

Введение в архитектуру ЭВМ

Юрий Анатольевич Богоявленский, заведующий кафедрой
Информатики и математического обеспечения, к.т.н., доцент:
ybgv@cs.petrSU.ru, <https://cs.petrSU.ru>

Веб страница дисциплины

<https://cs.petrSU.ru/~chistyak/architecture/>

Лекция 5. Символьные имена. Классы, значения, перемещение

План лекции

Введение.....	2
Атрибуты символьных имен.....	3
Синтаксис символьных имен.....	3
Классы и определение значений символьных имен....	4
Определение значений символьных имен - меток.....	4
.....	5
Специальное символьное имя точка «.».....	5
Определение значений символьных имена с произвольными значениями.....	5
Выражения времени сборки.....	6
Секции и перемещения значений символьных имен	12
Выводы.....	19

Введение

В лекции 1 даны начальные сведения о символьных именах. В этой лекции представлены:

- атрибуты символьных имен;

- синтаксис символьных имен;
- механизм присвоения адресных значение меткам при ассемблировании;
- выражения времени ассемблирования и присвоение символьным именам произвольных значений;
- называемый **перемещением** механизм замены редактором связей `ld` адресных значений символьных имен в объектных файлах на адресные значения времени выполнения в исполняемых файлах.

Эти сведения даются только для `ISA IA-32` и файлов формата `ELF`.

Атрибуты символьных имен

В файле формата `ELF` при ассемблировании с помощью `GNU Assembler (gas)` каждое символьное имя (метка, переменная или функция) обладает набором обязательных атрибутов:

- имя;
- тип;
- привязка;
- секция;
- значение;
- размер;
- видимость.

Мы рассмотрим виды имен и механизмы определения атрибута «значение». Позже, при изучении отдельной трансляции, мы также рассмотрим атрибут «привязка».

Синтаксис символьных имен

Синтаксически символьное имя это один или несколько символов из множества прописных и строчных букв английского алфавита, цифр и трех символов: подчеркивание «_», точка «.» и доллар «\$». Символьное имя не может начинаться с цифры. Прописные и строчные буквы считаются разными. Ограничений на длину нет.

Классы и определение значений символьных имен

Символьные имен делятся на два класса: метки и символьные имена с произвольными значениями.

Определение значений символьных имен - меток

Напомним, что меткой называется символьное имя, заканчивающееся на символ двоеточия «:».

Значениями меток являются адреса в ОП элементов объектного, а после работы `ld` — исполняемого файла — прежде всего адреса данных и адреса передач управления в операндах команд.

Метки получают значение в процессе ассемблирования при последовательном чтении строк исходного файла с переводом их в двоичные образы и размещением последних в объектном файле.

Для определения значение меток в каждой из секций, `.data`, `.text` `.bss` и других `gas` ведет независимые 32b целые счетчики размещения с нулевыми начальными значениями.

NBNB. Ассемблер присваивает метке значение счетчика размещения в объектном файле, которое он имел до обработки строки исходного кода, в которой метка определена. После этого формируется и размещается в объектном файле

двоичный образ этой строки и счетчик увеличивается на занимаемое этим образом количество байтов.

Метки используются для задания операндов команд и элементов выражений времени ассемблирования (см. ниже).

NBNB. При использовании меток как операндов команд ассемблера при формировании соответствующей команды **ISA** в процессорном формате значение метки помещается в поле «Смещение».

Заметим, что при обработке объектного файла редактором связей **ld** значения меток корректируются при перемещении объектного файла на адреса ОП времени выполнения чтобы соответствовать этим адресам данных и команд. Поэтому такие символьные имена называются **перемещаемыми**. Процесс перемещения описан ниже в разделе [«Секции и перемещения значений символьных имен»](#)

Специальное символьное имя точка « . »

В процессе ассемблирования специальное символьное имя точка «.» имеет текущее значение счетчика адреса размещения текущей ассемблируемой секции. Это имя программист может использовать в выражениях.

