Sistemas de Computação

Introdução a programação *Assembly*

Haroldo Gambini Santos

Universidade Federal de Ouro Preto - UFOP

5 de novembro de 2009

Assembly

Porque não C (ou outra linguagem de alto nível)?

- C é mais simples
  - Mais próximo da linguagem natural (inglês, no caso)
- C é portável
  - Um mesmo programa pode rodar nos SOs Linux ou Windows, usando processadores Power PC ou Intel

Assembly

Porque programar em *Assembly*

- Código em *Assembly* pode ser mais rápido e menor do que 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

Assembly

Porque programar em *Assembly*

- Código em *Assembly* pode ser mais rápido e menor do que 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

Conclusão

Saber *Assembly* é muito útil mesmo que nunca se programe diretamente nele!

O *Assembly* do 80386

Características

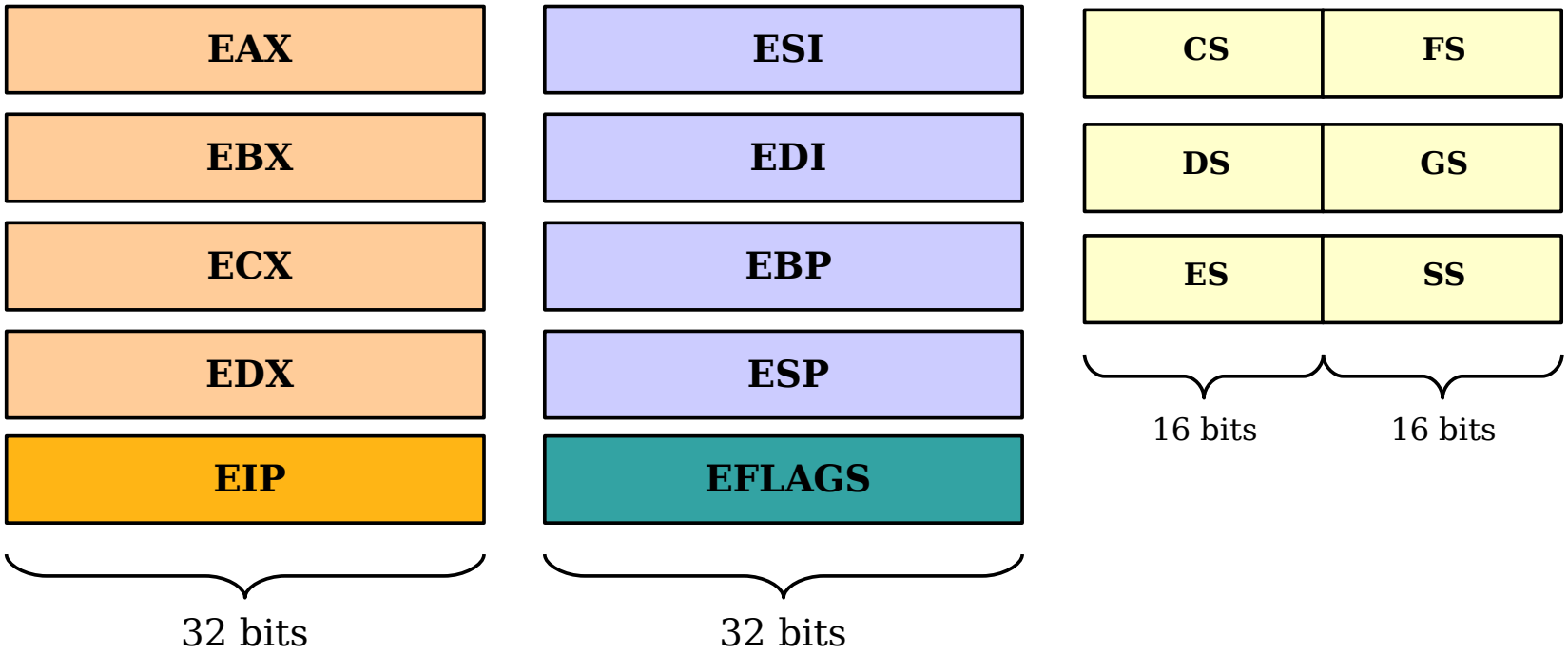
- Primeiro processador da Intel de 32 bits com recursos "modernos":
  - Modo protegido de memória (nas versões antigas, como o 8086 havia o "modo real", onde cada programa poderia bagunçar livremente a memória de algum outro)
  - Todos os sistemas operacionais modernos operam rodando sobre o modo protegido
  - Multitarefa
- Novos chips da Intel mantém compatibilidade: Pentium, Core2Duo, Atom, ...
- Existem diversas ferramentas livres na Internet que facilitam o desenvolvimento desse tipo de código

