PCS 3216
Sistemas de Programação

João José Neto

Programação em linguagem simbólica e Montador

Parte 1

O Uso de Linguagem Simbólica

Código binário

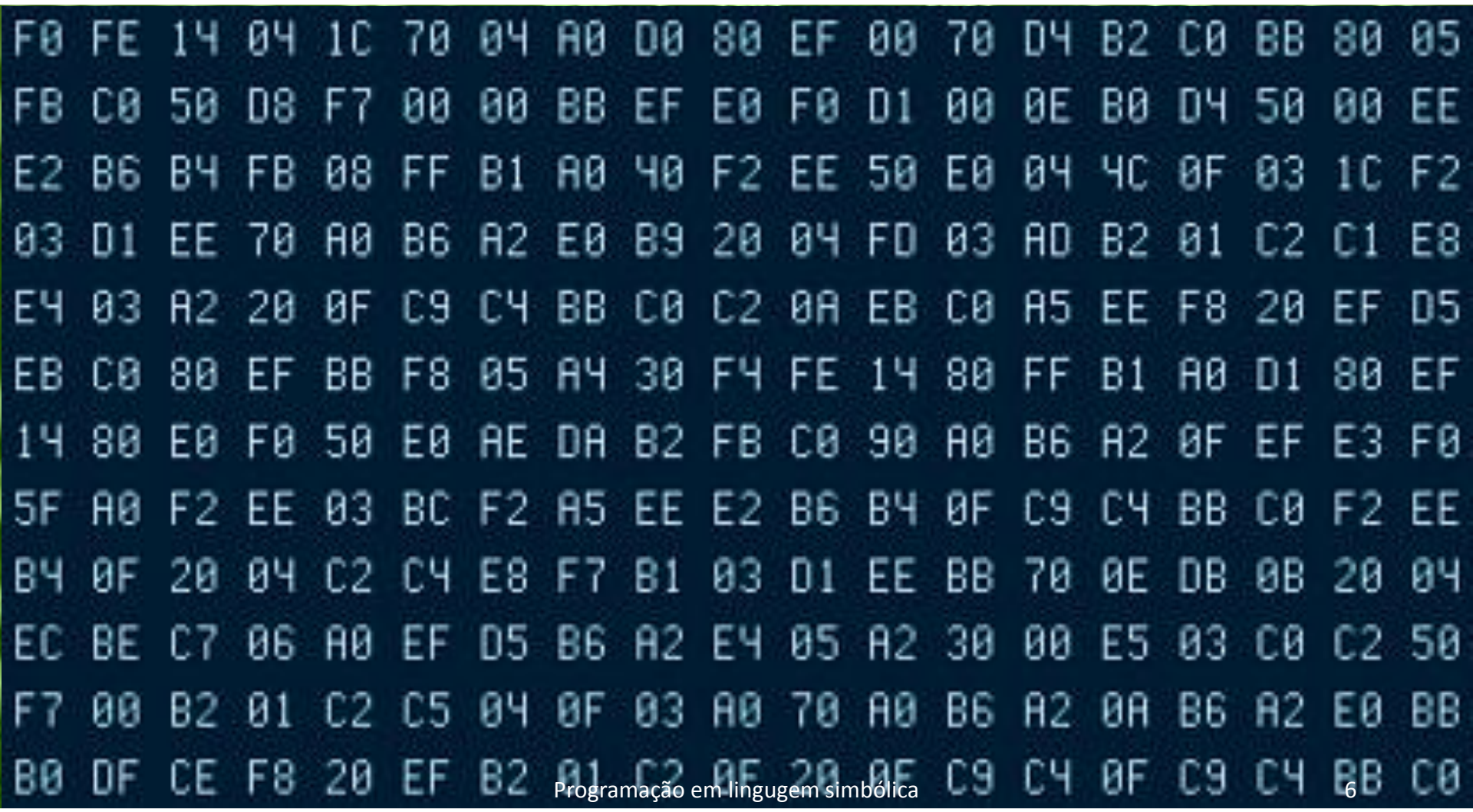
Codificação simbólica

O uso de **códigos não binários** pode melhorar a legibilidade e a documentação dos programas, Tendem a **facilitar a codificação** de programas em desenvolvimento e a melhorar a legibilidade de programas já codificados. Destacam-se:

- Uso de notações **numérica**, **não binárias**
- Uso de notação **não numérica**
- Utilização de notação **simbólica mnemônica**

- O uso de formatos numéricos não-binários, como é o caso das notações **octal**, **decimal** ou **hexadecimal**, tende a proporcionar aos programadores uma forma legível de acesso aos seus programas e dados, os quais agora podem ser denotados de uma forma mais compreensível e documentável.
- Em direção à meta de facilitar a codificação e melhorar a legibilidade dos programas, um grande passo foi dado em seguida, ao serem criadas formas de expressão de programas através de notações simbólicas, em substituição às notações numéricas.

Código hexadecimal



Dump de memória, nos formatos hexadecimal e ascii

Endereço Inicial (em hexa)

Cada linha mostra o conteúdo de 16 posições sucessivas de memória, em hexadecimal

Os mesmos 16 bytes, em ascii

```
0000: 4D 54 68 64 00 00 00 06 00 01 00 03 01 E0 4D 54   MThd..........MT
0010: 72 6B 00 00 00 19 00 FF 58 04 04 02 18 08 00 FF   rk......X.......
0020: 59 02 02 00 00 FF 51 03 0F 42 40 01 FF 2F 00 4D   Y.....Q..B@../.M
0030: 54 72 6B 00 00 00 71 00 C0 49 00 B0 79 00 00 B0   Trk...q..I..y...
0040: 40 00 00 B0 5B 30 00 B0 0A 40 00 B0 07 64 00 FF   @...[0...@...d..
0050: 03 01 20 00 90 3E 48 81 70 80 3E 00 00 90 40 4C   .. ..>H.p.>...@L
0060: 81 70 80 40 00 00 90 42 4E 81 70 80 42 00 00 90   .p.@...BN.p.B...
0070: 43 56 81 70 80 43 00 00 90 45 56 81 70 80 45 00   CV.p.C...EV.p.E.
0080: 00 90 47 54 81 70 80 47 00 00 90 49 52 81 70 80   ..GT.p.G...IR.p.
0090: 49 00 00 90 45 41 81 70 80 45 00 00 90 4A 50 87   I...EA.p.E...JP.
00A0: 40 80 4A 00 01 FF 2F 00 4D 54 72 6B 00 00 00 4D   @.J.../.MTrk...M
00B0: 00 C1 46 00 B1 79 00 00 B1 40 00 00 B1 5B 30 00   ..F..y...@...[0.
00C0: B1 0A 40 00 B1 07 64 00 FF 03 01 20 00 91 36 45   ..@...d.... ..6E
00D0: 83 60 81 36 00 00 91 32 4A 83 60 81 32 00 00 91   .`.6...2J.`.2...
00E0: 31 4A 83 60 81 31 00 00 91 2D 45 83 60 81 2D 00   1J.`.1...-E.`.-.
00F0: 00 91 32 4D 87 40 81 32 00 01 FF 2F 00            ..2M.@.2.../.
```

Mnemônicos

- Em oposição à linguagem de máquina, em que a codificação é feita de maneira totalmente numérica, procurou-se associar às diversas instruções de máquina os correspondentes códigos simbólicos mnemônicos, para permitir ao programador o desenvolvimento do programa sem a necessidade de manipular (ou, sequer, conhecer) numericamente os códigos binários da linguagem de máquina.
- A codificação mnemônica:
  - Usa códigos simbólicos, em geral formados por uma curta **sequência de letras** que **lembra a instrução** representada
  - **Dispensa** o programador de conhecer a **codificação numérica** do conjunto de instruções usado no seu programa
  - Até um certo ponto, pode-se dizer que essa notação fica **auto-documentada,** ainda que rudimentarmente .

