

Assembly Language for Intel-Based Computers, 5th Edition

Kip Irvine

Capítulo 3: Linguagem Assembly Fundamentos

Slides prepared by the author

Revision date: June 4, 2006

(c) Pearson Education, 2006-2007. All rights reserved. You may modify and copy this slide show for your personal use, or for use in the classroom, as long as this copyright statement, the author's name, and the title are not changed.

Tópicos da Aula

- Elementos básicos da linguagem Assembly
- Exemplo: adição e subtração de inteiros
- Montagem, “Linking”, e execução de programas
- Definição de dados
- Constantes simbólicas

Elementos básicos da linguagem “Assembly”

- Mnemônicos e operandos
- Instruções
- Labels (Rótulos)
- Palavras reservadas e identificadores
- Constantes de inteiros
- Expressões de inteiros
- Constantes de caracteres e cadeias
- Diretivas
- Comentários

Mnemônicos e Operandos

- Mnemônicos de instruções
 - ajuda a memorização
 - exemplos: MOV, ADD, SUB, MUL, INC, DEC
- Operandos
 - registrador
 - conteúdo na memória (label de dados c/ endereço)
 - constante
 - expressão de constantes

Constantes e expressões de constantes são também chamadas de valores imediatos

Instruções

- Traduzidas em código de máquina pelo assembler
- Executado em tempo de execução pelo CPU
- Nesta disciplina usam-se as instruções do Intel IA-32
- Todas as instruções contêm:
 - Label (opcional)
 - Mnemônico (necessário)
 - Operandos (depende da instrução)
 - Comentário (opcional)

Exemplo de formato de instruções

Sem operandos

stc ; set Carry flag

Um operando

inc eax ; register

inc myByte ; memory

Dois operandos

add ebx,ecx ; register, register

sub myByte,25 ; memory, constant

add eax,36 * 25 ; register, constant-expression

Labels (rótulos)

- Funcionam como marcadores de posição
 - marca o endereço de códigos e dados
- Segue a mesma regra dos identificadores
- Label para dados
 - deve ser único
 - exemplo: **myArray** (não tem dois pontos)
- Label para códigos
 - posição de instruções de **jump** e **loop**
 - exemplo: **L1:** (seguido por dois pontos)

Palavras reservadas e identificadores

- Palavras reservadas (não podem ser usadas como identificadores)
 - Mnemônicos de instruções, diretivas, atributos de tipo, operadores, símbolos pré-definidos
- Identificadores
 - 1-247 caracteres, incluindo dígitos
 - Insensível à caixa (maiúscula e minúscula)
 - primeiro caractere deve ser *letra, _, @, ?, ou \$*

Constantes de Inteiros

- Opção de ser acompanhado pelos sinais + ou –
- Dígitos: binário, decimal, hexadecimal, ou octal
- Sufixos comuns:
 - h – hexadecimal
 - d – decimal
 - b – binário
 - r – real

Exemplos: 30d, 6Ah, 42, 1101b

Hexadecimal começando com letra: 0A5h

Obs: Default: decimal – Ex: 25 = 25d

Expressões de Inteiros

- Operadores e níveis de precedência:

Operator	Name	Precedence Level
()	parentheses	1
+ , -	unary plus, minus	2
* , /	multiply, divide	3
MOD	modulus	3
+ , -	add, subtract	4

- Exemplos:

Expression	Value
16 / 5	3
-(3 + 4) * (6 - 1)	-35
-3 + 4 * 6 - 1	20
25 mod 3	1

Constantes de caracteres e cadeias

- Delimitar caracteres usando apóstrofos ou aspas
 - 'A', "x"
 - Caractere ASCII = 1 byte
- Delimitar cadeias usando apóstrofos ou aspas
 - "ABC"
 - 'xyz'
 - Cada caracter ocupa um byte numa cadeia
- Cadeia dentro da outra:
 - 'Say "Goodnight," Gracie'

Diretivas

- Comandos que são reconhecidos pelo **assembler**
 - Não fazem parte do conjunto de instruções do processador
 - Usados para declarar áreas de código, áreas de dados, selecionar modo de memória, declarar procedimentos, etc.
 - Insensível à caixa
- Diferentes assemblers podem ter diferentes diretivas
 - Por exemplo, NASM não é igual a MASM
 - Isso resulta em uma incompatibilidade de código fonte assembly, que é diferente de incompatibilidade de código de máquina.

