Лекция 12

3.2. Передача параметров в процедуру

Как видим, аналогии с передачей параметров в процедуру как в языках высокого уровня нет. Следовательно, нет списка параметров. Поэтому программист сам должен позаботиться о такой передаче как в процедуру, так и из нее.

Можно использовать несколько вариантов:

1. Через регистры;
2. Через стек;
3. Через общие области памяти

3.2.1. Передача параметров через регистры

Основная программа размещает параметры в определенных регистрах, а процедура их оттуда берет. Процедура записывает результаты в регистры, а программа их потом считывает.

Например:

```
определить c = max(a,b); a запишем в AX, b - в BX, а результат c - в AX.
основная программа
MOV AX, A
MOV BX, B
CALL MAX
MOV C, AX; защита AX
…….
MAX PROC FAR
CMP AX, BX
JGE MAX1
MOV AX, BX
MAX1: RET
MAX ENDP
```

Защита регистров в процедурах

Регистров в МП мало, поэтому процедура и программа используют одни и те же регистры. Чтобы процедура не изменяла содержание регистров в главной программе, нужно их защитить. То есть, переслать содержание регистров в стек, а затем обновить в обратном порядке. Это рекомендуется делать даже тогда, когда программа и процедура используют разные регистры. Со временем основная программа может измениться, и тогда процедура будет “мешать”.

Для облегчения таких действий в МП, начиная с 80186, внедрены команды **PUSHA** и **POPA**, которые записывают и считывают из стека содержание сразу 8 регистров: **AX, BX, CX, DX, DI, SI, SP, BP**. Также разрешено пересылать константы в стек - **PUSH 125**.

Передача параметров сложных типов

Кроме параметров значений, в процедуру можно передавать адреса. Когда параметром является сложный тип - структура или массив, то на машинном уровне передают не сами данные, а начальный адрес. Зная количество элементов и начальный адрес или поля, можно получить доступ к каждому элементу.

Например:

```
для двух массивов чисел без знака
X DB 100 DUP(?)
Y DB 50 DUP(?)
Определить  max (X[i])+max(Y[i]).
```

Ясно, что лучше сделать процедуру определения максимального элемента. У нее будет два параметра: начальный адрес и количество элементов.

```
; процедура MAX: AL = макс. элемент

MAX PROC
        PUSH BX ; начальный адрес
        PUSH CX
         MOV AL,0
MAX1: CMP [BX],AL
            JLE MAX2
            MOV AL[BX]
MAX2: INC BX; к следующему эл-ту
           LOOP MAX1
            POP CX
            POP BX
            RET
MAX ENDP
; Основная программа
LEA BX,X
MOV CX,100
CALL MAX
MOV DL,AL; защита AL
LEA BX,Y
```

Защита регистров

Если нет гарантии, что процедура может изменить содержание регистров, которые должны сохраниться после выхода из процедуры такими, как непосредственно перед вызовом процедуры, то их нужно на время выполнения заслать в стек.

Например:

```
все регистры общего назначения:
PUSH AX
PUSH BX
PUSH CX
PUSH DX
```

Перед выходом из процедуры - командой RET нужно эти регистры обновить:

```
POP DX
POP CX
POP BX
POP AX
```

Это делаем в обратном порядке. Это же касается индексных регистров и младшего байта регистра флагов.

3.2.2. Передача параметров через стек

Если в процедуре много параметров, то может не хватить регистров. Поэтому в этом случае можно использовать стек. Перед вызовом процедуры записываем параметры в стек в порядке, который определяет программист.

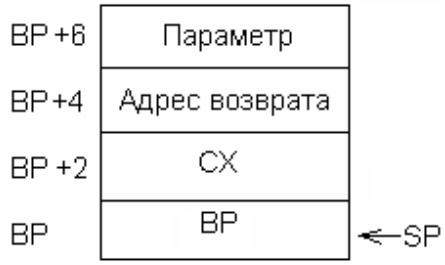
При этом применяется своеобразная методика работы со стеком не с помощью команд **push** и **pop**, а с помощью команд **mov** с косвенной адресацией через регистр **BP**, который архитектурно предназначен именно для адресации в стеке.

