1. Введение.

Базовые блоки. Локальная оптимизация.

1.1 Компиляторы

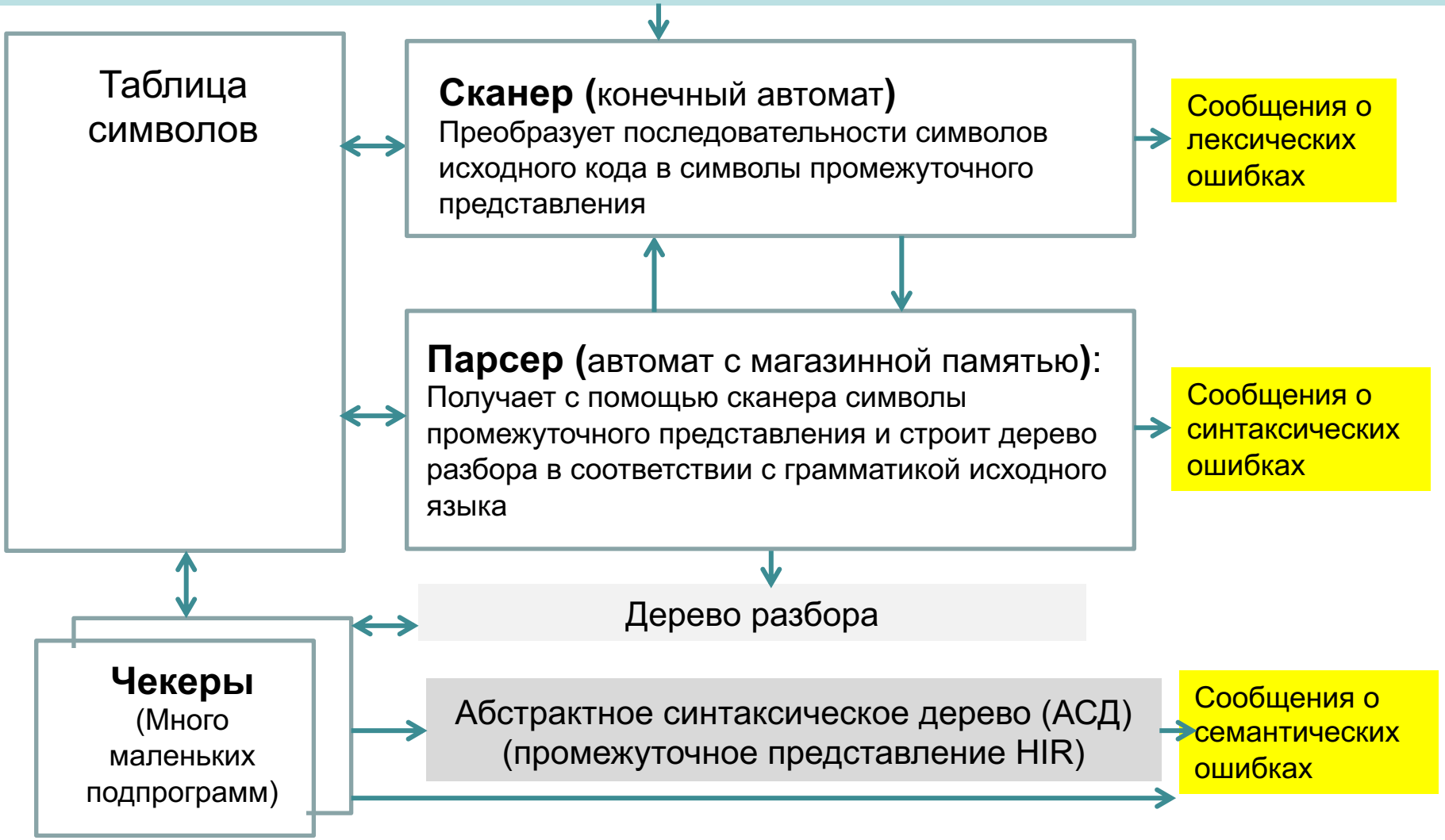
1.1.1. Неоптимизирующий компилятор

Структура неоптимизирующего компилятора

1.1 Компиляторы

1.1.2. Синтаксически управляемая трансляция

Исходный код – текст модуля компиляции в виде строки символов

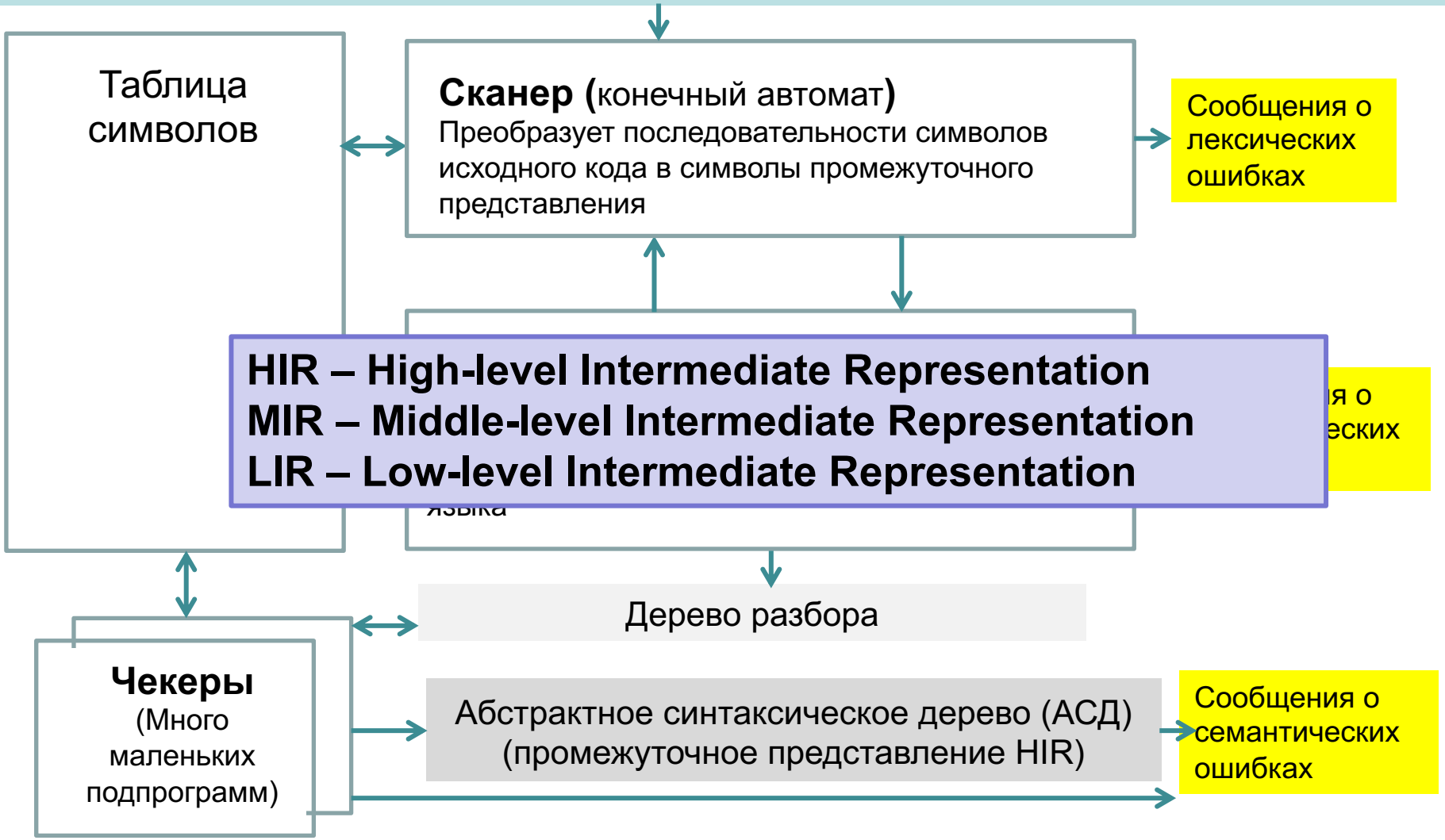


Каждый чекер вычисляет атрибут узлов дерева разбора и проверяет соответствуют ли атрибуты родительских и дочерних узлов; после чего либо строит соответствующую часть АСД, либо выдает сообщение об ошибке

1.1 Компиляторы

1.1.2. Синтаксически управляемая трансляция

Исходный код – текст модуля компиляции в виде строки символов



Каждый чекер вычисляет атрибут узлов дерева разбора и проверяет соответствуют ли атрибуты родительских и дочерних узлов; после чего либо строит соответствующую часть АСД, либо выдает сообщение об ошибке

1.1.2. Синтаксически управляемая трансляция

◊ По АСД (HIR) можно восстановить исходную программу, так как для каждого оператора исходной программы существует узел АСД. Промежуточные представления MIR и LIR, которые будут использоваться в данном курсе таким свойством не обладают даже в том случае, когда оптимизация была отключена.

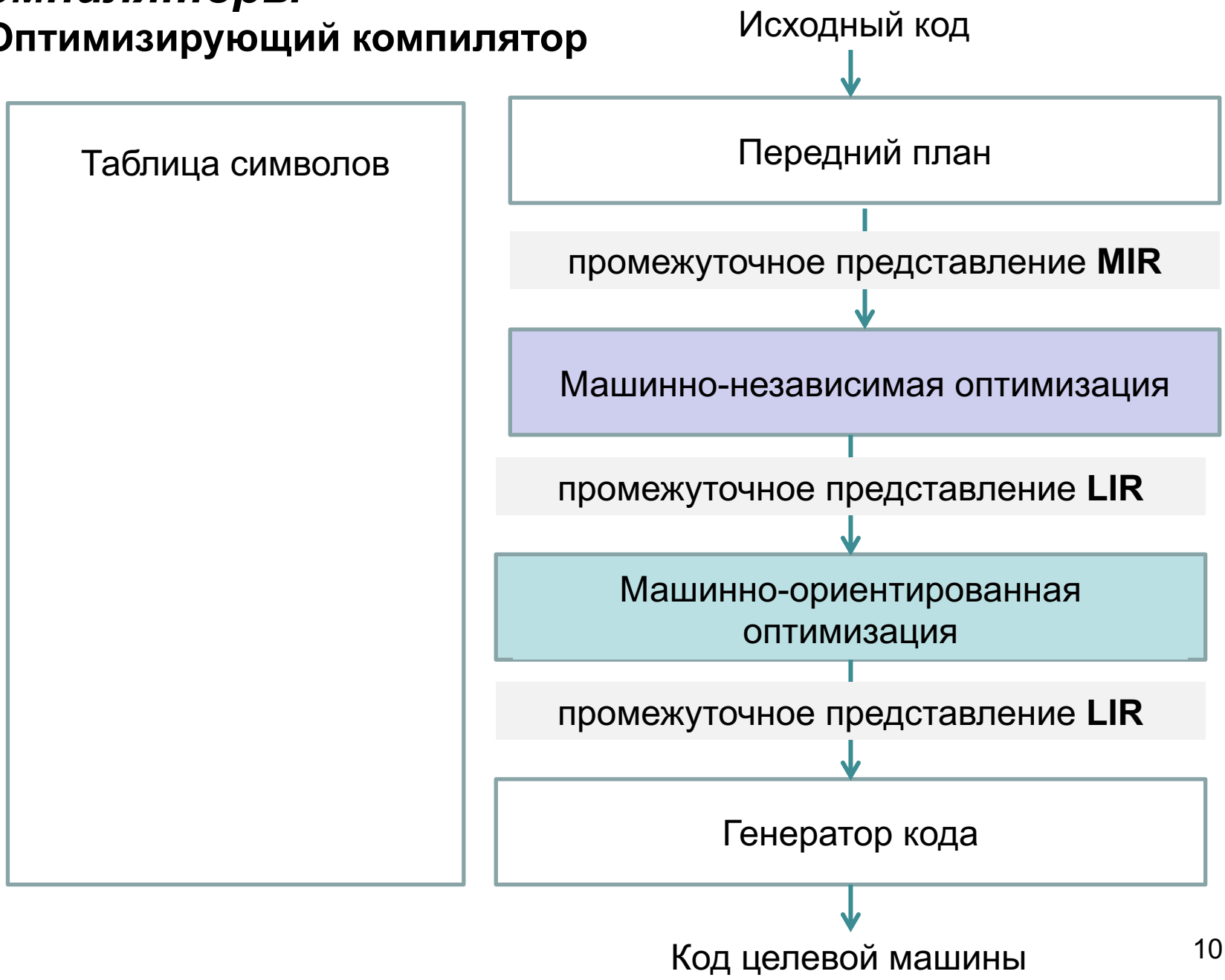
◊ Передний и задний планы компилятора изучаются в курсе «Конструирование компиляторов», а также в таких курса как «Теория алгоритмических языков», «Теория автоматов» и др.
Учебное пособие по этим темам:

В. А. Серебряков, М. П. Галочкин. **Теория и реализация языков программирования,** учебное пособие, 2-е изд., доп. и испр.
Москва : МЗ Пресс, 2006. - 350 с.