Comentários

- Comentários são úteis!
 - explica o propósito do programa
 - quando foi escrito e o autor
 - informação de revisão
 - técnicas e detalhes de codificação
 - explicação específica da aplicação
- Comentários numa mesma linha
 - Inicia-se com ponto e vírgula (;)
- Comentários em múltiplas linhas
 - **começa** com a diretiva COMMENT e um caracter escolhido pelo programador
 - **termina** com o mesmo caracter escolhido pelo programador

Próxima seção

- Elementos básicos da linguagem Assembly
- **Exemplo: adição e subtração de inteiros**
- Montagem, “Linking”, e execução de programas
- Definição de dados
- Constantes simbólicas

Exemplo: Adição e Subtração de Inteiros (AddSub.asm)

```
TITLE Add and Subtract                (AddSub.asm)

; Este programa soma e subtrai inteiros de 32-bits.

INCLUDE Irvine32.inc
.code
main PROC
    mov eax,10000h                    ; EAX = 10000h
    add eax,40000h                    ; EAX = 50000h
    sub eax,20000h                    ; EAX = 30000h
    call DumpRegs                    ; display registers
    exit
main ENDP
END main
```

Exemplo de saída

Saída do programa, mostrando registradores e flags:

EAX=00030000	EBX=7FFDF000	ECX=00000101	EDX=FFFFFFFF
ESI=00000000	EDI=00000000	EBP=0012FFF0	ESP=0012FFC4
EIP=00401024	EFL=00000206	CF=0	SF=0 ZF=0 OF=0

Sugestão de padrões de codificação (1 de 2)

- Algumas abordagens para uso de maiúsculas
 - não usar maiúsculas
 - usar só maiúsculas
 - maiúsculas para palavras reservadas, incluindo mnemônicos de instruções e nomes de registradores
 - maiúsculas somente para diretivas e operadores
- Outras sugestões
 - nomes de identificadores descritivos
 - espaços ao redor dos operadores aritméticos
 - linhas em branco entre procedimentos

Sugestão de padrões de codificação (2 de 2)

- Indentação e espaçamento
 - labels de código e dados – sem indentação
 - instruções executáveis – indentar 4-5 espaços
 - comentários: começar na coluna 40-45, alinhados verticalmente
 - 1-3 espaços entre mnemônico da instrução e operandos
 - ex: `mov ax,bx`
 - 1-2 linhas em branco entre procedimentos

Versão alternativa do AddSub.asm

```
TITLE Add and Subtract                                (AddSubAlt.asm)

; este programa soma e subtrai inteiros de 32-bits.
.386
.MODEL flat,stdcall
.STACK 4096

ExitProcess PROTO, dwExitCode:DWORD
DumpRegs PROTO

.code
main PROC
    mov eax,10000h                ; EAX = 10000h
    add eax,40000h                ; EAX = 50000h
    sub eax,20000h                ; EAX = 30000h
    call DumpRegs
    INVOKE ExitProcess,0
main ENDP
END main
```

Template de programa ASM

```
TITLE Program Template                (Template.asm)
```

```
; Program Description:
```

```
; Author:
```

```
; Creation Date:
```

```
; Revisions:
```

```
; Date:                               Modified by:
```

Instructors: please
customize as needed

```
INCLUDE Irvine32.inc
```

```
.data
```

```
    ; (insert variables here)
```

```
.code
```

```
main PROC
```

```
    ; (insert executable instructions here)
```

```
    exit
```

```
main ENDP
```

```
    ; (insert additional procedures here)
```

