

Весна 2021

Системное программное обеспечение

Онлайн-лекции

Лекция №6: **Объявление сегментов. Файлы COM. Обработка прерываний**

Доцент, к.т.н. ГОЛЬЦОВ Александр Геннадьевич

Лаб 4

- Ничего не вводим с клавиатуры! Описываем данные средствами языка ассемблера.
- Результатом может быть преобразованная исходная строка или другая строка, получаемая в ходе работы с исходной.
- Вывести исходную и результирующую строки на экран (возможно - посимвольно).
- **Отделять обработку от вывода:** сначала сформировать полностью строку-результат, а потом - вывести ее на экран целиком

Умножение

- Команды MUL и IMUL
- MUL <8-битный операнд>
 $AL * \text{операнд} \rightarrow AX$
- MUL <16-битный операнд>
 $AX * \text{операнд} \rightarrow DX:AX$
- MUL - трактует оба операнда как числа без знака
IMUL - как числа со знаком в доп. коде
- Операнд может быть регистром или ссылкой на ячейку памяти.

Умножение на степень двойки

- Используется сдвиг влево:

$$X * 2^n = X \text{ shl } n$$

- Правило действует в том числе для отрицательных чисел – просто сдвигаем дополнительный код
- Просто умножить на два - сложить число с самим собой

Деление

- Команды DIV и IDIV
- DIV <8-битный операнд>
AX / <операнд> → частное в AL, остаток в AH
- DIV <16-битный операнд>
DX:AX / <операнд> → частное в AX, остаток в DX
- DIV - трактует оба операнда как числа без знака
IDIV - как числа со знаком в доп. коде
- Операнд может быть регистром или ссылкой на ячейку памяти.
- При делении на 0 или при получении частного, не уместящегося в половинную разрядность, будет ошибка (аварийное завершение программы)
- Делитель (байт или слово) должен быть по модулю больше, чем старшая половина делимого (слова или двойного слова)

Деление на степень двойки

- Для положительных чисел используется сдвиг вправо:

$$X / 2^n = X \text{ shr } n$$

- Для отрицательных чисел допустимо применять арифметический сдвиг (для положительных $\text{sar} = \text{shr}$):

$$X / 2^n = X \text{ sar } n$$

но в этом случае округление пойдет не к 0, а в меньшую сторону, что верно с математической т.з., но часто не годится в алгоритме

$$-1 \text{ sar } 1 = -1$$

$$-2 \text{ sar } 1 = -1$$

$$-3 \text{ sar } 1 = -2$$

$$-4 \text{ sar } 1 = -2$$

Смысл замены умножения и деления сдвигами

- 8086:
деление - до 180 тактов
умножение - до 150 тактов
сдвиг регистра - 2 такта
сдвиг ячейки памяти - 16+ тактов
- Современные процессоры персональных компьютеров способны выдавать результаты умножения, деления и сдвигов в примерно одинаковом темпе - каждый такт.
- В микроконтроллерах может вообще не быть операций умножения и деления.

Директива ORG

- В процессе трансляции указывает ассемблеру установить текущее смещение в сегменте в заданное значение.
- Все последующие данные, команды, метки считаются расположенными, начиная с указанного смещения.

