Programação JHIPO ::MASM (parte 2)

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

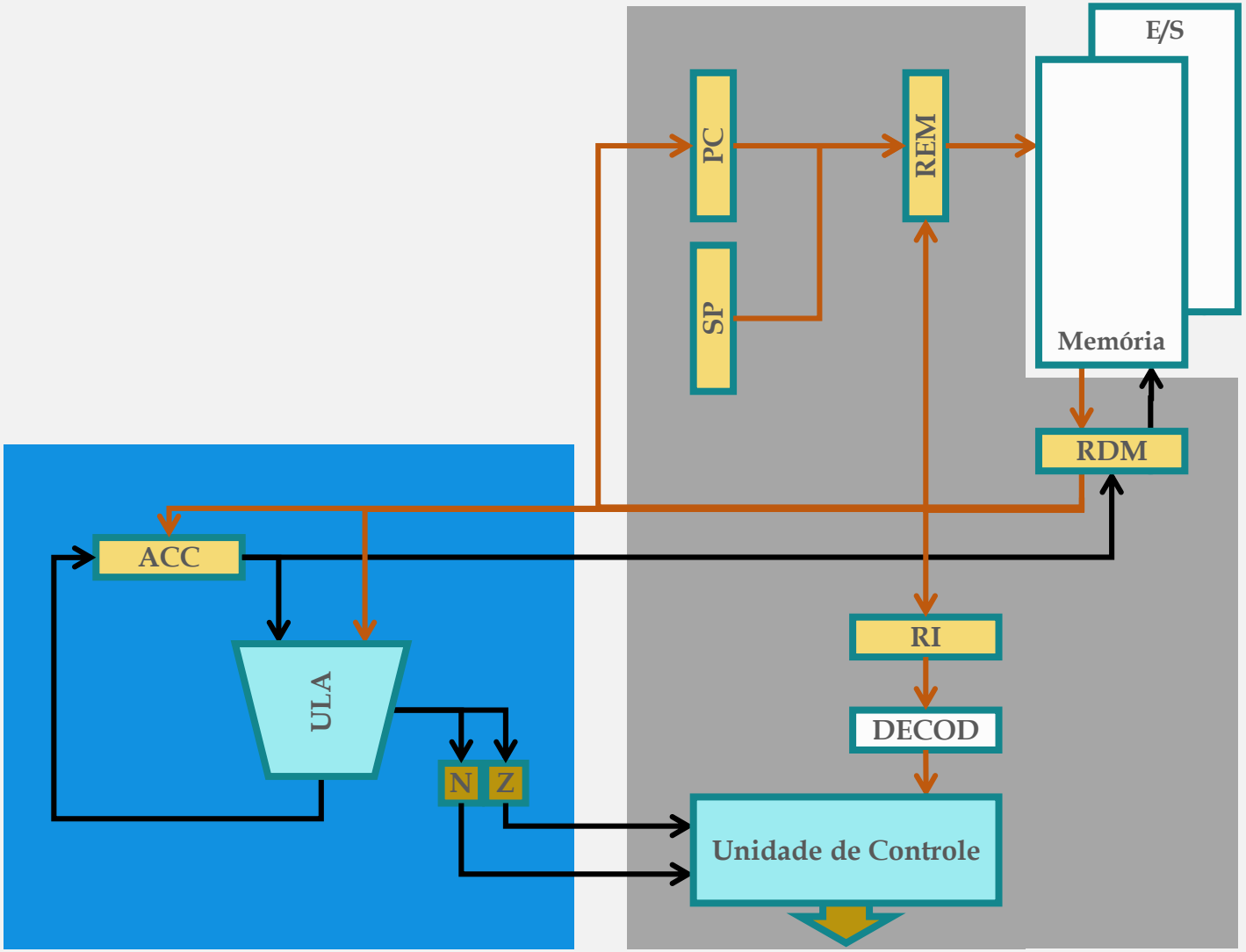
Análise e Desenvolvimento de Sistemas
FATEC – Jundiaí

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/Instruction Set

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.

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.

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.

