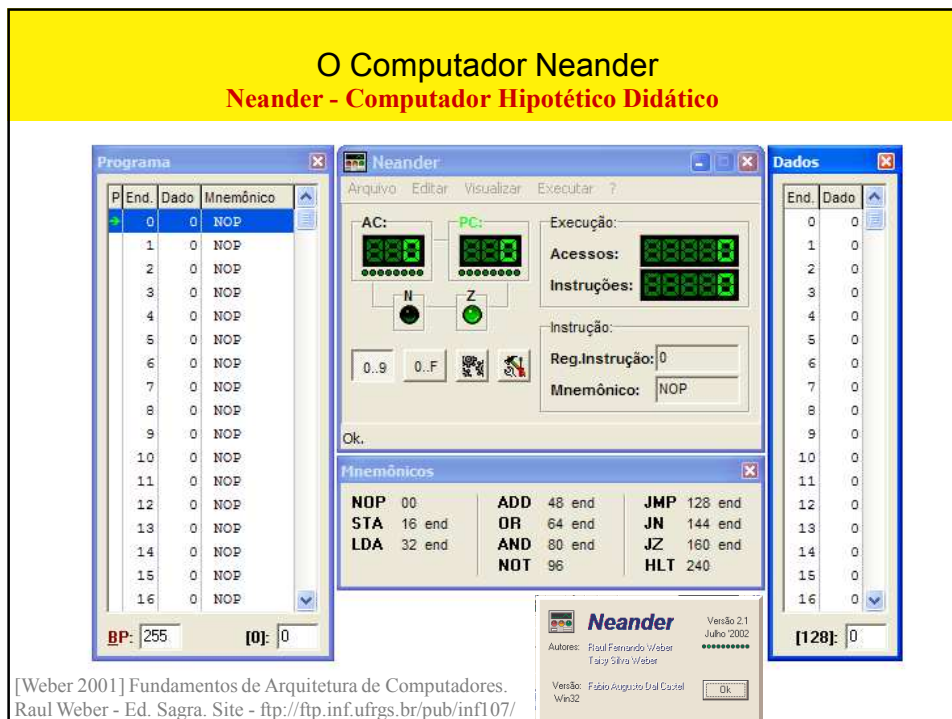


O Computador Neander

Neander - Computador Hipotético Didático



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)

* 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	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

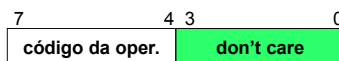
O Computador Neander

▶ Arquitetura: formato das instruções

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

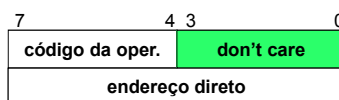
Instruções com um byte:

NOP, NOT



Instruções com dois bytes:

STA, LDA, ADD, OR, AND,
JMP, JN, JZ

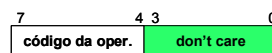


O Computador Neander

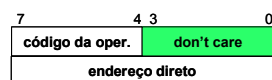
▶ Arquitetura: conjunto de instruções

código	instrução	comentário
0000	NOP	Nenhuma operação
0001	STA end	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

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)



▶ Arquitetura: o ciclo de execução

O Computador Neander

▶ Arquitetura/Organização: transferências entre regs.

A fase de busca: é igual para todas as instruções

$RI \leftarrow MEM(PC)$

$PC \leftarrow PC + 1$

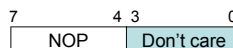
- Novo elemento é necessário: o **registrador de instrução (RI)**
- $MEM(PC)$ corresponde a um acesso à memória, usando o conteúdo do PC como fonte do endereço

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

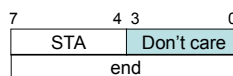
As transferências indicam quais caminhos de dados devem existir, mas não indicam os caminhos físicos reais entre os elementos (registradores e ULA)

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

Passos no nível RT:

7	4 3	0
LDA	Don't care	
end		

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

Execução: $\text{end} \leftarrow \text{MEM}(\text{PC})$
 $PC \leftarrow PC + 1$
 $AC \leftarrow \text{MEM}(\text{end});$ atualiza N e Z

