COMPILADORES

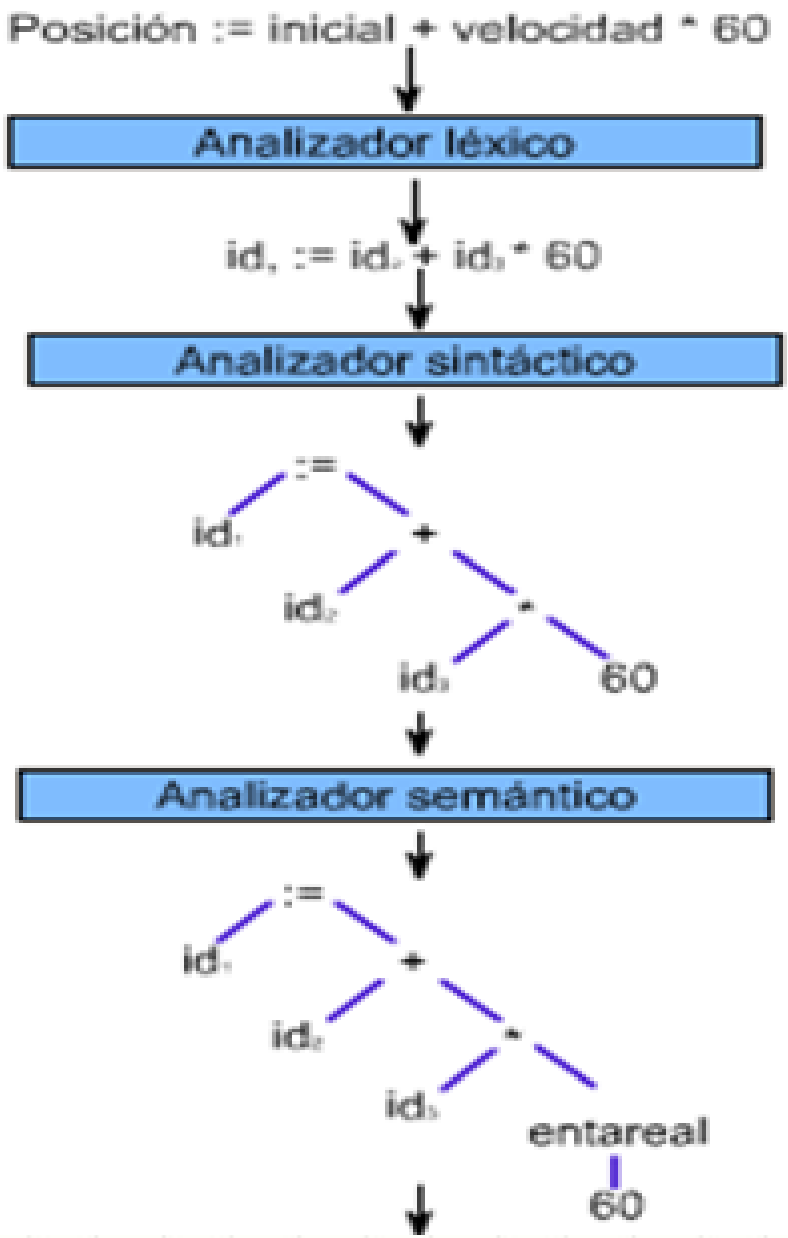
TEMA 6. OPTIMIZACIÓN DE CÓDIGO

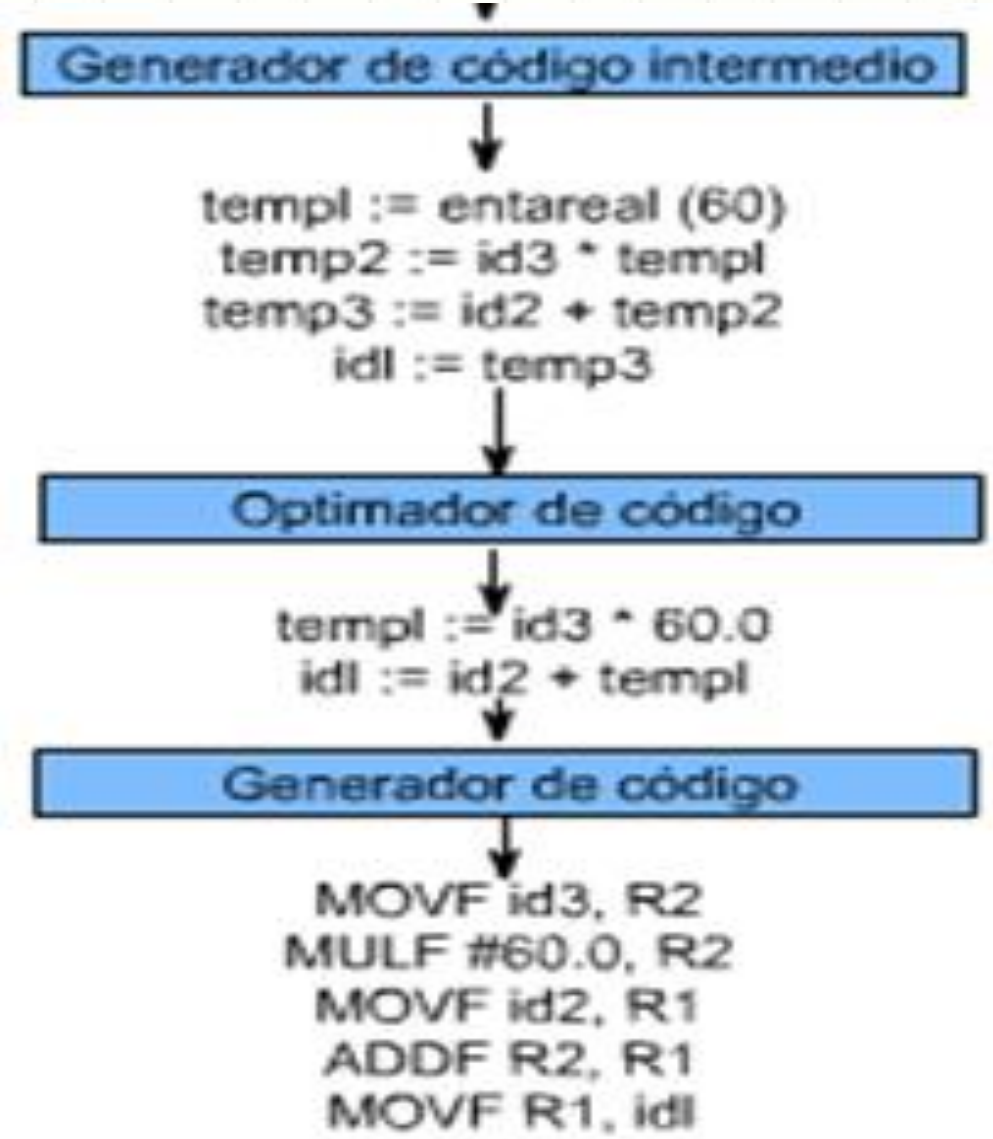
Presenta: Mtro. David Martínez Torres
Universidad Tecnológica de la Mixteca
Instituto de Computación
Oficina No. 37
dtorres@gs.utm.mx

Contenido

1. Principales fuentes para la optimización
2. Optimizaciones independientes de la máquina objeto
3. Optimización de bloques básicos
4. Lazos en los diagramas de flujo
5. Análisis del flujo de datos
6. Optimización dependiente de la máquina objeto: registros e instrucciones

Recordemos fases de un compilador con un ejemplo.