Linguagens simbólica e numérica

Bytes do código de máquina

```
B8 22 11 00 FF
01 CA
31 F6
53
8B 5C 24 04
8D 34 48
39 C3
72 EB
C3
```

Comandos em linguagem simbólica

```
foo:
movl $0xFF001122, %eax
addl %ecx, %edx
xorl %esi, %esi
pushl %ebx
movl 4(%esp), %ebx
leal (%eax,%ecx,2), %esi
cmpl %eax, %ebx
jnae foo
retl
```

Linguagem numérica

Linguagem simbólica

Cadeia de instruções

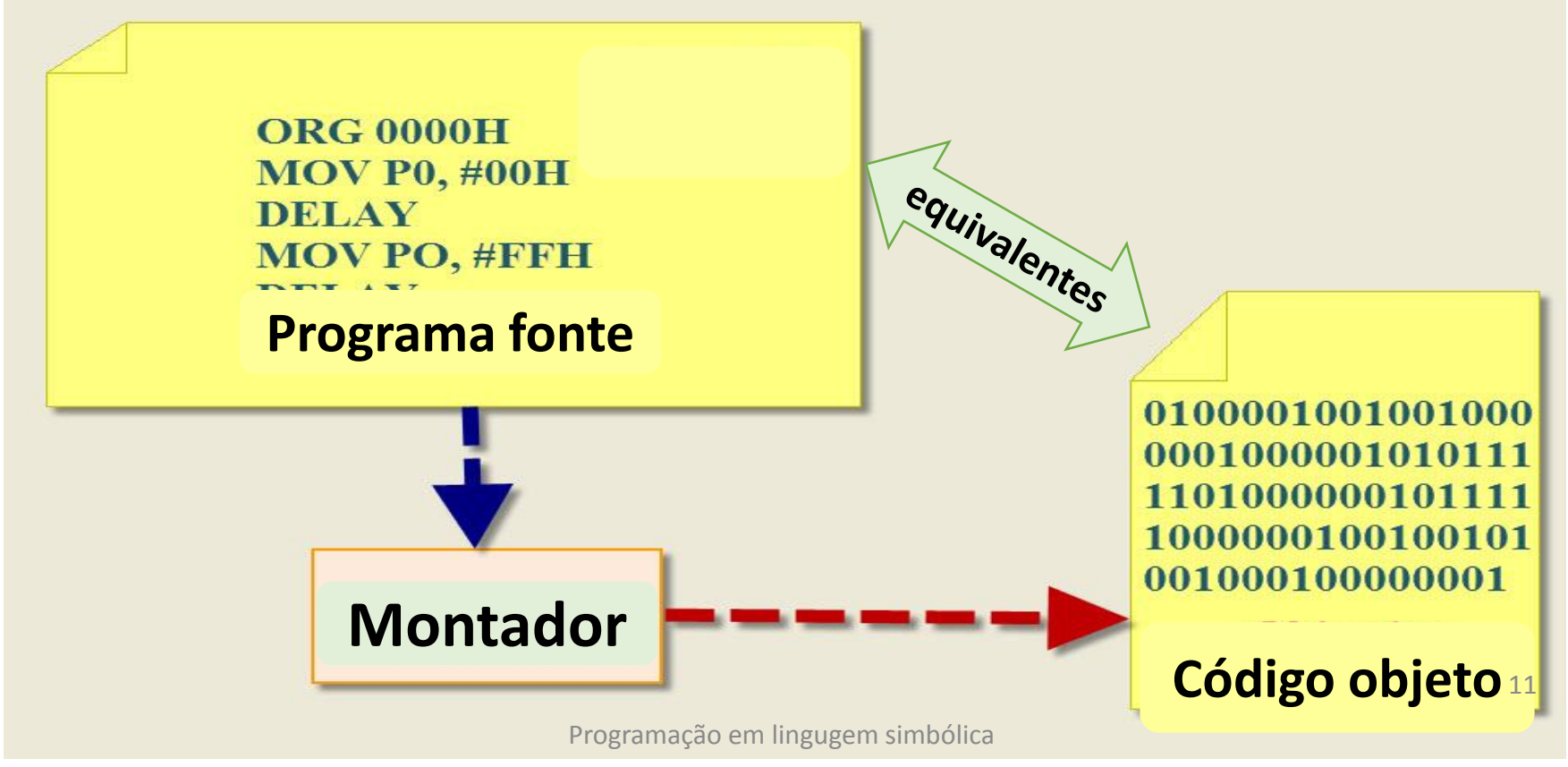
```
B8 22 11 00 FF 01 CA 31 F6 53 8B 5C 24
04 8D 34 48 39 C3 72 EB C3
```

A linguagem simbólica

- Como subproduto, o texto simbólico que representa o programa pode ser tornado mais inteligível, facilitando a compreensão do programa, quer por parte do próprio autor, em seu trabalho de desenvolvimento e de depuração, quer por parte de outros programadores e até de pessoas que não estejam familiarizadas com os detalhes do programa ou da máquina.
- O uso de linguagem simbólica:
  - **Facilita a leitura** do programa
  - **Simplifica o desenvolvimento** do programa
  - Simplifica a **depuração** do programa
  - **Facilita a compreensão** do programa por terceiros
  - **Simplifica a manutenção** posterior do programa

Tradução de programas simbólicos

- Para que um **programa** codificado em **linguagem simbólica** possa ser executado, é portanto necessário, antes, obter dele uma versão **equivalente**, em **linguagem de máquina**.



- Através de consultas a tabelas de correspondência entre os códigos mnemônicos e os códigos de máquina numéricos a eles associados, o programador pode, manualmente, efetuar com um certo conforto e agilidade a transcrição das instruções de seu programa simbólico para a forma de um código numérico correspondente.

- Essa transcrição vai além de uma simples substituição de mnemônicos por números, pois, sendo os próprios endereços das posições de memória do programa e dos seus dados representados como nomes simbólicos mnemônicos, precisam ser, também eles, convertidos para a forma numérica, como parte da operação de tradução do programa.

