

7. Форма статического единственного присваивания (*SSA*-форма)

6.1 Форма статического единственного присваивания (SSA-форма)

6.1.1. Постановка задачи

- ◇ *Форма статического единственного присваивания (SSA)* позволяет в каждой точке программы объединить
 - ◇ информацию об имени переменной
 - ◇ и информацию о текущем значении этой переменной (или, что то же самое, информацию о том, *какое из определений данной переменной определяет ее текущее значение в данной точке*).

6.1 Форма статического единственного присваивания (SSA-форма)

6.1.1. Постановка задачи

- ◇ *Форма статического единственного присваивания (SSA)* позволяет в каждой точке программы объединить
 - ◇ информацию об имени переменной
 - ◇ и информацию о текущем значении этой переменной (или, что то же самое, информацию о том, ***какое из определений данной переменной определяет ее текущее значение в данной точке***).
- ◇ Хотелось бы, чтобы программа в *SSA*-форме удовлетворяла двум условиям:
 - (1) каждое определение переменной имеет индивидуальное имя;
 - (2) каждое использование переменной ссылается на единственное определение.

6.1 Форма статического единственного присваивания (SSA-форма)

6.1.1. Постановка задачи

- ◇ *Форма статического единственного присваивания (SSA)* позволяет в каждой точке программы объединить
 - ◇ информацию об имени переменной
 - ◇ и информацию о текущем значении этой переменной (или, что то же самое, информацию о том, *какое из определений данной переменной определяет ее текущее значение в данной точке*).

Исходная программа Обычное представление Представление в SSA-форме

```
int bar();
void foo(int a) {
    int b = a + 3;
    b = bar();
    b = a + 3;
    b = b + a;
}
```



```
foo:
    param a
    b = a + 3
    b = bar(b)
    b = a + 3
    b = b + a
```



```
foo:
    param a.0
    b.0 = a.0 + 3
    b.1 = bar(b.0)
    b.2 = a.0 + 3 // <- b.0
    b.3 = b.2 + a.0
```

Исходная программа Обычное представление Представление в SSA-форме

```
int bar();
void foo(int a)
{
    int b = 0;
    b = bar();
    b = a + 3;
    if (a == 0)
    {
        b = b * 4;
    }
    b = b + a;
}
```



```
foo:
    param a
    b = 0
    b = bar()
    b = a + 3
    if (a != 0) goto next
    b = b * 4

next:
    b = b + a
```



```
foo:
    param a.0
    b.0 = 0
    b.1 = bar()
    b.2 = a.0 + 3
    if (a.0 == 0) goto l1 : l2

l1:
    b.3 = b.2 * 4

l2:
    b.4 = b.? + a.0
```

Исходная программа Обычное представление Представление в SSA-форме

```
int bar();  
void foo(int a)  
{  
    int b = 0;  
    b = bar();  
    b = a + 3;  
    if (a == 0)  
    {  
        b = b * 4;  
    }  
    b = b + a;  
}
```



```
foo:  
    param a  
    b = 0  
    b = bar()  
    b = a + 3  
    if (a != 0) goto next  
    b = b * 4  
  
next:  
    b = b + a
```

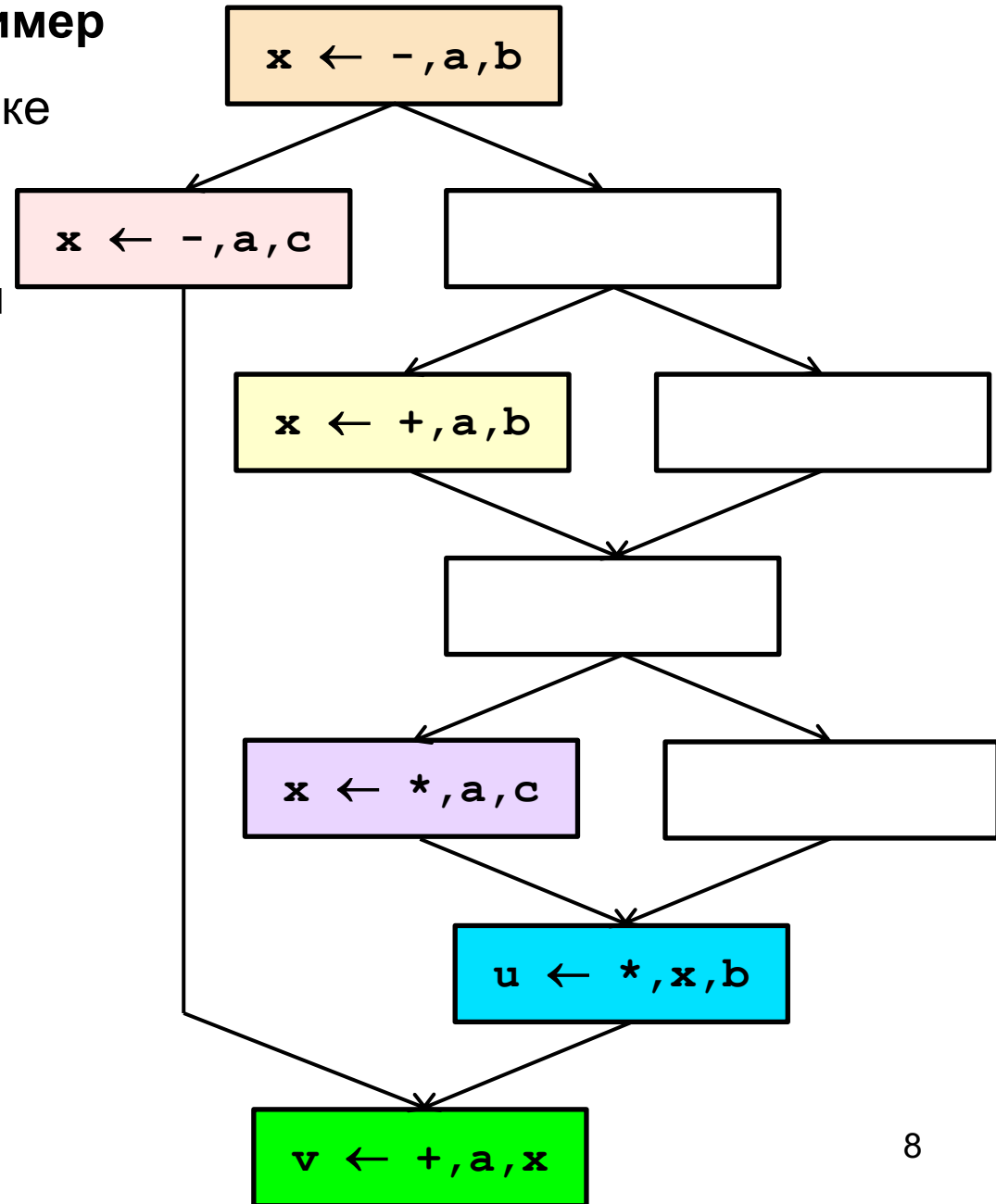


```
foo:  
    param a.0  
    b.0 = 0  
    b.1 = bar()  
    b.2 = a.0 + 3  
    if (a.0 == 0) goto l1 : l2  
  
l1:  
    b.3 = b.2 * 4  
  
l2:  
    b.4 = phi(b.2, b.3)  
    b.5 = b.4 + a.0
```

6.1 SSA-форма

6.1.1. Постановка задачи. Пример

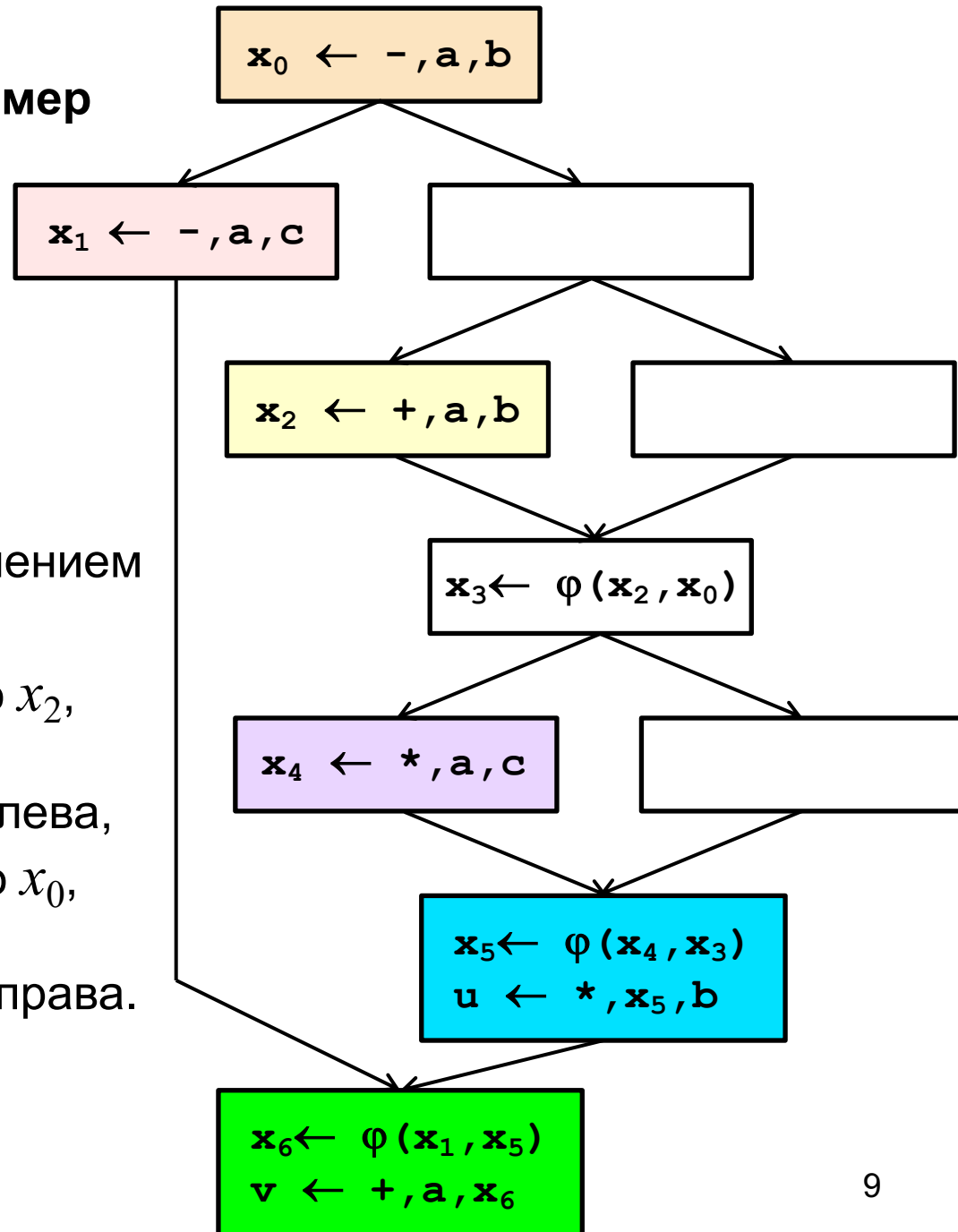
- ◇ Фрагмент ГПУ на рисунке содержит четыре определения **x**
- ◇ Использования в синем блоке достигают три определения **x**, в зеленом – четыре определения **x**
- ◇ Цель же состоит в том, чтобы *каждого использования достигало только одно определение*



6.1 SSA-форма

6.1.1. Постановка задачи. Пример

- ◇ Введем «функцию» объединения значений, или ϕ -функцию
- ◇ По определению $x_3 \leftarrow \phi(x_2, x_0)$ является новым определением переменной x :
 - ◇ значение x_3 равно x_2 , когда управление попадает в блок слева,
 - ◇ значение x_3 равно x_0 , когда управление попадает в блок справа.



6.1 SSA-форма

6.1.2. Определение ϕ -функции

- ◇ ϕ -функция определяет *SSA-имя* для значения своего аргумента, соответствующего ребру, по которому управление входит в блок.
- ◇ При входе в базовый блок все его ϕ -функции выполняются одновременно и до любого другого оператора, определяя целевые *SSA-имена*.

6.1 SSA-форма

6.1.2. Определение ϕ -функции. Пример

```
x =  
y = ...  
while (x < 100) {  
    x = x + 1  
    y = y + x  
}
```

6.1 SSA-форма

6.1.2. Определение ϕ -функции. Пример

```
x =  
y = ...  
while (x < 100) {  
    x = x + 1  
    y = y + x  
}
```

```
      x0 = ...  
      y0 = ...  
      if (x0 ≥ 100) goto next  
loop:  x1 =  $\phi(x_0, x_2)$   
      y1 =  $\phi(y_0, y_2)$   
      x2 = x1 + 1  
      y2 = y1 + x2  
      if (x2 < 100) goto loop  
next:  x3 =  $\phi(x_0, x_2)$   
      y3 =  $\phi(y_0, y_2)$ 
```

6.1 SSA-форма

6.1.2. Определение ϕ -функции. Пример

```
x =  
y = ...  
while (x < 100) {  
    x = x + 1  
    y = y + x  
}
```

```
x0 = ...  
y0 = ...  
if (x0 ≥ 100) goto next  
loop: x1 =  $\phi(x_0, x_2)$   
      y1 =  $\phi(y_0, y_2)$   
      x2 = x1 + 1  
      y2 = y1 + x2  
      if (x2 < 100) goto loop  
next: x3 =  $\phi(x_0, x_2)$   
      y3 =  $\phi(y_0, y_2)$ 
```

Каждая из ϕ -функций
объединяет значения своих
аргументов в новое значение,
определением которого она
является.

6.1 SSA-форма

6.1.2. Определение ϕ -функции. Пример

```
x =  
y = ...  
while (x < 100) {  
    x = x + 1  
    y = y + x  
}
```

```
      x0 = ...  
      y0 = ...  
      if (x0 ≥ 100) goto next  
loop: x1 =  $\phi(x_0, x_2)$   
      y1 =  $\phi(y_0, y_2)$   
      x2 = x1 + 1  
      y2 = y1 + x2  
      if (x2 < 100) goto loop  
next: x3 =  $\phi(x_0, x_2)$   
      y3 =  $\phi(y_0, y_2)$ 
```

Каждая из ϕ -функций
объединяет значения своих
аргументов в новое значение,
определением которого она
является.

До цикла –	x₀, y₀
На входе в цикл –	x₁, y₁
Внутри цикла –	x₂, y₂
После цикла –	x₃, y₃

6.1 SSA-форма

6.1.2. Определение ϕ -функции. Пример

```
x =  
y = ...  
while (x < 100) {  
    x = x + 1  
    y = y + x  
}
```

```
x0 = ...  
y0 = ...  
if (x0 ≥ 100) goto next  
loop: x1 =  $\phi(x_0, x_2)$   
      y1 =  $\phi(y_0, y_2)$   
      x2 = x1 + 1  
      y2 = y1 + x2  
      if (x2 < 100) goto loop  
next: x3 =  $\phi(x_0, x_2)$   
      y3 =  $\phi(y_0, y_2)$ 
```



Затруднение. Первый аргумент $\phi(x_0, x_2)$ определяется в блоке, который предшествует циклу, второй аргумент определяется позже в блоке, содержащем $\phi(x_0, x_2)$. Следовательно, при первом выполнении $\phi(x_0, x_2)$ ее второй аргумент еще не определен.

6.1 SSA-форма

6.1.2. Определение φ -функции. Пример

```
x =  
y = ...  
while (x < 100) {  
    x = x + 1  
    y = y + x  
}
```

```
x0 = ...  
y0 = ...  
if (x0 ≥ 100) goto next  
loop: x1 =  $\varphi(x_0, x_2)$   
      y1 =  $\varphi(y_0, y_2)$   
      x2 = x1 + 1  
      y2 = y1 + x2  
      if (x2 < 100) goto loop  
next: x3 =  $\varphi(x_0, x_2)$   
      y3 =  $\varphi(y_0, y_2)$ 
```

- ◇ **Выход из затруднения:** по определению любая φ -функция (а значит и $\varphi(x_0, x_2)$) **читает только один из своих аргументов**, а именно тот аргумент, который соответствует последнем пройденному ребру ГПУ.

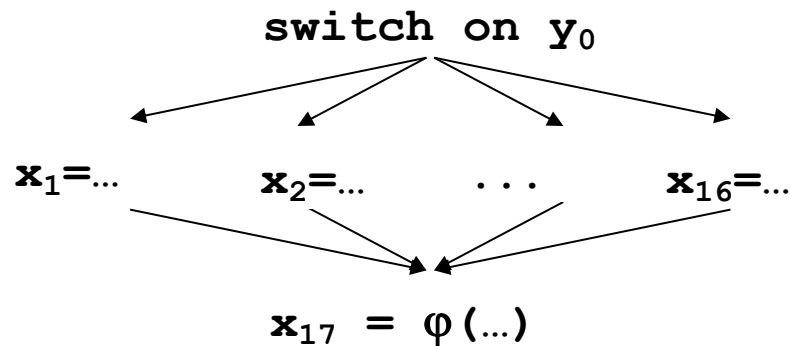
Поэтому φ -функция не прочитает неопределенного значения.

6.1 SSA-форма

6.1.3. Количество аргументов φ -функции

- ◇ По определению у φ -функции может быть любое число аргументов.

Например, на рисунке φ -функция, у которой 16 аргументов:



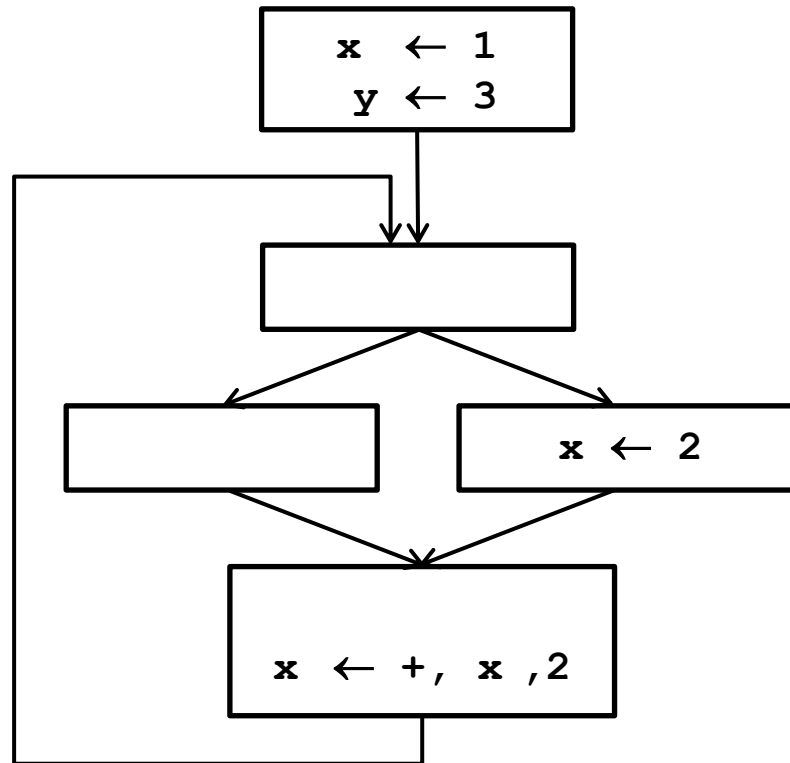
6.2 Построение SSA-формы

6.2.1. Постановка задачи

- ◇ Для преобразования процедуры в SSA-форму **компилятор** должен:
 - ◇ вставить в точки сбора необходимые ϕ -функции для каждой переменной;
 - ◇ переименовать переменные (в том числе и временные) таким образом, чтобы выполнялись следующие два правила:
 - (1) каждое определение имеет индивидуальное имя; и
 - (2) каждое использование ссылается на единственное определение.

6.2 Построение SSA-формы

6.2.3. Базовый алгоритм построения SSA-формы. Пример



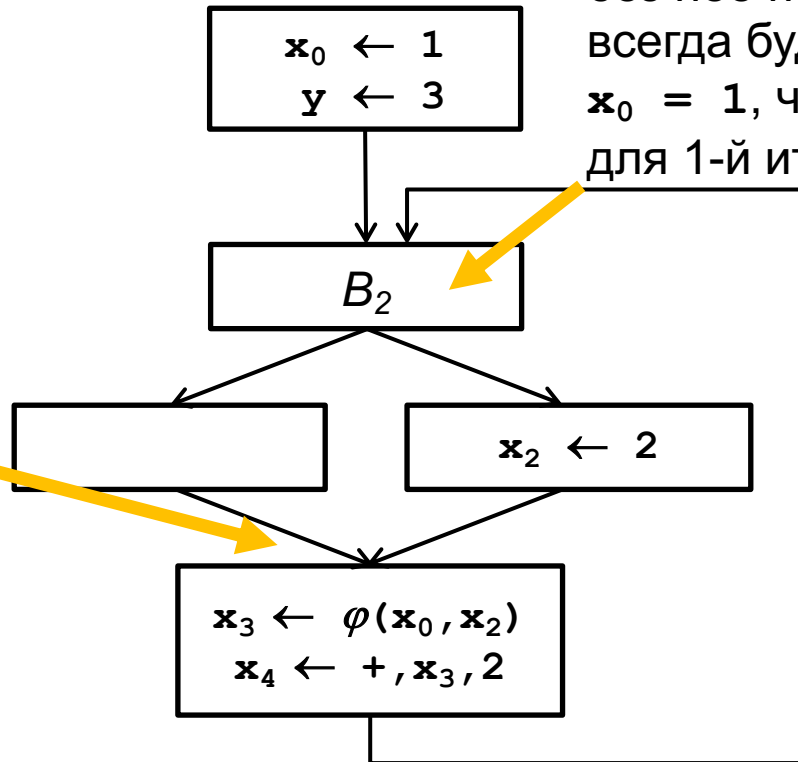
6.2 Построение SSA-формы

6.2.3. Базовый алгоритм построения SSA-формы. Пример

(Неправильное решение)

Не хватает еще одной ϕ -функции от x_0 и x_4 :
без нее по левой дуге всегда будет приходить $x_0 = 1$, что верно только для 1-й итерации цикла

Без $\phi(x_0, x_4)$ в B_2 по левой дуге всегда будет приходить $x_0 = 1$, и компилятор может подставить значение $x_0 = 1$ внутрь ϕ -функции, что приведет к генерации некорректного кода.



Как можно было избежать этой ошибки: при построении максимальной SSA-формы нужно вставлять ϕ -функции для ВСЕХ ГЛОБАЛЬНЫХ* переменных процедуры во ВСЕ точки сбора.

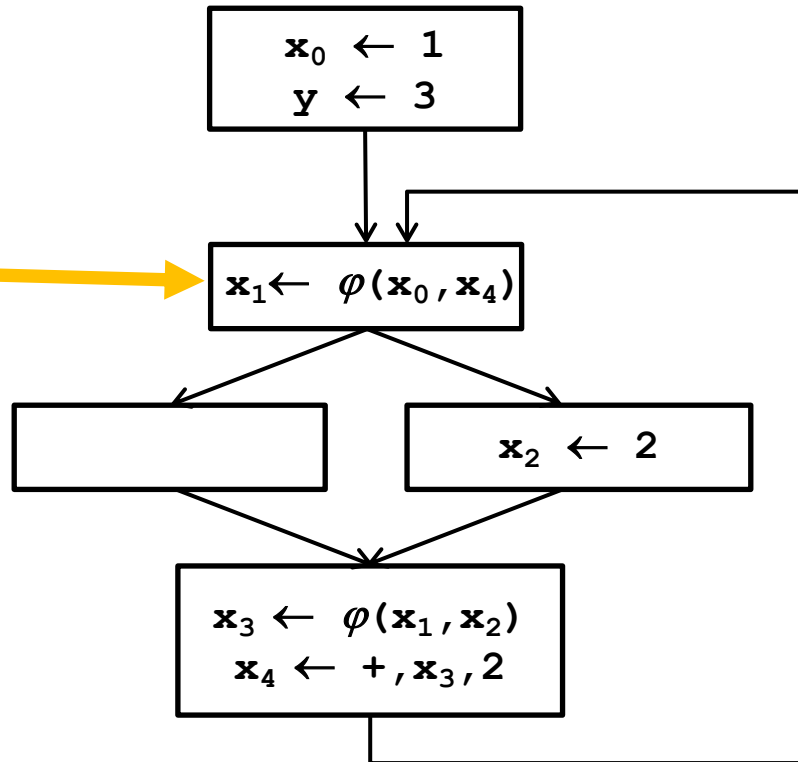
(*) Глобальные переменные – те, которые живы на границе базовых блоков

6.2 Построение SSA-формы

6.2.3. Базовый алгоритм построения SSA-формы. Пример

Как избежать такой ошибки: при построении максимальной SSA-формы нужно вставлять ϕ -функции для ВСЕХ ГЛОБАЛЬНЫХ переменных процедуры во ВСЕ точки сбора.

(Правильное решение)



Глобальные переменные – те переменные, которые живы на границе хотя бы одного из базовых блоков.

«Глобальные» – для целей построения SSA-формы (а не в смысле языка программирования). Было бы не совсем корректно говорить, что глобальные переменные – это те, которые имеют определение и использование в разных блоках. Например, определение может находиться после использования в одном и том же блоке внутри цикла. Кроме того, возможны вырожденные случаи: «мертвые» определения в разных блоках (т.е. без последующего использования), или использование без предшествующего определения. Во всех этих случаях могут потребоваться ϕ -функции.

6.2 Построение SSA-формы

6.2.2. Базовый алгоритм построения SSA-формы

- ◇ **Вход:** программа в промежуточном представлении
- ◇ **Выход:** промежуточное представление программы в SSA-форме
- ◇ **Метод:** Выполнить следующие действия:
 - (1) *Вставить φ -функции:*
в начало каждого блока B , у которого $|Pred(B)| > 1$ вставить φ -функцию вида $y = \varphi(y, y, \dots)$ для **каждого глобального** имени y , которое встречается в функции.

(Т.е. интервал жизни y должен пересекать границы базовых блоков).
Вставленная φ -функция должна иметь по одному аргументу для каждого $B' \in Pred(B)$:
Порядок вставляемых φ -функций несуществен.

6.2 Построение SSA-формы

6.2.2. Базовый алгоритм построения SSA-формы

- ◇ **Вход:** программа в промежуточном представлении
- ◇ **Выход:** промежуточное представление программы в SSA-форме
- ◇ **Метод:** Выполнить следующие действия:
 - (2) *Переименовать переменные:*
 - (a) вычислить достигающие определения; при этом только одно определение будет достигать любого использования; это гарантируют вставленные ϕ -функции, которые тоже являются определениями;
 - (b) переименовать каждое использование (как основных, так и временных) переменных таким образом, чтобы новое имя соответствовало единственному определению, которое достигает его.

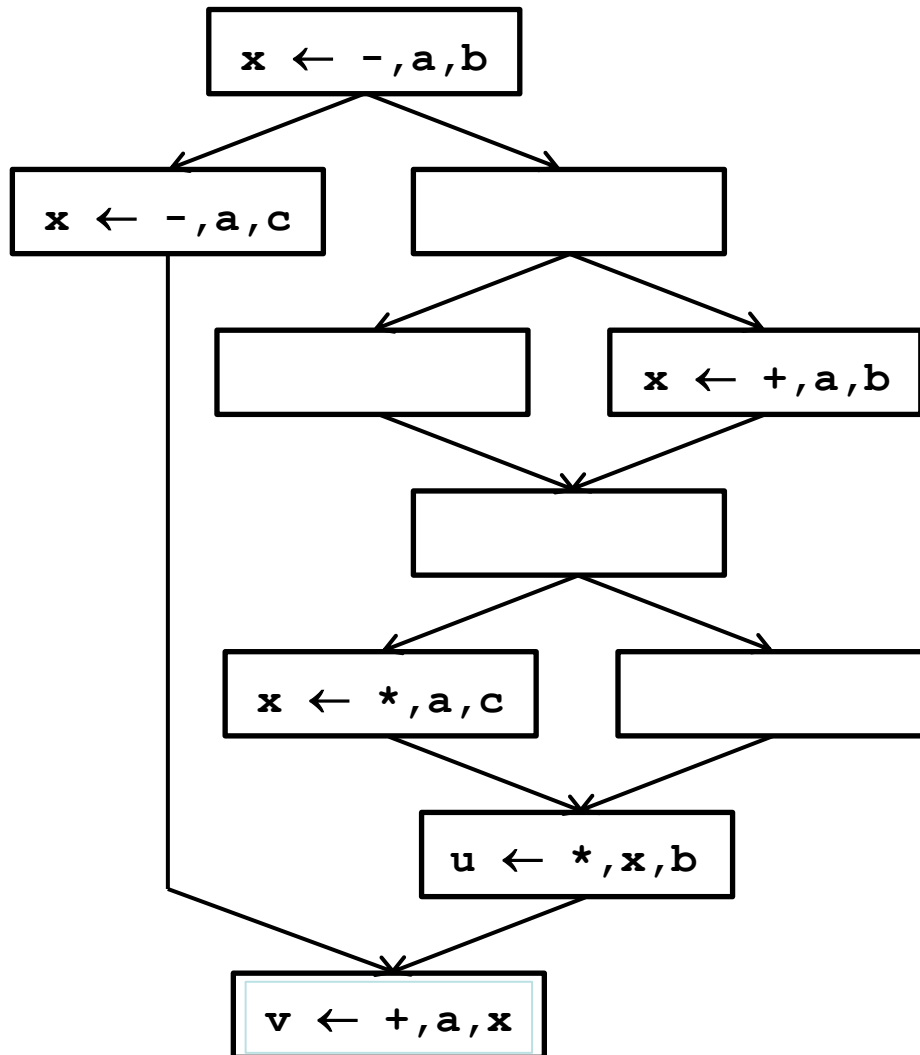
6.2 Построение SSA-формы

6.2.2. Базовый алгоритм построения SSA-формы

- ◇ **Вход:** программа в промежуточном представлении
- ◇ **Выход:** промежуточное представление программы в SSA-форме
- ◇ **Метод:** Выполнить следующие действия:
 - (3) *Отсортировать определения,*
достигающие каждой φ -функции, и для каждой φ -функции обеспечить соответствие имен ее аргументов путям, по которым определения этих аргументов достигают блока, содержащего указанную φ -функцию.

