Programação JHIPO ::MASM (parte 1)

Prof. Ms. Peter Jandl Junior
Arquitetura e Organização de Computadores

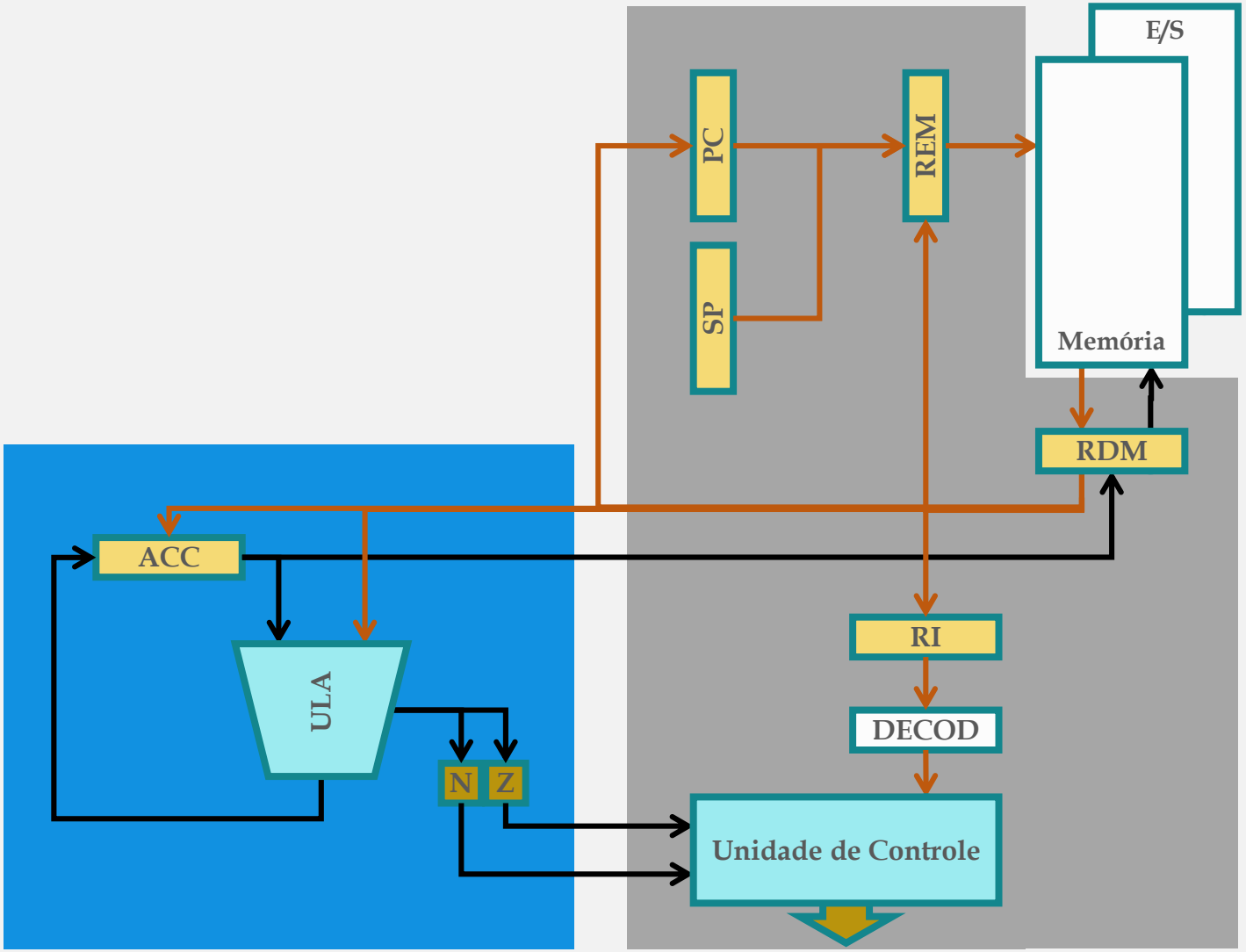
Análise e Desenvolvimento de Sistemas
FATEC – Jundiaí

