PCS3616

Programação de Sistemas

(Sistemas de Programação)

Semana 9, Aula 16

Monitor *Batch* Simples

Programação em linguagem de alto nível e em linguagem de montagem

Escola Politécnica da Universidade de São Paulo

Roteiro

1. Ambiente de execução
   - Retrospecto
2. Montador *Batch* simples
   - Estrutura
   - Linguagem de comandos

Programação de Sistemas – Visão Geral

- Neste ponto de evolução, podem ser visualizadas a programação de sistemas e as interfaces com outros programas básicos de sistema, como é o caso dos compiladores e sistemas operacionais.
- Uma **linguagem de alto nível** qualquer (como, por exemplo, Java) permite que o programador tenha preocupação apenas com o problema e com as abstrações que melhor resolvam o problema.
  - Um compilador que traduz um texto escrito em linguagem de alto nível para alguma forma intermediária
  - Alternativamente, um interpretador pode analisar e executar diretamente as operações especificadas no programa-fonte
  - Em ambos os casos, no momento da execução do programa, um ambiente de execução provê, na máquina hospedeira, um conjunto de funcionalidades que completam o programa-objeto gerado pelo compilador, ou então são diretamente acionadas pelo interpretador.

Processo de Execução de Linguagens de Alto Nível

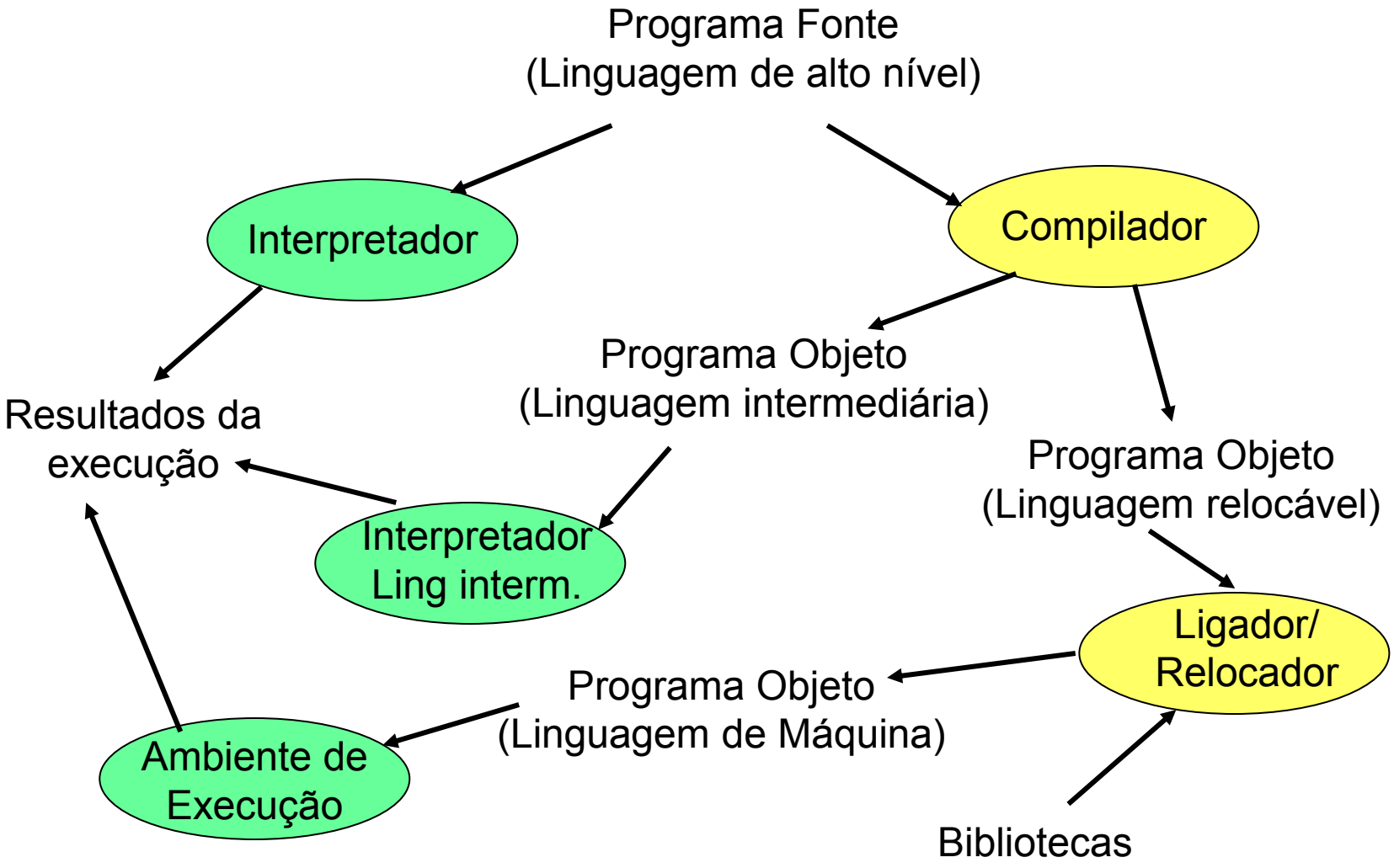
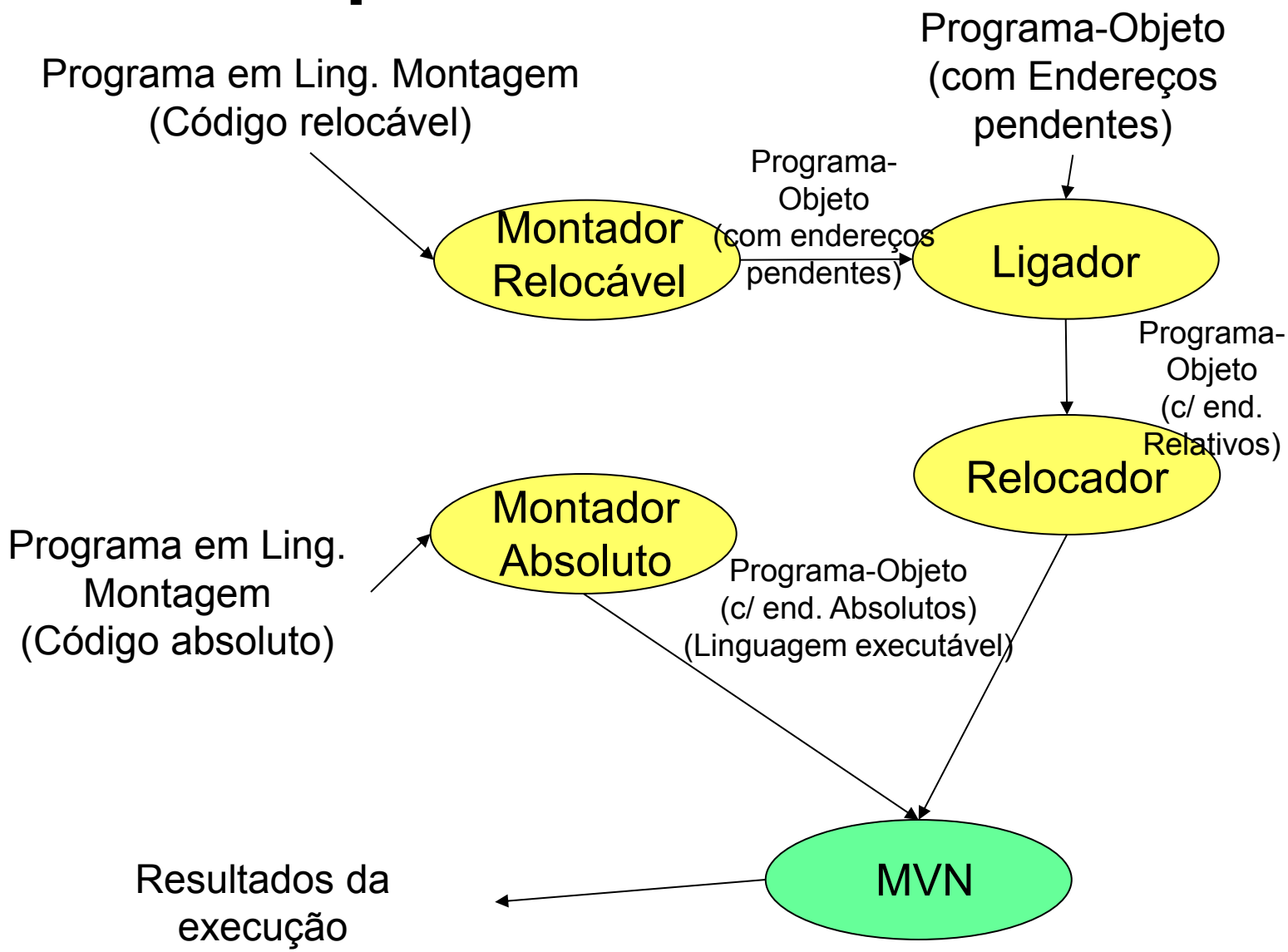


