Arquitetura e Organização de Computadores

Linguagem de Montagem e Linguagem de Máquina

Givanaldo Rocha de Souza
http://docente.ifrn.edu.br/givanaldorocha
givanaldo.rocha@ifrn.edu.br

Conceitos básicos

- **Linguagem/código de máquina**

Instruções que o processador é capaz de executar. Essas instruções, chamadas de código de máquina, são representadas por sequências de bits, normalmente limitadas pelo número de bits do registrador principal (8, 16, 32, 64 ou 128) da CPU.

Conceitos básicos

- **Linguagem de montagem ou assembly**

Notação legível por humanos para o código de máquina que uma arquitetura de computador específica utiliza.

Conceitos básicos

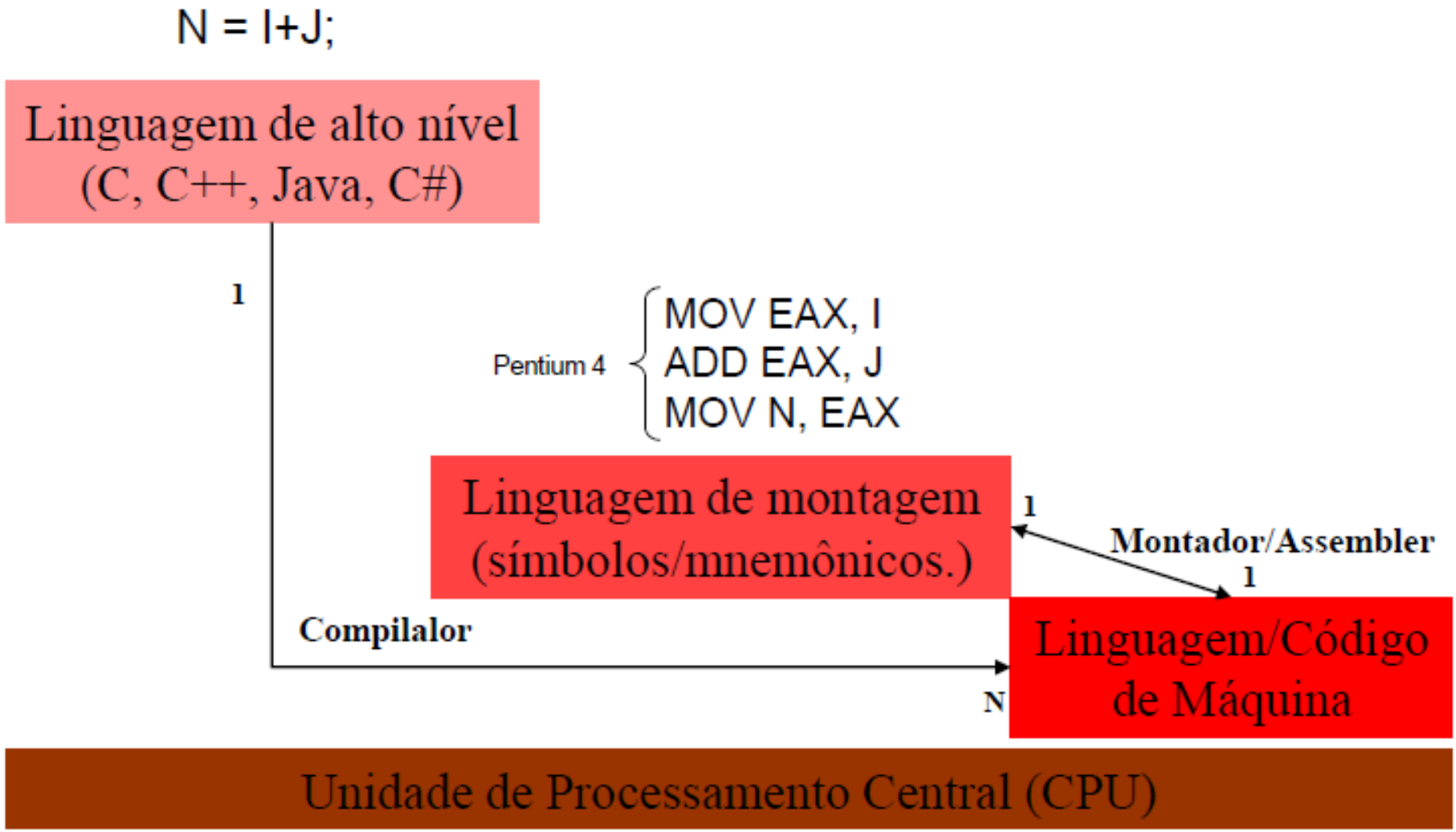
- **Tradutor**

Programas que convertem um programa usuário escrito em alguma linguagem (fonte) para uma outra linguagem (alvo).

