TRABALHO PRÁTICO 1: Emulador

Felipe Moraes Gomes - felipemoraes@dcc.ufmg.br

1 Introdução e Definição do Trabalho

Este trabalho descreve a implementação e operação de um simulador para um processador hipotético baseado na arquitetura RISC. A máquina virtual recebe como entrada um binário, e alguns parâmetros de execução, que ele então usa para interpretar o código lido.

O restante deste documento está organizado da seguinte forma: A 2ª seção trata da implementação da máquina virtual e sua organização no código; A 3ª seção explicita o formato de execução, entrada e saída do programa; A 4ª seção contém os testes realizados; A 5ª seção contém algumas considerações finais, concluindo o trabalho. Após isso, é colocado um apêndice contendo uma listagem dos arquivos do projeto.

2 Implementação e Organização

A máquina virtual carrega as instruções do arquivo de entrada especificado e as executa uma a uma até a conclusão do programa. A máquina virtual é implementada por meio de duas funções *Load* e *Execute*, delineada a seguir.

2.1 Dados e Variáveis

O algoritmo principal define várias estruturas de dados, todas de acesso global. Algumas das declarações são do tipo *int*, que é um inteiro de 32 bits.

- *int Mem[1000]*: A memória principal da máquina.
- *int PC, SP*: Os registradores de uso específico; O PC (Program Counter) mantém o índice da instrução sendo executada; O SP (Stack Pointer) aponta para o topo da pilha.
- *int R[8]*: Os 8 registradores de uso geral; Os nomes de cada registrador estão associados a seus respectivos índices pelo pseudônimos $RA = 0$, $RB = 1$, $RC = 2$ e $RD = 3$;
- *char PSW[2]*: Os flags setados durante as operações de $COPY$ e ALU, e usados pelas funções de branch; O primeiro (*PSW[0] - Zero Flag*) é

setado quando o resultado destas operações for 0, e o segundo (*PSW[1] - Sign Flag*) quando o bit de sinal (o mais significativo) for 1.

2.2 Métodos e Funções

- *int Load()*: Confere o arquivo binário e em seguida carrega o programa na memória.
- *int Execute(int code)*: Lê a instrução corrente e seus operandos da memória de acordo com o PC e a executa, retornando o status da execução daquela instrução. As instruções são decodificadas por um *switch*, que efetua as operações do conjunto de instruções, caso o status retornado identifique uma instrução halt, o emulador é finalizado. Quando uma instrução desconhecida é lida, a função não executa nenhuma instrução e incrementa o PC em um para o programa continuar a sua execução na próxima instrução.

2.3 Fluxo de Execução

O programa inicialmente lê os parametros passados a ele, verificando sua validade. Caso o formato esteja correto ele carrega o programa chamando *Load*. Havendo êxito, *Execute* é chamado. Ao retornar, o programa conclui sua execução caso status identifique uma instrução halt.

3 Controle & IO

3.1 Execução e Compilação do Emulador

O programa pode ser compilado através do gcc, pelo utilitário *make*, usando o makefile providenciado. Uma vez compilado, chamadas devem seguir o formato:

./emulador input.mk 's'| 'v'

Onde 's' especifica que a saída deve ser simples (somente com os resultados de execuções de *WRITE* e *READ*), enquanto 'v' especifica que o programa deve também imprimir o banco de registradores, flags e instrução corrente a cada passo. *input* é o nome do arquivo que contém o código binário a ser interpretado.

