Universidade Federal de Minas Gerais

Ingrid Rosselis Sant'Ana da Cunha

Ingrid Elizabeth Spangler

Software Básico

Trabalho Prático 2 - Ligador

Belo Horizonte

2017

1. Introdução:

O segundo trabalho prático da disciplina Software Básico requer a implementação de um ligador para a máquina Swombat, simulada no CPUSim, que deve agir em conjunto com o montador do primeiro trabalho prático. A motivação para esse projeto é o fato de que a maioria dos programas constitui de um ou mais procedimentos, onde cada um gera um módulo-objeto, e há um grande uso de bibliotecas externas às linguagens. Nesses casos, onde o programa constitui de mais módulos, é necessário para a execução que os módulos-objeto possuam um único espaço de endereçamento, então é preciso que sejam previamente ligados e concatenados após a tradução, e que apenas um arquivo binário total contendo todos os outros seja gerado ao final para a execução.

O montador implementado no primeiro trabalho é um programa que executa o processo de tradução por dois passos para um código cuja linguagem-fonte é Assembly e a linguagem-alvo é a própria linguagem de máquina. Tanto ele quanto o CPUSim, onde os programas gerados são testados, consideram que o programa é constituído de apenas um módulo e por isso geram o executável em binário diretamente. Para essa etapa, o montador deve ser alterado para que gere apenas um programa-objeto, o qual será entregue como entrada para o ligador.

O ligador, por sua vez, junta e ordena todos os programas-objeto, alterando os endereços, as referências à símbolos e a outros módulos, e sempre inicia pelo módulo principal. A seguir ele concatena os outros módulos em sequência e refaz os endereços somando-os com um offset, que é a quantidade de linhas de módulos que já foram escritas no arquivo.

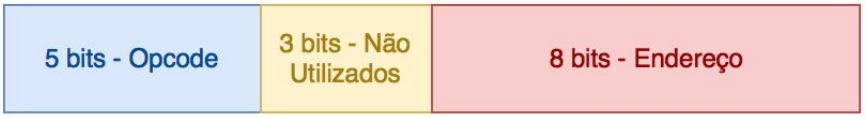
2. Modelagem:

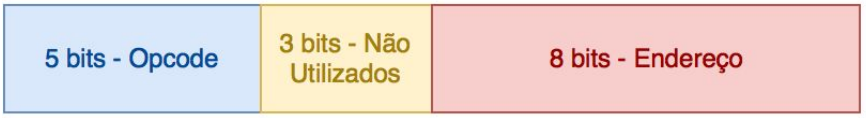
Mais especificamente, o ligador implementado opera em duas etapas: na primeira etapa os arquivos a serem ligados são lidos um por um e as informações de suas tabelas de símbolos são guardadas em uma tabela no programa (entry point table). O endereço final dos pontos iniciais de cada função também é calculado. Na segunda etapa os arquivos são lidos novamente, e linha por linha o código é impresso no arquivo .mif, já traduzindo as labels para binário por consulta da tabela e impressas.

Como o montador precisou ser modificado, e na primeira etapa ele construía o executável do programa com as chamadas das labels e o cabeçalho requeridos para a interpretação do CPUSim, estas funções foram removidas e transferidas para o ligador, ou seja, apenas o código parcialmente traduzido em linguagem de máquina, as labels em plaintext e a tabela de símbolos com referências relativas às declarações das funções internas são gerados pelo montador e impressos no arquivo objeto.

3. Decisões de Implementação:

Para o desenvolvimento do ligador foi dado um maior poder de decisão na escolha dos métodos a serem utilizados, especialmente na formulação do melhor modelo de objeto para auxiliar o ligador. Foi decidido colocar o número de linhas de cada um dos módulos dentro de seus próprios objetos pois os arquivos já haviam sido lidos pelo montador, o que facilitou o cálculo do offset, e todo o tratamento das labels foi delegado ao ligador, inclusive das labels internas. Por isso, com relação às instruções tratadas pelo montador, somente as do tipo J (jump, jmpn, jmpz e call) e a loadc (no caso de a constante do argumento ser uma variável local alocada com .data) foram afetadas pelas mudanças pois na arquitetura Swombat possuem a seguinte forma e podem receber labels como operandos:





Agora seu campo de endereço não é mais traduzido na etapa de montagem, mas são impressas no lugar as próprias labels, que servirão como symbolic links. Com relação à entry point table adicionada ao objeto, ela possui o seguinte formato (exemplo):

!12
!_label1 4
!_label2 8

!12 : número de linhas do módulo
!_label1 4 : declaração da label e endereço relativo da label

Decidimos que não é necessário utilizar uma tabela de referências externas, pois a arquitetura Swombat possui uma grande regularidade em seu formato de instruções, apenas a utilização das labels como links simbólicos foi suficiente para o correto funcionamento do ligador.

Para exemplificar o todo, segue abaixo o trecho de um objeto gerado pelo montador:

```
25 01000011
26 _pro
27 00100011
28 10000000
29 01000011
30 _med
31 00000000
32 00000000
33 10011000
34 _maior
35 00000000
36 00000000
37 10011000
38 _menor
39 00000000
40 00000000
41 10011000
42 _soma
43 00000000
44 00000000
45 10011000
46 _produto
47 00000000
48 00000000
49 10011000
50 _media
51 00000000
52 00000000
53 !52
54 !_mai 32
55 !_med 48
56 !_men 36
57 !_pro 44
58 !_som 40
59
```

Outras decisões de projeto são à respeito do tratamento de erros: caso o programa não encontre o endereço de uma label, uma mensagem avisa que há um módulo em falta e, a adição de módulos extras não interfere no funcionamento do programa original. Para facilitar a identificação das meta informações do objeto, o símbolo exclamação é colocado antes da label (identificação por expressão regular).

