

Programação JHIPO

Prof. Ms. Peter Jandl Junior

Arquitetura e Organização de Computadores

Análise e Desenvolvimento de Sistemas

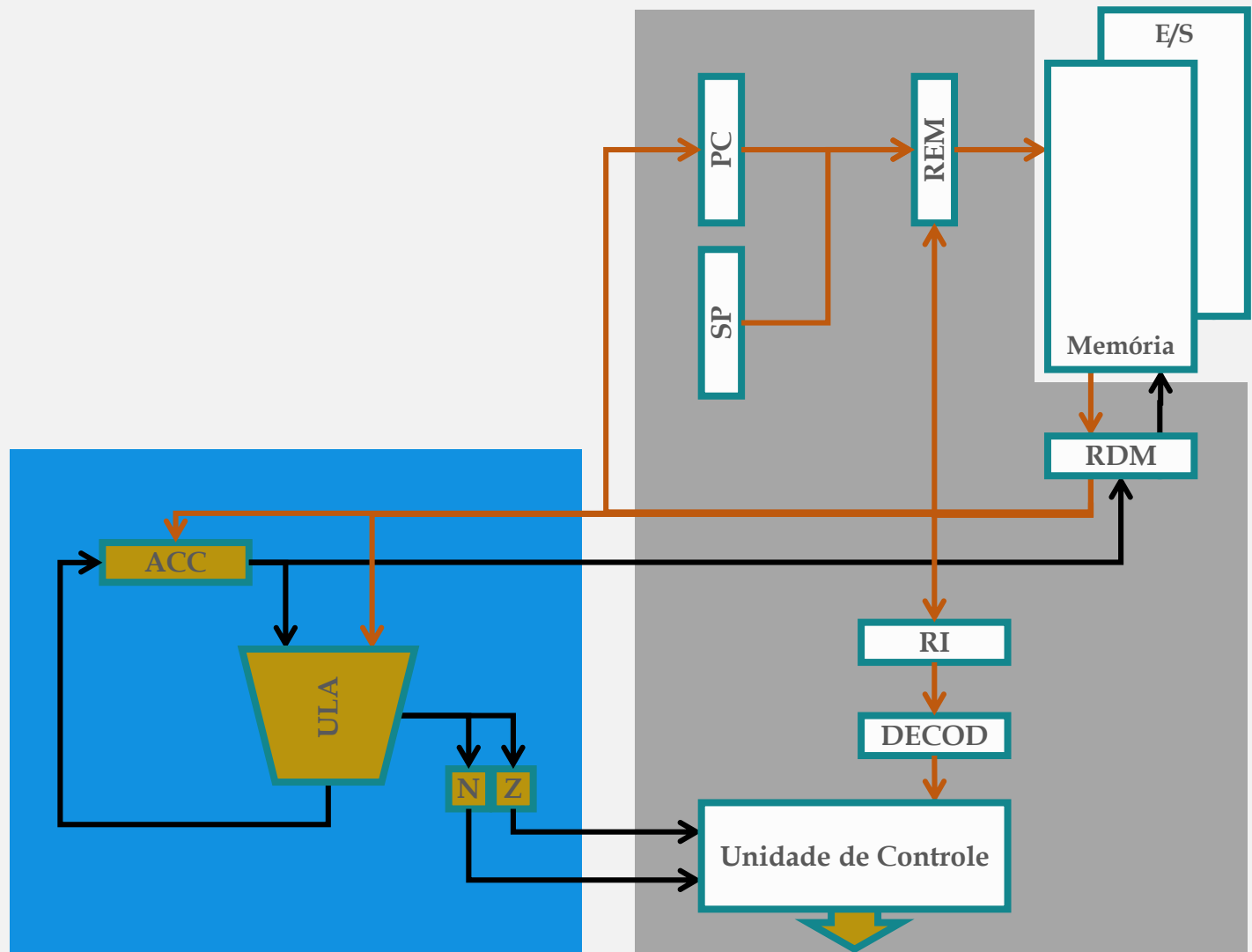
FATEC – Jundiaí

Computador hipotético HIPO

Computador Hipotético

- Barramento de dados de 8 bits
- Barramento de endereços de 8 bits
 $2^8 = 256$ posições de 8 bits (=1 byte) de memória
- PC (*Program Counter*), SP (*Stack Pointer*), RI (Reg. Instruções), RDM (Reg. Dados Memória), REM (Reg. End. Memória) e ACC (Acumulador) [todos de 8 bits].
- Duas *flags* de estado (códigos de condição): N (negativo) e Z (zero)

Computador Hipotético



Computador Hipotético:: Conjunto de Instruções

Código hexadecimal da instrução.

OpCode	Instrução	Bytes	Comentário Ação da instrução
0x00	NOP	1	Nenhuma Operação
0x10	STA end	2	$\text{MEM}[\text{end}] \leftarrow \text{ACC}$
0x20	LDA end	2	$\text{ACC} \leftarrow \text{MEM}[\text{end}]$
0x30	ADD end	2	$\text{ACC} \leftarrow \text{ACC} + \text{MEM}[\text{end}]$
0x40	SUB end	2	$\text{ACC} \leftarrow \text{ACC} - \text{MEM}[\text{end}]$
0x50	OR end	2	$\text{ACC} \leftarrow \text{ACC OR MEM}[\text{end}]$
0x60	AND end	2	$\text{ACC} \leftarrow \text{ACC AND MEM}[\text{end}]$
0x70	NOT	1	$\text{ACC} \leftarrow \text{NOT (ACC)}$
0x80	JMP end	2	$\text{PC} \leftarrow \text{end}$
0x90	JN end	2	IF N = 1 THEN $\text{PC} \leftarrow \text{end}$
0xA0	JZ end	2	IF Z = 1 THEN $\text{PC} \leftarrow \text{end}$
0xB0	CALL end	2	$\text{MEM}[\text{SP}] \leftarrow \text{PC}; \text{SP} \leftarrow \text{SP}-1; \text{PC} \leftarrow \text{end}$
0xC0	RET	1	$\text{PC} \leftarrow \text{MEM}[\text{SP}]; \text{SP} \leftarrow \text{SP}+1$
0xD0	IN end	2	$\text{ACC} \leftarrow \text{I/O}[\text{end}]$
0xE0	OUT end	2	$\text{I/O}[\text{end}] \leftarrow \text{ACC}$
0xF0	HLT	1	Término de execução

Mnemônicos das instruções.

Computador Hipotético:: Conjunto de Instruções

- A palavra *end* significa *endereço direto*.
- ACC é o registrador acumulador.
- MEM[*end*] significa o conteúdo da posição *end* da memória.
- Formato das instruções:
 - Possuem 1 ou 2 bytes.



Os 4 bits mais significativos contém o **OpCode** (código da operação ou da instrução).

É por isso que todo **OpCode** (código da instrução) acaba com valor zero.