Например конструкция в секции данных:

```
melvin: .long .
```

при ассемблировании присвоит **32b** целому, помеченному меткой **melvin:** значение адреса по которому ассемблер разместил это значение, т. е. метка будет адресом значения своего собственного адреса (ниже будут даны другие примеры).

Определение значений символьных имен с произвольными значениями

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

```
<символьное имя> = <выражение>
```

```
.set <символьное имя>, <выражение>
```

Значения таких символьных имен вычисляются при ассемблировании. Для хранения этих значений не выделяется память — вместо этого каждое вхождение такого имени замещается значением выражения.

Отметим, что значения этих символьных имен не корректируются редактором связей `ld` при перемещении объектного файла на адреса ОП времени выполнения, поэтому такие символьные имена называются **абсолютными**.

Такие символьные имена можно применять, например, для исключения т. н. «магических чисел», вычисления длин структур данных, например строк, для других целей. Например определение символьного имени:

`Buf_length = 50` позволяет исключить из исходного кода «магическое число» 50.

Ниже будут рассмотрены еще примеры.

Тип такого символьного имени зависит от типов членов выражения. Если же выражение определяет числовую константу то для этого символьного имени в таблице символов указывается тип ***ABS*** - «абсолютный» тип, (в выводе утилиты `nm - a`) — от примененного в описании `as` термина «absolute number».

Выражения времени ассемблирования.

Говоря о вычислении значений выражений обычно подразумевают, что это происходит в **процессе выполнения** программы.

Для удобства программиста в ассемблере `gas` поддерживается механизм вычисления выражений в **процессе ассемблирования** с помощью операций арифметики, логики и отношений над символьными именами и литералами. Этот механизм позволяет существенно улучшить качество исходного кода. Мы кратко рассмотрим его, полное описание можно найти в документе:

<https://kappa.cs.petrus.ru/~chistyak/architecture/docs/gas/gas-6.html>

Результатом выражения должно быть постоянное целое число или перемещение в определённой секции (см. раздел [«Тип данных указатель»](#)).

Выражение — это один или несколько аргументов, разделённых операторами. Термин **аргумент** описывает части выражения вместо обычно применяемого

термина **операнд**, который используется в нашем контексте при описании команд ЦП.

Аргументы — это символьные имена, литералы или подвыражения. Подвыражение — это левая скобка ‘(’, за которой следует целочисленное выражение, а затем правая скобка ‘)’; или префиксный оператор, за которым следует аргумент

Результатом выражения является 32-битное целое число со знаком в дополнительном коде, с указанием принадлежности его к одной из секций программы.

Мы рассмотрим только следующие операторы:

- префиксные ‘+’ и ‘-’ за которыми следует аргумент;
- инфиксные, находящиеся между двумя аргументами (приоритет слева направо) ‘*’, ‘/’, ‘%’, ‘+’, ‘-’.

Простые примеры.

```
.section .data
```

```
arr:      .long 1, 2, 3, 4
```

```
val:      .long 100
```

```
.section .text
```

```
.globl _start
```

```
_start:
```

```
    movl arr + 4*2, %eax    # берём 3-й элемент массива
```

```
    movl val + 8, %ebx    # смещение +8 от адреса val
```

Программа `Sym-names.S` — пример процесса ассемблирования

GAS LISTING `Sym-names.S`

page 1

```
1          #      Свойства символьных имен
```

```
2
```

```
3          .bss
```

```
4          buf_Length0 = 50  # абсолютное имя
```

```

5          .lcomm c, 1          #
6          .lcomm buf, buf_Length0 # 0П в bss
7
8          .data
9
10         buf_Length1 = buf_Length0 + 30 #
выражение времени ассемблирования (ВРА)
11
12         Sch_razm_0:
13 0000 000000000 .long .
14         Bait:
15 0004 F0       .byte 0xF0
16
17 0005 0000000 .align 4 # на адрес 0П кратный 4
18         Buf0:
19 0008 3200     .word buf_Length0
20         Buf1:
21 000a 5000     .word buf_Length1
22         Summ_buf:
23 000c 8200     .word  buf_Length0 + buf_Length1 #
постоянное выражение1
24
25 000e 0000     .align 4
26         Sch_razm_1:
27 0010 10000000 .long .
28         Slowo:
29 0014 CDAB     .word 0xABCD
30
31 0016 0000     .align 4
32         Sch_razm_2:

```



```

56                                     Adres_Stroki_v_0P:
57 0037 24000000                     .long Adres_Stroki
58
59
60
61
62                                     Adr_posledn_baita_Stroki_v_0P:
63 003b 36000000                     .long Adr_posledn_baita_Stroki
64
65                                     Dlina_Stroki_v_0P:
66 003f 13000000                     .long Str_Len
67
68                                     .end

```

GAS LISTING Sym-names.S

page 3

DEFINED SYMBOLS

```

Sym-names.S:4      *ABS*:00000032 buf_Length0
                   .bss:00000000 c
Sym-names.S:6      .bss:00000008 buf
Sym-names.S:10     *ABS*:00000050 buf_Length1
Sym-names.S:12     .data:00000000 Sch_razm_0
Sym-names.S:14     .data:00000004 Bait
Sym-names.S:18     .data:00000008 Buf0
Sym-names.S:20     .data:0000000a Buf1
Sym-names.S:22     .data:0000000c Summ_buf
Sym-names.S:26     .data:00000010 Sch_razm_1
Sym-names.S:28     .data:00000014 Slowo
Sym-names.S:32     .data:00000018 Sch_razm_2
Sym-names.S:34     .data:0000001c Dvoinoe_Slovo
Sym-names.S:37     .data:00000020 Sch_razm_3

```

```

Sym-names.S:40      .data:00000024 Adres_Stroki
Sym-names.S:43      .data:00000025
Adr_2_baita_Stroki
Sym-names.S:46      .data:00000024 Stroka
Sym-names.S:51      *ABS*:00000013 Str_Len
Sym-names.S:53      .data:00000036
Adr_posledn_baita_Stroki
Sym-names.S:56      .data:00000037
Adres_Stroki_v_0P
Sym-names.S:62      .data:0000003b
Adr_posledn_baita_Stroki_v_0P
Sym-names.S:65      .data:0000003f
Dlina_Stroki_v_0P

```

NO UNDEFINED SYMBOLS

Таблица символов исполняемого файла **Sym-names**

```

ybgv@ybgv-home:~/GnuAS/Labs/Sym-names.25> objdump -t Sym-
names

```

Sym-names: формат файла elf32-i386

SYMBOL TABLE:

```

00000000 l      df *ABS* 00000000 Sym-names.o
00000032 l          *ABS* 00000000 buf_Length0
08049098 l      0 .bss 00000001 c
080490a0 l      0 .bss 00000032 buf
00000050 l          *ABS* 00000000 buf_Length1
08049054 l          .data 00000000 Sch_razm_0

```

08049058	l	.data	00000000	Bait
0804905c	l	.data	00000000	Buf0
0804905e	l	.data	00000000	Buf1
08049060	l	.data	00000000	Summ_buf
08049064	l	.data	00000000	Sch_razm_1
08049068	l	.data	00000000	Slowo
0804906c	l	.data	00000000	Sch_razm_2
08049070	l	.data	00000000	Dvoinoe_Slovo
08049074	l	.data	00000000	Sch_razm_3
08049078	l	.data	00000000	Adres_Stroki
08049079	l	.data	00000000	Adr_2_baita_Stroki
08049078	l	.data	00000000	Stroka
00000013	l	*ABS*	00000000	Str_Len
0804908a	l	.data	00000000	Adr_posledn_baita_Stroki
0804908b	l	.data	00000000	Adres_Stroki_v_OP
0804908f	l	.data	00000000	Adr_posledn_baita_Stroki_v_OP
08049093	l	.data	00000000	Dlina_Stroki_v_OP
00000000		*UND*	00000000	_start
08049098	g	.bss	00000000	__bss_start
08049097	g	.data	00000000	_edata
080490d4	g	.bss	00000000	_end

Секции и перемещения значений символьных имен

В любом объектном файле секция `.text` начинается с адреса 0, за ней следует секция `.data`, а за ней — секция `.bss`. При этом адреса ОП команд и их операндов вычисляются при ассемблировании для такого распределения ОП.

