Весна 2021

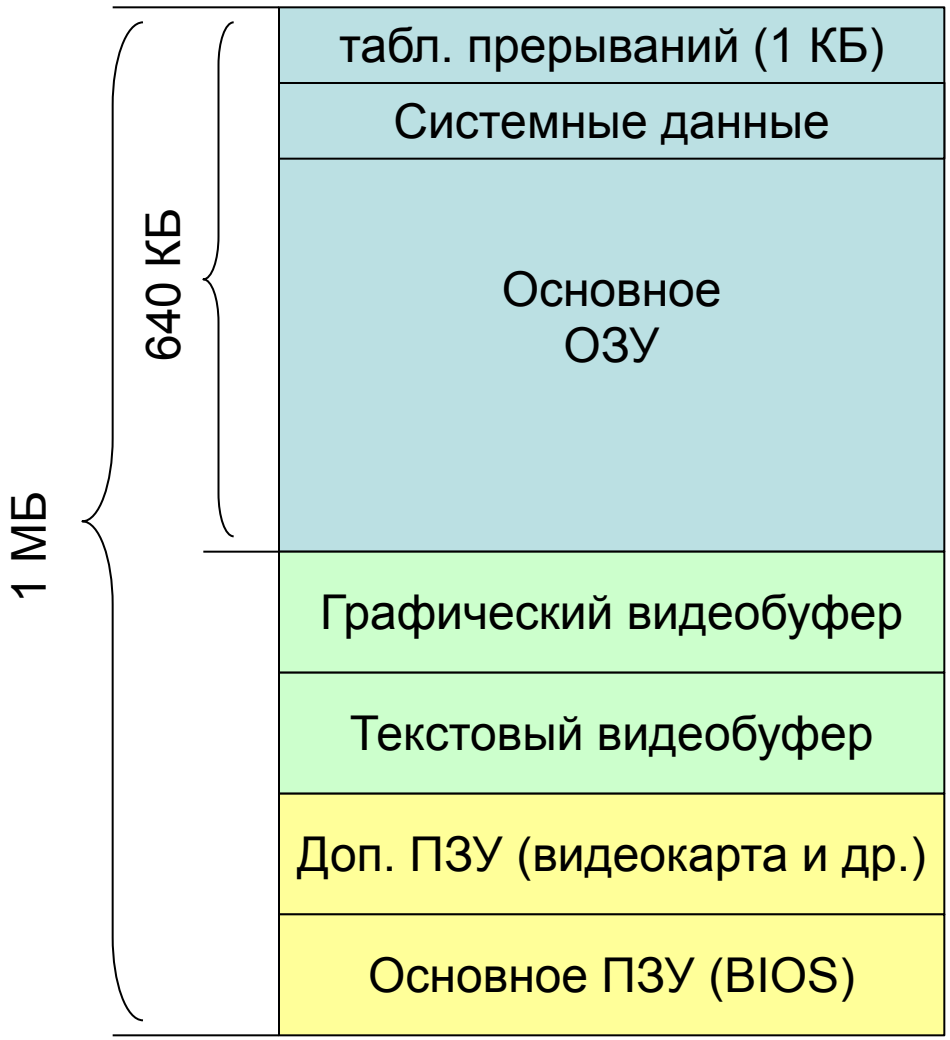
Системное программное обеспечение

Онлайн-лекции

Лекция №4: **Приемы работы с данными**

Доцент, к.т.н. ГОЛЬЦОВ Александр Геннадьевич

Карта памяти IBM-PC



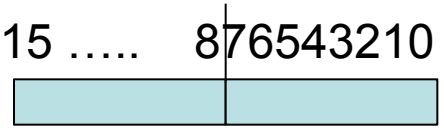
Параграф – блок памяти размером 16 байт.
1 МБ = 64к параграфов.

Сегментный регистр содержит номер первого параграфа сегмента.

Процессор стартует с адреса FFFF:0000

Биты и байты

- Система команд i8086 – байт-ориентированная
- Есть команды для работы с байтами, словами (2 байта), двойными словами (4 байта)
- Команды работы с отдельными битами появились в i80386 (не рассматриваем)
- Биты нумеруются справа-налево, номер младшего бита – 0



- Байты многобайтных значений хранятся в порядке: сначала младший, потом старший
  1234h ➔ 34 12 (байты дампа памяти)
- Существуют другие процессоры, в которых порядок байтов в многобайтовых значениях обратный

Дополнительный код

Система команд предполагает, что отрицательные числа всегда представляются в дополнительном коде.

