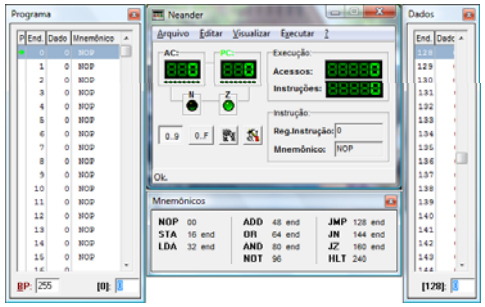
NEANDER



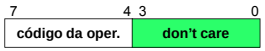
Um processador com

Arquitetura de Von Neumann

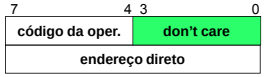
A Arquitetura: formato das instruções

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

Instruções com um byte:
NOP, NOT, HLT



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



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: 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)
- 2 códigos de condição: negativo (N) e zero (Z), implementados como um *registrador de estado* com 2 bits

O Computador Neander

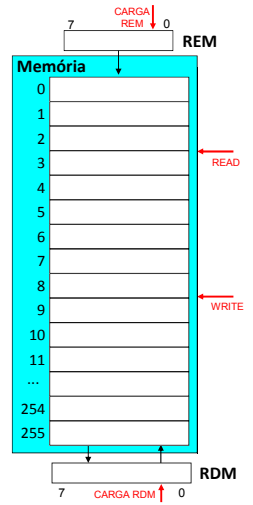
A Organização: elementos necessários

- **Um registrador de 8 bits para servir de acumulador (AC)**
- **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 com 256 posições (palavras) de 8 bits, endereçadas de 0 a 255**

Elementos da organização do NEANDER

1) Memória

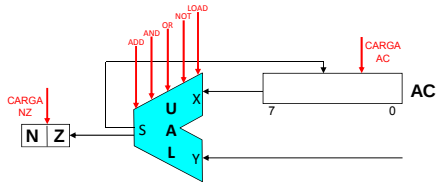
- Memória de 256 palavras de 8 bits, com endereços de 0 a 255.
- Registrador de endereços da memória (REM) de 8 bits, para armazenar endereços de 0 a 255.
- Registrador de dados da memória (RDM) de 8 bits para armazenar o dado lido de ou a ser escrito em uma palavra.
- Sinais de controle:
  - **Read** – ler palavra apontada pelo REM a carregar no RDM
  - **Write** – escrever o conteúdo do RDM na palavra apontada pelo REM
  - **Carga RDM/REM** – carregar no registrador os dados que estão na sua entrada



Elementos da organização do NEANDER

2) Unidade de Aritmética e Lógica (UAL)

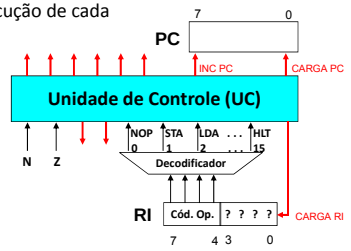
- 1 acumulador de 8 bits (AC) para armazenar resultados das operações.
- 2 registradores de 1 bit para armazenar os códigos de condição (N e Z) após a execução de uma operação pela UAL.
- Circuitos lógicos para executar cada operação da UAL e transferir o resultado da operação selecionada para a saída S
- Sinais de controle:
- **ADD**: S ← X + Y; AC ← S; ajusta N e Z
- **AND**: S ← X AND Y; AC ← S; ajusta N e Z
- **OR**: S ← X OR Y; AC ← S; ajusta N e Z
- **NOT**: S ← NOT X; AC ← S; ajusta N e Z
- **LOAD**: S ← Y; AC ← S; ajusta N e Z
- **Carga AC e NZ** – carregar no registrador os dados que estão na sua entrada



Elementos da organização do NEANDER

3) Unidade de Controle (UC)

- 1 registrador <u>contador</u> de 8 bits (PC) para armazenar o endereço da próxima instrução a ler da memória.
- 1 registrador de 4 (ou 8 ?) bits para armazenar a instrução lida da memória na fase de busca.
- Circuitos lógicos para ativar , no momento certo, os sinais de controle necessários à execução de cada instrução.
- Decodificador para ativar a linha correspondente à instrução que está no RI
- Sinais de controle:
- **INC PC**: incrementar o conteúdo do PC (somar 1)
- **Carga RI/PC** – carregar no registrador os dados que estão na sua entrada
- **Outros** – enviados para comandar os demais componentes do processador



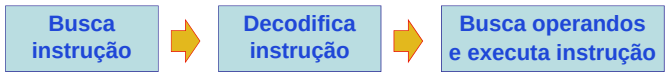
O Computador Neander

- Já vimos como funcionam UAL, UC, memória, registradores e decodificadores
- Mas como integrá-los e sincronizá-los para executar um programa?
- Já sabemos que a parte de controle do processador coordena o *ciclo de instrução*:

  Busca – Decodificação - Execução
- Vamos implementar este ciclo!

