
Simulador da Máquina de Von Neumann

Release 0.1

N. M. Cardoso, R. N. Fleury

10 mai. 2023

Contents:

1	API Reference	1
1.1	pyvnm.vm	1
1.2	pyvnm.system	13
	Índice de Módulos Python	19
	Índice	21

API Reference

pyvnm.vm

pyvnm.system

1.1 pyvnm.vm

Modules

pyvnm.vm.cpu

pyvnm.vm.device

pyvnm.vm.memory

pyvnm.vm.utils

pyvnm.vm.vnm

1.1.1 pyvnm.vm.cpu

Classes

<i>CPU</i>	Responsavel por executar um código carregado na memória
<i>CPUCallback</i>	
<i>CPUState</i>	Entidade que armazena o estado da Unidade Central de Processamento da Máquina de Von Neumann
<i>InstructionSet</i>	

CPU

class pyvnm.vm.cpu.CPU

Base: `object`

Responsavel por executar um código carregado na memória

initial_state: `MachineState` Estado inicial da máquina, usualmente gerado pelo carregador

_action_AD(*operand: int*)

Ação realizada pela instrução AD (Adição)

operand [int] Endereço da memória

_action_DV(*operand: int*)

Ação realizada pela instrução DV (Divisão)

operand [int] Endereço da memória

_action_GD(*operand: int*)

Ação realizada pela instrução GD (Get Data)

operand [int] Endereço do dispositivo

_action_HJ(*operand: int*)

Ação realizada pela instrução HJ (Jump after halt)

operand [int] Endereço da memória

_action_JN(*operand: int*)

Ação realizada pela instrução JN (Jump if acc < 0)

operand [int] Endereço da memória

_action_JP(*operand: int*)

Ação realizada pela instrução JP (Jump incondicional)

operand [int] Endereço da memória

_action_JZ(*operand: int*)

Ação realizada pela instrução JZ (Jump if acc = 0)

operand [int] Endereço da memória

_action_LD(*operand: int*)

Ação realizada pela instrução LD (Load)

operand [int] Endereço da memória

_action_ML(*operand: int*)

Ação realizada pela instrução ML (Jump multiplicação)

operand [int] Endereço da memória

_action_OS(*operand: int*)

Ação realizada pela instrução OS (Chamada no sistema operacional)

operand [int] Código

_action_PD(*operand: int*)

Ação realizada pela instrução PD (Put Data)

operand [int] Endereço do dispositivo

_action_RS(*operand: int*)

Ação realizada pela instrução RS (Return from subroutine)

operand [int] Endereço da memória

_action_SB(*operand: int*)

Ação realizada pela instrução SB (Subtração)

operand [int] Endereço da memória

_action_SC(*operand: int*)

Ação realizada pela instrução SC (Subroutine Call)

operand [int] Endereço da memória

_action_ST(*operand: int*)

Ação realizada pela instrução ST (Store)

operand [int] Endereço da memória

_increment_pc()

Incrementa o registrador PC em uma unidade

event_loop()

Inicia a execução de um programa a partir da posição indicada pelo registrador PC.

CPUCallback

class pyvnm.vm.cpu.CPUCallback

Base: `object`

on_event_loop_begin(*state: pyvnm.vm.cpu.CPUState*)

Função executada no início do loop de eventos

state [CPUState] O estado da CPU

CPUState: estado da CPU

on_event_loop_end(state: `pyvnm.vm.cpu.CPUState`)

Função executada no fim do loop de eventos

state [`CPUState`] O estado da CPU

`CPUState`: estado da CPU

on_instruction_begin(state: `pyvnm.vm.cpu.CPUState`)

Função executada imediatamente antes da execução da instrução

state [`CPUState`] O estado da CPU

`CPUState`: estado da CPU

on_instruction_end(state: `pyvnm.vm.cpu.CPUState`)

