Linguagens de montagem
Capítulo 14 - Montadores, Ligadores e Carregadores

Ricardo Anido
Instituto de Computação
Unicamp

Montagem de módulos separados

- Considere um programa que escreve uma cadeia de caracteres no console, implementado em dois arquivos: `escreve.s` e `mensagem.s`
- Como o programa é dividido em dois arquivos, o montador é acionado duas vezes para montar os dois arquivos separadamente.

Arquivo escreve.s

```
@ *********
@ escreve.s
@ *********
@ Procedimento para escrever no console uma cadeia de caracteres
@   terminada por zero
@ Entrada: r0 com endereço do início da cadeia
escreve:
                          @ inicialmente determina número de bytes
    mov      r2,#-1       @ contador de bytes
    mov      r1,r0        @ vamos usar r1 para percorrer a cadeia
escreve1:
    add      r2,#1        @ conta mais este caractere
    ldrb     r3,[r1,r2]   @ carrega o caractere
    cmp      r3,#0        @ verifica se chegou ao final da cadeia
    bne      escreve1     @ desvia se não é final da cadeia
                          @ e fazemos uma chamada a sistema
@ chamada a sistema: write(int fd, const void *buf, size_t count)
    mov     r1,r0         @ apontador para início da cadeia em r1
    mov     r0,#1         @ descritor de arquivo (1 é stdout)
    mov     r7,#4         @ write é chamada a sistema de tipo 4
    svc     #0x55         @ executa chamada
    bx      lr            @ e retorna
```

Arquivo mensagem.s

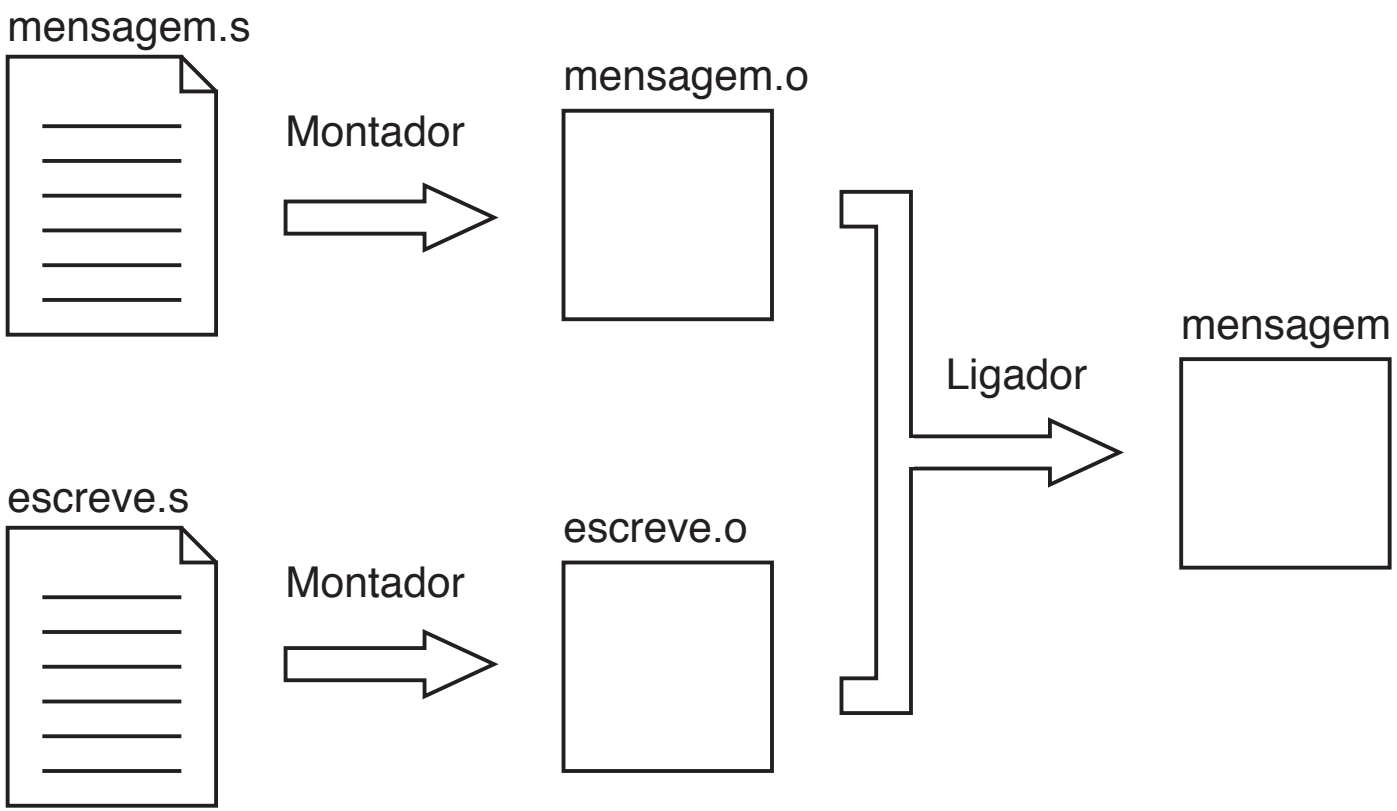
```
@ **********
@ mensagem.s
@ **********
@ Programa que escreve continuamente uma mensagem no console (stdout)
@   assume a existência de um procedimento 'escreve'

    .equ  INTERVALO,0x7fffff @ contador para passagem de tempo
    .org  0x100
inicio:
    ldr   r0,=INTERVALO    @ inicializa contador de tempo
loop:
    subs  r0,#1            @ espera contador de tempo zerar
    bne   loop
    ldr   r0,=cadeia       @ parâmetro: ender. do início da mensagem
    bl    escreve          @ imprime a mensagem
    b     main             @ e continua
cadeia:
     .asciiz   "Cuidado com o degrau!\n"  @ asciiz coloca 0 ao final
```