Изменение знака числа (команда NEG <операнд>):

инвертировать код числа и прибавить 1

Один и тот же код может представлять положительное или отрицательное число, трактовка определяется программистом (алгоритмом обработки):

65535 ➔ FFFFh

-1 ➔ FFFFh

Диапазоны представимых значений

- Байт:
  0..255 (00..FF)
  или
  -128..127 (80..FF,00,01..7F)
- Слово:
  0..65535 (0..FFFF)
  или
  -32768..32767 (8000..FFFF, 0000, 0001..7FFF)

Описание содержимого памяти

[метка] Dx <данные>

- Директивы Dx (Define Byte = DB, Define Word = DW…)
- Перед директивой может быть метка данных без двоеточия (метка может отсутствовать!)
- Далее – список значений соответствующей разрядности, перечисляемых через запятую
- В директиве DB допустимо использовать строку символов
- ? среди значений означает «не важно какое», не инициализированное значение
- Директива N DUP(M) означает N-кратное повторение значения M

Директивы Dx

- **DB** – define byte.
  1-байтовые числа, строки.
  db 1,-2,?,'Hello',5,?,?
- **DW** – define word.
  2-байтовые числа, ближние адреса.
  dw 1,5,65535,-4,?,'xy',Metka
- **DD** – define double word.
  4-байтовые числа, дальние адреса.
  dd 0,?,456789,Metka
- **DQ** – define quad word.
  8-байтовые и вещественные числа (double).
  dq 0,1,2,1.0,2.0,?
- **DT** – define ten-byte.
  10-байтовые и вещественные числа (extended).
  dt 0,1,2,1.0,2.0,?

Интерпретация директив определения данных

- Директивы определения данных (DB, DW и прочие) могут встречаться в любом сегменте, в т.ч. в сегменте кода
- Ассемблер генерирует бинарное представление описанных директивой данных и размещает его в текущей позиции в текущем сегменте – в том месте, где встретилась директива
- Если описать данные в сегменте кода - они могут выполниться, будучи интерпретированы как машинные коды

Данные в сегменте кода

```
.code
s               db 'Hello, world'
start:
   ......................
                end start
```

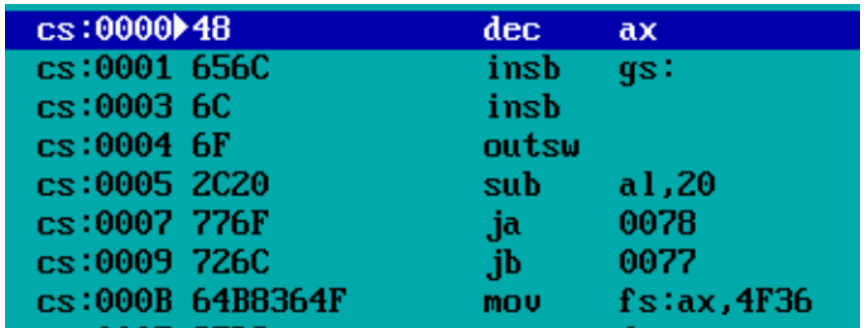
```
.code
   ......................
   jmp          somewhere
s               db 'Hello, world'
  ......................
somewhere:
   ......................
```

```
  .code
     ......................
     mov  ax,4C00h
     int 21h              ; выход из программы
  s               db 'Hello, world'
```

Управление не должно передаваться на область, в которой лежат байты данных, иначе они будут интерпретированы как коды каких-то команд и выполнены неизвестным образом

Некорректные данные в сегменте кода

```
.code
s               db 'Hello, world'
                mov ax,@data
.....................
                end     ; точка входа 0
```




```
.code
  .....................
                mov ax, 15
s               db 'Hello, world'
                mov bx, offset s
.....................
```

Символы при описании данных

- Символ (букву в кавычках) можно трактовать как цифру в системе счисления с основанием 256
- Последовательность символов (строка) произвольной длины описывается при помощи директивы DB
- Символы допустимы в директиве DW, два символа считаются двумя цифрами двухбайтового числа, первый – старший, второй – младший

db 'xyz ' ➔ ... 78 79 80 20 ...
db 'xy' ➔ … 78 79 …
dw 7879h ➔ … 79 78 …
dw 'xy' ➔ … 79 78 …