Função executada imediatamente após a execução da instrução

state [`CPUState`] O estado da CPU

`CPUState`: estado da CPU

`CPUState`

class `pyvnm.vm.cpu.CPUState`

Base: `object`

Entidade que armazena o estado da Unidade Central de Processamento da Máquina de Von Neumann

memory: `Memory` A memória da máquina

devices: `DeviceBus` O bus de dispositivos da máquina

acc: `int`, **opcional** O valor inicial do registrador acumulador, valor padrão: 0

pc: `int`, **opcional** O valor inicial do registrador PC, valor padrão: 0

pc_lock: `Lock` Cadeado usado para implementação da lógica de bloqueio do registrador PC (program counter)

sig_term: `bool` Sinal que indica a terminação da execução de um programa em execução

`InstructionSet`

class `pyvnm.vm.cpu.InstructionSet`

Base: `object`

Attributes

<i>AD</i>	Instrução AD (Adição)
<i>DV</i>	Instrução DV (Divisão)
<i>GD</i>	Instrução GD (Get Data)
<i>HJ</i>	Instrução HJ (Jump after halt)
<i>JN</i>	Instrução JN (Jump if acc < 0)
<i>JP</i>	Instrução JP (Jump incondicional)
<i>JZ</i>	Instrução JZ (Jump if acc = 0)
<i>LD</i>	Instrução LD (Load)
<i>ML</i>	Instrução ML (Multiplicação)
<i>OS</i>	Instrução OS (Chamada no sistema operacional)
<i>PD</i>	Instrução PD (Put Data)
<i>RS</i>	Instrução RS (Return from subroutine)
<i>SB</i>	Instrução SB (Subtração)
<i>SC</i>	Instrução SC (Subroutine Call)
<i>ST</i>	Instrução ST (Store)

classmethod `get_mnemonic(opcode: int) → str`

Obtém o mnemônico da instrução a partir de seu opcode

opcode [int] O código da operação

str O mnemônico correspondente

classmethod `get_opcode(mnemonic: str) → int`

Obtém o código da operação (opcode) a partir de seu símbolo

mnemonic [str] Mnemonico do opcode buscado

int Valor do opcode

AD = 4

Instrução AD (Adição)

DV = 7

Instrução DV (Divisão)

GD = 11

Instrução GD (Get Data)

HJ = 3

Instrução HJ (Jump after halt)

JN = 2

Instrução JN (Jump if acc < 0)

JP = 0

Instrução JP (Jump incondicional)

JZ = 1

Instrução JZ (Jump if acc = 0)

LD = 8
Instrução LD (Load)

ML = 6
Instrução ML (Multiplicação)

OS = 13
Instrução OS (Chamada no sistema operacional)

PD = 12
Instrução PD (Put Data)

RS = 0
Instrução RS (Return from subroutine)

SB = 5
Instrução SB (Subtração)

SC = 10
Instrução SC (Subroutine Call)

ST = 9
Instrução ST (Store)

1.1.2 pyvnm.vm.device

Classes

<i>CharScreen</i>	
<i>Device</i>	Interface que representa um hardware periférico de entrada e saída de dados
<i>DeviceBus</i>	Abstração de um bus de dispositivos.
<i>HardDisk</i>	
<i>Keyboard</i>	
<i>Screen</i>	

CharScreen

```
class pyvnm.vm.device.CharScreen
    Base: pyvnm.vm.device.Device
    read() → pyvnm.vm.memory.Word
        Lê dados do dispositivo
        Word Palavra recebida pelo periférico
    write(data: pyvnm.vm.memory.Word)
        Escreve dados no dispositivo
        data: Word Palavra a ser escrita no dispositivo
```

Device

class `pyvnm.vm.device.Device`

Base: `object`

Interface que representa um hardware periférico de entrada e saída de dados

read() → `pyvnm.vm.memory.Word`

Lê dados do dispositivo

Word Palavra recebida pelo periférico

write(data: `pyvnm.vm.memory.Word`)

