ДРАКОН

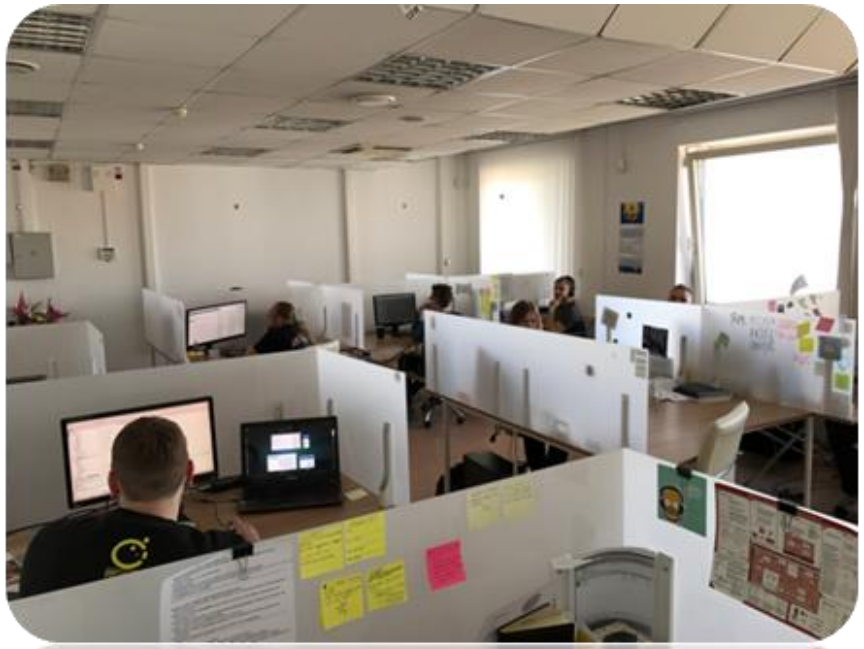
«Лаборатории Инвенто»

Артем Бразовский

руководитель отдела **R&D**
лаборатории **IoT**

inventolabs

«Лаборатории Инвенто»



- Основана в 2009 году
- С 2010 года – резидент ПВТ
- 100% частная компания
- Годовой доход - более $ 1 млн.
- Годовой прирост - более 50%
- Более 50 завершенных проектов для компаний из 15 стран мира

inventolabs

Что такое язык Дракон ?

Дружелюбный

Русский

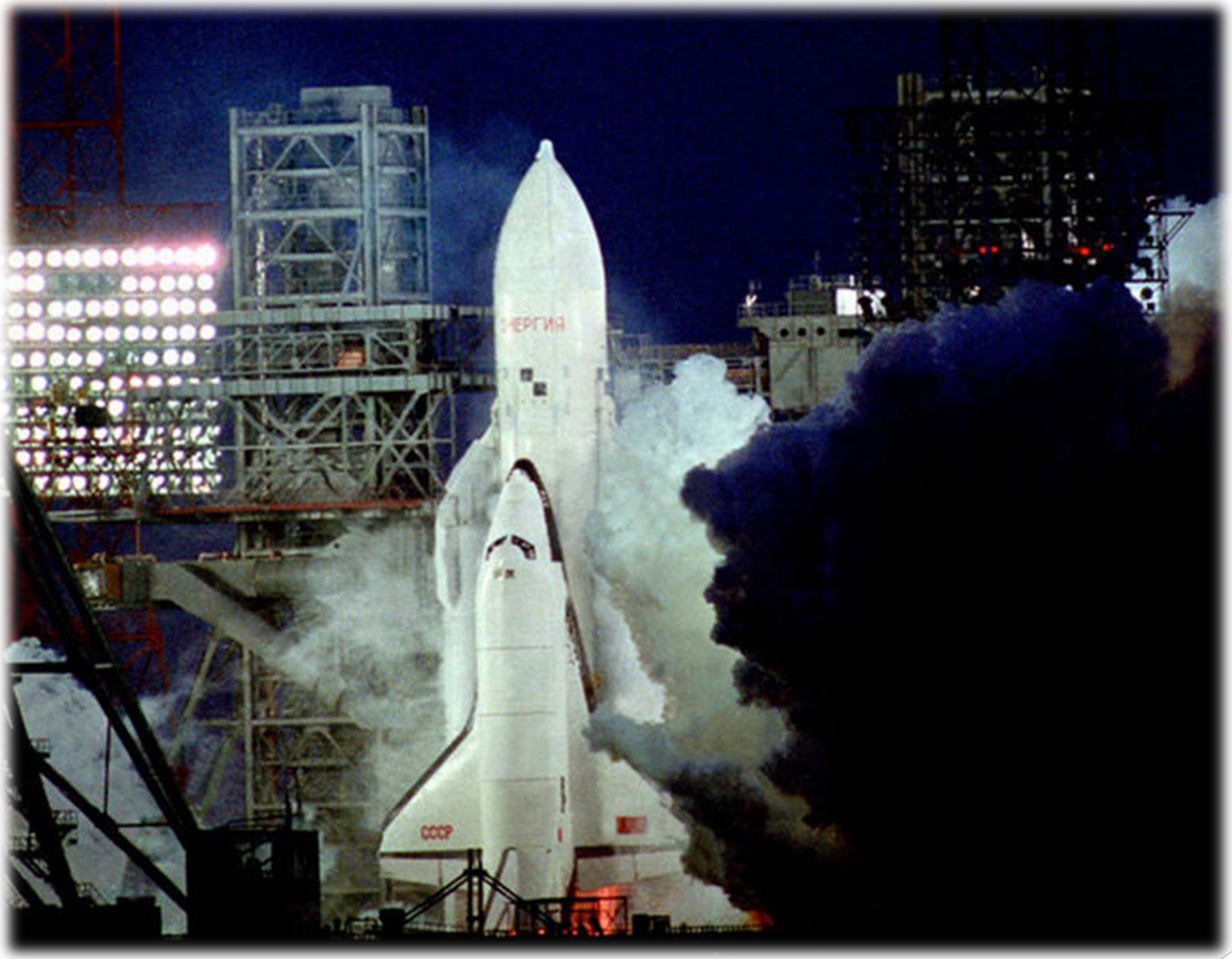
Алгоритмический язык

Который

Обеспечивает

Наглядность

inventolabs



inventolabs

Справка о языке ДРАКОН

- Родился в космической колыбели
- Разработка языка началась в 1986.
- Поступил в эксплуатацию в 1996

Системы управления космическими аппаратами

Высокая сложность

Высокая надежность

Высокая цена ошибки

Пуски ракет-носителей и разгонных блоков, при создании которых использовался язык ДРАКОН, производятся или производились с шести космодромов мира:

- Плесецк
- Байконур
- Kourou (Французская Гвиана)
- Плавучий космодром «Морской старт»
- Naro (Южная Корея)
- Восточный

Области применения языка ДРАКОН

- Формализация бизнес процессов
- Бизнес консалтинг
- Описание инструкций для обучения сотрудников
- Разработка систем учета
- Образование
- Врачебная практика
- Военная сфера
- …

inventolabs

Инженер

?

Программист

inventolabs

Инженер

Дракон

Программист

inventolabs

Джеймс Мартин (1933—2013)

inventolabs

inventolabs

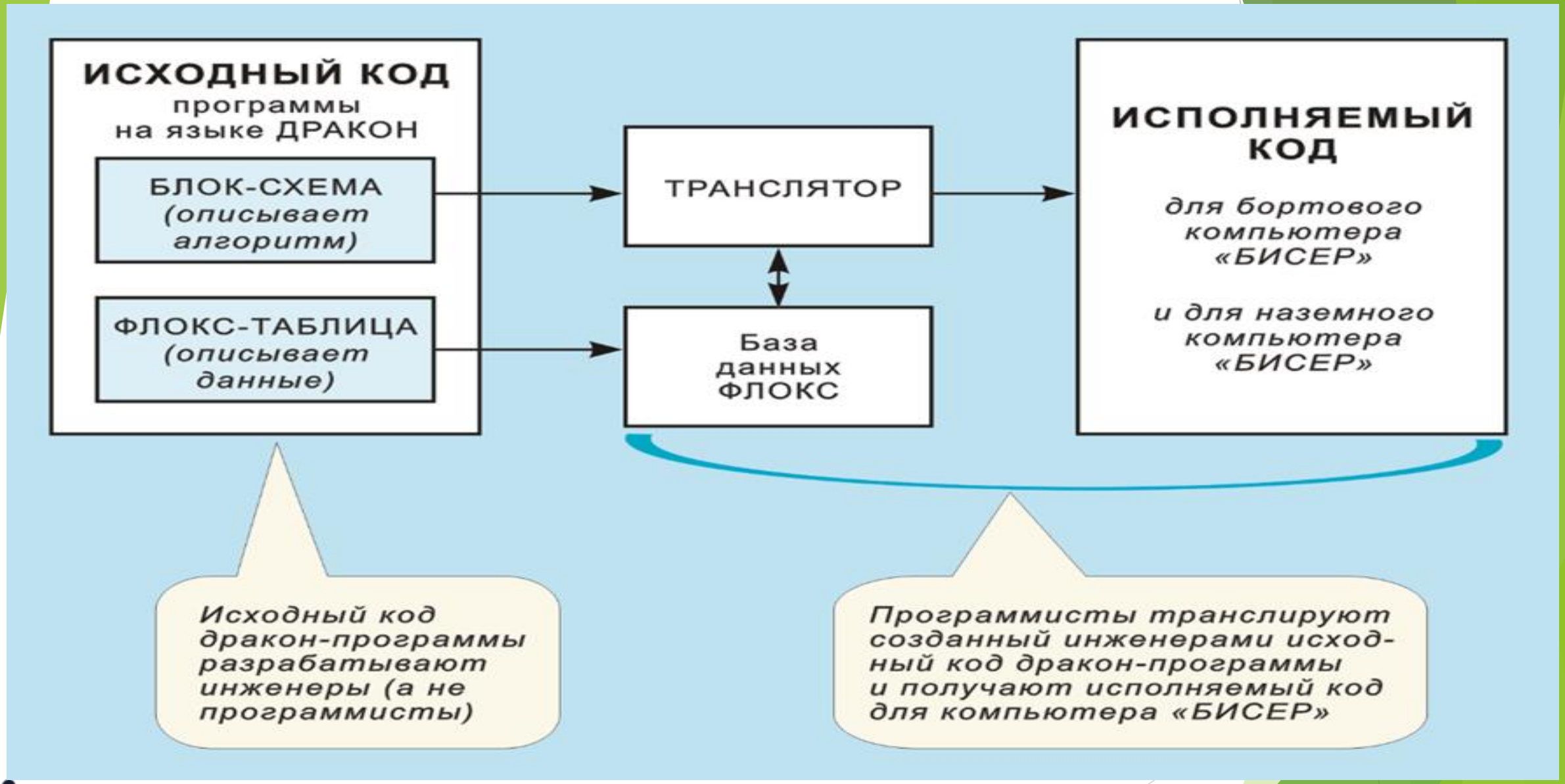
ДРАКОН-технология в НПЦАП —

это технология разработки

алгоритмов и программ

Графит – флокс

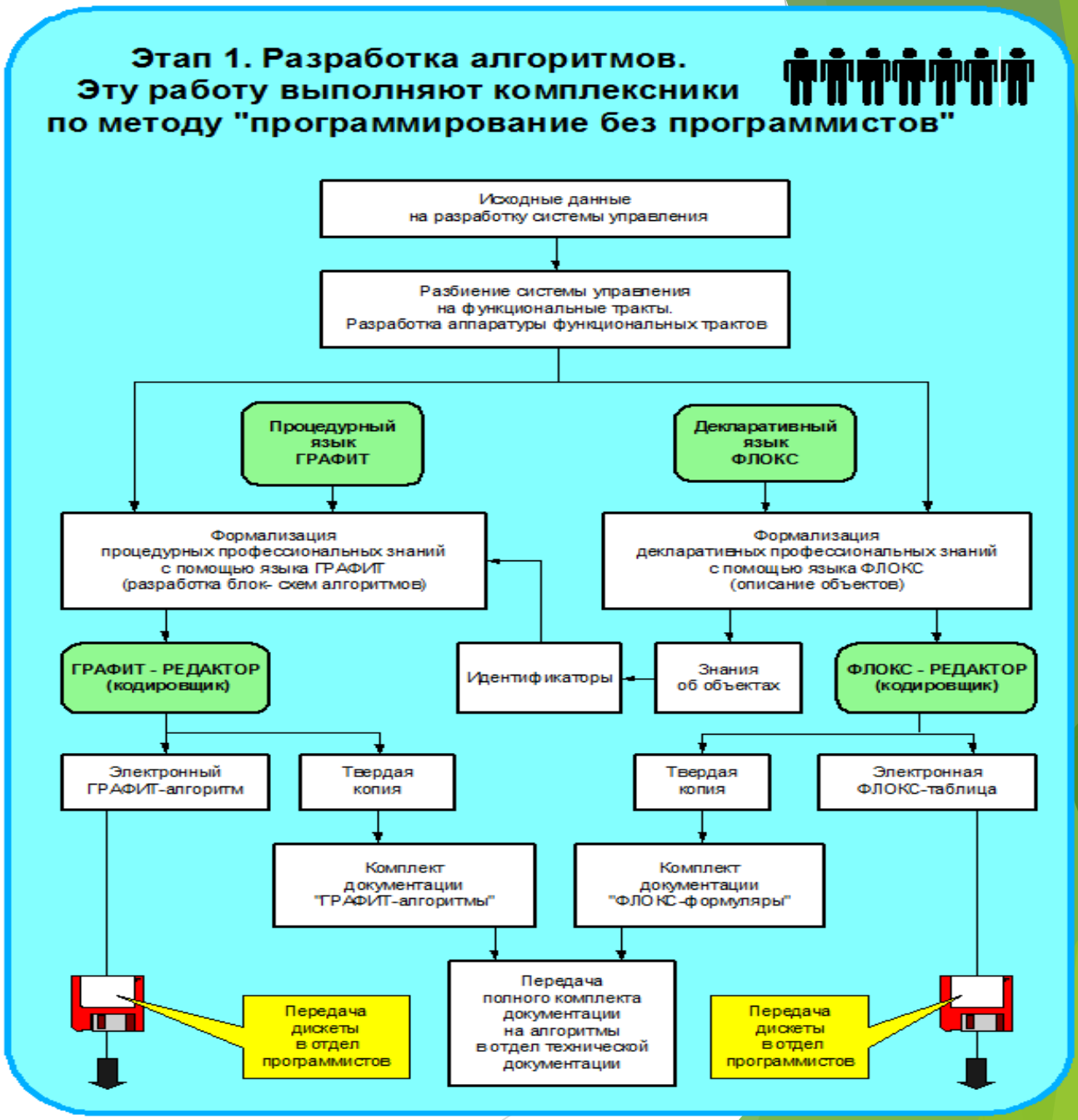
1996 — 2017



inventolabs

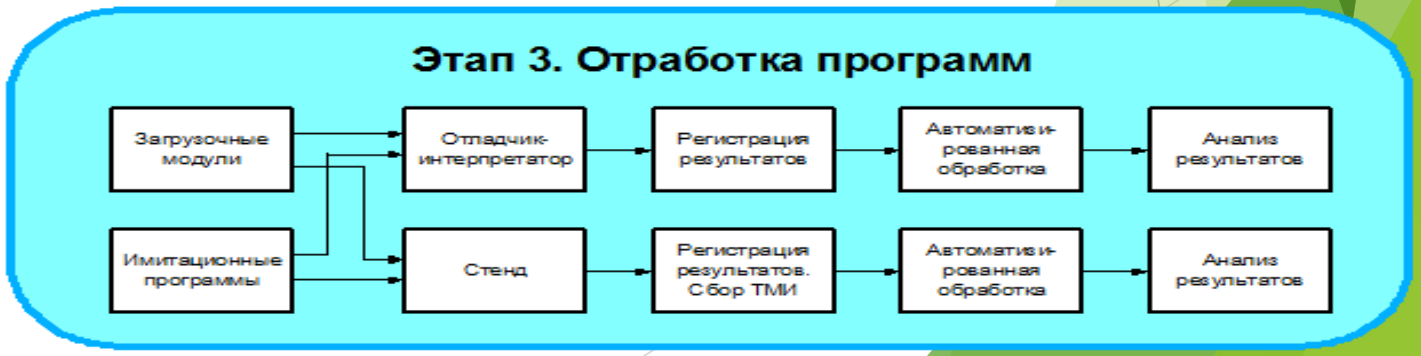
Инженеры

Кто обладает знаниями - тот и должен их формализовать.



