

UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
DCC008 – SOFTWARE BÁSICO – TB

Trabalho Prático I
Montador para a máquina Swombat R3.0

Alunos:
Bruno Varella Peixoto
Matheus Vargas

Professor:
Daniel Macedo

Belo Horizonte
18 de maio de 2018

1 Introdução

Salomon (SALOMON, 1993) considera um assembler como um tradutor que traduz instruções provenientes de uma linguagem simbólica em instruções de linguagem de máquina para uma determinada arquitetura alvo, sendo a tradução feita de um para um.

Uma das razões para o estudo de assemblers reside no fato de que as operações de um assembler refletem a arquitetura do computador alvo uma vez que a linguagem de montagem é fortemente dependente da organização interna do computador. Características arquiteturais tais como o tamanho da palavra de instrução/dados, formato numérico, codificação interna de caracteres, registradores de propósitos específicos e gerais, organização da memória, tipos de endereçamento e instruções, etc; afetam o modo de como as instruções são escritas e como o assembler lida com as instruções e diretivas associadas.

O assembler considerado neste trabalho é o assembler de dois passos, descrito no livro texto adotado na disciplina de Software Básico. Um assembler de dois passos consiste em uma implementação que faz a leitura do código fonte em linguagem simbólica duas vezes, a primeira passagem para a resolução da referência antecipada em que as definições de símbolos e rótulos de declarações são coletadas e armazenadas em uma tabela. A segunda passagem todos os valores simbólicos são conhecidos, não restando nenhuma referência antecipada, então, cada declaração lida pode ser montada e traduzida. O método de duas passagens apesar de requer uma passagem extra, é um método de implementação simples. Maiores detalhes sobre o processo de montagem por duas passagens podem ser obtidos em (TANENBAUM; AUSTIN, 2013) [cap. 7].

Tipicamente um assembler moderno possui duas entradas e saídas. A primeira entrada é a que ativa o assembler e especifica os parâmetros e o nome do arquivo de entrada. A segunda entrada é o arquivo fonte caminho (contendo código fonte em linguagem simbólica de máquina e diretivas de montagem). A primeira saída de um assembler típico é o arquivo objeto o qual contém as instruções montadas, ou seja, um programa em linguagem de máquina a ser posteriormente carregado e executado na máquina alvo. A segunda saída típica de um assembler é o arquivo de listagem o qual contém informações relevantes do processo de montagem tal como as instruções de máquina e diretivas (SALOMON, 1993).

1.1 Objetivo

O objetivo deste trabalho concerne o desenvolvimento de um montador (assembler), um software capaz de realizar a tradução de um programa escrito em uma linguagem de montagem para uma linguagem de máquina específica, neste caso, a Swombat R3.0, a qual será simulada através do software simulador CPUSim.

Espera-se na execução deste trabalho praticar os conceitos desenvolvidos na disciplina de Software Básico tais como o processo de tradução de linguagem simbólica para linguagem de máquina, uso de um software simulador de máquina alvo (CPUSim simulando a máquina Swombat R3.0), estruturas de dados adequadas para manipulação de arrays associativos, processo de tokenização, análise léxica, análise sintática e tradução para a máquina alvo.

2 Implementação

A implementação do assembler para a máquina Swombat R3.0 foi baseada nos conceitos do montador de dois passos descrito em (TANENBAUM; AUSTIN, 2013) [cap. 7]. Basicamente, o primeiro passo consiste na tokenização do arquivo contendo a implementação em linguagem simbólica, resolvido a questão da referência posterior através da montagem das tabelas de símbolos e pseudo-instruções. As tabelas foram elaboradas utilizando arrays associativos do tipo hash, facilitando tanto a inserção de símbolos quanto à pesquisa destes. A segunda passagem é realizada de forma que sempre que um símbolo com referência seja encontrado, a referência é pesquisada em uma das tabelas conforme o caso (tabela de símbolo ou tabela de pseudo-instruções) e então toda a instrução é montada e inserida no arquivo de saída. A composição do programa em sua totalidade poderá ser observada nos blocos da figura 1 a qual ilustra os blocos e o inter-relacionamento deles (nem todos os blocos foram implementados no trabalho).