Escreve dados no dispositivo

data: Word Palavra a ser escrita no dispositivo

DeviceBus

class `pyvnm.vm.device.DeviceBus`

Base: `object`

Abstração de um bus de dispositivos. É possível acessar qualquer um dos dispositivos registrados a partir desta classe

devices: Dict[int, Device] Dicionário usado para mapear todos os dispositivos de entrada/saída

add(code: int, device: `pyvnm.vm.device.Device`)

Adiciona um novo dispositivo no bus

code [int] Código do dispositivo (máx 16bits)

device [Device] Dispositivo a ser adicionado

get(code: int) → `pyvnm.vm.device.Device` | None

Obtém um dispositivo a partir de seu código

code [int] Código do dispositivo a ser retornado

Device | None O dispositivo requisitado, se o código constar no mapa de dispositivo, None caso contrário

remove(code: int)

Remove um dispositivo do bus

code [int] Código do dispositivo a ser removido

HardDisk

class `pyvnm.vm.device.HardDisk`

Base: `pyvnm.vm.device.Device`

read() → `pyvnm.vm.memory.Byte`

Lê dados do dispositivo

Word Palavra recebida pelo periférico

save()

write(*data*: `pyvnm.vm.memory.Word`)
Escreve dados no dispositivo
data: **Word** Palavra a ser escrita no dispositivo

Keyboard

class `pyvnm.vm.device.Keyboard`
Base: `pyvnm.vm.device.Device`
read() → `pyvnm.vm.memory.Word`
Lê dados do dispositivo
Word Palavra recebida pelo periférico
write(*data*: `pyvnm.vm.memory.Word`)
Escreve dados no dispositivo
data: **Word** Palavra a ser escrita no dispositivo

Screen

class `pyvnm.vm.device.Screen`
Base: `pyvnm.vm.device.Device`
read() → `pyvnm.vm.memory.Word`
Lê dados do dispositivo
Word Palavra recebida pelo periférico
write(*data*: `pyvnm.vm.memory.Word`)
Escreve dados no dispositivo
data: **Word** Palavra a ser escrita no dispositivo

1.1.3 `pyvnm.vm.memory`

Classes

<i>Byte</i>	
<i>Memory</i>	Memória de dados e de programa.
<i>Word</i>	Representação de uma palavra na memória e nos registradores.

Byte

class `pyvnm.vm.memory.Byte`

Base: `pyvnm.vm.memory.Word`

Attributes

<code>first_byte</code>	
<code>opcode</code>	Retorna o código da operação da respectiva instrução
<code>operand</code>	Retorna o operando da respectiva instrução
<code>second_byte</code>	
<code>size</code>	
<code>two_complement</code>	
<code>value</code>	Retorna o valor armazenado na palavra na representação inteira

static `convert_to_int(value: str | int) → int`

Converte uma string para um objeto do tipo inteiro. A base é infreira de acordo com a seguinte convensão:

- Números representados na base binária devem começar com os caracteres

`0b`, por exemplo: `0b10101010`; - Números representados na base hexadecimal devem começar com os caracteres `0x`, por exemplo: `0xfa`; - Números sem prefixo serão considerados decimais.

value [str | int] Número a ser representado como inteiro

int Valor inteiro do número indicado após a conversão de base

static `from_instruction(opcode: int, operand: int)`

is_empty() → bool

Varifica se a palavra nunca foi inicializada pelo carregador

bool True se a palavra é vazia (= None), False caso contrário

is_instruction() → bool

to(base: `Literal['x', 'hex', 'b', 'bin', 'd', 'dec', 'int', 'uint']`)

property `first_byte`

property `opcode`

Retorna o código da operação da respectiva instrução

int Representação inteira do código da operação

property `operand`

Retorna o operando da respectiva instrução

int Representação inteira do operando

property second_byte

size = 16

property two_complement: `int`

property value: `int` | `None`

Retorna o valor armazenado na palavra na representação inteira

`int` | `None` O valor inteiro da palavra ou `None`, caso ela nunca tenha sido inicializada

