Лекция 14. Стековые команды. Процедуры

Стековые команды

Процессор может обращаться к данным в памяти *без указания их адресов* по принципу LIFO

Last Input First Output – «последний записанный считывается первым»

- Такой способ обращения реализован в стековых командах:

  PUSH операнд (Запись в стек)

  POP операнд (Чтение из стека)

- Сегмент данных, к которому применяется такой способ обращения, называют «стековым» сегментом данных

Механизм выполнения стековых команд процессором

- В командах PUSH/POP адрес не задается!

  Процессор будет обращаться к памяти по адресу SS: SP

  SS – указатель сегмента стека,

  SP – внутрисегментный адрес последнего записанного в стек значения («вершина стека»)

- Процессор автоматически изменяет внутрисегментный адрес в регистре SP при выполнении стековых команд

- Команды PUSH и POP выполняются над словами или двойными словами (не байтами!).

Механизм выполнения стековой записи: PUSH

- Запись в стек (PUSH) процессор ведет в сторону уменьшения внутрисегментных адресов !
- Перед записью процессор уменьшает SP на длину записываемого операнда и только затем выполняет запись

Пример: Пусть, SP = 6 . В регистрах DX = 2053h, EBX = 1F 03 FA 05h . Выполнены команды записи в стек: push dx и push ebx

Механизм выполнения стекового чтения: POP

- Процессор выполняет чтение из памяти, начиная с адреса SS:SP
  Затем увеличивает SP на длину прочитанного операнда.

Пример: Пусть SP=0. После выполнения команды POP CX в регистре CX будет код FA 05

Состояние стека до чтения

SP→	0	05
	1	FA
	2	03
	3	1F
	4	53
	5	20
	6	??
	. .	??

Стек после чтения

	0	05
	1	FA
SP→	2	03
	3	1F
	4	53
	5	20
	6	??
	. .	??

Набор стековых команд в системе команд

- POP/PUSH r/m16 ; чтение/запись слов
  POP/PUSH r/m32 ; чтение/запись двойных слов

- PUSH i16 или i32 ; запись непосредственной величины (слово или двойное слово)

- PUSHA ; запись в стек всех 16-разр.регистров
  POPA ; чтение из стека в 16-разр.регистры

- PUSHAD ; запись в стек всех 32-разр. регистров
  POPAD ; чтение из стека в 32-разр. регистры

Использование стековых команд

- Стековыми командами пользуются для быстрого сохранения и последующего восстановления состояния регистров.
- Последовательность чтения из стека должна быть обратна последовательности записи !!

Пример:

```
        push bx    ; сохранили в стек коды из bx и cx
        push cx
    . . . . .
        pop cx     ; прочитали из стека значения обратно в cx и bx
        pop bx
```

Нужен ли собственный стековый сегмент в программе?

1) Создавать его не обязательно. Операционная система создает общий стек для использования всеми исполняемыми программами.

2) Если вы хотите создать свой стек (пример создания стека на 10 слов):

```
      stg  segment   stack  use16
           dw   10 dup (?)
      stg  ends
      . . . . .
      cod  segment   use16
    assume cs: cod, ss: stg, . . .
m1:    . . .
           mov       ax, stg
           mov ss, ax
         . . .
```

Использование указания stack в директиве segment позволяет не заботиться о занесении в SP адреса «вершины стека». Это сделает ОС при загрузке исполняемого кода в память.

Процедурная передача управления (процедуры)

- Это механизм передачи управления с возвратом в точку кодового сегмента, откуда был сделан вызов
- Команды процессора:

CALL адрес процедуры - Вызов процедуры (внутрисегментный или межсегментный CALL)

RET – Возврат из процедуры

Часто повторяющиеся фрагменты кода можно оформить как процедуру для сокращения объема машинного кода (но не времени его исполнения!)

Механизм выполнения команд CALL и RET

- Внутрисегментный CALL (пример: call met1)

  Процессор сохраняет в стеке текущее значение IP (адрес следующей за CALL команды):

  SP← SP -2; SS:SP ← IP

  и затем изменяет внутрисегментный адрес в IP, аналогично JMP

- Внутрисегментный RET

  Процессор считывает в IP сохраненный в стеке адрес :

  IP ← SS:SP; SP← SP +2

- При межсегментном CALL (пример: call far ptr cs:met2)

  процессор сохраняет в стеке текущие значения CS и IP.

