

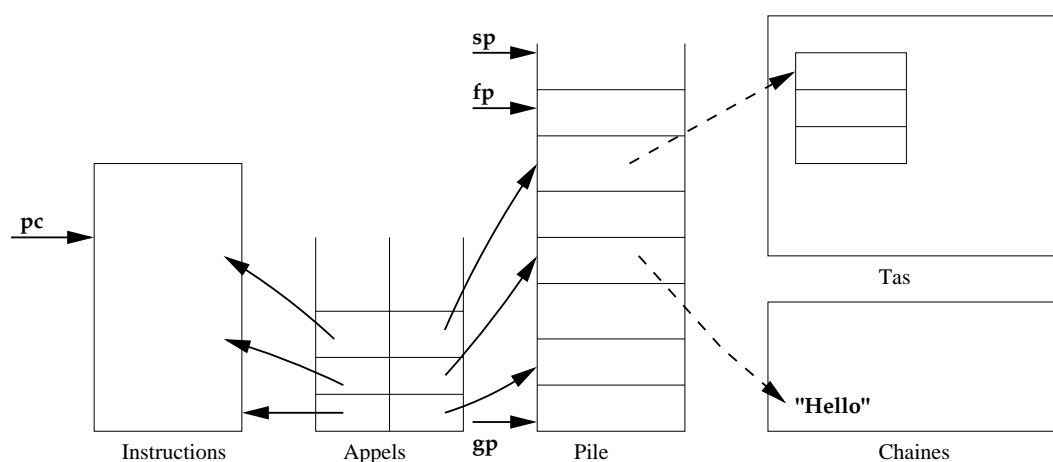
Máquina virtual para o projecto da disciplina de compiladores

Este documento é a tradução da documentação original fornecida com as fontes da máquina virtual aqui descrita.

1 Descrição

1.1 Organização da máquina

Trata-se duma máquina de pilhas (por oposição às máquinas de registos), composta duma pilha de execução, duma pilha de chamadas, duma zona de código, de duas *heaps* e de quatro registos.



A pilha de execução contém *valores*, que podem ser inteiros, reais ou endereços.

As duas *heaps* contêm, respectivamente, 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.

Três registos permitem o acesso a diferentes partes da pilha:

- O registo *sp* (stack pointer) aponta para o topo corrente da pilha. Ele aponta para a primeira célula livre da pilha.
- O registo *fp* (frame pointer) aponta para o endereço de base das variáveis locais.
- O registo *gp* contém o endereço de base das variáveis globais.

A máquina possui um registo *pc* que 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*.

1.2 As 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 `\`, `\n` ou `\\`),
- uma etiqueta simbólica designando uma zona no código.

1.2.1 Convenções

Utilizaremos as convenções seguintes:

- **Empilhar** um valor x significa colocar o valor x na célula $P[sp]$ e incrementar sp de 1.
- Empilhar n vezes um valor x significa iterar n vezes a operação anterior.
- Retirar, ou tirar da pilha n valor consiste em decrementar de n o valor de sp .
- O *topo* da pilha representa o último valor colocado na pilha, ou seja $P[sp-1]$, o valor anterior representa o penúltimo valor, o *sub-topo* colocado na pilha, ou seja $P[sp-2]$.

1.2.2 Operações de base

As operações aritméticas ou flutuantes envolvem os valores do topo e do sub-topo da pilha. Neste caso quando a operação envolvida é executada, os dois argumentos são retiradas da pilha (refira-se à secção das convenções para perceber o que é retirar valores da pilha) e o resultado é então empilhado. O resultado dum operação de comparação é um inteiro que vale 0 ou 1. O inteiro 0 representa o valor booleano *false* enquanto o valor 1 representa o valor *verdade*.

