UFAM

Universidade Federal do Amazonas
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Departamento de Ciência da Computação

Organização de Computadores (IEC482) – 2007/02

1º. Laboratório

Aluno: _____

Professor: Leandro Galvão

1. Objetivo

O objetivo deste laboratório é fazer com que o aluno se familiarize com o ciclo de desenvolvimento de programas escritos em linguagem assembly IA-32.

2. Ciclo de desenvolvimento

O ciclo de desenvolvimento de programas em linguagem de montagem geralmente consiste da edição programa fonte (atravé de um editor ASCII), montagem do programa fonte (através de um montador) e linkedição, gerando o programa executável. A Figura 1 mostra esse fluxo de desenvolvimento.

Um depurador também pode ser usado durante a fase de testes do programa, possibilitando a verificação, instrução por instrução, de seu comportamento.

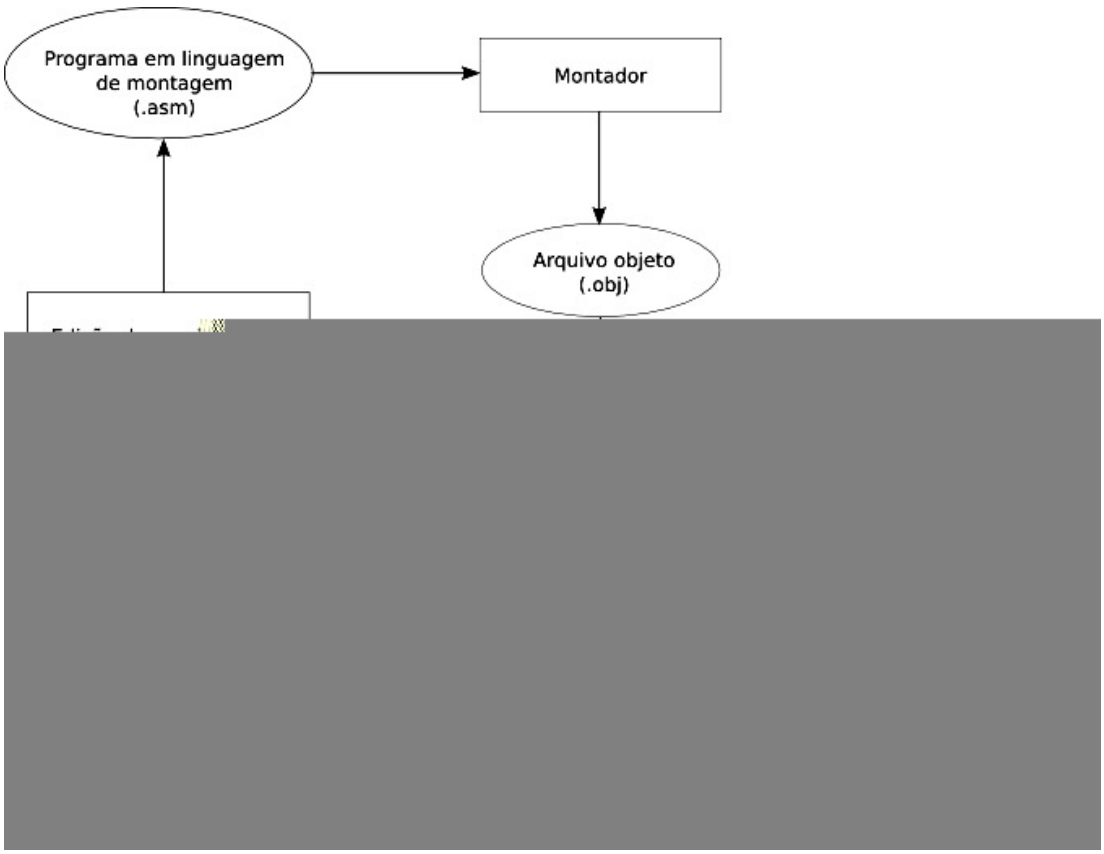

Figura 1 *- Ciclo de desenvolvimento de um programa em Assembly.*

3. Ambientes de trabalho

Você pode utilizar tanto o Windows como o Linux para realizar as tarefas do laboratório de Organização de Computadores (IEC-482). Entretanto, os programas não são portáveis de um sistema operacional para o outro.