Os módulos externos poderão conter e acessar variáveis estáticas (.data), sem problemas, o programa principal continua tendo esta funcionalidade.

4. Análise de Resultados:

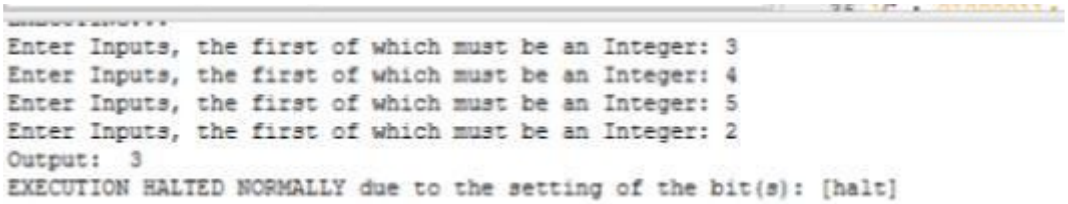
Para a análise dos resultados foi utilizado um programa produzido como definido na especificação, que deve receber 3 números e um operador e proceder alguma das 5 operações padrão do programa. Por escolha arbitrária, os três primeiros dados pedidos pelo programa são os operandos e o quarto é o operador.

A fim de demonstrar o funcionamento do montador, do ligador e do programa teste, seguem prints do funcionamento no CPUSim das 5 operações com os operandos 3,4 e 5.

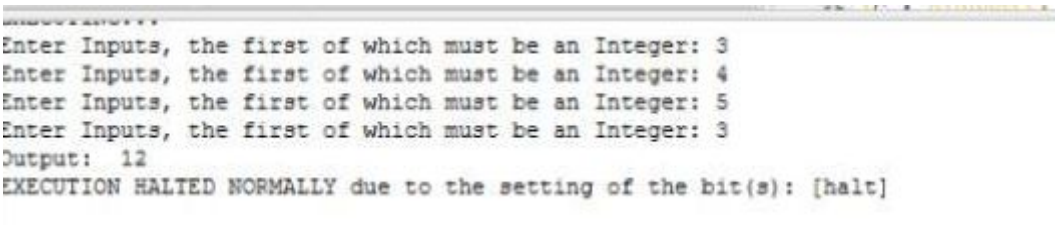
Operando 1, procura o maior inteiro dentre os três recebidos:

```
Enter Inputs, the first of which must be an Integer: 3
Enter Inputs, the first of which must be an Integer: 4
Enter Inputs, the first of which must be an Integer: 5
Enter Inputs, the first of which must be an Integer: 1
Output:  5
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [halt]
```

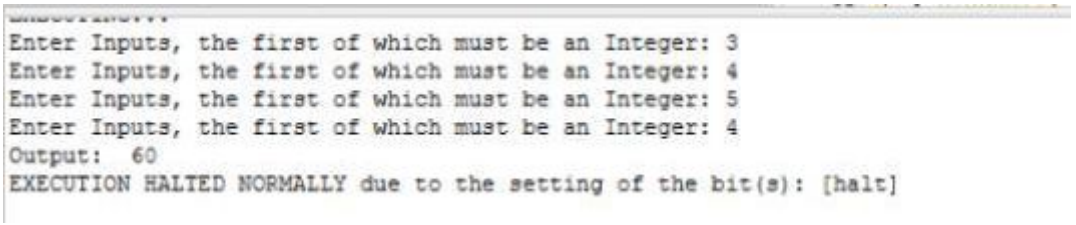
Operando 2, procura o menor inteiro dentre os três recebidos:



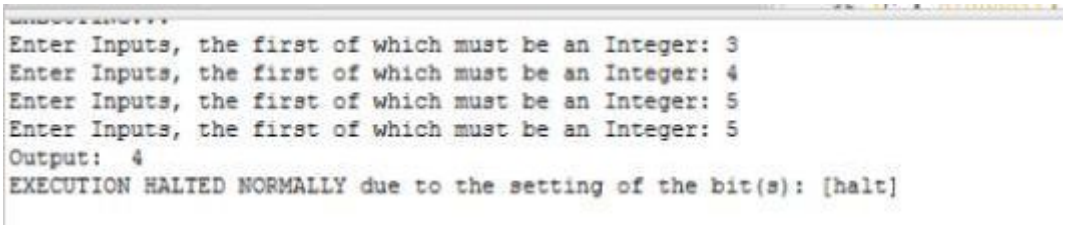
Operando 3, calcula a soma dos três operandos:



Operando 4, calcula o produto dos três operandos:



Operando 5, calcula a média entre os três operandos:



5. Conclusões:

Neste trabalho prático entendemos a importância e o funcionamento de um ligador dentro do processo de montagem e execução de procedimentos. O ligador é um passo importante para resolver o problema de realocação, como sistemas operacionais UNIX e Windows funcionam com um único espaço de endereçamento linear, programas com diferentes módulos não podem ser apenas traduzidos para instruções de máquina e carregados na memória sem antes passar por uma conversão tanto dos endereços quanto das referências externas, dois problemas que o ligador resolve de uma maneira simples.