UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

> Reinventa el mundo <

Compiladores

CS3025

Julio Eduardo Yarasca Moscol

TRANSFORMATEC

UTEC
UNIVERSIDAD DE INGENIERÍA TECNOLÓGICA

> Reinventa el mundo <

Indice:

- Unidad 5: Generación de código
  - 5.1. Llamadas a procedimientos y métodos en envío.
  - 5.2. Compilación separada; vinculación. Selección de instrucciones.
  - 5.3. Asignación de registros. Optimización por rendija (peephole)

TRANSFORMATEC

Logro de la sesión:

Discutir por qué la compilación separada limita la optimización debido a efectos de llamadas desconocidas. Discutir oportunidades para la optimización introducida por la traducción y enfoques para alcanzar la optimización, tales como la selección de la instrucción, planificación de instrucción, asignación de registros y optimización de tipo mirilla (peephole optimization).

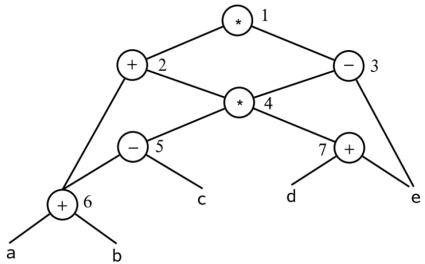
DAG de un bloque básico

- El DAG (Directed Acyclic Graph) de un bloque básico es una estructura utilizada en la optimización de compiladores para representar las dependencias entre operaciones dentro de un bloque básico.
- Es un **grafo acíclico dirigido (DAG)** que permite optimizar bloques de instrucciones en `Assembly x86-64`, mediante:
  - Representar operaciones aritméticas y movimientos (`addq`, `imulq`, `movq`, etc.) realizadas en un bloque básico.
  - Detectar y eliminar subexpresiones comunes (como multiplicaciones o sumas repetidas).
  - Reutilizar resultados ya calculados en registros, evitando operaciones redundantes.
- Elementos del DAG en `Assembly`:
  - **Nodos:** Representan registros, constantes inmediatas o resultados de instrucciones aritméticas.
  - **Aristas:** Indican dependencias de datos entre registros e instrucciones (por ejemplo, qué operandos se usan en una suma o multiplicación).

```
1: Inicializar un DAG vacío.
2: for cada instrucción op src, dest en el bloque do
3:     if op es movq then
4:         Buscar o crear nodo para src.
5:         Etiquetar ese nodo con dest.
6:     else if op es aritmética (addq, subq, imulq, etc.) then
7:         Buscar o crear nodo para src.
8:         Buscar o crear nodo para dest (valor previo antes de sobrescribir).
9:         Buscar si ya existe un nodo con operación op aplicada a dest y src.
10:        if existe then
11:            Reutilizar el nodo y etiquetarlo con dest.
12:        else
13:            Crear nodo nuevo con operación op, hijos: dest y src.
14:            Etiquetar el nodo con dest.
15:        end if
16:    end if
17: end for
18: for cada instrucción movq $c, reg con constante inmediata do
19:    if no existe nodo para $c then
20:        Crear nodo constante.
21:    end if
22:    Etiquetar ese nodo con reg.
23: end for
24: return DAG construido
```

Ejemplo

$$[(a+b)+((a+b-c)(d+e))][e-(d+e)]$$



Ejemplo

Ejemplo 1

```
t1  = a + b
t2  = a + b
t3  = t2 - c
t4  = d + e
t5  = t3 * t4
t6  = t1 + t5
t7  = d + e
t8  = e - t7
t9  = t6 * t8
```

Optimizado

```
t1 = a + b
t2 = t1 - c
t3 = d + e
t4 = t2 * t3
t5 = t1 + t4
t6 = e - t3
t7 = t5 * t6
```

Ejemplo

Ejemplo 1

```
movq $5, %rax
movq $5, %rbx
addq %rax, %rcx
addq %rbx, %rcx
```

Optimizado

```
addq $10, %rcx
```

Ejemplo

Ejemplo 2

```
movq %rdi, %rax
addq %rsi, %rax
movq %rax, %rbx
addq %rsi, %rdi
```

Optimizado

```
addq %rsi, %rdi
movq %rdi, %rax
movq %rdi, %rbx
```

Preguntas y cierre de la sesión

- ¿En qué consiste la optimización mirilla?

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

> Reinventa el mundo <

GRACIAS

JULIO EDUARDO YARASCA MOSCOL

TRANSFORMATEC