1. Principales fuentes para la optimización

Las transformaciones para mejorar el código pueden ser locales o globales.

- Se consideran locales cuando se aplica en proposiciones de un bloque básico
- Por el contrario, se consideran globales cuando la transformación se aplica a todos los bloques básicos.

Figura 3. Grafo de flujo.

1. Principales fuentes para la optimización

Las principales fuentes de transformación son las siguientes:

- Eliminación de subexpresiones comunes
- Propagación de copias
- Eliminación de código inactivo y,
- Cálculo previo de constantes.

1. Principales fuentes para la optimización

La optimización de código, normalmente se aplica a código intermedio. A continuación se mostrará un código fuente en C, del que se obtendrá código intermedio de tres direcciones, al cual, se le aplicará métodos de optimización.

```
void clasificacion_por_particiones(int m,int n) {
int i, j, v, x;
if(n <= m)
        return;
/*el fragmento comienza aquí*/
i=m-1;
j=n;
v=a[n];
while(1){
        do
                i=i+1;
        while(a[i]<v);
        do
                j=j-1;
        while(a[j]>v);
        if(i>=j)
                break;
        x=a[i];
        a[i]=a[j];
        a[j]=x;
        }
x=a[i];
a[i]=a[n];
a[n]=x;
/*el fragmento termina aquí*/
clasificacion_por_particiones(m,j);
clasificacion_por_particiones(i+1,n);
}
```

Figura 1. Código en C para clasificacion_por_particiones

1. Principales fuentes para la optimización

- Existen pocas transformaciones de mejora del código a nivel de programa fuente.
- A nivel de código intermedio hay más oportunidades, por ejemplo el código de tres direcciones especialmente en ciclos (lazos)
- Considere el código de tres direcciones para determinar el valor de a[i], suponiendo que cada elemento de la matriz ocupa cuatro bytes.

```
t1:=4*i
t2:= a[t1]
```

```
void clasificacion_por_particiones(int m,int n) {
int i, j, v, x;
if(n <= m)
        return;
/*el fragmento comienza aquí*/
i=m-1;
j=n;
v=a[n];
while(1){
        do
                i=i+1;
        while(a[i]<v);
        do
                j=j-1;
        while(a[j]>v);
        if(i>=j)
                break;
        x=a[i];
        a[i]=a[j];
        a[j]=x;
        }
x=a[i];
a[i]=a[n];
a[n]=x;
/*el fragmento termina aquí*/
clasificacion_por_particiones(m,j);
clasificacion_por_particiones(i+1,n);
}
```

Figura 1. Código en C para clasificacion_por_particiones

```
(1) i:=m-1
(2) j:=n
(3) t1:= 4*n
(4) v:= a[t1]
(5) i:= i+1
(6) t2:= 4*i
(7) t3:= a[t2]
(8) if t3 < v goto (5)
(9) j:= j-1
(10) t4:= 4*j
(11) t5:= a[t4]
(12) if t5 > v goto (9)
(13) if i >= j goto (23)
(14) t6:= 4*i
(15) x:= a[t6]
(16) t7:= 4*i
(17) t8:= 4*j
(18) t9:= a[t8]
(19) a[t7]:= t9
(20) t10:= 4*j
(21) a[t10]:= x
(22) goto (5)
(23) t11:= 4*i
(24) x:= a[t11]
(25) t12:= 4*i
(26) t13:= 4*n
(27) t14:= a[t13]
(28) a[t12]:= t14
(29) t15:= 4*n
(30) a[t15]:= x
```

Figura 2. Código de tres direcciones para el fra
la Figura 1.

```
(1) i:=m-1
(2) j:=n
(3) t1:= 4*n
(4) v:= a[t1]
(5) i:= i+1
(6) t2:= 4*i
(7) t3:= a[t2]
(8) if t3 < v goto (5)
(9) j:= j-1
(10) t4:= 4*j
(11) t5:= a[t4]
(12) if t5 > v goto (9)
(13) if i >= j goto (23)
(14) t6:= 4*i
(15) x:= a[t6]
(16) t7:= 4*i
(17) t8:= 4*j
(18) t9:= a[t8]
(19) a[t7]:= t9
(20) t10:= 4*j
(21) a[t10]:= x
(22) goto (5)
(23) t11:= 4*i
(24) x:= a[t11]
(25) t12:= 4*i
(26) t13:= 4*n
(27) t14:= a[t13]
(28) a[t12]:= t14
(29) t15:= 4*n
(30) a[t15]:= x
```

Figura 2. Código de tres direcciones para el fragmento de código de la Figura 1.

1. Principales fuentes para la optimización

- Con otras transformaciones intermedias, las variables temporales t1, t2,…, t15 no tienen que aparecer.
- El código intermedio del programa anterior asume que cada elemento del vector ocupa cuatro bytes.
- En el optimizador de código, los programas se representan mediante grafos de flujo, las aristas indican el flujo de control y los nodos representan bloques básicos.