В Интернете оно есть в формате pdf.
http://trpl7.ru/t-books/_TRYAPBOOK_pdf.pdf

1.1 Компиляторы

1.1.3. Оптимизирующий компилятор



1.2 Цель курса

◊ Цель курса
- изучение методов оптимизации программ (как машинно-независимой, так и машинно-ориентированной)
- изучение принципов конструирования оптимизирующих компиляторов.

◊ Основное учебное пособие:
А. В. Ахо, М. С. Лам, Р. Сети, Дж. Д. Ульман. Компиляторы: принципы, технологии и инструменты, 2-е издание.
М.: «И.Д. Вильямс», 2008
(в конце 2014 года был выпущен дополнительный тираж)

◊ В тех случаях, когда излагаемый материал в указанном учебном пособии отсутствует, используется еще одно учебное пособие:

Keith D. Cooper, Linda Torczon. Engineering a Compiler (Second Edition) Elsevier, Inc. 2012 (на русский оно пока не переведено)

либо статьи из журналов (как правило, на английском языке).

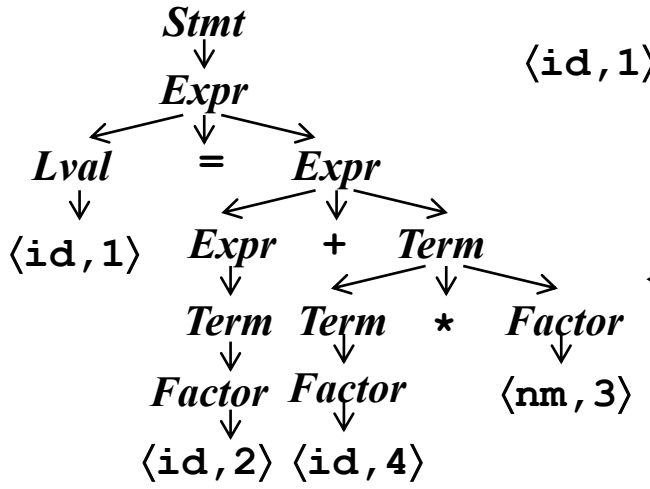
1.3 Генерация промежуточного представления MIR

`current_pos=initial_pos+16*scale` Строка исходной программы

Анализатор лексики

⟨id,1⟩⟨=⟩⟨id,2⟩⟨+⟩⟨nm,3⟩⟨*⟩⟨id,4⟩ Последовательность токенов

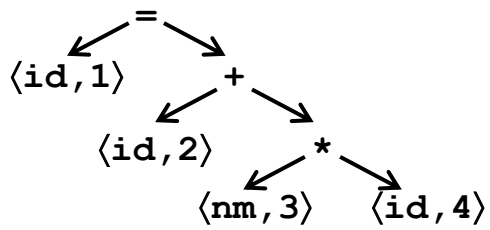
Анализатор синтаксиса



Дерево разбора

Дерево разбора

Анализатор семантики



Абстрактное синтаксическое дерево (**HIR**)

Генератор промежуточного представления

```
t1  ← *,nm3,id4
t2  ← +,id2,t1
id1 ← t2
```

MIR

Таблица символов

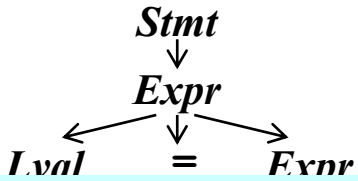
Внутрен-нее имя	Внешнее имя	Атрибуты
id1	current_pos	int
id2	initial_pos	int
nm3	16	int
id4	step	int

1.3 Генерация промежуточного представления MIR

current_pos=initial_pos+16*scale Строка исходной программы

Анализатор лексики

⟨**id,1**⟩⟨**=**⟩⟨**id,2**⟩⟨**+**⟩⟨**nm,3**⟩⟨***⟩⟨**id,4**⟩ Последовательность токенов



Анализатор синтаксиса

Почему для оптимизации необходимо промежуточное представление MIR? Потому что промежуточное представление HIR – это одна из форм линеаризованного представления дерева: скобочная структура или один из видов польской записи. И в том, и в другом случае трудно представить оптимизируемую программу в виде последовательности инструкций. Например, в инверсной польской записи строка исходной программы **current_pos=initial_pos+16*scale** будет иметь вид ⟨**=**⟩⟨**id,1**⟩⟨**+**⟩⟨**id,2**⟩⟨***⟩⟨**nm,3**⟩⟨**id,4**⟩, причем интерпретироваться она будет с правого, а не с левого конца.

id2	initial_pos	int
nm3	16	int
id4	step	int

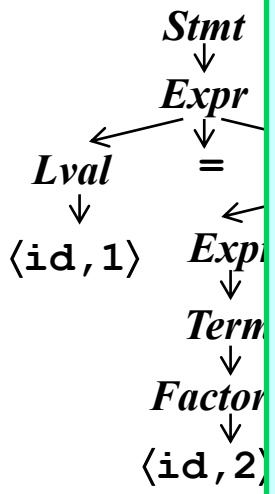
представления

```
t1  ← *,nm3,id4
t2  ← +,id2,t1
id1 ← t2
```

MIR

1.3 Генерация промежуточного представления MIR

`current_pos=initial_pos+16*scale` Строка исходной программы



ость токенов

збора

таксическое
во
)

Промежуточное представление **MIR** включает

- «четверки» (трехадресные инструкции),
- таблицу символов

Пример «четверки»: `*, t1, nm3, id4`
или `t1 ← *, nm3, id4`

Генератор промежуточного представления

Таблица символ

Внутрен-нее имя	Вн	
id1	current_pos	int
id2	initial_pos	int
nm3	16	int
id4	step	int

MIR

```
t1  ← *,nm3,id4
t2  ← +,id2,t1
id1 ← t2
```

1.4. Промежуточное представление MIR

1.4.1. Определение промежуточного представления MIR

◊ Инструкции присваивания (**x**, **y** и **z** – имена (адреса) переменных или константы):	
x ← op, y, z	выполнить бинарную операцию **op** над операндами **y** и **z** и поместить результат в **x**
x ← op, y	выполнить унарную операцию **op** над операндом **y** и поместить результат в **x**
x ← y	скопировать значение переменной **y** в переменную **x**
x[i] ← y	поместить значение **y** в **i**-ю по отношению к **x** ячейку памяти
x ← y[i]	поместить значение **i**-ой по отношению к **y** ячейки памяти в **x**

Не хватает указателей и адресной арифметики
(т.н. L-выражений в терминологии языка C):
x[i] – простейший частный случай L-выражения

1.4. Промежуточное представление MIR

1.4.1. Определение промежуточного представления MIR

◊ Инструкции перехода:	
`goto L`	Безусловный переход: следующей будет выполнена инструкция с меткой L
`ifTrue x goto L`	Условный переход: если x истинно, следующей будет выполнена инструкция с меткой L
`ifFalse x goto L`	Условный переход: если x ложно, следующей будет выполнена инструкция с меткой L

На рисунке справа – схема трансляции оператора **`if (expr) stmt1;`**

По аналогии можно составить схемы трансляции операторов **`else`**, **`while`**, **`for`**, **`switch`** и других операторов управления

	Инструкции внутреннего представления 2, вычисляющие значение выражения **`expr`** и присваивающие его **`x`**
	`ifFalse x goto L`
	Инструкции внутреннего представления 2, соответствующие оператору **`stmt1`**
`L:`	Инструкции внутреннего представления 2, соответствующие оператору, следующему за оператором **`if`**

1.4. Промежуточное представление MIR

1.4.1. Определение промежуточного представления MIR

◊ Процедуры:	
`param x`	Передача фактического параметра вызываемой процедуре (если вызываемая процедура имеет n параметров, то инструкции ее вызова предшествует n инструкций $param$)
`call p, n`	Вызов процедуры p, имеющей n параметров
`return y`	Возврат из процедуры y – возвращаемое значение
`param_decl x`	Объявление формального параметра внутри функции, является определением x для анализа достигающих переменных внутри функции

Замечание. Это промежуточное представление – учебное и не содержит некоторых деталей, важных для реализации.

Соответствующее промежуточное представление **MIR** компилятора *GCC* называется *Gimple,* В Clang/LLVM – *LLVM IR*.

1.4. Промежуточное представление MIR

1.4.2. Пример программы в промежуточном представлении MIR

```
{
int i, j;
int v, x;
if (n <= m) return;
/* Начало фрагмента */
i = m - 1;
j = n;
v = a[n];
while (1) {
do i = i + 1;
while (a[i] < v);
do j = j - 1;
while (a[j] > v);
if (i >= j) break;
/* Обмен a[i], a[j] */
x = a[i];
a[i] = a[j];
a[j] = x;
}
/* Обмен a[i], a[n] */
x = a[i];
a[i] = a[n];
a[n] = x;
/* Конец фрагмента */
quicksort(a,m,j); quicksort(a,i+1,n);
}
```

(1)	i ← -, m, 1	(16)	t7 ← *, 4, i
(2)	j ← n	(17)	t8 ← *, 4, j
(3)	t1 ← *, 4, n	(18)	t9 ← a[t8]
(4)	v ← a[t1]	(19)	a[t7] ← t9
(5)	L1: i ← +, i, 1	(20)	t10 ← *, 4, j
(6)	t2 ← *, 4, i	(21)	a[t10] ← x
(7)	t3 ← a[t2]	(22)	goto L1
(8)	ifTrue t3<v goto L1	(23)	L3: t11 ← *, 4, i
(9)	L2: j ← -, j, 1	(24)	x ← a[t11]
(10)	t4 ← *, 4, j	(25)	t12 ← *, 4, i
(11)	t5 ← a[t4]	(26)	t13 ← *, 4, n
(12)	ifTrue t5>v goto L2	(27)	t14 ← a[t13]
(13)	ifTrue i>=j goto L3	(28)	a[t12] ← t14
(14)	t6 ← *, 4, i	(29)	t15 ← *, 4, n
(15)	x ← a[t6]	(30)	a[t15] ← x

Слева – текст исходной Си-программы

1.5. Граф потока управления

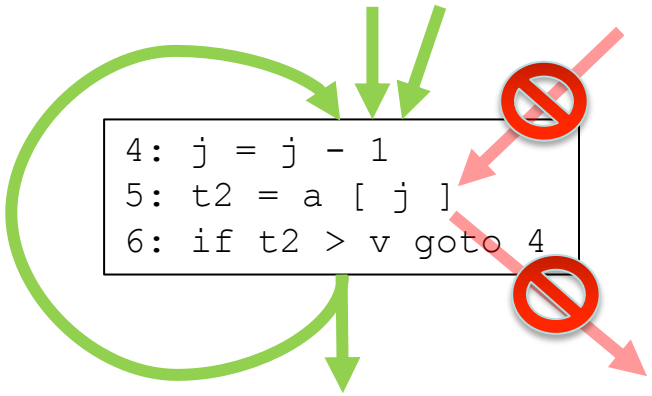
1.5.1. Базовый блок (определение)

◊ *Базовым блоком* (или *линейным участком*) называется последовательность следующих одна за другой инструкций **MIR**, со следующими свойствами:

(1) поток управления может входить в базовый блок только через его первую инструкцию, т.е. в программе нет переходов в средину базового блока;

(2) поток управления покидает базовый блок без останова или ветвления, за исключением, возможно, в последней инструкции базового блока.

