Лектор: А.Д.Хапугин

# Основы программного конструирования

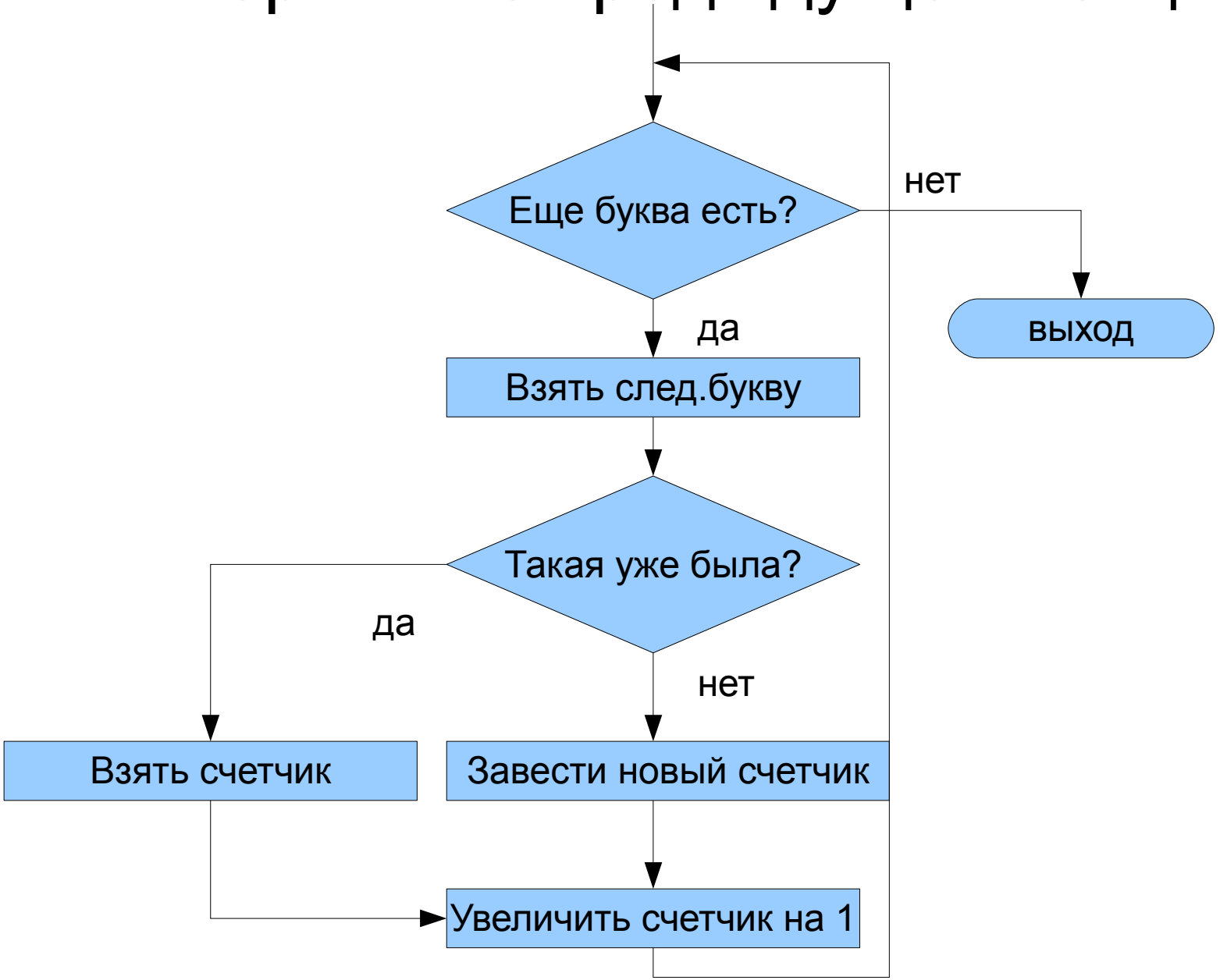
Лекция 4. Общие понятия архитектур ЭВМ - продолжение