Representación del código de 3 direcciones en un Grafo de flujo

(1) i:=m-1
(2) j:=n
(3) t1:= 4*n
(4) v:= a[t1]
(5) i:= i+1
(6) t2:= 4*i
(7) t3:= a[t2]
(8) if t3 < v goto (5)
(9) j:= j-1
(10) t4:= 4*j
(11) t5:= a[t4]
(12) if t5 > v goto (9)
(13) if i >= j goto (23)
(14) t6:= 4*i
(15) x:= a[t6]
(16) t7:= 4*i
(17) t8:= 4*j
(18) t9:= a[t8]
(19) a[t7]:= t9
(20) t10:= 4*j
(21) a[t10]:= x
(22) goto (5)
(23) t11:= 4*i
(24) x:= a[t11]
(25) t12:= 4*i
(26) t13:= 4*n
(27) t14:= a[t13]
(28) a[t12]:= t14
(29) t15:= 4*n
(30) a[t15]:= x

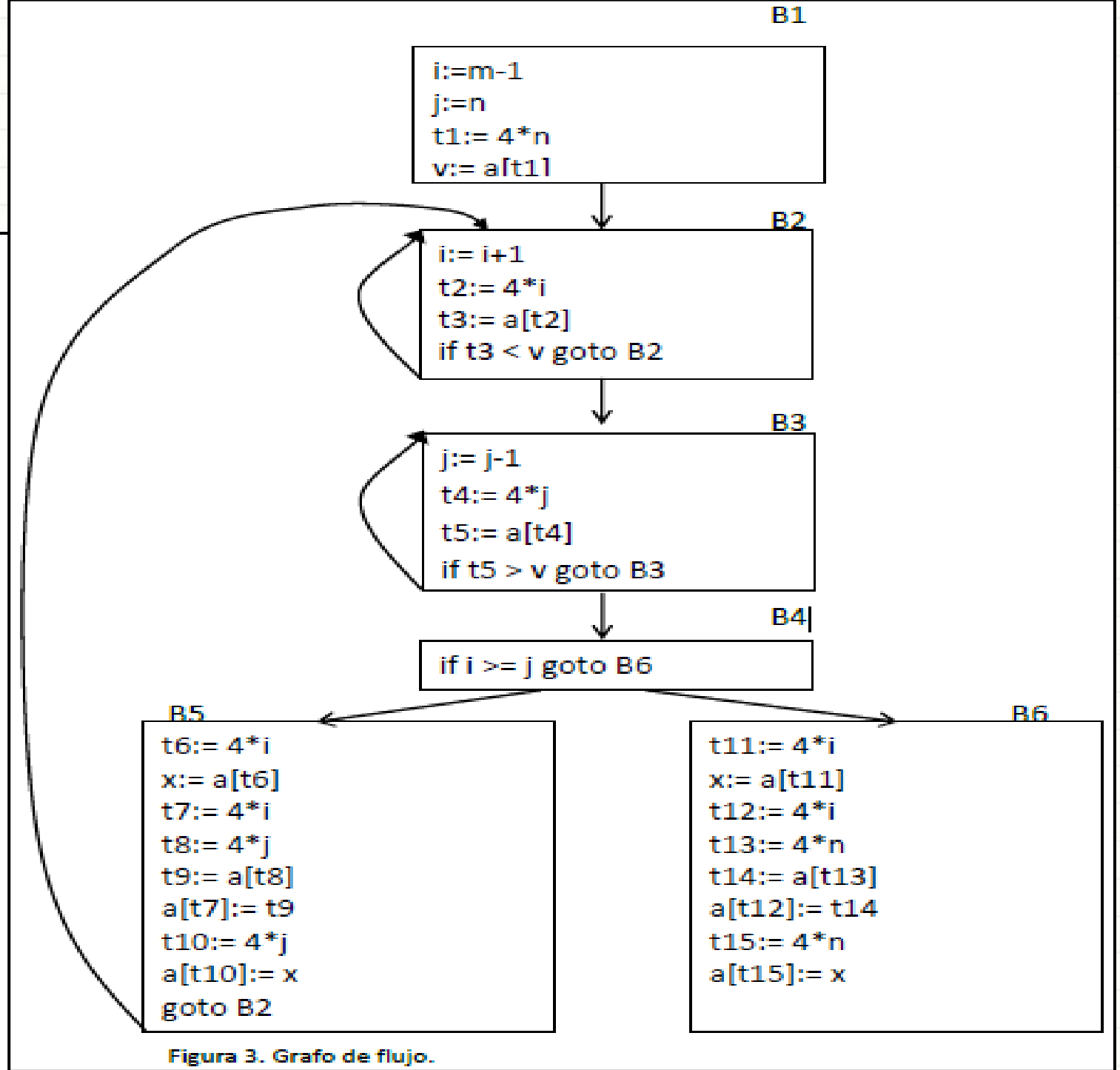


Figura 3. Grafo de flujo.