Memory

class `pyvnm.vm.memory.Memory`

Base: `object`

Memória de dados e de programa. Armazena toda informação não-persistente da máquina

size: `int` Número de endereços da memória

_data: `List[Word]` Estrutura de dados usada para armazenar todo conteúdo da memória

Attributes

<code>size</code>	O número de posições da memória
-------------------	---------------------------------

_check_valid_position(*address: int*)

Verifica se um determinado endereço existe na memória

address `[int]` Endereço pesquisado

ValueError Erro jogado caso haja tentativa de acesso à uma posição inválida

hexdump(*colors*)

read(*address: int*) → `pyvnm.vm.memory.Word`

Ler o conteúdo de um endereço específico da memória

address `[int]` Endereço a ser lido

Word Conteúdo contido no endereço especificado

write(*address: int, data: pyvnm.vm.memory.Word*)

Escrever conteúdo em um endereço específica da memória

address `[int]` Endereço a ser escrito

data: `Word` | `List[Word]` Palavra ou lista de palavras a serem escritas na memória

write_byte(*address: int, data: pyvnm.vm.memory.Byte*)

property size: `int`

O número de posições da memória

`int` O número de posições da memória

Word

class pyvnm.vm.memory.Word

Base: `object`

Representação de uma palavra na memória e nos registradores.

size: int Tamanho da palavra em bits

value: str Valor a ser armazenado na palavra

Attributes

<i>first_byte</i>	
<i>opcode</i>	Retorna o código da operação da respectiva instrução
<i>operand</i>	Retorna o operando da respectiva instrução
<i>second_byte</i>	
<i>size</i>	
<i>two_complement</i>	
<i>value</i>	Retorna o valor armazenado na palavra na representação inteira

static `convert_to_int(value: str | int) → int`

Converte uma string para um objeto do tipo inteiro. A base é infreira de acordo com a seguinte convensão:

- Números representados na base binária devem começar com os caracteres

`0b`, por exemplo: `0b10101010`; - Números representados na base hexadecimal devem começar com os caracteres `0x`, por exemplo: `0xfa`; - Números sem prefixo serão considerados decimais.

value [str | int] Número a ser representado como inteiro

int Valor inteiro do número indicado após a conversão de base

static `from_instruction(opcode: int, operand: int)`

`is_empty()` → `bool`

Varifica se a palavra nunca foi inicializada pelo carregador

bool True se a palavra é vazia (= None), False caso contrário

`is_instruction()` → `bool`

`to(base: Literal['x', 'hex', 'b', 'bin', 'd', 'dec', 'int', 'uint'])`

property `first_byte`

property `opcode`

Retorna o código da operação da respectiva instrução

int Representação inteira do código da operação

property operand

Retorna o operando da respectiva instrução

int Representação inteira do operando

property second_byte

size = 16

property two_complement: int

property value: int | None

Retorna o valor armazenado na palavra na representação inteira

int | None O valor inteiro da palavra ou None, caso ela nunca tenha sido inicializada

1.1.4 pyvnm.vm.utils

Classes

<i>Lock</i>	Representação de um cadeado para implementação da lógica de bloqueio de um determinado recurso
-------------	--

Lock

class pyvnm.vm.utils.Lock

Base: **object**

Representação de um cadeado para implementação da lógica de bloqueio de um determinado recurso

_locked: bool Representa o estado travado/destravado

acquire()

Trava o cadeado

is_locked() → bool

Verifica se o cadeado está travado

bool True se o cadeado estiver travado, False caso contrário.

release()

Destrava o cadeado

1.1.5 pyvnm.vm.vnm

Classes

<i>VonNeumannMachine</i>	O mais alto nível de abstração da Máquina de Von Neumann, todas as operações de entrada/saída, a manipulação de dados, as mudanças de estado e o carregamento de dispositivos estão abstraídos.
--------------------------	---

VonNeumannMachine

class pyvnm.vm.vnm.VonNeumannMachine

Base: `object`

O mais alto nível de abstração da Máquina de Von Neumann, todas as operações de entrada/saída, a manipulação de dados, as mudanças de estado e o carregamento de dispositivos estão abstraídos.