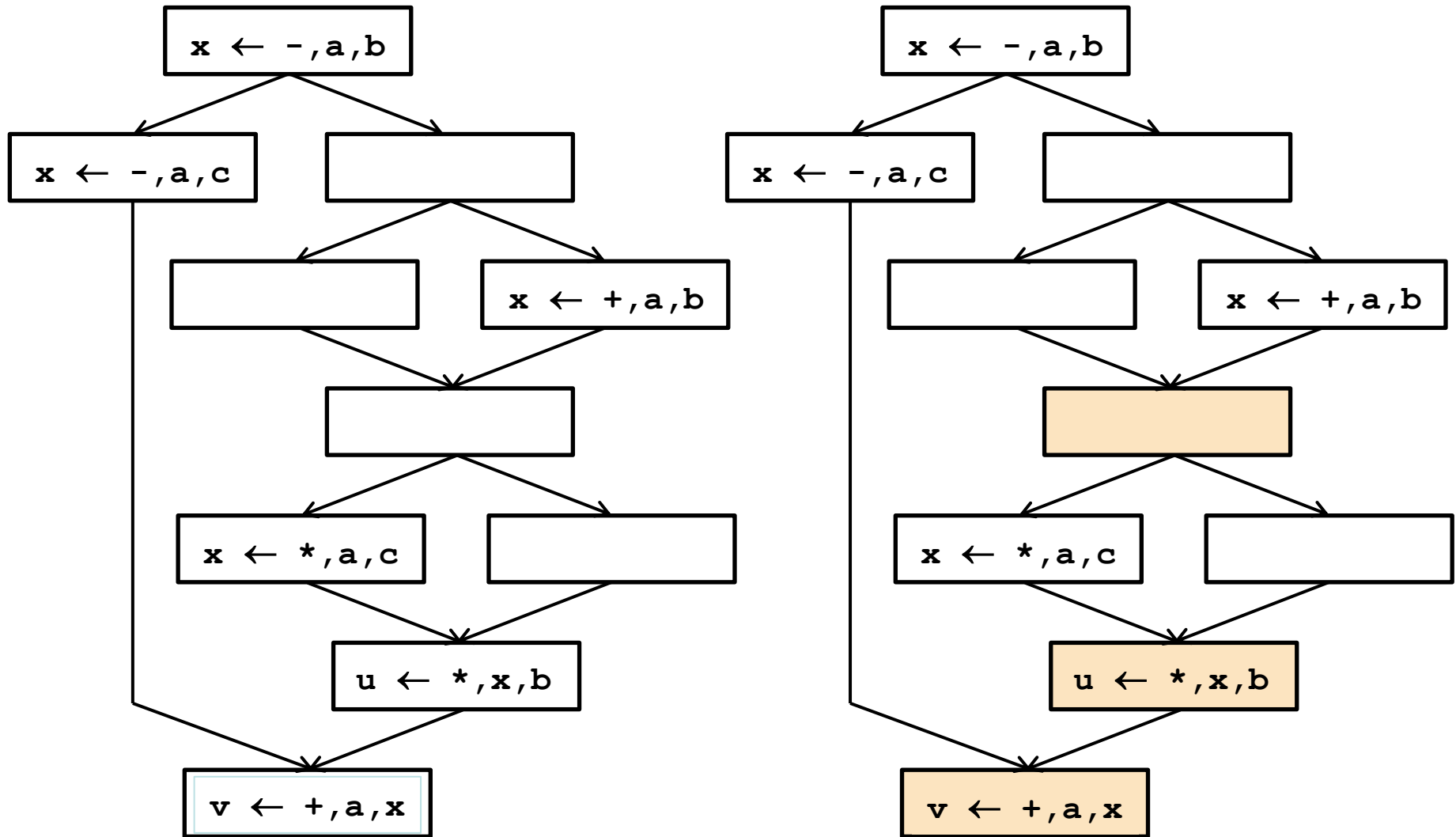
6.2 Построение SSA-формы

6.2.3. Базовый алгоритм построения SSA-формы. Пример



6.2 Построение SSA-формы

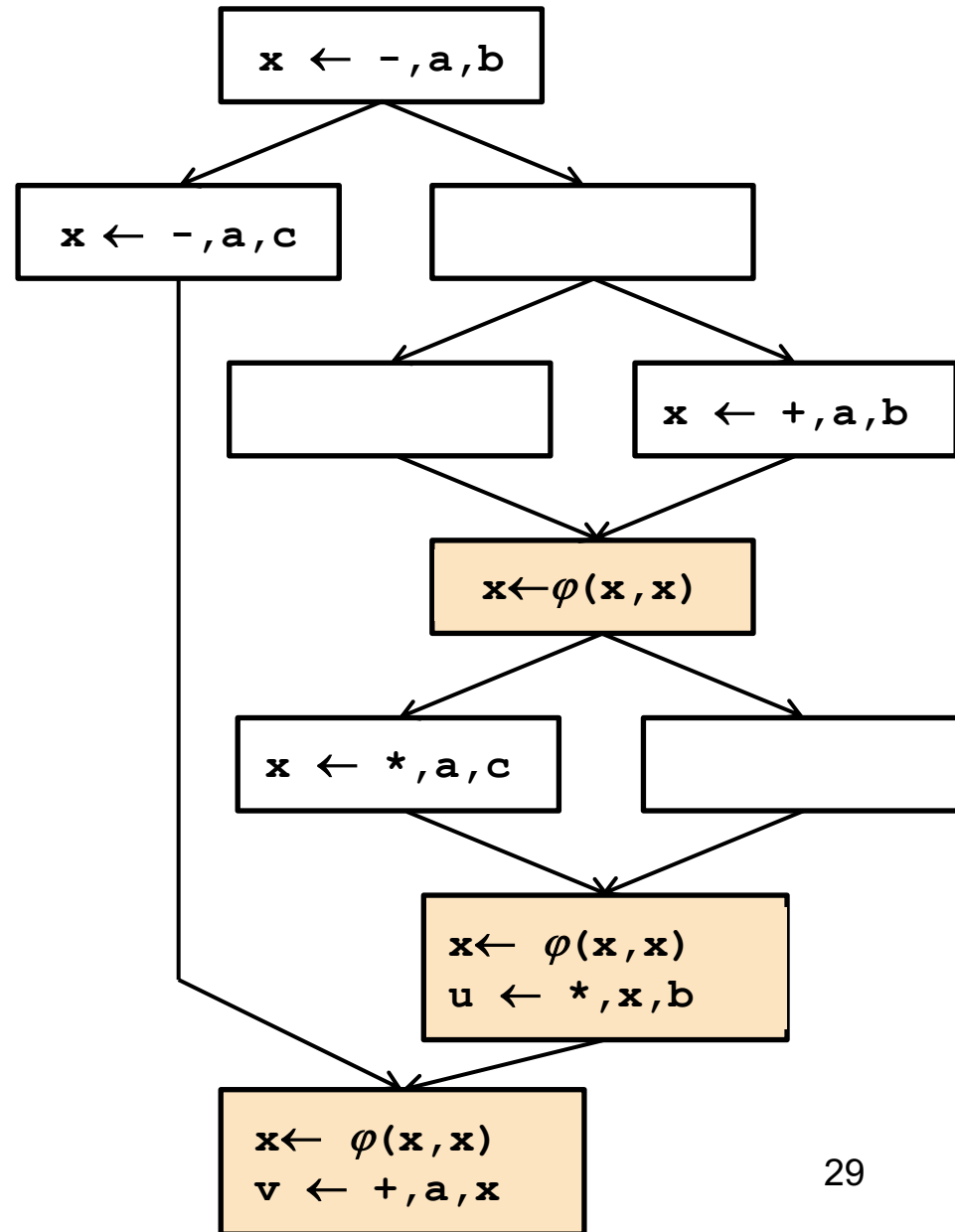
6.2.3. Базовый алгоритм построения SSA-формы. Пример



6.2 Построение SSA-формы

6.2.3. Базовый алгоритм построения SSA-формы. Пример

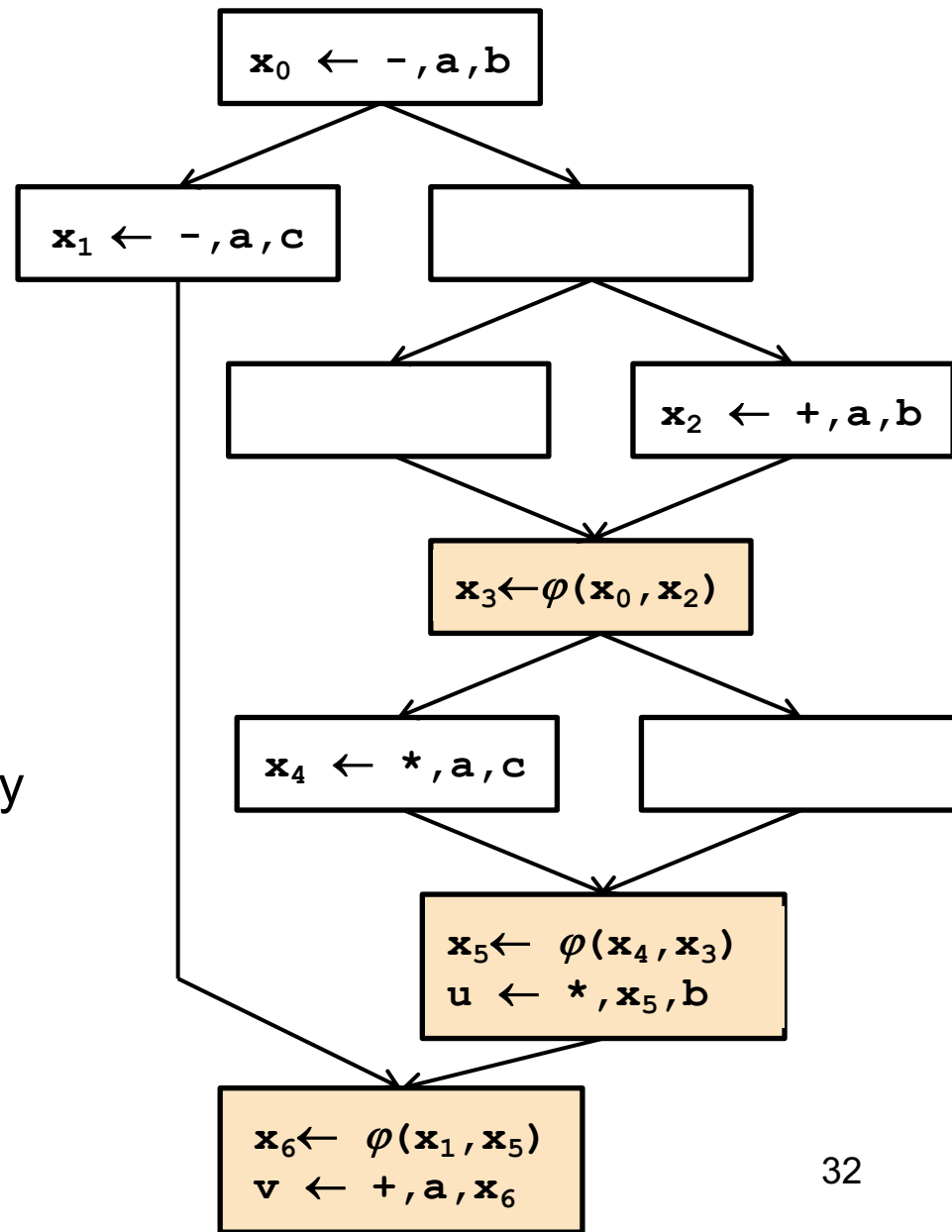
- ◇ Алгоритм вставил ϕ -функцию после каждой точки сбора ГПУ.
- ◇ Но это еще не SSA-форма, так как переменные еще не переименованы.



6.2 Построение SSA-формы

6.2.3. Базовый алгоритм построения SSA-формы. Пример

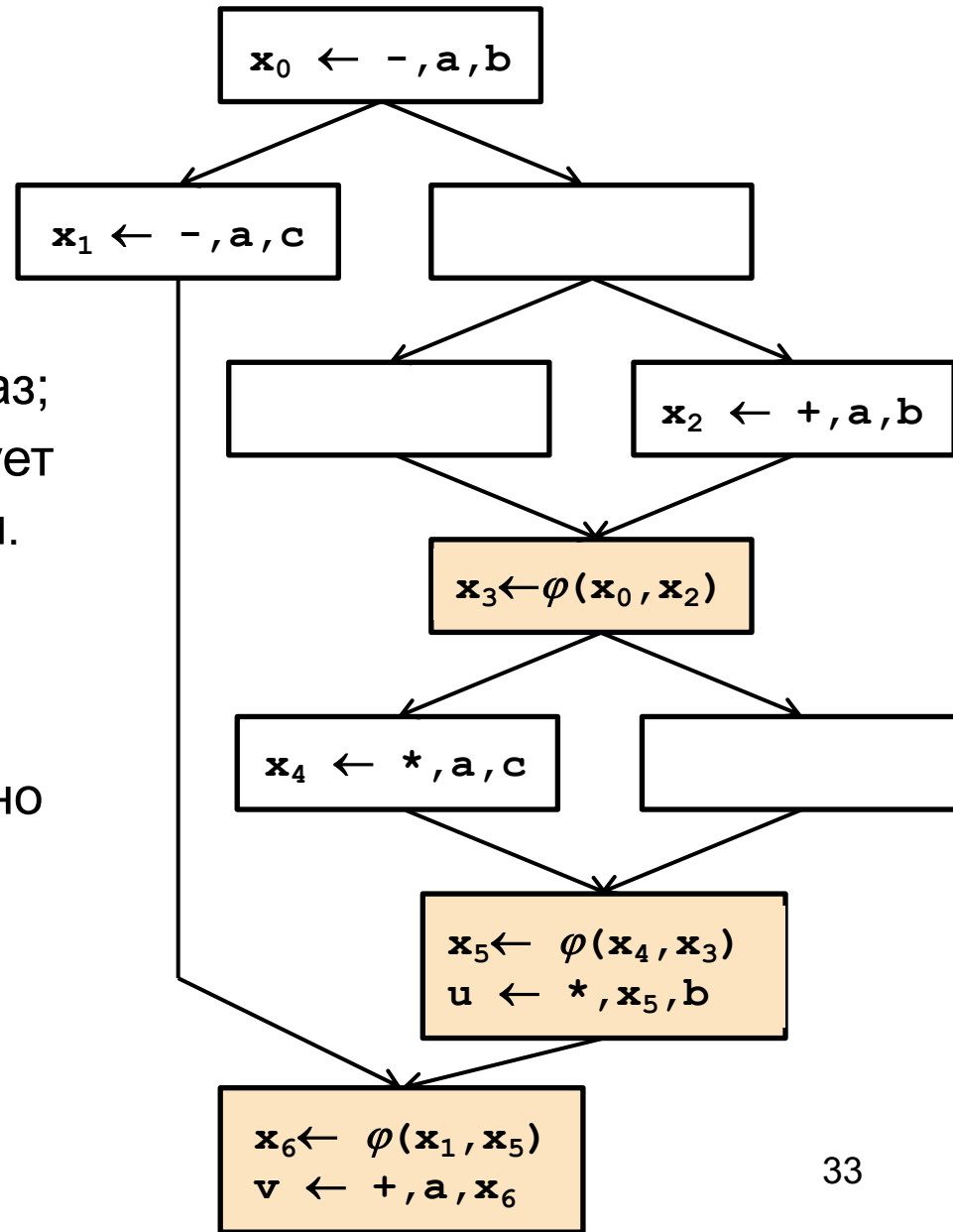
- ◇ Алгоритм вставил φ -функцию в каждой точке сбора ГПУ.
- ◇ После преобразования процедуры в SSA-форму
 - (1) внутри процедуры каждое определение создает уникальное имя;
 - (2) каждое использование обращается к единственному определению.



6.2 Построение *SSA*-формы

6.2.3. Базовый алгоритм построения *SSA*-формы. Пример

- ◇ Построенная *SSA*-форма корректна:
 - ◇ Каждая переменная определяется только один раз;
 - ◇ Каждое обращение использует имя отдельного определения.
- ◇ Построенная *SSA*-форма называется *максимальной SSA-формой*, так как она обычно содержит **намного больше** φ -функций, чем необходимо.



6.3 Построение частично усеченной SSA-формы

6.3.1. Постановка задачи

- ◇ По определению *частично усеченная SSA-форма* содержит меньше φ -функций, чем максимальная (но, к сожалению, как следует и из ее названия она **не всегда** содержит минимально возможное количество φ -функций).
- ◇ Частично-усеченная SSA-форма – один из вариантов «минимальной» SSA-формы, разработанных в конце прошлого века. Практика показала, что все эти формы дают для большей части программ одинаковые или незначительно отличающиеся результаты.
Наиболее простой алгоритм построения у частично-усеченной SSA-формы.

6.3 Построение частично усеченной SSA-формы

6.3.1. Постановка задачи

- ◇ По определению *частично усеченная SSA-форма* содержит меньше φ -функций, чем максимальная (но, к сожалению, как следует и из ее названия она **не всегда** содержит минимально возможное количество). Чтобы не всегда вставлять φ -функции необходимо **для каждой точки сбора** уметь выяснять, какие переменные нуждаются в φ -функциях, а какие нет.
- ◇ Частично усеченная SSA-форма — это форма прошлого века. Практика показала, что все эти формы дают для большей части программ одинаковые или незначительно отличающиеся результаты. Наиболее простой алгоритм построения у частично-усеченной SSA-формы.

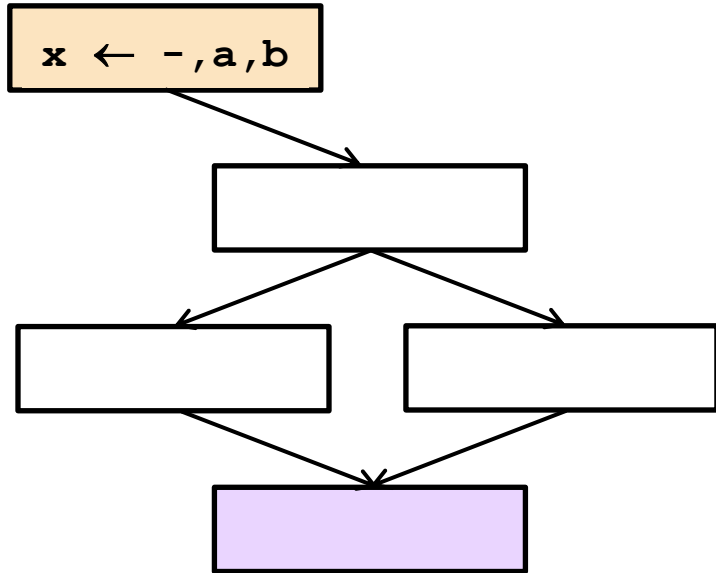
6.3 Построение частично усеченной SSA-формы

6.3.1. Постановка задачи

- ◇ По определению *частично усеченная SSA-форма* содержит меньше φ -функций, чем максимальная (но, к сожалению, как следует и из ее названия она **не всегда** содержит минимально возможное количество φ -функций). Чтобы не всегда вставлять φ -функции необходимо **для каждой точки сбора** уметь выяснять, какие переменные нуждаются в φ -функциях, а какие нет. Или «минимализация» SSA-формы — задача прошлого века. Практика: **для каждого определения переменной** уметь находить множество всех точек сбора, которые порождают значение, порожденное этим определением. И частично усеченной SSA-формы.
- ◇ Частично усеченная SSA-форма — это такая форма, в которой для каждой точки сбора можно найти минимальное количество φ -функций, которые необходимо вставить в код, чтобы сохранить значение переменной, порожденной этим определением.

6.3 Построение частично усеченной SSA-формы

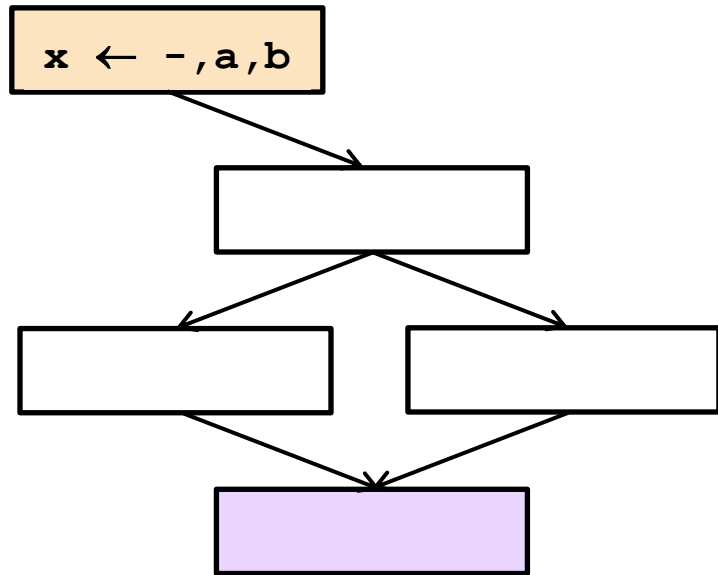
6.3.1. Постановка задачи



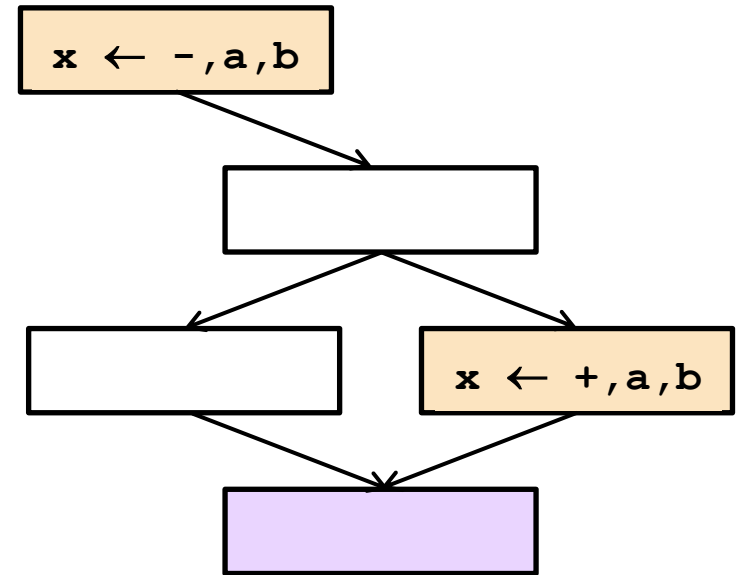
В лиловом блоке ϕ -функция для **x** **не нужна**, так как и слева, и справа приходит одно и то же значение **x**

6.3 Построение частично усеченной SSA-формы

6.3.1. Постановка задачи



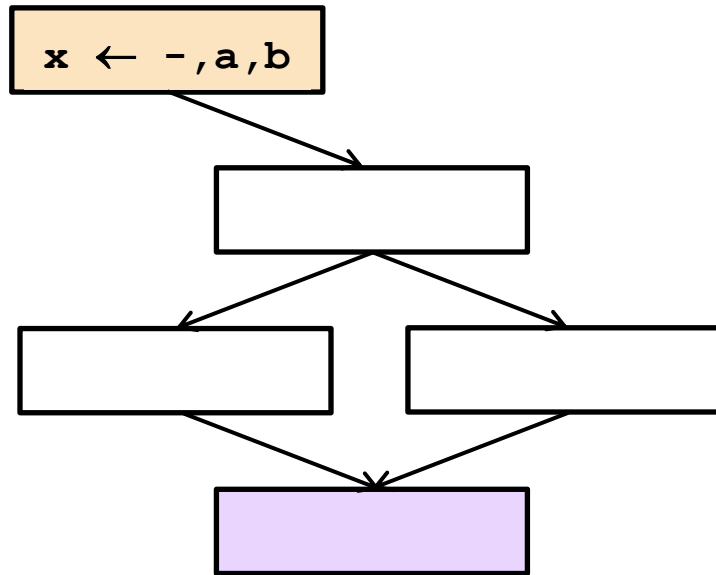
В лиловом блоке φ -функция для **x** **не нужна**, так как и слева, и справа приходит одно и то же значение **x**



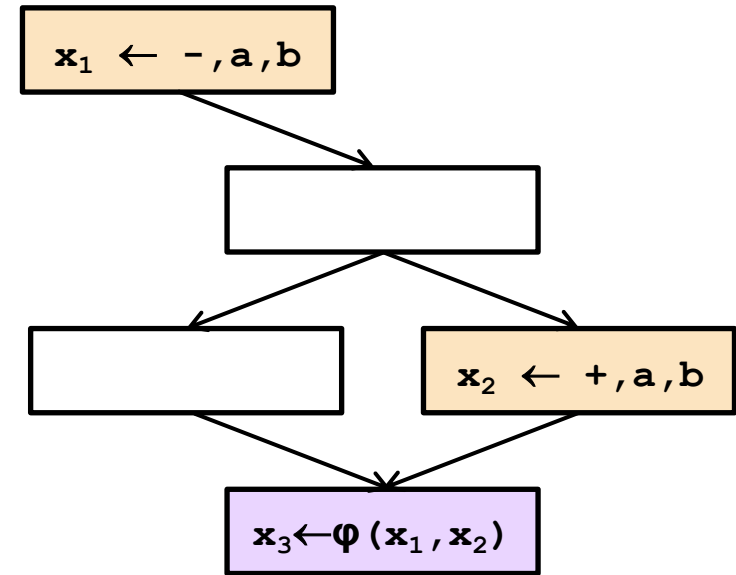
В лиловом блоке **нужна** φ -функция для **x** , так как справа появился блок, в котором вычисляется еще одно значение **x**

6.3 Построение частично усеченной SSA-формы

6.3.1. Постановка задачи



В лиловом блоке ϕ -функция для x **не нужна**, так как и слева, и справа приходит одно и то же значение x

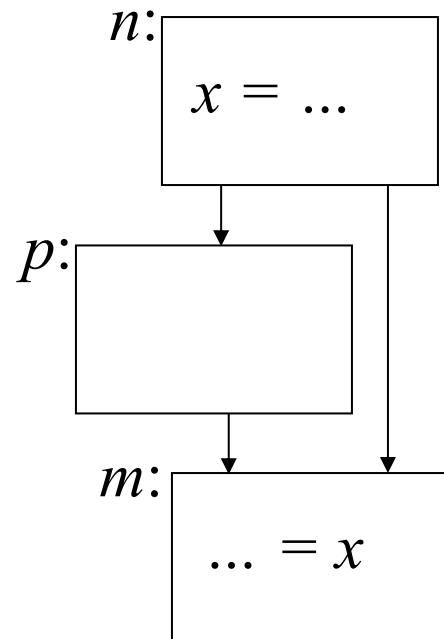


После переименования SSA-имена левого и правого аргументов ϕ -функции $\phi(x, x)$ и ее результата должны быть такими, как показано на рисунке, так как значение x_1 поступает слева, а значение x_2 — справа.

6.3 Построение частично усеченной SSA-формы

6.3.1. Постановка задачи

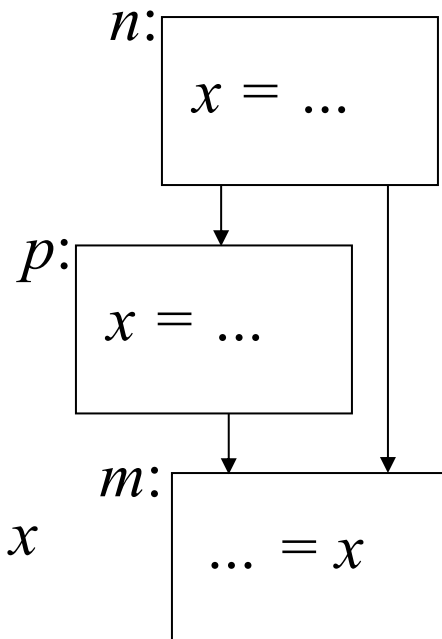
- ◇ Пусть $Dom(m)$ – множество доминаторов узла m .
- ◇ Если $n \in Dom(m)$, определение x_0 **в вершине n** не требует φ -функции в узле m , так как каждый путь, который достигает m , проходит через n .



6.3 Построение частично усеченной SSA-формы

6.3.1. Постановка задачи

- ◇ Пусть $Dom(m)$ – множество доминаторов узла m .
- ◇ Если $n \in Dom(m)$, определение x_0 **в вершине n** не требует φ -функции в узле m , так как каждый путь, который достигает m , проходит через n .
- ◇ Если $n \in Dom(m)$, единственным вариантом, при котором определение x_0 не достигнет узла m , является «вклинивание» еще одного определения x в некотором узле $p \notin Dom(m)$, расположенном между n и m .
При этом φ -функцию будет требовать не определение x в узле n , а его переопределение в узле p .



6.3 Построение частично усеченной SSA-формы

6.3.1. Постановка задачи

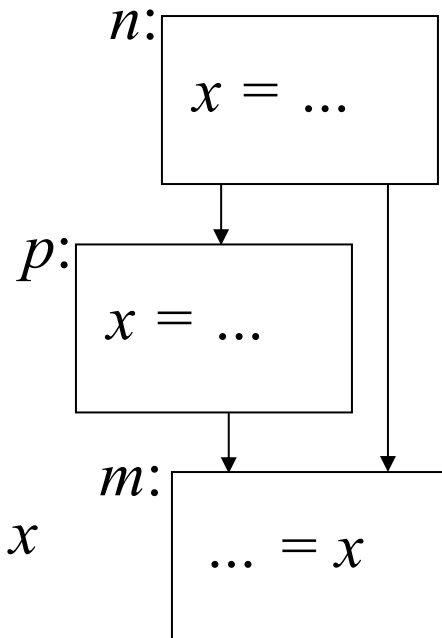
◇ Пусть $Dom(m)$ – множество доминаторов узла m .

◇ Нетрудно заметить, что блок с меткой m – граница доминирования блока с меткой p .

Если путь, по которому выполнение программы достигает m , проходит через n .

◇ Если $n \in Dom(m)$, единственным вариантом, при котором определение x_0 не достигнет узла m , является «вклинивание» еще одного определения x в некотором узле $p \notin Dom(m)$, расположенном между n и m .

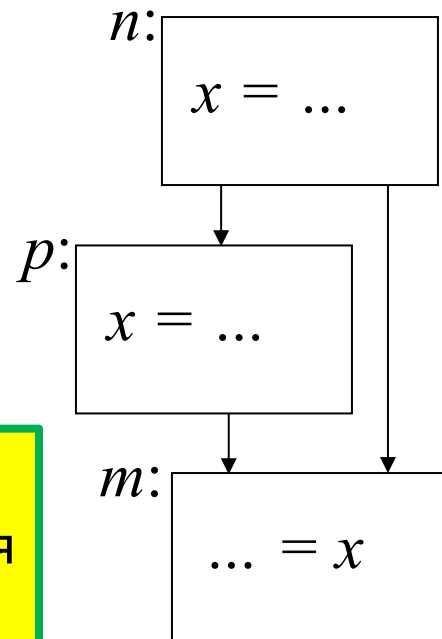
При этом φ -функцию будет требовать не определение x в узле n , а его переопределение в узле p .



6.3 Построение частично усеченной SSA-формы

6.3.1. Постановка задачи

- ◇ Пусть $Dom(m)$ – множество доминаторов узла m .
- ◇ Нетрудно заметить, что блок с меткой m – граница доминирования блока с меткой p .
Если блок с меткой p доминирует над блоком с меткой m , то любой путь, который достигает m , проходит через p .
- ◇ Если $n \in Dom(m)$, единственным вариантом, при котором определение x_0



Напоминание.

Границей доминирования $DF(n)$ узла n называется множество узлов m , удовлетворяющих условиям:

- (1) n является доминатором предшественника p узла m : $p \in Pred(m) \ \&\& \ n \in Dom(p)$
- (2) n не является строгим доминатором m :
$$n \notin (Dom(m) - \{m\})$$

6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение φ -функций

- ◇ Базовый алгоритм помещал по φ -функции для **каждой** глобальной переменной в начало **каждой** вершины сбора.
- ◇ Границы доминирования позволяют более точно определить, какие именно φ -функции необходимы в данной вершине.

Правило: Определение переменной x в базовом блоке B требует соответствующей φ -функции в каждой вершине из $DF(B)$.

При этом вставленная φ -функция становится новым определением x , так что могут потребоваться новые φ -функции.

6.3 Построение частично усеченной SSA-формы

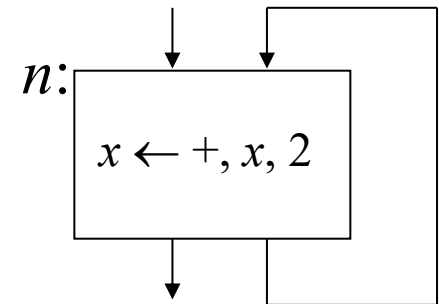
6.3.2. Размещение φ -функций

- ◇ Базовый алгоритм помещал по φ -функции для **каждой** глобальной переменной в начало **каждой** вершины сбора.
- ◇ Границы доминирования позволяют более точно определить, какие именно φ -функции необходимы в данной вершине.

Правило: Определение переменной x в базовом блоке B требует соответствующей φ -функции в каждой вершине из $DF(B)$.

При этом вставленная φ -функция становится новым определением x , так что могут потребоваться новые φ -функции.

- ◇ **Пример.** $DF(n) = \{n\}$



6.3 Построение частично усеченной SSA-формы

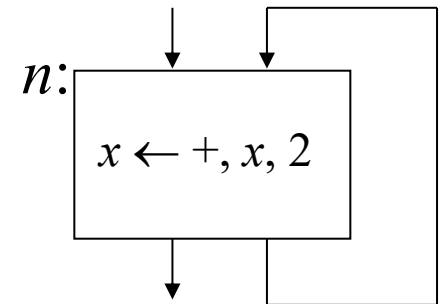
6.3.2. Размещение φ -функций

- ◇ Базовый алгоритм помещал по φ -функции для **каждой** глобальной переменной в начало **каждой** вершины сбора.
- ◇ Границы доминирования позволяют более точно определить, какие именно φ -функции необходимы в данной вершине.

Правило: Определение переменной x в базовом блоке B требует соответствующей φ -функции в каждой вершине из $DF(B)$.

!!! При этом вставленная φ -функция становится новым определением x , так что могут потребоваться новые φ -функции.

- ◇ **Пример.** $DF(n) = \{n\}$

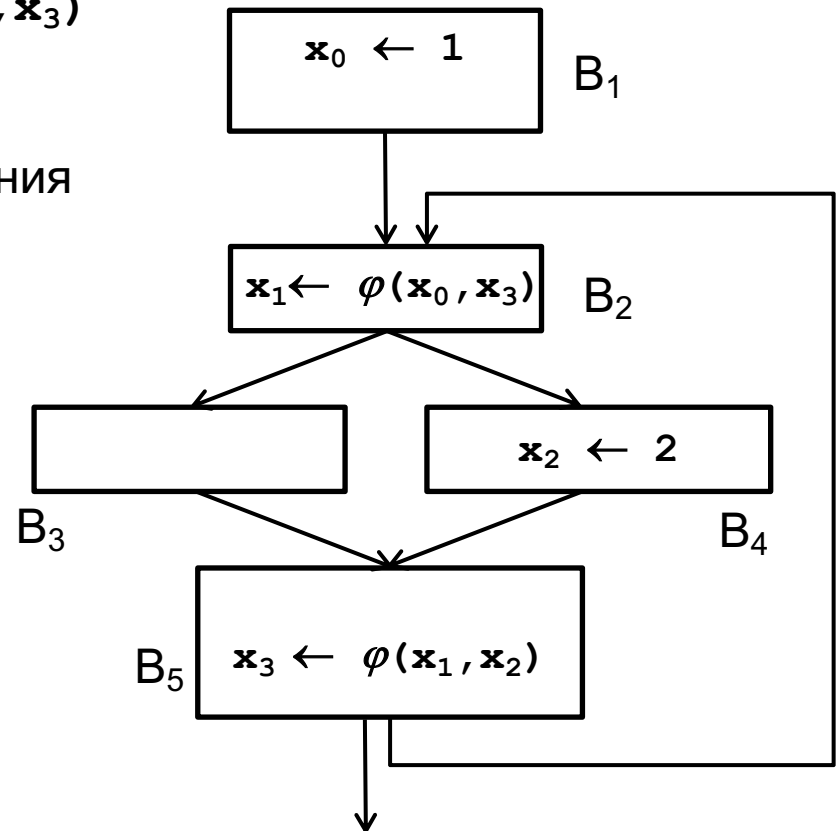


6.2 Построение SSA-формы

6.2.3. Базовый алгоритм построения SSA-формы. Пример

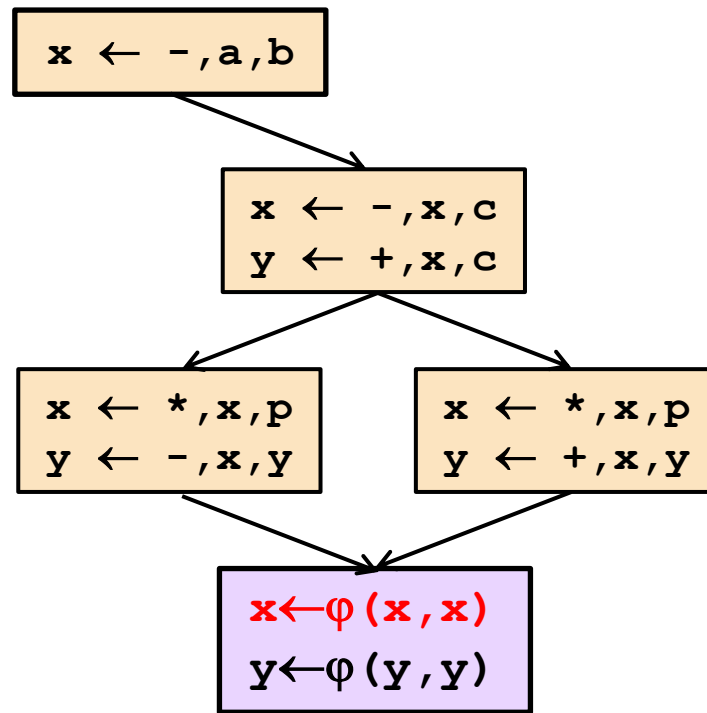
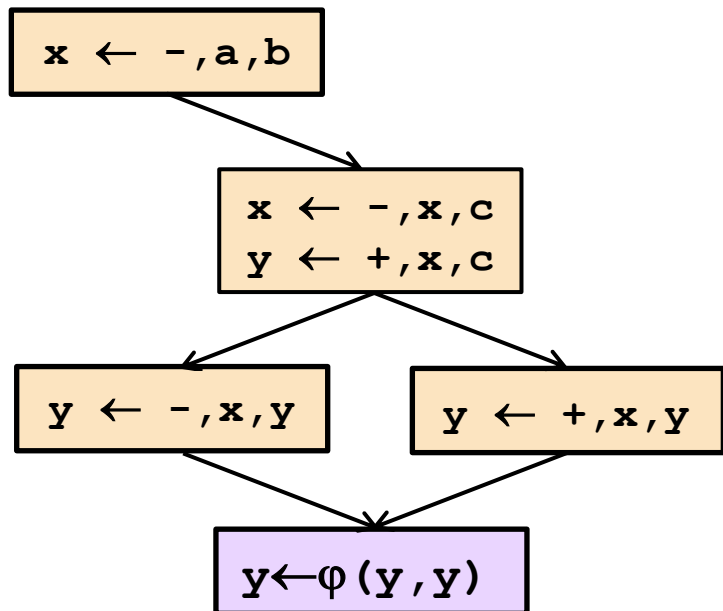
«При этом вставленная φ -функция становится новым определением x , так что могут потребоваться новые φ -функции.»

