

Весна 2021

Системное программное обеспечение

Онлайн-лекции

Лекция №7: **Процедуры**

Доцент, к.т.н. ГОЛЬЦОВ Александр Геннадьевич

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

MySeg:MyOfs

Мой обработчик

pusha

.....

pushf

call far OldSeg:OldOfs

.....

popa

iret

Старый обработчик
(системный?)

pusha

.....

popa

iret

или:

.....

popa

jmp far OldSeg:OldOfs

Запомнили OldSeg, OldOfs

Директива \$

Сущность, которой соответствует \$ - адрес. Это адрес текущей позиции компиляции, текущее смещение в текущем сегменте. Поместить это в регистр невозможно, сущность не представима 16 битами.

Не работает:

```
.data
aaa dw 25
bb = $           ;адрес
cc equ $+5       ;адрес
dd equ aaa-1     ;адрес
```

```
.code
mov ax,bb
mov bx,cc
mov dx,dd
```

Сработает:

```
.data
aaa dw 25
bb = offset $     ; число
cc equ $+5        ; адрес
dd equ $-aaa      ; число
```



```
.code
mov ax,bb
mov bx, offset cc ; еще seg, low, XX ptr
mov dx,dd
```

Процедура

- Фрагмент кода, существующий в памяти в единственном экземпляре, который можно многократно вызвать из любого места программы с последующим возвратом в место вызова.
- Может иметь параметры, задаваемые при каждом вызове. Параметры уточняют требуемые от процедуры действия.

Процедуры и стек

- Для возможности многократного вызова процедур любой степени вложенности используется стек.
- В стек при вызове процедуры всегда сохраняется адрес возврата - адрес места в программе, куда нужно перейти при выходе из процедуры.
- В процессорах i8086 в качестве адреса возврата сохраняется значение **IP** или **CS:IP** в случае ближнего и дальнего вызова соответственно.
- Принято предполагать, что процедуры имеют право портить все РОН; не принято сохранять РОН при входе в процедуры и восстанавливать при выходе - в отличие от обработчиков прерываний.

Ближний и дальний вызов

- Ближний - в пределах текущего сегмента кода. В коде команды прописывается число, **прибавляемое к текущему IP, т.е. смещение относительное**. За счет этого машинный код, содержащий только ближние вызовы, можно перемещать внутри сегмента кода без потери работоспособности.
- Дальний - с указанием сегмента и смещения процедуры, **независимо от того, в том же или в другом сегменте кода она расположена**. Позволяет адресоваться к любой точке в пределах 1М памяти.
- Дальний вызов в защищенном режиме старших моделей процессоров x86 протекает весьма долго.

Ближний и дальний возврат

- Команда RET на самом деле отсутствует, эта мнемоника уточняется ассемблером и генерируется одна из двух команд:
- RETN - ближний возврат, извлечь из стека сохраненный IP и за счет этого вернуться в точку вызова;
- RETF - дальний возврат, извлечь из стека CS:IP (младшее слово - IP, старшее - CS) и за счет этого вернуться в точку вызова.
- **Способ вызова (задается в каждом месте вызова) должен соответствовать способу возврата (задан в процедуре).**

Передача параметров

- **В глобальных ячейках памяти** - неудобно в случае необходимости рекурсивно вызывать процедуру: в этой ячейке уже лежат действующие параметры, а надо туда положить новые, а текущие - не потерять.
- **В регистрах** - вполне удобно, если параметров немного, а код процедуры не портит регистры активно.
- **В стеке** - удобно во всех случаях, но в регистрах может быть быстрее.

Метки и дальность

- Метки (просто метки в коде с двоеточием или имена процедур, см. ниже) всегда имеют атрибут "ближняя" (near) или "дальняя".
- Атрибут по умолчанию определяется **моделью памяти**, заданной для программы:
 - если моделью подразумевается **один сегмент кода** - метки ближние;
 - если возможно **много сегментов кода** - метки дальние.
- При использовании команды CALL дальность вызова определяется дальностью метки.

