

Введение в архитектуру ЭВМ

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

Веб страница дисциплины

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

Лекция 5. Структура команды. Режимы адресации данных

План лекции

Структура команды.....	2
Байт Mod R/M – Mode Register/Memory.....	3
Байт SIB – Scale Index Base.....	4
Поле «Смещение».....	5
Примеры размещения элементов команды в ее двоичном представлении.....	6
Режимы адресации данных в IA-32.....	7
Назначение режимов адресации данных.....	7
Вычисление адресов ОП элементов структур данных.....	8
Пример вычисления адресов ОП элементов матрицы.....	9
Механизм режимов адресации. Эффективный адрес.....	10
Терминология.....	13

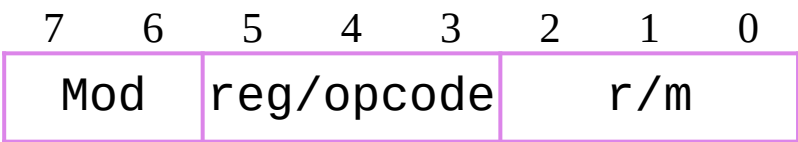
Структура команды

Поле двоичной команды	Префиксы	Код операции	Управление операндами в регистрах/памяти и режимами алресации байты ModR/M и SIB	Смещение	Непосредственный (Immrdiate) операнд
Длинп b	0 - 4 b	1 - 3 b	0 - 2b	0/1//2/4 b	0/1/2/4 b

Только один байт — кода операции присутствует всегда. Присутствие других полей зависит от структуры команды. Префиксы определяют дополнительные характеристики команды.

Байт Mod R/M – Mode Register/Memory

Если этот байт присутствует, то определяет расположение операндов (в регистре или в ОП) и способ вычисления эффективного адреса (режим адресации) операнда в ОП (если он указан). Кодирование этой информации происходит следующим образом.



Структура байта Mod R/M

Поля `mod` (`mode` — режим) и `r/m` — 32 комбинации:

- либо номер одного из 8 РОН со значением операнда;
- либо одна из 24 формул режима адресации для вычисления исполнительного адреса операнда в ОП.

Поле `reg/opcode` — в зависимости от байта кода операции:

- либо номер одного из 8 РОН со значением операнда;
- либо три дополнительных бита кода операции.

Поле `r/m` (`register/memory` — регистр/память):

- либо номер одного из 8 РОН, со значением второго операнда;

— либо, вместе с полем **mod**, одна из 24 формул режима адресации для вычисления исполнительного адреса операнда в ОП.

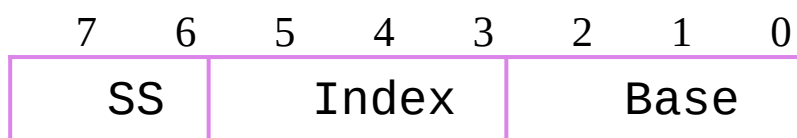
Определения

r/m-операнд — задается полями **mod** и **r/m**, может располагаться либо в регистре, либо в ОП.

Reg-операнд — задается полем **reg/opcode**, может располагаться только в регистре.

Итак, в ОП может располагаться только один из операндов.

Байт SIB – Scale Index Base



Структура байта SIB

Для определенных кодировок байта **ModR/M** требуется второй байт указания режимов адресации — **SIB**, который также включается в команду если указан режим адресации с индексным регистром (рассматривается ниже). Он также присутствует в команде при 32-разрядной адресации в форматах база плюс индекс и масштаб плюс индекс.

Более точное описание байтов **ModR/M** и **SIB** см. в документе «Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 2A: Instruction Set Reference, A-M», версия которого от ноября 2007 г. представлена на:

<https://cs.petrso.ru/~chistyak/architecture/docs/Intel-guides/inst--r-a-m.pdf>

- **SS** — код масштабного множителя (MM). MM имеет значения — 1, 2, 4 или 8. На него умножается значение индексного регистра при вычислении адреса операнда в ОП, которое выполняется перед каждым выполнением команды;
- **Index** — номер одного из семи индексных регистров (регистр **esp** использовать НЕЛЬЗЯ!);

— **Base** — номер номер одного из восьми (см. ниже) базовых регистров.