В данном примере φ -функция $x_1 \leftarrow \varphi(x_0, x_3)$ в B_2 появляется как раз потому, что только что добавленное определение $x_3 \leftarrow \varphi(x_1, x_2)$ в B_5 потребует добавления φ -функции в блок на своей границе доминирования (в B_2)



6.3 Построение частично усеченной SSA-формы

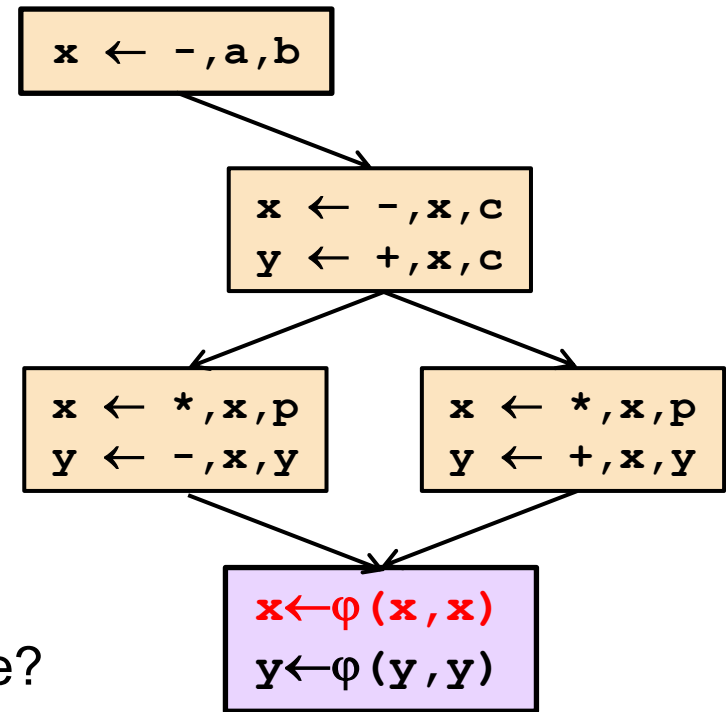
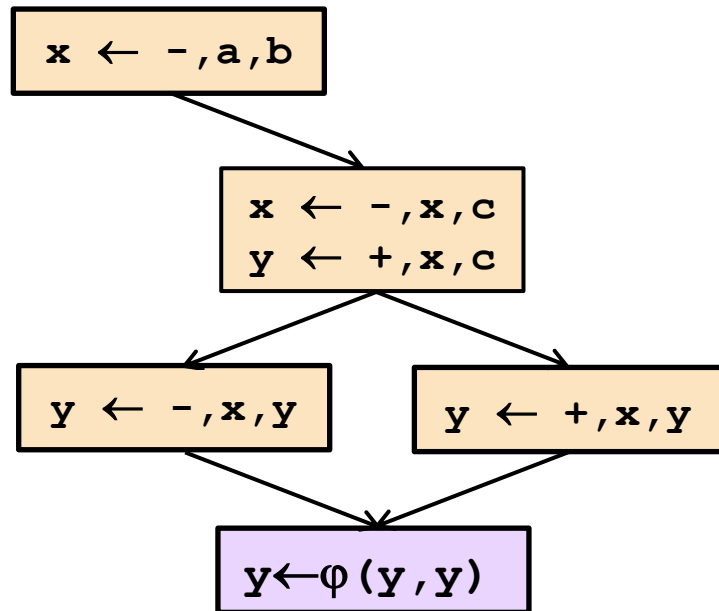
6.3.1. Постановка задачи. Пример



Какие φ -функции нужны в лиловом блоке?

6.3 Построение частично усеченной SSA-формы

6.3.1. Постановка задачи. Пример



Какие ϕ -функции нужны в лиловом блоке?

В ϕ -функцию, выделенную красным цветом, и справа, и слева поступают одинаковые значения x . Но алгоритм построения SSA формы этого в общем случае не учитывает, да и ничего с этим поделать не сможет, т. к. у каждого нового определения должен быть свой номер, а при слиянии путей, на которых одна переменная доступна с разными номерами, обязана быть ϕ -функция. Разве что одинаковое выражение поднять в доминатор, но в задачи при построении SSA-формы это не входит.

6.3 Построение частично усеченной SSA-формы

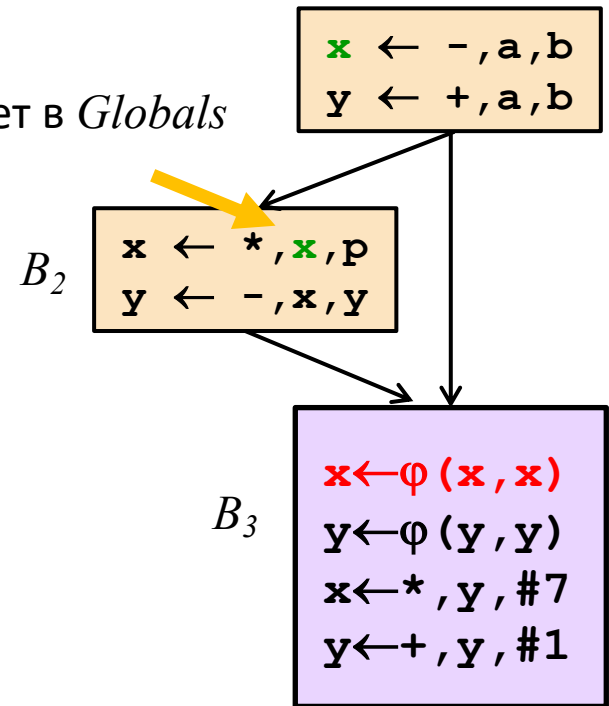
6.3.2.1. Частично-усеченная и усеченная SSA-формы

x используется в блоке до определения, т.е. x попадет в *Globals*

До построения SSA, в B_3 переменная x не используется до переопределения, поэтому не может иметь живой φ -функции.

Усеченная форма: x , определяемая в блоке B_2 , не жива за пределами блока, поэтому **φ -функция** (в B_3) для нее не требуется.

Но для того, чтобы это установить, нужно выполнить анализ живых переменных.



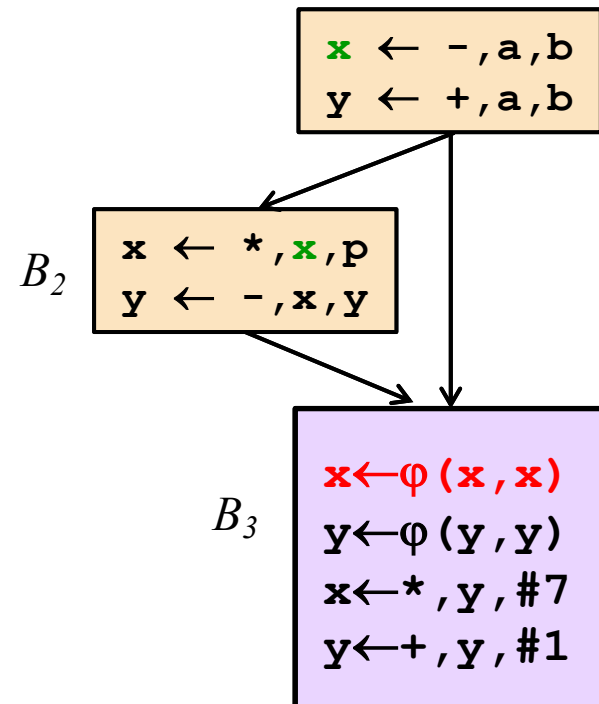
Частично-усеченная форма: Компромиссный вариант – не требует анализа живых переменных, φ -функции вставляются только для *глобальных имен*, т.е. имён, живых на границе какого-либо из базовых блоков.

Но это не гарантирует, что φ -функция жива (т.к. переменная могла попасть в глобальные, т.к. была жива на границе какого-то другого блока, а не того, в котором вставляется φ -функция), и может приводить к появлению избыточных **φ -функций** (например, в B_3).

6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение ϕ -функций

- ◇ ϕ -функция должна быть *живой*, т.е. живой должна быть переменная, которую эта ϕ -функция определяет. (пример на рисунке справа).
Отметим, что если выполнить анализ живых переменных до построения SSA, то $x, y \in def_{B_3}$, а $y \in use_{B_3}$ (на входе в блок x мертвая, а y живая)



6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение ϕ -функций

- ◇ Переменная, которая не жива на границах всех базовых блоков, например, сначала определяется и затем используется в одном блоке (и только в нем), не может иметь живой ϕ -функции. Поэтому ϕ -функции нужно вставлять только для глобальных имен, т.е. имен переменных, определение и использование которых находятся в разных базовых блоках (либо использование предшествует определению). Этого, однако, не достаточно, чтобы предотвратить ситуацию с предыдущего слайда.

6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение φ -функций

- ◇ Перед размещением φ -функций вычисляется множество *Globals* глобальных имен. При вычислении *Globals* используется множество def_B , в которое добавляются переменные, определяемые в текущем блоке.
- ◇ Алгоритм вычисления множества *Globals* попутно вычисляет для каждого базового блока B множество def_B и для каждой переменной $x \in Globals$ – множество $Blocks(x)$ базовых блоков B , в которых определяется x .

6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение φ -функций

- ◇ Итак, перед размещением φ -функций вычисляется множество глобальных имен *Globals*. При вычислении *Globals* можно также использовать результаты анализа живых переменных (т.е. вместо отдельного построения def_B , можно использовать множество def , которое вычисляется в ходе анализа LV).
- ◇ Представленный алгоритм вычисления множества *Globals* попутно сам вычисляет для каждого базового блока B множество def_B и для каждой переменной $x \in Globals$ – множество $Blocks(x)$ базовых блоков B , в которых определяется x .

6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение φ -функций

Алгоритм построения множеств *Globals* и *Blocks(x)*

- ◇ **Вход:** ГПУ
- ◇ **Выход:** множество *Globals*,
множества *Blocks(x)* для каждой переменной x – блоки, в которых определяется x
множества def_B для каждого базового блока B – переменные, которые определяются в B .
- ◇ **Метод:** Выполнить следующие действия:

6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение φ -функций

Алгоритм построения множеств *Globals* и *Blocks(x)*

Globals = $\emptyset$;

for each variable *x* **do** *Blocks(x)* = $\emptyset$;

for each block *B* **do** {

$def_B = \emptyset$;

for each instruction $i \in B$ **do** {

 // пусть команда i имеет вид: $x \leftarrow op, y, z$

if $y \notin def_B$ **then** *Globals* = *Globals* $\cup \{y\}$;

if $z \notin def_B$ **then** *Globals* = *Globals* $\cup \{z\}$;

 // $y, z \notin def_B$ – если они определены в другом блоке

$def_B = def_B \cup \{x\}$;

$Blocks(x) = Blocks(x) \cup \{B\}$

 }

}

6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение φ -функций

Алгоритм построения множеств *Globals* и *Blocks(x)*

$Globals = \emptyset$;

for each variable x **do**

for each block B **do** {

$def_B = \emptyset$;

for each instruction $i \in B$ **do** {

// пусть команда i имеет вид: $x \leftarrow op, y, z$

if $y \notin def_B$ **then** $Globals = Globals \cup \{y\}$;

if $z \notin def_B$ **then** $Globals = Globals \cup \{z\}$;

// $y, z \notin def_B$ – если они определены в другом блоке

$def_B = def_B \cup \{x\}$;

$Blocks(x) = Blocks(x) \cup \{B\}$

}

}

Аналогично нужно обрабатывать использования переменных и в других операциях, например, в условном операторе, или при вызове функций:

```
if (y < z) { }  
foo(y, z);
```

Другие определения обрабатываются так же, как левая часть присваивания (например, объявление параметра):

```
param_decl x
```

6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение φ -функций

Алгоритм размещения φ -функций

◇ **Вход:** исходный ГПУ

◇ **Выход:** преобразованный ГПУ

```
for each name  $x \in \text{Globals}$  do {  
     $\text{WorkList} = \text{Blocks}(x)$ ;  
    for each block  $B \in \text{WorkList}$  do {  
        for each block  $D \in \text{DF}(B)$  do {  
            if ( $D$  не содержит  $\varphi$ -функции для  $x$ ) {  
                Вставить  $\varphi$ -функцию для  $x$  в  $D$ ;  
                 $\text{WorkList} = \text{WorkList} \cup \{D\}$ ;  
            }  
        }  
         $\text{WorkList} = \text{WorkList} - \{B\}$ ;  
    }  
}
```

6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение ϕ -функций

Алгоритм размещения ϕ -функций

◇ **Вход:** исходный ГПУ

◇ **Выход:** преобразованный ГПУ

```
for each name  $x \in \text{Globals}$  do {  
     $\text{WorkList} = \text{Blocks}(x)$ ;  
    for each block  $B \in \text{WorkList}$  do {  
        for each block  $D \in \text{DF}(B)$  do {  
            if ( $D$  не содержит  $\phi$ -функции для  $x$ ) {  
                Вставить  $\phi$ -функцию для  $x$  в  $D$ ;  
                 $\text{WorkList} = \text{WorkList} \cup \{D\}$ ;  
            }  
        }  
         $\text{WorkList} = \text{WorkList} - \{B\}$ ;  
    }  
}
```

Чтобы учесть
новое
определение x ,
реализуемое
вставленной
 ϕ -функцией (!!!)

6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение φ -функций

- ◇ **Замечание 1.** Для каждого блока $B \in WorkList$ алгоритм вставляет φ -функцию в начало каждого блока $D \in DF(B)$.

Порядок φ -функций роли не играет.

После вставки φ -функции для x в блок D алгоритм добавляет D в *WorkList*, чтобы в дальнейшем учесть новое определение x в блоке D .

6.3 Построение частично усеченной SSA-формы

6.3.2. Размещение φ -функций

◇ **Замечание 2.** Для улучшения эффективности алгоритма необходимо избегать дублирования:

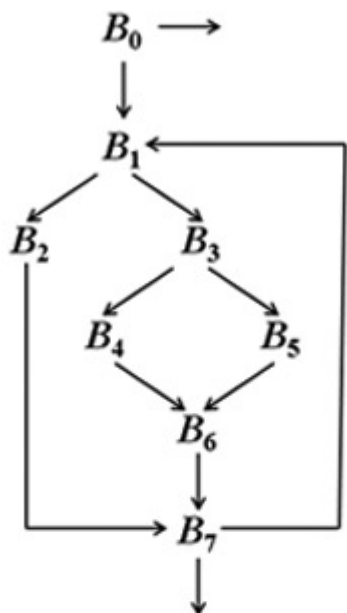
рассматриваемый блок может входить в состав границ доминирования нескольких блоков, входящих в *WorkList*; чтобы избежать вставления дублирующих φ -функций для переменной x , можно использовать множество блоков, которые уже содержат φ -функции для x , что быстрее, чем проверять каждый раз, какие φ -функции включены.

6.3 Построение частично усеченной SSA-формы

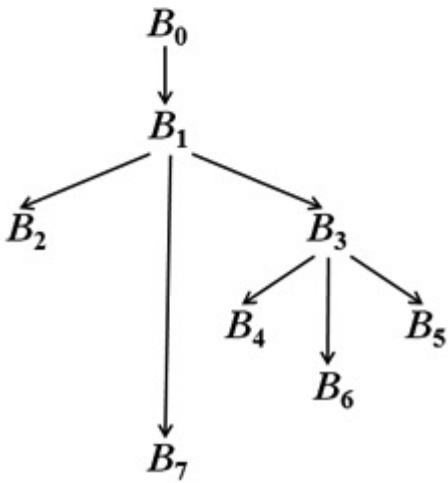
6.3.3. Размещение ϕ -функций. Пример.

B_0	$i \leftarrow 1$	B_5	$c \leftarrow$
B_1	$a \leftarrow$ $b \leftarrow$	B_3	$a \leftarrow$ $d \leftarrow$
B_2	$b \leftarrow$ $c \leftarrow$ $d \leftarrow$	B_7	$y \leftarrow +, a, b$ $z \leftarrow +, c, d$ $i \leftarrow +, i, 1$
B_6	$b \leftarrow$	B_4	$d \leftarrow$

Граф потока управления



Дерево доминаторов



6.3 Построение частично усеченной SSA-формы

6.3.3. Размещение φ -функций. Пример.

- ◇ Сначала вычисляются множества $Globals$ и $Blocks(x)$:
 $Globals = \{a, b, c, d, i\}$.

Множества $Blocks(x)$ представлены в таблице:

a	b	c	d	i
$\{B_1, B_3\}$	$\{B_1, B_2, B_6\}$	$\{B_2, B_5\}$	$\{B_2, B_3, B_4\}$	$\{B_0, B_7\}$

- ◇ Алгоритму размещения φ -функций потребуются также уже вычисленные границы доминирования:

n	0	1	2	3	4	5	6	7
$DF(B_n)$	$\emptyset$	$\{B_1\}$	$\{B_7\}$	$\{B_7\}$	$\{B_6\}$	$\{B_6\}$	$\{B_7\}$	$\{B_1\}$

6.3 Построение частично усеченной SSA-формы

6.3.3. Размещение φ -функций. Пример.

◇ Применяем алгоритм размещения φ -функций.

Берем первую переменную из $Globals = \{a, b, c, d, i\}$, т.е. a :

$Blocks(a) = \{B_1, B_3\}$, так что алгоритм должен вставить

φ -функцию для a в каждый блок из границ доминирования

$DF(B_1) = \{B_1\}$ и $DF(B_3) = \{B_7\}$.

Так, вставив φ -функцию для a в B_7 , получим новое определение a

– φ -функцию, вставленную в B_7 . Следовательно, необходимо

включить B_7 в $WorkList$ и вставить φ -функцию для a в каждый

блок из $DF(B_7) = \{B_1\}$. Но B_1 уже имеется в $WorkList$, так что

$DF(B_7)$ не добавляется к $WorkList$.

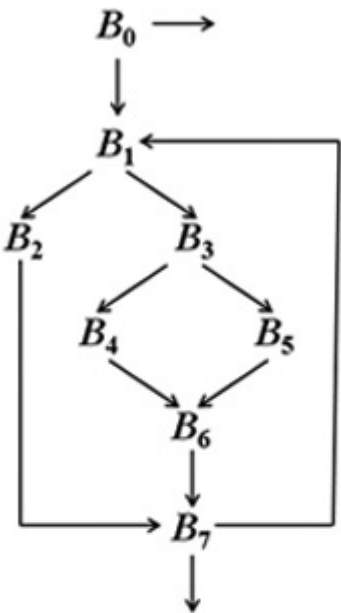
x	a	b	c	d	i
Insert $\varphi(x)$	$\{B_7, B_1\}$	$\{B_7, B_1\}$	$\{B_7, B_1, B_6\}$	$\{B_7, B_1, B_6\}$	$\{B_1\}$

6.3 Построение частично усеченной SSA-формы

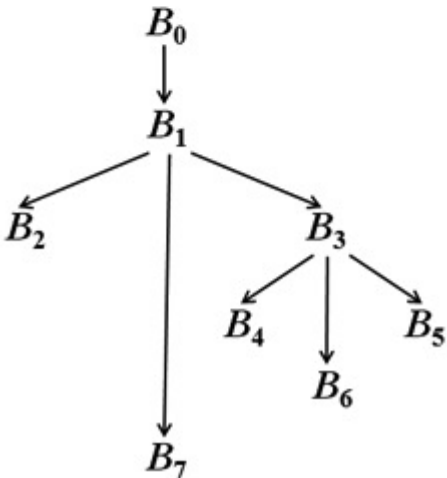
6.3.3. Размещение ϕ -функций. Пример.

B_0	$i \leftarrow 1$		
B_1	$a \leftarrow \phi(a, a)$ $b \leftarrow \phi(b, b)$ $c \leftarrow \phi(c, c)$ $d \leftarrow \phi(d, d)$ $i \leftarrow \phi(i, i)$ $a \leftarrow$ $b \leftarrow$	B_7	$a \leftarrow \phi(a, a)$ $b \leftarrow \phi(b, b)$ $c \leftarrow \phi(c, c)$ $d \leftarrow \phi(d, d)$ $y \leftarrow +, a, b$ $z \leftarrow +, c, d$ $i \leftarrow +, i, 1$
B_2	$b \leftarrow$ $c \leftarrow$ $d \leftarrow$	B_3	$a \leftarrow$ $d \leftarrow$
B_4	$d \leftarrow$	B_6	$c \leftarrow \phi(c, c)$ $d \leftarrow \phi(d, d)$ $b \leftarrow$
B_5	$c \leftarrow$		

Граф потока управления



Дерево доминаторов



6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

Алгоритм

- ◇ **Вход:** программа с размещенными φ -функциями
- ◇ **Выход:** программа, в которой каждой переменной сопоставлено ее *SSA-имя*.
- ◇ **Метод:** Сначала (в основном алгоритме) инициализируются стеки и счетчики, после чего из корня дерева доминаторов n_0 вызывается рекурсивная функция *Rename*.
Rename обрабатывает блок, рекурсивно вызывая его последователей по дереву доминаторов.
Закончив обрабатывать очередной блок, *Rename* выталкивает из стеков все имена, помещенные в них во время обработки блока.
Функция *NewName*, манипулируя со счетчиками и стеками, в случае необходимости создает новые имена.

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

Алгоритм

◇ Основной алгоритм:

```
for each  $i \in \text{Globals}$  do {  
    counter[i] = 0;  
    stack[i] =  $\emptyset$ ;  
};  
Rename ( $n_0$ ) ;
```

◇ Функция **newName**:

```
NewName(n) { // n – имя переменной  
    i = counter[n];  
    counter[n] += 1;  
    Push  $n_i$  onto stack[n];  
    return  $n_i$   
}
```

6.3 Построение частично усеченной SSA-формы

6.3.7. Переименование переменных.

Функция $Rename(B)$:

```
for each  $\phi$ -function  $\in B$ :  $x = \phi(\dots)$  do
    rename  $x$  as  $NewName(x)$  ;
for each instruction  $\in B$ :  $x \leftarrow op, y, z$  do{
    rewrite  $y$  as  $top(stack[y])$  ;
    rewrite  $z$  as  $top(stack[z])$  ;
    rewrite  $x$  as  $NewName(x)$  ;
}
for each successor of  $B$  in the flowgraph do
    fill in  $\phi$ -function parameters;
for each successor  $S$  of  $B$  in the
    dominator tree do  $Rename(S)$ 
for each instruction  $\in B$ :  $x \leftarrow op, y, z$  do
     $Pop(stack[x])$  ;
for each  $\phi$ -function  $\in B$ :  $x = \phi(\dots)$  do
     $Pop(stack[x])$  ;
```

6.3 Построение частично усеченной SSA-формы

6.3.7. Переименование переменных.

Функция $Rename(B)$:

```
for each  $\phi$ -function  $\in B$ :  $x = \phi(\dots)$  do
    rename  $x$  as  $newName(x)$  ;
for each instruction  $\in B$ :  $x \leftarrow op, y, z$  do{
    rewrite  $y$  as  $top(stack[n])$  ;
    rewrite  $z$  as  $top(stack[n])$  ;
    rewrite  $x$  as  $newName(x)$  ;
}
for each successor of  $B$  in the flowgraph do
    fill in  $\phi$ -function parameters;
for each successor  $S$  of  $B$  in the
    dominator tree do  $Rename(S)$ 
for each instruction  $\in B$ :  $x \leftarrow op, y, z$  do
     $Pop(stack[x])$  ;
for each  $\phi$ -function  $\in B$ :  $x = \phi(\dots)$  do
     $Pop(stack[x])$  ;
```

```
 $newName(n)$  {
     $i = counter[n]$  ;
     $counter[n] += 1$  ;
    Push  $n_i$  onto  $stack[n]$  ;
    return  $n_i$ 
}
```


6.3 Построение частично усеченной SSA-формы

6.3.7. Переименование переменных.

Функция $Rename(B)$:

```
for each  $\phi$ -function  $\in B$ :  $x = \phi(\dots)$  do
    rename  $x$  as  $newName(x)$  ;
for each instruction  $\in B$ :  $x \leftarrow op, y, z$  do{
    rewrite  $y$  as  $top(stack[x])$  ;
    rewrite  $z$  as  $top(stack[x])$  ;
    rewrite  $x$  as  $newName(x)$  ;
}
for each successor of  $B$  in the flowgraph do
    fill in  $\phi$ -function parameters;
for each successor  $S$  of  $B$  in the
    dominator tree do  $Rename(S)$ 
for each instruction  $\in B$ :  $x \leftarrow op, y, z$  do
     $Pop(stack[x])$  ;
```

```
 $newName(n)$  {
     $i = counter[n]$  ;
     $counter[n] += 1$  ;
    Push  $n_i$  onto  $stack[n]$  ;
    return  $n_i$ 
}
```

Типичная ошибка при выполнении $Pop(stack[x])$ – заодно уменьшить также и $counter[x]$. Этого делать не нужно!!!

6.3 Построение частично усеченной SSA-формы

6.3.7. Переименование переменных.

Функция $Rename(B)$:

```
for each  $\phi$ -fun  
    rename x
```

Аналогично нужно обрабатывать использования переменных и в других операциях, например, в условном операторе, или при вызове функций:

```
if (y < z) { }  
foo(y, z);
```

```
for each instruction  $\in B$ :  $x \leftarrow op, y, z$  do {  
    rewrite y as top(stack[y]);  
    rewrite z as top(stack[z]);  
    rewrite x as NewName(x);
```

```
}
```

```
for each succe  
    fill in  $\phi$ 
```

Другие определения обрабатываются так же, как левая часть присваивания (например, объявление параметра):

```
param_decl x
```

```
for each successor S of B in the  
    dominator tree do Rename(S)
```

```
for each instruction  $\in B$ :  $x \leftarrow op, y, z$  do  
    Pop(stack[x]);
```

```
for each  $\phi$ -function  $\in B$ :  $x = \phi(\dots)$  do  
    Pop(stack[x]);
```

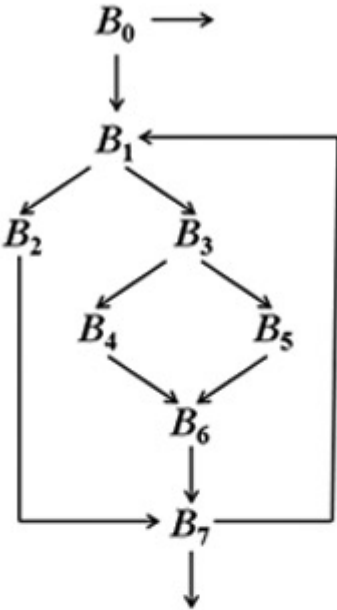
6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

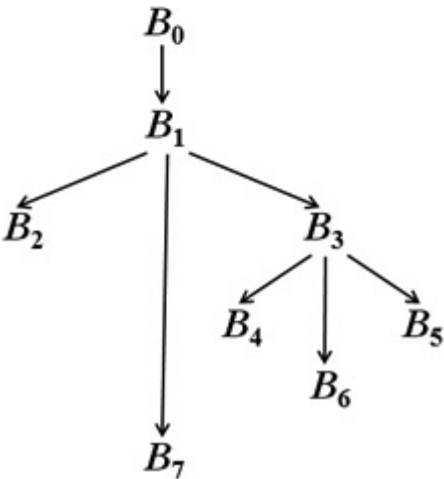
Применим алгоритм переименования к рассматриваемому примеру в предположении, что на входе в блок B_0 определены имена a_0, b_0, c_0, d_0 .

B_0	$i \leftarrow 1$	
B_1	$a \leftarrow \phi(a, a)$ $b \leftarrow \phi(b, b)$ $c \leftarrow \phi(c, c)$ $d \leftarrow \phi(d, d)$ $i \leftarrow \phi(i, i)$ $a \leftarrow$ $b \leftarrow$	B_7 $a \leftarrow \phi(a, a)$ $b \leftarrow \phi(b, b)$ $c \leftarrow \phi(c, c)$ $d \leftarrow \phi(d, d)$ $y \leftarrow +, a, b$ $z \leftarrow +, c, d$ $i \leftarrow +, i, 1$
B_2	$b \leftarrow$ $c \leftarrow$ $d \leftarrow$	B_3 $a \leftarrow$ $d \leftarrow$
B_4	$d \leftarrow$	B_6 $c \leftarrow \phi(c, c)$ $d \leftarrow \phi(d, d)$ $b \leftarrow$
B_5	$c \leftarrow$	

Граф потока управления



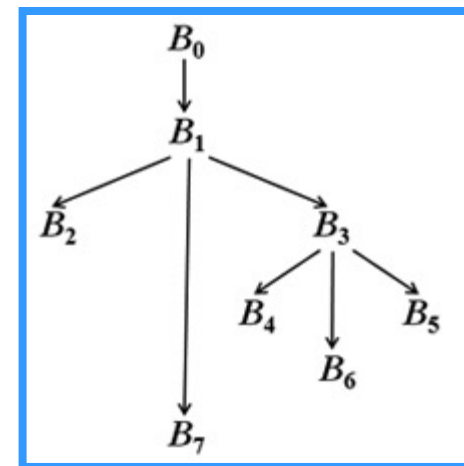
Дерево доминаторов



6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

- Корнем обрабатываемой части дерева доминаторов является блок B_0 . Поэтому «**Основной алгоритм**», обнулив счетчики и опустошив стеки для переменных из множества $Globals = \{a, b, c, d, i\}$, сделает вызов ***Rename(B_0)***.
- Порядок обработки базовых блоков определяется деревом доминаторов (обход в глубину):
 - первым обрабатывается блок B_0 .
 - во время обработки B_0 будет вызван ***Rename(B_1)***,
 - во время обработки B_1 будут вызваны ***Rename(B_2)***, ***Rename(B_7)*** и ***Rename(B_3)***,
 - во время обработки B_3 будет вызван ***Rename(B_4)***, ***Rename(B_6)***, ***Rename(B_5)***



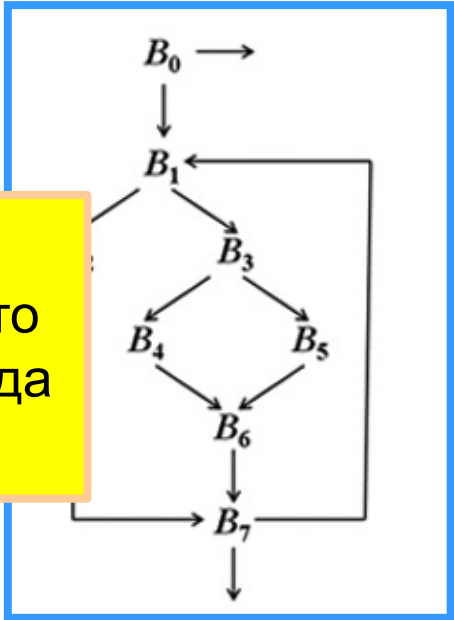
6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

- Корнем обрабатываемой части дерева доминаторов является блок B_0 . Поэтому «**Основной алгоритм**», обнулив счетчики и опустошив стеки для переменных из множества $Globals = \{a, b, c, d, i\}$, сделает вызов $Rename(B_0)$.

Вход в B_0	a	b	c	d	i
Счетчики	1	1	1	1	0
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	

```
B0: i ← 1;
```



Счетчики для переменных a, b, c, d в этом примере сразу имеют значение 1, а не 0, так как считается, что этим переменным были присвоены значения до входа в блок B_0

- Заполнение параметров φ -функций в B_1
- Вызов $Rename(B_1)$

```
B0: i0 ← 1;
```

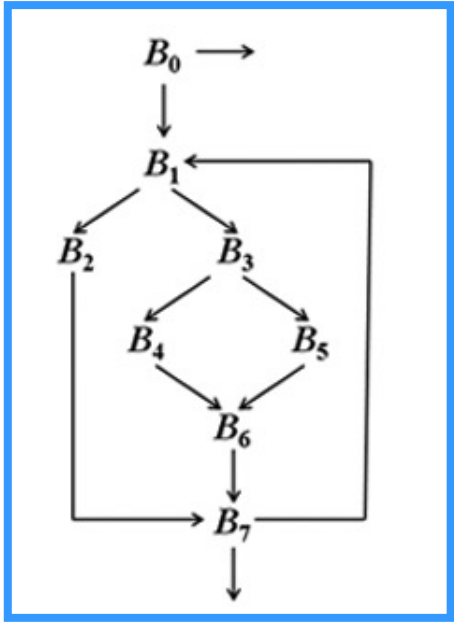
6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

- Корнем обрабатываемой части дерева доминаторов является блок B_0 . Поэтому «**Основной алгоритм**», обнулив счетчики и опустошив стеки для переменных из множества $Globals = \{a, b, c, d, i\}$, сделает вызов $Rename(B_0)$.

Вход в B_0	a	b	c	d	i
Счетчики	1	1	1	1	0
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	

$B_0:$ $i \leftarrow 1;$



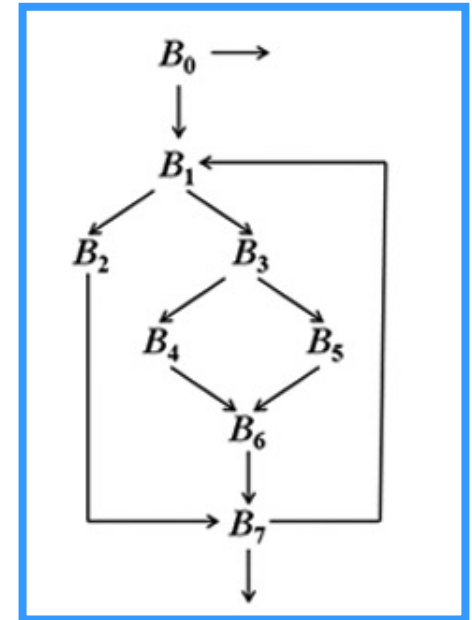
- $Rename(B_0)$:
 - Вызов $NewName(i) \rightarrow$ возвращает имя i_0
 - Замена $i \leftarrow 1;$ на $i_0 \leftarrow 1;$
 - Занесение i_0 в стек для i
 - Увеличение счетчика для i на 1
 - Заполнение параметров φ -функций в B_1
 - Вызов $Rename(B_1)$

$B_0:$ $i_0 \leftarrow 1;$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₀	$i_0 \leftarrow 1$	B₅	$c \leftarrow$
B₁	$a \leftarrow \varphi(a_0, a)$ $b \leftarrow \varphi(b_0, b)$ $c \leftarrow \varphi(c_0, c)$ $d \leftarrow \varphi(d_0, d)$ $i \leftarrow \varphi(i_0, i)$ $a \leftarrow$ $b \leftarrow$	B₇	$a \leftarrow \varphi(a, a)$ $b \leftarrow \varphi(b, b)$ $c \leftarrow \varphi(c, c)$ $d \leftarrow \varphi(d, d)$ $y \leftarrow +, a, b$ $z \leftarrow +, c, d$ $i \leftarrow +, i, 1$
B₂	$b \leftarrow$ $c \leftarrow$ $d \leftarrow$	B₃	$a \leftarrow$ $d \leftarrow$
B₄	$d \leftarrow$	B₆	$c \leftarrow \varphi(c, c)$ $d \leftarrow \varphi(d, d)$ $b \leftarrow$



<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
1	1	1	1	1
a₀	b₀	c₀	d₀	i₀

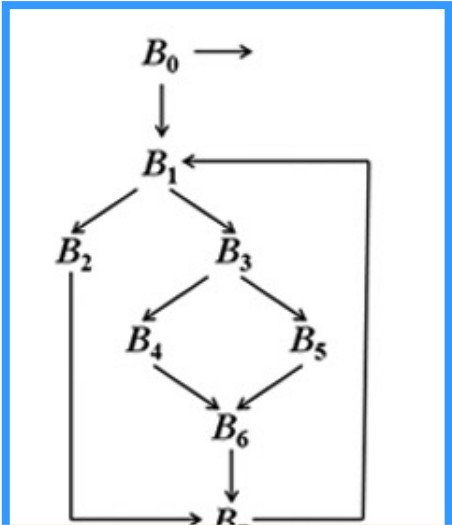
6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

- Корнем обрабатываемой части дерева доминаторов является блок B_0 . Поэтому «Основной алгоритм», обнулив счетчики и опустошив стеки для переменных из множества $Globals = \{a, b, c, d, i\}$, сделает вызов $Rename(B_0)$.

