| UFMG - ICEx DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO | Software Básico |
|---|---|
| | Trabalho Prático 2 – Implementação de um processador de calculadora |
| | Autore: Jhonatan Vinícius Mota Corrêa |
| | Data: 30/06/2019 |

# Descrição

Este documento descreve o trabalho prático que foi usado para treinar e fixar os conceitos aprendidos em sala de aula na disciplina Software Básica. O trabalho consiste de duas etapas: um emulador de uma máquina virtual básica, proposta para facilitar a implementação; e um montador para essa máquina, que permita a usuários escreverem programas em assembly para essa máquina básica.

# Desenvolvimento

## 1 - Máquina virtual:

A primeira parte do trabalho foi o desenvolvimento da máquina virtual descrita na especificação. O trabalho foi desenvolvido com base em C++ com o auxílio das bibliotecas iostream, fstream e string.

A entrada é passada via argumento, sendo a entrada o nome do arquivo com os dados necessários para a execução. O primeiro passo da máquina é analisar se esse argumento foi passado corretamente e segundo se o parâmetro “-v” (modo verbose) foi recebido.

Caso a entrada tenha sido recebida, a máquina tenta abrir o arquivo, e se bem sucedida, checa se o arquivo é válido (confere o cabeçalho “MV.EXE”). Caso positivo também, carrega os parâmetros do programa (PC, entry_point, tamanho, etc...).

Finalizada essa parte, a máquina carrega os dados da entrada para a memória e fecha o arquivo de entrada para evitar erros com o arquivo.

Com a memória carregada, a máquina começa a execução em si. Ela trabalha basicamente como um decodificador: carrega o decodificador correspondente ao PC atual, e a partir da decodificação, toma as ações necessárias (carrega registrador, memória, atualiza PC e etc). Como cada instrução tem um número de operandos diferente, a atualização de PC varia de instrução para instrução.

Esse decodificador é um while, que repete sempre, desde que a instrução não seja um “HALT” ou uma exceção que pare o loop. Nesse decodificador foi usado uma série de ifs e elses, já que o compilador usado (visual c++) não compilou com um switch de muitos cases.

Uma exceção é causada caso uma instrução desconhecida (código não determinado) for carregado, encerrando a execução.

No modo verbose, sempre que uma instrução é executada, ao final é impresso na tela uma linha com o dump dos registradores e da instrução atual. Os registradores foram criados basicamente com variáveis int (já que o processador trabalha basicamente com inteiros). O banco de registradores e a memória são arrays de inteiros. Por simplicidade, a memória é tratada com tamanho fixo de mil posições.

Por simplificação, uma função auxiliar (atualizaPep) foi utilizada. Basicamente se passa pra ela o valor do ultimo registrador calculado, e ela retorna o valor de pep correspondente.

Detalhe: para evitar exceções, todos os programas devem ser encerrados por “HALT”.

### 2 - Montador:

O montador dessa máquina segue um funcionamento similar ao da máquina virtual, decodificado instruções. Nesse algoritmo, são necessários dois argumentos: O nome do arquivo de entrada (com as instruções assembly da máquina virtual) e do arquivo de saída (que será gerado com o código de máquina).

Foram usadas aqui as mesmas bibliotecas da máquina virtual, com a adição da biblioteca sstream. Essa facilita na manipulação das instruções.

A abordagem usada aqui para a montagem do código foi a de duas etapas, uma para montar a tabela de símbolos e a segunda para a montagem do código de máquina em si. A tabela é basicamente um array de structs. Uma struct específica foi construída chamada “label”, na qual contem uma string “name” com o nome da label e um int “PC” com a posição da mesma. Por simplicidade, a tabela foi montada com tamanho fixo de 100.

A primeira passagem basicamente passa pelo código, identificando as labels. Isso é feito identificando os operadores que terminam com “:” como especificado na especificação do trabalho.

Após essa primeira passagem, o arquivo de entrada e fechado e aberto novamente, agora para a leitura das instruções e codificação da saída.

