Geração de Código

Bom dia!

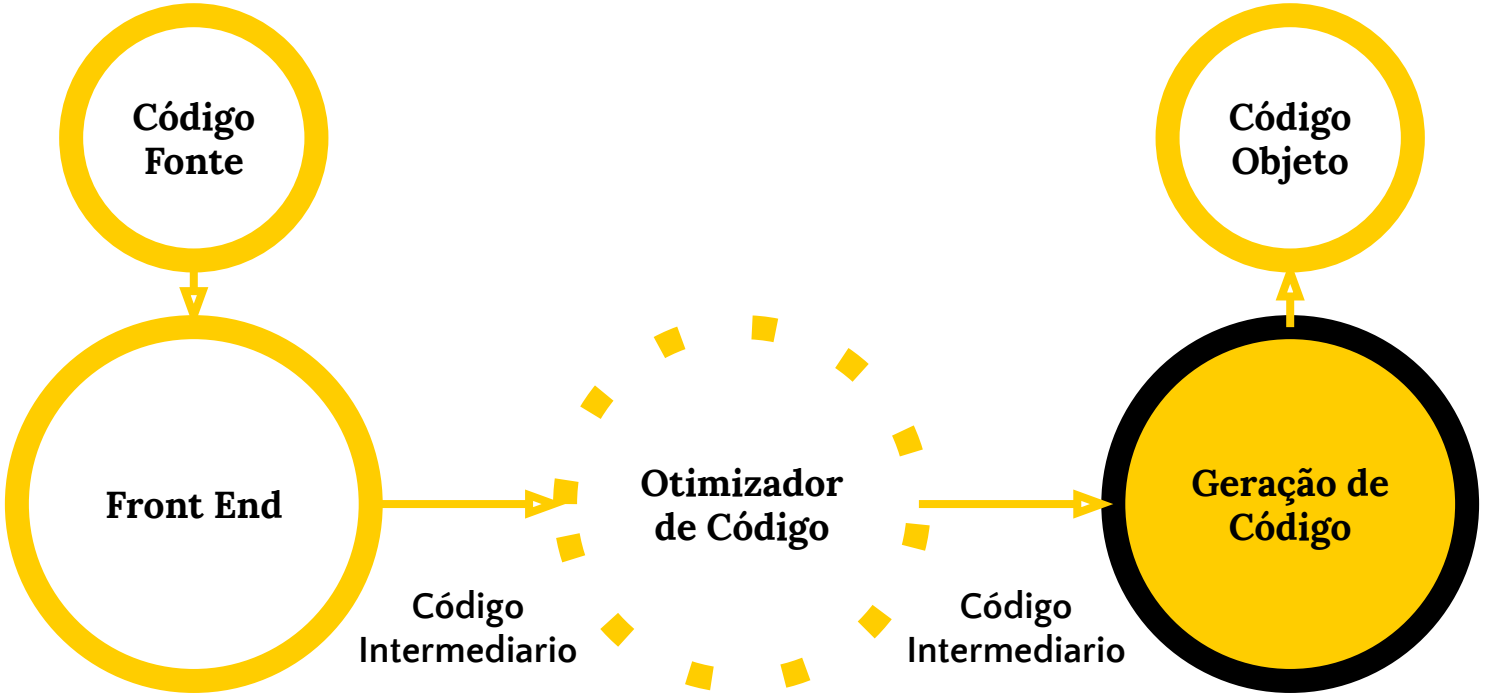
Alexandre* & *Matheus

Pode nos encontrar em:

github/pedrecal

github/matalmeida

Passos do Compilador



1 Gerador de Código

Gerador de Código

Necessidades

Preservar o significado semântico do código fonte;

Fazer uso efetivo dos recursos disponíveis na máquina alvo;

O gerador de código tem que ser eficiente.

Desafios

O problema de geração do código objeto otimizado é indecidível;

Muitos subproblemas encontrados na geração de código são computacionalmente intratáveis.

Tarefas principais

- Seleção de Instrução: Escolher as instruções apropriadas para a implementar a RI;
- Alocação de registrados e atribuição: Decidir quais valores manter em cada registrador;
- Ordenar instruções: Planejamento da ordem em que as instruções serão executadas.

