

## **Capítulo 7 – Nível da Linguagem Assembly**

- Presente em quase todos os computadores modernos.
- Implementado por tradução. Linguagem fonte => Linguagem alvo.
  - O programa no arquivo fonte não é executado diretamente => convertido para um programa objeto ou em um programa binário executável.
  - Durante a execução: nível da microarquitetura, ISA e sistema operacional.

### **7.1 Introdução à linguagem Assembly**

- Assembly: A linguagem fonte é uma representação simbólica da linguagem máquina.
  - Assembler: responsável pela tradução
- Compilador: tradutor utilizado quando a linguagem fonte é uma linguagem de alto nível, como C ou Java e a linguagem alvo é a linguagem máquina ou uma representação simbólica.

#### **7.1.1 O que é uma linguagem Assembly ?**

- Linguagem na qual cada comando produz exatamente uma instrução de máquina => correspondência direta entre as instruções de máquina e os comandos do programa em assembly.
- Muito mais fácil trabalhar com assembly do que com a linguagem máquina diretamente.
- O programador em assembly possui acesso a todas as características e instruções disponíveis na máquina alvo.
- Tudo o que pode ser feito pela linguagem de máquina pode ser feito na linguagem assembly.
- Um programa em linguagem assembly só pode ser executado em uma família de máquinas.

#### **7.1.2 Por que utilizar a linguagem Assembly ?**

- Performance
- Acesso a máquina

#### **7.1.3 Formato de um comando em Assembly**

Label	Opcode	Operands	Comments
FORMULA:	MOV	EAX,I	; register EAX = I
	ADD	EAX,J	; register EAX = I + J
	MOV	N,EAX	; N = I + J
I	DW	3	; reserve 4 bytes initialized to 3
J	DW	4	; reserve 4 bytes initialized to 4
N	DW	0	; reserve 4 bytes initialized to 0

(a)

Label	Opcode	Operands	Comments
FORMULA	MOVE.L	I, D0	; register D0 = I
	ADD.L	J, D0	; register D0 = I + J
	MOVE.L	D0, N	; N = I + J
I	DC.L	3	; reserve 4 bytes initialized to 3
J	DC.L	4	; reserve 4 bytes initialized to 4
N	DC.L	0	; reserve 4 bytes initialized to 0

(b)

Label	Opcode	Operands	Comments
FORMULA:	SETHI	%HI(I),%R1	! R1 = high-order bits of the address of I
	LD	[%R1+%LO(I)],%R1	! R1 = I
	SETHI	%HI(J),%R2	! R2 = high-order bits of the address of J
	LD	[%R2+%LO(J)],%R2	! R2 = J
	NOP		! wait for J to arrive from memory
	ADD	%R1,%R2,%R2	! R2 = R1 + R2
	SETHI	%HI(N),%R1	! R1 = high-order bits of the address of N
	ST	%R2,[%R1+%LO(N)]	
I:	.WORD	3	! reserve 4 bytes initialized to 3
J:	.WORD	4	! reserve 4 bytes initialized to 4
N:	.WORD	0	! reserve 4 bytes initialized to 0

(c)

### 7.1.4 Pseudoinstruções

- Um programa em assembly pode conter comandos para o assembler => pseudoinstruções ou diretivas do assembler.
- Exemplo: MASM

```
WORDSIZE EQU 16
```

```
IF WORDSIZE GT 16
```

```
WSIZE DW 32
```

```
ELSE
```

```
WSIZE DW 16
```

```
ENDIF
```

<b>Pseudoinstr</b>	<b>Meaning</b>
SEGMENT	Start a new segment (text, data, etc.) with certain attributes
ENDS	End the current segment
ALIGN	Control the alignment of the next instruction or data
EQU	Define a new symbol equal to a given expression
DB	Allocate storage for one or more (initialized) bytes
DD	Allocate storage for one or more (initialized) 16-bit halfwords
DW	Allocate storage for one or more (initialized) 32-bit words
DQ	Allocate storage for one or more (initialized) 64-bit double words
PROC	Start a procedure
ENDP	End a procedure
MACRO	Start a macro definition
ENDM	End a macro definition
PUBLIC	Export a name defined in this module
EXTERN	Import a name from another module
INCLUDE	Fetch and include another file
IF	Start conditional assembly based on a given expression
ELSE	Start conditional assembly if the IF condition above was false
ENDIF	End conditional assembly
COMMENT	Define a new start-of-comment character
PAGE	Generate a page break in the listing
END	Terminate the assembly program

## 7.2 Macros

- Necessidade de repetir seqüências de instruções diversas vezes em um programa.

