Introdução à Software Básico: Conceitos Básicos

Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade de Brasília

Sumário

Conceitos Básicos

Definição

Software de Sistema:

- Também conhecido como Software Básico, o Software de Sistema é o conjunto de programas necessário para operar e controlar o hardware de um computador, assim como para providenciar uma plataforma para poder rodar as aplicações.

Exemplos

Exemplos de Software de Sistema:

- O exemplo mais conhecido é o Sistema Operacional (OS). O OS administra todos os programas e hardware de um computador.
- A BIOS (*basic input/output system*) que testa e inicia os componentes de hardware do sistema. Chama o carregados (*bootloader*) do OS e proporciona uma camada abstrata para OS interagir com o hardware de I/O (*input/output*) como o teclado, mouse, impressora, etc.
- O *bootloader* que carrega o OS.
- Os *drivers* que controla um hardware especifico conectado ao computador, como um disco rígido, teclado, etc. O *driver* converte as instruções gerais de I/O do sistema operacional em mensagens que o dispositivo especifico possa entender.
- O *firmware* presente em computadores ou sistemas embarcados. Num sistema embarcado é o software que controla o sistema. A BIOS de um computador é um tipo de *firmware*.

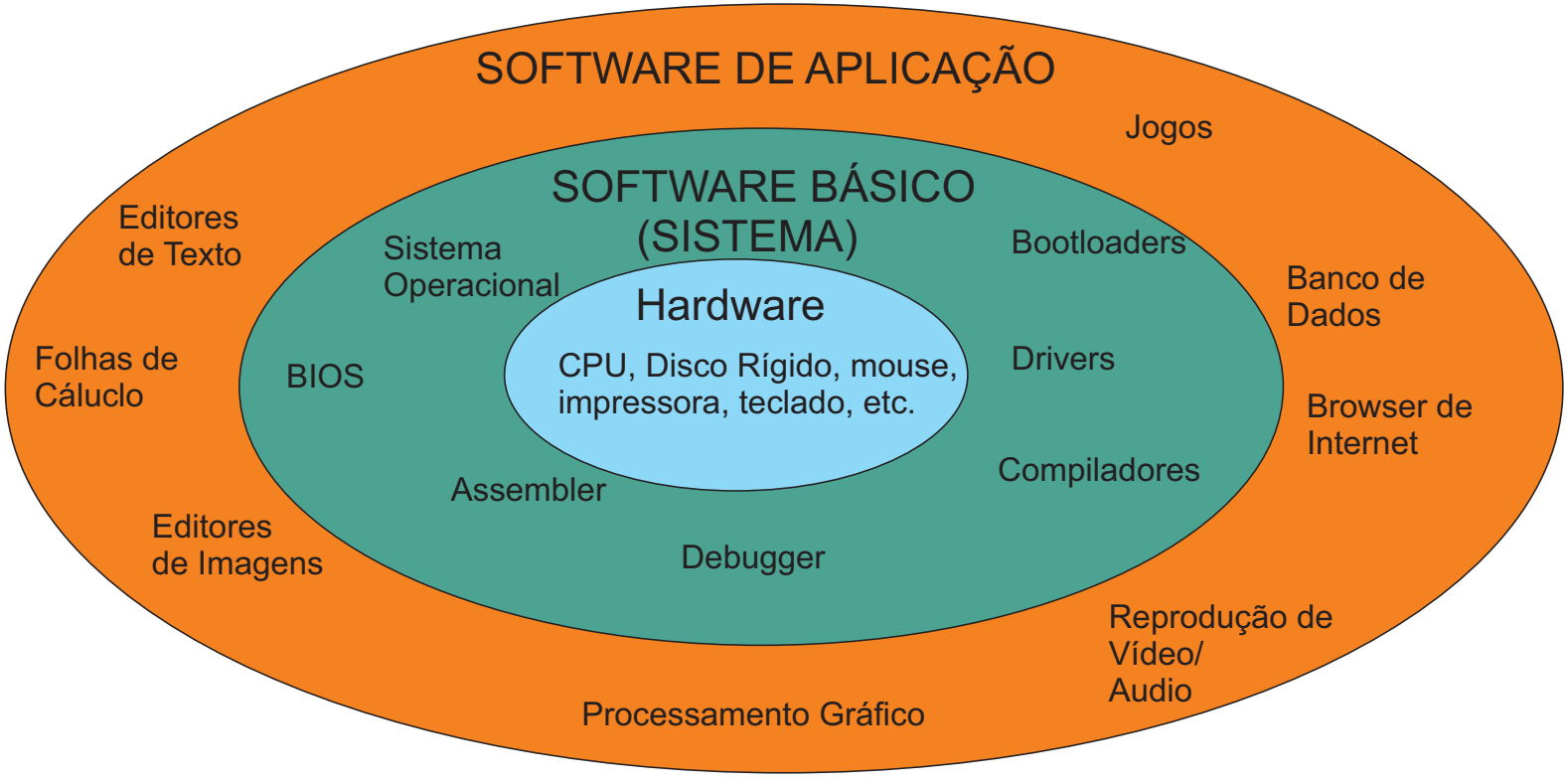

Figura: Hardware e Software de um sistema computacional

Exemplos

Outros Tipos de Software de Sistema:

- *Assembler* ou Montador que converte instruções em linguagem *assembly* para código máquina.
- *Loader* ou Carregador que carrega o programa a ser executado em memória.
- *Linker* ou Ligador quem combina dois ou mais arquivos objeto e proporciona as referências adequadas entre eles.
- Ferramentas de Desenvolvimento como Compiladores e Debuggers.
- Utilitários como desfragmentador de disco e *system restore*.

Nota

Alguns autores não consideram Utilitários como Software de Sistema.

Na Disciplina

Na disciplina vamos estudas com detalhes os Montadores, Carregadores e Ligadores. Estudaremos também programação *assembly* e novos conceitos de programação em C.

Estrutura da Memória de um Computador

- A memória principal de um computador é organizada em endereços. Cada endereço possui o mesma capacidade de armazenamento, basicamente somente pode ser armazenado um único número em cada endereço.
- Os endereços na maioria das arquiteturas possuem 8 bits (1 byte).
- Além de dados, a arquitetura Von Neumman especifica que os programas ativos também devem ser carregados em memória.