Operações sobre inteiros

Instrução	Descrição
ADD	tira da pilha n e m que devem ser inteiros e empilha o resultado $m + n$
SUB	tira da pilha n e m que devem ser inteiros e empilha o resultado $m - n$
MUL	tira da pilha n e m que devem ser inteiros e empilha o resultado $m \times n$
DIV	tira da pilha n e m que devem ser inteiros e empilha o resultado m/n
MOD	tira da pilha n e m que devem ser inteiros e empilha o resultado $m \bmod n$
NOT	tira da pilha n que deve ser um inteiro e empilha o resultado $n = 0$
INF	tira da pilha n e m que devem ser inteiros e empilha o resultado $m < n$
INFEQ	tira da pilha n e m que devem ser inteiros e empilha o resultado $m \leq n$
SUP	tira da pilha n e m que devem ser inteiros e empilha o resultado $m > n$
SUPEQ	tira da pilha n e m que devem ser inteiros e empilha o resultado $m \geq n$

Operações sobre flutuantes

Instrução	Descrição
FADD	tira da pilha n e m que devem ser reais e empilha o resultado $m + n$
FSUB	tira da pilha n e m que devem ser reais e empilha o resultado $m - n$
FMUL	tira da pilha n e m que devem ser reais e empilha o resultado $m \times n$
FDIV	tira da pilha n e m que devem ser reais e empilha o resultado m/n
FCOS	tira da pilha n que deve ser real e empilha o resultado $\cos(n)$
FSIN	tira da pilha n que deve ser real e empilha o resultado $\sin(n)$
FINF	tira da pilha n e m que devem ser reais e empilha o resultado $m < n$
FINFEQ	tira da pilha n e m que devem ser reais e empilha o resultado $m \leq n$
FSUP	tira da pilha n e m que devem ser reais e empilha o resultado $m > n$
FSUPEQ	tira da pilha n e m que devem ser reais e empilha o resultado $m \geq n$

Operações sobre endereços

Instrução	Descrição
PADD	tira da pilha n que deve ser um inteiro e a que deve ser um endereço e empilha o endereço $a + n$

Operações sobre cadeias de caracteres

Instrução	Descrição
CONCAT	tira da pilha n seguido de m que devem ser endereços de cadeias de caracteres (na heap apropriada) e empilha o endereço de uma cadeia de caracteres igual a concatenação da string presente no endereço n com a string de endereço m .

Operações sobre o heap dos blocos estruturados

Instrução	Argumento	Descrição
ALLOC	n inteiro	aloca no heap um bloco estruturado de tamanho n e empilha o endereço correspondente
ALLOCN		tira da pilha um inteiro n , aloca no heap dos blocos estruturados um bloco de tamanho n e empilha o endereço correspondente.
FREE		tira da pilha um endereço a e liberta o bloco estruturado alocado no endereço a

Igualdade

O teste de igualdade verifica que dois objectos (inteiros, reais ou endereços) da pilha são iguais. Um erro de execução ocorre quando os dois objectos não pertencem ao mesmo tipo. Duas strings são iguais quando são arquivadas no mesmo endereço. A instrução de teste de igualdade permite assim a verificação da igualdade de duas cadeias de caracteres.

Instrução	Descrição
EQUAL	tira da pilha n seguido de m que devem ser do mesmo tipo e empilha o resultado de $n = m$

Conversões

Diferentes instruções permitam converter uma cadeia de caracteres em inteiros ou reais, e reciprocamente.

Instrução	Descrição
ATOI	retira da pilha o endereço dum string e empilha a sua conversão em inteiro. Tal falha quando a string não representa um inteiro.
ATOF	retira da pilha o endereço dum string e empilha a sua conversão em real. Tal falha quando a string não representa um real.
ITOF	retira da pilha um inteiro e empilha a sua conversão para real.
FTOI	retira da pilha um real e empilha a parte inteira (obtida retirando as decimais).
STRI	retira da pilha um inteiro e empilha o endereço dum string representando este inteiro
STRF	retira da pilha um real e empilha o endereço dum string representando este real

1.2.3 Manipular dados

Se x designar um endereço na pilha então $x[n]$ designa um endereço situada n células por cima.

