NEANDERWIN

O NeanderWin é um simulador da máquina Neander, definida no livro do Raul F. Weber (UFRGS), Fundamentos de Arquitetura de Computadores, Ed. Sagra Luzzatto. A máquina original foi estendida aqui para incluir algumas instruções para carga de dados imediatos no acumulador e operações de entrada e saída de dados para dois dispositivos mapeados em nosso simulador: um teclado e um visor.

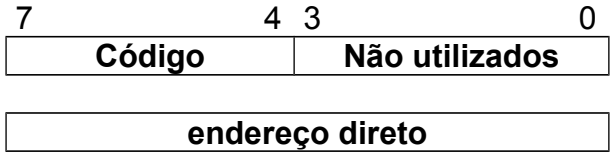
Algumas características do processador Neander são:

- Largura de dados e endereços de 8 bits
- Dados representados em complemento a dois
- 1 acumulador de 8 bits (AC)
- 1 apontador de instruções de 8 bits (PC)
- 1 registrador de código de condição com 2 bits: negativo (N) e zero (Z)

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, esse endereço corresponde à posição de memória onde está a próxima instrução a ser executada.
O NEANDERWIN foi estendido para ter mais dois modos de endereçamento: imediato e indireto. Maiores detalhes serão mostrados mais adiante.

1) Listagem geral das instruções

As instruções podem ter um ou dois bytes. Nas instruções com apenas um byte, os 4 bits mais significativos contém o código da operação. Nas instruções com dois bytes, que no Neander são aquelas que fazem referência a um operando na memória, o segundo byte contém o endereço de memória deste operando.



Nota:
Os 4 bits de mais baixa ordem do primeiro byte são reservados para futuras expansões. Existem também dois códigos que não têm instruções associadas.

'NOP' código 0

O comando NOP é usado apenas para gastar tempo.

'STA ender' código 1

O comando STA guarda o acumulador na posição de memória indicada pelo operando ender.

'LDA ender' código 2

O comando LDA atribui ao acumulador o conteúdo da posição de memória indicada pelo operando ender.

'ADD ender' código 3

O comando ADD soma ao acumulador o conteúdo de uma posição de memória indicada pelo operando ender.

'OR ender' código 4

O comando OR realiza um "ou" lógico entre o acumulador e o conteúdo de uma posição de memória indicada pelo operando ender.

'AND ender' código 5

O comando AND realiza um "e" lógico entre o acumulador e o conteúdo de uma posição de memória indicada pelo operando ender.

'NOT' código 6

O comando NOT inverte os bits do acumulador.

'JMP ender' código 8

O comando JMP (jump) desvia a execução do programa para o endereço indicado pelo operando ender.

'JN ender' código 9

O comando JN (jump if negative) desvia a execução do programa para o endereço indicado pelo operando ender, apenas quando a última operação realizada produziu um valor com o bit 7 ligado (negativo).

'JZ ender' código 10

O comando JZ (jump if zero) desvia a execução do programa para o endereço indicado pelo operando ender, apenas quando a última operação realizada produziu um valor zero.

'JNZ ender' código 11

O comando JNZ (jump if not zero) desvia a execução do programa para o endereço indicado pelo operando ender, apenas quando a última operação realizada produziu um valor diferente de zero.

'IN ender' código 12

O comando IN (input) traz para o acumulador o valor lido num dispositivo externo indicado pelo operando ender. No Neanderwin os dispositivos são: chaves (endereço 0) e o status de "dado disponível" das chaves (endereço 1).

'OUT ender' código 13

O comando OUT (output) descarrega o conteúdo do acumulador em um dispositivo externo indicado pelo operando ender. No Neanderwin o único dispositivo disponível é um visor (endereço 0).

'LDI imed' código 14

O comando LDI (load immediate) carrega no acumulador o valor dado pelo operando imed.

'HLT' código 15

O comando HLT (halt) para a máquina.

