Computador hipotético Neander

O computador NEANDER foi criado com intenções didáticas[1] e é extremamente simples. Tão simples que você, sem muito esforço, pode projetá-lo. Experimente!

4.1 Características

O computador NEANDER tem as seguintes características:

- Largura de dados e endereços de 8 bits
- Dados representados em complemento de dois
- 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)

4.2 Modos de endereçamento

O NEANDER só possui um modo de endereçamento: o modo direto (muitas vezes também chamado de absoluto).

No modo de endereçamento direto (Figura 4.1), a palavra que segue o código da instrução contém, nas instruções de manipulação de dados, o endereço de memória do operando.

Nas instruções de desvio, o endereço contido na instrução corresponde à posição de memória onde está uma instrução a ser executada.

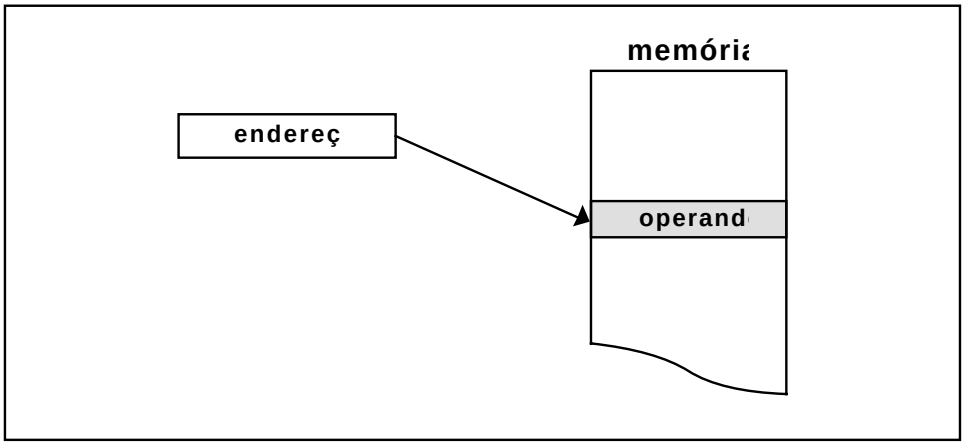

Figura 4.1- Modo de endereçamento direto

[1] Esta pseudo-máquina foi criada pelos Profs. Raul Weber e Taisy Weber para a antiga disciplina CPD148 - Arquitetura de Computadores I. Possui simulador e depurador associados, que podem ser vistos no apêndice A.

4.3 Conjunto de instruções

O conjunto de instruções de NEANDER compreende 11 instruções, codificadas através dos quatro bits mais significativos da palavra que contém o código da instrução (Tabela 4.1):

Código	Instrução		Comentário
0000	NOP		nenhuma operação
0001	STA	end	armazena acumulador - (store)
0010	LDA	end	carrega acumulador - (load)
0011	ADD	end	soma
0100	OR	end	"ou" lógico
0101	AND	end	"e" lógico
0110	NOT		inverte (complementa) acumulador
1000	JMP	end	desvio incondicional - (jump)
1001	JN	end	desvio condicional - (jump on negative)
1010	JZ	end	desvio condicional - (jump on zero)
1111	HLT		término de execução - (halt)

Tabela 4.1 - Conjunto de instruções do NEANDER

Na Tabela 4.1, **end** significa endereço direto. Nas instruções STA, LDA, ADD, OR e AND, **end** corresponde ao endereço de operando. Nas instruções JMP, JN e JZ, **end** corresponde ao endereço de desvio. As ações efetuadas por cada uma das instruções da Tabela 4.1 podem ser vistas na Tabela 4.2, a seguir:

Instrução		Comentário
NOP		nenhuma operação
STA	end	MEM(end) ← AC
LDA	end	AC← MEM(end)
ADD	end	AC← MEM(end) + AC
OR	end	AC← MEM(end) OR AC
AND	end	AC← MEM(end) AND AC
NOT		AC← NOT AC
JMP	end	PC← end
JN	end	IF N=1 THEN PC ← end
JZ	end	IF Z=1 THEN PC ← end

Tabela 4.2 - Ações executadas

Na Tabela 4.2 **AC** é o acumulador, **MEM(end)** significa conteúdo da posição **end** de memória, **N** e **Z** são os códigos de condição e ← representa uma atribuição.

4.4 Códigos de condição

A unidade lógica e aritmética de NEANDER fornece os seguintes códigos de condição, que são usados pelas instruções JN e JZ (vide Tabela 4.2):

N - (negativo) : sinal do resultado
1 - resultado é negativo
0 - resultado é positivo

Z - (zero) : indica resultado igual a zero
1 - resultado é igual a zero
0 - resultado é diferente de zero

As instruções lógicas e aritméticas (ADD, NOT, AND, OR) e a instrução de transferência LDA afetam os códigos de condição N e Z. As demais instruções (STA, JMP, JN, JZ, NOP e HLT) não alteram os códigos de condição.

4.5 Formato das instruções

As instruções de NEANDER são formadas por um ou dois bytes, ou seja, ocupam uma ou duas posições na memória (Figura 4.2).

