

# Арифметика и переходы

К лабе 1

# GCC Inline Assembly

```
#include <stdio.h>
```

```
int x;  
int main()  
{  
    x=0;  
    x += 10;  
    printf("%d", x);  
    return 0;  
}
```

```
#include <stdio.h>
```

```
int x;  
int main()  
{  
    x=0;  
    asm("addl $10, x");  
    printf("%d", x);  
    return 0;  
}
```

```
#include <stdio.h>
```

```
int x;  
int main()  
{  
    asm(  
        "mark: ""movl $0, %eax\n"  
        "addl $10, %eax\n"  
        "movl %eax, x"  
    );  
    printf("%d", x);  
    return 0;  
}
```

# Оператор цикла

**loop** *метка*

- уменьшить значение регистра **%ecx** на 1;
- если **%ecx** = 0, передать управление следующей за **loop** команде;
- если **%ecx** ≠ 0, передать управление на *метку*.

```
movl $0, %eax      /* в %eax будет результат, поэтому в начале его нужно обнулить */
movl $10, %ecx     /* 10 шагов цикла */
sum: addl %ecx, %eax /* %eax = %eax + %ecx */
     loop sum
/* %eax = 55, %ecx = 0 */
```

# Операторы перехода

```
    cmpl    $10, x    /* сравниваем x и 10 (x == 10) */
    je      equal      /* если равны, переходим на метку equal */
    movl    $0, res    /* сюда мы попадаем только если числа неравны,
                        выполняем действия для else */
    jmp     continue   /* переходим на метку (выход из оператора условия) */
equal:    movl    $1, res /* сюда мы попадаем только если числа равны,
                        выполняем действия для if
                        и переходим к следующему оператору
                        (выход из оператора условия) */
continue: /* дальнейшие действия */
```

**безусловный переход**  
(переходим всегда)


```
if (x == 10)
    res = 1;
else
    res = 0;
```

**условный переход**  
(переходим, если  
выполнено  
соответствующее  
условие)

# Операторы перехода

сmp  
j\*\*

операнд\_2, операнд\_1  
метка



The diagram illustrates the structure of comparison and jump instructions. A box at the top left shows the instruction format: 'сmp' followed by 'j\*\*'. An arrow points from this box to a table. Another arrow points from the bottom of the table to a box containing details about specific mnemonics.

| Мнемоника | Английское слово | Смысл            | Тип операндов |
|-----------|------------------|------------------|---------------|
| e         | equal            | равенство        | любые         |
| n         | not              | инверсия условия | любые         |
| g         | greater          | больше           | со знаком     |
| l         | less             | меньше           | со знаком     |
| a         | above            | больше           | без знака     |
| b         | below            | меньше           | без знака     |

- je — jne: равно — не равно;
- jg — jng: больше — не больше.

# Организация цикла оператором перехода

```
movl    $36, %ecx    /* смещение последнего элемента массива в байтах */
xor      %eax, %eax
mark:   addl    arr(%ecx), %eax    /* выполняем нужные действия */
subl     $4, %ecx    /* уменьшаем счетчик на размер ячейки,
                      чтобы перейти к предыдущему элементу */
cmpl     $0, %ecx    /* проверяем, не дошли ли мы до начала массива */
jg       mark        /* если содержимое счетчика больше нуля,
                      переходим на метку mark и продолжаем цикл */
movl     %eax, sum    /* если в счетчике ноль, выходим из цикла
                      и движемся дальше */
```