O Computador Neander

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



- **A Arquitetura: a fase de busca (fetch)**

O Computador Neander

- **Arquitetura/Organização**

  Transferências entre registradores

A fase de busca: é igual para todas as instruções

RI ← MEM(PC)
PC ← PC + 1

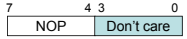
- **Novo elemento é necessário: o registrador de instrução (RI)**
- **MEM(PC) corresponde a um acesso à memória, usando o conteúdo do PC como fonte do endereço**
- **Quais são os barramentos necessários ?**
- **Que sinais de controle são usados ?**

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 UAL)

O Computador Neander

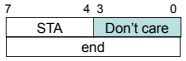
Arquitetura/Organização: transferências entre regs.

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

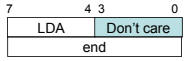
Arquitetura/Organização: transferências entre regs.

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

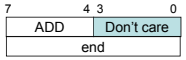
Arquitetura/Organização: transferências entre regs.

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

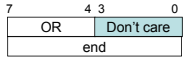
Arquitetura/Organização: transferências entre regs.

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

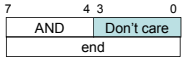
O Computador Neander

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

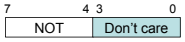
O Computador Neander

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

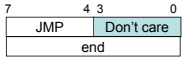
O Computador Neander

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

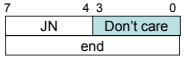
O Computador Neander

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:

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

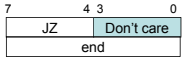
O Computador Neander

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:

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

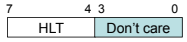
O Computador Neander

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

Instruções que envolvem a memória

Execução STA: end ← MEM(PC)
PC ← PC + 1
MEM(end) ← AC

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

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

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

O Computador Neander

Instruções que envolvem a memória

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

Execução JMP: end ← MEM(PC)
PC ← end

Execução JN: end ← MEM(PC)
PC ← end

Execução JZ: end ← MEM(PC)
PC ← end

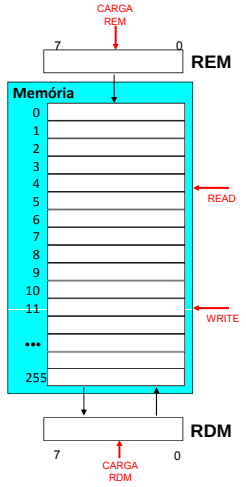
O Computador Neander

Organização do Sistema de Memória

Operações com a memória:

- Leitura (READ)

- Escrita (WRITE)



O Computador Neander

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 — ativa 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

O Computador Neander

Arquitetura/Organização

Operações com a memória

MEM(y) ← x descreve uma **escrita** na 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 — ativa 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

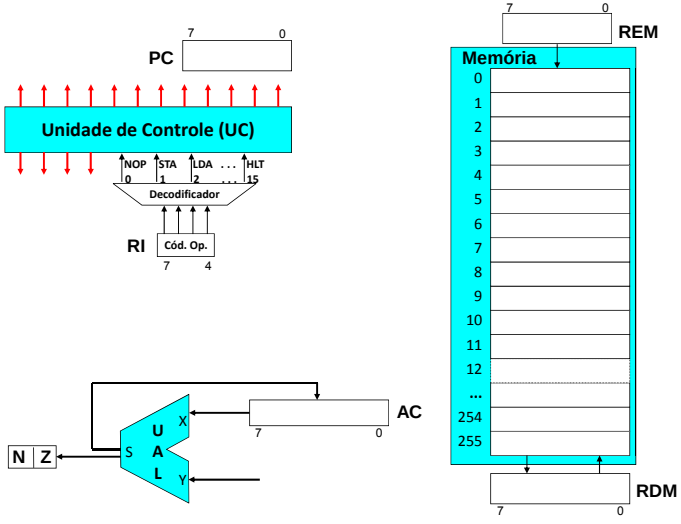
Observações (2)

- **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

- Já sabemos quais são os componentes necessários para implementar o Neander: uma memória, uma UAL, uma Unidade de Controle e barramentos
- Agora vamos detalhar um pouco mais as transferências entre registradores para ver o que falta (componentes e sinais de controle) para completar a organização



