Trabalho Prático 1:
Implementação de um montador

Lucas Peixoto, Rafael Grandsire

Software Básico – 2016/2

1 Introdução

Este trabalho simula o montador de uma máquina Wombat2 escrito em C++, com auxílio de suas bibliotecas padrão.

2 Decisões do Projeto

Para facilitar o desenvolvimento do programa, foram criados quatro mapas e um vetor de inteiros:

- O primeiro mapa associa a um label um inteiro que representa o PC da instrução que este referencia;
- O segundo associa a um opCode um inteiro que serve como identificador único de uma instrução definido no anexo da especificação;
- O terceiro associa uma string que representa um registrador a um inteiro que é o próprio índice do registrador;
- O último mapa associa a uma string um tipo *mem*, que descreve uma posição de memória alocada por *.data* interpreta-se a string "IO" como um *mem*, o que não gera ambiguidade, uma vez que "IO" é uma string reservada e não pode ser usada como label);
- O vetor de inteiros associa cada identificador de instrução a sua classe definida na Tabela 2.

No inicio da execução do programa, o vetor que classifica instruções e os mapas dos opCodes e registradores são preenchidos. Feito isso, para processar uma instrução, é necessário associar seu nome a um identificador e seu identificador a uma classe e, a partir daí, imprimi-la no formato definido na especificação. Após o preenchimento destas estruturas, o programa está pronto para começar o pré-processamento do arquivo de entrada.

2.1 Entrada e Saída

Para o funcionamento da aplicação, espera-se que o nome do arquivo de entrada e saída sejam recebidos por via de parâmetro.

```
./sb.exe entrada.a saida.mif
```

Como entrada, espera-se obter um arquivo assembly de extensão *.a*. A saída será fornecida em um arquivo *.mif* (formato escolhido para descrever o estado inicial da memória).

2.2 Tipos de Instrução

Todas as instruções previstas pela descrição do trabalho foram classificadas de acordo com seu tipo de entrada. Tenha como exemplo as instruções *exit* e *ret*: ambas não possuem argumentos e, portanto, foram classificadas como do mesmo tipo. As tabelas a seguir explicitam a classificação das instruções:

Classe	**Tipo**
0	Não recebe parâmetro.
1	Recebe 2 parâmetros. O primeiro deve ser um registrador e o segundo deve ser espaço de memória, rótulo ou constante.
2	Recebe 2 parâmetros. Todos eles deve ser registradores.
3	Recebe 3 parâmetros. Todos eles devem ser registradores.
4	Recebe 1 parâmetro. Deve ser um rótulo ou uma constante.

Tabela 1: tipos de instrução

Instrução	**Classe**		**Instrução**	**Classe**		**Instrução**	**Classe**
exit	0		loadi	1		storei	1
add	2		subtract	2		multiply	2
divide	2		jump	4		jumpz	1
jmpn	1		move	2		load	2
store	2		loadc	1		clear	1
moveSp	4		slt	3		call	4
loadSp	1		storeSp	1		ret	0
loadRa	4		storeRa	4		addi	1
sgt	3		seq	3		jmpp	1

Tabela 2: classificação das instruções suportadas

2.3 Politica de alocação na memória

O tratamento da pseudo-instrução *.data* foi feita do mesmo modo implementado no CPUSim: a alocação é feita exatamente na posição onde a pseudo-instrução foi chamada. Em outras palavras, se houver um *.data* antes do *exit*, a escrita do dado solicitado é feito entre as instruções do programa.

2.4 Pré-Processamento

No pré-processamento, são calculados os PCs finais de cada rótulo. O arquivo é lido linha por linha e o PC é devidamente acrescido a cada instrução e pseudo-instrução identificadas. Quando um rótulo é encontrado, salva-se seu nome e PC no mapa reservado para isso.

2.5 Processamento

Quando todo o arquivo de entrada estiver sido pré-processado a leitura é reiniciada, desta vez processando cada instrução e imprimindo-a no arquivo de saída. Sempre que uma instrução do tipo 1 ou 4 tiver um label como parâmetro, este será substituído pelo PC equivalente salvo no mapa.

