Compiladores

Geração de Código

Bruno Lopes

Instituto de Computação
UFF

Abstração de máquinas de pilha

- Usa uma pilha para operandos e resultados intermediários
- Não considera variáveis ou registradores de uso geral

Instituto de Computação

Abstração de máquinas de pilha

Cada instrução

- Pega seus operandos do topo da pilha
- Remove esses operandos da pilha
- Computa alguma operação com eles
- Coloca o resultado na pilha

Instituto de Computação

Exemplo de programa

Considere duas instruções

push i: coloca i no topo da pilha

add: remove (pop) dois elementos, os adiciona e coloca o resultado de volta na pilha

Um programa para computar $7 + 5$.

Instituto de Computação

Utilidade?

- Cada operação pega seus operandos do mesmo lugar e os coloca no mesmo local
- O que produz um esquema de compilação uniforme

Um compilador mais simples!

Instituto de Computação

Utilidade

A localização dos operandos é implícita

- Sempre no topo da pilha
- Não precisa especificá-los explicitamente

Não precisa especificar a localização do resultado
Instrução “add”, ao invés de “add r1, r2”

Instituto de Computação

Otimização

- A instrução “add” faz 3 operações na memória
- Ideia: manter o topo da pilha em um registrador específico (acumulador)

Instituto de Computação

De máquinas de pilha para MIPS

- O compilador gera código para uma máquina de pilhas com acumulador
- Queremos executar o código resultante em um processador MIPS (ou um simulador)
- Simulamos as intruções da máquina de pilha usando instruções MIPS e registradores

Como simular?

Instituto de Computação

MIPS assembly

- Operações aritméticas usam registradores para operandos e resultados
- Instruções load e store para usar operandos e colocar resultados em memória
- Registradores de 32 bits
- Usaremos $sp, $a0 and $t1 (temporário)

Como simular?

Instituto de Computação

```
li $a0 7
sw $a0  0($sp)
addiu $sp $sp -4
li $a0 5
lw $t1 4($sp)
add $a0 $a0 $t1
addiu $sp $sp 4
```

Instituto de Computação

```
acc <- 7
push acc
acc <- 5
acc <- acc + top_of_stack
pop
```

Instituto de Computação

- Como avaliar uma constante?
- Código para add?

Instituto de Computação

```
cgen(e1 + e2) =
  cgen(e1)
  sw $a0 0($sp)
  addiu $sp $sp - 4
  cgen(e2)
  lw $t1 4($sp)
  add $a0 $t1 $a0
  addiu $sp $sp 4
```

Instituto de Computação

```
cgen(e1 + e2) =
  cgen(e1)
  print "sw $a0 0($sp)"
  print "addiu $sp $sp - 4"
  cgen(e2)
  print "lw $t1 4($sp)"
  print "add $a0 $t1 $a0"
  print "addiu $sp $sp 4"
```

Instituto de Computação

Geração de Código

- Template com "buracos" para preencher com código que avalia expressão
- Código para máquinas de pilha é recursivo
- Geração de código pode ser escrita como uma AST recursiva descendente (ao menos para expressões)

Instituto de Computação

Condicionais

```
cgen(if e1 = e2 then e3 else e4) =
  cgen(e1)
  sw $a0 0($sp)
  addiu $sp $sp -4
  cgen(e2)
  lw $t1 4($sp)
  addiu $sp $sp 4
  beq $a0 $t1 true_branch
  false_branch:
    cgen(e4)
    b end_if
  true_branch:
    cgen(e3)
end_if:
```

Instituto de Computação

Registro de ativação

- Código para chamadas e definições de funções depende do layout do RA
- O resultado está sempre no acumulador
- Não tem necessidade de armazenar o resultado no RA
- O RA mantém os parâmetros atuais

Instituto de Computação

Registro de ativação

- A pilha sempre mantém o mesmo estado imediatamente antes e depois de uma operação
- É necessário guardar o endereço de retorno
- Um ponteiro para a ativação corrente é útil

Instituto de Computação

Chamadas de funções

- O chamador salva seu valor de fp
- Então ele salva os parâmetros correntes em ordem reversa
- Salva o endereço de retorno no registrador $ra
- O RA tem até agora 4*n+4 bytes
- Chamada para definição de funções?

Instituto de Computação

Geração de código para variáveis

- O valor de $sp é alterado de acordo com modificações oriundas de resultados intermediários
- Argumentos de funções não estão em um deslocamento fixo a partir do $sp
- Mas estão em um deslocamento fixo do fp

Instituto de Computação

Como está a pilha?

```
def f(x,y,z) =
  if x
  then g(y)
  else g(z)
def g(t) =
  t + 1
```

Instituto de Computação

Código?

```
def sumto(x) =
  if x = 0
  then 0
  else x + sumto(x-1)
```

Instituto de Computação

Temporários

- Manter temporários no RA
- O gerador de código deve associar uma localização fixa para cada temporário no RA
- Salvar e recuperar os valores sem ter que usar o sp para fazer essas manipulações

Instituto de Computação

- A geração de código deve saber quantos temporários serão usados em cada ponto
- Um novo argumento é adicionado na geração de código: a posição do próximo temporário disponível
- A área de temporários é usada como uma pequena pilha de tamanho fixo

Instituto de Computação

Linguagens OO

- Slogan em OO: se B é uma subclasse de A, então um objeto da classe B pode ser usado sempre que um objeto da classe A for esperado
- Código na classe A deve funcionar sem modificações para um objeto da classe B
- Como representar objetos na memória?
- Como implementar invocação dinâmica?

Similar a struct de C!

Instituto de Computação

Linguagens OO

- Slogan em OO: se B é uma subclasse de A, então um objeto da classe B pode ser usado sempre que um objeto da classe A for esperado
- Código na classe A deve funcionar sem modificações para um objeto da classe B
- Como representar objetos na memória?
- Como implementar invocação dinâmica?

Similar a struct de C!

Instituto de Computação