

# Trabalho Prático 2:

# Montador

## 1. Introdução

O trabalho tem por objetivo criar um montador, que será utilizado pela Máquina Virtual criada no Trabalho prático 1. Esse montador é implementado para fazer a tradução de um conjunto de instruções em Assembly para a linguagem de máquina, ou seja, para um código numérico. O código numérico gerado pode ser utilizado na Máquina Virtual, que receberá o código em sua entrada e executará as instruções do programa.

Além das instruções que serão traduzidas, este trabalho implementa ainda algumas pseudo-instruções, que são utilizadas apenas pelo montador. Esse tipo de instrução facilita o trabalho do montador e tem como objetivo principal permitir ao montador a realização de algumas tarefas extras.

## 2. Implementação

Ao ser executado, o montador deve receber 2 argumentos obrigatórios: arquivo de entrada e arquivo de saída dos dados. O terceiro argumento deve ser o modo de saída dos dados e é opcional. Este terceiro argumento determina se a saída será simples ou em modo “verbose”. Por padrão, o programa realiza a saída simples de dados. Caso o usuário necessite que a saída seja em modo “verbose”, a letra ‘v’ deverá ser passada como terceiro argumento.

A compilação do programa foi feita por um Makefile. Para gerar o arquivo binário executável, basta entrar no diretório que contem o arquivo Makefile pelo terminal e digitar “make”. Assim, um executável chamado “emulador” será criado na pasta *bin*. Quando é iniciado, o montador faz uma validação da entrada e, caso os argumentos obrigatórios não tenham sido passados na execução pelo terminal, o programa será finalizado. Dessa forma, para que o programa seja executado normalmente, é necessário executar o programa pelo terminal com o seguinte comando:

```
./emulador [arg1] [arg2] [arg3]*
```

O montador é composto por dois arquivos .c: “tp2\_montador.c” e “mont\_func.c”. O primeiro conta a função principal, onde é feita o montador. O segundo arquivo, por sua vez, contém a implementação das funções utilizadas no montador. Essas funções estão declaradas no arquivo “mont\_func.h”, que contém ainda a declaração de uma estrutura que agrupa informações a respeito das labels e é utilizada como tipo de dados.

Inicialmente, o montador verifica o número de linhas contidas no arquivo de entrada. Essa verificação é feita para que alguns vetores estáticos possam ser declarados com o tamanho do número de linhas do arquivo. Esses vetores representam as labels, os operadores e os operandos, e receberão em cada posição a informação trazida pela linha que está sendo lida.

A implementação do montador é dividida em duas partes: Passagem Um e Passagem Dois. A primeira tem por objetivo simular a execução da primeira passagem de um montador, onde é verificado todos os rótulos contidos no programa em Assembly e, a partir disso, é montada a **tabela de símbolos**. A tabela agrupa as labels e as posições de memória que elas representam. Dessa forma, o montador pode realizar a tradução do programa sem interrupções posteriormente. Além disso, a tabela permite ao usuário criar programas que utilize uma label como operando de uma instrução contida em uma posição de memória anterior à posição da label.

Na primeira passagem, o montador faz a leitura de toda a informação contida em uma linha. O programa utiliza uma *flag* para representar o conteúdo da linha que foi lida. Caso a linha contenha uma label, a *flag* assume o valor 1. Por outro lado, se a linha está em branco ou contém apenas comentários, a *flag* recebe o valor 2. Por fim, caso a linha contenha uma instrução sem label, a *flag* recebe o valor 3.

Se a linha lida tem label, esta é obtida e alocada em um vetor. Além disso, é verificado também o endereço de memória representado pela label. Ambas informações são alocadas também em um vetor do tipo “símbolos”, que representa a tabela de símbolos. Nessa passagem, o montador também verifica o tamanho de instrução, com a finalidade de determinar o valor do ILC na posição de cada label. Por fim, a passagem um realiza a alocação de memória, caso o operador seja WORD. Assim, o ciclo todo se repete até que sejam lidas todas as linhas do arquivo.

