

Весна 2022

Системное программное обеспечение

Онлайн-лекции

Лекция №2: Программы в формате EXE. Некоторые команды.

Доцент, к.т.н. ГОЛЬЦОВ Александр Геннадьевич

Регистры i8086

РОН

| | | |
|----|----|----|
| AH | AL | AX |
| BH | BL | BX |
| CH | CL | CX |
| DH | DL | DX |

Индексные

| |
|----|
| SI |
| DI |

Указатели

| |
|----|
| IP |
| SP |
| BP |

Сегментные

| |
|----|
| CS |
| DS |
| ES |
| SS |

Флаги

| |
|-------|
| Flags |
|-------|

- Можно изменять для нужд алгоритма пользователя
- Обычно используется для конкретной цели
- Управляет процессом вычислений
- Настраивается под нужды алгоритма

Регистр флагов

- Часть битов выставляются при совершении арифметических и логических операций и характеризуют результат:
 - Z - нулевой результат
 - C - перенос из старшего разряда
 - S - отрицательный знак числа
 - O - арифметическое переполнение
 - A - межтетрадный перенос (двоично-десятичные преобразования)
 - P - четное количество единиц в младшем байте результата
- Другие биты управляют работой процессора:
 - I - разрешение прерываний
 - T - режим трассировки
 - D - перебор операндов в порядке уменьшения/увеличения адресов в строковых командах
- Регистр в целом можно сохранить в стеке и восстановить PUSHF/POPF

Понятие сегмента

- Сегмент - блок памяти произвольного размера, в котором производится адресация с использованием смещения относительно начала сегмента.
- ОС выделяет память программе **сегментами**
- У сегмента (в "большой" операционной системе) есть **селектор** - что-то вроде идентификатора (2-4 байта), зная который, можно выяснить, с какого адреса сегмент начинается, каковы права доступа к нему и т.п.
- Адрес в сегментной модели памяти двухкомпонентный: **<селектор : смещение>**

Размер сегмента

- Максимальный размер определяется разрядностью смещения:
16 бит → 64к
32 бита → 4Г
- Обычно при выделении кратен степени двойки:
8, 16, 4096
- Минимальный размер - определяется кратностью степени двойки (условно - 1, 8, 16 байт)

Другие значения слова "сегмент"⁶

Применительно к i8086 сегментом называют:

- Блок памяти произвольного размера от 0 до 64 килобайт.

А также в разном контексте:

- Сегмент максимально возможного размера - "полный сегмент" в 64 килобайта.
- Сегментный регистр (CS,DS,ES,SS), хранящий селектор сегмента.
- Сегментная часть (селектор) в составе полного адреса

"Дальность" адреса

- Дальний адрес (far) - полный адрес ячейки памяти, включая селектор сегмента и смещение, 4 байта.
- Ближний адрес (near) - только смещение, селектор сегмента задан неявно, подразумевается где-то, 2 байта.
- Короткий адрес (short) - в пределах ± 128 адресов, 1 байт.

(Под "адресом" понимается значение, хранимое в ячейке памяти, регистре/регистрах или непосредственно заданное в коде команды.)

Устройство файла .EXE

- Программа состоит из одного или нескольких сегментов.
- Обычно это сегмент **кода** (один или несколько), сегмент **данных** (один или несколько), сегмент **стека**.
- На языке ассемблера описывается содержимое всех сегментов (для стека достаточно указать размер).
- Файл на диске имеет заголовок, в котором закодирована служебная информация для загрузчика, и собственно машинные коды, соответствующие сегментам программы.
- Отладочная информация: связь "смещение в сегменте → номер строки исходного текста", символьные имена для адресов (названия меток)

Модель памяти

Задаёт ассемблеру, сколько сегментов **предполагается** в программе

- tiny - один сегмент для всего-всего
- small - по одному сегменту на код, данные, стек
- compact - много данных - один кода
- medium - много кода - один данных
- large - много кода и данных

Директива ассемблера **.model**:

.model small

```

; общий комментарий - автор, особенности запуска и т.п.
.model small          ; один сегмент кода, данных и стека
.stack 100h           ; отвести под стек 256 байт
.data                 ; начало сегмента данных
S db 'Hello, world!$'
.code                 ; начало сегмента кода
; Начальная инициализация
mov ax,@data
mov ds,ax             ; настройка DS на начало сегмента данных

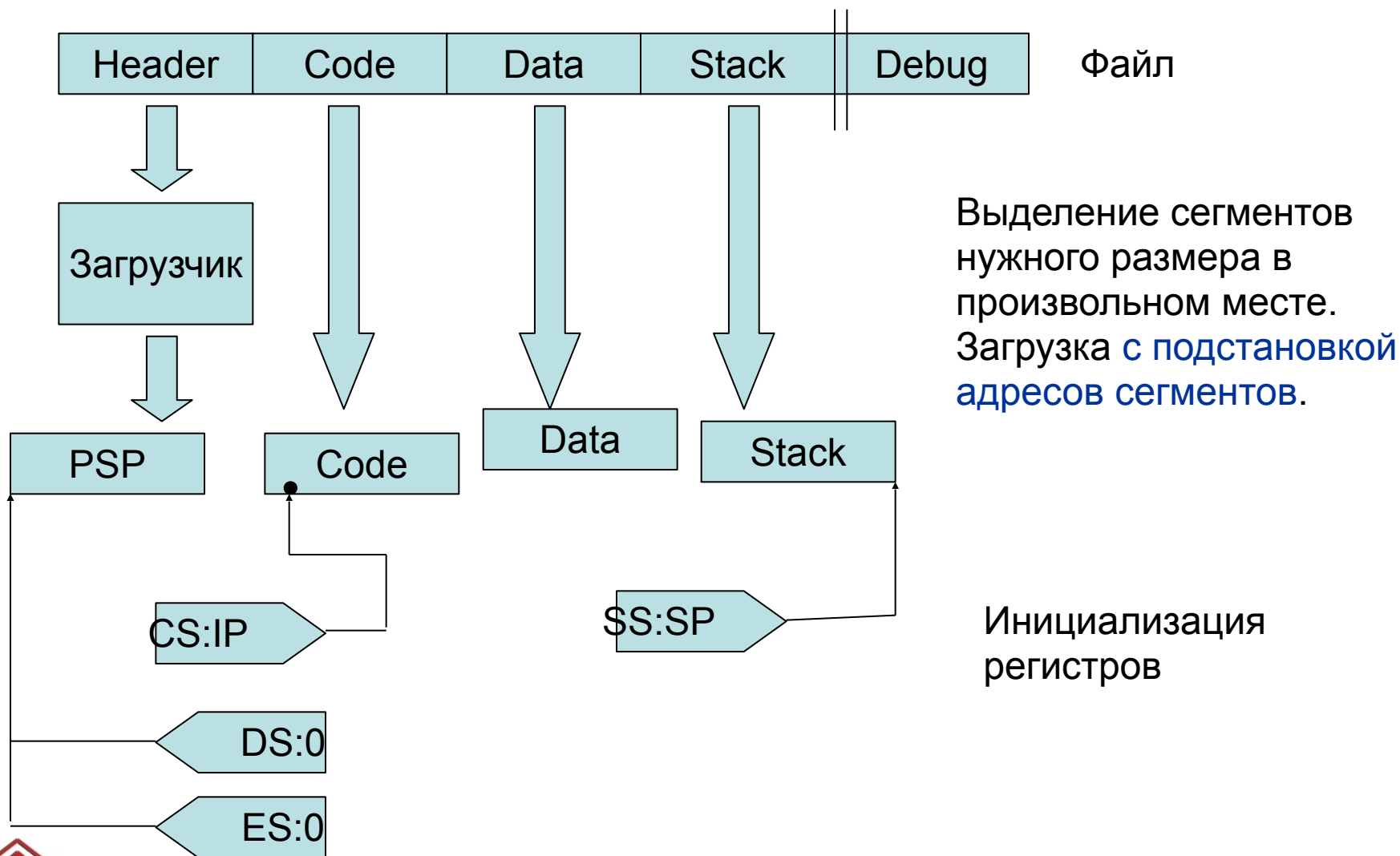
;-----
; Вывод строки на экран
mov ah,9              ; номер функции DOS
mov dx,offset S        ; DS:DX <- адрес строки S
                      ; DS уже проинициализирован ранее
int 21h                ; Вывод строки на экран в текущей позиции
курсора
;-----

; Стандартное завершение программы
mov ax,4C00h          ; ah = N функции, al = код возврата
int 21h                ; снять программу с выполнения

end                    ; конец текста программы

