

# Trabalho Prático 2 - Montador

Victor Pires Diniz

11 de Agosto de 2016

Software Básico - 2º Semestre de 2015

## 1 Descrição do trabalho

O segundo trabalho prático do semestre envolve o desenvolvimento de um montador para o código de montagem de uma máquina virtual especificada, para a qual foi feito um emulador no trabalho prévio. Esse montador teria suporte a processamento de *labels* para determinar posições de memória e também a algumas pseudo-instruções, além das vinte e duas instruções da máquina virtual. Com código de montagem, desenvolver programas para o emulador se torna muito mais simples, visto que o mesmo é de muito mais alto nível do que o código em linguagem de máquina, devido aos nomes mnemônicos, à presença das *labels* e à possibilidade de uso de comentários.

## 2 Implementação e decisões de projeto

O código do montador está dividido semanticamente entre vários módulos:

- *main.c*: recebe parâmetros por linha de comando e chama o montador apropriadamente.
- Map (*map.c*, *map.h*, *bucket.c*, *bucket.h*): implementa uma tabela de dispersão genérica, instanciada na tabela de símbolos e, também, na IDT.
- IDT (*idt.c*, *idt.h*): implementa uma tabela que relaciona o nome das instruções às suas características e ao procedimento relacionado às pseudo-instruções.
- Pseudo-instruções (*pseudo\_instr.c*, *pseudo\_instr.h*): contém dados sobre as pseudo-instruções, assim como funções que desempenham as ações associadas a elas.
- Metadados das instruções reais (*real\_md.c*, *real\_md.h*): contém vetores com metadados sobre cada instrução real, como, por exemplo, o número de operandos e o tipo de cada um deles.
- Função hash auxiliar (*hash\_aux.c*, *hash\_aux.h*): contém uma função para hashing de string.
- Montador (*assembler.c*, *assembler.h*): implementa o montador e os procedimentos relacionados a ele.

Os mais importantes deles serão analisados a seguir em mais detalhe.

### 2.1 Map

O módulo map contém a implementação de uma hash table totalmente genérica, com tratamento de colisão através de listas encadeadas, definidas nos arquivos *bucket.c* e *bucket.h*. A interface para o tipo contém, entre outras funcionalidades, um pseudo-iterador, que é utilizado na impressão da tabela de símbolos em modo verboso.

## 2.2 IDT

Esse módulo cria uma *instruction data table*, que relaciona o nome de cada instrução ou pseudo-instrução aos dados necessários para sua montagem (ou execução, no caso das pseudo-instruções). Para isso, são utilizados dados sobre essas instruções, disponíveis nos módulos *pseudo\_instr* e *real\_md*. A função *idtCreate* gera uma hash table indexada por strings, cujos valores são estruturas que contêm um tipo *PseudoInstr* ou *RealMD*, a depender de se tratar de uma instrução real ou não.

### 2.2.1 Montador

O montador opera em dois passos principais. Ambos mantêm o estado do montador através do *instruction location counter* (ILC), incrementado na leitura de cada operador ou operando. Além disso, eles compartilham o uso da função *asmParseLine*, que divide cada linha em seus componentes: *label*, *operador* e dois possíveis operandos.

O primeiro deles, desempenhado na função *asmBuildSymTable*, consiste em passar pelo código de montagem em busca de labels, registrando a posição de cada uma (de acordo com o ILC) na tabela de símbolos. Outras funções associadas à tabela de símbolos são *asmDestroySymTable* e *asmPrintSymTable*, que liberam a memória associada à tabela e a imprimem, respectivamente.

Ao passar pela segunda vez, com a função *asmReplaceAndSave*, a montagem real é realizada, imprimindo para o arquivo de saída o código de máquina associado a cada instrução. Nessa fase, as labels são substituídas pelo endereço presente na tabela de símbolos, com compensação relativa ao ILC atual, para garantir que os endereços sejam utilizados relativamente ao contador de programa, como proposto na especificação da máquina. A função *assembleOperand* é utilizada para imprimir diferentemente operandos registradores ou valores imediatos, visto que é necessário tratá-los de formas diferentes.

## 3 Compilação e execução

A compilação do montador pode ser realizada através da *makefile* disponibilizada ou diretamente através do GCC ou outro compilador C. Caso compilado através da *makefile*, o executável estará localizado na pasta *bin/*. A execução do programa deve ser realizada através da linha de comando, na seguinte forma,

```
{endereço do executável do montador} <input_addr> <output_addr> [output_mode]
```

em que:

- *input\_addr*: endereço para o arquivo de entrada, em código de montagem.
- *output\_addr*: endereço para o arquivo de saída a ser criado.
- *output\_mode*: parâmetro **opcional** que permite escolher entre os modos de execução simples e verboso. Caso não seja especificado, o programa é executado em modo simples, sem informação de depuração. Valores válidos: *s* (simples) ou *v* (verboso).