Diagrama do Sistema Desenvolvido na Disciplina até o momento

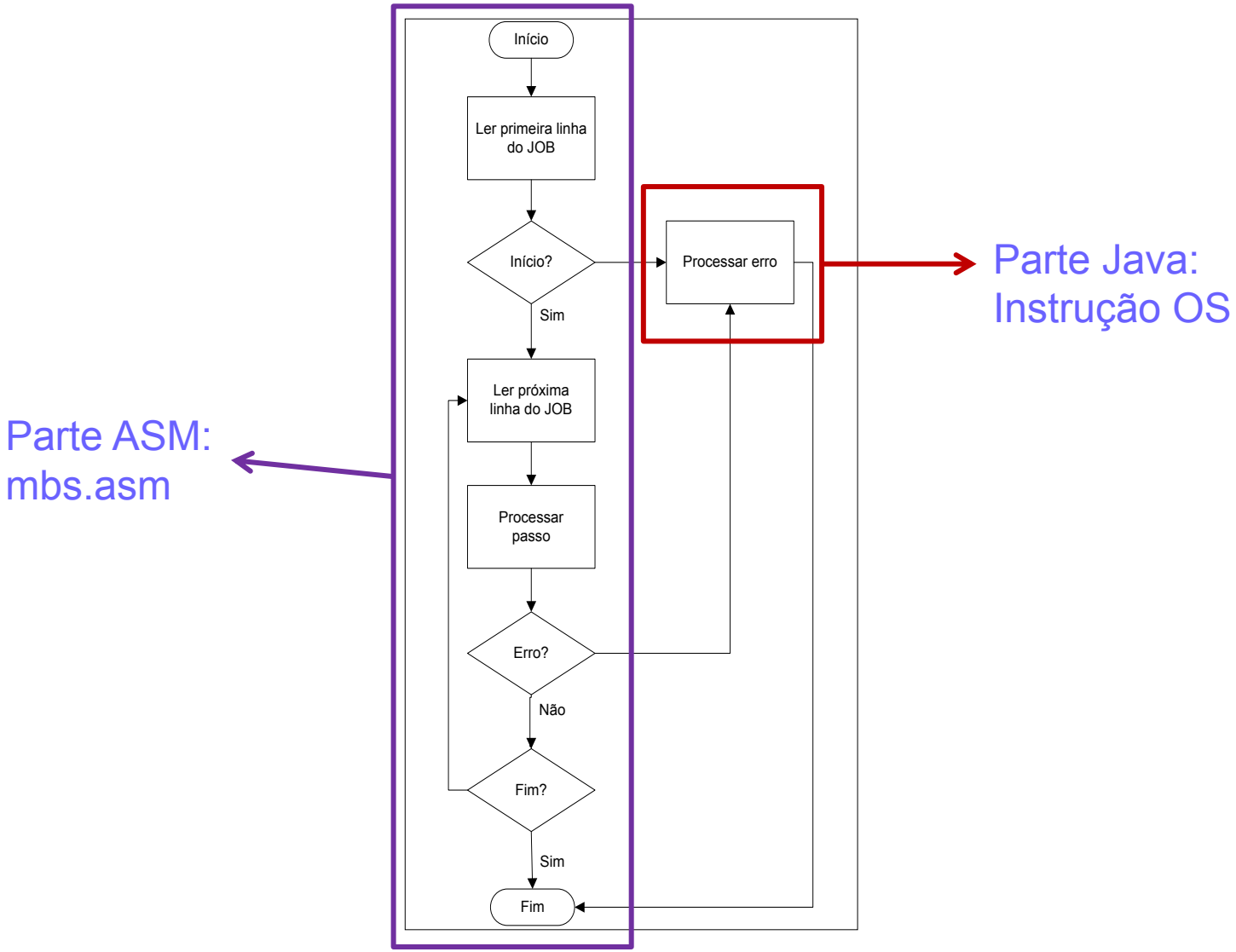


Monitor Batch Simples (MBS)

- O Monitor é um programa de sistema que controla a execução de tarefas (*jobs*) em um computador. Em um Monitor *Batch* (em lote), os *jobs* são processados sequencialmente.

- Assim, ele pode ser visto como um “sistema operacional” bastante rudimentar.

Fluxograma Básico do MBS



Sintaxe da Linguagem de Controle do MBS

arquivo_batch ::= <início>EOL{<comando>EOL}<final>
início ::= “//JB”
comando ::= “//”<cmd>
cmd ::= “DU” EOL <args_DUMP> | “LO” EOL <args_LOAD>
args_DUMP ::= <tamanho_bloco>bb<endereço_inicial>bb<tamanho_total>bb
<endereço_primeira_instrução>bb<LU>
args_LOAD ::= <LU>
LU ::= 0000..00FF/* Unidade Lógica (LU) do arquivo */
tamanho_bloco ::= 0002..0200
endereço_inicial ::= 0000..0FFE
tamanho_total ::= 0000..0800
endereço_primeira_instrução ::= /* Endereço da primeira instrução executável
do programa. Se não for um programa executável, o valor é
0xFFFF. */
final ::= “/*”

Sintaxe da Linguagem de Controle do MBS

<Início>EOL

{<Comando>EOL}

<Final>

- EOL representa uma quebra de linha no arquivo.
  - Windows: CRLF – 0x0D 0x0A ou \r\n
  - Linux: LF – 0x0A ou \n
- Espaços em branco são descritos com o caractere “espaço” (“b”).