◊ Пример базового блока:



Дуги, передающие управление в/из середины базового блока недопустимы (в таких случаях должны быть созданы отдельные базовые блоки)

1.5. Граф потока управления

1.5.1. Базовый блок (определение)

◊ *Базовым блоком* (или *линейным участком*) называется последовательность следующих одна за другой инструкций **MIR**, со следующими свойствами:

(1) поток управления может входить в базовый блок только через его первую инструкцию, т.е. в программе нет переходов в средину базового блока;

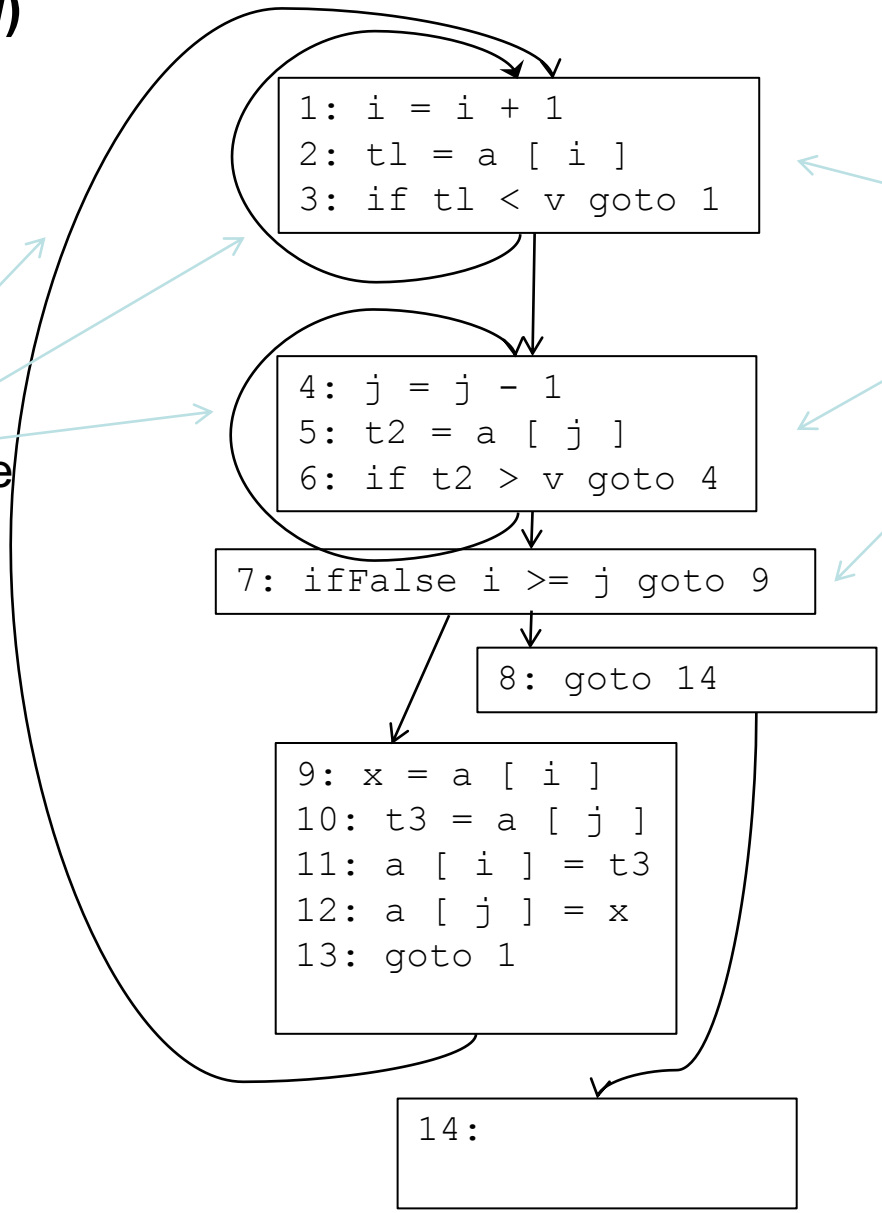
(2) поток управления покидает базовый блок без останова или ветвления, за исключением, возможно, в последней инструкции базового блока.

◊ Пример базового блока – инструкции (14) – (22) из примера 1.4.2

```
(14) t6 ← *, 4, i
(15) x ← a[t6]
(16) t7 ← *, 4, i
(17) t8 ← *, 4, j
(18) t9 ← a[t8]
(19) a[t7] ← t9
(20) t10 ← *, 4, j
(21) a[t10] ← x
(22) goto L1
```

1.5. Граф потока управления

1.5.1.1. Пример графа потока управления (*для другой исходной программы*)



Базовые блоки

Циклы в графе потока управления соответствуют циклам в программе (циклы могут быть вложенными)

Граф потока управления / Control Flow Graph (CFG)

1.5. Граф потока управления

1.5.2. Граф потока управления (определение)

◊ Вершинами графа потока управления являются базовые блоки, а дуги соединяют выход из вершины со входом в вершину, которая может выполняться следующей

В частности, если последней инструкцией базового блока является инструкция условного перехода, то из него будет выходить две дуги Если первая инструкция базового блока имеет метку, то в этот базовый блок будут входить дуги из всех базовых блоков, последняя инструкция которых будет инструкцией условного или безусловного перехода на эту метку

На первом курсе, изучая программирование, иногда строят граф потока управления программы, которую собираются составить, считая, что этот граф упрощает составление программы. Для некоторых небольших программ это действительно может помочь, но для больших программ, граф потока управления может иметь несколько сотен вершин, так что построить его, не имея текста программы, нереально

1.5. Граф потока управления

1.5.3. Алгоритм построения графа потока управления (ГПУ)

◊ **Вход**: последовательность трехадресных инструкций.

◊ **Выход**: список базовых блоков для данной последовательности инструкций, такой что каждая инструкция принадлежит только одному базовому блоку.

◊ **Метод**:

- Строится упорядоченное множество НББ (начал базовых блоков):
  - (a) первая инструкция программы
  - (b) помеченная инструкция программы (напр. с меткой **L1:**)
  - (c) инструкция, следующая за инструкцией перехода
- Каждому НББ соответствует ББ, который определяется как последовательность инструкций, содержащая само НББ и все инструкции до следующего НББ (не включая его) или до конца последовательности инструкций.
- Строится множество дуг графа потока управления:
  - если последняя инструкция ББ не является инструкцией перехода, строится дуга, соединяющая ББ со следующим ББ;
  - если последняя инструкция ББ является инструкцией безусловного перехода, строится дуга, соединяющая ББ с ББ, НББ которого имеет соответствующую метку;
  - если последняя инструкция ББ является инструкцией условного перехода, строятся обе дуги.

1.5. Граф потока управления

1.5.4. Пример

◊ Применим алгоритм 1.5.3 для построения ГПУ программы из примера 1.4.2.

(1)	i ← -, m, 1	(16)	t7 ← *, 4, i
(2)	j ← n	(17)	t8 ← *, 4, j
(3)	tl ← *, 4, n	(18)	t9 ← a[t8]
(4)	v ← a[tl]	(19)	a[t7] ← t9
(5)	L1: i ← +, i, 1	(20)	t10 ← *, 4, j
(6)	t2 ← *, 4, i	(21)	a[t10] ← x
(7)	t3 ← a[t2]	(22)	goto L1
(8)	ifTrue t3<v goto L1	(23)	L3: t11 ← *, 4, i
(9)	L2: j ← -, j, 1	(24)	x ←a[t11]
(10)	t4 ← *, 4, j	(25)	t12 ← *, 4, i
(11)	t5 ← a[t4]	(26)	t13 ← *, 4, n
(12)	ifTrue t5>v goto L2	(27)	t14 ← a[t13]
(13)	ifTrue i>=j goto L3	(28)	a[t12] ← tl4
(14)	t6 ← *, 4, j	(29)	t15 ← *, 4, n
(15)	x ← a[t6]	(30)	a[t15] ← x

1.5. Граф потока управления

1.5.4. Пример

◊ Найдем начала базовых блоков (НББ)

(1)	`i ← -, m, 1`	(16)	`t7 ← *, 4, i`
(2)	`j ← n`	(17)	`t8 ← *, 4, j`
(3)	`t1 ← *, 4, n`	(18)	`t9 ← a[t8]`
(4)	`v ← a[t1]`	(19)	`a[t7] ← t9`
(5)			
(6)			
(7)			
(8)			
(9)	`L2: j ← -, j, 1`	(24)	`x ←a[t11]`
(10)	`t4 ← *, 4, j`	(25)	`t12 ← *, 4, i`
(11)	`t5 ← a[t4]`	(26)	`t13 ← *, 4, n`
(12)	`ifTrue t5>v goto L2`	(27)	`t14 ← a[t13]`
(13)	`ifTrue i>=j goto L3`	(28)	`a[t12] ← t14`
(14)	`t6 ← *, 4, j`	(29)	`t15 ← *, 4, n`
(15)	`x ← a[t6]`	(30)	`a[t15] ← x`

НББ по определению это:

(a) первая инструкция программы

(b) помеченная инструкция программы (напр. с меткой **L1:**)

(c) инструкция, следующая за инструкцией перехода

1.5. Граф потока управления

1.5.4. Пример

◊ Найдем НББ

(*a*) первая инструкция программы
(*b*) помеченная инструкция программы (напр. с меткой **L1:**)
(*c*) инструкция, следующая за инструкцией перехода

(1)	`i ← -, m, 1` НББ (*a*)	(16)	`t7 ← *, 4, i`
(2)	`j ← n`	(17)	`t8 ← *, 4, j`
(3)	`tl ← *, 4, n`	(18)	`t9 ← a[t8]`
(4)	`v ← a[tl]`	(19)	`a[t7] ← t9`
(5)	`L1: i ← +, i, 1` НББ (*b*)	(20)	`t10 ← *, 4, j`
(6)	`t2 ← *, 4, i`	(21)	`a[t10] ← x`
(7)	`t3 ← a[t2]`	(22)	`goto L1`
(8)	`ifTrue t3<v goto L1`	(23)	`L3: t11 ← *, 4, i` НББ (*b*) (*c*)
(9)	`L2: j ← -, j, 1` НББ (*b*) (*c*)	(24)	`x ←a[t11]`
(10)	`t4 ← *, 4, j`	(25)	`t12 ← *, 4, i`
(11)	`t5 ← a[t4]`	(26)	`t13 ← *, 4, n`
(12)	`ifTrue t5>v goto L2`	(27)	`t14 ← a[t13]`
(13)	`ifTrue i>=j goto L3` НББ (*c*)	(28)	`a[t12] ← tl4`
(14)	`t6 ← *, 4, j` НББ (*c*)	(29)	`t15 ← *, 4, n`
(15)	`x ← a[t6]`	(30)	`a[t15] ← x`

1.5. Граф потока управления

1.5.4. Пример

◊ Найдем НББ

(*a*) первая инструкция программы
(*b*) помеченная инструкция программы (напр. с меткой **L1:**)
(*c*) инструкция, следующая за инструкцией перехода

(1)	`i ← -, m, 1` НББ (*a*)	(16)	`t7 ← *, 4, i`
(2)	`j ← n`	(17)	`t8 ← *, 4, j`
(3)	`tl ← *, 4, n`	(18)	`t9 ← a[t8]`
(4)	`v ← a[tl]`	(19)	`a[t7] ← t9`
(5)	`L1: i ← +, i, 1` НББ (*b*)	(20)	`t10 ← *, 4, j`
(6)	`t2 ← *, 4, i`	(21)	`a[t10] ← x`
(7)	`t3 ← a[t2]`	(22)	`goto L1`
(8)	`ifTrue t3<v goto L1`	(23)	`L3: t11 ← *, 4, i` НББ (*b*) (*c*)
(9)	`L2: j ← -, j, 1` НББ (*b*) (*c*)	(24)	`x ←a[t11]`
(10)	`t4 ← *, 4, j`	(25)	`t1`
(11)	`t5 ← a[t4]`	(26)	`t1`
(12)	`ifTrue t5>v goto L2`	(27)	`t1`
(13)	`ifTrue i>=j goto L3` НББ (*c*)	(28)	`a`
(14)	`t6 ← *, 4, j` НББ (*c*)	(29)	`t1`
(15)	`x ← a[t6]`	(30)	`a[t15] ← x`