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

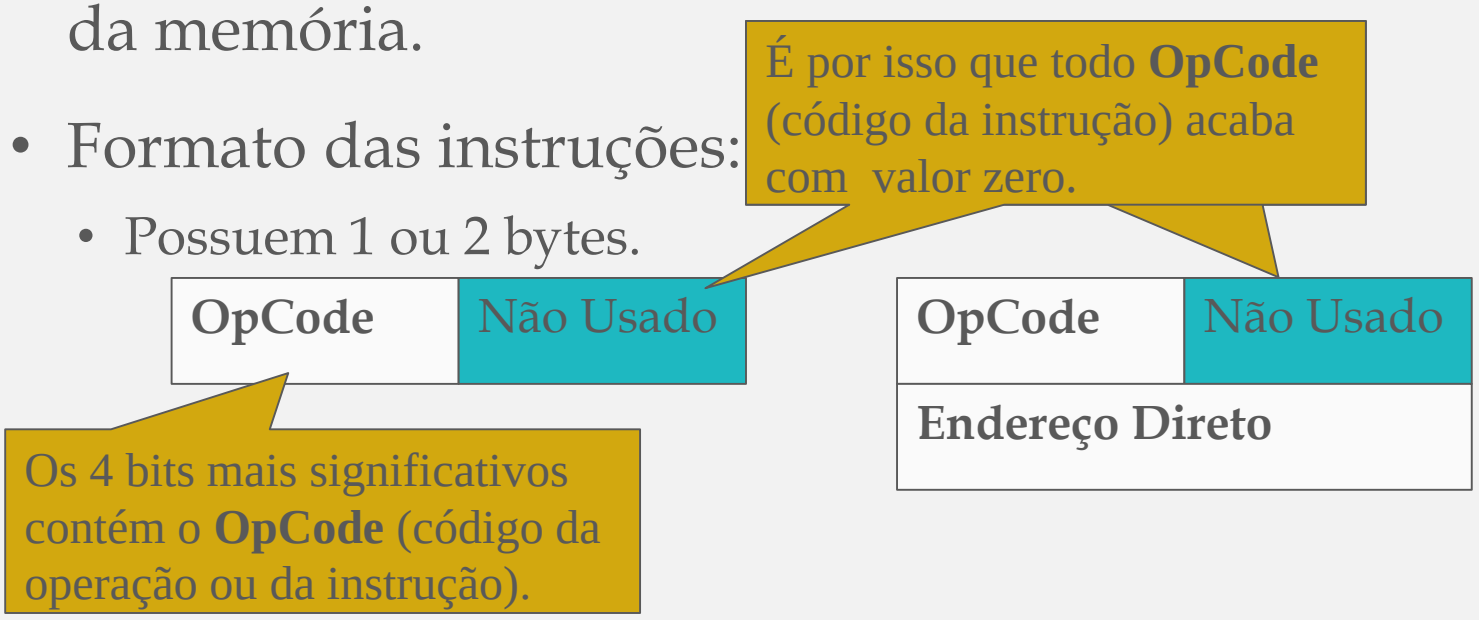
Código hexadecimal da instrução.

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



Aplicação das Instruções

Programação

- A programação de computadores envolve cinco capacidades essenciais:
  - sequenciação
  - computação
  - repetição
  - decisão
  - modularização
- Desta maneira, qualquer linguagem de programação, de baixo ou alto nível, deve oferecer meios para disponibilizar estas capacidades.

Programação JHIPO::Sequenciação

- A arquitetura e organização do JHIPO permite:
  - sequenciação
    - pois, instruções do processador podem ser arranjadas numa sequência para produção de resultados específicos.
- Exemplo:
  - **IN** 0x10
  - **STA** 0x7E
  - **NOP**
  - **OUT** 0x10
  - **HLT**

Aqui são exibidos os mnemônicos das instruções do JHIPO.

Os endereços indicados são apenas exemplos.

Programação JHIPO::Computação

- O *instruction set* do JHIPO dispõe de:
  - soma → **ADD** *endereço*
  - subtração → **SUB** *endereço*
    - cuja combinação permite obter operações matemáticas mais complexas.
  - e-lógico → **AND** *endereço*
  - ou-lógico → **OR** *endereço*
  - negação → **NOT**
    - cuja combinação permite obter operações lógicas mais complexas.

Programação JHIPO::Computação

- Além disso, também estão disponíveis:
  - entrada → **IN** *endereço*
  - saída → **OUT** *endereço*
    - que permitem a comunicação do sistema JHIPO com o exterior (interação com usuário).
  - armazenamento → **STA** *endereço*
  - recuperação → **LDA** *endereço*
    - que possibilitam armazenar e recuperar dados da memória do sistema.

Exercício 01

- 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.

Exercício 01

Aqui temos um programa que envolve sequenciação e computação

```
; File: Exerc10-01.masm
        IN 0x10         ; carrega acumulador com valor lido do I/O 0x10
        STA A           ; armazena valor do acumulador na variável A
        IN 0x10         ; carrega acumulador com valor lido do I/O 0x10
        STA B           ; armazena valor do acumulador na variável B
        IN 0x10         ; carrega acumulador com valor lido do I/O 0x10
        STA C           ; armazena valor do acumulador na variável C
        ADD B           ; acrescente conteúdo de B ao acumulador
        ADD A           ; acrescente conteúdo de A ao acumulador
        STA TOT         ; armazena valor do acumulador na variável TOT
        OUT 0x10        ; exibe valor do acumulador na saída padrão
        HLT             ; fim
; Variáveis
A:      0x00
B:      0x00
C:      0x00
TOT:    0x00            ; última posição de memória do programa
```

