Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Sistemas embarcados – 2021/1

Trabalho Prático 2 - Montador

Nome: Lecio Charlles Barbosa
Matrícula: 2016065120

Nome: Breno de Castro Pimenta
Matrícula: 2017114809

1. INTRODUÇÃO

O objetivo deste trabalho é criar um montador (assembler) para a máquina Swombat R3.0 que será executada utilizando o simulador CPUSim. Esse simulador é um ambiente de desenvolvimento construído em Java que permite projetar e implementar qualquer arquitetura utilizando a linguagem de montagem. O CPUSim possibilita projetar no nível dos registradores, especificar instruções e memórias, permitindo assim simular uma máquina completa, esse processo acontece através do carregamento de um arquivo XML com a descrição da arquitetura. A arquitetura fornecida foi projetada com 4 registradores de proposito geral: A0, A1, A2, A3 e 8 de proposito específico, estes últimos não são acessíveis ao programador. Temos também duas memorias: uma principal que armazena o programa e outra em formato de pilha, para manipular variáveis locais e outras operações. Tanto os dados, quanto as instruções possuem 16 bits, ocupando duas linhas da memória principal, que possui 1 byte cada. Já a pilha conta com 256 posições e os números são em complemento de 2 e o *endianess* é *little*.

2. IMPLEMENTAÇÃO

Nesta seção serão tratadas as decisões de implementação, desde o detalhamento da lógica à partes do código desenvolvido. O primeiro ponto notório a ser mencionado foi a escolha da linguagem, este trabalho foi implementado utilizando Python versão 3. Já as demais decisões estão descritas nos próximos subitens.

2.1. Arquivos

O projeto é dividido em três pastas, “assembler” onde está contido o código fonte do projeto (ver item 2.2), “tst” onde estão os programas de teste (ver item 3) e “doc” que tem como único objetivo armazenar a documentação.

2.2. Código

O arquivo de código que contém o montador é o assembler/main.py. A parte inicial do arquivo é a declaração de duas constantes, em formato de dicionário, “instrucao” e “reg”, que representam as instruções e os registradores respectivamente. Há 23 instruções representadas por um opcode de 5 bits cada e 4 registradores, mais um caso de don’t care, toda essa estrutura foi montada obedecendo às especificações da máquina Swombat R3.0. O restante do arquivo é dividido em três funções, a função principal *main* responsável por ler o arquivo de entrada (ver item 2.4) e gerenciar a chamada das outras funções, *traduzir* e *imprimir.* A função *traduzir* recebe uma instrução e é responsável por entender o seu opcode e executar a operação correspondente, onde terá uma lista com 128 posições, inicializada com zeros, que retrata a memória da máquina, a qual armazenará as operações das instruções. Após todas as instruções executadas a *main* irá chamar a função *imprimir*, a qual transformará e imprimirá essa memória final, já manipulada pelas instruções, no formato Intel HEX, explicado em detalhes no item (2.5).

2.3. Execução

Para executar o programa, pode-se utilizar o Makefile. Ao executar o comando make, ele executará os 4 programas escritos para a maquina Swombat em sequencia (ver item 3), gerando o arquivo em formato Intel HEX (ver item 2.5). Caso haja necessidade de se adicionar mais testes, basta criar um arquivo dentro da pasta *tst* com o padrao de nome “testeN.a” onde N é o numero do teste em questão e modificar o código apropriado no Makefile.

Outra forma de utilizar o programa é executar no terminal:

```
>python3 main.py “nome do arquivo assembly” “nome do arquivo de saida”
```

2.4. Entrada

A máquina recebe como entrada um arquivo com extensão *.a* onde cada linha desse arquivo representa uma instrução da máquina *Swombat R3.0*. As linhas podem conter comentários, porém é necessário utilizar o caractere “;” antes do comentário, pois será verificado como delimitador. As instruções possuem 0, 1 ou 2 operandos. Os labels são traduzidos para endereços de memória, devem começar com “_”, mas o montador implementado também os reconhecerá sem. A pseudo-instrução *.data* é usada para indicar uma área de dados na memória, ela sempre será precedida por um label. Esse tipo de instrução pode armazenar apenas um elemento de 16 bits.

Os procedimentos são chamados com **call** e retornados com **return**. Quando chamamos um procedimento com call, o valor do pc (instrução atual) é colocado na pilha e retirado quando encerramos com return. Definimos a passagem de parâmetros por registradores, o qual o A0 armazenara um valor ou a referência para um vetor de valores que podem ser usado tanto para passagem de parâmetros quanto para o retorno do procedimento.