Материалы доступны в Интернете по адресу: http://www.excelsior.ru/afti/

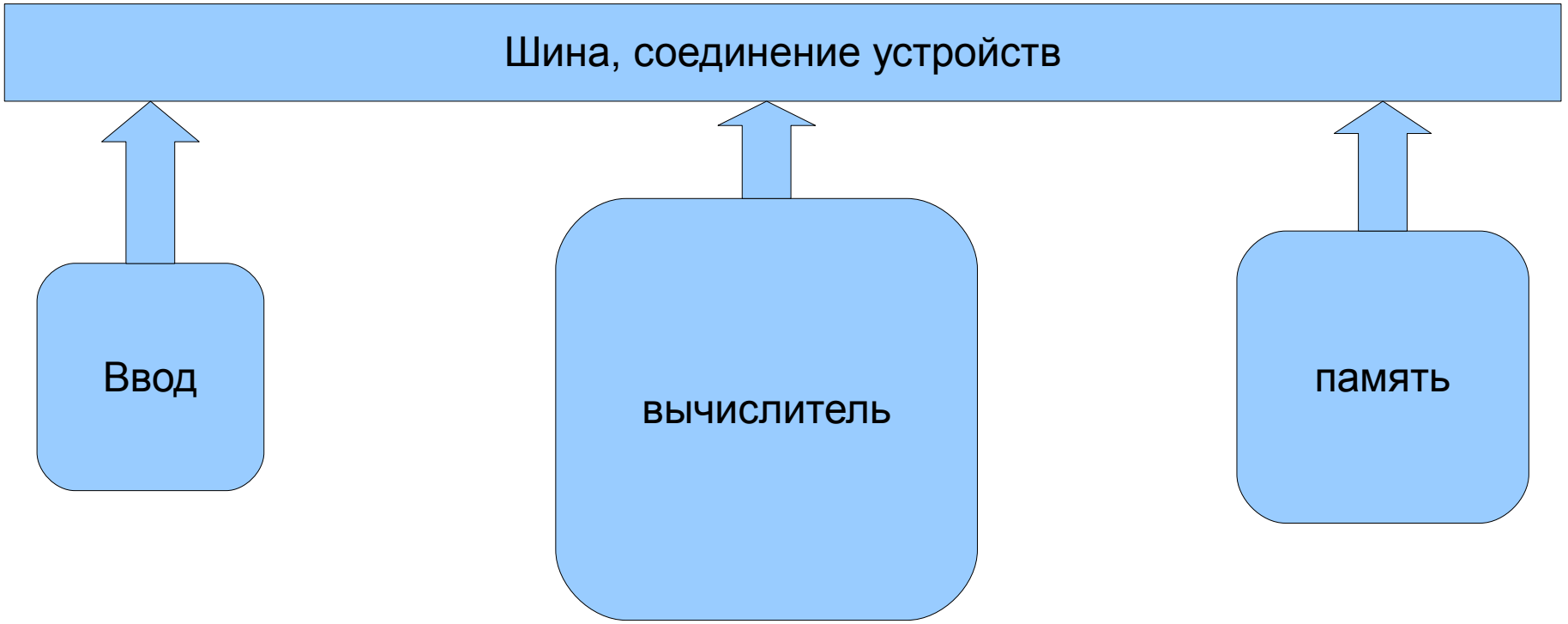
# Продолжаем придумывать устройство компьютера

Задача: для алгоритма, разработанного на прошлой лекции для статистического анализа шифрованных сообщений надо придумать набор команд, которого было бы достаточно для его реализации. При этом нужно не упускать из виду вторую цель проекта: сделать систему универсальной, то есть обеспечить возможность реализовывать любые алгоритмы без переделки аппаратуры.

# Алгоритм из предыдущей лекции

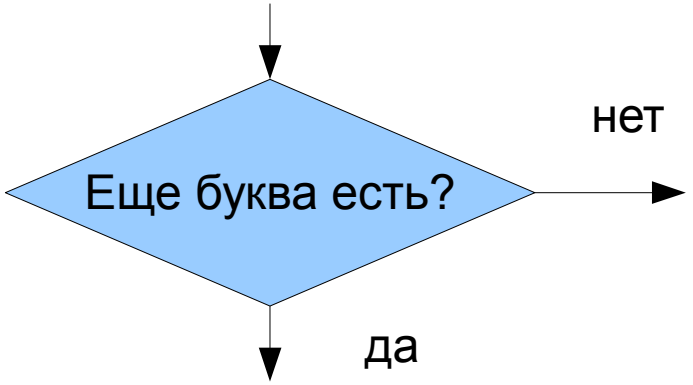


# Схема устройства системы из предыдущей лекции




Команды хранятся в памяти вместе с данными.

