Trabalho Prático 1: Montador

Software Básico - DCC008

Guilherme Resende Vieira - 2015004178 / Lucas Fonseca Mundim - 2015042134 / Pedro Nascimento Costa - 2015083388

1 Introdução

O trabalho tem por objetivo construir um montador para máquina *'Swombat'*, para tal, deve-se usar linguagem C ou $C++$, optou-se por C pelo costume maior em utilizá-la. Para a construção do montador, ou seja, a execução do trabalho foi decidido usar a montagem em dois passos, isso é feito devido ao problema de *'Forward jumping'*, que aconteceria caso contrário. Na primeira monta-se uma lista contendo todos os labels encontrados, na segunda, é feita uma leitura e tradução linha por linha do programa, convertendo para binário.

No desenvolvimento, decidiu-se tratar caso a caso durante a tradução para binário, inclusive, caso o programa encontre um número negativo, foi decidido utilizar complemento de dois para representá-los.

A entrada do programa é um arquivo contendo um código em assembly, seguindo algumas regras estabelecidas pela proposta do trabalho, como $.data$ estar sempre no final e registradores serem representados pela letra R ou A.

A saída é um arquivo de extensão mif contendo a tradução. Como a memória é estruturada de modo a armazenar palavras de tamanho 8 (bits) em cada endereço, representação é dada de 8 em 8 bits. As instruções da máquina *Swombat* são expressas na forma de 16 bits, portanto o PC(Program Counter) é normalmente incrementado de 2 em 2 para cada nova instrução.

2 Solução do Problema

Para a resolução do problema, optou-se pela criação de uma lista, que seria capaz de armazenar todas as *labels* presentes no código de entrada, emparelhadas com seu respectivo endereço de memória, dado pelo valor do PC relativo à linha em que a *label* se encontra. Essa lista é preenchida durante o primeiro passo de montagem, que consiste em fazer uma leitura completa do código de entrada buscando apenas as atribuições de *labels* iniciadas com um *underscore* e registrando-as na lista como descrito anteriormente.

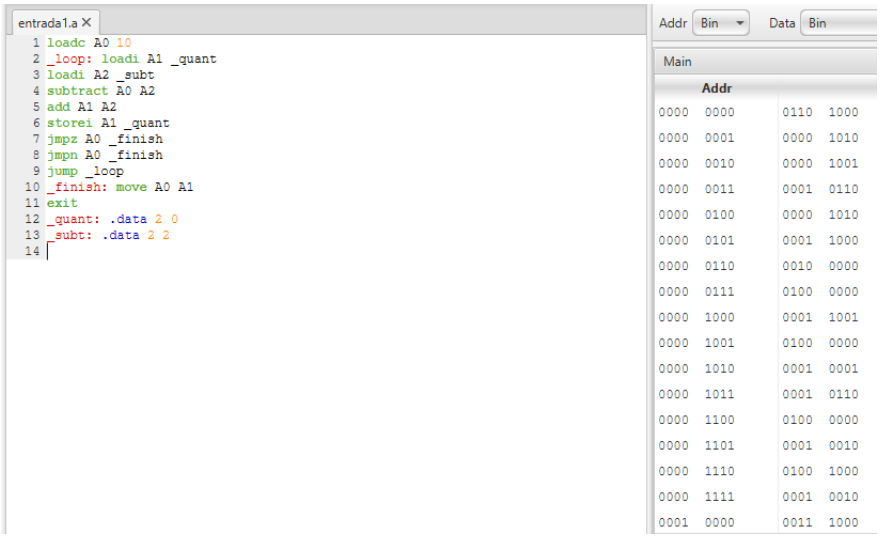
Após a captura de todos os endereços das *labels* na primeira passada, o montador faz uma segunda leitura traduzindo linha por linha do programa *assembly* em uma palavra de 16 bits. A tradução funciona, essencialmente, da seguinte forma: o programa lê caractere a caractere do arquivo *assembly* de entrada e procura por alguma instrução. Ao encontrar uma instrução, o programa interpreta qual foi encontrada e realiza as conversões necessárias de binário para retornar ao final a linha de código no formato de 16 bits, que é dividida em 2 palavras de 8 bits para se adequar à estrutura da memória e então são imediatamente registradas no arquivo de saída. Casos especiais ocorrem ao encontro de *labels*, em que o programa consulta a lista obter o endereço necessário, ao encontro da expressão *"IO"*, que faz referencia ao endereço destinado a entrada e saida de dados (posição 254 da memória), e ao encontro da pseudo instrução .data, que reserva uma determinada quantidade de bytes na memória para armazenar um número inteiro (neste caso o PC é incrementado de acordo com o número de bytes alocados).