Modos de Endereçamento:

- imediato

O segundo byte da instrução é o operando.
A única instrução que usa este modo de endereçamento é a LDI.

- direto

O segundo byte da instrução é o endereço de memória do operando.

- indireto

O segundo byte da instrução contém o endereço de memória onde está o endereço do operando (ou seja, o segundo byte da instrução é o endereço do ponteiro para o operando). Para indicar que um operando é indireto, deve-se precedê-lo pela letra "@" (arrôba)

2) Comentários no programa

Os comentários são começados por ponto e vírgula, e podem também ocorrer no final das linhas de instruções.

3) Rótulos

Um rótulo é um nome dado à próxima posição de memória.
O nome é seguido por dois pontos.

4) Pseudo Instruções:

ORG ender

A pseudo-instrução ORG (origin) indica ao montador que a próxima instrução será colocado na posição ender de memória.

var EQU imed

A pseudo-instrução EQU (equate) atribui um nome (rótulo) a um determinado valor.

Esse comando é frequentemente usado para especificar variáveis que são posicionadas em um endereço específico de memória. Por exemplo para posicionar a variável x no endereço 100 use: X EQU 100

END ender

A pseudo-instrução END indica que o programa fonte acabou.

O operando ender é usado para pré-carregar o PC com o endereço inicial de execução do programa.

DS imed

A pseudo-instrução DS (define storage) reserva um número de palavras na memória definido pelo valor imed.

DB imed

A pseudo-instrução DB (define bytes) carrega esta palavra com o valor dado pelo operando imed.

5) Exemplos de representação de números

Decimal: 30
Binário: 00110000b
Hexadecimal: 30h

Obs: Números hexadecimais maiores que 7Fh devem ser precedidos por um zero, p. ex. 0F3h

Tabela de Instruções

Código binário	Instrução	Descrição
0000	NOP	nenhuma operação
0001	STA ender	armazena acumulador (store)
0010	LDA ender	carrega acumulador (load)
0011	ADD ender	Soma
0100	OR ender	operação lógica “ou”
0101	AND ender	operação lógica “e”
0110	NOT	inverte (complementa) acumulador
1000	JMP ender	desvio incondicional (jump)
1001	JN ender	desvio condicional (jump on negative)
1010	JZ ender	desvio condicional (jump on zero)
1011	JNZ ender	desvio condicional (jump on not zero)
1100	IN ender	operação de entrada no dispositivo “ender”
1101	OUT ender	operação de saída no dispositivo “ender”
1110	LDI imed	carrega o valor imediato imed no acumulador
1111	HLT	término da execução (halt)

Instrução	Comentário
NOP	nenhuma operação
STA ender	MEM(ender) ← AC

LDA ender	AC ← MEM(ender)
ADD ender	AC ← AC + MEM(ender)
OR ender	AC ← AC or MEM(ender)
AND ender	AC ← AC and MEM(ender)
NOT	AC ← NOT AC
JMP ender	PC ← ender
JN ender	if N = 1 then PC ← ender
JZ ender	if Z = 1 then PC ← ender
JNZ ender	if Z = 0 then PC ← ender
IN ender	AC ← IN (ender)
OUT ender	OUT(ender) ← AC
LDI imed	AC ← imed
HALT	halt

Códigos de condição:

N – (negativo): sinal do resultado
1 – resultado é negativo
0 – resultado não é negativo

Z – (zero): indica resultado igual a zero
1 – resultado é igual a zero
0 – resultado 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, JNZ, IN, OUT, LDI, NOP e HLT) não alteram os códigos de condição.