JHIPOVM

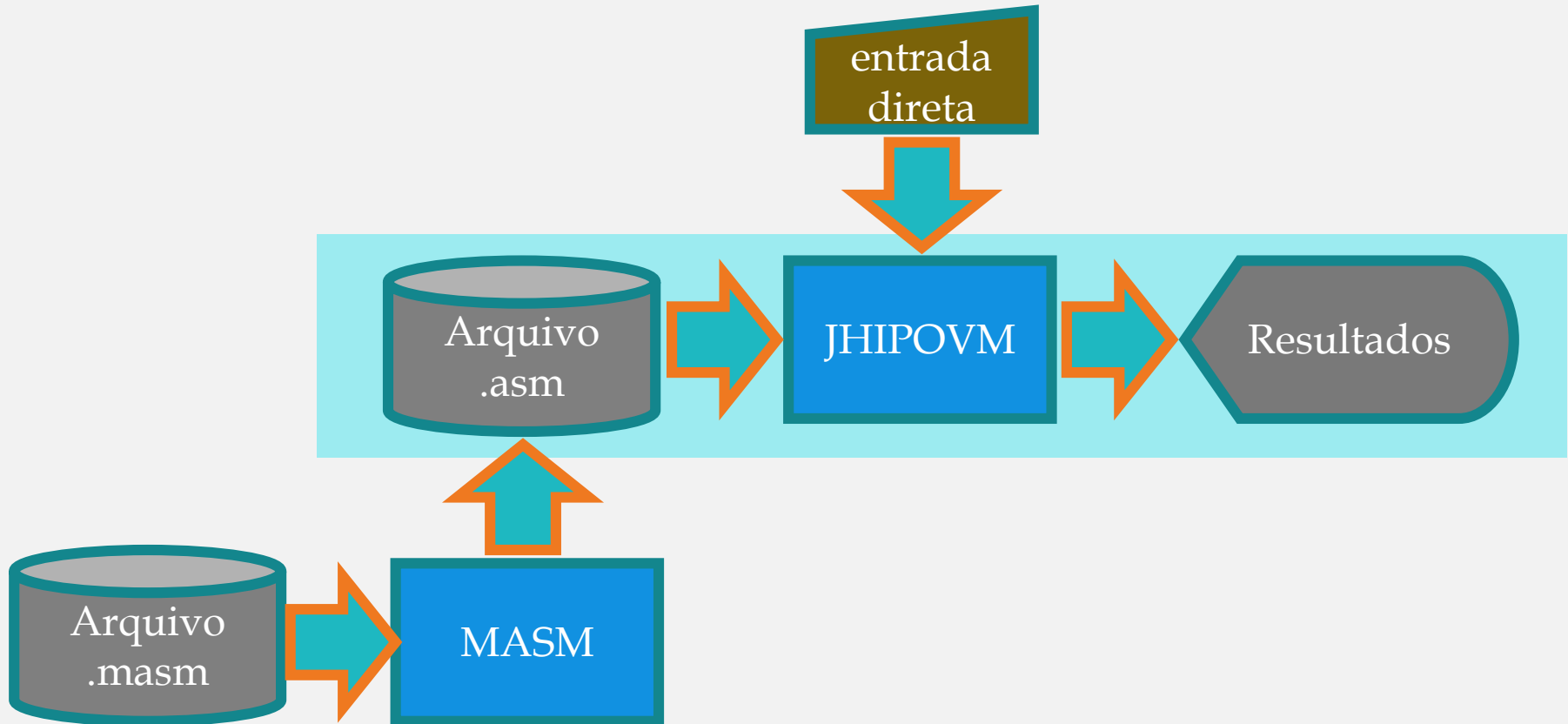
::A máquina virtual do HIPO

Um sistema de simulação para o HIPO desenvolvido em Java.

JHIPOVM

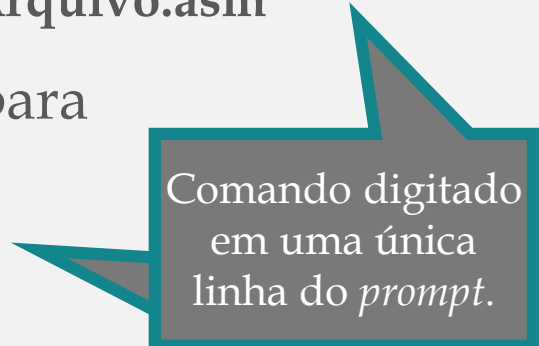
- Composta de simulador da arquitetura e organização do HIPO que permite:
 - Executar programas escritos na linguagem de máquina do HIPO.
 - Acompanhar a execução do programa, incluindo:
 - disposição do código em memória;
 - execução das microinstruções;
 - dispositivos conectados ao sistema.
- Também se inclui um *macro-assembler* que permite a construção de programas em linguagem de máquina.

JHIPOVM



Instruções para uso do JHIPOVM

1. Efetue o download da última versão:
jhipo-vaaaammdd.jar.
2. Mantenha os arquivos **.asm** no mesmo diretório que o arquivo **jar** ou em um subdiretório direto.
3. Em um prompt de comandos, navegue até o diretório onde está o arquivo **jar**.
4. Execute:
 - `java -cp jhipo-vaaaaddmm.jar jhipovm Arquivo.asm`
5. Verifique outras opções disponíveis para uso do JHIPOVM com:
 - `java -cp jhipo-vaaaaddmm.jar jhipovm ?`



Comando digitado em uma única linha do *prompt*.

JHIPOVM::opções

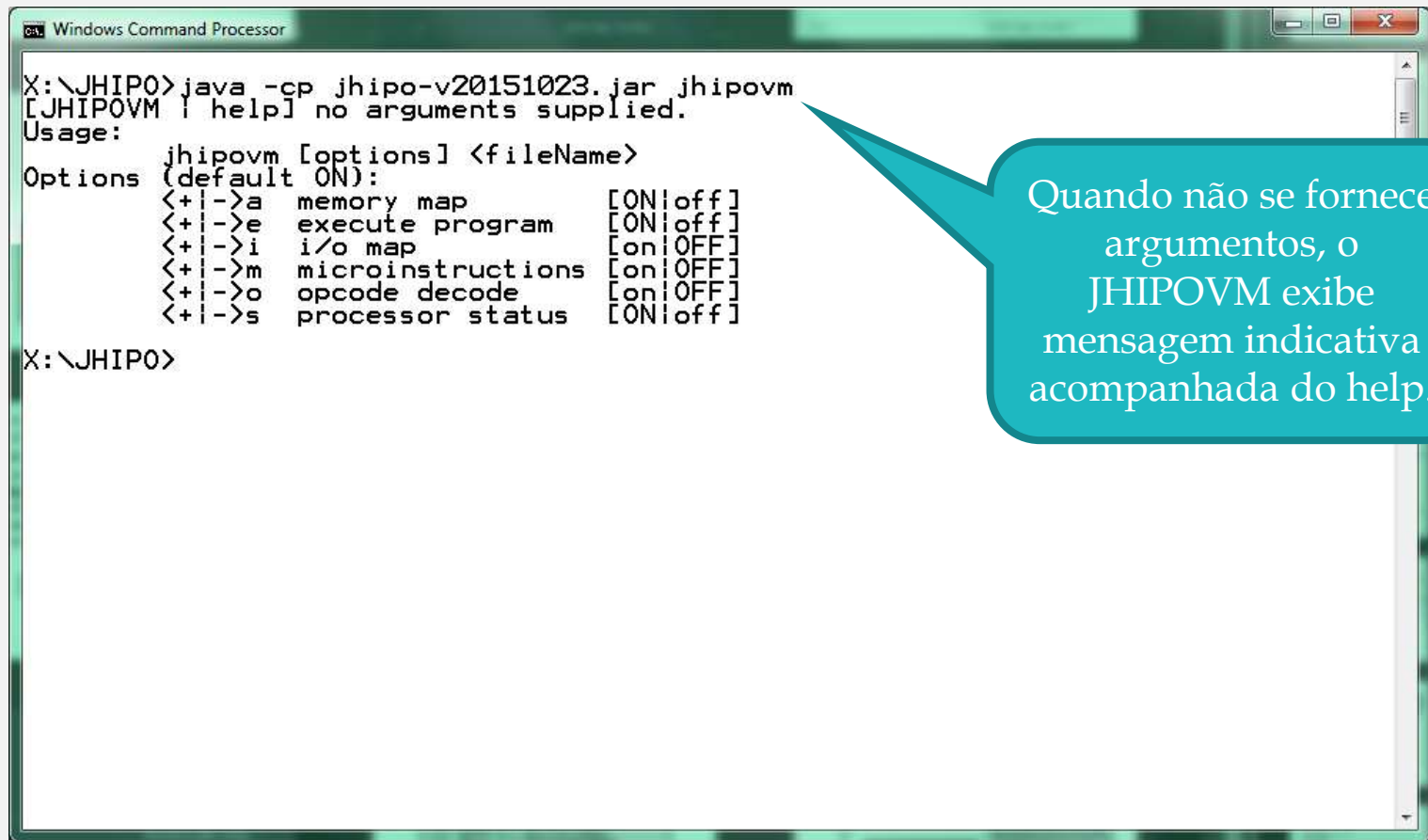
O arquivo jhipo-v20151023.jar foi copiado no subdiretório X:\JHIPO.

```
Windows Command Processor
X:\JHIPO>java -cp jhipo-v20151023.jar jhipovm ?
Usage:
    jhipovm [options] <fileName>
Options (default ON):
    <+|->a  memory map          [ON|off]
    <+|->e  execute program     [ON|off]
    <+|->i  i/o map             [on|OFF]
    <+|->m  microinstructions   [on|OFF]
    <+|->o  opcode decode       [on|OFF]
    <+|->s  processor status    [ON|off]
X:\JHIPO>
```

As opções padrão são aquelas com ON:

- mostra mapa de memória
- executa programa
- exhibe status do processador.

JHIPOVM::mensagens e help



```
Windows Command Processor

X:\JHIPO>java -cp jhipo-v20151023.jar jhipovm
[JHIPOVM | help] no arguments supplied.
Usage:
    jhipovm [options] <fileName>
Options (default ON):
    <+>->a memory map          [ON|off]
    <+>->e execute program      [ON|off]
    <+>->i i/o map              [on|OFF]
    <+>->m microinstructions     [on|OFF]
    <+>->o opcode decode         [on|OFF]
    <+>->s processor status      [ON|off]

X:\JHIPO>
```

Quando não se fornece argumentos, o JHIPOVM exibe mensagem indicativa acompanhada do help.