  SP ← SP-4; SS:SP← IP ; SS:SP+2 ← CS

  При выполнении RET он считывает их в CS и IP из стека

Описание процедуры для транслятора

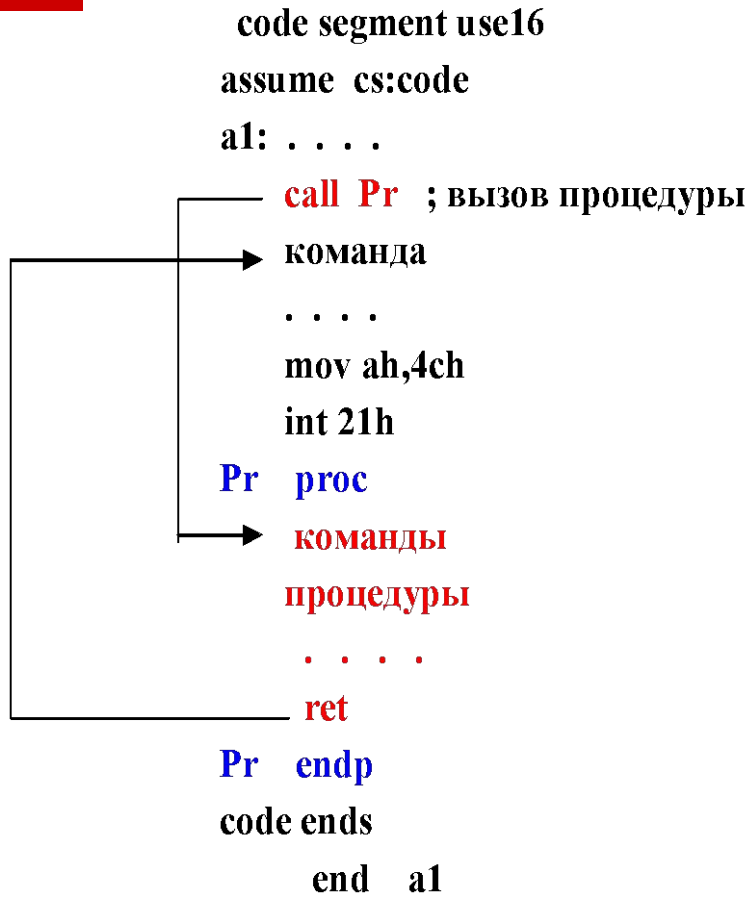
```
Имя процедуры   PROC      ; директива транслятора
                команды процедуры
                RET
Имя процедуры   ENDP      ; директива транслятора
```

- Процедуры размещают:
  - в отдельном кодовом сегменте для доступности другим кодам
  - локальную процедуру - в текущем кодовом сегменте после последней команды
- Параметры для процедуры (входные и/или выходные, если они предусмотрены) передают через регистры или память

  Адреса памяти в качестве параметров должны задаваться косвенно!

Пример:

Структура односегментной программы с внутренней локальной процедурой



Пример. Создать универсальную процедуру для обнуления любого массива байтов. Входные параметры: адрес массива в памяти и его длина. Использовать ее для обнуления своего массива данных.

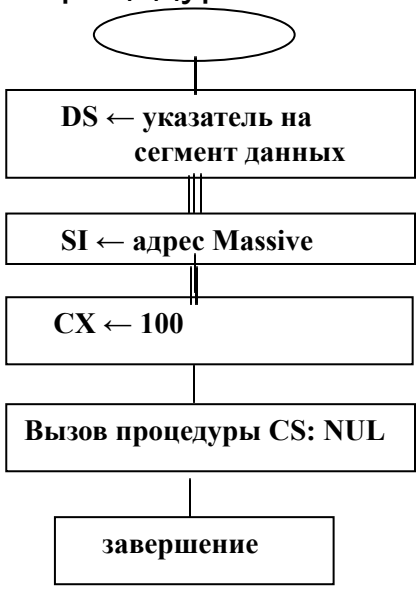
1. *Структура программы*
   - Сегмент данных Mydat, указатель на сегмент- ds
   - Кодовый сегменте Mycod, указатель на сегмент- cs
   - Кодовый сегмент Prcod с процедурой, указатель на сегмент- cs
2. *Размещение данных в памяти и регистрах*
   - ds:massive - адрес моего массива данных, 100 байт
3. *Параметры процедуры*

   Входные параметры: DS:SI – адрес массива,
   CX – длина массива (в байтах)

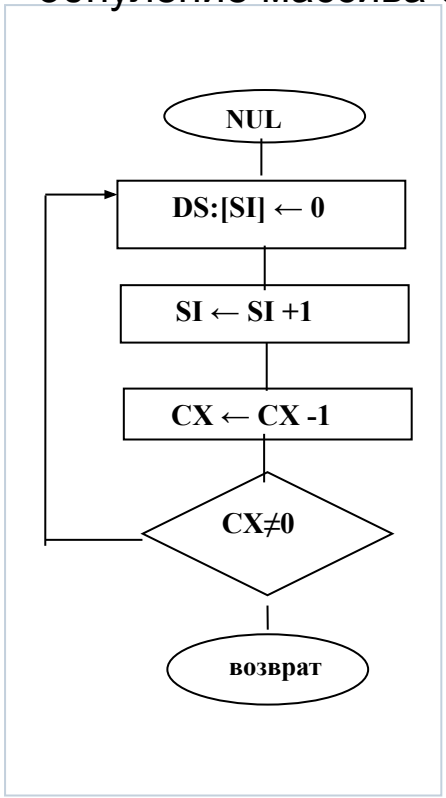
   Выходные параметры: нет

4. Алгоритмы основной программы и процедуры

Алгоритм основной программы:
- задание входных параметров для процедуры и ее вызов



Алгоритм процедуры NUL:
- обнуление массива байтов



5. Исходный текст программы

```
.386
Mydat  segment use16
    massive  db 100 dup (0FFh)
Mydat  ends

Mycod  segment use16
assume  ds:Mydat,  cs:Mycod
; загрузка регистра-указателя сегмента данных
 m1:  mov dx, Dseg
      mov ds, dx
      lea si, massive
      mov cx, 100
      call far ptr cs:nul
; выгрузка из памяти
     mov ah, 4ch
     int 21h
Mycod ends

Prcod  segment use16
assume cs:Prcod
   nul   proc
   zer:  mov byte ptr ds:[si], 0
         inc si
         loop zero
         ret
   nul   endp
Prcod ends
         end m1
```