

Богатов Р.Н.

# Программирование на языке высокого уровня

---

Лекция 4.

Низкоуровневое программирование.

Интерпретация, компиляция, компоновка.

Кафедра АСОИУ ОмГТУ, 2012

# Изобретём велосипед компьютер!

- **Примитивная машина:**

- Ячейки памяти нумеруются 0, 1, ...
- Ячейка может содержать любое число (в т.ч. адрес ячейки памяти)
- Процессор имеет один регистр R (ячейку для хранения промежуточных результатов)

- **Примитивный низкоуровневый язык:**

Инструкция	Действие
<code>move</code>	Запись значения <куда, что>
<code>add</code>	Сложение <куда, сколько>
<code>sub</code>	Вычитание <откуда, сколько>
<code>mult</code>	Умножение <что, на сколько>
<code>div</code>	Деление <что, на сколько>
<code>loop</code>	Цикл <счётчик, адрес начала>
<code>halt</code>	Останов
<code>goto</code>	Переход
<code>call</code>	Вызов процедуры
<code>ret</code>	Возврат из процедуры

- **Пример программы:**

```
// ввод значения с клавиатуры
call    Input_Register
// регистр содержит радиус
move    Radius, R
mult    R, Radius
mult    R, 3.1415
// регистр содержит площадь круга
// вывод содержимого регистра на экран
call    Print_Register
halt
```

# Машинные коды

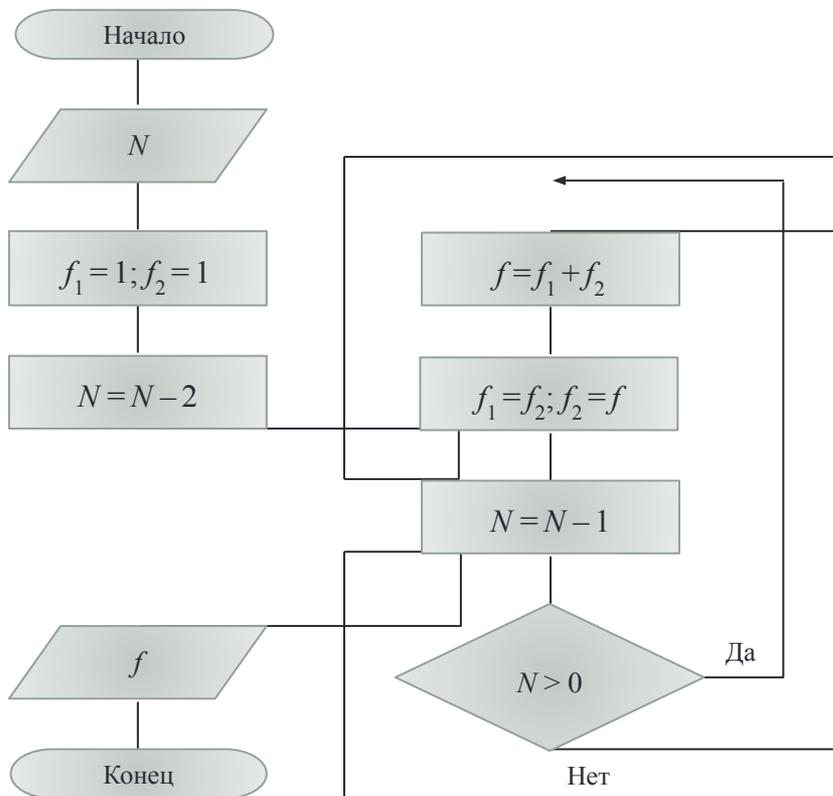
Инструкция	Действие	Синтаксис	Машинный код
move	Запись значения <куда, что>	move R, число move R, [адрес] move [адрес], число move [адрес], R move [адрес], [адрес]	4, число 5, адрес 6, адрес, число 7, адрес 8, адрес, адрес
add	Сложение <куда, сколько>	add R, число add R, [адрес] add [адрес], число add [адрес], R add [адрес], [адрес]	9, число 10, адрес 11, адрес, число 12, адрес 13, адрес, адрес
sub	Вычитание <откуда, сколько>	sub R, число sub R, [адрес] sub [адрес], число sub [адрес], R sub [адрес], [адрес]	14, число 15, адрес 16, адрес, число 17, адрес 18, адрес, адрес
mult	Умножение <что, на сколько>	...	19-23
div	Деление <что, на сколько>	...	24-28
loop	Цикл <счётчик, адрес начала>	loop R, [адрес] loop [адрес], [адрес]	29, адрес 30, адрес, адрес
halt	Останов	halt	0
goto	Переход	goto [адрес]	1, адрес
call	Вызов процедуры	call [адрес]	2, адрес
ret	Возврат из процедуры	ret	3

