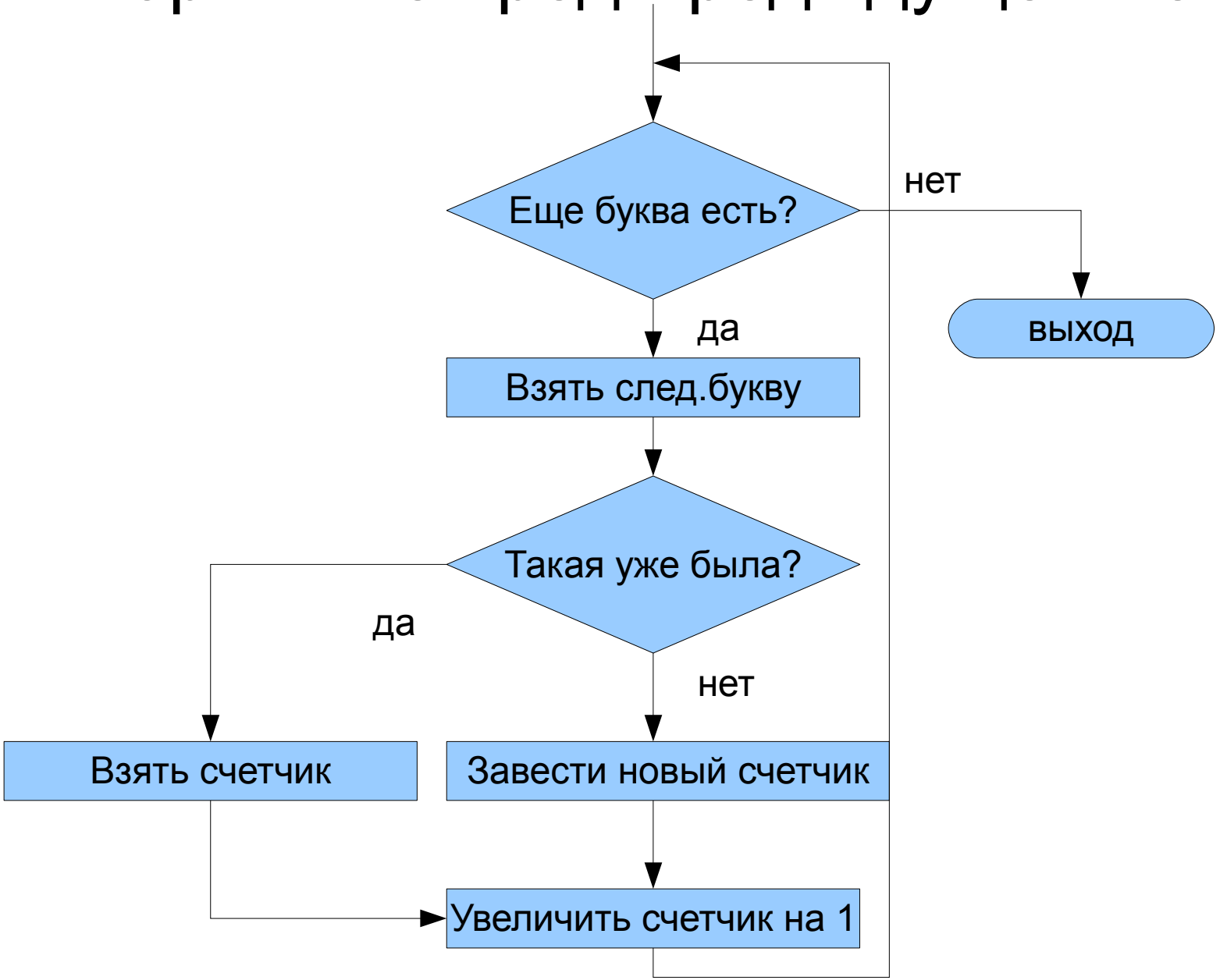
Лектор: А.Д.Хапугин

Основы программного конструирования

Лекция 5. Общие понятия архитектур ЭВМ - продолжение
(начало см. в лекциях №№3 и 4)

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

Алгоритм из пред-предыдущей лекции



Команды из предыдущей лекции

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]
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.