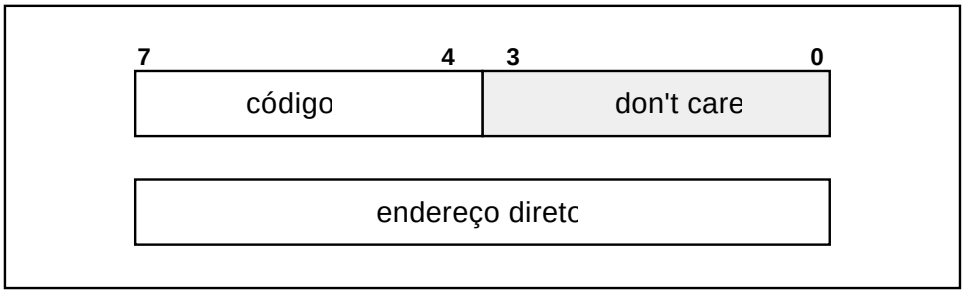

Figura 4.2 - Formato de instrução no NEANDER

Nas instruções de um byte, os 4 bits mais significativos contém o **código** da instrução. Nas instruções de dois bytes, o primeiro byte contém o **código** (também nos 4 bits mais significativos) e o segundo byte contém um **endereço**. Instruções de dois bytes, no NEANDER, são aquelas instruções que fazem referência à memória.

4.6 Exemplo de programação

Vamos considerar, como exemplo, um programa que realiza a soma de 3 posições consecutivas da memória e armazena o resultado numa quarta posição. Inicialmente, devem ser escolhidas a área de dados e a área de programa, ou seja, a localização das instruções e dados na memória. Não existem critérios para essa escolha, mas deve ser observado que a área de programa não pode invadir a área de dados e vice-versa. Seja, para esse programa, escolhida uma alocação de memória de tal forma que o programa ocupe a metade inferior da memória e os dados a metade superior, como segue:

área de programa		
início do programa	posição 0	(0H)
área de dados		
primeira parcela	posição 128	(80H)
segunda parcela	posição 129	(81H)
terceira parcela	posição 130	(82H)
resultado	posição 131	(83H)

O programa seria:

Simbólico	**Comentários**
LDA 128	% acumulador A recebe conteúdo da posição 128
ADD 129	% conteúdo de A é somado ao conteúdo da posição 129
ADD 130	% conteúdo de A é somado ao conteúdo da posição 130
STA 31	% conteúdo de A é copiado na posição 131
HLT	% processador para

Esse programa pode ser editado em linguagem de máquina (tanto em hexa como em decimal), depurado e executado usando o simulador/depurador NEANDER, cujos comandos foram apresentados no capítulo respectivo. A codificação em linguagem de máquina correspondente a cada uma das instruções mostradas acima seria:

Simbólico	**Hexa**		**Decimal**	
LDA 128	20	80	32	128
ADD 129	30	81	48	129
ADD 130	30	82	48	130
STA 131	10	83	16	131
HLT	F0		240	

4.7 Conclusão

NEANDER é um computador muito simples, desenvolvido apenas para fins didáticos. Processadores modernos são muito mais complexos que NEANDER. Entretanto, mesmo processadores utilizados nas mais sofisticadas estações de trabalho são baseados nos conceitos elementares que você aprendeu com NEANDER.

4.8 Exercícios de programação usando o NEANDER

Os exercícios apresentados aqui são apenas uma amostra do que pode ser programado com NEANDER. Na definição de novos problemas, o único cuidado que deve ser tomado é com a memória disponível para programa e dados, que compreende apenas 256 posições. Exceto onde explicitado, todos os números e endereços são representados na base decimal.

Para todos os programas sugeridos, vale a seguinte convenção:

início do programa - posição 0 (0H)
início da área de dados - posição 128 (80H)

Essa convenção é adotada apenas para facilitar a correção dos programas.

1. Limpar o acumulador: faça 4 programas diferentes que zerem o acumulador.

2. Somar duas variáveis de 8 bits: faça um programa para somar duas variáveis representadas em complemento de dois. As variáveis e o resultado estão dispostos segundo o mapa de memória abaixo:
   posição 128: primeira variável
   posição 129: segunda variável
   posição 130: resultado

3. Subtrair duas variáveis: faça um programa para subtrair duas variáveis de 8 bits representadas em complemento de dois. O resultado deve aparecer na posição de memória consecutiva às ocupadas pelas variáveis.
   posição 128: minuendo
   posição 129: subtraendo
   posição 130: resultado

4. Comparação: determine qual a maior de 3 variáveis positivas de 8 bits armazenadas em posições consecutivas de memória. O resultado (ou seja, a maior variável), deve aparecer na primeira posição livre de memória na área reservada aos dados.

5. Determinação de overflow na soma: faça um programa que determine a ocorrência de overflow na soma de duas variáveis. As variáveis são de 8 bits em complemento de dois e estão armazenadas em posições consecutivas de memória. O resultado da soma, também

em 8 bits, deve aparecer na primeira posição livre e overflow deve ser indicado da seguinte forma:

posição 130: conteúdo = 0H quando não ocorreu overflow
conteúdo = FFH quando ocorreu overflow

6. Limpeza de uma área de memória: faça um programa para zerar 32 posições consecutivas na memória.