inventolabs

Программисты



inventolabs

Знакомимся с ДРАКОН

Ц Е Л Ь

- **Облегчить и ускорить понимание алгоритмов Ч Е Л О В Е К О М**
- **Исключить ошибки в алгоритмах**
- **Специалисты из различных областей понимают друг друга. Начинает проявляться синергия.**

inventolabs

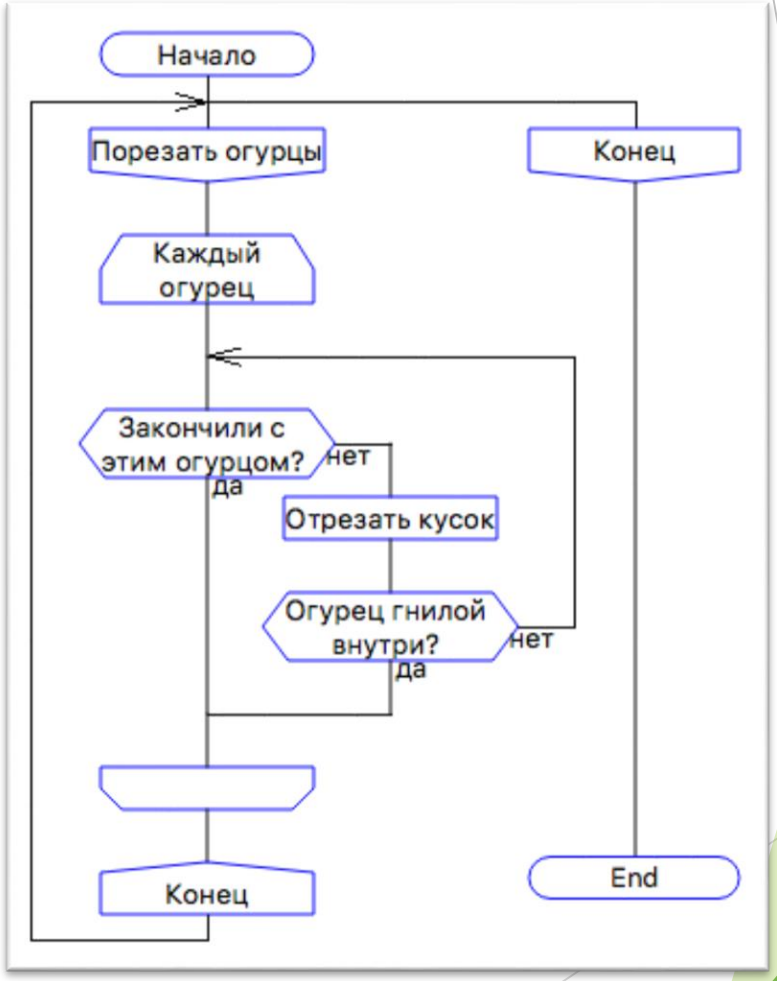
Текстовая информация

Каменный выступ на горе.
На высоте 700 метров.
С левой стороны горы высотой примерно 600 метров
С правой стороны горы высотой примерно 800 метров
Внизу озера
Погода: Переменная облачность
На выступе стоит какой-то человек

Визуальная информация

Сравнение визуального и текстового представления алгоритма.

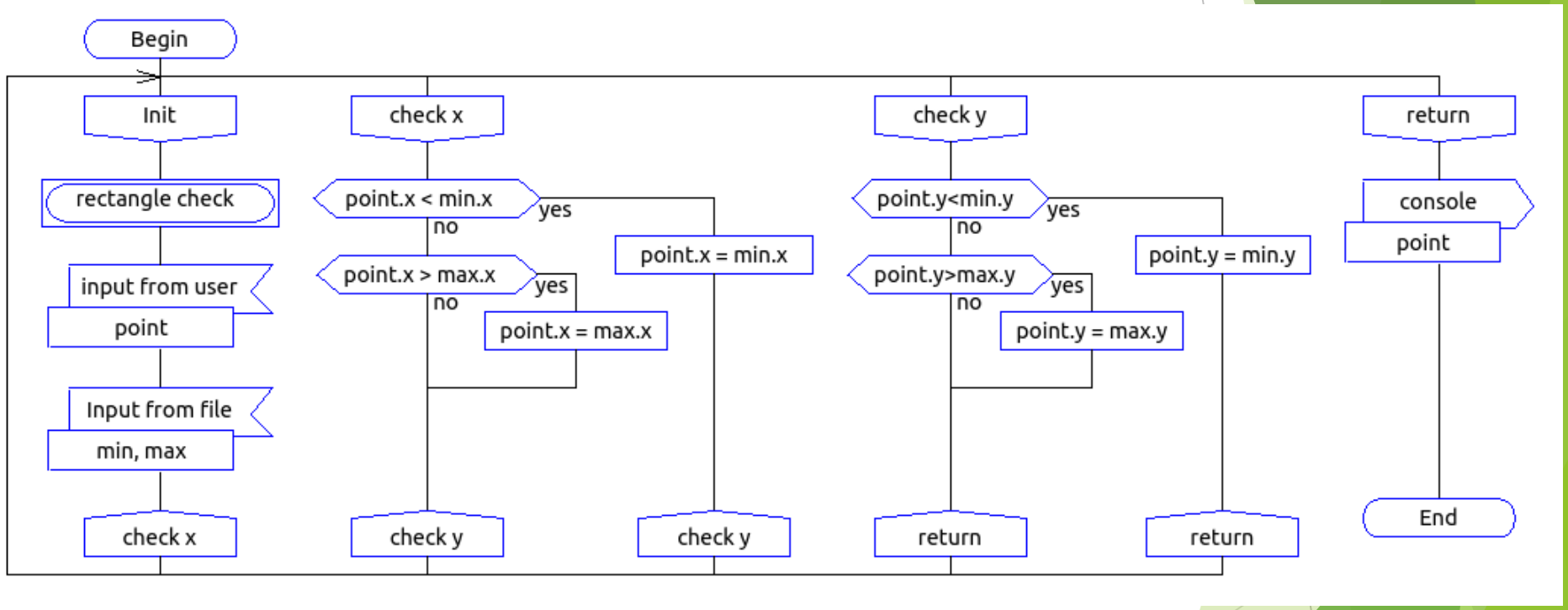
```
процедура "Почистить огурцы"
{
    каждый огурец
    {
        пока не закончили с огурцом
        {
            отрезать кусок();
            если огурец гнилой внутри
            {
              прервать цикл
            }
        }// прыгаем ли мы сюда после конца?
    }// или сюда?
}// или все же сюда?
```



inventolabs

Пример реализации на языке C++ и Дракон

```
Point point;
int tmp;
cout<<"enter X"<<endl;
cin>>tmp;
point.x = tmp;
cout<<endl<<"enter Y"<<endl;
cin>>tmp;
point.y = tmp;
Point min, max;
ifstream f("min_max.txt");
f>>min>>max;

if(point.x<min.x){
    point.x = min.x;
}
else{
    if(point.x > max.x ){
        point.x = max.x;
    }
}
if(point.y<min.y){
    point.y = min.y;
}
else{
    if(point.y > max.y ){
        point.y = max.y;
    }
}

cout<<point;
```



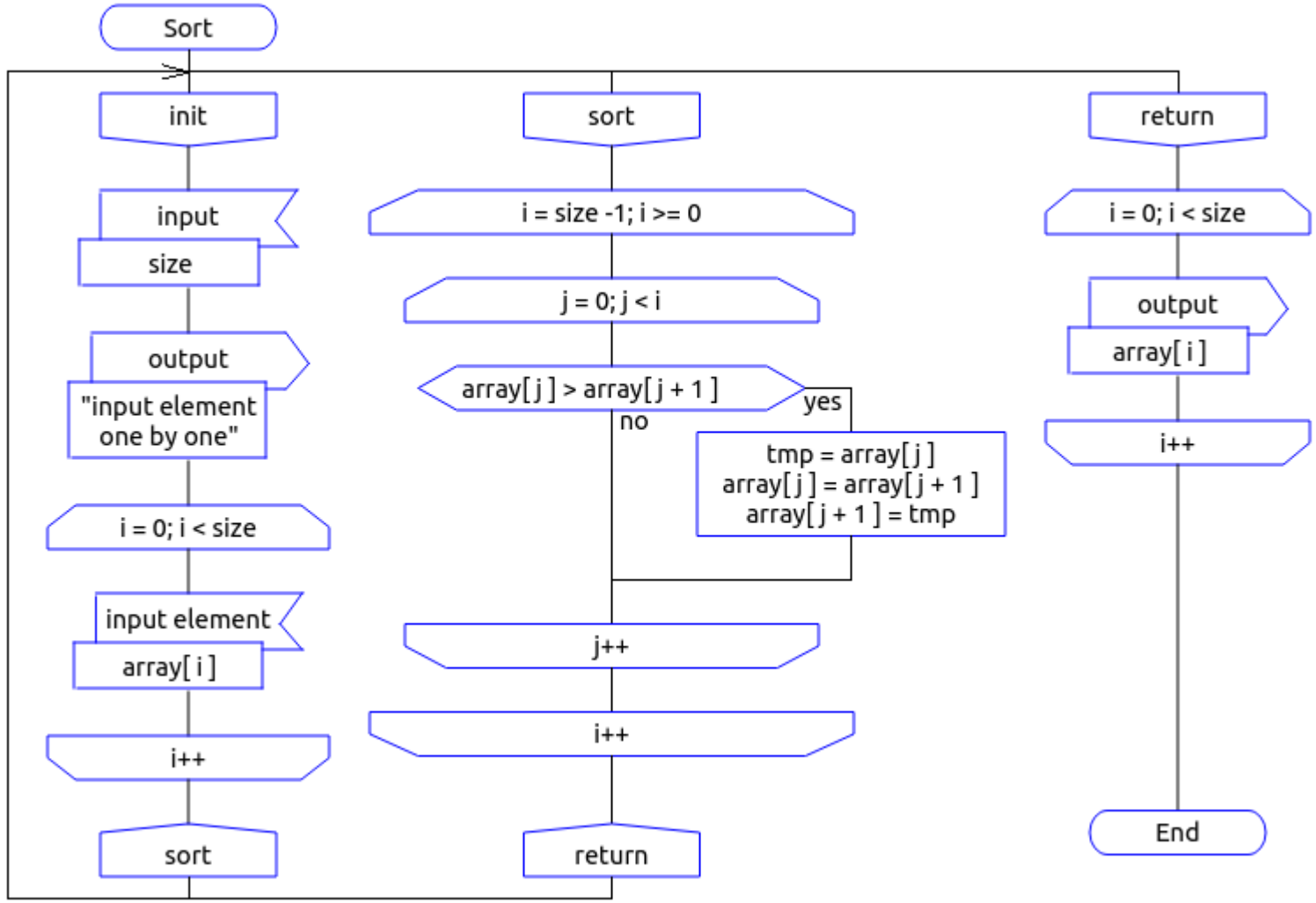
inventolabs

Сортировка на языке C++

```
int* array;
int size;

// init
cout<<"input size"<<endl;
cin>>size;
array = new int[size];
cout<<"input elements one by one"<<endl;
int tmp;
for(int i=0; i<size; i++){
  cin>>tmp;
  array[i]=tmp;
  cout<<endl;
}

// sort
for (int i = size - 1; i >= 0; i--)
{
  for (int j = 0; j < i; j++)
  {
    if (array[j] > array[j + 1])
    {
      int tmp = array[j];
      array[j] = array[j + 1];
      array[j + 1] = tmp;
    }
  }
}

// return
 for(int i=0; i<size; i++){
    cout<<array[i]<<endl;
}
```