.data

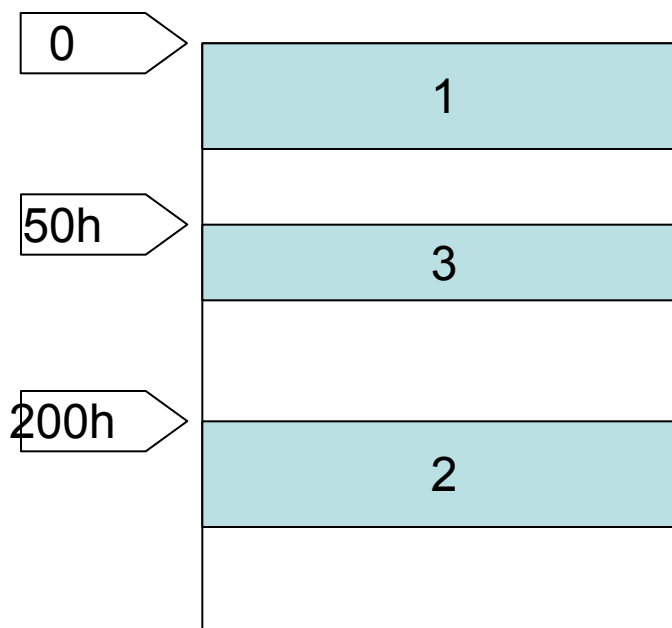
<порция данных 1>

org 200h

<порция данных 2>

org 50h

<порция данных 3>



Директива LABEL

<имя> LABEL <тип>

- Сопоставляет метку текущей позиции в текущем сегменте, ничего в сегменте не размещая и не резервируя ячеек памяти.
- Позволяет создавать псевдонимы для обращения к одному и тому же месту в памяти:

Pt	label	dword	; к следующим 4 байтам можно обращаться
			; как к двойному слову (там указатель)
Ofst	dw ?		; отдельно смещение
Sgm	dw ?		; отдельно сегмент
			; Ofst = Pt, Sgm = Pt+2

Директивы сегментации

- SEGMENT - ENDS
- Упрощенные директивы
- ASSUME

Упрощенные директивы сегментации

- Упрощенные директивы для программ с небольшим количеством сегментов:

`.code` [<имя>] - может быть несколько с разными именами

`.data`

`.stack` [<размер>]

`.fardata` [<имя>] - может быть несколько с разными именами
(и др.)

- Имя - без кавычек, как метка данных.
- Несколько однотипных - если предусмотрено моделью
- Символические имена для обозначения номера первого параграфа:

`@code`, `@stack`, `@data`, `@fardata`

Полные директивы сегментации

<Имя> **SEGMENT** [<выравнивание>] [<комбинирование>] ['<класс>']

.....

<Имя> **ENDS**

Имя - как метка.

Выравнивание: byte, word, dword, para, page

Комбинирование: private, public, common, stack, at <абс.адрес>

Классы (стандартные): 'code', 'data', 'stack'

По умолчанию атрибуты: para private <пустая строка>

Несколько сегментов с одинаковым именем "склеиваются" в один, содержимое последующего начинается после содержимого предыдущего.

Атрибуты сегмента

- **Выравнивание** - кратность адреса начала сегмента определенному количеству байт:
`byte` - 1 (без выравнивания), `word` - 2, `dword` - 4,
`para` - 16, `page` - 256
- **Комбинирование** - как комбинировать одноименные сегменты разных модулей (файлов `.asm/.obj` многомодульной программы):
 - `private` - не объединяются,
 - `public` - объединяются,
 - `common` - перекрываются, начинаясь с одного адреса (размер равен размеру самого большого)
 - `stack` - как `public`, дополнительно загрузчик настраивает на такой сегмент `ss:sp` (иначе - инициализировать вручную как `ds`)

Расположение сегментов в памяти¹⁴ после загрузки

- При использовании полных директив сегментации - в порядке определения сегментов в программе.
- При использовании упрощенных директив - в порядке, принятом в DOS (code - data - stack).
- Можно менять порядок загрузки с упрощенной сегментацией (не рассматриваем).

Директива ASSUME

`assume <регистр>:<сегмент>, <регистр>:<сегмент>`
где <сегмент> - имя сегмента или NOTHING

- Связывает сегментный регистр (DS, CS, SS, ES) с конкретным сегментом в программе.
- При упоминании меток данных, расположенных в этом сегменте, ассемблер при необходимости вставляет префикс, переопределяющий сегментный регистр по умолчанию.
- Программист должен заботиться, чтобы в соответствующем сегментном регистре был адрес нужного сегмента. Ассемблер всего лишь подставляет автоматически (или не подставляет) сегментные префиксы.