# Оператор условия внутри оператора цикла

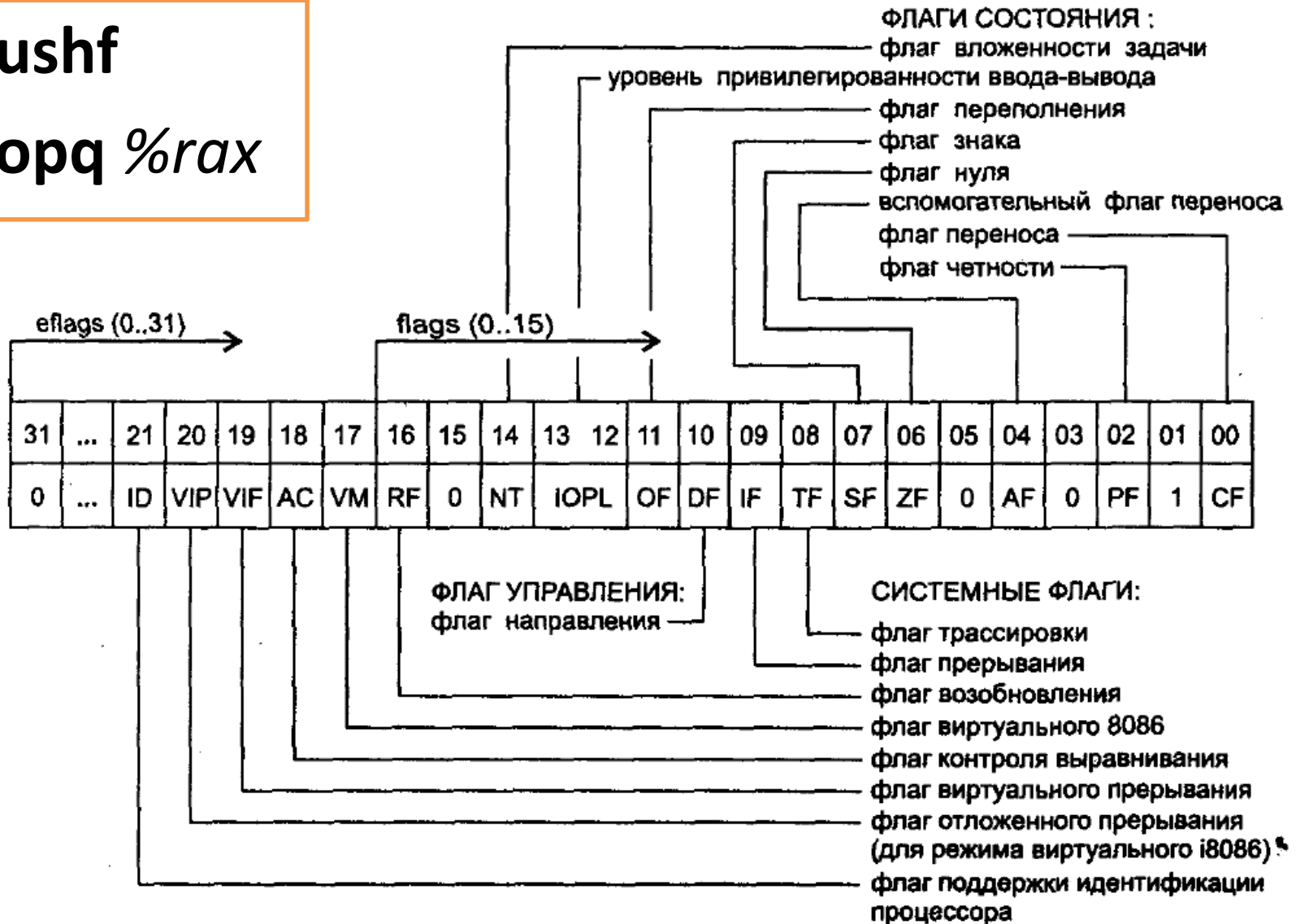
```
cycle:    movl    $36, %ecx          /* смещение последнего элемента массива в байтах */
          /* начинаем цикл, ищем в массиве числа 3 */
          cmpl   $3, arr(%ecx)     /* сравниваем текущий элемент с 3 */
          je     equal            /* если равны, переходим к метке equal */
          addl   $0, count         /* сюда мы попадаем только если числа неравны */
          jmp    continue        /* переходим на метку (выход из оператора условия) */

equal:    addl   $1, count         /* сюда мы попадаем только если числа равны */
          /* выход из оператора условия */
continue: subl   $4, %ecx         /* уменьшаем счетчик на размер ячейки,
                                чтобы перейти к предыдущему элементу */
          cmpl   $0, %ecx         /* проверяем, не дошли ли мы до начала массива */
          jnl    cycle           /* если содержимое счетчика больше нуля,
                                переходим на метку mark и продолжаем цикл */
          /* если в счетчике ноль, выходим из цикла
                                и движемся дальше */
```