3 Descrição dos Testes

3.1 test1.a

Programa que calcula o fatorial de um inteiro positivo. A primeira entrada recebida é a quantidade N de fatoriais a serem serem computados, seguida de N inteiros x. Como os registradores da Wombat2 só suportam inteiros de 16 bits, o maior x aceito é 7 (pois $8! = 40320 > 32767 = 2^(15) - 1$).

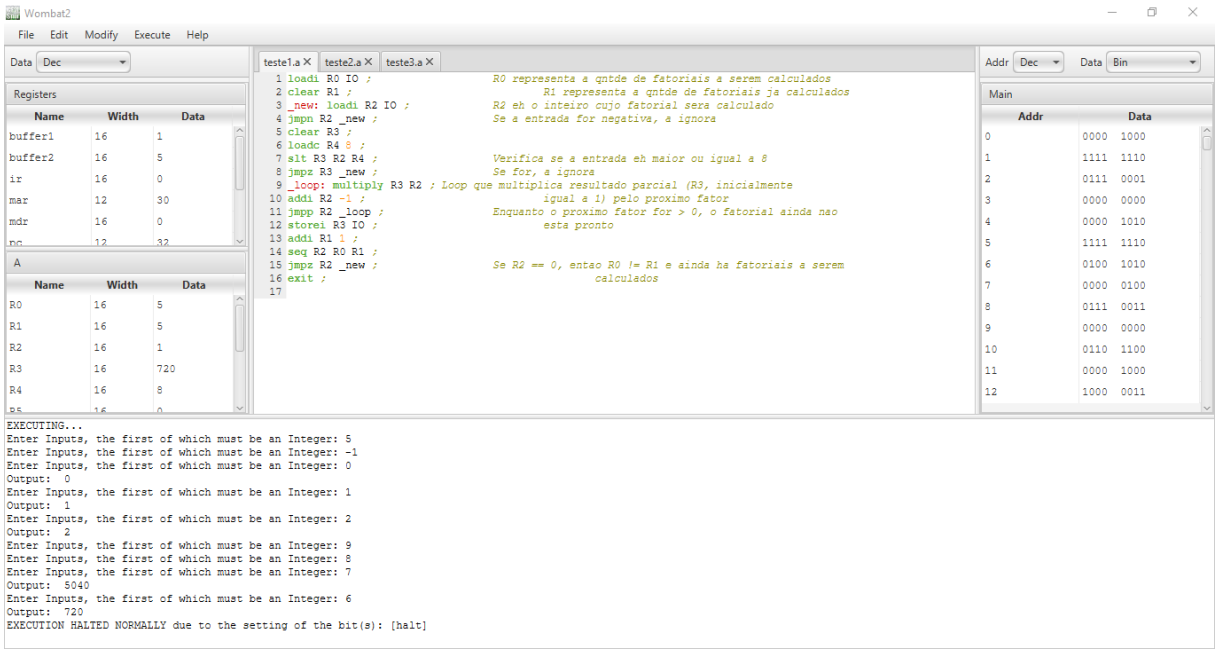

Figura 1: screenshot da execução de teste1.a no CPUSim

Note que quando o x não pertence ao intervalo $[0, 7]$, o programa para até que outro valor seja recebido.

3.2 teste2.a

Programa que calcula o resultado de uma equação de segundo grau da forma $a \cdot x^2 + b \cdot x + c$. As primeiras entradas devem especificar a, b e c, e as entradas seguintes, enquanto não nulas, indicam valores de x. Todos os resultados calculados são armazenados na pilha que, após o fim das entradas, é ordenada pelo algoritmo bubble sort. Quando a ordenação termina, a pilha é impressa em ordem crescente.