3 Avaliação Experimental

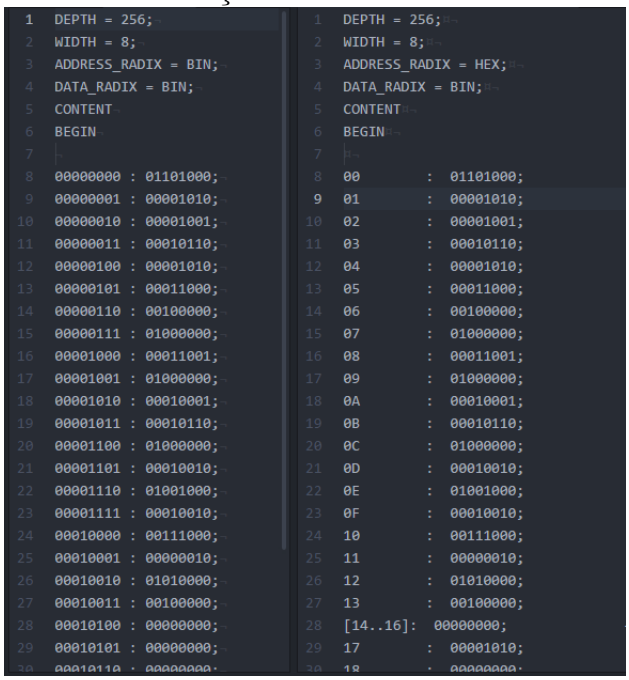
Para a avaliação experimental, foram criados dois programas simples em *Assembly* para que fosse feita a comparação entre os *outputs* gerados pelo *CPUSim* e pelo programa criado para o trabalho. Um pequeno detalhe que vale ser notado é que, por mais que o *CPUSim* permita a visualização da RAM com endereçamento em binário, ao exportá-la para um arquivo externo, só foi possível obter endereçamento em hexadecimal. Contudo, o referencial ainda se mantém igual. Além disso, o CPUSim coloca, em comentários, a que a linha binária impressa se refere em *assembly*, algo que não deveria ser trabalhado.

3.1 tst/entrada1.a

O *tst/entrada1.a* é um programa simples, transfere o numero 10, em intervalos de 2 em 2 de um registrador para outro, da forma: *(10,0); (8,2); (6,4); (4,6); (2,8); (0,10)*. E no final o registrador zerado recebe o valor cheio novamente.



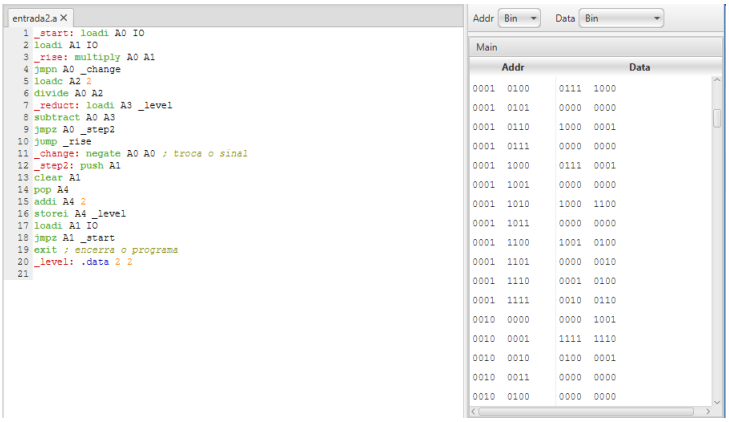
Simulação feita no CPUSim



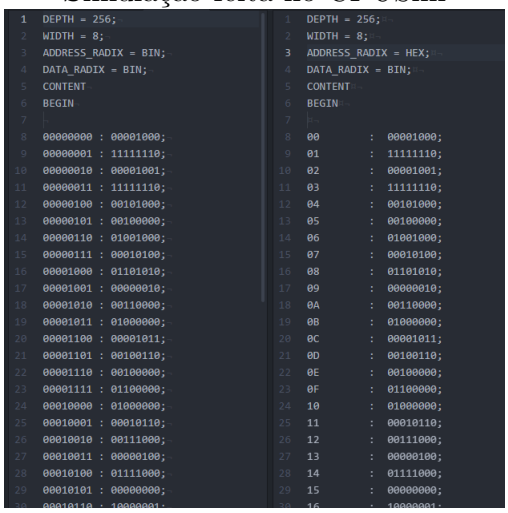
Comparação de Outputs, do programa à esquerda e do CPUSim à direita

3.2 tst/entrada2.a

A *entrada2.a* recebe dois números A e B de entrada, multiplicando o valor A por B ($A = A * B$). O resultado é dividido por 2 e então é decrementado pela variável *"_level"* ($A = (A/2) - \_level$), caso o resultado for diferente de 0, ele é multiplicado por B e o processo se repete até obter 0. Quando isso ocorre, B se torna o novo valor de *"_level"* e o programa pode começar um novo procedimento caso recebe 0 de entrada.



Simulação feita no CPUSim



Comparação de Outputs, do programa à esquerda e do CPUSim à direita

Como pode ser observado, o arquivo de saída obtido pelo programa feito é igual ao produzido pelo CPUSim, com exceção do que foi observado na introdução da seção. Nota-se, inclusive, o *tag ADDRESS_RADIX = HEX* ao topo do *output* do CPUSim. As capturas de tela com todos os componentes do CPUSim podem ser encontradas em *doc/CPUSimEntrada1* e *doc/CPUSimEntrada2*.

4 Conclusão

Este trabalho possibilitou o desenvolvimento de um montador para a máquina *Swombat*. O principal benefício proporcionado pelo projeto foi a oportunidade de trabalhar entre 2 camadas de níveis computacionais, e assim poder expandir a compreensão do funcionamento de um processador.