## 4 Testes realizados

Na pasta de testes presente no pacote deste trabalho, há diversos programas que foram utilizados para garantir o bom funcionamento do montador, cobrindo todas as instruções disponibilizadas pela especificação da máquina virtual. Vários deles foram implementados de acordo com o pedido na especificação do trabalho. Imagens da execução dos testes estão disponíveis no apêndice desta documentação. Segue abaixo uma breve descrição do comportamento de cada programa:

- *tspec.i*: teste disponibilizado na especificação do trabalho. Determina o maior número entre dois valores informados.
- *tfib.i*: calcula o número de Fibonacci de acordo com um índice fornecido pelo usuário. (Requerido na especificação.)

- `texp.i`: realiza a exponenciação de um número, de acordo com uma base e um expoente, ambos fornecidos pelo usuário. (Requerido na especificação.)
- `tdiv.i`: realiza a divisão por força bruta de dois números inteiros, retornando quociente e resto. (Requerido na especificação.)
- `tmdn.i`: determina a mediana de um conjunto de sete itens fornecidos pelo usuário. (Requerido na especificação.)
- `tothers.i`: realiza operações variadas com instruções que não foram cobertas pelos outros testes, para garantir o bom funcionamento de tudo.

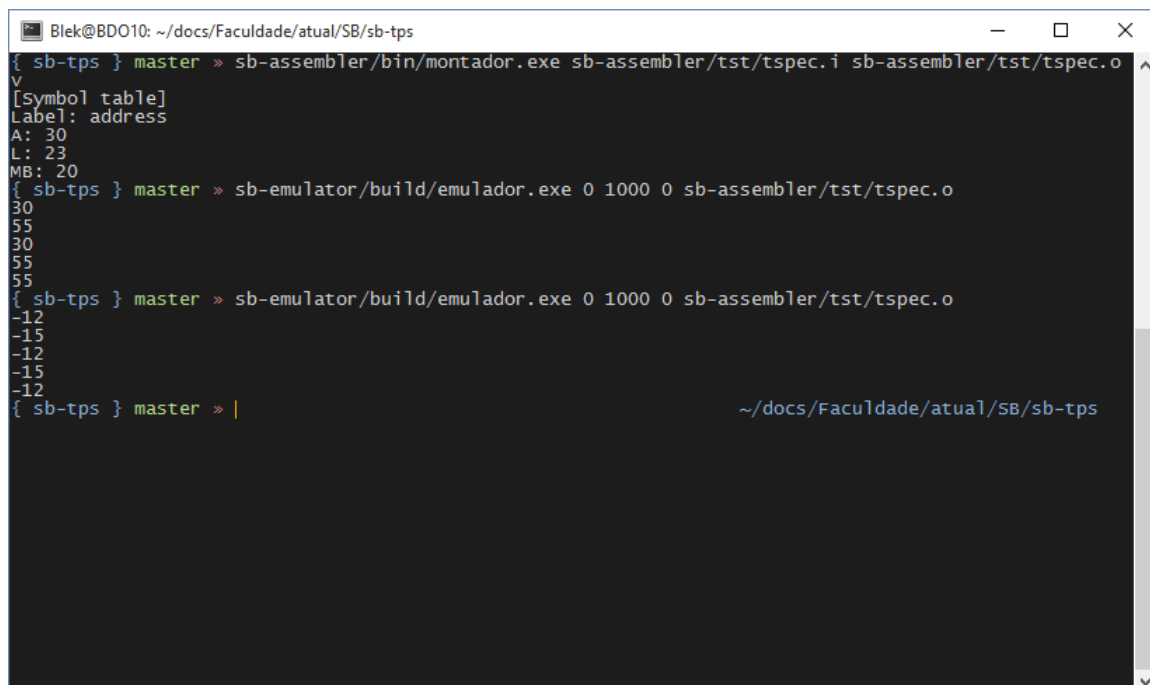
## 4.1 Testes unitários

Além das entradas de teste elaboradas, foram criados também diversos *unit tests*, com o propósito de testar a funcionalidade de cada módulo do montador. Esses testes estão disponíveis na pasta `unit-tests/`, dentro da pasta de testes, e podem ser compilados com o comando `make tests`, que utiliza uma funcionalidade adicional da *makefile* providenciada. Após a compilação, eles estarão localizados na pasta `bin/test-bin/`.

## 5 Conclusão

Neste trabalho, foi implementado um montador para uma máquina virtual, definido de acordo com a especificação fornecida. O comportamento e a implementação do montador foram discutidos no contexto dos testes realizados e da interação com o emulador criado no trabalho prático anterior, de forma a garantir que as instruções e pseudo-instruções funcionam como previsto.