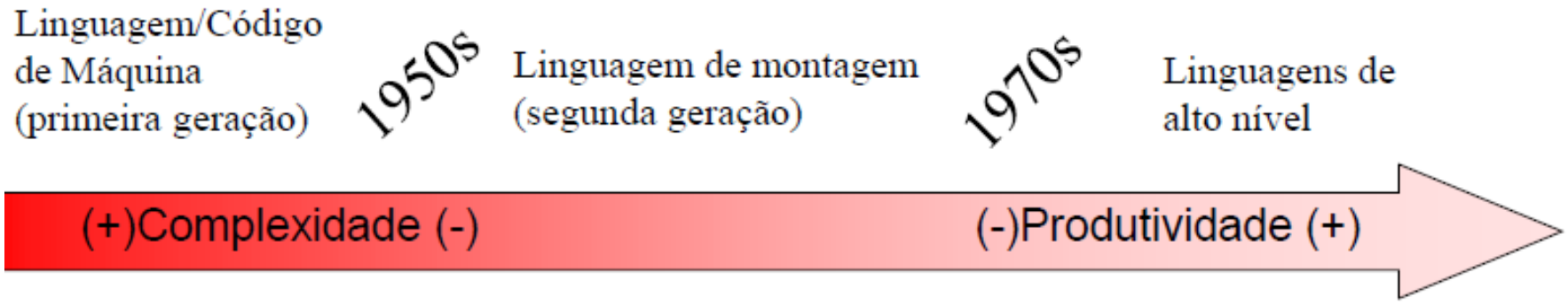
- **Montador/Assembler**

Um tradutor onde a linguagem fonte é a linguagem de montagem e a linguagem alvo é a linguagem de máquina.

Conceitos básicos (resumo)



História



Por que usar linguagem de montagem?

Código menor e mais rápido

- Cartão inteligente
- Drivers

- Tratadores de interrupções de baixo nível em um S.O.
- Controladores em sistemas embutidos de tempo real

Arquitetura

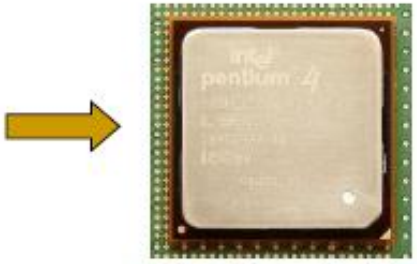
- **Arquitetura de computador:** própria linguagem de máquina e linguagem de montagem.
- Funções de alto nível diferente para cada arquitetura.

Motorola 680x0

```
MOVE.L I, D0
ADD.L J, DO
MOVE.L N, EAX
```

Pentium 4

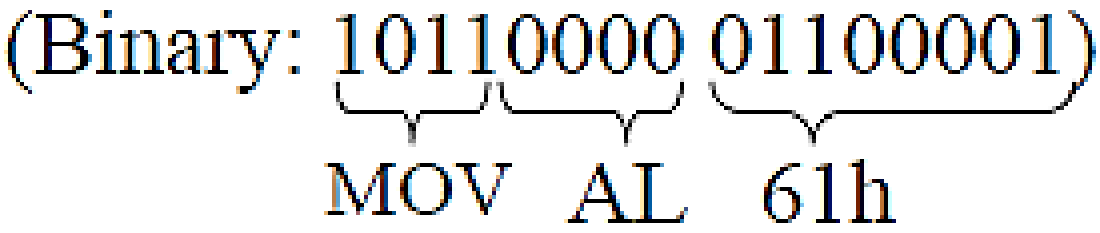
```
MOV EAX, I
ADD EAX, J
MOV N, EAX
```