- Пример программы, понятной процессору:**

Адрес	+0	+1	+2	+3	+4
0	8	112	27	8	113
5	28	8	114	29	2
10	101	10	29	10	6
15	30	3	8	130	27
20	8	131	28	8	132
25	29	8	133	30	2
30	115	...	...	...	...
35	...	...	...	...	...

# Ещё пример: числа Фибоначчи

## Схема алгоритма и его реализация (программа):



```
// ввод N с клавиатуры
call   Input_Register
// регистр содержит N
move   N, R
// инициализация
move   f1, 1
move   f2, 1
// основная часть
subN, 2
move   R, f2
A: // начало цикла
addR, f1
move   f1, f2
move   f2, R
loop   N, A // конец цикла
// регистр содержит искомое,
// выводим его на экран
call   Print_Register
halt
```

# Компиляция...

1. Проверка синтаксиса
2. Трансляция в машинные коды

```
call Input_Register
```

Адрес	Содержимое ячеек	Исходные инструкции
0	2, [Input_Register]	<code>call Input_Register</code>
2	7, 28	<code>move N, R</code>
4	6, 29, 1	<code>move f1, 1</code>
7	6, 30, 1	<code>move f2, 1</code>
10	16, 28, 2	<code>sub N, 2</code>
13	5, 30	<code>move R, f2</code>
15	10, 29	<code>A: add R, f1</code>
17	8, 29, 30	<code>move f1, f2</code>
20	7, 30	<code>move f2, R</code>
22	30, 28, 15	<code>loop N, A</code>
25	2, [Print_Register]	<code>call Print_Register</code>
27	0	<code>halt</code>
28	0	<code>N</code>
29	0	<code>f1</code>
30	0	<code>f2</code>

# Компоновка (связывание)...

## 1. К программе добавляется код процедур (компоновка)

Адрес	Содержимое ячеек	Исходные инструкции
0	2, 31	call Input_Register
2	7, 28	move N, R
4	6, 29, 1	move f1, 1
7	6, 30, 1	move f2, 1
10	16, 28, 2	sub N, 2
13	5, 30	move R, f2
15	10, 29	A: add R, f1
17	8, 29, 30	move f1, f2
20	7, 30	move f2, R
22	30, 28, 15	loop N, A
25	2, 58	call Print_Regist
27	0	halt
28	0	N
29	0	f1
30	0	f2
31	...	...
...	...	...
57	3	ret
58	...	...
...	...	...
73	3	ret

## 2. Производится связывание адресов

Адрес	+0	+1	+2	+3	+4
0	2	31	7	28	6
5	29	1	6	30	1
10	16	28	2	5	30
15	10	29	8	29	30
20	7	30	30	28	15
25	2	58	0	0	0
30	0	...	...	...	...
35	...	...	...	...	...
40	...	...	...	...	...
45	...	...	...	...	...
50	...	...	...	...	...
55	...	...	3	...	...
60	...	...	...	...	...
65	...	...	...	...	...
70	...	...	...	3	...

