Введение в архитектуру ЭВМ

Лекция 2. Системы счисления. Порядок байтов в ОП. Символьные имена

План лекции

Системы счисления

- Позиционные системы
- Алгоритмы переводов между системами с разными основаниями
  - Из системы с основанием p в десятичную систему
  - Из десятичной системы в систему с основанием p
  - Переводы между системами с основаниями 2, 8 и 16
- Порядок байтов в ОП при представлении чисел
- Символьные имена
  - Символьные имена с адресными значениями. Процесс ассемблирования
  - Символьные имена с произвольными значениями
  - Программа, иллюстрирующая процесс ассемблирования

Позиционные системы.

Вес цифры зависит от позиции $525 = 5 \cdot 10^2 + 2 \cdot 10^1 + 5 \cdot 10^0$

Общая формула позиционной системы счисления.

$$Q = a_n \cdot p^n + a_{n-1} \cdot p^{n-1} + \ldots a_1 \cdot p^1 + a_0 \cdot p^0 \qquad (1)$$

$a_n, a_{n-1}, \ldots a_1, a_0$ — цифры системы, p — ее основание.

Рассмотрим системы с основаниями 2, 8, 16.

p	цифры	
2	0	1

8	0	1	2	3	4	5	6	7							
16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
											10	11	12	13	14	15

Алгоритм перевода из системы с основанием p в десятичную систему.

1. Записать число по формуле (1).

2. Выполнить действия в десятичной системе.

Примеры.

$2 \to 10.\ 101 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 4 + 0 + 1 = 5$

$8 \to 10.\ 373 = 3 \cdot 8^2 + 7 \cdot 8^1 + 3 \cdot 8^0 = 192 + 56 + 3 = 251$

$16 \to 10.\ B9C = 11 \cdot 16^2 + 9 \cdot 16^1 + 12 \cdot 16^0 = 2816 + 144 + 12 = 2972$

Перевод из десятичной системы в систему с основанием p.

Нужно найти цифры преставления числа в системе с основанием p, зная его запись в десятичной системе. Это задача с неизвестным числом неизвестных. Заметим что если правую и левую части (1) поделить на p, выполняя деление в десятичной системе, то получим частные:

$$Q/p = a_n \cdot p^{n-1} + a_{n-1} \cdot p^{n-2} + \ldots a_1 \cdot p^0 \text{ и } a_0 \text{ в остатке} \qquad (2)$$

Т.е деление дает МЛАДШУЮ цифру представления числа Q в системе с основанием p. Из (2) видно, что если теперь значение Q/p поделить на p ещё раз, то в остатке получим a_1 — следующую цифру представления Q в системе с основание p. Легко видеть, что цифра a_n будет получена в остатке, когда частное будет равно нулю.

Алгоритм перевода из десятичной системы в систему с основанием p.

Последовательно делить нацело в десятичной системе на p **подлежащее переводу число, фиксируя получаемые остатки от деления. Прекратить деление когда частное получит значение ноль. Полученные остатки есть последовательные цифры искомого представления, причем младшей из них является первый полученный остаток.**

Примеры.

10→2. 30_{10}

30	2				
30	15	2			
0	14	7	2		
a0	1	6	3	2	
	a1	1	2	1	2
		a2	1	0	0
			a3	1	
				a4	

Результат: $30_{10} = 11110_2$

10→16. 2346_{10}

2346	16	
2336	146	16
10 (A)	144	9
a0	2	0
	a1	9
		a2

Результат: $2346_{10} = 92A_{16}$

Переводы между системами с основаниями 2, 8 **и** 16.

Эти переводы не требуют арифметических вычислений, а выполняются символьными заменами из таблицы.

Триада	8-я цифра	16-я цифра	Тетрада
`000	0	0	`0000
`001	1	1	`0001
`010	2	2	`0010
`011	3	3	`0011
`100	4	4	`0100
`101	5	5	`0101
`110	6	6	`0110
`111	7	7	`0111
		8	`1000
		9	`1001
		A	`1010
		B	`1011
		C	`1100
		D	`1101
		E	`1110
		F	`1111

Алгоритм перевода целого числа из двоичной системы в систему с основанием 8 (16).

1. Разбить двоичные цифры, начиная справа на триады (тетрады) и левую неполную из них, если есть, дополнить нулями.

2. Триады (тетрады) заменить соответствующими восьмеричными (шестнадцатеричными) цифрами из таблицы.