Arquitetura	Quantidade Registradores	Bits	Introduzido (Ano)
6502	3	8	1975
x86	16	16-32	1978
ARM	16	32	1983
IBM/360	16	32	1964
Z/Architecture	16	32-64	2000
UltraSPARC	32	64	1995
Alpha	32	64	1992
POWER	32	32-64	1992
x86-64	64	64	2000
Itanium	128	64	2001

Elementos de uma instrução

- Internamente, cada instrução é representada como uma sequência de bits dividida em campos, correspondentes aos elementos de uma instrução.



Elementos de uma instrução

- Cada instrução deve conter toda a informação necessária para que a CPU possa executá-la, ou seja:
  - **Código da operação:** especifica a operação a ser efetuada.
  - **Referência ao operando-fonte:** os dados envolvidos na operação devem estar referenciados na instrução.
  - **Referência ao operando-destino:** deve-se fornecer informações que possibilite o armazenamento do resultado gerado pela instrução.

Código da operação	Referência ao operando-fonte	Referência ao operando-destino

Representação de uma instrução

- Dificuldade do programador lidar com representações binárias de instruções de máquina.
- Uso de uma representação simbólica:
  - Os códigos de operação são representados por abreviações, chamadas mnemônicos.
    - ADD (adição)
    - SUB (subtração)
    - MPY (multiplicação)
  - Os operandos também são representados de maneira simbólica.
    - ADD AX, BX

Programação em Assembly

Por que não C ou outra linguagem de alto nível?

- C é mais simples, mais próxima da linguagem natural (inglês).

- C é portável, ou seja, um mesmo programa pode rodar em sistemas operacionais diferentes, usando processadores PowerPC ou Intel, por exemplo.

Programação em Assembly

Por que programar em Assembly?

- O código em Assembly pode ser mais rápido e menor do que o código gerado por compiladores.
- Assembly permite o acesso direto a recursos do hardware, o que pode ser difícil em linguagens de alto nível.
- Programar em Assembly permite que se ganhe um conhecimento profundo de como os computadores funcionam.

Programação em Assembly

Conclusão

- Saber programar em Assembly é muito útil mesmo que nunca se programe diretamente nele.

