Projeto (3º ano de LCC)

VMS, Simular Visual para a máquina de Stack Virtual VM

Relatório de Desenvolvimento

Adriano Campinho (a79032)

Vasco Leitão (a79220)

20 de Junho de 2018

Resumo

Este documento apresenta o relatório desenvolvido no âmbito da Unidade Curricular de Projecto, pretendendo explicitar o processo de conceção e implementação do mesmo, assim como as escolhas tomadas e as pretendidas resoluções para cada problema exposto no enunciado fornecido.

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

Conteúdo

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.

Capítulo 2

Análise e Especificação

2.1 Descrição Informal da Máquina

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

Code Esta pilha contem o conjunto das instruções do programa, esta e preenchida no inicio da execução do programa, mantendo-se igual ate ao fim da execução do mesmo;

Operation Stack Esta pilha contem valores guardados na maquina, este podem ser do tipo inteiro, real e podem ser enderecos;

Heap Esta pilha contem cadeias de caracteres (strings), cada uma destas referenciado por enderecos;

Call Stack Esta pilha contem blocos estruturados, cada um destes referenciado por enderecoes, cada bloco estruturdo contem um certo numero de valores (do mesmo tipo dos valores que se podem encontrar na Operation Stack)

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 contem 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 contem 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.

Instruções

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 Planeamento e objetivos

O simulador deve ser implementado de forma a funcionar num maior numero de ambientes UNIX possível: deve ser possível correr a simulação de vários modos possibilitando uma simulação interativa de cada programa. Existindo a possibilidade do jogador interagir com a simulação através da linha de comandos ou através de uma Interface gráfica a ser desenvolvida.

Podemos então dividir o projecto em 3 grandes grupos:

- **Parser:** um parser que receba o input (programas a ser simulados) partindo-os em partes para que possam ser geridas por um outro grupo, verificando se o ficheiro esta no formato correto, caso contrario apontando onde estarão os possíveis problemas;
- **Simulador:** um programa que através do input recebido do parser e, se necessário, do input do utilizador, faca a simulação em si do programa;
- **Interface:** uma interface que permita ao utilizador interagir com a execução do simulador, e dando-lhe acesso a informação sobre o estado atual da maquina; Cada vez que o utilizador interatue com a interface (e.g. clicando num botão), esta deve ser atualizada, refletindo as alterações feitas no estado na maquina e na interface.

2.3 Especificação do Requisitos

Esta secção de forma mais especifica os requisitos de cada um dos grupos de desenvolvimento demonstrando os problemas que surgiram durante cada fase:

2.3.1 Parser

Os programas por executar devem obedecer a sintaxe seguinte:

Convenções Lexicais:

- <INTEGER> ::= -?[0-9]+
- <FLOAT> ::= -?[0-9] + "."[0-9]+
- <LABEL> ::= [A-Z][A-Z0-9]*
- <COMMENT> ::= "//".* || "#". *

Funções:

"not" || "$equal$" || "add" || "sub" || "mul" || "div" || "mod" ||
	"inf"		"$infeq$"		"sup"		"$supeq$"	
	"$fadd$"		"$fsub$"		"$fmul$"		"$fdiv$"	
	"$fcos$"		"$fsin$"		"$ftan$"	
	"$finf$"		"$finfeq$"		"$fsup$"		"$fsupeq$"	
	"$padd$"		"$concat$"		"$alloc$"		"$allocn$"		"$free$"	
	"$atoi$"		"$atof$"		"$itof$"		"$ftoi$"		"$stri$"		"$strf$"	

$||"pushi"||"pushn"||"pushf"||"pushs"||"pushg"||"pushl"||"pushsp"||"pushfp"||"pushgp"||"load"||"loadn"||"dup"||"dupn"||$
$||"pop"||"popn"||"storel"||"storeg"||"store"||"storen"||$
$||"check"||"swap"||$
$||"writei"||"writes"||"writef"||"read"||$
$||"jump"||"jz"||"pusha"||$
$||"call"||"return"||$
$||"start"||"nop"||"err"||"stop"$