JHIPOVM::execução de programa

```
Windows Command Processor - java -cp jhipo-v20151023.jar jhipovm programs\Exemplo00.asm
X:\JHIPO>java -cp jhipo-v20151023.jar jhipovm programs\Exemplo00.asm
[JHIPOVM] warning: io configuration file not found, using default device.

[Memory | Map]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[Processor | Status]
PC : 00 | REM : 00
SP : FF | RDM : 00 | N:0
RI : 00 | ACC : 00 | Z:0

[Processor] started
[Processor] stopped

[Memory | Map]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Quando se fornece um nome de arquivo assembly (extensão .asm) como argumento, tal programa é executado.

Quando não existe, o arquivo de configuração `jhipovm-config.xml` é automaticamente criado.

JHIPOVM:jhipovm-config.xml

- Arquivo de configuração da máquina virtual.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
```

```
<properties>
```

```
  <comment>JHIPOVM I/O device list [auto-generated]</comment>
```

```
  <entry key="device0">jandl.aoc.jhipo.SimpleConsole</entry>
```

```
  <entry key="address0">10</entry>
```

```
</properties>
```

- Contém definição dos dispositivos de I/O conectados ao sistema.
- Dispositivo default (número 0):
 - console simples no endereço 0x10.

Pares
deviceN/addressN
que contêm nome da
classe e endereço do
dispositivo.

JHIPOVM:jhipovm-config.xml

- Arquivo de configuração da máquina virtual com dispositivos extras.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<!-- JHIPO Virtual Machine configuration file -->
<properties>
  <comment>JHIPO I/O device list<[edited]/comment>
  <!-- Default I/O device -->
  <entry key="device0">jandl.aoc.jhipo.SimpleConsole</entry>
  <entry key="address0">10</entry>

  <!-- Extra I/O device -->
  <entry key="device1">jandl.aoc.jhipo.SimpleConsole</entry>
  <entry key="address1">20</entry>

  <!-- Extra I/O device -->
  <entry key="device2">jandl.RedSingleLineHexaDisplay</entry>
  <entry key="address2">FF</entry>
</properties>
```

Cada par deve indicar um endereço diferente para o dispositivo.

JHIPOVM::execução de programa

Mapa de memória antes da execução

Status inicial do processador

Informação de execução

Mapa de memória depois da execução

Status final do processador

```
Windows Command Processor - java -cp jhipo-v20151023.jar jhipovm programs\Exemplo00.asm
X:\JHIPO>java -cp jhipo-v20151023.jar jhipovm programs\Exemplo00.asm
[JHIPOVM | warning] io configuration file not found, using default device.

[Memory | Map]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[Processor | Status]
PC : 00 | REM : 00
SP : FF | RDM : 00 | N:0
RI : 00 | ACC : 00 | Z:0

[Processor] started
[Processor] stopped

[Memory | Map]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[Processor | Status]
PC : 06 | REM : 05
SP : FF | RDM : FF | N:0
RI : FF | ACC : 00 | Z:0

<Press ENTER to shutdown JHIPOVM>
```

Pressione ENTER para finalizar a sessão de simulação!

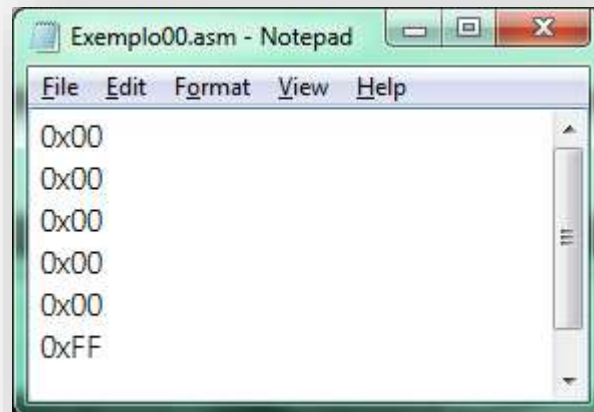
JHIPOVM::execução de programa

- O mapa de memória reflete a sequência de instruções do programa (arquivo **.asm** carregado).

- No caso:

- NOP
- NOP
- NOP
- NOP
- NOP
- HLT

```
[Memory | Map]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



JHIPOVM::execução de programa

```
Windows Command Processor - java -cp jhipo-v20151023.jar jhipovm +o -a programs\Exemplo00.asm

X:\JHIP0>java -cp jhipo-v20151023.jar jhipovm +o -a programs\Exemplo00.asm

[Processor | Status]
PC : 00 | REM : 00
SP : FF | RDM : 00 | N:0
RI : 00 | ACC : 00 | Z:0

[Processor] started
[Processor] decode: NOP | 0b0000 | sb
[Processor] decode: NOP | 0b0000 | sb
[Processor] decode: NOP | 0b0000 | sb
[Processor] decode: NOP | 0b0000 | sb
[Processor] decode: NOP | 0b0000 | sb
[Processor] decode: HLT | 0b1111 | sb
[Processor] stopped

[Processor | Status]
PC : 06 | REM : 05
SP : FF | RDM : FF | N:0
RI : FF | ACC : 00 | Z:0

<Press ENTER to shutdown JHIPOVM>
```

Status inicial do processador

Sem mapa de memória devido ao uso da opção -a

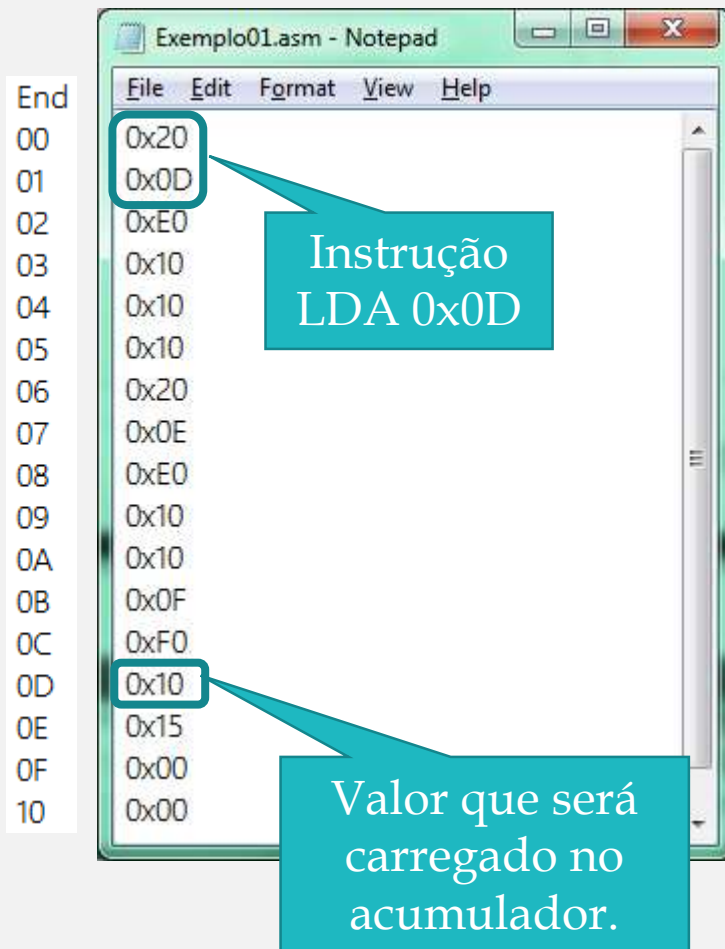
Instruções executadas exibidas devido ao uso da opção +o

Status final do processador

Exemplos

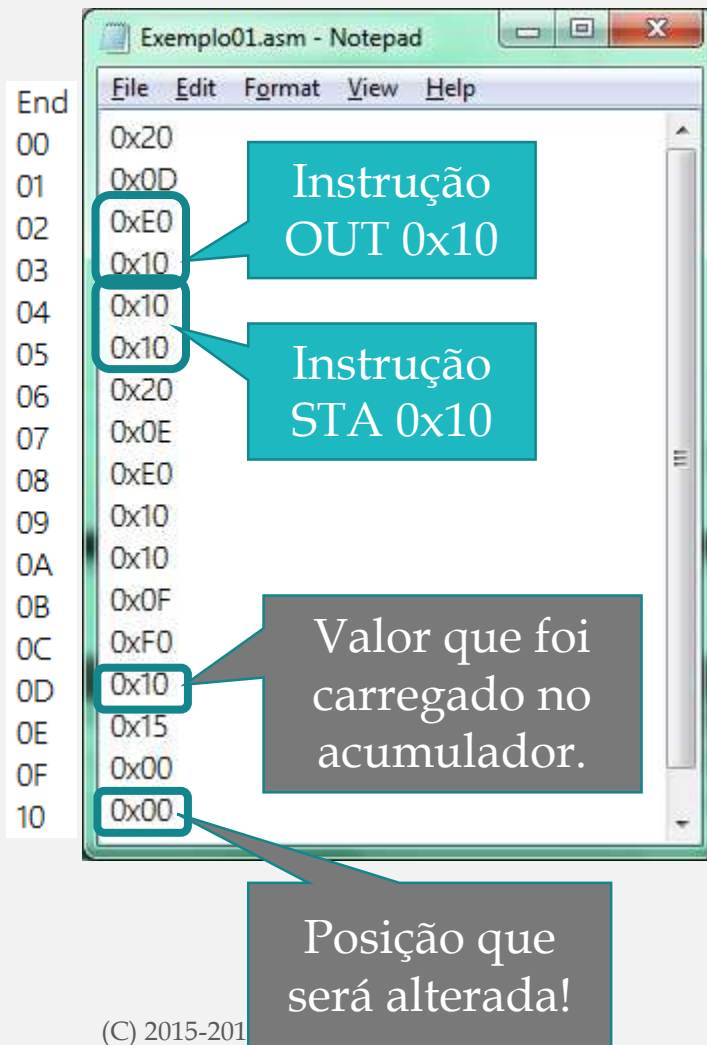
Programas que mostram o uso das diversas instruções do HIPO.