Exemplo de um arquivo MBS

```
//JB
//DU
0008bb0F00bb0022bbFFFFbb0001
//LO
0002
//LO
0005
/*
```

Obs.: lembre-se que “b” significa **espaço**

Chamada de Supervisor (SVC) na MVN

- Relembrando:

OS	Parâmetros	Operação

Exemplo: OS /01FF

OS	Instrução F (1 byte)
Parâmetros	Número de parâmetros (1 byte)
Operação	Código de operação [00 a FF] (2 bytes)

- Parâmetros devem ser posicionados no endereço de memória anterior ao da chamada, na forma de pilha (primeiro parâmetro vem logo acima da chamada OS)
  - Ex.: operação FF com dois parâmetros.
    - Passando ULs 1 e 2 como parâmetros:

```
UL2     K     /0002
UL1     K     /0001

        OS    /02FF
```

Exceções do MBS

- Usaremos a instrução OS /00EE (F0EE em hexa) para que a MVN imprima mensagens de sucesso ou erros caso estes ocorram.
- **Detalhes**:
  - Mensagens de sucesso/erro serão colocadas em um arquivo de log (**Disco cuja unidade lógica é FF**).
  - Códigos de sucesso/erro devem ser colocados no acumulador, e devem ser mantidos no acumulador após a execução de F0EE.
  - MVN não deve ter sua execução interrompida bruscamente caso o Disco na UL FF não exista: caso tente-se escrever no log e não exista um arquivo correspondente, execução deve continuar normalmente.

Exceções do MBS

Situação	Código (valor passado no acumulador)	Mensagem de texto a ser escrita no log	Instrução
Ausência de erros	00	OK	OS /0EE
Erro ou ausência: Indicador de início do job	01	ER:JOB	OS /0EE
Erro: nome do comando não reconhecido	02	ER:CMD	OS /0EE
Ausência de algum dos argumentos – separados por dois brancos – dos comandos	03	ER:ARG	OS /0EE
Erro ou ausência: Indicador de final do job	04	ER:END	OS /0EE

- Ex.: caso ocorra o erro 01, o MBS deve:

  (1) colocar no acumulador o valor 0001, e então

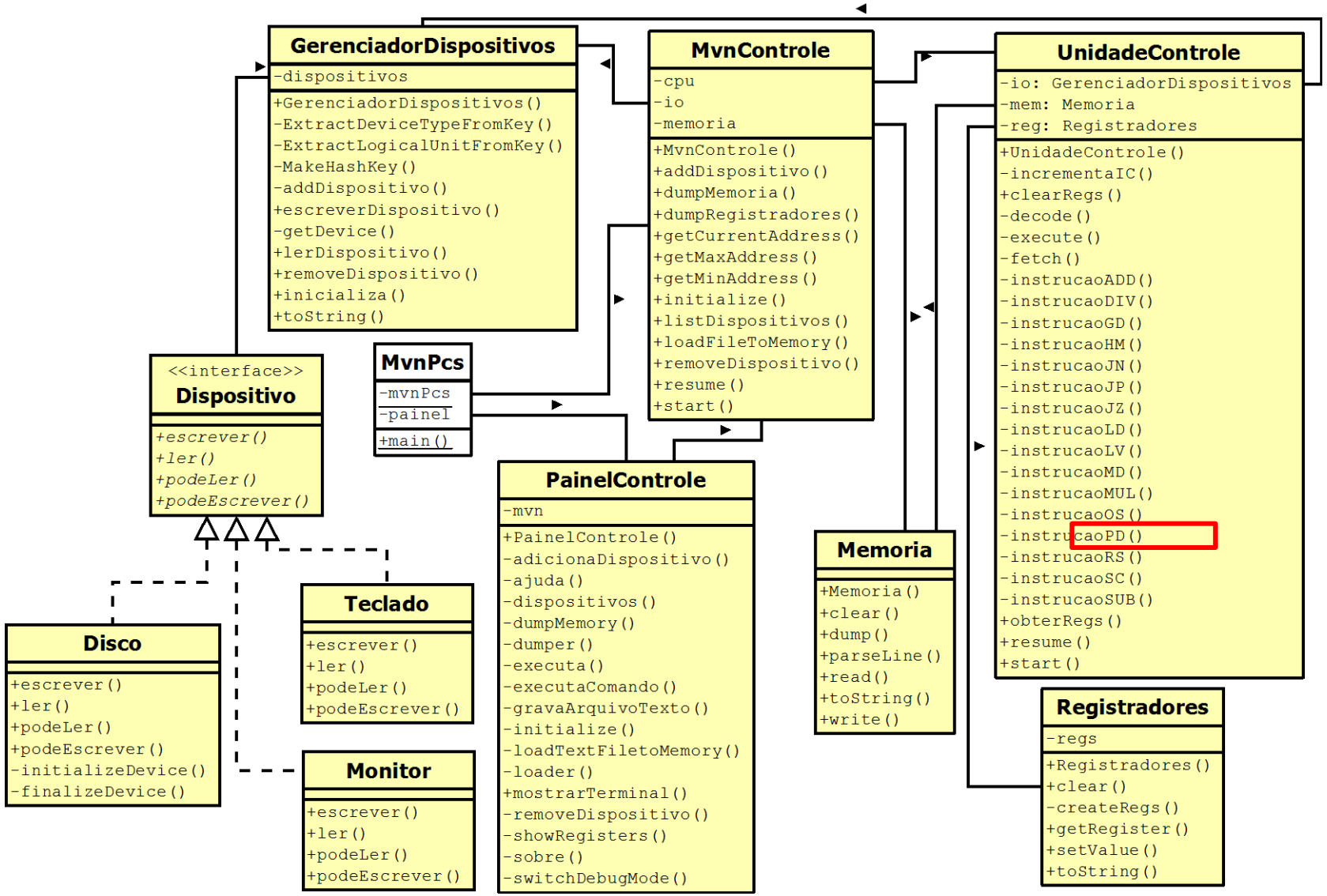
  (2) executar o comando “OS /00EE”

  ➔ O tratamento do comando OS é então feito pela MVN (i.e., no código Java)

