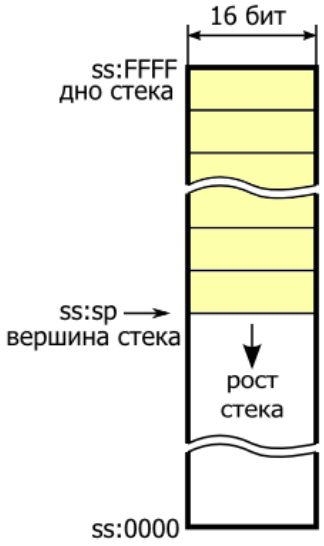
Лабораторная работа 4. Подпрограммы

Стек

Стеком называется структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является неотъемлемой частью архитектуры процессора и поддерживается на аппаратном уровне: в процессоре есть специальные регистры (SS, BP, SP) и команды для работы со стеком.

Обычно стек используется для сохранения адресов возврата и передачи аргументов при вызове процедур, также в нём выделяется память для локальных переменных. Кроме того, в стеке можно временно сохранять значения регистров.

Схема организации стека в процессоре 8086 показана на рисунке:



Стек располагается в оперативной памяти в сегменте стека, и поэтому адресуется относительно сегментного регистра SS. Шириной стека называется размер элементов, которые можно помещать в него или извлекать. В нашем случае ширина стека равна двум байтам или 16 битам. Регистр SP (указатель стека) содержит адрес последнего добавленного элемента. Этот адрес также называется вершиной стека. Противоположный конец стека называется дном.

Дно стека находится в верхних адресах памяти. При добавлении новых элементов в стек значение регистра SP уменьшается, то есть стек растёт в сторону младших адресов. Как вы помните, для COM-программ данные, код и стек находятся в одном и том же сегменте, поэтому если постараться, стек может разрастись и затереть часть данных и кода.

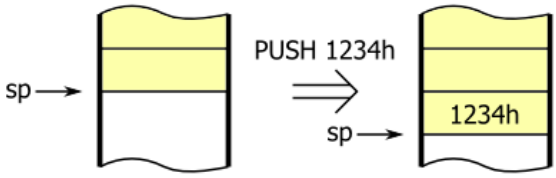
Для стека существуют всего две основные операции:

- добавление элемента на вершину стека (**PUSH**);
- извлечение элемента с вершины стека (**POP**);

Добавление элемента в стек

Выполняется командой PUSH. У этой команды один операнд, который может быть непосредственным значением, 16-битным регистром (в том числе сегментым) или 16-битной переменной в памяти. Команда работает следующим образом:

1. значение в регистре SP уменьшается на 2 (так как ширина стека — 16 бит или 2 байта);
2. операнд помещается в память по адресу в SP.



Примеры:

```
push -5           ;Поместить -5 в стек
push ax           ;Поместить AX в стек
push ds           ;Поместить DS в стек
push [x]          ;Поместить x в стек (x объявлен как слово)
push word [bx]    ;Поместить в стек слово по адресу в BX
```

Существуют ещё 2 команды для добавления в стек.

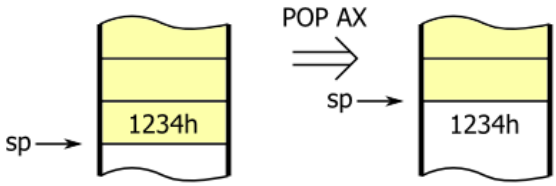
Команда **PUSHF** помещает в стек содержимое регистра флагов. Команда **PUSHA** помещает в стек содержимое всех регистров общего назначения в следующем порядке: AX, CX, DX, BX, SP, BP, SI, DI (значение DI будет на вершине стека). Значение SP помещается то, которое было до выполнения команды. Обе эти команды не имеют операндов.

Извлечение элемента из стека

Выполняется командой **POP**. У этой команды также один операнд, который может быть 16-битным регистром (в том числе сегментым, но кроме CS) или 16-битной переменной в памяти. Команда работает следующим образом:

1. операнд читается из памяти по адресу в SP;
2. значение в регистре SP увеличивается на 2.

Обратите внимание, что извлеченный из стека элемент не обнуляется и не затирается в памяти, а просто остаётся как мусор. Он будет перезаписан при помещении нового значения в стек.



Примеры:

```
pop cx            ;Поместить значение из стека в CX
pop es            ;Поместить значение из стека в ES
pop [x]           ;Поместить значение из стека в переменную x
pop word [di]     ;Поместить значение из стека в слово по адресу в DI
```

Соответственно, есть ещё 2 команды. **POPF** помещает значение с вершины стека в регистр флагов. **POPA** восстанавливает из стека все регистры общего назначения (но при этом значение для SP игнорируется).