Exemplo01



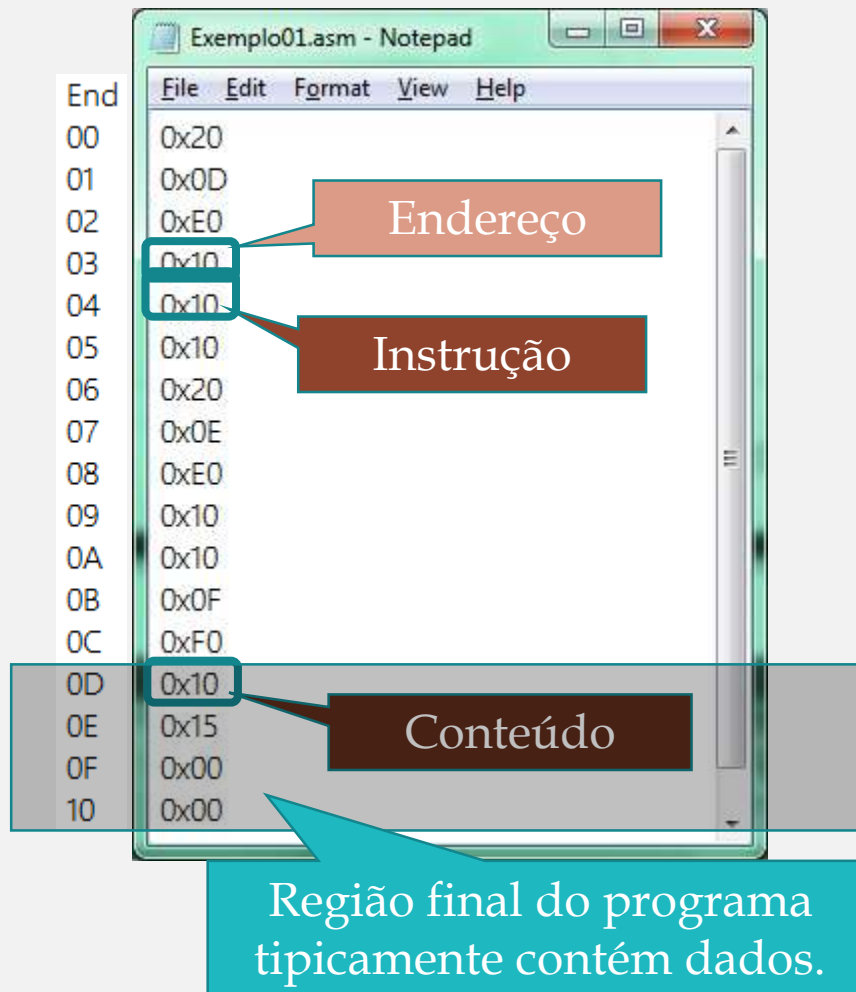
- Exemplo mostra o uso de LDA, OUT e STA.
- Código:
 - 0x20
 - 0x0D
- Corresponde à:
 - LDA 0x0D
 - Ou carrega acumulador com conteúdo de 0x0D.

Exemplo01



- Código:
 - 0xE0
 - 0x10
- Corresponde à:
 - OUT 0x10
 - Ou escreve conteúdo do acumulador no I/O 0x10.
- Código:
 - 0x10
 - 0x10
- Corresponde à:
 - STA 0x10
 - Ou escreve conteúdo do acumulador no endereço de memória 0x10.

Exemplo01



- Observe ainda que o valor **0x10**, ao longo programa, corresponde à:
 - um endereço;
 - uma instrução; e
 - um conteúdo de memória.
- *A correta interpretação do valor depende da situação em que o processador se encontra ao utilizar tal posição de memória, e não de seu valor!*

Exemplo01:: execução

```
Windows Command Processor - java -cp jhipo-v20150521.jar jhipovm +o programs\Exemplo01.asm
X:\JHIP0>java -cp jhipo-v20150521.jar jhipovm +o programs\Exemplo01.asm

[Memory | Map]
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
15 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[Processor | Status]
PC : 00 | REM : 00
SP : FF | RDM : 00 | N:0
RI : 00 | ACC : 00 | Z:0

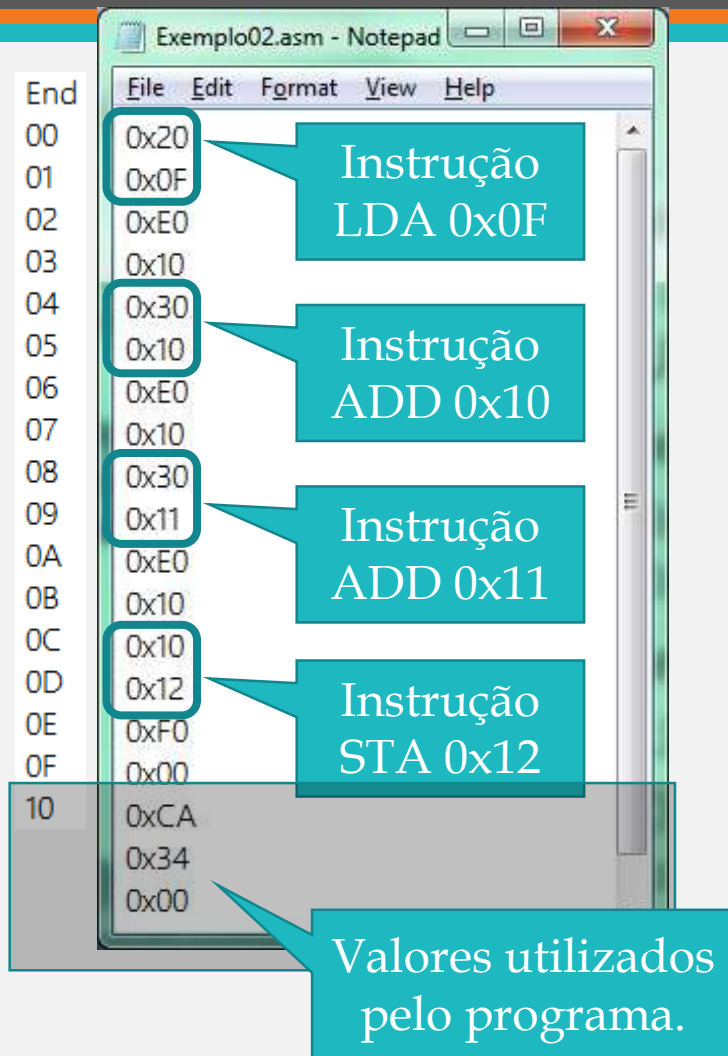
[Processor] started
[Processor] decode: LDA | 0b0010 | mb
[Processor] decode: OUT | 0b1110 | mb
I0[0x10] jandl.aoc.jhipo.SimpleConsole | write]
data: 10
[Processor] decode: STA | 0b0001 | mb
[Processor] decode: LDA | 0b0010 | mb
[Processor] decode: OUT | 0b1110 | mb
I0[0x10] jandl.aoc.jhipo.SimpleConsole | write]
data: 15
[Processor] decode: STA | 0b0001 | mb
[Processor] decode: HLT | 0b1111 | sb
[Processor] stopped

[Memory | Map]
20 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
15 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
15 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[Processor | Status]
PC : 0D | REM : 0C
SP : FF | RDM : F0 | N:0
RI : F0 | ACC : 15 | Z:0
```

Saída de dados
no console 0x10.