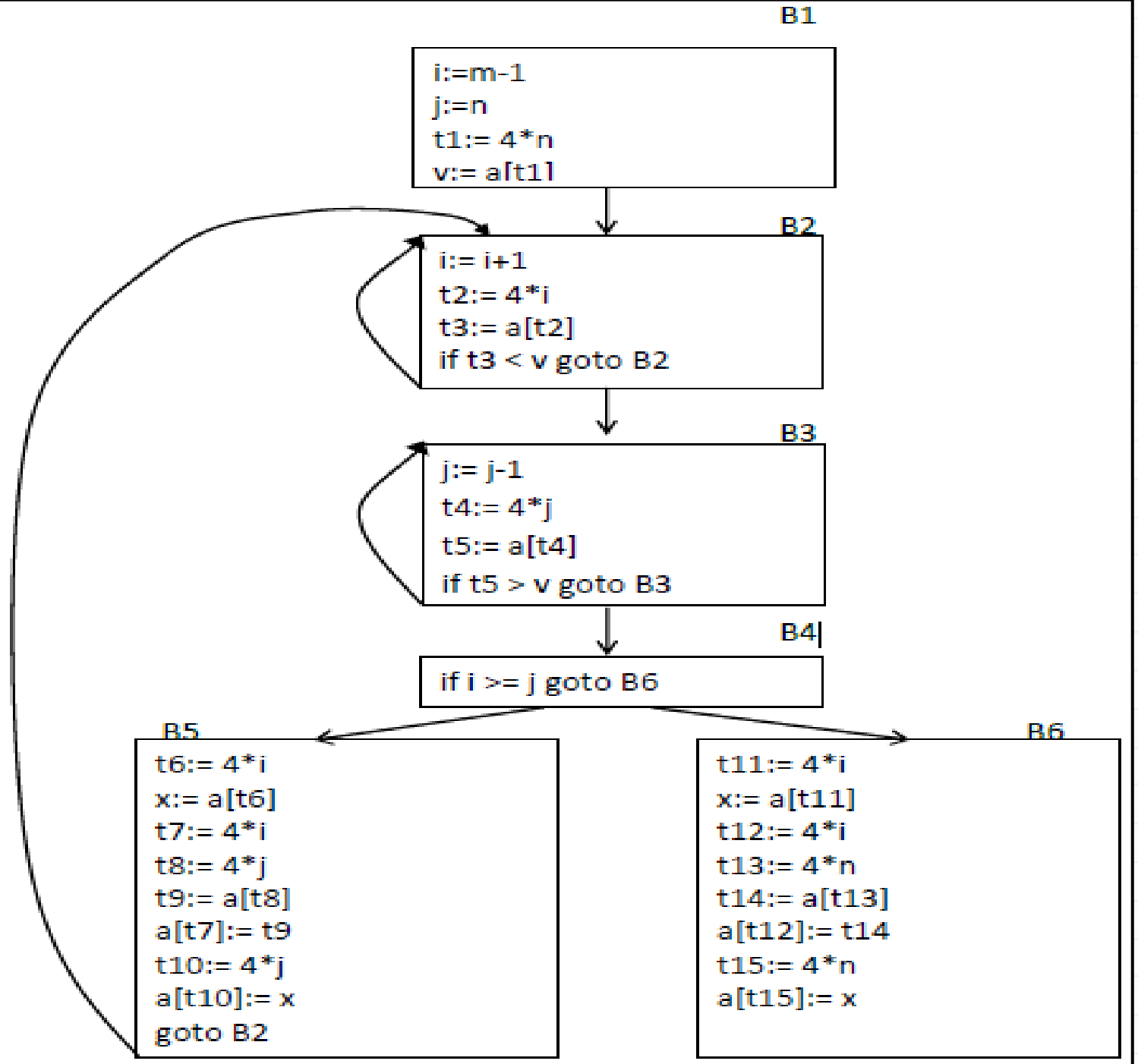


Figura 3. Grafo de flujo.

1. Principales fuentes para la optimización

- Todos los saltos condicionales e incondicionales hacia proposiciones en la figura 2, se han sustituido en la figura 3 por saltos al bloque del cual son líderes las proposiciones.
- En la figura 3, hay 3 lazos, B2 y B3 son lazos en sí mismos. Los bloques B2, B3, B4 y B5, juntos forman un lazo, con entrada B2.

1. Principales fuentes para la optimización

Eliminación de subexpresiones comunes.

- Una ocurrencia de una expresión E se denomina subexpresión común si E ha sido previamente calculada y los valores de las variables dentro de E no han cambiado desde el cálculo anterior.

1. Principales fuentes para la optimización

Por ejemplo, las asignaciones a t7 y a t10 tienen las subexpresiones comunes 4*i y 4*j respectivamente en el lado derecho de la figura 4(a). Han sido eliminadas en la figura 4(b) utilizando t6 en lugar de t7 y t8 en lugar de t10.

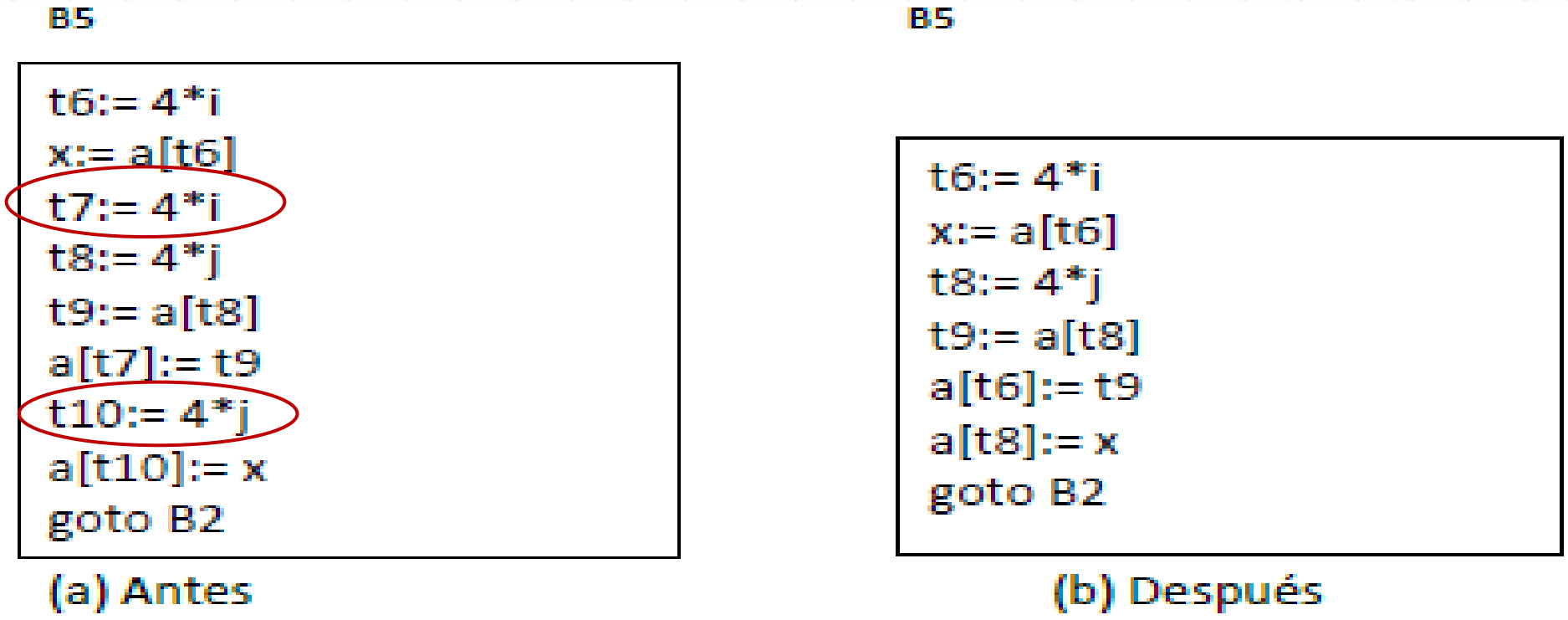


Figura 4. Eliminación de subexpresiones comunes locales.