```

Загрузка программы .EXE



Загрузка программы EXE

- Загрузчик считывает и анализирует заголовок
- Выделяются сегменты нужного размера под PSP, код, данные и стек
- Загрузчик при запуске программы в разных условиях выделит сегменты по разным адресам, эти адреса невозможно предугадать
- В сегменты считываются коды из соответствующих областей файла, заполняются поля в PSP
- Считанные из файла коды преобразуются с учетом RLT (relocation table, таблица преобразования адресов)
- Инициализируются регистры

PSP (префикс программного сегмента)

- PSP (program segment prefix) - структура данных, поддерживаемая DOS для каждой программы. Создается при запуске. Содержит системную информацию о состоянии процесса.
- Размер 256 байт.
- По смещению 81h от начала можно прочесть параметры командной строки, завершаемые 0Dh:

my.exe -t -v -delete

RLT (relocation table)

- Некоторые адреса в программе не определены до ее загрузки; как правило - это селекторы сегментов (сегментная часть адресов)
- В таблице RLT закодировано, в какие ячейки памяти после загрузки нужно прописать эти сегментные адреса

Написано в программе:

```
mov ax, @data
```

Хранится в файле:

```
mov ax, 0 ← и ссылка на эти два байта 00 00 в RLT
```

Будет после загрузки в сегменте кода (например):

```
mov ax, 5E48
```

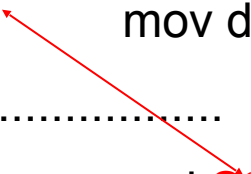
Инициализация регистров

- **SS:SP** указывают на ячейку, расположенную **сразу после сегмента стека**. При помещении чего-то в стек SP будет **уменьшаться**. SP всегда содержит смещение **последней занятой** ячейки стека (вершины стека).
- DS и ES привязываются к **PSP** (DS:0 и ES:0). **Чтобы DS указывал на сегмент данных - его в программе нужно изменить явно**.
- CS:IP устанавливаются на **точку входа** в программу. Это не обязательно первая команда в сегменте. Установка CS:IP означает начало выполнения программы.

Точка входа

- По умолчанию программа начинает исполняться с самой первой команды в сегменте кода.
- Чтобы задать точку входа, нужно объявить в соответствующем месте метку и **упомянуть эту метку после слова END**, завершающего программу:

```
.code  
Action:  ; какой-то код  
.....  
Action2: ; еще какой-то код  
.....  
Start:  mov ax, @data  
         mov ds,ax  
.....  
         end Start
```



Система команд (основные команды)

Команда MOV: пересылка данных

- MOV <куда>, <откуда>
- Операнды могут быть в регистрах, заданы непосредственно, прямо, косвенно.
- Операций память-память нет.
- Можно пересылать константу в память.
- Нет команд помещения константы в сегментный регистр.
- В Tasm если операндом указана метка данных, то это трактуется как прямой адрес (как будто метка заключена в квадратные скобки)

Использование меток данных

.data

s dw 10,20,30,40,50

x dw s+2 ; $x \leftarrow \text{offset } s + 2 = 0 + 2, x = 2$

.code

.....

mov ax, s ; $ax \leftarrow 10$

mov ax,[s] ; $ax \leftarrow 10$ (то же самое)

mov ax, offset s ; $ax \leftarrow 0$ (s в начале сегмента)

mov ax, seg s ; $ax \leftarrow @data$

mov ax, s+1 ; $ax \leftarrow 1 ((\text{offset } s) + 1)$

Арифметические команды

- ADD <куда>, <что> - сложение
- SUB <откуда>, <что> - вычитание
- NEG <что> - смена знака числа
- DEC <что> - уменьшение на единицу
- INC <что> - увеличение на единицу
- Результат → в первом параметре
- Команды изменяют флаги
- Инкремент и декремент не меняют флаг переноса!
- Операнды - в регистрах, непосредственные, в памяти, прямо и косвенно заданные

Тестовое задание

- Вычислить $A+B-C$ и поместить в регистр AX результат, где A, B и C - 16-разрядные числа:
 - $A=10$,
 - B лежит в AX,
 - C лежит по адресу 1234:5678h

add ax, 10 ; AX \leftarrow B+A

mov bx, 1234h

mov es, bx ; ES привязан к заданному сегменту

sub ax, es:[5678h] ; AX \leftarrow AX - C

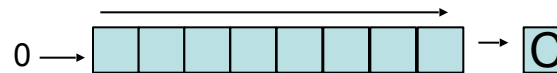
Логические команды

- AND <что>, <с чем> - побитовое И
- OR <что>, <с чем> - побитовое ИЛИ
- XOR <что>, <с чем> - побитовое \oplus
- NOT <что> - побитовая инверсия (НЕ)
- Результат \rightarrow в первом параметре
- Команды изменяют флаги
- Операнды - в регистрах, непосредственные, в памяти, прямо и косвенно заданные