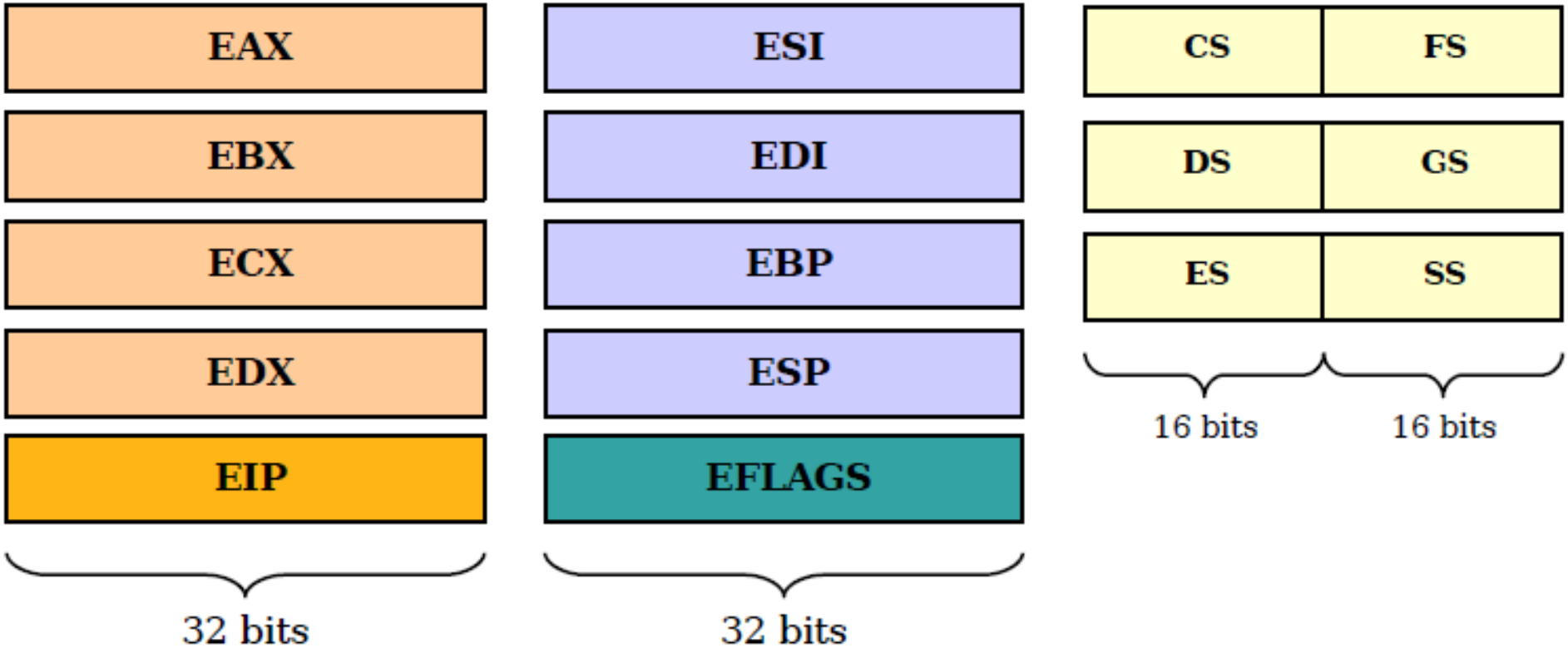
O Assembly do 80386

- Primeiro processador da Intel de 32 bits com recursos modernos:
  - Modo protegido de memória (nas versões antigas, como o 8086 havia o “modo real”, onde cada programa poderia bagunçar livremente a memória de algum outro).
  - Todos os sistemas operacionais modernos operam rodando sobre o modo protegido.
  - Multitarefa.