O Computador Neander

Arquitetura/Organização

Busca da Instrução

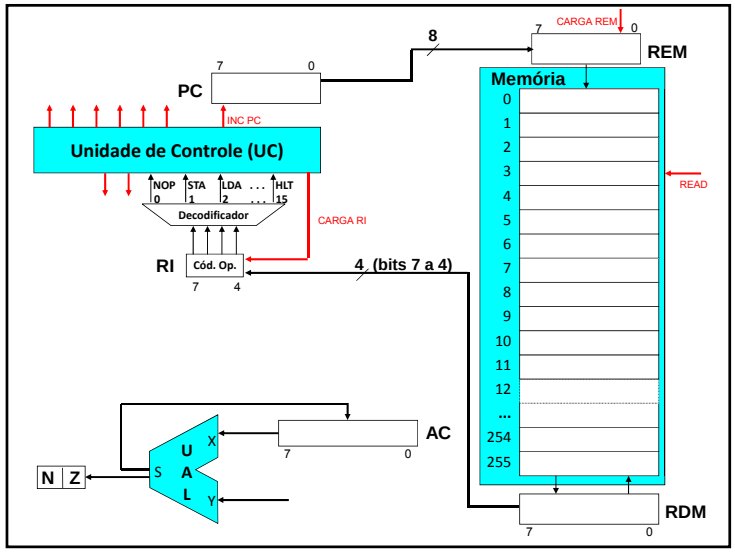
- **É uma operação de leitura da memória.**
- **O endereço a ser lido está no PC.**
- **O valor lido deve ser carregado no RI**
- **Passos no nível RT:**

REM ← PC

Read; PC ← PC + 1

RI ← RDM

Que elementos precisamos acrescentar na organização para fazer a busca ?



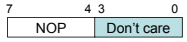
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução NOP (nenhuma operação)

Simbólico: NOP

RT: -

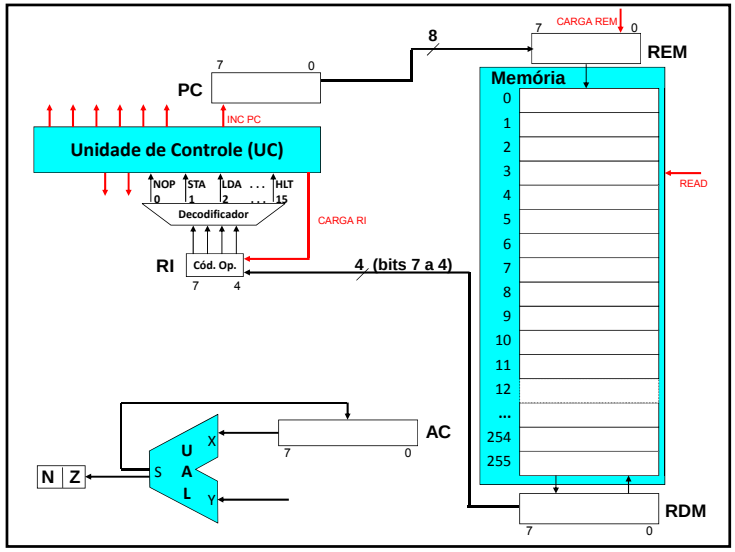


Passos no nível RT:

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

Execução: nenhuma operação

Precisamos acrescentar algum elemento à organização para executar esta instrução ? Qual(is) ?



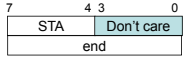
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução STA (armazena acumulador)

Simbólico: STA end

RT: MEM(end) ← AC

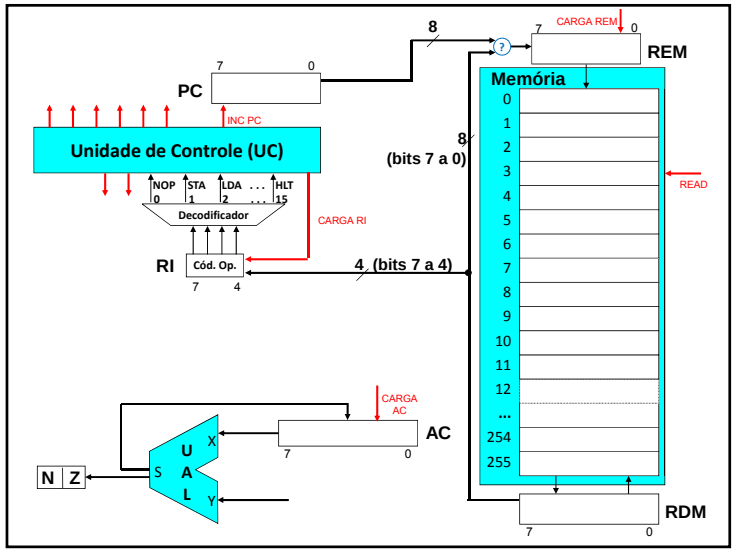


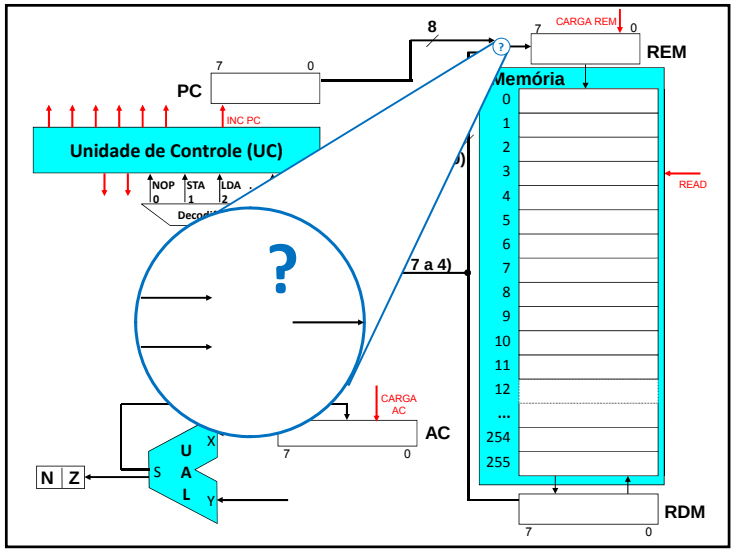
Passos no nível RT:

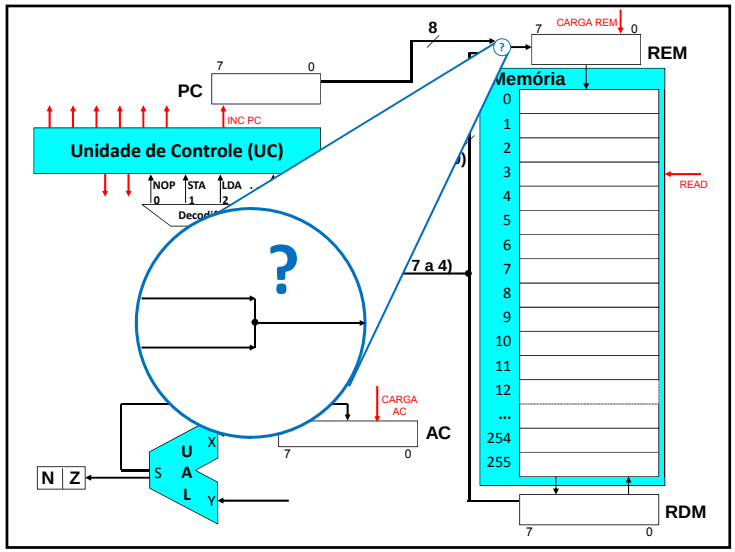
Busca: REM ← PC
Read; PC ← PC + 1
RI ← RDM

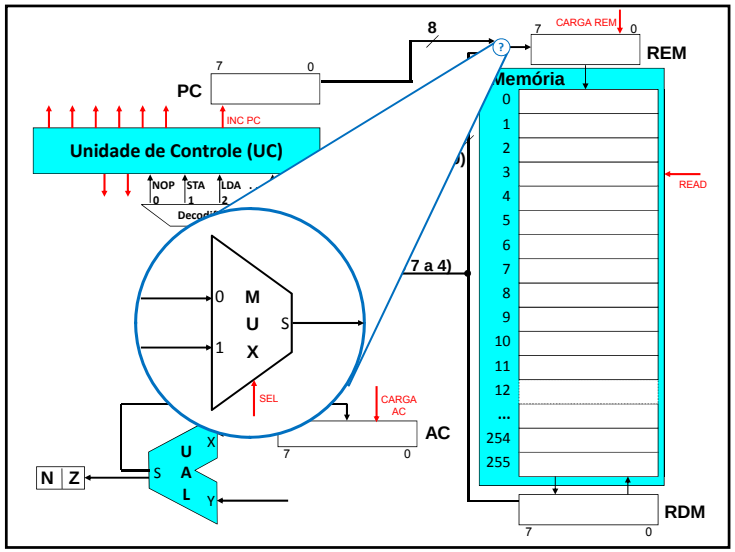
Execução: REM ← PC
Read; PC ← PC + 1
REM ← RDM
RDM ← AC
Write

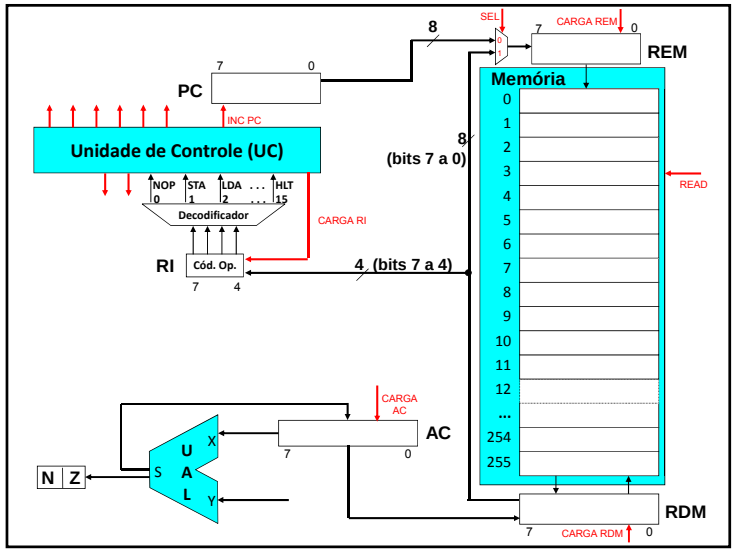
Precisamos acrescentar algum elemento à organização para executar esta instrução ? Qual(is) ?