Вход в B_0	a	b	c	d	i
Счетчики	1	1	1	1	0
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	

$B_0: \quad i \leftarrow 1;$



- $Rename(B_0)$:
 - Вызов $NewName(i) \rightarrow$ возвращает имя i_0
 - Замена $i \leftarrow 1;$ на $i_0 \leftarrow 1;$
 - Занесение i_0 в стек для i
 - Увеличение счетчика для i на 1
 - Заполнение параметров φ -функций в B_1
 - Вызов $Rename(B_1)$

$B_1 = Succ(B_0)$ по ГПУ

$B_0: \quad i_0 \leftarrow 1;$

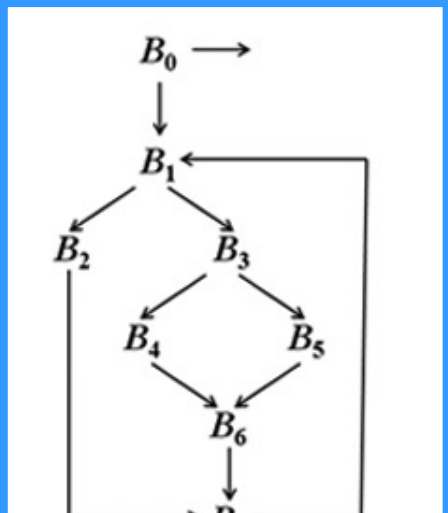
6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

- Корнем обрабатываемой части дерева доминаторов является блок B_0 . Поэтому «**Основной алгоритм**», обнулив счетчики и опустошив стеки для переменных из множества $Globals = \{a, b, c, d, i\}$, сделает вызов $Rename(B_0)$.

Вход в B_0	a	b	c	d	i
Счетчики	1	1	1	1	0
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	

$B_0: \quad i \leftarrow 1;$



- $Rename(B_0)$:
 - Вызов $NewName(i) \rightarrow$ возвращает имя i_0
 - Замена $i \leftarrow 1;$ на $i_0 \leftarrow 1;$
 - Занесение i_0 в стек для i
 - Увеличение счетчика для i на 1
 - Заполнение параметров φ -функций в B_1
 - Вызов $Rename(B_1)$

$B_1 = Succ(B_0)$ по ГПУ

$B_1 = Succ(B_0)$ по дереву доминаторов

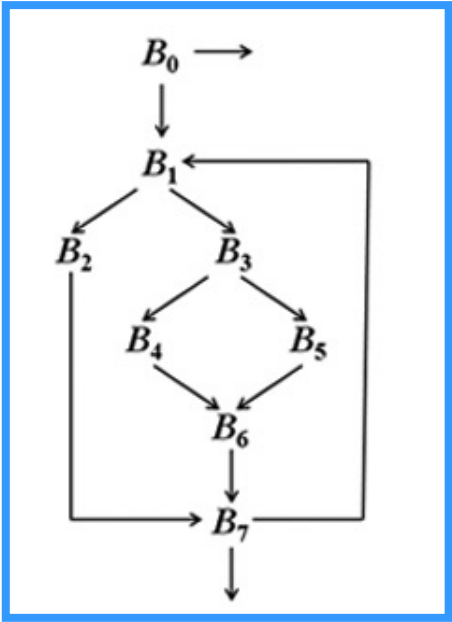
6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

◇ Корнем обрабатываемой части дерева доминаторов является блок B_0 . Поэтому «*Основной алгоритм*», обнулив счетчики и опустошив стеки для переменных из множества $Globals = \{a, b, c, d, i\}$, сделает вызов $Rename(B_0)$.

Вход в B_0	a	b	c	d	i
Счетчики	1	1	1	1	0
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	

$B_0:$ $i \leftarrow 1;$



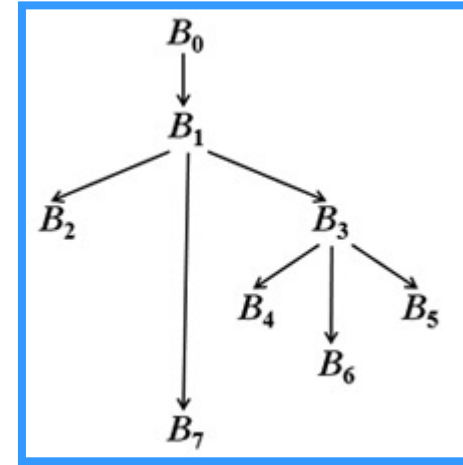
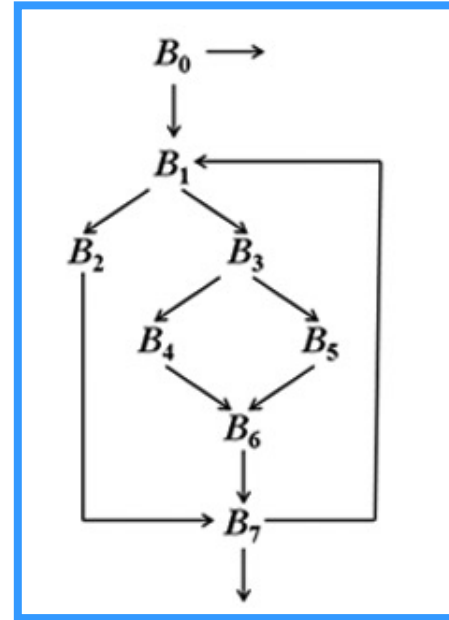
- ◇ $Rename(B_0)$:
 - ◇ Вызов $NewName(i) \rightarrow$ возвращает имя i_0
 - Замена $i \leftarrow 1;$ на $i_0 \leftarrow 1;$
 - Занесение i_0 в стек для i
 - Увеличение счетчика для i на 1
 - ◇ Заполнение параметров φ -функций в B_1
 - ◇ Вызов $Rename(B_1)$

$B_0:$ $i_0 \leftarrow 1;$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_1	a	b	c	d	i
Счетчики	1	1	1	1	1
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	i_0



Rename(B_1):

1. Переименование φ -функций

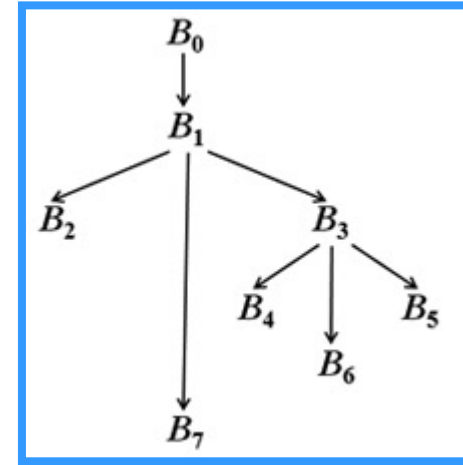
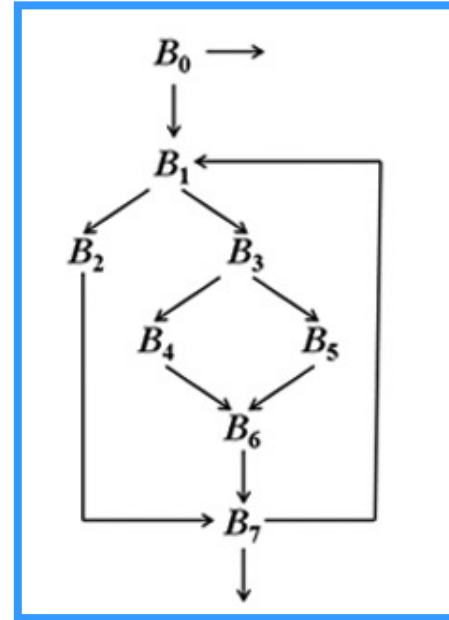
```

 $B_1$    $a \leftarrow \varphi( a_0 , a )$ 
         $b \leftarrow \varphi( b_0 , b )$ 
         $c \leftarrow \varphi( c_0 , c )$ 
         $d \leftarrow \varphi( d_0 , d )$ 
         $i \leftarrow \varphi( i_0 , i )$ 
         $a \leftarrow$ 
         $b \leftarrow$ 
  
```

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₁	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	1	1	1	1	1
Стеки (↓)	a₀	b₀	c₀	d₀	i₀



Rename(B₁):

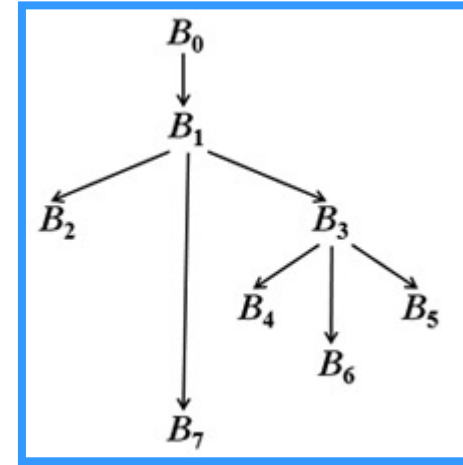
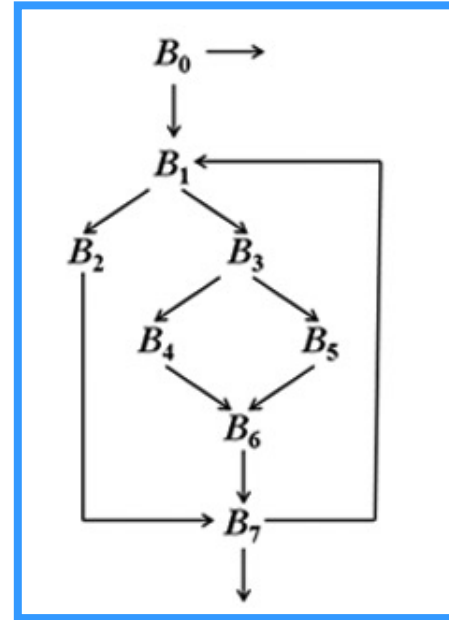
1. Переименование ϕ -функций

B₁	a	←	ϕ(a₀ , a)
	b	←	ϕ(b₀ , b)
	c	←	ϕ(c₀ , c)
	d	←	ϕ(d₀ , d)
	i	←	ϕ(i₀ , i)
	a	←	
	b	←	

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₁	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	2	1	1	1	1
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁				



Rename(B₁):

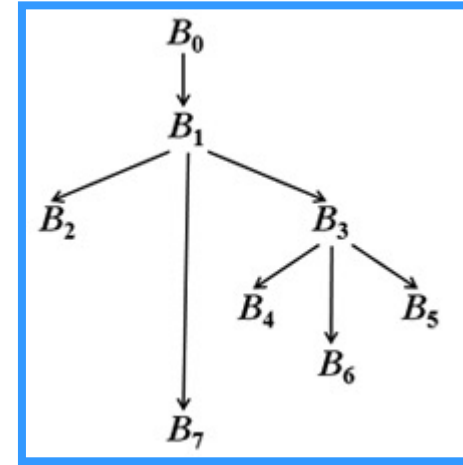
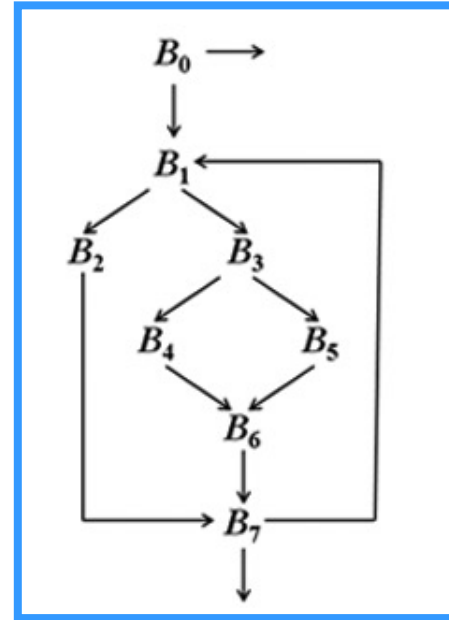
1. Переименование ϕ -функций

B ₁	a ₁ ← ϕ (a ₀ , a)
	b ← ϕ(b₀ , b)
	c ← ϕ (c ₀ , c)
	d ← ϕ (d ₀ , d)
	i ← ϕ (i ₀ , i)
	a ←
	b ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₁	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	2	2	1	1	1
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁			



Rename(B₁):

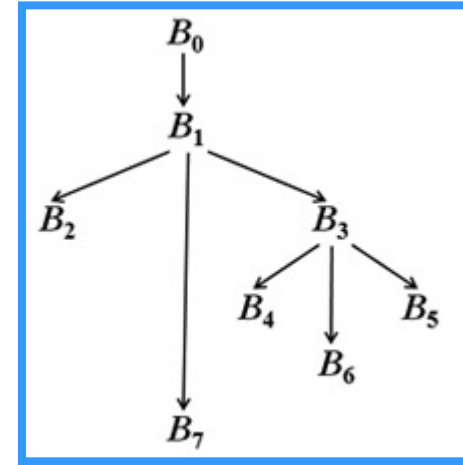
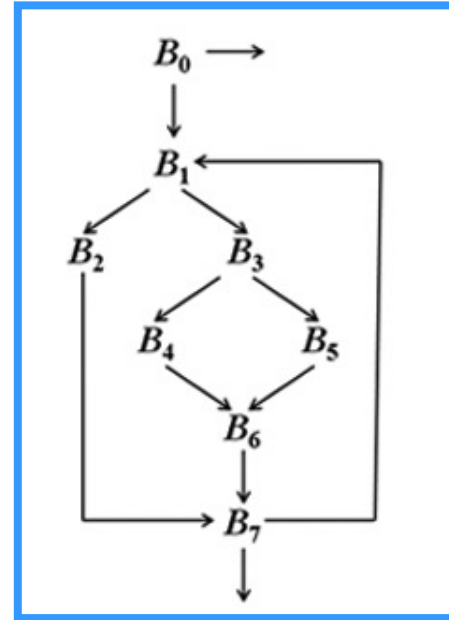
1. Переименование ϕ -функций

```
B1  a1 ←  $\phi$ ( a0 , a )
      b1 ←  $\phi$ ( b0 , b )
      c ←  $\phi$ ( c0 , c )
      d ←  $\phi$ ( d0 , d )
      i ←  $\phi$ ( i0 , i )
      a ←
      b ←
```

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₁	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	2	2	2	1	1
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁		



Rename(B₁):

1. Переименование ϕ -функций

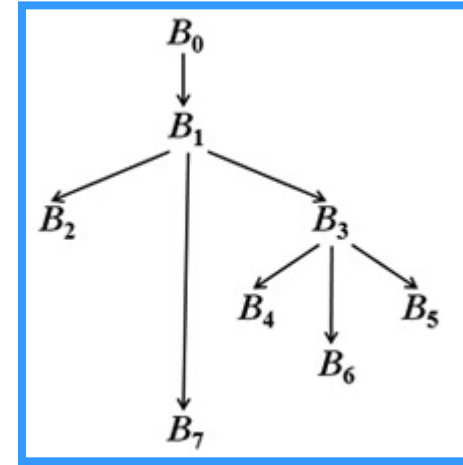
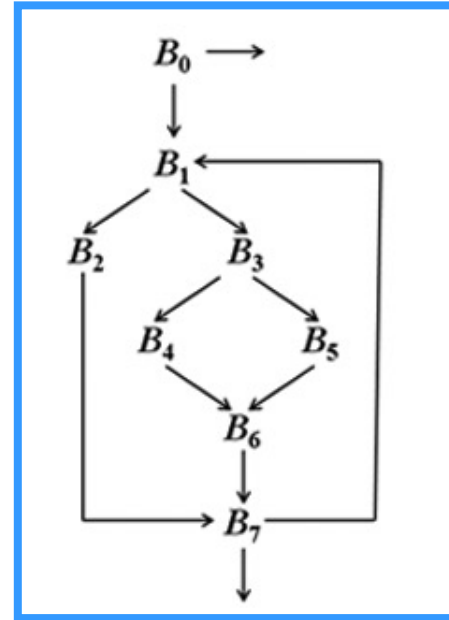
```

B1  a1 ← φ( a0 , a )
      b1 ← φ( b0 , b )
      c1 ← φ( c0 , c )
      d ← φ( d0 , d )
      i ← φ( i0 , i )
      a ←
      b ←
  
```

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₁	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	2	2	2	2	1
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	



Rename(B₁):

1. Переименование ϕ -функций

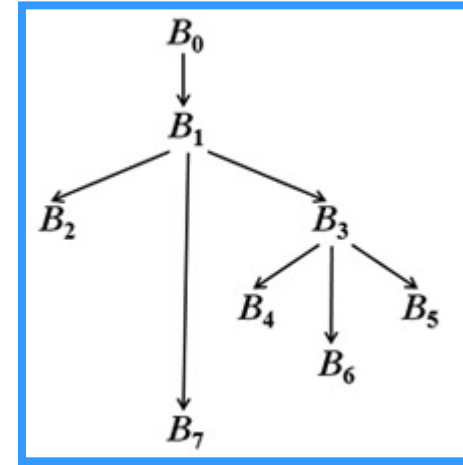
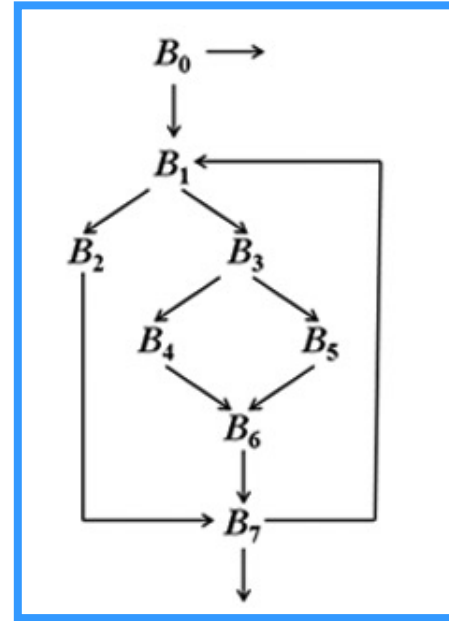
```

B1  a1 ← φ( a0 , a )
      b1 ← φ( b0 , b )
      c1 ← φ( c0 , c )
      d1 ← φ( d0 , d )
      i ← φ( i0 , i )
      a ←
      b ←
  
```


6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₁	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	2	2	2	2	2
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁



Rename(B₁):

1. Переименование ϕ -функций
2. Переименование инструкций

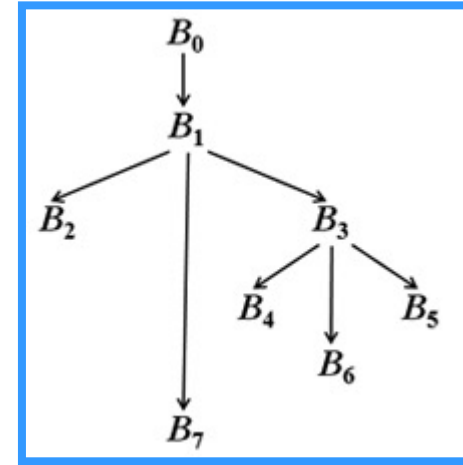
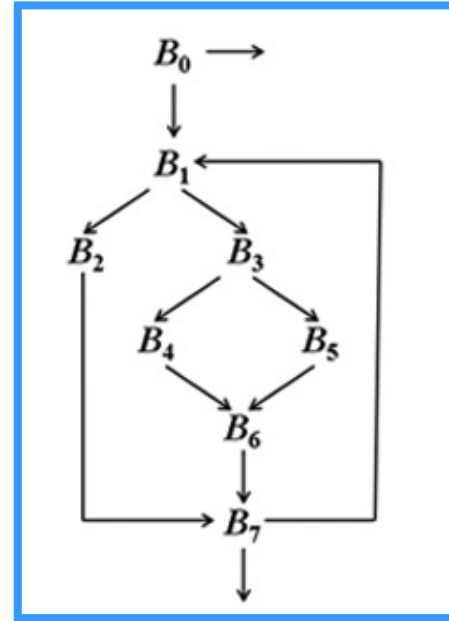
```

B1  a1 ← φ( a0 , a )
      b1 ← φ( b0 , b )
      c1 ← φ( c0 , c )
      d1 ← φ( d0 , d )
      i1 ← φ( i0 , i )
      a ←
      b ←
  
```

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₁	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	2	2	2	2	2
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁



Rename(B₁):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

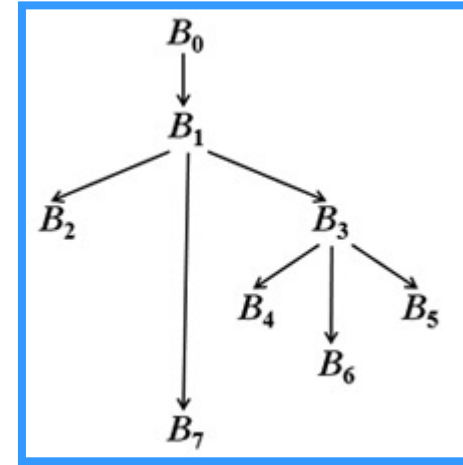
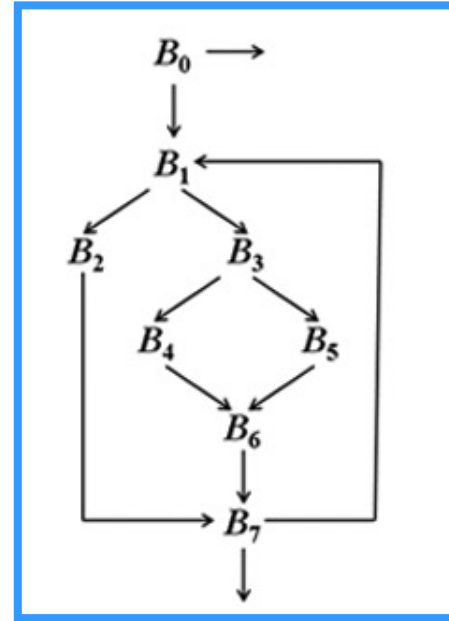
```

B1  a1 ← φ( a0 , a )
      b1 ← φ( b0 , b )
      c1 ← φ( c0 , c )
      d1 ← φ( d0 , d )
      i1 ← φ( i0 , i )
      a ←
      b ←
  
```

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₁	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	3	2	2	2	2
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂				



Rename(B₁):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

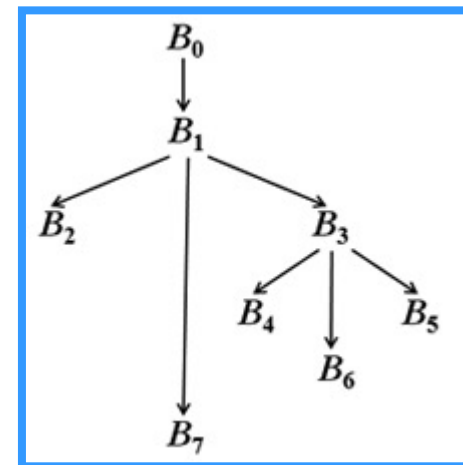
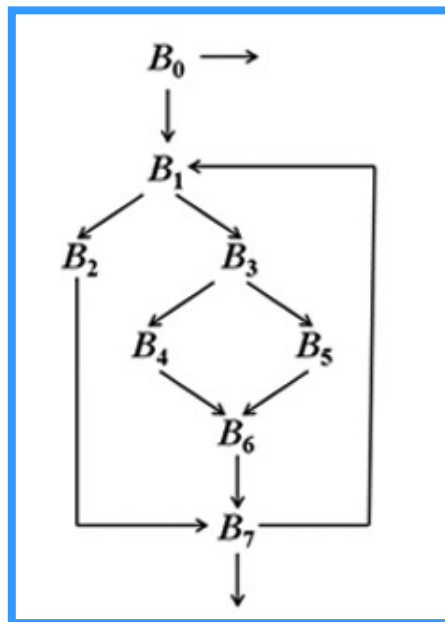
```

B1  a1 ← φ( a0 , a )
      b1 ← φ( b0 , b )
      c1 ← φ( c0 , c )
      d1 ← φ( d0 , d )
      i1 ← φ( i0 , i )
      a2 ←
      b ←
  
```

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₁	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	3	3	2	2	2
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂			



Rename(B₁):

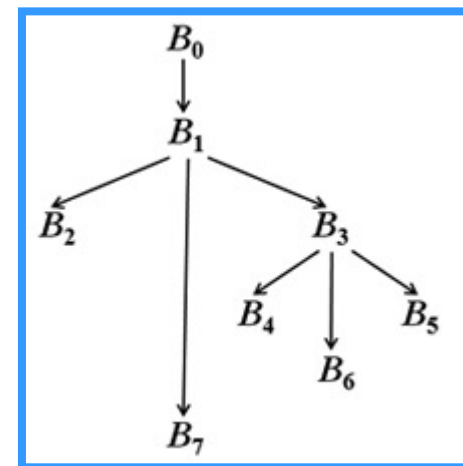
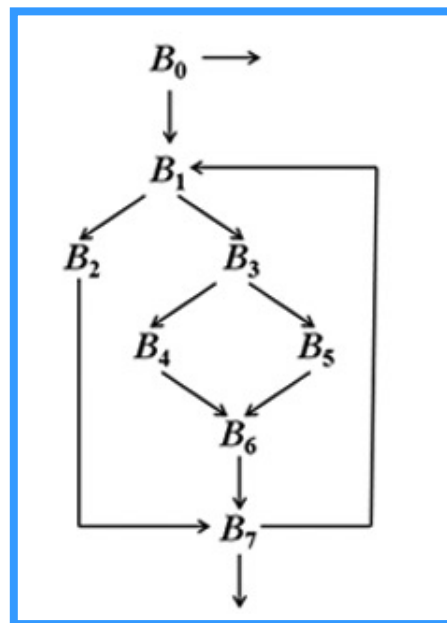
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG (в потомках B₁ нет ϕ -функций)

B₁ a₁ ← ϕ (a₀ , a)
 b₁ ← ϕ (b₀ , b)
 c₁ ← ϕ (c₀ , c)
 d₁ ← ϕ (d₀ , d)
 i₁ ← ϕ (i₀ , i)
 a₂ ←
 b₂ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₁	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	3	3	2	2	2
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂			



Rename(B₁):

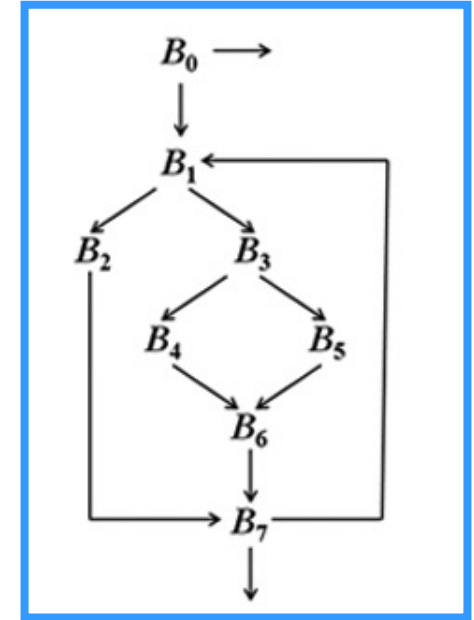
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG (в потомках B₁ нет ϕ -функций)
4. **Rename**(B₂)

B₁ a₁ ← ϕ (a₀ , a)
 b₁ ← ϕ (b₀ , b)
 c₁ ← ϕ (c₀ , c)
 d₁ ← ϕ (d₀ , d)
 i₁ ← ϕ (i₀ , i)
 a₂ ←
 b₂ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₀	i₀ ← 1	B₅	c ←
B₁	a₁ ← φ(a₀ , a) b₁ ← φ(b₀ , b) c₁ ← φ(c₀ , c) d₁ ← φ(d₀ , d) i₁ ← φ(i₀ , i) a₂ ← b₂ ←	B₇	a ← φ(a , a) b ← φ(b , b) c ← φ(c , c) d ← φ(d , d) y ← +, a , b z ← +, c , d i ← +, i , 1
B₂	b ← c ← d ←	B₃	a ← d ←
B₄	d ←	B₆	c ← φ(c , c) d ← φ(d , d) b ←



	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
	3	3	2	2	2
a₀	b₀	c₀	d₀	i₀	
a₁	b₁	c₁	d₁	i₁	
a₂	b₂				

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_2

a *b* *c* *d* *i*

Счетчики

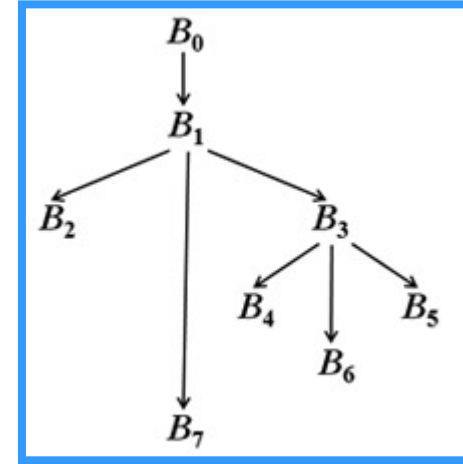
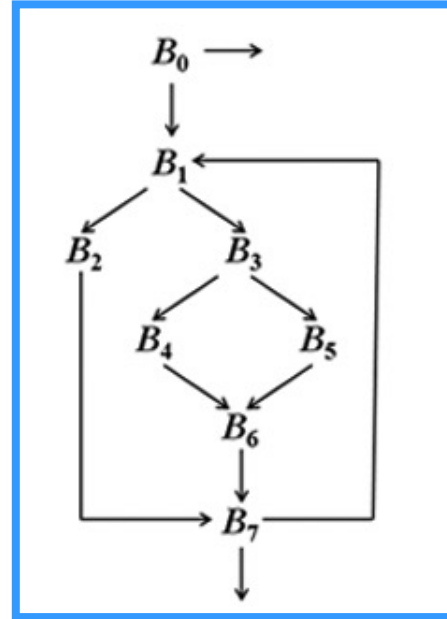
3 3 2 2 2

Стеки ($\downarrow$)

a_0 **b_0** **c_0** **d_0** **i_0**

a_1 **b_1** **c_1** **d_1** **i_1**

a_2 **b_2**



Rename(B_2):

1. Переименование ϕ -функций

B_2 **b** $\leftarrow$

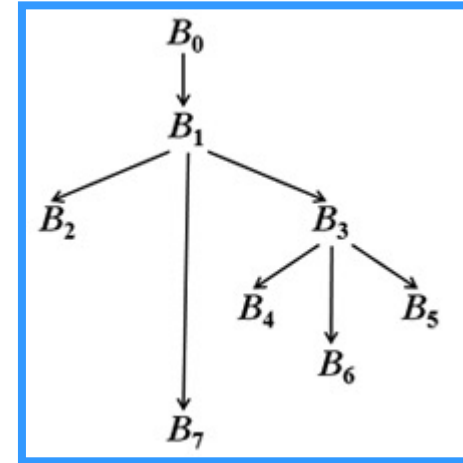
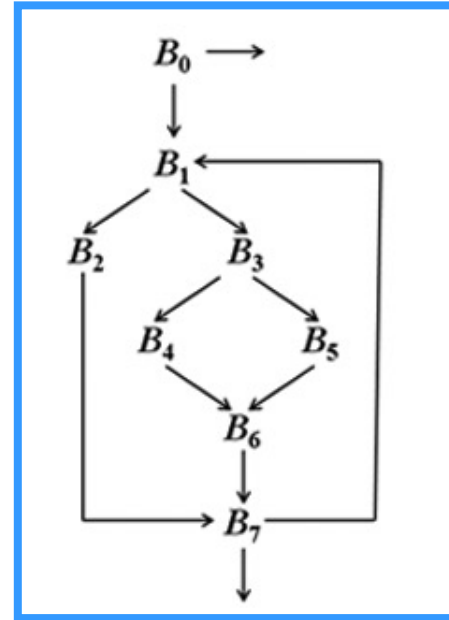
c $\leftarrow$

d $\leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₂	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	3	3	2	2	2
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂			



Rename(B₂):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₂	b ←
	c ←
	d ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₂

a *b* *c* *d* *i*

Счетчики

3 4 2 2 2

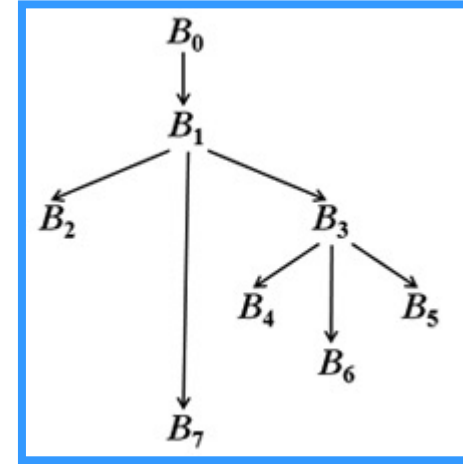
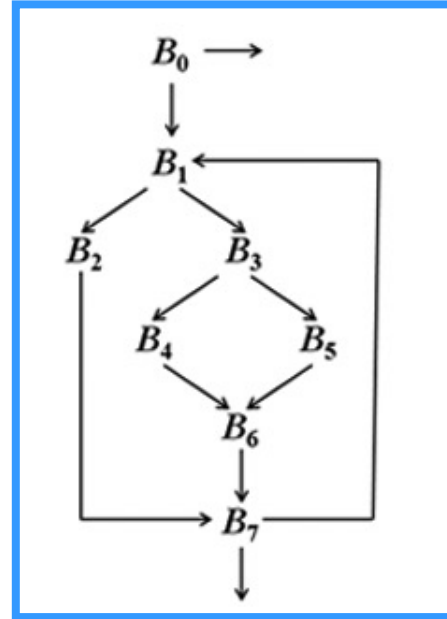
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₁** **d₁** **i₁**

a₂ **b₂** **c₁** **d₁** **i₁**



Rename(B₂):

1. Переименование φ-функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₂ **b₃ ←**

c ←

d ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₂

a *b* *c* *d* *i*

Счетчики

3 4 3 2 2

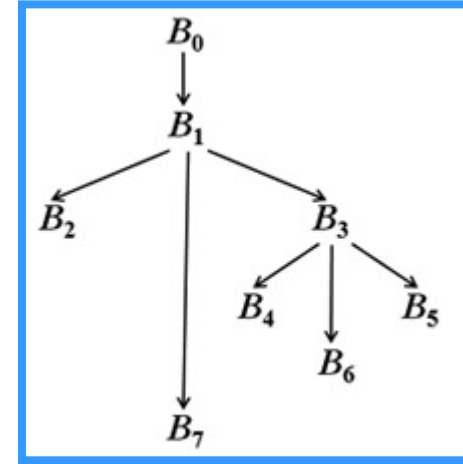
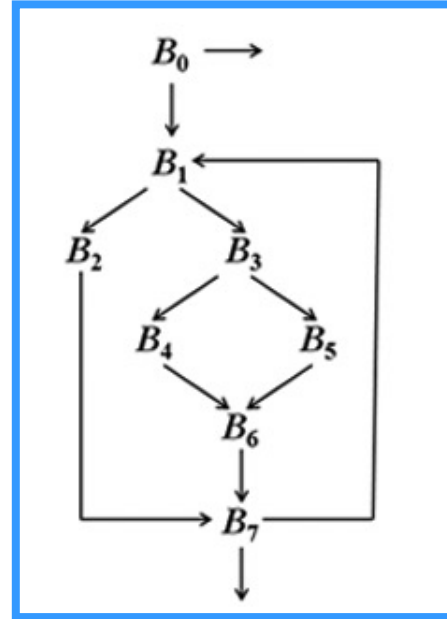
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₂**

b₃



Rename(B₂):