Esta classe permite a criação de uma máquina de Von Neumann, bem como a carga e descarga de conteúdo em sua memória e execução de programas sem necessidade de manipulação de componentes de hardware

memory_size: int Quantidade de posições de memória da Máquina criada

initial_pc: int Valor inicial do contador de programa

boot()

Carregamento inicial dos principais programas de sistema na máquina

dump() → `str`

Aciona o Dumper para a persistência dos dados carregados na memória para outra mídia

output_path: str | Path Caminho para onde o arquivo será salvo

output_base: str Base numérica do arquivo a ser descarregado. Valores usados: 'x' para hexadecimal e 'b' para binário. Valor padrão: 'x'

str Valor da memória codificado na base especificada

execute_program()

Aciona a Unidade de Controle para o início da execução do programa

load()

1.2 pyvnm.system

Modules

pyvnm.system.assembler

pyvnm.system.bootloader

pyvnm.system.os

1.2.1 pyvnm.system.assembler

Classes

<i>Assembler</i>	
<i>AssemblerInstructions</i>	
<i>LineTokens</i>	LineTokens(label: str = None, mnemonic: str = None, operand: str = None, index: int = None, mem_addr: int = None, empty: bool = None)

Assembler

```
class pyvnm.system.assembler.Assembler
    Base: object
    _check_labels()
    _check_mnemonics()
    _check_operands()
    _compute_checksum(program: str) → pyvnm.vm.memory.Byte
    _decode_operands(lbl_tokens: List[pyvnm.system.assembler.LineTokens], inst_tokens:
        List[pyvnm.system.assembler.LineTokens])
    _tokenize() → Tuple[List[pyvnm.system.assembler.LineTokens], List[pyvnm.system.assembler.LineTokens]]
    _tokenize_line(line: str, line_index: int, memory_offset: int) → pyvnm.system.assembler.LineTokens
    assemble() → str
```

AssemblerInstructions

```
class pyvnm.system.assembler.AssemblerInstructions
    Base: object
```

Attributes

<i>AREA</i>
<i>DATA</i>
<i>END</i>
<i>ORG</i>
AREA = 'AREA'
DATA = 'DATA'

END = 'END'

ORG = 'ORG'

LineTokens

class pyvnm.system.assembler.LineTokens

Base: `object`

LineTokens(label: str = None, mnemonic: str = None, operand: str = None, index: int = None, mem_addr: int = None, empty: bool = None)

Attributes

empty

index

label

mem_addr

mnemonic

operand

empty: `bool` = None

index: `int` = None

label: `str` = None

mem_addr: `int` = None

mnemonic: `str` = None

operand: `str` = None

1.2.2 pyvnm.system.bootloader

Classes

BootLoader

Carregador do binário e hexadecimal.

BootLoader

class pyvnm.system.bootloader.BootLoader

Base: `object`

Carregador do binário e hexadecimal. Efetua o carregamento do conteúdo de um arquivo (usualmente, código + dados) na memória da máquina

initial_state: `MachineState` O estado a da máquina a ser modificado com o carregamento dos dados na memória

input_base: `str`, **optional** Base numérica que em que se encontra o programa objeto a ser carregado. Valores usados: 'x' para hexadecimal e 'b' para binário. Valor padrão: 'x'

_to_byte(*value*) → `pyvnm.vm.memory.Byte`

_to_word(*value*) → `pyvnm.vm.memory.Word`

load(*program_obj*: `str` | `pathlib.Path`) → `int`

Carrega os principais programas de sistema na memória

program_obj [`str` | `Path`] Caminho ou conteúdo do programa objeto a ser carregado

int O endereço da memória onde o programa foi carregado

1.2.3 pyvnm.system.os

Classes

OS

OS

class pyvnm.system.os.OS

Base: `object`

Attributes

LOADER_CHECKSUM_MISMATCH

SIG_TERM

SIG_TRAP

flags

classmethod codes()

