

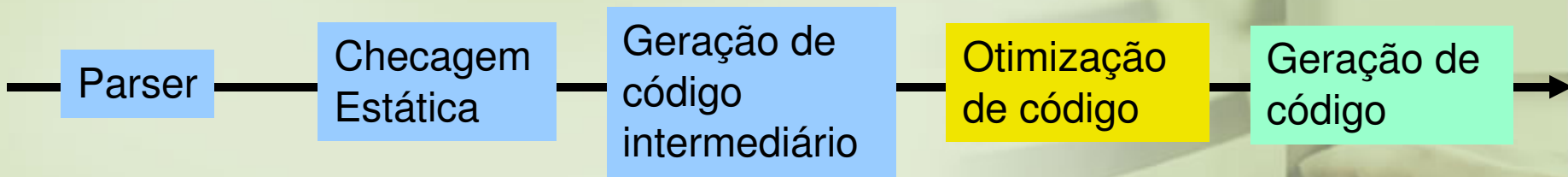
Compiladores

Otimização de código



Otimização de código

- Recebe uma representação intermediária e produz como saída código de máquina eficiente



Otimização de Código

- **Aspectos conflitantes:**
 - uso de memória
 - velocidade de execução
- **Código ótimo:**
 - problema não decidível.
 - utilizar heurísticas
 - “Melhoria” em vez de “otimização”!



Otimização de Código

- **Otimização do código intermediário**
 - eliminar atribuições redundantes
 - suprimir subexpressões comuns
 - eliminar temporários desnecessários
 - trocar instruções de lugar
- **Otimização do código objeto.**
 - troca de instruções de máquina por instruções mais rápidas
 - melhor utilização de registradores.

Otimização de código

- Usuário

```
int x[100000];
main()
{
    int i;
    for (i=0; i < 100000;
        i++)
        x[i] = 1;
}
```



Otimização de código

• Usuário

```
int x[100000];
main()
{
    int i;
    for (i=0; i < 100000; i++)
        x[i] = 1;
}
```

```
int x[100000];
main()
{
    register int *p;
    for (p=(int *)x;p<(int*)x+100000;)
        *p++ = 1;
}
```

Otimização de Código Intermediário

- otimização de blocos seqüenciais de código intermediário
 - redução do número de instruções através de grafos acíclicos orientados
 - **Blocos Básicos** (B. B.)
- transformações de expressões aritméticas
 - visando ao uso otimizado de registradores.

Blocos

- **Comando “líder”:**
 - referido por um goto ou quando segue a um goto (caso em que deverá possuir um rótulo).
 - primeiro comando de um programa também é um “líder”.
- **Bloco básico:**
 - trecho de programa
 - inicia por um comando “líder”
 - Vai até o líder seguinte
 - não apresenta comandos de desvio,
 - a não ser eventualmente o último, o qual pode ser um comando de desvio.
- **Grafo de fluxo:**
 - DAG que indica o fluxo de controle entre BB

Geração de DAG

- **Algoritmo:**

Considere três tipos de instruções:

(i) $x = y \text{ op } z$ (ii) $x = \text{op } y$ (iii) $x = y$

Para todas instruções do bloco básico:

SE nodo y não existe no grafo

ENTÃO criar nodo folha para y

SE instrução for tipo (iii) e z não existe no DAG

ENTÃO criar nodo folha para z

SE instrução for tipo (i) e existe nodo para o operador op no grafo com filhos y e z na mesma ordem

ENTÃO chamar o nodo também de x

SE instrução for tipo (ii) e existe um nodo op com um único filho y

ENTÃO chamar o nodo de x

SENÃO criar o nodo com orientação para y

SE instrução for tipo (iii)

ENTÃO chamar o nodo x também de y

Geração de DAG

Exemplo:

$t1 = a + b$

$t2 = a - b$

$t3 = t1 * t2$

Código objeto:

```
LD  a
ADD b
STO t1
LD  a
SUB b
STO t2
LD  t1
MLT t2
STO t3
```

Arquitetura básica

Processador com um registrador.
Conjunto de instruções

```
LD  E
STO E
ADD E
SUB E
MLT E
```