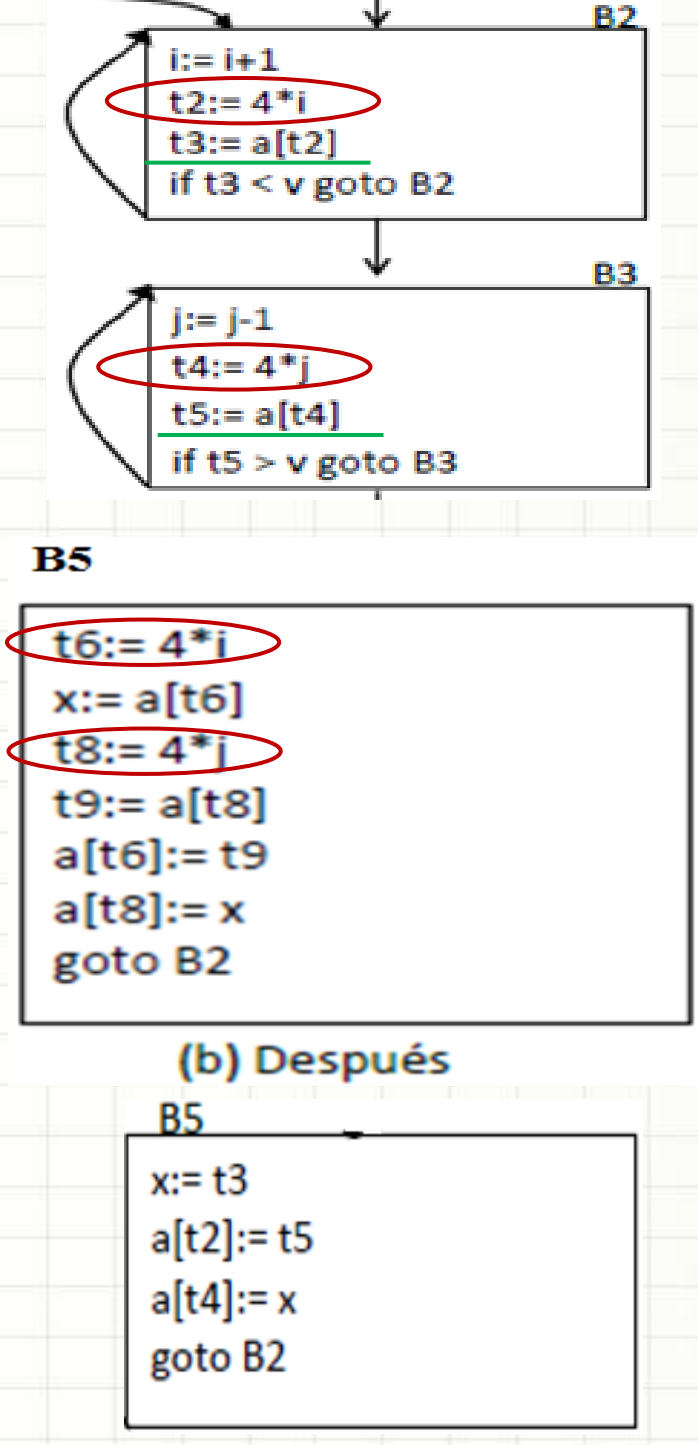
1. Principales fuentes para la optimización

- Después de eliminar las subexpresiones comunes, B5 todavía tiene que evaluar 4*i y 4*j, en la figura 4(b). Ambas son subexpresiones comunes.

  t8:= 4*j t9:= a[t8] a[t8]:= x

- Considerando t2:=4*i en B2, y t4:= 4*j en B3, entonces en B5, se pueden sustituir por:

  t9:= a[t4] y esta por a[t4]:= x



Grafo después de la eliminación de subexpresiones comunes en B5 y B6

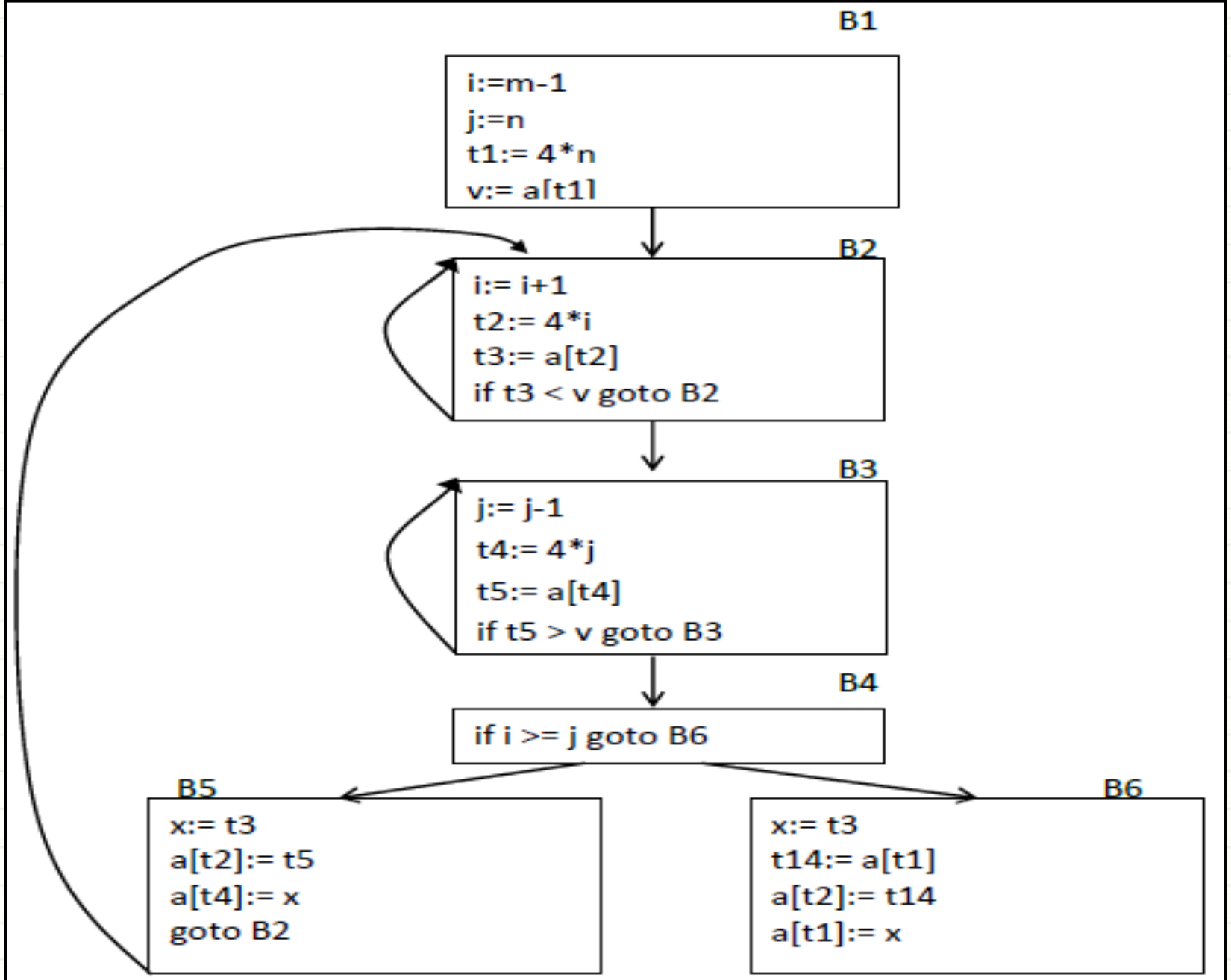
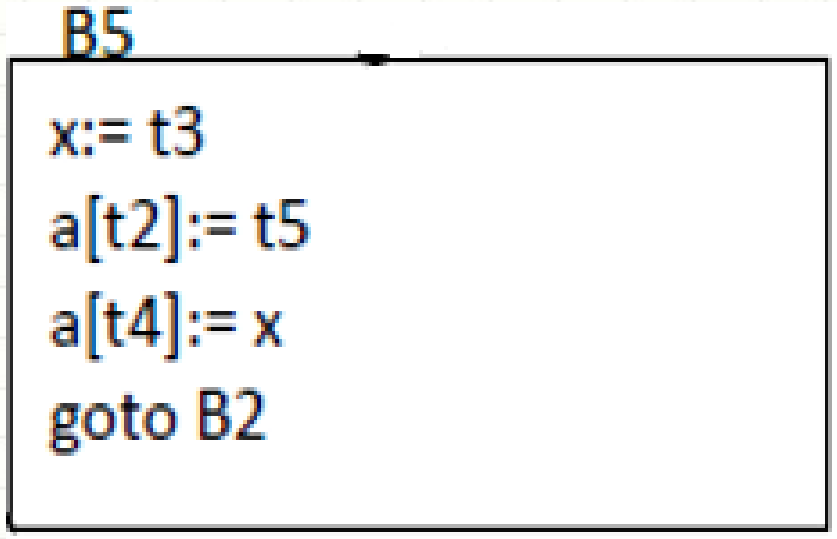