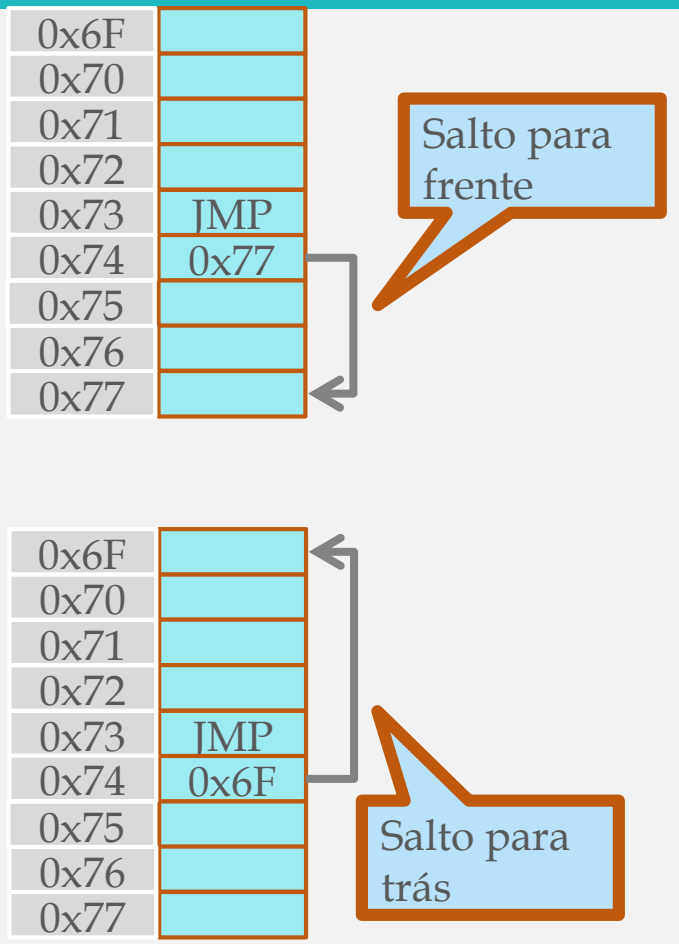
CONDICIONAL

- **JN**
  Jump if Negative

- **JZ**
  Jump if Zero

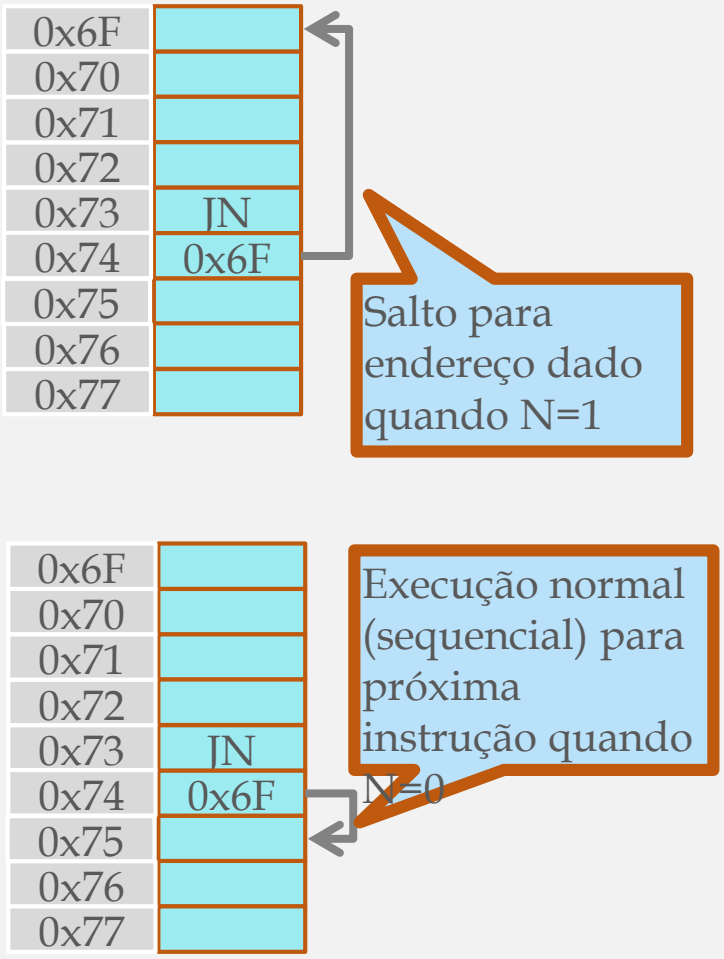
JMP::Jump

- **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.



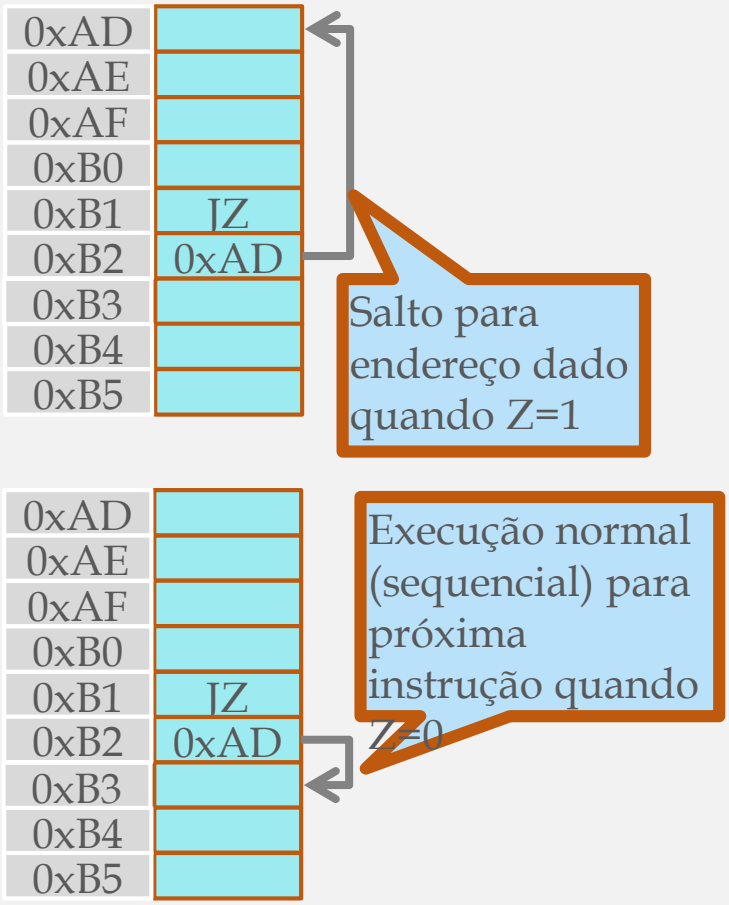
JN::Jump if Negative
JZ::Jump if Zero

- **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



Subprogramas

Estratégia de divisão de programas em partes menores, que podem simplificar o desenvolvimento e também promover o reuso do código.