Началами базовых блоков являются инструкции с номерами: (1), (5), (9). (13), (14), (23). Алгоритм 1.5.3 позволяет построить следующие ББ

1.5. Граф потока управления

1.5.4. Пример

Блок A

```
(1)     i ← -, m, 1
(2)     j ← n
(3)     t1 ← *, 4, n
(4)     v ← a[t1]
```

Блок B

```
(5) L1:  i ← +, i, 1
(6)      t2 ← *, 4, i
(7)      t3 ← a[t2]
(8)   ifTrue t3<v gotoL1
```

Блок C

```
(9)  L2:  j ← -, j, 1
(10)      t4 ← *, 4, j
(11)      t5 ← a[t4]
(12) ifTrue t5>v goto L2
```

Блок D

```
(13) ifTrue i>=j goto L3
```

Блок E

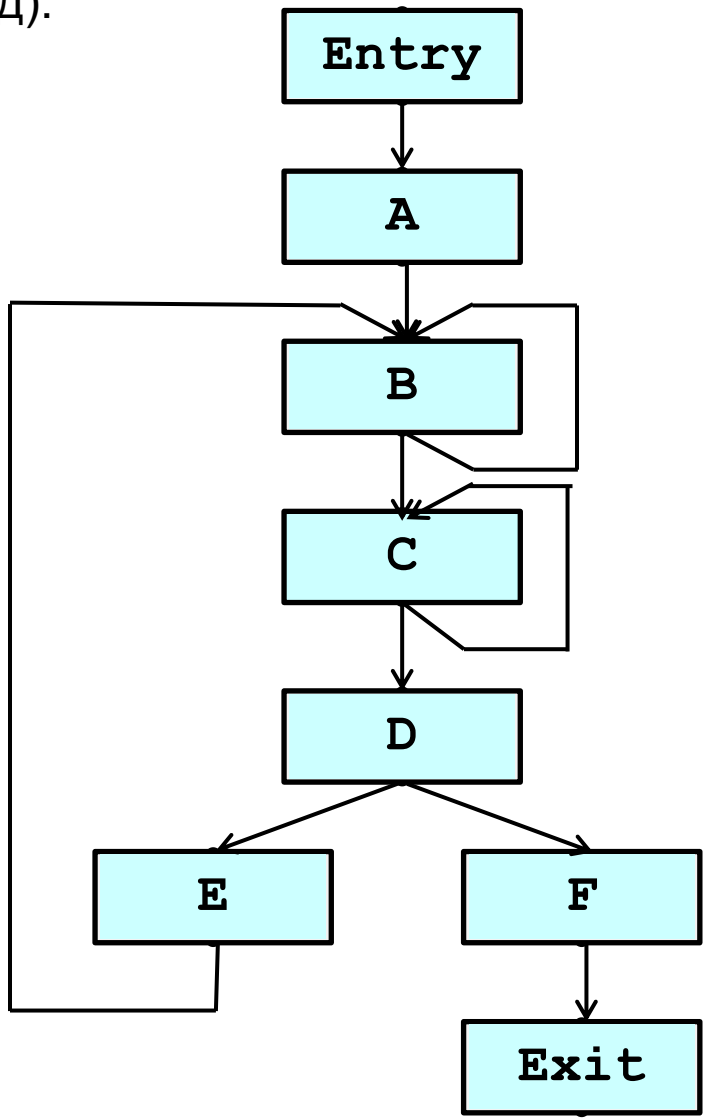
```
(14)   t6 ← *, 4, i
(15)   x ← a[t6]
(16)   t7 ← *, 4, i
(17)   t8 ← *, 4, j
(18)   t9 ← a[t8]
(19)   a[t7] ← t9
(20)   t10← *, 4, j
(21)   a[t10] ← x
(22)   goto L1
```

Блок F

```
(23) L3: t11 ← *, 4, i
(24)     x ← a[t11]
(25)     t12 ← *, 4, i
(26)     t13 ← *, 4, n
(27)     t14 ← a[t13]
(28)     a[t12] ← t14
(29)     t15 ← *, 4, n
(30)     a[t15] ← x
```

1.5. Граф потока управления

1.5.4. Пример. Построив дуги согласно алгоритму 1.5.3, получим следующий ГПУ. Для удобства анализа к ГПУ добавлены два дополнительных пустых узла: *entry* (вход) и *exit* (выход).



1.6 Локальная оптимизация (*оптимизация базовых блоков*)

1.6.1 Постановка задачи локальной оптимизации

◊ **Определение**. В контексте задач локальной оптимизации, будем считать, что базовый блок, помимо инструкций программы, имеет также два множества *Input* и *Output,* т.е. представляет собой тройку

$$B = \langle P, Input, Output \rangle,$$

где P – последовательность инструкций,

Input – множество переменных, определенных до входа в блок B,

Output – множество переменных, используемых после выхода из блока B.

◊ Оптимизация – это выполнение в ББ следующих преобразований:

- Удаление *общих подвыражений* (инструкций, повторно вычисляющих уже вычисленные значения).
- Удаление *мертвого кода* (инструкций, вычисляющих значения, которые впоследствии не используются).
- *Сворачивание констант*.
- Изменение порядка инструкций, там, где это возможно, чтобы сократить время хранения временного значения на регистре.

1.6 Локальная оптимизация (*оптимизация базовых блоков*)

1.6.1 Постановка задачи локальной оптимизации

◊ **Определение**. В контексте задач локальной оптимизации, будем считать, что базовый блок, помимо инструкций программы, имеет также два множества *Input* и *Output,* т.е. представляет собой тройку

$$B = \langle P, Input, Output \rangle,$$

Все перечисленные преобразования можно выполнить за один просмотр ББ, представив его в виде *ориентированного ациклического графа* (ОАГ).

◊ Оптимизация – это выполнение в ББ следующих преобразований:

- ◈ Удаление *общих подвыражений* (инструкций, повторно вычисляющих уже вычисленные значения).
- ◈ Удаление *мертвого кода* (инструкций, вычисляющих значения, которые впоследствии не используются).
- ◈ *Сворачивание констант*.
- ◈ Изменение порядка инструкций, там, где это возможно, чтобы сократить время хранения временного значения на регистре.

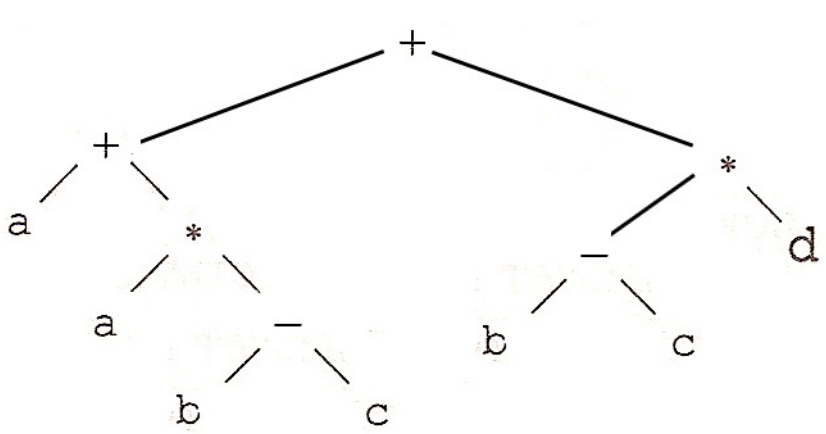
1.6 Локальная оптимизация (*оптимизация базовых блоков*)

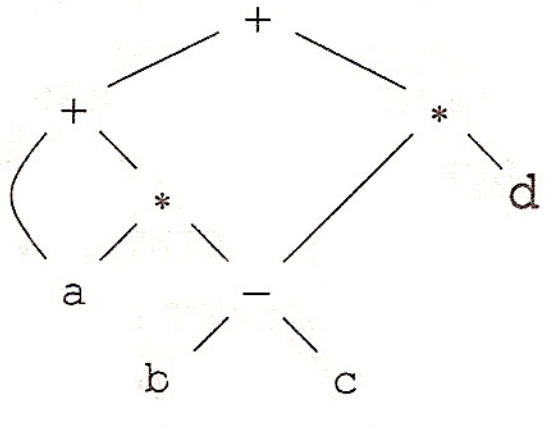
1.6.2 Представление базового блока в виде ориентированного ациклического графа

◊ **Простейший пример**. Выражение

```
a + a * (b-c) + (b-c) * d
```

можно представить в виде фрагмента АСД (левый рисунок) либо в виде ориентированного ациклического графа (правый рисунок) – *DAG* (*Directed Acyclic Graph*)





◊ Основное отличие DAG от АСД в том, что в DAG каждое значение представляется только один раз: узлы, представляющие в АСД одинаковые значения, «склеиваются»

1.6 Локальная оптимизация

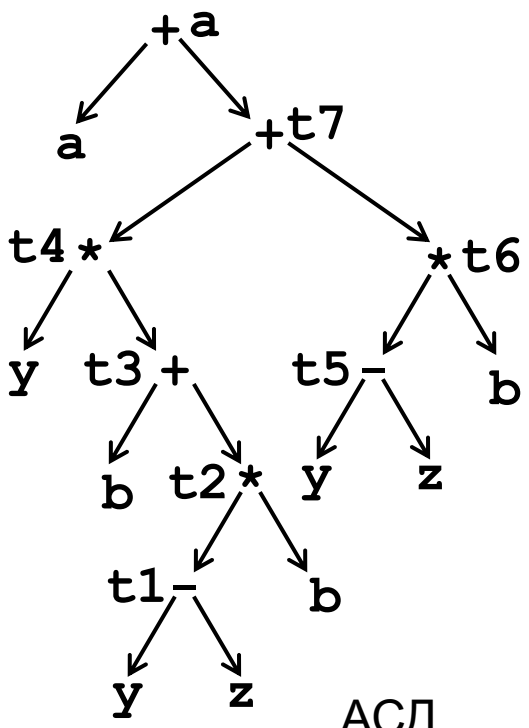
1.6.2 Представление базового блока в виде ориентированного ациклического графа

◊ **Пример**. Выражение в исходном коде:

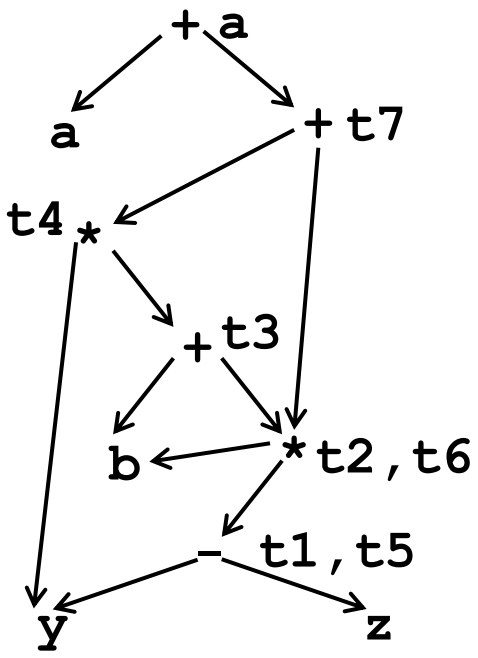
```
a = a + y*(b +(y-z)*b)+(y-z)* b
```

```
t1 ← -, y, z
t2 ← *, t1, b
t3 ← +, b, t2
t4 ← *, y, t3
t5 ← -, y, z
t6 ← *, t5, b
t7 ← +, t4, t6
a  ← +, a, t7
```