Exemplo02



- Exemplo mostra o uso de LDA, ADD, STA e OUT.
- Código:
 - 0x30
 - 0x0D
- Corresponde à:
 - ADD 0x0D
 - Ou soma conteúdo de 0x0D ao acumulador.

Exemplo02:: execução

```
Windows Command Processor - java -cp jhipo-v20150521.jar jhipovm +o programs\Exemplo02.asm
X:\JHIPO>java -cp jhipo-v20150521.jar jhipovm +o programs\Exemplo02.asm

[Memory | Map]
20 CA 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0F 34 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00
11 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
12 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[Processor | Status]
PC : 00 | REM : 00
SP : FF | RDM : 00 | N:0
RI : 00 | ACC : 00 | Z:0

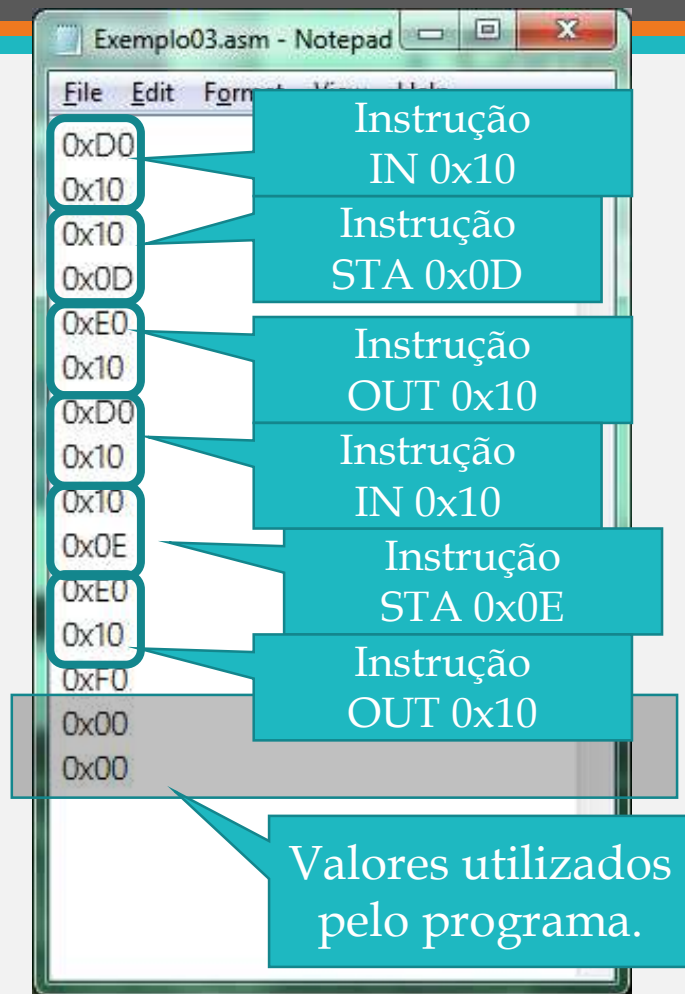
[Processor] started
[Processor] decode: LDA | 0b0010 | mb
[Processor] decode: OUT | 0b1110 | mb
I/O 0x10 | jandl.aoc.jhipo.SimpleConsole | write]
data: 00
[Processor] decode: ADD | 0b0011 | mb
[Processor] decode: OUT | 0b1110 | mb
I/O 0x10 | jandl.aoc.jhipo.SimpleConsole | write]
data: CA
[Processor] decode: ADD | 0b0011 | mb
[Processor] decode: OUT | 0b1110 | mb
I/O 0x10 | jandl.aoc.jhipo.SimpleConsole | write]
data: FE
[Processor] decode: STA | 0b0001 | mb
[Processor] decode: HLT | 0b1111 | sb
[Processor] stopped

[Memory | Map]
20 CA 00 00 00 00 00 00 00 00 00 00 00 00 00
0F 34 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 FE 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00
11 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
12 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[Processor | Status]
PC : 0F | REM : 0E
SP : FF | RDM : F0 | N:0
RI : F0 | ACC : FE | Z:0
```

Saída de dados
no console 0x10.

Exemplo03



- Exemplo mostra o uso de IN, STA e OUT.
- Código:
 - 0xD0
0x10
- Corresponde à:
 - IN 0x10
 - Ou carrega acumulador com valor lido do I/O 0x10.

Exemplo03:: execução

```
Windows Command Processor - java -cp jhipo-v20150521.jar jhipovm +o programs\Exemplo03.asm
X:\JHIPO>java -cp jhipo-v20150521.jar jhipovm +o programs\Exemplo03.asm

[Memory | Map]
D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[Processor | Status]
PC : 00 | REM : 00
SP : FF | RDM : 00 | N:0
RI : 00 | ACC : 00 | Z:0

[Processor] started
[Processor] decode: IN | 0b1101 | mb
10 0x10 | jandl.aoc.jhipo.SimpleConsole | read]
data: 12
[Processor] decode: STA | 0b0001 | mb
[Processor] decode: OUT | 0b1110 | mb
10 0x10 | jandl.aoc.jhipo.SimpleConsole | write]
data: 12
[Processor] decode: IN | 0b1101 | mb
10 0x10 | jandl.aoc.jhipo.SimpleConsole | read]
data: 34
[Processor] decode: STA | 0b0001 | mb
[Processor] decode: OUT | 0b1110 | mb
10 0x10 | jandl.aoc.jhipo.SimpleConsole | write]
data: 34
[Processor] decode: HLT | 0b1111 | sb
[Processor] stopped

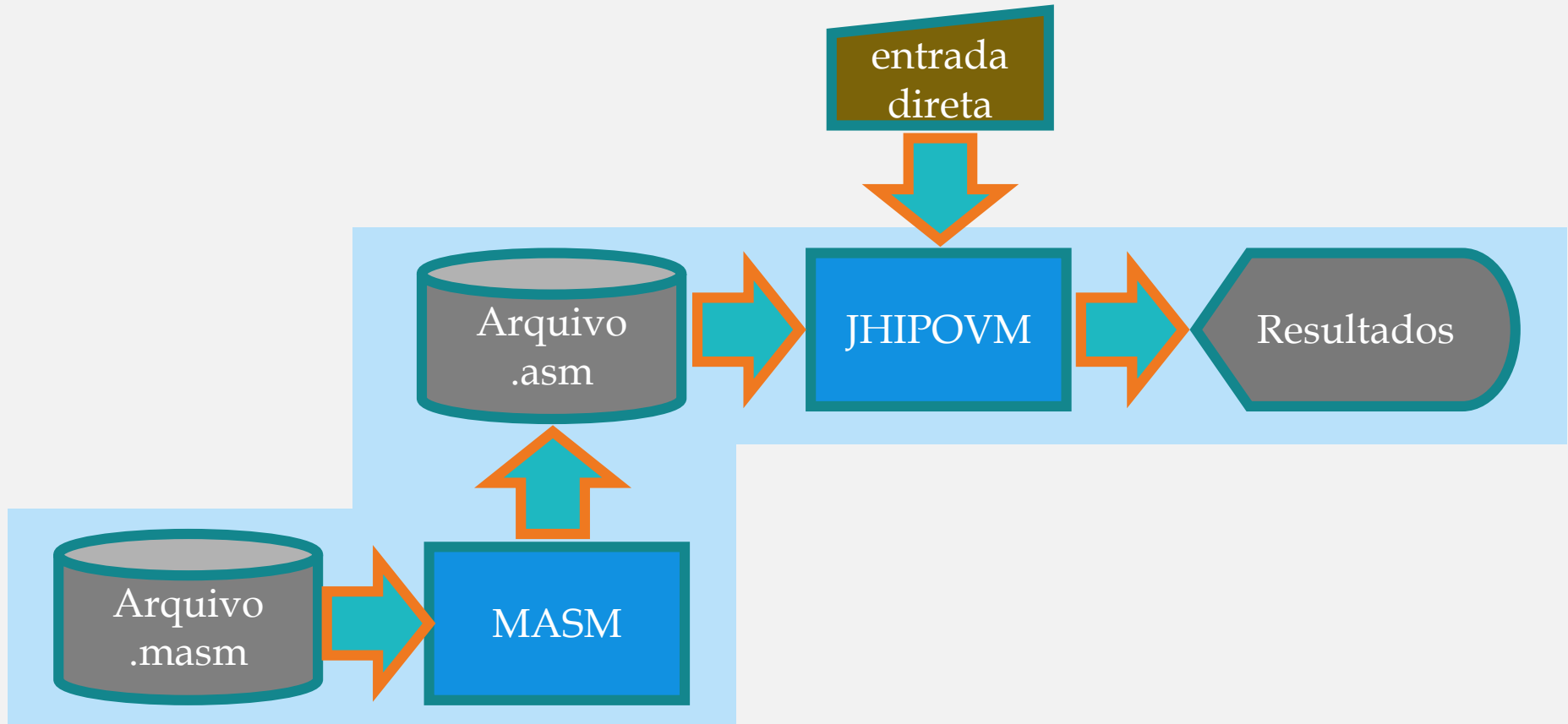
[Memory | Map]
D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
12 00 00 00 00 00 00 00 00 00 00 00 00 00 00
34 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Entrada e saída
de dados no
console 0x10.

MACROASSEMBLER

Uma linguagem construída sobre outra
linguagem!

JHIPOVM



Sintaxe MASM [macro-assembly]

- **Program** →
|
OneByteInstr Program
TwoByteInstr Address Program
Label ":" ONEBYTEINSTR Program
Label ":" TWOBYTEINSTR Address Program
Label ":" byteValue Program
ε
- **Address** →
|
byteValue
Label
- **OneByteInstr** → "NOP" | "NOT" | "RET" | "HLT"
- **TwoByteInstr** →
|
"STA" | "LDA" | "ADD" | "SUB" | "OR" | "AND"
"JMP" | "JN" | "JZ" | "CALL" | "IN" | "OUT"
- **Label** → Char [Char | Digit]⁺
- **Char** → "A" .. "Z" | "a" .. "z"
- **Digit** → "0" .. "9"

Sintaxe MASM [macro-assembly]

- Um *programa* é uma sequência de:
 - *instrução* (de um byte)
 - *instrução* (de dois bytes) com *endereço*
 - um *rótulo*, dois-pontos, seguido de *instrução* (de um byte)
 - um *rótulo*, dois-pontos, seguido de *instrução* (de dois bytes) com *endereço*
 - um *rótulo*, dois-pontos, seguido de um *valor* (constante de um byte)

Sintaxe MASM [macro-assembly]

- Um *rótulo* é uma sequência:
 - iniciada por caractere (alfabético apenas)
 - seguida por um ou mais caracteres (alfabético apenas) ou dígitos
- Um *endereço* é:
 - um *valor* (constante de um byte)
 - um *rótulo*

Sintaxe MASM [macro-assembly]

- As *instruções* de um byte são:
 - NOP, NOT, RET, HLT
- As *instruções* de dois bytes (que vem acompanhadas de um endereço) são:
 - STA, LDA, ADD, SUB, AND, OR, JMP, JZ, JN, CALL, IN, OUT

Exemplo00

Comentários são iniciados por um ; (ponto-e-vírgula) e não são considerados parte do programa.