- O montador processa sem problemas o arquivo `escreve.s`.
- Ao processar o arquivo `mensagem.s`, ao final do passo 1 o montador não tem informação sobre o endereço do rótulo `escreve`. E portanto o comando de chamada ao procedimento `escreve` na linha 15 do arquivo `mensagem.s` não pode ser corretamente montado no passo 2.
- Isto não é um erro do programador, mas apenas uma situação gerada pelo fato de o programa estar dividido em dois arquivos.

Montagem de módulos separados

- O processo de tradução de um programa de arquivo texto para arquivo executável é normalmente realizado por dois programas: o montador e o ligador.
- O *montador* traduz um arquivo texto para um arquivo objeto, ainda não preparado para execução.
- O *ligador* é responsável por gerar um arquivo executável a partir de um ou mais arquivos objeto produzidos pelo montador.

O montador e o ligador



O montador e o ligador

- O montador processa o arquivo texto `mensagem.s` gerando o arquivo objeto `mensagem.o` e processa o arquivo texto `escreve.s` gerando o arquivo objeto `escreve.o`.
- Nenhum desses dois arquivos objeto está ainda no formato executável.
- O ligador então processa os dois arquivos objeto, resolve todas as pendências de endereços de rótulos e produz um arquivo executável (na figura, `mensagem`).
- Para que o montador e o ligador possam gerar o arquivo executável é necessário que o programador forneça mais informações nos arquivos-fonte, através de diretivas do montador.

Rótulos indefinidos

- Os rótulos utilizados mas não definidos no arquivo sendo processado não são sinalizados como erro.
- O montador assume que esses rótulos serão definidos posteriormente em outro arquivo.
- O arquivo objeto criado pelo montador contém uma lista de rótulos *indefinidos*, e para cada um desses rótulos uma lista com as instruções que dependem do valor desse rótulo e devem ser finalizadas pelo ligador.