- O programa final, convertido para a forma numérica, é obtido, portanto, pela montagem de cada instrução binária de máquina, geralmente a partir de duas componentes:
  - a primeira, proveniente da operação básica de máquina associada ao mnemônico;
  - a segunda, referente a uma eventual referência a alguma posição de memória a ela associada.

Conversão de programas simbólicos em numéricos

- **Transcrição** (manual ou não) dos mnemônicos em **códigos de máquina numéricos**, em geral com a ajuda de tabelas
- **Tabelas de símbolos** memorizam os endereços numéricos associados aos endereços simbólicos
- São utilizadas para facilitar a **conversão** dos **endereços simbólicos em numéricos**
- A **montagem** da instrução efetua a composição do **código numérico** da instrução com o endereço numérico associado aos nomes de **posições de memória referenciadas** simbolicamente.

Montagem (*assembly*)

- O termo "**montagem**" designa a atividade de **tradução** de programas **fonte**, disponíveis na forma de um **código simbólico**, para a forma de um **código de máquina binário equivalente**.
- Para que programas em notação simbólica possam ser executados por um computador, é preciso portanto executar antes sua **montagem**
- Tal operação costuma ser **automatizada** com a ajuda de programas de sistema denominados **montadores** (***assemblers***).

Observações

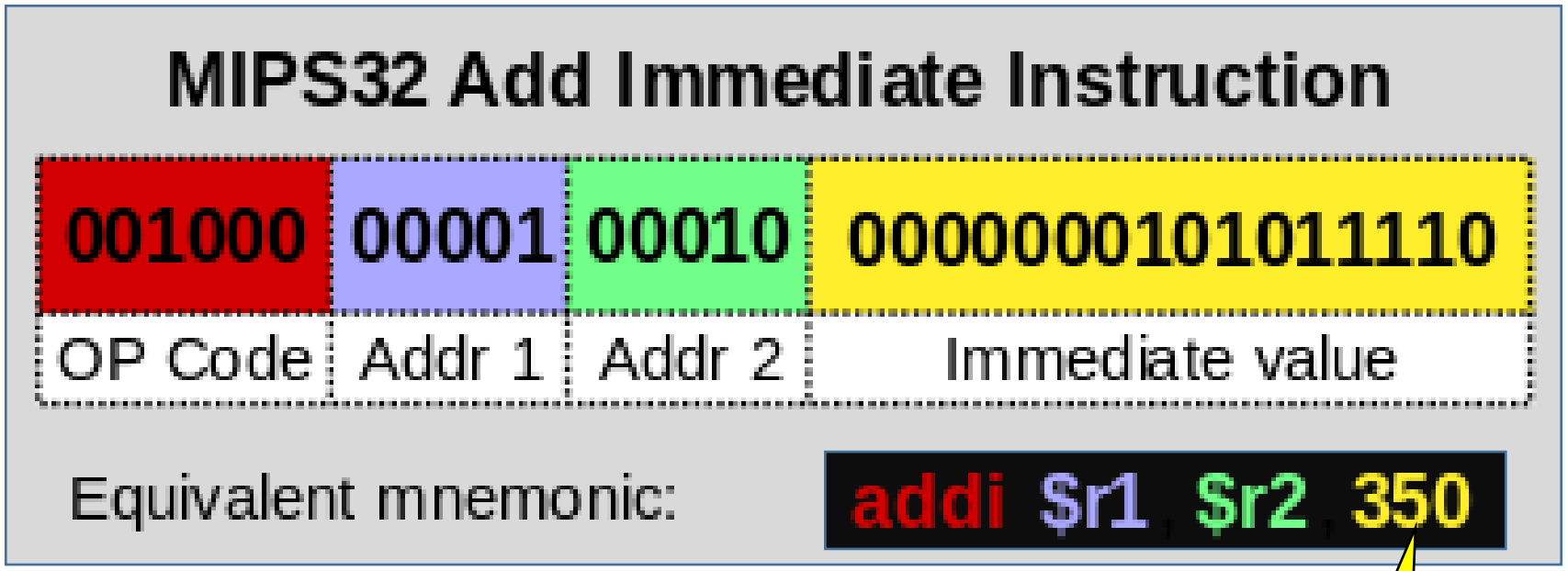
- Usa-se o termo **montagem** para designar essa atividade de tradução de programas-fonte, denotados em código simbólico, para a forma numérica de um código de máquina binário equivalente.

- É fácil constatar que, embora facilite o trabalho de escrita e melhore o nível de legibilidade e de compreensão do programa para o ser humano, o uso de uma linguagem simbólica força o programador a efetuar, em troca, a atividade adicional de montagem do programa, para que este possa, enfim, ser executado.

Montadores

- A operação de montagem é um trabalho tedioso e mecânico, e, por sua característica metódica e repetitiva, pode ser facilmente automatizado, por meio de programas de sistema denominados **montadores** (“***assemblers***”), que portanto se encarregam da tradução de programas dados em linguagem simbólica de montagem para um de código numérico em formato executável.

Exemplo: estrutura de uma instrução



$350_{10} = 15E_{16} = 0001\ 0101\ 1110_2$

Instrução simbólica, em linguagem de montagem

MOV AL,54H

MOV

Tabela de Mnemônicos

ADD	55H
BRA	43H
CMP	21H
MOV	88H
POP	32H
RET	01H
SHR	42H

Código binário de máquina

01001010101
01001010001
01010101010
01000111010
100100 ?????

88H

Referência simbólica, em linguagem de montagem

MOV AX, PayRate

PayRate

Tabela de Símbolos

Total	0100
Average	0102
Pay	0401
PayRate	2031
Date	2011

Posição de memória referenciada, em formato numérico

2031

Montagem manual é tarefa cansativa

- Tarefa simples, mas **tediosa** e **mecânica**
- Pode ser facilmente **automatizada**
- Os programas que fazem este trabalho são os **Montadores** (***Assemblers***)
- Montadores **traduzem** programas-fonte em **notação simbólica**, convertendo-os nos correspondentes códigos **numéricos** executáveis equivalentes.

Linguagens de montagem

- As linguagens aceitas pelos montadores são denominadas **linguagens de montagem** (“***assembly languages***”), e, em sua forma mais evoluída, incorporam diversos recursos adicionais de programação, que complementam a linguagem básica de máquina, facilitando a programação e permitindo melhorar a qualidade da documentação do programa:
  - **Montadores** não se limitam a **traduzir** programas simbólicos em programas numéricos.
  - Complementam a linguagem de máquina com **outros recursos** de programação que facilitam a produção e ajudam a melhorar a documentação e a legibilidade dos programas.