2.5. Saída

O montador ao ler o arquivo, traduzi-lo e processar as instruções, gera um arquivo texto com a extensão *.hex*, onde cada linha representa um byte da memória, logo cada arquivo terá 128 linhas (tamanho da memória da máquina Swombat R3.0) + 1, onde a linha adicional “:00000001FF” represesta o final do arquivo. Esse formato é conhecido como Intel HEX, muito comum parar programar chips. As linhas que representam os bytes são formatadas da seguinte forma: o primeiro dígito sempre é o caractere “:”; os próximos dois dígitos representam o número de bytes, logo no caso deste trabalho sempre será “01”, como cada linha sempre representará um único byte da memória; os próximos 4 dígitos representam o endereçamento; os próximos dois dígitos são relativos ao dado que aquele byte representa; os próximos n dígitos em hexadecimal representam os dados, que nos casos usados para esse trabalho não ultrapassaram dois dígitos (salve a linha indicativa de final de arquivo); finalmente os últimos dois dígitos são valores que permitem verificar se o registro contém algum erro.

3. PROGRAMAS TESTES

Os testes estão presentes na pasta *tst,* onde para cada teste há um arquivo de entrada com as instruções com a extensão *.a,* como os respectivos arquivos de saída no formato HEX. Este projeto traz quatro programas testes que juntos utilizam 18 das 23 instruções da máquina, a Figura 01 ilustra a execução do teste 3 no simulador CPUSim.

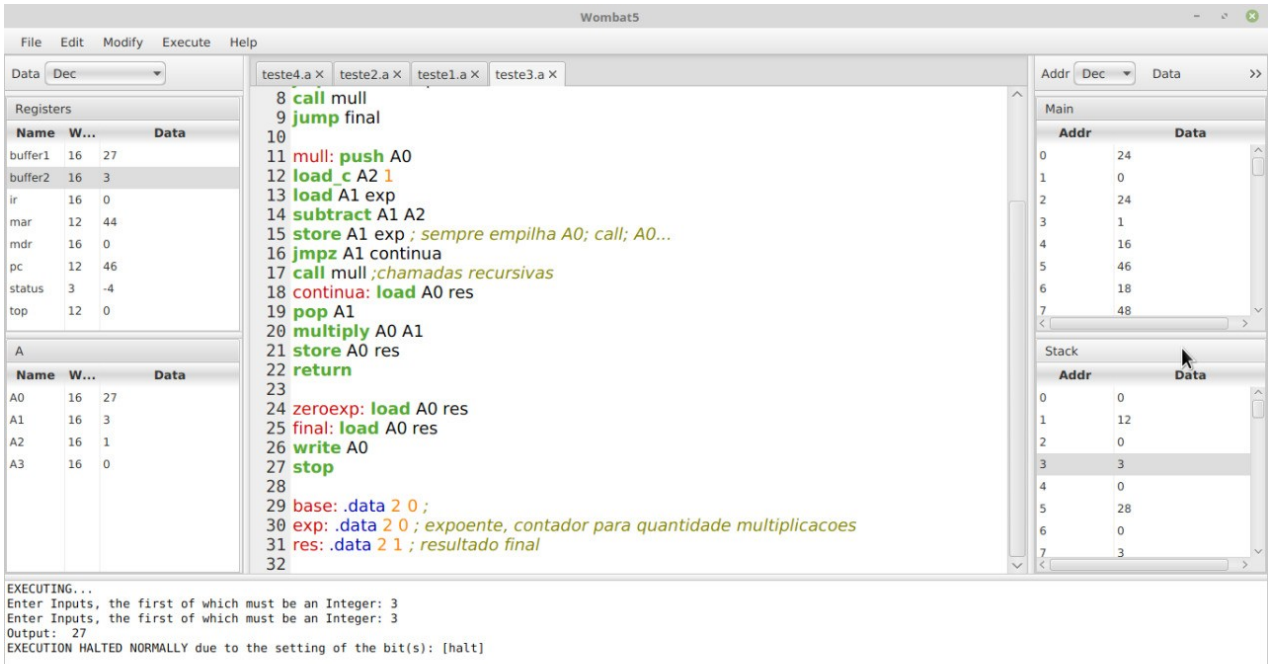

Figura 01: CPUSim executando o teste3.a.

Abaixo há a descrição dos quatro programas testes disponibilizados no projeto:

O Programa teste1.a calcula o fatorial do número dado como entrada e imprime o valor na tela, fizemos este usando chamadas recursivas.

O Programa teste2.a lê inteiros da entrada ate que o 0 apareca, em seguida ele imprimira no console o maior número da entrada.

O Programa teste3.a recebe dois parâmetros, podemos chamá-los de a e b, em seguida calcula a potência a^b e imprime o resultado. Este programa utiliza chamadas recursivas para realizar a multiplicação.

O Programa teste4.a recebe um número como parâmetro e em seguida calcula iterativamente a sequência de fibonacci desse número.