Nessa segunda passagem, a abordagem é parecida com a máquina virtual. Cada operador é lido, e de acordo com a operação, é lido os operandos (um ou dois). Nesse caso, duas funções auxiliares foram usadas. A primeira decodeReg, recebe o nome de um registrado e decodifica em um registrador correspondente (basicamente retorna o índice deste no banco).

A outra função é a countPC (também usada na primeira passagem e justamente por isso usada). Como cada instrução tem um número de operandos diferente, o contador da máquina atualiza de maneira diferente. Logo convém ter essa instrução que retorna o quanto o PC vai ser atualizado em cada instrução. Toda essa codificação é gravada numa string que ao final vai para o arquivo de saída.

Para instruções com label, sempre que um é lido, o label correspondente é buscado na tabela. Como o endereçamento é relativo ao PC, esse endereço é calculado de acordo com o PC da instrução corrente.

Ao final dessa codificação, o cabeçalho é gravado, junto aos parâmetros do programa(tamanho, pc, ap e entry point). Por simplicidade, foi definido que os programas vão sempre ter entry point, pc em 0 e ap em 999, já que isso não foi especificado anteriormente.

Por fim, a string com o programa codificado é gravado no arquivo de saída, e os arquivos (tanto entrada quanto saida) são fechados para não dar erro.

Detalhe: os comentários devem estar separados do último operando para funcionar.

## Execução

A execução do programa é feita em duas etapas: Montagem e execução. Com os programas compilados, e com um arquivo de entrada em assembly, a execução é dada da seguinte forma:

Diretório: “Montador.cpp” “TPSB.cpp” “programa.txt” – O montador.cpp tem o algoritmo do montador, TPSB tem o programa com a máquina virtual e  programa.txt tem o assembly do programa.

Executar com os seguintes comandos:

Montador programa.txt executável.txt

TPSB executável.txt -v (o -v é opcional e serve para ativar o modo verbose da máquina).

Testes

O programa foi testado em Windows e Linux com os programas “assembly.txt”, “fibonacci.txt” e “mediana.txt” (todos anexos ao programa). Os resultados estão abaixo:

```
C:\Users\Jhon\Documents\Matérias\Software Básico\TP\Testes>Montador assembly_tp.txt saida_assembly_tp.txt

C:\Users\Jhon\Documents\Matérias\Software Básico\TP\Testes>TPSB saida_assembly_tp.txt
50
10
50
10
50
Fim da execu├º├úo

C:\Users\Jhon\Documents\Matérias\Software Básico\TP\Testes>TPSB saida_assembly_tp.txt -v
50
3 2 999 1 50 0 0 0
10
3 4 999 1 50 10 0 0
2 7 999 1 50 10 0 0
20 10 999 0 40 10 0 0
1 13 999 0 50 10 0 0
11 15 999 0 50 10 0 0
5 18 999 0 50 10 50 0
8 23 999 0 50 10 50 0
50
4 25 999 0 50 10 50 0
10
4 27 999 0 50 10 50 0
50
4 29 999 0 50 10 50 0
Fim da execu├º├úo
```

```
C:\Users\Jhon\Documents\Matérias\Software Básico\TP\Testes>Montador mediana.txt saida_mediana.txt

C:\Users\Jhon\Documents\Matérias\Software Básico\TP\Testes>TPSB saida_mediana.txt
1
77
25
59
20
25
Fim da execu├º├úo
```

*Modo verbose não adicionado por ser muito longo

```
C:\Users\Jhon\Documents\Matérias\Software Básico\TP\Testes>TPSB saida_fibonacci.txt
8
13
Fim da execu├º├úo
```

*Modo verbose não adicionado por ser muito longo

Conclusão

No presente trabalho foram exercitados os conceitos estudados durante a disciplina, tais como montagem, execução de um processador, decodificação de instruções dentre outros conceitos.

Alguns problemas foram encontrados, desde problemas com o compilador, lógica com o compilador, até endereçamentos. Mas até a conclusão deste relatório, todos os testes passaram como esperados.