Абсолютный сегмент

VMem SEGMENT at 0B800h; b800:0000

TxtBuf label word ; знакоместо = слово (аттр:символ)

VMem ENDS

.....
assume es:VMem

mov ax,VMem

mov es, ax

; настроить ES на VMem надо руками

.....
mov TxtBuf+40*2, 1E00h + 'A' ; будет подставлено ES:

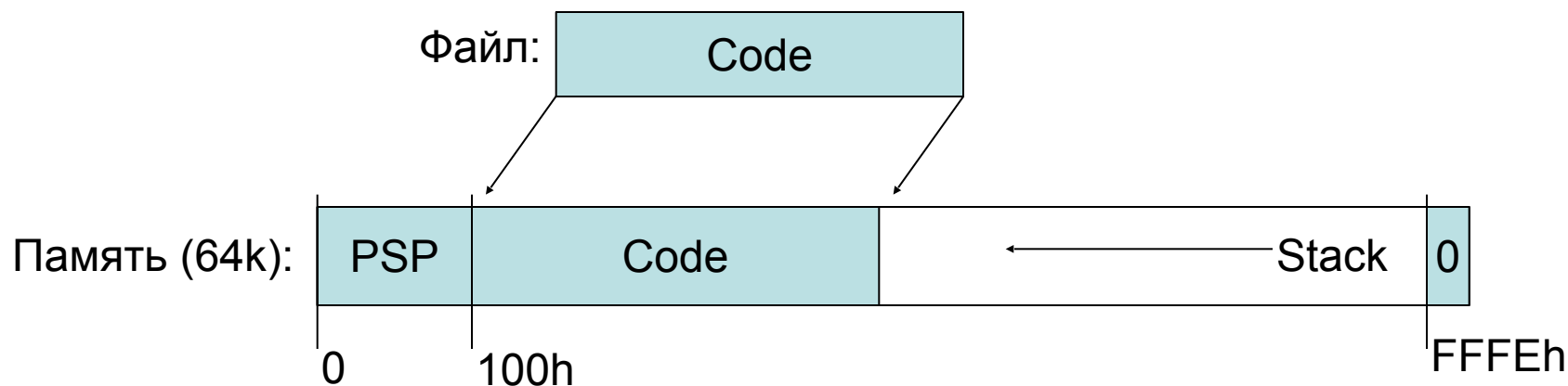
ES будет подставлено, потому что ASSUME и метка внутри VMem. Настройка ES на VMem на это не влияет!

Выводит букву А цветом "желтый по синему" в позиции (0, 40) - середина верхней строки

Формат COM для исполнимых файлов

- Придуман в ОС CP/M (1974)
- Унаследован в DOS (1981) для совместимости
- Прimitивный, неэффективный
- Файл содержит ТОЛЬКО исполнимый код
- Нет заголовка, нет RLT
- Грузится фиксированным образом
- Один сегмент для всего-всего
- Данные объявляются в сегменте кода
- Подходит для небольших программ

Загрузка СОМ-файла



CS=DS=ES=SS = номер параграфа начала сегмента

SP=0, push 0 → SP=FFFh

IP = 100h

Замечания по загрузке

- У файла нет заголовка, нет структуры, невозможно задать точку входа, при загрузке код никаким преобразованиям не подвергается
- Всегда выделяется 64к - полный сегмент
- Все сегментные регистры настраиваются на начало сегмента
- Первые 100h=256 байт отводятся под PSP
- Код грузится со смещения 100h (после PSP) и стартует с самой первой команды (IP=100h)
- Под стек отводится весь конец сегмента после кода
- В стек при загрузке помещается 0

Исходный код

.model tiny

.code

org 100h; учесть способ загрузки

Beg:

jmp Start ; обойти данные

s db 'Hello, world'

Start:

mov ah,9

mov dx, offset s

int 21h

mov ax, 4C00h

int 21h

; или данные в конце после команды выхода

end Beg ; точку входа указывать обязательно

Особенности исходного кода

- Модель TINY означает один единственный сегмент .CODE
- ORG 100h - не причина, а следствие того, что код будет загружен по смещению 100h.
- Без ORG 100h смещение строки S, например, не соответствовало бы действительности.
- Точка входа должна быть явно указана и должна быть по смещению 100h
- Данные нужно размещать так, чтобы они не начали исполняться.