### 7.2.1 Definição, Chamadas e Expansão de Macros

- Definição de macro: Forma de fornecer um nome a uma parte de texto.
  - Após a definição da macro basta fornecer o seu nome.
- Chamadas de macro: Utilização do nome da macro
- Expansão de macro: substituição do nome da macro pelo seu corpo.
  - Ocorre durante o processo de montagem, pelo assembler.

<pre> MOV  EAX,P MOV  EBX,Q MOV  Q,EAX MOV  P,EBX  MOV  EAX,P MOV  EBX,Q MOV  Q,EAX MOV  P,EBX           </pre> <p style="text-align: center;">(a)</p>	<pre> SWAP  MACRO       MOV EAX,P       MOV EBX,Q       MOV Q,EAX       MOV P,EBX       ENDM  SWAP  SWAP           </pre> <p style="text-align: center;">(b)</p>
--	--

- A macro é uma instrução para o assembler.
- Uma chamada de procedimento é uma instrução para a máquina.

Item	Macro call	Procedure call
When is the call made?	During assembly	During execution
Is the body inserted into the object program every place the call is made?	Yes	No
Is a procedure call instruction inserted into the object program and later executed?	No	Yes
Must a return instruction be used after the call is done?	No	Yes
How many copies of the body appear in the object program?	One per macro call	1

### 7.2.2 Macros com parâmetros

- Geralmente um programa possui uma seqüência de comandos que não são exatamente iguais, mas parecidos => macros com parâmetros.

<pre> MOV  EAX,P MOV  EBX,Q MOV  Q,EAX MOV  P,EBX  MOV  EAX,R MOV  EBX,S MOV  S,EAX MOV  R,EBX </pre>	<pre> CHANGE  MACRO P1, P2 MOV EAX,P1 MOV EBX,P2 MOV P2,EAX MOV P1,EBX ENDM  CHANGE P, Q  CHANGE R, S </pre>
(a)	(b)

### 7.2.3 Características Avançadas

- Problema com macro: Duplicação de labels (alvos)
  - Nome do label passado como parâmetro
  - Declaração do label como sendo local => geração de um novo label a cada expansão
  - Numeração automática do label a cada expansão
- Definição de uma macro dentro de outra e utilizando condicional.
- Macro chamando outra macro.

### 7.2.4 Implementação de macros em um Assembler

- O assembler deve ser capaz de:
  - Salvar a definição da macro
  - Expandir a macro em suas chamadas

## 7.3 Processo de Montagem

### 7.3.1 Assemblers de duas Etapas

- Problema de referência futura
  - Leitura do programa fonte em duas etapas
    - Etapa 1: Armazenamento de definições de símbolos e macros em tabelas. Expansão das macros.
    - Etapa 2: leitura e tradução de cada comando.
  - Leitura e conversão do programa em uma forma intermediária, armazenada na memória. A segunda etapa é feita no forma intermediária e não no programa fonte.

### 7.3.2 Primeira Etapa

- Criação da tabela de símbolos: contém os valores de todos os símbolos.
  - Símbolo: label ou valor associado a um nome simbólico por pseudoinstrução
  - Label: necessidade de conhecer o endereço da instrução durante a execução => ILC (Instruction Location Counter).

Label	Opcode	Operands	Comments	Length	ILC
MARIA:	MOV	EAX, I	EAX = I	5	100
	MOV	EBX, J	EBX = J	6	105
ROBERTA:	MOV	ECX, K	ECX = K	6	111
	IMUL	EAX, EAX	EAX = I * I	2	117
	IMUL	EBX, EBX	EBX = J * J	3	119
MARILYN:	IMUL	ECX, ECX	ECX = K * K	3	122
	ADD	EAX, EBX	EAX = I * I + J * J	2	125
	ADD	EAX, ECX	EAX = I * I + J * J + K * K	2	127
STEPHANY:	JMP	DONE	branch to DONE	5	129

- Maioria dos assembler utilizam três tabelas na etapa 1:
  - Tabela de símbolos
  - Tabela de pseudoinstruções
  - Tabela de opcodes

Symbol	Value	Other information
MARIA	100	
ROBERTA	111	
MARILYN	125	
STEPHANY	129	

Opcode	First operand	Second operand	Hexadecimal opcode	Instruction length	Instruction class
AAA	—	—	37	1	6
ADD	EAX	immed32	05	5	4
ADD	reg	reg	01	2	19
AND	EAX	immed32	25	5	4
AND	reg	reg	21	2	19