Rótulos exportados

- Um rótulo somente é exportado pelo montador para ser visível para o ligador se o programador o declarou como sendo *global*, o que é feito através da diretiva do montador .GLOBAL:

  `.global` *lista_de_rótulos*

- onde *lista_de_rótulos* é uma sequência de rótulos separados por vírgula.
- Os rótulos na *lista_de_rótulos* são exportados pelo montador, ou seja, serão visíveis externamente ao módulo corrente.
- O montador inclui no arquivo objeto a informação sobre cada rótulo exportado e seu respectivo endereço.

O ligador

- O ligador processa os arquivos objeto em alguma ordem pré-especificada (por exemplo, se o ligador é chamado da linha de comando, a ordem é a mesma em que os arquivos são descritos na linha de comando).
- O ligador inicia com o apontador de montagem no endereço zero, processa o primeiro arquivo, e para cada um dos arquivos seguintes o ligador usa como valor inicial do apontador de mensagem o seu valor corrente.
- Ou seja, para cada arquivo objeto a não ser o primeiro, o ligador modifica o valor da posição inicial onde o arquivo é efetivamente montado, em relação ao que foi assumido pelo montador.

Relocação

A modificação da posição inicial de um fragmento de código objeto é chamada de *relocação*. Em relação a relocação, um fragmento de código objeto pode ser classificado em três tipos:

- *Código de endereço absoluto* (ou *código não relocável*) é um fragmento de código que deve ser montado em um endereço de memória específico para ser executado.
- *Código independente de endereço* (ou *código autorrelocável*) é um fragmento de código que pode ser montado em qualquer endereço de memória sem que seja necessário alterar o fragmento de código.
- *Código relocável* é um fragmento de código objeto que contém informação suficiente para ser movido para qualquer endereço de memória, desde que o fragmento de código seja alterado.

Código de endereço absoluto

- Quando o programador usa a diretiva .ORG, o fragmento de código desse ponto em diante, até o final do arquivo sendo processado, torna-se não relocável.

Código independente de endereço

- Se um fragmento de código utiliza apenas endereços relativos, ele é naturalmente relocável
- Nenhuma informação adicional é necessária para o ligador relocar o fragmento.
- Exemplo: arquivo `escreve.s`.

Código relocável

- Mesmo quando um fragmento não é independente de endereço, se há informação suficiente no código objeto o ligador pode relocar o fragmento, realizando o ajuste em algumas instruções.

Seções

- O sistema operacional pode designar algumas regiões de memória como *protegidas* contra acessos leitura, escrita ou execução.
- esse esquema de proteção garante que programas de usuário não acessem memória de maneira imprópria, por exemplo alterando o código do sistema operacional.

Seções

A memória de uma programa em execução pode ser classificada pelo sistema operacional em cinco categorias

-
- *código*, que contém o código executável do programa sendo executado. (acesso para leitura e execução, mas não para escrita)
- *dados estáticos de leitura apenas* (acesso para leitura mas não para escrita ou execução).
- *dados estáticos* (acesso para leitura e para escrita, mas não para execução)
- *dados dinâmicos* (em inglês, *heap*) (acesso para leitura e escrita, mas não para execução).
- *pilha*, (acesso para leitura e escrita, mas não para execução).

Seções

O montador GNU-ARM reconhece as seguintes diretivas para a criação de seções:

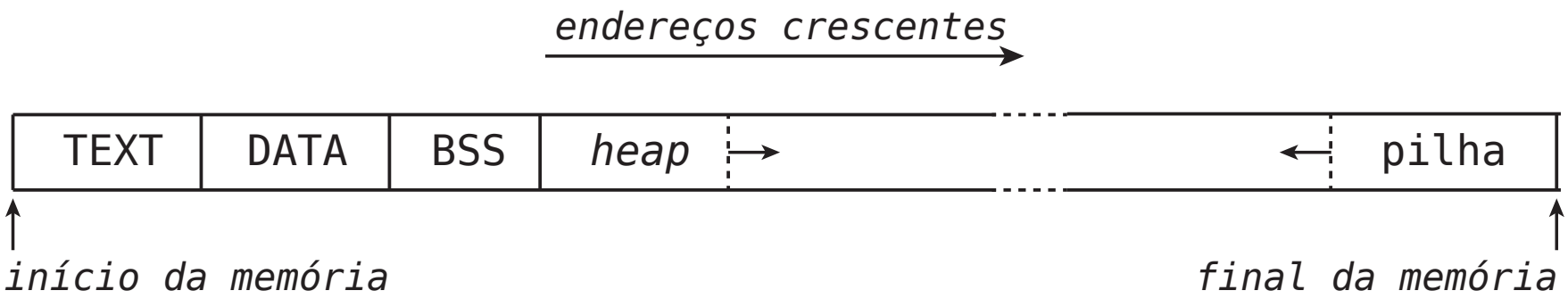
- .TEXT para indicar que o bloco de programa após a diretiva deve ser carregado na área de código (portanto, com proteção contra escrita).
- .DATA para indicar que o bloco de programa após a diretiva deve ser carregado na área de dados, com proteção contra execução; esta é a seção em que são alocados os dados inicializados.
- .BSS para indicar que o bloco de programa após a diretiva deve ser carregado na área de dados, com proteção contra execução; esta é a seção em que são alocados os dados não inicializados.

Exemplo de uso de Seções

```
@ uma seção de dados
.data
.global a,b,c
.align 2
a: .word 1
b: .word 2
c: .byte 4
@ uma seção de código
.text
.align 2
.global main
main:
stmfd sp!, {fp, lr}
        ...                          @ instruções do programa, não mostradas
ldmfd sp!, {fp, lr}
bx lr
@ outra seção de dados, não inicializados
.bss
.align 2
x: .skip   100
y: .skip   200
```

Layout da memória

Disposição das seções em um sistema operacional que usa a abordagem de *memória virtual*, como Linux ou Windows.



Nessa abordagem, o sistema operacional utiliza os recursos de gerenciamento de memória fornecidos pelo processador para fazer com que cada programa execute em um espaço de memória independente, que engloba todo o espaço de endereçamento do processador.

O carregador

- O ligador produz um arquivo objeto executável.
- Para executar o programa, o sistema operacional precisa ainda carregar o conteúdo do arquivo executável na memória do computador.
- O módulo do sistema operacional responsável por essa tarefa é chamado de *carregador* (em inglês, *loader*).
- O carregador lê um arquivo contendo código executável para um programa e o carrega na memória, no endereço especificado.
- Note que o trabalho do carregador é muito parecido com o trabalho executado pelo ligador.
- O carregador pode efetuar ligação e relocação de trechos de código, numa abordagem chamada de *ligação dinâmica*.

Bibliotecas e Ligação dinâmica

Mesmo o programa mais simples em C em geral não descreve todos os comandos necessários para sua execução.

```
#include <stdio.h>

int main(void)
{
  printf("Hello World!\n");
  return 0;
}
```

Bibliotecas e Ligação dinâmica

O código objeto que o ligador produz para o programa `hello` pode ser:

- *estaticamente ligado*, se o arquivo `hello` incorpora o código executável para a função `printf`. É auto-contido, pode ser copiado para outro computador independente se este contém as biliotecas corretas, mas pode ter tamanho grande.
- *dinamicamente ligado*, se o arquivo `hello` não incorpora o código executável para a função `printf`, contendo apenas a informação de onde o código pode ser encontrado (ou seja, em qual arquivo de biblioteca). Tem a vantagem de ter tamanho reduzido, mas para ser executado em outro computador este deve ter as bibliotecas corretas, e o trabalho do carregador é mais complexo.