Exercício01:: execução

Windows Command Processor - java -cp jhipo-v20151023.jar jhipovm +o -s programs\Exerc10-01.asm

```
X:\JHIPO>java -cp jhipo-v20151023.jar jhipovm +o -s programs\Exerc10-01.asm

[Memory | Map]
D0  10  00  00  00  00  00  00  00  00  00  00  00  00  00  00
10  18  00  00  00  00  00  00  00  00  00  00  00  00  00  00
10  E0  00  00  00  00  00  00  00  00  00  00  00  00  00  00
15  10  00  00  00  00  00  00  00  00  00  00  00  00  00  00
D0  F0  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
16  00  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  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
17  00  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  00
16  00  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  00
15  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00

[Processor] started
[Processor] decode:   IN | 0b1101 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | read]
> 3
[Processor] decode:  STA | 0b0001 | mb
[Processor] decode:   IN | 0b1101 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | read]
> 6
[Processor] decode:  STA | 0b0001 | mb
[Processor] decode:   IN | 0b1101 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | read]
> 12
[Processor] decode:  STA | 0b0001 | mb
[Processor] decode:  ADD | 0b0011 | mb
[Processor] decode:  ADD | 0b0011 | mb
[Processor] decode:  STA | 0b0001 | mb
[Processor] decode:  OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 1B
[Processor] decode:  HLT | 0b1111 | sb
[Processor] stopped

[Memory | Map]
D0  10  00  00  00  00  00  00  00  00  00  00  00  00  00  00
10  18  00  00  00  00  00  00  00  00  00  00  00  00  00  00
10  E0  00  00  00  00  00  00  00  00  00  00  00  00  00  00
15  10  00  00  00  00  00  00  00  00  00  00  00  00  00  00
D0  F0  00  00  00  00  00  00  00  00  00  00  00  00  00  00
10  03  00  00  00  00  00  00  00  00  00  00  00  00  00  00
10  06  00  00  00  00  00  00  00  00  00  00  00  00  00  00
16  12  00  00  00  00  00  00  00  00  00  00  00  00  00  00
D0  1B  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
17  00  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  00
16  00  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  00
15  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
```

Resultado Correto:
03 + 06 + 12 = 1B
(tudo em hexadecimal)

Exercício 02

- 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.

Exercício 02

Outro programa que envolve sequenciação e computação

Repetição de código: cinco blocos idênticos!

```
LDA ZERO      ; carrega acumulador com valor da vari. ZERO
STA COUNT     ; armaz valor do acumulador na var.  COUNT
OUT 0x10      ; exibe valor do acumulador na saída padrão
ADD INC       ; acrescenta conteúdo de INC ao acumulador
STA COUNT     ; armazena valor do acumulador na variável TOT
OUT 0x10      ; exibe valor do acumulador na saída padrão
ADD INC       ; acrescenta conteúdo de INC ao acumulador
STA COUNT     ; armazena valor do acumulador na variável TOT
OUT 0x10      ; exibe valor do acumulador na saída padrão
ADD INC       ; acrescenta conteúdo de INC ao acumulador
STA COUNT     ; armazena valor do acumulador na variável TOT
OUT 0x10      ; exibe valor do acumulador na saída padrão
ADD INC       ; acrescenta conteúdo de INC ao acumulador
STA COUNT     ; armazena valor do acumulador na variável TOT
OUT 0x10      ; exibe valor do acumulador na saída padrão
ADD INC       ; acrescenta conteúdo de INC ao acumulador
STA COUNT     ; armazena valor do acumulador na variável TOT
OUT 0x10      ; exibe valor do acumulador na saída padrão
HLT           ; fim
; Variáveis
ZERO:   0x00
COUNT:  0x00
INC:    0x01
```

Exercício 02::Execução

Execução tomou 19 instruções!

Programa ocupou 40 bytes!

```
Windows Command Processor - java  -cp jhipo-v20151023.jar jhipovm +o -a programs\Exerc10-02.asm
X:\JHIPO>java -cp jhipo-v20151023.jar jhipovm +o -a programs\Exerc10-02.asm

[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:  STA | 0b0001 | mb
[Processor] decode:  OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 00
[Processor] decode:  ADD | 0b0011 | mb
[Processor] decode:  STA | 0b0001 | mb
[Processor] decode:  OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 01
[Processor] decode:  ADD | 0b0011 | mb
[Processor] decode:  STA | 0b0001 | mb
[Processor] decode:  OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 02
[Processor] decode:  ADD | 0b0011 | mb
[Processor] decode:  STA | 0b0001 | mb
[Processor] decode:  OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 03
[Processor] decode:  ADD | 0b0011 | mb
[Processor] decode:  STA | 0b0001 | mb
[Processor] decode:  OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 04
[Processor] decode:  ADD | 0b0011 | mb
[Processor] decode:  STA | 0b0001 | mb
[Processor] decode:  OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 05
[Processor] decode:  HLT | 0b1111 | sb
[Processor] stopped

[Processor | Status]
PC  : 25 | REM : 24
SP  : FF | RDM : F0 | N:0
RI  : F0 | ACC : 05 | Z:0
```

Exercício 02::considerações

- A execução sequencial apresenta como **vantagens**:
  - simplicidade, pois tarefas desejadas são executadas diretamente;
  - desempenho, sendo mais simples é, em geral mais eficiente.
- Ao mesmo tempo apresenta **desvantagens**:
  - tamanho inadequado, pois a medida que cresce o número de repetições de uma tarefa, mais código torna-se necessário, ocupando mais memória;
  - esta estratégia não permite solucionar situações onde o número de repetições de uma tarefa é definido dinamicamente, isto é, durante a execução do programa.