Empilhar

Instrução	Argumentos	Descrição
PUSHI	n inteiro	empilha n
PUSHN	n inteiro	empilha n vezes o valor inteiro 0
PUSHF	n real	empilha n
PUSHS	n string	arquiva n na zona das strings e empilha o endereço
PUSHG	n inteiro	empilha o valor localizado em $gp[n]$
PUSHL	n inteiro	empilha o valor localizado em $fp[n]$
PUSHSP		empilha o valor do registo sp
PUSHFP		empilha o valor do registo fp
PUSHGP		empilha o valor do registo gp
LOAD	n inteiro	retira da pilha um endereço a e empilha o valor na pilha ou no heap (dependendo do tipo de a) em $a[n]$
LOADN		retira da pilha um inteiro n , um endereço a e empilha o valor na pilha ou no heap (dependendo do tipo de a) em $a[n]$
DUP	n inteiro	duplica e empilha os n valores de topo na pilha
DUPN		retira da pilha o inteiro n , duplica e empilha os n valores de topo na pilha

Retirar da pilha

Instrução	Argumento	Descrição
POP	n inteiro	retira n valores da pilha
POPN		retira um inteiro n da pilha e retira n valores da pilha

Arquivar

Instrução	Argumento	Descrição
STOREL	n inteiro	retira um valor da pilha e arquiva-a na pilha em $fp[n]$
STOREG	n inteiro	retira um valor da pilha e arquiva-a na pilha em $gp[n]$
STORE	n inteiro	retira da pilha um valor v e um endereço a , arquiva v em $a[n]$ na pilha ou na heap (dependendo do tipo de a)
STOREN		retira da pilha um valor v , um inteiro n e um endereço a , arquiva v no endereço $a[n]$ na pilha ou na heap

Diversos

Instrução	Argumento(s)	Descrição
CHECK	n, p inteiros	verifica que o topo da pilha é um inteiro i tal que $n \leq i \leq p$. Caso contrário um erro é lançado
SWAP		retira da pilha um valor n seguido de m e empilha n seguido de m

1.2.4 Input-Output

As instruções seguintes permitam a gestão das entradas e saídas.

Instrução	Descrição
WRITEI	retira um inteiro da pilha e imprime o valor na saída standard
WRITEF	retira um real da pilha e imprime o valor na saída standard
WRITES	retira um endereço de uma string da pilha e imprime a string correspondente na saída standard
READ	lê uma string do teclado (concluída por um "\n") e arquiva esta string (sem o "\n") na heap e coloca (empilha) o endereço na pilha..

1.2.5 Primitivas gráficas

As instruções seguintes permitam a realização de desenhos utilizando a versão gráfica da máquina virtual. A origem da janela gráfica está em cima a esquerda.

Instrução	Descrição
DRAWPOINT	retira da pilha m seguida de n (ambos inteiros) e desenha um ponto de coordenada (n, m)
DRAWLINE	retira q, p, m e n da pilha (todos eles inteiros) e desenha o segmento que começa em (n, m) acaba em (p, q)
DRAWCIRCLE	retira da pilha p, m e n (todos eles inteiros) e desenha um círculo de centro (n, m) e de raio p
OPENDRAWINGAREA	retira os inteiros h e w da pilha e abre uma nova janela gráfica de largura w e de altura h
CLEARDRAWINGAREA	remove a saída gráfica e reinicializa a cor corrente para o preto
SETCOLOR	retira da pilha os valores inteiros b, g e r e muda a cor corrente para a cor cujo o valor RGB é determinada pelos três inteiros r, g e b (cada valor entre 0 e 65535)
REFRESH	refresca a janela gráfica, i.e. torna invisível qualquer operação gráfica efectuada desde a última reinicialização.

1.2.6 Operações de controlo

Após a execução de uma instrução, o registo pc é normalmente incrementado de 1. As operações de controlo permitam uma modificação deste comportamento.