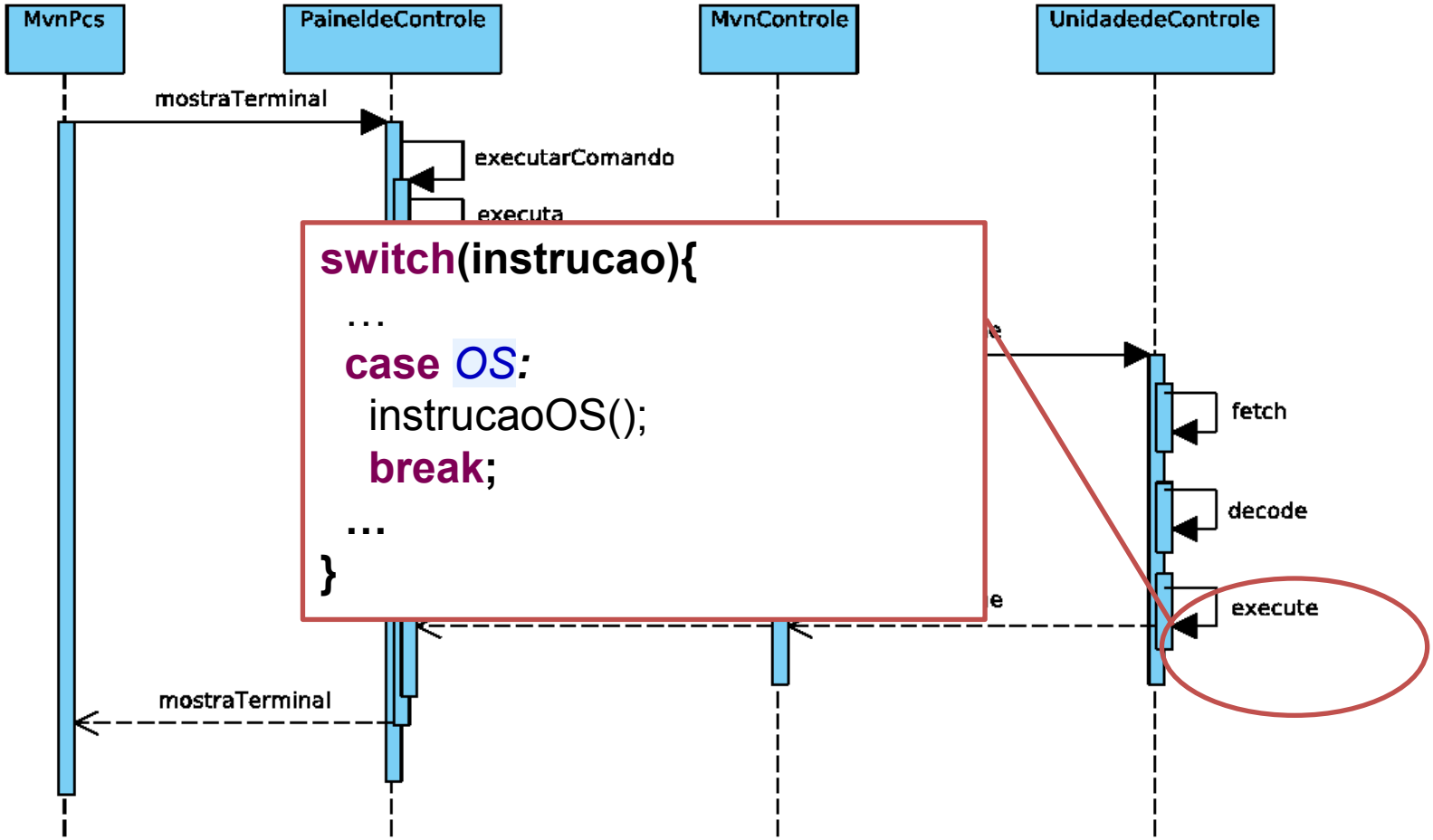
Exceções do MBS

- Casos de teste
  - **Erro ou ausência: Indicador de início do job**
    - Inicie o job com “JB” e com “//JJ”
  - **Erro: nome do comando não reconhecido**
    - Use os comandos “DU”, “//DUMP” e “//NO”
  - **Ausência de algum dos argumentos – separados por dois brancos – dos comandos**
    - Passe como argumentos do DU: “0001bb”, “” e “0008bb0F00bb0022bbFFFF”
    - Passe como argumentos do LO: “0001bb0002” e “”
  - **Erro ou ausência: Indicador de final do job**
    - Finalize o job com “”

Macro Arquitetura da MVN



Macro Arquitetura da MVN



Monitor Batch Simples (MBS)

- Estendemos o MBS com um comando que permite a execução de programas MVN: o comando EX (EXecutar).
- Para a execução de um programa, o MBS:
  - Invoca o programa *Loader* , carregando na memória um programa-objeto gravado em um arquivo. Este arquivo deve conter em sua última palavra o endereço da primeira instrução executável do programa.
  - Em seguida, invoca o simulador MVN para proceder à execução o programa carregado a partir do endereço definido no acumulador.
  - Retoma seu fluxo de execução ao final do programa carregado.

Sintaxe da Linguagem de Controle do MBS

arquivo_batch ::= <início>EOL{<comando>EOL}<final>

início ::= “//JB”

comando ::= “//”<cmd>

cmd ::= “DU” EOL <args_DUMP> | “LO” EOL <args_LOAD>

| “EX” EOL <args_LOAD>

args_DUMP ::= <tamanho_bloco>bb<endereço_inicial>bb<tamanho_total>bb
<endereço_primeira_instrução>bb<LU>

args_LOAD ::= <LU>

LU ::= 0000..00FF/* Unidade Lógica (LU) do arquivo */

tamanho_bloco ::= 0002..0200

endereço_inicial ::= 0000..0FFE

tamanho_total ::= 0000..0800 /* Tamanho em palavras (*words*) */

endereço_primeira_instrução ::= /* Endereço da primeira instrução executável
do programa. Se não for um programa executável, o valor é
0xFFFF. */

final ::= “/*”

Exemplo de um arquivo MBS

- Vamos supor que um arquivo, com unidade lógica 2, seja a imagem de um programa que tenha 26 palavras (1A em hexadecimal) e cuja primeira instrução executável deva estar no endereço 0x0F00.
- A título de exemplo, vamos supor que, inicialmente, esse arquivo seja gerado pelo programa *Dumper*. Um exemplo de arquivo de texto a ser submetido ao MBS é:

```
//JB
//DU
0008bb0F00bb001Abb0F00bb0002
//EX
0002
/*
```

Obs.: lembre-se que “b” significa **espaço.**

Exceções do MBS

Situação	Código (valor passado no acumulador)	Mensagem de texto a ser escrita no log	Instruçã o
Ausência de erros	00	OK	OS /0EE
Erro ou ausência: Indicador de início do job	01	ER:JOB	
Erro: nome do comando não reconhecido	02	ER:CMD	
Ausência de algum dos argumentos – separados por dois brancos – dos comandos	03	ER:ARG	
Erro ou ausência: Indicador de final do job	04	ER:END	
Erro no comando EX: endereço da primeira instrução executável inválido	**05**	**ER:EXE**	

- Ex.: caso ocorra o erro 05, seu MBS deve:
  (1) colocar no acumulador o valor 0005, e então
  (2) executar o comando “OS /00EE”

  ➔ O tratamento do comando OS é então feito pela MVN (i.e., no seu código Java)