Na passagem dois, o montador faz nova leitura do arquivo de entrada. Nessa segunda leitura, a cada linha ele realiza novamente a checagem do conteúdo e atualiza o valor das *flags*, assim como foi feito na primeira passagem. A principal diferença na leitura do arquivo em relação à Passagem Um é que, desta vez, o montador verifica também o operador e os operandos. Para isso, o programa precisa saber o número de operandos utilizados por cada operador. Com esse objetivo foi criada a função “quant\_operandos”, que verifica o operador e retorna o número de operandos necessários.

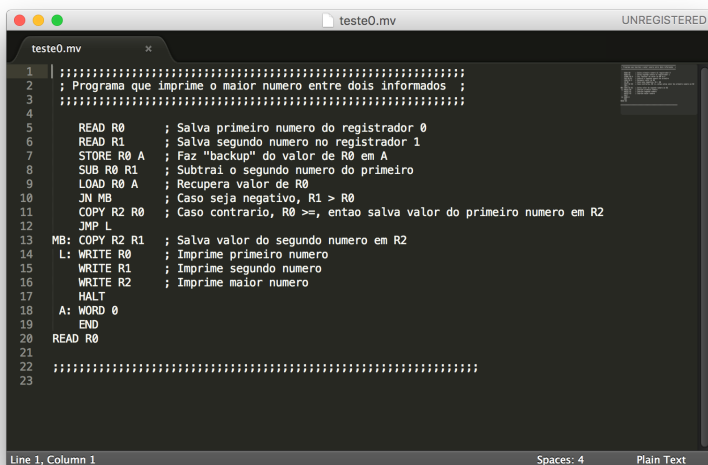
Após verificar o operador e os operandos da linha, o programa utiliza as funções “codigo\_operador” e “codigo\_op” para verificar os códigos numéricos aos quais cada operando ou operador correspondem. Esses códigos são utilizados na impressão dos dados no arquivo de saída. Para imprimir os dados, o programa utiliza a função “imprime\_saida”. Essa função recebe os códigos numéricos e os armazena, um por um, em cada linha do arquivo de saída. Com a impressão dos dados no arquivo de saída, a Passagem Dois é finalizada e o loop é repetido até que todo o arquivo de entrada seja lido.

## 3. Testes

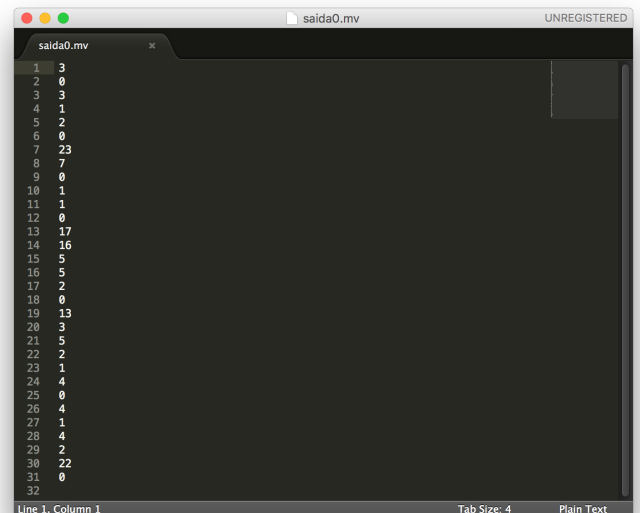
Seis testes foram realizados com o montador. Os testes estão armazenados no diretório *tp2\_alefmon/tst* e foram nomeados como teste0.mv, teste1.mv, teste2.mv, teste3.mv, teste4.mv e teste5.mv, respectivamente. Cada um dos testes utilizou o montador para gerar um código numérico, o qual foi posteriormente inserido na Máquina Virtual. O detalhamento de cada um dos testes será realizado a seguir.

