PCS3616

Programação de Sistemas

(Sistemas de Programação)

Semana 8, Aula 13

Montador Relocável

Programação em linguagem de montagem

Escola Politécnica da Universidade de São Paulo

Roteiro

1. Necessidade de programas relocáveis
2. Implicações na linguagem simbólica
   - Novas pseudo-instruções
   - Novo formato de instrução
3. Montador relocável
   - Diagrama de classes
   - Exemplo de funcionamento

Necessidade de Programas Relocáveis (1)

- Programas absolutos são executáveis estritamente nas posições de memória em que foram criados
- Tornam difícil a manutenção e o trabalho em equipe
  - Exigem gerência cuidadosa das áreas de memória ocupadas e dos endereços de cada parte do programa
  - Toda vez que um programa é modificado, pode ser necessário recodificá-lo parcial ou totalmente
  - Se a área ocupada pelo novo código for maior que a antiga, é preciso alojar o programa em outra parte da memória

Necessidade de Programas Relocáveis (2)

- Programas relocáveis permitem sua execução em qualquer posição de memória
  - As referências à memória devem ser previamente ajustadas
  - Um gerenciador da ocupação da memória deve ser utilizado
- Tornam possível utilizar partes de código projetadas externamente
  - Uso de bibliotecas
  - Exigem que se possa montar parcialmente um programa, sem todos os endereços resolvidos!

Implicações na linguagem simbólica

- Para que se possa exprimir um programa relocável e com possibilidade de construção em módulos, separadamente desenvolvidos, é necessário que:
  - Haja a possibilidade de representar e identificar endereços absolutos e endereços relativos
  - Um programa possa ser montado sem que os seus endereços simbólicos estejam todos resolvidos
  - Seja possível identificar, em um módulo, símbolos que possam ser referenciados simbolicamente em outros módulos

Implicações no montador

- No montador, tornam-se necessários:
  - **endereços relativos** – uma pseudo-instrução especial deve indicar que se trata de origem relativa
  - **importar símbolos** – para que um símbolo **X** de outro programa possa ser referenciado no programa
  - **exportar símbolos** – para que um ponto **X** do programa possa ser referenciado em outros programas
  - anexar, ao final da montagem, todos os **símbolos não-resolvidos** ao programa-objeto, para que essa informação possa ser passada posteriormente ao programa ligador (*linker*).
  - Gerar **código-objeto no formato compatível** com o *loader* hexadecimal (função **P** do simulador MVN)

Alterações no Montador

- A inserção das seguintes modificações no montador absoluto são necessárias:
  - Inclusão e tratamento das novas pseudo instruções, para declarar:
    - **&** – Origem relocável
    - **>** – Endereço simbólico de entrada (entry point)
    - **<** – Endereço simbólico externo (external)
  - Geração de código-objeto no novo formato:
    - Origem absoluta e relocável
    - Operandos absoluto e relocável

Exemplos

- **&** – Origem relocável

```
ABC & /01AC ; ASSOCIA A ORIGEM CORRENTE AO SÍMBOLO ABC
            ; NOVA ORIGEM (RELOCÁVEL) É /01AC + BASE DE RELOCAÇÃO
    & /0000 ; INICIA A ORIGEM (RELOCÁVEL) EM 0
XYZ ...     ; XYZ FICA ASSOCIADO AO ENDEREÇO RELOCÁVEL 0
```

- **>** – Endereço simbólico de entrada (entry point)

```
ABC >       ; ASSOCIA A ORIGEM CORRENTE AO ENTRY-POINT ABC
```

- **<** – Endereço simbólico externo (external)