Сборка СОМ-файла

```
tasm my.asm
```

```
tlink /t my.obj
```

- Недопустимы директивы SEG, недопустимы метки в качестве элементов массивов DD, т.к. нет заголовка, нет RLT и загрузчик не может подставить фактический адрес сегмента.
- компоновщик не может присоединить к файлу COM отладочную информацию.

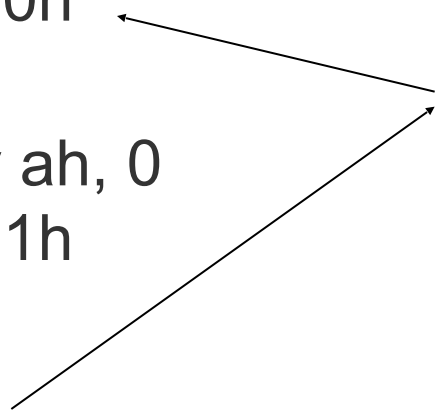
Способы выхода из COM-программы

- `mov ah, 4Ch`
`mov al, <code>`
`int 21h`

- `int 20h`

- `mov ah, 0`
`int 21h`

- `ret`

- 
- при загрузке в стек был помещен 0
 - `ret` помещает 0 в `ip`
 - переходим на адрес `cs:0`
 - это - начало PSP
 - там код `CD 20` = `int 20h`

Сравнение COM и EXE

- При загрузке COM расходуется 64к памяти - для EXE выделяется столько, сколько нужно, не больше.
- В COM ограничен объем доступной памяти, большой алгоритм не уместится - в EXE можно делать много сегментов, каждый до 64к.
- Размер файла COM меньше за счет отсутствия заголовка - но место на диске выделяется кластерами, маленькие файлы все равно занимают целый кластер (кластер = 4к, 8к, 32к...).
- Формат COM накладывает дополнительные ограничения: нельзя произвольно указать точку входа, использовать директивы SEG.

COM: добраться до PSP

```
.model tiny
```

```
.code
```

```
org 81h
```

```
ParamStr label byte ; данные не объявлены, только их адрес
```

```
org 100h
```

```
beg:
```

```
.....Использовать данные по метке ParamStr.....
```

```
end beg
```

Компиляция в COM и EXE

Чтобы один и тот же файл можно было скомпилировать и в COM, и в EXE:

- Модель tiny
- Один сегмент
- Точка входа 100h
- Явно инициализировать DS, потому что при загрузке EXE для PSP выделяется ОТДЕЛЬНЫЙ сегмент:

```
push cs  
pop ds
```

- Выход через функцию, годящуюся и для EXE, и для COM

Обработка прерываний

- Процессор поддерживает 256 различных источников прерываний
- Прерывания делятся на:
 - внутренние
 - аппаратные (деление на 0, отладка...)
 - программные (int n)
 - внешние (они аппаратные)
 - немаскируемые (вход NMI → int 2)
 - маскируемые (вход INTR, номер прерывания передает внешний контроллер прерываний)
- Флаг IF влияет только на вход INTR и не влияет на другие прерывания (команды CLI / STI)

Вызов обработчика

- Таблица прерываний расположена по адресу 0000:0000 (в начале памяти) и содержит 256 двойных слов - дальних указателей на обработчики прерываний (**вектора прерываний**).
- Когда случается прерывание номер N, из N-й ячейки таблицы извлекается вектор (младшее слово - смещение, старшее - сегментная часть адреса).
- Текущие значения CS, IP и флагов сохраняются в стеке (6 байт).
- Сбрасывается флаг IF (запрещаются внешние аппаратные прерывания).
- Вектор загружается в CS:IP, начинает исполняться обработчик.