### 7.3.3 Segunda Etapa

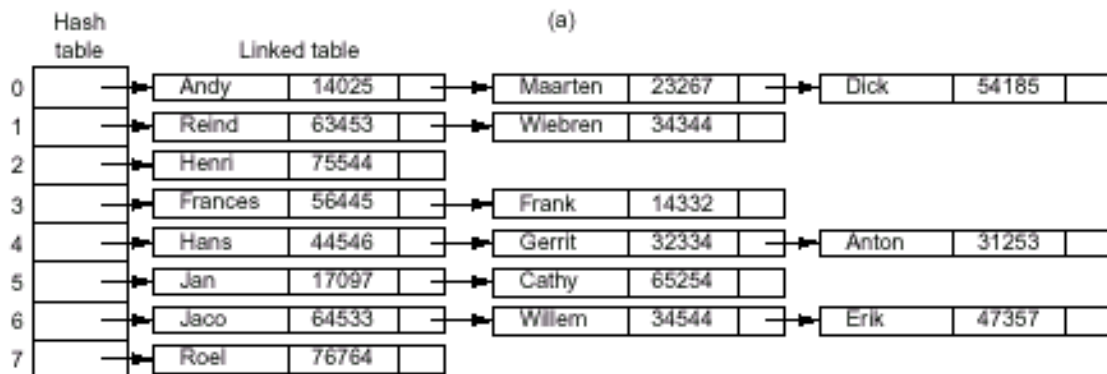
- Geração do programa objeto e possível listagem do assembly.

- Utilização da tabela de símbolos.
- Fornecimento de certas informações necessárias na linkagem

### 7.3.4 Tabela de Símbolos

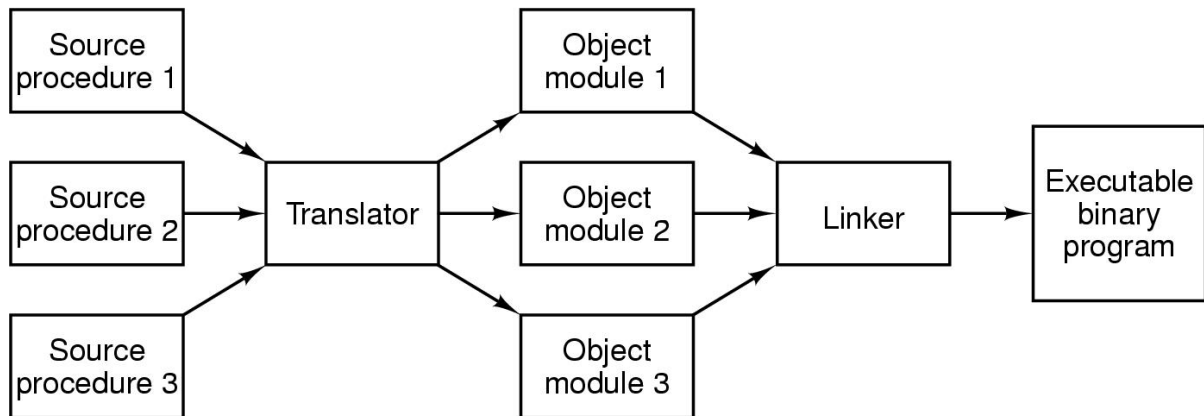
- Implementada como um memória associativa:
  - conjunto de pares símbolo:valor
- Busca linear por determinado símbolo:
- Algoritmo de busca binária: Tabela organizada alfabeticamente
- Hash coding: função que mapeia os símbolos em inteiros de 0 a  $k-1$ . Cada símbolo é, então, armazenado em um slot correspondente ao seu número. Símbolos de mesmo número são colocados em cadeia.

Andy	14025	0
Anton	31253	4
Cathy	65254	5
Dick	54185	0
Erik	47357	6
Frances	56445	3
Frank	14332	3
Gerrit	32334	4
Hans	44546	4
Henri	75544	2
Jan	17097	5
Jaco	64533	6
Maarten	23267	0
Reind	63453	1
Roel	76764	7
Willem	34544	6
Wiebren	34344	1