# Первый шаг алгоритма

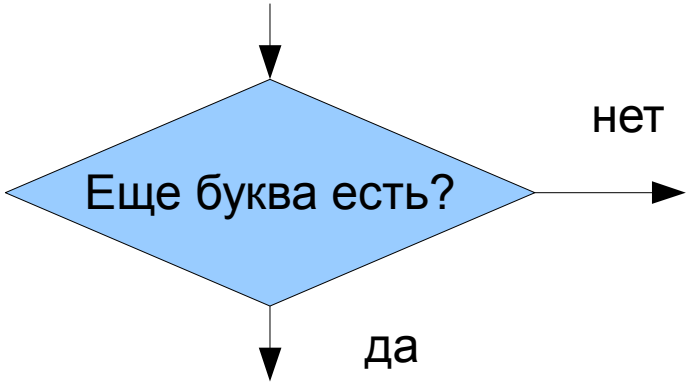



Проблема №1: этот шаг алгоритма слишком конкретный. Делать для него одну отдельную команду — значит терять универсальность. Решение: для этого шага алгоритма скорее всего потребуется несколько команд. Сначала нужно ввести символ.

Проблема №2: нужно отличать конец сообщения от других символов. Решение: договариваемся, что символ с кодировкой 0 («ноль») будет обозначать конец сообщения. Соответственно устройства ввода должны работать в соответствии с этой договоренностью.

Проблема №3: непонятно, откуда процессор вообще будет брать команду. В этом месте дискуссии снова возникали предложения отказаться от хранения команд вместе с данными в одной памяти. Решили: отказываться не будем, а для разрешения противоречий договариваемся: при включении питания процессор сразу начинает исполнять команду, лежащую по адресу «ноль». При этом устраиваем систему так, что до включения питания мы можем положить в память все, что нам нужно.

# Первый шаг алгоритма - 2




## Команда «Читать с устройства ввода»

**Формат команды** (то есть то, как она представляется в памяти: [0] (ровно одно слово «ноль»)

**Исполнение:**

Считывает что-то (символ или признак завершения сообщения) с устройства ввода.

Снова проблема: что делать со считанным из УВВ словом?

Нужно сохранить. Куда?

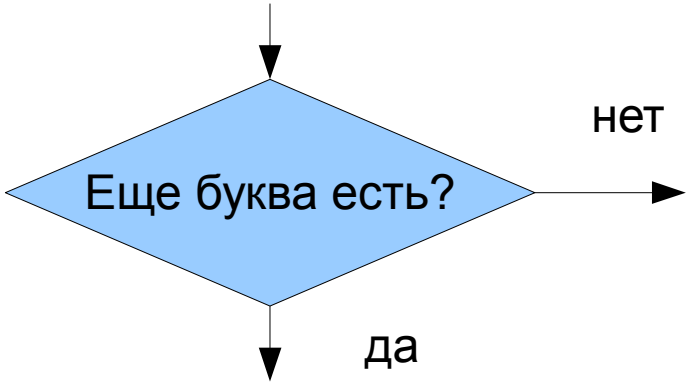
Решаем: в первое свободное слово в памяти. Что такое «первое свободное слово в памяти», пока непонятно, но решили, что уточним позднее.

Уточняем описание исполнения:

**Исполнение (уточненное):**

Считывает что-то (символ или признак завершения сообщения) с устройства ввода и сохраняет в первое свободное слово в памяти.

# Первый шаг алгоритма - 3




Символ считан, что дальше?
Нужно сравнить с признаком окончания сообщения.

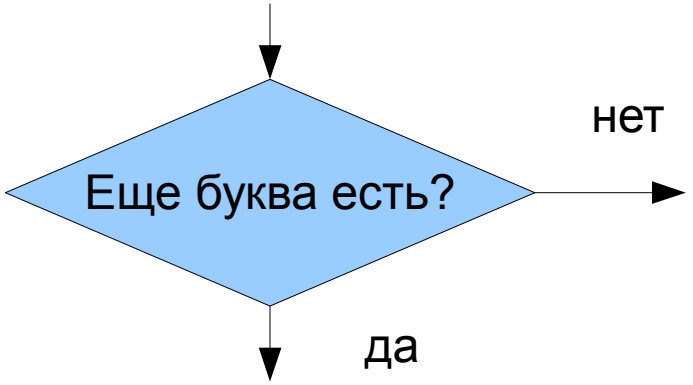
Вводим команду сравнения.
Код команды — 1.