Estrutura da Memória de um Computador

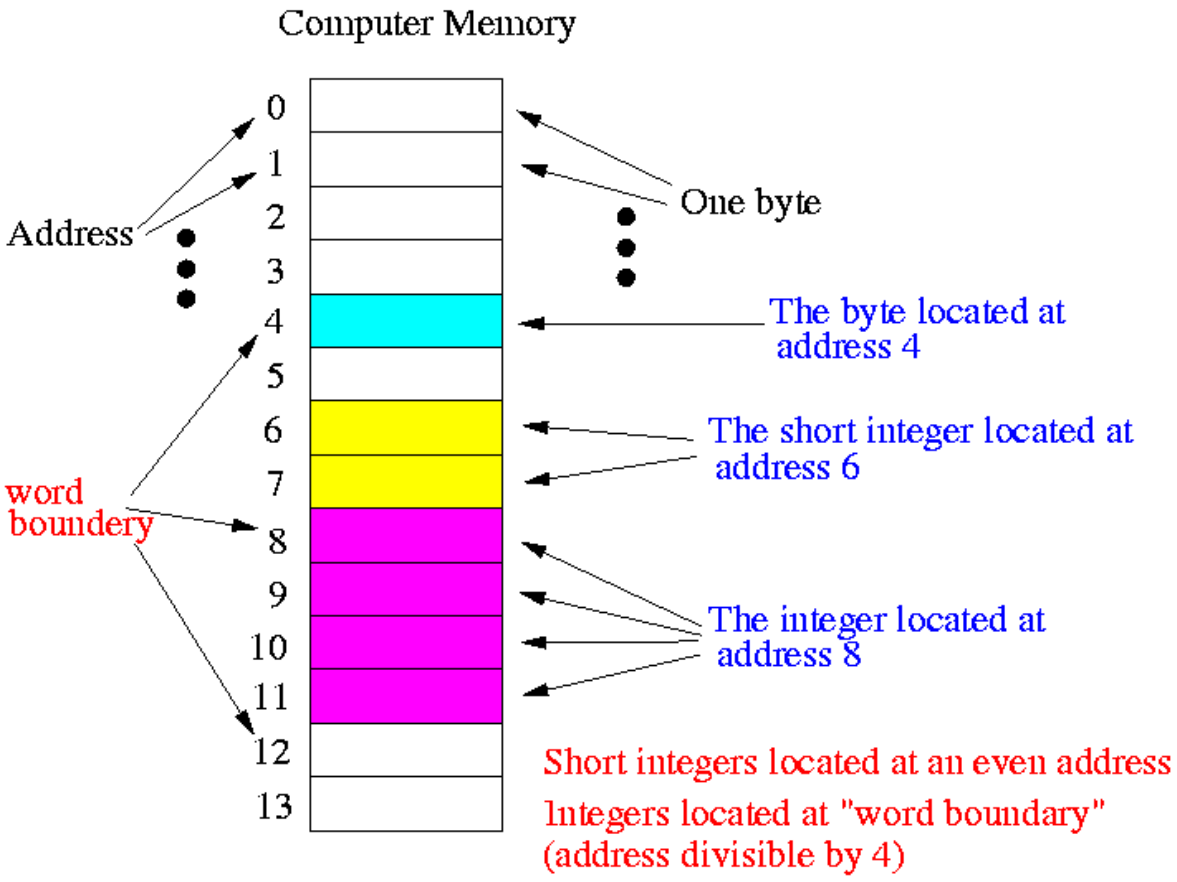

Figura: Exemplo da estrutra da memória principal de um computador de 32 bits.

CPU

A CPU lê uma instrução da memória principal por vez e a executa. Isto é conhecido como ciclo de execução. Para fazer isso precisa dos seguintes componentes:

- Contador de Programa: Indica ao CPU a posição em memória da próxima instrução.
- Decodificador de Instrução: Identifica o significado de cada instrução.
- Bus de Dados: É o canal pelo qual a memória e a CPU se comunicam, tanto para a transferência de instruções como de dados.
- Registradores: Existem 2 tipos de registradores. Os registradores de uso geral e os de uso especifico. Os registradores de uso geral, são onde a maioria de operações são realizadas (soma, multiplicação, etc.). Os de uso especifico dependem de cada arquitetura, e são acessados por comandos específicos.
- Unidade Lógica e Aritmética: E nesta unidade que as instruções matemáticas e lógicas são executadas, para isso a ALU faz uso dos registradores.

Código Máquina

- O código máquina é um conjunto de instruções executada diretamente pela CPU.
- Cada instrução realiza uma tarefa especifica como carregar um dado de memória para um registrador, pular o contador de programa, realizar uma operação na ALU, etc.
- Código Máquina é o único tipo de instrução que o Decodificador de Instrução da CPU entende.
- O Código máquina é expresso em números. É possível programar diretamente em código máquina, porém é extremadamente difícil e trabalhoso devido a que os códigos numéricos não são fáceis de serem interpretados por um ser humano.

Baixo Nível

- O código máquina é a programação mais baixo nível que existe.
- A linguagem montadora (*Assembly*) é uma representação da linguagem de máquina através de códigos mnemônicos.
- Cada instrução de linguagem (*Assembly*) possui uma única instrução correspondente de código máquina.
- Cada processador possui seu próprio conjunto de instruções (na disciplina vamos estudar o IA-32 (INTEL)).

Programação Alto Nível vs. Baixo Nível

Alto Nível

- As linguagens de alto nível são mais próximas da linguagem usada pelo ser humano.
- Cada instrução de uma linguagem de alto nível não indica apenas uma única instrução de código máquina, mas pode indicar um conjunto de instruções.
- As linguagens de alto nível possuem uma enorme flexibilidade e capacidade de abstração.
- As estruturas de alto nível facilitam a programação, e permitem que os programas sejam portáveis.
- Exemplo de linguagens de alto nível: C/C++, PASCAL, Java, Cobol, FORTRAN, etc.