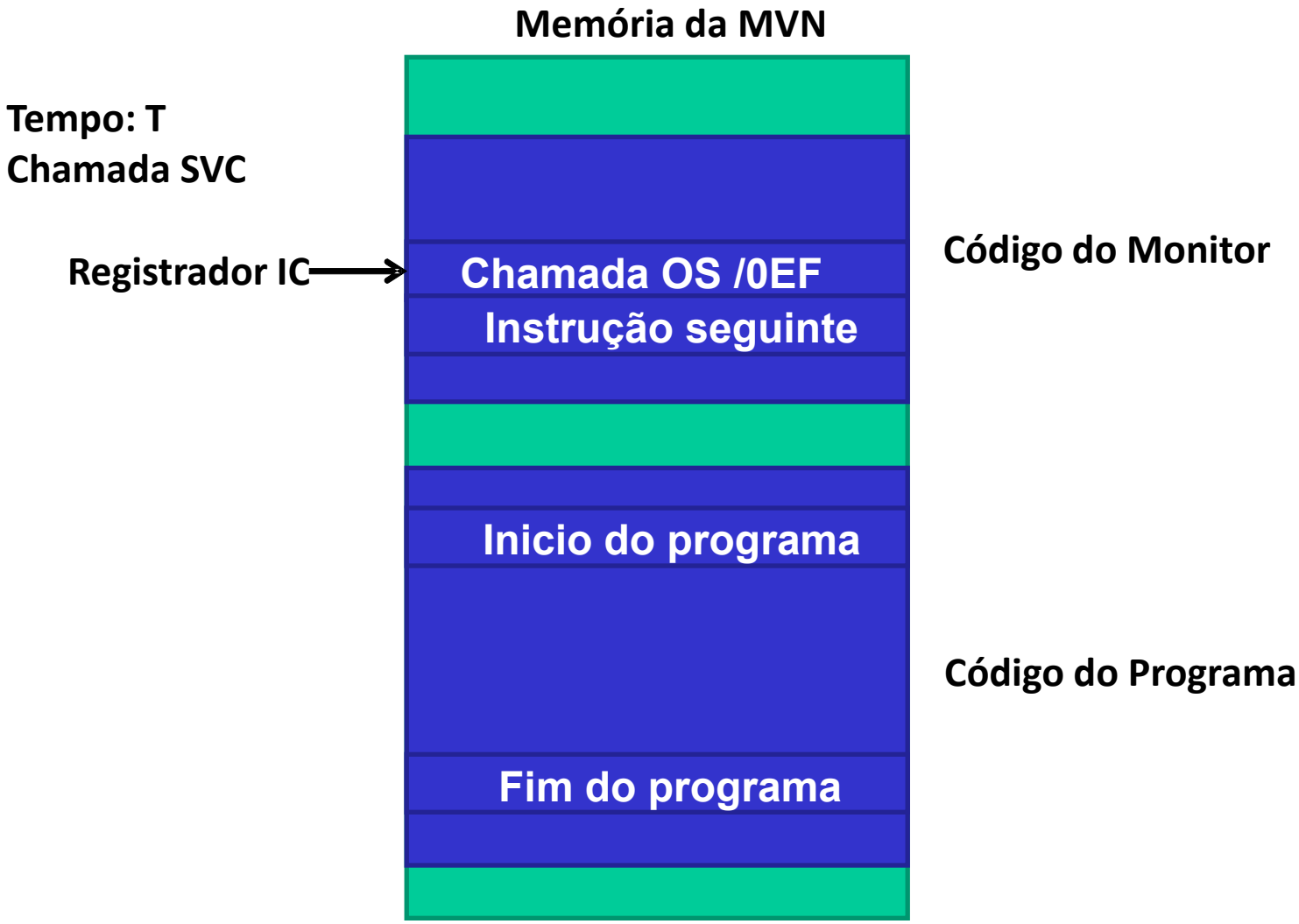
Problema Enfrentado

- Problema:
  - Se o programa do MBS simplesmente fizesse um *jump* para o ponto de entrada do programa carregado na memória, ao final deste programa a MVN pararia, pois haveria acabado de executar uma instrução HM.
  - Obviamente, não se pode admitir “gambiarras” específicas em um programa a ser executado pelo MBS, nem tampouco modificações de instruções originais da MVN! O programa carregado é um programa cujo código-fonte montado e “*linkado*” pode ser carregado e executado normalmente pelo painel de controle, com as opções ‘p’ e ‘r’.

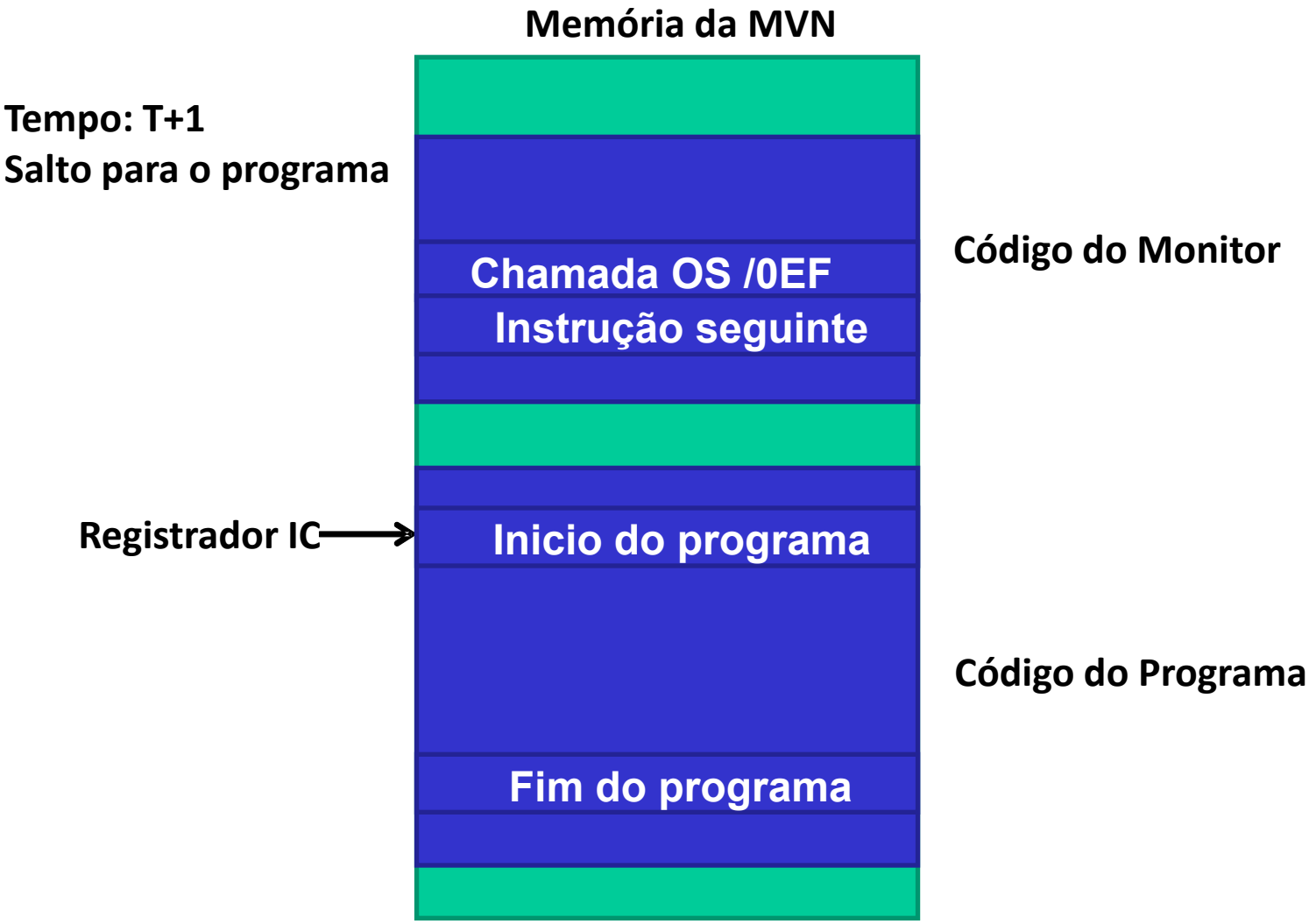
Solução do Problema

- Solução:
  - Inicialmente, o MBS deve instruir a MVN para executar o programa carregado começando no endereço armazenado no acumulador.
  - Para isso criamos uma nova chamada SVC sem parâmetros, chamada com o comando *OS /0EF*, pois basta o valor armazenado no acumulador.
  - Devemos nos lembrar que, ao término da execução do programa carregado pelo MBS, a MVN precisa retomar a execução do MBS, como ilustrado nas transparências que seguem.