Прерывания и стек

- Аппаратные прерывания происходят всегда, они используют тот стек, на который сейчас указывают SS:SP
- Если вы увеличили SP, а потом тут же уменьшили (освободили ячейку стека и снова ее заняли) - в это время могло произойти прерывание, и кратковременно освобожденная ячейка стека уже не содержит того, что в ней было.
- При любых манипуляциях с настройкой SS:SP нужно запрещать прерывания.
- Аналогично - при изменениях в таблице прерываний.

Возврат из обработчика

- Команда IRET
- Извлекает из стека флаги, IP и CS, возвращаемся в точку, в которой случилось прерывание.
- Восстановление флагов приводит флаг IF в то состояние, которое было до обработки прерывания (очевидно, для внешнего аппаратного прерывания - прерывания были разрешены)

Внутри обработчика

- **ВСЕ регистры**, которые модифицируются в обработчике, должны быть сохранены и восстановлены при выходе, т.к. **НЕИЗВЕСТНО, КАКОЙ КОД БЫЛ ПРЕРВАН**, не обязательно написанный вами - возможно, работала сервисная или библиотечная функция.
- Есть команды **PUSHA** и **POPA**
- Обработчик должен быстро завершаться
- В обработчике можно явно разрешить прерывания, но есть опасность его же рекурсивного вызова. Реентерабельный ли он?

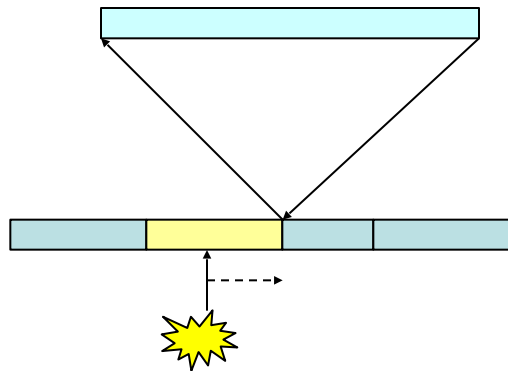
Реентерабельность

- Реентерабельность - свойство кода сохранять работоспособность, будучи прерванным и запущенным сначала, а затем - продолженным с точки прерывания.
- Для обеспечения реентерабельности все переменные, используемые алгоритмом, должны храниться в стеке или в сохраняемых в памяти при входе регистрах (не должны лежать в фиксированных ячейках памяти)

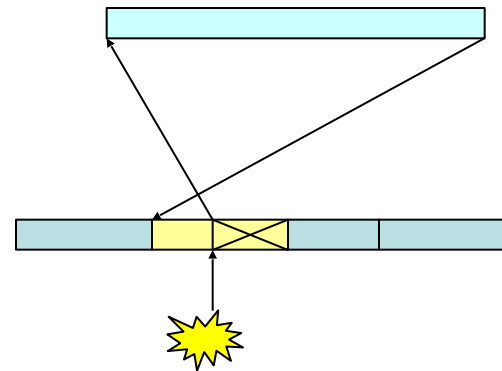
Прерывания и исключения

- Исключение (exception) - разновидность внутренних прерываний с особой логикой обработки (повтором команды).

Обычное прерывание: команда, во время которой случилось прерывание, выполняется до конца, затем вызывается обработчик, затем выполнение продолжается со следующей команды.



Исключение: команда, во время которой случилось прерывание, не выполняется до конца, сразу вызывается обработчик, затем прерванная команда выполняется еще раз.