Programação JHIPO::Repetição

- O *instruction set* do JHIPO não dispõe de operação específica para repetição, mas oferece:
  - desvio/salto → JMP end
  - que permite deslocar a execução do programa para outro ponto (***desvio incondicional***), possibilitando a repetição.
- Exemplo:
  - LOOP: LDA X
  - ADD INC
  - :
  - JMP LOOP

Programação JHIPO::Decisão

- O *instruction set* do JHIPO dispõe de operação específica para decisão:
  - salta se zero → JZ end
  - salta se negativo → JN end
  - que permitem deslocar a execução do programa para outro ponto ***quando*** ocorre a condição requerida (valor zero ou negativo), possibilitando o desvio e a repetição.
- Exemplo:
  - JZ NAO
  - SIM: : ; código para condição não zero
  - :
  - JMP CONT
  - NAO: : ; código para condição zero
  - :
  - CONT: : ; continuação

Programação JHIPO::Modularização

- O *instruction set* do JHIPO dispõe de operações que possibilitam a modularização do código:
  - chama de sub-rotina → CALL end
  - retorno de sub-rotina → RET
  - que permitem que a execução do programa seja deslocada para outro bloco de código, possibilitando o retorno para o ponto de chamada.
- Exemplo:
  - CALL FUNC
  - :
  - FUNC: LDA X
  - :
  - RET

Instruções de desvio

As instruções que permite ações de desvio, de decisão e de repetição, além da modularização de programas.

Instruções de Desvio

INCONDICIONAL

- **JMP**
  Jump

- **CALL**
  Subprogram Call
- **RET**
  Subprogram Return

Instruções específicas para criação de subprogramas.