Попытаемся записать программу.

Начало программы из предыдущей лекции

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

Счетчики храним в памяти в такой структуре: 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.

Нужна команда сложения.

Варианты команды сложения

[3] [Adr] - увеличить на 1 слово по адресу Adr
[4] [Adr] [Val] — увеличить слово по адресу Adr на значение Val
[5] [Adr1] [Adr2] — увеличить слово по адресу Adr1 на значение, взятое по адресу Adr2
[6] [A1] [A2] [Ares] - увеличить слово по адресу A1 на значение, взятое по адресу A2 и результат сохранить по адресу Ares

Пытаемся дописать программу
0 500 (считать символ с УВВ и положить в ячейку 500)
1 500 501 502 (сравнить ячейки 500 и 501, результат сравнения занести в 502)
2 502 200 (взять результат сравнения из адреса 502, и если не тот, то перейти на 200)
4 500 600 (прибавить к слову по адресу 500 значение 600)
3 500 (увеличить слово по адресу 500 на 1)

Оп-па!
В ячейке 500 лежит не сам счетчик, а его адрес. Увеличивать адрес счетчика на 1 — совсем не то, что нужно! Вывод: изменим смысл команды с кодом «3» - теперь ее операнд — это не адрес, по которому лежит увеличиваемое значение, а адрес адреса этого увеличиваемого значения.
Это называется обращением к памяти с ***двойной косвенностью***.

Дальше?

Команда безусловного ветвления

Первоначальный вариант: ввести команду безусловного ветвления:
код команды «7».
[7] [Adr-Next]
Исполняется так: после исполнения этой команды следующая команда будет выбрана не следом за этой, а по адресу Adr-Next.
Тогда программа перепишется вот так:
0 500 (считать символ с УВВ и положить в ячейку 500)
1 500 501 502 (сравнить ячейки 500 и 501, результат сравнения занести в 502)
2 502 200 (взять результат сравнения из адреса 502, и если не тот, то перейти на 200)
4 500 600 (прибавить к слову по адресу 500 значение 600)
3 500 (увеличить слово по адресу 500 на 1)
7 0 (безусловно перейти на команду с адресом 0)

В принципе все сделано, программа работает.

При обсуждении на лекции была предложена оптимизация, при которой переход в конце программы становился условным, по условию в ячейке 502. Этот вариант в данной пезентации отсутствует, читателю предлагается придумать его самостоятельно.

Интересные моменты дискуссии

1. При работе над записью программы в виде чисел мы в какой-то момент стали путаться с обозначением как команд, так и операндов: все часла так или иначе выглядят слишком похоже.
Потому для удобства дискуссии перешли на мнемоническое обозначение кодов команд.
Программа при этом начинает выглядеть примерно вот так:

```
in      500 (считать символ с УВВ и положить в ячейку 500)
cmp     500 501 502 (сравнить ячейки 500 и 501, результат сравнения занести в 502)
jc      502 200 (если результат сравнения по адр. 502 «не тот», то перейти на 200)
add     500 600 (прибавить к слову по адресу 500 значение 600)
inc     500     (увеличить слово по адресу 500 на 1)
jmp     0  (безусловно перейти на команду с адресом 0)
```

Мнемоники строим отталкиваясь от английских слов:
in — (INput = ввод),
cmp — (CoMPare = сравнить) и т.д.,
Мнемоники подсказывают значение команд, что удобнее чем просто числа.

Интересные моменты дискуссии

2. При переделках программы, даже такой маленькой как наша, источником затруднений является то, что необходимо помнить, в какой ячейке памяти что хранится. Так же при попытках переставлять команды, заметили, что приходится вручную пересчитывать длины переходов. И там и там очень легко ошибиться!

Никаких средств для преодоления этих затруднений придумывать пока не стали.

Какого размера нужно сделать слово?