Linguagem C

- A linguagem C foi desenvolvida por Dennis M. Ritchie na década de 70 (1969 - 1973)
- A origem da linguagem C esta relacionada ao desenvolvimento do OS Unix. O Unix foi inicialmente desenvolvido para o PDP-7, eventualmente foi incorporado ao PDP-11. Nesse momento o Unix era totalmente feito em linguagem *Assembly*. Foi decidido que para melhorar a portabilidade deveria ser rescrito em linguagem B. Porém, a linguagem B era uma linguagem de alto nível que não tinha muita pouca capacidade de interagir com o hardware. Logo, foi criada a linguagem C.
- ANSI C
  - Padrão da linguagem C
  - Versões de 89 e 90 são semelhantes
  - Revisão em 99 deu origem ao C99, atual padrão
  - O uso do C padrão facilita a portabilidade do programa

Programação Alto Nível vs. Baixo Nível

Por que estudar linguagem *Assembly*, dada a existência de linguagens de alto Nível?

- Para aprender como códigos de mais alto nível podem ser traduzidos a código máquina.
- Para aprender a dependência entre hardware/software de um computador.
- Para realizar programas mais eficientes. Programas que ocupem menos memória e possuam maior velocidade de execução.

Programação Alto Nível vs. Baixo Nível

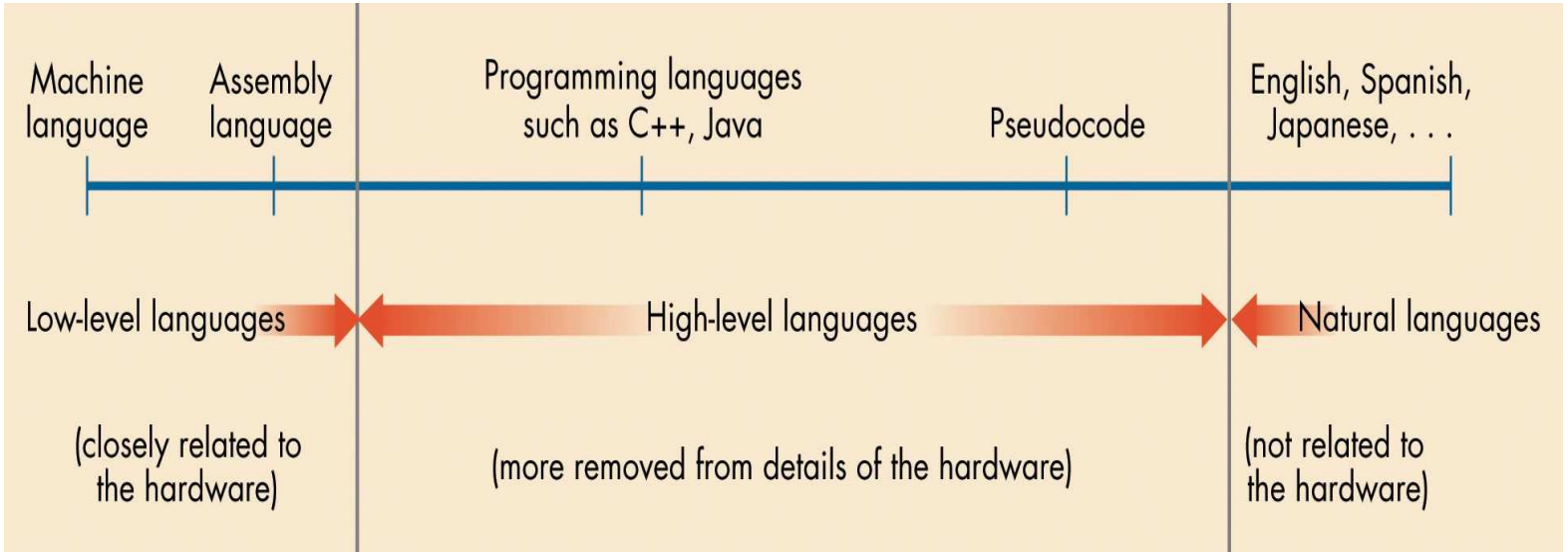

Figura: Alto Nível vs. Baixo Nível.