```
*****  
;  
; File: Exemplo00.masm  
; Purpose: Uso de NOP e HLT  
; Author: P. Jandl Jr.  
; Date: 2015-10-23  
*****  
;
```

NOP ; nenhuma ação/operação

NOP

NOP

NOP

NOP

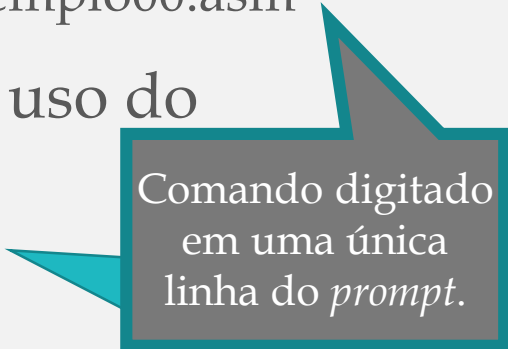
HLT ; parada do processador, ou seja, fim

Um programa em *macro-assembly* é uma sequência de instruções indicada por meio de seus mnemônicos.

Todo programa em *macro-assembly* é finalizado por um HLT.

Instruções para uso do MASM

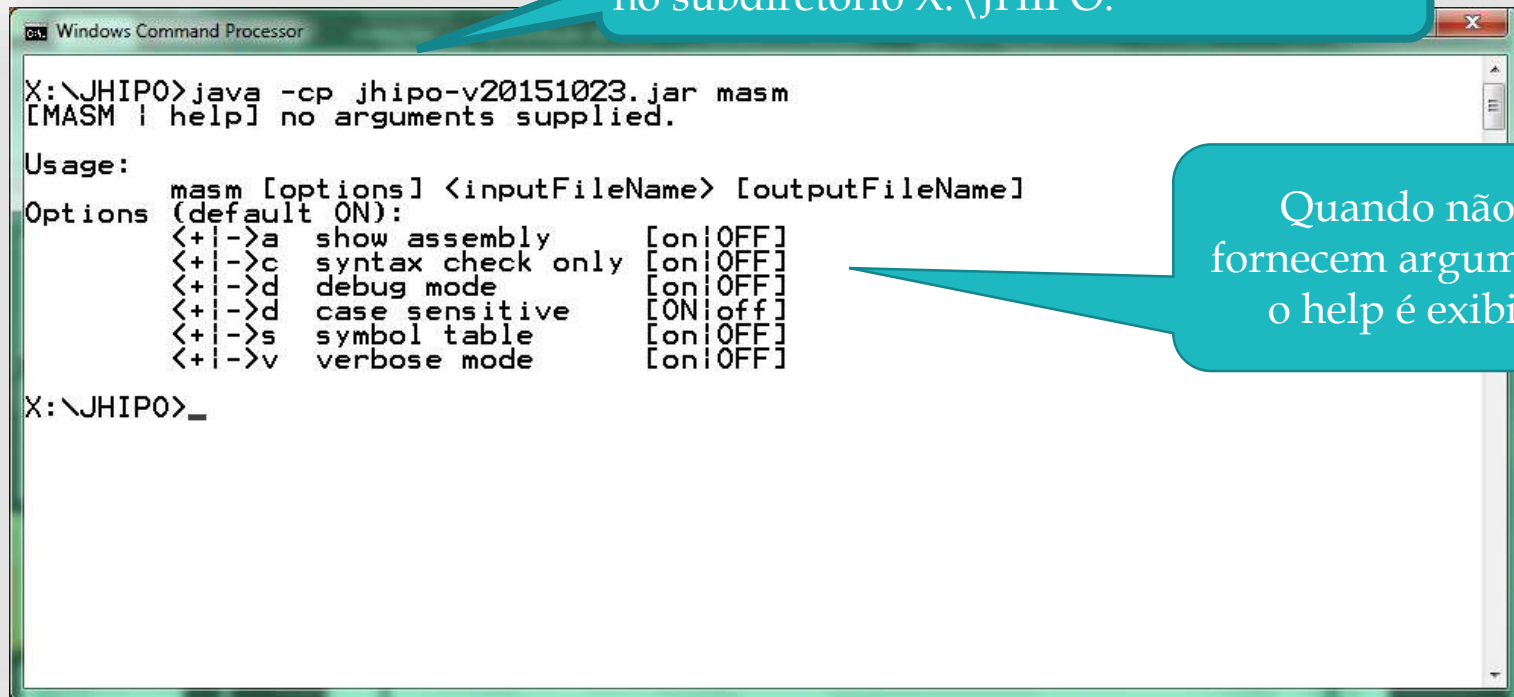
1. Mantenha os arquivos `.masm` no mesmo diretório que o arquivo `jar` correspondente ao JHIPO ou em um subdiretório direto.
2. Em um prompt de comandos, navegue até o diretório onde está o arquivo `jar`.
3. Execute:
 - `java -cp jhipo-vyyyyddmm.jar masm
program\Exemplo00.masm program\Exemplo00.asm`
4. Verifique as opções disponíveis para uso do MASM com:
 - `java -cp jhipo-vyyyyddmm.jar masm`



Comando digitado
em uma única
linha do *prompt*.

Opções do MASM

O arquivo jhipo-v20151023.jar foi copiado no subdiretório X:\JHIPO.