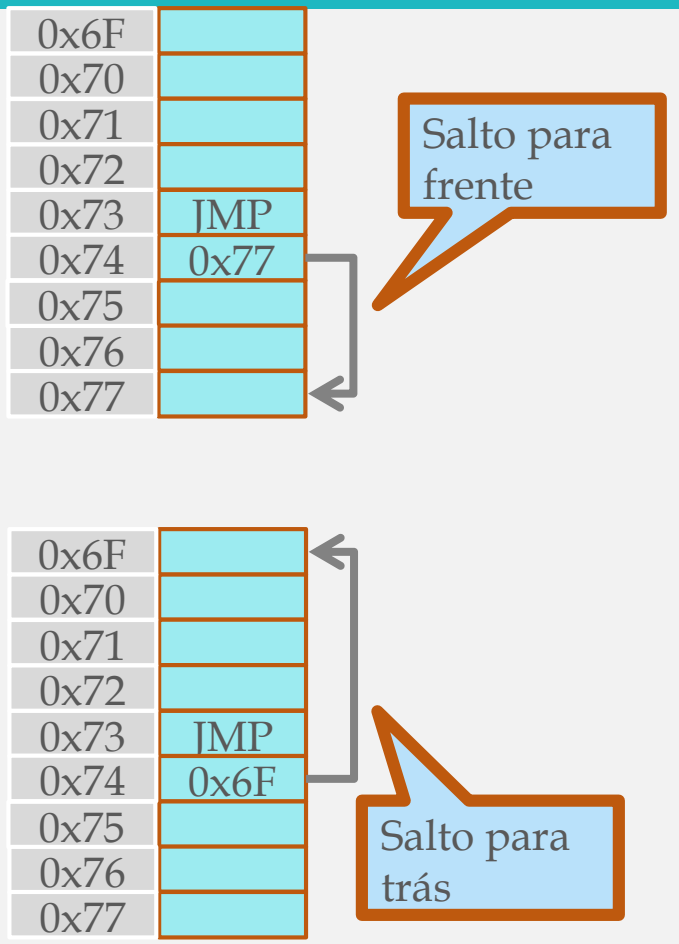
CONDICIONAL

- **JN**
  Jump if Negative

- **JZ**
  Jump if Zero

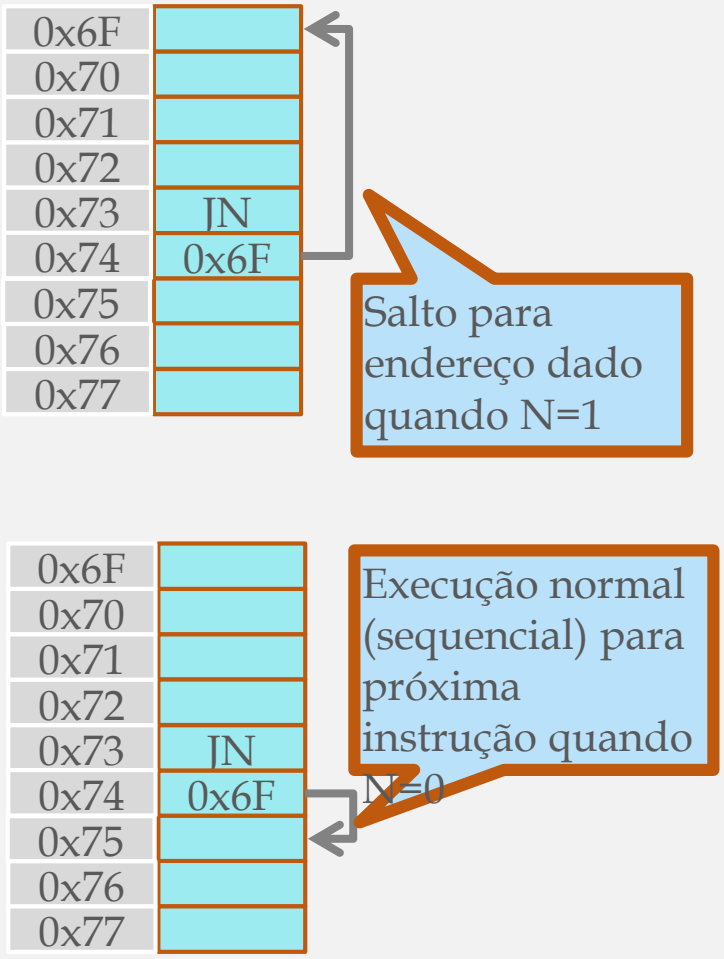
JMP::Jump

- **JMP** *end*
- Provoca um desvio (salto) na execução do código para o endereço de seu operando.
- O desvio pode ocorrer para posição à frente ou atrás da instrução JMP.
- Execução ***não depende*** de qualquer condição.



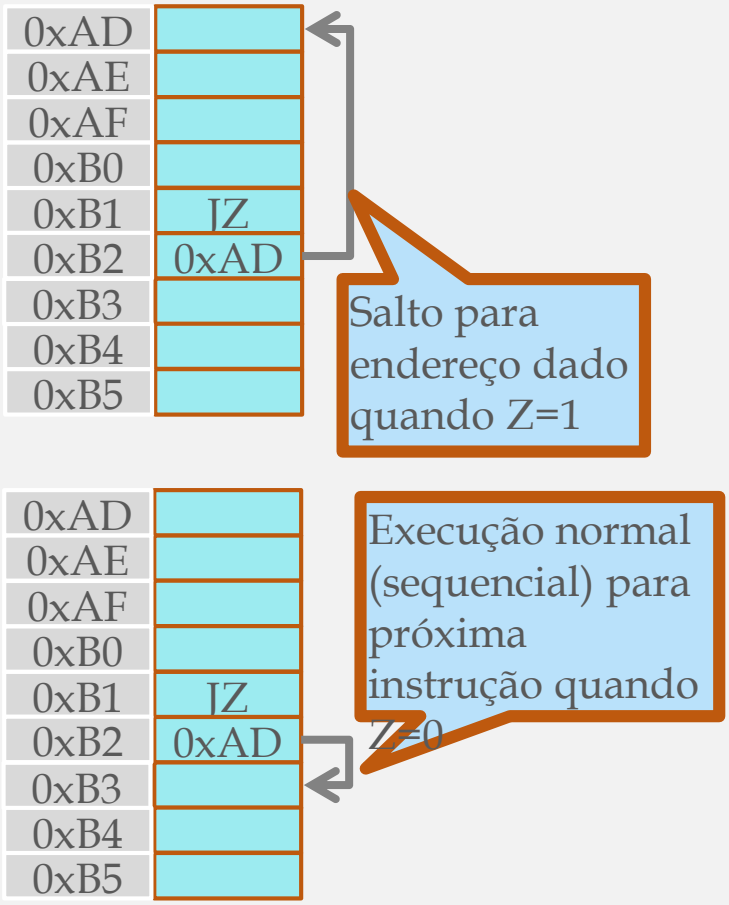
JN::Jump if Negative
JZ::Jump if Zero

- **JN** *end* | **JZ** *end*
- Provoca um desvio (salto) na execução do código para o endereço de seu operando quando ocorre uma condição específica.
- O desvio pode ocorrer para posição à frente ou atrás da instrução **JN|JZ**.
- Execução ***depende*** de qualquer condição.



JN::Jump if Negative
JZ::Jump if Zero

- A condição específica é indicada por uma *flag*:
  - N (Negative) para JN
  - Z (Zero) para JZ
- Tais *flags* são modificadas por meio de operações envolvendo o acumulador (ACC):
  - LDA, ADD, SUB, AND, OR, NOT, IN



Exercício 02B

- Escreva um programa em linguagem de máquina do HIPO capaz de exibir uma contagem de 0 até $(05)_{10}$ no console.
  - Utilizando repetição da execução de código proporcionada pelo uso da instrução JZ ou JN para a construção de um laço de repetição.

Exercício 02B

Bloco a ser repetido inserido numa estrutura de repetição!

Controle da repetição: se contagem – MAX resulta negativo, ainda não repetiu o suficiente!

```
        LDA ZERO        ; carrega acumulador com valor da variável ZERO
        STA COUNT       ; armazena valor do acumulador na variável COUNT
LOOP:   LDA COUNT       ; carrega acumulador com valor da variável COUNT
        OUT 0x10        ; exibe valor do acumulador na saída padrão
        ADD INC         ; acrescenta conteúdo de INC ao acumulador
        STA COUNT       ; armazena valor do acumulador na variável TOT
        SUB MAX         ; decresce acumulador com valor de MAX
        JN LOOP         ; se negativo salta para LOOP
        LDA COUNT       ; carrega acumulador com valor da variável COUNT
        OUT 0x10        ; exibe valor do acumulador na saída padrão
        HLT             ; fim
; Variáveis
ZERO:   0x00
COUNT:  0x00
INC:    0x01
MAX:    0x05
```

Para contornar efeito colateral da operação de controle realizada no acumulador!