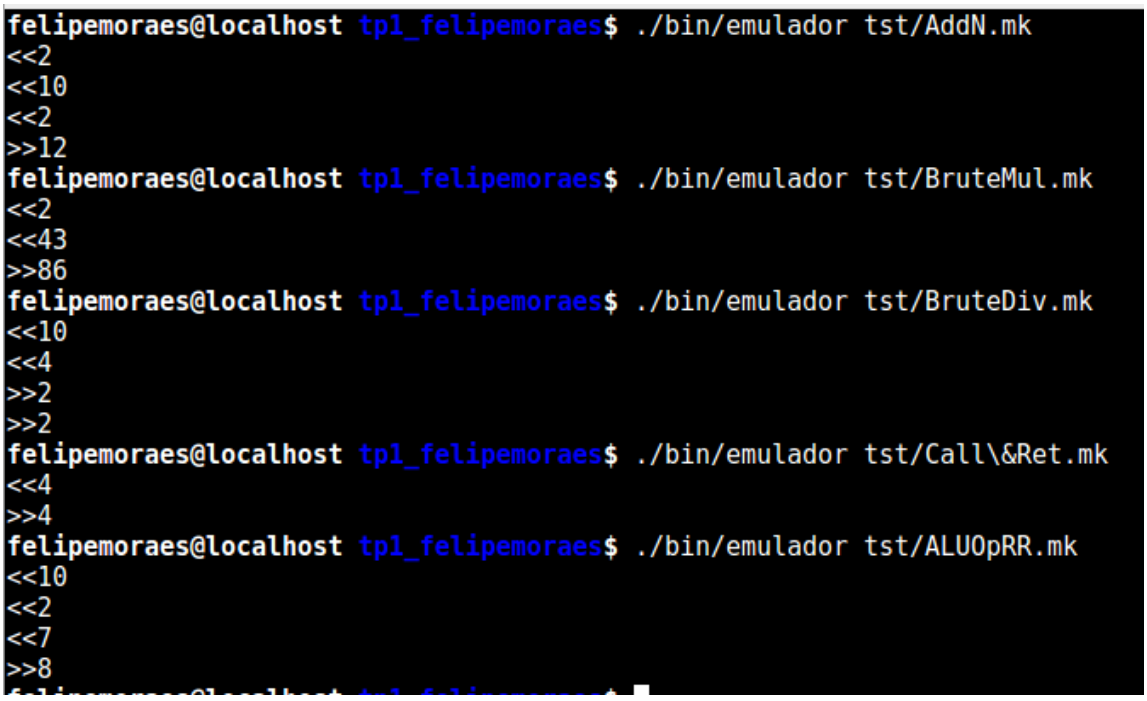

Figura 1: Execução dos testes

3.2 Formato dos Binários

Os programas executados pela máquina virtual são codificados em arquivos binários, onde os dois primeiros bytes contém a assinatura do binário (”M” e ”K”), os próximos 4 bytes o PC e em seguida 4 bytes que corresponde ao valor do s próximos bytes correspondem exatamente a um *int*, que representa um dado ou uma instrução, segundo a codificação apresentada na especificação.

3.3 Interação em Tempo de Execução

São definidas duas instruções que permitem interagir com os programas em tempo de execução, que são *READ* e *WRITE*, que usam a saida padrão (terminal) para se comunicarem com o usuário.

Execuções de *READ* requisitam a entrada de um número na parte do usuário; Execuções de *WRITE* imprime o valor corrente do endereço de memória especificado na instrução.

4 Testes

Múltiplos testes foram efetuados sobre o emulador, de forma a obter cobertura total do código e do conjunto de instruções. Os testes foram executados com *PC= 0* e *SP = 1000*. As execuções de cada programa são ilustradas na Fig. 1.

- *Soma N (AddN.mk):* Calcula e imprime a soma de N números, onde N é o primeiro parâmetro, que vem seguido pelos N números a serem somados.
- *Divisão força bruta (BruteDiv.mk):* Faz a divisão força bruta de dois números (passados como parâmetros), e imprime o quociente e o resto obtido.
- *Multiplicação força bruta (BruteMul.mk):* Faz a multiplicação força bruta de dois números (passados como parâmetros), e imprime o produto obtido.
- *Cobertura (ALUOpRR.mk)*: Testa as instruções não abordadas pelos programas anteriores; O teste lê dois parâmetros A, B e C, e imprime um valor de acordo com o parâmetro C o tipo de operação lógica e aritmética de acordo com o código da instrução de operações Registrador-Registradore.
- *Cobertura (Call&Ret.mk)*: Testa a máquina virtual para chamadas de funções e retorno de parâmetros. Recebe um parâmetro A e retorna o mesmo valor caso todas chamadas foram bem sucedidas.

5 Conclusão

O emulador é um interpretador eficiente e eficaz da linguagem especificada, oferecendo reconhecimento de instruções comparáveis com aquelas fornecidas pelo RISC.

O baixo número de registradores e a falta de endereçamento imediato, no entanto, tornam a arquitetura desta máquina incompatível com problemas de maior escala, que teria de recorrer à memória com uma alta frequência, acarretando em delays intoleráveis para muitas aplicações de uso comum.

Evidentemente, conceitos mais avançados associados a essa arquitetura (como pipeline) foram omitidas pelo fato das mesmas não contribuírem para o melhor reconhecimento da linguagem.

A execução do trabalho transcorreu sem maiores dificuldades, e os resultados obtidos correspondem ao esperado.

A Apêndice

A.1 Listagem de Arquivos

- Código Fonte:
  - *src/main.c:* Interpreta os parâmetros de entrada e controla o fluxo do programa (Seção 2.1).
  - *src/func.c, src/func.h:* Implementa a máquina virtual (Seção 2.2).
- Testes:
  - *tst/AddN.mk:* (Seção 4 item 1).
  - *tst/BruteDiv.mk:* (Seção 4 item 2).
  - *tst/BruteMul.mk:* (Seção 4 item 3).
  - *tst/ALUOpRR.mk:* (Seção 4 item 4).
  - *tst/Call&Ret.mk:* (Seção 4 item 3).