### 3.1 Teste 0

O Teste 0 foi feito utilizando o programa contido na especificação do TP2.



```
1 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2 ; Programa que imprime o maior numero entre dois informados ;
3 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
4
5 READ R0 ; Salva primeiro numero do registrador 0
6 READ R1 ; Salva segundo numero no registrador 1
7 STORE R0 A ; Faz "backup" do valor de R0 em A
8 SUB R0 R1 ; Subtrai o segundo numero do primeiro
9 LOAD R0 A ; Recupera valor de R0
10 JN MB ; Caso seja negativo, R1 > R0
11 COPY R2 R0 ; Caso contrario, R0 >=, entao salva valor do primeiro numero em R2
12 JMP L
13 MB: COPY R2 R1 ; Salva valor do segundo numero em R2
14 L: WRITE R0 ; Imprime primeiro numero
15 WRITE R1 ; Imprime segundo numero
16 WRITE R2 ; Imprime maior numero
17 HALT
18 A: WORD 0
19 END
20 READ R0
21
22 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
23
```



```
1 3
2 0
3 3
4 1
5 2
6 0
7 23
8 7
9 0
10 1
11 1
12 0
13 17
14 16
15 5
16 5
17 2
18 0
19 13
20 3
21 5
22 2
23 1
24 4
25 0
26 4
27 1
28 4
29 2
30 22
31 0
32
```

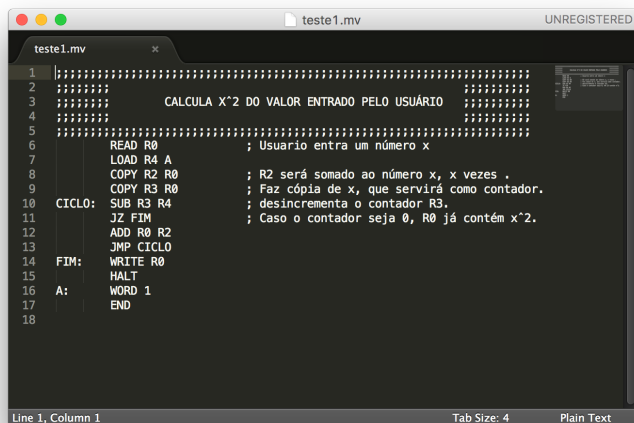


```
bin -- -bash -- 81x24
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida0.mv s
3 4
3
4
4
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida0.mv s
59 830
59
830
830
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida0.mv s
100567 99000
100567
99000
100567
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida0.mv s
0 48
0
48
48
Alef:bin alefmonteiros$
```

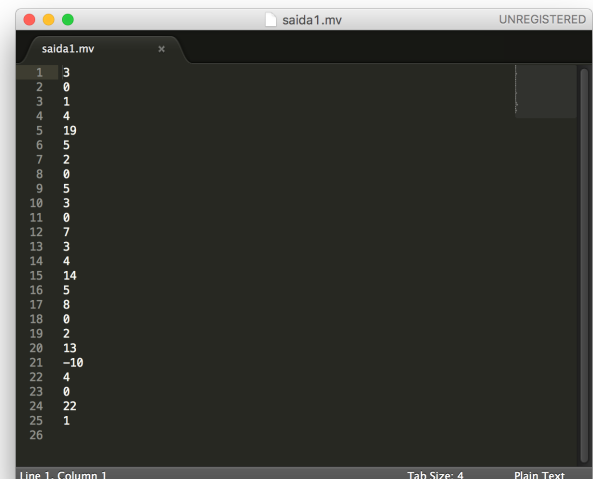
## 3.2 Teste 1

O Teste 1 foi feito com um programa que calcula o quadrado de um número inserido pelo usuário. O programa pede ao usuário que digite um número e o aloca no registrador 0. Duas cópias deste valor são feitas e colocadas nos registradores 2 e 3. O registrador 4, por sua vez, contém o valor 1, que será utilizado para decrementar o registrador 3(contador).

A cada ciclo do programa de teste, o valor inserido pelo usuário contido no registrador 2 é somado ao valor total. Esse ciclo é executado n vezes, de acordo com o número inserido pelo usuário. Ao final, é impresso na tela o número inserido ao quadrado.



```
1  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
2  ::::::::::
3  :::::::::: CALCULA X^2 DO VALOR ENTRADO PELO USUÁRIO ::::::::::
4  ::::::::::
5  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
6  ::::::::::
7  READ R0 ; Usuario entra um número x
8  LOAD R4 A
9  COPY R2 R0 ; R2 será somado ao número x, x vezes .
10 COPY R3 R0 ; Faz cópia de x, que servirá como contador.
11 CICLO: SUB R3 R4 ; desincrementa o contador R3.
12 JZ FIM ; Caso o contador seja 0, R0 já contém x^2.
13 ADD R0 R2
14 JMP CICLO
15 FIM: WRITE R0
16 A: WORD 1
17 END
18
```



```
1  3
2  0
3  1
4  4
5  19
6  5
7  2
8  0
9  5
10 3
11 0
12 7
13 3
14 4
15 14
16 5
17 8
18 0
19 2
20 13
21 -10
22 4
23 0
24 22
25 1
26
```