Nos sistemas antigos

- Programação em linguagens de **baixo nível de abstração:**
  - Linguagem de **máquina**:
    - Autor = Programador = Operador
  - Linguagem **simbólica**
    - Programadores/Codificadores
    - Analistas
- Uma preocupação permanente do responsável pela elaboração de um programa era manter em foco o desempenho do mesmo em relação à eficiência dos programas construídos.
- O **responsável** pelo programa era, portanto, encarregado de garantir **vários quesitos**, entre os quais:
  - eficiência da atividade de programação
  - quantidade de memória ocupada
  - tempo de resposta do programa

Com a evolução da tecnologia

- Com o barateamento do hardware e com a crescente imposição do software como determinante do custo do sistema, mudaram também alguns objetivos de projeto: em lugar de buscar reduções da área de memória ocupada, passou-se a procurar meios de construir programas bem estruturados, de fácil entendimento, com bons recursos para a execução de testes e para a implementação do programa.
- Um objetivo não muito perseguido nos primeiros tempos da computação, e que hoje se mostra essencial, é o de garantir ao projeto a maior facilidade possível de manutenção, importante meta da Engenharia de Software.

Nos sistemas recentes

- **O hardware** sofreu considerável barateamento
- **O software** tornou-se o principal determinante do **custo do sistema**
- Mudaram os **objetivos** do programador:
  - Boa **estruturação** dos programas
  - Fácil **entendimento** do programa
  - Bons recursos para **testes** e implementação
  - Maior facilidade de **manutenção**

Programação independente de máquina

- Apesar de facilitar muito o trabalho dos programadores e analistas, o uso de linguagens simbólicas ainda mantém uma considerável proximidade em relação à máquina, deles exigindo, portanto, um treinamento específico para cada particular hardware a ser utilizado.
- Isto pode ser muito prejudicial em relação aos aspectos de intercâmbio dos programas desenvolvidos, e, portanto, da sua portabilidade, o que sugere fortemente aos programadores a desvinculação, até onde for possível, entre o programa e a máquina em que serão executados.

Deficiências da linguagem simbólica

- Exige **proximidade excessiva** do usuário com a máquina
- Cada computador é **diferente** dos demais
- Demanda um **treinamento** específico e detalhado
- A **efetividade** desse conhecimento é **efêmera**, dada a velocidade dos avanços, tanto no hardware como no software

- **Intercâmbio difícil** de programas
- **Portabilidade limitada** dos programas
- **Vinculação** forte entre **programa e máquina**

- Isto sugere que se busquem formas de **desacoplamento** entre o programa e a máquina em que é executado.
- Uma solução foi adotar **linguagens** de programação que operassem em um **nível superior de abstração**

Alternativas

- A busca de alternativas para exprimir algoritmos e ideias em alguma forma independente da particular máquina em que devem ser executados proporcionou o desenvolvimento das denominadas linguagens de alto nível, em substituição às linguagens de montagem e de máquina.
- Entre essas alternativas, destacam-se as notações inspiradas na linguagem humana, como por exemplo, os diagramas da lógica dos programas e os pseudocódigos.

Linguagens de alto nível

- Com a ajuda de **diagramas** e de **pseudo-código,** é possível exprimir os programas de maneira bastante **independente de máquina.**
- A criação de **notações** apropriadas (denominadas **linguagens de alto nível**), visando à codificação dos programas com a ajuda de **linguagens artificiais,** move a atividade de programação para mais perto da linguagem humana, e para mais longe das linguagens de máquina numéricas.
- Nessas linguagens, **fórmulas matemáticas** são codificadas em um formato muito próximo ao da notação matemática usualmente empregada nos estudos teóricos.

Fórmulas

- As **primeiras linguagens** de alto nível que surgiram colocavam seu foco na **codificação de fórmulas** matemáticas
- A uma das primeiras dessas linguagens de alto nível foi dado o nome de

  FORTRAN = **FOR**mula **TRAN**slation
- Entre outras das **linguagens** mais **antigas** citam-se:

  o Basic, o Cobol e o Lisp
- Incrivelmente, todas essas linguagens pioneiras continuam a ser **intensamente utilizadas** para finalidades práticas **até os dias de hoje.**

Dificuldade de tradução

- Apesar da simplificação obtida pelo emprego introduzido de linguagens de alto nível, continua sendo muito elevada a complexidade relativa dessas linguagens de alto nível em relação à das linguagens de máquina.
- A tarefa da tradução dessas linguagens apresenta um grau de dificuldade suficientemente grande para inviabilizar que no dia-a-dia essa atividade possa ser efetuada manualmente.

- Por esta razão, torna-se necessária a elaboração dos programas de sistema chamados processadores de linguagens de alto nível:
  - Programas de sistema especializados na tradução automática de programas denotados em uma linguagem de alto nível, para a linguagem básica da máquina específica em que se deseje utilizar esses programas. Ou então,
  - Programas que analisam o programa em linguagem de alto nível e promovem a execução das operações por ele especificadas.

Tradução dos programas escritos em linguagem de alto nível

- A obtenção de **códigos executáveis** a partir das **linguagens de alto nível** não é trivial.
  - Apesar de toda a simplificação notacional
  - Apesar da grande limitação de sua flexibilidade
- Isso exige a utilização de processos **automáticos** para a sua **tradução** ou **interpretação.**
- **Compiladores e Interpretadores** são os programas que executam essas tarefas.

Compiladores

- **Traduzem** para código executável programas expressos em linguagem de alto nível
  (processo conceitualmente **similar ao da montagem**, embora muito mais complexo)
- Recebem como **entrada** o programa-**fonte**, em **linguagem de alto nível**
- Geram como **saída** um programa equivalente em **linguagem de máquina** (em geral, **relocável**)

Interpretadores

- Uma **alternativa** para o processamento de programas em linguagem de alto nível é a utilização de **interpretadores.**
- Esses programas de sistema **percorrem o texto do programa** a executar, analisando-o e decidindo **passo a passo** as ações esperadas, e por fim, executando-as.
- Não geram código de máquina, mas **simulam a execução** direta **dos comandos** do programa.
- O processamento de linguagens de alto nível é tema para um estudo detalhado posterior.