Assembly com o NASM

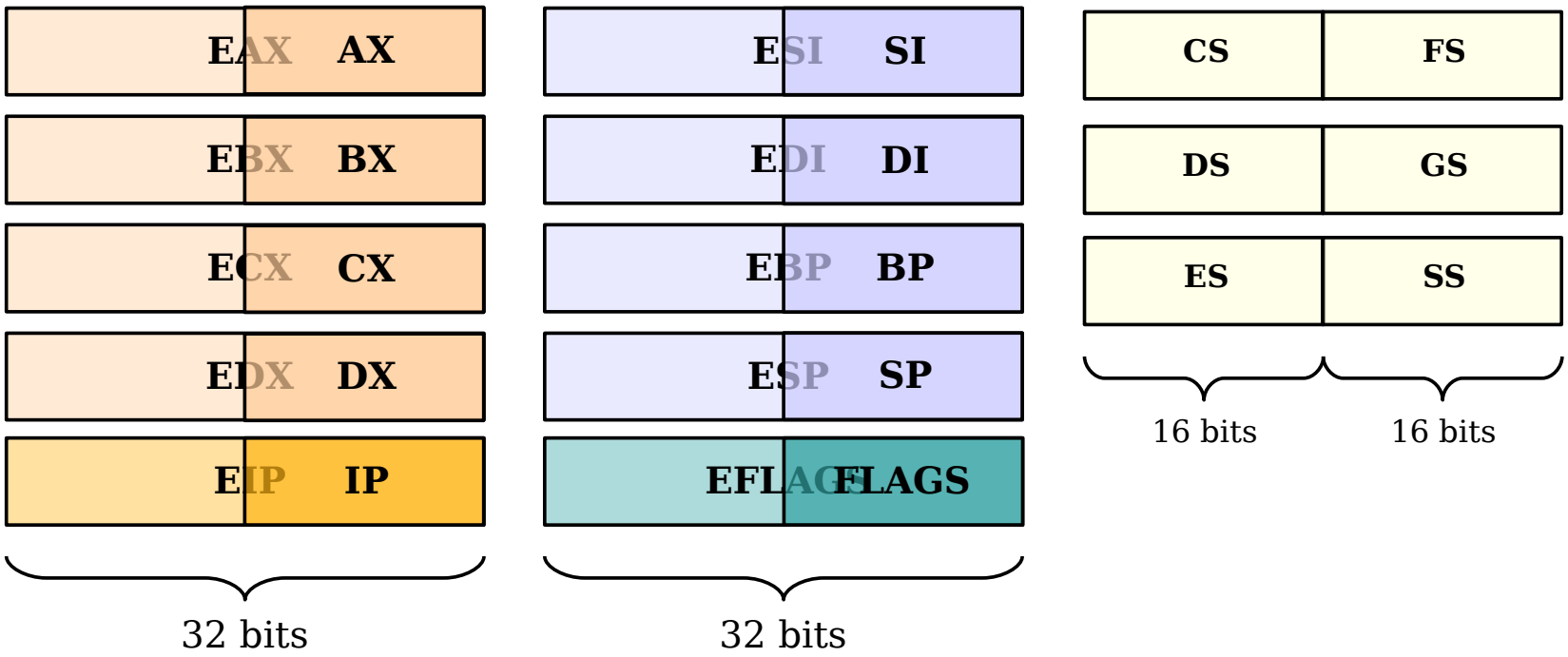
Netwide Assembler: NASM

- Software livre
- Portável: roda em vários sistemas operacionais Windows e Linux
- Simples de usar e com suporte a macros

Registradores 80386



Registradores pré-80386



Registradores 80386

Registradores de Propósito Geral - Uso Típico

EAX : Registrador acumulador. Usado para endereçar E/S, aritimética, etc.

EBX : Registrador base. Usado como ponteiro para acesso à memória, interrupções.

ECX : Registrador contador. Usado como contador em laços, interrupções.

EDX : Registrador de dados. Usado para endereçar E/S, aritmética, interrupções.

Registradores 80386

Registradores de Endereço

EIP Ponteiro de índice: guarda um índice indicando a próxima instrução a ser executada.