3.1 Ambiente Linux

Para configurar um ambiente completo de programação assembly IA-32 no Linux, recomendamos a instalação dos seguintes programas:

- Gedit – editor de texto (utilize um destacador de sintaxe para facilitar a tarefa de escrever programas assembly: www.dcc.ufam.edu.br/~dcc_oc/asm.lang)
- NASM - Netwide assembler (montador)
- LD – GNU linker
- GDB – GNU Debugger
- DDD - Data Display Debugger

3.2 Ambiente Windows

Uma sugestão de um bom editor de texto para escrever códigos de programas assembly IA-32 no Windows é o ConTEXT (www.inet.hr/~edkirin/ConTEXTsetup.exe). Sua instalação padrão oferece o recurso de realce de sintaxe para programas scritos em assembly IA-32, o que torna mais agradável a tarefa de programar.

Uma segunda opção de editor de texto é TextPad (www.textpad.com). Uma versão rudimentar de arquivo de definição de sintaxe assembly IA-32 para o TextPad pode ser encontrada no site da disciplina (www.dcc.ufam.edu.br/~dcc_oc/asm.syn). Você está convidado a aprimorá-la. Esse arquivo deverá ser copiado no diretório **...\TextPad 4\system**. É necessário ainda ir ao menu Configurar → Nova classe de documento e criar uma classe de documento do tipo asm.

No arquivo **tools_dos.zip**, disponível em www.dcc.ufam.edu.br/~dcc_oc/tools_dos.zip, estão as seguintes ferramentas:

- NASM.EXE - Netwide assembler (montador)
- TLINK.EXE - Turbo linker (ligador)
- TD.EXE - Turbo Debugger (depurador)

4. Atividade 1 – Hello World!

Nesta atividade, você vai escrever em assembly um programa que imprime a mensagem “Hello World” na tela do computador. Por enquanto, não se concetre em entender o que as instruções do programa fazem, mas sim em utilizar as ferramentas de desenvolvimento.

4.1 Passo 1 – Edição

Abra o editor Gedit (Linux) ou ConTEXT (Windows), ou algum outro de sua preferência, e escreva o programa apresentado na *Figura 2* (Linux) ou *Figura 3* (Windows), escritos em assembly IA-32 para NASM.

```
%include "io.mac"

SEGMENT stack stack   ; segmento da pilha
resb 0x100

.DATA                 ; segmento de dados
HelloMessage     db   "Hello World! ", 13, 10, 13, 10


.CODE                 ; segmento de codigo

      .STARTUP

      ; macro para escrita de string no video
      PutStr HelloMessage

      .EXIT
```

***Figura 2** - Código "Hello World" em assembly IA-32 (NASM) para Linux.*

O arquivo **io.mac** que acompanha os exemplos do livro "Guide to Assembly Language Programming in Linux", do Dandamudi, define várias macros que facilitam a leitura e escrita de caracteres, strings e inteiros sinalizados no terminal. Esse arquivo de macros está disponível na página da disicplina: www.dcc.ufam.edu.br/~dcc_oc/tools_linux.zip.

```
SEGMENT stack stack        ; segmento da pilha
resb 0x100

SEGMENT data               ; segmento de dados
HelloMessage     db   'Hello World! ', 13, 10, '$'


SEGMENT code               ; segmento de codigo
start:                     ; ponto de entrada

      ; faz registrador DS apontar para segmento de dados
      mov eax, data
      mov ds, eax

      ; interrupcao para escrita de string no video
      mov ah, 9
      mov dx, HelloMessage  ; ds:dx -> aponta para mensagem
      int 0x21

      ; interrupcao para devolver controle para DOS
exit:
      mov ah, 0x4C
      int 0x21
```

***Figura 3** - Código "Hello World" em assembly IA-32 (NASM) para Windows.*

No DOS, o caracter '$' marca o fim de uma string de caracteres para alguns serviços da BIOS, não sendo exibido na tela. Por isso ele deve ser obrigatoriamente colocado no final de strings quando você estiver programando para DOS.

