Software Básico - Trabalho Prático

Aluno: Thiago Henrique dos Santos
Matrícula: 2015086590
Data: 30/06/2019

Introdução

O problema proposto consiste em construir um emulador para uma máquina virtual básica e um montador que permite que usuários escrevam programas para essa máquina.

Emulador

A máquina virtual que deve ser emulada possui uma memória de 1000 posições capaz de armazenar apenas números inteiros. Possui também um total de 7 registradores, dos quais 3 são de propósito específico e 4 são de propósito geral. Os registradores de propósito específico são os seguintes:

- PC - Contador de Programa: Contém o endereço da próxima instrução a ser executada.
- AP - Apontador de Pilha: Aponta para o elemento no topo da pilha.
- PEP - Palavra de Estado do Processador: Armazena o estado da última operação lógica/aritmética realizada pelo programa. Consiste em dois bits, um para indicar que operação resultou em 0 e o outro para indicar que resultou em um número negativo.

A máquina virtual é capaz de executar 25 instruções, que estão especificadas no quadro a seguir:

Cód.	Símb.	Oper.	Definição	Ação
0	HALT		Parada	
1	LOAD	R M	Carrega Registrador	Reg[R] ← Mem[M+PC]
2	STORE	R M	Armazena Registrador	Mem[M+PC] ← Reg[R]
3	READ	R	Lê valor para registrador	Reg[R] ← "valor lido"
4	WRITE	R	Escreve conteúdo do registrador	"Imprime" Reg[R]
5	COPY	R1 R2	Copia registrador	Reg[R1] ← Reg[R2] *
6	PUSH	R	Empilha valor do registrador	AP ← AP−1; Mem[AP] ← Reg[R]
7	POP	R	Desempilha valor no registrador	Reg[R] ← Mem[AP]; AP ← AP+1
8	JUMP	M	Desvio incondicional	PC←PC+M
9	JZ	M	Desvia se zero	Se PEP [zero], PC ← PC+M
10	JNZ	M	Desvia se não zero	Se !PEP [zero], PC ← PC+M
11	JN	M	Desvia se negativo	Se PEP [negativo], PC ← PC+M
12	JNN	M	Desvia se não negativo	Se !PEP [negativo], PC ← PC+M
13	CALL	M	Chamada de subrotina	AP ← AP – 1; Mem[AP] ← PC; PC←PC+M
14	RET		Retorno de subrotina	PC ← Mem[AP]; AP ← AP+1
15	AND	R1 R2	AND (bit a bit) de dois registradores	Reg[R1] ← Reg[R1] AND Reg[R2] *
16	OR	R1 R2	OR (bit a bit) de dois registradores	Reg[R1] ← Reg[R1] OR Reg[R2] *
17	NOT	R1	NOT (bit a bit) de um registrador	Reg[R1] ← NOT Reg[R1] *
18	XOR	R1 R2	XOR (bit a bit) de dois registradores	Reg[R1] ← Reg[R1] XOR Reg[R2] *
19	ADD	R1 R2	Soma dois registradores	Reg[R1] ← Reg[R1] + Reg[R2] *
20	SUB	R1 R2	Subtrai dois registradores	Reg[R1] ← Reg[R1] − Reg[R2] *
21	MUL	R1 R2	Multiplica dois registradores	Reg[R1] ← Reg[R1] × Reg[R2] *
22	DIV	R1 R2	Dividendo entre dois registradores	Reg[R1] ← dividendo(Reg[R1] ÷ Reg[R2]) *
23	MOD	R1 R2	Resto entre dois registradores	Reg[R1] ← resto(Reg[R1] ÷ Reg[R2]) *
24	CMP	R1 R2	Compara dois registradores	Reg[R1] – Reg[R2] *
25	TST	R1 R2	Testa dois registradores	Reg[R1] AND Reg[R2] *

Montador

O montador deve ser de dois passos e deve receber como entrada um programa em linguagem simbólica com linhas no seguinte formato:

```
[<label>:] <operador> <operando1> <operando2> [; comentário]
```

As labels, caso existam, devem estar definidas no início da linha e sempre terminam com o caractere ':'. Além disso o montador deve permitir comentários, que são iniciados com o caractere ';'. Os operandos podem ser tanto registradores de propósito geral (R0, R1, R2, R3) como posições de memória do programa, identificadas por labels.