1. Переименование φ-функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₂ **b₃ ←**

c₂ ←

d ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₂

a *b* *c* *d* *i*

Счетчики

3 4 3 3 2

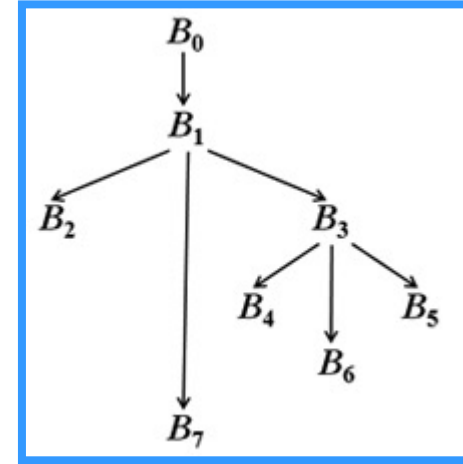
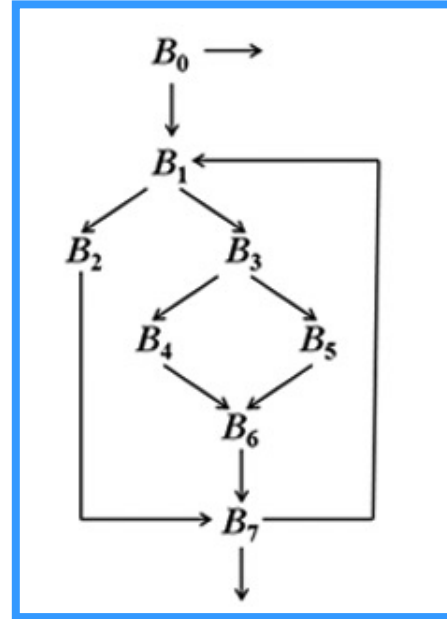
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₂** **d₂**

b₃



Rename(B₂):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₂ **b₃ ←**

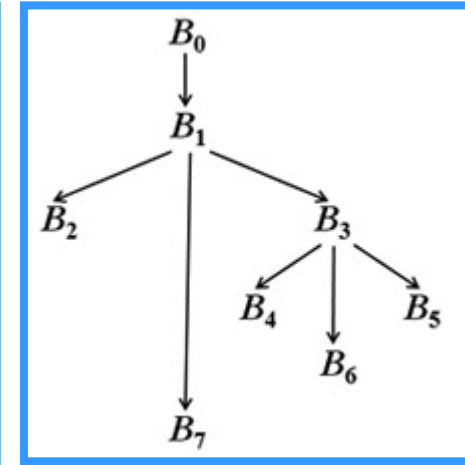
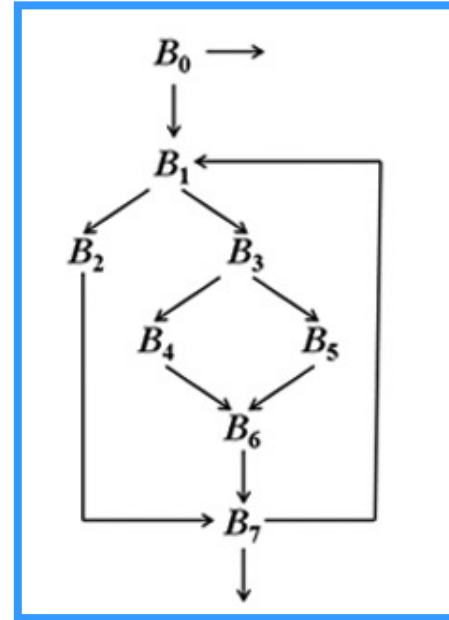
c₂ ←

d₂ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₂	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	3	4	3	3	2
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂	c₂	d₂	
		b₃			



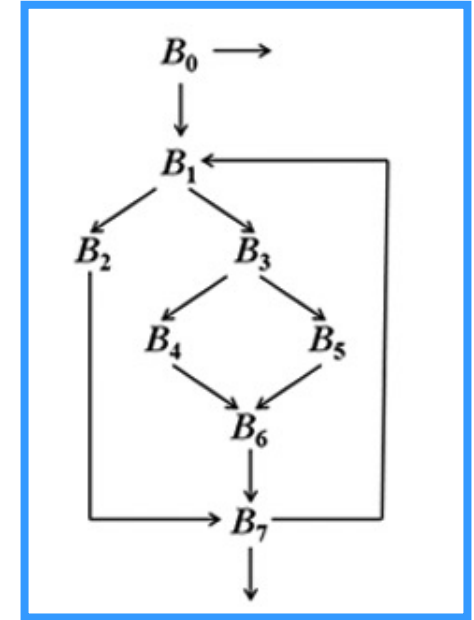
Rename(B₂):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG

B₂ b₃ ←
 c₂ ←
 d₂ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных



a *b* *c* *d* *i*

3	4	3	3	2
a ₀	b ₀	c ₀	d ₀	i ₀
a ₁	b ₁	c ₁	d ₁	i ₁
a ₂	b ₂	c ₂	d ₂	
	b ₃			

B ₀	<i>i</i> ₀ ← 1	B ₅	<i>c</i> ←
B ₁	<i>a</i> ₁ ← φ(<i>a</i> ₀ , <i>a</i>) <i>b</i> ₁ ← φ(<i>b</i> ₀ , <i>b</i>) <i>c</i> ₁ ← φ(<i>c</i> ₀ , <i>c</i>) <i>d</i> ₁ ← φ(<i>d</i> ₀ , <i>d</i>) <i>i</i> ₁ ← φ(<i>i</i> ₀ , <i>i</i>) <i>a</i> ₂ ← <i>b</i> ₂ ←	B ₇	<i>a</i> ← φ(a ₂ , <i>a</i>) <i>b</i> ← φ(b ₃ , <i>b</i>) <i>c</i> ← φ(c ₂ , <i>c</i>) <i>d</i> ← φ(d ₂ , <i>d</i>) <i>y</i> ← +, <i>a</i> , <i>b</i> <i>z</i> ← +, <i>c</i> , <i>d</i> <i>i</i> ← +, <i>i</i> , 1
B ₂	b ₃ ← c ₂ ← d ₂ ←	B ₃	<i>a</i> ← <i>d</i> ←
B ₄	<i>d</i> ←	B ₆	<i>c</i> ← φ(<i>c</i> , <i>c</i>) <i>d</i> ← φ(<i>d</i> , <i>d</i>) <i>b</i> ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₂

a *b* *c* *d* *i*

Счетчики

3 4 3 3 2

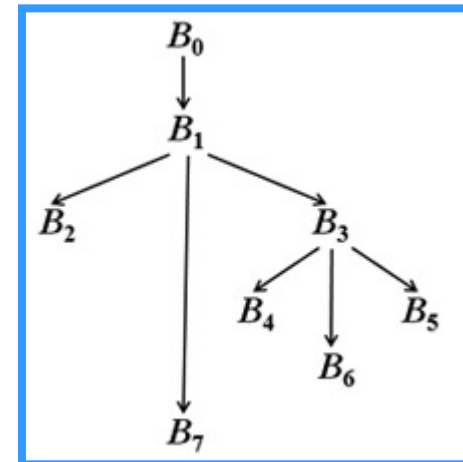
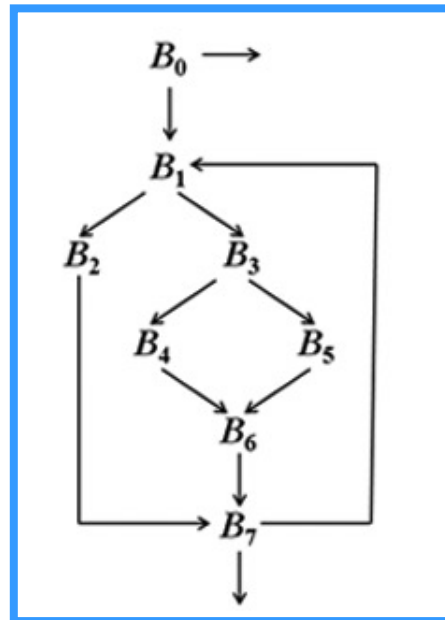
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₂** **d₂**

b₃



Rename(B₂):

1. Переименование φ-функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров φ-функций в потомках в CFG
4. Нет потомков в дереве доминаторов

B₂ **b₃ ←**

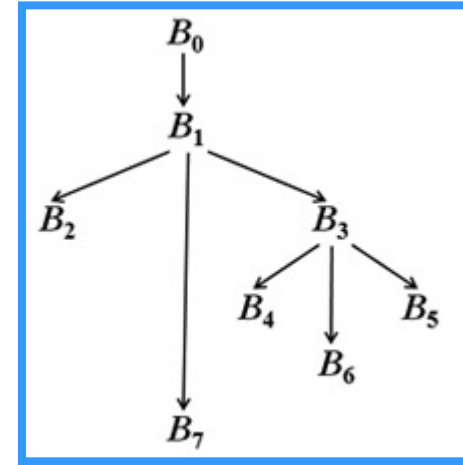
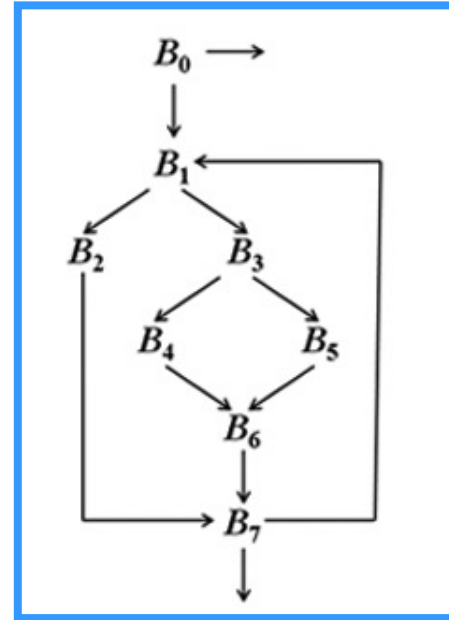
c₂ ←

d₂ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₂	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	3	4	3	3	2
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂	c₂	d₂	
		b₃			



Rename(B₂):

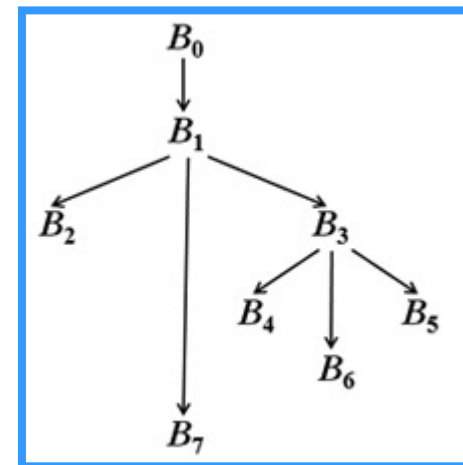
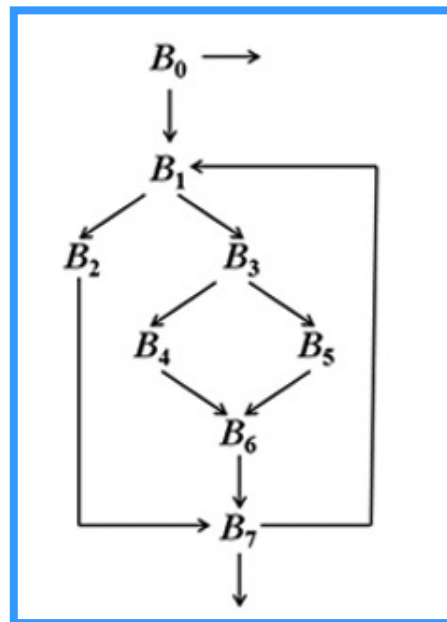
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG
4. Нет потомков в дереве доминаторов
5. Очистка стека

B₂ **b₃ ←**
 c₂ ←
 d₂ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₂	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	3	4	3	3	2
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂	c₂	d₂	
		b₃			



Rename(B₂):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG
4. Нет потомков в дереве доминаторов
5. Очистка стека

B₂	b₃	←
	c₂	←
	d₂	←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₂

a *b* *c* *d* *i*

Счетчики

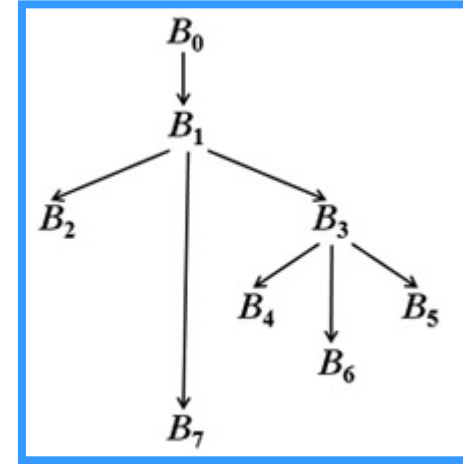
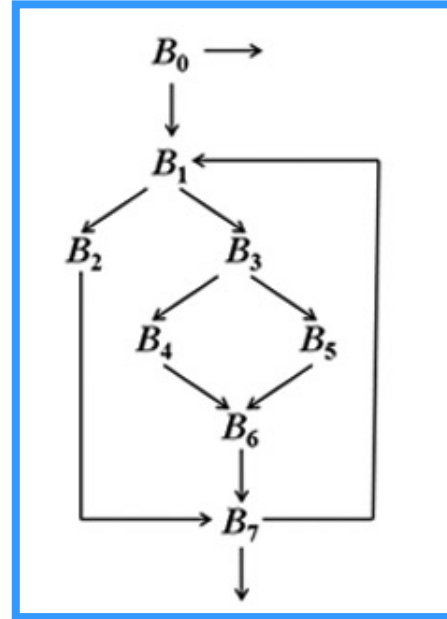
3 4 3 3 2

Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂**



Rename(B₂):

1. Переименование φ-функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров φ-функций в потомках в CFG
4. Нет потомков в дереве доминаторов
5. Очистка стека. **return** // -> B₁

B₂ **b₃ ←**

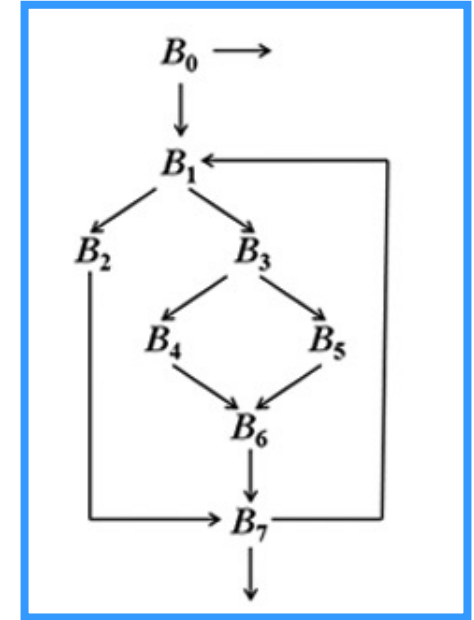
c₂ ←

d₂ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₀	i₀ ← 1	B₅	c ←
B₁	a₁ ← φ(a₀ , a) b₁ ← φ(b₀ , b) c₁ ← φ(c₀ , c) d₁ ← φ(d₀ , d) i₁ ← φ(i₀ , i) a₂ ← b₂ ←	B₇	a ← φ(a₂ , a) b ← φ(b₃ , b) c ← φ(c₂ , c) d ← φ(d₂ , d) y ← +, a , b z ← +, c , d i ← +, i , 1
B₂	b₃ ← c₂ ← d₂ ←	B₃	a ← d ←
B₄	d ←	B₆	c ← φ(c , c) d ← φ(d , d) b ←



	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
	3	4	3	3	2
a₀	b₀	c₀	d₀	i₀	
a₁	b₁	c₁	d₁	i₁	
a₂	b₂				

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_1

a b c d i

Счетчики

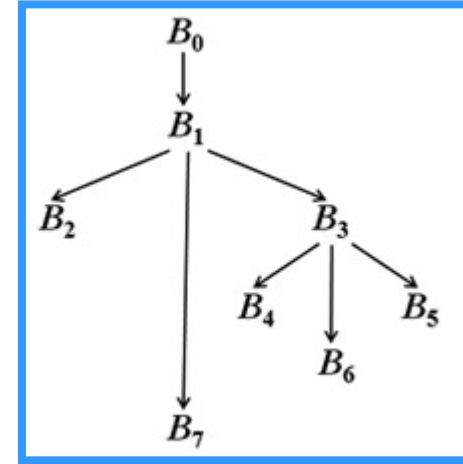
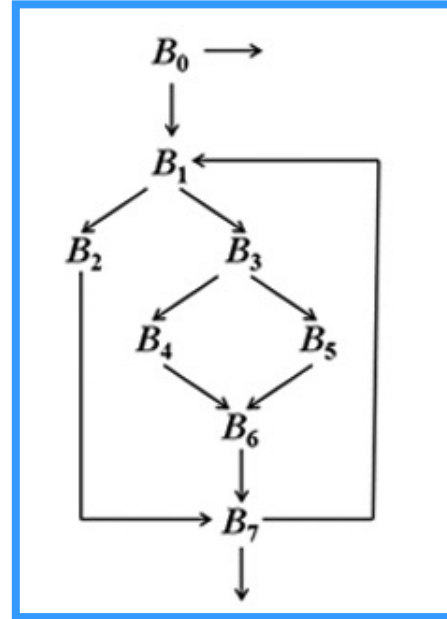
3 4 3 3 2

Стеки ($\downarrow$)

a_0 b_0 c_0 d_0 i_0

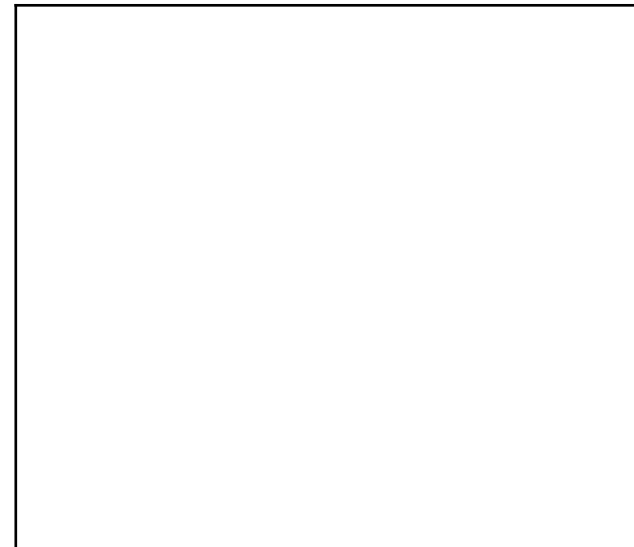
a_1 b_1 c_1 d_1 i_1

a_2 b_2



Rename(B_1):

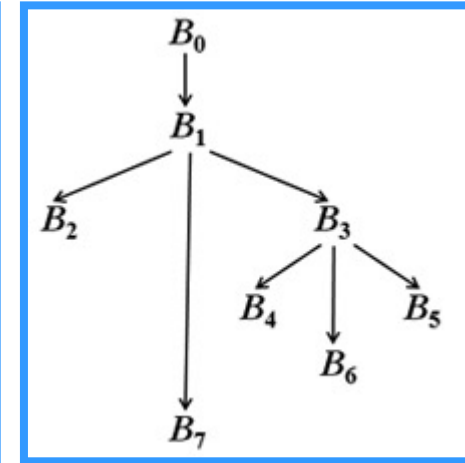
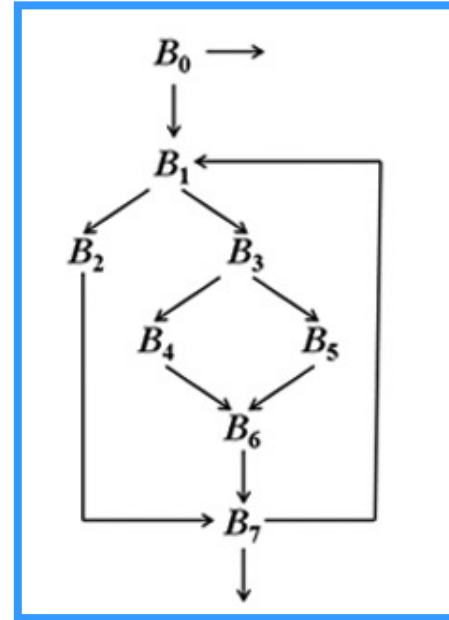
Вызов ***Rename***(B_7) так как B_7 следующий потомок в дереве доминаторов



6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₇	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	3	4	3	3	2
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂			



Rename(B₇):

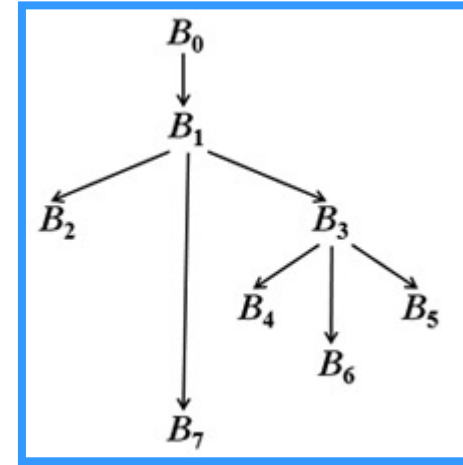
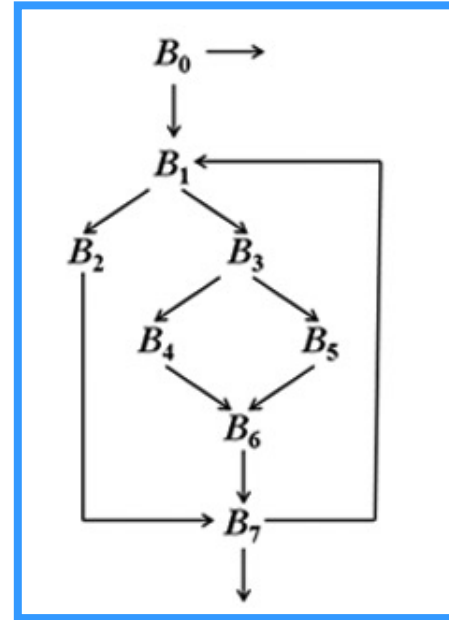
1. Переименование ϕ -функций

B₇	a	←	φ (a₂ , a)
	b	←	φ (b₃ , b)
	c	←	φ (c₂ , c)
	d	←	φ (d₂ , d)
	y	←	+ , a , b
	z	←	+ , c , d
	i	←	+ , i , 1

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₇	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	4	4	3	3	2
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂			
	a₃				



Rename(B₇):

1. Переименование ϕ -функций

```

B7  a3 ←  $\phi$ ( a2 , a )
      b ←  $\phi$ ( b3 , b )
      c ←  $\phi$ ( c2 , c )
      d ←  $\phi$ ( d2 , d )
      y ←      +, a , b
      z ←      +, c , d
      i ←      +, i , 1
  
```

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₇

a *b* *c* *d* *i*

Счетчики

4 5 3 3 2

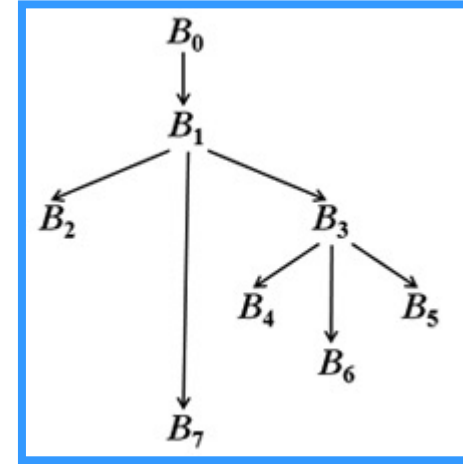
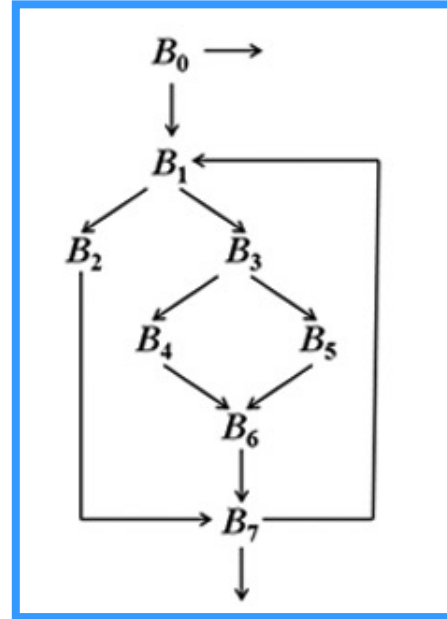
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂**

a₃ **b₄**



Rename(B₇):

1. Переименование φ-функций

B₇ **a₃ ← φ (a₂ , a)**

b₄ ← φ (b₃ , b)

c ← φ (c₂ , c)

d ← φ (d₂ , d)

y ← + , a , b

z ← + , c , d

i ← + , i , 1

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₇

a *b* *c* *d* *i*

Счетчики

4 5 4 3 2

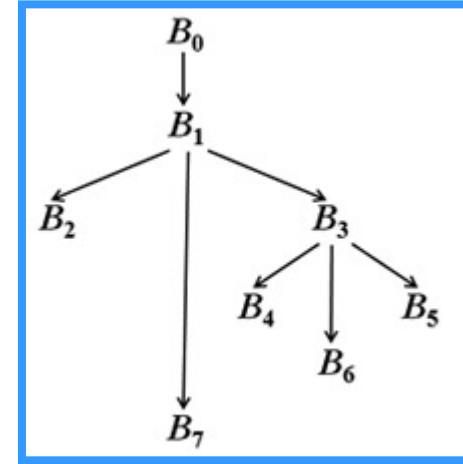
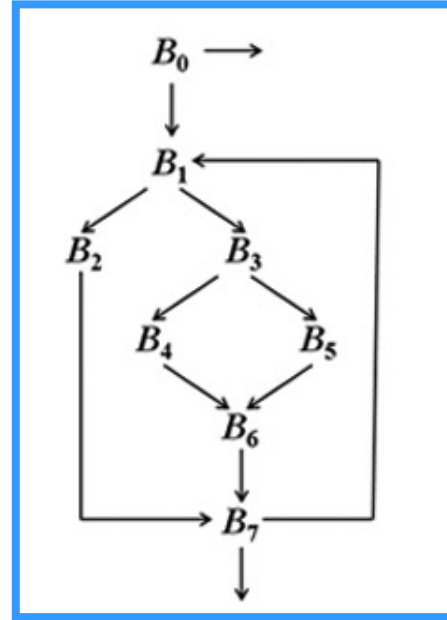
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₃**

a₃ **b₄**



Rename(B₇):

1. Переименование φ-функций

B₇ **a₃ ← φ (a₂ , a)**

b₄ ← φ (b₃ , b)

c₃ ← φ (c₂ , c)

d ← φ (d₂ , d)

y ← + , a , b

z ← + , c , d

i ← + , i , 1

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₇

a *b* *c* *d* *i*

Счетчики

4 5 4 4 2

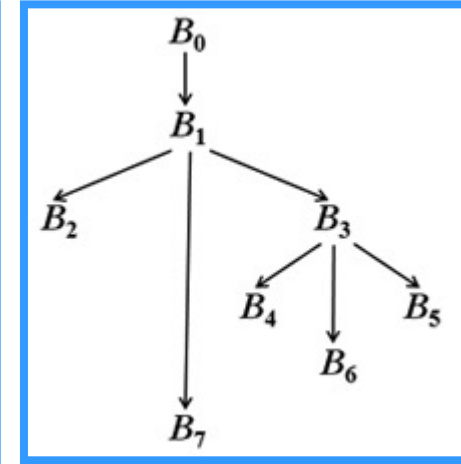
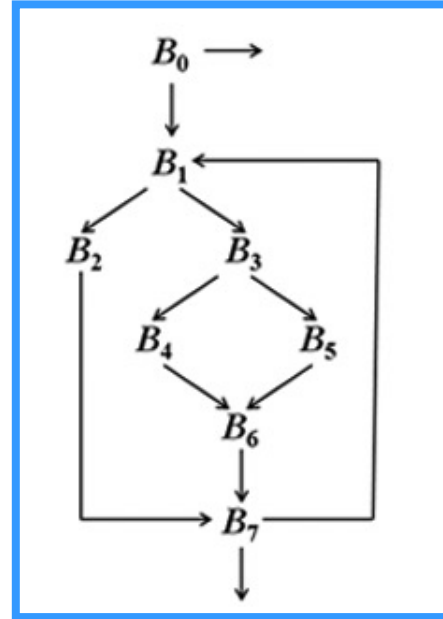
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₃** **d₃**

a₃ **b₄**



Rename(B₇):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₇ **a₃ ← ϕ (a₂ , a)**
 b₄ ← ϕ (b₃ , b)
 c₃ ← ϕ (c₂ , c)
 d₃ ← ϕ (d₂ , d)
 y ← +, a , b
 z ← +, c , d
 i ← +, i , 1

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₇

a *b* *c* *d* *i*

Счетчики

4 5 4 4 2

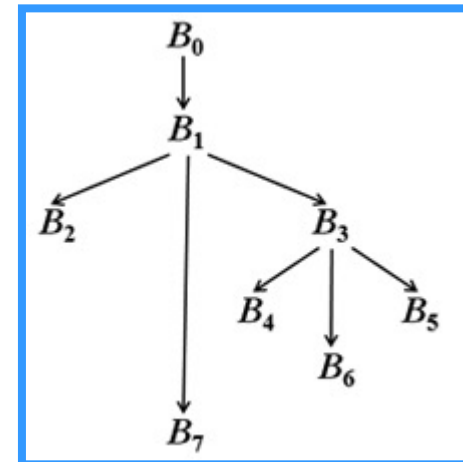
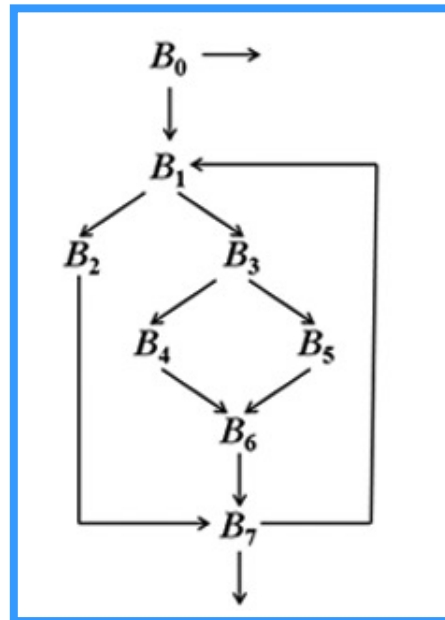
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₃** **d₃**

a₃ **b₄**



Rename(B₇):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₇ **a₃ ← ϕ (a₂ , a)**

b₄ ← ϕ (b₃ , b)

c₃ ← ϕ (c₂ , c)

d₃ ← ϕ (d₂ , d)

y ← +, a , b

z ← +, c , d

i ← +, i , 1

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₇

a *b* *c* *d* *i*

Счетчики

4 5 4 4 2

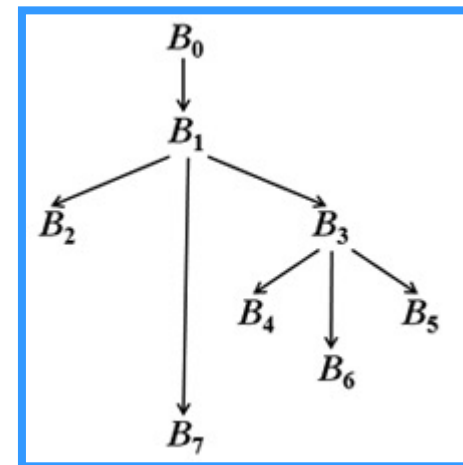
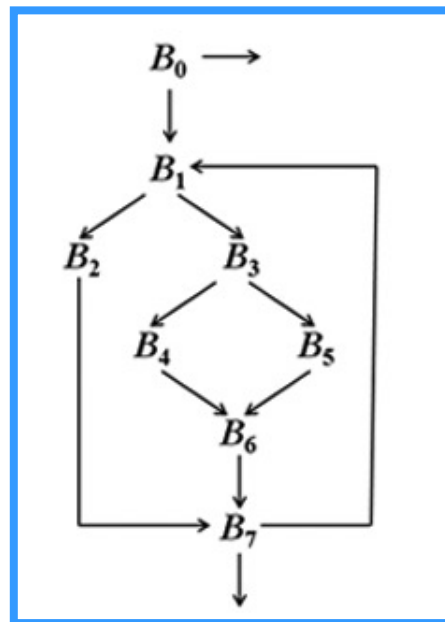
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₃** **d₃**

a₃ **b₄**



Rename(B₇):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₇ a₃ ← ϕ (a₂ , a)

 b₄ ← ϕ (b₃ , b)

 c₃ ← ϕ (c₂ , c)

 d₃ ← ϕ (d₂ , d)

y ← + , a₃ , b₄

 z ← + , c , d

 i ← + , i , 1

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_7

a b c d i

Счетчики

4 5 4 4 2

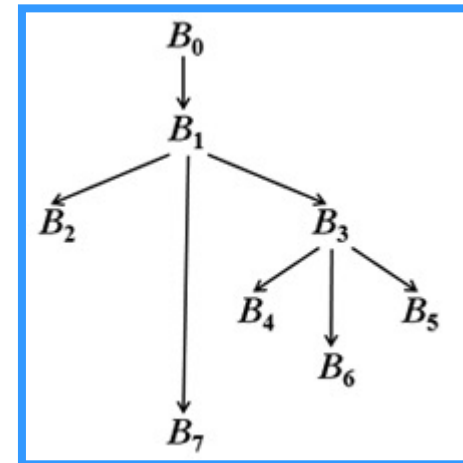
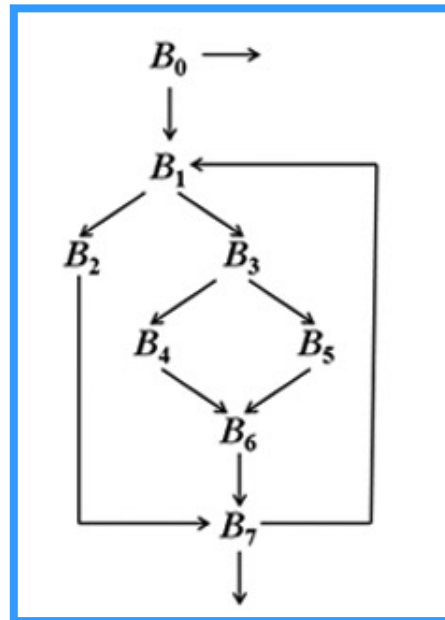
Стеки ($\downarrow$)

a_0 b_0 c_0 d_0 i_0

a_1 b_1 c_1 d_1 i_1

a_2 b_2 c_3 d_3

a_3 b_4



Rename(B_7):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование
результатирующего операнда