Модели памяти

Директива **.model** задает 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

Процедуры без "процедур"

Команды вызова и возврата существуют на уровне системы команд, и можно написать "программу с процедурами" без использования дополнительных "процедурных" синтаксических конструкций, пользуясь только метками:

```
.code
```

```
    mov    ax, 10 ; сколько раз повторять  
    call   MyProc
```

```
    .....
```

```
MyProc:
```

```
    ; что-то делать AX раз  
    ret
```

Команда CALL

Прямой вызов:

- CALL метка (или имя процедуры)
- CALL far/near ptr метка

Косвенный вызов (по указателю):

- CALL регистр (ближний вызов)
- CALL метка данных (dw - ближний, dd - дальний)
- CALL [косвенный адрес указателя] (если тип данных непонятен из адреса, то word/dword ptr)

Заносит в стек IP или CS:IP и помещает в IP или CS:IP адрес заданной точки входа в процедуру.

далее -
пример

```

.model large
.stack 100h
.data
ProcPtr dd MyProc
.code Other
MyProc label far
        inc ax
        retf
.code

        mov ax,@data
        mov ds,ax

        call far ptr m1
        call MyProc
        call ProcPtr
        mov cx, offset m2
        call cx
        mov bx, offset ProcPtr
        call dword ptr [bx]

        mov ax,4C00h
        int 21h

        m1:    retf
        m2:    retn

        end

```

далее -
в отладчике

cs:0000▶B8364F	mov	ax,4F36
cs:0003 8ED8	mov	ds,ax
cs:0005 0E	push	cs
cs:0006 E81900	call	0022
cs:0009 90	nop	
cs:000A 9A0400354F	call	4F35:0004
cs:000F FF1E0000	call	far [0000]
cs:0013 B92300	mov	cx,0023
cs:0016 FFD1	call	cx
cs:0018 BB0000	mov	bx,0000
cs:001B FF1F	call	far [bx]
cs:001D B8004C	mov	ax,4C00
cs:0020 CD21	int	21
cs:0022 CB	retf	
cs:0023 C3	ret	

Директива PROC

метка PROC тип_вызова

.....

метка ENDP

- Может использоваться в любом месте программы.
- Позволяет повысить читаемость кода.
- Позволяет задать дальность для вызова и возврата одновременно.
- Позволяет воспользоваться локальными символами (метками).
- Позволяет описать тип и имена параметров, передаваемых через стек.
- Позволяет сгенерировать код входа и выхода, как ЯВУ.

Отличие от процедур ЯВУ

- Команду выхода нужно писать явно, иначе выполнение просто продолжится следующей после **endp** командой.
- В процедуру можно попасть после выполнения написанной перед **proc** команды, как если бы в коде была объявлена обычная метка, а не начало процедуры.
- Можно (но нужно ли?) осуществлять переход извне на метку, расположенную внутри процедуры; это иногда удобно, если параметры передаются НЕ через стек.

Комментарии к процедуре

- Перед процедурой **ОБЯЗАТЕЛЬНО** должен быть комментарий:
 - ЧТО делает процедура в целом (назначение)
 - Какие ПАРАМЕТРЫ ей передаются и как
 - На какие глобальные данные полагается алгоритм
 - КАК она делает свое дело, каковы особенности алгоритма, какие предварительные действия требуются, что "портит" и т.п.

```

.model small
.stack 100h
.code
Incr    proc far
        inc     ax
        ret
Incr    endp

Start:
        call    Incr

        mov     ax, 4C00h
        int     21h

        end     Start

```

cs:0000	40	inc	ax
cs:0001	CB	retf	
cs:0002	0E	push	cs
cs:0003	E8FAFF	call	0000
cs:0006	B8004C	mov	ax, 4C00
cs:0009	CD21	int	21

```

.model small
.stack 100h
.data
Hello    db 'Hello, world$'
sPtr     dw ?
.code
;-----
; Выводит строку, смещение в sPtr
;-----
PrStr    proc
        mov     dx, sPtr
        mov     ah, 9
        int     21h

        ret
PrStr    endp

Start:
        mov     ax, @data
        mov     ds, ax

        mov     sPtr, offset Hello
        call    PrStr

        mov     ax, 4C00h
        int     21h

        end     Start

