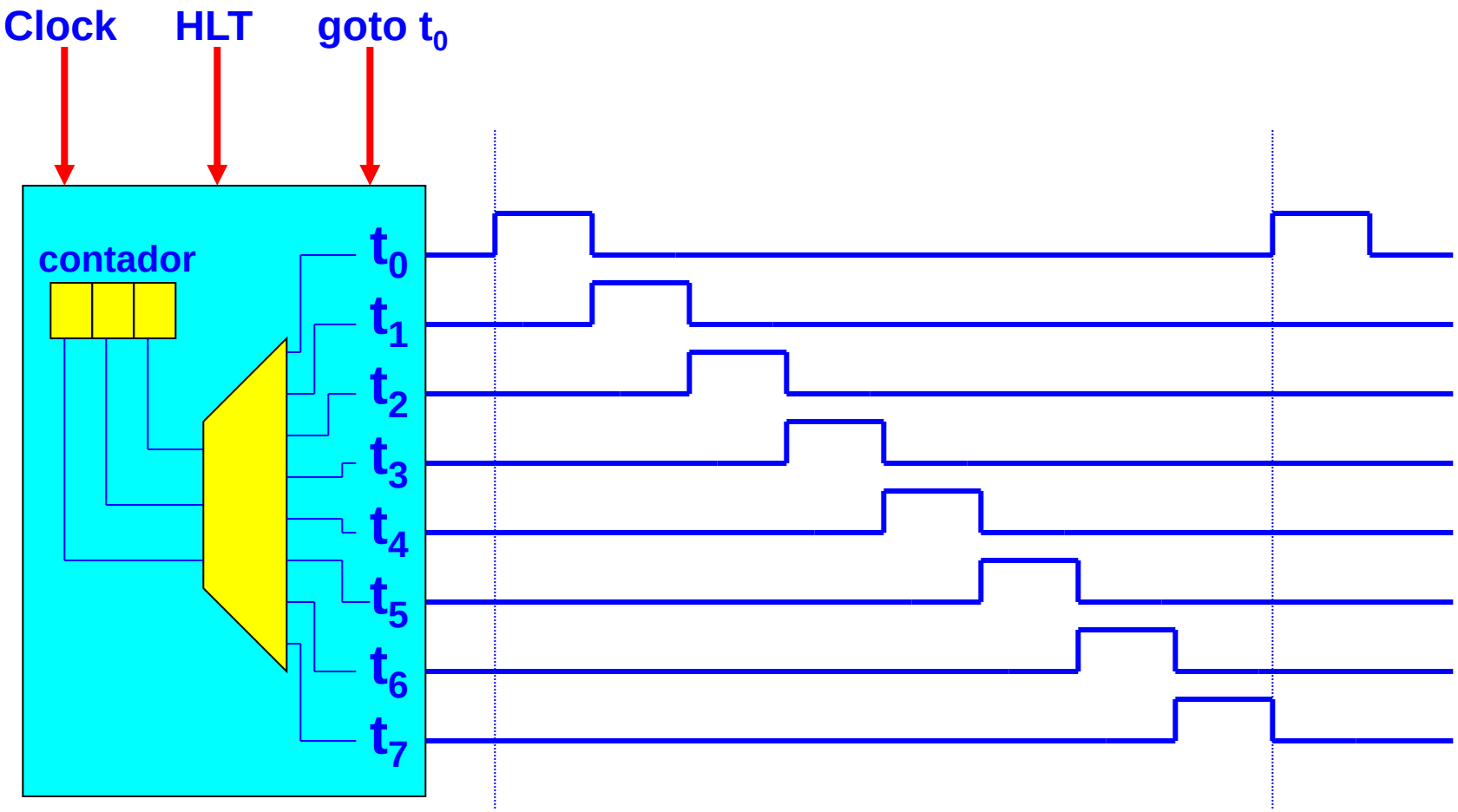
O Computador Neander

Temporização dos sinais de controle (parte 2)

tempo	**JMP**	**JN, N=1**	**JN, N=0**	**JZ, Z=1**	**JZ, Z=0**	**NOP**	**HLT**
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 = t0 + t3.(STA+LDA+ADD+OR+AND+JMP+JN.N+JZ.Z + t5.(STA+LDA+ADD+OR+AND)

incrementa PC = t1 + t4.(STA+LDA+ADD+OR+AND) + t3.(JN.N' + JZ.Z')

carga RI = t2

sel = t5.(STA+LDA+ADD+OR+AND)

carga RDM = t6.STA

Read = t1 + t4.(STA+LDA+ADD+OR+AND+JMP+JN.N+JZ.Z) + t6.(LDA+ADD+OR+AND)

Write = t7.STA

UAL(Y) = t7.LDA

UAL(ADD) = t7.ADD

UAL(OR) = t7.OR

UAL(AND) = t7.AND

UAL(NOT) = t3.NOT

carga AC = t7.(LDA+ADD+OR+AND) + t3.NOT

carga NZ = t7.(LDA+ADD+OR+AND) + t3.NOT = carga AC

carga PC = t5.(JMP+JN.N+JZ.Z)

goto t0 = t7.(STA+LDA+ADD+OR+AND) + t3.(NOP+NOT+JN.N'+JZ.Z') + t5.(JMP+JN.N+JZ.Z)