Trabalho Prático 1 - Emulador

Victor Pires Diniz

11 de Agosto de 2016

Software Básico - 2º Semestre de 2015

1 Descrição do trabalho

O primeiro trabalho prático do semestre propunha a implementação de um emulador que simulasse uma máquina virtual chamada "Máquina de Khattab". Esse programa deveria receber como entrada um programa em linguagem de máquina e emular sua execução, instrução a instrução.

1.1 Detalhes sobre a Máquina de Khattab

A máquina virtual se trata de uma arquitetura registrador-registrador (conhecida também como arquitetura *load/store*) com instruções de no máximo dois operandos, que podem corresponder a registradores ou posições de memória, dependendo da instrução. Além dos 16 registradores de R0 a R15, há, também, alguns registradores de propósito específico:

- **PC**: armazena posição atual a ser executada no programa. Modificado em instruções que alteram o fluxo de execução, como branches, jumps e chamadas a subrotinas.
- **PSW**: guarda informação sobre o resultado da última operação aritmética realizada, permitindo que instruções funcionem de maneira diferente se tal operação obteve resultado positivo, nulo ou negativo. (Útil para instruções de branch.)
- **SP**: mantém posição atual da pilha da máquina. Modificado em chamadas a subrotinas ou em instruções que empilham ou desempilham explicitamente.

A memória da máquina virtual tem pelo menos 1000 posições capazes de armazenar um inteiro. Inteiros são a menos unidade endereçável e o único tipo de dados tratado pela MV. Além dos dados presentes no programa carregado, a máquina atua também na E/S padrão através de instruções de leitura e escrita. Programas são executados até encontrarem uma instrução *HALT*, que indica o fim da execução.

2 Implementação e decisões de projeto

A fim de modularizar e garantir boa manutenibilidade, o código do emulador foi dividido em três módulos principais. O módulo *Instr* lida com o comportamento de cada instrução, cada uma com sua própria função. Ele fornece, também, uma função que retorna, com base no *opcode* de uma instrução, a função correspondente. *Emulator*, por sua vez, define o comportamento e a estrutura da máquina virtual em si, permitindo instanciar um emulador, carregar um programa de um arquivo e executar a máquina. Finalmente, o arquivo *main.c* apenas trata os parâmetros de execução em linha de comando e instancia o emulador. Por ser muito simples, não terá sua implementação analisada em mais detalhe, mas detalhes sobre os módulos restantes vêm a seguir.

2.1 Instr

Esse módulo é composto de dois arquivos. O header *"instr.h"*, contém, além dos protótipos das funções acessíveis externamente, a definição de um *enum "instr"*, que pode assumir vários nomes diferentes que associam os *opcodes* aos nomes mnemônicos das instruções, com a finalidade de melhorar a legibilidade do código.

O arquivo fonte *"instr.c"* contém funções que desempenham a operação correspondente a cada instrução do conjunto de instruções definido na máquina virtual. Além dos operandos de cada instrução, essas funções recebem também como parâmetro uma referência ao emulador e seus atributos, permitindo acesso e modificação do banco de registradores, de posições na memória e dos registradores especiais.

Além disso, *"instr.c"* define a função *fetchInstr*, que recebe um *opcode* como parâmetro e retorna a função correspondente à instrução. No entanto, em C, ponteiros de funções com protótipo diferente não são iguais, o que dificulta o retorno da função. Para contornar essa dificuldade, foi definido no header um *union* chamado *"oper"*, que contém um dos três tipos de função das instruções (zero, um ou dois operandos). Para permitir a decodificação do *union*, o módulo define também um vetor que associa o opcode de cada instrução ao seu número de operandos.

2.2 Emulator

O módulo responsável pelo emulador da máquina virtual disponibiliza duas funções externamente no arquivo de *headers "emulator.h"*. *"emuFromFile"* cria um novo emulador a partir de um arquivo de entrada, carregando o programa presente nesse arquivo para a memória. *"emuRun"* inicializa o loop de execução do emulador, executando o programa carregado com os parâmetros especificados. Essa função permite, também, acionar o modo *verbose*, que exibe informações de depuração para que o usuário possa avaliar se o programa está se comportando de acordo com o esperado. Ainda nos *headers* do módulo, é definido um *struct "emulator"*, que contém todos os elementos da máquina virtual: banco de registradores, memória etc. Também são declarados *enums* para os modos de execução (simples e verboso) e para os três valores que o *PSW* pode assumir (negativo, nulo e positivo). O arquivo *"emulator.c"* contém a implementação das funções do módulo. A função *"runNext"*, presente nele, executa a próxima instrução no loop de execução da máquina virtual, apontada pelo contador de programa. Para isso, ela chama a função *"fetchInstr"*, definida no módulo *Instr*, e executa a instrução recebida de acordo com seu número de operandos. A função retorna o identificador da instrução executada, para que o loop de execução possa identificar a instrução "HALT" e interromper o fluxo do programa. As outras funções desse arquivo não são tão relevantes, e estão documentadas no próprio código fonte.

3 Compilação e execução

A compilação do emulador pode ser realizada através da *makefile* disponibilizada ou diretamente através do *GCC* ou outro compilador C. A execução do programa deve ser realizada através da linha de comando, na seguinte forma,

```
{endereço do executável do emulador} <pc> <sp> <load_pos> <input> [output_mode]
```

em que:

- `pc`: valor inicial do **PC** no emulador.
- `sp`: valor inicial do **SP** no emulador.
- `load_pos`: posição a partir do qual o programa de entrada será carregado na memória.
- `input`: endereço para o arquivo de entrada, em linguagem de máquina.
- `output_mode`: parâmetro **opcional** que permite escolher entre os modos de execução simples e verboso. Caso não seja especificado, o programa é executado em modo simples, sem informação de depuração. Valores válidos: `s` (simples) ou `v` (verboso).