Definições

Compilador

- Programa responsável por traduzir uma linguagem de programação a outra. Um compilador pode por exemplo traduzir Linguagem C para *Assembly* ou diretamente para código máquina.

Montador

- É um tipo especifico de Compilador. É responsável por traduzir os mnemônicos de programa em linguagem *Assembly* para código máquina.

Figura: Compilador.

Estágio de Compilação/Montagem

Estágio 1

Pré-processamento

- Realizado por um programa chamado pré-processador.
- Elimina comentários e espaços em branco do código fonte.
- O código fonte sofre alterações, mas permane escrito na linguagem original.

Estágio 2

Compilação/Montagem

- Realizado pelo compilador/montador.
- Lê o código gerado pelo pré-processador e gera um código de saída, normalmente o código objeto (código máquina).
- Verifica erros de sintaxe e mostra avisos (*warnings*).
- Escreve o código de sáida em disco. No caso de um montador ou compilador para código máquina, nesta étapa é gerado o arquivo objeto (.o).
- Em caso de erros, o código de saída não é gerado.

Estágio 3

Ligação (Linking)

- Combina o código objeto com outros códigos objetos para então gerar o arquivo executável.
- Outros códigos objetos:
  - *Run-Time Library*,
  - Outras bibliotecas ou outros códigos objetos criados.
- Não havendo erros, o código executável é gravado no disco.