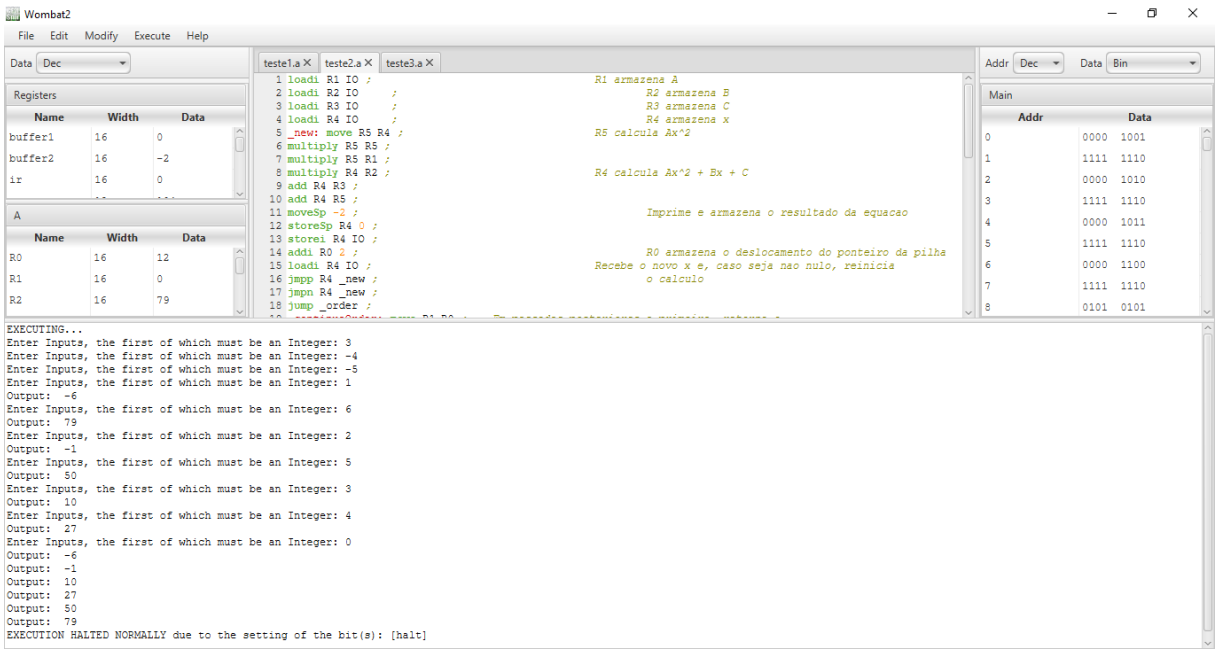

Figura 2: screenshot da execução de teste2.a no CPUSim

3.3 teste3.a

Programa que recebe uma quantidade arbitrária de inteiros inserindo-os na pilha e procura dentre estes um valor especifico. São recebidos e empilhados inteiros até que o valor -1 seja identificado, o que indica o fim do empilhamento. A seguir é recebido um inteiro x e, se este estiver na pilha, é impresso (caso contrário imprime-se -1).