Пример.

```
; Написать программу формирования строки из исходной путем
; отображения исходной строки. Пример:
; Исходная строка: "mama myla ramu$"
; Итоговая строка: "umar alym amam"
use16
```

```
org 100h

n = 32          ; количество символов исходной строки
mov dx, str1    ; помещаем в dx для вывода на экран
mov ah, 9       ;
int 21h

mov di, str1
mov bx, str2

;посимвольно запихнем строку в стек
repeat n
      mov ax, [di]
      push ax
      inc di
end repeat

;вытолкнем символы из стека в строку
mov di, str2
repeat n
      pop ax
      mov [di], ax
      inc di
end repeat

;сделаем последним символом "$"
mov ax, '$'
mov [di],ax

; перенесем на новую строку
mov ah, 9
mov dx, strr
int 21h

; выведем итоговую строку
mov dx, str2
int 21h

; задержка экрана
mov ah, 8
int 21h

; конец программы
mov ax, 4C00h
int 21h
;----------------
str1 db "mama myla ramu, papa myl mashinu$"
str2 rb 35
strr db 13,10,'$'
```

Процедуры

Процедура представляет собой код, который может выполняться многократно и к которому можно обращаться из разных частей программы. Обычно процедуры предназначены для выполнения

каких-то отдельных, законченных действий программы и поэтому их иногда называют подпрограммами.

Команды CALL и RET

Для работы с процедурами предназначены команды CALL и RET. С помощью команды CALL выполняется вызов процедуры. Эта команда работает почти также, как команда безусловного перехода (JMP), но с одним отличием — одновременно в стек сохраняется текущее значение регистра IP. Это позволяет потом вернуться к тому месту в коде, откуда была вызвана процедура. В качестве операнда указывается адрес перехода, который может быть непосредственным значением (меткой), 16-разрядным регистром (кроме сегментных) или ячейкой памяти, содержащей адрес.

Возврат из процедуры выполняется командой RET. Эта команда восстанавливает значение из вершины стека в регистр IP. Таким образом, выполнение программы продолжается с команды, следующей сразу после команды CALL. Обычно код процедуры заканчивается этой командой. Команды CALL и RET не изменяют значения флагов (кроме некоторых особых случаев в защищенном режиме).

Небольшой пример разных способов вызова процедуры:

```
use16                   ;Генерировать 16-битный код
org 100h                ;Программа начинается с адреса 100h

    mov ax,myproc
    mov bx,myproc_addr
    xor si,si

    call myproc         ;Вызов процедуры (адрес перехода - myproc)
    call ax             ;Вызов процедуры по адресу в AX
    call [myproc_addr]  ;Вызов процедуры по адресу в переменной
    call word [bx+si]   ;Более сложный способ задания адреса

    mov ax,4C00h        ;\
    int 21h             ;/ Завершение программы

;-----------------------------------------------------------------
;Процедура, которая ничего не делает
myproc:
    nop                 ;Код процедуры
    ret                 ;Возврат из процедуры
;-----------------------------------------------------------------
myproc_addr dw myproc   ;Переменная с адресом процедуры
```

Передача параметров

Очень часто возникает необходимость передать процедуре какие-либо параметры. Например, если вы пишете процедуру для вычисления суммы элементов массива, удобно в качестве параметров передавать ей адрес массива и его размер. В таком случае одну и ту же процедуру можно будет использовать для разных массивов в вашей программе. Самый простой способ передать параметры — это поместить их в регистры перед вызовом процедуры.

Возвращаемое значение

Кроме передачи параметров часто нужно получить какое-то значение из процедуры. А если процедура что-то делает, то полезно узнать, завершилось действие успешно или возникла ошибка. Существуют разные способы возврата значения из процедуры, но самый часто используемый — это поместить значение в один из регистров. Обычно для этой цели используют регистры AL и AX.

Сохранение регистров

Хорошим приёмом является сохранение регистров, которые процедура изменяет в ходе своего выполнения. Это позволяет вызывать процедуру из любой части кода и не беспокоиться, что значения в регистрах будут испорчены. Обычно регистры сохраняются в стеке с помощью команды PUSH, а перед возвратом из процедуры восстанавливаются командой POP. Естественно, восстанавливать их надо в обратном порядке. Примерно вот так:

```
myproc:
    push bx             ;Сохранение регистров
    push cx
    push si
    ...                 ;Код процедуры
    pop si              ;Восстановление регистров
    pop cx
    pop bx
    ret                 ;Возврат из процедуры
```

Получение строки символов

Функция 0Ah прерывания 21h позволяет вводить строку длиной до 254 символов, выдавая эхо на терминал. Эта процедура продолжает ввод поступающих символов до тех пор, пока не нажата клавиша «возврат каретки». DX указывает на адрес памяти, куда должна быть помещена строка. При входе первый байт в этой позиции должен содержать число байтов, отводимых для этой строки. После того как строка введена, второй байт даст число реально введенных символов.