Пример:

Нужно, чтобы параметр (условная величина задержки) передавался в подпрограмму через стек. Вызов подпрограммы delay выполняется следующим образом:

```
0001  50 55 8B EC C7 46 02+  push 2000 ;Проталкиваем в стек значение параметра 07D0 5D
000B  E8 0005          call delay   ;Вызываем подпрограмму delay
000E  B8 4C00          mov ax,4c00h
0011  CD 21            int 21h
0013                   MAIN ENDP
0013                   delay proc          ;Процедура-подпрограмма
0013  51               push CX             ;Сохраним CX      основной программы
0014  55               push BP             ;Сохраним BP
0015  8B EC            mov BP,SP           ;Настроим BP      на текущую вершину стека
0017  8B 4E 06         mov CX, [BP+6]      ;Скопируем из стека параметр
001A  51               del1:  push CX      ;Сохраним его
001B  B9 0000          mov CX,0            ;Счетчик внутреннего      цикла
001E  E2 FE            del2:  loop del2    ;Внутренний цикл(64К      шагов)
0020  59               pop CX              ;Восстановим      внешний      счетчик
0021  E2 F7            loop del1           ;Внешний цикл
0023  5D               pop BP              ;Восстановим      BP
0024  59               pop CX              ;и CX программы
0025  C2 0002          ret  2              ;Возврат и снятие со стека ненужного параметра
0028                   delay ENDP
```

Команда **call**, передавая управление подпрограмме, сохраняет в стеке адрес возврата в основную программу. Подпрограмма сохраняет в стеке еще два 16-разрядных регистра. В результате стек оказывается в состоянии:



После сохранения в стеке исходного содержимого регистра **BP** (в основной программе нашего примера этот регистр не используется, однако в общем случае это может быть и не так), в регистр **BP** копируется содержимое указателя стека, после чего в **BP** оказывается смещение вершины стека. Далее командой mov в регистр **CX** заносится содержимое ячейки стека, на 6 байтов ниже текущей вершины. В этом месте стека как раз находится передаваемый в подпрограмму параметр, как это показано в левом столбце рис.

```
CPU 80486                                                    1
#lec12ed#29: push BP ;сохраним BP                    ax 0E35   c=0
  cs:0014 55              push   bp                  bx 2DB3   z=0
#lec12ed#30: mov BP,SP ;Настроим BP на текущую вершину стека  cx 2DB3   s=0
  cs:0015▶8BEC            mov    bp,sp               dx 24C4   o=0
#lec12ed#31: mov CX, [BP+6] ;Скопируем из стека параметр 2000 si 3092   p=0
  cs:0017 8B4E06          mov    cx,[bp+06]          di 3092   a=0
#lec12ed#del1: del1: push CX ;Сохраним его           bp 0100   i=1
  cs:001A 51              push   cx                  sp 0136   d=0
#lec12ed#33: mov CX,0 ;счетчик внутреннего цикла     ds 5BCE
  cs:001B B90000          mov    cx,0000             es 5BCE
#lec12ed#del2: del2: loop del2 ;внутренний цикл(64k шагов - 6 ss 5BDE
  cs:001E E2FE            loop   #lec12ed#del2 (001E) cs 5BF4
#lec12ed#35: pop CX ;восстановим внешний счетчик     ip 0015
  cs:0020 59              pop    cx
#lec12ed#36: loop del1 ;внешний цикл CX=CX-1

  ds:0000 CD 20 FF 9F 00 9A F0 FE =  Я ъЁ▮           ss:013E 5BCE
  ds:0008 1D F0 E0 01 68 22 AA 01 ↔Ёр☺h"к☺          ss:013C 07D0
  ds:0010 68 22 89 02 C3 1C 35 0E h"Й☻├∟5♫          ss:013A 000E
  ds:0018 01 01 01 00 02 FF FF FF ☺☺☺ ☻              ss:0138 2DB3
  ds:0020 FF FF FF FF FF FF FF FF                    ss:0136▶0100
```

Конкретную величину смещения относительно вершины стека надо для каждой подпрограммы определять индивидуально, исходя из того, сколько слов сохранено ею в стеке к этому моменту. Напомним, что при использовании косвенной адресации с регистром **BP** в качестве базового, по умолчанию адресуется стек, что в данном случае и требуется.

