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

O Computador Neander

A Organização: transferências necessárias

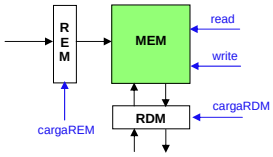
Analisando todas as descrições RT, a agrupando pelo registrador destino, tem-se:

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



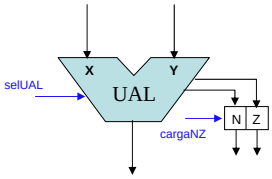
Associados à Memória:
- RDM (dados)
- REM (endereços)
- sinal de escrita (write)
- sinal de leitura (read)

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)**

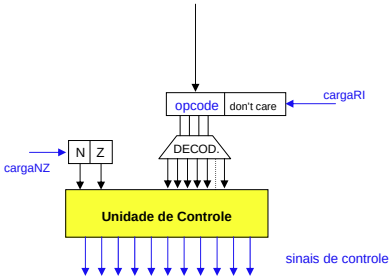


Flip-Flops devem ter sinal de carga

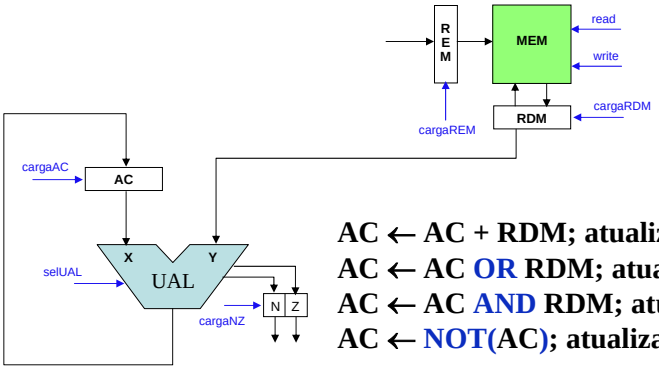
Organização do Registrador de Instrução

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**



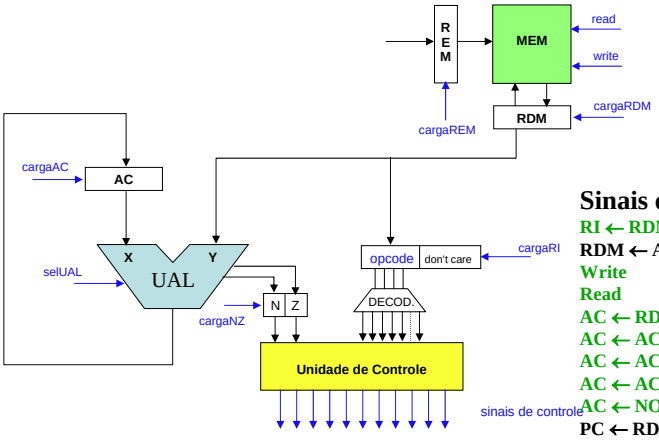
Operações na UAL



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

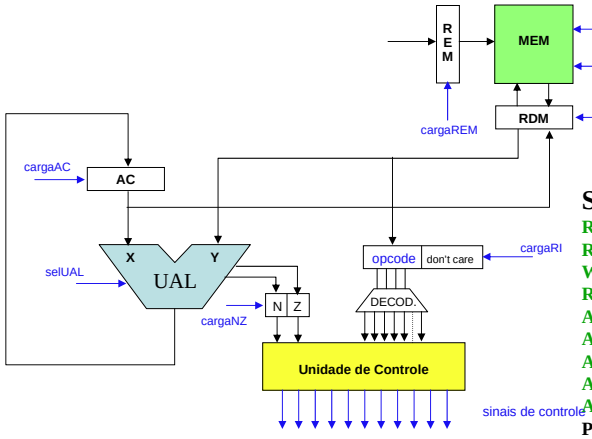
Dúvida:
AC ← RDM; atualiza N e Z (via UAL)

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)

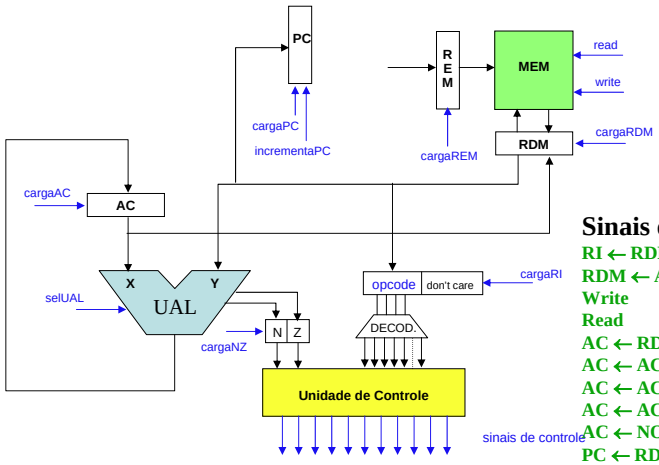
O incremento do PC pode ser feito:

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

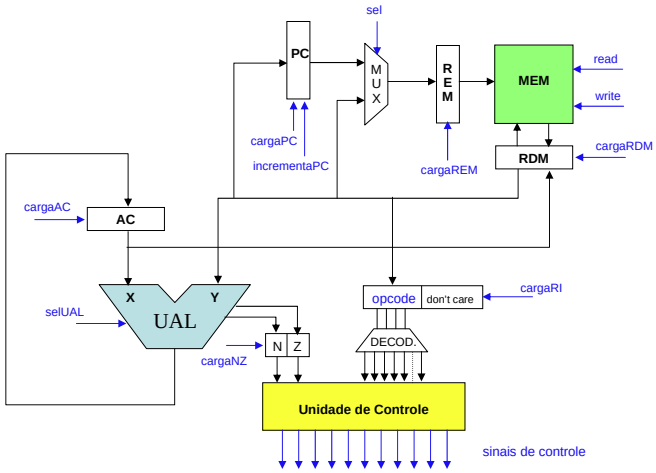
Valores para o REM

- 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

Organização final



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