## A Imagens de execução dos testes



```

Blek@BDO10: ~/docs/Faculdade/Atual/SB/sb-tps
{ sb-tps } master » sb-assembler/bin/montador.exe sb-assembler/tst/tspec.i sb-assembler/tst/tspec.o
v
[Symbol table]
Label: address
A: 30
L: 23
MB: 20
{ sb-tps } master » sb-emulator/build/emulador.exe 0 1000 0 sb-assembler/tst/tspec.o
30
55
30
55
55
{ sb-tps } master » sb-emulator/build/emulador.exe 0 1000 0 sb-assembler/tst/tspec.o
-12
-15
-12
-15
-12
{ sb-tps } master » |
~/docs/Faculdade/Atual/SB/sb-tps

```

Figura 1: Montagem em modo verboso e execução do teste `tspec.i`.

```
Blek@BDO10: ~/docs/Faculdade/Atual/SB/sb-tps
{ sb-tps } master » sb-assembler/bin/montador.exe sb-assembler/tst/tmdn.i sb-assembler/tst/tmdn.o v
[Symbol table]
Label: address
SORT_INNER_LOOP: 137
LOADR_PC_OFFSET: 58
SORT_OUTER_LOOP: 120
STORER_POS: 96
SWAP_END: 182
READ_LOOP: 12
ARR_POS: 55
STORER_PC_OFFSET: 59
ARR_SIZE: 56
STORER: 81
ONE: 54
LOADR_POS: 75
MEDIAN_INDEX: 57
SORT_INNER_LOOP_END: 187
SORT_OUTER_LOOP_END: 192
SORT: 102
READ_LOOP_END: 32
LOADR: 60
{ sb-tps } master » sb-emulator/build/emulador.exe 0 1000 0 sb-assembler/tst/tmdn.o
77 44 33 22 11 55 66
44
{ sb-tps } master » sb-emulator/build/emulador.exe 0 1000 0 sb-assembler/tst/tmdn.o
1 2 3 5 6 7 4
4
{ sb-tps } master » |
~/docs/Faculdade/Atual/SB/sb-tps
```

Figura 2: Montagem em modo verboso e execução do teste tmdn.i.

```
Blek@BDO10: ~/docs/Faculdade/Atual/SB/sb-tps
{ sb-tps } master » sb-assembler/bin/montador.exe sb-assembler/tst/texp.i sb-assembler/tst/texp.o v
[Symbol table]
Label: address
ZERO: 75
MULLP: 64
MUL: 34
END: 31
BOTHN: 53
FSTN: 59
ONE: 76
SNDN: 47
MULE: 74
LOOP: 13
{ sb-tps } master » sb-emulator/build/emulador.exe 0 1000 0 sb-assembler/tst/texp.o
27 3
19683
{ sb-tps } master » sb-emulator/build/emulador.exe 0 1000 0 sb-assembler/tst/texp.o
3 4
81
{ sb-tps } master » sb-emulator/build/emulador.exe 0 1000 0 sb-assembler/tst/texp.o
2 30
1073741824
{ sb-tps } master » sb-emulator/build/emulador.exe 0 1000 0 sb-assembler/tst/texp.o
2 31
-2147483648
{ sb-tps } master »
~/docs/Faculdade/Atual/SB/sb-tps
```

Figura 3: Montagem em modo verboso e execução do teste texp.i.

```
Blek@BDO10: ~/docs/Faculdade/atual/SB/sb-tps/sb-assembler
{ sb-assembler } master » make tests
mkdir bin//test-bin/
gcc -std=c99 -g -pg -Wall -Wextra -Werror -Isrc/ -o bin//test-bin//bucket tst/unit-tests//bucket.c s
rc/bucket.c
gcc -std=c99 -g -pg -Wall -Wextra -Werror -Isrc/ -o bin//test-bin//map tst/unit-tests//map.c src/map
.c src/bucket.c
gcc -std=c99 -g -pg -Wall -Wextra -Werror -Isrc/ -o bin//test-bin//pseudo_instr tst/unit-tests//pseu
do_instr.c src/pseudo_instr.c
gcc -std=c99 -g -pg -Wall -Wextra -Werror -Isrc/ -o bin//test-bin//real_md tst/unit-tests//real_md.c
src/real_md.c
gcc -std=c99 -g -pg -Wall -Wextra -Werror -Isrc/ -o bin//test-bin//idt tst/unit-tests//idt.c src/map
.c src/bucket.c src/pseudo_instr.c src/real_md.c src/idt.c src/hash_aux.c
{ sb-assembler } master » bin/test-bin/bucket.exe ~/docs/Faculdade/atual/SB/sb-tps/sb-assembler
All tests passed successfully.
{ sb-assembler } master » bin/test-bin/idt.exe ~/docs/Faculdade/atual/SB/sb-tps/sb-assembler
14
All tests passed successfully.
{ sb-assembler } master » bin/test-bin/map.exe ~/docs/Faculdade/atual/SB/sb-tps/sb-assembler
All tests passed successfully.
{ sb-assembler } master » bin/test-bin/pseudo_instr.exe
33022033
All tests passed successfully.
{ sb-assembler } master » bin/test-bin/real_md.exe ~/docs/Faculdade/atual/SB/sb-tps/sb-assembler
All tests passed successfully.
{ sb-assembler } master » |
~/docs/Faculdade/atual/SB/sb-tps/sb-assembler
```

Figura 4: Compilação e execução dos testes unitários providenciados.