Выполнив возложенную на нее задачу, подпрограмма восстанавливает сохраненные ранее регистры и осуществляет возврат в основную программу с помощью команды **ret**, в качестве аргумента которой указывается число байтов, занимаемых в стеке отправленными туда перед вызовом подпрограммы параметрами. В нашем случае единственный параметр занимает 2 байта. Если здесь использовать обычную команду **ret** без аргумента, то после возврата в основную программу параметр останется в стеке, и его надо будет оттуда извлекать (между прочим, не очень понятно, куда именно, ведь все регистры у нас могут быть заняты). Команда же с аргументом, осуществив возврат в вызывающую программу, увеличивает содержимое указателя стека на значение ее аргумента, тем самым осуществляя логическое снятие параметра. Физически этот параметр, как, впрочем, и все остальные данные, помещенные в стек, остается в стеке и будет затерт при дальнейших обращениях к стеку.

Разумеется, в стек можно было поместить не один, а сколько угодно параметров. Тогда для их чтения надо было использовать несколько команд **mov** со значениями смещения **BP+6, BP+8, BP+0Ah** и т.д.(см. видео приложение к лекции)

Рассмотренная методика может быть использована и при дальних вызовах подпрограмм, но в этом случае необходимо учитывать, что дальняя команда **call** сохраняет в стеке не одно, а два слова, что повлияет на величину рассчитываемого смещения относительно вершины стека.

3.2.3. Передача параметров через общие области памяти

Остановимся на последнем варианте. Если процедура и вызов к ней размещены в разных сегментах кода, то данные можно разместить в самом начале и использовать их имена с директивами **PUBLIC** и **EXTRN**.

Директива **EXTRN** предназначена для объявления некоторого имени внешним по отношению к данному модулю. Это имя в другом модуле должно быть объявлено в директиве **PUBLIC**.

Директива **PUBLIC** предназначена для объявления некоторого имени, определенного в этом модуле и видимого в других модулях.

Синтаксис этих директив следующий:

extrn имя:тип, имя:тип

public имя,...,имя

Здесь **имя** — идентификатор, определенный в другом модуле. В качестве идентификатора могут выступать:

- имена переменных, определенных директивами типа **DB**, **DW** и т. д.;
- имена процедур;
- имена констант, определенных операторами = и **EQU**.

Аргумент тип определяет тип идентификатора. Указание типа необходимо для того, чтобы транслятор правильно сформировал соответствующую машинную команду. Действительные адреса вычисляются на этапе редактирования, когда будут разрешаться внешние ссылки.

Возможные значения типа определяются допустимыми типами объектов для этих директив:

- имя переменной - тип может принимать значения **DB**, **DW** и т. д.;
- имя процедуры - тип может принимать значения **near** или **far**;
- имя константы - тип должен быть **abs**.

Покажем принцип использования директив **EXTRN** и **PUBLIC** на схеме связи модулей 1 и 2.

```
;Модуль 1
masm
.model small
.stack 256
.data
.code
my_proc_1 proc
my_proc_1 endp
my_proc_2 proc
my_proc_2 endp
public my_proc_1 ;объявляем процедуру my_proc_1 видимой извне

start:
mov ax,@data
end start
```

```
;Модуль 2
masm
.model small
.stack 256
.data
. code
extrn my_proc_1 ;объявляем процедуру roy_proc_1 внешней
start:
mov ax,@data
call my_proc_1 ;вызов my_proc_1 из модуля 1
end start
```

Если в процедуре, описанном в одном сегменте есть описание переменных, но в главной программе, где осуществляется вызов процедуры, можно использовать и другие имена этих элементов данных.

Например:

```
добавление элементов массива:
DATA SEGMENT COMMON
ARY DW 100 DUP(?)
COUNT DW ?
SUM DW 0
DATA ENDS
```

Например:

NUM вместо ARY,
N вместо COUNT и
TOTAL вместо SUM.

Для этого этот сегмент нужно определить так:

```
DATA SEGMENT COMMON
NUM DW 100 DUP(?)
N DW ?
TOTAL DW 0
DATA ENDS
```

Во время редактирования связей сегменты ***DATA*** совмещаются и первые 100 слов ***DATA*** будут иметь в разных сегментах одинаковые адреса.