Parte 2

Como funciona um Montador

Apresentação

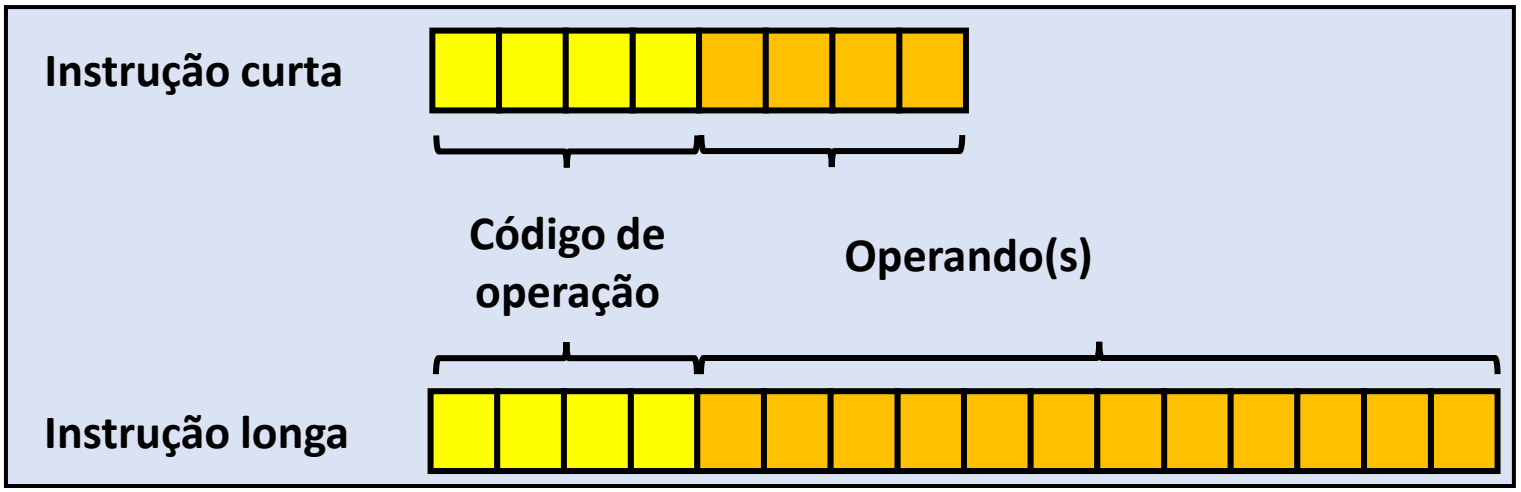
- Este jogo de slides procura registrar, embora não literalmente, um exemplo do uso do mecanismo utilizado pelos montadores para efetuar a conversão de um pequeno programa em notação simbólica, escrito em linguagem de montagem, para a forma de um programa executável equivalente, expresso em linguagem de máquina.
- Use-o como guia para um estudo panorâmico do funcionamento de montadores e para tirar as principais dúvidas conceituais sobre o processo de montagem de programas simbólicos.
- Detalhes particulares, acerca de especificidades e tópicos mais avançados sobre o assunto, são estudados mais adiante nesta disciplina.
- *Convém alertar que este exemplo ilustrativo, embora similar, não atende a todas as especificações do projeto desta disciplina, portanto para a elaboração do projeto deverão ser feitos os ajustes necessários para adequá-los às especificações.*

Máquina hospedeira

- O hardware do processador em que o programa estudado neste exemplo deverá ser executado tem, entre outras, as seguintes características
  - Acumulador, registrador aritmético de 1 byte (8 bits)
  - Memória de 4kB, com palavras de 1 byte (8 bits)
  - Instruções: os 4 primeiros bits designam seu tipo
  - Instruções de 1 byte (curtas): STOP, READ, WRITE, RTN
    - Os demais 4 bits designam operando (valor = 0 a 15)
  - Instruções de 2 bytes (longas): JUMP, JUMP0, JUMPN, ADD, SUB, MUL, DIV, CALL
    - Os demais 12 bits designam operando (endereço de memória)

Formatos das instruções

- Na máquina simples que estamos usando para ilustrar este material, há dois formatos apenas de instruções: as curtas, de 1 byte, e as longas de 2 bytes.
- Em ambas, os 4 primeiros bits determinam qual é a particular instrução:



Informações sobre as Instruções

Instrução	mnemônico	código	tamanho
Desvio incondicional	JUMP	0000	2
Desvio se zero	JUMP0	0001	2
Desvio se negativo	JUMPN	0010	2
Soma	ADD	0011	2
Subtração	SUB	0100	2
Multiplicação	MUL	0101	2
Divisão	DIV	0110	2
Carregar acumulador	LOAD	0111	2
Guardar na memória	STORE	1000	2
Chamar subrotina	CALL	1001	2
		1010	
		1011	
Retorno de subrotina	RTN	1100	1
Parar o programa	STOP	1101	1
Ler um byte da fita p/o acumulador	READ	1110	1
Perfurar fita c/ byte do acumulador	WRITE	1111	1

Informações sobre as Pseudo-Instruções

- Definem meta-dados para uso do montador, que não se referem a instruções de máquina.

Pseudo	mnemônico	Operando	Efeito
Origem	ORG	Endereço	Define endereço de origem do código a seguir
Fim	END	Nenhum	Define o final físico do programa fonte
Dados	DATA	Valor numérico	Define o conteúdo endereço que está sendo preenchido
Área	AREA	Comprimento	Reserva área de memória, de comprimento especificado

Exemplo simples de montagem

- A seguir, mostra-se um programa muito curto, que servirá para auxiliar na explicação do processo de tradução de um programa fonte simbólico para formato executável.
- Para não complicar o entendimento, os números utilizados nos slides a seguir estão todos denotados em **hexadecimal**, salvo ressalva em contrário.
- Esse programa, usado como exemplo, serve apenas para ilustrar a mecânica desse processo, e não se tem a intenção de representar nenhum algoritmo real.

Programa fonte a ser montado

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
		2	INICIO	LOAD	X
		3		ADD	Y
		4		STORE	Z
		5		STOP	
		6		JUMP	INICIO
		7	X	DATA	10
		8	Y	DATA	25
		9	Z	AREA	3
		A		END	

- Nos slides seguintes, o texto-fonte acima será processado passo a passo pelo montador, e as tabelas do próximo slide serão preenchidas à medida que as linhas deste programa forem sendo tratadas pelos algoritmos do montador.

Situação inicial

TABELA DE SÍMBOLOS

Símbolo	endereço

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

(vazia)

CÓDIGO GERADO

Endereço	Conteúdo

Linha 1: ORG 29E

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E		2	INICIO	LOAD	X
		3		ADD	Y
		4		STORE	Z
		5		STOP	
		6		JUMP	INICIO
		7	X	DATA	10
		8	Y	DATA	25
		9	Z	AREA	3
		A		END	

- Não tem rótulo
- ORG é uma pseudo instrução que define endereço de origem para o código a seguir
- 29E é esse endereço

- Associar à linha seguinte (linha 2) o endereço 29E
- Não gera código

Dados após tratar a linha 1

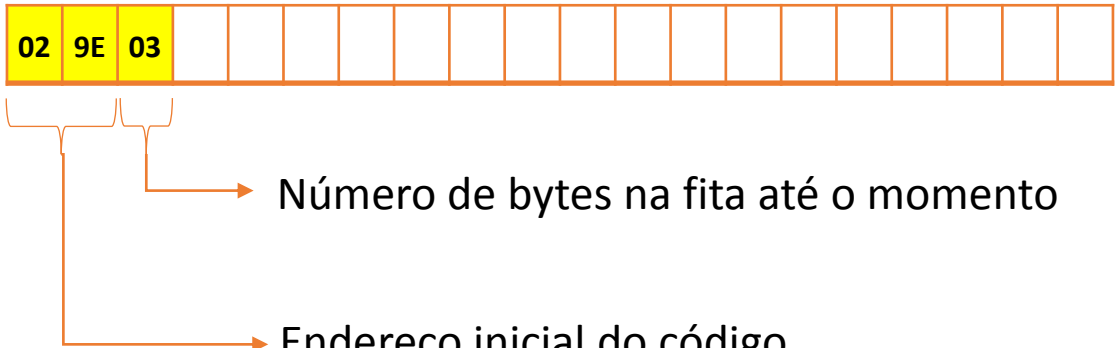
TABELA DE SÍMBOLOS

Símbolo	endereço

CÓDIGO GERADO

Endereço	Conteúdo

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)



Linha 2: INICIO LOAD X

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	7???	2	INICIO	LOAD	X
2A0		3		ADD	Y
		4		STORE	Z
		5		STOP	
		6		JUMP	INICIO
		7	X	DATA	10
		8	Y	DATA	25
		9	Z	AREA	3
		A		END	

- O rótulo INICIO deve ser associado ao endereço 29E
- LOAD é a instrução com código 7 (ou 0111_2), e ocupa 2 bytes de memória.
- X é um endereço simbólico, por enquanto indefinido.