4.2 Passo 2 – Montagem

Vá até o diretório em que salvou o arquivo **hello.asm** no passo anterior. Para montar o programa, digite:

LINUX: `nasm -f elf hello.asm`

DOS: `NASM -f obj hello.asm`

Em ambos os casos, não deve aparecer nenhuma mensagem de erro. Execute o comando **`ls`** do Linux ou **`dir`** do DOS e note que o arquivo objeto **hello.o** ou **hello.obj** foi criado.

4.3 Passo 3 – Linkedição

A linkedição (ou ligação) é feita no Linux pelo LD (GNU linker) e no Windows pelo TLINK (Turbo linker). Digite um dos comandos abaixo:

LINUX: `ld -s -o hello hello.o io.o`

DOS: `TLINK hello`

Esse passo deve ter criado o arquivo executável em seu diretório de trabalho.

4.4 Passo 4 – Execução

Para executar o programa, simplesmente digite **`./hello`** no terminal do Linux ou **`hello`** no prompt do DOS.

A mensagem "Hello World!" deve ter aparecido.

5. Atividade 2 – Depuração

Nesta atividade, serão vistas as ferramentas de depuração GDB/DDD, para Linux, e Turbo Debugger (TD), para DOS.

4.1 Data Display Debugger

O GNU Debugger (GDB) é um depurador de linha de comando. Uma boa interface visual para ele é provida pelo Data Display Debugger (DDD). Antes de usá-los, é necessário preparar o programa de uma maneira diferente:

```
nasm -f elf -g -F stabs hello.asm
ld -s -o hello hello.o io.o
```