O Computador Neander

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

Instrução ADD (soma)

Simbólico: ADD end

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

Passos no nível RT:

7	4 3	0
ADD	Don't care	
end		

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

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

7	4 3	0
OR	Don't care	
end		

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:

7	4 3	0
AND	Don't care	
end		

Busca: RI ← MEM(PC)
PC ← PC + 1

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

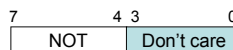
O Computador Neander

► **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

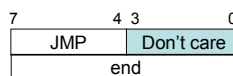
O Computador Neander

► **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

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:

7	4 3	0
JN	Don't care	
end		

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

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:

7	4 3	0
JZ	Don't care	
end		

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

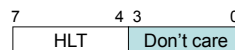
▶ Arquitetura/Organização: transferências entre regs.

Instrução HLT (término de execução - halt)

Simbólico: HLT

RT: --

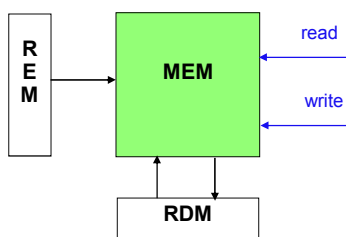
Passos no nível RT:



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

O Computador Neander

▶ Organização do Sistema de Memória



O Computador Neander

▶ **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**

O Computador Neander

▶ **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

O Computador Neander

▶ 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

O Computador Neander

▶ 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

O Computador Neander

▶ Arquitetura/Organização

Então, detalhando mais as transferências entre registradores...

O Computador Neander

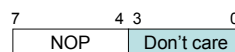
▶ Arquitetura/Organização: transferências entre regs.

Instrução NOP (nenhuma operação)

Simbólico: NOP

RT: -

Passos no nível RT:



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:

7	4 3	0
STA	Don't care	
end		

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:

7	4 3	0
LDA	Don't care	
end		

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:**

7	4 3	0
ADD	Don't care	
end		

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:**

7	4 3	0
OR	Don't care	
end		

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:**

7	4 3	0
AND	Don't care	
end		

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

O Computador Neander

▶ **Arquitetura/Organização:** transferências entre regs.**Instrução NOT (complementa acumulador)****Simbólico:** NOT**RT:** AC ← NOT AC**Passos no nível RT:**

7	4 3	0
NOT	Don't care	

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

7	4 3	0
JMP	Don't care	
end		

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:

7	4 3	0
JN	Don't care	
end		

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:

7	4 3	0
JZ	Don't care	
end		

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:

7	4 3	0
HLT	Don't care	

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

Execução: 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	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

O Computador Neander

► **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

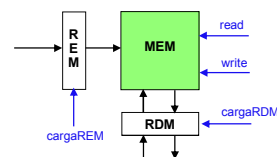
O Computador Neander

► 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)

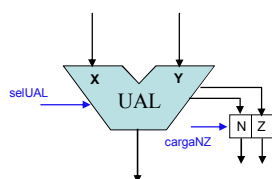
Cada registrador é controlado por um sinal de carga

O Computador Neander

► Organização da Unid. Aritmética e Lógica

Associados à UAL:

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



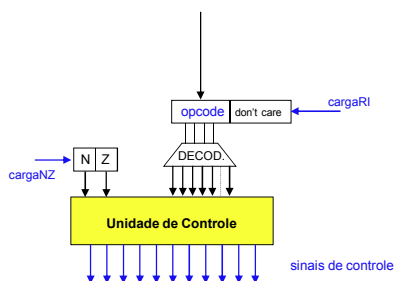
Flip-Flops devem ter sinal de carga

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

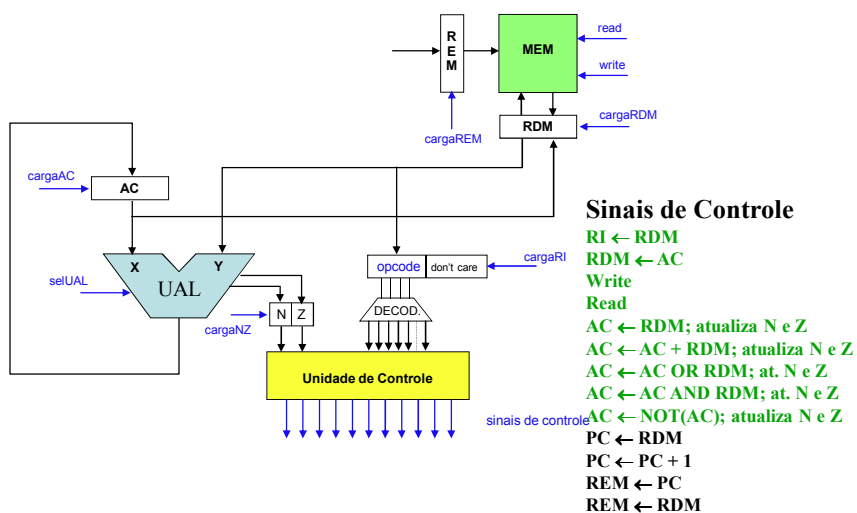
▶ **Situação até aqui**

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



O Computador Neander

► Acrescentado Program Counter (PC)

O incremento do PC pode ser feito:

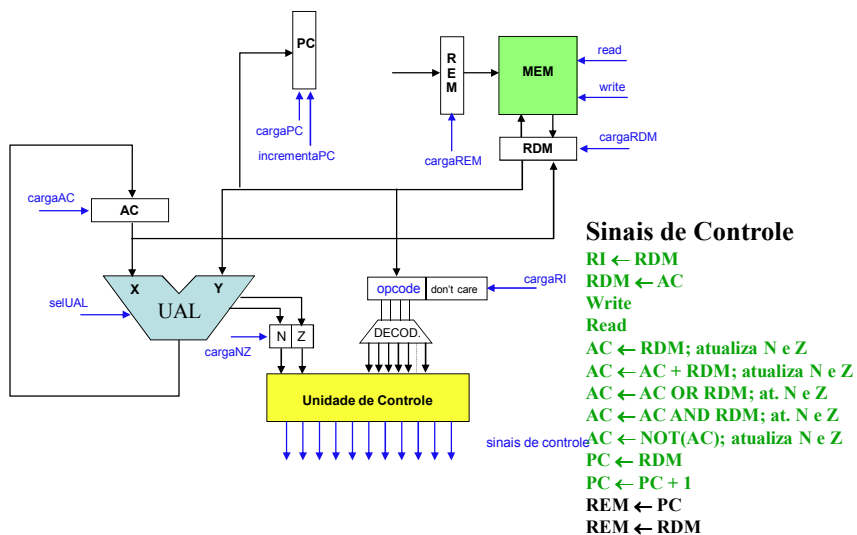
Por meio de um somador dedicado

Usando a ULA

Por meio de um registrador-contador

O Computador Neander

► Acrescentado Program Counter (PC)



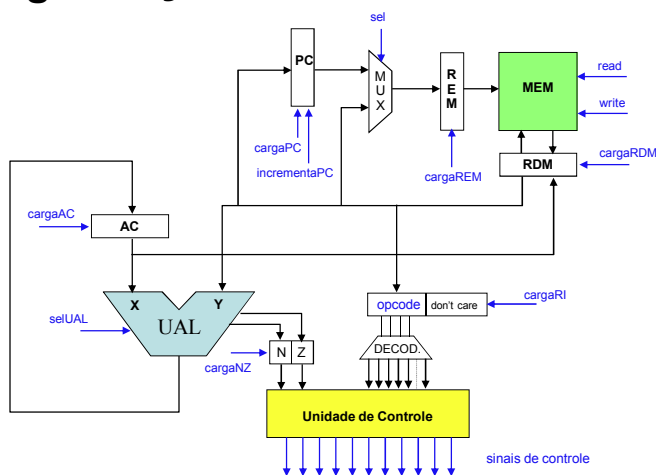
O Computador Neander

► Valores para o REM

- Existem duas transferências para o REM
 - $REM \leftarrow PC$
 - $REM \leftarrow RDM$
- 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	