inventolabs

Сортировка на языке дракона



inventolabs

Базовые правила

Граф с вертикальными и горизонтальными рёбрами

Отсутствие визуальных помех

Нет пресечений ребер

Другие правила

inventolabs

Прямоугольный плоский граф

Разрешить

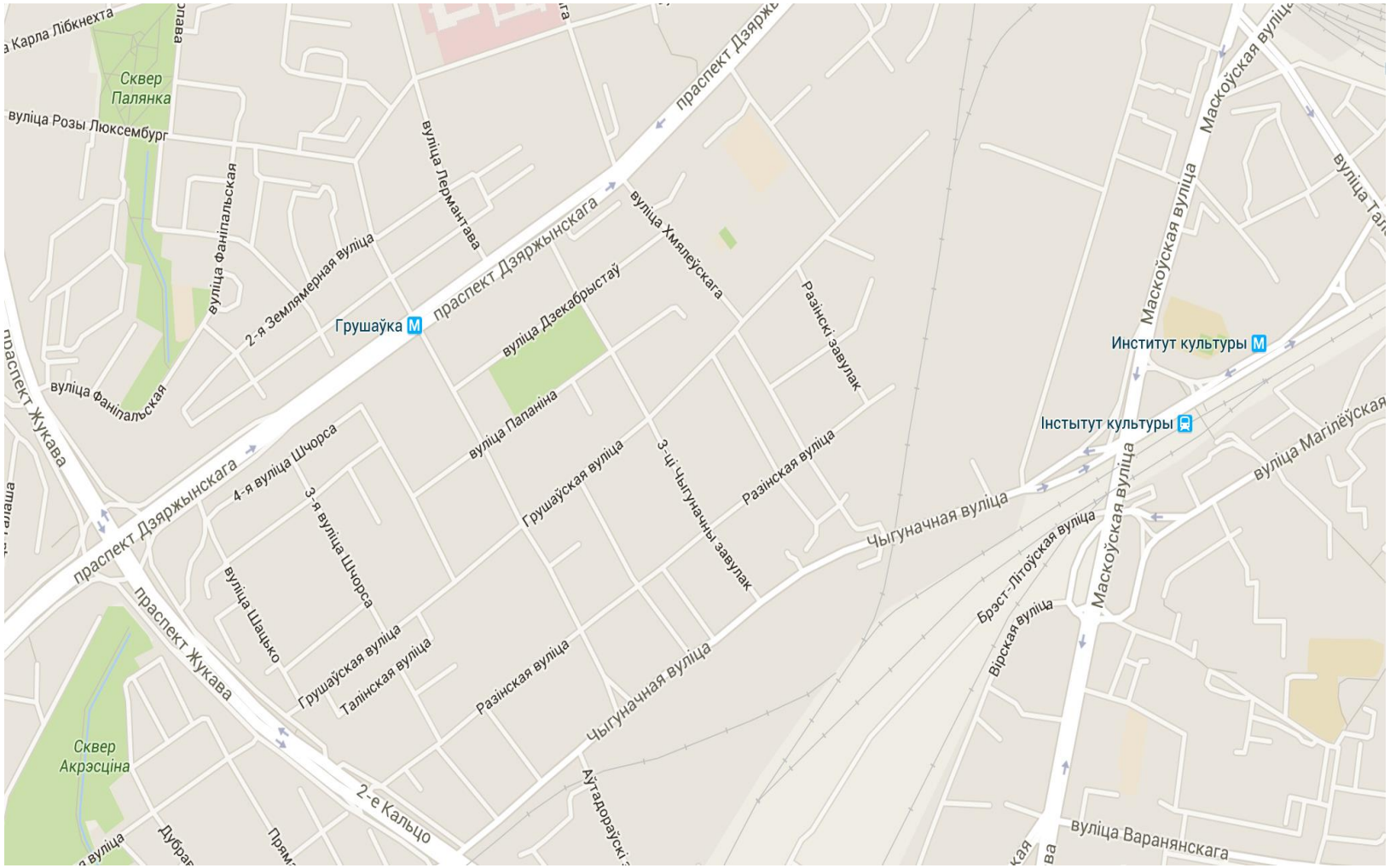
- Прямые линии
- Вертикальные или горизонтальные

Запретить

- Кривые линии
- Наклонные или диагональные

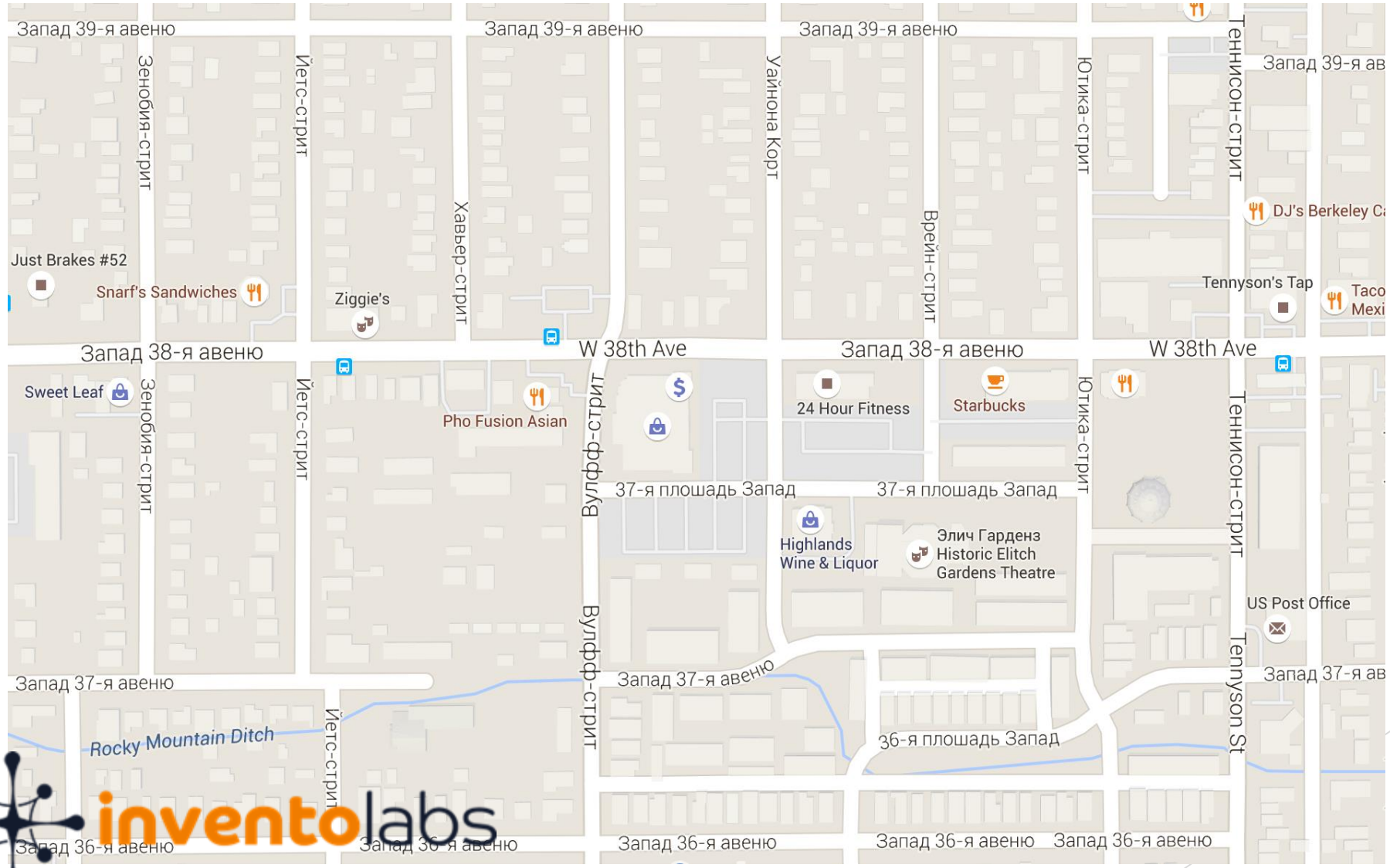
inventolabs

Пример произвольного плоского графа



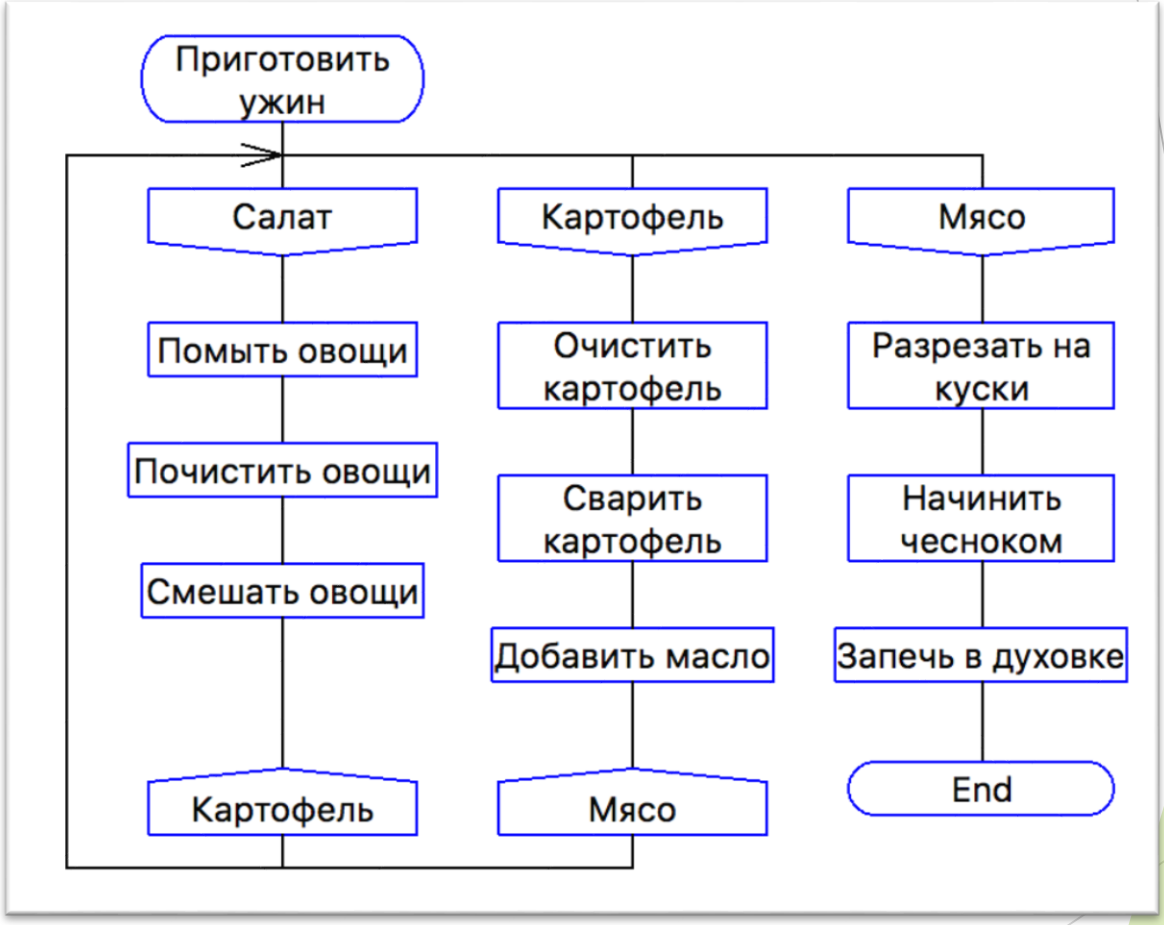
inventolabs

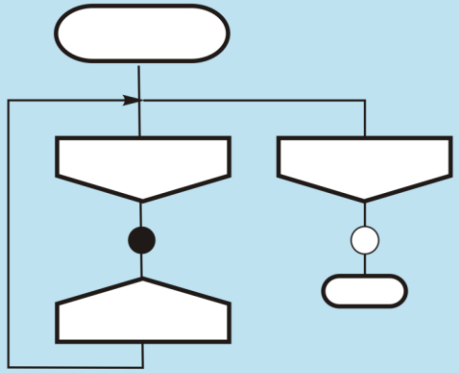
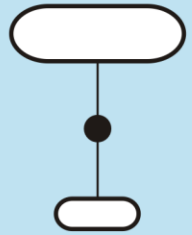
Граф с вертикальными и горизонтальными рёбрами



inventolabs

Использование шампуров, для логического разделения алгоритма



Аксиома-силуэт	Аксиома-примитив
	

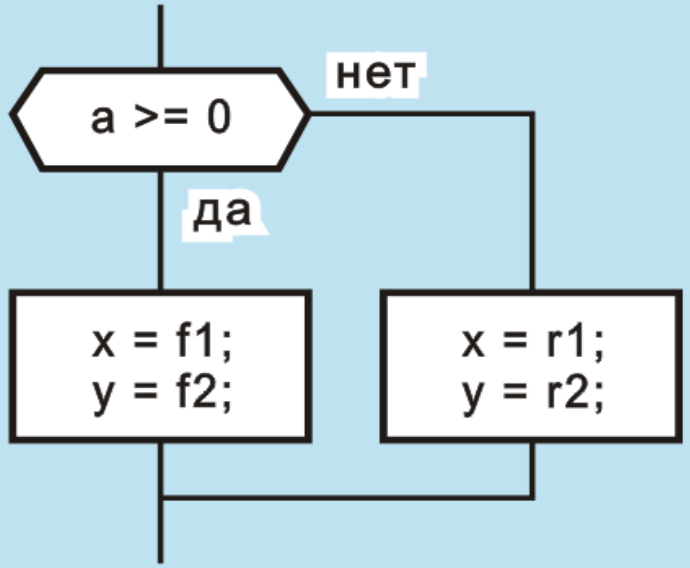
inventolabs

- **Отказаться от управляющих ключевых слов:**
  if, then, else, case, switch, break, while, do, repeat, until, for, foreach, continue, loop, exit, when, last и т. д.
- **Заменить их на управляющую графику, то есть использовать двумерное структурное программирование**

inventolabs

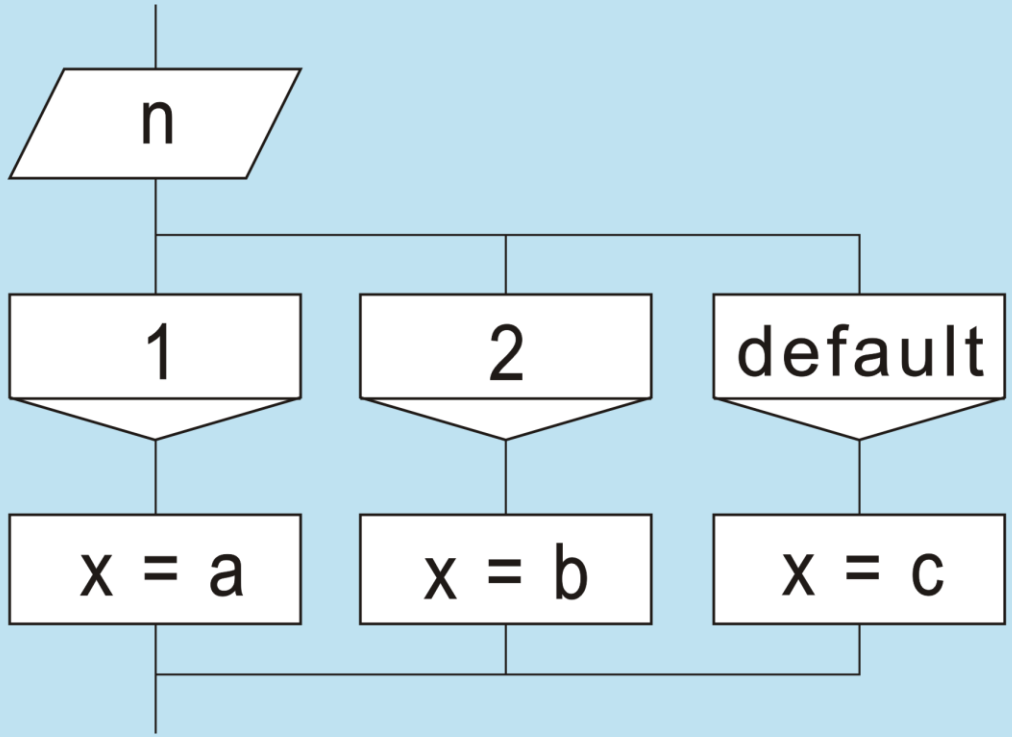
Оператор if else

Программа на языке Си	Программа на языке Дракон–Си

`if ( a >= 0 )`
`{`
`  x = f1;`
`  y = f2;`
`}`
`else`
`{`
`  x = r1;`
`  y = r2;`
`}`
	