Значения полей байта SIB в 32-разрядном режиме адресации

Base	Базовый регистр	Index	Индексный регистр	SS	MM
000	EAX	000	EAX	00	1
001	ECX	001	ECX	01	2
010	EDX	010	EDX	10	4
011	EBX	011	EBX	11	8
100	ESP	100	Не исп.		
101	EBP / смещ32	101	EBP		
110	ESI	110	ESI		
111	EDI	111	EDI		

Поле «Смещение»

В архитектуре **IA-32** в плоской (flat) модели памяти для команд и данных доступно адресное пространство ОП объемом в 4 ГБ с диапазоном адресов байтов от **0x00000000** до **0xFFFFFFFF** (от 0 до $2^{32} - 1$) байт, т.е.

для задания адреса операнда в ОП необходимо **ЧЕТЫРЕ БАЙТА**, для чего в структуре команды предусмотрено поле «Смещение». Оно получает значение при формировании команды ISA.

В простейшем случае, если операнд команды ассемблера задан меткой, а режимы адресации не указаны, поле «Смещение» будет содержать адресное значение этой метки.

Если же операнд записан в форме с рассматриваемыми ниже режимами адресации, то значение поля «Смещение» будет одним из слагаемых при вычислении адреса операнда в ОП, которое выполняется перед каждым выполнением команды.

Назначение поля «Операнд в команде» говорит само за себя.

На стр. книги 147-149 описаны байты Mod R/M и SIB.

Примеры размещения элементов команды в ее двоичном представлении

1. movl \$0, %ebx — переслать значение 0 длиной 32 бита (непосредственный операнд) в регистр **ebx**;

Поле двоичной команды	Префиксы	Код операции	Задана пара операндов I, R и регистр %ebx	Смещение	Непосредственный (Immediate) операнд
Длинн b	0 - 4 b	1 - 3 b	ModR/M - 1b	нет	\$0 - 32b ноль

2. movl n, %eax — переслать **ЗНАЧЕНИЕ** из области ОП, распределенной для символического имени **n** в регистр **eax**;

Поле двоичной команды	Префиксы	Код операции	Задана пара операндов M, R и регистр %eax	Смещение	Непосредственный (Immediate) операнд
Длинн b	0 - 4 b	1 - 3 b	ModR/M - 1b	Значение адреса метки n в ОП	нет

3. movl \$n, %eax — переслать **АДРЕС** значения из области ОП, распределенной для символического имени **n** в регистр **eax**;

Поле двоичной команды	Префиксы	Код операции	Задана пара операндов M, R и регистр %eax	Смещение	Непосредственный (Immediate) операнд
Длинн b	0 - 4 b	1 - 3 b	ModR/M - 1b	нет	Значение адреса метки n в ОП

4. `movl %ebx, length` — переслать значение из регистра `ebx` длиной 32 бита в область ОП, распределенную для символьного имени `length`;

Поле двоичной команды	Префиксы	Код операции	Задана пара операндов R, M и регистр %ebx	Смещение	Непосредственный (Immrdiate) операнд
Длинн b	0 - 4 b	1 - 3 b	ModR/M - 1b	Значение адреса метки length в ОП	нет

Режимы адресации данных в IA-32

Назначение режимов адресации данных

В задаче `task3` (см. лекцию 3) для продвижения по буферу, хранящему в ОП

байты коды поступающих с клавиатуры цифр, использовалась команда

```
movb %al, buf(%esi)
```

в которой операнд-приемник задан с помощью режимов адресации.

Адрес байта	buf+0	buf+1	buf+2	..	buf+n
Значение <code>%esi</code>	0	1	2		n
№ веденного байта кода	1	2	3		n+1

При такой записи операнда говорят, что `%esi` является базовым регистром, а

адрес этого операнда **A** вычисляется процессором перед каждым выполнением этой команды по формуле:

$$A = \&buf + \%esi,$$

как показано в таблице выше.