# Регистр флагов

**pushf**

**popq %rax**





# Флаги состояния

|    |   |
|----|---|
| CF | <b>Флаг переноса (бит 0).</b> На самом деле он имеет разное назначение в зависимости от выполняемой инструкции. В арифметических операциях над целыми числами этот флаг, будучи установленным, показывает наличие переноса или заёма (это можно рассматривать как «беззнаковое переполнение»), а будучи сброшенным — отсутствие переноса или заёма. Кроме того, этот флаг применяется в некоторых других инструкциях и тем или иным образом характеризует полученный результат. Подробно использование этого флага в каждой конкретной инструкции указывается в её описании |
| PF | <b>Флаг чётности (бит 2).</b> Устанавливается, если младший байт результата содержит чётное число единичных битов, в противном случае сбрасывается  |
| AF | <b>Флаг вспомогательного переноса (бит 4).</b> Устанавливается при возникновении переноса или заёма из 4-ого разряда в 3-ий разряд. Сбрасывается при отсутствии такового. Используется командами десятичной коррекции.  |
| ZF | <b>Флаг нуля (бит 6).</b> Устанавливается при получении нулевого результата, сбрасывается в противном случае.   |
| SF | <b>Флаг знака (бит 7).</b> Устанавливается, если в результате операции получено отрицательное число, т.е. если старший разряд результата равен единице. В противном случае сбрасывается   |
| OF | <b>Флаг переполнения (бит 11).</b> Устанавливается, если в результате арифметической операции зафиксировано знаковое переполнение, то есть если результат, рассматриваемый как число со знаком, не помещается в операнд-приёмник. Если знакового переполнения нет, этот флаг сбрасывается   |

# Логическая арифметика

«и» - &

«или» - |

«исключающее или» - ^

«не» - !

«логическое сравнение»

Побитовые операции:

- **and** *источник, приёмник*
- **or** *источник, приёмник*
- **xor** *источник, приёмник*
- **not** *операнд*
- **test** *операнд\_1, операнд\_2*

```
testb $0b00001000, %al /* установлен ли 3-й (с нуля) бит? */  
je     not_set  
        /* нужные биты установлены */  
not_set:  
        /* биты не установлены */
```

# Команды сдвигов

```
/* SHift logical Left, Shift Arithmetic Left */  
shl, sal количество_сдвигов, назначение
```

До сдвига:

|         |                                  |
|---------|----------------------------------|
| +---+   | +-----+                          |
| ?       | 10001000100010001000100010001011 |
| +---+   | +-----+                          |
| Флаг CF | Операнд                          |

Сдвиг влево на 1 бит:

|         |  |
|---------|--|
| +---+   | +-----+                                  |
| 1   <-- | 00010001000100010001000100010110   <-- 0 |
| +---+   | +-----+                                  |
| Флаг CF | Операнд                                  |

Сдвиг влево на 3 бита:

|         |         |  |
|---------|---------|--|
| +-----+ | +---+   | +-----+                                    |
| 10      | 0   <-- | 01000100010001000100010001011000   <-- 000 |
| +-----+ | +---+   | +-----+                                    |
| Улетели | Флаг CF | Операнд                                    |

в никуда

# Команды сдвигов

```
/* SHift logical Right*/
```

**shr** количество\_сдвигов, назначение

До сдвига:

|   |  |
|---|--|
| <pre> +-----+   10001000100010001000100010001011   +-----+ Операнд </pre> | <pre> +---+   ?   +---+ Флаг CF </pre> |
|---|--|