- Adicionar INICIO à tabela de símbolos, e associá-lo ao endereço 29E
- Preencher o código de operação (7) da instrução LOAD
- Adicionar X à tabela de símbolos, marcando X como ainda indefinido (???).
- Na linha 3, somar 2 bytes ao endereço 29E

Dados após tratar a linha 2

TABELA DE SÍMBOLOS

Símbolo	endereço
INICIO	29E
X	???

CÓDIGO GERADO

Endereço	Conteúdo
29E	7?
29F	??

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

Número de bytes na fita até o momento

Código parcial da linha 2

Linha 3: ADD Y

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	7???	2	INICIO	LOAD	X
2A0	3???	3		ADD	Y
2A2		4		STORE	Z
		5		STOP	
		6		JUMP	INICIO
		7	X	DATA	10
		8	Y	DATA	25
		9	Z	AREA	3
		A		END	

- Não há rótulo aqui.
- ADD tem código 3 (0011_2) e ocupa 2 bytes
- Y é um endereço simbólico ainda ausente na tabela de símbolos

- Preencher o código de operação (3) da instrução ADD,
- Manter indefinido (???) o seu operando
- Na linha 4, somar 2 ao endereço 2A0
- Incluir Y (indefinido) na tabela de símbolos

Dados após tratar a linha 3

TABELA DE SÍMBOLOS

Símbolo	endereço
INICIO	29E
X	???
y	???

CÓDIGO GERADO

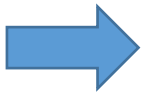
Endereço	Conteúdo
29E	7?
29F	??
2A0	3?
2A1	??

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

Linha 4: STORE Z

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	7???	2	INICIO	LOAD	X
2A0	3???	3		ADD	Y
2A2	8???	4		STORE	Z
2A4		5		STOP	
		6		JUMP	INICIO
		7	X	DATA	10
		8	Y	DATA	25
		9	Z	AREA	3
		A		END	

- Não há rótulo aqui.
- STORE tem código 8 (1000_2) e ocupa 2 bytes
- Z é um endereço simbólico ainda ausente na tabela de símbolos



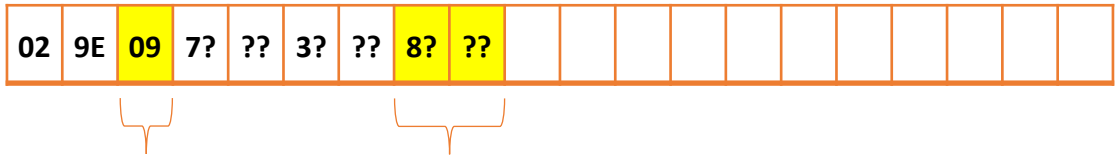
- Preencher o código (8) de operação da instrução STORE,
- Manter indefinido (???) o seu campo de operando
- Na linha 5, somar 2 bytes ao endereço 2A2
- Incluir Z (indefinido) na tabela de símbolos

Dados após tratar a linha 4

TABELA DE SÍMBOLOS

Símbolo	endereço
INICIO	29E
X	???
y	???
Z	???

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)



CÓDIGO GERADO

Endereço	Conteúdo
29E	7?
29F	??
2A0	3?
2A1	??
2A2	8?
2A3	??

Linha 5: STOP

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	7???	2	INICIO	LOAD	X
2A0	3???	3		ADD	Y
2A2	8???	4		STORE	Z
2A4	D0	5		STOP	
2A5		6		JUMP	INICIO
		7	X	DATA	10
		8	Y	DATA	25
		9	Z	AREA	3
		A		END	

- Não há rótulo
- STOP tem código D (1101_2), não tem operandos e ocupa 1 byte de memória

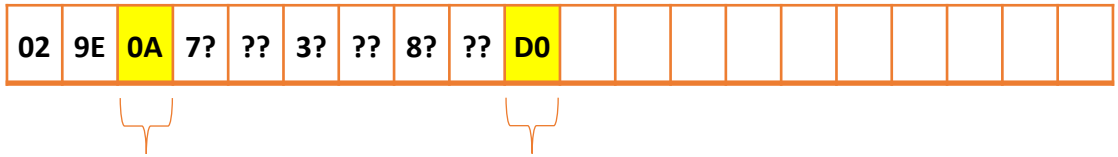
- Não tendo operandos, o conteúdo do campo de operando do código objeto é irrelevante, e será preenchido com zeros. O código resultante é D0.
- Na linha 6, somar 1 byte ao endereço 2A4
- A tabela de símbolos permanece inalterada.

Dados após tratar a linha 5

TABELA DE SÍMBOLOS

Símbolo	endereço
INICIO	29E
X	???
y	???
Z	???

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)



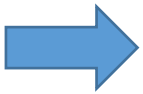
CÓDIGO GERADO

Endereço	Conteúdo
29E	7?
29F	??
2A0	3?
2A1	??
2A2	8?
2A3	??
2A4	D0

Linha 6: JUMP INICIO

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	7???	2	INICIO	LOAD	X
2A0	3???	3		ADD	Y
2A2	8???	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7		7	X	DATA	10
		8	Y	DATA	25
		9	Z	AREA	3
		A		END	

- Não há rótulo
- JUMP tem código 0 (0000_2), e ocupa 2 bytes de memória
- INICIO é seu operando, está na tabela de símbolos associado ao endereço 29E



- Preencher o campo de código de operação do JUMP com 0
- Preencher o campo de operando do JUMP com 29E
- Na linha 7, somar 2 bytes ao endereço 2A5
- A tabela de símbolos permanece inalterada.

Dados após tratar a linha 6

TABELA DE SÍMBOLOS

Símbolo	endereço
INICIO	29E
X	???
y	???
Z	???

