



## 4 Instruções

Instrução	opcode	Comentário
load	0 (00000000 <sub>2</sub> )	memória
load	1 (00000001 <sub>2</sub> )	valor
store	2 (00000010 <sub>2</sub> )	—
add	3 (00000011 <sub>2</sub> )	—
sub	4 (00000100 <sub>2</sub> )	—
mul	5 (00000101 <sub>2</sub> )	—
div	6 (00000110 <sub>2</sub> )	—
inc	7 (00000111 <sub>2</sub> )	—
dec	8 (00001000 <sub>2</sub> )	—
and	9 (00001001 <sub>2</sub> )	—
or	10 (00001010 <sub>2</sub> )	—
not	11 (00001011 <sub>2</sub> )	—
jmp	12 (00001100 <sub>2</sub> )	—
jz	13 (00001101 <sub>2</sub> )	—
jnz	14 (00001110 <sub>2</sub> )	—
hlt	15 (00001111 <sub>2</sub> )	fim do programa

### 4.1 LOAD

A instrução **load** opera de duas maneiras distintas: carrega um valor **direto** para o acumulador ou carrega um valor em **memória** para o acumulador. O trecho de código abaixo exibe as diferenças.

Exemplo de código:

```
load 10 ; acc = 10. Carrega o valor 10 no acumulador
```

```
load $10 ; acc = mem[10]. Carrega o valor que tem endereço 10 no acumulador
```

Formato de Código : load \$addr

Código da Operação : 00

Função : Carrega o valor da posição de memória 'addr' em acc

Exemplo de instrução:

```
      opcode      addr
+-----+-----+
| 00000000 | 00001010 |
+-----+-----+
load      $10
```

Formato de Código : load num

Código da Operação : 01

Função : Carrega o valor de 'num' em acc

Exemplo de instrução:

```
      opcode      addr
+-----+-----+
| 00000001 | 00001011 |
+-----+-----+
load      11
```

### 4.2 STORE

Formato de Código : store \$addr

Código da Operação : 02

Função : Armazena o valor do acumulador na posição de memória definida por 'addr'

Exemplo de instrução:

opcode	addr
+-----+-----+	
00000010	10000111
+-----+-----+	
store	\$135; mem[135] = acc

### 4.3 ADD

Operação de **adição** entre o acumulador e um valor em memória.

Formato de Código : add \$addr  
 Código da Operação : 03  
 Função : acc = acc + mem[addr]

**Exemplo de instrução:**

opcode	addr
+-----+-----+	
00000011	00100101
+-----+-----+	
add	\$37

### 4.4 SUB

Operação de **subtração** entre o acumulador e um valor em memória

Formato de Código : sub \$addr  
 Código da Operação : 04  
 Função : acc = acc - mem[addr]

**Exemplo de instrução:**

opcode	addr
+-----+-----+	
00000100	11000010
+-----+-----+	
sub	\$194

### 4.5 MUL

Operação de **multiplicação** entre o acumulador e um valor em memória.

Formato de Código : mul \$addr  
 Código da Operação : 05  
 Função : acc = acc \* mem[addr]

**Exemplo de instrução:**

opcode	addr
+-----+-----+	
00000101	00100100
+-----+-----+	
mul	\$36

### 4.6 DIV

Operação de **divisão** entre o acumulador e um valor em memória.

Formato de Código : div \$addr  
 Código da Operação : 06  
 Função : acc = acc / mem[addr]

**Exemplo de instrução:**

opcode	addr
+-----+-----+	
00000110	00100100
+-----+-----+	
div	\$36

## 4.7 INC

Incrementa em uma unidade o acc.

Formato de Código : inc  
 Código da Operação : 07  
 Função :  $acc = acc + 1$

**Exemplo de instrução:**

opcode	addr
+-----+-----+	
00000111	00000000
+-----+-----+	
inc	

## 4.8 DEC

Decrementa em uma unidade o acc.

Formato de Código : dec  
 Código da Operação : 08  
 Função :  $acc = acc - 1$

**Exemplo de instrução:**

opcode	addr
+-----+-----+	
00001000	00000000
+-----+-----+	
dec	

## 4.9 AND

Operação de **E-lógico** entre o acumulador e um valor em memória.