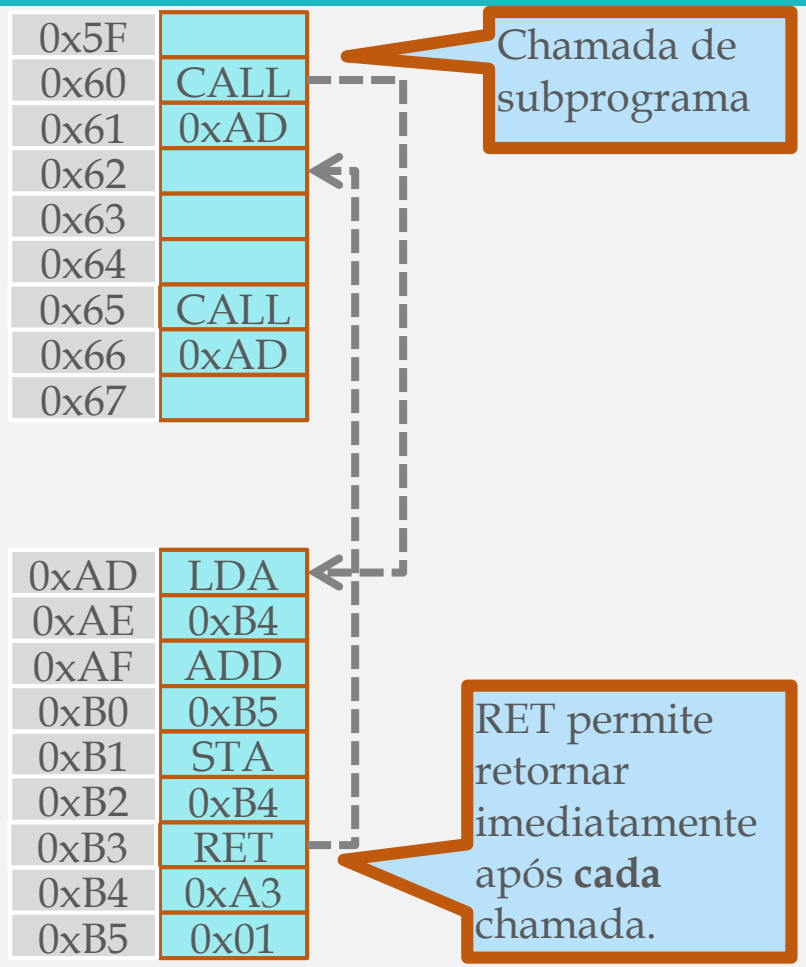
Subprogramas

- Um ***subprograma*** (ou ***sub-rotina***) é um trecho de código, *com responsabilidades próprias*, que pode ser utilizado repetidas vezes.
- São conhecidos como:
  - *funções*,
  - *métodos* ou
  - *procedimentos*
- nas linguagens de programação.

- Tem como propósitos:
  - Evitar repetições;
  - Simplificar o desenvolvimento;
  - Promover o reuso do código;
  - Simplificar a manutenção;
  - Além de reduzir o tamanho do código.

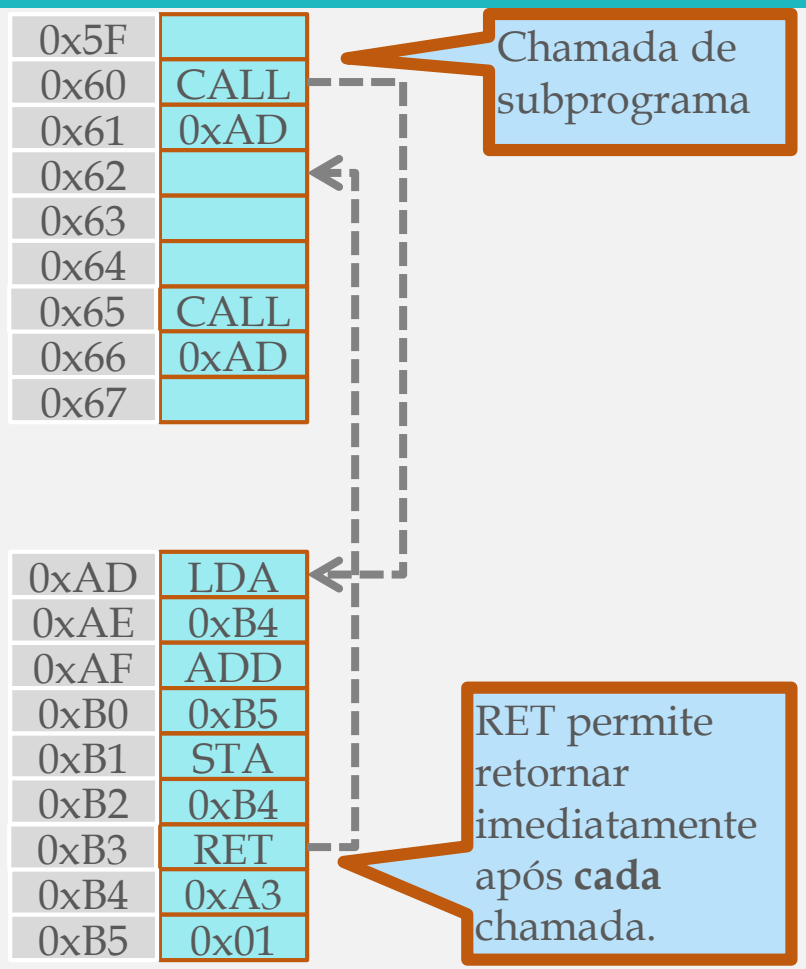
Subprogramas

- Para funcionem corretamente, os subprogramas exigem duas instruções:
  - CALL
    Efetua a chamada do subprograma.
  - RET
    Efetua o retorno do subprograma.



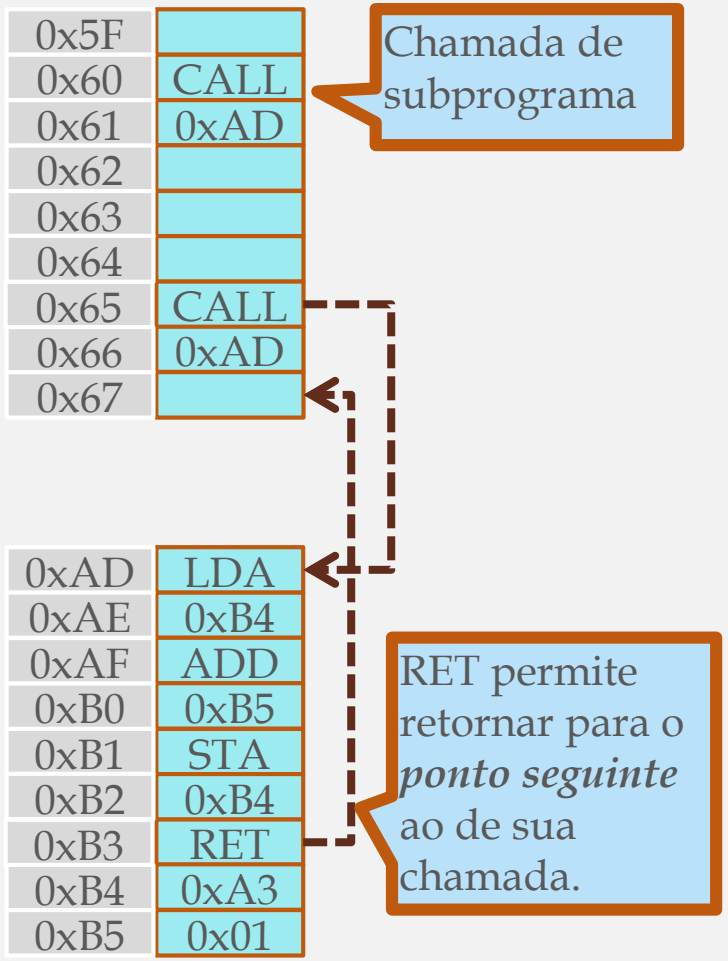
CALL::Subprogram Call
RET::Subprogram Return

- **CALL** *end*
- A instrução **CALL** desvia a execução do programa para o endereço dado incondicionalmente.
- **CALL** também armazena o endereço seguinte (ponto de retorno) na *pilha*.