Сама строка начинается с третьего байта. Таким образом, нужно отвести достаточно памяти для строки нужной длины, плюс 2 байта для дескриптора строки и 1 добавочный байт для возврата каретки (CR). Например, для получения 50-символьной строки надо отвести 53 байта памяти.

Пример. Ввести строку с клавиатуры, вывести её на экран.

```
use16
org 100h

    mov dx, s1   ;адрес строки
    mov [s1], 51 ;1-й байт - макс. количество символов в строке
    mov ah, 0ah  ;функция DOS для ввода строки
    int 21h

    call endline ;вызываем функцию перехода на новую строку

    mov dx, s1   ; в dx адрес введенной строки
    call printline  ;функция вывода введенной строки

    call endline ;вызываем функцию перехода на новую строку

    mov ah,8
    int 21h

int 20h

;------------ функции ----------

;------------------------------
;  перевод на новую строку
;------------------------------
endline:
```

```
        push ax   ;помещаем данные из регистров
        push dx   ;в стек
        mov ah, 9
        mov dx, endlen ;адрес строки, хранящей 10,13,'$'
        int 21h
        pop dx    ;восстанавливаем регистры
        pop ax
        ret
    ;------------------------------

    ;------------------------------
    ; вывод введенной строки
    ;------------------------------
    printline:
        push ax        ;помещаем данные из регистров
        push bx        ;в стек
        push dx

        mov ah,9
        mov bx,dx      ;1й байт хранит макс. число символов в строке
        inc bx         ;а 2й - фактическое
        mov al,[bx]    ;фактическое число символов в региср AL
        add bl,al        ;прибавляем это число к bl, чтобы найти конец
строки
        inc bl         ;и переходим к следующему символу
        mov byte [bx],'$' ;ставим там конец строки
        add dx,2       ;пропускаем первые 2 байта
        int 21h        ;и выводим строку

        pop dx
        pop bx
        pop ax
        ret
    ;------------------------------

    s1 db 53 dup(?)
    endlen db 10,13,'$'
```

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Выбрать вариант в соответствии со списком. Данные вводить с клавиатуры. Результат выводить на экран. При решении использовать процедуры.

ВАРИАНТЫ

1. Сколько раз в веденной строке встречается число 10?
2. Повторить все символы строки заданное количество раз.
3. Подсчитать в строке число букв A и B, если букв A больше, чем B, то удалить в строке все символы B.
4. Подсчитать количество слов в данном предложении.
5. Определить, сколько раз в строке встречается данный символ.
6. Дана символьная строка, заканчивается точкой. Найти длину самого длинного слова.
7. Имеется строка символов, содержащая буквы латинского алфавита и цифры. Определить количество цифр в строке.
8. Преобразовать строку, заменив все «:» на запятые, встречающиеся среди первых [n/2] символов и заменив точками все «!», встречающиеся среди символов, стоящих после [n/2] символов.

9. В предложении найти слова, начинающиеся и заканчивающиеся на одну и ту же букву.
10. Заменить в заданной строке знак "!" на сочетание "???".
11. Определить сколько раз в тексте встречаются гласные буквы.
12. В данной строке подсчитать сумму цифр, содержащихся в ней.
13. В заданной строке удвоить каждый символ.
14. Вводится строка. Поставить запятую после каждого пробела данной строки.
15. Вводится строка. Удалить из строки лишние пробелы.
16. Вводится строка. Удалить из строки пробелы, стоящие перед запятыми.
17. Дана строка и число n. Верно ли, что в ней есть по крайней мере n подряд идущих букв a?
18. Дана строка. Удалить из нее последовательности символов, расположенные между скобками. Скобки также удалить. После первой открывающей скобки другие открывающие скобки игнорируются. Если скобка не закрывается до конца строки, то удалять все до конца строки. Закрывающая скобка без парной открывающей игнорируется.
19. Заменить в исходной строке все 0 на заданный символ.
20. В исходной строке вместо заданного символа вставьте 0.
21. Удалить из строки все одинаковые символы, идущие подряд.
22. Добавить в строку недостающие пробелы (после знаков препинания обязательно должен быть пробел).
23. Подсчитать в строке число букв A и B. Найти разницу N между этими числами и вывести N раз букву, которая встречается чаще.
24. Определить является ли заданная строка палиндромом без учета регистра.
25. Написать программу, разбивающую текст на строки длиной не N символов.
26. Поменять местами в предложении все 0 и 1.