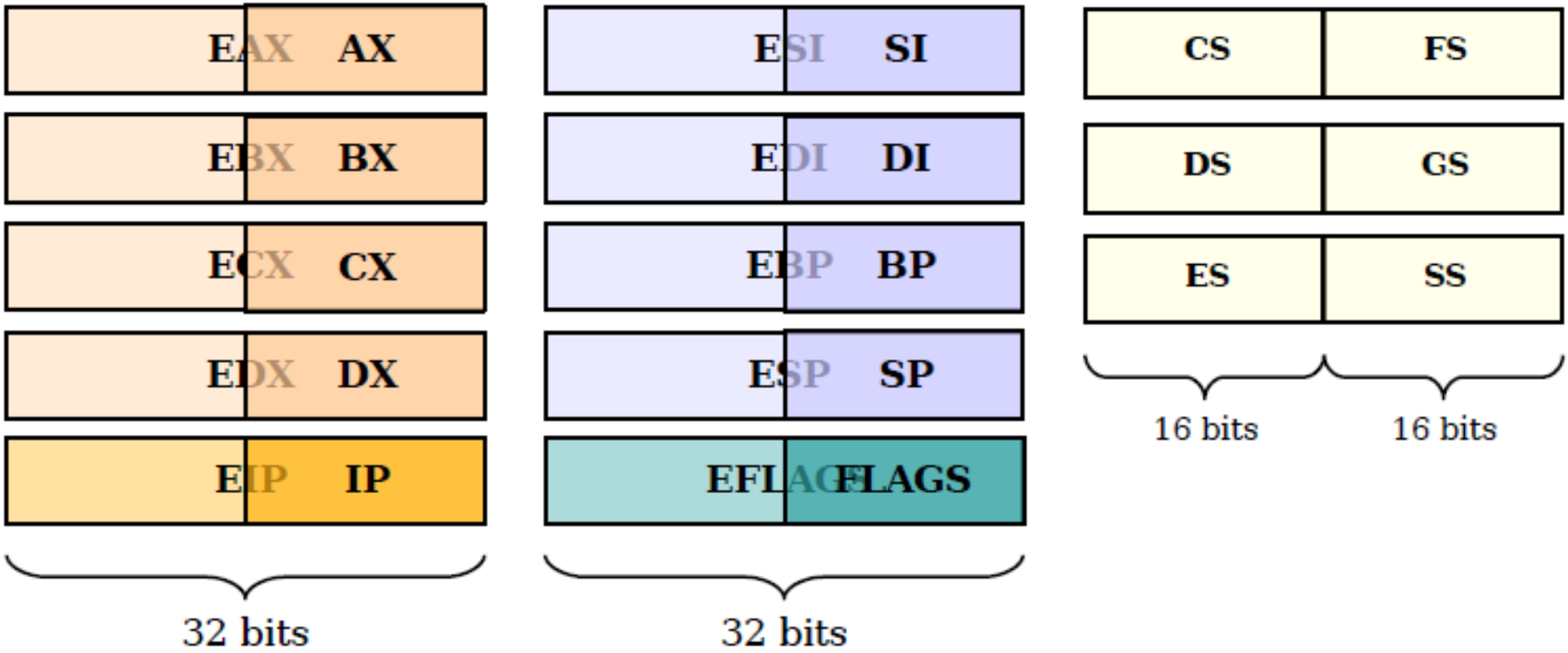
O Assembly do 80386

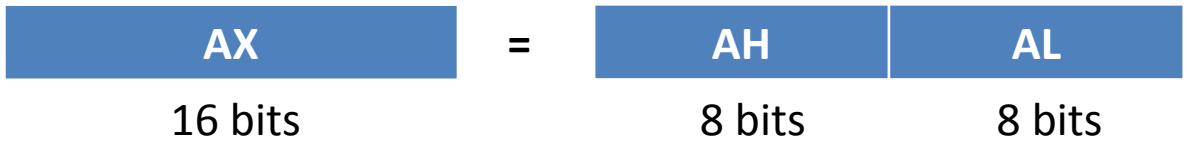
- Novos chips da Intel mantêm compatibilidade: Pentium, Core2Duo, Atom etc.
- Existem diversas ferramentas que facilitam o desenvolvimento desse tipo de código.
  - NASM (Netwide Assembler)
  - TASM (Turbo Assembler)
  - MASM (Macro Assembler)

Registradores do 80386



Registradores pré-80386





Também vale para BX, CX e DX

Registradores do 80386

- **Registradores de propósito geral – uso típico**
  - **EAX:** Registrador acumulador, usado para endereçar E/S, aritmética etc.
  - **EBX:** Registrador base, usado como ponteiro para acesso à memória e interrupções.
  - **ECX:** Registrador contador, usado como contador em laços e interrupções.
  - **EDX:** Registrador de dados, usado para endereçar E/S, aritmética e interrupções.

Registradores do 80386

- **Registradores de endereço**
  - **EIP:** Ponteiro de índice, guarda um índice indicando a próxima instrução a ser executada.
  - **EBP:** Endereço da base da pilha.
  - **ESP:** Endereço do topo da pilha.
  - **EDI:** Índice do destino na operação de cópia de strings.
  - **ESI:** Índice da fonte na operação de cópia de strings.

Registradores do 80386

- **Registrador EFLAG**
  - Cada um de seus 32 bits controla ou exibe algum estado final ou intermediário de uma operação.
  - Exemplos:
    - *Zero flag* (6): indica se o resultado de uma operação foi zero.
    - *Direction flag* (10): usado no processamento de strings, indica quando o processamento deve ser feito do início para fim ou vice-versa.
    - *Overflow flag* (11): usado por operações aritméticas que podem gerar estouro de pilha (overflow).

