Trabalho Prático 0
Software Básico
Assembler Simples

João Paulo Martino Bregunci - jpbregunci@gmail.com
Ronald Davi Rodrigues Pereira - ronald.drp11@gmail.com

Outubro de 2016

1 Introdução

A tradução de cógido *Assembly* para código de máquina é um dos problemas mais antigos da Ciência da Computação, dada a obvia dificuldade da programação diretamente em código de máquina. Desse problema, iniciou-se a proposta desse trabalho que será a construção de um *Assembler* simples, o qual é capaz de gerar código executavel para a máquina *Wombat2*. Este dado assembler em questão criado possui uma esquemática classica de *2 passadas*, a fim de resolver qualquer tipo de referência posterior em instruções de desvio. Ao final de todo o processo, cria-se um novo arquivo executável de saída que possa ser diretamente executado pela máquina. Vale ressaltar que todas as instruções de máquina possuem, nesta implementação, uma paridade com cada *pseudo-instrução* de máquina.

2 Explanação do Problema

2.1 Apresentação do Problema

Como enunciado anteriormente na Introdução, o problema consiste na implementação de *Assembler* simples. Sendo o maior problema relevante a tal a existência de referências futuras, ou seja, saltos na memória de instrução, cujo endereço ainda é desconheccido.

2.2 Solução Generalizada do Problema

Para resolver o prinicipal problema inerente, a existência de referências futuras, optou-se pela implementação de um *Assembler de 2 passadas*, utilizando para tal uma *Tabela de símbolos*, na qual armazena-se o endereço relativo de

todos os procedimentos existentes no código, a fim de, na segunda passada, resolver tais referências futuras. A imagem a seguir auxília na compreensão do *Assembler*:

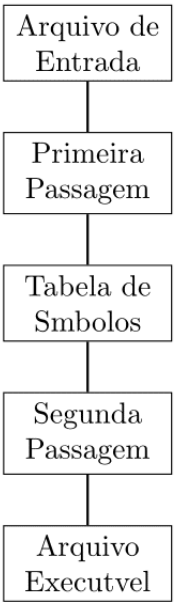

Figure 1: Fluxograma com os passos do *Assembler*

O arquivo de entrada é representado acima pelo primeiro nodo e em cima de tal realiza-se a primeira passada por todo arquivo pelo *Assembler*. Nessa passagem somente geram-se duas tabelas de referências, uma para os *.data* e outra para a posição de todos os *labels* no código, contando o PC *program counter*, a partir do início do arquivo. Essa tabela é posteriormente utilizada para resolver todas as referências posteriores existentes, realizando assim a segunda passada. Ao final de todo esse processo é gerado o arquivo executável, que pode ser lido diretamente pela máquina.

2.3 Implementação das Passagens

Na primeira passagem há a criação de duas listas encadeadas, as quais são as duas tabelas de símbolos. Uma dessas tabelas contem todos os *.data*, que serão prontamente substituidos e a outra tabela contem referências para os PC (*program counter*) de labels encontradas na primeira passagem do *Assembler*. Escolheram-se listas encadeadas para a estruturação da Tabela de Símbolos, pois não existem problemas eventuais na procura de elementos da lista em *O(n)*, já que o número de *labels* e *.data* não prejudica severamente a performance do código. Depois disso se realiza o processo corriqueiro de decodificação por meio

de uma série de *if, else if e else* para a leitura da instrução e a tradução do valores de eventuais registradores para os respectivos valores binários. Após todo esse processo se encontra o arquivo executável gerado.

3 Casos de Teste

Dois casos de teste foram formulados para o teste do montador, cada qual procurando demonstrar, testar e utilizar de comandos peculiares da arquitetura da máquina *Wombat2*.

O primeiro programa realiza a busca do maior de três valores distintos e retorna tal valor ao quadrado acrescido de 1. Esse caso de teste é muito interessante, porque na implementação realizada utilizaram-se vários diferentes tipos de, cuja lista é: [*loadi, subtract, move, jump, jmpp, jmpn, storei, exit, multiply, clear, addi, add*].

O segundo prorama emula muita das caracteristicas do primeiro, mas esse recebe 3 inteiros e retorna a média aritmetica de todos os elementos, sendo que elementos repetidos são contados somente uma vez. Algumas operações aqui foram testadas que não foram utilizadas no teste1 e essas foram:[*loadc, seq, divide, sgt, slt, call*].

No total entre os dois arquivos de teste foram testadas 18 instruções das 26 possíveis relativas a máquina *Wombat2*. A codificação destas instruções para línguagem de máquina foi sucedida e externou o êxito do *Assembler* aqui criado. Abaixo estão algums imagens dos casos testes rodando nativamente em uma máquina *Wombat2*.