Переменная y

не является глобальной

B_7 $a_3 \leftarrow \phi(a_2 , a)$
 $b_4 \leftarrow \phi(b_3 , b)$
 $c_3 \leftarrow \phi(c_2 , c)$
 $d_3 \leftarrow \phi(d_2 , d)$
 $y \leftarrow +, a_3 , b_4$
 $z \leftarrow +, c , d$
 $i \leftarrow +, i , 1$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₇

a *b* *c* *d* *i*

Счетчики

4 5 4 4 2

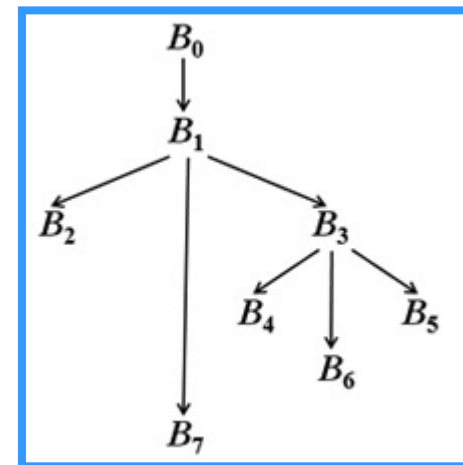
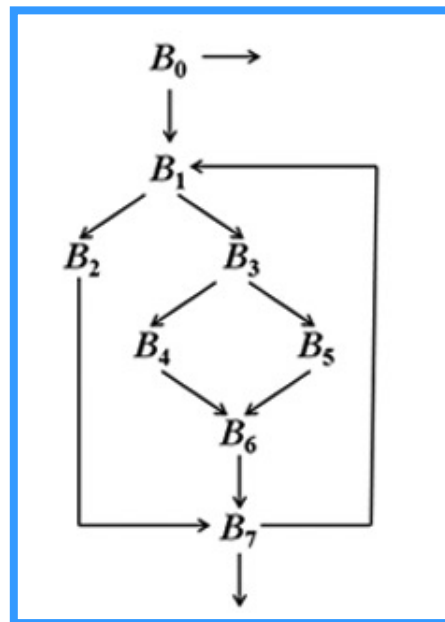
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₃** **d₃**

a₃ **b₄**



Rename(B₇):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₇ a₃ ← ϕ (a₂ , a)

 b₄ ← ϕ (b₃ , b)

 c₃ ← ϕ (c₂ , c)

 d₃ ← ϕ (d₂ , d)

 y ← + , a₃ , b₄

z ← + , c , d

 i ← + , i , 1

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₇

a *b* *c* *d* *i*

Счетчики

4 5 4 4 2

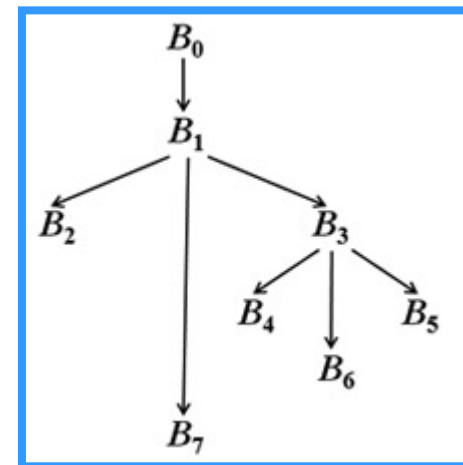
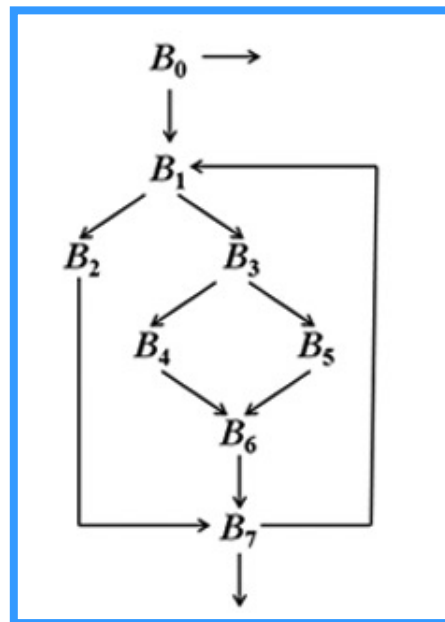
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₃** **d₃**

a₃ **b₄**



Rename(B₇):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₇ a₃ ← ϕ (a₂ , a)

 b₄ ← ϕ (b₃ , b)

 c₃ ← ϕ (c₂ , c)

 d₃ ← ϕ (d₂ , d)

 y ← + , a₃ , b₄

 z ← + , c₃ , d₃

i ← + , i , 1

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₇

a *b* *c* *d* *i*

Счетчики

4 5 4 4 3

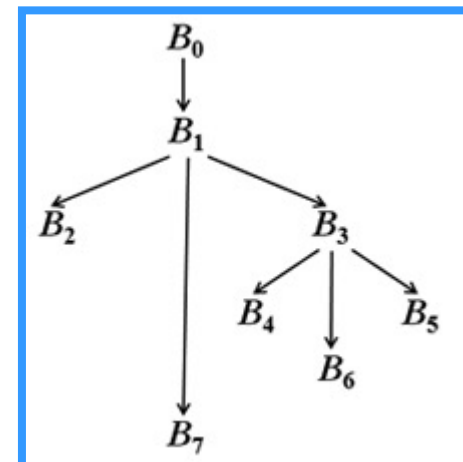
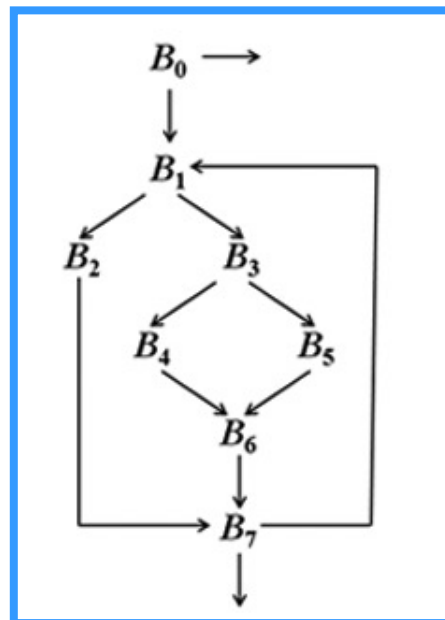
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₃** **d₃** **i₂**

a₃ **b₄**



Rename(B₇):

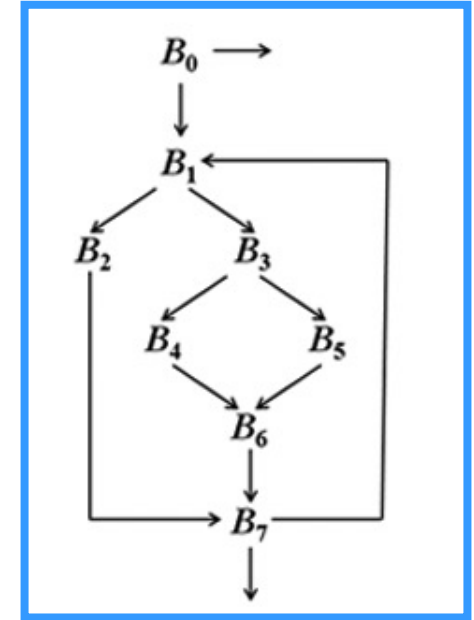
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG

B₇ **a₃ ← ϕ (a₂ , a)**
 b₄ ← ϕ (b₃ , b)
 c₃ ← ϕ (c₂ , c)
 d₃ ← ϕ (d₂ , d)
 y ← + , a₃ , b₄
 z ← + , c₃ , d₃
 i₂ ← + , i₁ , 1

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₀	$i_0 \leftarrow 1$	B₅	$c \leftarrow$
B₁	$a_1 \leftarrow \varphi(a_0, a_3)$ $b_1 \leftarrow \varphi(b_0, b_4)$ $c_1 \leftarrow \varphi(c_0, c_3)$ $d_1 \leftarrow \varphi(d_0, d_3)$ $i_1 \leftarrow \varphi(i_0, i_2)$ $a_2 \leftarrow$ $b_2 \leftarrow$	B₇	$a_3 \leftarrow \varphi(a_2, a)$ $b_4 \leftarrow \varphi(b_3, b)$ $c_3 \leftarrow \varphi(c_2, c)$ $d_3 \leftarrow \varphi(d_2, d)$ $y \leftarrow +, a_3, b_4$ $z \leftarrow +, c_3, d_3$ $i_2 \leftarrow +, i_1, 1$
B₂	$b_3 \leftarrow$ $c_2 \leftarrow$ $d_2 \leftarrow$	B₃	$a \leftarrow$ $d \leftarrow$
B₄	$d \leftarrow$	B₆	$c \leftarrow \varphi(c, c)$ $d \leftarrow \varphi(d, d)$ $b \leftarrow$



<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
4	5	4	4	3
a₀	b₀	c₀	d₀	i₀
a₁	b₁	c₁	d₁	i₁
a₂	b₂	c₃	d₃	i₂
a₃	b₄			

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₇

a *b* *c* *d* *i*

Счетчики

4 5 4 4 3

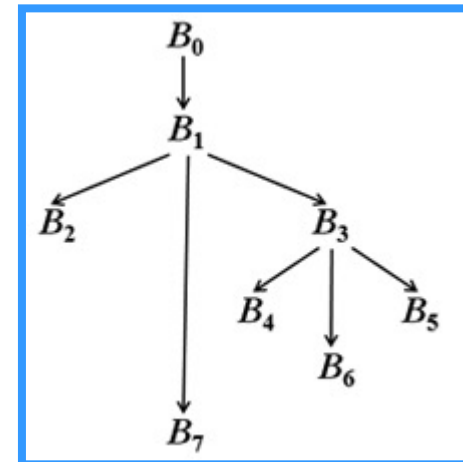
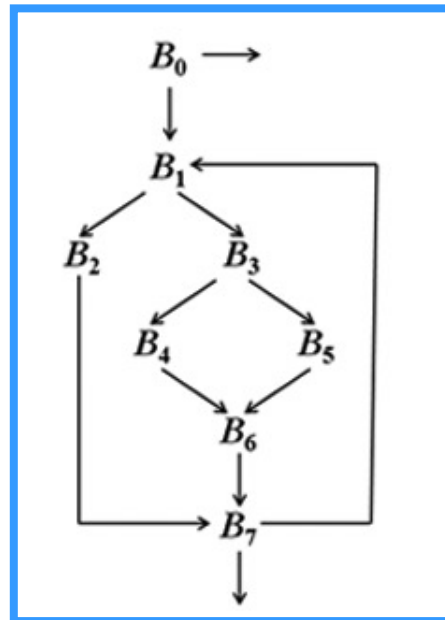
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₃** **d₃** **i₂**

a₃ **b₄**



Rename(B₇):

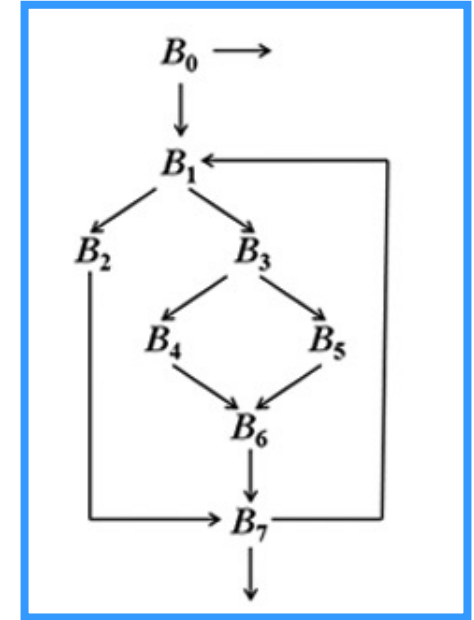
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG
4. Нет потомков в дереве доминаторов
5. Очистка стека. **return** // -> B₁

B₇ a₃ ← ϕ (a₂ , a)
 b₄ ← ϕ (b₃ , b)
 c₃ ← ϕ (c₂ , c)
 d₃ ← ϕ (d₂ , d)
 y ← + , a₃ , b₄
 z ← + , c₃ , d₃
 i₂ ← + , i₁ , 1

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₀	i₀ ← 1	B₅	c ←
B₁	a₁ ← φ(a₀ , a₃) b₁ ← φ(b₀ , b₄) c₁ ← φ(c₀ , c₃) d₁ ← φ(d₀ , d₃) i₁ ← φ(i₀ , i₂) a₂ ← b₂ ←	B₇	a₃ ← φ(a₂ , a) b₄ ← φ(b₃ , b) c₃ ← φ(c₂ , c) d₃ ← φ(d₂ , d) y ← +, a₃ , b₄ z ← +, c₃ , d₃ i₂ ← +, i₁ , 1
B₂	b₃ ← c₂ ← d₂ ←	B₃	a ← d ←
B₄	d ←	B₆	c ← φ(c , c) d ← φ(d , d) b ←



<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
4	5	4	4	3
a₀	b₀	c₀	d₀	i₀
a₁	b₁	c₁	d₁	i₁
a₂	b₂			

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_3

a b c d i

Счетчики

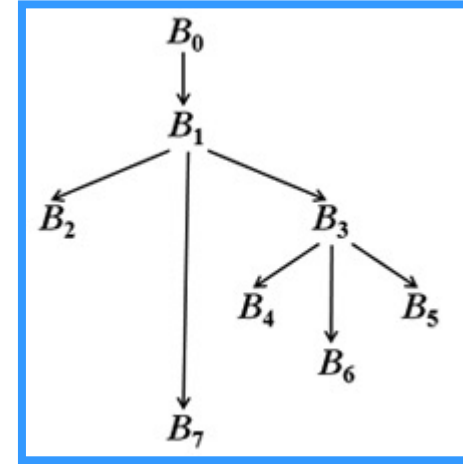
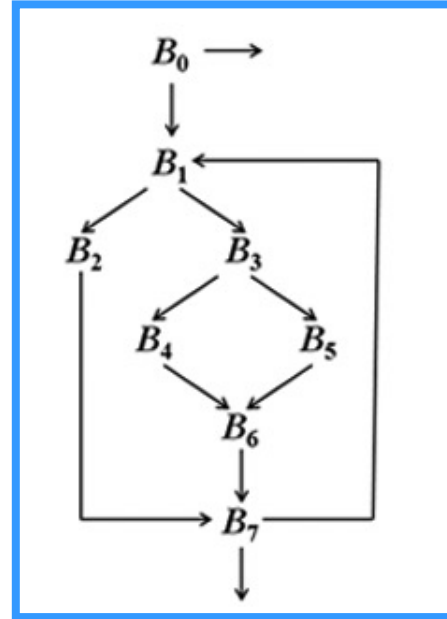
4 5 4 4 3

Стеки ($\downarrow$)

a_0 b_0 c_0 d_0 i_0

a_1 b_1 c_1 d_1 i_1

a_2 b_2 $$ $$ $$



Rename(B_3):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B_3 a $\leftarrow$

d $\leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_3

a b c d i

Счетчики

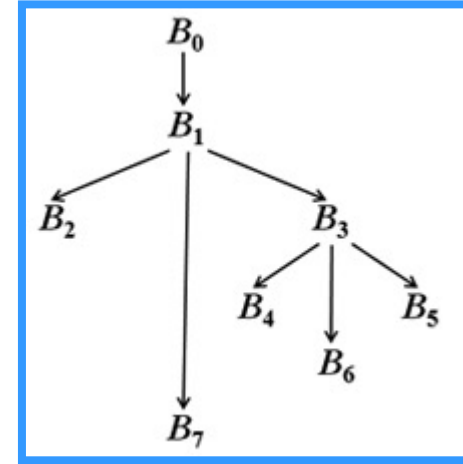
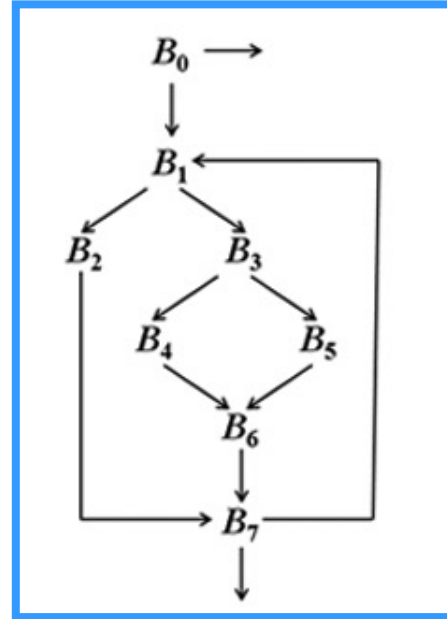
4 5 4 4 3

Стеки ($\downarrow$)

a_0 b_0 c_0 d_0 i_0

a_1 b_1 c_1 d_1 i_1

a_2 b_2 $$ $$ $$



Rename(B_3):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B_3

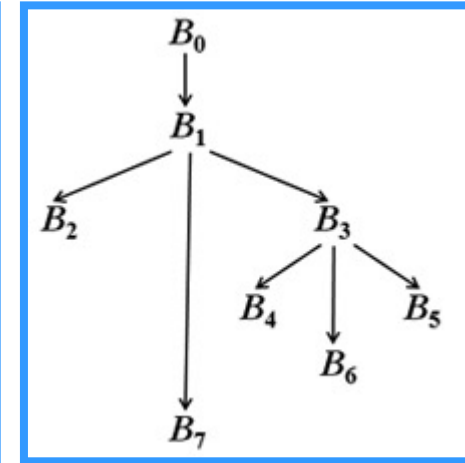
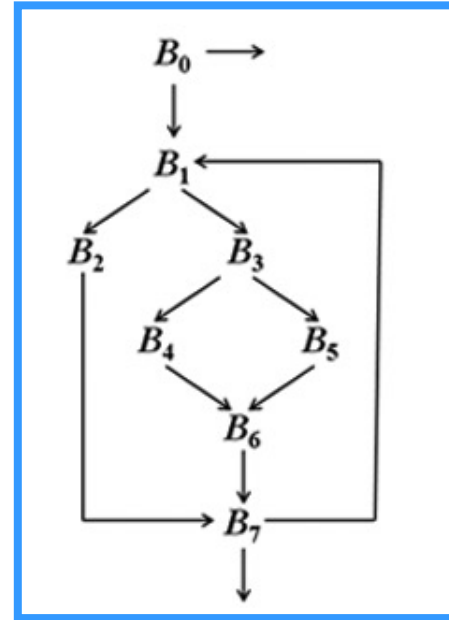
$a \leftarrow$

$d \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_3	a	b	c	d	i
Счетчики	5	5	4	4	3
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	i_0
	a_1	b_1	c_1	d_1	i_1
	a_2	b_2			
	a_4				



Rename(B_3):

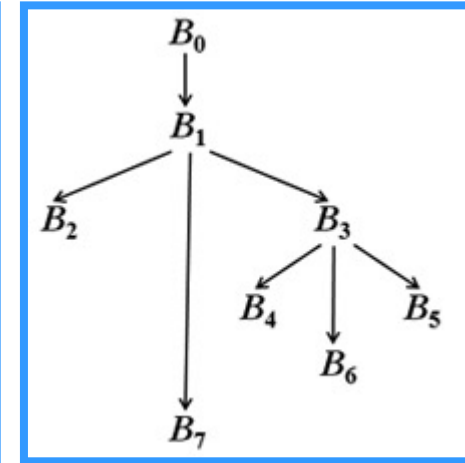
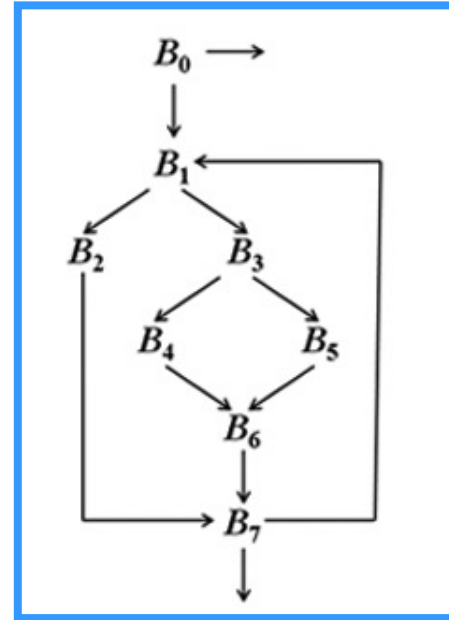
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B_3	$a_4 \leftarrow$
	$d \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_3	a	b	c	d	i
Счетчики	5	5	4	5	3
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	i_0
	a_1	b_1	c_1	d_1	i_1
	a_2	b_2		d_4	
	a_4				



Rename(B_3):

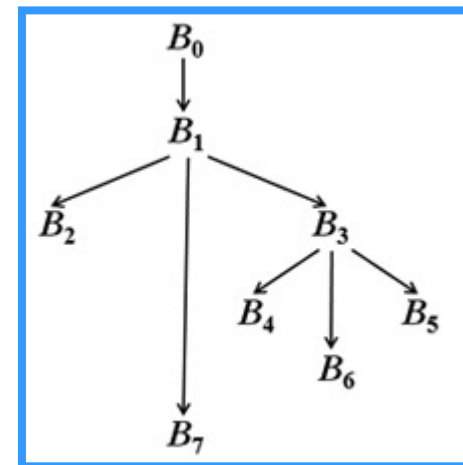
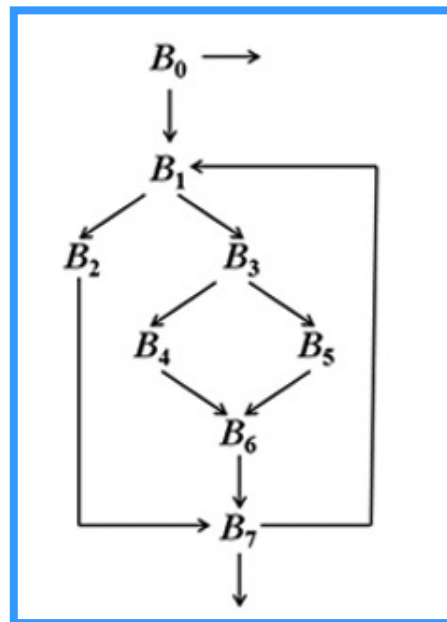
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG

B_3 $a_4 \leftarrow$
 $d_4 \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_3	a	b	c	d	i
Счетчики	5	5	4	5	3
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	i_0
	a_1	b_1	c_1	d_1	i_1
	a_2	b_2		d_4	
	a_4				



Rename(B_3):

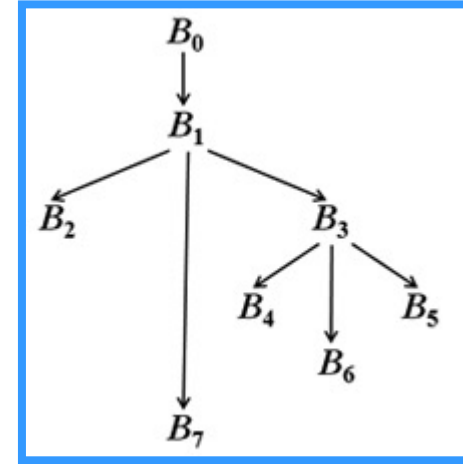
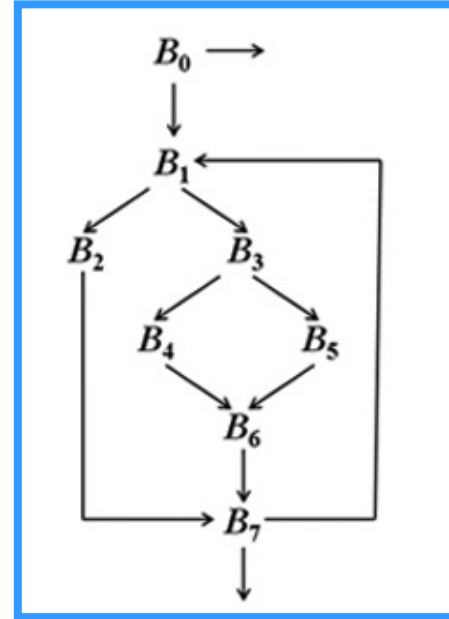
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG
4. Rename потомков в дереве доминаторов

B_3 $a_4 \leftarrow$
 $d_4 \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_3	a	b	c	d	i
Счетчики	5	5	4	5	3
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	i_0
	a_1	b_1	c_1	d_1	i_1
	a_2	b_2		d_4	
	a_4				



Rename(B_1):

- **Rename**(B_2) // Done
- **Rename**(B_7) // Done
- **Rename**(B_3)
 - **Rename**(B_4)

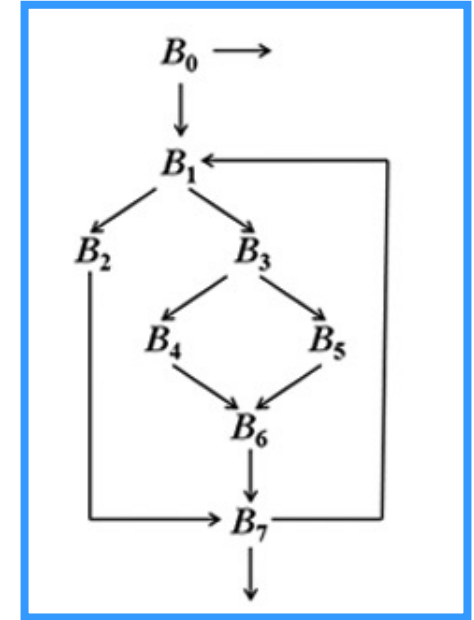
B_3 $a_4 \leftarrow$

$d_4 \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₀	i₀ ← 1	B₅	c ←
B₁	a₁ ← φ(a₀ , a₃) b₁ ← φ(b₀ , b₄) c₁ ← φ(c₀ , c₃) d₁ ← φ(d₀ , d₃) i₁ ← φ(i₀ , i₂) a₂ ← b₂ ←	B₇	a₃ ← φ(a₂ , a) b₄ ← φ(b₃ , b) c₃ ← φ(c₂ , c) d₃ ← φ(d₂ , d) y ← +, a₃ , b₄ z ← +, c₃ , d₃ i₂ ← +, i₁ , 1
B₂	b₃ ← c₂ ← d₂ ←	B₃	a₄ ← d₄ ←
B₄	d ←	B₆	c ← φ(c , c) d ← φ(d , d) b ←



	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
	5	5	4	5	3
a₀	b₀	c₀	d₀	i₀	
a₁	b₁	c₁	d₁	i₁	
a₂	b₂		d₄		
a₄					

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₄

a *b* *c* *d* *i*

Счетчики

5 5 4 5 3

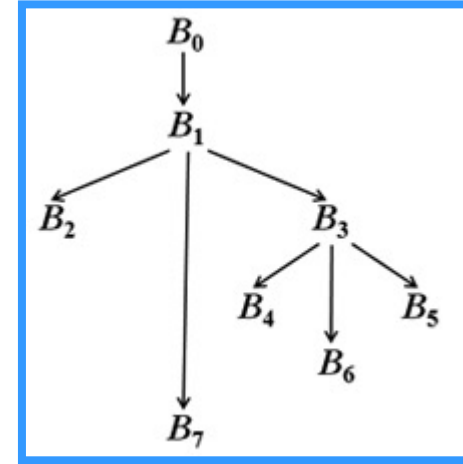
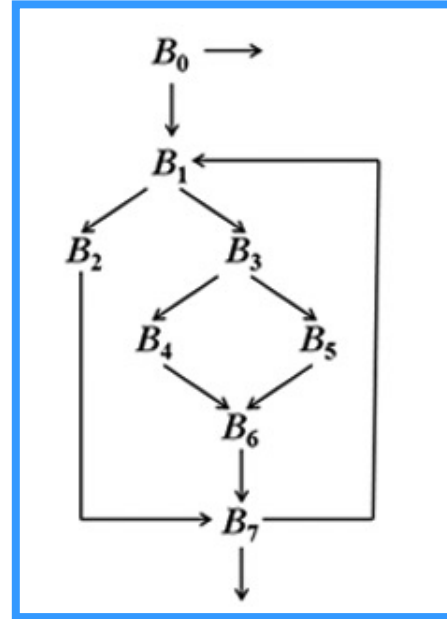
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₁** **d₄** **i₁**

a₄ **b₂** **c₁** **d₄** **i₁**



Rename(B₄):

1. Переименование φ-функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₄

d ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_4

a b c d i

Счетчики

5 5 4 6 3

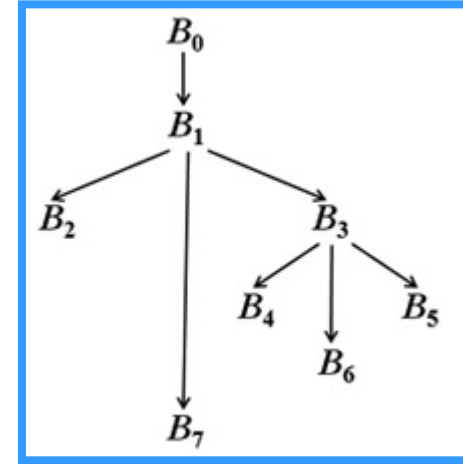
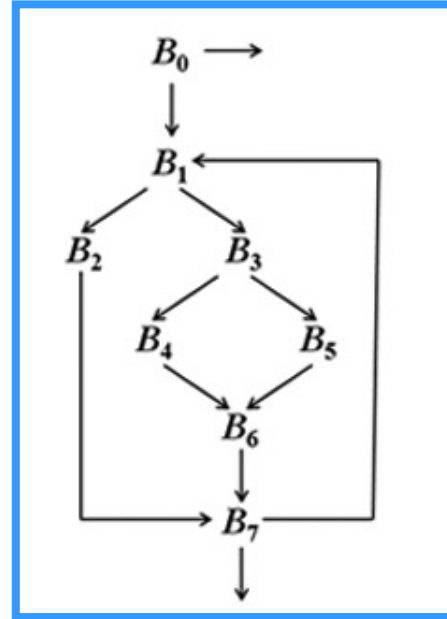
Стеки ($\downarrow$)

a_0 b_0 c_0 d_0 i_0

a_1 b_1 c_1 d_1 i_1

a_2 b_2 d_4

a_4 d_5



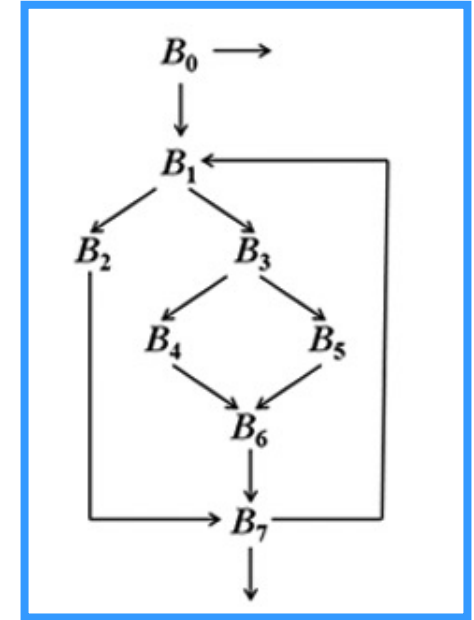
Rename(B_4):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG

B_4 $d_5 \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных



a *b* *c* *d* *i*

5	5	4	6	3
a_0	b_0	c_0	d_0	i_0
a_1	b_1	c_1	d_1	i_1
a_2	b_2		d_4	
a_4			d_5	

B_0	$i_0 \leftarrow 1$	B_5	$c \leftarrow$
B_1	$a_1 \leftarrow \varphi(a_0, a_3)$ $b_1 \leftarrow \varphi(b_0, b_4)$ $c_1 \leftarrow \varphi(c_0, c_3)$ $d_1 \leftarrow \varphi(d_0, d_3)$ $i_1 \leftarrow \varphi(i_0, i_2)$ $a_2 \leftarrow$ $b_2 \leftarrow$	B_7	$a_3 \leftarrow \varphi(a_2, a)$ $b_4 \leftarrow \varphi(b_3, b)$ $c_3 \leftarrow \varphi(c_2, c)$ $d_3 \leftarrow \varphi(d_2, d)$ $y \leftarrow +, a_3, b_4$ $z \leftarrow +, c_3, d_3$ $i_2 \leftarrow +, i_1, 1$
B_2	$b_3 \leftarrow$ $c_2 \leftarrow$ $d_2 \leftarrow$	B_3	$a_4 \leftarrow$ $d_4 \leftarrow$
B_4	$d_5 \leftarrow$	B_6	$c \leftarrow \varphi(c_1, c)$ $d \leftarrow \varphi(d_5, d)$ $b \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₄

a *b* *c* *d* *i*

Счетчики

5 5 4 6 3

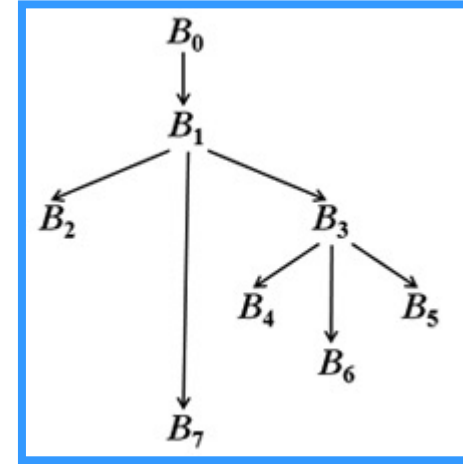
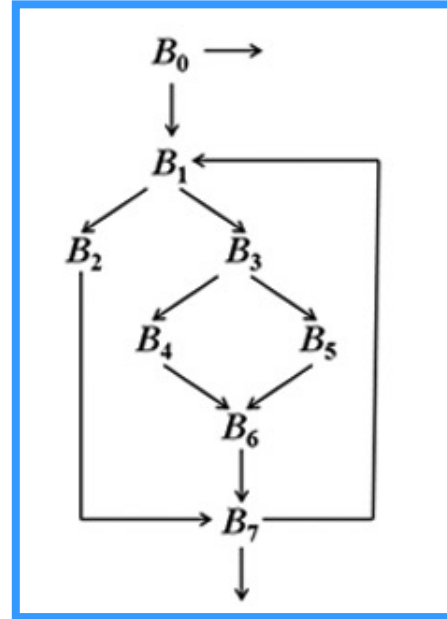
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **d₄**

a₄ **d₅**



Rename(B₄):