CALL::Subprogram Call
RET::Subprogram Return

- **RET**
- A instrução **RET** desvia a execução do programa para o endereço seguinte ao da última instrução **CALL**.
- Tais endereços (pontos) de retorno são armazenado na *pilha*.



Subprogramas

CALL

- O (sub)programa que executa uma instrução **CALL** é denominado *caller* ou acionador.
- A execução de uma instrução **CALL** é denominada:
  - chamada;
  - invocação; ou
  - acionamento de subprograma.

RET

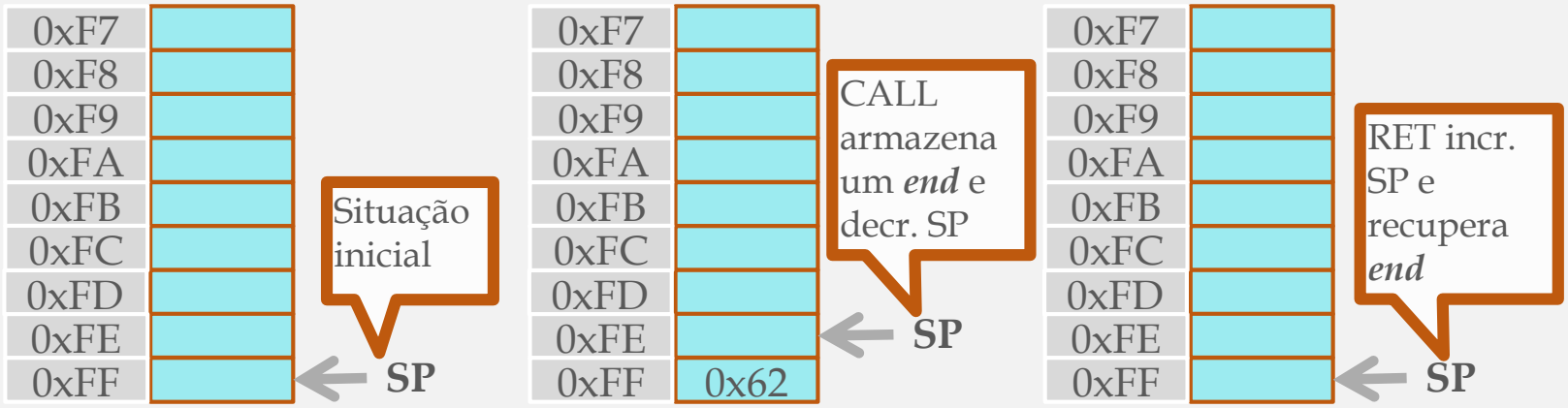
- O uso da instrução **RET** garante que o subprograma retorna para instrução seguinte ao **CALL** utilizada para invocação do subprograma.
- O retorno do subprograma é sempre para seu *caller*/acionador.

Pilha de Endereços de Retorno

- O funcionamento das instruções CALL e RET requer o armazenamento de um ou mais endereços de retorno.
- É inconveniente armazenar estas informações no processador.
- Utiliza-se uma região de memória particular para este fim (em geral a porção final).
- Um registrador especial do processador controla o uso desta região:
  - *Stack Pointer* (SP)

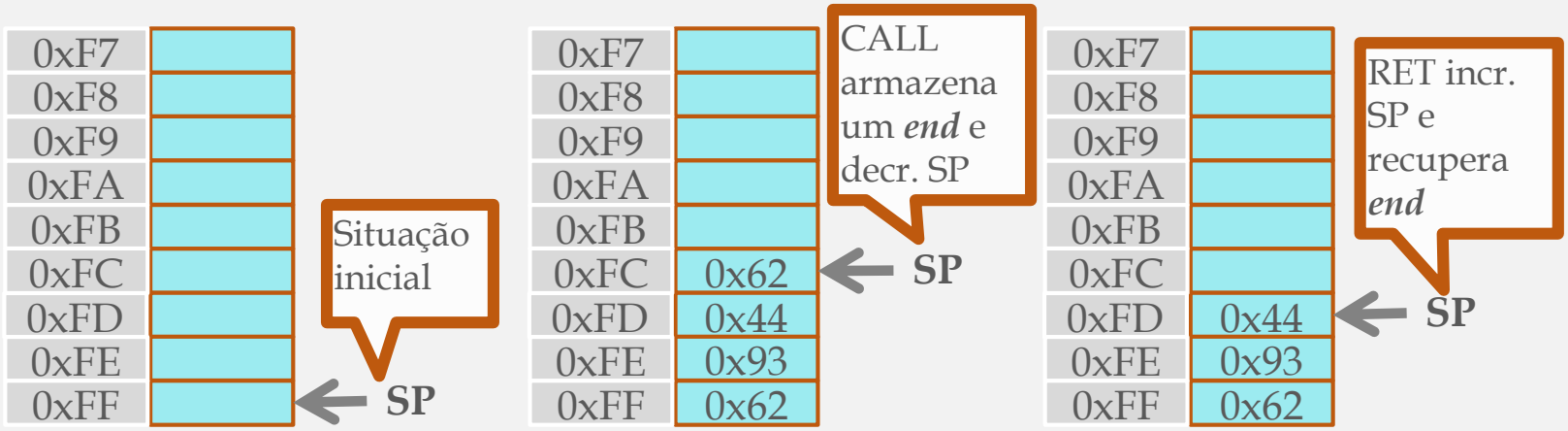
Pilha de Endereços de Retorno

- **SP** inicialmente aponta para última posição da memória.
- **CALL** armazena o endereço de retorno na posição indicada por **SP** e decrementa **SP**.
- **RET** incrementa **SP** e recupera o endereço de retorno.