Estágio de Compilação/Montagem

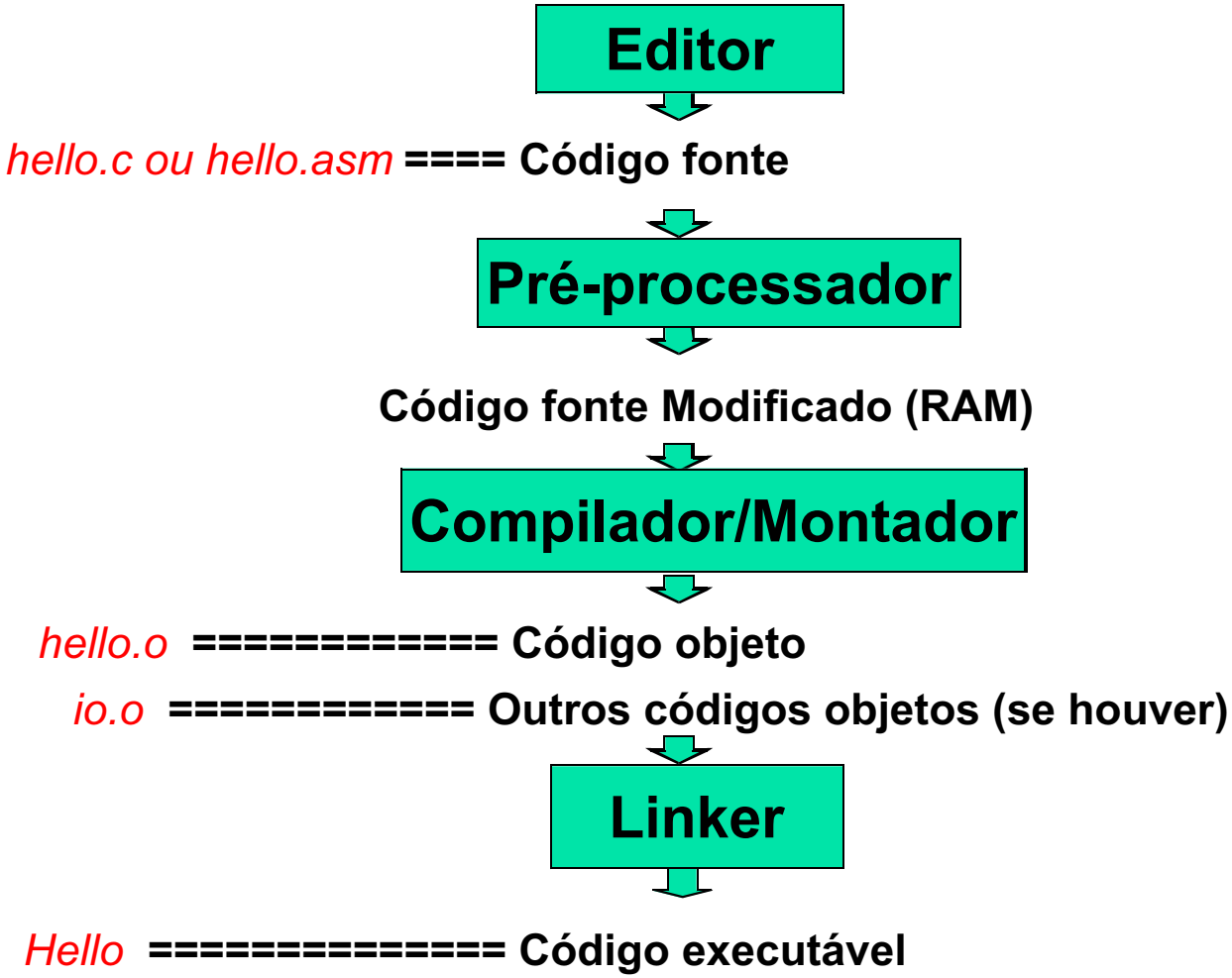

Figura: Estágio de Compilação/Montagem

Escrevendo Programa em C

- Editores de texto são usados para criar/modificar o código fonte
  - Unix: Vi, Emacs, xemacs, pico, etc
  - Windows: Notepad, word, edit, etc.
  - Também podem ser usados ambientes de desenvolvimento.
- Uma vez criado o código fonte, o compilador gera o código objeto, e ligador gera o código executável
- Nós utilizaremos o compilador **gcc**. Disponível no ambiente Linux, Windows, e Unix. GCC realiza as funções de compilador e ligador.

Escrevendo Programa em C

- Compilando um programa chamado *hello.c* em gcc:
  gcc -ansi -Wall hello.c -o hello
  - *hello.c* é o código fonte em C
  - *-ansi*, opção que informa utilizar o padrão ANSI C
  - *-Wall*, opção para que o **compilador** identifique todas as alertas (*warnings*). Um programa deve possuir ZERO *warnings*
  - *-o hello*, opção que informa qual o nome do arquivo executável a ser gerado.
- Caso não existam erros na compilação, o código executável será criado
- Para executar o programa, basta digitar ./hello no prompt do Linux

Escrevendo Programa em *Assembly*