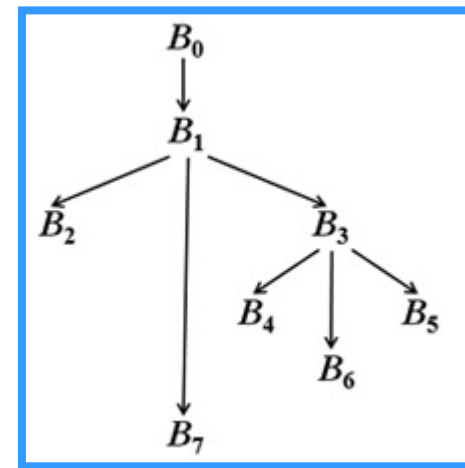
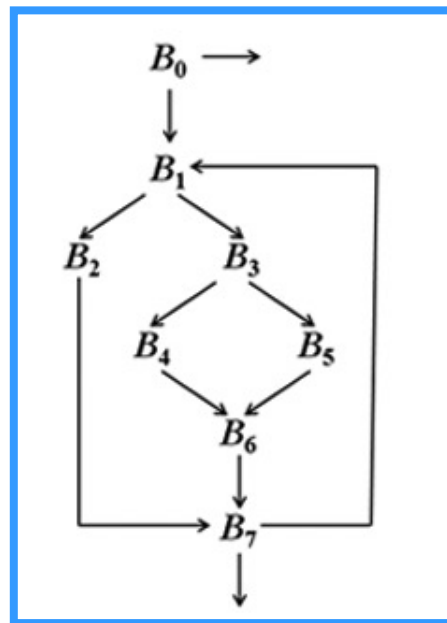
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG
4. Нет потомков в дереве доминаторов
5. Очистка стека. **return** // -> B₃

B₄ d₅ ←

6.3 Построение частично усеченной SSA-формы

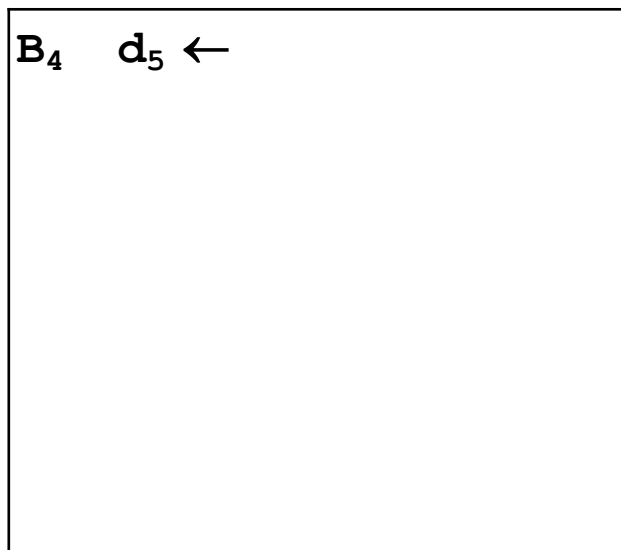
6.3.4. Переименование переменных

B₄	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	5	5	4	6	3
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂		d₄	
	a₄				



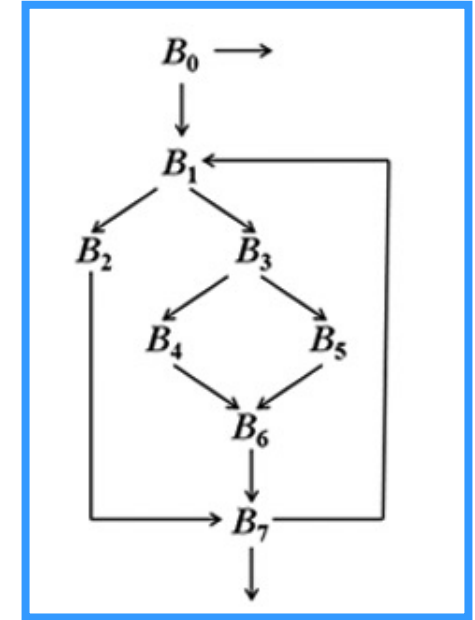
Rename(B₄):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG
4. Нет потомков в дереве доминаторов
5. Очистка стека. **return** // -> B₃



6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных



a *b* *c* *d* *i*

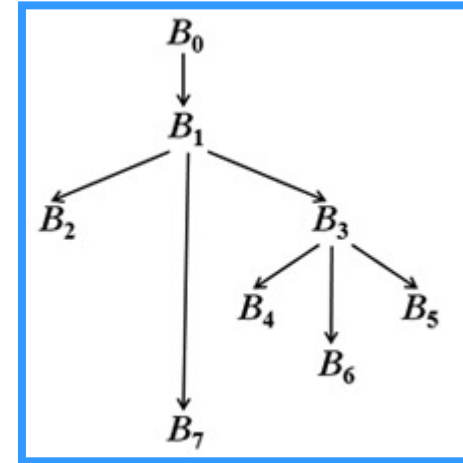
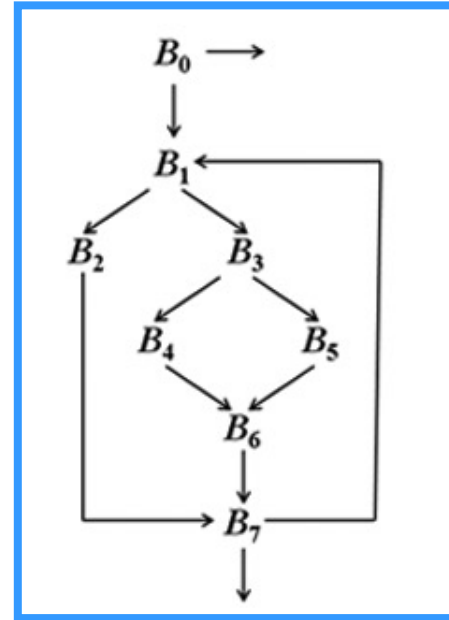
5	5	4	6	3
a_0	b_0	c_0	d_0	i_0
a_1	b_1	c_1	d_1	i_1
a_2	b_2		d_4	
a_4				

B_0	$i_0 \leftarrow 1$	B_5	$c \leftarrow$
B_1	$a_1 \leftarrow \varphi(a_0, a_3)$ $b_1 \leftarrow \varphi(b_0, b_4)$ $c_1 \leftarrow \varphi(c_0, c_3)$ $d_1 \leftarrow \varphi(d_0, d_3)$ $i_1 \leftarrow \varphi(i_0, i_2)$ $a_2 \leftarrow$ $b_2 \leftarrow$	B_7	$a_3 \leftarrow \varphi(a_2, a)$ $b_4 \leftarrow \varphi(b_3, b)$ $c_3 \leftarrow \varphi(c_2, c)$ $d_3 \leftarrow \varphi(d_2, d)$ $y \leftarrow +, a_3, b_4$ $z \leftarrow +, c_3, d_3$ $i_2 \leftarrow +, i_1, 1$
B_2	$b_3 \leftarrow$ $c_2 \leftarrow$ $d_2 \leftarrow$	B_3	$a_4 \leftarrow$ $d_4 \leftarrow$
B_4	$d_5 \leftarrow$	B_6	$c \leftarrow \varphi(c_1, c)$ $d \leftarrow \varphi(d_5, d)$ $b \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_3	a	b	c	d	i
Счетчики	5	5	4	6	3
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	i_0
	a_1	b_1	c_1	d_1	i_1
	a_2	b_2		d_4	
	a_4				



Rename(B_1):

- **Rename**(B_2) // Done
- **Rename**(B_7) // Done
- **Rename**(B_3)
 - **Rename**(B_4)
 - **Rename**(B_6)

B_3 $a_4 \leftarrow$
 $d_4 \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₃

a *b* *c* *d* *i*

Счетчики

5 5 4 6 3

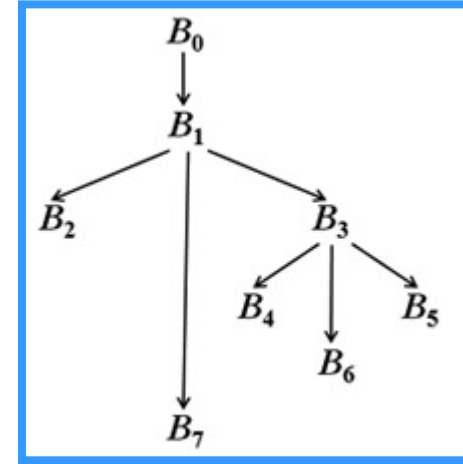
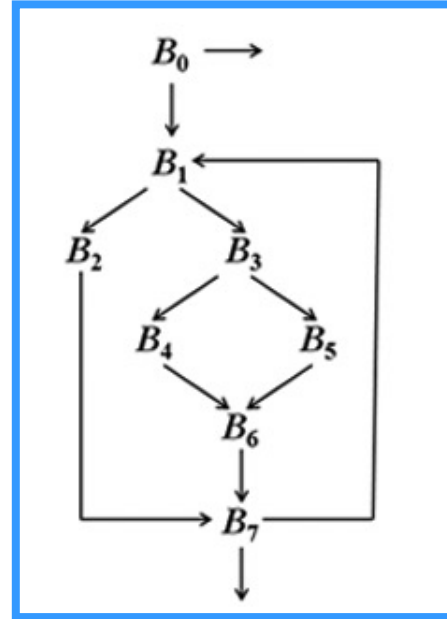
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **d₄**

a₄



Rename(B₆):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₆

c ← ϕ (c₁ , c)

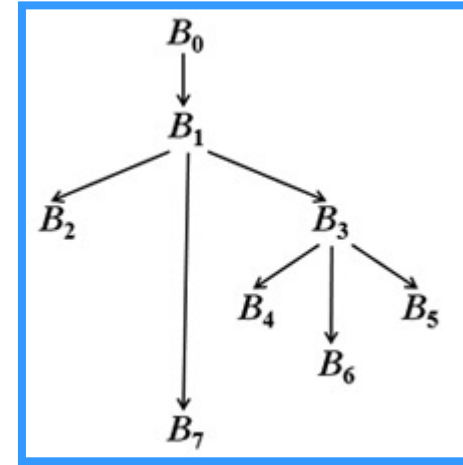
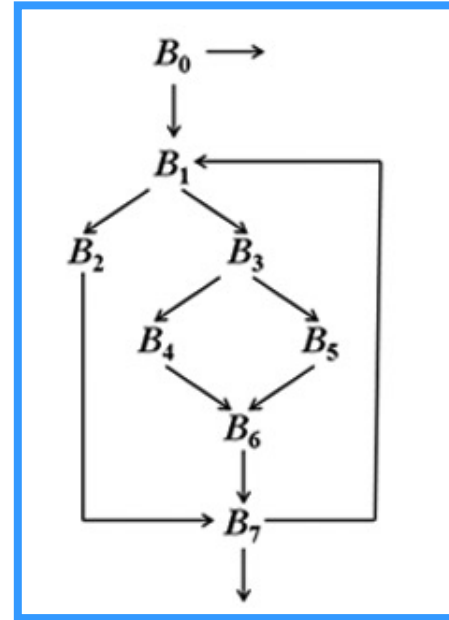
d ← ϕ (d₅ , d)

b ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_3	a	b	c	d	i
Счетчики	5	5	5	6	3
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	i_0
	a_1	b_1	c_1	d_1	i_1
	a_2	b_2	c_4	d_4	
	a_4				



Rename(B_6):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B_6 $c_4 \leftarrow \phi(c_1, c)$
 $d \leftarrow \phi(d_5, d)$
 $b \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₃

a *b* *c* *d* *i*

Счетчики

5 5 5 7 3

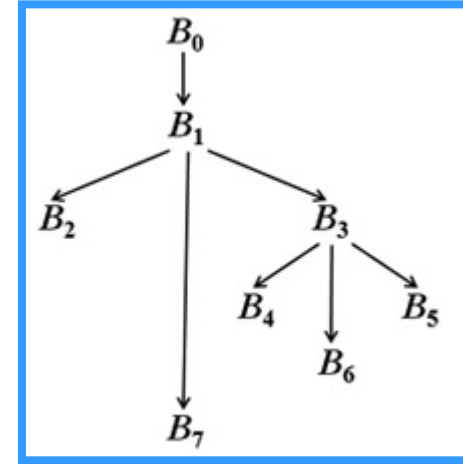
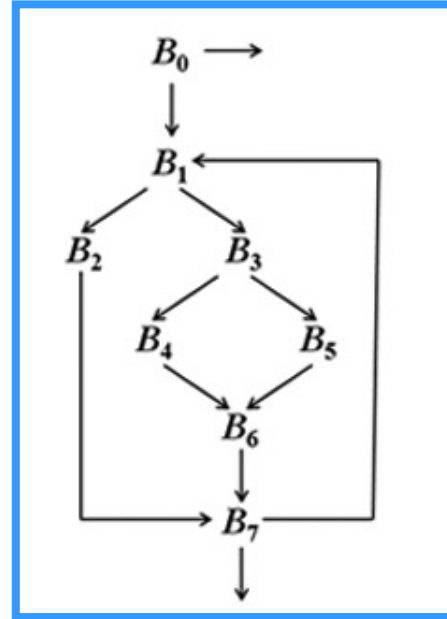
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **c₄** **d₄**

a₄ **d₆**



Rename(B₆):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда

B₆ **c₄ ← ϕ (c₁ , c)**

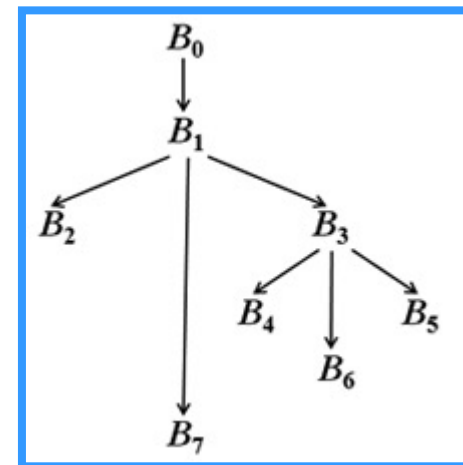
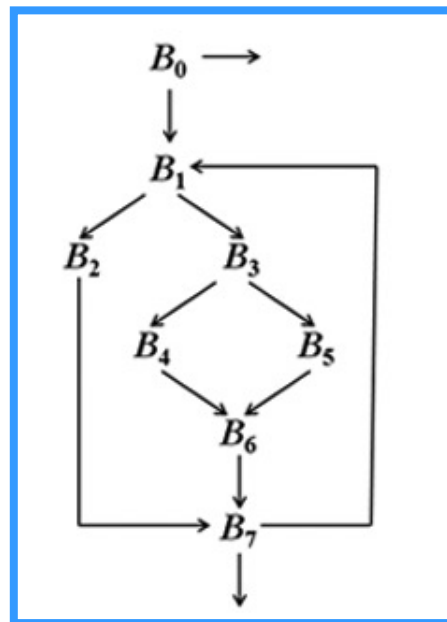
d₆ ← ϕ (d₅ , d)

b ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₃	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	5	6	5	7	3
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂	c₄	d₄	
	a₄	b₅		d₆	



Rename(B₆):

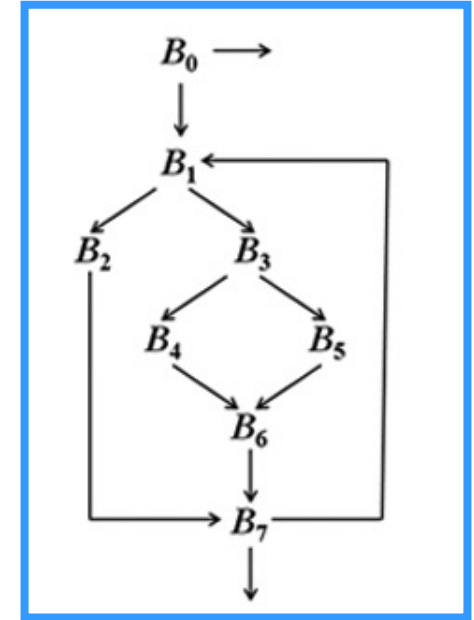
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG

B₆ c₄ ← ϕ (c₁ , c)
 d₆ ← ϕ (d₅ , d)
 b₅ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₀	$i_0 \leftarrow 1$	B₅	$c \leftarrow$
B₁	$a_1 \leftarrow \varphi(a_0, a_3)$ $b_1 \leftarrow \varphi(b_0, b_4)$ $c_1 \leftarrow \varphi(c_0, c_3)$ $d_1 \leftarrow \varphi(d_0, d_3)$ $i_1 \leftarrow \varphi(i_0, i_2)$ $a_2 \leftarrow$ $b_2 \leftarrow$	B₇	$a_3 \leftarrow \varphi(a_2, a_4)$ $b_4 \leftarrow \varphi(b_3, b_5)$ $c_3 \leftarrow \varphi(c_2, c_4)$ $d_3 \leftarrow \varphi(d_2, d_6)$ $y \leftarrow +, a_3, b_4$ $z \leftarrow +, c_3, d_3$ $i_2 \leftarrow +, i_1, 1$
B₂	$b_3 \leftarrow$ $c_2 \leftarrow$ $d_2 \leftarrow$	B₃	$a_4 \leftarrow$ $d_4 \leftarrow$
B₄	$d_5 \leftarrow$	B₆	$c_4 \leftarrow \varphi(c_1, c)$ $d_6 \leftarrow \varphi(d_5, d)$ $b_5 \leftarrow$



<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
5	6	5	7	3
a₀	b₀	c₀	d₀	i₀
a₁	b₁	c₁	d₁	i₁
a₂	b₂	c₄	d₄	
a₄	b₅		d₆	

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₃

a *b* *c* *d* *i*

Счетчики

5 6 5 7 3

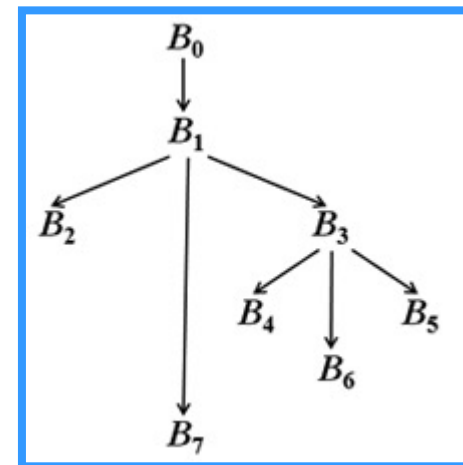
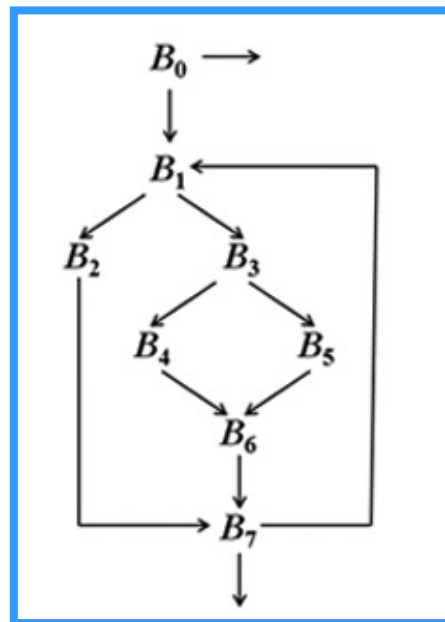
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **d₄**

a₄



Rename(B₆):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG
4. Нет потомков в дереве доминаторов
5. Очистка стека. **return** // -> B₃

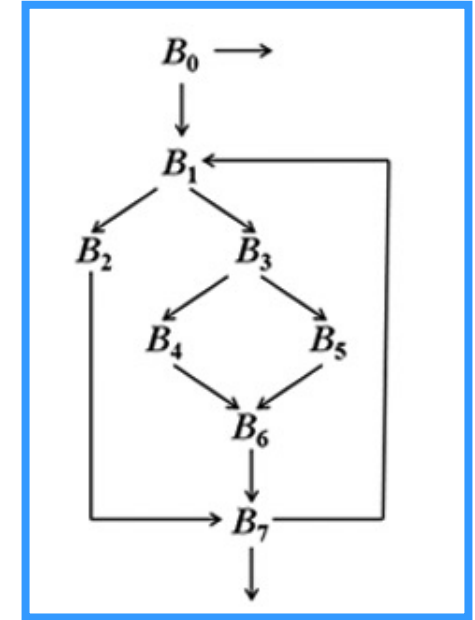
B₆ **c₄ ← ϕ (c₁ , c)**

d₆ ← ϕ (d₅ , d)

b₅ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных



a *b* *c* *d* *i*

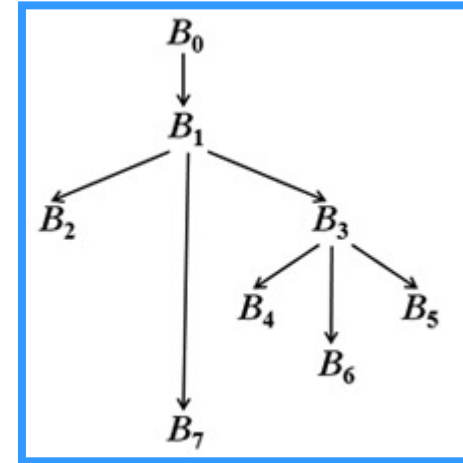
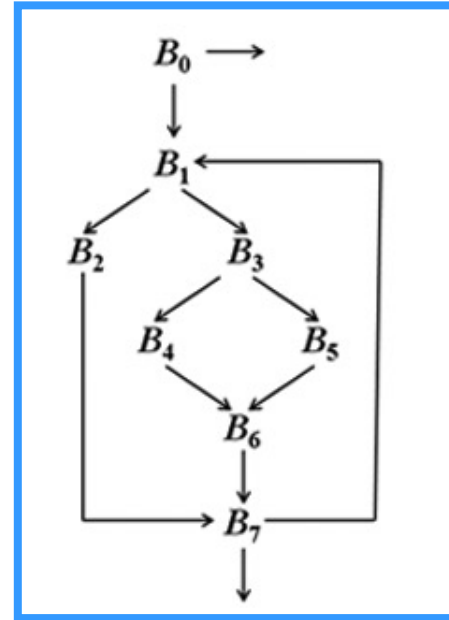
5	6	5	7	3
a_0	b_0	c_0	d_0	i_0
a_1	b_1	c_1	d_1	i_1
a_2	b_2		d_4	
a_4				

B_0	$i_0 \leftarrow 1$	B_5	$c \leftarrow$
B_1	$a_1 \leftarrow \varphi(a_0, a_3)$ $b_1 \leftarrow \varphi(b_0, b_4)$ $c_1 \leftarrow \varphi(c_0, c_3)$ $d_1 \leftarrow \varphi(d_0, d_3)$ $i_1 \leftarrow \varphi(i_0, i_2)$ $a_2 \leftarrow$ $b_2 \leftarrow$	B_7	$a_3 \leftarrow \varphi(a_2, a_4)$ $b_4 \leftarrow \varphi(b_3, b_5)$ $c_3 \leftarrow \varphi(c_2, c_4)$ $d_3 \leftarrow \varphi(d_2, d_6)$ $y \leftarrow +, a_3, b_4$ $z \leftarrow +, c_3, d_3$ $i_2 \leftarrow +, i_1, 1$
B_2	$b_3 \leftarrow$ $c_2 \leftarrow$ $d_2 \leftarrow$	B_3	$a_4 \leftarrow$ $d_4 \leftarrow$
B_4	$d_5 \leftarrow$	B_6	$c_4 \leftarrow \varphi(c_1, c)$ $d_6 \leftarrow \varphi(d_5, d)$ $b_5 \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₃	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	5	6	5	7	3
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂		d₄	
	a₄				



Rename(B₁):

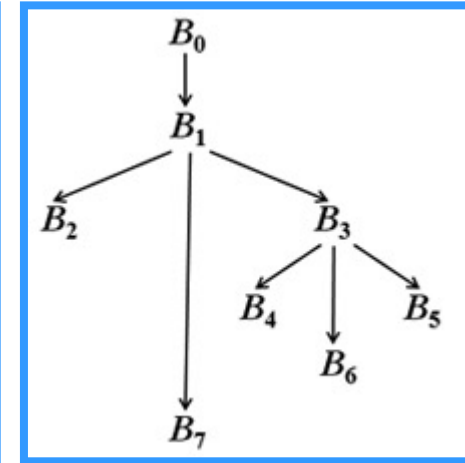
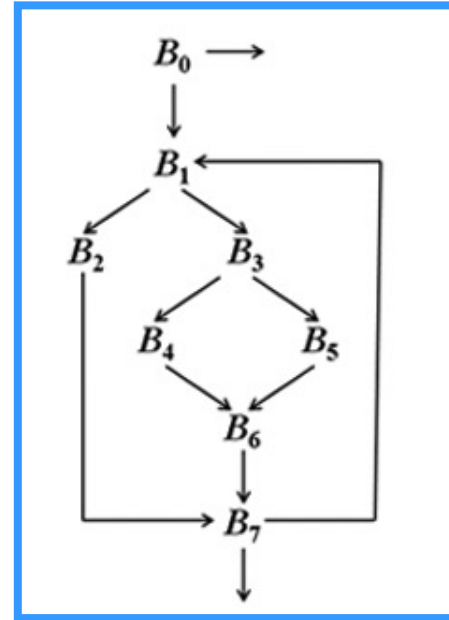
- **Rename(B₂)** // Done
- **Rename(B₇)** // Done
- **Rename(B₃)**
 - **Rename(B₄)**
 - **Rename(B₆)**
 - **Rename(B₅)**

B₃ a₄ ←
d₄ ←

6.3 Построение частично усеченной SSA-формы

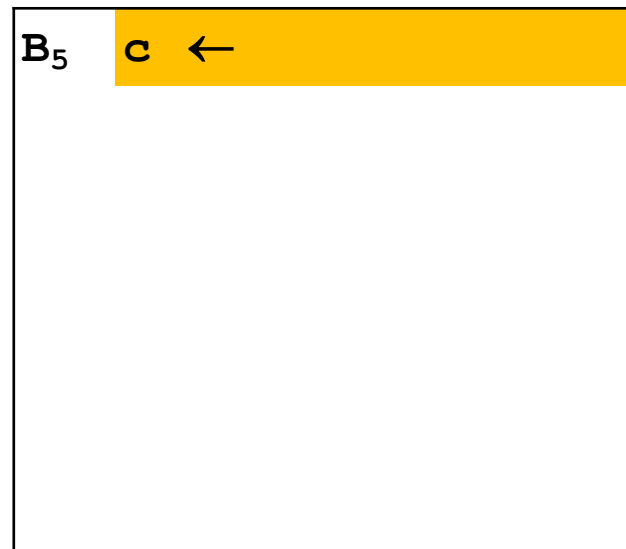
6.3.4. Переименование переменных

B₅	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	5	6	5	7	3
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂		d₄	
	a₄				



Rename(B₅):

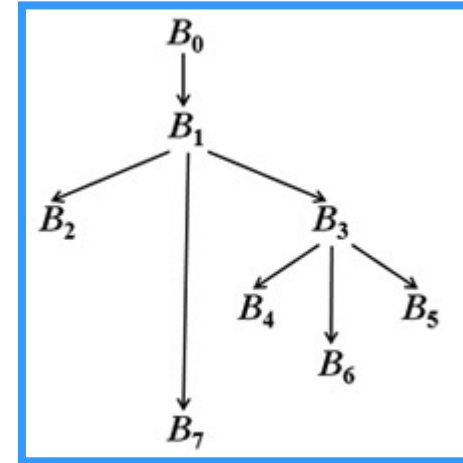
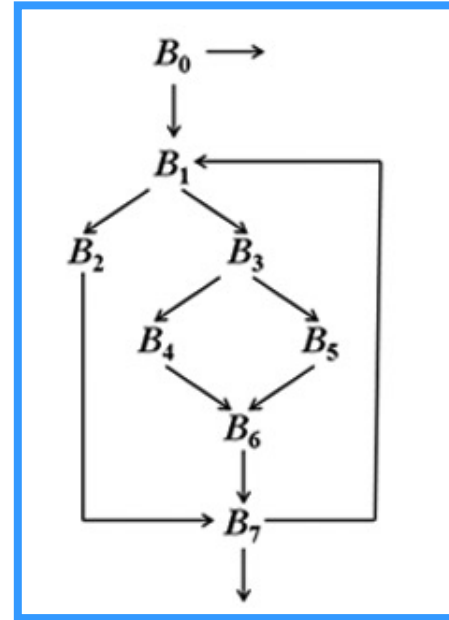
1. Переименование φ-функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда



6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₅	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	5	6	6	7	3
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂	c₅	d₄	
	a₄				



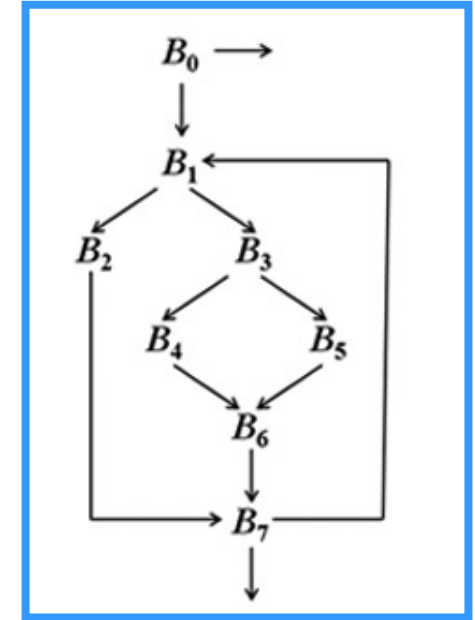
Rename(B₅):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG

B₅ c₅ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных



a *b* *c* *d* *i*

5	6	6	7	3
a_0	b_0	c_0	d_0	i_0
a_1	b_1	c_1	d_1	i_1
a_2	b_2	c_5	d_4	
a_4				

B_0	$i_0 \leftarrow 1$	B_5	$c_5 \leftarrow$
B_1	$a_1 \leftarrow \varphi(a_0, a_3)$ $b_1 \leftarrow \varphi(b_0, b_4)$ $c_1 \leftarrow \varphi(c_0, c_3)$ $d_1 \leftarrow \varphi(d_0, d_3)$ $i_1 \leftarrow \varphi(i_0, i_2)$ $a_2 \leftarrow$ $b_2 \leftarrow$	B_7	$a_3 \leftarrow \varphi(a_2, a_4)$ $b_4 \leftarrow \varphi(b_3, b_5)$ $c_3 \leftarrow \varphi(c_2, c_4)$ $d_3 \leftarrow \varphi(d_2, d_6)$ $y \leftarrow +, a_3, b_4$ $z \leftarrow +, c_3, d_3$ $i_2 \leftarrow +, i_1, 1$
B_2	$b_3 \leftarrow$ $c_2 \leftarrow$ $d_2 \leftarrow$	B_3	$a_4 \leftarrow$ $d_4 \leftarrow$
B_4	$d_5 \leftarrow$	B_6	$c_4 \leftarrow \varphi(c_1, c_5)$ $d_6 \leftarrow \varphi(d_5, d_4)$ $b_5 \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₅

a *b* *c* *d* *i*

Счетчики

5 6 6 7 3

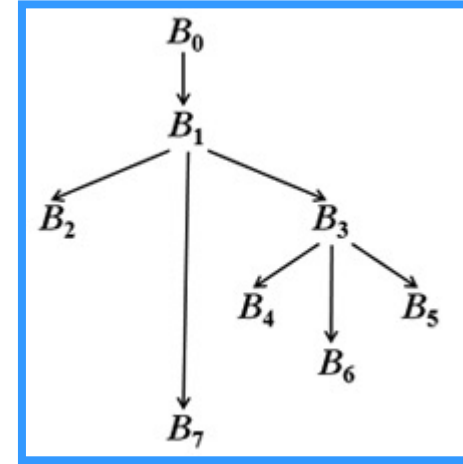
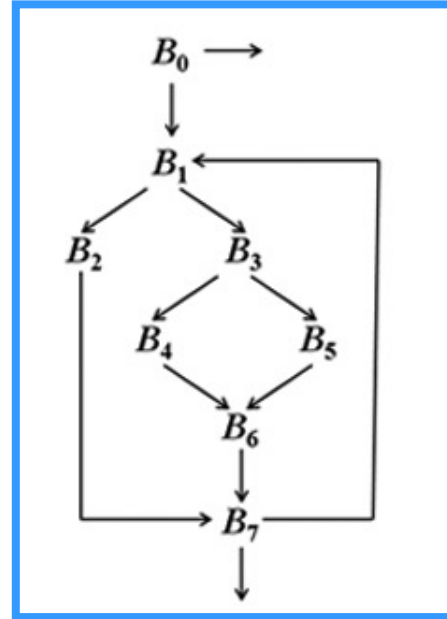
Стеки (↓)

a₀ **b₀** **c₀** **d₀** **i₀**

a₁ **b₁** **c₁** **d₁** **i₁**

a₂ **b₂** **d₄**

a₄



Rename(B₅):

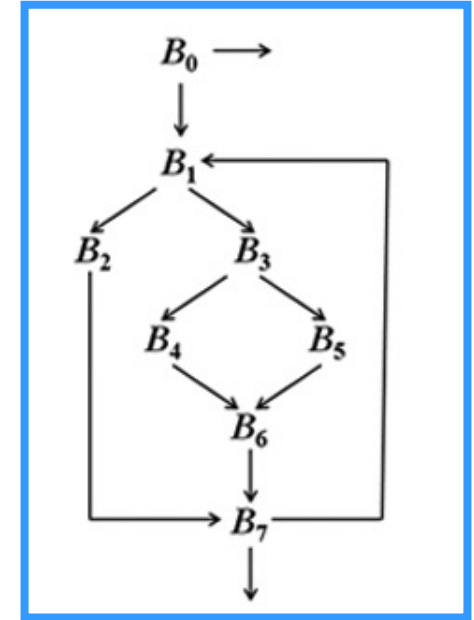
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG
4. Нет потомков в дереве доминаторов
5. Очистка стека. **return** // -> B₃

B₅ c₅ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₀	$i_0 \leftarrow 1$	B₅	$c_5 \leftarrow$
B₁	$a_1 \leftarrow \varphi(a_0, a_3)$ $b_1 \leftarrow \varphi(b_0, b_4)$ $c_1 \leftarrow \varphi(c_0, c_3)$ $d_1 \leftarrow \varphi(d_0, d_3)$ $i_1 \leftarrow \varphi(i_0, i_2)$ $a_2 \leftarrow$ $b_2 \leftarrow$	B₇	$a_3 \leftarrow \varphi(a_2, a_4)$ $b_4 \leftarrow \varphi(b_3, b_5)$ $c_3 \leftarrow \varphi(c_2, c_4)$ $d_3 \leftarrow \varphi(d_2, d_6)$ $y \leftarrow +, a_3, b_4$ $z \leftarrow +, c_3, d_3$ $i_2 \leftarrow +, i_1, 1$
B₂	$b_3 \leftarrow$ $c_2 \leftarrow$ $d_2 \leftarrow$	B₃	$a_4 \leftarrow$ $d_4 \leftarrow$
B₄	$d_5 \leftarrow$	B₆	$c_4 \leftarrow \varphi(c_1, c_5)$ $d_6 \leftarrow \varphi(d_5, d_4)$ $b_5 \leftarrow$