Логический сдвиг вправо на 1 бит:

|       |                               |     |                |
|-------|-------------------------------|-----|----------------|
|       | +-----+                       |     | +---+          |
| 0 --> | 01000100010001000100010001001 | --> | 1              |
|       | +-----+                       |     | +---+          |
|       | <b>Операнд</b>                |     | <b>Флаг CF</b> |

Логический сдвиг вправо на 3 бита:

|     |     |                                  |     |         |                  |
|-----|-----|----------------------------------|-----|---------|------------------|
| 000 | --> | 00010001000100010001000100010001 | --> | 0       | 11               |
|     |     | Операнд                          |     | Флаг CF | Улетели в никуда |

# Команды сдвигов

```
/* Shift Arithmetic Right*/
```

```
sar количество_сдвигов, назначение
```

До сдвига:

|                                  |       |
|----------------------------------|-------|
| +-----+                          | +---+ |
| 10001000100010001000100010001011 | ?     |
| +-----+                          | +---+ |

Операнд

Флаг CF

старший бит равен 1 ==>

==> значение отрицательное ==>

==> "вдвинуть" бит 1 ---+

|         |
|---------|
|         |
| +-----+ |
|         |

V      Арифметический сдвиг вправо на 1 бит:

|       |                                  |         |
|-------|----------------------------------|---------|
|       | +-----+                          | +---+   |
| 1 --> | 11000100010001000100010001000101 | -->   1 |
|       | +-----+                          | +---+   |

Операнд

Флаг CF

Арифметический сдвиг вправо на 3 бита:

|         |                                  |         |         |
|---------|----------------------------------|---------|---------|
|         | +-----+                          | +---+   | +-----+ |
| 111 --> | 11110001000100010001000100010001 | -->   0 | 11      |
|         | +-----+                          | +---+   | +-----+ |

Операнд      Флаг CF      Улетели в никуда

# Команды сдвигов

```
/* ROtate Right */
```

```
ror количество_сдвигов, назначение
```

До сдвига:

```
+-----+ +---+  
| 10001000100010001000100010001011 | | ? |  
+-----+ +---+
```

Операнд    Флаг CF

Циклический сдвиг вправо на 1 бит:

```
          +-----+ 1      1 +---+  
+--- | 11000100010001000100010001000101 | ---+--> | 1 |  
|      +-----+ |      +---+  
^ Операнд                      V      Флаг CF  
|                              |  
+-----<---<---<-----+ 1
```

Циклический сдвиг вправо на 3 бита:

```
          +-----+ 011    0 +---+  
+--- | 01110001000100010001000100010001 | ---+--> | 0 |  
|      +-----+ |      +---+  
^ Операнд                      V      Флаг CF  
|                              |  
+-----<---<---<-----+ 011
```

# Команды сдвигов

```
/* ROtate Left */
```

```
rol количество_сдвигов, назначение
```

До сдвига:

```
+---+ +-----+
| ? | | 10001000100010001000100010001011 |
+---+ +-----+
```

Флаг CF Операнд

Циклический сдвиг влево на 1 бит:

```
+---+ 1      1 +-----+
| 1 | <---+--- | 00010001000100010001000100010111 | ---+
+---+ | +-----+ |
Флаг CF V      Операнд ^
      |
      +----->--->--->-----+ 1
```

Циклический сдвиг влево на 3 бита:

```
+---+ 0      100 +-----+
| 0 | <---+--- | 01000100010001000100010001011100 | ---+
+---+ | +-----+ |
Флаг CF V      Операнд ^
      |
      +----->--->--->-----+ 100
```

# Команды сдвигов

```
/* Rotate through Carry Right */
```

```
rcr количество_сдвигов, назначение
```

```
/* Rotate through Carry Left */
```

```
rcl количество_сдвигов, назначение
```

До сдвига:

```
+---+ +-----+
| X | | 10001000100010001000100010001011 |
+---+ +-----+
```