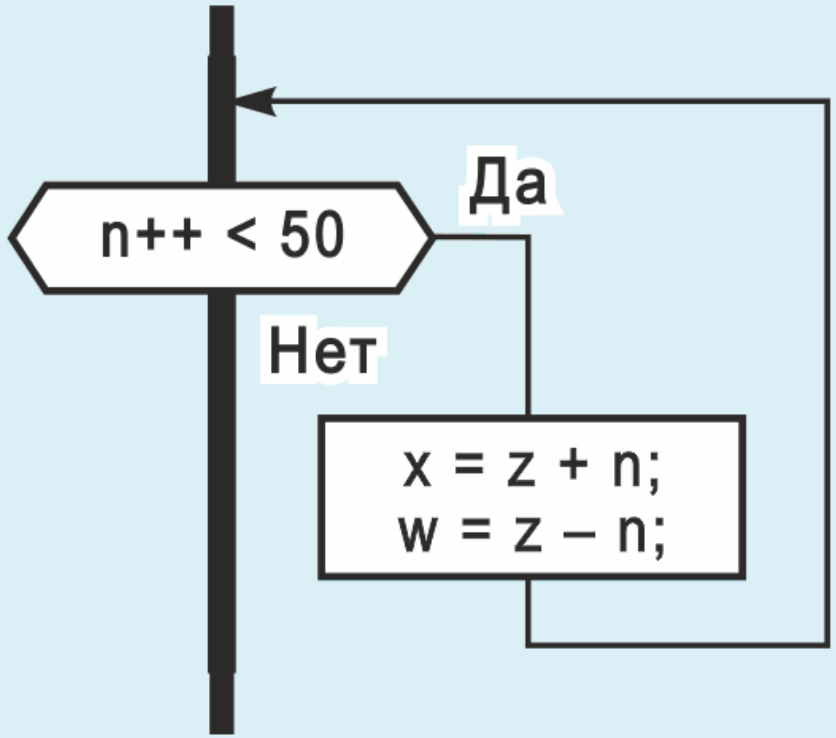
Операторы switch case break

Программа на языке Си	Программа на языке Дракон–Си

switch (n)
{
case 1:
x = a;
break;
case 2:
x = b;
break;
default:
x = c;
} /* switch */
	

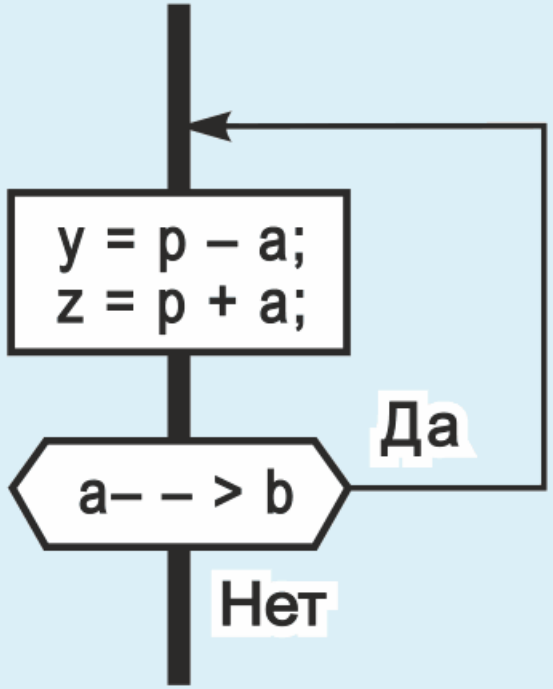
Оператор while

Программа на языке Си	Программа на языке Дракон–Си

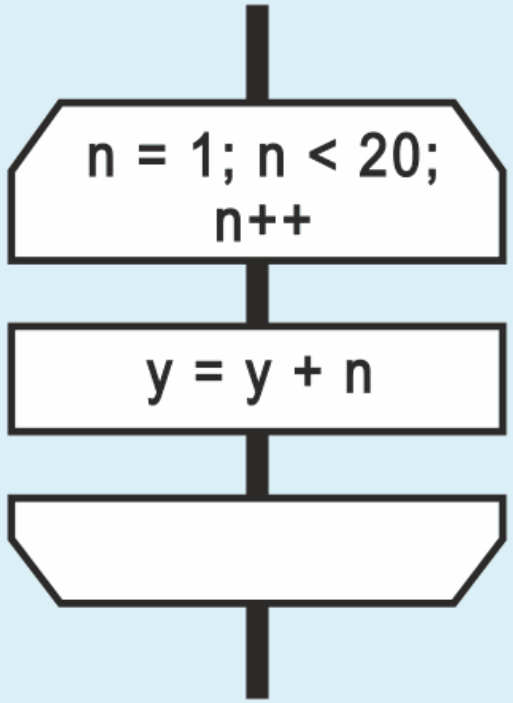
while (n++ < 50)
{
x = z + n;
w = z – n;
} /* while */
	

Оператор do while

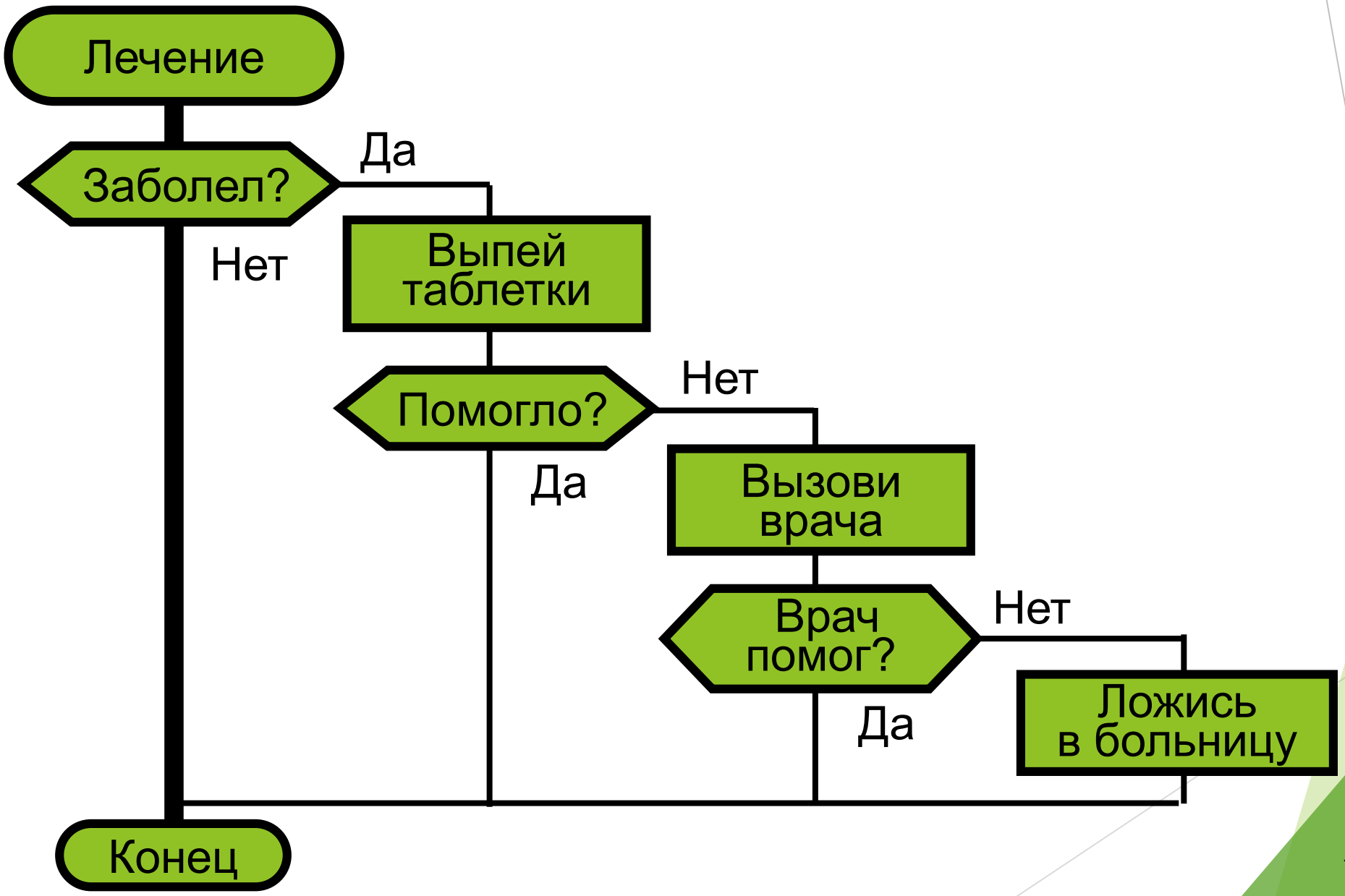
Программа на языке Си	Программа на языке Дракон–Си

do
{
y = p – a;
z = p + a;
} while (a– – > b);
/* do while */
	

Оператор for

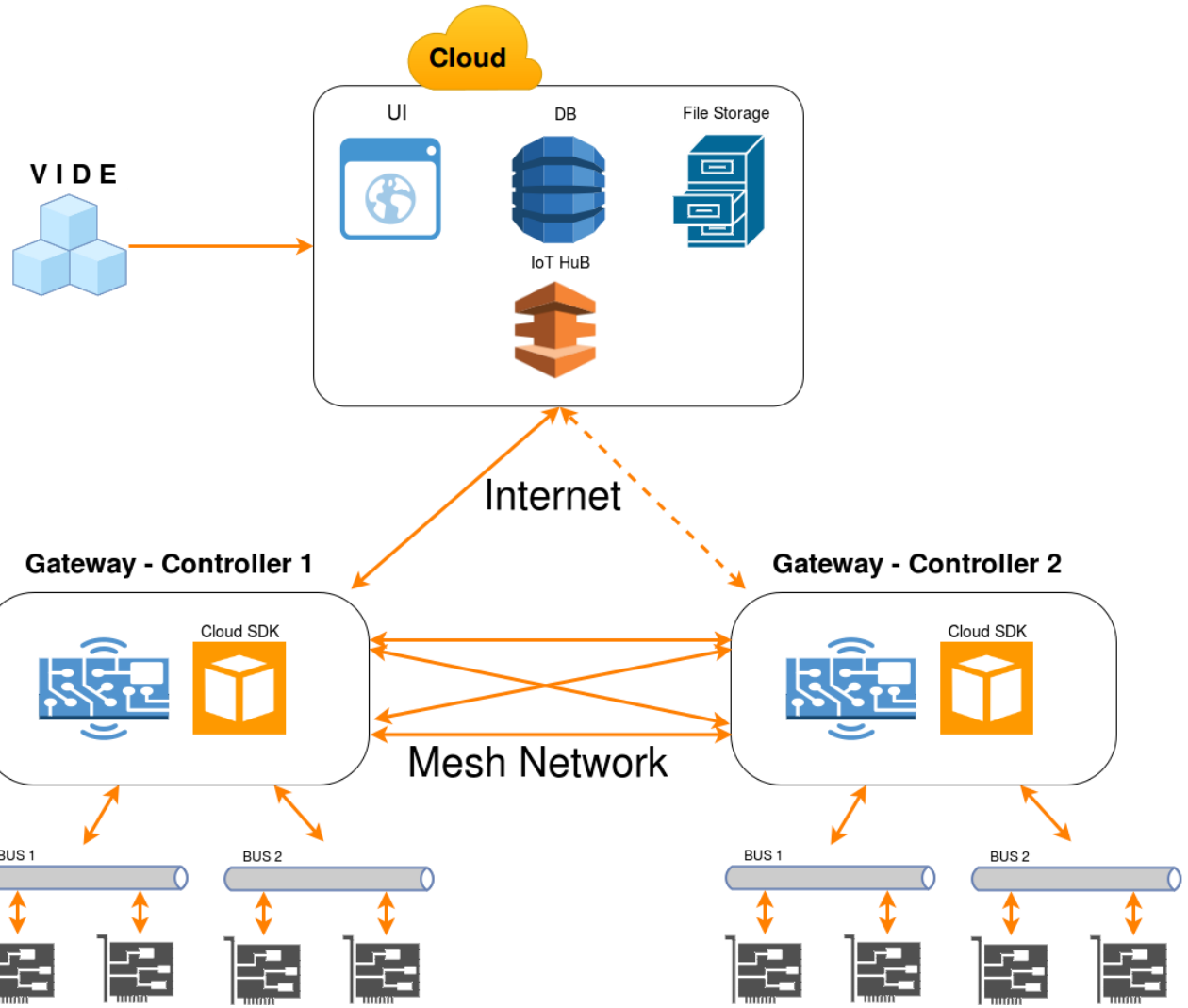
Программа на языке Си	Программа на языке Дракон–Си
for (n=1; n<20; n++) y = y + n;	