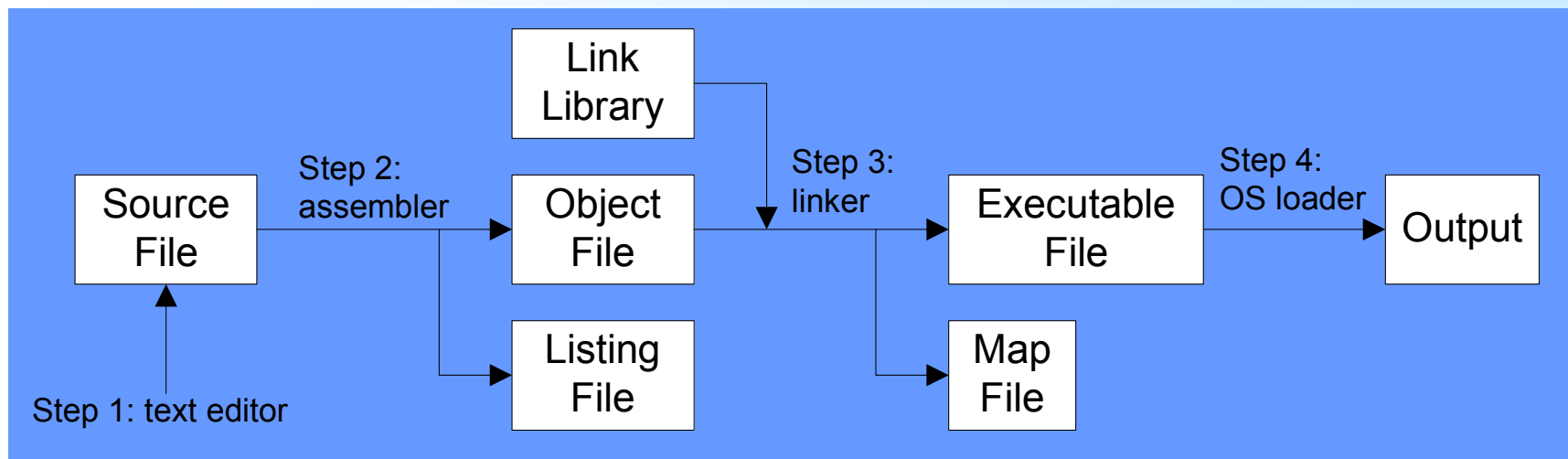
```
END main
```

Próxima seção

- Elementos básicos da linguagem Assembly
- Exemplo: adição e subtração de inteiros
- Montagem, “Linking”, e execução de programas
- Definição de dados
- Constantes simbólicas

Ciclos de Montagem, “Linkagem” e Execução

- O seguinte diagrama descreve os passos a partir da criação do programa fonte até a execução do programa.
- Se o código fonte é modificado, os passos 2 a 4 devem se repetir.



Arquivo de Listagem (List File)

- Mostra como o programa é montado
- Contem
 - código fonte
 - endereços
 - código objeto (linguagem de máquina)
 - nomes de segmentos
 - símbolos (variáveis, procedimentos, e constantes)

Arquivo de Mapeamento (Map File)

- Informações sobre cada segmento de programa
 - endereço de início
 - endereço de fim
 - tamanho
 - tipo de segmento

Próxima seção

- Elementos básicos da linguagem Assembly
- Exemplo: adição e subtração de inteiros
- Montagem, “Linking”, e execução de programas
- **Definição de dados**
- Constantes simbólicas



Intervalo ?

Definição de dados

- Tipos de dados intrínsecos
- Sentenças para definição de dados
- Definição de BYTE e SBYTE
- Definição de WORD e SWORD
- Definição de DWORD e SDWORD
- Definição de QWORD
- Definição de TBYTE
- Definição número Real
- Ordem Little Endian
- Adicionando variáveis ao programa AddSub
- Declaração de dados não-inicializados

Tipos de dados instrínsecos (1 de 2)

- BYTE, SBYTE
 - 8-bit unsigned integer; 8-bit signed integer
- WORD, SWORD
 - 16-bit unsigned & signed integer
- DWORD, SDWORD
 - 32-bit unsigned & signed integer
- QWORD
 - 64-bit integer
- TBYTE
 - 80-bit integer

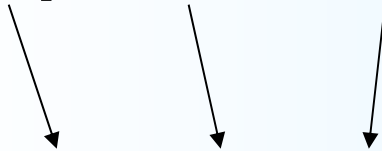
Tipos de datos intrínsecos (2 de 2)

- REAL4
 - 4-byte IEEE short real
- REAL8
 - 8-byte IEEE long real
- REAL10
 - 10-byte IEEE extended real

Sentença de definição de dados

- Uma sentença de definição de dados define a forma de armazenamento da variável na memória.
- Pode opcionalmente atribuir um nome (label) ao dado
- Sintaxe:

[name] directive initializer [,initializer] . . .



value1 BYTE 10

- *Todos os inicializadores (initializers) são traduzidos pelo assembler em dados binários na memória*

Definição de dados BYTE e SBYTE

Cada sentença seguinte define o armazenamento de um byte:

<code>value1 BYTE 'A'</code>	<code>; character constant</code>
<code>value2 BYTE 0</code>	<code>; smallest unsigned byte</code>
<code>value3 BYTE 255</code>	<code>; largest unsigned byte</code>
<code>value4 SBYTE -128</code>	<code>; smallest signed byte</code>
<code>value5 SBYTE +127</code>	<code>; largest signed byte</code>
<code>value6 BYTE ?</code>	<code>; uninitialized byte</code>

- Se for declarada uma variável SBYTE, o Microsoft debugger mostra automaticamente o seu valor decimal com sinal.

Definição de *Byte Arrays* (vetores de bytes)

Usam múltiplos inicializadores:

```
list1 BYTE 10,20,30,40
list2 BYTE 10,20,30,40
        BYTE 50,60,70,80
        BYTE 81,82,83,84
list3 BYTE ?,32,41h,00100010b
list4 BYTE 0Ah,20h,'A',22h
```

Definição de cadeias (Strings) (1 de 3)

- Uma cadeia é implementada como um vetor de caracteres
 - por conveniência, é usualmente cercada com apóstrofes ou aspas
 - geralmente termina com zero
- Exemplos:

```
str1 BYTE "Enter your name",0
str2 BYTE 'Error: halting program',0
str3 BYTE 'A','E','I','O','U'
greeting BYTE "Welcome to the Encryption program "
          BYTE "created by Kip Irvine.",0
```


Definição de cadeias (Strings) (2 de 3)

- Para continuar uma cadeia na linha seguinte, essa linha deve terminar com vírgula:

```
menu BYTE "Checking Account",0dh,0ah,0dh,0ah,  
        "1. Create a new account",0dh,0ah,  
        "2. Open an existing account",0dh,0ah,  
        "3. Credit the account",0dh,0ah,  
        "4. Debit the account",0dh,0ah,  
        "5. Exit",0ah,0ah,  
        "Choice> ",0
```

Definição de cadeias (Strings) (3 de 3)

- Seqüência de caracteres de fim de linha e linha seguinte:
 - 0Dh = carriage return
 - 0Ah = line feed

```
str1 BYTE "Enter your name:    ",0Dh,0Ah  
      BYTE "Enter your address: ",0  
  
newLine BYTE 0Dh,0Ah,0
```

Dica: Definir todas as cadeias usadas no programa na mesma área do segmento de dados.

Usando o operador DUP

- Usar DUP para alocar (criar espaço para) um vetor ou cadeia.
- Sintaxe: *counter* DUP (*argument*)
- *Counter* e *argument* devem ser constantes ou expressões de constantes
 - counter* – indica o número de dados
 - argument* – indica o valor do dado

```
var1 BYTE 20 DUP(0)           ; 20 bytes, all equal to zero
var2 BYTE 20 DUP(?)           ; 20 bytes, uninitialized
var3 BYTE 4 DUP("STACK")      ; 20 bytes: "STACKSTACKSTACKSTACK"
var4 BYTE 10,3 DUP(0),20      ; 5 bytes
```

Definição de WORD e SWORD

- Definição de armazenamento de inteiros de 16-bits
 - ou dupla de caracteres
 - valor único ou múltiplos valores

<code>word1</code>	<code>WORD</code>	<code>65535</code>	<code>; largest unsigned value</code>
<code>word2</code>	<code>SWORD</code>	<code>-32768</code>	<code>; smallest signed value</code>
<code>word3</code>	<code>WORD</code>	<code>?</code>	<code>; uninitialized, unsigned</code>
<code>word4</code>	<code>WORD</code>	<code>"AB"</code>	<code>; double characters</code>
<code>myList</code>	<code>WORD</code>	<code>1,2,3,4,5</code>	<code>; array of words</code>
<code>array</code>	<code>WORD</code>	<code>5 DUP(?)</code>	<code>; uninitialized array</code>