LOADER_CHECKSUM_MISMATCH = 400

```
SIG_TERM = 15  
SIG_TRAP = 5  
flags = {}
```


p

- `pyvnm.system`, 13
- `pyvnm.system.assembler`, 13
- `pyvnm.system.bootloader`, 15
- `pyvnm.system.os`, 16
- `pyvnm.vm`, 1
- `pyvnm.vm.cpu`, 2
- `pyvnm.vm.device`, 6
- `pyvnm.vm.memory`, 8
- `pyvnm.vm.utils`, 12
- `pyvnm.vm.vnm`, 12

Símbolos

_action_AD() (método *pyvnm.vm.cpu.CPU*), 2
 _action_DV() (método *pyvnm.vm.cpu.CPU*), 2
 _action_GD() (método *pyvnm.vm.cpu.CPU*), 2
 _action_HJ() (método *pyvnm.vm.cpu.CPU*), 2
 _action_JN() (método *pyvnm.vm.cpu.CPU*), 2
 _action_JP() (método *pyvnm.vm.cpu.CPU*), 2
 _action_JZ() (método *pyvnm.vm.cpu.CPU*), 2
 _action_LD() (método *pyvnm.vm.cpu.CPU*), 2
 _action_ML() (método *pyvnm.vm.cpu.CPU*), 3
 _action_OS() (método *pyvnm.vm.cpu.CPU*), 3
 _action_PD() (método *pyvnm.vm.cpu.CPU*), 3
 _action_RS() (método *pyvnm.vm.cpu.CPU*), 3
 _action_SB() (método *pyvnm.vm.cpu.CPU*), 3
 _action_SC() (método *pyvnm.vm.cpu.CPU*), 3
 _action_ST() (método *pyvnm.vm.cpu.CPU*), 3
 _check_labels() (método *pyvnm.system.assembler.Assembler*), 14
 _check_mnemonics() (método *pyvnm.system.assembler.Assembler*), 14
 _check_operands() (método *pyvnm.system.assembler.Assembler*), 14
 _check_valid_position() (método *pyvnm.vm.memory.Memory*), 10
 _compute_checksum() (método *pyvnm.system.assembler.Assembler*), 14
 _decode_operands() (método *pyvnm.system.assembler.Assembler*), 14
 _increment_pc() (método *pyvnm.vm.cpu.CPU*), 3
 _to_byte() (método *pyvnm.system.bootloader.BootLoader*), 16
 _to_word() (método *pyvnm.system.bootloader.BootLoader*), 16
 _tokenize() (método *pyvnm.system.assembler.Assembler*), 14
 _tokenize_line() (método *pyvnm.system.assembler.Assembler*), 14

A

AD (atributo *pyvnm.vm.cpu.InstructionSet*), 5
 add() (método *pyvnm.vm.device.DeviceBus*), 7
 aquire() (método *pyvnm.vm.utils.Lock*), 12
 AREA (atributo *pyvnm.system.assembler.AssemblerInstructions*), 14
 assemble() (método *pyvnm.system.assembler.Assembler*), 14
 Assembler (clase en *pyvnm.system.assembler*), 14
 AssemblerInstructions (clase en *pyvnm.system.assembler*), 14

B

boot() (método *pyvnm.vm.vnm.VonNeumannMachine*), 13
 BootLoader (clase en *pyvnm.system.bootloader*), 16
 Byte (clase en *pyvnm.vm.memory*), 9

C

CharScreen (clase en *pyvnm.vm.device*), 6
 codes() (método de clase *pyvnm.system.os.OS*), 16
 convert_to_int() (método estático *pyvnm.vm.memory.Byte*), 9
 convert_to_int() (método estático *pyvnm.vm.memory.Word*), 11
 CPU (clase en *pyvnm.vm.cpu*), 2
 CPUCallback (clase en *pyvnm.vm.cpu*), 3
 CPUState (clase en *pyvnm.vm.cpu*), 4