Pilha de Endereços de Retorno

- O uso sucessivo de **CALL** armazena uma série de endereços de retorno nas posições indicadas por **SP**.
- Como **SP** é decrementado, forma-se uma pilha de endereços de retorno.
- **RET** incrementa **SP** e recupera sempre o endereço de retorno que corresponde ao último **CALL**.



Exemplo

Programa

- Programa que, a partir do conteúdo de duas variáveis A e B, realiza:
  - A = 2*A + 1
  - B = 2*B + 1

SubPrograma

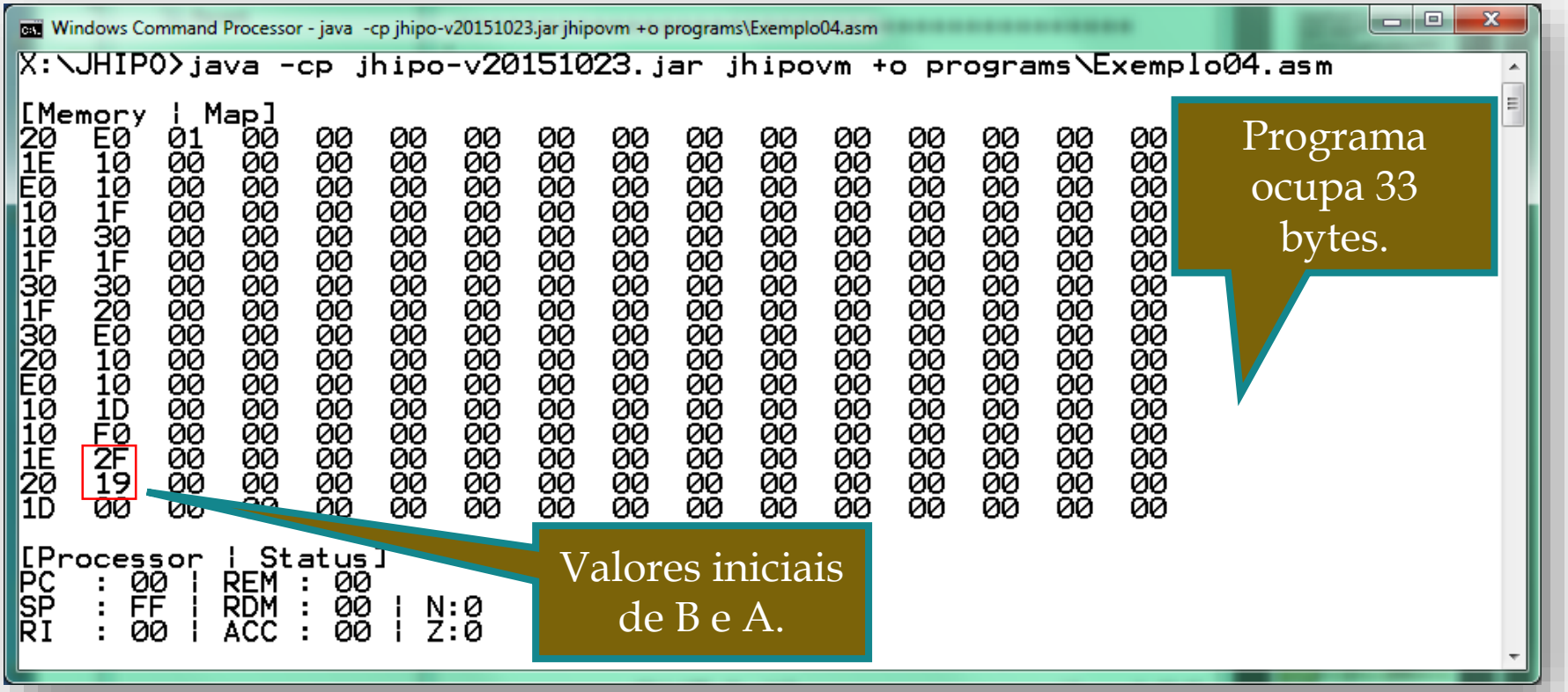
- É facilmente observável que o programa realiza, repetidas vezes:
  - f(x) = 2*x + 1

