

Projeto (3º ano de LCC)

VMS, Simular Visual para a máquina de Stack Virtual VM
Relatório de Desenvolvimento

Adriano Campinho
(a79032)

Vasco Leitão
(a79220)

17 de Junho de 2018

Resumo

O objetivo principal deste relatório técnico revolve em explicar e esclarecer os passos que os alunos comprometeram neste projecto e as pretendidas resoluções para cada problema exposto no enunciado fornecido.

Supervisores: Pedro Rangel Henriques e José João
Área: Processamento de Linguagens

Conteúdo

1	Introdução	2
2	Análise e Especificação	3
2.1	Descrição Informal da Máquina	3
2.2	Especificação do Requisitos	4
2.2.1	Parser	4
2.2.2	Processador	4
2.2.3	Interface	4
3	Concepção/desenho da Resolução	5
3.0.1	Parser	5
3.0.2	Processador	5
3.0.3	Interface	5
4	Guia de Utilização e Testes	6
4.1	Modo de utilização	6
4.1.1	Mensagens de erro	6
4.2	Alternativas, Decisões e Problemas de Implementação	6
4.3	Testes realizados e Resultados	6
5	Conclusão	7

Capítulo 1

Introdução

Nos últimos anos, temos usado nas disciplinas de Processamento de Linguagens (e Compiladores) a máquina virtual VM, uma máquina de stack que suporta inteiros, reais e strings e tem uma heap para permitir um uso dinâmico da memória (necessário por exemplo para implementar listas ligadas). A VM é programada em Assembly e que disponibiliza um conjunto de instruções máquina mínimo e muito compreensível, o que a torna pedagogicamente muito relevante como máquina objeto (destino) em tarefas de compilação.

Para que o uso da VM seja realmente um bom instrumento de trabalho nestes cursos, é fundamental que exista um simulador que permita testar (de preferência passo a passo) o código gerado. Atualmente existe, como é de conhecimento de todos, um Assembler para traduzir o Assembly produzido numa lista de códigos máquina e existe um interpretado que executa esse código e que fornece uma interface visual que permite acompanhar a execução verificando a evolução dos vários blocos de memória e dos registo de controlo da máquina.

Esta versão, codificada em Java, está funcional e pode ser usada, mas tem alguns problemas de implementação, nomeadamente o elevado tempo de execução, quando os programas aumentam um pouco.

Neste projeto pretende-se que os alunos reconstruam o simulador com interface interativo e visual para a máquina VM desde o início, em Java ou em C (conforme preferência do grupo) produzindo um executável que evite os problemas atuais.

Análise e Especificação

Trata-se duma máquina de pilhas (por oposição às máquinas de registos),... Esta é composta pelas seguintes 4 pilhas:

OPStack Esta pilha contém `www www www www www www www www www www www www www www www www`
`www www www www www www www`

Heap Esta pilha contém `www www www www www www www www www www www www www www www www`
`www www www www www www www`

[illegible]

A pilha de execução contém valores, que podem ser inteiros, reais ou endereços. As duas heaps contêm, respetivamente, cadeias de caracteres (strings) e blocos estruturados. Cada um destes tipos de dados é referenciado por endereços. Cada bloco estruturado contém um certo número de valores (do mesmo tipo dos valores que se podem encontrar na pilha). Um endereço pode apontar para quatro tipos de informação: para código, para a pilha, para um bloco estruturado ou para uma string. Ao longo da execução do programa são guardados os seguintes quatro registos

- **SP** (Stack Pointer) o registo aponta para o topo corrente da pilha/ para a primeira célula livre da pilha.
- **FP** (Frame Pointer) o registo aponta para o endereço de base das variáveis locais.
- **GP** (Global Pointer) o registo contém o endereço de base das variáveis globais.
- **PC** (Program Counter) o registo aponta para a instrução corrente (da zona de código) por executar.

A pilha de chamada permite guardar as chamadas: contém pares de apontadores (i, f). O endereço i guarda o registo de instrução pc e f o registo fp.

As instruções são designadas por um nome e podem aceitar um ou dois parâmetros. Estes podem ser:

- Constantes inteiras,

- Constantes reais,
- Cadeias de caracteres delimitadas por aspas. Estas cadeias de caracteres seguem as mesmas regras de formatação que as cadeias da linguagem C (em particular no que diz respeito aos caracteres especiais como ; ou \\\),
- Uma etiqueta simbólica designando uma zona no código.

O conjunto da maioria das intrucoes pode ser encontrado no relatorio ... A estas foram adicionadas

2.2 Especificação do Requisitos

Os requisitos deste projeto podem ser divididos em 3 grupos:

2.2.1 Parser

`code ::= < instr > *`

`instr ::= < ident >:`

`| < instratom >`
`| < instrint > < integer >`
`| pushf < float >`
`| (pushs|err) < string >`
`| check < integer >, < integer >`
`| (jump|jz|pusha) < ident >`

`|instratom > ::= add|sub|mul|div|mod|not|inf|infeq|sup|`
`|supeq|fadd|fsub|fmul|fdiv|fcos|fsin|`
`|finf|finfeq|fsup|fsupeq|concat|equal|atoi|atof|`
`|itof|ftoi|stri|strf|`
`|pushsp|pushfp|pushgp|loadn|storen|swap|`
`|writei|writef|writes|read|call|return|`
`|drawpoint|drawline|drawcircle|`
`|cleardrawingarea|opendrawingarea|setcolor|refresh|`
`|start|nop|stop|allocn|free|dupn|popn|`
`< instrint > ::= pushi|pushn|pushg|pushl|load|`
`|dup|pop|storel|storeg|alloc`

2.2.2 Processador

2.2.3 Interface

Capítulo 3

Concepção/desenho da Resolução

3.0.1 Parser

3.0.2 Processador

3.0.3 Interface

Capítulo 4

Guia de Utilização e Testes

4.1 Modo de utilização

Esta máquina destina-se a uma utilização interactiva do utilizador, para visualizar a execução de um programa. A execução poderá ser conduzida passo a passo, ou ser feita de uma só vez. Sendo feita da seguinte forma:

```
vms [opção] ficheiro.vm OU vms opção [ficheiro.vm]
```

O projeto desenvolvido, tem 3 opções de utilização, estas são:

Máquina em Modo Silencioso (sem flag) O programa passado como argumento, e executado de início ao fim, apenas permitindo ao utilizador introduzir input quando pedido;

Máquina em Modo Debug (-d) O modo Debug permite ao utilizador o acesso a mais informação sobre o programa, disponibilizando os seguintes comandos, que permitem uma execução interativa do programa:

- **run:** executa o programa ate ao fim;
- **next 'n':** executa 'n' passos;
- **file 'f':** "carrega"o ficheiro 'f' para ser executado;
- **reload:** recarrega o ficheiro na posição do ultimo ficheiro executado;
- **quit:** termina a execução do simulador;

Máquina em Modo Interface (-g) Coiso

4.1.1 Mensagens de erro

4.2 Alternativas, Decisões e Problemas de Implementação

4.3 Testes realizados e Resultados

Mostram-se a seguir alguns testes feitos (valores introduzidos) e os respectivos resultados obtidos:

Capítulo 5

Conclusão

Conclusão e não conclusão da conclusão