Чем правее тем хуже



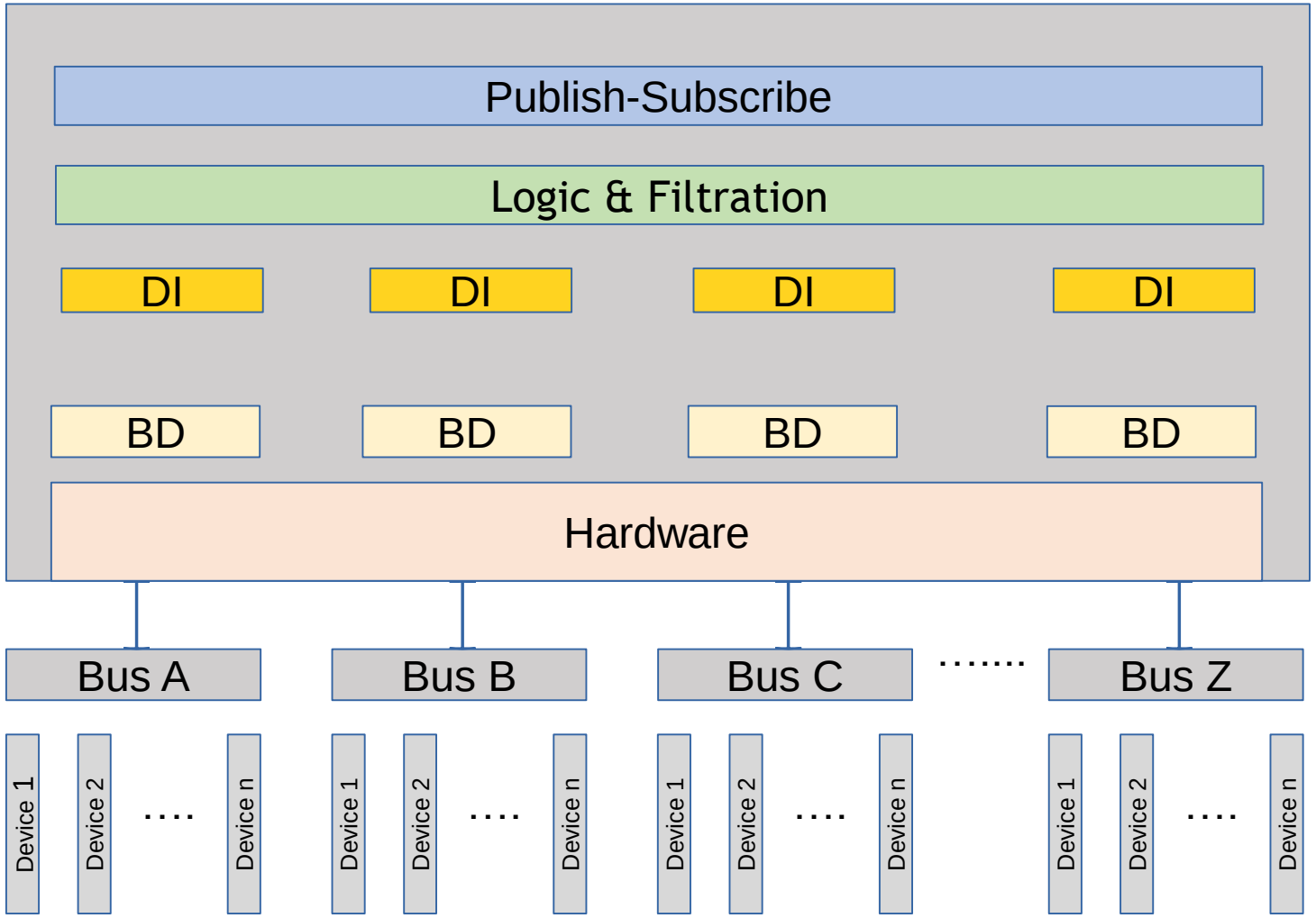
ДРАКОН в IoT
VIDE

Gateway и IoT платформа



inventolabs

Gateway



inventolabs

Три преимущества

Декомпозиция, оптимизация, предотвращение ошибок.

- Декомпозиция

Дракон делает сложные алгоритмы наглядными, что позволяет декомпозировать сложную задачу на простые составляющие.

- Оптимизация

Двумерное представление алгоритмов позволяет проще заметить возможность для оптимизации.

- Предотвращение ошибок

Высокая наглядность алгоритма позволяет заметить логические ошибки до этапа кодирования, что позволяет заметно сэкономить время на разработку.

VIDE = Visual IDE

Использование языка Assembler для создания драйверов.

Использование языка C для программирования бизнес-логики.

Визуальная форма алгоритмов упрощает процесс создание ПО.

Эргономичный интерфейс.

Интеллектуальный редактор.

inventolabs

Преимущества языка ассемблера

- Этот язык позволяет использовать все возможности процессора
- Код написанный на ассемблере позволяет достигнуть максимум производительности при минимальном размере машинного кода.

Недостатки языка ассемблера

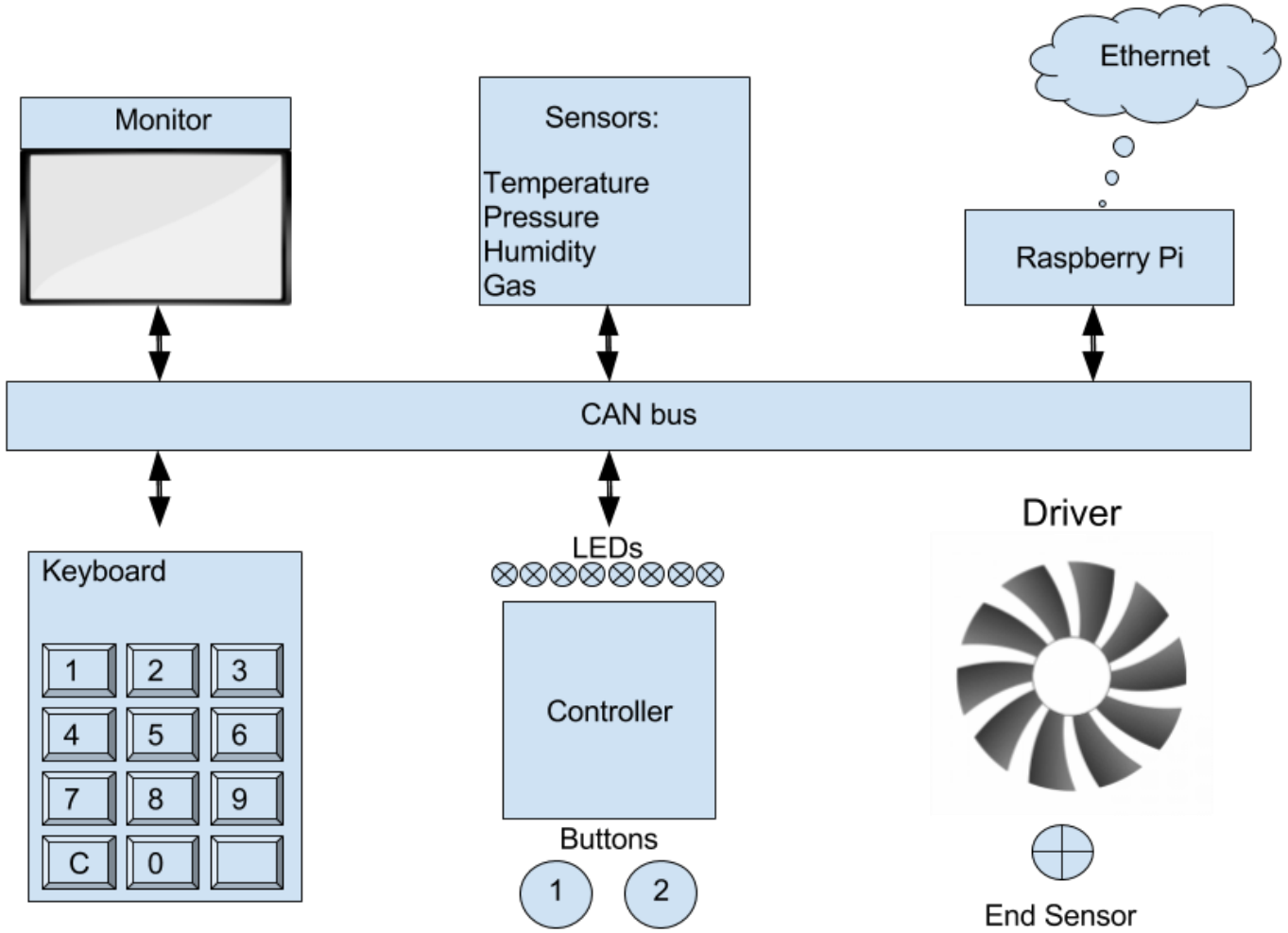
- Объем кода. То, что можно поместить в одну строчку на языках более высокого уровня может занят десятки строк ассемблерного кода.
- Плохая читабельность. Код на нём изобилует командами условных и безусловных переходов. Разобраться в порядке выполнения кода проблематично.
- Разработка на ассемблере занимает больше времени и средств.

Представляем
Visual Assembler

- Визуальные объекты наглядно представляют команды переходов, избавляясь от одного из главных недостатков ассемблера.
- Замена мнемоники на понятный синтаксис.
- Программа становится наглядной и “удобно-читабельной”.

inventolabs

Демонстрационный стенд



inventolabs

Наш контроллер

inventolabs

Что мы имеем

- VIDE - инструмент с которым удобно думать
- Просто использовать
- Легко создавать техническое задание
- Можем видеть логику создаваемого кода и делать его аудит
- Разработка и документирование в одном флаконе
- Алгоритм легко изменять, улучшать, оптимизировать
- Сокращение времени тестирования
- Высокая скорость разработки
- Меньшая зависимость от персонала
- Быстрый вход новых людей в проект

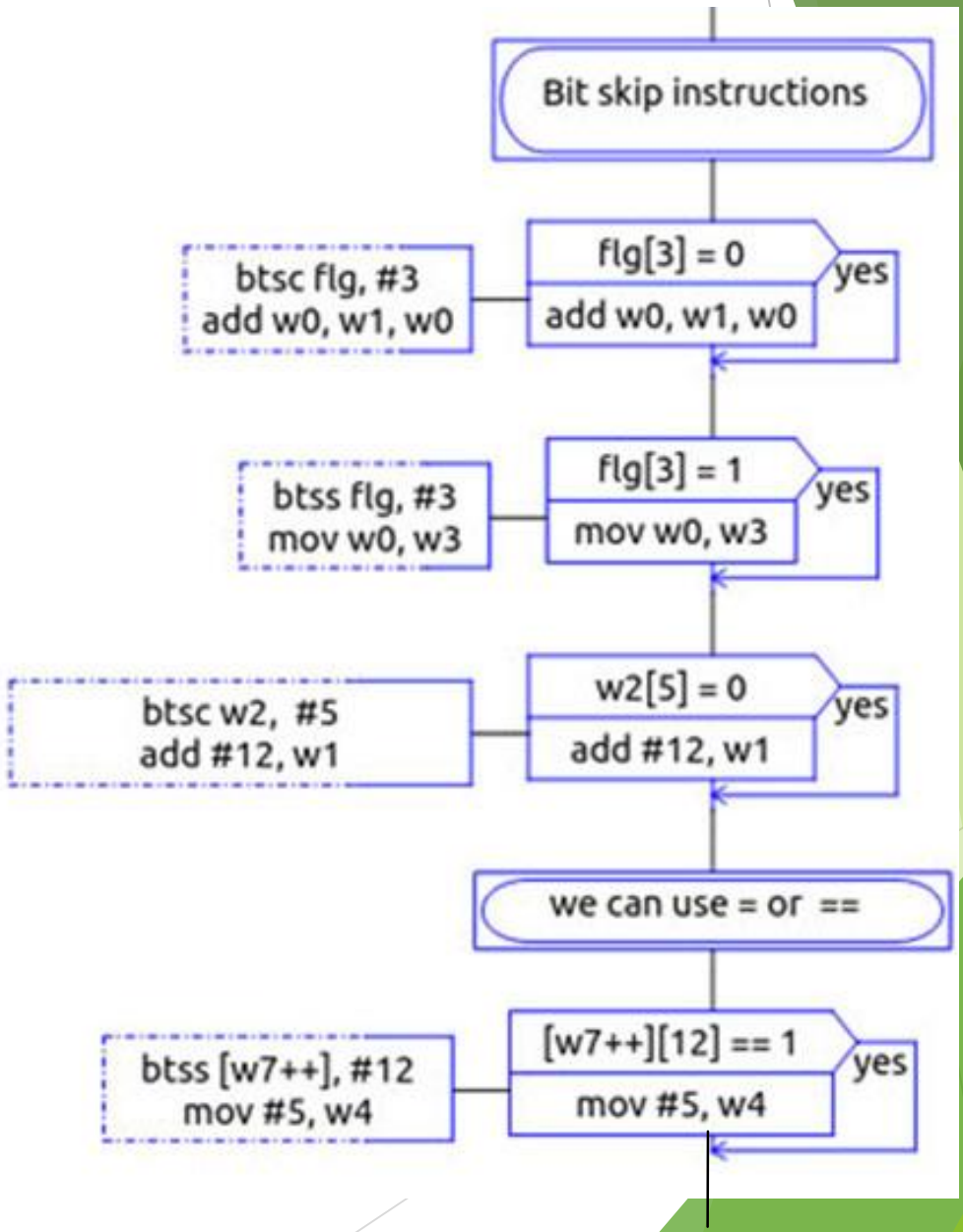
inventolabs

Инструкция Skip

Assembler

Operation	Operand 1	Op 2	Description
BTSC{.B}	f	#bit4	Bit test f, skip if clear
BTSC	Ws or [Ws]*	#bit4	Bit test Ws, skip if clear
BTSS{.B}	f	#bit4	Bit test f, skip if set
BTSS	Ws or [Ws]*	#bit4	Bit test Ws, skip if set