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

Número de bytes na fita até o momento

Código da linha 6

CÓDIGO GERADO

Endereço	Conteúdo
29E	7?
29F	??
2A0	3?
2A1	??
2A2	8?
2A3	??
2A4	D0
2A5	02
2A6	9E

Linha 7: X DATA 10

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	7???	2	INICIO	LOAD	X
2A0	3???	3		ADD	Y
2A2	8???	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8		8	Y	DATA	25
		9	Z	AREA	3
		A		END	

- O rótulo X, indefinido na tabela de símbolos, agora fica associado ao endereço 2A7
- DATA não tem código de operação associado (pseudo instrução)
- O operando 10 dá o valor que preencherá o endereço 2A7
- Esse valor ocupa 1 byte de memória

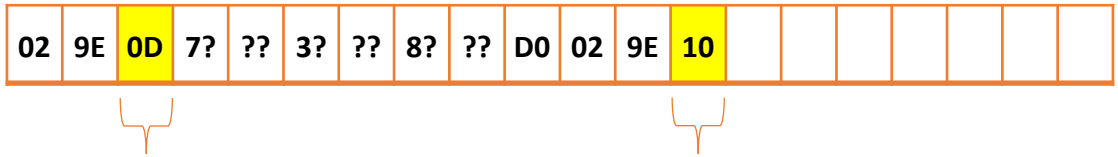
- Definir na tabela de símbolos o rótulo X associando-o ao endereço 2A7
- No endereço 2A7 do código, gravar o valor 10
- Na linha 8, somar 1 byte ao endereço 2A7

Dados após tratar a linha 7

TABELA DE SÍMBOLOS

Símbolo	endereço
INICIO	29E
X	2A7
y	???
Z	???

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)



CÓDIGO GERADO

Endereço	Conteúdo
29E	7?
29F	??
2A0	3?
2A1	??
2A2	8?
2A3	??
2A4	D0
2A5	02
2A6	9E
2A7	10

Linha 8: Y DATA 25

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	7???	2	INICIO	LOAD	X
2A0	3???	3		ADD	Y
2A2	8???	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
		A		END	

- O rótulo Y, indefinido na tabela de símbolos, agora fica associado ao endereço 2A8
- DATA não tem código de operação associado (pseudo instrução)
- O operando 25 dá o valor que preencherá o endereço 2A8
- Esse valor ocupa 1 byte de memória

- Definir na tabela de símbolos o rótulo Y associando-o ao endereço 2A8
- No endereço 2A8 do código, gravar o valor 25
- Na linha 9, somar 1 byte ao endereço 2A8

Dados após tratar a linha 8

TABELA DE SÍMBOLOS

Símbolo	endereço
INICIO	29E
X	2A7
Y	2A8
Z	???

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

Número de bytes na fita até o momento

Código da linha 8

CÓDIGO GERADO

Endereço	Conteúdo
29E	7?
29F	??
2A0	3?
2A1	??
2A2	8?
2A3	??
2A4	D0
2A5	02
2A6	9E
2A7	10
2A8	25

Linha 9: Z AREA 3

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	7???	2	INICIO	LOAD	X
2A0	3???	3		ADD	Y
2A2	8???	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- O rótulo Z, indefinido na tabela de símbolos, agora fica associado ao endereço 2A9
- AREA não tem código de operação associado (é pseudo instrução)
- O operando 3 dá o número de bytes reservados a partir do endereço 2A9
- Essa área ocupa 3 bytes de memória

- Definir na tabela de símbolos o rótulo Z associando-o ao endereço 2A9
- Nenhum código é gerado por esta pseudo instrução
- Na linha A, somar 3 bytes ao endereço 2A9

Dados após tratar a linha 9

TABELA DE SÍMBOLOS

Símbolo	endereço
INICIO	29E
X	2A7
y	2A8
Z	2A9

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

Número de bytes na fita até o momento

Código da linha 9

CÓDIGO GERADO

Endereço	Conteúdo
29E	7?
29F	??
2A0	3?
2A1	??
2A2	8?
2A3	??
2A4	D0
2A5	02
2A6	9E
2A7	10
2A8	25
2A9	??
2AA	??
2AB	??

Linha A: END

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	7???	2	INICIO	LOAD	X
2A0	3???	3		ADD	Y
2A2	8???	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- Não há rótulo
- END é uma pseudo casos instrução sem código associado, e sem reserva de memória
- Não há operandos neste caso

- O endereço 2AC aponta para a posição de memória que sucede a última posição ocupada pelo programa que acabamos de estudar.
- A pseudo instrução END finaliza o programa fonte.

Dados após tratar a linha A

TABELA DE SÍMBOLOS

Símbolo	endereço
INICIO	29E
X	2A7
Y	2A8
Z	2A9

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

02	9E	11	7?	??	3?	??	8?	??	D0	02	9E	10	25	??	??	??			

CÓDIGO GERADO

Endereço	Conteúdo
29E	7?
29F	??
2A0	3?
2A1	??
2A2	8?
2A3	??
2A4	D0
2A5	02
2A6	9E
2A7	10
2A8	25
2A9	??
2AA	??
2AB	??

Tabela de Símbolos após linha A

Símbolo	endereço
INICIO	29E
X	2A7
y	2A8
Z	2A9

Ao final dessa fase, em que o montador já percorreu todo o programa fonte:

- Não há símbolos indefinidos, ou seja, símbolos que não tenham um endereço associado.
- Por isso, pode-se considerar que o programa não contém erros de referências simbólicas.

Código Gerado, após linha A

CÓDIGO GERADO

Endereço	Conteúdo
29E	7?
29F	??
2A0	3?
2A1	??
2A2	8?
2A3	??
2A4	D0
2A5	02
2A6	9E
2A7	10
2A8	25
2A9	??
2AA	??
2AB	??

Códigos com preenchimento incompleto

Observando-se a tabela de código gerado pelo montador ao final desta fase, pode-se notar que há diversos bytes ainda não preenchidos.

Para completar esse preenchimento, é necessário percorrer novamente o programa fonte a partir de seu início, agora de posse da tabela de símbolos completamente preenchida.

Isso permitirá completar o preenchimento de endereços referenciados adiante no programa fonte, o que não foi possível até aqui.

A imagem da fita objeto incompleta, gerada até este ponto é:

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

02	9E	11	7?	??	3?	??	8?	??	D0	02	9E	10	25	??	??	??		

Segundo passo de montagem

- Montadores que operam da forma aqui mostrada são chamados montadores de dois passos.
- No primeiro passo, o objetivo é uma tabela de símbolos contendo todas as informações sobre o programa, para que seja possível gerar o código objeto completo.
- No segundo passo, usa-se o conteúdo dessa tabela e o código incompleto gerado no primeiro passo, para compor o código objeto completo na forma de uma fita perfurada (ou sua imagem, em um arquivo em disco).
- O programa fonte é lido pela segunda vez, e à medida que as instruções que lhes deram origem vão sendo reencontradas, as lacunas vão sendo preenchidas com a ajuda das informações coletadas na tabela de símbolos.

Linha 1: ORG 29E

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	7???	2	INICIO	LOAD	X
2A0	3???	3		ADD	Y
2A2	8???	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- Voltando à linha 1, a informação nela contida não é motivo para qualquer modificação no conteúdo da fita objeto.

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

02	9E	11	7?	??	3?	??	8?	??	D0	02	9E	10	25	??	??	??			