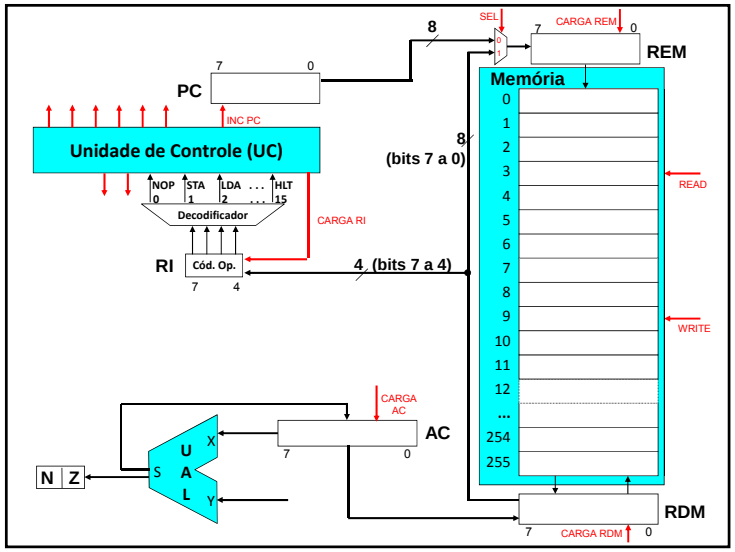

8
CARGA REM
REM
PC
INC PC
Memória
Unidade de Controle (UC)
8
(bits 7 a 0)
READ
NOP
STA
LDA
HLT
Decodificador
CARGA RI
RI
Cód. Op.
4 (bits 7 a 4)
CARGA
AC
AC
U
A
L
N
Z
RDM


8
CARGA REM
REM
PC
INC PC
Memória
Unidade de Controle (UC)
READ
?
7 a 4)
CARGA
AC
AC
U
A
L
N
Z
RDM


8
CARGA REM
REM
PC
INC PC
Memória
Unidade de Controle (UC)
READ
?
7 a 4)
CARGA
AC
AC
U
A
L
N
Z
RDM


8
CARGA REM
REM
PC
INC PC
Memória
Unidade de Controle (UC)
READ
0
M
U
X
S
1
SEL
7 a 4)
CARGA
AC
AC
U
A
L
N
Z
RDM





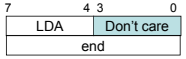
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução LDA (carrega acumulador)

Simbólico: LDA end

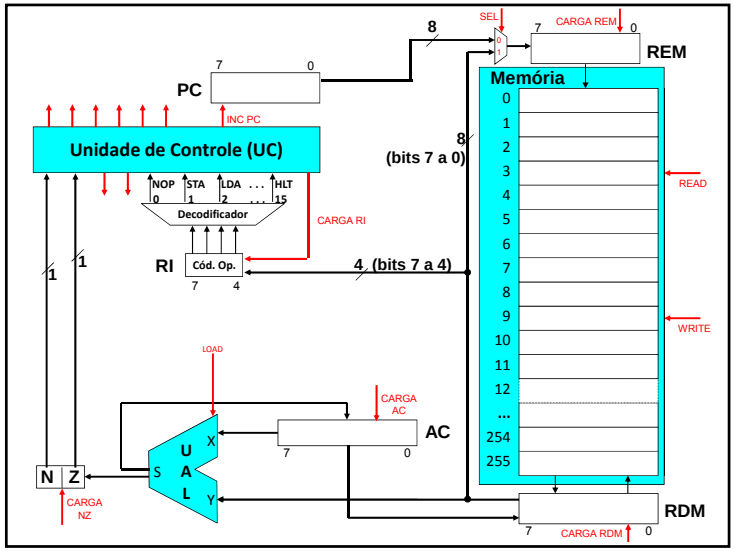
RT: AC ← MEM(end)



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

Precisamos acrescentar algum elemento à organização para executar esta instrução ? Qual(is) ?



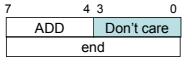
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução ADD (soma)

Simbólico: ADD end

RT: AC ← MEM(end) + AC

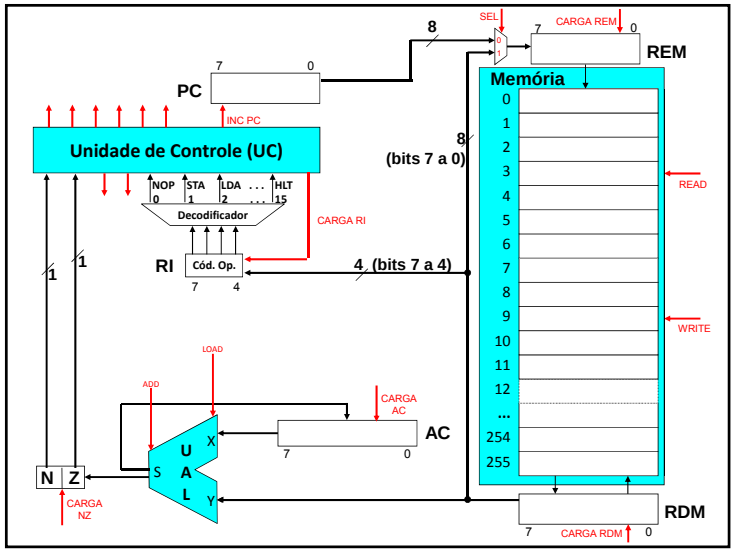


Passos no nível RT:

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

Precisamos acrescentar algum elemento à organização para executar esta instrução ? Qual(is) ?