4 Testes realizados

Na pasta de testes presente no pacote deste trabalho, há diversos programas que foram utilizados para garantir o bom funcionamento do emulador, cobrindo todas as instruções disponibilizadas pela especificação da máquina virtual. Segue abaixo uma breve descrição do comportamento de cada programa:

- `t_spec.i`: teste disponibilizado na especificação do trabalho, soma 100 a um número fornecido pelo usuário.
- `t_fib.i`: calcula o número de Fibonacci de acordo com um índice fornecido pelo usuário.
- `t_exp.i`: realiza a exponenciação de um número, de acordo com uma base e um expoente, ambos fornecidos pelo usuário.
- `t_div.i`: realiza a divisão por força bruta de dois números inteiros, retornando quociente e resto.
- `t_mdn.i`: determina a mediana de um conjunto de sete itens fornecidos pelo usuário.

O assembly correspondente a cada teste pode ser encontrado em uma pasta interna à pasta de testes.

4.1 Imagens de execução dos testes

```
┌─[±][master U:1 ?:5 ✗][/scratch/victor/sb/sb-emulator]
└─▪ TNAME=t_spec
┌─[±][master U:1 ?:5 ✗][/scratch/victor/sb/sb-emulator]
└─▪ build/emulador 0 1000 0 test-io/$TNAME.i v
----------------------------------------
PC: 0 (3), SP: 1000, PSW: 0
R00:        0 | R01:        0 | R02:        0 | R03:        0
R08:        0 | R09:        0 | R10:        0 | R11:        0
R0> 123
READ R0 (0) <- stdin (123)
----------------------------------------
PC: 2 (1), SP: 1000, PSW: 0
R00:      123 | R01:        0 | R02:        0 | R03:        0
R08:        0 | R09:        0 | R10:        0 | R11:        0
LOAD R1 (0) <- Mem[6 + PC(5)] (100)
----------------------------------------
PC: 5 (8), SP: 1000, PSW: 0
R00:      123 | R01:      100 | R02:        0 | R03:        0
R08:        0 | R09:        0 | R10:        0 | R11:        0
ADD R0 (123) += R1 (100)
----------------------------------------
PC: 8 (4), SP: 1000, PSW: 0
R00:      223 | R01:      100 | R02:        0 | R03:        0
R08:        0 | R09:        0 | R10:        0 | R11:        0
WRITE R0 (223) -> stdout
223
----------------------------------------
PC: 10 (22), SP: 1000, PSW: 0
R00:      223 | R01:      100 | R02:        0 | R03:        0
R08:        0 | R09:        0 | R10:        0 | R11:        0
HALT
```

Figura 1: Execução do teste t_spec.i em modo verboso. Alguns registradores nulos não aparecem na imagem, para permitir melhor visualização.

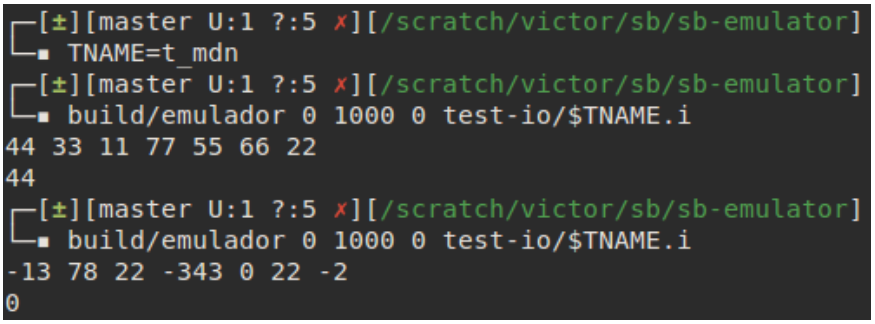

Figura 2: Execução do teste t_mdn.i com dois conjuntos diferentes de entrada.

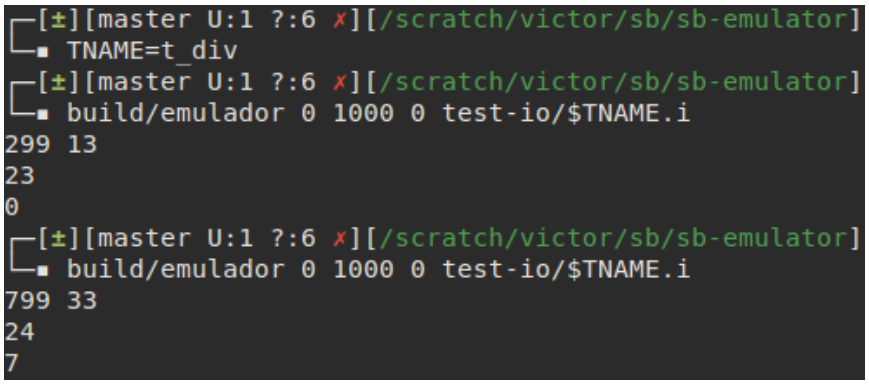

Figura 3: Execução do teste t_div.i com dois pares diferentes de entrada.

5 Conclusão

Neste trabalho, foi implementado um emulador para uma máquina virtual, definido de acordo com a especificação fornecida. O comportamento e a implementação do emulador foram discutidos no contexto dos testes realizados, de forma a garantir que as instruções funcionam como previsto.