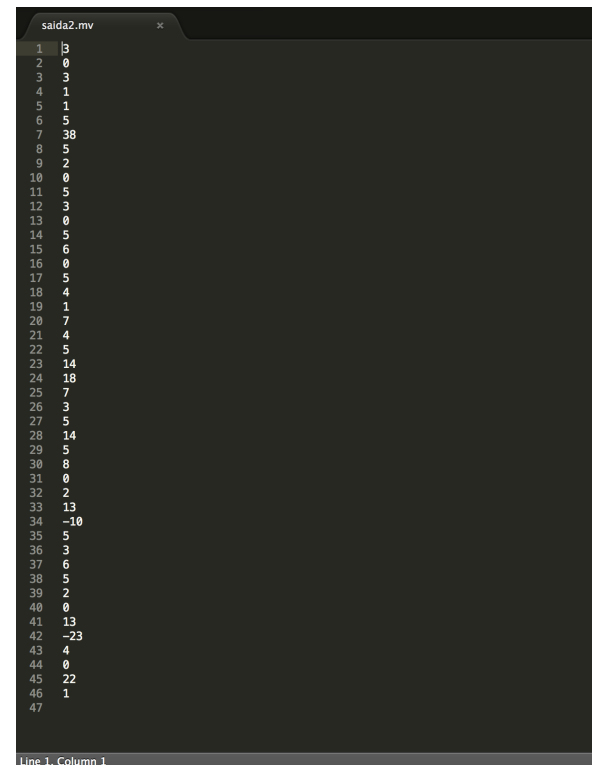
```
bin -- -bash -- 81x24
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida1.mv s
4
16
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida1.mv s
6
36
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida1.mv s
25
625
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida1.mv s
900
810000
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida1.mv s
37
1369
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida1.mv s
21
441
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida1.mv s
50
2500
Alef:bin alefmonteiro$
```

## 3.3 Teste 2

O Teste 2 é o programa “Exponenciação”, que lê dois números (base, expoente), faz a exponenciação e imprime o resultado. A implementação deste programa é similar à do Teste 1, mas desta vez é lido um valor extra, que será o expoente da operação. Outra diferença é que neste teste o ciclo é executado um número de vezes correspondente ao expoente lido.



```
1  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
2  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
3  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
4  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
5  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
6  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
7  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
8  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
9  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
10 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
11 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
12 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
13 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
14 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
15 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
16 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
17 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
18 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
19 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
20 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
21 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
22 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
23 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
24 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
25 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
26 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```



```
1 3
2 0
3 3
4 1
5 1
6 5
7 38
8 5
9 2
10 0
11 5
12 3
13 0
14 5
15 6
16 0
17 5
18 4
19 1
20 7
21 4
22 5
23 14
24 18
25 7
26 3
27 5
28 14
29 5
30 8
31 0
32 2
33 13
34 -10
35 5
36 3
37 6
38 5
39 2
40 0
41 13
42 -23
43 4
44 0
45 22
46 1
47
```



```
bin -- -bash -- 81x24
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida2.mv s
3 4
81
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida2.mv s
2 8
256
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida2.mv s
2 10
1024
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida2.mv s
10 8
10000000
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida2.mv s
9 3
729
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida2.mv s
5 4
625
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida2.mv s
8 2
64
Alef:bin alefmonteiro$
```

## 3.4 Teste 3

O Teste 3 é o programa “Divisão”, que lê dois números e faz a divisão força bruta. Por fim, o programa imprime o quociente e o resto da divisão.

Após a leitura dos dois números inseridos pelo usuário (divisor e dividendo), o programa faz sucessivas subtrações, onde o dividendo é subtraído do divisor, até que o

resultado seja negativo. Quando o resultado se torna negativo, o número do dividendo é somado uma vez ao divisor, resultando no resto da divisão. Para encontrar o quociente, o programa utiliza um contador, no registrador 2, que é incrementado em um a cada subtração feita pelo divisor. Dessa forma, ao fim do programa são impressos na tela o quociente e o resto da divisão.