Задав ключ `-h` команды `objdump` можно получить сводную информацию об этой структуре из заголовков секций объектного файла:

```
ybgv@localhost:~/GnuAS/Labs/1.No.Debug> objdump -h task1-exe-S.o
```

task1-exe-S.o: формат файла elf32-i386

Разделы:

Idx	Name	Разм	VMA	LMA	Фа смещ.	Выр.
0	.text	0000002e	00000000	00000000	00000034	2**0
		CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE				
1	.data	0000000c	00000000	00000000	00000062	2**0
		CONTENTS, ALLOC, LOAD, DATA				
2	.bss	00000000	00000000	00000000	0000006e	2**0
		ALLOC				

Структуру объектного файла на диске можно представит в виде следующей схемы:

Раздел файла	Заголовок ELF		.text		.data		.bss
Диапазон адресов	0x0	0x33	0x34	0x61	0x62	0x6D	Начало 0x6E
Размер в байтах	0x34		0x2E		0xC		0

Здесь каждая секция описана двумя строками. В первой из них даны имя секции, размер, виртуальный (VMA) и локальный адреса (LMA), перемещение от начала объектного файла на диске (Фа смещ.) и на какую степень двойки должен быть выровнен адрес начала секции в ОП (Выр.). Во второй строке указаны атрибуты секции:

- CONTENTS — секция имеет содержимое в файле на диске;
- ALLOC — секция занимает место в ОП во время выполнения программы, которое будет выделено при загрузке;
- LOAD — данные из секции должны быть перенесены из файла на диске в ОП;

RELOC — указывает на то, что в секции содержатся записи о перемещениях (релокации), которые должны быть разрешены до начала выполнения программы. Такие секции типичны для перемещаемых объектных файлов где адреса некоторых символьных ещё не известны на момент связывания.

READONLY — доступ к данным секции в ОП разрешён только для чтения;

CODE — секция содержит исполняемый код — команды процессора;

В процессе перемещения нужно корректировать значения поля «Смещение» двоичных команд так, чтобы они являлись адресами ОП соответствующих данных и команд времени выполнения, полученных при перемещении секций объектных файлов.

Ассемблер **as** формирует и размещает в объектном файле данные, необходимые редактору связей **ld** для такой коррекции. Для каждого адреса ОП размещаются следующие данные:

- Адрес в объектном файле где начинается перемещаемый адрес ОП.
- Длина этого адреса в байтах.
- Секция на которую ссылается адрес и значение начального адрес секции.
- Тип адреса.

Проиллюстрируем эти положение на примере файлов задачи **task1**, полученных при ассемблировании без отладочной информации (не задан ключ **-gstabs+**).

В исходном файле ниже удалены ненужные комментарии и цветом выделены:

- в секции **.data** определения символьных имен подлежащие перемещению и их значения, полученные ассемблером;
- в секции **.text** эти же символьные имена, указанные программистом как операнды команд и их шестнадцатеричные значения, размещенные ассемблером в полях «Смещение» двоичных команд.

```

2      * Вычисление количества позиций символов для
3      * символьного представления неотрицательного
4      * 32-битового целого числа
5      50      */
55  .data      # секция данных, распределение памяти
56              # соотв. конструкция языка C и коммент.
57  0000 29090000      n:      .long 2345      # int n = 2345;
58  0004 00000000      length: .long 0      # int length =0;
59  0008 0A000000      ten:    .long 10

62  .text # секция команд процессора
63
64  .global _start      # точка входа - глобальная метка
65
66  _start:
67  0000 90            nop      # пустая операция
72  0001 BB000000      movl $0, %ebx      # counter = 0
73  0006 A1000000      movl n, %eax      # готовим к команде деления
75  nextdigit:
76  000b BA000000      movl $0, %edx      # еще готовим, уже в цикле
88  0010 F73D0800      idivl ten      # делим объединенные регистры
88      0000
91  0016 43            incl %ebx      # ++counter
96  0017 83F800      cmpl $0, %eax      # частное > 0 ?
97  001a 7FEF          jg  nextdigit      # ДА, продолжаем
120 001c 891D0400      movl %ebx, length      # length = counter;
120      0000
122              Finish      # конец работы,
126      .end      # последняя строка исходного текста

GAS LISTING task1.S                                page 4

```

DEFINED SYMBOLS

```

task1.S:57      .data:0000000000000000 n
task1.S:58      .data:0000000000000004 length
task1.S:59      .data:0000000000000008 ten

```

```
task1.S:66      .text:0000000000000000 _start
task1.S:75      .text:0000000000000000b nextdigit
```

NO UNDEFINED SYMBOLS

Применим команду `objdump` с ключами:

`f` — заголовки разделов

`F` — перемещение от начала объектного файла на диске

`d` — дизассемблировать

`r` — данные о перемещении

Здесь в дизассемблированной секции `.text` цветом выделены шестнадцатеричные значения, размещенные ассемблером в полях «Смещение» двоичных команд и данные об их перемещении.

Заметим, что указанный тип перемещаемого адреса `R_386_32` означает, что в указанный перемещаемый адрес нужно записать абсолютный 32b адрес символьного имени, не зависящий от текущей команды.

```
ybgv@localhost:~/GnuAS/Labs/1.No.Debug> objdump -fFdr
task1-exe-S.o
```

```
task1-exe-S.o:      формат файла elf32-i386
архитектура: i386, флаги 0x00000011:
HAS_RELOC, HAS_SYMS
начальный адрес 0x00000000
```

Дизассемблирование раздела `.text`:

00000000 <_start> (файловое смещение: 0x34):

```
0:      90                      nop
1:      bb 00 00 00 00        mov     $0x0,%ebx
6:      a1 00 00 00 00        mov     0x0,%eax
```

7: R_386_32 .data

0000000b <nextdigit> (файловое смещение: 0x3f):

b: ba 00 00 00 00 mov \$0x0,%edx

10: f7 3d 08 00 00 00 idivl 0x8

12: R_386_32 .data

16: 43 inc %ebx

17: 83 f8 00 cmp \$0x0,%eax

1a: 7f ef jg b <nextdigit>

(файловое смещение: 0x3f)

1c: 89 1d 04 00 00 00 mov %ebx,0x4

1e: R_386_32 .data

22: bb 00 00 00 00 mov \$0x0,%ebx

27: b8 01 00 00 00 mov \$0x1,%eax

2c: cd 80 int \$0x80

ybgv@localhost:~/GnuAS/Labs/1.No.Debug>

Теперь рассмотрим результаты применения аналогичных команд к полученному ld исполняемому файлу. Здесь цветом выделены шестнадцатеричные значения, размещенные ассемблером в полях «Смещение» двоичных команд после выполнения перемещения редактором связей на адреса времени выполнения.

Получим информацию о секции .data.

objdump -t -j .data task1-exe-S

task1-exe-S: формат файла elf32-i386

SYMBOL TABLE:

080490a2 l .data 00000000 n

080490a6 l .data 00000000 length

```

080490aa l      .data 00000000 ten
080490ae g      .data 00000000 __bss_start
080490ae g      .data 00000000 _edata
080490b0 g      .data 00000000 _end

```

Получим информацию о секции .text.

```

ybgv@localhost:~/GnuAS/Labs/1.No.Debug> objdump -fFdr
task1-exe-S

```

```

task1-exe-S:      формат файла elf32-i386
архитектура: i386, флаги 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
начальный адрес 0x08048074