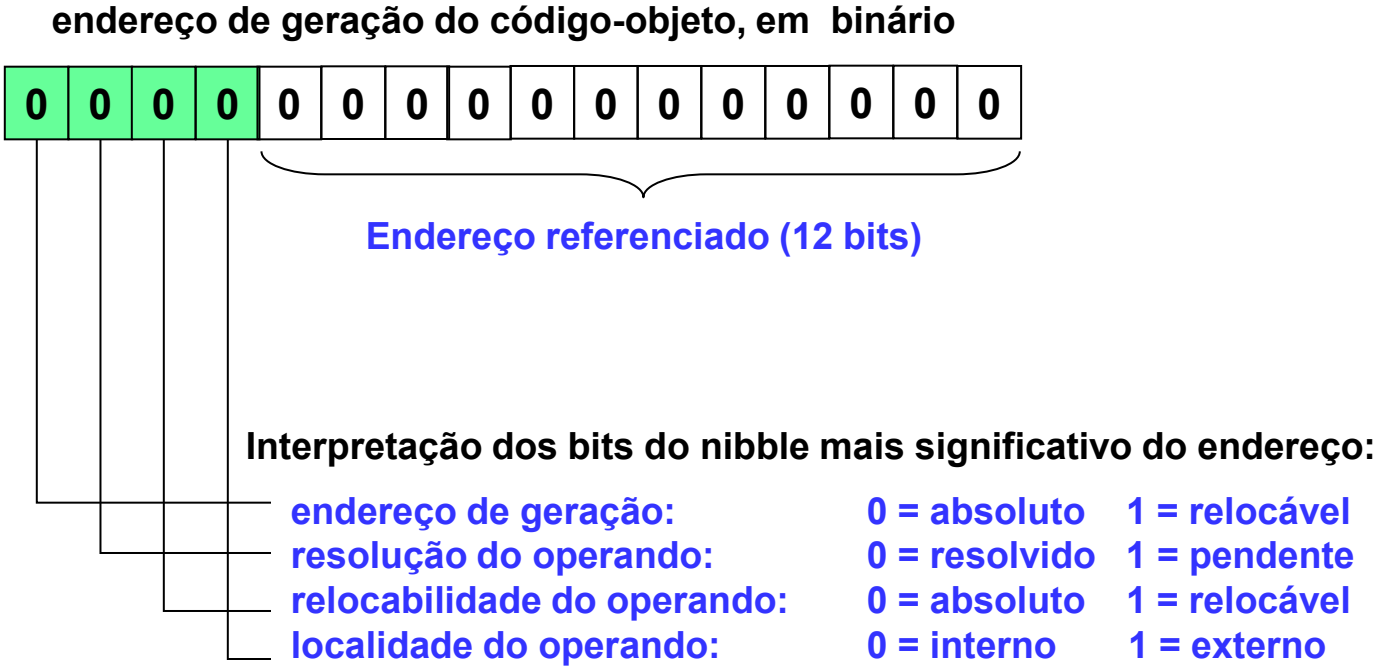
```
ABC <       ; DECLARA O SÍMBOLO EXTERNO ABC ESTÁ SENDO IMPORTADO
```

Tipos de endereços no programa-objeto

- Há dois aspectos a considerar:
  - o endereço onde será gerado o código
  - os endereços referenciados pelo código
- Endereço onde o código deve ser gerado
  - Absoluto ou relocável
- Endereço referenciado pelo código
  - Resolvido ou não-resolvido (endereços **externos** são não-resolvidos, endereços locais não-resolvidos são erros!)
  - Absoluto ou relocável (somente para endereços **locais**, para endereços externos designa-se como absoluto)
  - Interno ou externo (em relação à localidade do endereço referenciado (operando), todos os endereços **importados** no módulo são considerados **externos**, os demais (exportáveis e rótulos locais) são considerados **internos**.

Formatos no programa-objeto relocável

- Cada código gerado incorpora duas componentes de endereço:
  - O Endereço onde deve ser gerada a instrução (absoluto/relocável)
  - Operando referenciado (resolvido/não, absoluto/relocável, interno/externo)
- Pode-se codificar esses atributos nos quatro bits mais significativos do endereço onde o código deve ser gerado (até aqui, esses bits sempre foram nulos), já que o endereço ocupa apenas 12 bits

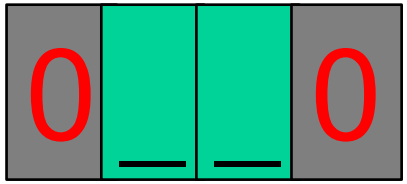


Combinações possíveis

Pseudo-instruções

- **Entry point >**
  - **`ABC >`**
- **External <**
  - **`ABC <`**

- Utilização dos bits.



- **Endereço de geração:** não utilizado (zero)
- **Resolução do operando:** utilizado
- **Relocabilidade do operando:** utilizado
- **Localidade do operando:** não utilizado (zero)

Combinações possíveis

Pseudo-instruções

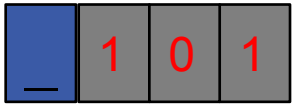
- 3 combinações possíveis
  - Declaração de Variável externa (importada)
  - Declaração de Variável interna (exportada) com endereço absoluto (Terceiro bit igual a zero)
  - Declaração de Variável interna (exportada) com endereço relativo (Terceiro bit igual a um)

Combinações possíveis

Instruções

- Instrução com variáveis externas (*)
  - `SOMADOR <  ;  Pseudo-instrução`
  - `MM SOMADOR  ;  Instrução com variável externa`

- **Endereço de geração:** utilizado
- **Resolução do operando:** utilizado (um)
- **Relocabilidade do operando:** não utilizado (zero)
- **Localidade do operando:** utilizado (um)



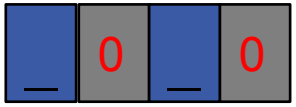
(*) Combinações diferentes destas citadas são casos de erros e devem ser corretamente tratadas.

