Trabalho Prático 1

Software Básico

Enzo Roiz

1- Introdução

Visando entender melhor os conceitos de linguagem de máquina que serão abordados durante o curso, uma série de trabalhos práticos serão desenvolvidos como uma forma de fixar tais conceitos. Este primeiro trabalho visa criar um emulador para uma máquina virtual básica que será utilizada como base para os próximos trabalhos ao longo do curso.

A máquina virtual consiste em 16 registradores de uso geral, 3 registradores de uso específico, uma memória de no mínimo 1000 posições. Além disto, a máquina virtual tem um conjunto de 22 instruções.

2 – Implementação

Abaixo seguem a estrutura de dados uilizada para a implementação da máquina virtual bem como as funções utilizadas para a construção do emulador.

2.1 – Estruturas de dados:

```
#define REGISTERS_SIZE 16
#define MEMORY_SIZE 1000

typedef struct VirtualMachine {
  int PC, SP; Registradores específicos
  short unsigned int PSW[2]; Registradores específicos [0] = zero / [1] = negativo
  int memoryAddress[2]; Enderecos de memoria [0] = inicio / [1] = final
  int registers[REGISTERS_SIZE]; Registradores gerais
  int memory[MEMORY_SIZE]; Memória
  int verbose; Modo da VM
} VirtualMachine;
```

2.2 – Funções:

void createVM *Inicializa a VM com os argumentos passados por linha de comando*
void readInstructions *Lê as instruções do arquivo passado por linha de comando*
void runProgram *Roda o programa enquanto houver instruções ou enquanto a instrução não for “halt”*
int getInstruction *Seleciona e executa a instrução através do seu código*
int getNextInstruction *Busca o código da próxima instrução*
void printInstructionName *Imprime na tela o nome da instrução de acordo com seu código*
void updatePSW *Atualiza registrador PSW com relação ao resultado da última operação*

3 – Decisões de projeto

A fim de se criar uma maior modularidade no código, bem como o tornar mais fácil de se alterar, este foi dividido em varias funções menores que executam tarefas específicas. Assim a função principal do programa tem conteúdo reduzido.

Foi utilizado um tipo abstrato de dados chamado VirtualMachine. Com este TAD é possível, caso necessário em trabalhos futuros, a adição de novos campos ao emulador construído neste trabalho, mantendo as estruturas de dados em uma forma organizada e de fácil entendimento.

O código foi dividido em arquivos ***.c*** e ***.h***:

main.c
Arquivo que contém a função *main* do programa. A partir deste arquivo são chamadas as principais funções que fazem o emulador construído rodar.

virtualmachine.c
Arquivo que contém funções chamadas pela *main* além de funções auxiliares que realizam operações específicas, rodando o emulador.

virtualmachine.h
Arquivo que contém a estrutura de dados construída para o emulador e o cabeçalho das funções utilizadas no arquivo *virtualmachine.h.*

4 – Compilação e Execução

4.1 – Compilação

O programa pode ser compilado utilizando *Makefile* através do comando *make* ou alternativamente pode ser compliando utilizando-se:

```
gcc -Wall main.c virtualmachine.c -o emulador
```

4.2 – Execução

Para executar o programa compilado é necessário utilizar o comando:

```
./emulador <PC> <SP> <M> <arquivo.mv> [s|v]
```

Onde:

PC – Valor inicial do PC (contador de programa)
SP – Valor inicial do SP (apontador para pilha)
M – Posição da memória a partir da qual o programa será carregado
arquivo.mv – Localização do arquivo que contém o programa em linguagem de máquina
[s|v] – Parâmetro opcional, escolhe se programa executará em modo verboso

4.3 – Arquivo de entrada

Um exemplo do arquivo de entrada de instruções em linguagem de máquina pode ser visto abaixo:

03
00
01
01
06
08
00
01
04
00
22

4.4 – Saída

Caso o argumento para “verboso” seja *s* ou o argumento seja omitido, o programa é executado sem que sejam exibidas na tela o fluxo de execução do programa e seus registradores. Caso o programa seja executado em modo verboso é possível acompanhar seu fluxo de execução, bem como o estado de seus registradores por meio de mensagens exibidas como abaixo:

READ
PC: 2 SP: 0 PSW[ZERO]: 0 PSW[NEGATIVE]: 0
[0]: 0 [1]: 0 [2]: 0 [3]: 0 [4]: 0 [5]: 0 [6]: 0 [7]: 3 [8]: 0 [9]: 0 [10]: 0 [11]: 0 [12]: 0 [13]: 0 [14]: 0 [15]: 0

5 – Testes

Alguns testes foram realizados de modo a garantir que as instruções seriam executadas de modo correto e sem erros. Abaixo pode-se ver a execução do programa.

```
enzoroiz@enzoroiz:/media/enzoroiz/Storage/Dropbox/Faculdade/Semestre V/SB/TP 1 V1/tp1_enzoroiz/build/emulador$ ./emulador 0 0 0 test_jmp.mv v
Type the number to be read: 10
READ
PC: 2 SP: 0 PSW[ZERO]: 0 PSW[NEGATIVE]: 0
[0]: 10 [1]: 0 [2]: 0 [3]: 0 [4]: 0 [5]: 0 [6]: 0 [7]: 0 [8]: 0 [9]: 0 [10]: 0 [11]: 0 [12]: 0 [13]: 0 [14]: 0 [15]: 0
JMP
PC: 5 SP: 0 PSW[ZERO]: 0 PSW[NEGATIVE]: 0
[0]: 10 [1]: 0 [2]: 0 [3]: 0 [4]: 0 [5]: 0 [6]: 0 [7]: 0 [8]: 0 [9]: 0 [10]: 0 [11]: 0 [12]: 0 [13]: 0 [14]: 0 [15]: 0
Writing register: 10
WRITE
PC: 7 SP: 0 PSW[ZERO]: 0 PSW[NEGATIVE]: 0
[0]: 10 [1]: 0 [2]: 0 [3]: 0 [4]: 0 [5]: 0 [6]: 0 [7]: 0 [8]: 0 [9]: 0 [10]: 0 [11]: 0 [12]: 0 [13]: 0 [14]: 0 [15]: 0
HALT
PC: 8 SP: 0 PSW[ZERO]: 0 PSW[NEGATIVE]: 0
[0]: 10 [1]: 0 [2]: 0 [3]: 0 [4]: 0 [5]: 0 [6]: 0 [7]: 0 [8]: 0 [9]: 0 [10]: 0 [11]: 0 [12]: 0 [13]: 0 [14]: 0 [15]: 0
```

Figura 1 – Execução em modo verboso

6 – Conclusão

Para este trabalho foi construído o emulador da máquina virtual a ser utilizada para os próximos trabalhos. Com isto foi importante entender a especificação e o problema propostos a fim de que num futuro não seja necessário preocupar com a base desenvolvida. Além disto todas as funções foram testadas e a máquina atende às especificações propostas.