Программа, обрабатывающая прерывания

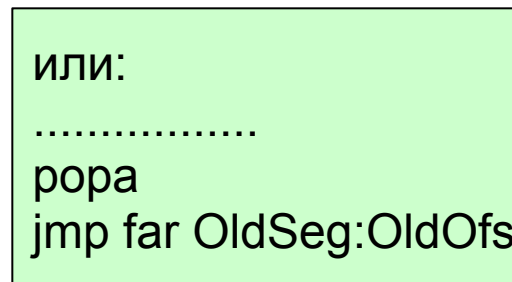
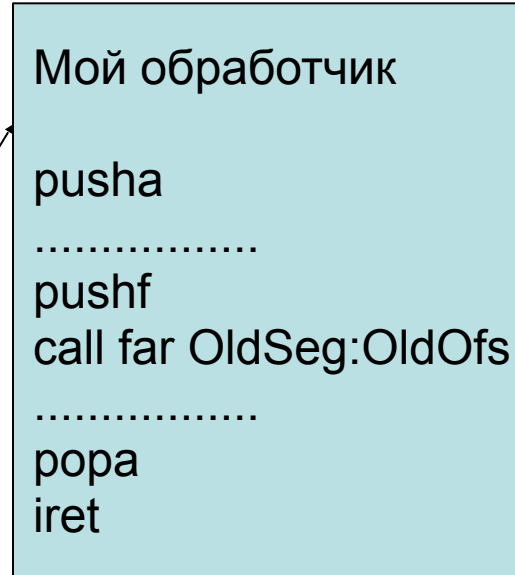
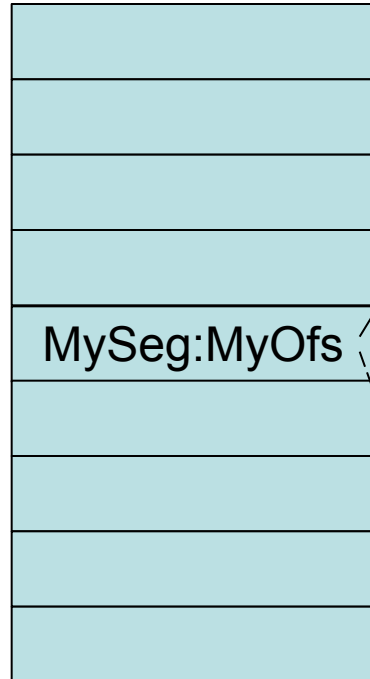
- Если программа завершается обычным образом - перед завершением она должна вернуть таблицу прерываний (и настройки контроллера) "как было"
- Программа может не завершаться, а оставаться резидентной в системе (**TSR - Terminate but Stay Resident, int 27h** или **функция 31h прерывания 21h**) - тогда обработчик в ее составе активируется при возникновении какого-то события, например, при нажатии комбинации клавиш, когда работают другие программы.

Цепочка обработчиков

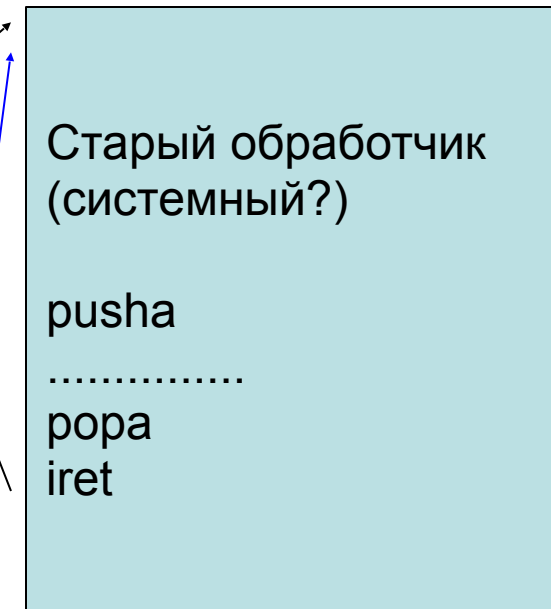
- Чаще всего в IBM PC в программах пытаются обрабатывать прерывания от таймера или клавиатуры (int 8 и int 9).
- Часто необходимо "поймать" событие, но оставить работоспособными системные обработчики, иначе, например, собьются часы компьютера.
- Устанавливая адрес своего обработчика в таблице прерываний, необходимо запомнить адрес старого обработчика и вызывать его из своего обработчика:
 - или при выходе
 - или вначале или в середине

Цепочка обработчиков

Таблица прерываний



Запомнили OldSeg, OldOfs



Спасибо за внимание.