2 Problemas de Design

Entrada para o Gerador

- A não padronização da RI
  - Representações de três endereços
  - Representações de Máquina Virtual
  - Representações Lineares
  - Representações Gráficas

Problemas de Design (Código Objeto)

- Arquitetura das instruções (RISC, CISC);
- Produção de programa absoluto em linguagem de máquina;
- Produção de programa relocável em linguagem de máquina;
- Produção de programa em linguagem de montagem.

Problemas de Design (Seleção de Instruções)

A complexidade de mapear a RI para uma sequência de código para a máquina alvo depende de:

- Nível da RI (Alto-nível ou Baixo-nível);
- Natureza do grupo de instruções (Suporte de tipos de dados);
- Qualidade desejada do código gerado (Velocidade e Tamanho).

Problemas de Design (Seleção de Instruções)

Código Fonte

```
X = Y + Z
```

Código Gerado

```
LD R0, Y
ADD R0, R0, Z
ST X, R0
```

Problemas de Design (Seleção de Instruções)

Código Fonte

```
A = B + C
D = A + E
```

Código Gerado

```
LD R0, B
ADD R0, R0, C
LD R0, A
ADD R0, R0, E
ST D, R0
```

Problemas de Design (Registradores)

- Selecionar um grupo de variáveis que será mantida nos registradores em cada ponto do programa;
- Escolher o registrador específico que aquela variável irá ficar.

Problemas de Design (Ordem de Evaluação)

- Selecionar a ordem na qual ocorrerá computação;
- Afeta a eficiência do código objeto;
- Escolher a melhor ordem é NP-Completo;
- Algumas ordens usam menos registradores que outras.

3 A Linguagem Objeto

Modelo Simples de Máquina Alvo (Exemplo do livro)

- Carregar/armazenar
  - `LD dest, end`
  - `ST X, r0`
- Operações Computacionais
  - `OP dest, fonte1, fonte2`
- Desvios
  - Condicionais: `Bcond r0, L1`
  - Incondicionais: `BR L1`

Modelo Simples de Máquina Alvo (Exemplo do livro)

- Modos de endereçamento
  - Nome de variavél x
  - a(r) significa:
    - conteúdo(a + conteúdo(r))
  - *a(r) significa:
    - conteúdo(conteúdo(a + conteúdo(r)))
  - Imediato: #constante
    - ex: `LD R1, #100`

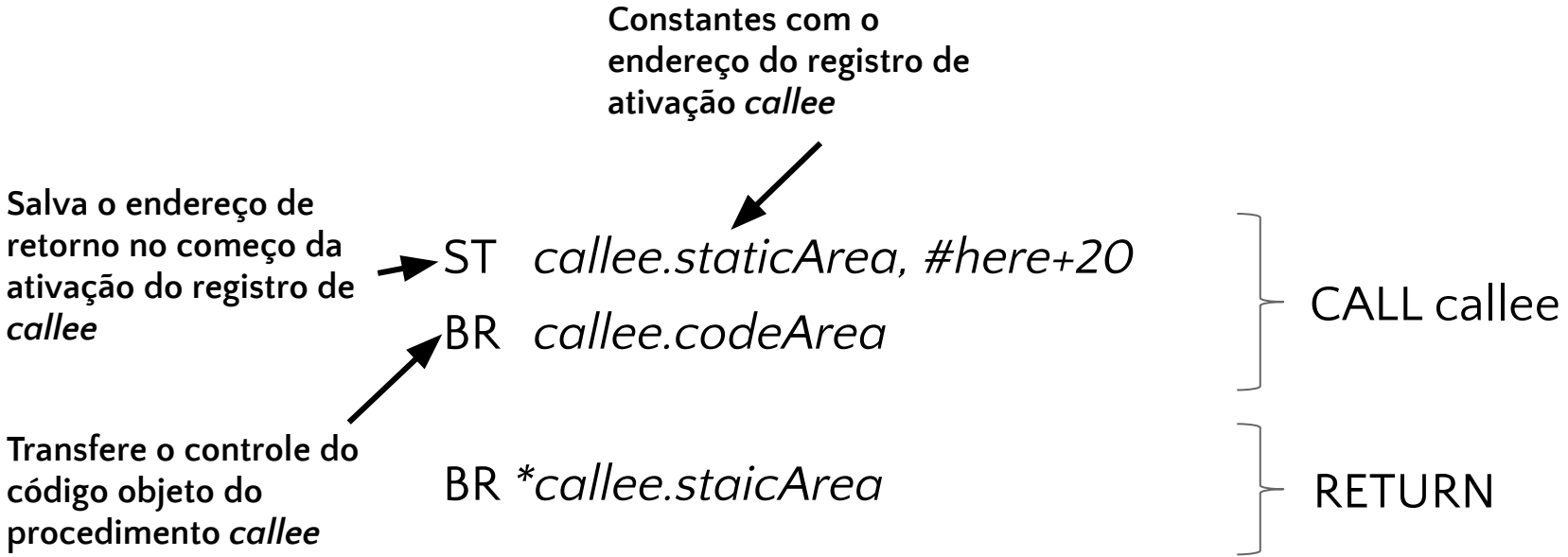
Modelo Simples de Máquina Alvo (Exemplo do livro)