```

Параметры через фиксированную область памяти

Параметры через регистр AX

```

.model small
.stack 100h
.data
Hello db 'Hello, world$'
.code
;-----
; Выводит строку, смещение в AX
;-----
PrStr    proc
        mov     dx, ax
        mov     ah, 9
        int     21h

        ret
PrStr    endp

Start:
        mov     ax, @data
        mov     ds, ax

        mov     ax, offset Hello
        call    PrStr

        mov     ax, 4C00h
        int     21h

        end     Start

```

Параметры через стек

```

.model small
.stack 100h
.data
Hello    db 'Hello, world$'
.code
;-----
; Выводит строку, смещение в стеке
;-----
PrStr    proc
        mov     bp, sp
        mov     dx, [bp+2]
        mov     ah, 9
        int     21h

        ret
PrStr    endp

Start:
        mov     ax, @data
        mov     ds, ax

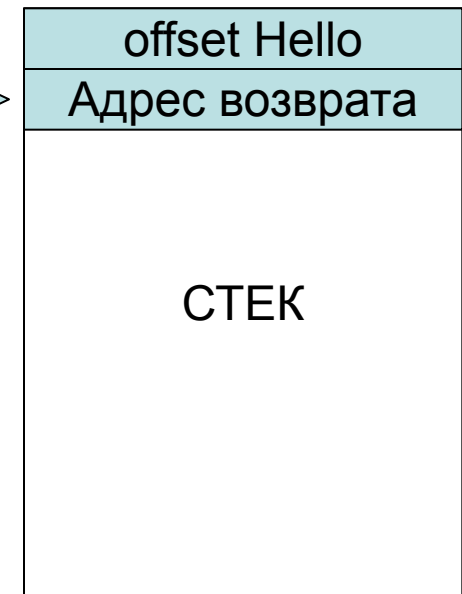
        push     offset Hello
        call     PrStr

        mov     ax, 4C00h
        int     21h

        end     Start

```

bp = sp



Обратите внимание

- Из `push <константа>` получается "нечто":

```
push    ax
push    bp
mov     bp,sp
mov     word ptr [bp+02],0000
pop     bp
```

- А если написать `.286`, то гораздо лучше (нужная команда появилась позже):

```
push    0000
```

- Если процедуру объявить `far` или поменять модель на `large`, то `[bp+2] → [bp+4]`

Что плохо в примере

- После вызова процедуры параметры остались болтаться в стеке! Стек должен быть возвращен в исходное состояние.
- Хотелось бы называть параметры внутри процедуры именами, а не `[bp+2]`.
- Если такую процедуру вызывать из другой, тоже использующей `bp`, надо `bp` сохранять. Так **принято**.
- Если в процедуре есть локальные переменные - их удобно расположить в стеке НИЖЕ адреса возврата по адресам `[bp-2]` и т.д.

Начало и конец процедуры

- **Начало:**

push bp ; для сохранения
mov bp, sp ; база - копия bp
sub sp, Vars ; память под локальные переменные

- **Теперь**

[bp] → сохраненная копия bp

[bp+2] → адрес возврата

[bp+4] → первый параметр

sp указывает на первый байт области локальных переменных,
помещение чего-то в стек пойдет ниже