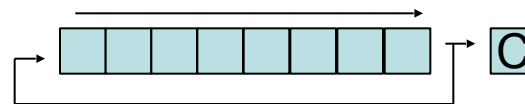
Сдвиги

- <СДВИГ> <что>,1 - сдвиг на 1 разряд
- <СДВИГ> <что>,cl - сдвиг на несколько разрядов, заданы в CL
- <СДВИГ> <что>,n - 286+, сдвиг на несколько разрядов

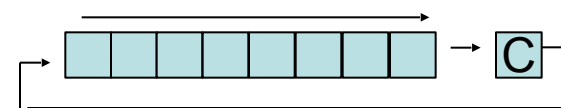
- SHL/SHR - линейный



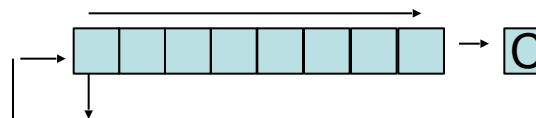
- ROL/ROR - циклический



- RCL/RCR - циклический с переносом



- SAR - арифметический (влево - идентичен линейному)
не изменяет знак числа, так можно делить на 2 отрицательные



Сравнение

- CMP <что>, <с чем> - неразрушающее вычитание, флаги формируются, результат отбрасывается

```
cmp ax, 15  
je m1
```

- TEST <число>, <маска> - неразрушающее И, флаги формируются, результат отбрасывается, часто применяется для проверки отдельных битов числа

```
test ax, 00100000b  
jnz m1 ; переход, если взведен 5-й бит
```


Переходы

- JMP <метка> - безусловный
 - JZ (JE) <метка> - если флаг нуля взведен (равенство при сравнении)
 - JNZ (JNE) <метка> - если флаг нуля сброшен (неравенство при сравнении)
 - JC - если есть перенос
 - JNC - если нет переноса
 - JG - если первый операнд сравнения/вычитания больше
 - JA - то же, но сравниваемые числа трактуются как числа со знаком
 - JLE - меньше или равно
 - JBE - то же, но операнды со знаком
 - и т.д (множество условий, все задаются комбинациями флагов)
-
- Команды условных переходов **КОРОТКИЕ**, т.е не далее ± 128 от текущего адреса.
 - Безусловный переход - ближний или дальний, ассемблер сам распознает, видя в каком сегменте метка.

Условный переход "вдаль"

- Сформировать обратное условие и обойти безусловный переход

Хочется:

```
cmp ax,10  
je Action
```

..... ; много кода

Action: ; метка далеко

Пишем:

```
cmp ax,10  
jne m1  
jmp Action
```

m1:

..... ; много кода

Action: ; метка далеко

Организация цикла

- LOOP <метка> - команда для организации циклов с известным количеством повторений.
- Уменьшает CX, и если CX не равен нулю - переходит на метку.

```
xor ax,ax    ; обнулить ax
mov cx,10    ; счетчик повторов
LoopStart:
add ax,cx    ; сумм(1..10) → AX
loop LoopStart
```

Операции со стеком

- PUSH <что> - регистр, сег, память, константу (286)
- POP <что>
- PUSHA/POPA - все регистры (286)
- PUSHF/POPF - регистр флагов

PUSH AX

- уменьшает SP на 2
- AX → mem[SS:SP]

Вызов и возврат из процедуры

- CALL <метка>
занести IP (или CS:IP) в стек
IP (или CS:IP) ← адрес метки
- RET (RETN, RETF)
извлечь из стека IP (или CS:IP)
- RET <n>
извлечь из стека IP (или CS:IP)
дополнительно увеличить SP на n

Вызов прерывания и возврат

- INT n
сохранить в стеке CS, IP, флаги
взять в таблице прерываний адрес обработчика n
сбросить флаг IF (запретить прерывания)
поместить адрес обработчика в CS:IP
- IRET
извлечь из стека флаги, IP, CS

Спасибо за внимание.