```

teste3.mv
1  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
2  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
3  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
4  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
5  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
6  READ R0          ; Lê o divisor.
7  READ R1          ; Lê o dividendo.
8  LOAD R4 A        ; Armazena o valor 1 em R4, que servirá para
9  Incrementar o quociente.
10 LOOP: SUB R0 R1   ; Loop executado até que o resto seja menor que o
11 dividendo.
12 JN FIM           ; Incrementa R2, que é o quociente da divisão.
13 ADD R2 R4        ; Incrementa R2, que é o quociente da divisão.
14 JMP LOOP
15 FIM: ADD R0 R1    ; O resto é armazenado em R3.
16 COPY R3 R0       ; Imprime o quociente da divisão.
17 WRITE R2         ; Imprime o resto da divisão.
18 HALT
19 A: WORD 1
20 END

```

```

saida3.mv
1  3
2  0
3  3
4  1
5  1
6  4
7  21
8  7
9  0
10 1
11 16
12 5
13 8
14 2
15 4
16 13
17 -10
18 0
19 0
20 1
21 5
22 3
23 0
24 4
25 2
26 4
27 3
28 22
29 1
30

```

```

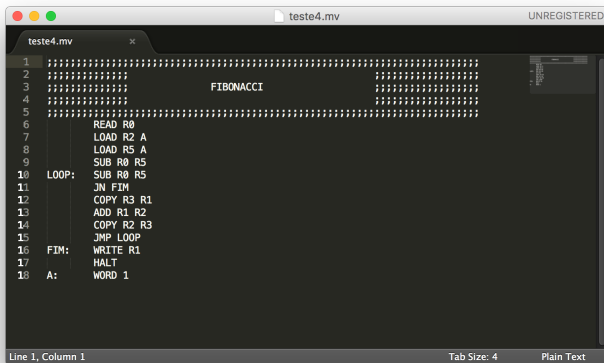
bin -- -bash -- 81x25
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida3.mv s
107 20
5
7
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida3.mv s
248 2
124
0
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida3.mv s
237 45
5
12
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida3.mv s
900 32
28
4
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida3.mv s
10 5
2
0
Alef:bin alefmonteiro$ ./emulador 0 0 1000 saida3.mv s
1740 31
56
4
Alef:bin alefmonteiro$

```

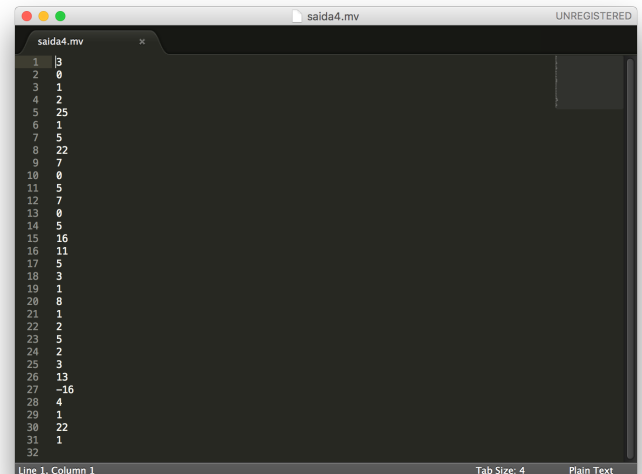
## 3.5 Teste 4

O Teste 4 é o programa “Fibonacci”, que lê um número N e imprime na tela do usuário o N-ésimo termo da sequência de Fibonacci.

O programa lê o número N e o armazena no registrador 0. Além disso, o programa utiliza outros três registradores para formar a sequência de Fibonacci. Um desses registradores é iniciado com 1 e os outros dois com zero. Um ciclo é executado N vezes e para quando o valor do registrador 0 fica negativo. Esse registrador, que contém o número inserido pelo usuário, serve como contador e é decrementado a cada ciclo. O ciclo executa a adição de dois valores seguidos da sequência e armazena o resultado no registrador 1, que representa o próximo termo da sequência de Fibonacci. Por fim, o valor do registrador 1 é impresso na tela assim que o ciclo é finalizado.



