

# 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 saída 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.

entrads1.a X	Addr	Bin	Data	Bin
1 loadc A0 10	Main			
2 _loop: loadi A1 _quant	Addr			
3 loadi A2 _subt	0000	0000	0110	1000
4 subtract A0 A2	0000	0001	0000	1010
5 add A1 A2	0000	0010	0000	1001
6 storei A1 _quant	0000	0011	0001	0110
7 jmpz A0 _finish	0000	0100	0000	1010
8 jmpn A0 _finish	0000	0101	0001	1000
9 jump _loop	0000	0110	0010	0000
10 _finish: move A0 A1	0000	0111	0100	0000
11 exit	0000	1000	0001	1001
12 _quant: .data 2 0	0000	1001	0100	0000
13 _subt: .data 2 2	0000	1010	0001	0001
14	0000	1011	0001	0010
	0000	1100	0100	0000
	0000	1101	0001	0010
	0000	1110	0100	1000
	0000	1111	0001	0010
	0001	0000	0011	1000

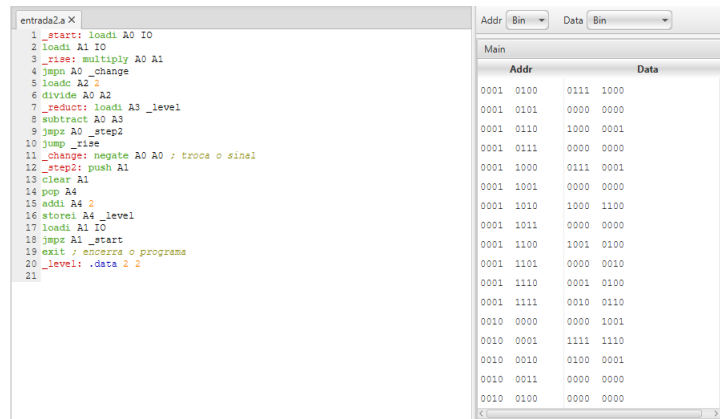
### Simulação feita no CPUSim

1 DEPTH = 256;	1 DEPTH = 256;
2 WIDTH = 8;	2 WIDTH = 8;
3 ADDRESS_RADIX = BIN;	3 ADDRESS_RADIX = HEX;
4 DATA_RADIX = BIN;	4 DATA_RADIX = BIN;
5 CONTENT	5 CONTENT
6 BEGIN	6 BEGIN
7	7
8 00000000 : 01101000;	8 00 : 01101000;
9 00000001 : 00001010;	9 01 : 00001010;
10 00000010 : 00001001;	10 02 : 00001001;
11 00000011 : 00010110;	11 03 : 00010110;
12 00000100 : 00001010;	12 04 : 00001010;
13 00000101 : 00011000;	13 05 : 00011000;
14 00000110 : 00100000;	14 06 : 00100000;
15 00000111 : 01000000;	15 07 : 01000000;
16 00001000 : 00011001;	16 08 : 00011001;
17 00001001 : 01000000;	17 09 : 01000000;
18 00001010 : 00010001;	18 0A : 00010001;
19 00001011 : 00010110;	19 0B : 00010110;
20 00001100 : 01000000;	20 0C : 01000000;
21 00001101 : 00010010;	21 0D : 00010010;
22 00001110 : 01001000;	22 0E : 01001000;
23 00001111 : 00010010;	23 0F : 00010010;
24 00010000 : 00111000;	24 10 : 00111000;
25 00010001 : 00000010;	25 11 : 00000010;
26 00010010 : 01010000;	26 12 : 01010000;
27 00010011 : 00100000;	27 13 : 00100000;
28 00010100 : 00000000;	28 [14..16]: 00000000;
29 00010101 : 00000000;	29 17 : 00001010;
30 00010110 : 00000000;	30 18 : 00000000;

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” ( $A = (A/2) - \text{\_level}$ ), 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

1 DEPTH = 256;	1 DEPTH = 256;
2 WIDTH = 8;	2 WIDTH = 8;
3 ADDRESS_RADIX = BIN;	3 ADDRESS_RADIX = HEX;
4 DATA_RADIX = BIN;	4 DATA_RADIX = BIN;
5 CONTENT	5 CONTENT
6 BEGIN	6 BEGIN
7	7
8 00000000 : 0001000;	8 00 : 00001000;
9 00000001 : 11111110;	9 01 : 11111110;
10 00000010 : 00001001;	10 02 : 00001001;
11 00000011 : 11111110;	11 03 : 11111110;
12 00000100 : 00101000;	12 04 : 00101000;
13 00000101 : 00100000;	13 05 : 00100000;
14 00000110 : 01001000;	14 06 : 01001000;
15 00000111 : 00010100;	15 07 : 00010100;
16 00001000 : 01101010;	16 08 : 01101010;
17 00001001 : 00000010;	17 09 : 00000010;
18 00001010 : 00110000;	18 0A : 00110000;
19 00001011 : 01000000;	19 0B : 01000000;
20 00001100 : 00001011;	20 0C : 00001011;
21 00001101 : 00100110;	21 0D : 00100110;
22 00001110 : 00100000;	22 0E : 00100000;
23 00001111 : 01100000;	23 0F : 01100000;
24 00010000 : 01000000;	24 10 : 01000000;
25 00010001 : 00010110;	25 11 : 00010110;
26 00010010 : 00111000;	26 12 : 00111000;
27 00010011 : 00000100;	27 13 : 00000100;
28 00010100 : 01111000;	28 14 : 01111000;
29 00010101 : 00000000;	29 15 : 00000000;
30 00010110 : 10000001;	30 16 : 10000001;

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.