После продожительной дискуссии решили, что слово будет размером 16 бит.
Следовательно это слово сможет хранить целые числа в диапазоне [0..65535].
Этого диапазона хватает для всех действий нашей программы кроме одного: увеличения счетчика на 1.
Решили, что все-таки такой размер слова утвердим, но счетчик будем увеличивать на 1 не одной командой, а несколькими, с примерно таким смыслом:
а) проверить, а не стал ли уже счетчик максимально возможным
б) если не стал, то перейти к шагу «г»
в) увеличить старшую половину данного счетчика на 1, а сам этот счетчик обнулить
в) увеличить счетчик на 1

В виде команд полностью этот алгоритм записывать не стали, для желающих это пусть будет самостоятельным заданием

Неэффективность

Код команды по нашему соглашению хранится в одном слове, то есть занимает 16 бит.
Слово может хранить 65536 различных значений.
Если попытаться посчитать, сколько всего команд должно быть у нашей вычислительной системе, то с трудом набирается 200-300, на что хватает 8-9 бит.

Это значит, что слова с кодами команд «расходуются» не экономно.

Не проблема, но стоит отметить как место, которое нужно улучшить.

Соображение

Посмотрим повнимательнее на то, как исполняется практически любая команда, например, вот эта:
cmp 500 501 502

По шагам:
1. выбрать из памяти команду cmp
2. по выбранной команде сообразить, что у нее есть 3 операнда
3. выбрать операнд 1 (значение 500).
4. взять по адресу 500 сравниваемое значение
5. выбрать операнд 2 (значение 501).
6. взять по адресу 501 сравниваемое значение
7. произвести сравнение
8. выбрать операнд 3 (значение 502).
9. положить по адресу 502 результат сравнения

Заметим, что сравниваемые значения от момента вычитывания из памяти до момента действительного выполнения операции должны где-то храниться внутри процессора, иначе процессор просто не будет работать как надо! Причем таких ячеек для временного хранения должно быть несколько.

Нельзя ли предоставить программисту возможность явно указывать такую ячейку как операнд команды?

Регистры

Предложение: сделать внутри процессора несколько (мало) ячеек памяти, и дать им «имена», которые можно использовать в командах.
Такие ячейки называются «регистры».
Например, договариваемся, что у нас будет 8 регистров с номерами 0-7. Тогда для записи номера регистра хватает 3-х бит. Поскольку для записи кода команды хватает 8-9 бит, как раньее обнаружено, то получается, что в одном слове можно закодировать полностью двухоперандную команду, например, договорившись, что младшие (правые) 6 бит обозначают номера регистров, где хранятся операнды, а остальные биты слова — код самой команды.

Тогда в систему команд необходимо добавить специальные команды копирования из памяти в регистр и из регистра в память, и наша программа улучшится сразу по двум параметрам:
- сама программа займет меньше места в памяти
- при ее исполнении станет гораздо меньше обращений к памяти.

Попробуем...

Пример программы с использованием регистров

В тексте программы регистры будут обозначаться r0, r1 ... r7

```
in    r0 (считать слово из УВВ в регистр 0)
movri     r1,0 (занести слово 0 в регистр 1)
cmp       r0,r1,502 (трехопрадная команда, придется оставить третий операнд в памяти?)
jc        502 200 (если результат сравнения по адр. 502 «не тот», то перейти на 200)
movri     r2,600 (занести 600 в r2)
add       r0,r2 (прибавить к r0 содержимое r2)
movra     r3,r0 (занести в r3 слово, адрес которого взять из r0)
inc       r3 (увеличить содержимое регистра r3 на 1)
movma     r3,r0 (записать значение из r3 по адресу, взятому из r0)
jmp       0  (безусловно перейти на команду с адресом 0)
```

Команд стало больше, но они занимают почти в 2 раза меньше памяти, а также почти в 2 раза реже обращаются к памяти за данными.

Проверить самостоятельно.

Главные итоги

Работу по разработке архитектуры ЭВМ на этом прерываем.

Понятно, что дальше мы вполне можем довести ее до победного конца, но это большая и тщательная работа, которая выходит за рамки данного курса.
Самое главное это то, что нам понятно как ее дальше делать.

Главный итог проделанной работы — это возможность увидеть основные понятия архитектуры ЭВМ вообще, а также откуда они берутся.

Архитектура и система команд вычислительной системы — это самый нижний слой, доступный программисту, а значит это одно из оснований программирования.

Завершение следует...