Выражение в промежуточном представлении 2



АСД



ОАГ

1.6 Локальная оптимизация

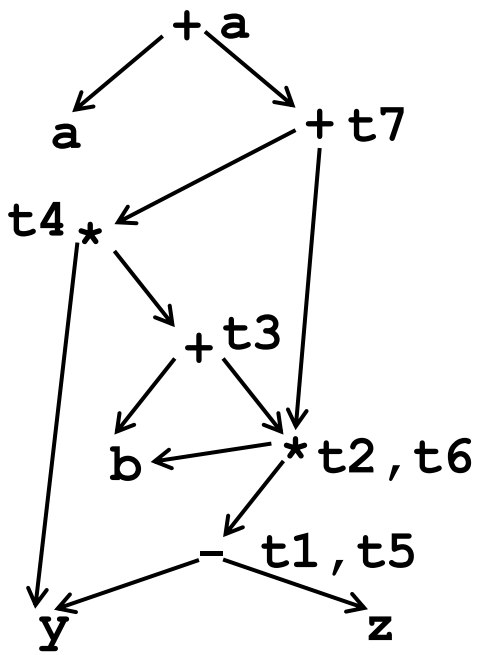
1.6.2 Представление базового блока в виде ориентированного ациклического графа

◊ **Пример (окончание).**

На ОАГ для выражения видно (в отличие от АСД):

- переменная **b** используется в двух вычислениях
- выражение **y-z** можно вычислить только один раз
- выражение **(y-z)* b** можно вычислить только один раз

```
t1 ← -, y, z
t2 ← *, t1, b
t3 ← +, b, t2
t4 ← *, y, t3
t5 ← -, y, z
t6 ← *, t5, b
t7 ← +, t4, t6
a  ← +, a, t7
```



ОАГ

1.6.3 Метод нумерации значений – алгоритм построения ОАГ

Представим *ОАГ* в виде хеш-таблицы

- ◊ Каждая строка хеш-таблицы представляет один узел ОАГ.
- ◊ Первое поле каждой записи представляет собой код операции
- ◊ Каждой правой части операции

  ⟨**op, left, right**⟩,

  где **op** – код операции, а **left** и **right** – левый и правый операнды, соответствует ее *сигнатура*

  ⟨**op, #left, #right**⟩,

  где **op** – код операции, а **#left** и **#right** – номера значений левого и правого операндов (у унарных операций **#right** равен 0).
- ◊ Унарные операции **id** и **nm** определяют соответственно имена переменных и константы (листовые узлы).
- ◊ Номер значения – это номер строки таблицы значений (ТЗ), в которой определяется это значение
- ◊ Для определения номера строки ТЗ обычно используют хеш-функцию, аргументом которой является сигнатура

1.6.4. Упрощенный алгоритм построения ОАГ для базового блока B

Алгоритм (на псевдокоде) построения ОАГ для базового блока B, содержащего n инструкций вида $t_i \leftarrow Op_i, l_i, r_i$.

Функция **#val(s)** определяет номер значения, определяемого сигнатурой

s = (Op, #val(l), #val(r)) .

```
for each "t_i ← Op_i, l_i, r_i" do
   s_i = (Op_i, #val(l_i), #val(r_i))
   if (ТЗ содержит s_j == s_i)
      then
            вернуть j в качестве номера значения #val(s_i)
      else
            завести в ТЗ новую строку ТЗ_k
            записать сигнатуру s_i в строку ТЗ_k
            вернуть k в качестве номера значения #val(s_i)
```

1.6.4. Упрощенный алгоритм построения ОАГ для базового блока B

Алгоритм (на псевдокоде) построения ОАГ для базового блока B, содержащего n инструкций вида $t_i \leftarrow Op_i, l_i, r_i$.

Функция **#val(s)** определяет номер значения, определяемого сигнатурой

s = (Op, #val(l), #val(r)) .

```
for ea
    s_i =
    if (ТЗ содержит s_j == s_i)
       then
            вернуть j в качестве номера значения #val(s_i)
       else
            завести в ТЗ новую строку ТЗ_k
            записать сигнатуру s_i в строку ТЗ_k
            вернуть k в качестве номера значения #val(s_i)
```

Подробный алгоритм построения таблицы значений, удаления общих подвыражений вместе со сверткой констант, а также удаления мертвого кода приведен в конце лекции (см. 1.8.1-1.8.2)

1.6 Локальная оптимизация

1.6.3 Представление *ОАГ* в виде таблицы значений

◊ Таблица значений рассматриваемого примера имеет вид:

$t1^5 \leftarrow -, y^3, z^4$
$t2^6 \leftarrow *, t1^5, b^2$
$t3^7 \leftarrow +, b^2, t2^6$
$t4^8 \leftarrow *, y^3, t3^7$
$t5^5 \leftarrow -, y^3, z^4$
$t6^6 \leftarrow *, t5^5, b^2$
$t7^9 \leftarrow +, t4^8, t6^6$
$a^{10} \leftarrow +, a^1, t7^9$

1	id	ссылка в ТС		a
2	id	ссылка в ТС		b
3	id	ссылка в ТС		y
4	id	ссылка в ТС		z
5	-	3	4	t1, t5
6	*	5	2	t2, t6
7	+	2	6	t3
8	*	3	7	t4
9	+	8	6	t7
10	+	1	9	a
# значения	КОП	# операнда	# операнда	Присоединенные переменные
	Определение значения (сигнатура)			

1.6 Локальная оптимизация

1.6.4.1. Пример

```
t1 ← -, y, z
t2 ← *, t1, b
t3 ← +, b, t2
t4 ← *, y, t3
t5 ← -, y, z
t6 ← *, t5, b
t7 ← +, t4, t6
a  ← +, a, t7
```

(a) Блок E
до оптимизации

```
t1^5 ← -, y^3, z^4
t2^6 ← *, t1^5, b^2
t3^7 ← +, b^2, t2^6
t4^8 ← *, y^3, t3^7
t5^5 ← -, y^3, z^4
t6^6 ← *, t5^5, b^2
t7^9 ← +, t4^8,t6^6
a^10 ← +, a^1, t7^9
```

(c) Блок E
после нумерации значений

```
t1 ← -, y, z
t2 ← *, t1, b
t3 ← +, b, t2
t4 ← *, y, t3
t5 ← t1
t6 ← t2
t7 ← +, t4, t6
a  ← +, a, t7
```

(c) Блок E
после оптимизации

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0  ← *, i,  c
i   ← +, t0, b
c   ← −, c,  #2
j   ← *, i,  c
t1  ← −, a,  #1
t2  ← +, t0, b
t3  ← *, t2, c
j   ← −, t3, t2
t2  ← *, i,  c
t3  ← −, t0, #1
t3  ← −, t3, t2
t3  ← +, c,  b
t1  ← −, j,  i
ifTrue j < #256 goto L1
```

LiveNow								
a	b	c	i	j	t0	t1	t2	t3

Таблица значений				
#значения	КОП	#операнда	#операнда	Присоединенные переменные

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0  ← *, i5,  c3
i   ← +, t0, b
c   ← −, c,  #2
j   ← *, i,  c
t1  ← −, a,  #1
t2  ← +, t0, b
t3  ← *, t2, c
j   ← −, t3, t2
t2  ← *, i,  c
t3  ← −, t0, #1
t3  ← −, t3, t2
t3  ← +, c,  b
t1  ← −, j,  i
ifTrue j < #256 goto L1
```

LiveNow								
a	b	c	i	j	t0	t1	t2	t3
1	2	3	5					

Таблица значений				
#значения	КОП	#операнда	#операнда	Присоединенные переменные
1	id			*a*
2	id			*b*
3	id			*c*
4	nm			#0
5	id			*i*
6	nm			#2
7	nm			#1

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0[8] ← *, i[5],  c[3]
i     ← +, t0,  b
c     ← −, c,   #2
j     ← *, i,   c
t1    ← −, a,   #1
t2    ← +, t0,  b
t3    ← *, t2,  c
j     ← −, t3,  t2
t2    ← *, i,   c
t3    ← −, t0,  #1
t3    ← −, t3,  t2
t3    ← +, c,   b
t1    ← −, j,   i
ifTrue j < #256 goto L1
```

LiveNow								
a	**b**	**c**	**i**	**j**	**t0**	**t1**	**t2**	**t3**
1	2	3	5		8			

Таблица значений				
#значения	**КОП**	**#операнда**	**#операнда**	**Присоединенные переменные**
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0[8]  ← *, i[5],  c[3]
i[9]   ← +, t0[8], b[2]
c      ← −, c,    #2
j      ← *, i,    c
t1     ← −, a,    #1
t2     ← +, t0,   b
t3     ← *, t2,   c
j      ← −, t3,   t2
t2     ← *, i,    c
t3     ← −, t0,   #1
t3     ← −, t3,   t2
t3     ← +, c,    b
t1     ← −, j,    i
ifTrue j < #256 goto L1
```

LiveNow								
a	b	c	i	j	t0	t1	t2	t3
1	2	3	5		8			
			9					

Таблица значений				
#значения	КОП	#операнда	#операнда	Присоединенные переменные
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	i

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0⁸  ← *, i⁵,  c³
i⁹   ← +, t0⁸, b²
c¹⁰  ← −, c³,  #2⁶
j    ← *, i,   c
t1   ← −, a,   #1
t2   ← +, t0,  b
t3   ← *, t2,  c
j    ← −, t3,  t2
t2   ← *, i,   c
t3   ← −, t0,  #1
t3   ← −, t3,  t2
t3   ← +, c,   b
t1   ← −, j,   i
ifTrue j < #256 goto L1
```

LiveNow								
a	b	c	i	j	t0	t1	t2	t3
1	2	3	5		8			
		10	9					

Таблица значений				
#значения	КОП	#операнда	#операнда	Присоединенные переменные
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	i
10	−	3	6	c

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0⁸  ← *, i⁵,  c³
i⁹   ← +, t0⁸, b²
c¹⁰  ← −, c³,  #2⁶
j¹¹  ← *, i⁹,  c¹⁰
t1   ← −, a,   #1
t2   ← +, t0,  b
t3   ← *, t2,  c
j    ← −, t3,  t2
t2   ← *, i,   c
t3   ← −, t0,  #1
t3   ← −, t3,  t2
t3   ← +, c,   b
t1   ← −, j,   i
ifTrue j < #256 goto L1
```

LiveNow								
a	**b**	**c**	**i**	**j**	**t0**	**t1**	**t2**	**t3**
1	2	3	5	11	8			
		10	9					

Таблица значений				
#значения	**КОП**	**#операнда**	**#операнда**	**Присоединенные переменные**
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	i
10	−	3	6	c
11	*	9	10	j

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0[8]  ← *, i[5],  c[3]
i[9]   ← +, t0[8], b[2]
c[10]  ← −, c[3],  #2[6]
j[11]  ← *, i[9],  c[10]
t1[12] ← −, a[1],  #1[7]
t2     ← +, t0, b
t3     ← *, t2, c
j      ← −, t3, t2
t2     ← *, i,  c
t3     ← −, t0, #1
t3     ← −, t3, t2
t3     ← +, c,  b
t1     ← −, j,  i
ifTrue j < #256 goto L1
```

LiveNow								
a	**b**	**c**	**i**	**j**	**t0**	**t1**	**t2**	**t3**
1	2	3	5	11	8	12		
		10	9					

Таблица значений				
#значения	**КОП**	**#операнда**	**#операнда**	**Присоединенные переменные**
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	i
10	−	3	6	c
11	*	9	10	j
12	−	1	7	$t1$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0⁸  ← *, i⁵,  c³
i⁹   ← +, t0⁸, b²
c¹⁰  ← −, c³,  #2⁶
j¹¹  ← *, i⁹,  c¹⁰
t1¹² ← −, a¹,  #1⁷
t2   ← +, t0⁸, b²
t3   ← *, t2,  c
j    ← −, t3,  t2
t2   ← *, i,   c
t3   ← −, t0,  #1
t3   ← −, t3,  t2
t3   ← +, c,   b
t1   ← −, j,   i
ifTrue j < #256 goto L1
```

LiveNow								
a	**b**	**c**	**i**	**j**	**t0**	**t1**	**t2**	**t3**
1	2	3	5	11	8	12		
		10	9					