Definição de DWORD e SDWORD

Definição de armazenamento de inteiros não-sinalizados ou sinalizados de 32-bits:

```
val1 DWORD 12345678h ; unsigned
val2 SDWORD -2147483648 ; signed
val3 DWORD 20 DUP(?) ; unsigned array
val4 SDWORD -3,-2,-1,0,1 ; signed array
```

Definição de dados QWORD, TBYTE e Real

Definição de armazenamento para quadwords, tenbytes, e números reais:

```
quad1 QWORD 1234567812345678h
val1 TBYTE 1000000000123456789Ah
rVal1 REAL4 -2.1
rVal2 REAL8 3.2E-260
rVal3 REAL10 4.6E+4096
ShortArray REAL4 20 DUP(0.0)
```

Ordem Little Endian

- Na arquitetura IA-32 todos os tipos de dados maiores que um byte armazenam os seus bytes na **memória** em ordem reversa, o seja, o **byte meno significativo** no **menor endereço de memória**.

- Exemplo:

`val1 DWORD 12345678h`

0000:	78
0001:	56
0002:	34
0003:	12

*Na prática, em Little Endian os bytes de um dado registrador ou valor constante aparecem **invertidos** na memória.
Não esqueça isso!*

Adicionando Variáveis ao AddSub

```
TITLE Add and Subtract, Version 2                (AddSub2.asm)
; este programa soma e subtrai inteiros não-sinalizados de 32-bits
; e armazena a soma numa variavel.
INCLUDE Irvine32.inc
.data
val1 DWORD 10000h
val2 DWORD 40000h
val3 DWORD 20000h
finalVal DWORD ?
.code
main PROC
    mov eax,val1                ; start with 10000h
    add eax,val2                ; add 40000h
    sub eax,val3                ; subtract 20000h
    mov finalVal,eax            ; store the result (30000h)
    call DumpRegs              ; display the registers
    exit
main ENDP
END main
```


Declaração de dados não-inicializados

- Usar a diretiva `.data?` para declarar um segmento de dados não inicializados:

```
.data?
```

- Dentro do segmento, declarar variáveis com inicializadores `"?"` :
`smallArray DWORD 10 DUP(?)`

vantagem: reduz o tamanho do arquivo EXE do programa.

Próxima seção

- Elementos básicos da linguagem Assembly
- Exemplo: adição e subtração de inteiros
- Montagem, “Linking”, e execução de programas
- Definição de dados
- **Constantes simbólicas**

Constantes simbólicas

(representação de constantes usando símbolos)

- Diretiva de sinal de igual
- Cálculo de tamanho de vetores e cadeias
- Diretiva EQU
- Diretiva TEXTEQU

Diretiva de sinal de igual (=)

- *name = expression (expressão de inteiros)*
 - expression é um **inteiro** de 32-bits (expressão ou constante)
 - **pode ser redefinido**
 - *name* é chamado de constante simbólica
- Um bom estilo de programação é usar símbolos

```
COUNT = 500  
  
.  
.  
mov al,COUNT
```

Cálculo do tamanho de um vetor de bytes

- endereço da posição atual, onde está sendo calculado é dado pelo contador de posição: **\$**
 - subtrair o endereço de *list*
 - a diferença é o número de bytes

```
list BYTE 10,20,30,40  
ListSize = ($ - list)
```

Cálculo do tamanho de um vetor de Words (tamanho = número de words)

Divide o número total de bytes por 2 (tamanho de um word)

```
list WORD 1000h,2000h,3000h,4000h  
ListSize = ($ - list) / 2
```

Cálculo do tamanho de um vetor de Doublewords

Divide o número total de bytes por 4 (tamanho de um doubleword)

```
list DWORD 1,2,3,4  
ListSize = ($ - list) / 4
```

Diretiva EQU

- Define um símbolo como um inteiro ou expressão de texto
 - *name EQU expression (expressão de inteiros)*
 - *name EQU symbol (símbolo que foi definido anteriormente)*
 - *Name EQU <text>*
- Não pode ser redefinido

Diretiva EQU

Exemplos:

```
PI EQU <3.1416>
```

```
DIM_1 EQU 10 * 10
```

```
DIM_2 EQU <10 * 10>
```

```
pressKey EQU <"Press any key to continue...",0>
```

```
.data
```

```
prompt BYTE pressKey
```

```
Matriz1 WORD DIM_1       $\longrightarrow$  100
```

```
Matriz2 WORD DIM_2       $\longrightarrow$  10 X 10
```