Выражения

- При описании данных (и операндов команд) допустимы выражения, вычислимые на этапе компиляции
- Выражения могут включать числа, метки, символы, знаки арифметических и логических операций и операторы ассемблера
- Выражения не могут включать значения регистров и ячеек памяти
- Знаки операций:
  + – * / mod (минус может быть унарным)
  not and or xor (побитовые)
  скобки
  shl shr (логические сдвиги)
  eq ne gt ge lt le (результат равен 0 или -1)

Некоторые операторы

- $ - текущее смещение в текущем сегменте:
  jmp $; зациклиться, перейти на свой собственный адрес
- OFFSET <метка> – смещение метки в том сегменте, в котором она объявлена
- SEG <метка> - сегментная часть адреса метки, значение не определено до загрузки программы
- LENGTH <метка> - количество элементов данных, описанных при помощи DUP после метки
- TYPE <метка> - количество байт в элементах данных, описанных после метки (1,2,4,8,10 и др.)
  со структурой дает размер структуры

Метки внутри описания данных

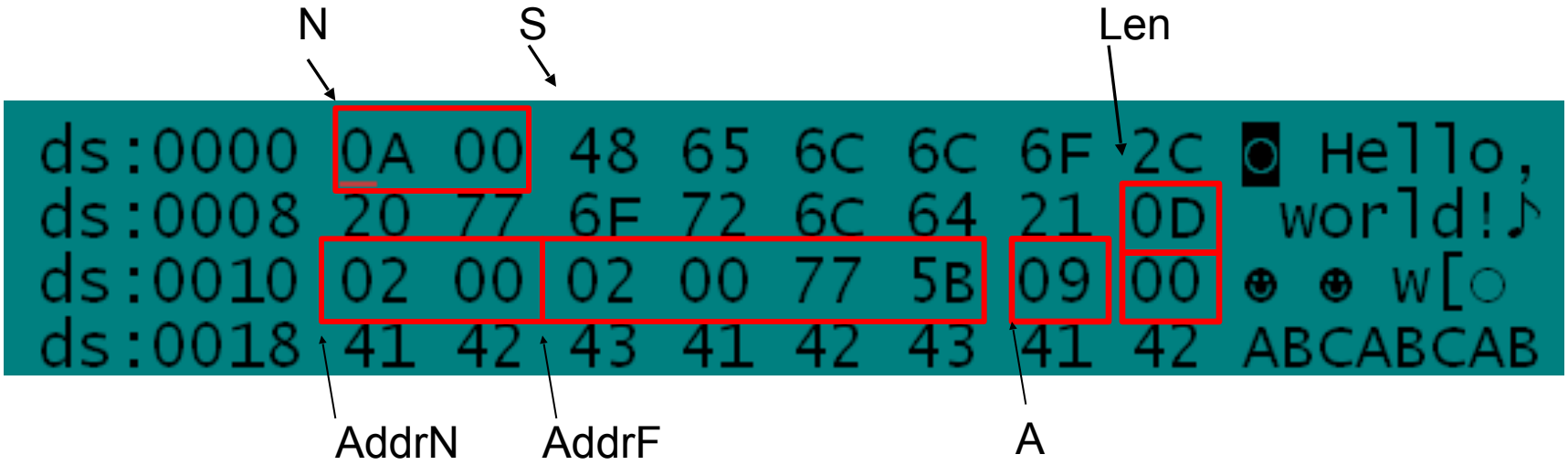
- Метка всегда имеет значение, равное ее адресу
- Если упомянуть метку в DW или DD – будет подставлен ее адрес, а не значение, расположенное по этой метке
- Если упомянуть метку в составе выражения – будет использован адрес, соответствующий метке

```
.data
    a dw 1          ; 1 по смещению 0
    b dw 10         ; 10 по смещению 2
    c dw a+b        ; 2 = 0+2 по смещению 4
    pt dd a, b, c   ; массив дальних указателей:
                    ; @data:0, @data:2, @data:4
```

При ссылке на метки, объявленные по тексту НИЖЕ, потребуется двухпроходная трансляция:
tasm /m2 my.asm