Проблема №1: как указать что и с чем сравнивать?
Решаем, что сразу за кодом команды будем укладывать 2 слова, в которых будут храниться адреса слов в памяти, где лежит «что» и «с чем».
Проблема №2: А что делать с результатом?
Решаем, что результат будем сохранять в памяти, для чего добавим в формат команды третье слово. Процессор должен понимать, что выбрав команду с кодом «1» из памяти, он должен выбрать еще три слова за ним и использовать как операнды.

## Команда «сравнить на равенство»

**Формат:** [1] [AdrOp1] [AdrOp2] [AdrRes]
**Исполнение:** из памяти выбираются значения слов по адресам AdrOp1 и AdrOp2, сравниваются между собой и результат сравнения (как-то представленные значения для «да» или «нет») сохраняется в память по адресу AdrRes.

# Первый шаг алгоритма - 4




Символ считали, сравнили с нулем, что дальше?
Нужна команда, соответствующая выбору стрелочки «да» или «нет» на картинке.

Вводим команду ветвления.
Код команды — 2.

Проблема №1: как указать то место «куда» переходить?
Решение: поскольку команды лежат в памяти, то это место можно просто обозначить адресом.

## Команда «условное ветвление»

**Формат:** [2] [AdrCond] [AdrNext1] [AdrNext2] (трехоперандная)
**Исполнение:** из памяти выбираются значение слова по адресу AdrCond. Если считано слово, обозначающее «да», то следующую команду брать из адреса AdrNext1, а если нет — то из AdrNext2.

# Первые итоги

## Команда «Читать с устройства ввода»

**Формат команды:** [0]
**Исполнение:** Считывает что-то (символ или признак завершения сообщения) с устройства ввода и сохраняет в первое свободное слово в памяти.

## Команда «сравнить на равенство»

**Формат:** [1] [AdrOp1] [AdrOp2] [AdrRes]
**Исполнение:** из памяти выбираются значения слов по адресам AdrOp1 и AdrOp2, сравниваются между собой и результат сравнения (как-то представленные значения для «да» или «нет») сохраняется в память по адресу AdrRes.

## Команда «условное ветвление»

**Формат:** [2] [AdrCond] [AdrNext1] [AdrNext2] (трехоперандная)
**Исполнение:** из памяти выбираются значение слова по адресу AdrCond. Если считано слово, обозначающее «да», то следующую команду брать из адреса AdrNext1, а если нет — то из AdrNext2.

Попытаемся записать программу.

# Программа, первая попытка

```
(0) = [0]
(1) = [1] [AdrOp1] [AdrOp2] [AdrRes]
(5) = [2] [AdrCond] [AdrNext1] [AdrNext2]
```

Все бы хорошо, но вместо обозначений операндов мы должны в память укладывать конкретные числа — адреса или значения.

Проблема №1: нужно договариваться, как именно для этой программы мы распределим память, то есть в какое место памяти что положим.
Решаем так: адреса от 0 до 499 — отводим под код, с 500 — под данные.
Подробнее:
(500) — храним последний считанный с УВВ символ
(501) — храним ноль (для сравнения)
(502) — сохраняем результат сравнения символа с нулем

В области кода нужно «отвести» пару подобластей: куда ответвимся если конец сообщения (выбираем адрес 100) и куда — если не конец (выбираем адрес 200).

Тогда программа перепишется так:
... см. дальше...

# Программа

Тогда программа перепишется так:

```
(0) = 0
(1) = 1 500 501 502
(5) = 2 502 100 200
```

**В программе есть ошибка!!!**
Первая команда вводит символ с УВВ в то место, которое мы назвали «первым свободным словом памяти» и еще не уточнили что это означает. А вторая команда ожидает, что он будет по адресу 500.

Решение: изменяем команду ввода:

## Команда «Читать с устройства ввода»

**Формат команды:** [0] [Adr]
**Исполнение:** Считывает что-то (символ или признак завершения сообщения) с устройства ввода и сохраняет по адресу Adr.