2.3.2 Processador

2.3.3 Interface

A interface deve permitir ao utilizador executar as seguintes ações:

- Executar 1 passo na execução do programa;
- Executar n passos (passado como argumento) na execução do programa;
- Executar o programa ate ao fim;
- Carregar um programa para ser simulado;
- Recarregar o programa que esta atualmente a ser simulado;
- Em qualquer momento da execução do programa, mostrar o estado actual da maquina (stacks e pointers);
- Permitir ao utilizador introduzir input para o programa quando necessário;
- Permitir ao utilizador introduzir múltiplas linhas de input de uma vez;
- Carregar um ficheiro com o input para ser lido por um programa;

Capítulo 3

Concepção/desenho da Resolução

3.1 Organização do código

Em termos de organização do código, este foi dividido nos seguintes ficheiros:

- **check.sh*** - Coisas
- **README** - Coisas
- **structs/** - Coisas
- **vmsMan.1** - Coisas
- **inst_err** - Coisas
- **lex.l** - Coisas
- **semantic.c** - Coisas
- **syntax.y** - Coisas
- **interface.c** - Coisas
- **makefile** - Coisas
- **semantic.h** - Coisas
- **vms.c** - Coisas

3.1.1 Estruturas

Em termos de estruturas criadas, estas foram divididas nos seguintes grupos (sendo que cada grupo representa um par de ficheiros .h e .c):

- **array** - Coisas
- **callStack** - Coisas
- **code** - Coisas
- **heap** - Coisas
- **opStack** - Coisas
- **types** - Coisas

3.2 Fases de Desenvolvimento

3.2.1 Parser

Falar flex etc.

3.2.2 Processador

Estruturas de dados etc.

3.2.3 Interface

Nao faco ideia

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 de opção) O programa passado como argumento, e executado de inicio 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 no "local"do ultimo ficheiro executado;
- **quit:** termina a execução do simulador;

Máquina em Modo Interface (-g) O modo Interface

4.2 Mensagens de erro

- "Error:"
- "Error: Index out of array"
- "Error: fp == sp"
- "Error: Opening file"
- "Error: You must specify a '-d' or '-g' option, or a program file'man vms' for more information."

- "Error: Invalid type write"
- "Error: Invalid type not"
- "Error: Invalid type equal"
- "Error: Invalid type padd second arg not int"
- "Error: Invalid type padd first argument not adress"
- "Error: Invalid type alocn"
- "Error: Invalid type free"
- "Error: Invalid type atox arg is not heap adress"
- "Error: Invalid type itof"
- "Error: Invalid type ftoi"
- "Error: Invalid type stri"
- "Error: Invalid type strf"
- "Error: Invalid type loadn"
- "Error: Invalid type load"
- "Error: Invalid type dupn"
- "Error: Invalid type popn"
- "Error: Invalid type pon"
- "Error: Invalid type storen"
- "Error: Invalid type jz"
- "Error: Invalid type call"

4.3 Alternativas, Decisões e Problemas de Implementação

4.4 Testes realizados e Resultados

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

Capítulo 5

Conclusão

Neste relatório descreveu-se o processo de conceção e implementação do projecto pedido pelo enunciado, nomeadamente o parser, o simulador e a interface gráfica, alem de descrever em detalhe o funcionamento da maquina referida.

Apesar de estarmos satisfeitos com o trabalho desenvolvido, há ainda espaço para muitas melhorias desta implementação. Enumeram-se algumas ideias que, por falta de tempo, não foram (ainda) concretizadas:

- Criar um conjunto grande de ficheiros de testes, que sirvam tanto para provar/verificar a implementação da maquina, assim como servir de exemplo da correcta instalação da mesma;
- Fazer melhorias no espaçamento da interface gráfica, permitindo a utilização mais fácil em janelas mais pequenas;