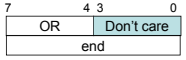
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução OR ("ou" lógico, bit a bit)

Simbólico: OR end

RT: AC ← MEM(end) OR AC

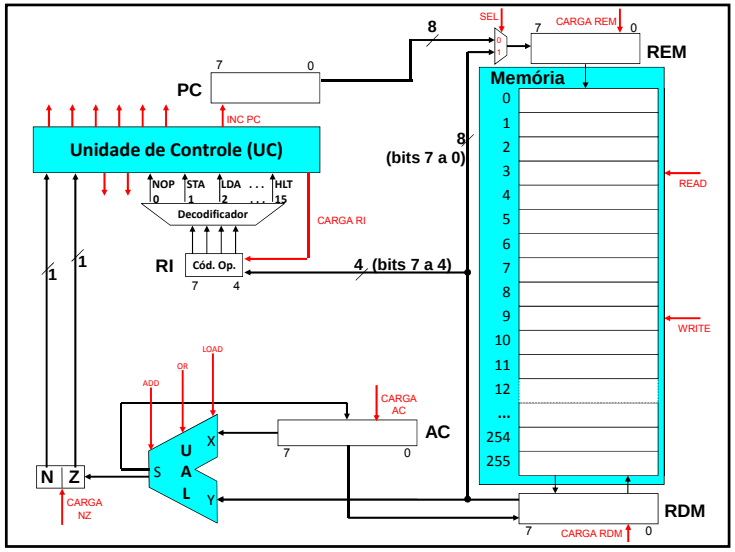


Passos no nível RT:

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

Precisamos acrescentar algum elemento à organização para executar esta instrução ? Qual(is) ?



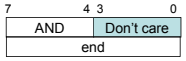
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução AND ("e" lógico, bit a bit)

Simbólico: AND end

RT: AC ← MEM(end) AND AC

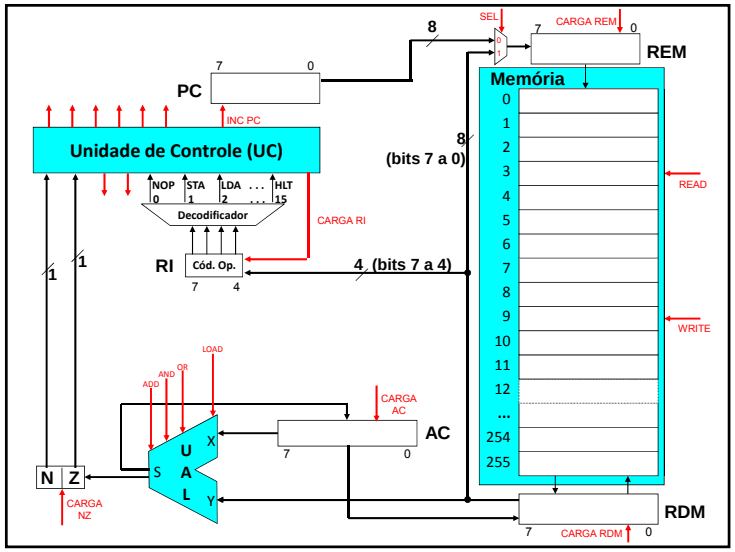


Passos no nível RT:

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

Precisamos acrescentar algum elemento à organização para executar esta instrução ? Qual(is) ?



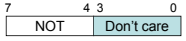
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução NOT (complementa acumulador)

Simbólico: NOT

RT: AC ← NOT AC

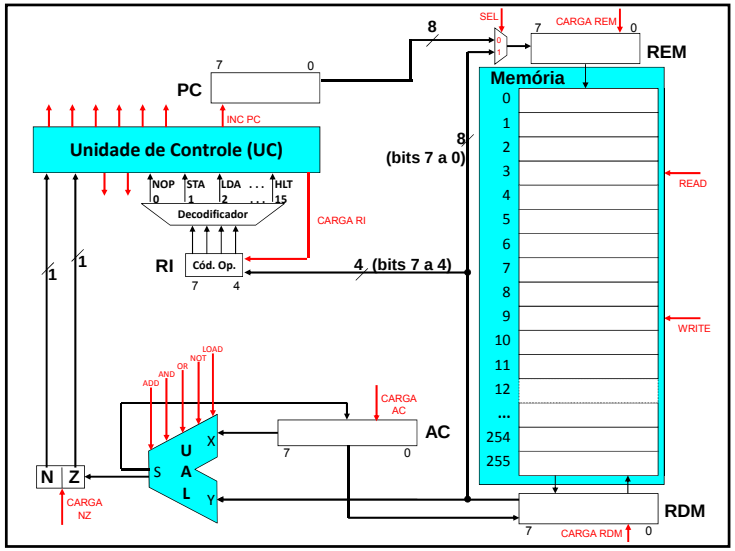


Passos no nível RT:

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

Execução: AC ← NOT(AC); atualiza N e Z

Precisamos acrescentar algum elemento à organização para executar esta instrução ? Qual(is) ?