Diretiva TEXT EQU

- Define um símbolo como um inteiro ou expressão de inteiros ou texto.
- Chamado um macro de texto
 - *name* TEXT EQU %constexpression (expressão de inteiros)
 - *name* TEXT EQU symbol (definido anteriormente)
 - *name* TEXT EQU <text>
- Pode ser redefinido

Diretiva TEXTEQU

Exemplos:

```
continueMsg TEXTEQU <"Do you wish to continue (Y/N)?">
rowSize = 5
.data
prompt1 BYTE continueMsg
count TEXTEQU %(rowSize * 2)      ; evaluates the expression
setupAL TEXTEQU <mov al,count>

.code
setupAL                          ; generates: "mov al,10"
```

Exercícios

Resolver os exercícios da lista da semana (no Moodle)

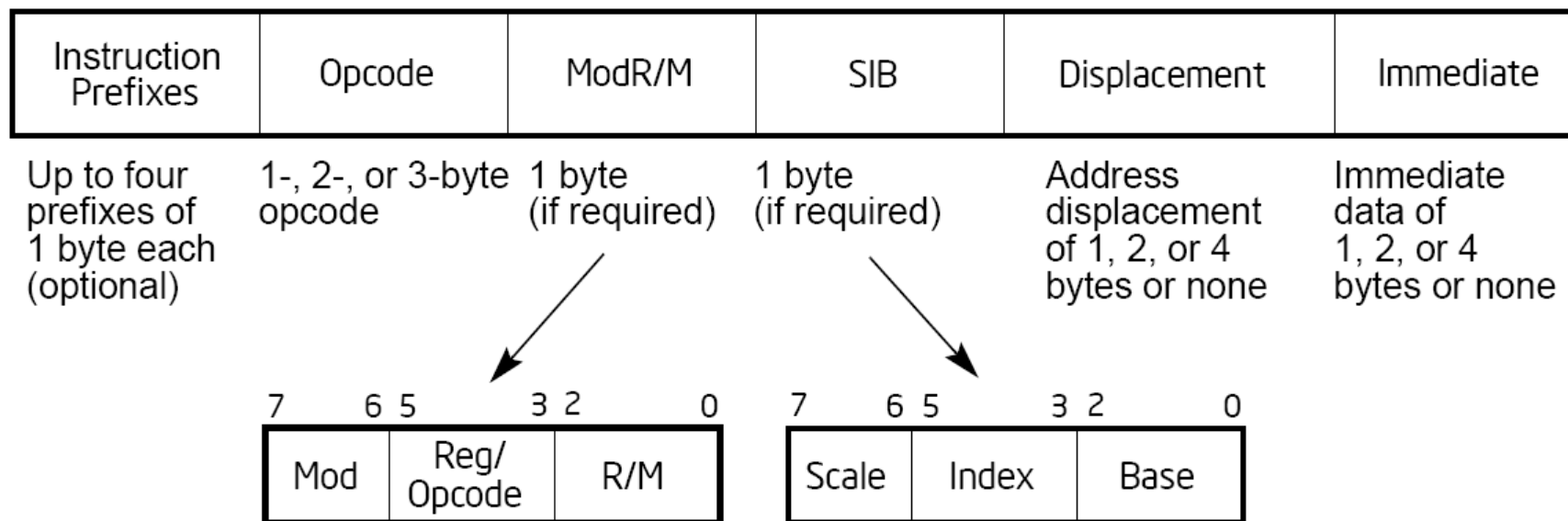