В примере `2to10` (см. лекцию 4) для продвижения по байтам строки (по сути также буфера) `digits_of_n`, хранящей байты кодов цифр переводимого в десятичную систему двоичного числа, использовалась команда

```
movb %dl,digits_of_n(%esi)
```

в которой операнд -приемник также задан с помощью регистровой адресации.

Адрес байта <code>digits_of_n</code> +	0	1	2	3	4
Значение <code>%esi</code>	0	1	2	3	4
Цифра разряда	a4	a3	a2	a1	a0

При этом адрес этого операнда **A** вычисляется процессором перед каждым выполнением этой команды по формуле:

$$A = \& \text{digits_of_n} + \%esi,$$

как показано в таблице выше, а значение `%esi` должно меняться в цикле от 4 до 0 (если получаем все пять цифр).

Вычисление адресов ОП элементов структур данных

ОП это линейная последовательность пронумерованных байтов, что существенно затрудняет работу со структурами данных.

NBNB. Вычислять адреса произвольных элементов структур данных приходится используя формулы или алгоритмы, что может существенно замедлять выполнение программ. Мы видели, как вычисляется адрес элемента вектора в приведенных примерах работы с буферами.

NBNB. В архитектуре IA-32 реализован обобщенный механизм режимов адресации, ускоряющий вычисление адресов элементов одномерных и двумерных массивов. Рассмотрим пример такой задачи для двумерных массивов (матриц).

Пример вычисления адресов ОП элементов матрицы

Пусть задана квадратная матрица S размера 3×3 , такая как матрица с двухбайтовыми элементами S_{matrix} в рассматриваем ниже примере, расположенная в ОП по строкам. Если адрес в ОП элемента a_{11} имеет значение α , то расположение по строкам для двухбайтовых элементов двух первых строк матрицы имеет вид:

Адрес ОП	Элемент
$\alpha + 0$	a_{11}
$\alpha + 2$	a_{12}
$\alpha + 4$	a_{13}
$\alpha + 6$	a_{21}
$\alpha + 8$	a_{22}
$\alpha + A$	a_{23}

Пусть:

- Матрица расположена по строкам;
- N — порядок матрицы;
- L - длина одного элемента матрицы в байтах;
- α — адрес в ОП элемента a_{11} .

Задача. Построить функцию $A(N, L, \alpha, i, j)$ вычисляющую адрес в ОП элемента a_{ij} по заданным N, L, α, i и j .

Решение. Эта функция имеет вид:

$$A(N, L, \alpha, i, j) = \alpha + N \cdot (i - 1) \cdot L + (j - 1) \cdot L$$

Адрес ОП	Элемент	Правая часть формулы
$\alpha + 0$	a_{11}	$\alpha + 0 + 0 = \alpha$
$\alpha + 2$	a_{12}	$\alpha + 0 + 2 = \alpha + 2$
$\alpha + 4$	a_{13}	$\alpha + 0 + 4 = \alpha + 4$
$\alpha + 6$	a_{21}	$\alpha + 6 + 0 = \alpha + 6$
$\alpha + 8$	a_{22}	$\alpha + 6 + 2 = \alpha + 8$
$\alpha + A$	a_{23}	$\alpha + 6 + 4 = \alpha + A$
$\alpha + C$	a_{31}	$\alpha + C + 0 = \alpha + C$
$\alpha + E$	a_{32}	$\alpha + C + 2 = \alpha + E$
$\alpha + 0 \times 10$	a_{33}	$\alpha + C + 4 = \alpha + 0 \times 10$

NBNB. Для вычисления адреса элемента необходимо выполнить две операции сложения, две операции вычитания и три операции умножения, что будет занимать значительной время.

Т.к. обработка одномерных и двумерных массивов встречается очень часто в архитектуре реализован механизм режимов адресации данных, позволяющий вычислить адрес операнда соответствующего произвольному элементу вектора или матрицы **ЗА ВРЕМЯ ВЫПОЛНЕНИЯ ОДНОЙ КОМАНДЫ!**.

Механизм режимов адресации. Эффективный адрес

Для описания этого механизма нам потребуется три определения.

В лекции 2 было отмечено:

«**NBNB**. Т.о. значением метки в секциях **.text**, **.data**, **.bss** является количество байтов ОП от начала секции до адреса, соответствующего метке».

