Sistemas Embarcados - Trabalho Prático 3

Ligador

Arthur Gomes de Lima (2017023650)
arthur.lima@dcc.ufmg.br

João Gabriel de Oliveira Bicalho (2017015134)
jgobi@ufmg.br

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil
2021

1. Introdução

Esta documentação descreve o desenvolvimento de um ligador, um programa responsável por gerar arquivos executáveis a partir de arquivos-módulo. Os arquivos-módulo são gerados por um programa montador e são unificados pelo ligador, gerando os arquivos executáveis que utilizamos no sistema operacional alvo.

Neste trabalho, foi implementado um programa ligador para a arquitetura *Swombat R3.0,* com o objetivo de gerar programas para essa arquitetura em conjunto com o montador implementado no trabalho anterior.

2. Entrada e saída

O ligador desenvolvido recebe uma lista de arquivos-objeto como entrada na linha de comando, e gera um arquivo **mif** (Memory Initialization File) como saída contendo o programa montado e ligado.

Os arquivos-objeto de entrada contêm o código objeto e a tabela de relocação associada gerados a partir da linguagem *assembly* da CPU pelo programa montador. Estes arquivos seguem o formato **mif** e possuem a tabela de relocação escrita como um comentário, dessa forma escolhemos utilizar a extensão `.mifo` (MIF Object).

Após a compilação com **make**, pode-se executar o montador e o ligador com os comandos:

```
montador entrada.a [arquivo-saida]
ligador entrada1.mifo [, … entradaN.mifo] arquivo-saida
```

Caso não haja “arquivo-saida” no montador, ou caso o nome do arquivo de saída no ligador seja “-”, então a saída padrão será usada como buffer de saída.

3. Considerações gerais de implementação

a. Modificação do programa montador

O montador desenvolvido no primeiro trabalho foi implementado de forma a gerar um arquivo executável a partir de um único arquivo de entrada. Para que o programa ligador funcione corretamente, o montador foi modificado para gerar arquivos intermediários: os módulos.

Um **módulo** é um arquivo que contém instruções já montadas, juntamente com informações adicionais sobre as referências de memória do programa. Estas informações permitem que o ligador construa um arquivo final a partir de vários módulos, corrigindo as referências conforme necessário.

O montador modificado neste trabalho gera um arquivo `.mifo` composto de duas seções: um programa parcialmente montado (sem o endereço de referências externas) e uma seção de metadados (contendo a tabela de relocação do módulo e o tamanho do módulo em bytes).

b. Tabela de relocação e metadados

A tabela de relocação contém registros sobre todas as referências encontradas em um módulo. Para facilitar a marcação de globais e valores externos, foram adicionadas à linguagem de montagem as pseudo-instruções: `.externD`, `.externT`, `.globalD` e `.globalT`. Nesta implementação, todas estas pseudo-instruções devem ser definidas no começo do arquivo, pois elas são criadas no primeiro passo do montador e usadas durante os dois passos.

A representação da tabela no arquivo `.mifo` segue o seguinte formato:

```
!internal: loc1 loc2 … locN
var1 E loc1 loc2 … locN
func1 E loc1 loc2 … locN
globlalvar1 G addr
globalfunc1 G addr
```

Na primeira linha são informados os endereços em bytes das instruções que referenciam as labels globais ou locais definidas dentro do módulo.

Nas linhas seguintes, cada linha é uma entrada da tabela de relocação, onde o primeiro termo representa o nome da referência e o segundo representa o tipo de referência: E para referência externa ou G para label global. Não é feito diferenciação entre labels de dados e de código. Os termos seguintes de cada linha dependem do tipo de referência:
- Para globais, a linha contém mais um valor indicando o endereço de memória onde essa label foi definida
- Para labels externas, a linha contém todas as posição de memória onde essa label é referenciada no módulo.