Exercício02B:: execução

Execução tomou 35 instruções!

Programa ocupou 25 bytes!

```
Windows Command Processor - java  -cp jhipo-v20151023.jar jhipovm +o -a programs\Exerc10-02B.asm
X:\JHIPO>java -cp jhipo-v20151023.jar jhipovm +o -a programs\Exerc10-02B.asm

[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:   STA | 0b0001 | mb
[Processor] decode:   LDA | 0b0010 | mb
[Processor] decode:   OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 00
[Processor] decode:   ADD | 0b0011 | mb
[Processor] decode:   STA | 0b0001 | mb
[Processor] decode:   SUB | 0b0100 | mb
[Processor] decode:    JN | 0b1001 | mb
[Processor] decode:   LDA | 0b0010 | mb
[Processor] decode:   OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 01
[Processor] decode:   ADD | 0b0011 | mb
[Processor] decode:   STA | 0b0001 | mb
[Processor] decode:   SUB | 0b0100 | mb
[Processor] decode:    JN | 0b1001 | mb
[Processor] decode:   LDA | 0b0010 | mb
[Processor] decode:   OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 02
[Processor] decode:   ADD | 0b0011 | mb
[Processor] decode:   STA | 0b0001 | mb
[Processor] decode:   SUB | 0b0100 | mb
[Processor] decode:    JN | 0b1001 | mb
[Processor] decode:   LDA | 0b0010 | mb
[Processor] decode:   OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 03
[Processor] decode:   ADD | 0b0011 | mb
[Processor] decode:   STA | 0b0001 | mb
[Processor] decode:   SUB | 0b0100 | mb
[Processor] decode:    JN | 0b1001 | mb
[Processor] decode:   LDA | 0b0010 | mb
[Processor] decode:   OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 04
[Processor] decode:   ADD | 0b0011 | mb
[Processor] decode:   STA | 0b0001 | mb
[Processor] decode:   SUB | 0b0100 | mb
[Processor] decode:    JN | 0b1001 | mb
[Processor] decode:   LDA | 0b0010 | mb
[Processor] decode:   OUT | 0b1110 | mb
IO[0x10| jandl.aoc.jhipo.SimpleConsole | write]
> 05
[Processor] decode:   HLT | 0b1111 | sb
[Processor] stopped

[Processor | Status]
PC   : 15 | REM : 14
SP   : FF | RDM : F0 | N:0
RI   : F0 | ACC : 05 | Z:0
```

Comparação

Execução Sequencial

- Programa ocupa **mais** espaço em memória.
- Execução toma **menos** instruções.
- *Maior, mas rápido!*

Execução com Repetição

- Programa ocupa **menos** espaço em memória.
- Execução toma **mais** instruções.
- *Menor, mas lento!*

Comparação

Cinco blocos	Cinquenta blocos
• Código sequencial:	• Código sequencial:
• *19 instruções executadas*	• *154 instruções executadas*
• *40 bytes de memória*	• *310 bytes de memória*
• Código repetitivo:	• Código repetitivo:
• *35 instruções executadas*	• *305 instruções executadas*
• *25 bytes de memória*	• *25 bytes de memória*

Maior

Mais lento

Exercício 03

- 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 correspondente ao programa e exibindo tal valor no console.

Exercício 03

- JHIPO não dispõe de instrução de multiplicação.
- Mas a multiplicação é uma soma sucessiva:
  - X = 3 x 4 = 0 + 4 + 4 + 4
  - ou seja, 3 vezes 4 é o mesmo que a 3 somas do valor 4 a partir de zero!
- Para realizar a soma são necessários:
  - uma variável para armazenar a soma progressiva, que no final constitui o resultado desejado;
  - uma variável para armazenar o valor a ser somado;
  - outra variável para armazenar o número total de somas;
  - e outra variável para contar quantas somas já foram efetuadas.