Figura 5. B5 y B6 después de la eliminación de subexpresiones comunes.

1. Principales fuentes para la optimización

- **Propagación de copias.**
  - El bloque B5 de la figura 5 se puede mejorar aún más eliminando x por medio de dos nuevas transformaciones. Una concierne a las asignaciones de la forma f:= g llamadas *proposiciones de copia o copias* simplemente.



1. Principales fuentes para la optimización

- Por ejemplo, en la figura 6, cuando se elimina la subexpresión común c:= d+e, el algoritmo utiliza una nueva variable t para guardar el valor d+e.

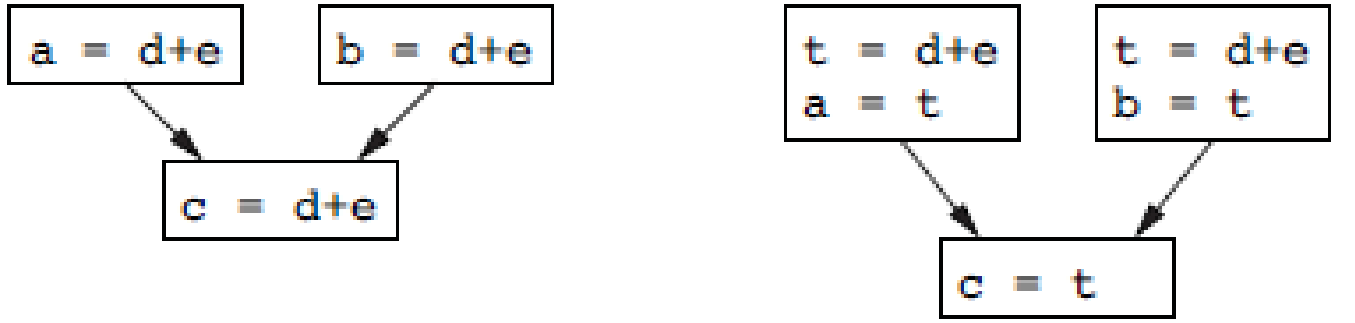
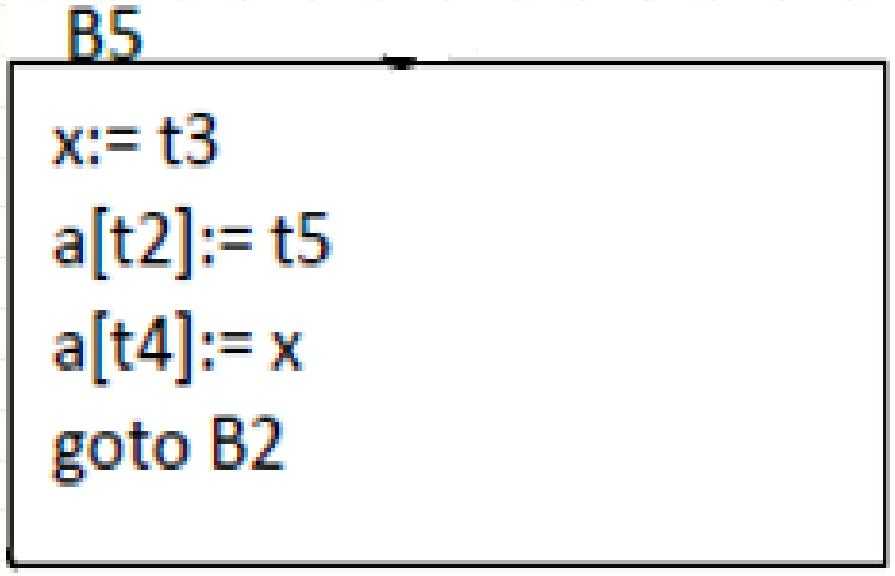


Figura 6. Copias introducidas durante la eliminación de subexpresiones comunes.

- La idea que se basa la transformación de copias es utilizar g por f, siempre que sea posible después de la proposición de copia f:= g

1. Principales fuentes para la optimización

- Por ejemplo la asignación $x := t3$ en el bloque B5 de la figura 5 es una copia. La propagación de copias aplicada a B5 produce:



- Puede no parecer una mejora, pero veremos que da la oportunidad de eliminar las asignaciones a x.

1. Principales fuentes para la optimización

- **Eliminación de código inactivo**
  - Una variable está activa en un punto de un programa si su valor puede ser utilizado posteriormente; en caso contrario está inactiva en ese punto.
  - Una idea afín es el código inactivo o inútil, proposiciones que calculan valores que nunca se llegan a utilizarse.

1. Principales fuentes para la optimización

- Una ventaja de la propagación de copias es que normalmente convierte proposiciones de copia en código inactivo.
- Ejemplo, el código de bloque B5, puede quedar de la siguiente forma.