Modificação do registo pc

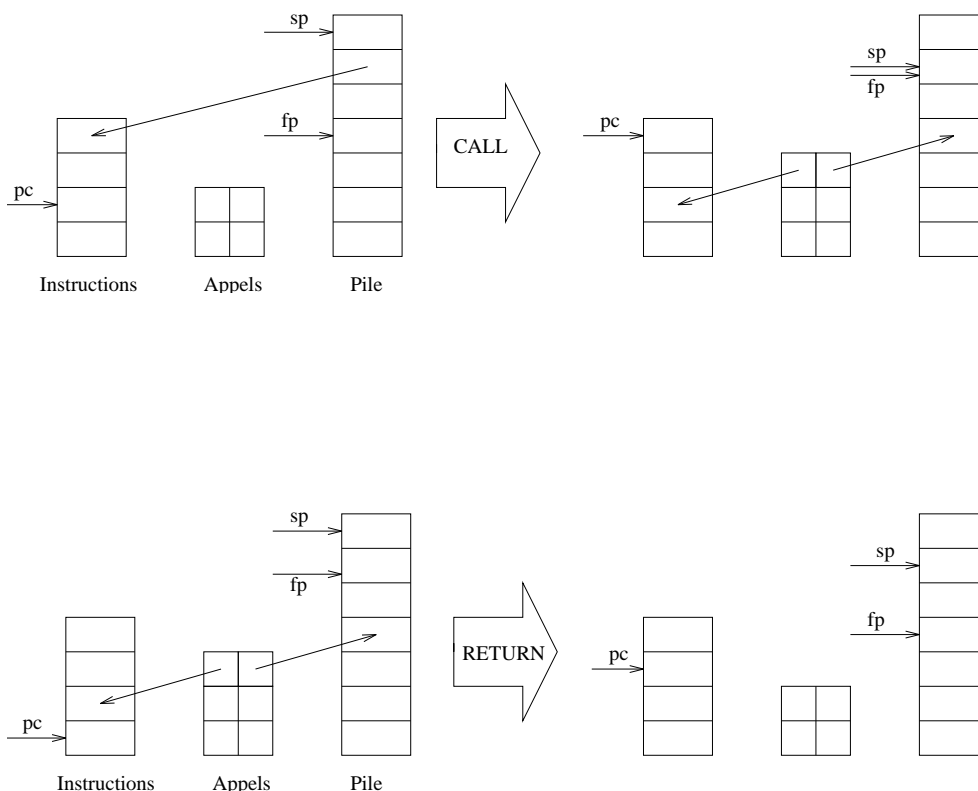
Instrução	Argumento	Descrição
JUMP	<i>label</i> etiqueta	atribui ao registo pc o endereço no código que corresponde a <i>label</i> (pode ser um inteiro ou um valor simbólico).
JZ	<i>label</i> etiqueta	retira da pilha um valor. Se este for nulo então é atribuído ao registo pc o endereço correspondente à <i>label</i> , incrementa simplesmente pc de 1, caso contrário.
PUSHA	<i>label</i> etiqueta	empilha o endereço de programa correspondente a etiqueta <i>label</i>

Procedimentos

Durante a chamada de procedimentos é necessário salvaguardar os registos de instruções e de variáveis locais que serão restaurados no momento do retorno.

Instrução	Descrição
CALL	retira da pilha um endereço de programa a , salvaguarda pc e fp na pilha das chamadas, afecta a fp o valor corrente de sp e a pc o valor de a .
RETURN	afecta a sp o valor corrente de fp , restaura da pilha de chamadas os valores de fp e de pc , incrementa pc de 1 por forma a encontrar a instrução a seguir a chamada.

O comportamento das instruções CALL e RETURN pode ser descrito pelas figuras seguintes, onde só constam os registo de facto modificados pela acção das instruções:



1.2.7 Inicialização e fim

No estado inicial, o registo pc aponta para a primeira instrução do programa. A pilha das chamadas e a pilha de execução estão vazias. Os registos gp e sp apontam para a base da pilha, o registo fp está indefinido. Este registo deve ser inicializado pela instrução START que só poderá ser utilizada uma única vez. As instruções seguintes são utilizadas para parar a máquina no fim do programa ou em caso de erro.

Instrução	Argumento	Descrição
START		Afecta o valor de <i>sp</i> a <i>fp</i>
NOP		não faz nada.
ERR	<i>x</i> string	levanta um erro com a mensagem <i>x</i> .
STOP		para a execução do programa

2 Implementação

Uma implementação da máquina virtual é fornecida sob a forma de dois programas *vm* e *gvm*.