EBP Endereço base da pilha.

ESP Endereço do topo da pilha.

EDI Índice do destino - na operação de cópia de cadeias de caracteres.

ESI Índice da fonte - na operação de cópia de cadeias de caracteres.

Registradores 80386

Registador `EFLAG`

Cada um de seus 32 bits controla ou exibe algum estado final/ intermediário de uma operação.
Exemplo:

6 *Zero Flag:* indica se o resultado de uma operação foi zero.

10 *Direction Flag:* usado no processamento de strings, indica quando o processamento deve ser feito do início para o fim ou o contrário.

11 *Overflow Flag:* usado por operações aritméticas que podem gerar *overflow*.

Palavra - *word*

Palavra

Registradores de 32 bits, mas palavra no Assembly do 80386 tem 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 - posição constante ou informada em um registrador;
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 *src* conteúdo de *src*. Ex.:

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

`add`

adiciona inteiros. Ex.:

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

`sub` : mesmo formato de `add`

`inc` : ex.: `inc eax;` *eax++*

`dec` : ex.: `dec eax;` *eax--*

Diretivas

%define

define um valor constante a ser usado no programa:

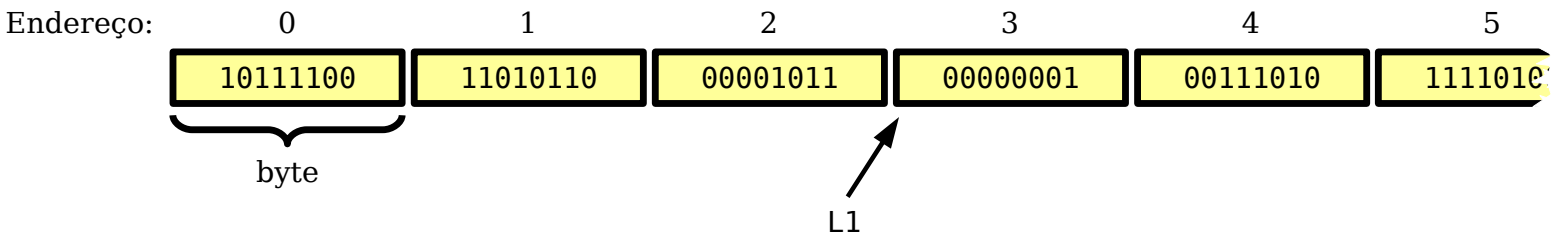
```
%define SIZE 100
    mov eax SIZE
```

Diretivas de Dados

Reservam de espaços de memória. Os espaços são marcados com rótulos (*labels*):

```
L1 db 0 ; byte rotulado L1 com valor inicial 0
L2 dw 1000 ; palavra rotulada L2 com valor inicial 1000
L3 db 110101b ; byte inicializado para 110101 em binário
L4 db 12h ; byte inicializado to 12 em hexadecimal
L5 db 17o ; byte inicializado to 17 em octal
L6 dd 1A92h ; double word inicializado 1A92 em hexadecimal
L7 resb 1 ; 1 byte não inicializado
L8 db "A" ; byte inicializado o código da letra A (65)
L9 db 0, 1, 2, 3 ; define 4 bytes
L10 db "w", "o", "r", 'd', 0 ; uma string compatível com C
L11 db 'word', 0 ; mesmo que L10
```

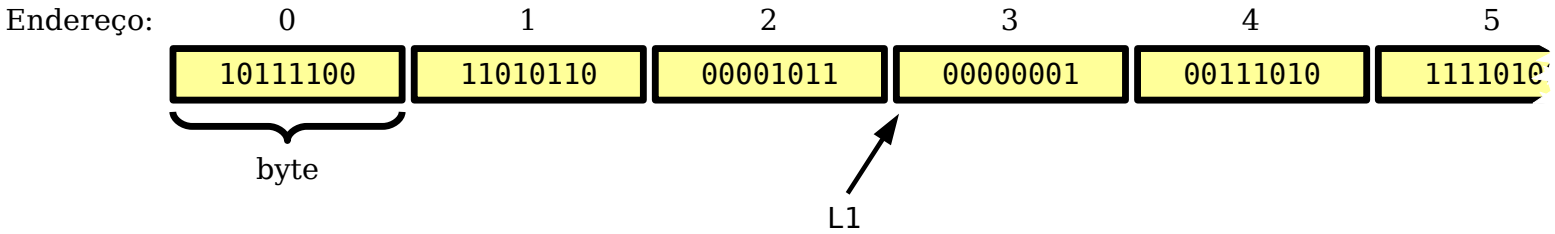
Rótulos (*labels*)