B5

```
x:= t3
a[t2]:= t5
a[t4]:= x
goto B2
```

```
a[t2]:= t5
a[t4]:= t3
goto B2
```

2. Optimizaciones independientes de la máquina objeto

- Optimización de lazos
  - Es donde los programas tienden a emplear la mayor parte de su tiempo.
  - El tiempo de ejecución de un programa se puede mejorar si se disminuye la cantidad de instrucciones en un lazo interno, incluso si se incrementa la cantidad de código fuera del lazo.
  - Hay tres técnicas para la optimización de lazos: **traslado de código, las variables de inducción y la reducción de intensidad.**

3. Optimización de bloques básicos

- **Traslado de código:**
  - Esta técnica se puede visualizar en el siguiente ejemplo:

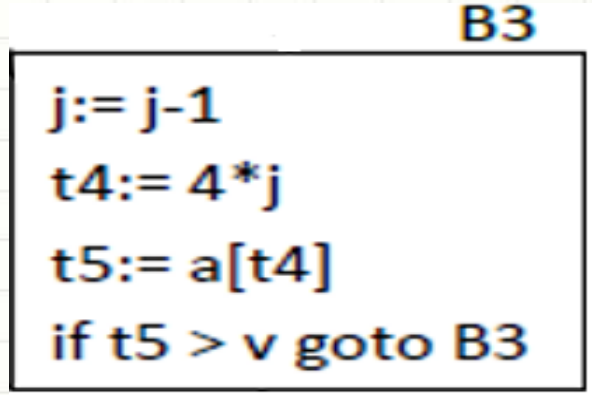
```
/* la siguiente expresión no cambia limite*/
while (i<= limite-2)
```

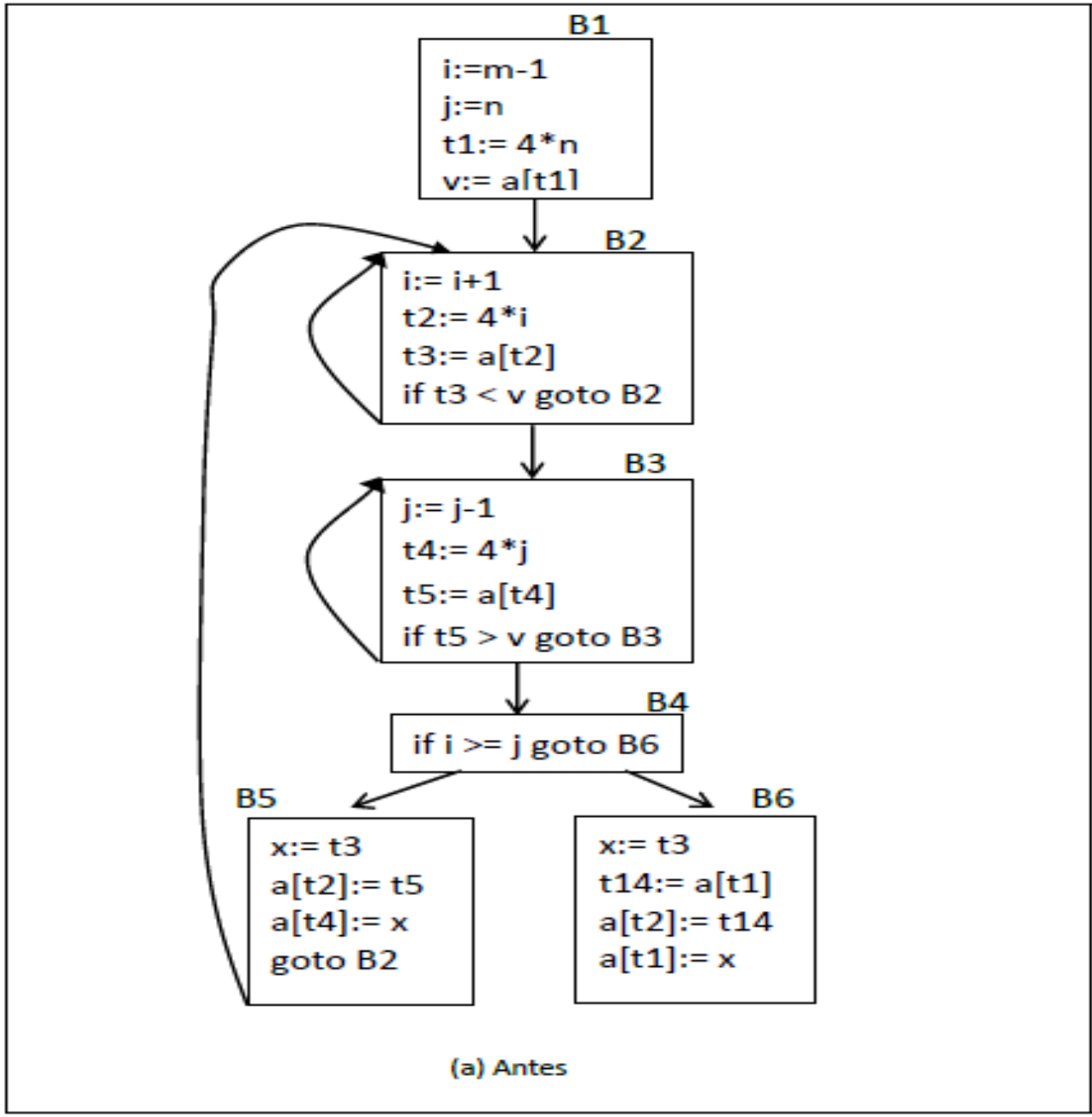
  - El traslado de código es equilavente a:

```
t = limite -2
while (i<=t)     /* la expresión no cambia limite ni t*/
```