```
teste4.mv
1  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
2  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
3  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
4  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
5  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
6  READ R0
7  LOAD R2 A
8  LOAD R5 A
9  SUB R0 R5
10 LOOP: SUB R0 R5
11      JN FIM
12      COPY R3 R1
13      ADD R1 R2
14      COPY R2 R3
15      JMP LOOP
16 FIM:  WRITE R1
17      HALT
18 A:    WORD 1
```



```
saida4.mv
1  1
2  1
3  1
4  2
5  2
6  5
7  5
8  22
9  7
10 0
11 5
12 7
13 0
14 5
15 16
16 11
17 5
18 3
19 1
20 8
21 1
22 2
23 5
24 2
25 3
26 13
27 -16
28 4
29 1
30 22
31 1
32
```

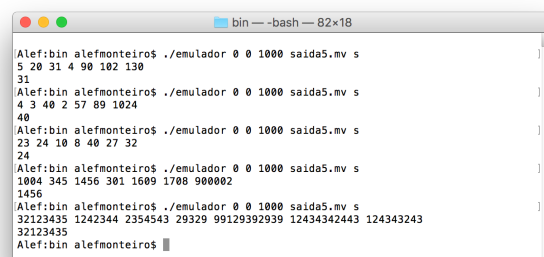


```
bin -- -bash -- 81x25
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida4.mv s
8
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida4.mv s
13
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida4.mv s
1
0
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida4.mv s
2
1
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida4.mv s
3
1
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida4.mv s
4
2
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida4.mv s
5
3
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida4.mv s
6
5
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida4.mv s
15
377
Alef:bin alefmonteiros$
```

## 3.6 Teste 5

O Teste 5 é o programa “Mediana”. Este programa lê 7 inteiros e os armazena nos registradores R0 a R6. Após a leitura dos valores, o programa compara o valor de R0 com o valor dos outros registradores e troca os valores de posição caso o valor de R0 seja maior. Dessa forma, ao fim das comparações, o valor de R0 é o menor entre os valores lidos.

O programa repete esse procedimento e compara R1 com todos os outros valores, exceto R0. Depois, compara também o R2 com os outros, exceto R0 e R1. Por fim, o programa compara R3 com os outros registradores restantes, de maneira que, ao final das comparações, R3 contém a mediana. Assim, o valor de R3 é impresso na tela do usuário.



```
bin -- -bash -- 82x18
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida5.mv s
5 20 31 4 90 102 130
31
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida5.mv s
4 3 40 2 57 89 1024
40
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida5.mv s
23 24 10 8 40 27 32
24
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida5.mv s
1004 345 1456 301 1609 1708 900002
1456
Alef:bin alefmonteiros$ ./emulador 0 0 1000 saida5.mv s
32123435 1242344 2354543 29329 99129392939 12434342443 124343243
32123435
Alef:bin alefmonteiros$
```

## 4. Conclusão

A criação de um programa que simula o montador é uma tarefa complexa e que exige um amplo conhecimento a respeito do funcionamento não só do montador, mas também de outras camadas de um computador. Toda abordagem do conteúdo feita pela disciplina de Software Básico colabora muito para a execução do trabalho.

Finalmente, o Trabalho Prático mostrou a grande diferença do código numérico para o código em Assembly. Durante a criação dos programas de teste, a facilidade do uso da linguagem Assembly torna-se evidente, já que os programas exigiriam um esforço extremamente maior caso a criação devesse ser feita em linguagem de máquina. O grande código numérico gerado pelo montador, a partir de um programa de poucas linhas, ilustra bem esse contraste entre as linguagens.

## 5. Referências

[1] TANENBAUM, Andrew S.; AUSTIN, Todd. Organização Estruturada de Computadores. 6. ed. : Pearson, 2013.