E: endereço de memória

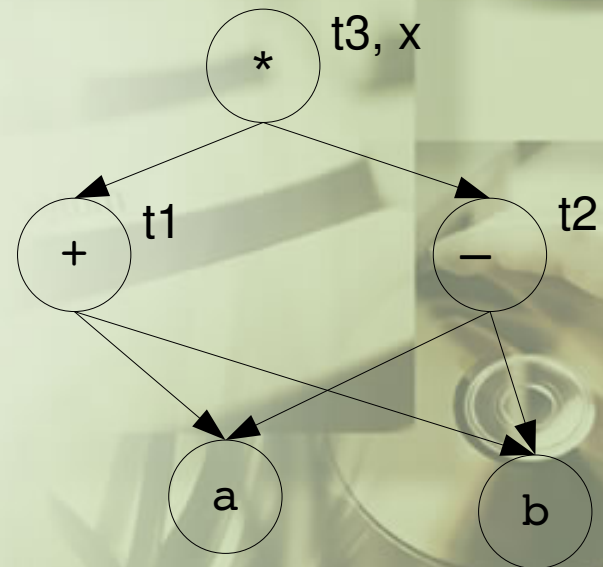
Geração de DAG

Exemplo:

$$t1 = a + b$$

$$t2 = a - b$$

$$t3 = t1 * t2$$



Geração de DAG

Exemplo:

$t1 = a + b$

$t2 = a - b$

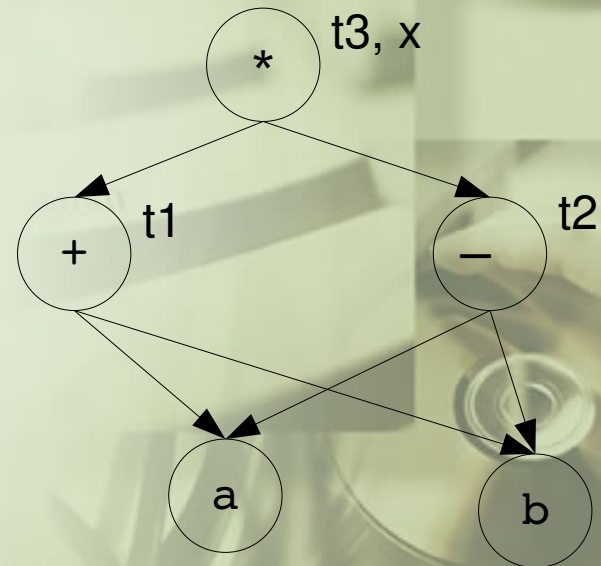
$t3 = t1 * t2$

Assembly:

O grafo gerado permite otimizar o uso dos registradores, reaproveitando resultados de computações anteriores

Código objeto:

```
LD a
ADD b
STO t1
LD a
SUB b
MLT t1
STO t3
```



Geração de DAG

Exemplo:

$t1 = a + b$

$t2 = a - b$

$t3 = t1 * t2$

$t4 = a + b$

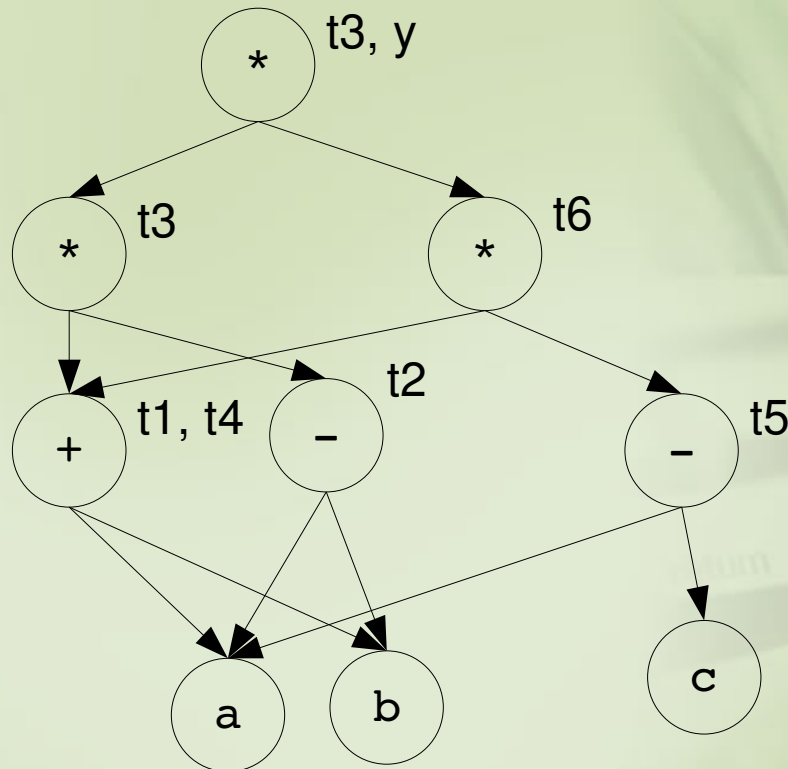
$t5 = a - c$

$t6 = t4 * t5$

$t7 = t3 * t6$

$y = t7$

DAG:



Código objeto:

LD a

ADD b

STO t1

LD a

SUB b

MLT t1

LD a

ADD b

STO t4

LD a

SUB c

MLT t4

MLT t3

STO y

Otimização de DAG

Algoritmo:

Construa um conjunto L de nodos

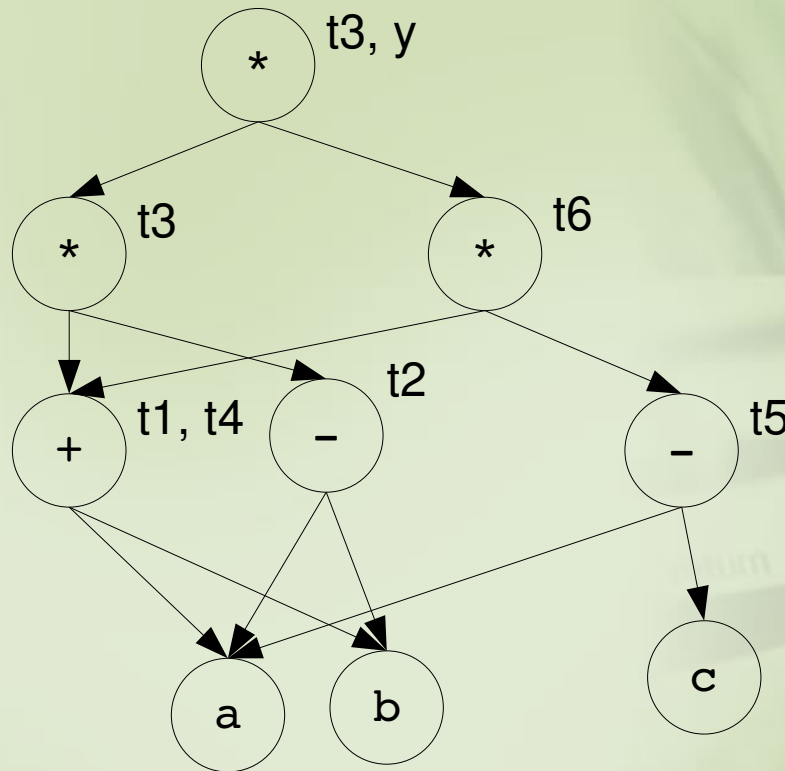
- 1) $L = \{\}$
- 2) Selecionar um nodo n no DAG que não esteja em L e que não possua arestas incidentes partindo de nodos que não estejam em L
 - 2.1) Adicionar nodo n em L
 - 2.2) Se não existe tal nodo, encerrar algoritmo
- 3) Sendo n o último nodo adicionado em L
 - 3.1) SE a aresta mais a esquerda de n incide sobre um nodo m que não está em L e todos os predecessores diretos de m estão em L
ENTÃO: adiciona m em L e segue para o passo 3
SENÃO: segue para o passo 2

Otimização de DAG

Exemplo:

$t1 = a + b$
 $t2 = a - b$
 $t3 = t1 * t2$
 $t4 = a + b$
 $t5 = a - c$
 $t6 = t4 * t5$
 $t7 = t3 * t6$
 $y = t7$

DAG:



Gerado:

$t2 = a - b$
 $t1 = a + b$
 $t3 = t1 * t2$
 $t5 = a - c$
 $t6 = t4 * t5$
 $t7 = t3 * t6$
 $y = t7$

$L = \{ t7, t6, t5, t3, t1 \text{ (ou } t4), t2 \}$

Otimização de DAG

Reordenado:

$$t2 = a - b$$

$$t1 = a + b$$

$$t3 = t1/t4 * t2$$

$$t5 = a - c$$

$$t6 = t1/t4 * t5$$

$$t7 = t3 * t6$$

$$y = t7$$

$L = \{ t7, t6, t5, t3, t1 \text{ (ou } t4), t2 \}$

Obs: em aliases (como $t1/t4$, é necessário salvar o dado.

Código objeto:

LD a

SUB b

STO t2

LD a

ADD b

STO t1/t4

MLT t2

STO t3

LD a

SUB c

MLT t1/t4

MLT t3

STO y

Exercício

Considere:

$$t1 = a + b$$

$$t2 = a - b$$

$$t3 = t1 * t2$$

$$t4 = a - c$$

$$t5 = b - c$$

$$t5 = t2 * t4$$

$$t7 = t6 * t5$$

Construa o DAG deste código intermediário, seu código objeto. Indique o ordenamento deste DAG para minimizar o número de instruções no código objeto gerado e o código objeto final.

(Exercício em Price e Toscani, página 149.)