Visual Assembler

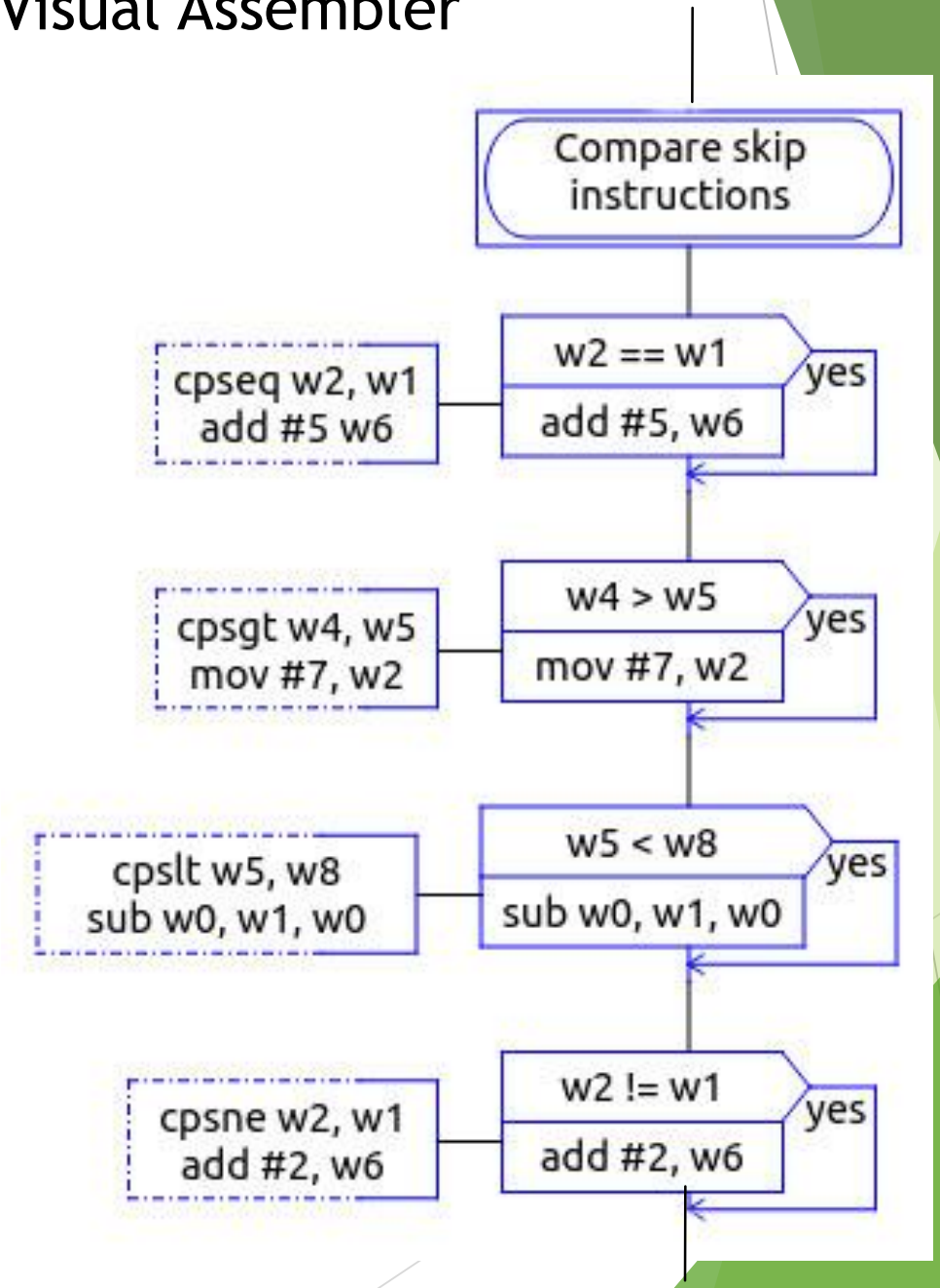


Skip instructions

Assembler

CPSEQ{.B}	Wb	Wn	Compare (Wb – Wn), skip if =
CPSGT{.B}	Wb	Wn	Compare (Wb – Wn), skip if >
CPSLT{.B}	Wb	Wn	Compare (Wb – Wn), skip if <
CPSNE{.B}	Wb	Wn	Compare (Wb – Wn), skip if ≠

Visual Assembler

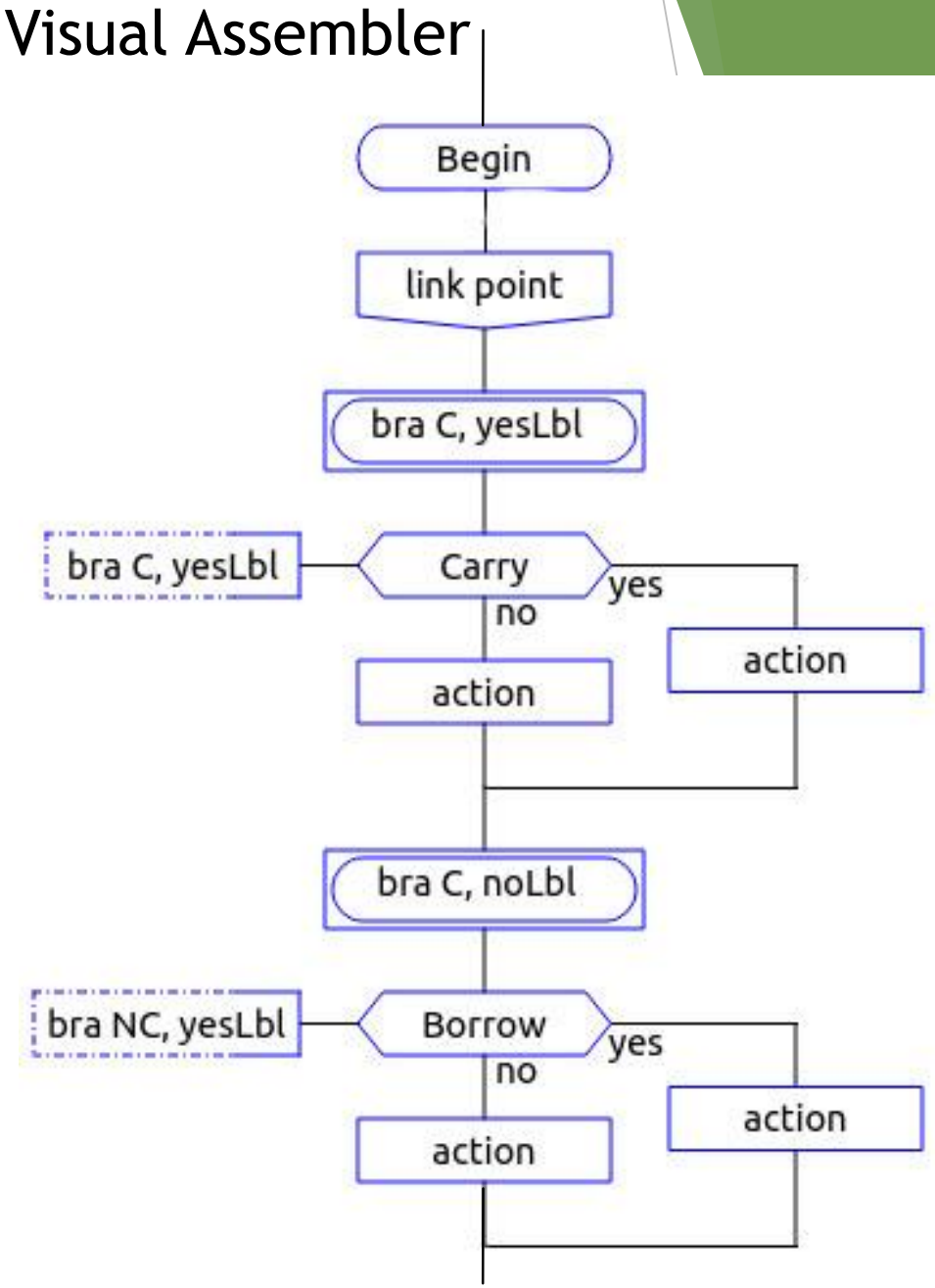


Инструкция условного перехода по флагу

Assembler

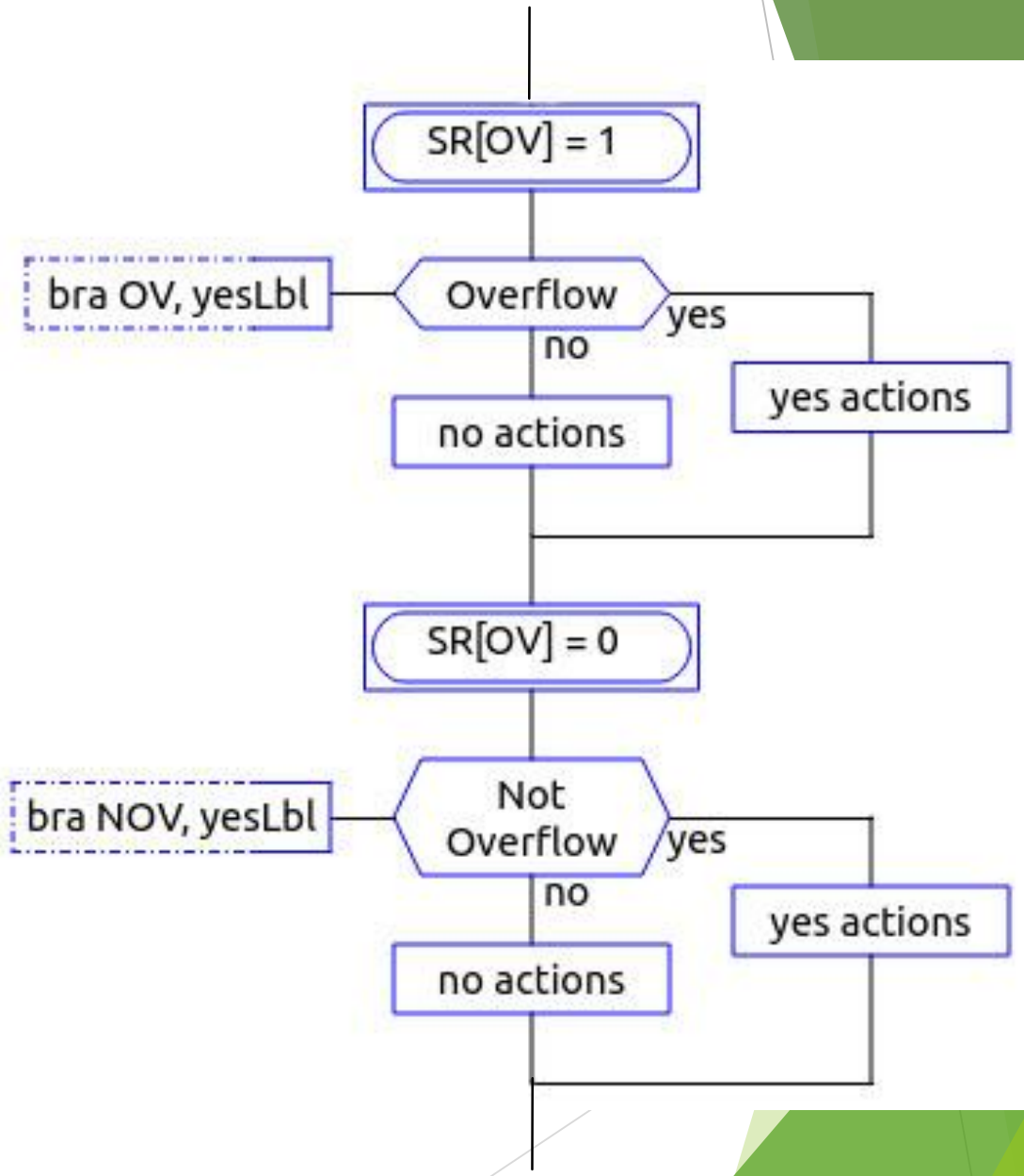
Operation	Operand 1	Op 2	Description
BRA	C	Expr	Branch if Carry (no Borrow)
BRA	NC	Expr	Branch if not Carry (Borrow)

Visual Assembler



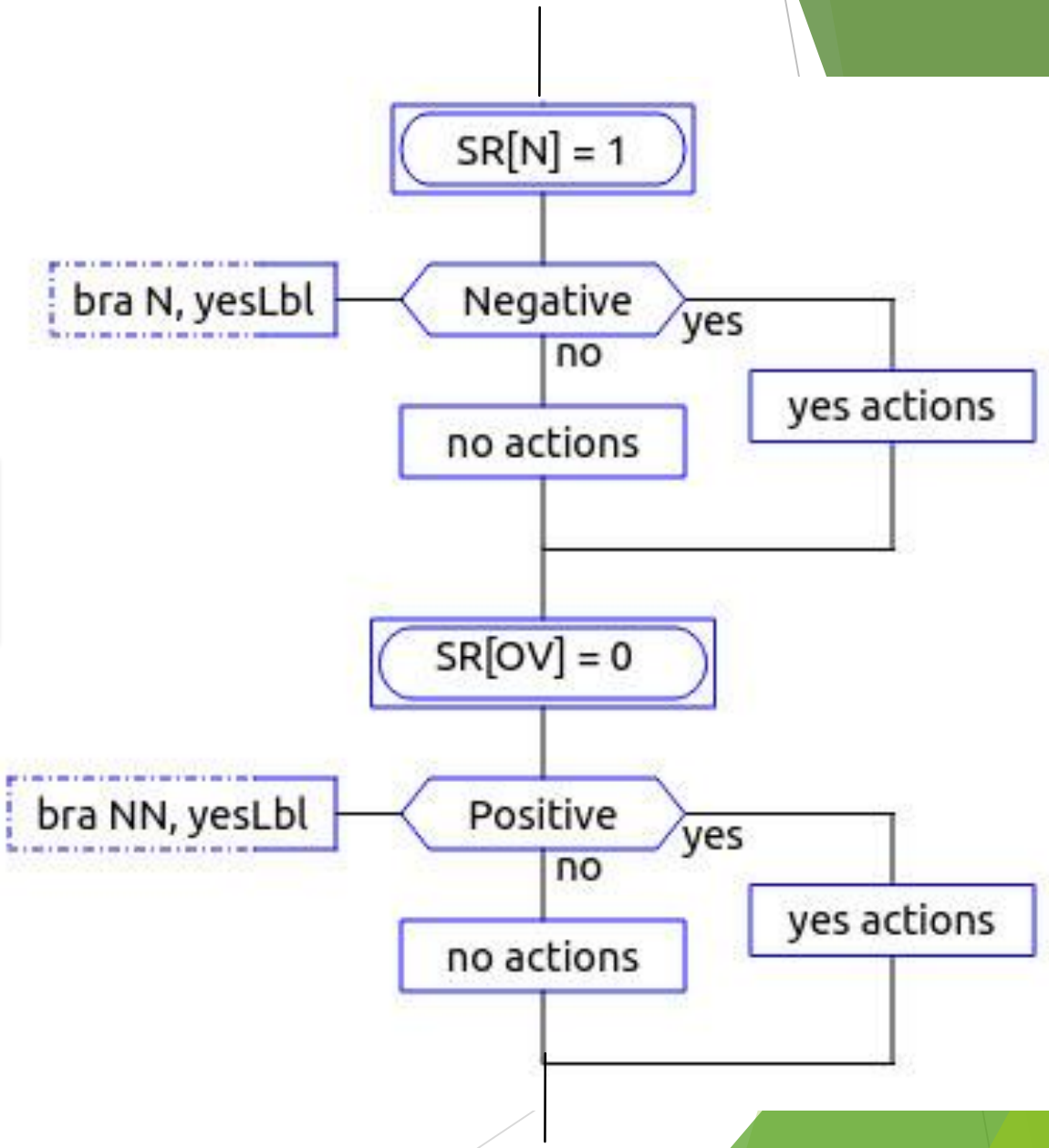
Branch flags

Operation	Operand 1	Op 2	Description
BRA	OV	Expr	Branch if Overflow
BRA	NOV	Expr	Branch if not Overflow