Примеры.

010 110 001 101 111 010 = 2615728

010 110 001 101 111 010 = 0001 0110 0011 0111 1010 = 1637A16

Алгоритм перевода целого числа из системы с основанием 8 (16) в

двоичную систему.

Заменить восьмеричные (шестнадцатеричные цифры) на соответствующие триады (тетрады) из таблицы.

Примеры.

47358 = 100111011101

9AC516 = 1001101011000101

Домашнее задание.

Выполнить по 20 переводов в направлениях 2 ⇆ 10, 16 ⇆ 10, 8 ⇆ 10, 2 ⇆ 16, 2 ⇆ 8. Через 2-3 недели будет проводиться контрольная работа на владение алгоритмами переводов. Ее успешное выполнение обязательно для получения зачета/экзамена.

Представление двоичного числа байтами с помощью 16-х цифр.

Проницательный студент уже догадался, что шестнадцатеричные (реже восьмеричные) числа используются в файлах листинга, документации, программных инструментах и т. п. для представления двоичных чисел. Это особенно удобно для восьми битовых байтов (октетов) в которых каждому полубайту (nibble, nybble) однозначно соответствуют шестнадцатеричная цифра и ее тетрада. Таким образом 32 разрядное двоичное число — с весьма длинной и непонятной записью:

`10010011111010100001101001111111`

`(тетрады – 1001 0011 1110 1010 0000 1101 0011 1111)`

будет представлено в четырех байтах (восьми полубайтах) в естественном порядке как 0x93EA0D3F (0x указывает, что число представлено в шестнадцатеричной системе).

Порядок байтов в ОП при представлении чисел

В том случае, если число не может быть представлено одним байтом, нужно зафиксировать в архитектуре системы, в каком порядке байты будут храниться в ОП или передаваться по линиям связи. Принято два таких порядка.

Little endian - байты младших цифр имеет меньший адрес. Число 93EA0D3F в этом порядке будет представлено так:

Адрес ОП	a+0	a+1	a+2	a+3
Байты числа	3F	0D	EA	93

Преимущество. Адрес двойного слова, младшего слова и младшего байта совпадают и равны, для этого примера - a+0. Для порядке Big endian это не так.

Big endian - байты младших цифр имеет больший адрес. Число 93EA0D3F в этом порядке будет представлено так:

Адрес ОП	a+0	a+1	a+2	a+3
Байты числа	93	EA	0D	3F

Порядок байтов выбирает конструктор архитектуры ЭВМ.

Порядок Little endian (остроконечный) принят а архитектурах Intel, Big endian (тупоконечный) - в сетях TCP/IP (network byte order), архитектурах Motorola, соответствует порядку десятичной системы счисления.

Примеры Little endian.

Рассмотрим фрагменты файла task1.lst.

```
55      .data    # секция данных, распределение памяти
56                  # соотв. конструкция языка С и коммент.
57 0000 29090000    n:      .long 2345      # int n = 2345; число
58 0004 00000000    length: .long 0         # int length =0; результат
59 0008 0A000000    ten:    .long 10        # определяем константу ЯВНО
```

Конструкция n: .long 2345 с помощью директивы ассемблера .long определяет символьное имя — метку (см. ниже) n: как 32 битовое ДВОИЧНОЕ число, значение которого в десятичной системе есть 2345. Ассемблер переводит это значение в двоичную систему, получая в естественном порядке байтов число 0x00000929. Т.к. эта директива первая в секции данных, ассемблер размещает это число в объектном файле в адресах ОП этой секции 0000 - 0003 в порядке Little endian.

Затем, обрабатывая директиву length: .long 0, ассемблер размещает в адресах ОП 0004 — 0007 переменную для хранения количества символов с начальным нулевым значение. Конструкция ten: .long 10 размещает в адресах 0008 - 000b двоичный аналог значения 10, опять же в порядке Little endian.

В фрагменте:

```
62      .text # секция команд процессора
63
64      .global _start     # точка входа - глобальная метка
.
```

```
   .
   88 0010 F73D0800      idivl ten  # делим объединенные регистры edx:eax
на 10
   88      0000
```

представлен сгенерированный ассемблером двоичный код команды на ЯКЦП. Здесь `F7` — код операции деления, `3D` управляющий байт, `08000000` — представленное в порядке `Little endian` *32* битовое значение адреса `0008` операнда `ten`, определенного в секции данных.