Если вместо метки иметь в виду любой адрес ОП то получаем обобщение в виде понятия перемещения.

1. Определение. *Перемещение* адреса ОП – это расстояние в байтах от первого байта некоторого блока ОП до какого-либо адреса ОП, английский термин — offset.

NBNB. ЕСЛИ АДРЕС ПЕРВОГО БАЙТА БЛОКА РАВЕН НУЛЮ, ТО ЛЮБОЙ БАЙТ ОП ИМЕЕТ ПЕРЕМЕЩЕНИЕ РАВНОЕ ЕГО АДРЕСУ В ЭТОМ БЛОКЕ.

2. Определение . *Смещение* – это поле в команде (см. Рис. 14, стр. 146), слагаемое для вычисления эффективного адреса (effective address) операнда — его перемещения в ОП. Английский термин — displacement.

NB В документации Intel используются именно эти два термина — offset и displacement и именно в смысле приведенных выше определений. Во многих книгах на русском языке эти два термина переводятся одним русским термином, что неверно.

Поле двоичной команды	Префиксы	Код операции	Управление операндами в регистрах/памяти и режимами алресации байты ModR/M и SIB	Смещение	Непосредственный (Immrdiate) операнд
Длинп b	0 - 4 b	1 - 3 b	0 - 2b	0/1//2/4 b	0/1/2/4 b

Структура двоичного представления команды процессора

3. Определение . *Эффективный адрес* EA (effective address) операнда команды есть его перемещение в памяти, которое вычисляется процессором НЕПОСРЕДСТВЕННО ПЕРЕД ИСПОЛНЕНИЕМ КОМАНДЫ путем сложения значений поля «Смещение» в команде со значением базового регистра и значением индексного регистра, умноженным на масштабный множитель по формуле:

$$EA = \text{Смещение} + \text{База} + \text{Инд} \cdot \text{ММ}$$

Для того, чтобы **EA** операнда вычислялся по этой формуле необходимо задать в команде операнд в ОП в виде:

$$\text{Смещение}(\text{Баз.рег}, \text{Инд.рег}, \text{ММ})$$

При этом:

- должен присутствовать хотя бы один из элементов: **Баз.рег**, **Инд.рег**;
- значения отсутствующих элементов принимаются равными нулю;
- если присутствует **Инд.рег**, а **ММ** отсутствует, то его значение считается равным 1.

Таким образом возможны, например, формы:

- без **Смещения** — **(Баз.рег, Инд.рег, ММ)**;
- без регистров — **Смещение**;
- без **Инд.рег** — **Смещение(Баз.рег)**;
- без **Баз.рег** — **Смещение(, Инд.рег, ММ)**;
- без **ММ** — **Смещение(, Инд.рег)**.

NNBB. Из формулы вычисления **EA** следует, что если не заданы регистры режимов адресации— т. е. применена форма — **Смещение**, или, если регистры заданы, но их значения равны нулю, то имеет место равенство **EA = Смещение**. Напомним также, что по определению **EA** операнда равен его **перемещению**.

Использование регистров общего назначения как базовых и/или индексных определяется формулой:

			Значения регистров		
EA	=	Смещение	+	База	+
				Инд	·
				ММ	
				eax	
				ebx	
				ecx	
		нет		ecx	1

8	edx	edx	2
16	esp	нет	4
32	ebp	ebp	8
	esi	esi	
	edi	edi	

Терминология

Режимы адресации и их терминология не стандартизованы и не совпадают в разных архитектурах ЦП. Для этих режимов использовались термины регистровая, непосредственная, прямая, косвенная (базовая и индексная), косвенная со смещением (базовая и индексная), базово-индексная со смещением и др.

С введением в архитектуре IA-32 возможности использовать для адресации данных все регистров общего назначения целесообразно различать три вида адресации:

- непосредственная (**immediate**) — операнд задан в поле «Операнд в команде»;
- прямая (**direct**) — базовый и индексный регистры не заданы, т.е. адрес ОП операнда определен в поле «Смещение» команды;
- регистровая — задано использование базового и/или индексного регистров.