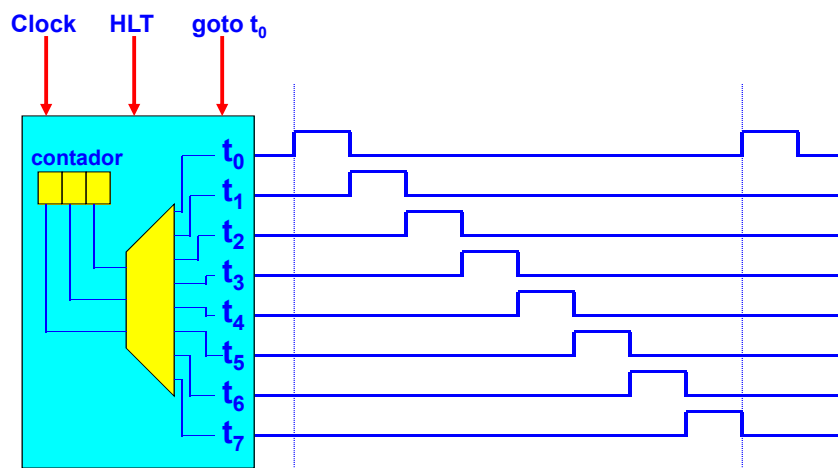
O Computador Neander

Temporização dos sinais de controle (parte 2)

tempo	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 = $t_0 + t_3 \cdot (\text{STA} + \text{LDA} + \text{ADD} + \text{OR} + \text{AND} + \text{JMP} + \text{JN.N} + \text{JZ.Z} + t_5 \cdot (\text{STA} + \text{LDA} + \text{ADD} + \text{OR} + \text{AND}))$

incrementa PC = $t_1 + t_4 \cdot (\text{STA} + \text{LDA} + \text{ADD} + \text{OR} + \text{AND}) + t_3 \cdot (\text{JN.N}' + \text{JZ.Z}')$

carga RI = t_2

sel = $t_5 \cdot (\text{STA} + \text{LDA} + \text{ADD} + \text{OR} + \text{AND})$

carga RDM = $t_6 \cdot \text{STA}$

Read = $t_1 + t_4 \cdot (\text{STA} + \text{LDA} + \text{ADD} + \text{OR} + \text{AND} + \text{JMP} + \text{JN.N} + \text{JZ.Z}) + t_6 \cdot (\text{LDA} + \text{ADD} + \text{OR} + \text{AND})$

Write = $t_7 \cdot \text{STA}$

UAL(Y) = $t_7 \cdot \text{LDA}$

UAL(ADD) = $t_7 \cdot \text{ADD}$

UAL(OR) = $t_7 \cdot \text{OR}$

UAL(AND) = $t_7 \cdot \text{AND}$

UAL(NOT) = $t_3 \cdot \text{NOT}$

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

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

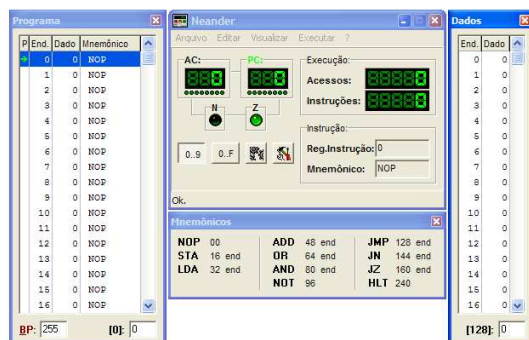
carga PC = $t_5 \cdot (\text{JMP} + \text{JN.N} + \text{JZ.Z})$

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

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 :-)

Material baseado nos slides originais sobre o Neander disponíveis no site - <ftp://ftp.inf.ufg.br/pub/inf107/>