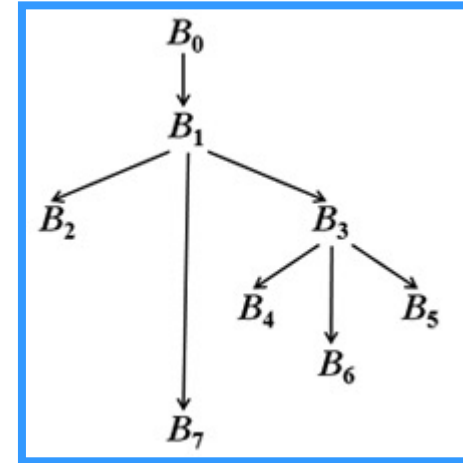
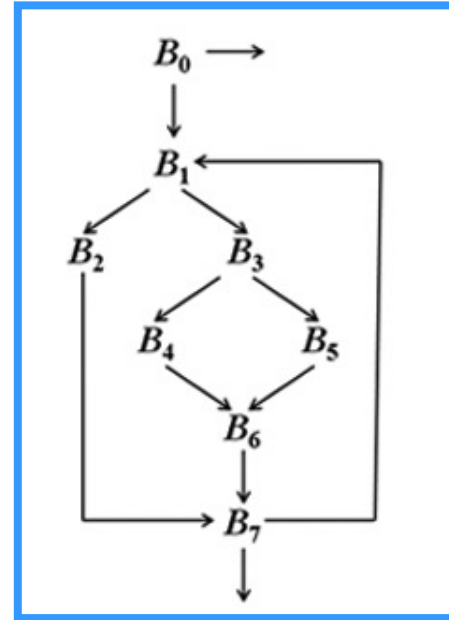


	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
	5	6	6	7	3
a₀		b₀	c₀	d₀	i₀
a₁		b₁	c₁	d₁	i₁
a₂		b₂		d₄	
a₄					

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₃	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	5	6	6	7	3
Стеки (↓)	a₀	b₀	c₀	d₀	i₀
	a₁	b₁	c₁	d₁	i₁
	a₂	b₂		d₄	
	a₄				



Rename(B₁):

- **Rename(B₂)** // Done
- **Rename(B₇)** // Done
- **Rename(B₃)**
 - **Rename(B₄)**
 - **Rename(B₆)**
 - **Rename(B₅)**

B₃ **a₄ ←**
 d₄ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_3

a b c d i

Счетчики

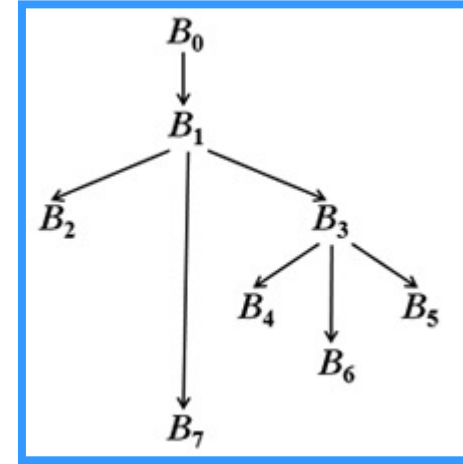
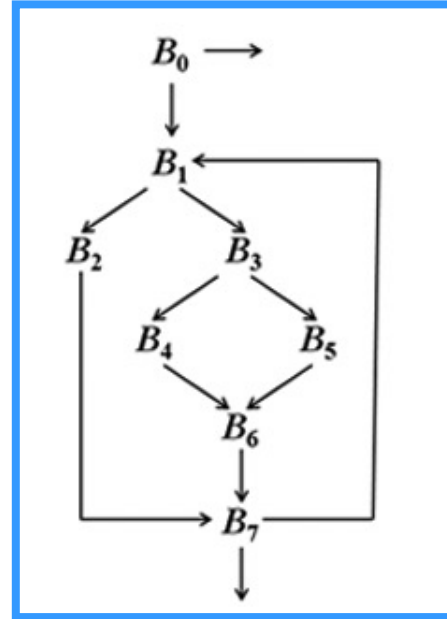
5 6 6 7 3

Стеки ($\downarrow$)

a_0 b_0 c_0 d_0 i_0

a_1 b_1 c_1 d_1 i_1

a_2 b_2



Rename(B_3):

1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG
4. *Rename* потомков в в дереве дом.
5. Очистка стека. ***return*** B_1

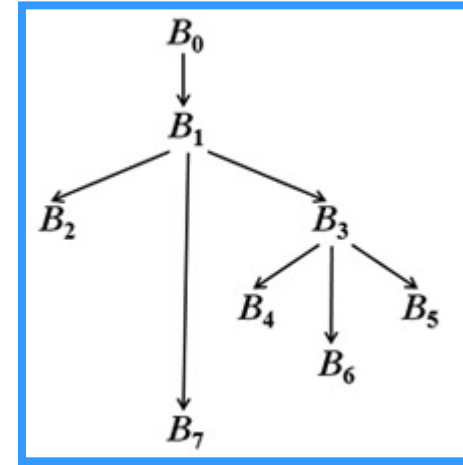
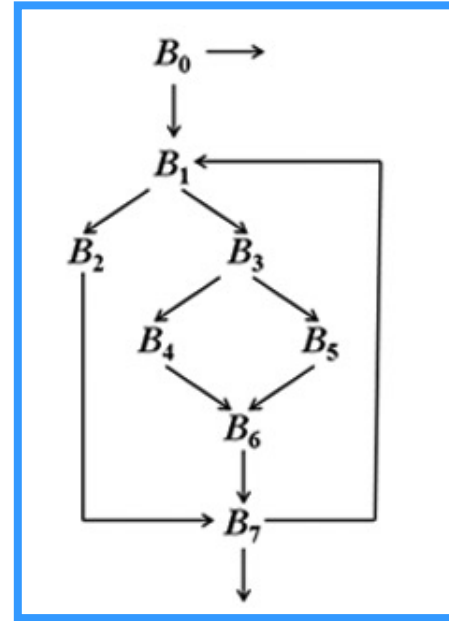
B_3 $a_4 \leftarrow$

$d_4 \leftarrow$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₁	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
Счетчики	5	6	6	7	3
Стеки (↓)	a₀	b₀	c₀	d₀	i₀



Rename(B₁):

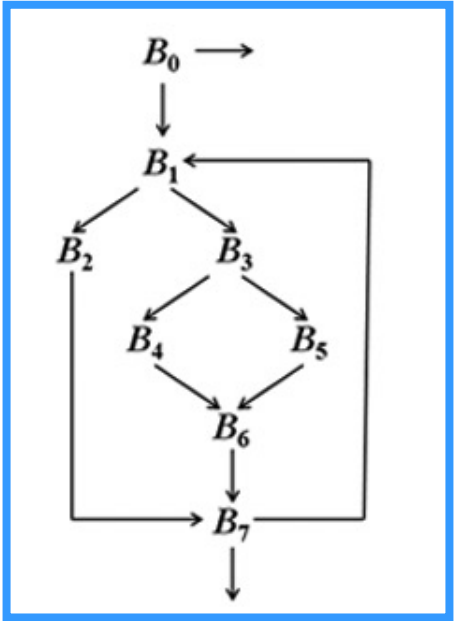
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG
4. Rename потомков в в дереве дом.
5. Очистка стека. **return** B₀

B₁ a₁ ← ϕ (a₀ , a₃)
 b₁ ← ϕ (b₀ , b₄)
 c₁ ← ϕ (c₀ , c₃)
 d₁ ← ϕ (d₀ , d₃)
 i₁ ← ϕ (i₀ , i₂)
 a₂ ←
 b₂ ←

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B ₀	i ₀ ← 1	B ₅	c ₅ ←
B ₁	a ₁ ← φ(a ₀ , a ₃) b ₁ ← φ(b ₀ , b ₄) c ₁ ← φ(c ₀ , c ₃) d ₁ ← φ(d ₀ , d ₃) i ₁ ← φ(i ₀ , i ₂) a ₂ ← b ₂ ←	B ₇	a ₃ ← φ(a ₂ , a ₄) b ₄ ← φ(b ₃ , b ₅) c ₃ ← φ(c ₂ , c ₄) d ₃ ← φ(d ₂ , d ₆) y ← +, a ₃ , b ₄ z ← +, c ₃ , d ₃ i ₂ ← +, i ₁ , 1
B ₂	b ₃ ← c ₂ ← d ₂ ←	B ₃	a ₄ ← d ₄ ←
B ₄	d ₅ ←	B ₆	c ₄ ← φ(c ₁ , c ₅) d ₆ ← φ(d ₅ , d ₄) b ₅ ←

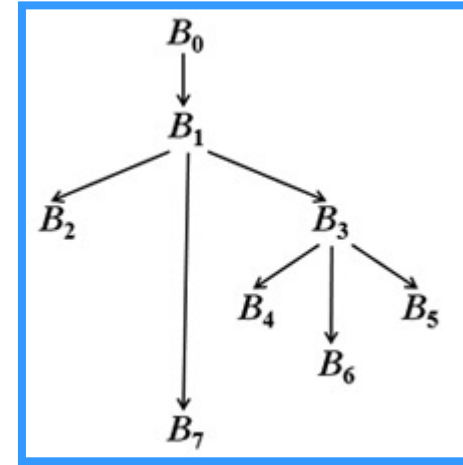
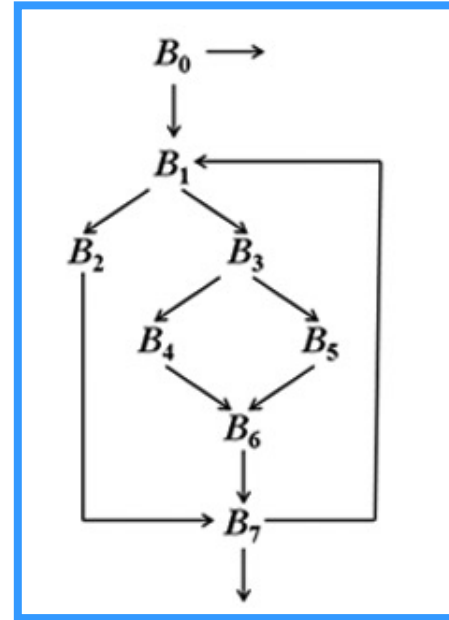


	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
	5	6	6	7	3
a ₀		b ₀	c ₀	d ₀	i ₀

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B_0	a	b	c	d	i
Счетчики	5	6	6	7	3
Стеки ($\downarrow$)	a_0	b_0	c_0	d_0	



Rename(B_0):

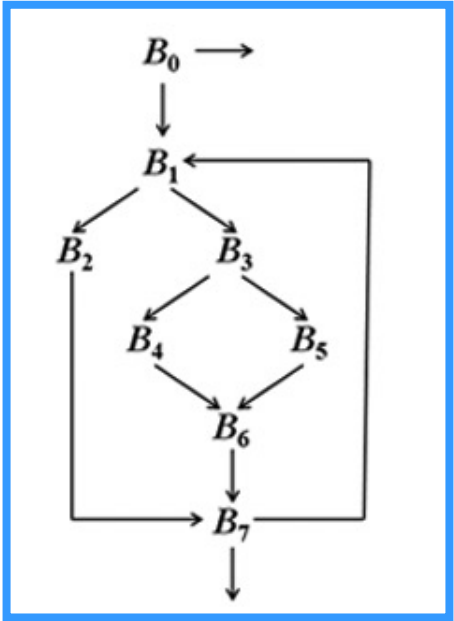
1. Переименование ϕ -функций
2. Переименование инструкций
 1. Переименование операндов
 2. Переименование результирующего операнда
3. Заполнение параметров ϕ -функций в потомках в CFG
4. Rename потомков в в дереве дом.
5. Очистка стека. **return**

$B_0 \quad i_0 \leftarrow 1$

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

B₀	i₀ ← 1	B₅	c₅ ←
B₁	a₁ ← φ (a₀ , a₃) b₁ ← φ (b₀ , b₄) c₁ ← φ (c₀ , c₃) d₁ ← φ (d₀ , d₃) i₁ ← φ (i₀ , i₂) a₂ ← b₂ ←	B₇	a₃ ← φ (a₂ , a₄) b₄ ← φ (b₃ , b₅) c₃ ← φ (c₂ , c₄) d₃ ← φ (d₂ , d₆) y ← +, a₃ , b₄ z ← +, c₃ , d₃ i₂ ← +, i₁ , 1
B₂	b₃ ← c₂ ← d₂ ←	B₃	a₄ ← d₄ ←
B₄	d₅ ←	B₆	c₄ ← φ (c₁ , c₅) d₆ ← φ (d₅ , d₄) b₅ ←



<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>i</i>
5	6	6	7	3
a₀	b₀	c₀	d₀	

6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

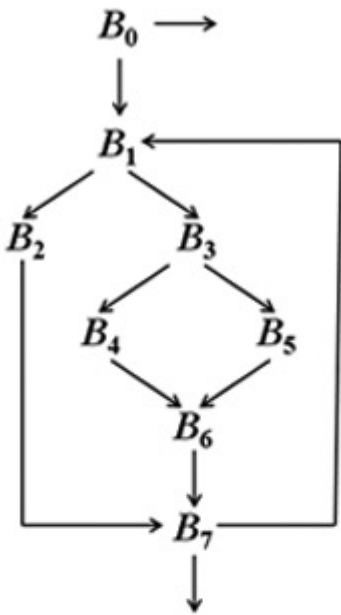
Блок B_0

Когда в конце обхода дерева доминаторов управление снова попадает в B_0 *Rename* выталкивает значение из стека для i и происходит выход из *Rename*.

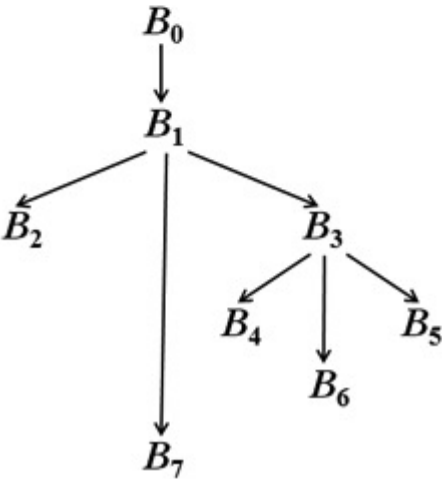
Состояние счетчиков и стеков в момент, когда *Rename*(B_0) уже закончила обрабатывать блок B_0 , но еще не удалила имена, связанные с B_0 из соответствующих стеков:

Глобальные переменные	a	b	c	d	i
Счетчики	5	6	6	7	3
Стеки	a_0	b_0	c_0	d_0	i_0

Граф потока управления



Дерево доминаторов



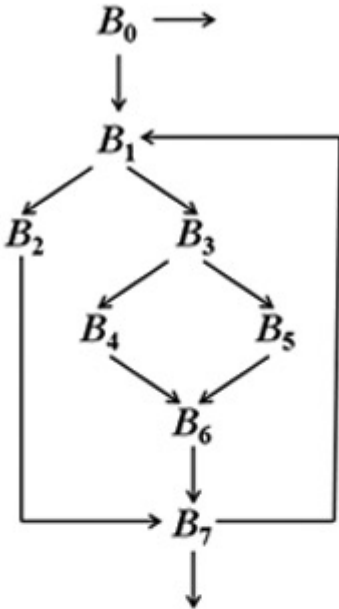
6.3 Построение частично усеченной SSA-формы

6.3.4. Переименование переменных

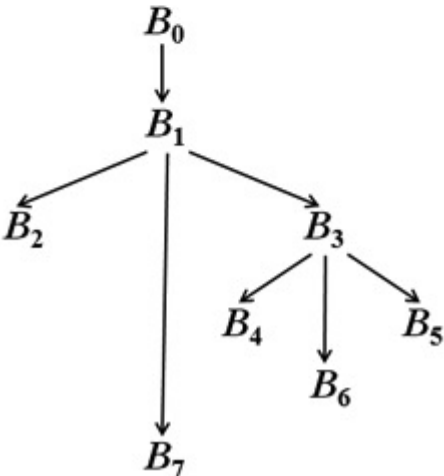
После окончания работы алгоритма получим

B₀ i₀ ← 1	B₅ c₅ ←
B₁ a₁ ← φ (a₀ , a₃) b₁ ← φ (b₀ , b₄) c₁ ← φ (c₀ , c₃) d₁ ← φ (d₀ , d₃) i₁ ← φ (i₀ , i₂) a₂ ← b₂ ←	B₇ a₃ ← φ (a₂ , a₄) b₄ ← φ (b₃ , b₅) c₃ ← φ (c₂ , c₄) d₃ ← φ (d₂ , d₆) y ← +, a₃ , b₄ z ← +, c₃ , d₃ i₂ ← +, i₁ , 1
B₂ b₃ ← c₂ ← d₂ ←	B₃ a₄ ← d₄ ←
B₄ d₅ ←	B₆ c₄ ← φ (c₁ , c₅) d₆ ← φ (d₅ , d₄) b₅ ←

Граф потока управления



Дерево доминаторов



6.4 Восстановление кода из SSA-формы

6.4.1. Постановка задачи

- ◇ Поскольку современные процессоры не выполняют ϕ -функций, компилятор должен перевести программу в *SSA*-форме в выполняемый код.
- ◇ Рассмотрение примеров наводит на мысль, что **компилятору достаточно попросту опустить индексы имен и удалить ϕ -функции.**
Такой возврат к обычной форме будет корректным только сразу же после перехода в *SSA*-форму, но будет уже некорректным после выполнения оптимизаций, при которых могут появиться несколько *одновременно живых* переменных с разными индексами и одним базовым именем.
- ◇ Простое переименование с помощью «отбрасывания» индексов **может привести к некорректному коду.**

6.4 Восстановление кода из SSA-формы

6.4.1. Постановка задачи

◇ **Пример.** Внизу слева показан базовый блок из четырех команд и его оптимизация с помощью ЛНЗ над исходными именами. Справа показан тот же самый пример с использованием *SSA*-имен. Вследствие того, что исходное имя *a* имеет разные *SSA*-имена a_0 и a_1 , удалось оптимизировать последнее присваивание. Заметим однако, что если у имен опустить индексы, получится некорректный код: *c* будет присвоено значение 17.

Исходные имена		SSA-имена	
$a \leftarrow +, x, y$	$a \leftarrow +, x, y$	$a_0 \leftarrow +, x_0, y_0$	$a_0 \leftarrow +, x_0, y_0$
$b \leftarrow +, x, y$	$b \leftarrow a$	$b_0 \leftarrow +, x_0, y_0$	$b_0 \leftarrow a_0$
$a \leftarrow 17$	$a \leftarrow 17$	$a_1 \leftarrow 17$	$a_1 \leftarrow 17$
$c \leftarrow +, x, y$	$c \leftarrow +, x, y$	$c_0 \leftarrow +, x_0, y_0$	$c_0 \leftarrow a_0$

6.4 Восстановление кода из SSA-формы

6.4.2. Замена ϕ -функций группами инструкций копирования

- ◇ Можно оставить *SSA*-имена неизменными, заменив каждую ϕ -функцию группой команд копирования (по одной для каждого входного ребра), предоставив разбираться с именами оптимизирующему преобразованию «Распространение копий». В рассматриваемом примере при удалении ϕ -функций будет:

```
B7  a4 ← φ(a2, a3)
      b4 ← φ(b2, b3)
      c6 ← φ(c3, c5)
      d6 ← φ(d2, d5)
      y  ← +, a4, b4
      z  ← +, c6, d6
      i2 ← +, i1, 1
```

```
B6  c5 ← φ(c2, c4)
      d5 ← φ(d4, d3)
      b3 ←
```

```
B2  b2 ←
      c3 ←
      d2 ←
```

```
B7  y  ← +, a4, b4
      z  ← +, c6, d6
      i2 ← +, i1, 1
```

```
B6  b3 ←
      a4 ← a2
      b4 ← b2
      c6 ← c3
      d6 ← d2
```

```
B2  b2 ←
      c3 ←
      d2 ←
      a4 ← a3
      b4 ← b3
      c6 ← c5
      d6 ← d5
```

6.4 Восстановление кода из SSA-формы

6.4.2. Замена ϕ -функций группами инструкций копирования

B_7 $a_4 \leftarrow \phi(a_2, a_3)$
 $b_4 \leftarrow \phi(b_2, b_3)$
 $c_6 \leftarrow \phi(c_3, c_5)$
 $d_6 \leftarrow \phi(d_2, d_5)$
 $y \leftarrow +, a_4, b_4$
 $z \leftarrow +, c_6, d_6$
 $i_2 \leftarrow +, i_1, 1$

B_6 $c_5 \leftarrow \phi(c_2, c_4)$
 $d_5 \leftarrow \phi(d_4, d_3)$
 $b_3 \leftarrow$

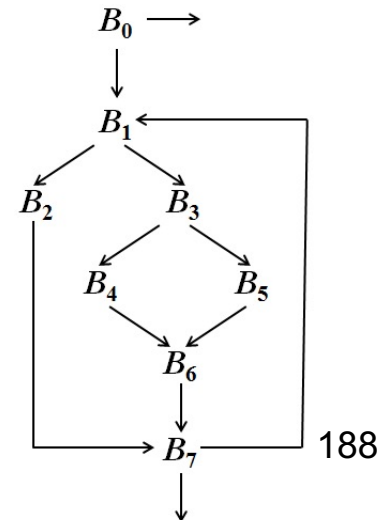
B_2 $b_2 \leftarrow$
 $c_3 \leftarrow$
 $d_2 \leftarrow$

B_7 $y \leftarrow +, a_4, b_4$
 $z \leftarrow +, c_6, d_6$
 $i_2 \leftarrow +, i_1, 1$

B_6 $b_3 \leftarrow$
 $a_4 \leftarrow a_2$
 $b_4 \leftarrow b_2$
 $c_6 \leftarrow c_3$
 $d_6 \leftarrow d_2$

B_2 $b_2 \leftarrow$
 $c_3 \leftarrow$
 $d_2 \leftarrow$
 $a_4 \leftarrow a_3$
 $b_4 \leftarrow b_3$
 $c_6 \leftarrow c_5$
 $d_6 \leftarrow d_5$

- ◇ Удаление ϕ -функций из блока B_6 породит инструкции копирования в блоках B_4 и B_5 .
- ◇ Удаление ϕ -функций из блока B_1 должно породить инструкции копирования в блоках B_0 и B_7 . Но этого делать нельзя!



6.4 Восстановление кода из SSA-формы

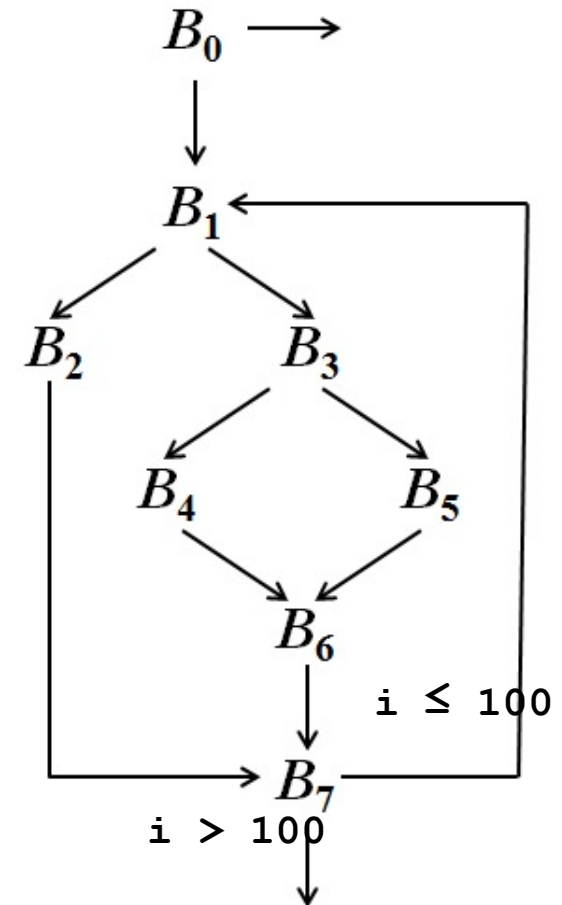
6.4.2. Замена Φ -функций группами инструкций копирования

- ◇ Удаление ϕ -функций из блока B_1 должно породить инструкции копирования в блоках B_0 и B_7 .

Но этого делать нельзя! Почему?

- ◇ У блоков B_0 и B_7 по два последующих блока (вторые блоки не попали на схему, так как они находятся вне анализируемого цикла).

Если вставить копирования в блоки B_0 и B_7 , они будут выполняться не только на ребрах, внутри рассматриваемого цикла, но и на ребрах выводящих из цикла.



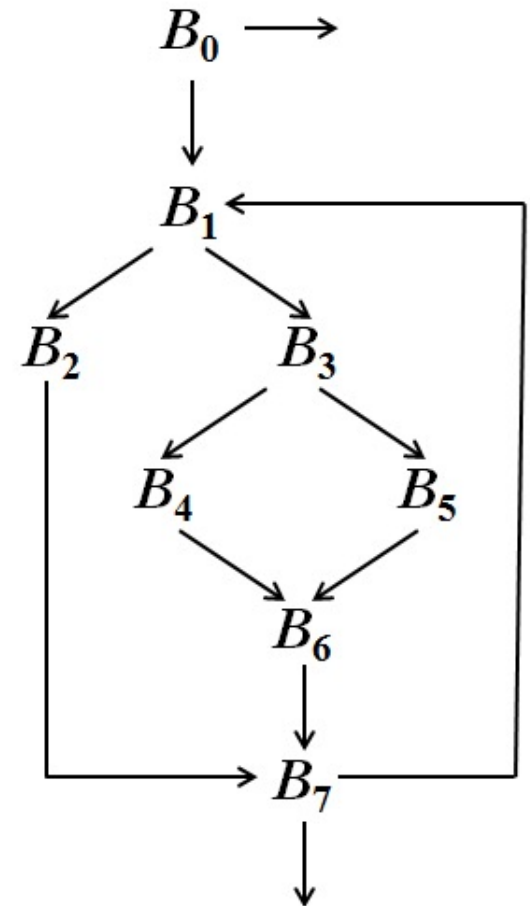
6.4 Восстановление кода из SSA-формы

6.4.2. Замена Φ -функций группами инструкций копирования

- ◇ По определению ребро, выходящее из вершины с несколькими потомками, и входящее в вершину с несколькими предками, называется *критическим*.

Ребра $(B_0 B_1)$ и $(B_7 B_1)$ – критические.

- ◇ Критические ребра обычно исключают, разрывая их, и вставляя в разрыв дополнительные вершины.
- ◇ Разорвем критические ребра и добавим два новых блока B_8 и B_9 , поместив в них требуемые инструкции копирования.



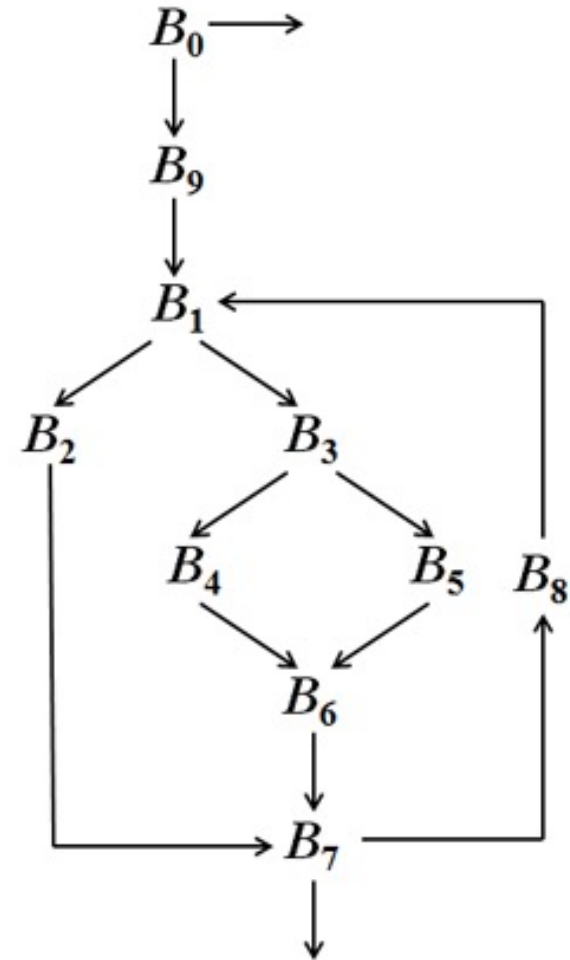
6.4 Восстановление кода из SSA-формы

6.4.2. Замена Φ -функций группами инструкций копирования

- По определению ребро, выходящее из вершины с несколькими потомками, и входящее в вершину с несколькими предками, называется *критическим*.

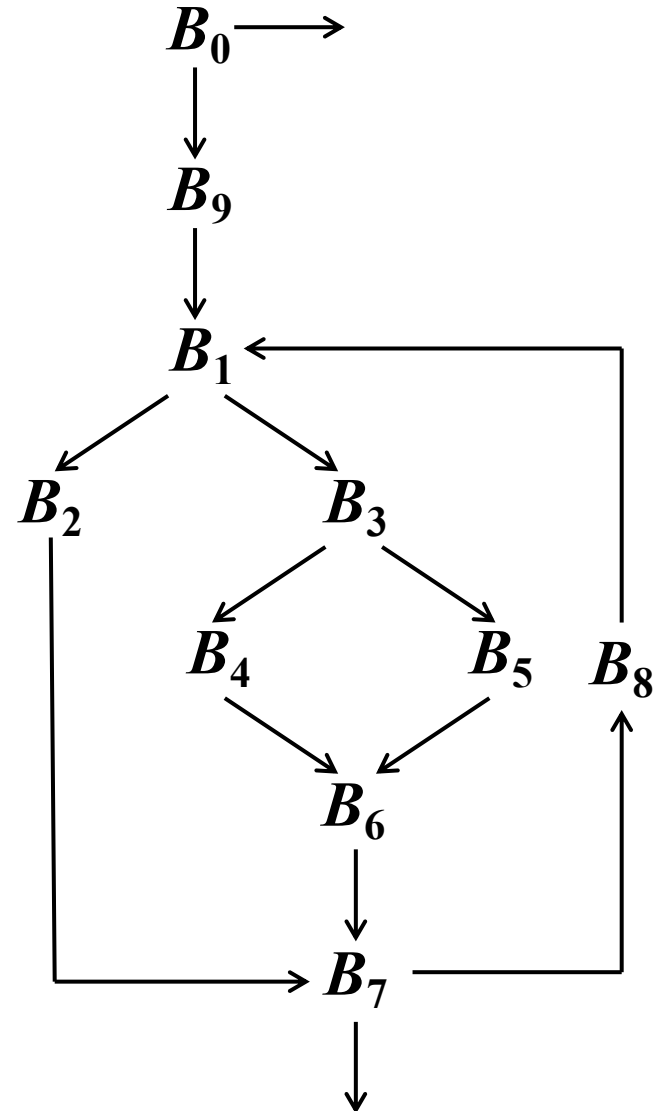
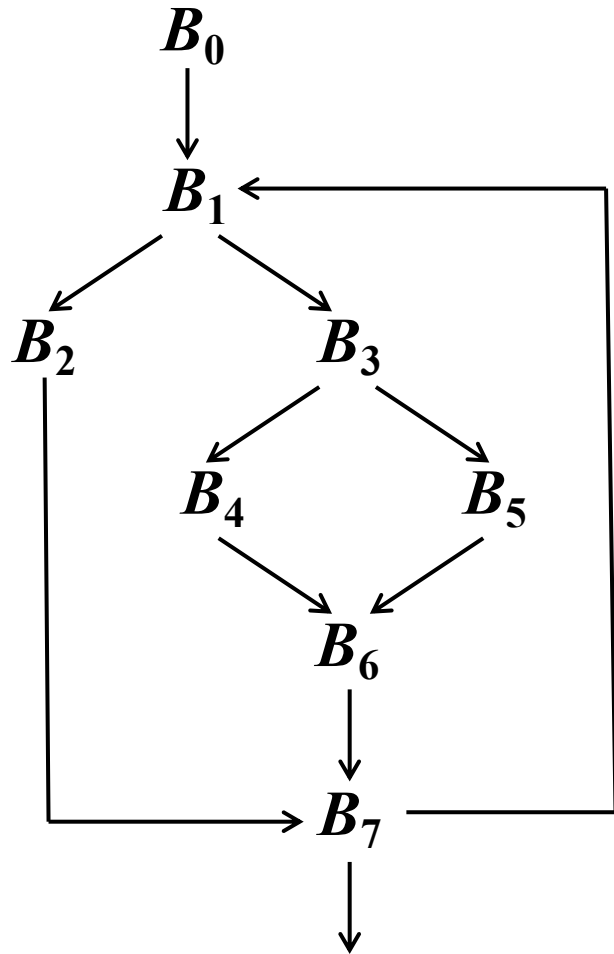
Ребра $(B_0 B_1)$ и $(B_7 B_1)$ – критические.

- Критические ребра обычно исключают, разрывая их, и вставляя в разрыв дополнительные вершины.
- Разорвем критические ребра и добавим два новых блока B_8 и B_9 , поместив в них требуемые инструкции копирования.



6.4 Восстановление кода из SSA-формы

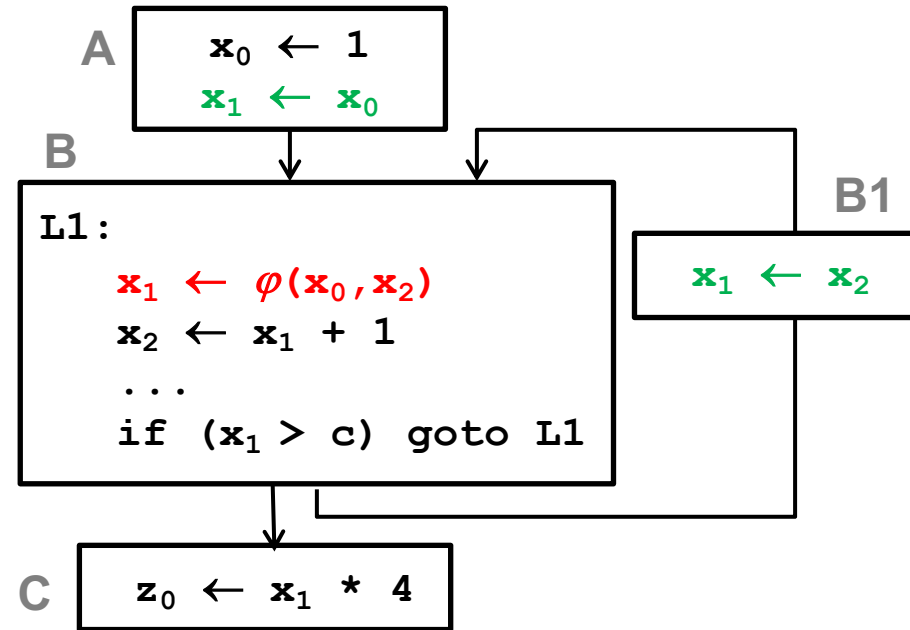
6.4.2. Замена Φ -функций группами инструкций копирования



6.4 Восстановление кода из SSA-формы

6.4.2. Замена ϕ -функций группами инструкций копирования

При удалении ϕ -функции из блока (**B**), в который входит критическая дуга, операция копирования должна быть добавлена в новом блоке (**B1**), созданном на той дуге, по которой приходит соответствующий аргумент. Добавлять копирование $x_1 \leftarrow x_2$ просто в конец предка, из которого исходит дуга (**B**), некорректно:



- Копирование из **B1** нельзя вставить в конец **B** после `if`, т.к. базовый блок должен заканчиваться переходом. Кроме того, в блоке **C** используется значение x_1 , которое копирование перезапишет.
- Копирование из **B1** нельзя вставить в конец **B** перед `if`, т.к. x_1 читается в условии перехода.

В данном примере в `if` и **C** используется не самое последнее определение x (x_1 а не x_2). Такой код не встречается в только что построенном SSA, но может появиться позже в результате оптимизаций.