Linha 2: INICIO LOAD X

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	72A7	2	INICIO	LOAD	X
2A0	3???	3		ADD	Y
2A2	8???	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- Aqui a referência a X, que havia ficado incompleta, pode ser corrigida: consultando a tabela de símbolos, temos X=2A7

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

02	9E	11	72	A7	3?	??	8?	??	D0	02	9E	10	25	??	??	??		

Linha 3: ADD Y

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	72A7	2	INICIO	LOAD	X
2A0	32A8	3		ADD	Y
2A2	8???	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- Aqui a referência a Y, que havia ficado incompleta, pode ser corrigida: consultando a tabela de símbolos, temos Y=2A8

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

02	9E	11	72	A7	32	A8	8?	??	D0	02	9E	10	25	??	??	??		

Linha 4: STORE Z

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	72A7	2	INICIO	LOAD	X
2A0	32A8	3		ADD	Y
2A2	82A9	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- Aqui a referência a Z, que havia ficado incompleta, pode ser corrigida: consultando a tabela de símbolos, temos Z=2A9

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

02	9E	11	72	A7	32	A8	82	A9	D0	02	9E	10	25	??	??	??	

Linha 5: STOP

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	72A7	2	INICIO	LOAD	X
2A0	32A8	3		ADD	Y
2A2	82A9	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- Aqui não há nada a corrigir.

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

02	9E	11	72	A7	32	A8	82	A9	D0	02	9E	10	25	??	??	??			

Linha 6: JUMP INICIO

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	72A7	2	INICIO	LOAD	X
2A0	32A8	3		ADD	Y
2A2	82A9	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- Aqui também não há nada a corrigir.

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

02	9E	11	72	A7	32	A8	82	A9	D0	02	9E	10	25	??	??	??			

Linha 7: X DATA 10

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	72A7	2	INICIO	LOAD	X
2A0	32A8	3		ADD	Y
2A2	82A9	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- Nada a corrigir.

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

02	9E	11	72	A7	32	A8	82	A9	D0	02	9E	10	25	??	??	??		

Linha 8: Y DATA 25

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	72A7	2	INICIO	LOAD	X
2A0	32A8	3		ADD	Y
2A2	82A9	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- Nada a corrigir.

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

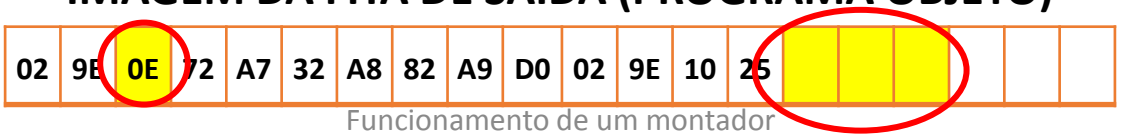
02	9E	11	72	A7	32	A8	82	A9	D0	02	9E	10	25	??	??	??		

Linha 9: Z AREA 3

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	72A7	2	INICIO	LOAD	X
2A0	32A8	3		ADD	Y
2A2	82A9	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- Esta pseudo instrução apenas reserva uma área de 3 bytes, mas não deve preencher essa área com dados.
- Assim, ela não deve gerar nenhum código no programa objeto.
- Portanto, corrigir o número de bytes subtraindo 3 do número de dados.

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)



Linha A: END

Endereço	Código	Linha	rótulo	mnemônico	operando
		1		ORG	29E
29E	72A7	2	INICIO	LOAD	X
2A0	32A8	3		ADD	Y
2A2	82A9	4		STORE	Z
2A4	D0	5		STOP	
2A5	029E	6		JUMP	INICIO
2A7	10	7	X	DATA	10
2A8	25	8	Y	DATA	25
2A9		9	Z	AREA	3
2AC		A		END	

- Esta pseudo instrução sinaliza o final do programa fonte, e promove a conversão da imagem da fita, construída pelo montador na memória, em uma fita física de papel.

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

02	9E	0E	72	A7	32	A8	82	A9	D0	02	9E	10	25					

Fita de papel (saída final do montador)

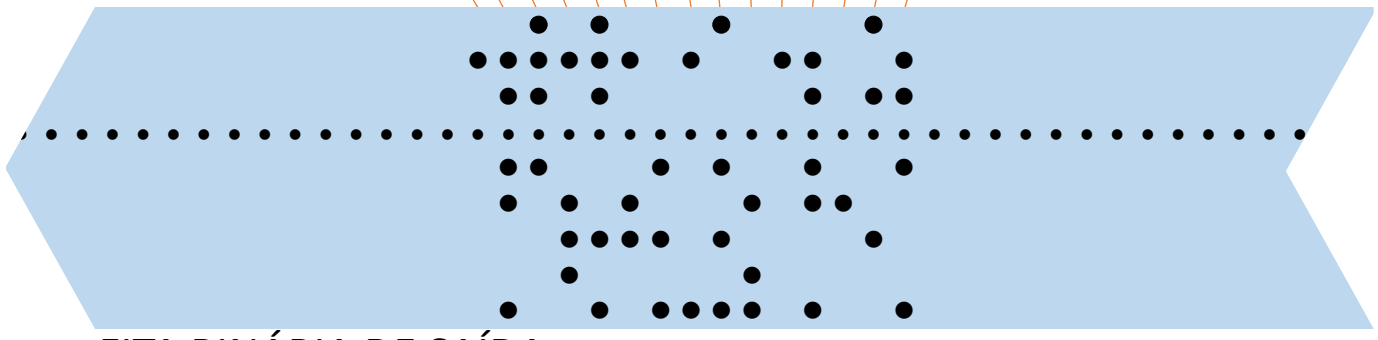
- A imagem hexadecimal abaixo, da fita de papel, construída na memória pelo montador, agora deve ser convertida para binário, e enviada para uma perfuradora de fita, obtendo-se enfim a seguinte fita física de papel:

IMAGEM DA FITA DE SAÍDA (PROGRAMA OBJETO)

02	9E	0F	72	A7	32	A8	82	A9	D0	02	9E	10	25	8E					

Acrescenta-se 1 ao número de bytes devido à inserção do checksum

Inserção de checksum: a soma de todos os bytes anteriores é 572, cujo byte menos significativo é 72. Seu complemento de 2 é 8E (para que a soma seja nula)



FITA BINÁRIA DE SAÍDA

Conclusão

- A **fita objeto** binária assim construída já está em condições de fazer parte de uma biblioteca de programas **prontos para o processamento**.
- Para executar esse programa, é necessário que um **loader binário** do estilo daquele que foi anteriormente estudado transfira para as posições especificadas da memória do computador os dados contidos na fita binária.
- Endereçando-se então a primeira instrução do programa, o processador já poderá iniciar sua execução.
- Percebe-se que o **processo de montagem** manual, embora seja **simples**, é muito **trabalhoso**, portanto adequado para ser automatizado com a ajuda de um programa de computador.
- É exatamente para isso que servem os **montadores**.

FIM