Registradores do 80386

- **Registrador EFLAG**

Intel x86 FLAGS register			
Bit #	Abbreviation	Description	Category[1]
FLAGS			
0	CF	Carry flag	S
1	1	Reserved	
2	PF	Parity flag	S
3	0	Reserved	
4	AF	Adjust flag	S
5	0	Reserved	
6	ZF	Zero flag	S
7	SF	Sign flag	S
8	TF	Trap flag (single step)	X
9	IF	Interrupt enable flag	X
10	DF	Direction flag	C
11	OF	Overflow flag	S
12, 13	IOPL	I/O privilege level (286+ only)	X
14	NT	Nested task flag (286+ only)	X
15	0	Reserved	
EFLAGS			
16	RF	Resume flag (386+ only)	X
17	VM	Virtual 8086 mode flag (386+ only)	X
18	AC	Alignment check (486SX+ only)	X
19	VIF	Virtual interrupt flag (Pentium+)	X
20	VIP	Virtual interrupt pending (Pentium+)	X
21	ID	Able to use CPUID instruction (Pentium+)	X
22	0	Reserved	
23	0	Reserved	
24	0	Reserved	
25	0	Reserved	
26	0	Reserved	
27	0	Reserved	
28	0	Reserved	
29	0	Reserved	
30	0	Reserved	
31	0	Reserved	
RFLAGS			
32-63	0	Reserved	

1. ^ S: Status flag
   C: Control flag
   X: System flag

Palavra - *word*

- Apesar do 80386 ser de 32 bits, o tamanho da sua palavra é 16 bits.

unidade de memória	tam. em bytes
word	2 bytes
double word	4 bytes
quad word	8 bytes
paragraph	16 bytes

Tipos de operandos

- **Registrador:** o operando refere-se diretamente ao conteúdo de um registrador da CPU.
- **Memória:** refere-se a um dado em memória.
- **Imediato:** valores fixos expressos diretamente na instrução.
- **Implicado:** valor não mostrado diretamente. Ex: operação de incremento.

Instruções básicas

mov *dest*, *src*

- copia em *dest* o conteúdo de *src*.

```
mov eax, 3;   grava 3 no registrador eax
mov ebx, eax;   grava o conteúdo de eax em ebx
```

add

- adiciona inteiros.

```
add eax, 4;   eax = eax + 4
add ebx, eax;   ebx = ebx + eax
```

sub : mesmo formato de add

inc : inc eax; *eax++*

dec : dec eax; *eax--*

Programa básico (MASM)

Comentários

Área de dados

Área de instruções

Inclusão de funções externas

main.asm

```
1  TITLE MASM Template                    (main.asm)
2
3  ; Description:
4  ;
5  ; Revision date:
6
7  INCLUDE Irvine32.inc
8  .data
9  myMessage BYTE "MASM program example",0dh,0ah,0
10
11 .code
12 main PROC
13     call Clrscr
14
15     mov  edx,OFFSET myMessage
16     call WriteString
17
18     exit
19 main ENDP
20
21 END main
```

Prática de laboratório

- Acessar o site kipirvine.com/asm e clicar no link “Getting started with MASM and Visual Studio 2010”.
- Seguir o tutorial para executar o primeiro programa em Assembly.
- Para mais detalhes sobre a programação em Assembly utilizando o MASM, procurar pelo livro de Kip Irvine, “Assembly Language for x86 Processors (6th edition)”.
- Para mais detalhes sobre a programação em Assembly utilizando o NASM, procurar pelo livro de Paul Carter, “PC Assembly Language”.

Exercícios

- Faça um programa para carregar dados em EAX e EBX e subtraí-los, deixando o resultado em EAX.
- Faça um programa para carregar dados em EAX e EBX e somá-los, deixando o resultado em EBX.
- Faça um programa para carregar dados quaisquer nos quatro registradores de uso geral e exiba o conteúdo deles na tela.
- Faça um programa para retornar se um número é par ou ímpar. Em um primeiro momento, armazene esse número direto em AX. Em um próximo passo, entre com esse número pelo teclado.