Rótulos

- Reservam espaço em algum lugar na memória e opcionalmente inicializam seu conteúdo
- No decorrer do programa o mesmo espaço de memória é facilmente acessado através de seu rótulo
- Exemplo:
  - `L1 db 1`
  - No caso, `L1` foi associado a célula de memória com endereço 3 e seu conteúdo inicializado com 1

Rótulos (*labels*)



Usando rótulos

- Similar ao conceito de ponteiros em `C`
- Pode-se trabalhar com o *endereço* do rótulo ou com o seu *conteúdo*
- `mov eax, L1`
  - Registrador eax recebe 3
- `mov eax, [L1]`
  - Registrador eax recebe 1

Rótulos de mais de um byte

- `L1 dd 35`
  - O tipo `dd` refere-se a um *double word* do 80386 (32 bits), bom para armazenar inteiros
- `mov [L1], 15`
  - Não funciona ! O assembler não sabe se deve trabalhar com byte, palavra ou palavra dupla, por exemplo
- Correto:

```
mov dword [L1], 15
```

Tipos de Dados

`res*` e `d*` - sufixos:

- `B` byte
- `W` word
- `D` double word
- `Q` quad word
- `T` ten bytes

Estrutura

```
segment .data
prompt1 db "Digite um numero: ", 0
prompt1 db "Digite outro numero: ", 0
msg1 db "A soma é: ", 0

segment .bss
x resd 1
y resd 1

segment .text
asm_main:
    enter 0,0
    pusha
    mov eax, prompt1
    call print_string
    call read_int
    mov [x], eax
```

Comandos de Entrada e Saída

Usualmente dependente do sistema operacional.
Solução mais portável:

`asm_io.inc`

Biblioteca com comandos simplificados:
Linux:
http://www.drpaulcarter.com/pcasm/linux-ex.zip
Windows:
http://www.drpaulcarter.com/pcasm/ms-ex.zip

Imprimindo

```
%include "asm_io.inc"

segment .data
prompt1 db "Digite um numero: ", 0

segment .text
    ...
    mov eax, prompt1
    call print_string
    ...
```

Lendo um Inteiro

```
%include "asm_io.inc"

segment .bss
x resd 1

segment .text
    ...
    call read_int
    mov [x], eax
    ...
```

Desvios

Rótulos de Código

Marcam determinada posição no código. Ex.:

```
mov ebx, [y]
mov eax, [x]
call print_string
...
```

Instrução *Jump*: `JMP`

```
jmp rotulo_de_codigo
```

Faz com a próxima instrução executada seja aquela situada imediatamente após o rótulo "`rotulo_de_codigo`".

Repetição

Instrução `LOOP`

Formato: `LOOP` *posição_para_saltar*

Verifica o valor do registrador `ECX`, caso o mesmo seja diferente de zero pula para a parte do código rotulada em *posição_para_saltar*.

Repetição

Equivalente em C

```
for (int i=0 ; (i<10) ; i++ )
    printf("%d\n", i);
```

Assembly

```
%include "asm_io.inc"
segment .text
   ...
   mov [ECX] 10
inicio:
   mov eax, ecx
   call print_int
   loop inicio
   ...
```

Testes condicionais

Executando o "`if`" em assembly

Duas partes:

1. Executa uma comparação (instrução `CMP`)
   - Altera valor do registrador `EFLAGS`
2. Avalia resultado (instruções `JE`, `JNE`, ...)

Testes condicionais

Equivalente em C

```
if ( x==0 )
        comandoA;
   else
        comandoB;
```

Assembly

```
        ...
        mov eax, [x]
        cmp eax, 0
        je posicaoA
        jmp posicaoB
posicaoA:
        comandoA
        jmp depoisTeste
posicaoB:
        comandoB
        jmp depoisTeste
depoisTeste:
        ...
```

Desvios

Incondicional

Referências

PC Assembly Language:
http://www.drpaulcarter.com/pcasm/
(livro em PDF)

The Netwide Assembler:
http://www.nasm.us/

Writing a Useful Program with NASM:
http://leto.net/writing/nasm.php

GNU Compiler Collection:
http://gcc.gnu.org/

MingW: Minimalist GNU for Windows
http://www.mingw.org/