И переписываем программу:

```
(0) = 0 500
(1) = 1 500 501 502
(5) = 2 502 100 200
```

**Теперь работать будет правильно!**

# Оптимизация

Возникло предложение отменить третий операнд у команды ветвления. Вместо него использовать просто адрес следующей команды за командой ветвления. Это экономит память.
Изменяем описание:

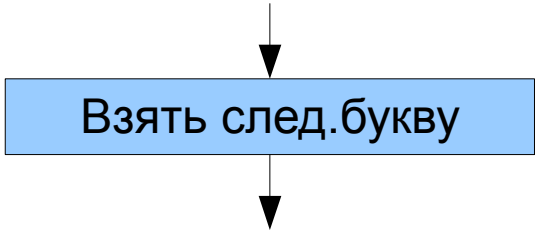
## Команда «условное ветвление»

**Формат:** [2] [AdrCond] [AdrNext1]
**Исполнение:** из памяти выбирается значение слова по адресу AdrCond. Если считано слово, обозначающее «нет», то следующую команду брать из адреса AdrNext1, а если «да» — то как обычно, следующую команду за этой.

Изменяем программу слева в скобках адрес первого слова памяти, занятого данной командой):

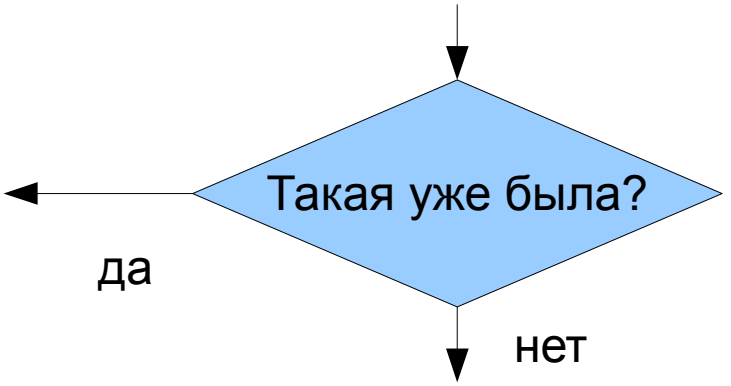
```
(0) = 0 500
(1) = 1 500 501 502
(5) = 2 502 100
```

# Второй шаг алгоритма




Этот шаг уже выполнен. Следующая буква лежит по адресу 500.
Новая команда не нужна.

# Следующий шаг




Проблема: нужно как-то определить что значит «такая буква уже была». Это явно связано с тем, как мы определим что значит «взять счетчик» или «завести новый счетчик» из последующих шагов алгоритма.

Вывод: нужно сначала решить как мы представим счетчики в памяти.

# Предложение 1

В памяти укладываем последовательность пар слов вида: (символ, счетчик), начиная с какого-то адреса (например, 503). Конец последовательности обозначаем символом «ноль».
Например, так (для сообщения «Hello, World!»):

```
(503) = [«H»] [1]
(505) = [«e»] [1]
(507) = [«l»] [3]
...
(517) = [«!»] [1]
(519) = [0]
```

Тогда «такая буква есть» - это будет последовательный просмотр последовательности, сравнение первого слова из пары (символа) с тем, что считали сейчас, и если равно, то вот он счетчик — в следующем слове. Если не равно, идем дальше, пока не наткнемся на ноль.
Если наткнулись на ноль, то вместо него положим новый символ, следом за ним ноль (значение нового счетчика равно нулю), и затем еще один ноль (признак конца последовательности).

Идея выглядит правильной, должно работать.

# Предложение 2

Вспоминаем, что у нас символы кодируются числами и количество возможных символов ограничено.
Делаем структуру проще: в памяти отводим сразу место под хранение всех возможных счетчиков, причем укладываем их один за другим (например, с адреса 503).
Договариваемся, что для символа с кодом 1 счетчик будет храниться по адресу (503+1), для символа с кодом 2 — (503+2) и так далее. Перед включением компьютера в эти места укладываем нули.

***Примечание:*** *понятно, что для этого сначала необходимо задать кодировки всех символов, то есть сделать таблицу. Понятно так же, что это не проблема, поэтому здесь подробности не рассматриваем, а считаем что такая таблица нами уже составлена.*

Тогда действие «такая буква уже есть?» - отменяется совсем.
Действие «завести счетчик» - отменяется совсем.
Остается только «взять счетчик и увеличить его не 1».

Выбираем этот вариант как более простой, под него будем сочинять новые команды.

**Нужна команда сложения.**

# Конец лекции 4.
# Продолжение следует...