- Editores de texto são usados para criar/modificar o código fonte
  - Unix: Vi, Emacs, xemacs, pico, etc
  - Windows: Notepad, word, edit, etc.
  - Alguns ambientes de desenvolvimento permitem programar em *Assembly* INTEL.
- Uma vez criado o código fonte, o **montador** gera o código objeto, e o ligador gera o código executável
- Nós utilizaremos o compilador **nasm** e o ligador **ld**. Disponível no ambiente Linux (existem versões para Windows).

Desenvolvendo Programas

Escrevendo Programa em *Assembly*

- Montado um programa chamado *hello.asm* em nasm:
  nasm -f elf -o hello.o hello.asm
  - *-f elf*, indica que o formato do arquivo objeto deve ser elf (*executable and linkable format*) 32 bits. Para compilar em 64 bits usar *elf*64.
  - *-hello.o*, nome do arquivo objeto de saída.
  - *-hello.asm*, nome do arquivo de entrada
- Gerando o executável com o ligador ld:
  ld -o hello hello.o
- Executando o programa em Linux: Basta digitar *./hello* no terminal.

Nota 1

Se o seu OS Linux é nativo de 64 bits, então para fazer a ligação deve utilizar:
ld -m elf_i386 -o hello hello.o

Nota 2

Também existe o montador GAS (comando **as**). Em formato GAS o código *Assembly* é ligeramente diferente que o NASM.