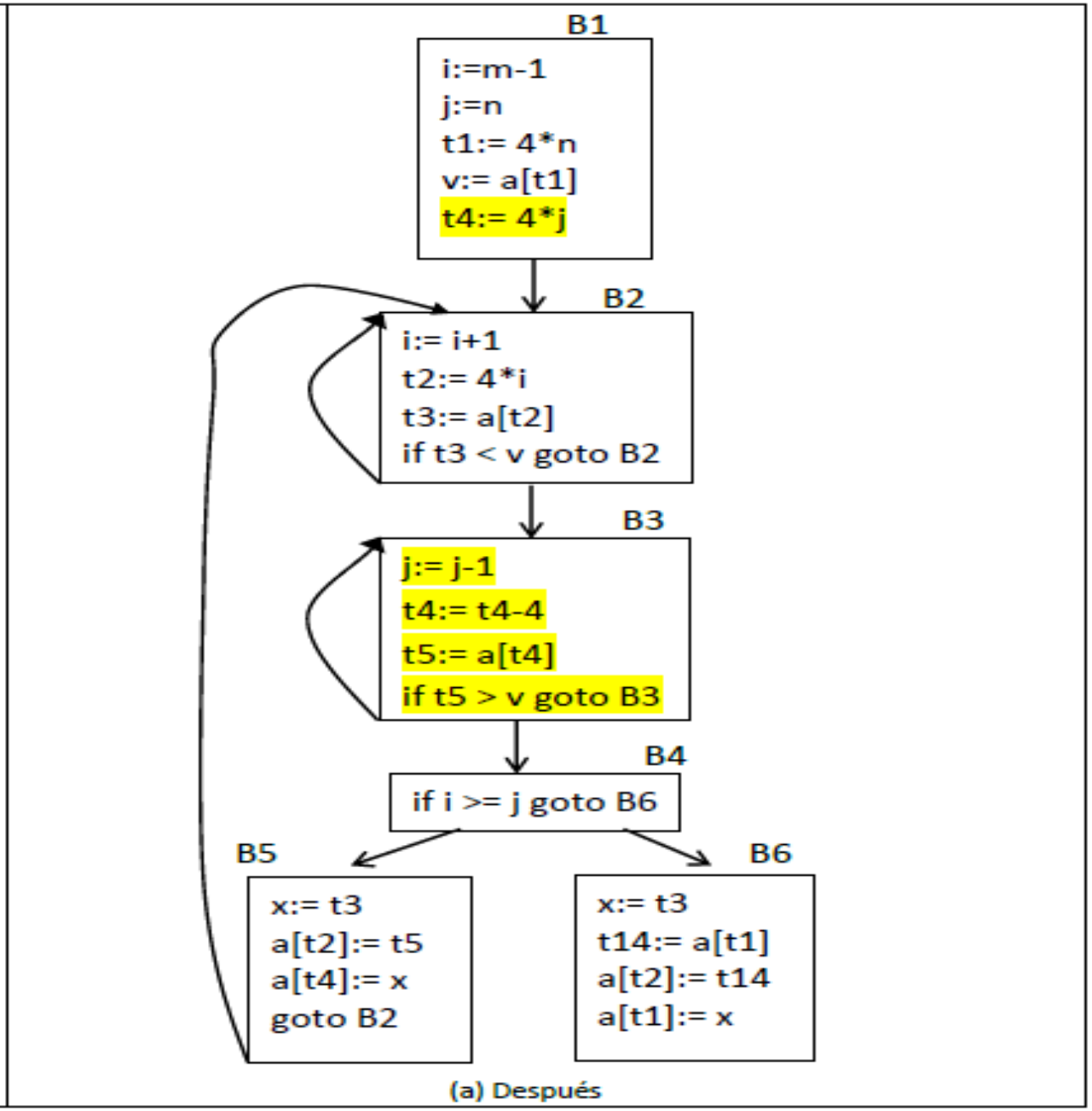
4. Lazos en los diagramas de flujo

- **Variables de inducción y reducción de intensidad**
  - Revisar el bloque B3 de la figura 7.
  - Observe que los valores de j y t4 permanecen atados; cada vez que el valor de j disminuye en 1, el de t4 disminuye en 4, porque 4*j se asigna a t4.
  - Dichos identificadores se denominan variables de inducción





(a) Antes



(a) Después

Figura 7. Reducción de intensidad aplicada a 4*j en el bloque B3.

Grafo final, después de las optimizaciones realizadas.

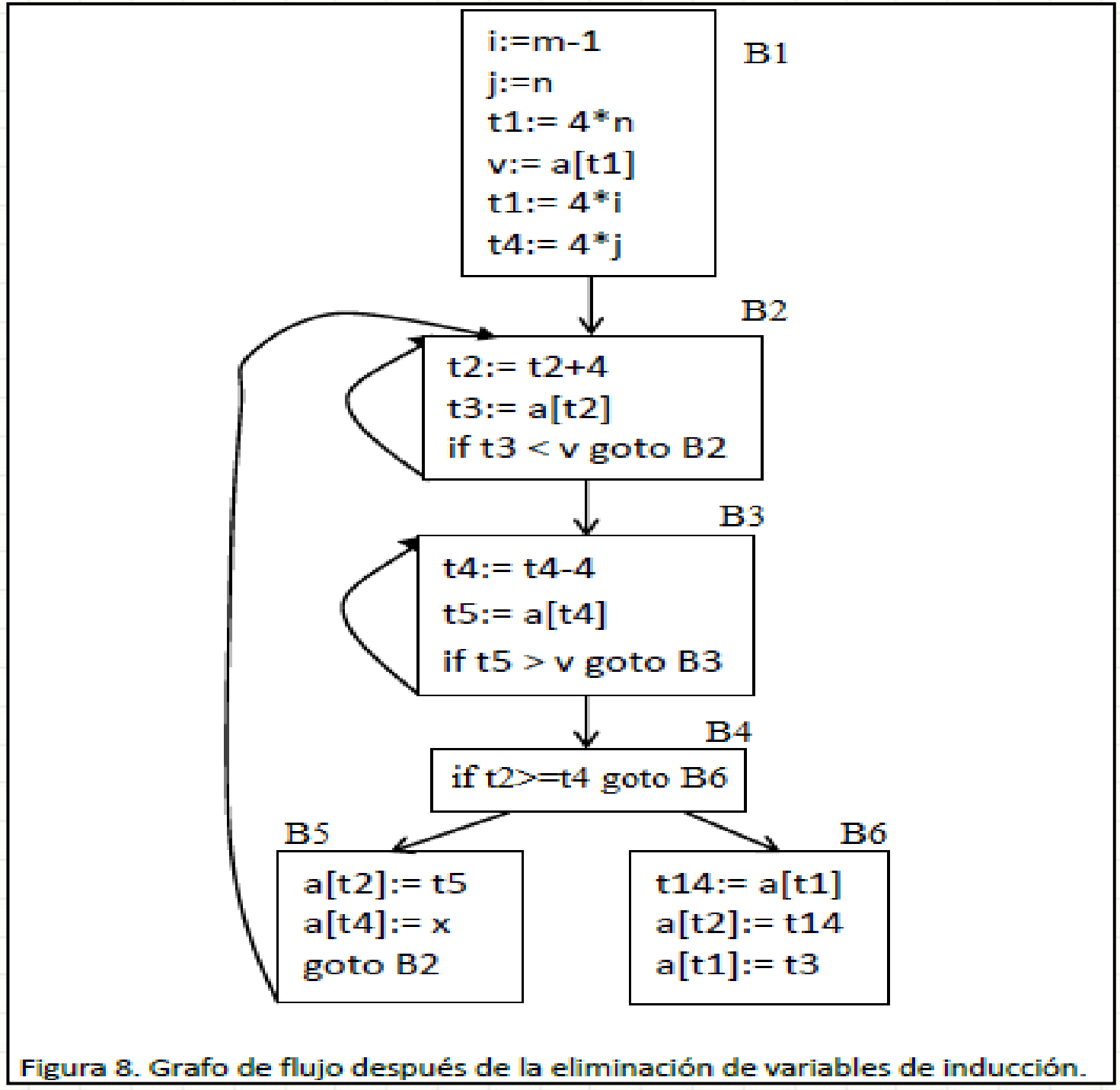


Figura 8. Grafo de flujo después de la eliminación de variables de inducción.

Referencias

1. Compiladores. Principios, Técnicas y Herramientas (2a ed.). Aho, A.V. Pearson Educación. 2008.
2. Construcción de Compiladores. Principios y práctica. Louden, K. C. Thomson Editores. 2005