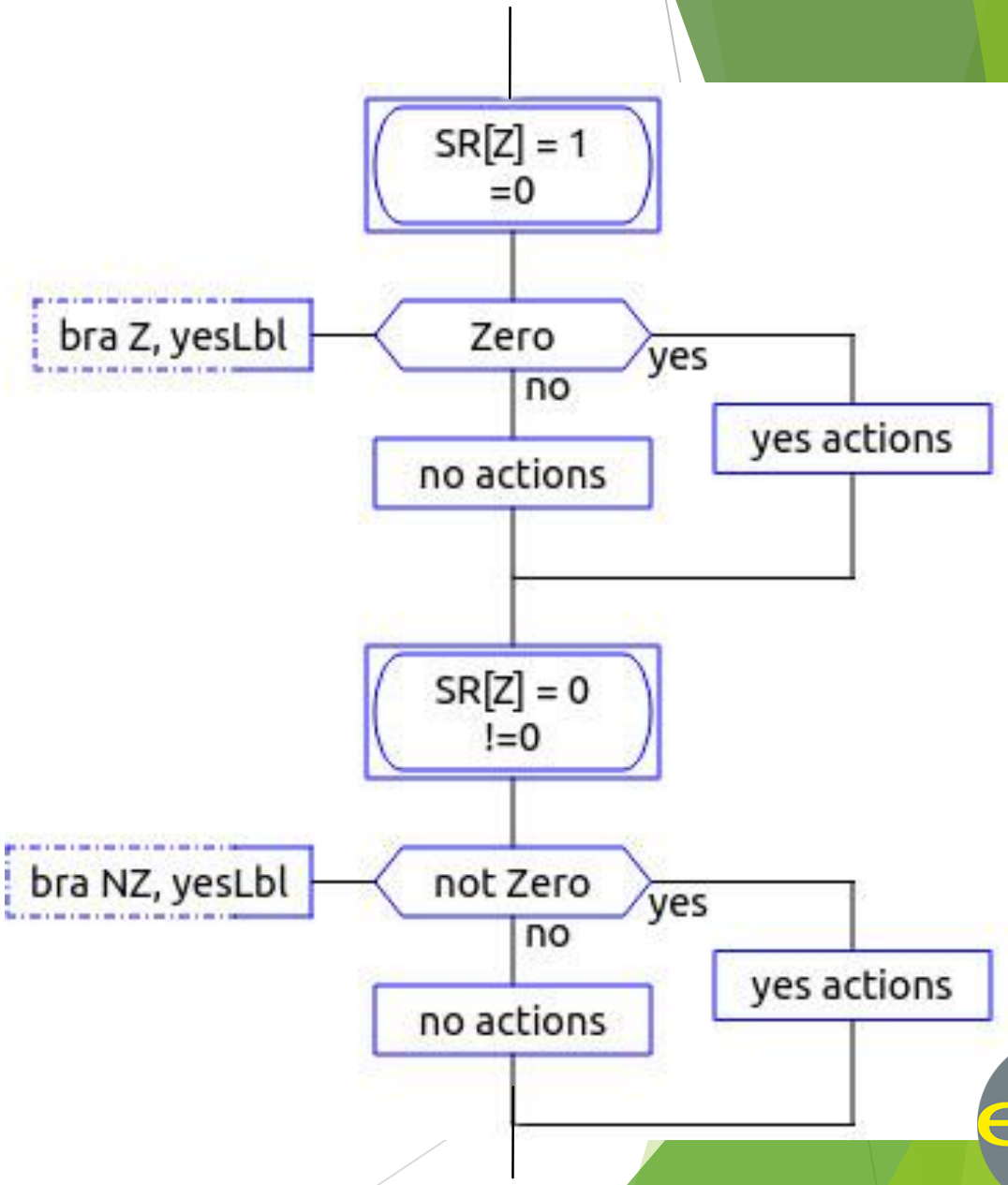
Branch flags

Operation	Operand 1	Op 2	Description
BRA	N	Expr	Branch if Negative
BRA	NN	Expr	Branch if not Negative



Branch flags

Operation	Operand 1	Op 2	Description
BRA	Z	Expr	Branch if Zero
BRA	NZ	Expr	Branch if not Zero

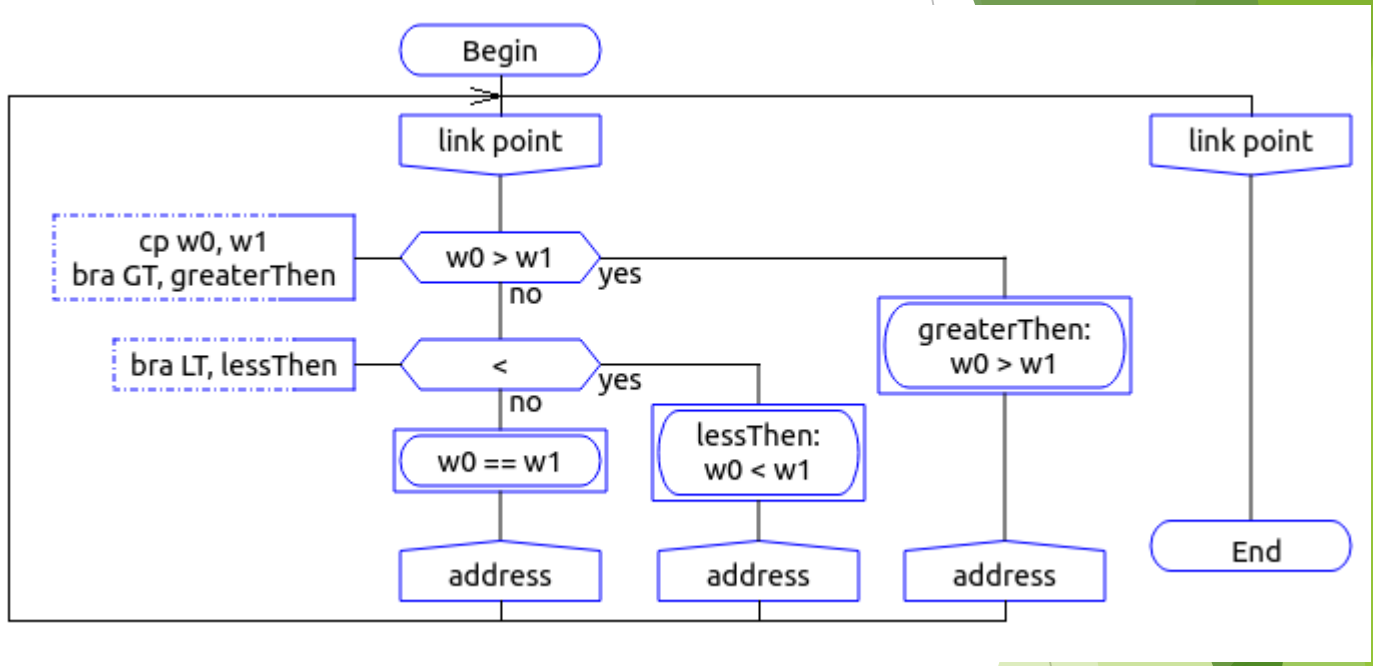


Инструкция условного перехода

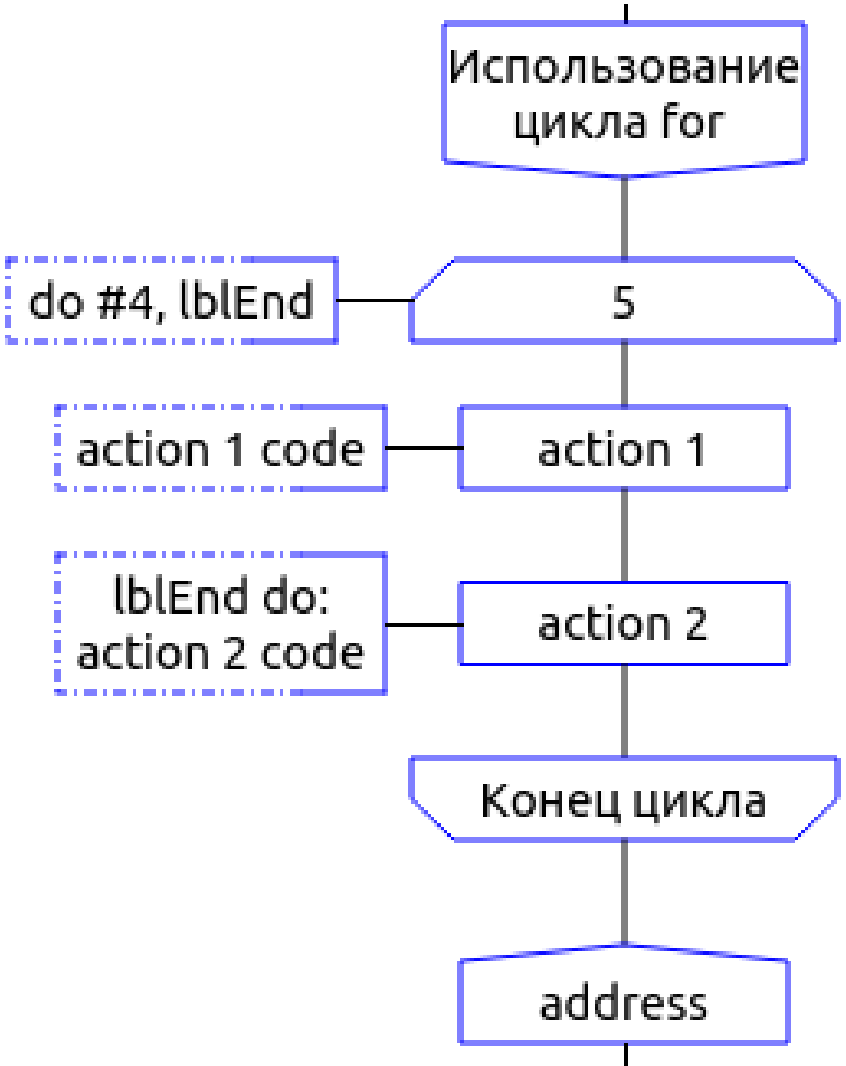
Знаковые	
Больше	>
Больше либо равно	>=
Меньше	<
Меньше либо равно	<=