Estrutura de um programa NASM

```
;brief title of program          file name
;               Objectives:
;               Inputs:
;               Outputs:
section .data
 (initialized data go here)

section .bss
 (uninitialized data go here)

section .text
 (Directives and Instructions go here)
```

Figura: Anatomia de um programa Assembly Intel 32 bits, em formato NASM

Estrutura de um programa NASM

Seções

- A seção de texto (.section .text) é obrigatória em qualquer programa *Assembly*, já que nessa seção ficam os mnemônicos das instruções do programa.
- As seções de dados (section .data e section .bss) são opcionais e declaram elementos que são as variáveis do programa:
  - A seção de dados (section .data) declara elementos de dados cujo valor inicial é declarado no programa.
  - Já a seção de dados não inicializados (section .bss) declara elementos de dados inicializados com zero (ou para alguns montadores, não inicializados).

Estrutura de um programa NASM

Comentários

- Comentários em nasm: "';"' no início da linha

Formato

- Formato de uma declaração (linha de programa):
  [Símbolo] [Operação] [Operando(s)] [;Comentário]
- Símbolos podem ser:
  - Endereço de um ponto no código
  - Nome de uma posição de memória
  - Nome de constante

Operação: instruções ou diretivas

- Instruções são para o processador
  - Dependem do hardware alvo
- Diretivas são comandos para o montador
  - Não dependem do hardware
  - São palavras chave do montador, e não do processador. Exemplos:
    - %include : inclui o código fonte de um arquivo externo
    - section ou segment : agrega o código em seções separadas e nomeadas
    - db: reserva bytes de memória, com valor de incialização
    - resb: reserva um byte de memória sem inicialização

Algoritmo 1 Exit

```
1:  ;PURPOSE: Simple program that exits and returns a status code back to the
2:  ;Linux kernel.
3:  ;INPUT: none.
4:  ;OUTPUT: returns a status code. This can be viewed by typing echo $? after
5:  ;running the program.
6:  ;VARIABLES:
7:  ; eax holds the system call number
8:  ; ebx holds the return status
9:  section .data
10: section .text
11: global _start
12: _start:
13: mov eax,1   ;this is the linux system call for exiting a program.
14: mov ebx,0   ;this is the status number we will return to the operating system.
15: int 0x80    ;this wakes up the kernel to run the exit command.
```

Comentários

- Sempre incluir nos comentários o objetivo do programa, em resumo do processo envolvido e dados de entrada e saída

section .text

- A primeira diretiva é *global _start*. Isto indica ao montador que *_start* é um símbolo global. Durante a montagem ou ligação o símbolo será substituído por um outro valor.
- A linha seguinte é a definição do símbolo *_start*. Neste caso o símbolo *_start* é uma etiqueta (*label*). Uma etiqueta indica ao montador que o valor do símbolo deve ser substituído pelo endereço de memória da instrução seguinte à definição do símbolo. A etiqueta *_start* é obrigatória, todos as próximas definições de etiquetas são relativas à etiqueta *_start*.

section .text

- O primeira instrução é *mov eax*,1. Este comando transfere o valor 1 para o registrador de uso geral chamado *eax*.
- O número 1 é o código da chamada ao sistema *exit*. Uma chamada ao sistema é quando um programa chama diretamente uma instrução do kernel. Em *Assembly* o código que indica qual chamada ao sistema estamos fazendo deve estar contido no registrador *eax*.
- A chamada ao sistema *exit*, simplesmente sai do programa em execução e retorna o valor contido em *ebx* ao sistema operacional.
- Convencionalmente se o programa rodou sem erros deve ser retornado o valor 0. Por isso a instrução *mov ebx*,0.
- A última instrução *int* é uma interrupção. O número hexadecimal ao lado de *int* indica o tipo de interrupção que será feita.
- 0x80 significa que a interrupção é uma chamada ao sistema.

Próxima Aula

Tradução, Compilação e Interpretação.