Gramática do Assembler Neander

Arquivo: assembler.pdf
Autor: Ygor Silva

Visão Geral
Este documento descreve a gramática reconhecida pelo assembler desenvolvido para a arquitetura Neander. O montador é capaz de interpretar instruções de baixo nível e gerar um binário `.mem` com cabeçalho e representação de 256 bytes de memória em ASCII hexadecimal.

Formato do Arquivo de Entrada `.asm`

Um programa `.asm` é composto por duas seções principais:

1. Seção de Dados (`.DATA`)
Contém variáveis e valores literais utilizados no programa.

```
.DATA
X DB 10
Y DB ?
UM DB 1
ZERO DB 0
```

- Cada linha define uma variável e seu valor inicial.
- O valor `?` indica que o dado não é inicializado (armazenado como `0`).

2. Seção de Código (`.CODE`)
Define a lógica do programa.

```
.CODE
.ORG 0
LDA X
ADD Y
STA X
HLT
```

- `.ORG` define o endereço inicial onde o código será carregado (em geral 0).
- As instruções seguem o formato da linguagem Neander.
- Podem ser precedidas por rótulos:

```
LOOP: LDA X
    ADD Y
    JMP LOOP
```

Instruções Suportadas

Mnemônico	Opcode (hex)	Operando	Tamanho
NOP	0x00	Nenhum	1 byte
STA	0x10	Sim	2 bytes
LDA	0x20	Sim	2 bytes
ADD	0x30	Sim	2 bytes

Gramática do Assembler Neander

OR	0x40	Sim	2 bytes
AND	0x50	Sim	2 bytes
NOT	0x60	Não	1 byte
JMP	0x80	Sim	2 bytes
JN	0x90	Sim	2 bytes
JZ	0xA0	Sim	2 bytes
HLT	0xF0	Não	1 byte

Gramática BNF Simplificada

<programa> ::= <data_section> <code_section>
<data_section> ::= '.DATA' <declaracao>*
<code_section> ::= '.CODE' '.ORG' <numero> <instrucoes>*
<declaracao> ::= <identificador> 'DB' (<numero> | '?')
<instrucoes> ::= [<rotulo> ':'] <instrucao>
<instrucao> ::= 'NOP' | 'STA' <operando> | 'LDA' <operando> | 'ADD' <operando>
| 'OR' <operando> | 'AND' <operando> | 'NOT' | 'JMP' <operando>
| 'JN' <operando> | 'JZ' <operando> | 'HLT'
<operando> ::= <identificador> | <numero>
<numero> ::= [0-9]+
<identificador> ::= [A-Za-z_][A-Za-z0-9_]*

Observações

- Labels (rótulos) devem terminar com `:` e preceder a instrução.
- Comentários com `;` são ignorados pelo assembler.
- A memória de dados é alocada a partir do final (endereço 255 para trás).
- Os programas são limitados a 256 bytes totais (código + dados).

Exemplo Completo

```
.DATA
X DB 10
Y DB 5
TMP DB ?
UM DB 1
ZERO DB 0

.CODE
.ORG 0
LDA X
ADD Y
STA TMP
HLT
```