Além da tabela de relocação, também está incluso na seção de metadados uma linha com o tamanho do módulo, com o formato:

```
!end: address_end
```

Nessa linha, *address_end* representa a posição de memória do primeiro byte livre após o término do módulo.

Esta seção de tabela e metadados é representada no final do arquivo **.mifo** como comentários, e todos os números estão em base decimal, assim como no exemplo abaixo:

```
---- START OF THE RELOCATION TABLE ----
-- !end: 44
-- !internal: 8 16 30
--
-- _fib G 0
---- END OF THE RELOCATION TABLE ----
```

c. Política de organização dos módulos

A política de ligação desta implementação é do tipo *first-come-first-served*: o primeiro módulo declarado na linha de comando será o primeiro a aparecer no executável final.

A organização dos módulos é bastante simples: a seção de texto e de dados de um módulo é mantida em sua ordem original no arquivo de saída. Isto é, se o ligador foi chamado com o comando `ligador modulo1.mifo modulo2.mifo saida.mif`, a estrutura do arquivo final será, de forma simplificada:

```
Texto do modulo_1.mifo
Dados do modulo_1.mifo
Texto do modulo_2.mifo
Dados do modulo_2.mifo
```

Uma consequência direta disso é que a primeira instrução a ser executada pela máquina será a primeira instrução do arquivo `modulo_1.mifo`.

4. Resumo da implementação

A execução do ligador segue o roteiro:

a. Análise dos arquivos de entrada

Cada arquivo de entrada é processado e guardado na memória na ordem dos parâmetros de entrada. As seções de código e de metadados são guardadas em vetores diferentes.

b. Edição de referências

Primeiramente, as posições de início e fim dos módulos são ajustadas: o início de um módulo `i` será igual ao fim do módulo `i-1` e o fim deste módulo i será igual a sua posição de início somada ao seu tamanho.

Em seguida, o endereço das labels locais e globais definidas dentro do módulo atual são ajustadas para serem condizentes com a nova posição de início do módulo.

Logo após, as referências externas são ajustadas. Ao encontrar uma referência externa em um módulo, o programa procura por uma label global nas tabelas de relocação dos outros módulos, substituindo a referência pelo endereço da label. O programa acusa um erro caso a referência global não seja encontrada.

c. Geração do arquivo totalmente ligado

Após os passos anteriores, o arquivo final é gerado simplesmente pela concatenação sequencial dos módulos de entrada, formatando-os de acordo com a especificação do formato **mif**.

5. Testes

A fim de comprovar o correto funcionamento do programa ligador, desenvolvemos alguns programas de teste. Os programas estão divididos em módulos para avaliar o funcionamento do programa ligador.

a. basic/teste1.a, basic/max_func.a, basic/mean_func.a

Este programa recebe 3 números do usuário e imprime o maior deles, seguido da média dos 3 valores. O programa é bastante simples: o arquivo teste1.a é responsável por chamar as outras funções e imprimir o valor na tela. O programa realiza testes básicos da capacidade do ligador, pois contém regiões de memória externas aos módulos e as funções ficam em arquivos separados.

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 10
Enter Inputs, the first of which must be an Integer: 20
Enter Inputs, the first of which must be an Integer: 30
Output:  30
Output:  20
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [halt]
```

b. link/main.a, link/t1.a, link/t2.a

Este programa faz uma série de verificações sobre o funcionamento do ligador. Cada módulo possui labels e referências locais e globais, e o intuito principal do programa é verificar se seus endereços foram corretamente ajustados pelo ligador. No fim de uma execução bem sucedida, o programa escreve na tela os números 1, 2, 3, 4 e 5, nesta ordem. Qualquer número diferente impresso ou desvio na ordem de impressão significa um erro de ligação, assumindo que o programa de montagem está funcionando corretamente. Como pode ser visto na figura abaixo é possível ver que o programa não indicou erros para o ligador que implementamos.

```
EXECUTING...
Output:  1
Output:  2
Output:  3
Output:  4
Output:  5
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [halt]
```