Combinações possíveis

Instruções

- Instruções com variáveis internas (*)
  - `SAIDA > ; Pseudo-instrução`
  - `LD SAIDA ; instrução com variável interna`

- **Endereço de geração:** utilizado
- **Resolução do operando:** utilizado (zero)
- **Relocabilidade do operando:** utilizado
- **Localidade do operando:** utilizado (zero)



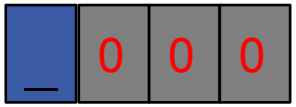
(*) Combinações diferentes destas citadas são casos de erros e devem ser corretamente tratadas.

Combinações possíveis

Instruções

- Declaração de variáveis (*)
  - `SAIDA > ; Pseudo-instrução`
  - `SAIDA K /100 ; Pseudo-instrução`

- **Endereço de geração:** utilizado
- **Resolução do operando:** não utilizado (zero)
- **Relocabilidade do operando:** não utilizado (zero)
- **Localidade do operando:** não utilizado (zero)



- (*) Combinações diferentes destas citadas são casos de erros e devem ser corretamente tratadas.

Alterações complementares

- Para atingir toda a sua funcionalidade, as seguintes adições posteriores serão necessárias:
  - Geração de código-objeto no novo formato, incluindo:
    - Operando simbólico
    - Endereços simbólicos de entrada e externos
    - Outras referências simbólicas não-resolvidas
  - Alteração do dumper hexadecimal: incluir referências simbólicas
  - Algoritmo de relocação a partir de uma base estabelecida
  - Alteração do loader hexadecimal: incluir relocação

Novas pseudo-instruções

Em adição às pseudo-instruções já utilizadas:

- @ (define uma ORIGEM ABSOLUTA para o código a ser gerado)
  - Exemplo: @ /50 ;indica /050 como origem do código seguinte
- # (define o FIM físico do programa)
  - Exemplo: # X ; indica que X é o endereço de execução do programa.
- K (define uma área preenchida por uma CONSTANTE de 2 bytes)
  - Exemplo: XYZ K /10 ; Gera /10 na posição correspondente a XYZ
- $ (define um BLOCO DE MEMÓRIA com número especificado de words)
  - Exemplo: XYZ $ =30 ; reserva 30 words, e o primeiro chama-se XYZ (Operando = número de words a serem reservadas para o bloco)

incluir-se-ão as seguintes novas pseudo-instruções:

- & (define uma ORIGEM RELOCÁVEL para o código a ser gerado)
  - Exemplo: & /50 ;indica que o próximo código se localizará no endereço /050, relativo à origem do código corrente.
- > (define um endereço simbólico local como entry-point do programa)
  - Exemplo: ABC > ; indica que o símbolo ABC está sendo exportado
- < (define um endereço simbólico que referencia um entry-point externo)
  - Exemplo: ABC < ; indica que ABC é um símbolo importado

