UFMG

Trabalho Prático
Montador de Assembly
- Software Básico -

Gustavo Vieira Maia | 2013030627
Lucas Renan Vieira de Freitas Pereira | 2016006689

15 de maio de 2017

Introdução

O proceso de montagem de um código em Assembly consiste em traduzir o código alfa numérico para linguagem de máquina ou código binário.

Infelizmente a tradução imediata nem sempre é possível dado que o código a ser traduzido pode ter referências a regiões do código posteriores àquela que o montador já traduziu.

A estratégia escolhida para a montagem nesse trabalho foi a de montar o código em dois passos, o primeiro passo consiste em gerar um código intermediário ainda incompleto sem tratar as referências posteriores e em seguida, no segundo passo, editar as referências desconhecidas não tratadas de início.

Implementação

A implementação do trabalho foi feita utilizando C++, por ser de maior domínio entre os integrantes do grupo.

Como dito na introdução, na primeira passada pelas instruções de um código, as referências anteriores e as instruções que não dependem de outras instruções são montadas de prontidão. As demais instruções são montada parcialmente, deixando as referências posteriores para serem tratadas depois. Isso garante com que a montagem seja mais eficiente, já que o segundo passo não passará, necessariamente, por todas as instruções.

A memória da máquina cujo montador foi implementado tem 256 posições de memória, sendo 2 dessas direcionadas exclusivamente a Entrada/Saída. Assim, como só há funções que alocam regiões de memória e não há funções que liberam, foi decidido que a política de gerenciamento de memória seria feita da seguinte maneira: as instruções são alocadas a partir da posição 0 de memória, de forma crescente e as regiões alocadas pela pseudo-instrução “.data” são alocadas a partir da posição 253, de forma decrescente. Assim, no pior caso, toda região de memória é utilizada, sem nenhuma posição deixada para trás.

Além disso, o paradigma binário utilizado foi Little Endian. Em outras palavras, os bits mais significativos de uma palavra aparecem antes dos menos significativos. Isso é válido tanto para as instruções como para as alocações.

Os registradores A0, A1, A2, A3, A4, A5, A6, A7 foram mapeados para os inteiros de 3 bits 0, 1, 2, 3, 4, 5, 6, 7, respectivamente.

Testes

testFile0.a:
O código de teste 0 implementa uma lógica de programa muito simples para teste geral do montador. O programa pede por duas entradas de dois números inteiros do usuário (A, B), se $A < B$ o programa retorna a soma dos dois inteiros ($A + B$), se $A \geq B$ o programa retorna a diferença dos dois inteiros ($A - B$). O programa usa diversas instruções inclusive referências posteriores.

testFile1.a:
O código de teste 1 não segue nenhuma lógica útil, mas possui todas as intruções do conjunto de instruções da arquitetura de modo a forçar o montador a montar pelo menos uma vez cada uma das instruções. O programa possui diversas intruções de output, mas para teste das instruções de jump essas instruções de output devem ser ignoradas durante a execução exceto por uma que retorna o número 123 para indicar o final da execução.

Conclusão

Após feitos os testes com os dois diferentes arquivos de teste nota-se que o funcionamento do montador feito está dentro do esperado. Os arquivos de saída foram testado no CPUSim e o resultado foi semelhante ao resultado quando traduzido pelo CPUSim.

A única diferença entre as duas montagens é que a região da memória definida para alocação de memória usando a pseudo-instrução $.data$ é diferente, dado que as políticas de alocação de memória são diferentes.

Portanto, conclui-se que a implementação do montador foi bem sucedida, cumprindo o papel de contribuir na fixação e no entendimento da matéria por parte dos alunos.