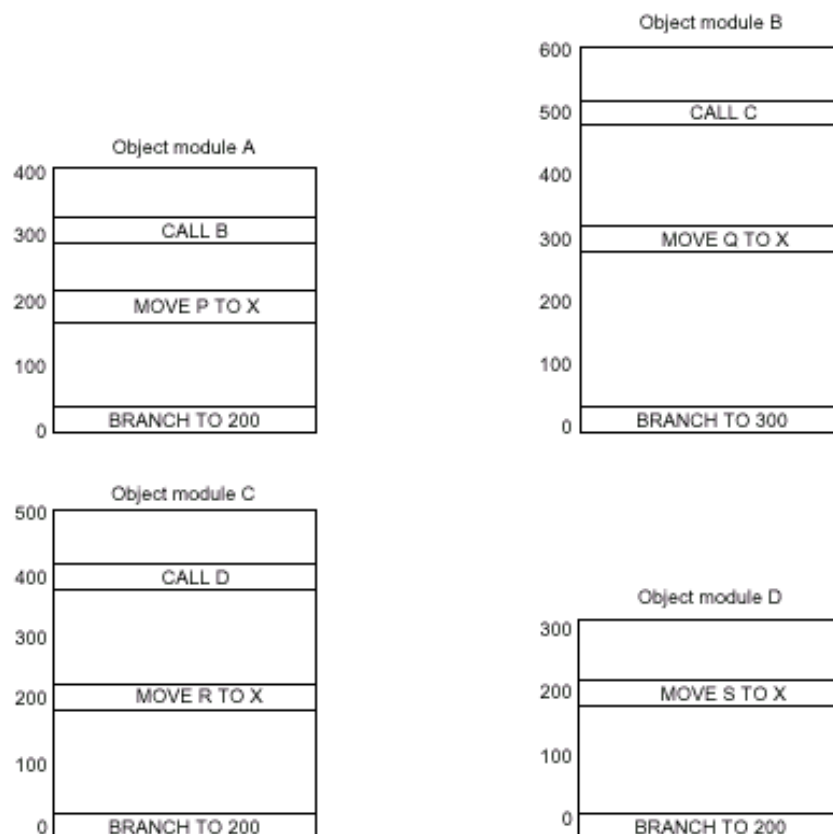
### 7.4 Ligação e Carregamento

- Maioria dos programas: mais de um procedimento => antes da execução todos os procedimentos compilados ou traduzidos devem ser montados juntos, adequadamente.
- Execução de um programa:
  - Compilação ou montagem do fonte dos procedimentos
  - Ligação dos módulos objetos



### 7.4.1 Tarefas Executadas pelo Ligador

- Problema de Realocação: Cada módulo é gerado considerando o endereço base igual a 0.
- Problema de referência externa: Um módulo pode chamar uma rotina definida em outro módulo.