```

Дизассемблирование раздела .text:

```

08048074 <_start> (файловое смещение: 0x74):
8048074: 90                nop
8048075: bb 00 00 00 00    mov     $0x0,%ebx
804807a: a1 a2 90 04 08    mov     0x80490a2,%eax

0804807f <nextdigit> (файловое смещение: 0x7f):
804807f: ba 00 00 00 00    mov     $0x0,%edx
8048084: f7 3d aa 90 04 08 divl    0x80490aa
804808a: 43                inc     %ebx
804808b: 83 f8 00          cmp     $0x0,%eax
804808e: 7f ef            jg      804807f <nextdigit>
                                (файловое смещение: 0x7f)
8048090: 89 1d a6 90 04 08 mov     %ebx,0x80490a6

```

```
8048096:  bb 00 00 00 00  mov    $0x0,%ebx
804809b:  b8 01 00 00 00  mov    $0x1,%eax
80480a0:  cd 80          int     $0x80
ybgv@localhost:~/GnuAS/Labs/1.No.Debug>
```

Выводы

- В файлах формата ELF GNU Assembler (gas) для каждого символического имени (метка, переменная или функция) формирует семь обязательных атрибутов.
- Символические имена делятся на два класса: метки (перемещаемые символические имена) и символические имена с произвольными значениями (абсолютные). Они отличаются по способу использования и методу присвоения им значений.
- Значениями меток являются адреса в ОП элементов объектного, а после работы ld — исполняемого файла — прежде всего адреса данных и адреса передач управления в операндах команд.
- Для определения значения меток в каждой из секций, .data, .text .bss и других gas ведет независимые 32b целые счетчики размещения с нулевыми начальными значениями.
- Ассемблер присваивает метке значение счетчика размещения в объектном файле, которое он имел ДО обработки строки исходного кода, в которой метка определена. После этого формируется и размещается в объектном файле двоичный образ этой строки и счетчик увеличивается на занимаемое этим образом количество байтов.
- В процессе ассемблирования специальное символическое имя точка «.» имеет текущее значение счетчика адреса размещения текущей ассемблируемой секции.
- При обработке объектного файла редактором связей ld значения меток корректируются при перемещении объектного файла на адреса ОП времени выполнения чтобы соответствовать этим адресам данных и команд. Поэтому такие символические имена называются перемещаемыми.

- Программист может присвоить символьному имени в конце которого отсутствует символ «:» любое значение, задаваемое выражением и не связанное со счетчиком размещения, с помощью знака «=» или директивы `.set`.

- Такие символьные имена называются абсолютными, их значения вычисляются при ассемблировании. Для хранения этих значений не выделяется память — вместо этого каждое вхождение такого имени в исходный текст замещается значением выражения.

- Значения абсолютных символьных имен не корректируются редактором связей `ld` при перемещении объектного файла на адреса ОП времени выполнения,

- Для удобства программиста в ассемблере `gas` поддерживается механизм вычисления выражений в процессе ассемблирования с помощью операций арифметики, логики и отношений над символьными именами и литералами.

- В любом объектном файле секция `.text` начинается с адреса 0, за ней следует секция `.data`, а за ней — секция `.bss`. При этом адреса ОП команд и их операндов вычисляются при ассемблировании для такого распределения ОП.

- По команде `objdump -h <имя_файла>.o` можно получить список сущей файла с указанием их адресов, размеров и атрибутов.

- Редактор связей `ld` читает объектные файлы и объединяет их в исполняемый файл решая при этом две задачи:

1. Назначает объектным файлам адреса так, чтобы они не перекрывались, т. к. все они начинаются с адреса 0.

2. Перемещает объектные файлы на их адреса времени выполнения. Это процесс называется **перемещением (relocation)** и выполняется и для одного входного файла.

- Ассемблер `as` формирует и размещает в объектном файле данные, необходимые редактору связей `ld` для такой выполнения перемещения.