O Computador Neander

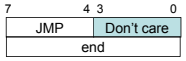
Arquitetura/Organização: transferências entre regs.

Instrução JMP (desvio incondicional - jump)

Simbólico: JMP end

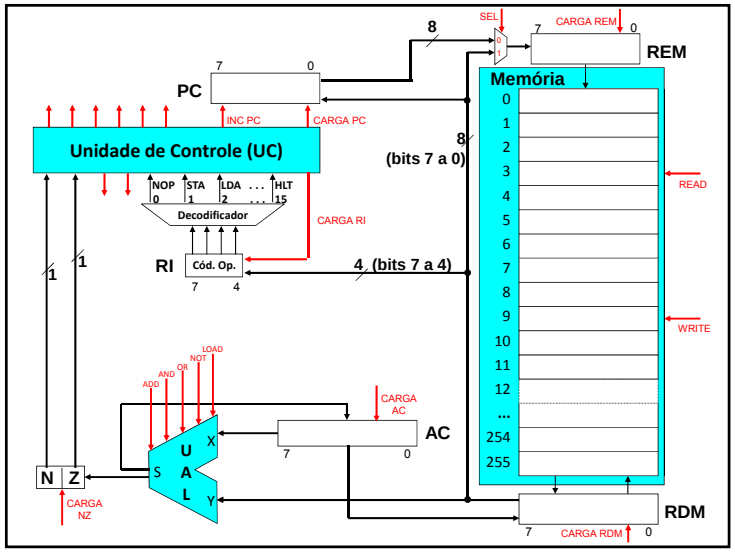
RT: PC ← end

Passos no nível RT:



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

Precisamos acrescentar algum elemento à organização para executar esta instrução ? Qual(is) ?



O Computador Neander

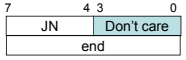
Arquitetura/Organização: transferências entre regs.

Instrução JN (desvio condicional - jump on negative)

Simbólico: JN end

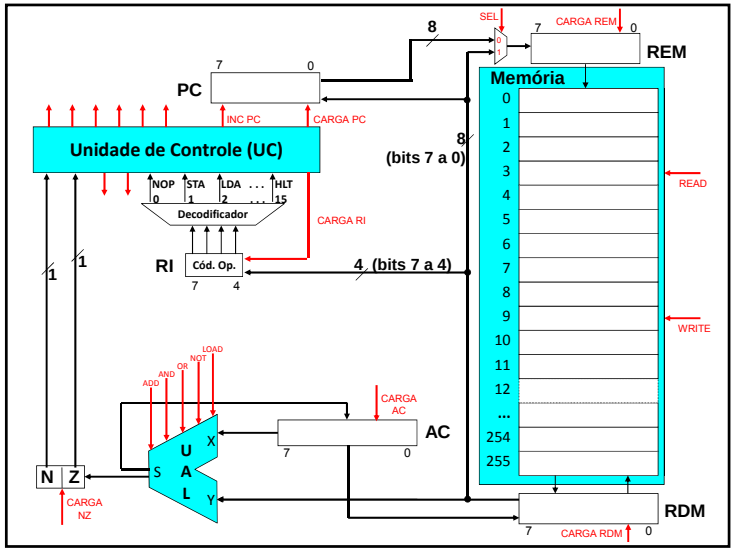
RT: IF N = 1 THEN PC ← end

Passos no nível RT:



Se N=1 (desvio ocorre)		Se N=0 (desvio não ocorre)	
Busca:	REM ← PC	Busca:	REM ← PC
	Read; PC ← PC + 1		Read; PC ← PC + 1
	RI ← RDM		RI ← RDM
Execução:	REM ← PC	Execução:	PC ← PC + 1
	Read		
	PC ← RDM		

Precisamos acrescentar algum elemento à organização para executar esta instrução ? Qual(is) ?