Флаг CF Операнд

Циклический сдвиг влево через CF на 1 бит:

```
      X +---+ +-----+
+<-| 1 | <--- | 0001000100010001000100010001011X | ---+
|      +---+ +-----+
V      флаг CF      Операнд
|
+----->-->-->-----+
```

Циклический сдвиг влево через CF на 3 бита:

```
      X10 +---+ +-----+
+<-| 0 | <--- | 01000100010001000100010001011X10 | ---+
|      +---+ +-----+
V      флаг CF      Операнд
|
+----->-->-->-----+
```



# Команды сдвигов

```
/* SHift logical Left, Shift Arithmetic Left */  
shl, sal количество_сдвигов, назначение
```

До сдвига:

|         |                                  |
|---------|----------------------------------|
| +---+   | +-----+                          |
| ?       | 10001000100010001000100010001011 |
| +---+   | +-----+                          |
| Флаг CF | Операнд                          |

Сдвиг влево на 1 бит:

|         |  |
|---------|--|
| +---+   | +-----+                                  |
| 1   <-- | 00010001000100010001000100010110   <-- 0 |
| +---+   | +-----+                                  |
| Флаг CF | Операнд                                  |

Сдвиг влево на 3 бита:

|         |         |  |
|---------|---------|--|
| +-----+ | +---+   | +-----+                                    |
| 10      | 0   <-- | 01000100010001000100010001011000   <-- 000 |
| +-----+ | +---+   | +-----+                                    |
| Улетели | Флаг CF | Операнд                                    |

в никуда

# Команды сдвигов

```
/* SHift logical Right*/
```

**shr** количество сдвигов, назначение

До сдвига:

|   |  |
|---|--|
| <pre> +-----+   10001000100010001000100010001011   +-----+ Операнд </pre> | <pre> +---+   ?   +---+ Флаг CF </pre> |
|---|--|

Логический сдвиг вправо на 1 бит:

|         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |   |   |   |   |
|---------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---------|---|---|---|---|
| 0 -->   | + | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | + | +       | - | - | - | + |
|         | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1       | 0 | 1 |   | 1 |
|         | + | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | + | +       | - | - | - | + |
| Операнд |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   | Флаг CF |   |   |   |   |

Логический сдвиг вправо на 3 бита:

|         |                                  |         |                     |
|---------|----------------------------------|---------|---------------------|
| 000 --> | +-----+                          | +---+   | +----+              |
|         | 00010001000100010001000100010001 | 0       | 11                  |
|         | +-----+                          | +---+   | +----+              |
| Операнд |                                  | Флаг CF | Улетели<br>в никуда |

# Команды сдвигов

```
/* Shift Arithmetic Right*/
```

```
sar количество_сдвигов, назначение
```

До сдвига:

|                                  |       |
|----------------------------------|-------|
| +-----+                          | +---+ |
| 10001000100010001000100010001011 | ?     |
| +-----+                          | +---+ |

Операнд

Флаг CF

старший бит равен 1 ==>

==> значение отрицательное ==>

==> "вдвинуть" бит 1 ---+

|

+-----+

|

V

Арифметический сдвиг вправо на 1 бит:

|       |                                  |         |
|-------|----------------------------------|---------|
|       | +-----+                          | +---+   |
| 1 --> | 11000100010001000100010001000101 | -->   1 |
|       | +-----+                          | +---+   |

Операнд

Флаг CF

Арифметический сдвиг вправо на 3 бита:

|         |                                  |         |         |
|---------|----------------------------------|---------|---------|
|         | +-----+                          | +---+   | +-----+ |
| 111 --> | 11110001000100010001000100010001 | -->   0 | 11      |
|         | +-----+                          | +---+   | +-----+ |

Операнд

Флаг CF Улетели

в никуда

# Команды сдвигов

```
/* ROtate Right */
```

```
ror количество_сдвигов, назначение
```

До сдвига:

```
+-----+ +---+  
| 10001000100010001000100010001011 | | ? |  
+-----+ +---+
```