Таблица значений				
#значения	**КОП**	**#операнда**	**#операнда**	**Присоединенные переменные**
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	i
10	−	3	6	c
11	*	9	10	j
12	−	1	7	$t1$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0^8  ← *, i^5,  c^3
i^9   ← +, t0^8, b^2
c^10  ← −, c^3,  #2^6
j^11  ← *, i^9,  c^10
t1^12 ← −, a^1,  #1^7
t2^9  ← i^9
t3    ← *, t2, c
j     ← −, t3, t2
t2    ← *, i,  c
t3    ← −, t0, #1
t3    ← −, t3, t2
t3    ← +, c,  b
t1    ← −, j,  i
ifTrue j < #256 goto L1
```

LiveNow								
a	**b**	**c**	**i**	**j**	**t0**	**t1**	**t2**	**t3**
1	2	3	5	11	8	12	9	
		10	9					

Таблица значений				
#значения	**КОП**	**#операнда**	**#операнда**	**Присоединенные переменные**
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	j
12	−	1	7	$t1$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0[8]  ← *, i[5],  c[3]
i[9]   ← +, t0[8], b[2]
c[10]  ← −, c[3],  #2[6]
j[11]  ← *, i[9],  c[10]
t1[12] ← −, a[1],  #1[7]
t2[9]  ← i[9]
t3[11] ← *, t2[9], c[10]
j      ← −, t3, t2
t2     ← *, i,  c
t3     ← −, t0, #1
t3     ← −, t3, t2
t3     ← +, c,  b
t1     ← −, j,  i
ifTrue j < #256 goto L1
```

LiveNow								
a	**b**	**c**	**i**	**j**	**t0**	**t1**	**t2**	**t3**
1	2	3	5	11	8	12	9	
		10	9					

Таблица значений				
#значения	**КОП**	**#операнда**	**#операнда**	**Присоединенные переменные**
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	j
12	−	1	7	$t1$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0^8  ← *, i^5,  c^3
i^9   ← +, t0^8, b^2
c^10  ← −, c^3,  #2^6
j^11  ← *, i^9,  c^10
t1^12 ← −, a^1,  #1^7
t2^9  ← i^9
t3^11 ← j^11
j     ← −, t3, t2
t2    ← *, i,  c
t3    ← −, t0, #1
t3    ← −, t3, t2
t3    ← +, c,  b
t1    ← −, j,  i
ifTrue j < #256 goto L1
```

LiveNow								
a	b	c	i	j	t0	t1	t2	t3
1	2	3	5	11	8	12	9	11
		10	9					

Таблица значений				
#значения	КОП	#операнда	#операнда	Присоединенные переменные
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3$
12	−	1	7	$t1$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0⁸  ← *, i⁵,  c³
i⁹   ← +, t0⁸, b²
c¹⁰  ← −, c³,  #2⁶
j¹¹  ← *, i⁹,  c¹⁰
t1¹² ← −, a¹,  #1⁷
t2⁹  ← i⁹
t3¹¹ ← j¹¹
j¹³  ← −, j¹¹, i⁹
t2¹¹ ← j¹¹
t3¹⁴ ← −, t0⁸, #1⁷
t3¹⁵ ← −, t3¹⁴, j¹¹
t3¹⁶ ← +, c¹⁰,  b²
t1¹⁷ ← −, j¹³,  i⁹
ifTrue j¹³ < #256 goto L1
```

LiveNow								
a	**b**	**c**	**i**	**j**	**t0**	**t1**	**t2**	**t3**
1	2	3	5	11	8	12	9	11
		10	9	13		17	11	14
								15
								16

Таблица значений				
#значения	**КОП**	**#операнда**	**#операнда**	**Присоединенные переменные**
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3, t2$
12	−	1	7	$t1$
13	−	11	9	j
14	−	8	7	$t3$
15	−	14	11	$t3$
16	+	10	2	$t3$
17	−	13	9	$t1$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0^8  ← *, i^5,  c^3
i^9   ← +, t0^8, b^2
c^10  ← −, c^3,  #2^6
j^11  ← *, i^9,  c^10
t1^12 ← −, a^1,  #1^7
t2^9  ← i^9
t3^11 ← j^11
j^13  ← −, j^11, i^9
t2^11 ← j^11
t3^14 ← −, t0^8, #1^7
t3^15 ← −, t3^14, j^11
t3^16 ← +, c^10,  b^2
t1^17 ← −, j^13,  i^9
ifTrue j^13 < #256 goto L1
```

LiveNow								
a	**b**	**c**	**i**	**j**	**t0**	**t1**	**t2**	**t3**
1	2	3	5	11	8	12	9	11
		10	9	13		17	11	14
								15
								16

Таблица значений				
#значения	**КОП**	**#операнда**	**#операнда**	**Присоединенные переменные**
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3, t2$
12	−	1	7	$t1$
13	−	11	9	j
14	−	8	7	$t3$
15	−	14	11	$t3$
16	+	10	2	$t3$
17	−	13	9	$t1$

$$Output_B = \{a, b, c, i, j, t0\}$$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0^8  ← *, i^5,  c^3
i^9   ← +, t0^8, b^2
c^10  ← −, c^3,  #2^6
j^11  ← *, i^9,  c^10
t1^12 ← −, a^1,  #1^7
t2^9  ← i^9
t3^11 ← j^11
j^13  ← −, j^11, i^9
t2^11 ← j^11
t3^14 ← −, t0^8, #1^7
t3^15 ← −, t3^14, j^11
t3^16 ← +, c^10,  b^2
t1^17 ← −, j^13,  i^9
ifTrue j^13 < #256 goto L1
```

LiveNow								
a	b	c	i	j	t0	t1	t2	t3
1	2	3	5	11	8	12	9	11
		10	9	13		17	11	14
								15
								16

Таблица значений				
#значения	КОП	#операнда	#операнда	Присоединенные переменные
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3, t2$
12	−	1	7	$t1$
13	−	11	9	j
14	−	8	7	$t3$
15	−	14	11	$t3$
16	+	10	2	$t3$
17	−	13	9	$t1$

Пусть известны живые переменные в конце базового блока: $Output_B = \{a, b, c, i, j, t0\}$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0⁸  ← *, i⁵,  c³
i⁹   ← +, t0⁸, b²
c¹⁰  ← −, c³,  #2⁶
j¹¹  ← *, i⁹,  c¹⁰
t1¹² ← −, a¹,  #1⁷
t2⁹  ← i⁹
t3¹¹ ← j¹¹
j¹³  ← −, j¹¹, i⁹
t2¹¹ ← j¹¹
t3¹⁴ ← −, t0⁸, #1⁷
t3¹⁵ ← −, t3¹⁴, j¹¹
t3¹⁶ ← +, c¹⁰,  b²
t1¹⁷ ← −, j¹³,  i⁹
ifTrue j¹³ < #256 goto L1
```

LiveNow								
a	**b**	**c**	**i**	**j**	**t0**	**t1**	**t2**	**t3**
1	2	3	5	11	8	12	9	11
		10	9	13		17	11	14
								15
								16

Таблица значений				
#значения	**КОП**	**#операнда**	**#операнда**	**Присоединенные переменные**
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3, t2$
12	−	1	7	$t1$
13	−	11	9	j
14	−	8	7	$t3$
15	−	14	11	$t3$
16	+	10	2	$t3$
17	−	13	9	$t1$

$$Output_B = \{a^1, b^2, c^{10}, i^9, j^{13}, t0^8\}$$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0⁸ ← *, i⁵,  c³
i⁹  ← +, t0⁸, b²
c¹⁰ ← −, c³,  #2⁶
j¹¹ ← *, i⁹,  c¹⁰
t1¹² ← −, a¹,  #1⁷
t2⁹ ← i⁹
t3¹¹ ← j¹¹
j¹³ ← −, j¹¹, i⁹
t2¹¹ ← j¹¹
t3¹⁴ ← −, t0⁸, #1⁷
t3¹⁵ ← −, t3¹⁴, j¹¹
t3¹⁶ ← +, c¹⁰,  b²
t1¹⁷ ← −, j¹³,  i⁹
ifTrue j¹³ < #256 goto L1
```

инструкцию $t2^9 \leftarrow i^9$ не помечаем полезной, т.к. $t2 \notin Output_B$

Таблица значений				
#значения	КОП	#операнда	#операнда	Присоединенные переменные
1	id			a
2	id			b
3	id			c
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3, t2$
12	−	1	7	$t1$
13	−	11	9	j
14	−	8	7	$t3$
15	−	14	11	$t3$
16	+	10	2	$t3$
17	−	13	9	$t1$

LiveNow								
a	b	c	i	j	t0	t1	t2	t3
1	2	3	5	11	8	12	9	11
		10	9	13		17	11	14
								15
								16

$$Output_B = \{a^1, b^2, c^{10}, i^9, j^{13}, t0^8\}$$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0^8  ← *, i^5,  c^3
i^9   ← +, t0^8, b^2
c^10  ← −, c^3,  #2^6
j^11  ← *, i^9,  c^10
t1^12 ← −, a^1,  #1^7
t2^9  ← i^9
t3^11 ← j^11
j^13  ← −, j^11, i^9
t2^11 ← j^11
t3^14 ← −, t0^8, #1^7
t3^15 ← −, t3^14, j^11
t3^16 ← +, c^10,  b^2
t1^17 ← −, j^13,  i^9
ifTrue j^13 < #256 goto L1
```

определения $t3^{11}$ и $t2^{11}$ не помечаем полезными, т.к. $t2, t3 \notin Output_B$

				ний
			р	Присоединенные переменные
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3, t2$
12	−	1	7	$t1$
13	−	11	9	j
14	−	8	7	$t3$
15	−	14	11	$t3$
16	+	10	2	$t3$
17	−	13	9	$t1$

LiveNow								
a	b	c	i	j	t0	t1	t2	t3
1	2	3	5	11	8	12	9	11
		10	9	13		17	11	14
								15
								16

$$Output_B = \{a^1, b^2, c^{10}, i^9, j^{13}, t0^8\}$$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0⁸  ← *, i⁵,  c³
i⁹   ← +, t0⁸, b²
c¹⁰  ← −, c³,  #2⁶
j¹¹  ← *, i⁹,  c¹⁰
t1¹² ← −, a¹,  #1⁷
t2⁹  ← i⁹
t3¹¹ ← j¹¹
j¹³  ← −, j¹¹, i⁹
t2¹¹ ← j¹¹
t3¹⁴ ← −, t0⁸, #1⁷
t3¹⁵ ← −, t3¹⁴, j¹¹
t3¹⁶ ← +, c¹⁰,  b²
t1¹⁷ ← −, j¹³,  i⁹
ifTrue j¹³ < #256 goto L1
```

LiveNow								
a	**b**	**c**	**i**	**j**	**t0**	**t1**	**t2**	**t3**
1	2	3	5	11	8	12	9	11
		10	9	13		17	11	14
								15
								16

Таблица значений				
#значения	**КОП**	**#операнда**	**#операнда**	**Присоединенные переменные**
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3, t2$
12	−	1	7	$t1$
13	−	11	9	j
14	−	8	7	$t3$
15	−	14	11	$t3$
16	+	10	2	$t3$
17	−	13	9	$t1$

$$Output_B = \{a^1, b^2, c^{10}, i^9, j^{13}, t0^8\}$$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0^8  ← *, i^5,  c^3
i^9   ← +, t0^8, b^2
c^10  ← −, c^3,  #2^6
j^11  ← *, i^9,  c^10


j^13  ← −, j^11, i^9
```

```
ifTrue j^13 < #256 goto L1
```

LiveNow								
a	b	c	i	j	t0	t1	t2	t3
1	2	3	5	11	8	12	9	11
		10	9	13		17	11	14
								15
								16

Таблица значений				
#значения	КОП	#операнда	#операнда	Присоединенные переменные
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3, t2$
12	−	1	7	$t1$
13	−	11	9	j
14	−	8	7	$t3$
15	−	14	11	$t3$
16	+	10	2	$t3$
17	−	13	9	$t1$

$$Output_B = \{a^1, b^2, c^{10}, i^9, j^{13}, t0^8\}$$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0   ← *, i5,  c3
i9   ← +, t0,  b2
c10  ← −, c3,  #26
j11  ← *, i9,  c10

j13  ← −, j11, i9

ifTrue j13 < #256 goto L1
```

Таблица значений				
#значения	КОП	#операнда	#операнда	Присоединенные переменные
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3, t2$
12	−	1	7	$t1$
13	−	11	9	j
14	−	8	7	$t3$
15	−	14	11	$t3$
16	+	10	2	$t3$
17	−	13	9	$t1$