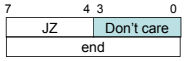
O Computador Neander

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:

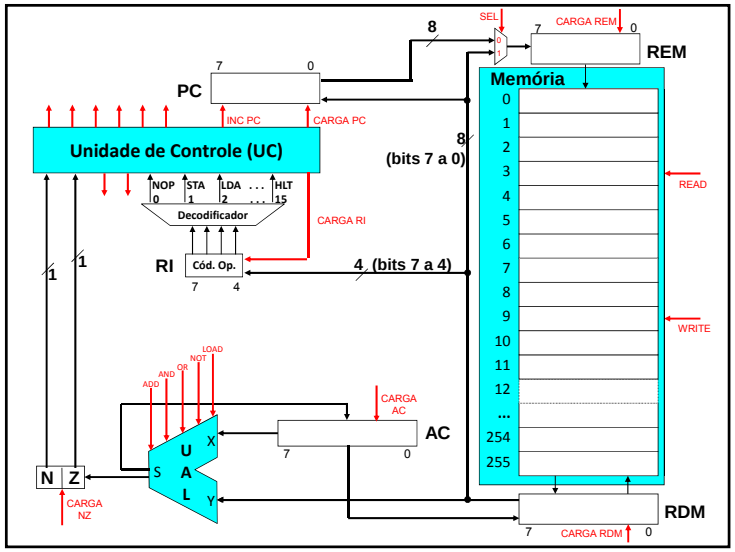
Se Z=1 (desvio ocorre)

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

Precisamos acrescentar algum elemento à organização para executar esta instrução ? Qual(is) ?



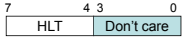
O Computador Neander

Arquitetura/Organização: transferências entre regs.

Instrução HLT (término de execução - halt)

Simbólico: HLT

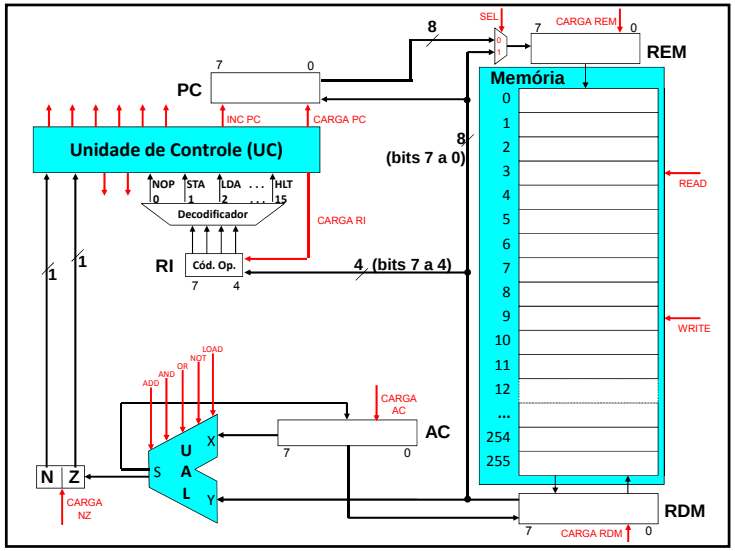
RT: --



Passos no nível RT:

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

Precisamos acrescentar algum elemento à organização para executar esta instrução ? Qual(is) ?



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(LOAD), 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(LOAD), 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	

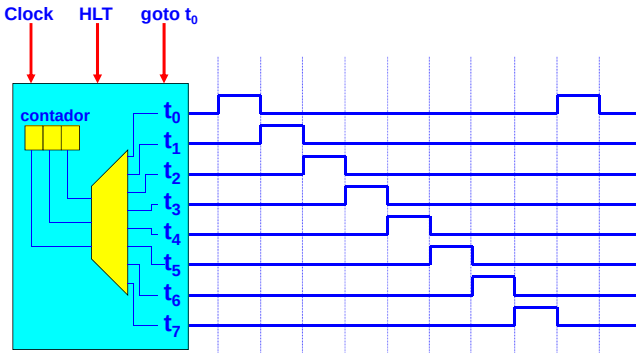
O Computador Neander

Temporização dos sinais de controle (parte 2)

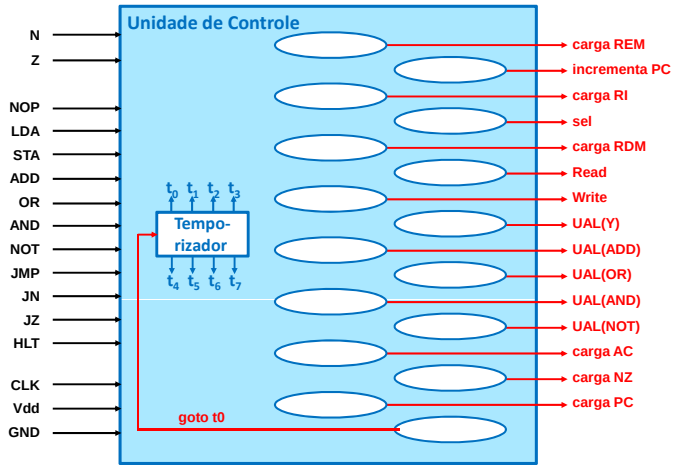
tempo	JMP	JN, N=1	JN, N=0	JZ, Z=1	JZ, Z=0	NOP	HLT
t0	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM
t1	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, 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 - exemplo



O Computador Neander



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)