Операнд    Флаг CF

Циклический сдвиг вправо на 1 бит:

```
          +-----+ 1      1 +---+  
+--- | 11000100010001000100010001000101 | ---+--> | 1 |  
|      +-----+ |      +---+  
^ Операнд                                V      Флаг CF  
|                                         |  
+-----<---<---<-----+ 1
```

Циклический сдвиг вправо на 3 бита:

```
          +-----+ 011    0 +---+  
+--- | 01110001000100010001000100010001 | ---+--> | 0 |  
|      +-----+ |      +---+  
^ Операнд                                V      Флаг CF  
|                                         |  
+-----<---<---<-----+ 011
```

# Команды сдвигов

```
/* ROtate Left */
```

```
rol количество_сдвигов, назначение
```

До сдвига:

```
+---+ +-----+
| ? | | 10001000100010001000100010001011 |
+---+ +-----+
```

Флаг CF Операнд

Циклический сдвиг влево на 1 бит:

```
+---+ 1      1 +-----+
| 1 | <---+--- | 00010001000100010001000100010111 | ---+
+---+ | +-----+ |
Флаг CF V      Операнд ^
|
+----->--->--->-----+ 1
```

Циклический сдвиг влево на 3 бита:

```
+---+ 0      100 +-----+
| 0 | <---+--- | 01000100010001000100010001011100 | ---+
+---+ | +-----+ |
Флаг CF V      Операнд ^
|
+----->--->--->-----+ 100
```

# Команды сдвигов

```
/* Rotate through Carry Right */
```

```
rcr количество_сдвигов, назначение
```

```
/* Rotate through Carry Left */
```

```
rcl количество_сдвигов, назначение
```

До сдвига:

```
+---+ +-----+
| X | | 10001000100010001000100010001011 |
+---+ +-----+
```

Флаг CF Операнд

Циклический сдвиг влево через CF на 1 бит:

```
      X +---+ +-----+
+-<-| 1 | <--- | 0001000100010001000100010001011X | ---+
|      +---+ +-----+
V      флаг CF      Операнд
|
+----->-->-->-----+
```

Циклический сдвиг влево через CF на 3 бита:

```
      X10 +---+ +-----+
+-<-| 0 | <--- | 01000100010001000100010001011X10 | ---+
|      +---+ +-----+
V      флаг CF      Операнд
|
+----->-->-->-----+
```

# Работа с глобальными массивами

```
#include <stdio.h>
int sum = 0, arr[10] = {0,1,2,3,4,5,6,7,8,9}, fifth;
int main()
{
    asm(
        "movl    (arr+20), %ecx    \n"
        "movl    %ecx,    fifth   \n"
        "movl    $40,      %ecx    \n"
        "xor      %eax,     %eax    \n"
        "mark:    addl      arr(%ecx), %eax \n"
        "subl     $4,       %ecx    \n"
        "cmpl     $0,       %ecx    \n"
        "jg       mark      \n"
        "movl     %eax,     sum     \n"
    );
    printf("fifth element: %d; sum: %d", fifth, sum);
}
```

# Память

## Методы адресации:

- Непосредственная
- Регистровая
- Прямая (абсолютная)
- Косвенная (базовая)
- Относительная





# Память: методы адресации

- Прямая

```
int num=10;
int main() {
    asm("movl  num, %eax\n");
    asm("movl  6295608, %eax\n");
}
```

- Косвенная

```
int num=10;
int main() {
    asm("movl  $num, %ebx\n"
        "movl  (%ebx), %eax\n");
}
```

```
int x=10, *u, res=0;
int main()
{
    u = &x;
    asm(
        "movl  x, %eax  \n"

        //      "movl  $x, %eax  \n"
        //      "movl  u, %eax   \n"
        //      "movl  6295608, %eax \n"

        //      "movl  $x, %ebx  \n"
        //      "movl  (%ebx),%eax \n"

        "movl  %eax, res \n"
    );
    printf("res=%d\n",res);
    return 0;
}
```