Пример описания данных

```
.data                          ; загрузчик выделил память с 5B77:0000
N       dw      10             ; 000Ah ➔  A0 00
S       db      'Hello, world!'
Len     db      $ - S          ; 13 = 0Dh
AddrN   dw      S              ; 2
AddrF   dd      S              ; @data:2
A       db      9, ?, 3 dup('ABC')
```



Оператор PTR

- byte ptr, word ptr, dword ptr
- Необходим, когда размер операнда не следует из мнемоники команды и требует явного указания

```
mov [bx],1              ; непонятно, слово или байт, слово по умолчанию
mov byte ptr [bx], 1    ; размер указан явно
mov [bx],al             ; 1 байт, потому что AL
```

- Также применяется, когда идет обращение к элементу данных, не соответствующему определению DB/DW/DD

```
N           dw 1234h
.....................
mov         al, byte ptr N          ; загрузить младший байт слова N
mov         ah,byte ptr N+1         ; загрузить старший байт
```

Директива "равно"

<символ> = <числовое выражение>

- Сопоставляет символу число - результат вычисления выражения
- Не резервируется память, действует на этапе компиляции
- Может встречаться в любом месте в программе
- Выражение вычисляется в месте использования директивы =, это актуально при использовании $ внутри выражения
- Везде, где написан символ, считается, что написано сопоставленное число

```
a=1
mov ax, a     ; mov ax, 1
```

- Значение символа в дальнейшем можно переопределять:

```
a=1
.............
a=100
```

- Можно ссылаться на свое старое значение

```
a=a+1
```

Директива EQU

<символ> EQU <определение>

- Это макроопределение: везде в исходном тексте, где употреблен символ, считается, что написана правая часть директивы
- Аналог #DEFINE языка Си
- Может встречаться в любом месте в программе
- Не резервируется память, действует на этапе компиляции
- В качестве определения может использоваться любая последовательность символов, а не только выражение, например – целая команда процессора

Примеры EQU

```
a equu 4C00h
b equ a+5
c equ [bx]
exec equ int 21h
ddef equ db 'Hello!'
mov ax,a            ; mov ax, 4C00h
ideal
mov bx, [b]         ; mov bx, [4C05h] или mov bx,4C05h в MASM
MASM
mov cx,c;           mov cx, [bx]
exec                ; int 21
dat ddef            ; dat db 'Hello!'
```

Режимы кода ideal и MASM

- http://citforum.ru/programming/tasm3/index.shtml - Turbo Assembler - руководство пользователя
- MASM - другой ассемблер ("макро-ассемблер"), с которым синтаксически совместим TASM по умолчанию
- Ideal - собственный, более "правильный" набор синтаксических правил, включается в программе явно
- В любом месте кода сколько угодно раз могут встречаться директивы Ideal и MASM для переключения режима компиляции

Директива STRUC

- Определяет структуру области памяти и, возможно, начальные значения полей

  <имя> STRUC

  <имя поля> Dx <значение> или ?

  <имя поля> Dx <значение> или ?

  .....

  <имя> ENDS
- Не выделяет память - аналог объявления типа записи в ЯВУ
- Затем имя структуры используется аналогично DB-DW-DD при выделении памяти
- MASM: Поля в структурах - глобальные символы, в разных структурах не могут быть одинаковые имена полей, имя поля не может совпадать с "отдельной" меткой данных

Работа со структурами

```
Point STRUC
    xpt dw ?
    ypt dw ?
Point ENDS
; Значения полей задаются через запятую в угловых скобках
PtArr    Point <1,1>, <1,2>, <2,2>, <2,1>
Pt       Point ?

         mov ax, PtArr.ypt
         mov Pt.ypt, ax

         mov bx, offset PtArr
         add  bx, 2* TYPE PtArr
         mov [bx].ypt,ax
```

Разновидности косвенной адресаций i8086

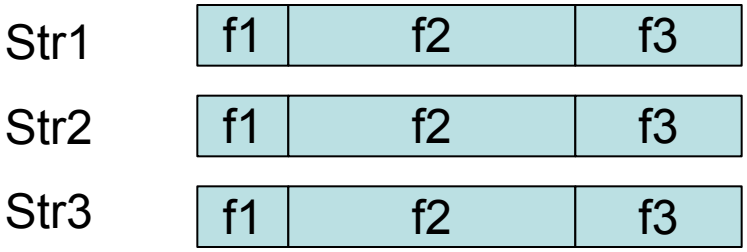
- Косвенный адрес формируется по схеме

$$\left[\begin{matrix} bx \\ bp \end{matrix} + \begin{matrix} si \\ di \end{matrix} + const\right]$$

