

Compiladores – Geração de Código

Fabio Mascarenhas – 2017.1

<http://www.dcc.ufrj.br/~fabiom/comp>

Máquinas de Pilha

- Uma máquina de pilha é um tipo de processador em que todos os valores temporários são armazenados em uma pilha
 - Não são usados registradores
- Toda operação em uma máquina de pilha desempilha seus operandos, faz a operação e empilha o resultado
- Instruções também podem empilhar valores constantes, ou o conteúdo de variáveis locais e endereços da memória (variáveis globais)
- Compilar para máquinas de pilha é bem fácil, mas menos eficiente que usar registradores

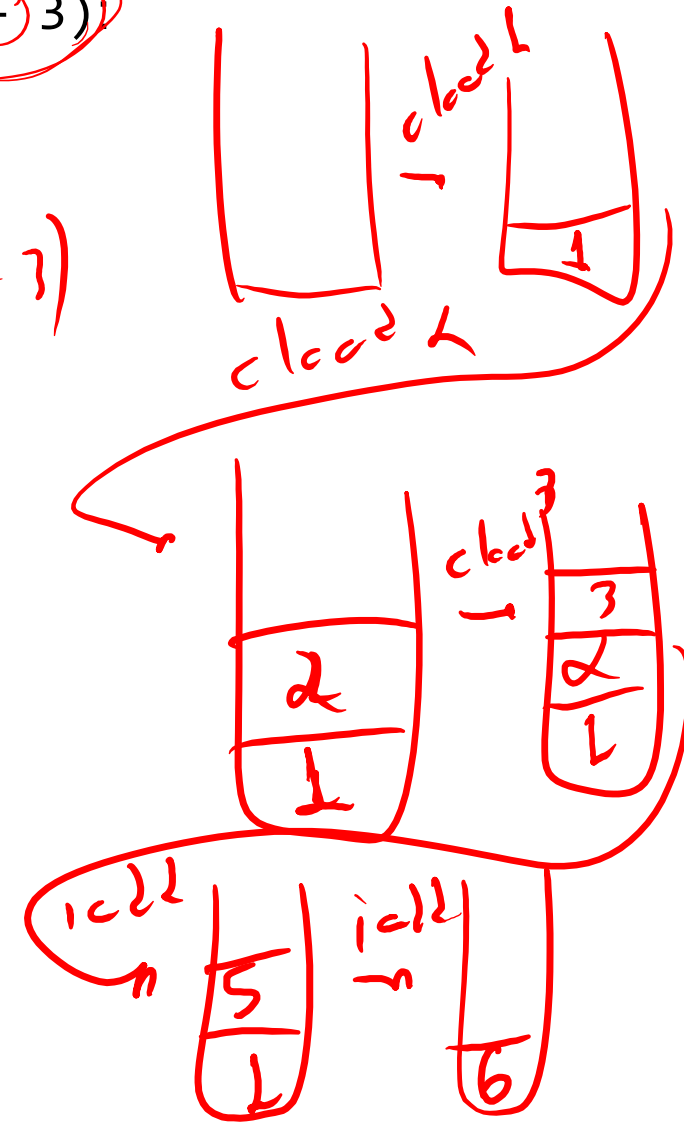
Compilando expressões

- Para ter uma intuição de como a geração de código funciona para uma máquina de pilha, vamos gerar código para $1 + (2 + 3)$.

cloud 1
cloud 2
cloud 3
iadd
iadd

} 2 + 3

} 1 + (2 + 3)



Geração de Código para TINY em x64

- Vamos usar um modelo de máquina de pilha para gerar código para TINY com procedimentos para x64
- As instruções de nossa máquina de pilha serão implementadas por instruções de x64, usando a pilha do processador como a pilha
- Nossa máquina de pilha terá as instruções: *globais* *liberais* *locais*
chamada
getglobal, putglobal, cload, iload,
fload, istore, fstore, iadd, fadd, isub, fsub, imul, fmul, idiv, fdiv, invoke,
if_icmpneq, if_fcmpneq, if_icmpge, if_fcmpge, jmp, iread, fread, bread, iwrite,
fwrite, bwrite, pop, iret, fret *I/O*
- A organização e nomes lembram os de máquinas virtuais de pilha, como a JVM

Contexto de Geração de Código

- Vamos criar uma classe para ser o *contexto* de geração de código
- O contexto implementa as instruções da máquina de pilha, gerando código x64 para elas em um buffer
- Vamos usar um contexto para cada procedimento, e depois costurar o código dos procedimentos junto com o código do corpo principal do programa e o código que declara variáveis globais
- Ele gerencia também os *labels* do programa, usados nas instruções de salto
- Os métodos de geração de código da AST só vão precisar de preocupar em chamar os métodos do contexto que correspondem às instruções da máquina

Tabelas de Símbolos e Endereços

- A geração de código também precisa de tabelas de símbolos que irão mapear nomes de variáveis e procedimentos em *endereços*
- O endereço diz se ela é global ou local, e como acessá-la: com seu *nome simbólico*, se a variável é global, ou com sua *profundidade*, se ela é local
- Com o local de uma variável podemos gerar código para empilhar seu valor, ou para desempilhar o que está no topo da pilha e escrevê-lo na variável
- O contexto de geração de código precisa também guardar uma *marca d'água*, a profundidade mais alta usada para uma local naquele contexto

Geração de Código - Expressões

- Expressões devem deixar a pilha com um elemento a mais no topo: o valor final da expressão
- A geração depende do contexto e da tabela de símbolos de endereços, como nos comandos

$x + (2 + y)$



```
getglobal x  
cload 2  
load 1      ; y é local 1  
iadd  
iadd
```

- Cada subexpressão da expressão acima tem o efeito de empilhar o seu valor; ao fim a pilha será a original, mais o valor de toda a expressão

Geração de Código - Comandos

- Comandos precisam deixar a pilha do mesmo jeito que encontraram
- A geração depende do *contexto* corrente, e da tabela de símbolos que associa nomes a endereços

```
read x;  
read y;  
y := x + (2 + y);  
write y
```



```
iread  
putglobal x  
iread  
istore 1      ; y é local 1  
getglobal x  
cload 2  
load 1        ; y  
iadd  
iadd  
istore 1      ; y  
iload 1       ; y  
iwrite
```