Exercício 03

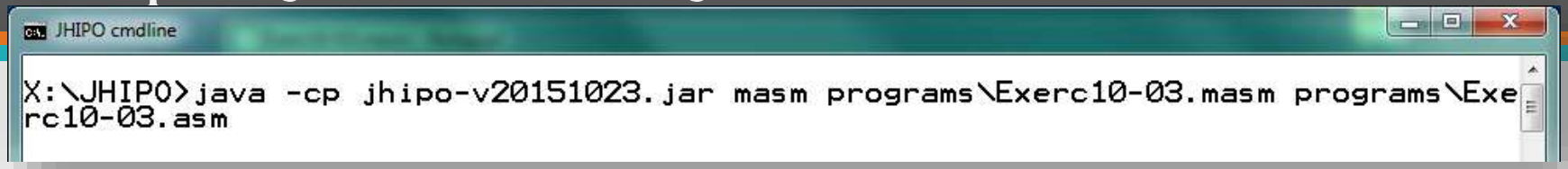
Bloco opcional

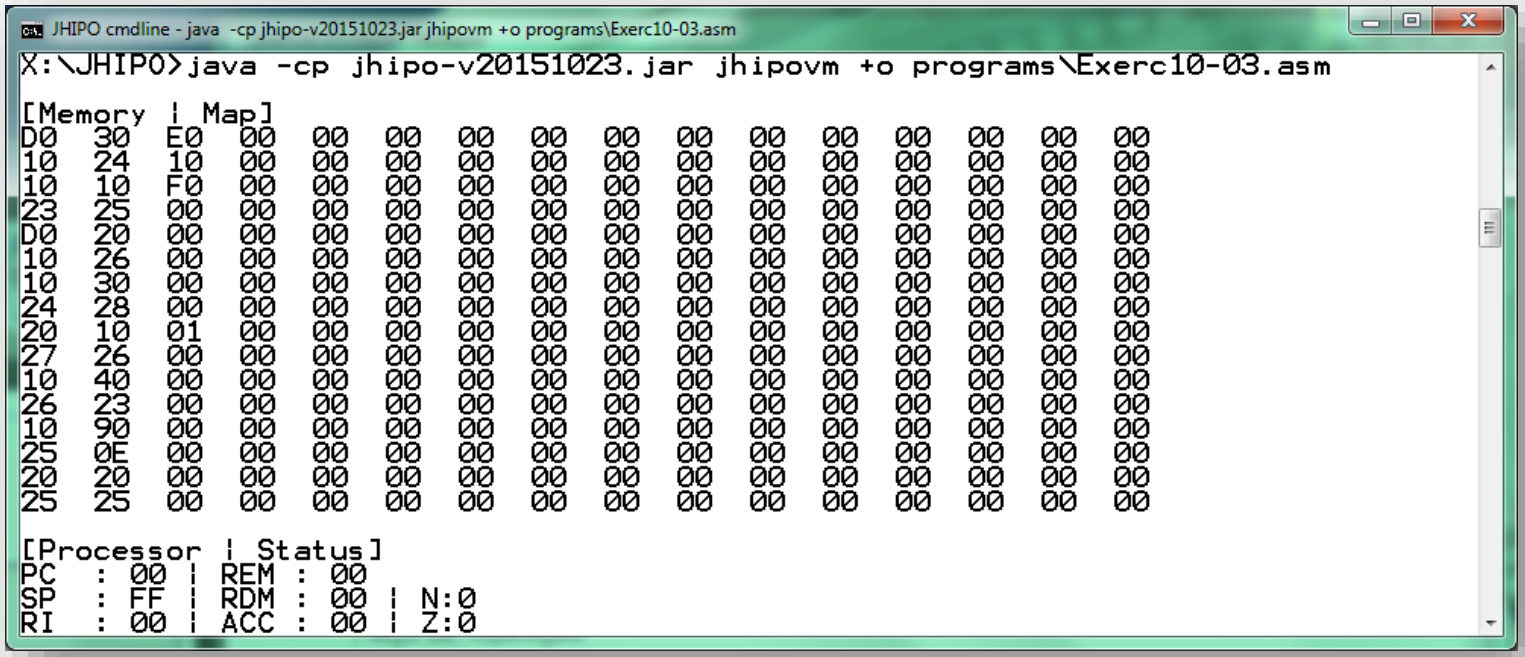
Laço que contém multiplicação

Exibição do resultado

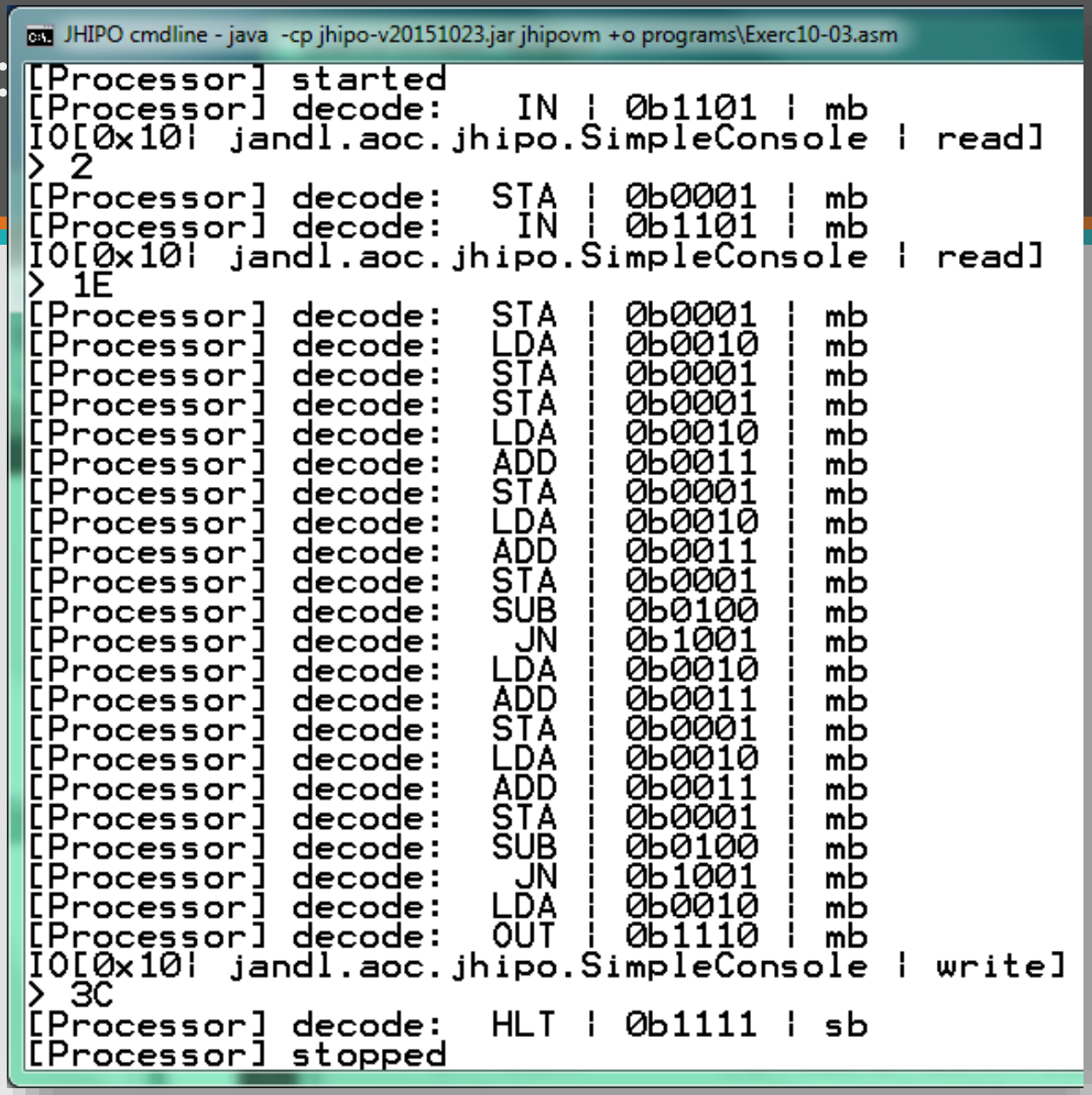
```
        IN 0x10     ; efetua leitura de valor na entrada padrão
        STA A       ; armazena ACC em  A
        IN 0x10     ; efetua leitura de valor na entrada padrão
        STA B       ; armazena ACC em B
        ; garante valores iniciais CONT = 0 e RES = 0
        LDA ZERO            ; armazena ZERO no ACC (ACC=0)
        STA CONT; armazena ZERO em CONT (CONT=0)
        STA RES     ; armazena ACC em RES (RES=0)
        ; laço de repetição
LOOP:   LDA RES     ; armazena RES no ACC (ACC=RES)
        ADD B       ; adiciona B no ACC (ACC=RES+B)
        STA RES     ; armazena ACC em RES           (RES=RES+B)
        LDA CONT            ; carrega ACC com CONT (ACC=CONT)
        ADD UM      ; soma UM ao ACC (ACC=CONT+1)
        STA CONT; armazena ACC em CONT (CONT=CONT+1)
        SUB A       ; subtrai A do ACC (A é num vezes que somamos  B)
        JN LOOP     ; se negativo, salta para LOOP (zero indica que CONT=A)
        ; exibição do resultado RES
        LDA RES     ; carrega acumulador com RES
        OUT 0x10    ; exibe ACC (valor RES) na saída padrão
        HLT         ; fim
; Variáveis