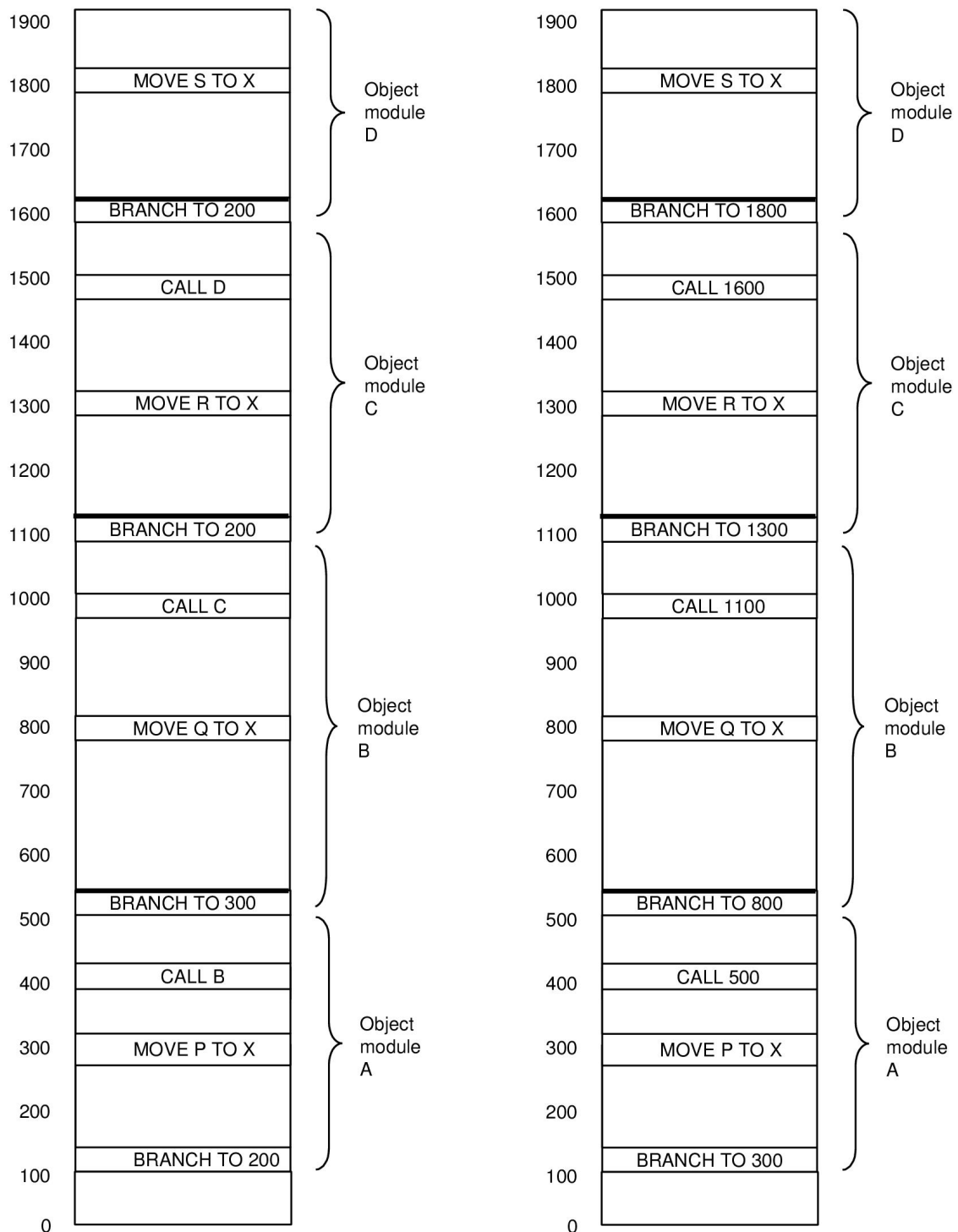


– O ligador deve unir os espaços de endereçamento de cada módulo em um único espaço de endereçamento:

- 1 – Constrói uma tabela de todos os módulos e seus tamanhos
- 2 – Baseada na tabela ele marca um endereço inicial para cada módulo

3 – Procura por todas as instruções que referenciam a memória e adiciona a cada uma constante de realocação , igual ao endereço inicial do módulo.

4 – Procura por instruções que referenciam outros procedimentos, inserindo o endereço destes procedimentos.



### 7.4.2 Estrutura de um módulo Objeto

– Dividido em seis partes:

1 – Identificação: Nome do módulo e informações necessárias ao ligador

2 – Lista de símbolos definidas no módulo que outros módulos podem referenciar, bem como seus valores.

3 – Lista de símbolos que são utilizados no módulo mas que são definidos em outros módulos juntamente com a lista de quais instruções de máquina que utilizam estes símbolos.

4 – Código montado e constantes. Parte carregada na memória e que será executada

5 – Dicionário de relocação. Informa quais instruções utilizam endereços e quais utilizam constantes. A informação sobre quais endereços devem ser realocados está presente nesta parte.

6 – Indicação de fim de módulo. Pode ser um checksum.

End of module
Relocation dictionary
Machine instructions and constants
External reference table
Entry point table
Identification

– Maioria dos ligadores: Necessárias duas etapas:

– Leitura de todos os módulos objetos e construção da tabela de nomes de módulos e tamanhos, além da tabela de símbolos globais

– Leitura dos módulos, relocação e ligação de um módulo por vez.

### 7.4.3 Binding Time e Realocação dinâmica

– Binding time: momento no qual um endereço real de memória é determinado:

– Quando o programa é escrito

– Quando o programa é montado

– Quando o programa é ligado mas antes de ser carregado

– Quando o programa é carregado

– Quando um registrador base utilizado para o endereçamento é carregado

– Quando a instrução contendo o endereço é executada

- Problema quando o programa é movido na memória principal => realocação
  - Páginas: apenas a tabela de páginas precisa ser alterada, não o programa
  - Runtime Relocation Register: O registrador sempre aponta para o endereço físico de memória do começo do programa que está sendo executado. Todas as referências de memória são adicionadas ao registrador de realocação, por hardware, antes de acessarem a memória.
  - Acesso a memória com endereço relativo ao valor de PC => independente da posição.

#### **7.4.4 Ligação Dinâmica**

- Maioria dos programas possuem módulos que só são chamados em condições muito especiais.
- Ligação dinâmica: Cada módulo é ligado na primeira vez que ele for chamado.