Ciclo de busca da instrução:

RI ← MEM(PC)
PC ← PC + 1

Sequência de operações para execução da instrução:

Instrução:	**Ciclo**	**operações**
NOP	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	nenhuma
STA	Busca	RI ← MEM(PC) PC ← PC + 1

	Execução	ender ← MEM(PC) PC ← PC + 1
LDA	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) PC ← PC + 1 AC ← MEM(ender)

ADD	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) PC ← PC + 1 AC ← AC + MEM(ender)
OR	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) PC ← PC + 1 AC ← AC or MEM(ender)
AND	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) PC ← PC + 1 AC ← AC and MEM(ender)
NOT	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	AC ← not AC
JMP	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) PC ← ender
JN caso N = 1	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) PC ← ender
JN caso N = 0	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) PC ← PC + 1

JZ caso Z = 1	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) PC ← ender
JZ caso Z = 0	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) PC ← PC + 1
JNZ caso Z = 1	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) PC ← PC + 1
JNZ caso Z = 0	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) PC ← ender
IN ender	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) IN (ender)
OU ender	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) OUT (ender)
LDI imed	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	ender ← MEM(PC) AC ← ender
HLT	Busca	RI ← MEM(PC) PC ← PC + 1
	Execução	parar o processamento

Sequências considerando operações de leitura e escrita na memória

Instrução:	**Ciclo**	**Operações**
NOP	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	Nenhuma
STA	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	REM ← PC Read; PC ← PC + 1 REM ← RDM RDM ← AC write
LDA	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
ADD	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

OR	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
AND	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
NOT	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	AC ← not AC
JMP	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	REM ← PC Read; PC ← RDM
JN caso N = 1	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	REM ← PC Read; PC ← RDM

JN caso N = 0	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	PC ← PC + 1
JZ caso Z = 1	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	REM ← PC Read; PC ← RDM
JZ caso Z = 0	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	PC ← PC + 1
JNZ caso Z = 1	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	PC ← PC + 1
JNZ caso Z = 0	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	REM ← PC Read; PC ← RDM
IN	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	REM ← PC Read; PC ← PC + 1 REM ← RDM input AC ← RDM; atualiza N e Z
OUT	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	REM ← PC Read; PC ← PC + 1 REM ← RDM RDM ← AC; output
LDI	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	REM ← PC

		Read; PC ← PC + 1 AC ← RDM; atualiza N e Z
HLT	Busca	REM ← PC Read; PC ← PC + 1 RI ← RDM
	Execução	Parar o processamento

Exemplos 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 área de programa não pode invadir a área de dados e vice-versa. Ou seja, para esse programa, foi 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

X  EQU  128     ; Endereço da variável X definido como 128
Y  EQU  129     ; Endereço da variável Y definido como 129
W EQU   130     ; Endereço da variável W definido como 130
Z  EQU  131     ; Endereço da variável Z definido como 131


 ORG  0
LDA   X         ; acumulador A recebe conteúdo de X
ADD   Y         ; conteúdo de A é somado ao conteúdo de Y
ADD   W         ; conteúdo de A é somado ao conteúdo de W
STA   Z         ; conteúdo de A é copiado para Z
HLT             ; processador pára
```

Esse programa pode ser editado em linguagem de montagem, depurado e executado usando o simulador/depurador NEANDERWIN, cujos comandos foram apresentados anteriormente.

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.

O NEANDERWIN estende o conjunto de instruções do NEANDER e oferece uma interface de programação amigável, com a entrada do código em representação simbólica, com diversas facilidades para o programador, que tornam muito mais fácil o uso do processador NEANDER como ferramenta de ensino. Estão disponíveis versões

tanto para o sistema operacional Windows e Linux. O código fonte está disponível mediante solicitação.

Exercícios de programação usando o NEANDERWIN

Os exercícios apresentados aqui são apenas uma amostra do que pode ser programado com NEANDERWIN. 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.

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 a dois e verifique se ocorreu "overflow". Indique a ocorrência de "overflow" da seguinte maneira:

   conteúdo = 0 quando não ocorreu overflow
   conteúdo = 1 quando ocorreu overflow

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 de entrada.

4. Comparação: determine qual a maior valor de um vetor com 10 elementos de 8 bits cada. Coloque os valores iniciais do vetor com uso das diretivas do montador.