

Universidade Federal De Ouro Preto

Montador para a Máquina ECEE R19.1

Grupo:

Daniel Henrique Batista de Magalhães - 18.1.8042

José Martins da Rosa Júnior - 18.1.8137

Luiz Henrique Marcos Ferreira - 18.1.8020

Miriele dos Santos Campos - 18.2.8177

Professor: Carlos Henrique Gomes Ferreira

Relatório de documentação descritiva referente ao trabalho prático de implementação de uma máquina, a "ECEE R19.1", na disciplina de Organização e Arquitetura de Computadores I da Universidade Federal de Ouro Preto.

Maio
2019

Conteúdo

1	Introdução	1
2	Assembler.h	1
2.1	<i>Linha</i>	1
2.2	<i>Passo1</i>	1
3	main.c	2
4	Assembler.c	2
4.1	<i>lendoPrograma</i>	2
4.2	<i>primeiroPasso</i>	2
4.3	<i>montandoEscrita</i>	2
4.4	<i>converteBinario</i>	3
4.5	<i>auxiliarEscrita</i>	3
5	Programas desenvolvidos	3
6	Testes	4
6.1	Fibonacci.a	4
6.2	SomaRecursivo.a	5
6.3	Quadrado.a	6

1 Introdução

O montador implementado, intitulado como **Máquina ECEE R19.1**, teve por objetivo fazer a simulação de um *assembler* onde, basicamente, é passado um arquivo como entrada, usando instruções especificadas no trabalho, contendo a extensão **".a"**, que é convertido em outro arquivo, porém em linguagem de máquina, contendo a extensão **".mif"**. O ambiente utilizado foi o terminal do **Ubuntu 16**, e para verificar se o arquivo gerado estava correto, foi utilizado o programa CPUSim, pré-requisitado pelo professor. O código fonte foi implementado em formato *TAD* (Tipo Abstrato de Dados), logo o mesmo contém dois arquivos com extensão **".c"** e um com extensão **".h"**, onde o arquivo **".h"** contém o título das sub-rotinas e as estruturas utilizadas, fora o fato de que foi feito um *MakeFile*, para facilitar a compilação, como exigência do próprio trabalho.

2 Assembler.h

As seguintes estruturas foram utilizadas para facilitar a manipulação e leitura do código;

2.1 *Linha*

A estrutura linha é composta por 3 itens. O primeiro, e principal, é um vetor de string **cod**. Nele é armazenado os códigos de uma linha, para manipulação futura. Os outros dois itens são auxiliares. **qtCod** é usado para contar a quantidade de instruções que tem na linha e **numLinha** é usada para contar a linha em que está sendo lida as instruções do programa **".a"**.

2.2 *Passo1*

Essa estrutura tem como objetivo armazenar os **Label's** (rótulos) e os números de referência. O vetor de strings **label**, armazenado todos os rótulos encontrados no programa **".a"**. O vetor de strings **"numeroReferencia"**, armazena em binário, a posição da linha que o rótulo estava. Devido às especificações do montador, a posição da linha em que o rótulo está é multiplicada por 2.

3 main.c

A função principal, localizada no arquivo **main.c** é responsável por coletar o nome do arquivo com extensão **".a"** no qual estamos interessados em converter para linguagem de máquina, é passado o nome digitado para a função *lendoPrograma*, que retorna a quantidade de linhas para uma variável inteira criada, logo em seguida faz a chamada da função *primeiroPasso* que recebe a quantidade de linhas do arquivo, após isso é feita a chamada da função de *montandoEscrita* que recebe o a quantidade de linhas do arquivo e o seu nome.

4 Assembler.c

Para o cumprimento da tarefa passada, foram implementadas as seguintes funções.

4.1 *lendoPrograma*

A função de leitura do arquivo em assembly, localizada no arquivo **Assembler.c**, percorre o arquivo de entrada colocando as informações de cada linha na posição de um vetor de estrutura *Linha* alocado dinamicamente e faz algumas contagens, como a contagem da quantidade de linhas, que será utilizada por outras partes do código.