LiveNow								
a	b	c	i	j	t0	t1	t2	t3
1	2	3	5	11	8	12	9	11
		10	9	13		17	11	14
								15
								16

$$Output_B = \{a^1, b^2, c^{10}, i^9, j^{13}, t0\}$$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0   ← *, i5,  c3
i9   ← +, t0,  b2
c10  ← −, c3,  #26
j11  ← *, i9,  c10

j13  ← −, j11, i9

ifTrue j13 < #256 goto L1
```

Таблица значений				
#значения	КОП	#операнда	#операнда	Присоединенные переменные
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3, t2$
12	−	1	7	$t1$
13	−	11	9	j
14	−	8	7	$t3$
15	−	14	11	$t3$
16	+	10	2	$t3$
17	−	13	9	$t1$

LiveNow								
a	b	c	i	j	t0	t1	t2	t3
1	2	3	5	11	8	12	9	11
		10	9	13		17	11	14
								15
								16

$$Out[B] = \{a^1, b^2, c^{10}, i^9, j^{13}, t0\}$$

1.6 Локальная оптимизация

1.6.4.2. Пример

```
t0   ← *, i5,  c3
i    ← +, t0,  b
c    ← −, c3,  #26
j11  ← *, i,   c

j    ← −, j11, i

ifTrue j < #256 goto L1
```

Таблица значений				
#значения	КОП	#операнда	#операнда	Присоединенные переменные
1	id			a
2	id			b
3	id			c
4	nm			#0
5	id			i
6	nm			#2
7	nm			#1
8	*	5	3	$t0$
9	+	8	2	$i, t2$
10	−	3	6	c
11	*	9	10	$j, t3, t2$
12	−	1	7	$t1$
13	−	11	9	j
14	−	8	7	$t3$
15	−	14	11	$t3$
16	+	10	2	$t3$
17	−	13	9	$t1$

LiveNow								
a	b	c	i	j	t0	t1	t2	t3
1	2	3	5	11	8	12	9	11
		10	9	13		17	11	14
								15
								16

$$Out[B] = \{a, b, c, i, j, t0\}$$

1.6 Локальная оптимизация

1.6.5 Представление *ОАГ* в виде таблицы значений. Пример

◊ Заполнить таблицу значений для базового блока **B**:

```
B = ⟨P, {x, y, z},
       {v, w, z}⟩
```

```
P = {
 1. x1  ← 31
 2. y2  ← 52
 3. t14 ← +, x1, y2
 4. t24 ← +, x1, y2
 5. w5  ← *, t14,t24
 6. t36 ← –, x1, y2
 7. t47 ← *, w5, x1
 8. v8  ← –, t47,z3
 9. t54 ← +, x1, y2
10. y9  ← +, t54,z3
11. x10 ← +, x1, y9
12. v11 ← +, x10,y9
13. z12 ← +, z3, y9
14. y13 ← *, x10,z12
15. x14 ← *, t36,t47}
```

Можно выполнить сворачивание констант

Общие подвыражения

# значения	КОП	# операнда	# операнда	Переменные
	Определение значения (сигнатура)			
1	nm	ссылка в ТС		3 x
2	nm	ссылка в ТС		5 y
3	id	ссылка в ТС		z
4	+	1	2	t1 t2 t5
5	*	4	4	w
6	–	1	2	t3
7	*	5	1	t4
8	–	7	3	v
9	+	4	3	y
11	+	1	9	x
10	+	10	9	v
12	+	3	9	z
13	*	10	12	y
14	*	6	7	x

1.6 Локальная оптимизация

1.6.6 Удаление общих подвыражений

◊ Общие подвыражения обнаруживаются при построении узлов *ОАГ* с помощью алгоритма 1.6.3 (нумерация значений).
Так, в примере 1.6.5 три раза вычисляется одно и то же значение № 4 (в инструкциях № 3, 4 и 9). Удаление общих подвыражений позволит заменить инструкции 4 и 9 на **`t2 ← t1`** и **`t5 ← t1.`**

1.6.7 Сворачивание констант

◊ *Сворачивание констант* заключается в вычислении констант в процессе компиляции и замене константных выражений их значениями.
Например, выражение 2 * 3.1415926 можно заменить значением 6.2831852.

В примере 1.6.5 значение №4 равно **`x + y`**, причем **`x = 3, y = 5`**.
Сворачивание константы позволит заменить присваивание
`t1 ← +, x, y` на присваивание **`t1 ← 8`**

1.6 Локальная оптимизация

1.6.8 Снижение стоимости вычислений

◊ Еще один класс алгебраических оптимизаций включает локальное *снижение стоимости вычислений,* т.е. заменяет более дорогие с операции более дешевыми (см. таблицу). Так в примере 1.6.5 инструкция № 5 могла бы быть написана как **w ← ↑, t1, #2** (↑ обозначает операцию возведения в степень), но, как следует из таблицы, операция умножения, использованная в инструкции, является более дешевой. Если программист использовал не умножение, а возведение в степень, оптимизатор исправил бы его оплошность.

Дорогие	Дешевые
x^2	x × x
2 × x	x + x
x/2	x × 0.5
x/2	x >> 1 *(если x - целое)*
x*2^n	x << n *(если x - целое)*

1.6 Локальная оптимизация

1.6.8.1 Более сложный пример снижения стоимости вычислений (алгоритм в данном курсе не рассматривается)

◊ **Вычислить X = X * 81 в две команды без использования MUL**

Решение:

x*81 = x*9*9 = (8*x + x)*9 = ((x << 3) + x)*9 =

= ((x << 3) + x)*8 + ((x << 3) + x) =

= ((x << 3) + x) << 3 + ((x << 3) + x) = (r1 << 3) + r1

В архитектуре ARM (32-bit) сдвиг аргумента (`asl`) может выполняться «бесплатно» вместе со сложением за 1 такт (при этом «честная» команда умножения `MUL` выполняется 5 тактов):

```
add r1, r0, r0 asl #3    // r1 = r0 * 9 = r0 + r0 * 8 = r0 + r0 << 3
add r2, r1, r1 asl #3    // r2 = r0 * 81 = r1 * 9
```

Другой вариант разложения: 81 = 5 * 16 + 1 = (((2 << 1) + 1) << 4) + 1

1.6 Локальная оптимизация
1.6.9 Удаление мертвого кода

◊ Преобразование *ОАГ*, соответствующее удалению мертвого кода, состоит в удалении из *ОАГ* любого корня (**Op**), с которым не связаны живые переменные.

◊ *Живыми* называются переменные, значения которых, вычисленные в рассматриваемом базовом блоке, используются в других базовых блоках

◊ Для полноценного удаления мертвого кода необходимо уточнить определение базового блока.

◊ **Определение**. Базовым блоком называется тройка

$$B = \langle P, Input, Output \rangle,$$

где P – последовательность инструкций блока B,
$Input$ – множество переменных, определенных до входа в блок B,
$Output$ – множество переменных, используемых после выхода из блока B.

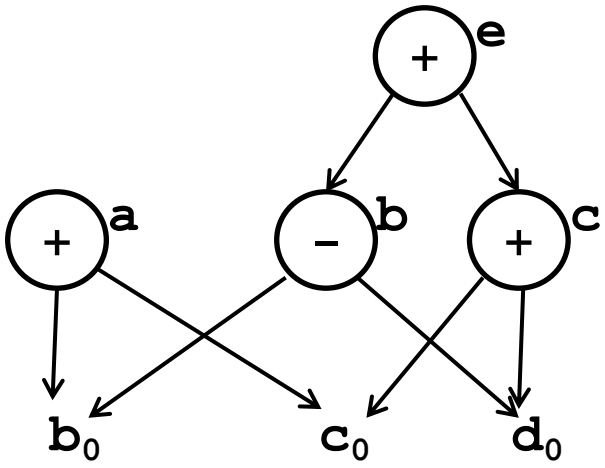
1.6 Локальная оптимизация
1.6.9 Удаление мертвого кода

Вычисление множеств **Input** и **Output** – отдельная задача, которая находится за пределами локальной оптимизации.
В множество Output входят переменные, значение которых используется вне данного базового блока.

◊ **Пример**. Рассмотрим базовый блок $B = \langle P, \{a, b, c, d\}, \{a, b\} \rangle$

```
a ← +, b, c
b ← -, b, d
c ← +, c, d
e ← +, b, c
```

Базовый блок B до удаления мертвого кода



ОАГ базового блока B

```
a ← +, b, c
b ← -, b, d
```

Новые значения **c** и **e** можно не вычислять, так как **c** и **e** не входят в множество *Output*

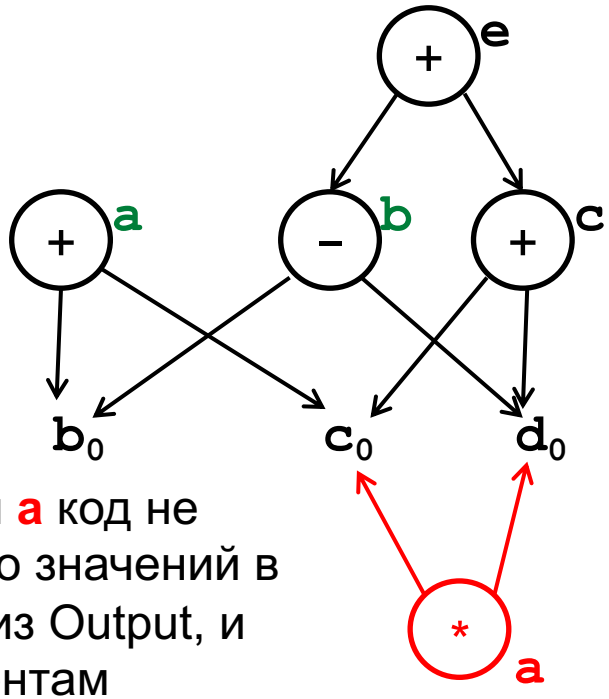
Базовый блок B после удаления мертвого кода

1.6 Локальная оптимизация
1.6.9 Удаление мертвого кода

Вычисление множеств **Input** и **Output** – отдельная задача, которая находится за пределами локальной оптимизации.
В множество Output входят переменные, значение которых используется вне данного базового блока.

◊ **Пример**. Рассмотрим базовый блок $B = \langle P, \{a, b, c, d\}, \{a, b\} \rangle$