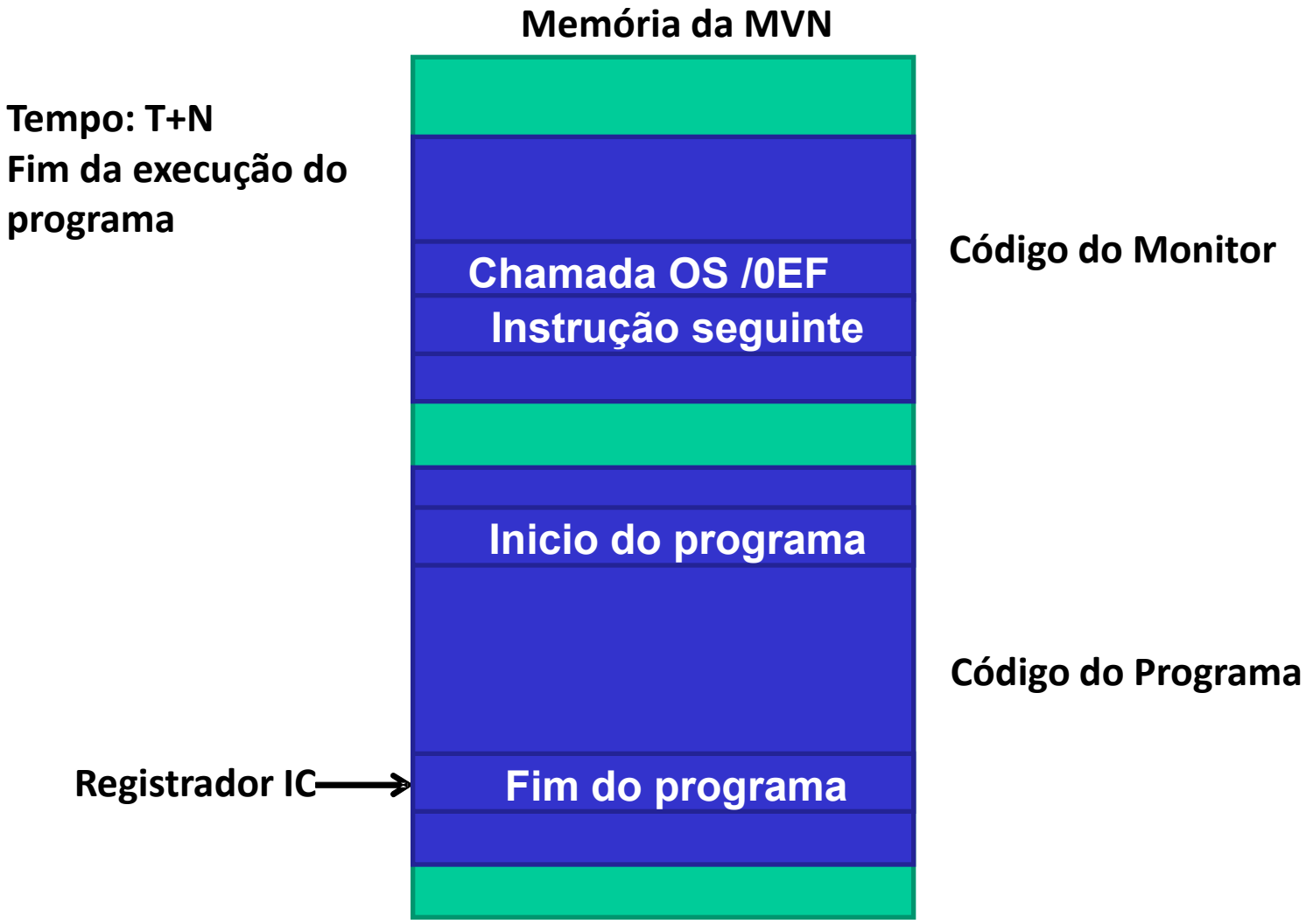
Solução


Memória da MVN
Tempo: T
Chamada SVC
Registrador IC
Chamada OS /0EF
Instrução seguinte
Inicio do programa
Fim do programa
Código do Monitor
Código do Programa

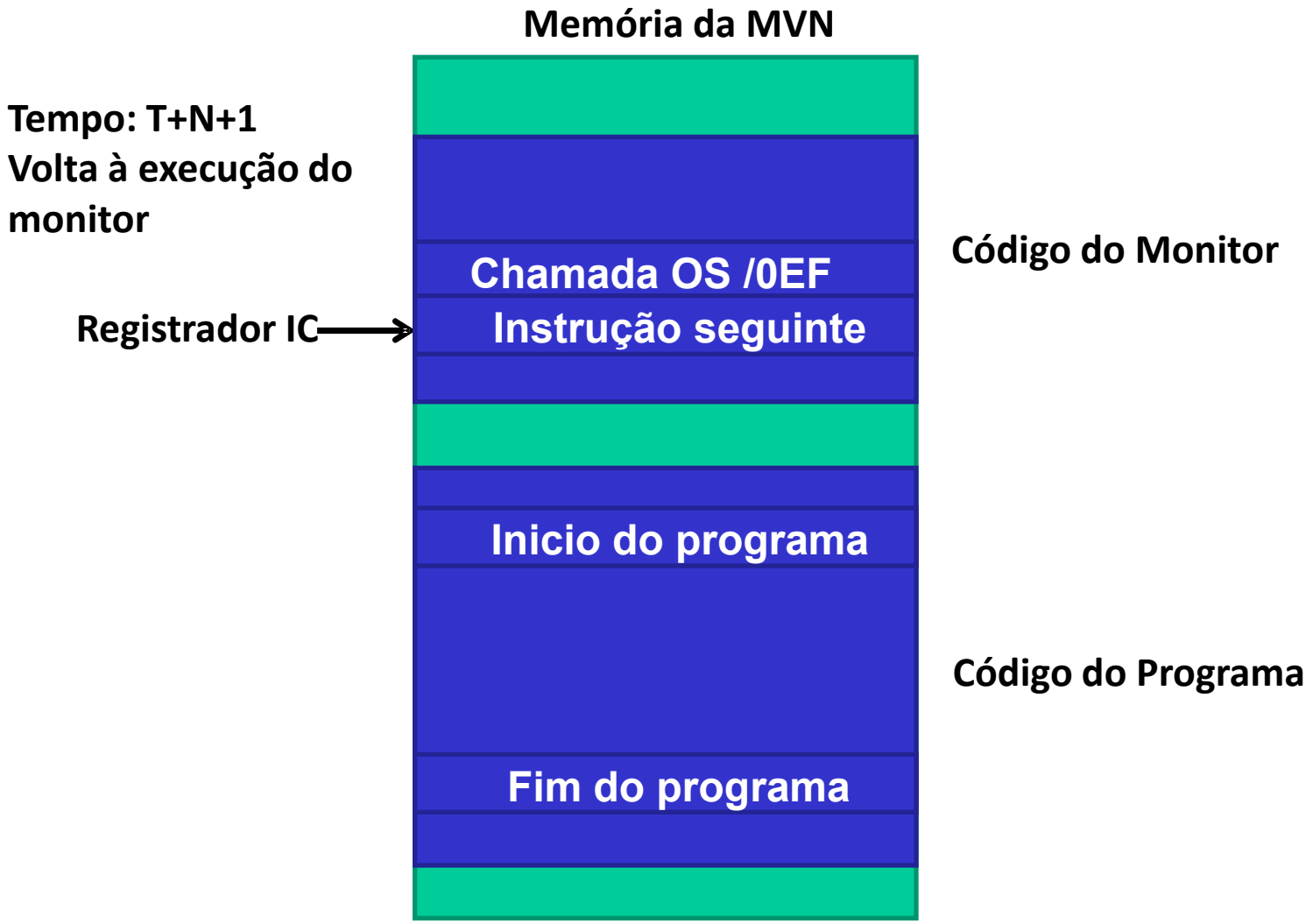
Solução


Memória da MVN
Tempo: T+1
Salto para o programa
Chamada OS /0EF
Instrução seguinte
Código do Monitor
Registrador IC
Inicio do programa
Código do Programa
Fim do programa