- Custo
  - Custo de uma instrução = 1 + custo do operando
    - Ex: `LD R0, R1` => Custo 1
  - Custo do operando de registro = 0
    - Ex: `a(r)` => Custo 0
  - Custo envolvendo memória e constantes = 1
    - Ex: `LD, 100(R2)` => Custo 2
  - Custo do programa = soma dos custos de instruções

Geração de Código para lidar com a Pilha

Tamanho e layout dos registros de ativação são determinados pelo gerador de código usando informações da tabela de symbolo.

Geração de Código para lidar com a Pilha



```
action1
call p
action2
halt

action3
retorno
```

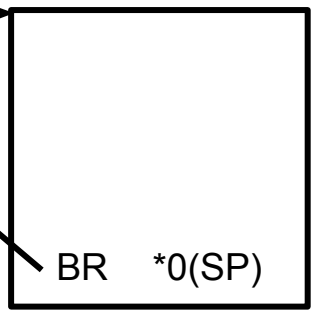
```
100: ACTION1      // Código para action1
120: ST 364, #140 // Salva e retorna o endereço 140 na
localização 364
132: BR 200       // Chama p
140: ACTION2
160: HALT         // Retorna para o Sistema Operacional
…
200: ACTION3
220: BR *364      // Retorna para o endereço salvo em 364
…
300:              // 300-364 mantém o registro de ativação para c
…
364:              // 364-451 mantém o registro de ativação para p
```

Alocação de Pilha

- Não se tem a posição do registro de ativação até a hora de execução
- Se faz necessário o uso de endereço relativo para acessar os elementos do registro de ativação
- Precisamos de um registrador apontando para o topo da pilha

```
LD   SP, #stackStart
ADD  SP, SP, #callee.recordSize
ST       *SP, #here + 16
BR       callee.codeArea

SUB      SP, SP, #callee.recordSize
HALT
```



4 Blocos Básicos e Grafos de Fluxo

Blocos Básicos - Regras

- O fluxo de controle só pode entrar no bloco básico pela primeira instrução deste bloco
- O controle sairá do bloco somente na última instrução

Gerando Blocos Básicos

- A primeira instrução é um líder
- A primeira instrução, dada por um desvio, é um líder
- A primeira instrução, imediatamente após um desvio, é um líder

Código Fonte

```
for i from 1 to 10 do
   for j from 1 to 10 do
      a[i,j] = 0.0;
for i from 1 to 10 do
   a[i,j] = 1.0;
```

Código Intermediário

```
 1)  i = 1
 2)  j = 1
 3)  t1 = 10 * i
 4)  t2 = t1 + j
 5)  t3 = 8 * t2
 6)  t4 = t3 - 88
 7)  a[t4] = 0.0
 8)  j = j + 1
 9)  if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

5 Loops

Loops

- A maioria dos programas gasta a maior parte do seu tempo executando loops
- É muito importante gerar bons códigos para loops

Obrigado!