Software Básico - Trabalho 1

Matheus Vieira Portela - 10/0017959
matheus.v.portela@gmail.com

Lucas de Levy Oliveira - 10/0015248
lucasdelevy@gmail.com

8 de maio de 2014

1 Estrutura

Este projeto implementa um montador e simulador para a linguagem assembly didática. Para tal fim, escolheu-se utilizar um pré-processador de duas passagens e montador de passagem única.

Tabelas foram implementadas por meio de *hash tables* com resolução de colisões por listas encadeadas.

2 Organização

O projeto está dividido em diretórios de acordo com a seguinte organização:

- `asm`: Arquivos-fonte e cabeçalho para o montador;
- `bin`: Arquivos executáveis para o montador e o simulador;
- `sim`: Arquivos-fonte para o simulador;
- `test`: Arquivos-fonte em assembly didático para fins de teste.

3 Compilação

O projeto utiliza *GNU Make* para compilação automática. Para tanto, basta executar o comando

```
$ make
```

que irá invocar *Makefile* recursivos nos diretórios `asm` e `sim` para compilar tanto o montador quanto o simulador automaticamente.

4 Utilização

Uma vez gerado os arquivos executáveis, que estarão disponíveis no diretório `bin`, pode-se utilizá-los para montar ou simular arquivos.

4.1 Montagem

Para a montagem, utiliza-se a seguinte sintaxe:

```
Usage: assembler <input> <preprocessing> <output>
```

Portanto, caso estejamos no diretório raiz do projeto e queiramos montar o arquivo `fibonacci.asm` no diretório de arquivos-teste, utiliza-se o seguinte comando:

```
$ ./bin/assembler test/fibonacci.asm fibonacci.pre fibonacci.obj
```

gerando o arquivo `fibonacci.pre` após o pré-processamento e o arquivo `fibonacci.obj` com código-objeto no diretório raiz.

4.2 Simulação

Uma vez compilado, um arquivo-objeto pode ser simulado utilizando o simulador, cuja sintaxe é:

```
Usage: simulator <input>
```

Portanto, caso estejamos no diretório raiz do projeto e queiramos simular o arquivo-objeto`fibonacci.obj`, utiliza-se o seguinte comando:

```
$ ./bin/simulator fibonacci.obj
```

5 Testes

O diretório `test` possui códigos-fonte de arquivos em assembly para fins de teste, demonstrando montagens bem-sucedidas e possíveis erros de simulação e execução.

Estão presentes os seguintes arquivos e suas utilizações:

5.1 Testes de programas

- `array.asm`: Lê uma array de 4 elementos da entrada padrão e os imprime multiplicados por 3.
- `fibonacci.asm`: Imprime a sequência de Fibonacci até o número limite lido por input de teclado.
- `triangle.asm`: Calcula a área de um triângulo baseado no valor da base e da altura obtidos pela entrada padrão.

5.2 Testes de erro

- `error_calculate_with_text.asm`: Erro por utilizar label de código, ao invés de dados, com uma instrução aritmética;
- `error_conditional_jump.asm`: Erro por realizar um salto condicional para a seção de dados;
- `error_data_in_text.asm`: Erro por definir uma variável de dados na seção de código;
- `error_division_by_zero.asm`: Erro por dividir por uma constante inicializada com o valor zero;
- `error_double_definition.asm`: Erro por redefinir uma variável;
- `error_invalid_instruction.asm`: Erro por utilizar uma instrução inválida;
- `error_invalid_token.asm`: Erro por definir uma label inválida;
- `error_malformed_directive.asm`: Erro por utilizar uma diretiva indevidamente;
- `error_malformed_instruction.asm`: Erro por utilizar uma instrução indevidamente;
- `error_missing_data_section.asm`: Erro por ausência da seção de dados;
- `error_missing_definition.asm`: Erro por ausência da definição da label;
- `error_missing_text_section.asm`: Erro por ausência da seção de código;
- `error_text_in_data.asm`: Erro por utilizar instrução na seção de dados;
- `error_unconditional_jump.asm`: Erro por realizar um saldo incondicional para a seção de dados;
- `error_write_at_const.asm`: Erro por escrever em um espaço de memória reservado para constante.