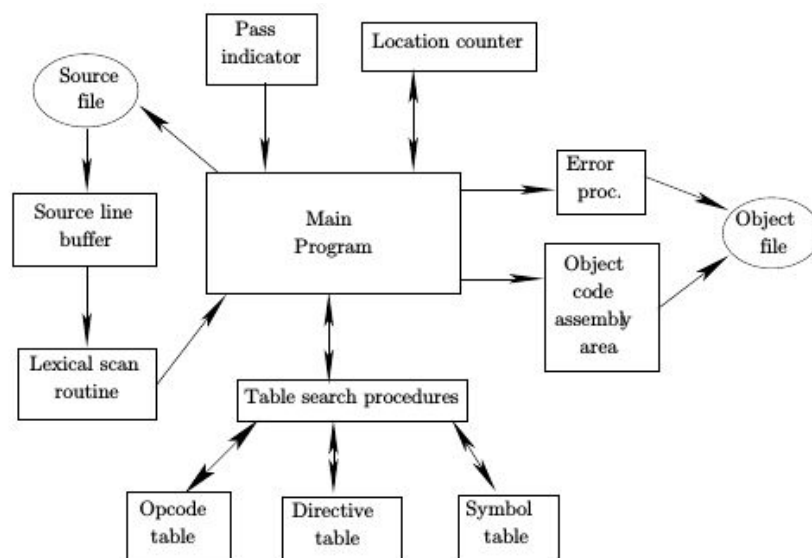
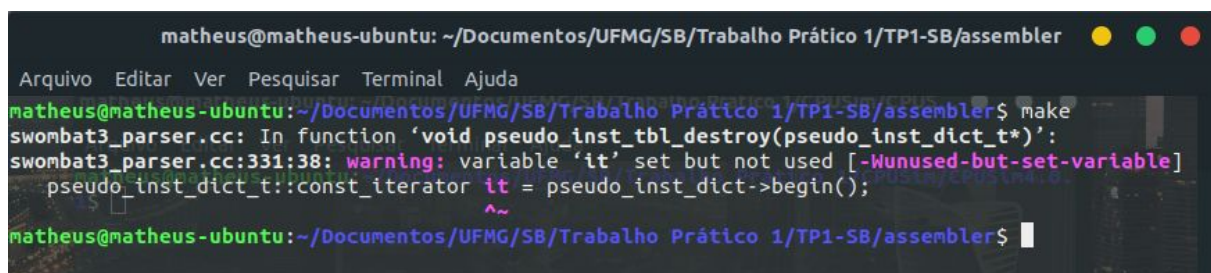


Figura 1 Composição do assembler para a máquina Swombat (SALOMON, 1993)

3 Ambiente de desenvolvimento

3.1 Compilação

Para compilar o programa, basta ir ao diretório onde encontra-se o código fonte (diretório assembler) e através do utilitário *make*, executar o comando. Automaticamente o utilitário fará a leitura do arquivo makefile o qual tem uma espécie de “receita” para a compilação do programa.



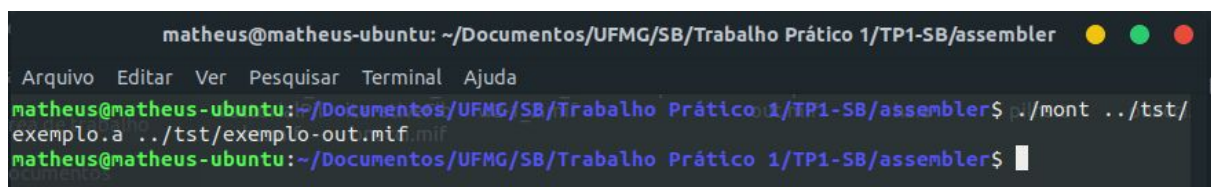
```
matheus@matheus-ubuntu: ~/Documentos/UFGM/SB/Trabalho Prático 1/TP1-SB/assembler
Arquivo Editar Ver Pesquisar Terminal Ajuda
matheus@matheus-ubuntu:~/Documentos/UFGM/SB/Trabalho Prático 1/TP1-SB/assembler$ make
swombat3_parser.cc: In function 'void pseudo_inst_tbl_destroy(pseudo_inst_dict_t*)':
swombat3_parser.cc:331:38: warning: variable 'it' set but not used [-Wunused-but-set-variable]
    pseudo_inst_dict_t::const_iterator it = pseudo_inst_dict->begin();
                                     ^~
matheus@matheus-ubuntu:~/Documentos/UFGM/SB/Trabalho Prático 1/TP1-SB/assembler$
```