Программа «Демонстрация режимов адресации»

GAS LISTING adr-mode.S

page 1

```

1
2      /*
3
4      Демонстрация режимов адресации.
5
6      */
7      .include "my-macro"      # подключение файла с
макроопределениями
1
2      .macro Finish
3          movl $0, %ebx # first argument: exit code

```

```

4          movl $1, %eax # sys_exit index
5          int $0x80 # kernel interrupt
6          .endm
7
8
9          .data # секция данных, распределение памяти
10
11 0000 00          B1: .byte    0
12 0001 0A          B2: .byte    10
13
14 0002 0000        S1: .short    0
15 0004 1400        S2: .short    20
16
17          #      Одномерный массив
18
19 0006 0F001000    SSS: .short    15,16,31,32,63,64
19          1F002000
19          3F004000
20
21 0012 1E000000    I1: .long     30
22
23          #      Двумерный массив 3 x 3
24
25          Smatrix:
26
27 0016 0B000C00          .short 11,12,13
27          0D00
28 001c 15001600          .short 21,22,23
28          1700
29 0022 1F002000          .short 31,32,33
29          2100
30
31          .text # секция команд процессора
32
33          .global _start # точка входа - глобальная метка
34
35          _start:
36
37 0000 90          nop
38
39          #      Непосредственная (Immediate) адресация
40
41 0001 B005          movb $5, %al

```

B0 код операции, 05 — операнд в команде

```

42 0003 C6050000          movb $15, B1
42          00000F

```

C6 код операции, 05 — байт Mod R/M, 00000000 — Смещение B1, 0F — операнд в команде

```

43 000a 66B80E00          movw $14, %ax

```

66 — префикс 16 битного операнда, B8 — код операции, 0E00 — операнд в команде

44 000e B8050000 movl \$5, %eax

GAS LISTING adr-mode.S page 2

44 00

45

46 # Прямая (direct) адресация, Эффективный адрес =
Смещение

47

48 0013 A0010000 movb B2, %al

48 00

49 0018 66C70502 movw \$15, S1

49 0000000F

49 00

50 0021 66A10400 movw S2, %ax

50 0000

51 0027 A1120000 movl I1, %eax

51 00

52

53 # Разные виды регистровой адресации

54

55 # РАБОТА С ВЕКТОРОМ

56

57 # Эффективный адрес = значение в базовом регистре

58 #

59 # EBX - базовый регистр

60

61 002c BB060000 movl \$SSS,%ebx# адрес массива в ebx

61 00

62 0031 668B03 movw (%ebx), %ax # 1-ый элемент в рег.

63 0034 83C302 addl \$2,%ebx # наращиваем адрес на 2

64 0037 668B03 movw (%ebx), %ax # 2-ой элемент в рег

65 003a 83C302 addl \$2,%ebx # и т.д.

66 003d 668B03 movw (%ebx), %ax

67

68 # Эффективный адрес = значение в базовом регистре +
Смещение

69 #

70 # EBX - базовый регистр

71

72 0040 29DB subl %ebx,%ebx# база = 0 - на 1-ый элемент

73 0042 668B8306 movw SSS(%ebx), %ax # 1-ый элемент в рег.

73 000000

74 0049 83C302 addl \$2,%ebx # наращиваем адрес на 2

75 004c 668B8306 movw SSS(%ebx), %ax # 2-ой элемент в рег

75 000000

76 0053 83C302 addl \$2,%ebx # и т.д.

77 0056 668B8306 movw SSS(%ebx), %ax

77 000000

78

```

79      #      Эффективный адрес = значение в инд. рег * ММ. +
      #      Смещение
80      #
81      #      EBX - индексный регистр
82
83 005d 29DB      subl %ebx,%ebx # индекс = 0 - на 1-ый элемент
84 005f 668B045D  movw SSS(,%ebx,2), %ax # 1-ый элемент в рег.
84      06000000
85      # база опущена, ММ = 2
86 0067 43      incl %ebx      # наращиваем индекс на 1
87 0068 668B045D  movw SSS(,%ebx,2), %ax # 2-ой элемент в рег
87      06000000
88 0070 43      incl %ebx      # и т.д.
89 0071 668B045D  movw SSS(,%ebx,2), %ax