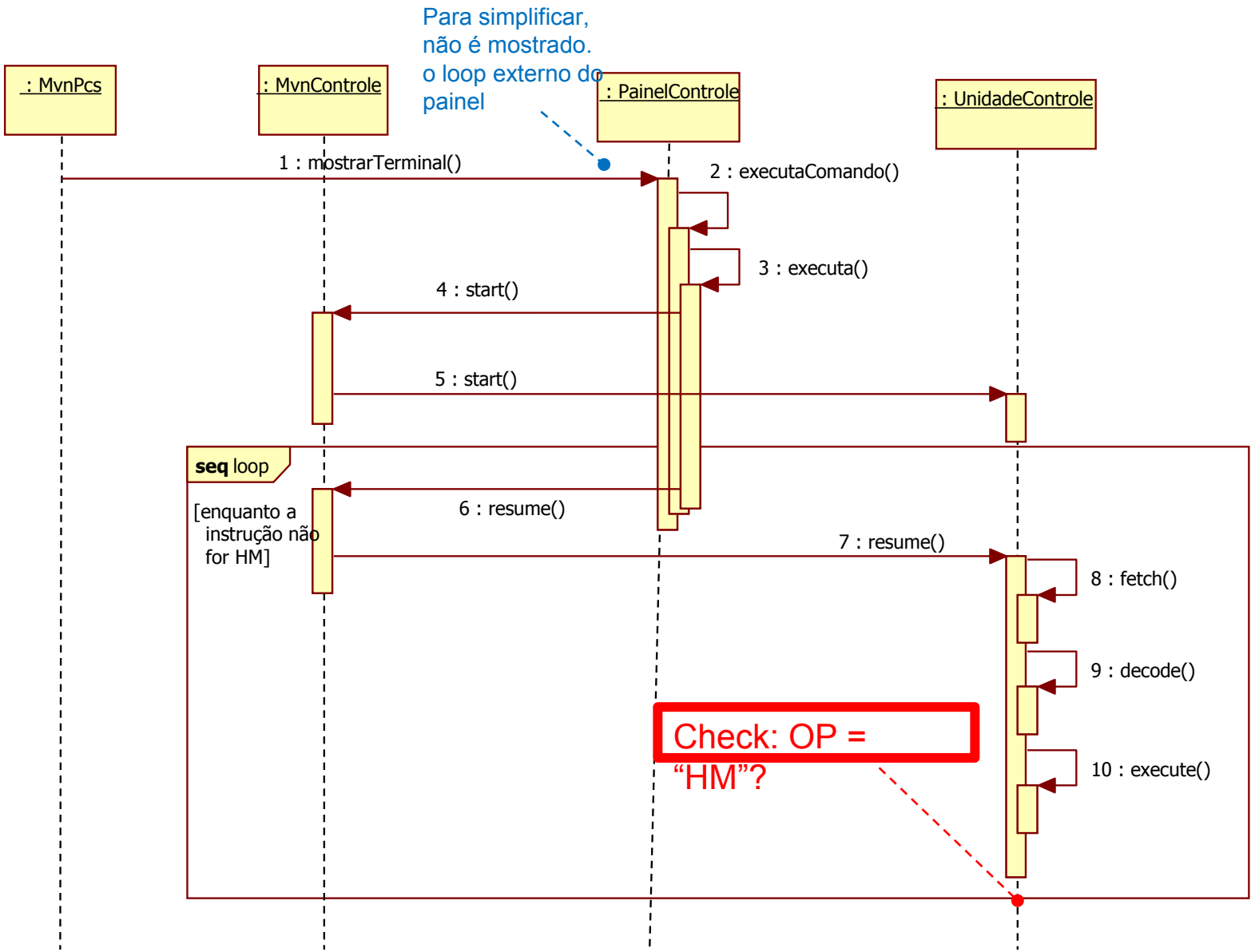
Solução


Memória da MVN
Tempo: T+N
Fim da execução do programa
Chamada OS /0EF
Instrução seguinte
Código do Monitor
Inicio do programa
Código do Programa
Registrador IC
Fim do programa

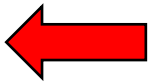
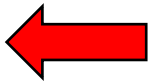
Solução


Memória da MVN
Tempo: T+N+1
Volta à execução do monitor
Chamada OS /0EF
Registrador IC
Instrução seguinte
Código do Monitor
Inicio do programa
Código do Programa
Fim do programa

Lembrete: MVN – Execução de comandos

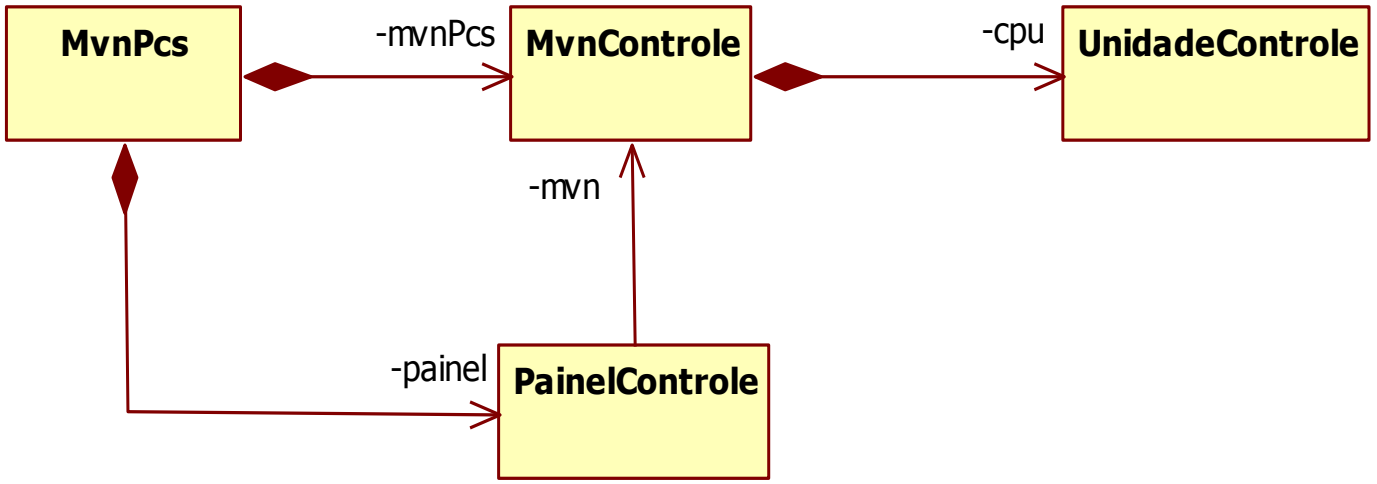


DICAS: MVN – Execução de comandos

- **`PainelControle::mostrarTerminal()`**
  - Invoca o método privado **`executaComando()`**, passando o comando do simulador e seus argumentos.
- **`PainelControle::executaComando()`** 
  - Identifica o comando do simulador. Se for o comando de executar um programa (opção 'r' do painel), então invoca o método privado **`executa()`** passando os argumentos do comando.
- **`PainelControle::executa()`**
  - Extrai os argumentos do comando e invoca o método **`start()`** da **`UnidadeControle`**, passando o endereço inicial da primeira instrução executável do programa.
  - Entra em laço que continua invocando o método **`resume()`** da **`UnidadeControle`** enquanto o registrador OP não contém a instrução HM. 
  - **`resume()`** contém o ciclo FetchDecodeExecute do simulador.

Simulador MVN

- A figura mostra o diagrama de classes de parte do simulador contendo as classes envolvidas diretamente na execução de comandos do simulador e de instruções da MVN.
- Note que, no código original da MVN, um objeto do tipo UnidadeControle não tem referência (direta) para um objeto do tipo PainelControle.