Arquitetura: Aspecto Técnico

Característica do Conjunto de Instruções IA-32 →

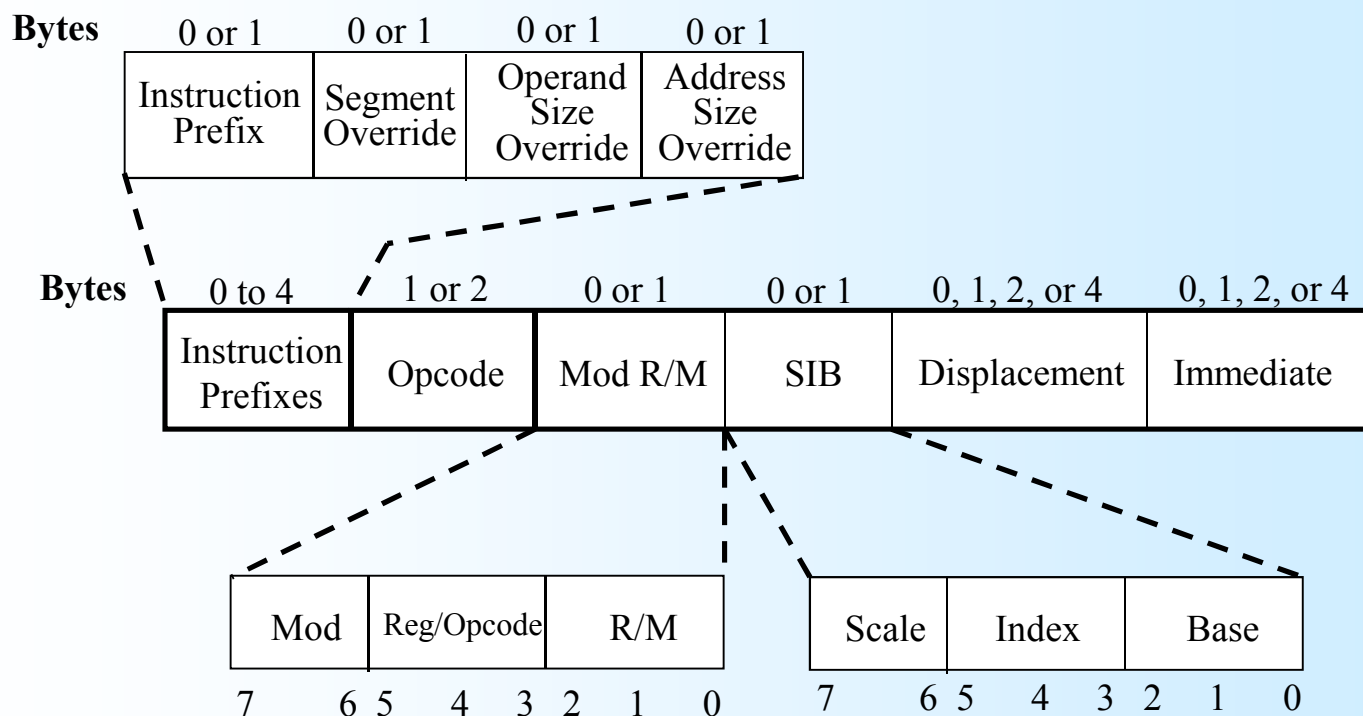
Formato de instruções Intel IA-32

- Instruções de tamanho variável
 - Decisões de projeto desde o processador 8086 são empecilhos para arquiteturas modernas
 - Uma instrução pode ter de 1 a 17 bytes !!
 - Que parte da cpu é mais sobrecarregada por esse fato ?

Formato de instruções Intel IA-32



Formato de instruções Intel IA-32



Formato de instruções Intel IA-32

- Campos individuais do formato de instruções:
 - Prefix (0-4 bytes)
 - Opcode (1-3 bytes)
 - R/M Modifier (0-1 byte)
 - SIB Modifier (0-1 byte)
 - Displacement Modifier (0-4 bytes)
 - Data elements (0-4 bytes)