- **Конец:**

add sp, Vars; очистить локальные переменные

pop bp ; восстановить bp

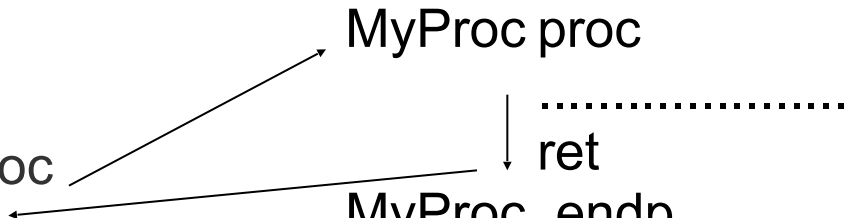
ret ; или ret N возврат

Освобождение стека от параметров

- Стиль языка Си: освобождает вызывающий код

```
push 10  
push 20  
call MyProc  
add sp,4
```

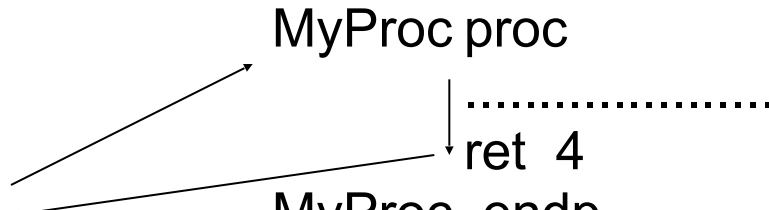
MyProc proc
.....
ret
MyProc endp



- Стиль Паскаля и Windows API: освобождает процедура

```
push 10  
push 20  
call MyProc
```

MyProc proc
.....
ret 4
MyProc endp



Директива ARG

- Пишется ниже PROC
- Позволяет назначить параметрам процедуры символьные имена
- Эти имена являются локальными идентификаторами, действующими внутри процедуры
- Устанавливает связь каждого имени с $[bp+N]$
- Имена перечисляются в том порядке, в котором перечислялись бы на ЯВУ
- Дополнительно есть модель .tpascal и .pascal, задающая прямой порядок расположения параметров в стеке (на Си первый параметр лежит последним, т.е. по адресу $[bp+max]$)

Директива ARG

ARG arg1, arg2, ... [= sze [RETURNS res1, res2, ...]]

- Подразумевается, что результаты подлежат оставлению в стеке, а аргументы - вычищаются
- На ЯВУ часто результаты функций возвращаются в AL, AX или DX:AX (или в стеке)

- Аргумент это:

Name [: Type [: Count]]

Type = Byte, Word, Dword,
Near ptr <Type>, Far ptr <Type>
(например Near ptr word)

- Имени sze сопоставляется размер области параметров

Примеры

My proc near

ARG a: word, b: word
; a=[bp+4], b=[bp+6]

My proc far

ARG a, b
; a = [bp+6], b=[bp+8]

My proc far

ARG a: word, b: far ptr qword:4, c:dword = size
; a=[bp+6], b=[bp+8], c = [bp+24], size = 22

Директива LOCAL

LOCAL var1, var2, [= size]

- Аналогично ARG, ассоциирует имена локальных переменных с [bp-N]
- **НЕ производит** резервирования памяти, т.е. не генерирует код, вычитающий из SP размер локальных переменных
- Но для этого удобно вычисляющееся Size
- var<i> записываются по тем же правилам, что и arg<i> в директиве ARG

Директива USES

USES элем1, элем2, ...

- Элементом является регистр или метка данных.
- Генерирует код, помещающий в стек указанные элементы при входе в процедуру и извлекающий при выходе.
- Метка должна соответствовать ОДНОЙ единице данных, а не последовательности байт, слов и т.д.
- Требуется явное указание в модели языка:
.model Small, C
.model Compact, Pascal

Локальные идентификаторы

- LOCALS - включает локальность внутри процедур
- NOLOCALS - выключает локальность
- По умолчанию, локальные идентификаторы должны иметь префикс @@, например, @@m1
- Префикс можно задать, указав в директиве LOCALS, например

```
LOCALS @1
PrStr  proc far
        ARG a,b
        USES ax, bx, n
        → @1m1:
            mov     ax, a
            mov     bx, b
            |
            ret
        PrStr  endp

AA      proc
        → @1m1:
            endp
```


Спасибо за внимание.