Extensão do MBS – inserção do comando Clear File (CL)

- Durante o processo de testes de um programa é sempre importante garantir que a sua saída seja sempre a mesma para uma determinada entrada. Assim, espera-se que ele tenha um comportamento determinístico.
- Entretanto, caso um programa escreva dados ao fim de um arquivo a cada vez que o mesmo seja testado, o arquivo de saída terá mais e mais informações, o que dificulta os testes do mesmo (ex.: é exatamente o que ocorre com nosso Dumper!)
- Visando resolver este problema para os programas executados no MBS, é possível estender o monitor de forma que ele seja capaz de “limpar” (reduzir o tamanho do arquivo a 0 bytes) os dados dos arquivos que serão usados pelos programas antes que estes sejam executados. Isso é feito com a adição do comando Clear File (CL).

Extensão do MBS – inserção do comando Clear File (CL)

- **Passo 1**: Criar uma chamada ao supervisor (instrução OS) de código “C0” que tem como parâmetros as unidades lógicas a serem limpas (uma ou mais).
  - Ex.: Limpando UL 03 ➔

```
UL1   K   /0003
      OS  /1C0
```

  Ex.: Limpando ULs 01 e 05

```
UL1   K   /0001
UL2   K   /0005
      OS  /2C0
```

- Em caso de erro durante a execução da operação (ex.: inexistência de unidade lógica) a chamada deve (1) cancelar todos os clears e (2) retornar 0xFFFF no acumulador

Extensão do MBS – inserção do comando Clear File (CL)

- Alguns detalhes importantes:
  - (i) Um arquivo só pode ser apagado se não houver qualquer referência (InputStream ou OutputStream) aberta a ele;
  - (ii) A chamada deve ser testada com pelo menos 2 casos:
    - Todas as unidades lógicas existentes:
      - limpa o arquivo das unidades lógicas e retorna 0x0000 no acumulador
    - Uma unidade lógica inexistente entre as passadas:
      - não realiza a operação de limpeza de qualquer UL e retorna 0xFFFF no acumulador.

Extensão do MBS – inserção do comando Clear File (CL)

- **Passo 2**: Implementação de uma extensão do MBS adicionando um comando “CL” que usa a instrução OS definida no passo 1 para limpar uma unidade lógica.
- A sintaxe da linguagem foi estendida para permitir sua definição (ver próximo slide).

Sintaxe da Linguagem de Controle do MBS

arquivo_batch ::= <início>EOL{<comando>EOL}<final>

início ::= “//JB”

comando ::= “//”<cmd>

cmd ::= “DU” EOL <args_DUMP> | “LO” EOL <args_LOAD> | “EX” EOL <args_LOAD>

| “CL” EOL <args_CL>

args_DUMP ::= <tamanho_bloco>bb<endereço_inicial>bb<tamanho_total>bb
<endereço_primeira_instrução>bb<LU>

args_LOAD ::= <LU>

args_CL ::= <LU>{bb<LU>}

LU ::= 0000..00FF/* Unidade Lógica (LU) do arquivo */

tamanho_bloco ::= 0002..0200

endereço_inicial ::= 0000..0FFE

tamanho_total ::= 0000..0800

endereço_primeira_instrução ::= /* Endereço da primeira instrução executável
do programa. Se não for um programa executável, o valor é
0xFFFF. */

final ::= “/*”

Exemplo de uso do CL

```
//JB
//CL
0003
//CL
0004bb0001bb0002
/*
```

Tabela de mnemônicos para a MVN (de 2 caracteres)

Operação **0** **Jump** Mnemônico **JP**	Operação **1** Jump if **Zero** Mnemônico **JZ**	Operação **2** Jump if **Negative** Mnemônico **JN**	Operação **3** Load **Value** Mnemônico **LV**
Operação **4** **Add** Mnemônico **+**	Operação **5** **Subtract** Mnemônico **–**	Operação **6** **Multiply** Mnemônico *****	Operação **7** **Divide** Mnemônico **/**
Operação **8** **Load** Mnemônico **LD**	Operação **9** Move to **Memory** Mnemônico **MM**	Operação **A** **Subroutine** Call Mnemônico **SC**	Operação **B** **Return** from Sub. Mnemônico **RS**
Operação **C** **Halt** Machine Mnemônico **HM**	Operação **D** **Get** Data Mnemônico **GD**	Operação **E** **Put** Data Mnemônico **PD**	Operação **F** **Operating System** Mnemônico **OS**

Tabela de caracteres ASCII (7 bits. Ex: "K" = 4b)

	0	1	2	3	4	5	6	7
0	NUL		SP	0	@	P	`	p
1			!	1	A	Q	a	q
2			"	2	B	R	b	r
3			#	3	C	S	c	s
4			$	4	D	T	d	t
5			%	5	E	U	e	u
6			&	6	F	V	f	v
7	BEL		'	7	G	W	g	w
8			(	8	H	X	h	x
9			)	9	I	Y	i	y
a	LF		*	:	J	Z	j	z
b		ESC	+	;	K	[	k	{
c			,	<	L	\	l	|
d	CR		-	=	M	]	m	}
e			.	>	N	^	n	~
f			/	?	O	_	o	DEL

Bibliografia (Programação de Sistemas)

Relíquias Preciosas

Barron, D. W. ***Assemblers and Loaders*** (3rd. ed.) MacDonald/Elsevier, 1978

Beck, L. L. ***System Software - An Introduction to Systems Programming*** Addison-Wesley, 1996

Calingaert, P. ***Assemblers, Compilers and Program Translation*** Computer Science Press, 1979

Donovan, J. J. ***Systems Programming*** McGraw-Hill, 1972

Duncan, F.G. ***Microprocessor Programming and Software Development*** Prentice Hall, 1979.

Freeman, P. ***Software System Principles*** SRA, 1975

Gear, C. W. ***Computer Organization and Programming (3rd. ed.)*** McGraw-Hill, 1980

Graham, R. M. ***Principles of Systems Programming*** John Wiley & Sons, 1975

Gust, P. ***Introduction to Machine and Assembly Language Programming*** Prentice Hall, 1985

Maginnis, J. B. ***Elements of Compiler Construction*** Appleton-Century-Crofts, Meredith Co., 1972

Presser, L. and White, J. R. ***Linkers and Loaders*** ACM Comp. Surveys, vol. 4, n. 3, pp. 149-168, 1972

Rosen, S. (ed.) ***Programming Systems and Languages*** McGraw-Hill, 1967

Tseng, V. (ed.) ***Microprocessor Development and Development Systems*** McGraw-Hill, 1982

Ullman, J. D. ***Fundamental Concepts of Programming Systems*** Addison-Wesley, 1976

Wegner, P. ***Progr. Languages, Inf. Structures and Machine Organization*** McGraw-Hill, 1968.

Welsh, J. and McKeag, M. ***Structured System Programming*** Prentice-Hall,1980

Referências Bibliográficas

Bryant R. E. and O'Hallaron, D. R. ***Computer Systems: A Programmer's Perspective***, 2010.

DONOVAN, J. ***Systems Programming***, 1972.

Leitura complementar:

UM SIMULADOR-INTERPRETADOR PARA A LINGUAGEM DE MÁQUINA DO PATINHO FEIO.

(João José Neto, Aspectos do Projeto de Software de um Minicomputador, Dissertação de Mestrado, EPUSP, S. Paulo, 1975, cap.3)

Transparências extraídas e alteradas de:

José Neto, J., Sichman, J. S., Silva, P.S.M., Rocha, R.L.A. ***Material didático da disciplina PCS 2024 – Laboratório de Fundamentos da Engenharia de Computação***, PCS/EPUSP, São Paulo, SP. 2005-2015.