Exemplo:: Solução Sequencial

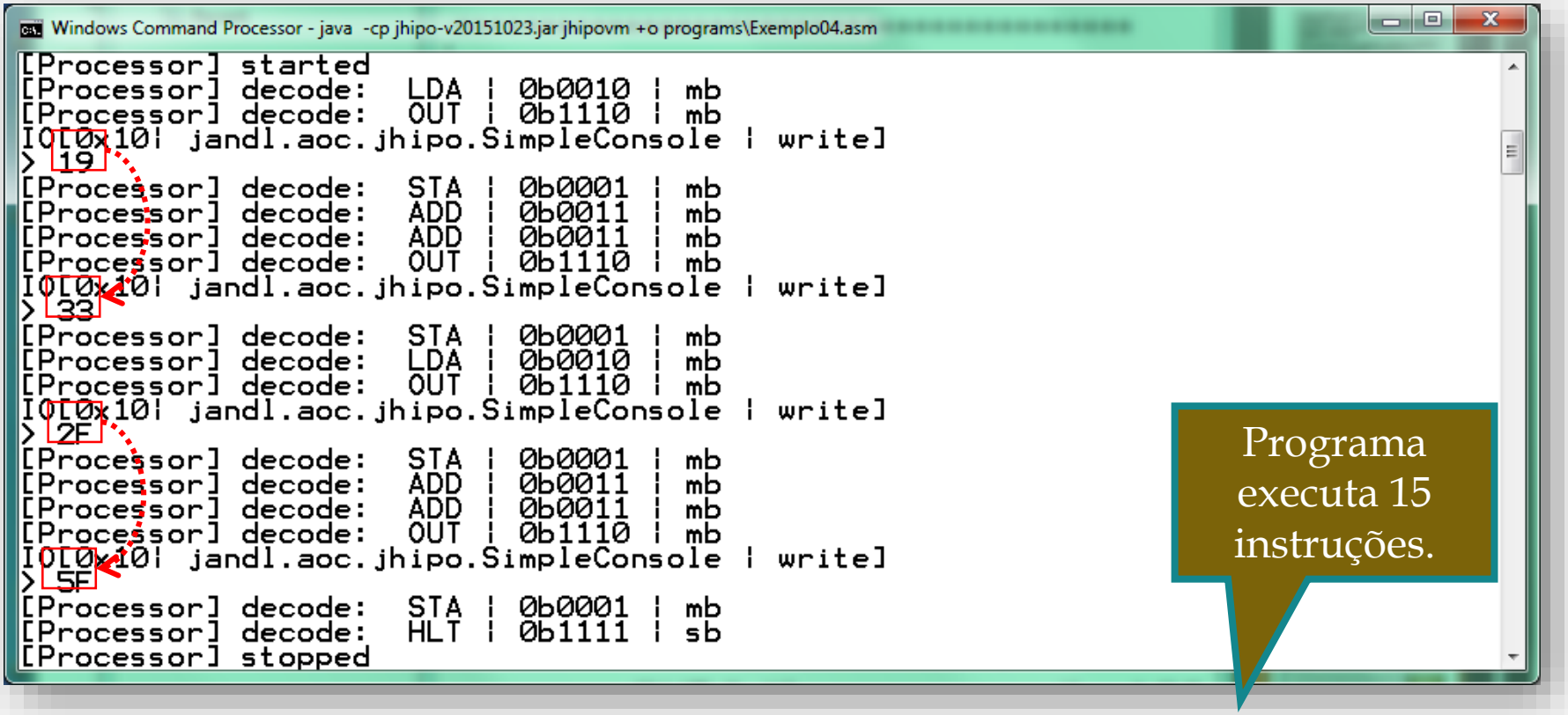
Trecho de código repetido!

```
; Programa principal
INIT:   LDA A       ; carrega ACC com  A
        OUT 0x10    ; exibe ACC (valor A) na saída padrão
        STA AUX     ; armazena ACC em AUX
        ADD AUX     ; acrescenta AUX ao ACC (faz AUX*2)
        ADD UM      ; soma UM ao ACC
        OUT 0x10    ; exibe ACC (valor A) na saída padrão
        STA A       ; armazena ACC em A
        LDA B       ; carrega ACC com B
        OUT 0x10    ; exibe ACC (valor B) na saída padrão
        STA AUX     ; armazena ACC em AUX
        ADD AUX     ; acrescenta AUX ao ACC (faz AUX*2)
        ADD UM      ; soma UM ao ACC
        OUT 0x10    ; exibe ACC (valor A) na saída padrão
        STA B       ; armazena ACC em B
        HLT         ; fim
; Variáveis
B:      0x2F        ; variável B com constante 47(decimal)
A:      0x19        ; variável A com constante 25(decimal)
AUX:    0x00        ; variável auxiliar
UM:     0x01        ; constante 1(decimal)
```