Exemplo: programa em linguagem simbólica

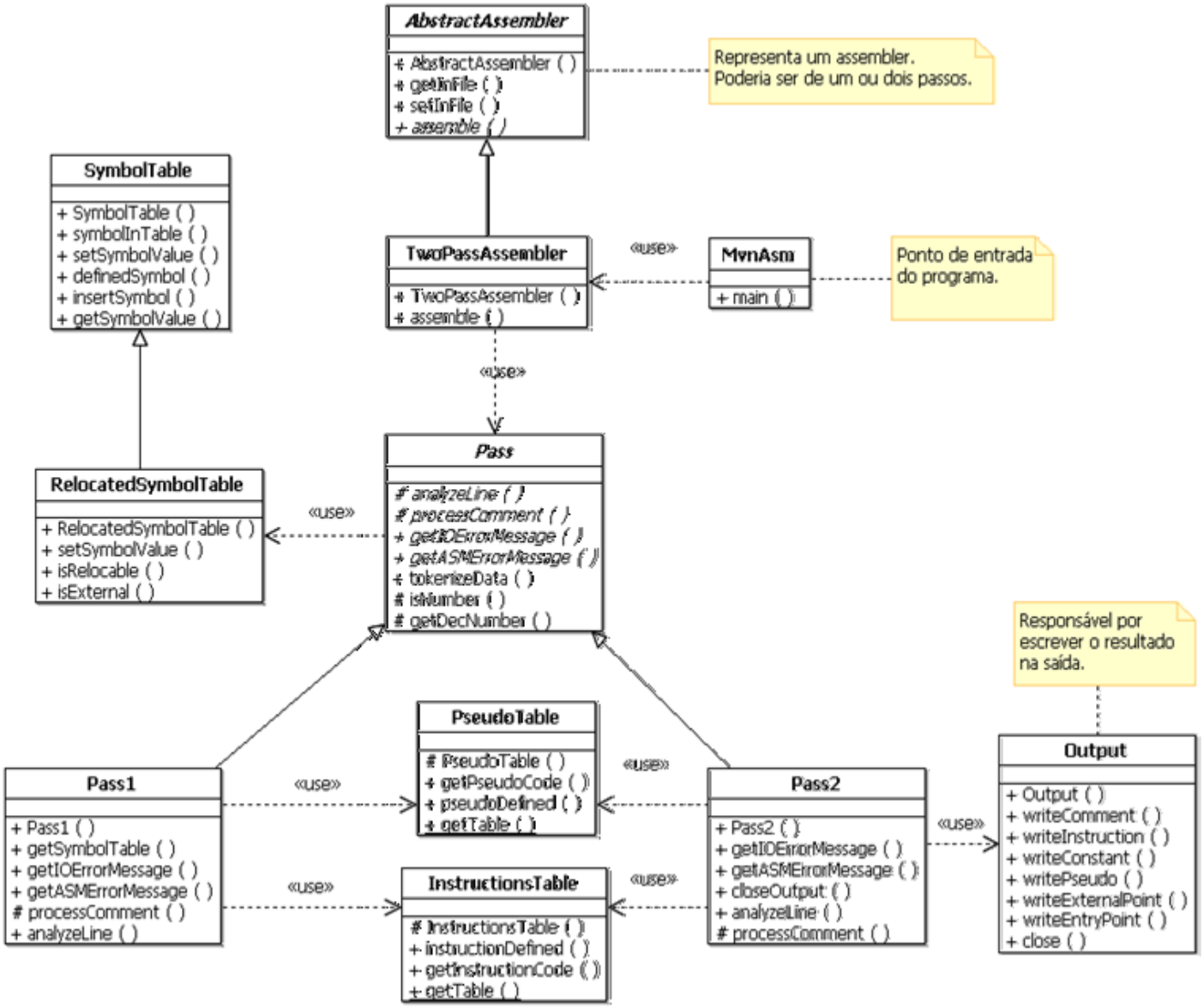
O programa abaixo, que foi dado como exemplo na aula 5:

```
A100          8F00   Obtém o endereço para onde se deseja mover o dado
A102          4F02   Compõe o endereço com o código de operação Move
A104          9106   Guarda instrução montada para executar em seguida
8106          9000   Executa a instrução recém-montada
A108          ....   Provavelmente, o código seguinte altera o conteúdo de 0F00
....
A15C          0100   Volta a repetir o procedimento, para outro endereço.
....
AF00          034C   Endereço (34C) para onde se deseja mover o dado
8F02          9000   Código de operação Move, com operando 000
```

codificado em linguagem simbólica, fica com o seguinte aspecto:

```
  & /0100     ; &=origem do código 0100=posição de memória (em hexadecimal)
P LD E        ; P=rótulo LD=load E=endereço simbólico da constante 034C
  +  M        ; +=add M=rótulo de onde está uma instrução Move 0000
  MM X        ; MM=move X=endereço da instrução seguinte
X MM /0       ; reservado para guardar a instrução recém-montada
  ...
  JP P        ; JP=jump (desvio) P=rótulo da primeira instrução deste programa
  ...
E K  /034C    ; E=rótulo K=constante 034C=operando numérico, em hexadecimal
M MM /0000    ; M=rótulo MM=move 0000=operando zero
  # P         ; #=final físico P=rótulo da primeira instrução a ser executada
```

Diagrama de Classes do Montador relocável



Classes do Montador Relocável

- O montador é definido a partir da classe abstrata (*AbstractAssembler*). O montador construído para esta disciplina é de dois passos (*TwoPassAssembler*). As demais classes (oriundas do montador absoluto) são:
  - **Tabela de instruções** (*InstructionsTable*): define as instruções válidas (símbolo e valor).
  - **Tabela de pseudo-instruções** (*PseudoTable*): define as pseudo-instruções válidas (símbolo e valor).
  - **Tabela de símbolos** (*SymbolTable*): permite armazenar e recuperar os rótulos (símbolo e endereço real).
  - **Passo** (*Pass*): define a estrutura dos passos, que são derivados desta classe (*Pass1* e *Pass2*).
  - **Saída** (*Output*): responsável por toda saída de dados para os arquivos.
  - **Ponto de entrada** (*MvnAsm*): contém o aplicativo que inicia o montador.
- A nova classe é:
  - **Tabela de símbolos relocáveis** (*RelocatedSymbolTable*): armazena e recupera os símbolos relocáveis.
- Nas classes existentes devem ser introduzidas mudanças.