Por último, basta executar o DDD:

```
ddd hello &
```

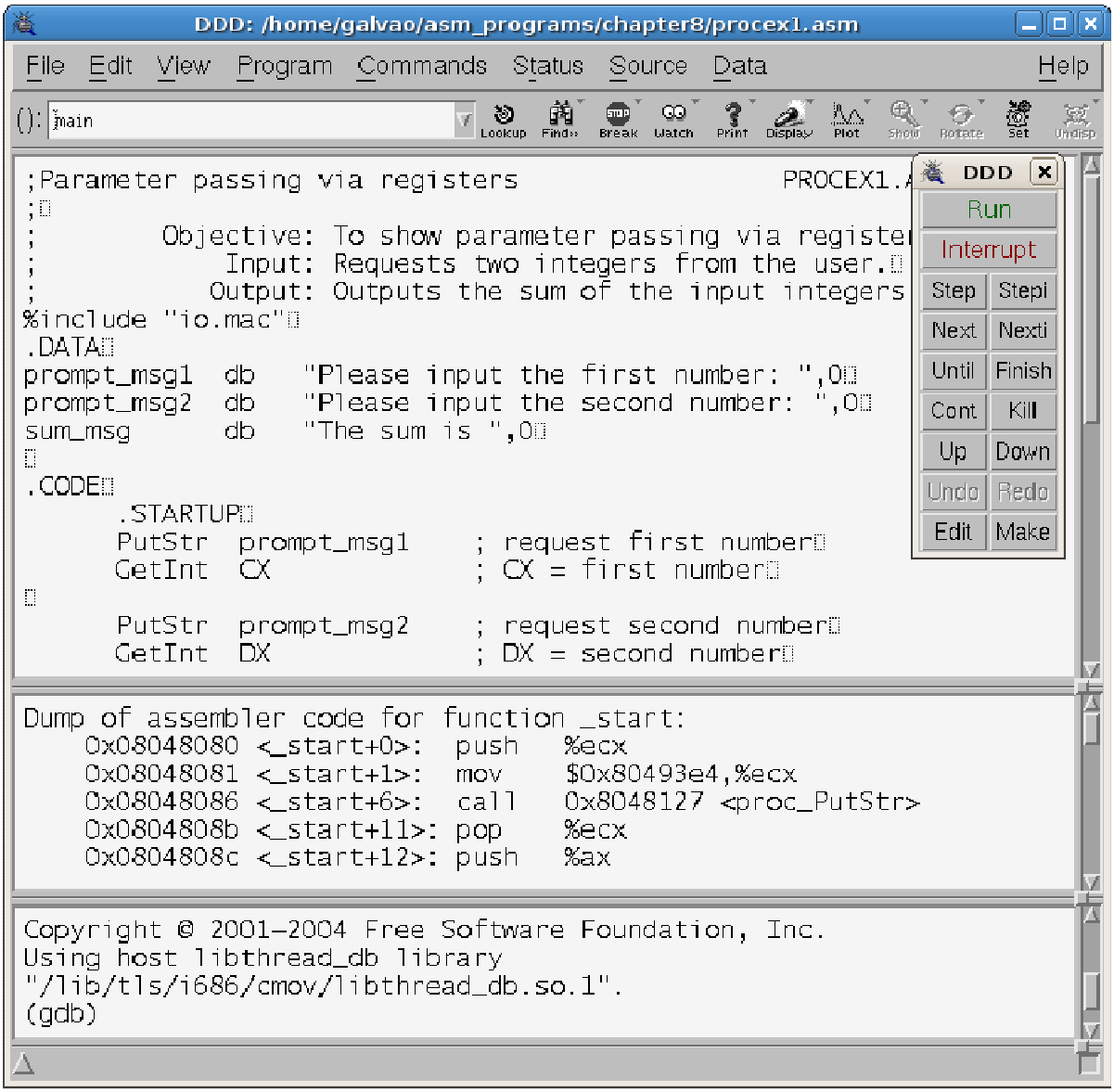

***Figura 4** - Tela inicial do DDD.*

4.2 Turbo Debugger

Antes de se executar o Turbo Debugger, é necessário informar ao ligador (TLink) que queremos informações extras de depuração no arquivo executável. Assim sendo, volte a executar o TLINK usando agora a opção **/v** na linha de comando. Ela informa ao TLINK que desejamos incorporar informacões de depuração ao arquivo executável.

```
TLINK /v hello
```

Agora o arquivo executável hello.exe está apto para ser depurado no TD. Digite a seguinte linha de comando para executar o Turbo Debugger:

```
TD hello
```

As principais janelas do TD estão destacadas na *Figura 5*. Neste primeiro laboratório vamos apenas mostrar os comandos mais simples de depuração.