```
Windows Command Processor

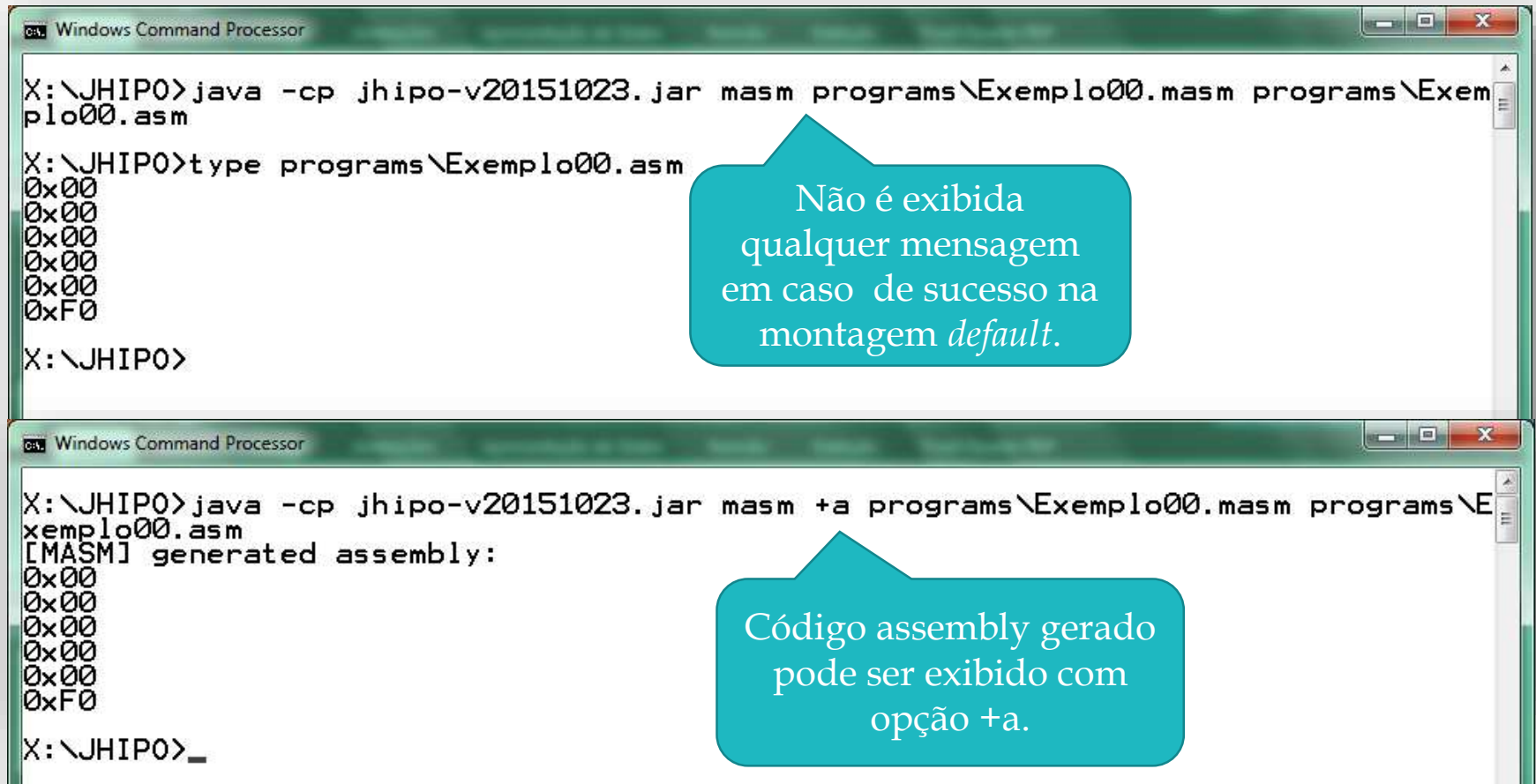
X:\JHIPO>java -cp jhipo-v20151023.jar masm
[MASM | help] no arguments supplied.

Usage:
Options  masm [options] <inputFileName> [outputFileName]
          (default ON):
          <+|->a  show assembly          [on|OFF]
          <+|->c  syntax check only      [on|OFF]
          <+|->d  debug mode              [on|OFF]
          <+|->d  case sensitive          [ON|off]
          <+|->s  symbol table            [on|OFF]
          <+|->v  verbose mode            [on|OFF]

X:\JHIPO>_
```

Quando não se fornecem argumentos, o help é exibido.

Opções do MASM



```
Windows Command Processor

X:\JHIP0>java -cp jhipo-v20151023.jar masm programs\Exemplo00.masm programs\Exem
plo00.asm

X:\JHIP0>type programs\Exemplo00.asm
0x00
0x00
0x00
0x00
0x00
0xF0

X:\JHIP0>
```

Não é exibida
qualquer mensagem
em caso de sucesso na
montagem *default*.

```
Windows Command Processor

X:\JHIP0>java -cp jhipo-v20151023.jar masm +a programs\Exemplo00.masm programs\Ex
emplo00.asm
[MASM] generated assembly:
0x00
0x00
0x00
0x00
0x00
0xF0

X:\JHIP0>_
```

Código assembly gerado
pode ser exibido com
opção +a.

Exemplo01

Compare o código
masm com seu
equivalente asm.

```
*****  
; File: Exemplo01.masm  
; Purpose: Uso de STA, OUT e LDA  
; Author: P. Jandl Jr.  
; Date: 2015-10-23  
*****  
LDA A ; carrega valor da variável A no acumulador  
OUT 0x10 ; exibe valor do acumulador na saída  
STA D ; armazena valor do acumulador na variável D  
LDA B  
OUT 0x10 ; exibe valor do acumulador na saída  
STA C ; armazena valor do acumulador na variável C  
HLT ; fim  
*****  
; Variáveis  
A: 0x10  
B: 0x15  
C: 0x00  
D: 0x00
```

Observe o uso de
rótulos e valores.

Exemplo01.asm - Notepad

File Edit Format View Help

0x20
0x0D
0xE0
0x10
0x10
0x10
0x20
0x0E
0xE0
0x10
0x10
0x0F
0xF0
0x10
0x15
0x00
0x00

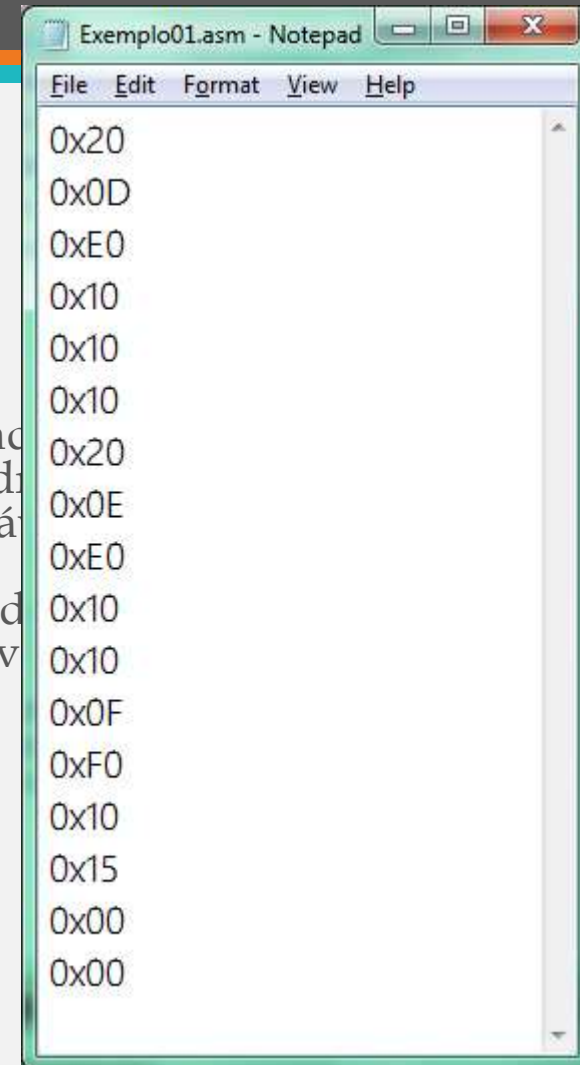
Rótulos (ou *labels*)
são traduzidos como
endereços.

Exemplo01

Compare o código
masm com seu
equivalente asm.

```
.*****  
;  
; File: Exemplo01.masm  
; Purpose: Uso de STA, OUT e LDA  
; Author: P. Jandl Jr.  
; Date: 2015-10-23  
.*****  
;  
    LDA A    ; carrega valor da variável A no acumulador  
    OUT 0x10 ; exibe valor do acumulador na saída padrão  
    STA D    ; armazena valor do acumulador na variável D  
    LDA B  
    OUT 0x10 ; exibe valor do acumulador na saída padrão  
    STA C    ; armazena valor do acumulador na variável C  
    HLT      ; fim  
.*****  
;  
; Variáveis  
A:    0x10  
B:    0x15  
C:    0x00  
D:    0x00
```

É interessante
posicionar variáveis e
seus conteúdos no final
do programa.