2.1 Sintaxe concreta

Os programas por executar devem obedecer a sintaxe seguinte.

2.1.1 Convenções lexicais

Espaços, tabulações e *carrier-return* formam os espaços autorizados. Os comentários começam por // e vão até ao fim da linha corrente. Os identificadores seguem a expressão regular *<ident>* seguinte :

$$\begin{aligned}
 \langle digit \rangle & ::= 0-9 \\
 \langle alpha \rangle & ::= a-z \mid A-Z \\
 \langle ident \rangle & ::= (\langle alpha \rangle \mid _)(\langle alpha \rangle \mid \langle digit \rangle \mid _ \mid ')*
 \end{aligned}$$

As constantes inteiras e reais são definidas pelas expressões regulares seguintes:

$$\begin{aligned}
 \langle integer \rangle & ::= -? \langle digit \rangle + \\
 \langle float \rangle & ::= -? \langle digit \rangle + (\cdot \langle digit \rangle +)^? ((e \mid E) (+ \mid -)? \langle digit \rangle +)^?
 \end{aligned}$$

com a convenção que uma constante é real se e só se não é uma constante inteira (um real tem então pelo menos um ponto decimal ou um expoente).

As strings são delimitadas pelo caracter ", e podem conter este mesmo caracter se estiver precedido do caracter \. Ou seja:

$$\langle string \rangle ::= " ([^ "] \mid \backslash ")^* "$$

Todos os identificadores constituindo as instruções (ver sintaxe a seguir) formam as palavras reservadas e são *case-insensitive*.

2.1.2 Sintaxe

Os programas seguem a sintaxe dada na figura 1.


```

<code> ::= <instr>*
<instr> ::= <ident> :
          | <instr_atom>
          | <instr_int> <integer>
          | pushf <float>
          | (pushs | err) <string>
          | check <integer> , <integer>
          | (jump | jz | pusha) <ident>
<instr_atom> ::= 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 |
<instr_int> ::= pushi | pushn | pushg | pushl | load |
                | dup | pop | storel | storeg | alloc

```

Figura 1: Sintaxe dos programas

2.2 Utilização

2.2.1 Máquina em modo texto vm

Esta máquina é destinada a uma utilização não-interactiva. Utiliza-se da seguinte forma:

```
vm [opções] [ficheiro.vm]
```

Quando o ficheiro é omitido, o código é obtido da entrada standard. As opções são as seguintes:

-dump

mostra algumas informações sobre a máquina no fim da execução do programa (valor dos registos, topo da pilha, etc.)

-silent

execução silenciosa

-count

mostra o número de instruções executadas

-ssize *inteiro*

estabelece o tamanho da pilha (10000 por defeito)

-csize *inteiro*

estabelece o tamanho da pilha de chamadas (100 por defeito)

2.2.2 Interface gráfica gvm

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

```
gvm [opções] [ficheiro.vm]
```

As únicas opções são, a semelhança da máquina `vm`, `-ssize` e `-csize`.

Só os elementos presentes debaixo de SP são visualizados (SP tem então por valor o índice da primeira linha vazia). A linha verde corresponde a FP. Os valores de SP, FP e PC são mostrado em baixo a direita. A execução dum `READ` provoca a abertura duma janela de introdução de dados e a execução de instruções `WRITE` tem por efeito a escrita duma mensagem na barra inferior.

2.3 Mensagens de erro

Os diferentes erros de execução possíveis são:

Illegal Operand

provocado quando o(s) valores retirados da pilha não têm o tipo esperado

Segmentation Fault

provocado por um acesso indevido a zona de código, as duas heaps ou a pilha.

Stack Overflow

provocada por qualquer tentativa de empilhamento numa pilha cheia (de execução ou de chamada)

Division By Zero

provocada aquando duma divisão inteira por zero.

Error "*mensagem*"

provocada pela instrução `err`

Anomaly

Este erro não deverá ser produzida. Tratar-se-á provavelmente dum bug da máquina virtual.