3.2 Execução

A execução do programa é dada da seguinte forma (ambiente GNU/Linux):

```
$ ./mont <file_in.a> <file_out.mif>
```

Ao executar o comando acima, o programa: mont será invocado, tendo como argumento o arquivo de entrada: file_in.a e opcionalmente um nome para o arquivo de saída file_out.mif. Caso usuário não especifique um arquivo de saída, um nome default é dado para o arquivo de saída (a.mif), neste caso, a forma de invocação do programa é: \$./assembler <file_in.a>. Caso não ocorra nenhum erro de montagem um arquivo com a extensão *.mif será dado. Caso ocorra alguma anomalia, uma mensagem de erro é dada na saída de erros padrão (stderr) e a execução do programa é encerrada.



```
matheus@matheus-ubuntu: ~/Documentos/UFGM/SB/Trabalho Prático 1/TP1-SB/assembler
Arquivo Editar Ver Pesquisar Terminal Ajuda
matheus@matheus-ubuntu:~/Documentos/UFGM/SB/Trabalho Prático 1/TP1-SB/assembler$ ./mont ../tst/
exemplo.a ../tst/exemplo-out.mif.mif
matheus@matheus-ubuntu:~/Documentos/UFGM/SB/Trabalho Prático 1/TP1-SB/assembler$
```

4 Resultados

Para testarmos o montador criamos três programas simples na linguagem de montagem da máquina Swombat, apresentados nas seguintes subseções:

Os códigos dos programas em linguagem de montagem foram omitidos na documentação uma vez que foram fornecidos juntamente com os demais arquivos do assembler.

4.1 Programa 1 - Fatorial usando a Pilha

O primeiro programa é uma função do cálculo do fatorial usando a pilha interna da máquina. O número que deseja-se calcular o fatorial é dado através da leitura da entrada padrão (teclado), o fatorial é então calculado e o valor impresso na saída padrão (monitor). O valor máximo do fatorial calculado é o 8. Acima deste valor, por restrições da máquina, o resultado é imprevisível. Opcionamente poderia ser feito um programa de maior complexidade, mas devido ao tempo escasso, optou-se por implementar o algoritmo de forma mais simples, porém suficiente para verificar o funcionamento correto da implementação e do assembler.

The screenshot displays the Swombat assembler interface with three open files: `teste-funcoes-out.mif`, `multiplicacoes-out.mif`, and `fatorial-pilha-out.mif`. The `fatorial-pilha-out.mif` file is active, showing assembly code for calculating a factorial using a stack. The code includes instructions for setting registers, pushing values onto the stack, and popping them to calculate the result. The interface also shows the state of registers, main memory, and the stack.

Registers:

Name	Width	Data
buffer1	16	24
buffer2	16	1
ir	16	0
mar	12	8
mdr	16	0
pc	12	10
status	3	-4

Stack:

Addr	Data
00	00
01	08
02	00
03	16
04	00
05	16
06	00

Execution Log:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 4
Output: 24
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [halt]
```

4.2 Programa 2 - Multiplicação de dois inteiros por somas sucessiva

O segundo programa em linguagem de máquina, em linhas gerais, realiza a multiplicação de dois inteiros através de soma sucessiva (sem o uso da instrução de máquina de multiplicação).

The screenshot displays a MIPS simulator window with the following components:

- Registers:** A table showing the state of MIPS registers.

Name	Width	Data
buffer1	16	6
buffer2	16	1
ir	16	0
mar	12	26
mdr	16	0
pc	12	28
status	3	-4
- Memory (Main and Stack):** Two tables showing memory contents.

Addr	Data
00	18
01	00
02	18
03	01
04	62
05	02
06	60

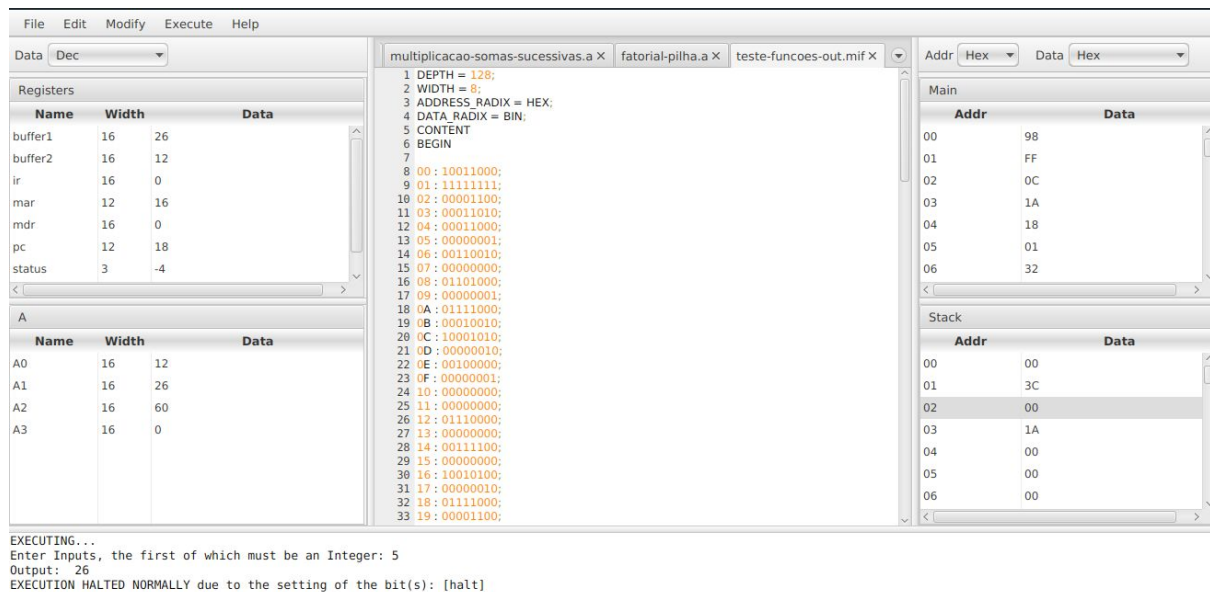
Addr	Data
00	00
01	00
02	00
03	00
04	00
05	00
06	00
- Assembly Code:** A list of 33 instructions in MIPS assembly, including constants, register assignments, and arithmetic operations.

```
1 DEPTH = 128;
2 WIDTH = 8;
3 ADDRESS_RADIX = HEX;
4 DATA_RADIX = BIN;
5 CONTENT
6 BEGIN
7
8 00: 00011000;
9 01: 00000000;
10 02: 00011000;
11 03: 00000001;
12 04: 01100010;
13 05: 00000010;
14 06: 01100000;
15 07: 00000001;
16 08: 00001110;
17 09: 00011100;
18 0A: 00110100;
19 0B: 00000011;
20 0C: 01001000;
21 0D: 00001110;
22 0E: 01010100;
23 0F: 00011000;
24 10: 00101000;
25 11: 00000001;
26 12: 00001110;
27 13: 00011100;
28 14: 00110100;
29 15: 00000011;
30 16: 01001000;
31 17: 00001110;
32 18: 00100000;
33 19: 00000000;
```
- Execution Log:** A text area at the bottom showing the execution progress:

```
EXECUTING...
Enter Inputs, the first of which must be an Integer: 2
Enter Inputs, the first of which must be an Integer: 3
Output: 6
EXECUTION HALTED NORMALLY due to the setting of the bit(s): [halt]
```

4.3 Programa 3 - Testes de chamada de pilha em funções

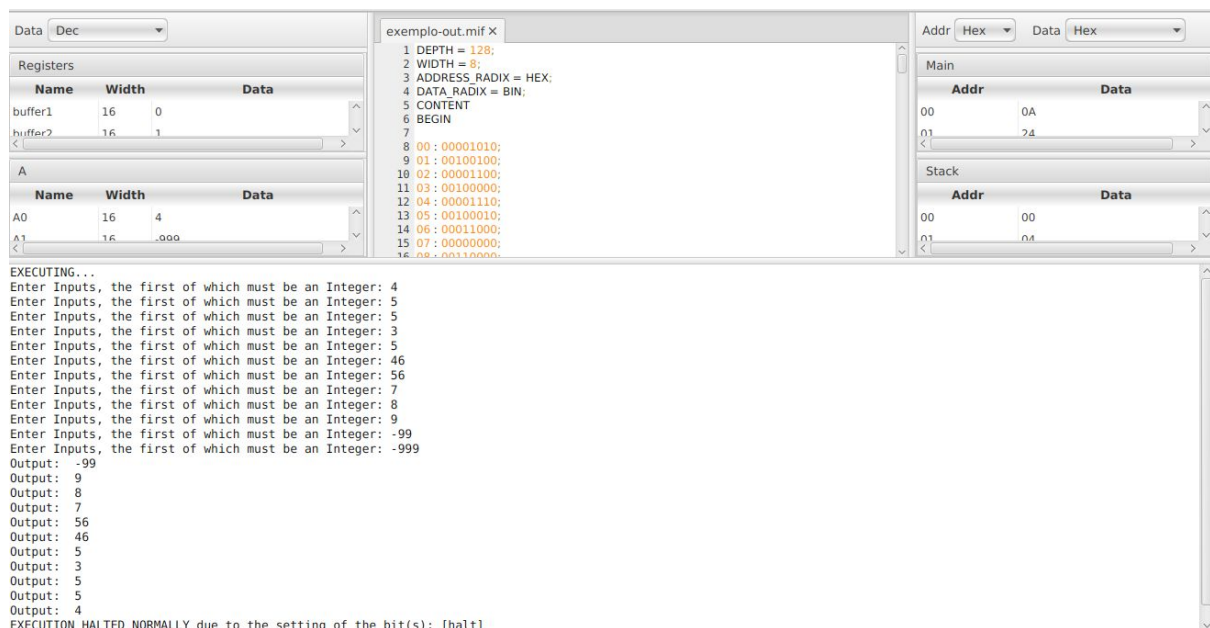
O último programa do conjunto de testes, visa verificar o funcionamento das instruções de armazenamento em pilha como argumentos para chamadas posteriores de funções.



4.4 Programa 4 - Exemplo Fornecido

A seguir serão apresentados os resultados da execução do programa de testes exemplo fornecido como base para implementação.

Após executar o montador sob a entrada de exemplo “exemplo.a”, a saída “out.mif” foi carregada na memória da máquina Swombat resultando no comportamento correto do programa.



Obs: O CPUSim deve ser constantemente limpo (memória, programas carregados). Caso o funcionamento não seja o correto, pode ser aconselhável reiniciar o programa.

5 Conclusões

O trabalho prático propiciou maior entendimento da composição (programação) e organização (estruturação) de um pequeno assembler para uma máquina específica (Swombat). Seguindo as orientações contidas no livro texto da disciplina (TANENBAUM; AUSTIN, 2013)[cap. 7] referente à composição de um assembler de dois passos.

Objetivando princípios de engenharia de software o assembler foi escrito de forma multimodular (vários arquivos) de forma a ter baixo acoplamento entre as partes integrantes do projeto e alta reusabilidade, aproveitando a capacidade de expressão da linguagem utilizada (Linguagem C++). As principais estruturas de dados utilizadas (tabela de dispersão ou tabela hash, lista, strings, etc) foram as disponíveis da biblioteca padrão da linguagem e também da STL. A compreensão da manipulação e alocação das estruturas inicialmente foi tida como difícil, dado o paradigma de visualizar mentalmente as estruturas de dados utilizadas e como manipulá-las durante a composição do código fonte. A ferramenta de debug e checagem de vazamento de memória Valgrind foi utilizada com o intuito de checar as alocações e desalocações de recursos ao longo do programa.

8 Referências Bibliográficas

SALOMON, D. Assemblers and Loaders. 1993. Disponível em: <<http://www.davidsalomon.name/assem.advertis/asl.pdf>>.

TANENBAUM, A. S.; AUSTIN, T. Structured Computer Organization. 6. ed. [S.l.]: Pearson, 2013. ISBN 0-13-291652-5.