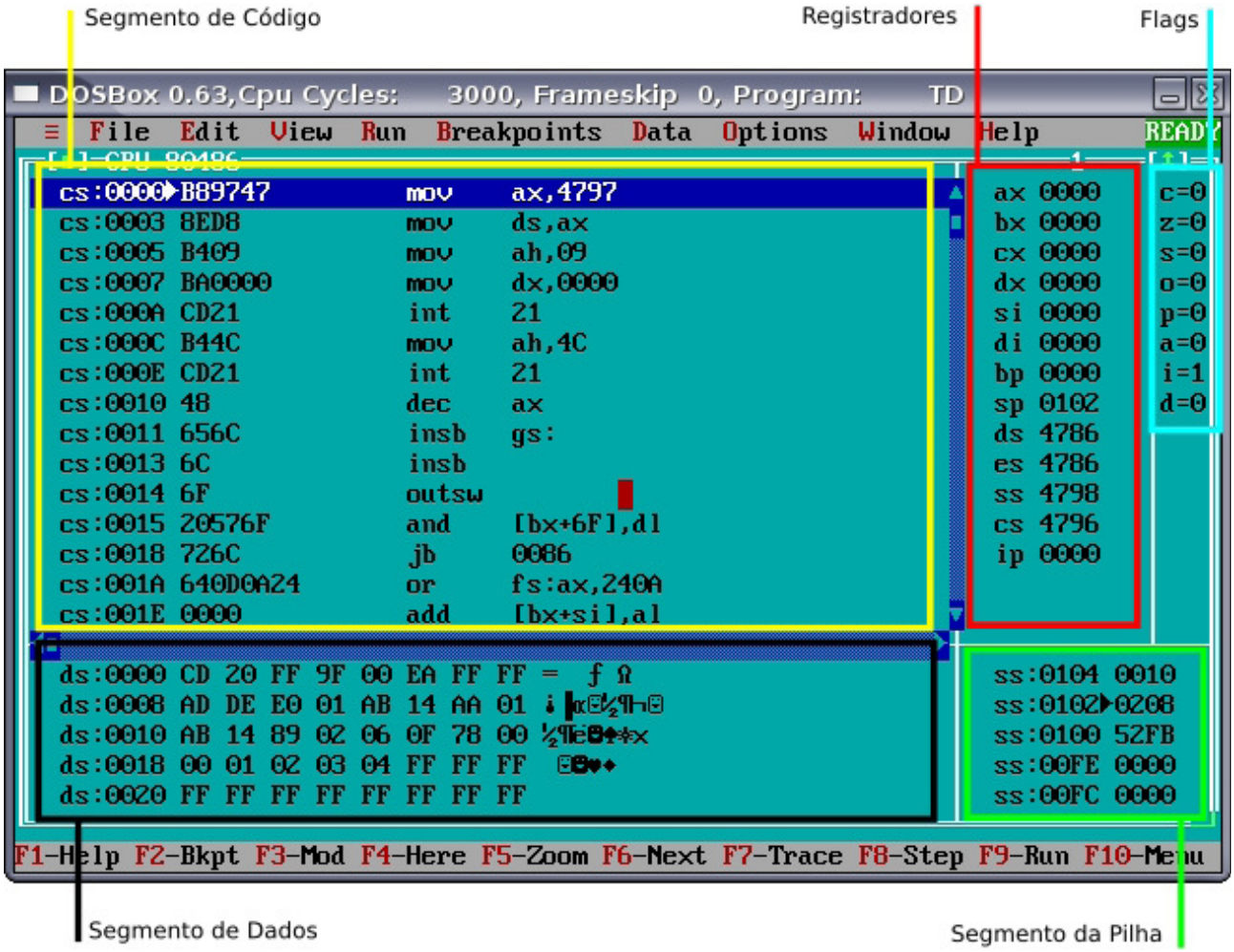

Figura 5 *- Tela inicial do Turbo Debugger.*

Uma seta branca na janela de segmento de código mostra a instrução corrente (que está para ser executada). Para caminhar pelo programa, instrução por instrução, use as teclas F7 ou F8. Pressionando a tecla F7, executam-se as subrotinas, se houver, passo a passo (*trace into*). Com a tecla F8 a subrotina é executada inteiramente em um único passo (*step over*). No exemplo "Hello World", elas têm o mesmo comportamento, já que não há subrotinas. Caminhe pelas instruções até o programa terminar com a mensagem **`Terminated, exit code xx`**.

Para resetar o programa, use CTRL+F2. A tecla F9 executa o programa todo de uma vez só. É possível criar pontos de parada (*breakpoints*) e executar o programa até esses pontos. Para isso, mova com o teclado a linha azul horizontal até a instrução desejada e pressione F2. Essa linha vai ficar marcada em vermelho. Pressionando a tecla F9, o programa será executado até que o primeiro breakpoint seja encontrado (ou que o programa termine).

Como exemplo, use CTRL+F2 e resete o programa. Agora caminhe até a instrução **`mov ah, 0x4C`**. Em seguida pressione F2 para marcar um ponto de parada nessa instrução.

Agora tecle F9 e note que todas as instruções desde o começo do programa até o ponto de parada foram executadas. Para desmacar o breakpoint, basta pressionar F2 novamente. Breakpoints são muito úteis quando se deseja investigar uma determinada área do programa (por exemplo, um procedimento) sem perder tempo caminhando por todas as outras instruções.

Para sair do Turbo Debugger utilize a tecla ALT+X.