Exemplo: Somador

- Programa somador.asm

```
; Somador
; *******
; Somador que recebe duas entradas, nas posições
; ENTRADA1 e ENTRADA2, e coloca o resultado da
; soma na posição SAIDA (externa).

SOMADOR >
ENTRADA1 >
ENTRADA2 >
SAIDA <

& /0000                    ; Origem relocável
; Entradas do programa.
ENTRADA1 K /0000
ENTRADA2 K /0000

; Programa
SOMADOR  JP /000           ; Ponto de entrada da subrotina
INICIO   LD ENTRADA1
         +  ENTRADA2
         MM SAIDA          ; Colocando na saída
         RS SOMADOR        ; Retornando
# INICIO
```

Exemplo: Somador

- Tabela de símbolos

	isRelocable	isExternal	Endereco (hexa)
SOMADOR	true	false	0004
ENTRADA1	true	false	0000
ENTRADA2	true	false	0002
SAIDA	?	true	?

- Código

	Endereço de geração	Resolução do operando	Relocabilidade do operando	Localidade do operando
SOMADOR >	0	0	1	0
ENTRADA1 >	0	0	1	0
ENTRADA2 >	0	0	1	0
SAIDA <	0	1	0	0
& /0000				
ENTRADA1 K /0000	1	0	0	0
ENTRADA2 K /0000	1	0	0	0
SOMADOR JP /000	1	0	0	0
INICIO LD ENTRADA1	1	0	1	0
+ ENTRADA2	1	0	1	0
MM SAIDA	1	1	0	1
RS SOMADOR	1	0	1	0

```
2004 0000 ; "SOMADOR>"
2000 0000 ; "ENTRADA1>"
2002 0000 ; "ENTRADA2>"
4000 0000 ; "SAIDA<"

8000 0000
8002 0000
8004 0000
a006 8000
a008 4002
d00a 9000
a00c b004
```

Exemplo: Somador

- Programa principal.asm

```
; Principal
; *********
; Programa principal que chama o somador.

SOMADOR <
ENTRADA1 <
ENTRADA2 <
SAIDA >

@ /0000
          JP INICIO
VALOR1    K =50              ; valor 1 a somar
VALOR2    K #101101          ; valor 2 a somar
SAIDA     K /0000

INICIO    LD VALOR1          ; passando as variáveis
          MM ENTRADA1
          LD VALOR2
          MM ENTRADA2
          SC SOMADOR         ; chamando o somador
          HM /00
# INICIO
```

Exemplo: Somador

- Tabela de símbolos

	isRelocable	**isExternal**	**Endereco (hexa)**
SOMADOR	?	true	?
ENTRADA1	?	true	?
ENTRADA2	?	true	?
SAIDA	false	false	0006
INICIO	false	false	0008
VALOR1	false	false	0002
VALOR2	false	false	0004

Exemplo: Somador

- Código

	Endereço de geração	Resolução do operando	Relocabilidade do operando	Localidade do operando
SOMADOR <	0	1	0	0
ENTRADA1 <	0	1	0	0
ENTRADA2 <	0	1	0	0
SAIDA >	0	0	0	0
@ /0000				
JP INICIO	0	0	0	0
VALOR1 K =50	0	0	0	0
VALOR2 K #101101	0	0	0	0
SAIDA K /0000	0	0	0	0
INICIO LD VALOR1	0	0	0	0
MM ENTRADA1	0	1	0	1
LD VALOR2	0	0	0	0
MM ENTRADA2	0	1	0	1
SC SOMADOR	0	1	0	1
HM /00	0	0	0	0

```
4000 0000 ; "SOMADOR<"
4001 0000 ; "ENTRADA1<"
4002 0000 ; "ENTRADA2<"
0006 0000 ; "SAIDA>"

0000 0008
0002 0032
0004 002d
0006 0000
0008 8002
500a 9001
000c 8004
500e 9002
5010 a000
0012 c000
```

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.