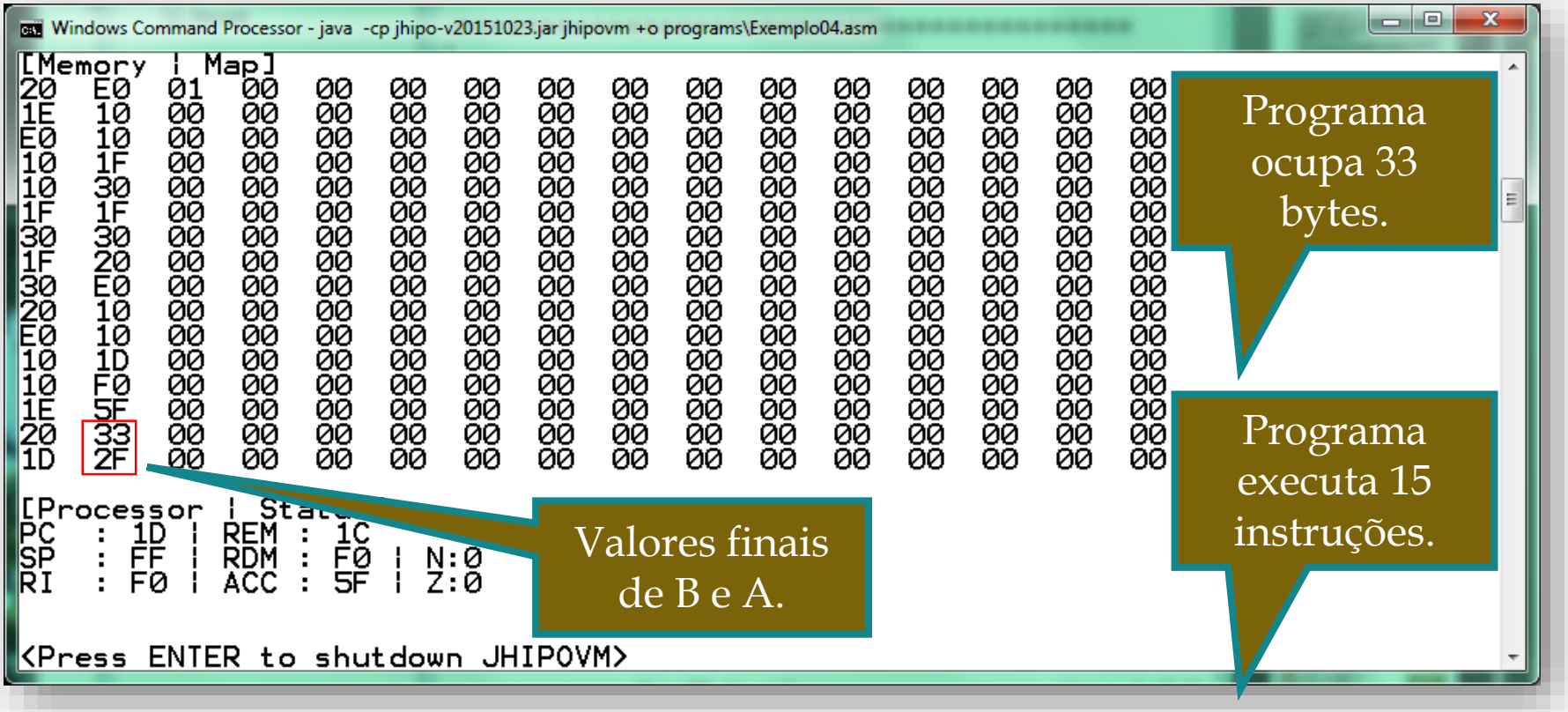
Exemplo:: Solução Sequencial



Exemplo:: Solução Sequencial



Exemplo:: Solução Sequencial



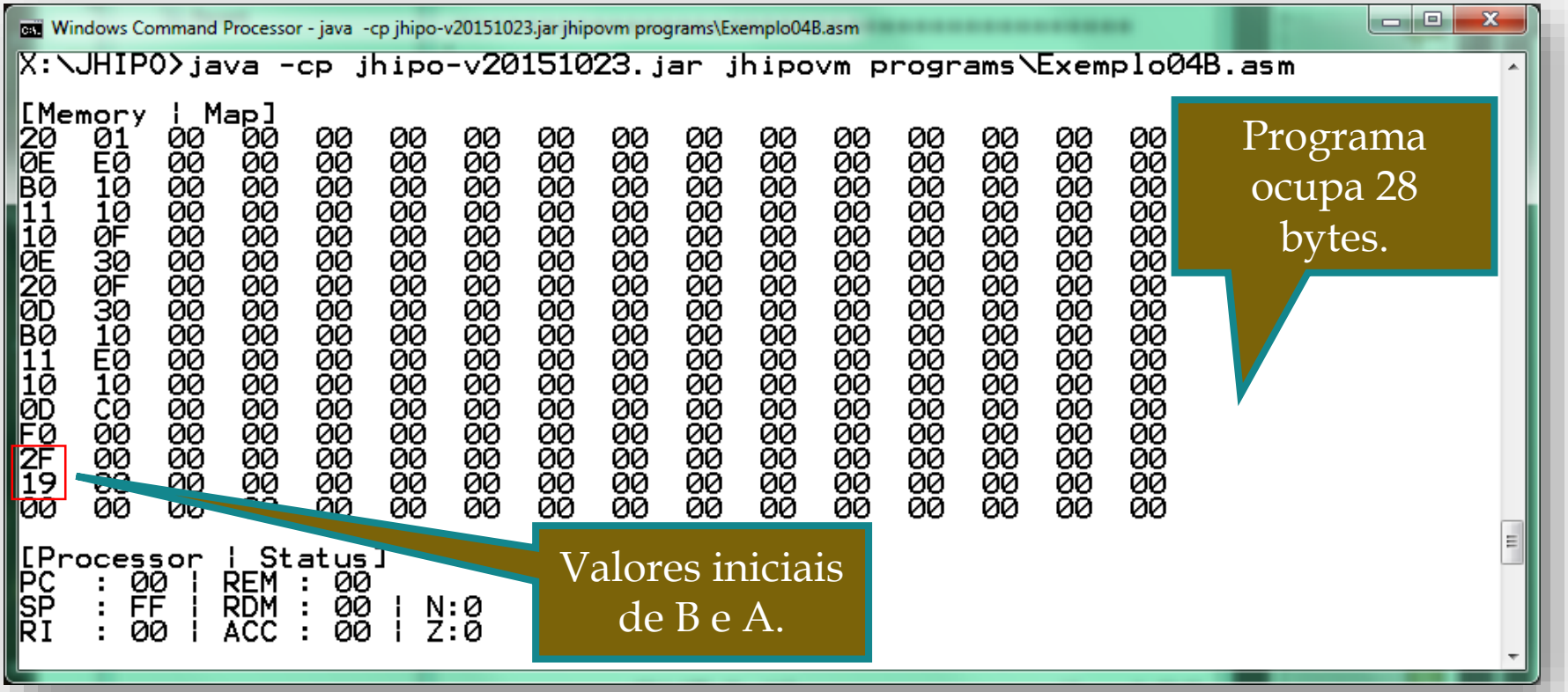
Exemplo::Solução com Subprograma

Acionamento de subprograma simplifica código original!

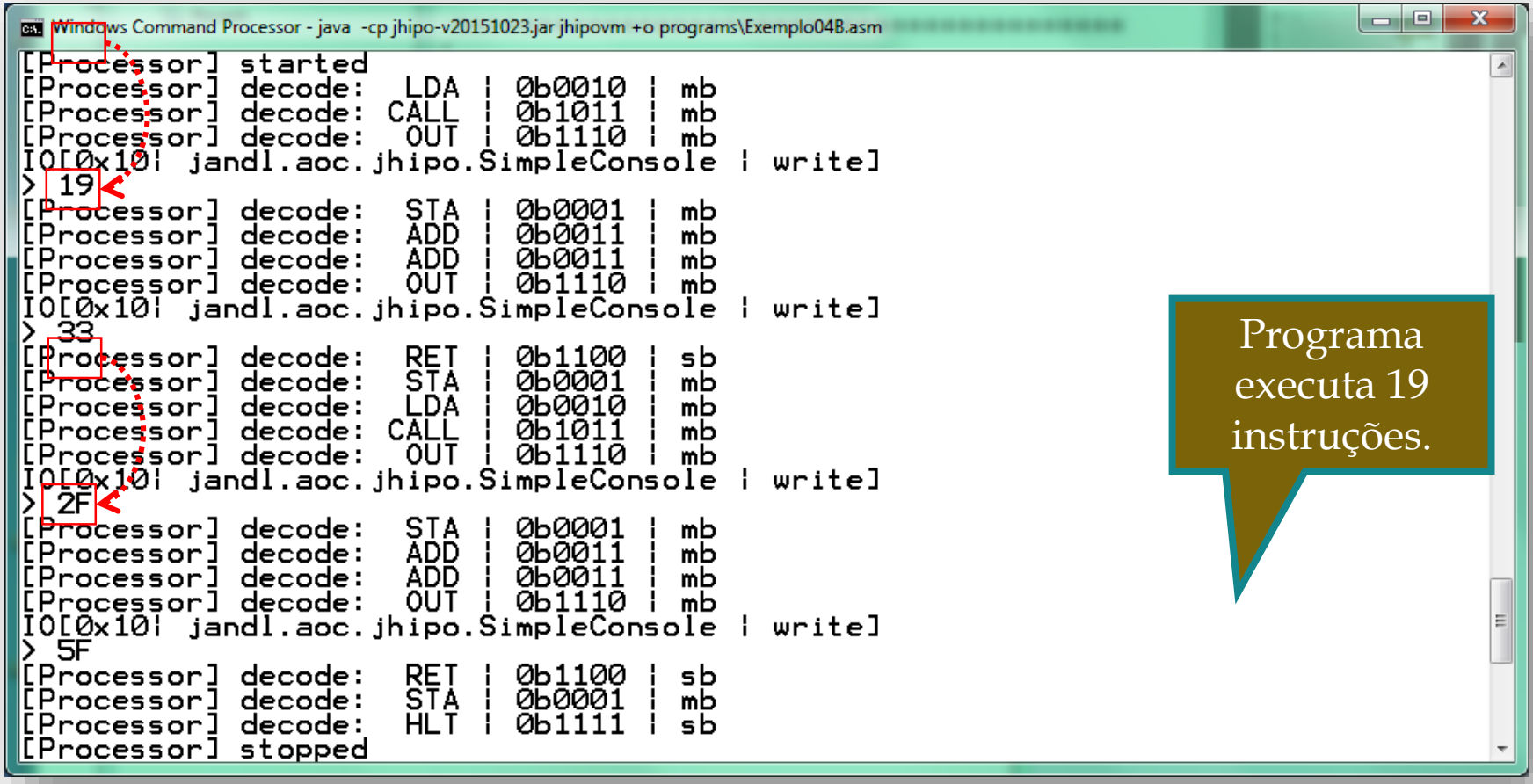
Subprograma contém trecho de código que será usado repetidas vezes!

```
; Programa principal
INIT:   LDA A           ; carrega ACC com  A
        CALL SUBPRG     ; aciona subprograma SUBPRG
        STA A           ; armazena ACC em A
        LDA B           ; carrega ACC com B
        CALL SUBPRG     ; aciona subprograma SUBPRG
        STA B           ; armazena ACC em B
        HLT             ; fim
; Variáveis
B:      0x2F            ; variável B com constante 47(decimal)
A:      0x19            ; variável A com constante 25(decimal)
; Subprograma SUBPRG
AUX:    0x00            ; variável auxiliar
UM:     0x01            ; constante 1(decimal)
SUBPRG: OUT 0x10        ; exibe ACC (valor B) na saída padrão
        STA AUX         ; armazena ACC em AUX
        ADD AUX         ; acrescenta AUX ao ACC (faz AUX*2)
        ADD UM          ; soma UM ao ACC
        OUT 0x10        ; exibe ACC (valor A) na saída padrão
        RET             ; retorna ao progr. acionador (caller)
```