```
Exemplo01.asm - Notepad  
File Edit Format View Help  
0x20  
0x0D  
0xE0  
0x10  
0x10  
0x10  
0x20  
0x0E  
0xE0  
0x10  
0x10  
0x0F  
0xF0  
0x10  
0x15  
0x00  
0x00
```

Labels ou Rótulos

- São uma interessante característica dos macro-assemblers.
- Um rótulo é um identificador (um nome) para um endereço do programa.
- Ele permite "marcar" uma posição do programa, sem necessidade da contagem do endereço correspondente daquela posição.
- O macro-assembler, durante a compilação do código macro-assembly em assembly, determina o endereço físico de memória correspondente a cada rótulo.

Labels ou Rótulos

- Rótulos devem ser formados por uma sequência de letras e números, iniciada por letra, sem espaços ou quaisquer outros símbolos.
- Assim, é possível escrever:
 - A: 0x10
 - Que define um conteúdo 0x10 para um posição de memória que pode ser referenciada pelo rótulo/label A.
- Isto permite que, em qualquer ponto do programa macro-assembly seja feito:
 - ADD A
- Ao invés de:
 - ADD 0x20 ; imaginando que o conteúdo 0x10 acima estivesse definido no endereço 0x20 da memória.
- Os dois comandos realizam a mesma operação: acrescentam o conteúdo da posição de memória indicada ao acumulador.

Labels ou Rótulos

- Também é possível escrever:
 - PONTO: ADD A
 - Que define um nome PONTO para a instrução ADD A (que também pode usar rótulos ao invés de endereços).
- Com isso, uma instrução de salto (JMP, JN ou JZ) pode indicar o rótulo como destino ao invés do endereço físico:
 - JMP PONTO
 - JN PONTO
 - JZ PONTO
- Isto simplifica a organização do programa.

Exemplo02

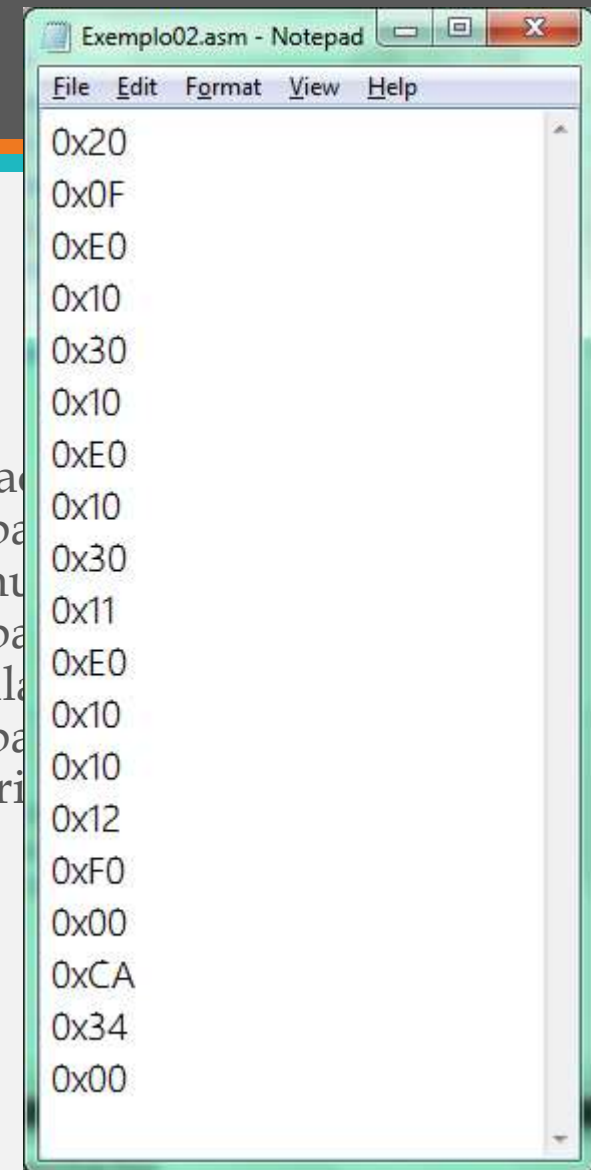
Compare o código
masm com seu
equivalente asm.

```
*****  
;  
; File: Exemplo02.masm  
; Purpose: Uso de LDA, ADD, STA e OUT  
; Author: P. Jandl Jr.  
; Date: 2015-10-27  
*****  
;
```

```
LDA ZERO ; carrega valor da variável ZERO no acumulador  
OUT 0x10 ; exibe valor do acumulador na saída paralela  
ADD A ; adiciona o valor da variável A ao acumulador  
OUT 0x10 ; exibe valor do acumulador na saída paralela  
ADD B ; adiciona o valor da variável B ao acumulador  
OUT 0x10 ; exibe valor do acumulador na saída paralela  
STA TOT ; armazena valor do acumulador na variável TOT  
HLT ; fim
```

```
*****  
;  
; Variáveis  
ZERO: 0x00  
A: 0xCA  
B: 0x34  
TOT: 0x00
```

Observe o uso de
rótulos e valores, que
funcionam como
variáveis.



```
Exemplo02.asm - Notepad  
File Edit Format View Help  
0x20  
0x0F  
0xE0  
0x10  
0x30  
0x10  
0xE0  
0x10  
0x30  
0x11  
0xE0  
0x10  
0x10  
0x10  
0x12  
0xF0  
0x00  
0xCA  
0x34  
0x00
```

Exemplo03

Compare o código
masm com seu
equivalente asm.

```
.*****  
;  
; File: Exemplo03.masm  
; Purpose: Uso de IN, STA e OUT  
; Author: P. Jandl Jr.  
; Date: 2015-10-27  
.*****  
;
```

```
    IN 0x10 ; carrega acumulador com valor lido do teclado  
    STA A  ; armazena valor do acumulador na variável A  
    OUT 0x10 ; exibe valor do acumulador na porta 0x10  
    IN 0x10 ; carrega acumulador com valor lido do teclado  
    STA B  ; armazena valor do acumulador na variável B  
    OUT 0x10 ; exibe valor do acumulador na porta 0x10  
    HLT    ; fim
```

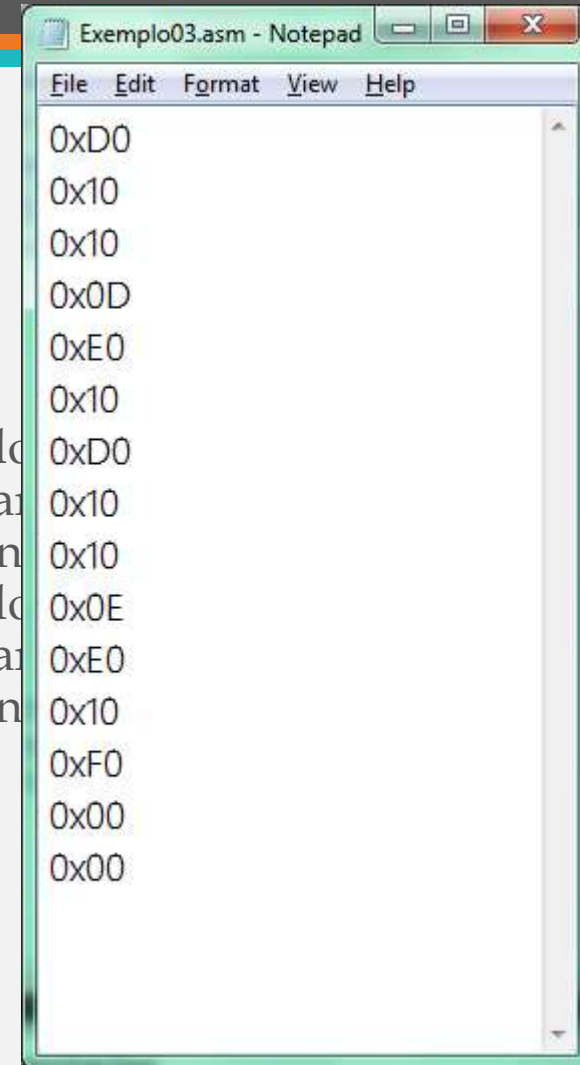
```
.*****  
;
```

; Variáveis

A: 0x00

B: 0x00

Observe o uso de
rótulos e valores, que
funcionam como
variáveis.



```
Exemplo03.asm - Notepad  
File Edit Format View Help  
0xD0  
0x10  
0x10  
0x0D  
0xE0  
0x10  
0xD0  
0x10  
0x10  
0x0E  
0xE0  
0x10  
0xF0  
0x00  
0x00
```

Exercícios de Fixação

Exercícios

1. Escreva um programa em linguagem de máquina do HIPO capaz de ler três valores dados pelo usuário, armazenando o total obtido na última posição de memória correspondente ao programa e exibindo tal valor no console.
2. Escreva um programa em linguagem de máquina do HIPO capaz de exibir uma contagem de 0 até (05)10 no console.
 - Observe a repetição de código.
 - Considere a possibilidade de uso da instrução JZ ou JN para a construção de um laço de repetição.

Exercícios

3. Escreva um programa em linguagem de máquina do HIPO capaz de multiplicar dois valores dados pelo usuário, armazenando o total obtido na última posição de memória ocupada pelo programa e exibindo tal valor no console.