4.2 *primeiroPasso*

O objetivo desta função, em resumo, é encontrar os **Label's**(rótulos) das funções presentes no arquivo. Em cada posição do vetor de estrutura descrito na seção *lendoPrograma*, a função irá percorrê-lo, e como sabemos que os **Label's** estão sempre entre os caracteres **"_"** e **":"**, usamos os mesmos como método de identificação desses títulos, e estes serão depositados em um outro vetor de estrutura *Passo1* alocado dinamicamente. Outro objetivo desta sub-rotina é salvar os endereços de cada rótulo((posição da linha)*2 em binário), para assim termos a referência de cada um.

4.3 *montandoEscrita*

Essa função, utiliza o vetor criado na função *lendoPrograma* e o vetor usado na função *primeiroPasso*. Ela percorre o vetor de estruturas *Linha* e escreve em um outro vetor de strings o correspondente em binário

da linha do arquivo de entrada **".a"**. Ou seja, ela verifica quais instruções estão naquela linha e as escreve nesse vetor. Os valores das instruções foram pré-definidos no roteiro apresentado pelo professor. Feito isso, é iniciado a escrita do arquivo **".mif"** e a impressão da mesma informação no terminal. Visto que cada linha de um programa **".a"** corresponde a 16 bits e que o montador utiliza 8 bits por linha, a forma de escrita e impressão dos valores em binário, foram de 8 bits por linha, ou seja, imprime a metade de uma posição do vetor de strings, é saltada uma linha e é impresso a outra metade, cumprindo assim as especificações do montador.

4.4 *converteBinario*

Essa função converte um valor inteiro em um valor binário correspondente de 9 bits. Inclusive é tratado o caso de valores negativos (complemento de zero).

4.5 *auxiliarEscrita*

O objetivo desta função é reconhecer quais registradores estão sendo utilizados naquela linha e a quantidade de bits necessária para esses registradores, visto que as instruções variam a quantidade de bits dedicada para cada parâmetro. Como pode se perceber ela é uma função auxiliar da função *montandoEscrita*.

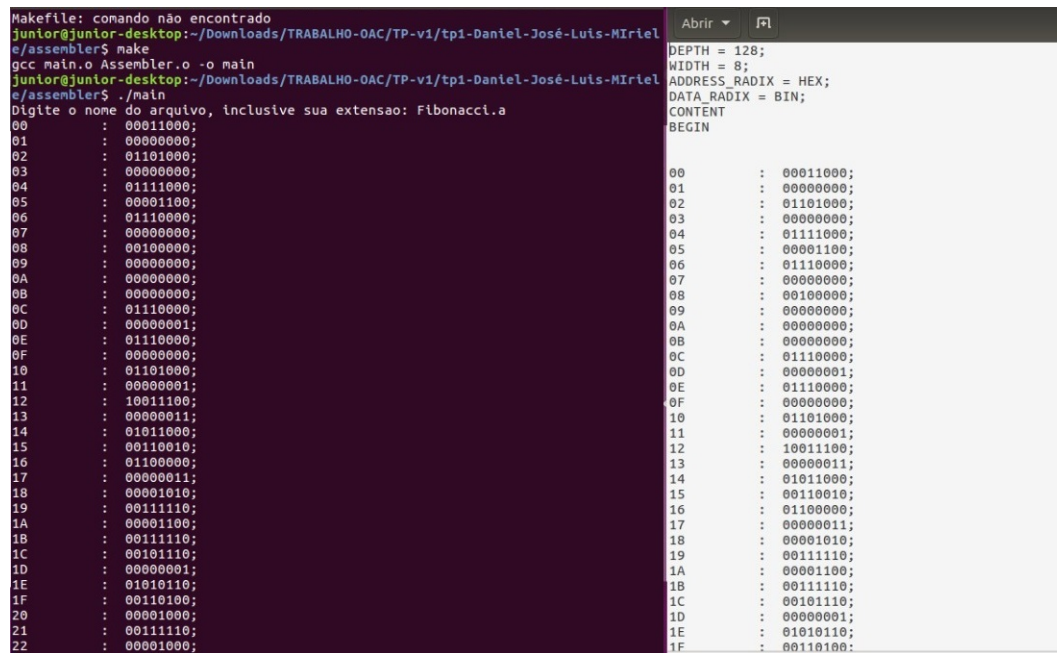
5 Programas desenvolvidos

Para fazer realizar os testes no código fonte, utilizamos como base o arquivo **"Fibonacci.a"**, passado junto ao programa *CPUSim*, porém, além destes, foram implementados dois outros códigos; **"SomaRecursivo.a"** e **"Quadrado.a"**, onde o primeiro foi desenvolvido com um objetivo aparentemente simples, somar dois números, mas para poder explorar bem a linguagem utilizada, o código foi implementado de maneira recursiva, o segundo foi implementado de maneira que a entrada possa ser qualquer número inteiro digitado pelo usuário e a saída seja o quadrado do mesmo.