D

DATA (atributo *pyvnm.system.assembler.AssemblerInstructions*), 14
 Device (clase en *pyvnm.vm.device*), 7
 DeviceBus (clase en *pyvnm.vm.device*), 7
 dump() (método *pyvnm.vm.vnm.VonNeumannMachine*), 13
 DV (atributo *pyvnm.vm.cpu.InstructionSet*), 5

E

`empty` (atributo `pyvnm.system.assembler.LineTokens`), 15
`END` (atributo `pyvnm.system.assembler.AssemblerInstructions`), 14
`event_loop()` (método `pyvnm.vm.cpu.CPU`), 3
`execute_program()` (método `pyvnm.vm.vnm.VonNeumannMachine`), 13

F

`first_byte` (propriedade `pyvnm.vm.memory.Byte`), 9
`first_byte` (propriedade `pyvnm.vm.memory.Word`), 11
`flags` (atributo `pyvnm.system.os.OS`), 17
`from_instruction()` (método `pyvnm.vm.memory.Byte`), 9
`from_instruction()` (método `pyvnm.vm.memory.Word`), 11

G

`GD` (atributo `pyvnm.vm.cpu.InstructionSet`), 5
`get()` (método `pyvnm.vm.device.DeviceBus`), 7
`get_mnemonic()` (método `pyvnm.vm.cpu.InstructionSet`), 5
`get_opcode()` (método `pyvnm.vm.cpu.InstructionSet`), 5

H

`HardDisk` (classe em `pyvnm.vm.device`), 7
`hexdump()` (método `pyvnm.vm.memory.Memory`), 10
`HJ` (atributo `pyvnm.vm.cpu.InstructionSet`), 5

I

`index` (atributo `pyvnm.system.assembler.LineTokens`), 15
`InstructionSet` (classe em `pyvnm.vm.cpu`), 4
`is_empty()` (método `pyvnm.vm.memory.Byte`), 9
`is_empty()` (método `pyvnm.vm.memory.Word`), 11
`is_instruction()` (método `pyvnm.vm.memory.Byte`), 9
`is_instruction()` (método `pyvnm.vm.memory.Word`), 11
`is_locked()` (método `pyvnm.vm.utils.Lock`), 12

J

`JN` (atributo `pyvnm.vm.cpu.InstructionSet`), 5
`JP` (atributo `pyvnm.vm.cpu.InstructionSet`), 5
`JZ` (atributo `pyvnm.vm.cpu.InstructionSet`), 5

K

`Keyboard` (classe em `pyvnm.vm.device`), 8

L

`label` (atributo `pyvnm.system.assembler.LineTokens`), 15
`LD` (atributo `pyvnm.vm.cpu.InstructionSet`), 5
`LineTokens` (classe em `pyvnm.system.assembler`), 15

`load()` (método `pyvnm.system.bootloader.BootLoader`), 16
`load()` (método `pyvnm.vm.vnm.VonNeumannMachine`), 13
`LOADER_CHECKSUM_MISMATCH` (atributo `pyvnm.system.os.OS`), 16
`Lock` (classe em `pyvnm.vm.utils`), 12

M

`mem_addr` (atributo `pyvnm.system.assembler.LineTokens`), 15
`Memory` (classe em `pyvnm.vm.memory`), 10
`ML` (atributo `pyvnm.vm.cpu.InstructionSet`), 6
`mnemonic` (atributo `pyvnm.system.assembler.LineTokens`), 15
módulo
 `pyvnm.system`, 13
 `pyvnm.system.assembler`, 13
 `pyvnm.system.bootloader`, 15
 `pyvnm.system.os`, 16
 `pyvnm.vm`, 1
 `pyvnm.vm.cpu`, 2
 `pyvnm.vm.device`, 6
 `pyvnm.vm.memory`, 8
 `pyvnm.vm.utils`, 12
 `pyvnm.vm.vnm`, 12