```

GAS LISTING adr-mode.S

page 3

```

89      06000000
90
91      #      РАБОТА С МАТРИЦЕЙ
92
93      #      Эффективный адрес = значение в базовом регистре +
94      #      значение в индексном регистр * ММ + смещение
95      #
96      #      EDX - базовый регистр - адрес первого элемента
97      #      i-ой строки
98      #      EBX — индекс. регистр - номер столбца матрицы - j
99      #      Установка начального значения цикла по строкам:
100
101 0079 BA160000      mov $Smatrix,%edx      # i = 1
101      00
102
103      Next_String:
104      #      Установка начального значения цикла по столбцам
105
106 007e 29DB      subl %ebx,%ebx # j = 1 соотв. %ebx=0
107
108      #      Внимание! i и j управляются разными способами!
109
110      #      Для перехода к след. строке к i нужно прибавить
111      #      длину строки матрицы в байтах - в нашем случае - 6
112
113      #      Для перехода к след. столбцу к j нужно прибавить 1
114
115      #      Последовательная передача элементов матрицы в рег. %ax
116
117      Next_Row:
118
119 0080 8D0C5A      leal (%edx,%ebx,2), %ecx # демонстрация
120      # эффективного адреса
121
122      #      команда leal вычисляет Исп. адрес и посылает его в
123      #      регистр %ecx
124      #      команда присутствует только для учебных целей

```

```

125 0083 668B045A      movw    (%edx,%ebx,2), %ax # элемент ij в %eax
126
127 0087 43          incl %ebx    # j++
128 0088 83FB03      cmpl $3,%ebx  # столбцы закончились?
129 008b 75F3      jne Next_Row    # НЕТ - на обработку
130                      # элемента след. столбца
131
132                # ДА - присвоить %edx адрес первого элемента следующей
строки
133
134 008d 83C206      addl $6, %edx  # i++ (но по 6 байтов !)
135 0090 81FA2800    cmpl $Smatrix+18, %edx # строки закончились?
136                0000
137                jne Next_String    # НЕТ – на след строку
138                      # ДА - на след. команду
139                Finish # конец работы, возврат в ОС (макро из файла
my-macro)
139 0098 BB000000    > movl $0,%ebx
139                00
139 009d B8010000    > movl $1,%eax
139                00

```

GAS LISTING adr-mode.S page 4

```

139 00a2 CD80      > int $0x80
140                .end    # последняя строка исходного текста
GAS LISTING adr-mode.S page 5

```

DEFINED SYMBOLS

adr-mode.S:11	.data:000000000000000000	B1
adr-mode.S:12	.data:000000000000000001	B2
adr-mode.S:14	.data:000000000000000002	S1
adr-mode.S:15	.data:000000000000000004	S2
adr-mode.S:19	.data:000000000000000006	SSS
adr-mode.S:21	.data:000000000000000012	I1
adr-mode.S:25	.data:000000000000000016	Smatrix
adr-mode.S:35	.text:000000000000000000	_start
adr-mode.S:103	.text:00000000000000007e	Next_String
adr-mode.S:117	.text:000000000000000080	Next_Row

NO UNDEFINED SYMBOLS

Выводы

- Только один байт — кода операции присутствует всегда в двоичном коде команды.
- Байт Mod R/M определяет расположение операндов (в регистре или в ОП) и способ вычисления эффективного адреса.

- Байт SIB включается в команду если указан режим адресации с индексным регистром.
- Значение поля «Смещение» будет одним из слагаемых при вычислении адреса операнда в ОП, которое выполняется перед каждым выполнением команды.
- Механизм режимов адресации позволяет вычислять адреса элементов одномерных и двумерных массивов за время выполнения одной команды..
- Эффективный адрес EA (effective address) операнда команды вычисляется процессором НЕПОСРЕДСТВЕННО ПЕРЕД ИСПОЛНЕНИЕМ КОМАНДЫ путем сложения значений поля «Смещение» в команде со значением базового регистра и значением индексного регистра, умноженным на масштабный множитель