Без знаковое:	
Больше	u>
Больше либо равно	u>=
Меньше	u<
Меньше либо равно	u<=

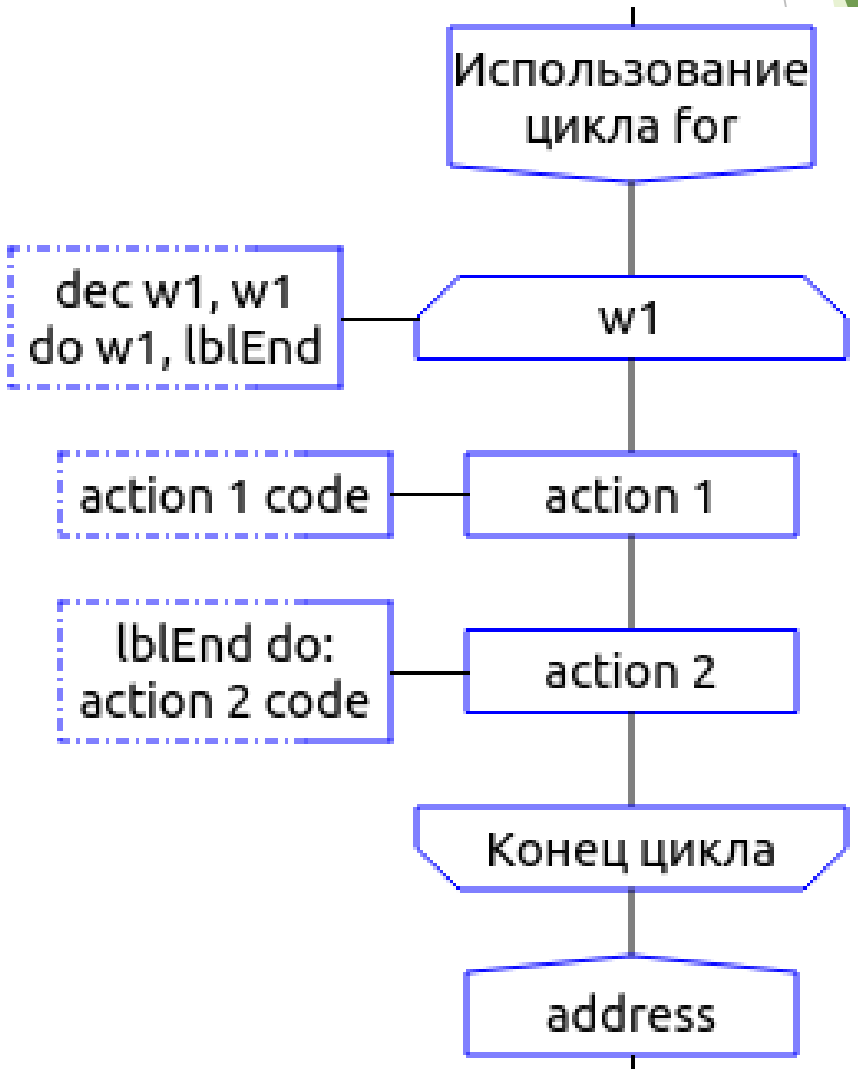
Равно	**== или =**
Не равно	**!=**



Пример цикла for с константой

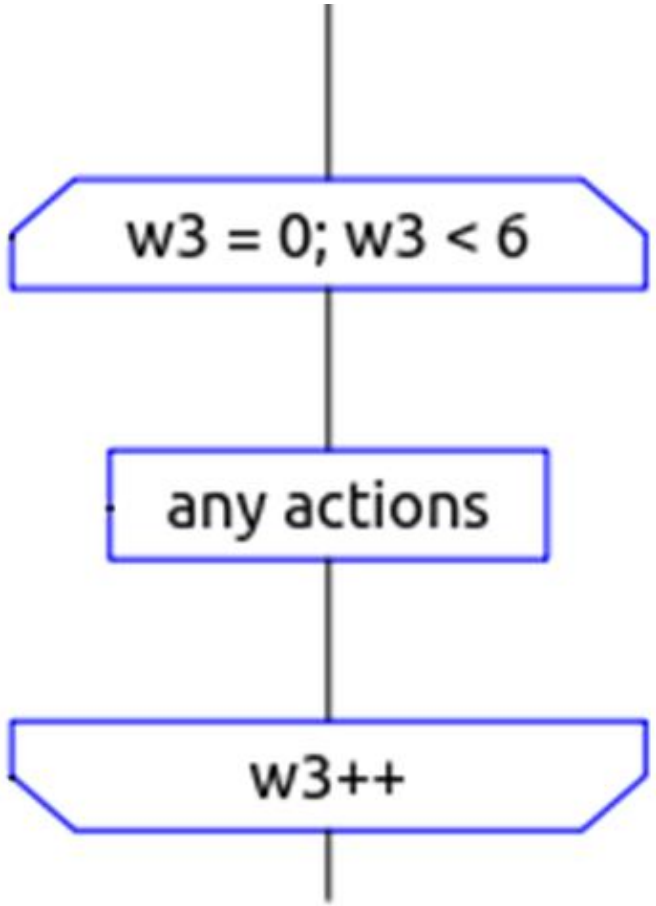


регистром

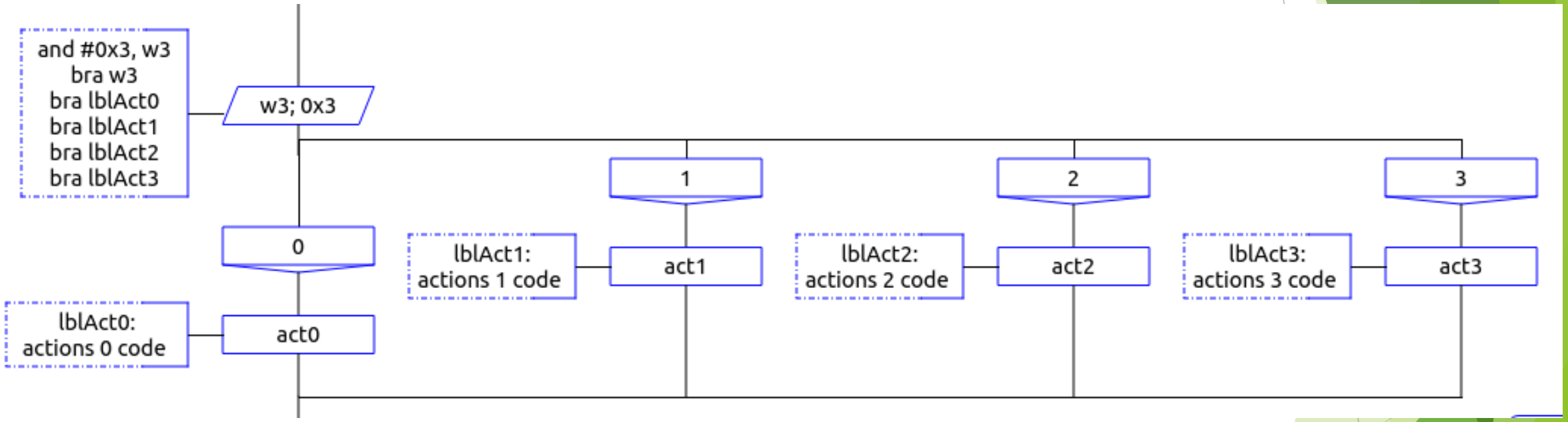


Пример стандартного цикла с условием

```
mov #0, w3
mov #6, w1
lblForBegin:

any actions code

inc w3, w3
cp w3, w1
bra LT, lblForBegin
```



Пример с оператором управления switch



Сложение с ячейкой памяти

Операция	Op1	Op2	Пояснение	Assembler	Visual assembler
INC{.B}	f	{WREG}	Destination = f + 1	inc f	f = f + 1 f += 1 ++f f++
				inc f, wreg	wreg = f + 1
INC2{.B}	f	{WREG}	Destination = f + 2	inc2 f	f = f + 2 f += 2
				inc2 f, wreg	wreg = f + 2
ADD{.B}	f	{WREG}	Destination = f + WREG	add f	f = f + wreg f += wreg
				add f, wreg	wreg = f + wreg wreg += f

Сложение с константой

Операция	Op1	Op2	Op3	Пояснение	Assembler	Visual assembler
ADD{.B}	#lit10	Wn		Wn = lit10 + Wn	add #27, w1	w1 = w1 + 27 w1 += 27
ADD{.B}	Wb	#lit5	Wd or [Ws]*	Wd = Wb + lit5	add w0, #8, w1	w1 = w0 + 8

Сложение с рабочими регистрами

Операция	Op1	Op2	Op3	Пояснение	Assembler	Visual assembler
INC{.B}	Ws or [Ws]*	Wd or [Ws]*		Wd = Ws + 1	asm: inc w1, w1	w1 = w1 + 1 w1 += 1 ++w1 w1++
					asm: inc w0, w2	w2 = w0 + 1
INC2{.B}	Ws or [Ws]*	Wd or [Ws]*		Wd = Ws + 2	asm: inc2 w1, w1	w1 = w1 + 2 w1 += 2
					asm: inc2 w0, w2	w2 = w0 + 2
ADD{.B}	Wb	Ws or [Ws]*	Wd or [Ws]*	Wd = Wb + Ws	asm: add w0, w5, w1 asm: add w0, [w5++], [w1]	w1 = w0 + w5 [w1] = w0 + [w5++]

Вычитание с ячейкой памяти

Операция	Op1	Op2	Пояснение	Assembler	Visual assembler
DEC{.B}	f	{WREG}	Destination = f – 1	dec f	f = f + 1 f += 1 ++f f++
				dec f, wreg	wreg = f + 1
DEC2{.B}	f	{WREG}	Destination = f – 2	dec2 f	f = f + 2 1.1 f += 2
				dec2 f, wreg	wreg = f + 2
SUB	f	{WREG}	Destination = f – WREG	sub f	f = f - wreg f -= wreg
				sub f, wreg	wreg = f — wreg
SUBR	f	{WREG}	Destination = WREG – f	sub f	f = wreg — f
				sub f, wreg	wreg = wreg — f wreg -= f

Вычитание с константой

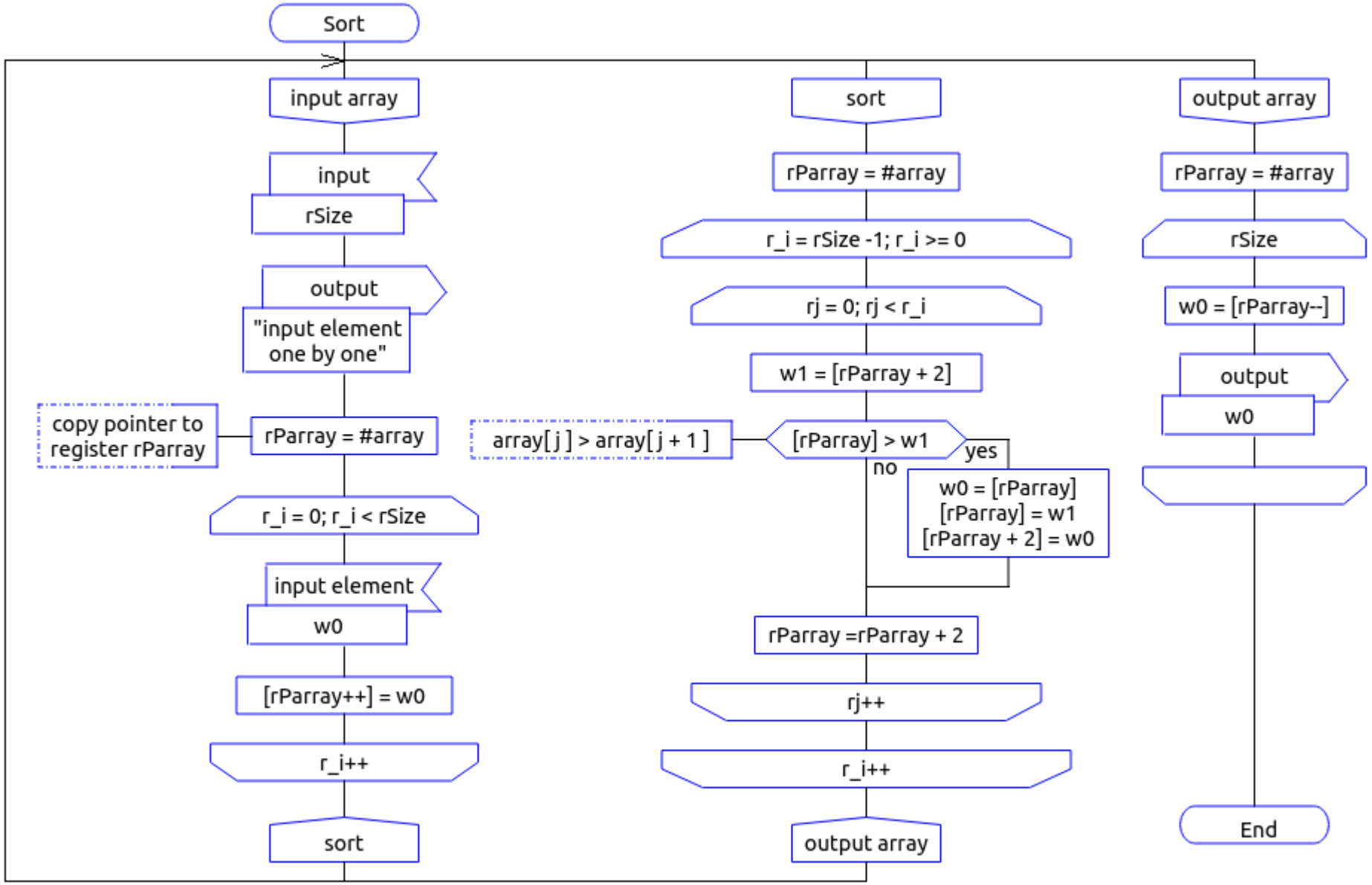
Операция	Op1	Op2	Op3	Пояснение	Assembler	Visual assembler
SUB	#lit10	Wn		Wn = Wn – lit10	sub #27, w1	w1 = w1 - 27 1.1 w1 -= 27
SUB	Wb	#lit5	Wd or [Ws]*	Wd = Wb – lit5	sub w0, #8, w1	w1 = w0 - 8
SUBR	Wb	#lit5	Wd or [Ws]*	Wd = lit5 – Wb	sub w0, #8, w1	w1 = 8 — w0

Вычитание с рабочими регистрами

Операция	Op1	Op2	Op3	Пояснение	Assembler	Visual assembler
DEC{.B}	Ws or [Ws]*	Wd or [Ws]*		Wd = Ws – 1	sub w1, w1	w1 = w1 - 1 w1 -= 1 --w1 w1--
					sub w0, w2	w2 = w0 - 1
DEC2	Ws or [Ws]*	Wd or [Ws]*		Wd = Ws – 2	sub2 w1, w1	w1 = w1 - 2 w1 -= 2
					sub2 w0, w2	w2 = w0 - 2
SUB	Wb	Ws or [Ws]*	Wd or [Ws]*	Wd = Wb – Ws	sub w0, w5, w1 sub w0, [w5++], [w1]	w1 = w0 - w5 [w1] = w0 - [w5++]
SUBR	Wb	Ws or [Ws]*	Wd or [Ws]*	Wd = Ws – Wb	sub w0, w5, w1 sub w0, [w5++], [w1]	w1 = w5 - w0 [w1] = [w5++] - w0

Пример на Visual Assembler

Алгоритм сортировки на “Visual Assembler”



Тот же алгоритм на сортировке на языке C++

```
int* array;
int size;

// init
cout<<"input size"<<endl;
cin>>size;
array = new int[size];
cout<<"input elements one by one"<<endl;
int tmp;
for(int i=0; i<size; i++){
  cin>>tmp;
  array[i]=tmp;
  cout<<endl;
}

// sort
for (int i = size - 1; i >= 0; i--)
{
  for (int j = 0; j < i; j++)
  {
    if (array[j] > array[j + 1])
    {
      int tmp = array[j];
      array[j] = array[j + 1];
      array[j + 1] = tmp;
    }
  }
}

// return
 for(int i=0; i<size; i++){
    cout<<array[i]<<endl;
 }
```

Выводы

- На “visual assembler” структура разбита на логические части.
- Основной алгоритм становиться наглядным.
- Для того чтобы подчеркнуться структуру алгоритма на C++ существует множества ключевых слов и открывающихся и закрывающихся фигурных скобок.
- На “visual assembler” эти ключевые слова полностью заменены на графические символы
- Компактность когда.

Дракон является графическим языком для объяснения алгоритмов.

Это инструмент для легкого и быстрого понимания между людьми, когда они говорят о программах. Лозунг ДРАКОНА "взглянул – понял".

Он пришёл из космоса

Дракон был разработан в рамках космической программы «Буран» при участии Федерального космического агентства и Российской академии наук.

Понимание — это ключ к созданию программ.

Лёгкое чтение кода программы является основой хорошего программного процесса.

- Очень важно заметить ошибки на ранних стадиях программирования перед запуском программы. Значение этого факта невозможно недооценивать. Нет необходимости тратить долгие часы на отладку и тестирование. В результате тратиться гораздо меньше времени на написание самой программы.

- Большой программный проект как облако информации постоянно растущий создаётся многими людьми. Каждое новое пополнение должно соответствовать старым требованиям. Новый код должен работать хорошо с существующем кодом. Понимание того, что другие уже сделали имеет важное значение для работы в крупном проекте. Следовательно, люди сотрудничают более эффективно, если проект легко понять.

Понимание, восприятие должно стать проще.

С какими трудностями связано восприятие программ?

1. Программы становятся сложнее
2. Их становится тяжелее понять.
3. Понимание код - работа
4. Продуктивность такой работы непростительно низкая.

Вывод: необходимо сделать восприятие кода проще.

Как этого добиться?

Инновационность ДРАКОН-а в упрощение восприятия программ основывается на математически строгости и эргономичности.

С одной стороны ДРАКОН математически строг, что означает:

1. точность и недвусмысленность
2. доказано, что, используя ДРАКОН, можно представить ЛЮБОЙ алгоритм

С другой стороны одной математической строгости недостаточно, поэтому ДРАКОН так сконцентрирован на удобности и эргономичности.

Что такое эргономика?

Это искусство превращения изматывающе скучной работы в легкую и удобную.
Суть эргономики - делать вещи удобными для людей.
Эргономичность дракона:

1. ДРАКОН прост.
2. ДРАКОН графический язык.
3. Нотация ДРАКОН-а оптимизирована.

Почему Графический язык?

В текущей момент текст является основной формой представления алгоритмов, и все воспринимают это как естественный и неизбежный факт. Диаграммы используются
от случая к случаю.
Это - проблема, потому что:

1. Человеческие глаза и мозг оптимизированы под восприятие визуальной информации, составляющей большую часть информации получаемой человеком. Мы хуже воспринимаем текст и лучше воспринимаем картинки. Для повышения нашей продуктивности мы должны использовать наши тела и разумы наиболее подходящим способом.
2. Изображение - более компактный способ представления информации, чем текст.
3. Инженеры давно перешли к использованию изображений. они построили мир современных технологий используя изображения и чертежи. Чем хуже разработчики программного обеспечения? Будущее за визуальными языками, так