O

`on_event_loop_begin()` (método `pyvnm.vm.cpu.CPUCallback`), 3
`on_event_loop_end()` (método `pyvnm.vm.cpu.CPUCallback`), 3
`on_instruction_begin()` (método `pyvnm.vm.cpu.CPUCallback`), 4
`on_instruction_end()` (método `pyvnm.vm.cpu.CPUCallback`), 4
`opcode` (propriedade `pyvnm.vm.memory.Byte`), 9
`opcode` (propriedade `pyvnm.vm.memory.Word`), 11
`operand` (atributo `pyvnm.system.assembler.LineTokens`), 15
`operand` (propriedade `pyvnm.vm.memory.Byte`), 9
`operand` (propriedade `pyvnm.vm.memory.Word`), 11
`ORG` (atributo `pyvnm.system.assembler.AssemblerInstructions`), 15
`OS` (atributo `pyvnm.vm.cpu.InstructionSet`), 6
`OS` (classe em `pyvnm.system.os`), 16

P

`PD` (atributo `pyvnm.vm.cpu.InstructionSet`), 6
`pyvnm.system`
 módulo, 13
`pyvnm.system.assembler`
 módulo, 13
`pyvnm.system.bootloader`

módulo, 15
 pyvnm.system.os
 módulo, 16
 pyvnm.vm
 módulo, 1
 pyvnm.vm.cpu
 módulo, 2
 pyvnm.vm.device
 módulo, 6
 pyvnm.vm.memory
 módulo, 8
 pyvnm.vm.utils
 módulo, 12
 pyvnm.vm.vnm
 módulo, 12

R

read() (método pyvnm.vm.device.CharScreen), 6
 read() (método pyvnm.vm.device.Device), 7
 read() (método pyvnm.vm.device.HardDisk), 7
 read() (método pyvnm.vm.device.Keyboard), 8
 read() (método pyvnm.vm.device.Screen), 8
 read() (método pyvnm.vm.memory.Memory), 10
 release() (método pyvnm.vm.utils.Lock), 12
 remove() (método pyvnm.vm.device.DeviceBus), 7
 RS (atributo pyvnm.vm.cpu.InstructionSet), 6

S

save() (método pyvnm.vm.device.HardDisk), 7
 SB (atributo pyvnm.vm.cpu.InstructionSet), 6
 SC (atributo pyvnm.vm.cpu.InstructionSet), 6
 Screen (classe em pyvnm.vm.device), 8
 second_byte (propriedade pyvnm.vm.memory.Byte), 9
 second_byte (propriedade pyvnm.vm.memory.Word),
 12
 SIG_TERM (atributo pyvnm.system.os.OS), 16
 SIG_TRAP (atributo pyvnm.system.os.OS), 17
 size (atributo pyvnm.vm.memory.Byte), 10
 size (atributo pyvnm.vm.memory.Word), 12
 size (propriedade pyvnm.vm.memory.Memory), 10
 ST (atributo pyvnm.vm.cpu.InstructionSet), 6

T

to() (método pyvnm.vm.memory.Byte), 9
 to() (método pyvnm.vm.memory.Word), 11
 two_complement (propriedade pyvnm.vm.memory.Byte
), 10
 two_complement (propriedade pyvnm.vm.memory.Word
), 12

V

value (propriedade pyvnm.vm.memory.Byte), 10
 value (propriedade pyvnm.vm.memory.Word), 12

VonNeumannMachine (classe em pyvnm.vm.vnm), 13

W

Word (classe em pyvnm.vm.memory), 11
 write() (método pyvnm.vm.device.CharScreen), 6
 write() (método pyvnm.vm.device.Device), 7
 write() (método pyvnm.vm.device.HardDisk), 7
 write() (método pyvnm.vm.device.Keyboard), 8
 write() (método pyvnm.vm.device.Screen), 8
 write() (método pyvnm.vm.memory.Memory), 10
 write_byte() (método pyvnm.vm.memory.Memory), 10