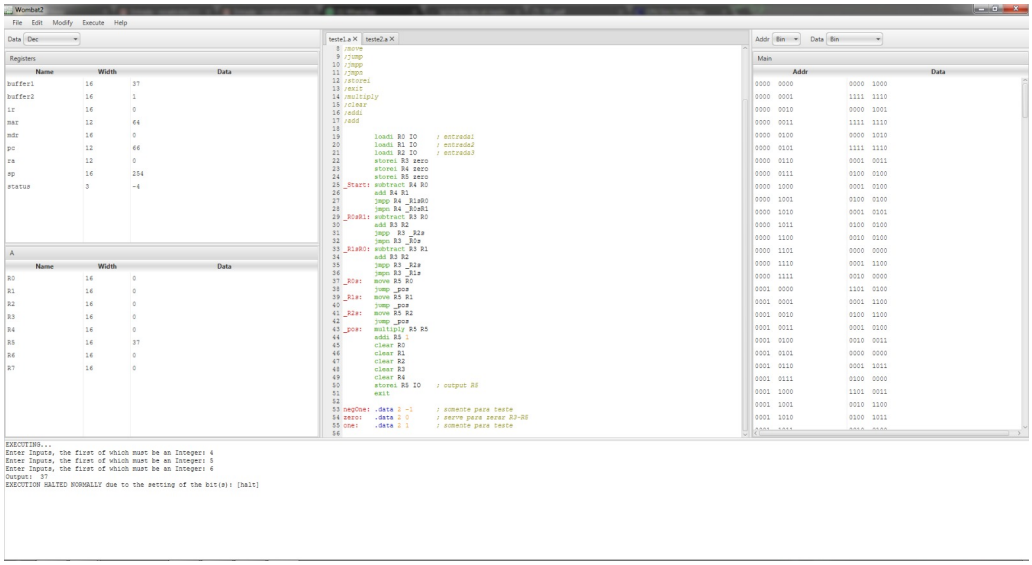

Figure 2: Funcionamento do primeiro caso de teste

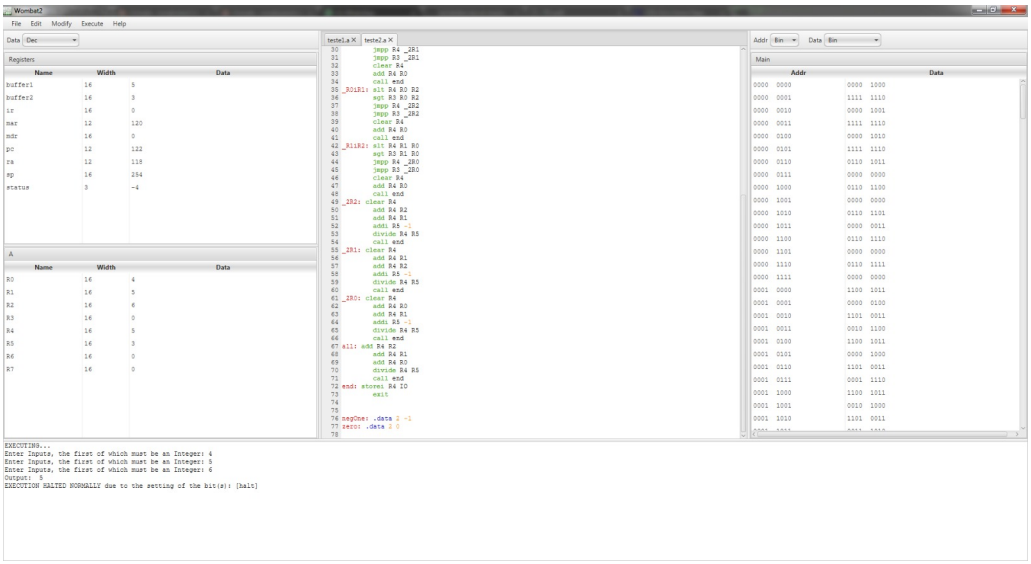

Figure 3: Funcionamento do segundo caso de teste

Para o leitor que desejar chegar tais testes ele pode acessar a pasta contendo os testes, a fim de comparar as saídas do montador implementado. Os arquivos *answers.mif* possuem a saída gerada pelo montador da própria máquina, os arquivos *output.mif* possuem as saídas geradas pelo montador aqui desenvolvido e os arquivos de *teste.a* são os programas em questão desenvolvidos para o teste do montador.

4 Conclusões

Por meio desse trabalho foi possível desenvolver sucessivamente um montador de duas passadas que gera código executável para a máquina *Wombat2*. Contudo, o maior ganho desse projeto não veio na simples elaboração desse *Assembler*, mas sim na oportunidade de expandir o horizonte de conhecimento sobre o funcionamento de montadores e sobre as técnicas de programação voltadas a construção desses.

References

[1] Colby College, Official CPUsim Website
`http://www.cs.colby.edu/djskrien/CPUSim/`