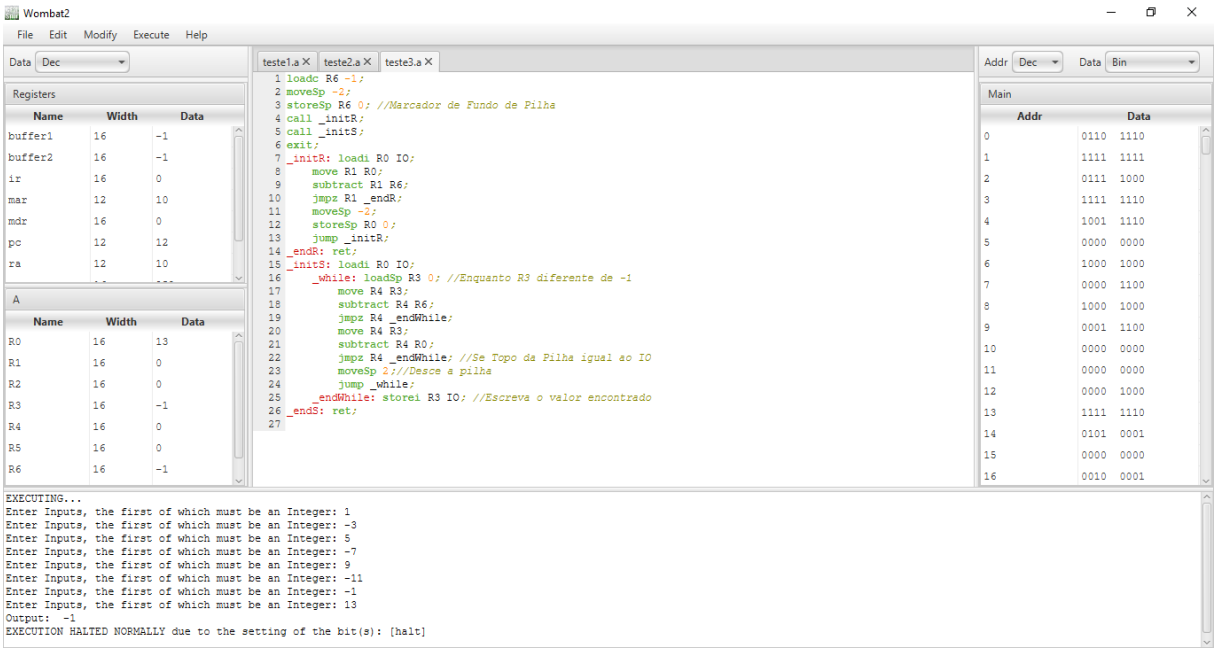

Figura 3: screenshot 1 da execução de teste3.a no CPUSim

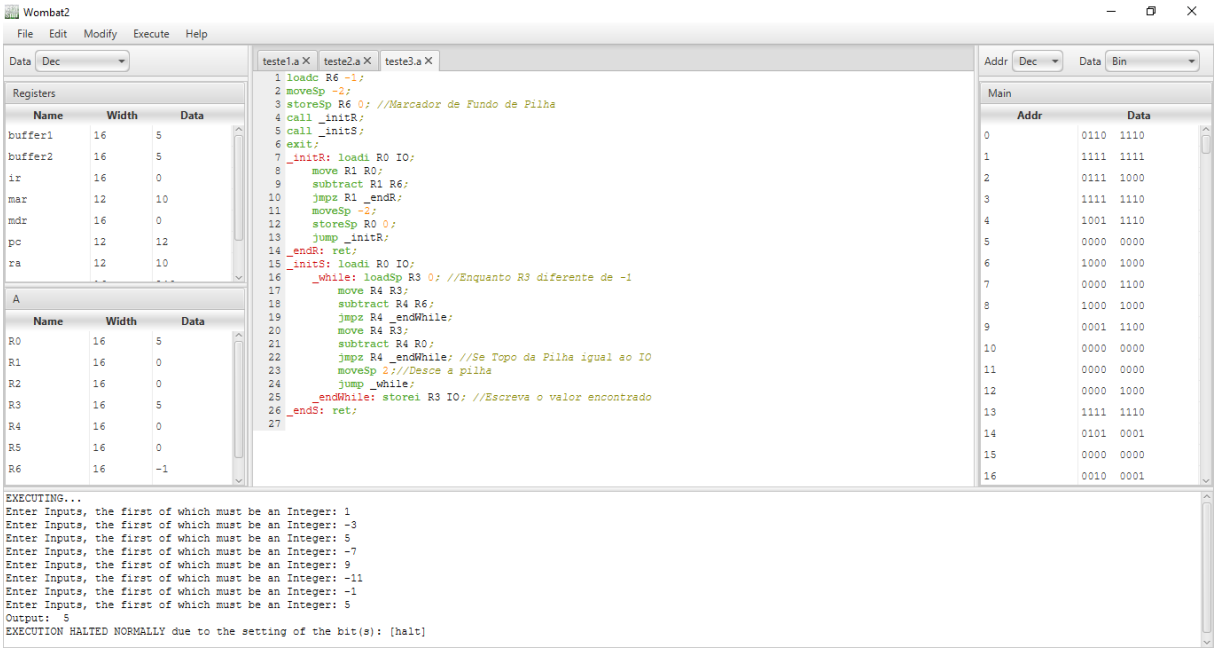

Figura 4: screenshot 2 da execução de teste3.a no CPUSim