# Память: методы адресации

- Относительная

(адресация по базе с индексированием и масштабированием)

```
int res=0, mas[10] = {0,1,2,3,4,5,6,7,8,9};
int main()
{
    asm(
//          "movl  mas,      %eax      \n"
//          "movl  $mas,     %eax      \n"
/*          "movl  $mas,     %ecx      \n"
          "movl  20(%ecx),   %eax      \n"*/
/*          "movl  $5,       %ecx      \n"
          "movl  mas(,%ecx,4), %eax      \n"*/
          "movl  $5,       %ecx      \n"
          "movl  $mas,      %edx      \n"
          "movl  (%edx,%ecx,4), %eax    \n"

          "movl  %eax,      res       \n"
    );
    printf("res=%d\n",res);
    return 0;
}
```

# Команда *Load Effective Address*

*lea* источник, назначение

```
leal    0x32, %eax      /* аналогично movl $0x32, %eax      */
leal    some_var, %eax  /* аналогично movl $some_var, %eax  */
leal    $0x32, %eax     /* вызовет ошибку при компиляции,
                        так как $0x32 - непосредственное
                        значение                                     */
leal    $some_var, %eax /* аналогично, ошибка компиляции:
                        $some_var - это непосредственное
                        значение, адрес                             */

movl    $5, %ecx
movl    $10, %edx
leal    (%ecx,%edx,4), %eax    /* %eax = %ecx + 4 * %edx    */
```

# Память

адреса байтов в многобайтовых переменных  
расположены последовательно в обратном порядке

```
char    a='q', b='w', c='e', d='r',
        e, f, g, h;

int ch=0;
int main()
{
    asm(
        "xorl    %eax,    %eax    \n"
        "movb    a,      %al    \n"
        "shll    $8,      %eax    \n"
        "movb    b,      %al    \n"
        "shll    $8,      %eax    \n"
        "movb    c,      %al    \n"
        "shll    $8,      %eax    \n"
        "movb    d,      %al    \n"
        "movl    %eax,    ch      \n"
```

...

...

```
        "xorl    %eax,    %eax    \n"
        "movb    ch,      %al    \n"
        "movb    %al,     e      \n"
        "movb    ch+1,    %al    \n"
        "movb    %al,     f      \n"
        "movb    ch+2,    %al    \n"
        "movb    %al,     g      \n"
        "movb    ch+3,    %al    \n"
        "movb    %al,     h      \n"
    );
    printf("%c %c %c %c\n", e, f, g, h);
    return 0;
}
```

# Память

```

+---+ +---+ +---+ +---+
|   |   |   |   |   |
+---+ +---+ +---+ +---+    <- %eax
                                %al

```

```

+---+ +---+ +---+ +---+
|   |   |   |   |   | 'q' |
+---+ +---+ +---+ +---+    <- movb a,%al
+---+ +---+ +---+ +---+
|   |   |   |   | 'q' |   |
+---+ +---+ +---+ +---+    <- shll $8,%eax
+---+ +---+ +---+ +---+
|   |   |   |   | 'q' | 'w' |
+---+ +---+ +---+ +---+    <- movb a,%al

```

```

.
.
.
+---+ +---+ +---+ +---+
| 'q' | 'w' | 'e' | 'r' |
+---+ +---+ +---+ +---+    <- %eax

```

```

.
+-----+ 0x0000F046
+-----+
| 'r' |
+-----+ 0x0000F047    <- ch
+-----+
| 'e' |
+-----+ 0x0000F048    <- ch+1
+-----+
| 'w' |
+-----+ 0x0000F049    <- ch+2
+-----+
| 'q' |
+-----+ 0x0000F050    <- ch+3
.
+-----+
| данные |
+-----+ 0x0000FFF8
+-----+
| данные |
+-----+ 0x00010000    <- дно стека

```