A:      0x00        ; variável A
B:      0x00        ; variável B
RES:    0x00        ; variável auxiliar para resultado
CONT:   0x00        ; variável auxiliar contador
ZERO:   0x00        ; constante 0 (decimal)
UM:     0x01        ; constante 1 (decimal)
```

Exercício 03::
compilação & execução





Exercício 03:: execução



Exercício 03:: execução

Exercícios de Fixação

Exercícios

1. Escreva um programa que efetue a divisão inteira de um valor A por outro valor B, apresentando o quociente inteiro e também o resto da divisão. Os valores de A e B devem ser fornecidos pelo usuário.

2. Escreva um programa que preencha uma região de memória de 32 bytes com um valor constante fornecido pelo usuário. A região a ser preenchida se localiza a partir (inclusive) do endereço 0xE0.

Exercícios

3. Escreva um programa que preencha uma região de memória de 32 bytes com uma contagem de 0 a 31 (no seu equivalente hexadecimal). A região a ser preenchida se localiza a partir (inclusive) do endereço 0xE0.

4. Escreva um programa que compare dois valores A e B dados pelo usuário, escrevendo na saída padrão (0x10) o maior valor.

5. Escreva um programa que verifique se o bit 5 de um valor A dado pelo usuário é zero ou um, escrevendo na saída padrão (0x10):
   - o valor 0x00 se tal bit for zero; e
   - o valor 0x01 se tal bit for um.