Formato de Código : and \$addr  
 Código da Operação : 09  
 Função :  $acc = acc \& mem[addr]$

**Exemplo de instrução:**

opcode	addr
+-----+-----+	
00001001	00000100
+-----+-----+	
and	\$4

## 4.10 OR

Operação de **OU-lógico** entre o acumulador e um valor em memória.

Formato de Código : or \$addr  
 Código da Operação : 10  
 Função :  $acc = acc | mem[addr]$

**Exemplo de instrução:**

opcode	addr
+-----+-----+	
00001010	00001100
+-----+-----+	
or	\$12

#### 4.11 NOT

Operação de **negação** de todos os bits do acumulador: executa o complemento a 2 no acumulador.

Formato de Código : not  
 Código da Operação : 11  
 Função : acc = ~acc

**Exemplo de instrução:**

opcode	addr
+-----+-----+	
00001011	00001100
+-----+-----+	
not	\$12

#### 4.12 JMP

Desvio incondicional para o label.

Formato de Código : jmp #label  
 Código da Operação : 12  
 Função : desvia a execução para  
 a definição de 'label'

**Exemplo de instrução:**

opcode	addr
+-----+-----+	
00001100	00001101
+-----+-----+	
jmp	13

#### 4.13 JZ

Desvia caso o registrador acc seja igual a 0. Caso contrário, prossiga no código. Desta forma, o registrador pc será modificado se o acc for igual a 0.

Formato de Código : jnz #label  
 Código da Operação : 13  
 Função : Se acc == 0, desvia a execução para  
 a definição de 'label'

**Exemplo de instrução:**

opcode	ref.
+-----+-----+	
00001101	01001110
+-----+-----+	
jz	78

#### 4.14 JNZ

Desvia caso o registrador acc seja diferente de 0. Caso contrário, prossiga no código. Desta forma, o registrador pc será modificado se o acc for diferente de 0.

Formato de Código : jz #label  
 Código da Operação : 14  
 Função : Se acc != 0, desvia a execução para a definição de 'label'

O assembler *vas* traduz cada *label* em uma posição do programa. **Exemplo de instrução:**

```

opcode      ref.
+-----+-----+
| 00001110 | 00000111 |
+-----+-----+
      jz      7
  
```

No exemplo acima, se no momento que foi executada a função **jz** o acumulador for zero *program counter* deverá ser alterado para 7.

## 4.15 HLT

Indica o fim do programa.

Formato de Código : hlt  
 Código da Operação : 15  
 Função : Fim do programa

**Exemplo de instrução:**

```

opcode
+-----+-----+
| 00001111 | 00000000 |
+-----+-----+
      hlt
  
```

## 5 Registrador STAT

STAT (8 bits)

```

+-----+
| xxxxx0CZ |
+-----+
  
```

O : Overflow  
 C : Carry  
 Z : Zero ACC  
 x : não utilizado

## 6 Exemplos

```

; soma acc = 10 + 20
load  20    ; acc = 20
store $1    ; mem[1] = 20
load  10    ; acc = 10
add   $1    ; acc = acc + mem[1]
hlt                    ; fim do programa
  
```

Programa executável:

```

      load      20      store      $1      load      10      add      $1      hlt
00000001 00010100 00000010 00000001 00000001 00001010 00000011 00000001 00001111 00000000
|          |
+-----16 bits-----+
  
```

O programa executável constitui apenas os valores. Se esse programa fosse visualizado por bytes em representação de base 16 (hexadecimal) teríamos o seguinte programa:

```
0x01 0x14 0x02 0x01 0x01 0x0A 0x03 0x01 0x0F 0x00
```

```
; Exemplo de loop
; acc = 10 + 9 + 8 + ... + 1
load 0
store $1 ; $1 = 0

load 1
store $3 ; $3 = 1

load 10 ; acc = 10

#While:
    store $2 ; $2 = acc
    load $1 ; acc = $1
    add $2 ; acc = $2 + $1
    store $1 ; $1 = acc
    load $2 ; acc = $2
    sub $3 ; acc = acc - 1
    jnz #While
load $1

hlt
```