Символьные имена

Идея ассемблера — программист работает в терминах понятных ему символьных имен, которые автоматически переводятся в двоичные коды, выполняемые центральным (или другим) процессором.

Символьные имена команд задаются в описании ассемблера и соответствуют (не взаимно однозначно) двоичным кодам команд ЯКЦП.

NBNB. Уже на этом уровне проявляются преимущества языка ассемблера.

Символьным именам команд соответствует, как правило, несколько команд ЯКЦП с разными двоичными кодами, например команда `mov` (стр. 85) имеет 20 различных кодов в зависимости от типа (в ОП или в регистре), размера `{b|w|l}` операндов и комбинации приемник/источник..

NBNB. Символьные имена, определяемые программистом, используются для именования данных, адресов переходов и функций и других сущностный в программе. Они используются редактором связей для объединения раздельно транслируемых и библиотечных

модулей, а также отладчиком.

Синтаксически символьное имя это один или несколько символов из множества букв (обоих регистров), цифр и трех знаков: «_», «.» и «$». Символьное имя не может начинаться с цифры. Прописные и строчные буквы считаются разными. Ограничений на длину не существует, все знаки значимы.

Символьные имена ограничиваются знаками не из этого множества (например пробел, табуляция) или началом файла (так как исходная программа должна кончаться новой строкой, конец файла не может считаться ограничителем символа).

NBNB. Символьное имя имеет двоичное значение и тип, которые определяются в процессе ассемблирования. Значение символьного имени в большинстве случаев 32 битовое.

Символьные имена с адресными значениями. Процесс ассемблирования

Ассемблер последовательно читает строки исходного файла, переводит их в двоичные коды и размещает в объектном файле.

При этом в каждой из секций, в том числе в секции данных `.data`, секции команд `.text` и в секции глобальных неинициализированных данных `.bss` (рассмотрим позже) ассемблер ведет независимые `счетчики размещения` с нулевыми начальными значениями. При размещении в объектном файле двоичного кода очередной команды языка ассемблера счетчик увеличивается на занимаемое этим кодом количество байтов.

Определение. Метка — это символьное имя за которым без пробела

следует символ «:».

Метки используются прежде всего для именования данных и адресов передачи управления. Ассемблер присваивает метке значение счетчика размещения в объектном файле, которое он имел ДО обработки строки исходного кода, в которой метка определена. Метками можно задавать операнды команд ассемблера.

NBNB. При построении соответствующей двоичной команды ЯКЦП метка заменяется ее значением.

NBNB. Т.о. значением метки в секциях `.text, .data, .bss` является количество байтов ОП от начала секции до адреса, соответствующего метке. Тип метки соответствуют типу секции где она определена и обозначается буквами `t, d, b`.

NBNB. Итак, наличие символа «:» в конце символьного имени заставляет ассемблер определять его значение с помощью счетчика размещения. Текущее значение счетчика размещения всегда доступно программисту с помощью встроенного в ассемблер специального символьного имени «.».

Символьные имена с произвольными значениями.

Программист может присвоить символьному имени в конце которого отсутствует символ «:» любое значение с помощью конструкции:

<символьное имя> = <выражение>

Здесь выражение строится из определенных в исходном тексте символьных имен, констант и знаков арифметических и логических

операций. Мы будем использовать только простейшие выражения. Такие символьные имена можно применять например для исключения т. н. «магических числе», вычисления длин структур данных, например строк, для других целей. Ниже буду рассмотрены примеры.

Тип такого символьного имени зависит от типов членов выражения, но они также могут иметь и т. н. «абсолютный» тип. Дело в том, что при построении исполняемого файла редактор связей изменяет базовые (начальные) адреса секций `text, .data, .bss` и, следовательно, адресные значения определенных в них символьных имен, прежде всего меток, определенных в этих секциях . Такие имена называются перемещаемыми.

В тоже время ясно, что если определить символьное имя как, например:

```
Buf_length = 50
```

то его значение не требует перемещения и не будет изменяться редактором связей. Говорят, что такие символьные имена имеют *абсолютный* тип.