6 Testes

Os testes registrados levam em conta, principalmente, o funcionamento do código fonte no terminal e o arquivo sendo testado no programa *CPUSim*.

6.1 Fibonacci



The image shows a terminal window on the left and a file editor on the right. The terminal window displays the following commands and output:

```
Makefile: comando não encontrado
junior@junior-desktop:~/Downloads/TRABALHO-OAC/TP-v1/tp1-Daniel-José-Luis-Miriel
e/asmblers$ make
gcc main.o Assembler.o -o main
junior@junior-desktop:~/Downloads/TRABALHO-OAC/TP-v1/tp1-Daniel-José-Luis-Miriel
e/asmblers$ ./main
Digite o nome do arquivo, inclusive sua extensão: Fibonacci.a
00 : 00011000;
01 : 00000000;
02 : 01101000;
03 : 00000000;
04 : 01111000;
05 : 00001100;
06 : 01110000;
07 : 00000000;
08 : 00100000;
09 : 00000000;
0A : 00000000;
0B : 00000000;
0C : 01110000;
0D : 00000001;
0E : 01110000;
0F : 00000000;
10 : 01101000;
11 : 00000001;
12 : 10011100;
13 : 00000011;
14 : 01011000;
15 : 00110010;
16 : 01100000;
17 : 00000011;
18 : 00001010;
19 : 00111110;
1A : 00001100;
1B : 00111110;
1C : 00101110;
1D : 00000001;
1E : 01010110;
1F : 00110100;
20 : 00001000;
21 : 00111110;
22 : 00001000;
```

The file editor on the right shows the following content:

```
DEPTH = 128;
WIDTH = 8;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN
00 : 00011000;
01 : 00000000;
02 : 01101000;
03 : 00000000;
04 : 01111000;
05 : 00001100;
06 : 01110000;
07 : 00000000;
08 : 00100000;
09 : 00000000;
0A : 00000000;
0B : 00000000;
0C : 01110000;
0D : 00000001;
0E : 01110000;
0F : 00000000;
10 : 01101000;
11 : 00000001;
12 : 10011100;
13 : 00000011;
14 : 01011000;
15 : 00110010;
16 : 01100000;
17 : 00000011;
18 : 00001010;
19 : 00111110;
1A : 00001100;
1B : 00111110;
1C : 00101110;
1D : 00000001;
1E : 01010110;
1F : 00110100;
```

Figura 1: Teste do código fonte no terminal a esquerda e arquivo gerado à direita