```
a ← *, c, d
a ← +, b, c
b ← -, b, d
c ← +, c, d
e ← +, b, c
```




```
a ← +, b, c
b ← -, b, d
```

Новые значения **c** и **e** можно не вычислять, так как **c** и **e** не входят в множество *Output*

Для первого присваивания **a** код не сгенерируется, т.к. начав со значений в LiveNow для переменных из Output, и переходя по дугам (аргументам выражений в строке ТЗ), мы не посетим строку в ТЗ, соответствующую **a ← *, c, d**

Базовый блок B после удаления мертвого кода

1.7 Восстановление базового блока по его ОАГ

1.7.1. Правила

- ◊ Для каждого узла с одной или несколькими связанными переменными строится трехадресная инструкция, которая вычисляет значение одной из этих переменных.
- ◊ Если у узла несколько присоединенных живых переменных, то следует добавить команды копирования, которые присвоят корректное значение каждой из этих переменных.

1.7 Восстановление базового блока по его ОАГ

1.7.2. Пример

◊ Пусть ОАГ представлен следующим массивом записей.

1	b_0	ссылка в ТС		
2	c_0	ссылка в ТС		
3	d_0	ссылка в ТС		
4	+	1	2	a
5	-	4	3	d, b
6	+	5	2	c
8	a	ссылка в ТС		
9	b	ссылка в ТС		
10	c	ссылка в ТС		
11	d	ссылка в ТС		

◊ Применяя правила из п. 1.7.1, получим:

Базовый блок

$B = \langle P, Input, Output \rangle$

P: $\mathbf{a} \leftarrow +, b_0, c_0$

$\mathbf{d} \leftarrow -, a, d_0$

$\mathbf{c} \leftarrow +, d, c_0$

$\mathbf{b} \leftarrow d$

$Input = \{b_0, c_0, d_0\}$

$Output = \{a, b, c, d\}$

1.7 Восстановление базового блока по его ОАГ

1.7.2. Пример

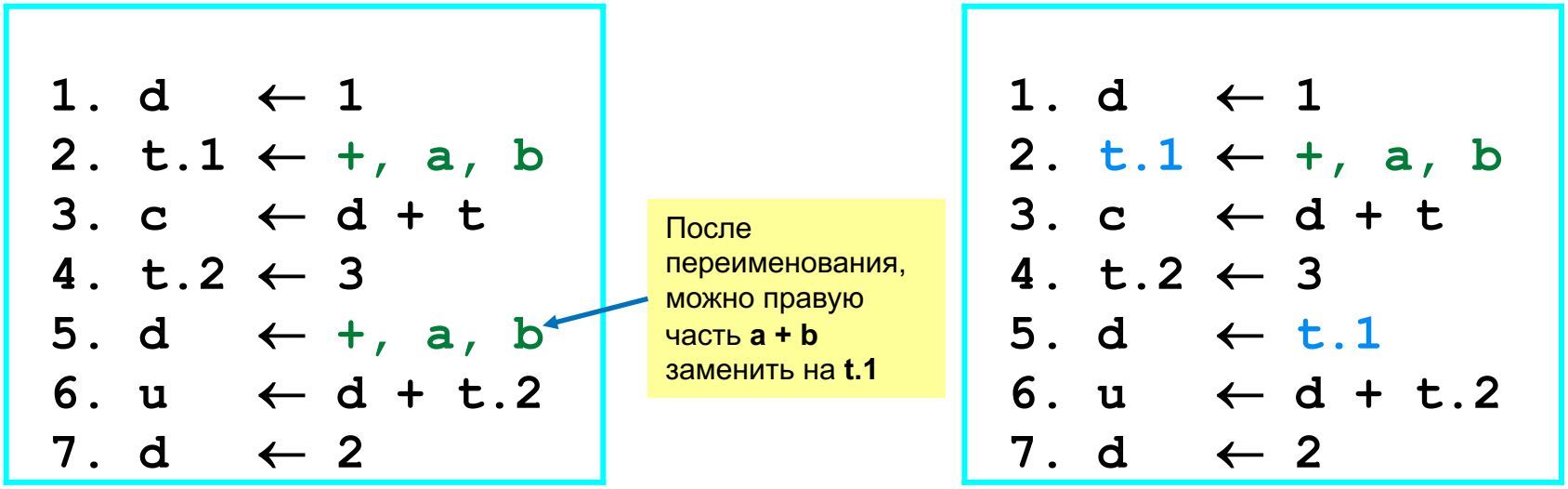
При восстановлении базового блока, в котором есть переопределения переменных, обеспечить корректность кода получится одним из следующих способов:

1) запретить использование тех значений, для которых есть переопределение переменной, с которой оно связано (например, проверять при замене выражения, является ли 1-я переменная в правой части строки ТЗ «живой» в таблице **LiveNow**).

Этот способ может привести к упущенным возможностям оптимизации: например, в строке 5 нельзя будет использовать **t** в правой части вместо **a + b**.

1.7 Восстановление базового блока по его ОАГ

1.7.2. Пример



2) переименовать часть переменных (каждое определение t будет иметь уникальный номер).

2-й способ приводит к необходимости доработки алгоритма, т.к. потребуется поменять имя у одной из одновременно «живых» переменных. Идея о том, что каждое (пере-) определение переменной порождает собственное уникальное имя переменной, лежит в основе SSA-формы, которая будет рассмотрена на следующих лекциях.

Операция копирования `d ← t.1` должна остаться на том же месте относительно других операций базового блока, чтобы сохранить семантику программы.

1.7 Восстановление базового блока по его ОАГ

1.7.3. Пример

◊ Пусть ОАГ представлен следующим массивом записей.

1	b_0	ссылка в ТС		
2	c_0	ссылка в ТС		
3	d_0	ссылка в ТС		
4	+	1	2	a
5	-	4	3	d, b
6	+	5	2	c
8	a	ссылка в ТС		
9	b	ссылка в ТС		
10	c	ссылка в ТС		
11	d	ссылка в ТС		

◊ Применяя правила из п. 1.7.1, получим:

Базовый блок

$B = \langle P, Input, Output \rangle$

P: $a_4 \leftarrow +, b_0, c_0$

$d_5 \leftarrow -, a_4, d_0$

$c_6 \leftarrow +, d_5, c_0$

$b_5 \leftarrow d_5$

$Input = \{b_0, c_0, d_0\}$

$Output = \{a_4, b_5, c_6, d_5\}$

При восстановлении базового блока переменным нужно присвоить новые имена (индексы), в соответствии с номерами их значений в ТЗ.

1.8.1. Подробный алгоритм построения таблицы значений (1/4)

$VT[\,]$ – хэш-таблица значений (ТЗ), состоящая из записей вида:
⟨ индекс, Op, $Lnum$, $Rnum$, $vars[\,]$ ⟩ для операций;
⟨ индекс, "nm", $constVal$, $vars[\,]$ ⟩ для констант;
⟨ индекс, "id", $vars[\,]$ ⟩ для переменных.

$LiveNow\,[\,]$ –таблица, отображающая имя переменной на значение (ключ) для $VT[\,]$, активное в данный момент

Перед началом работы алгоритма:
- для каждой переменной t в $VT[\,]$ есть запись вида
⟨ $valueNumber$, "id", [t] ⟩, а при этом $LiveNow\,[t] = valueNumber$

- для каждой константы со значением $Const$ в $VT[\,]$ есть запись вида
⟨ $valueNumber$, "nm", $Const$, [ ] ⟩

1.8.1. Подробный алгоритм построения таблицы значений (2/4)

for each instr $= \langle\, t_i \leftarrow Op_i, l_i, r_i \,\rangle \in BasicBlock$ do

$L_n = LiveNow[l_i]$ // получаем актуальные в данной точке

$R_n = LiveNow[r_i]$ // номера значений аргументов l_i и r_i

// 1) если оба аргумента – константы, пробуем сделать свертку констант:

if $VT[L_n].constVal$ && $VT[R_n].constVal$

// Вычисляем новую константу

$newConst = evalExpr(Op_i, VT[L_n].constVal, VT[R_n].constVal)$

// Обновляем таблицу значений: создаем новую строку с новой константой

$row = \langle "nm", newConst \,\rangle$

$num = getHash(row)$

if $VT.hasKey(num)$ // Если такая константа уже есть в ТЗ, присоединяем

$VT[num].vars.add(t_i^{num})$ // ещё одну переменную в её строку

else // Либо если такая константа ранее не встречалась,

$VT.add(num, row, [t_i^{num}])$ // добавляем её в ТЗ

$instr.replace(\langle t_i^{num} \leftarrow newConst^{num} \rangle)$ // изменяем код, добавляем номера

$LiveNow[t_i] = num$ // Обновляем актуальный номер значения для t_i

continue // переходим к следующей инструкции

1.8.1. Подробный алгоритм построения таблицы значений (3/4)

// (продолжение цикла) for each instr $= \langle\, t_i \leftarrow Op_i, l_i, r_i \,\rangle \in BasicBlock$ do

// 2) пробуем найти общие подвыражения (среди сохраненных в ТЗ)

$num = getHash(\langle Op_i, L_n, R_n \rangle)$

if $VT.hasKey(num)$ // Если такое выражение уже есть в ТЗ

$t_j^{num} = VT[num].vars.first()$ // Берем первую переменную для значения

$instr.replace(\langle t_i^{num} \leftarrow t_j^{num} \rangle)$ // Заменяем правую часть, добавляем

// в коде номера значений к переменным (верхние индексы)

$VT[num].vars.add(t_i^{num})$ // добавляем t_i^{num} к списку присоединенных

// переменных в ТЗ

$LiveNow[t_i] = num$ // обновляем $LiveNow[t_i]$ – актуальное значение

// переменной t_i сейчас находится в строке ТЗ с номером num

continue // переходим к следующей инструкции

// 3) Если операция Op_i коммутативна, то нужно также попробовать

// повторить п. (2) для другого порядка аргументов: $\langle Op_i, R_n, L_n \rangle$

1.8.1. Подробный алгоритм построения таблицы значений (4/4)

// (продолжение цикла) for each $\mathbf{instr} = \langle\, t_i \leftarrow Op_i, l_i, r_i \,\rangle \in BasicBlock$ do

// 4) иначе, если такого выражения ранее не встречалось, добавим его в
// таблицу значений:

$VT.add(num, \langle Op_i, L_n, R_n \rangle)$

$LiveNow[t_i] = num$ // обновляем $LiveNow[t_i]$

// Находим главные (первые) переменные для значений аргументов:

$l_{main} = VT[L_n].vars.first()$

$r_{main} = VT[R_n].vars.first()$

$instr.replace(\langle t_i^{num} \leftarrow Op_i, l_{main}^{L_n}, r_{main}^{R_n} \rangle)$

// Заменяем в правой части вхождения присоединенных переменных
// на главные (это позволяет позже удалить избыточные копирования),
// добавляем верхние индексы переменным в соответствии с их номерами
// значений

// конец цикла – переходим к следующей инструкции

1.8.2. Алгоритм удаления мертвого кода для базового блока (1/3)

$\mathbf{workList} = \emptyset$

$\boldsymbol{usefulInstrs} = \emptyset$

for each $\mathbf{var_i} \in \boldsymbol{Output}$ do

$\boldsymbol{num_i} = \boldsymbol{LiveNow}[\boldsymbol{var_i}]$ // какому значению соответствует var_i в конце ББ

$\boldsymbol{workList}.\boldsymbol{add}(\boldsymbol{num_i})$ // добавляем в список все корни ОАГ для выражений

1.8.2. Алгоритм удаления мертвого кода для базового блока (2/3)

while $workList \neq \emptyset$ do
 $num = workList.removeFirst()$
 for each $t^{num} \in VT[num].vars$ do
  пусть $instr = \langle t^{num} \leftarrow Op, l, r \rangle$ – инструкция, в которой определяется t^{num}
  $\langle num, Op, L_n, R_n, vars \rangle = VT[num]$
  // Обрабатываем главные и присоединенные переменные по-разному
  if $t^{num} = vars.first()$
   // Помечаем инструкцию, как «полезную»
   $usefulInstrs.add(instr)$
   $workList.add(L_n)$
   $workList.add(R_n)$
 else
  // Определение присоединенной переменной «полезно», только если
  // переменная входит в Output, и это её последнее переопределение
  if $t \in Output$ && $LiveNow[t] = val$
   $usefulInstrs.add(instr)$

1.8.2. Алгоритм удаления мертвого кода для базового блока (3/3)

// удаляем все инструкции, не отмеченные в качестве «полезных»

for each $instr \in BasicBlock$ do

  if $instr \notin usefulInstrs$

    $BasicBlock.remove(instr)$

1.8.3. Переименование индексов переменных

// удаляем индексы у всех переменных, с которыми они входят в $\boldsymbol{Output}$

for each $v \in \boldsymbol{Output}$ do

$\boldsymbol{ind} = \boldsymbol{LiveNow}[v]$

$\boldsymbol{BasicBlock.replaceVar}\left(v^{\boldsymbol{ind}}, v\right)$

// Аналогично, нужно удалить индексы у всех переменных, с которыми они входят в $\boldsymbol{Input}$. Здесь для каждой переменной из $\boldsymbol{Input}$ следует взять тот индекс, с которым она впервые попала в $\boldsymbol{LiveNow}$ перед началом работы алгоритма.

1.9 Локальная оптимизация. Заключительные замечания

1.9.1. Заключительные замечания

- ◊ Стремление разработать полноценную локальную оптимизацию привело к необходимости построить связи по данным оптимизируемого базового блока с другими базовыми блоками, т.е. к необходимости глобального анализа оптимизируемой процедуры (функции, метода).
- ◊ Необходимо научиться строить базовые блоки больших размеров, чтобы повысить эффективность локальной оптимизации.
- ◊ Ссылаться на базовые блоки по их идентификаторам неудобно. Необходимо научиться нумеровать базовые блоки, чтобы их можно было называть B_1, B_2 и т.д. Решение этой задачи связано с обходом ГПУ.
- ◊ Далее будут рассмотрены методы решения перечисленных задач и некоторых других задач, которые будут возникать по ходу дела.