Программа, иллюстрирующая процесс ассемблирования

```
GAS LISTING Sym-names.S                 page 1


   1                    #    Свойства символьных имен
   2
   3                    .bss
   4                        buf_Length = 50
   5                        .lcomm c, 1
   6                        .lcomm buf, buf_Length
   7
   8                    .data
   9 0000 00000000          Sch_razm_0: .long .
```

```
10 0004 F0                      Bait:  .byte 0xF0
11 0005 05000000          Sch_razm_1: .long .
12 0009 CDAB                    Slowo: .word 0xABCD
13 000b 0B000000          Sch_razm_2: .long .
14
15 000f 00000000          .align 32
15      00000000
15      00000000
15      00000000
15      00
16 0020 DCFE4512            Dvoinoe_Slovo: .long 0x1245FEDC
17 0024 24000000           Sch_razm_3: .long .
18                    Adres_Stroki  = .
19                    Adr_2_baita_Stroki = . + 1
20 0028 31323334                Stroka: .ascii "1234 ABCD АБВГ "
20      20414243
20      4420D090
20      D091D092
20      D09320
21                       Str_Len = . - Stroka
22                  Adr_posledn_baita_Stroki = . - 1
23 003b 13000000              Dlina_Stroki_v_OP: .long Str_Len
24 003f 3F000000          Sch_razm_5: .long .
25
26
27              .text
28
29              .global _start
30
31              _start:
32 0000 8A1D0400          movb Bait,    %bl
32      0000
33 0006 668B0D09          movw Slowo, %cx
33      000000
34 000d 8B152000          movl Dvoinoe_Slovo, %edx
34      0000
35
36 0013 8B3D3B00          movl Dlina_Stroki_v_OP, %edi
36      0000
37
38 0019 66BE1300          movw $Str_Len, %si
39
40 001d B8290000          movl $Adr_2_baita_Stroki, %eax
40      00
41
42 0022 A1290000          movl Adr_2_baita_Stroki, %eax
42      00
43
```

```
GAS LISTING Sym-names.S                 page 2


  44                # системный вызов 1 -  возврат управленяи ос
  45
  46 0027 B8010000          movl $1, %eax
  46      00
  47 002c CD80              int  $0x80
  48
  49                .end

GAS LISTING Sym-names.S                 page 3


DEFINED SYMBOLS
         Sym-names.S:4      *ABS*:0000000000000032 buf_Length
                             .bss:0000000000000000 c
         Sym-names.S:6      .bss:0000000000000008 buf
         Sym-names.S:9      .data:0000000000000000 Sch_razm_0
         Sym-names.S:10     .data:0000000000000004 Bait
         Sym-names.S:11     .data:0000000000000005 Sch_razm_1
         Sym-names.S:12     .data:0000000000000009 Slowo
         Sym-names.S:13     .data:000000000000000b Sch_razm_2
         Sym-names.S:16     .data:0000000000000020 Dvoinoe_Slovo
         Sym-names.S:17     .data:0000000000000024 Sch_razm_3
         Sym-names.S:18     .data:0000000000000028 Adres_Stroki
         Sym-names.S:19     .data:0000000000000029 Adr_2_baita_Stroki
         Sym-names.S:20     .data:0000000000000028 Stroka
         Sym-names.S:21     *ABS*:0000000000000013 Str_Len
         Sym-names.S:22     .data:000000000000003a Adr_posledn_baita_Stroki
         Sym-names.S:23     .data:000000000000003b Dlina_Stroki_v_OP
         Sym-names.S:24     .data:000000000000003f Sch_razm_5
         Sym-names.S:31     .text:0000000000000000 _start

NO UNDEFINED SYMBOLS
```

Полная информация о символьных именах

Команда nm <флаги> <имя_объектного файла> **предоставляет программисту информацию о обнаруженных в файле символьных именах, их значениях и типах.**

Пример.

```
ybgv@ybgv-home:~/KAPPA/.../Sym-names> nm Sym-names.o
00000029 d Adr_2_baita_Stroki
```

```
00000028 d Adres_Stroki
0000003a d Adr_posledn_baita_Stroki
00000004 d Bait
00000008 b buf
00000032 a buf_Length
00000000 b c
0000003b d Dlina_Stroki_v_OP
00000020 d Dvoinoe_Slovo
00000000 d Sch_razm_0
00000005 d Sch_razm_1
0000000b d Sch_razm_2
00000024 d Sch_razm_3
0000003f d Sch_razm_5
00000009 d Slowo
00000000 T _start
00000013 a Str_Len
00000028 d Stroka
```