6.2 SomaRecursivo.a

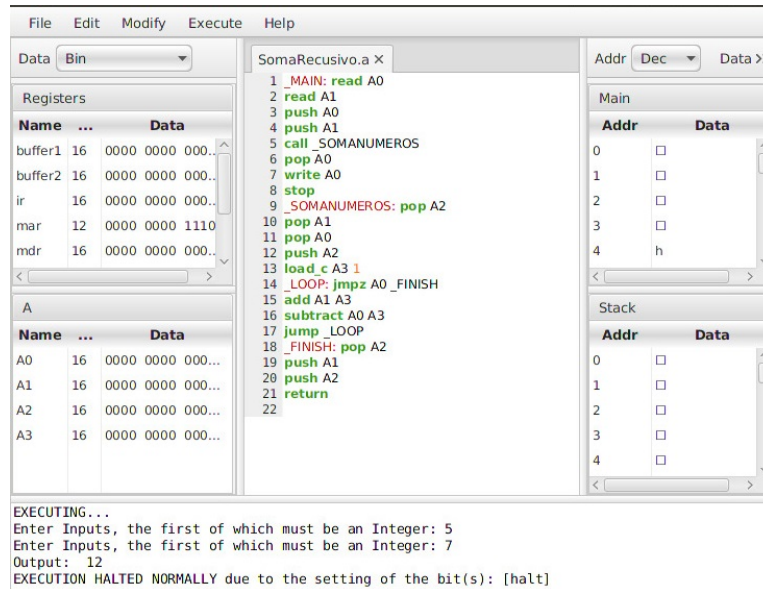


Figura 2: Código desenvolvido no *CPUSim*

```

junior@junior-desktop:~/Downloads/TRABALHO-OAC/TP-v1/tp1-Daniel-Jose-Luis-Miriele/asm$ ./main
Digite o nome do arquivo, inclusive sua extensao: /home/junior/Downloads/TRABALHO-OAC/TP-v1/tp1-Daniel-Jose-Luis-Miriele/tst/somaIntervalo.a
00 : 00011000;
01 : 00000000;
02 : 00011000;
03 : 00000001;
04 : 01101000;
05 : 00000000;
06 : 01101000;
07 : 00000001;
08 : 01111000;
09 : 00010000;
0A : 01110000;
0B : 00000000;
0C : 00100000;
0D : 00000000;
0E : 00000000;
0F : 00000000;
10 : 01110000;
11 : 00000010;
12 : 01110000;
13 : 00000001;
14 : 01110000;
15 : 00000000;
16 : 01101000;
17 : 00000010;
18 : 10011110;
19 : 00000001;
1A : 01010000;
1B : 00100100;
1C : 00101010;

```

Figura 3: Teste do código fonte no terminal

6.3 Quadrado.a

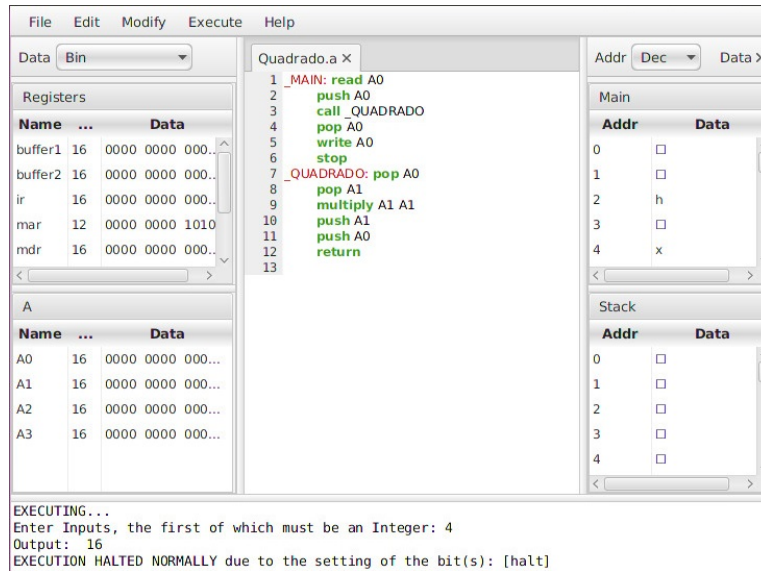


Figura 4: Código desenvolvido no *CPUSim*

```
junior@junior-desktop:~/Downloads/TRABALHO-OAC/TP-v1/tp1-Daniel-Jose-Luis-Miriele/assembly$ ./main
Digite o nome do arquivo, inclusive sua extensao: /home/junior/Downloads/TRABALHO-OAC/TP-v1/tp1-Daniel-Jose-Luis-Miriele/tst/quadrado.a
00 : 00011000;
01 : 00000000;
02 : 01101000;
03 : 00000000;
04 : 01111000;
05 : 00001100;
06 : 01110000;
07 : 00000000;
08 : 00100000;
09 : 00000000;
0A : 00000000;
0B : 00000000;
0C : 01110000;
0D : 00000000;
0E : 01110000;
0F : 00000001;
10 : 00111010;
11 : 00000001;
12 : 01101000;
13 : 00000001;
14 : 01101000;
15 : 00000000;
16 : 10000000;
17 : 00000000;
[18..7F] : 00000000;
END;
```

Figura 5: Teste do código fonte no terminal