# Интерпретация. Трансляция. Компиляция

- **Трансляция программы** — преобразование программы, представленной на одном из языков программирования, в равнозначную программу на другом языке.
- **Компиляция** — трансляция программы в машинно-ориентированный язык.
- **Интерпретация** — пооператорная обработка и выполнение исходной программы (без формирования кодов для последующего исполнения).

## Компоновка

- **Исполнимый модуль** (*от англ. executable*) — файл, содержащий машинные коды, готовые для исполнения в определённой операционной системе.
- **Объектный модуль** (*англ. object file*) — файл с промежуточным представлением отдельного модуля программы, полученный в результате работы компилятора.
- **Компоновщик** (также **редактор связей**, линкер — *от англ. link editor, linker*) — программа, которая производит компоновку: принимает на вход один или несколько объектных модулей и собирает по ним исполнимый модуль.

# Эволюция «уровня» языка программирования

## Машинные коды

Адрес	+0	+1	+2	+3	+4
0	8	112	27	8	113
5	28	8	114	29	2
10	101	10	29	10	6
15	30	3	8	130	27
20	8	131	28	8	132
25	29	8	133	30	2
30	115	...	...	...	...
35	...				

## Ассемблер

```
// прячем окружность
move    hide_x, c_x
move    hide_y, c_y
move    hide_R, c_R
call    Circle_Hide
// меняем радиус и цвет
addc_R, 10
move    c_Color, 3
// рисуем окружность
move    show_x, c_x
move    show_y, c_y
move    show_R, c_R
move    show_C, c_Color
call    Circle_Show
```

## Домохозяйка 2.0

Кружочек:  
Спрятаться  
Увеличиться чуть-чуть  
Цвет красный  
Показаться

## LazyTalk 0.99beta

```
c:  
Hide  
Grow Small  
Color Red  
Show
```

## Фантаст 47.0.2RC

Кружочек,  
стань чуть больше  
и красным

## C++, Java, C# и т.п.

```
c.Hide();  
c.R += SizeDelta.Small;  
c_Color = Colors.Red;  
c.Show();
```

## Си (без структур)

```
Circle_Hide(c_x, c_y, c_R);  
c_R = c_R + 10;  
c_Color = 3;  
Circle_Show(c_x, c_y, c_R, c_Color);
```

# Домашнее задание

- **Написать программу в машинных кодах**

для вычисления суммы ряда  $1 + \frac{1}{1!} + \frac{1}{2!} + \dots + \frac{1}{n!}$  для заданного  $n$ .

(При  $n \rightarrow \infty$  сумма ряда сходится к числу Эйлера  $e$ .)

- **Пишем программу от простого к сложному:**

- Получить в R значение  $4!$  (не используя цикл **loop**) и вывести его на экран
- Допisać программу, чтобы полученное в R домножалось на 5 и давало  $5!$
- Изменить: полученное  $4!$  сохранить в переменную (например: **move Z, R**), а после получения в регистре  $5!$  добавить результат к переменной (**add Z, R**). (Теперь программа позволяет получать суммы вида  $1! + 2! \dots + n!$ , но для фиксированного  $n$  и без использования цикла.)
- Потренироваться создавать циклы: пользователь вводит  $n$ , запоминаем это значение в переменной N и делаем цикл **loop N, A**, где A – метка, к которой будет происходить возврат (см. пример). Команда **loop** работает так: если  $N \neq 0$ , то N уменьшается на единицу и происходит переход к метке A; иначе **loop** ничего не делает, возврата не происходит.
- Теперь внутри цикла помещаем код, который умеет из предыдущего факториала (например, содержащегося в R) получать следующий (путём домножения R на текущее значение N) и добавлять его к уже накопленной сумме (Z). (Теперь программа вычисляет суммы вида  $1! + 2! \dots + n!$  с помощью цикла для любого  $n$ , введённого с клавиатуры.)
- Осталось подумать, как внутри цикла получать дробь, и добавить к сумме лишнюю единицу