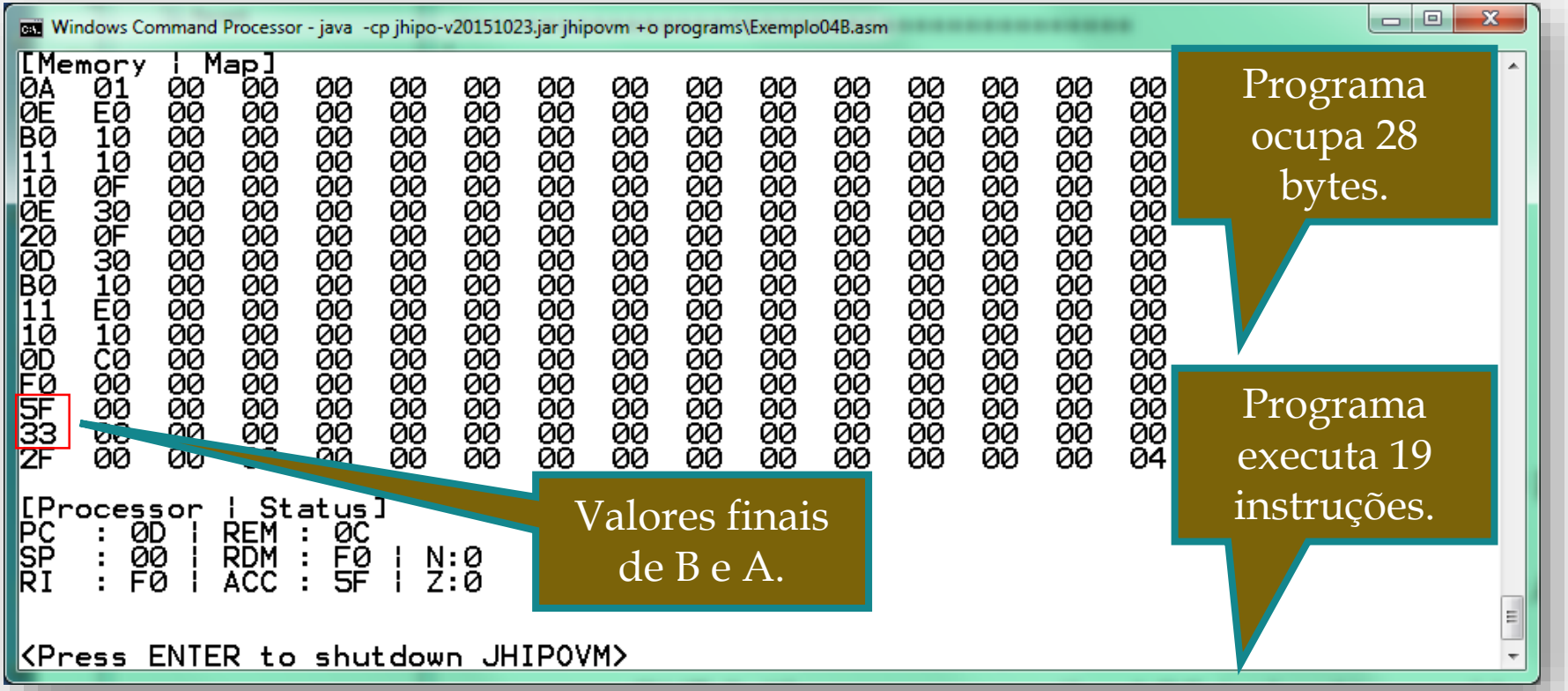
Exemplo::Solução com Subprograma



Exemplo::Solução com Subprograma



Exemplo::Solução com Subprograma



Exercícios de Fixação

Exercícios

1. Escreva um programa que utilize um subprograma para efetuar a leitura de um valor menor do que 5. A leitura deve ser repetida até que o valor atenda a condição estabelecida. O valor lido deve ser exibido no programa principal (e não no subprograma).
2. Escreva um programa que some cinco valores menores do que 5, obtidos por meio de um subprograma como do exercício anterior. A soma deve ser exibida no programa principal.