- Любой элемент суммы может отсутствовать
- BX или BP, SI или DI
- Если есть BP - по умолчанию обращение к стеку
- const - это любое выражение, вычисляемое на этапе компиляции

Базовая адресация

- Регистр косвенно адресует начало блока памяти (обычно - структуры), относительно регистра исчисляются фиксированные смещения (обычно - к конкретным полям)
- ЛЮБОЙ из 4 возможных регистров может задавать базу, не обязательно BX или BP




```
mov bx, offset str2
mov ax, [bx+f3]
mov ax, [bx].f3
```

Индексная адресация

- Начало структуры данных (часто - массива) задано константой, относительно начала смещение задается в регистре
- ЛЮБОЙ из 4 возможных регистров может задавать смещение, не обязательно SI или DI

```
Str1   db  'Hello, world!'
Str2   db  13 DUP (?)
.....................................
        mov si,3
        mov al, [Str1+si]
        mov al, Str1[si]
        mov al, [Str1.si]
        mov al,si.str1
```

Базово-индексная адресация

- Задействуются одновременно два косвенных регистра.
- База (не обязательно BX!) адресует начало блока памяти, индекс (не обязательно SI/DI!) задает смещение внутри блока

Явное задание сегмента

- Если операнд адресуется прямо или косвенно без использования BP, то сегментная часть адреса берется из DS
- Если в составе косвенного адреса есть BP, то сегментная часть адреса берется из SS
- Можно явно указать сегментный регистр, относительно которого задается адрес операнда:

```
mov ax, es:[bx]
mov cs:[di], ax
```

- В машинном коде перед кодом "обычной" команды появится байт-префикс, меняющий сегментный регистр

Команда LEA

- "Load effective address"
- Второй аргумент - ячейка памяти, адрес которой задан в любом формате, в том числе косвенном
- Загружает в первый аргумент адрес (смещение) ячейки памяти
- lea ax, s ➔ есть такая команда, но ее код 4-байтовый и она эквивалентна трехбайтовой mov ax, offset x
- lea ax, [bx+si+75] ➔ сложение сразу трех чисел!

Команда XLAT

- Выборка байта из массива по индексу ("преобразование по таблице")
- Без аргументов (жестко использует AL и BX)

mem(ds:[BX+AL]) ➔ AL

- Можно подменить сегмент по умолчанию: xlat es:

Формирование шестнадцатеричной записи числа

```
.286
.data
HEX     db '01234567890ABCDEF'
OutStr  db 4 dup(?),'$'
N       dw 1234h
.code
.....................
        mov     bx, offset HEX

        mov     al,byte ptr N+1
        shr     al,4
        xlat
        mov     OutStr[0],al

        mov     al,byte ptr N+1
        and     al,0Fh
        xlat
        mov     OutStr[1],al

        mov     al,byte ptr N
        shr     al,4
        xlat
        mov     OutStr[2],al

        mov     al,byte ptr N
        and     al,0Fh
        xlat
        mov     OutStr[3],al

        mov     ah,9
        mov     dx,offset OutStr
int 21h

        .....................
```

Если не написать .286, то

shr al,4 ➔

```
shr al,1
shr al,1
shr al,1
shr al,1
```

Сложение многобайтных чисел

```
A       dd 1
B       dd 2
SUM     dd ?
.........................
mov     ax, word ptr A
add     ax, word ptr B
mov     word ptr SUM, ax

mov     ax, word ptr A+2
adc     ax, word ptr B+2
mov     word ptr SUM+2, ax
```

```
Mas     dw 1,2,3,4,5
SUM     dd ?
.........................
        xor     bx,bx
        mov     word ptr Sum,bx
        mov     word ptr Sum+2, bx

        mov     cx,5

m1:     mov     ax,mas[bx]
        add     bx,2
        add     word ptr SUM,ax
        adc     word ptr SUM+2, 0

        loop m1
```

Спасибо за внимание.