Formato de instruções Intel IA-32

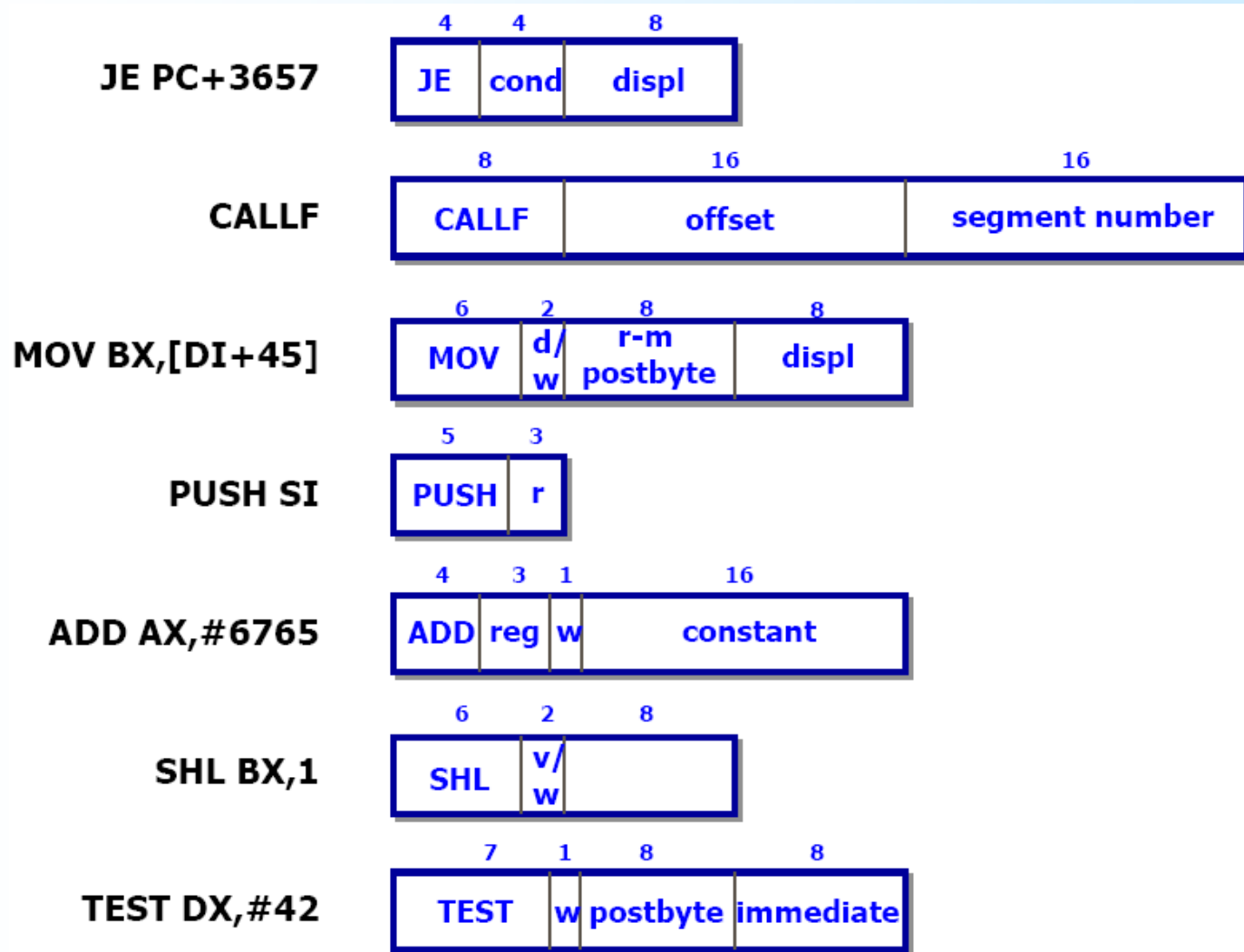
- Prefix (0-4 bytes)
 - Alerta a CPU que o tamanho de um endereço ou operando será modificado
- Opcode (1-3 bytes)
 - Código da operação a ser executada. Operações comuns tem apenas um byte, outras menos frequentes podem ter até três.
- R/M Modifier (0-1 byte)
 - Especifica o modo de endereçamento: Registrador ou Memória

Formato de instruções Intel IA-32

- SIB: Scale / Index / Base (0-1 byte)
 - Indica se o registrador é usado como índice ou base, além de determinar o fator de escala
- Displacement Modifier (0-4 bytes)
 - Offset de dados adicional
- Data elements (0-4 bytes)
 - Imediatos (constantes): dados ou endereços

Formato de instruções Intel IA-32

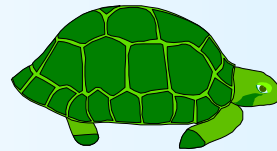
Exemplos



Formato de instruções Intel IA-32

Ponto crucial em termos de arquitetura:

- Formato variável e complexo resulta em uma unidade de controle mais complicada.



46 69 6E 69 73