Além do conjunto de instruções definido na tabela acima, o montador deve oferecer as seguintes instruções:

- WORD I : Reserva a posição de memória e a inicializa com o valor I
- END : Indica o final do programa para o montador

Solução

Emulador

Para a solução do emulador foi criado o módulo `virtual_machine`, que contém as variáveis referentes à memória e aos registradores da máquina virtual. Esse módulo também contém a implementação das instruções e o método `execute`, que recebe o código e os operandos da instrução e chama o método correspondente para executá-la.

O emulador é responsável por ler o arquivo de entrada com as instruções, e carregá-lo para a memória na posição indicada. A partir daí ele passa a buscar instruções na memória e executá-las seguindo o seguinte *loop*:

```
ENQUANTO (PC <= enderecoCarregamento + tamanhoPrograma)
      instrucao = MEMORIA[PC]
      operandos = &MEMORIA[PC + 1]

      PC = PC + 1 + NUMERO_OPERANDOS[instrucao]
      EXECUTAR(instrucao, operandos)
```

Para concentrar informações sobre o conjunto de instruções suportado, foi criado o módulo `instructions_info`. Ele contém a definição dos códigos de cada instrução e os vetores `INSTRUCTION_SYMBOLS` e `INSTRUCTION_NUMBER_OF_OPERANDS`, que armazenam respectivamente os símbolos e o número de operandos de cada uma das instruções. Esse módulo é utilizado tanto pelo emulador como pelo montador.

A execução do emulador deve ser feita com o seguinte comando:

```
./emulator [-v] <arquivo>
```

O argumento `<arquivo>` corresponde ao nome do arquivo que contém o programa em linguagem de máquina. A opção `-v` pode ser utilizada para executar o programa no modo *verbose*.

Montador

O montador implementado faz duas passagens no arquivo recebido como entrada. Na primeira passagem o programa mantém o contador `instructionLocationCounter` (`ILC`), que é incrementado a cada instrução ou operando encontrados. Quando uma label é encontrada, seu valor é armazenado no vetor `symbols` e o valor de ILC, que corresponde à sua posição absoluta no programa, é armazenado no vetor `symbolsLocation`.

Para a segunda passagem, o programa reinicia o `ILC` e passa a identificar as instruções e convertê-las para respectivos códigos. Em instruções do tipo `LOAD`, `STORE`, `JUMP` e `CALL`, que recebem labels como operando, o vetor `symbolsLocation` é consultado para que seja calculada a posição relativa da label em relação à posição da instrução atual:

```
POSICAO_RELATIVA = POSICAO_ABSOLUTA - (ILC + 1 + NUMERO_OPERANDOS_INSTRUCAO)
```

A execução do montador deve ser feita com o seguinte comando:

```
./assembler <arquivo>
```

O argumento `<arquivo>` corresponde ao nome do arquivo que contém o programa em linguagem de montagem.

Testes

Para a realização de testes foram implementados em linguagem de montagem os programas propostos e dois outros. Todos eles se encontram no diretórios “testes”.

O primeiro programa recebe 5 números e calcula a mediana deles. Utilizando os números 5, 7, 4, 9 e 3 como entrada, o resultado foi 5.

```
tp $./assembler testes/mediana > testes/mediana_exec
tp $./emulator testes/mediana_exec
5
7
4
9
3
5
```

O segundo programa lê um número N da entrada e imprime o N-ésimo número da sequência de Fibonacci. Utilizando o valor 11 como entrada, o valor exibido foi 55.

```
tp $./assembler testes/fibonacci > testes/fibonacci_exec
tp $./emulator testes/fibonacci_exec
11
55
```

Para testar as instruções de operações lógicas e aritméticas foi implementado um programa que recebe dois números N1 e N2 da entrada e imprime o resultado das seguintes operações:

- N1 AND N2
- N1 OR N2
- NOT N1
- N1 XOR N2
- N1 + N2
- N1 - N2

- N1 x N2
- N1 : N2
- N1 mod N2

Abaixo se encontram os resultados para N1=240 e N2 = 170:

```
tp $./assembler testes/operacoes > testes/operacoes_exec
tp $./emulator testes/operacoes_exec
240
170
160
250
-241
90
410
70
40800
1
70
```

Para testar as instruções PUSH e POP foi implementado um programa que lê 5 números da entrada e os imprime em ordem contrária. Abaixo se encontra o resultado para os números 3, 9, 1, 5 e 10:

```
tp $./assembler testes/pilha > testes/pilha_exec
tp $./emulator testes/pilha_exec
3
9
1
5
10
10
5
1
9
3
```