Parameter Passing

- Most subroutines require parameters
- Can sometimes pass parameters via registers

Assume subroutine 'line' will compute the value:

y = m*x + b

where m,x,b are signed byte values, and 'y' is a 16 bit value.

Could pass the values in registers:

al = m, dl = x, cl = b

Procedure 'line'

```
; computes y = m*x + b
; al=m, dl = x,  cl = b
; result returns in ax
proc        line
            imul        dl    ;ax = al*dl
            mov         dx,ax  ; save ax
            mov         al,cl
            cbw               ; ax=b sign extended
            add         ax,dx
            ret               ;return ax=y
```


Calling procedure *line*

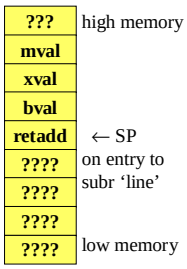
```
            .data
mval        db ?
xval        db ?
bval        db ?
sum         dw ?

            .code
            mov   ax, @data
            mov   dx, ax
            mov   al, byte ptr [mval]
            mov   dl, byte ptr [xval]
            mov   cl, byte ptr [bval]
            call  line
            mov   [sum],ax
```

Passing Parameters on Stack

Often need to pass parameters on the stack because registers cannot hold all of the parameters. Pass m,x,b on stack, return Y in ax.

```
.code
mov   ax, @data
mov   ds, ax
xor   ah,ah
mov   al, byte ptr [mval]
push  ax
mov   al, byte ptr [xval]
push  ax
mov   al, byte ptr [bval]
push  ax
call  line
mov   [sum],ax
```



Procedure 'line' with parameters on stack

```
; computes y = m*x + b
; al=m, dl = x,  cl = b
; result returns in ax
line  proc
        mov bp,sp
        mov ax,[bp+6]     ; get oper_m
        mov cx,[bp+4]     ; get oper_x
        imul cl           ; ax = al * cl = a * x
        mov dx,ax         ; save ax in dx
        mov ax,[bp+2]     ; get oper_b
        cbw              ; sign extend al to 16 bits
        add ax,dx         ; ax = a*x + b
        ret 6           ; return and inc SP by 6
line   endp
```


Some notes on 'line' example

- Cannot use register 'sp' as an address register
  - mov ax, [sp+2] is illegal
  - must use 'bp' (base pointer), transfer sp to bp first
  - default segment for 'bp' is stack segment
- It is usually the subroutine's job to clean up the stack of passed parameters
  - the code 'ret 6' will pop off the return address, then increment the stack pointer by 6 to clean up stack.
- The calling code could also clean up the stack if desired:

  call line
  add sp, 6 ; clean up stack

  In this case, the subroutine would just use the 'ret' instruction with no arguments.

Saving the BP value

Note that BP offsets increased by 2

```
; computes y = m*x + b
; al=m, dl = x,  cl = b
; result returns in ax
line  proc
        push bp           ;save BP value
        mov bp,sp
        mov ax,[bp+8]     ; get oper_m
        mov cx,[bp+6]     ; get oper_x
        imul cl           ; ax = al * cl = a * x
        mov dx,ax         ; save ax in dx
        mov ax,[bp+4]     ; get oper_b
        cbw              ; sign extend al to 16 bits
        add ax,dx        ; ax = a*x + b
        pop bp           ; restore BP

        ret 6           ; return and inc SP by 6

line   endp
```


Stack Picture with BP

???	
mval	bp + 8
xval	bp + 6
bval	bp +4
retadd	
old BP	← new BP = SP
????	after push of BP, mov bp,sp
????	
????	

Local Memory Storage on Stack

What if the subroutine needed some temporary memory locations? Could allocate them on the stack!!! Assume we need 2 words of memory.

???	
mval	bp + 8
xval	bp + 6
bval	bp +4
retadd	
old BP	← new BP
local A	
local B	← SP
????	

```
suba proc
     push bp
     mov bp,sp
     sub sp,4
     ...
     ...
     mov sp,bp
     pop bp
     ret 6
```

A Stack Frame

A generalized picture of a *stack frame* used by a subroutine is shown below:

any number of words

return address

← new BP

any number of words

← SP

parameters to subroutine available at [bp+4], [bp+6], etc from BP

stack memory storage for subroutine available at [bp–2] from Basepointer

BP also known as *Frame pointer*

Code for Stack Frame in the Subroutine

```
suba proc
     push bp      <-- Save old basepointer
     mov bp,sp    <-- Get new basepointer
     sub sp, 8    <-- allocate local storage if needed
     ...
     ...
     ...
     mov sp,bp    <-- Restore SP, free local storage
     pop bp       <-- restore old basepointer
     ret 6        <-- pop return address and clean stack of passed parameters
```


Enter/Leave instructions (+286)

Enter creates stack frame, *leave* removes stack frame.
1st parameter to *Enter* is number of bytes of local storage.
We will ignore 2nd parameter, and always leave as 0.

```
suba proc
     push bp
     mov bp,sp
     sub sp, 8
     ...
     ...
     ...
     mov sp,bp
     pop bp
     ret 6
```

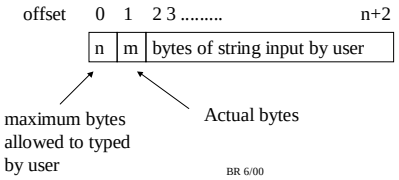
```
suba proc
     enter 8,0


     ...
     ...
     ...
     leave

     ret 6
```

mecho example

- Write a subroutine that gets an input string from the user keyboard, string stored on stack
- Subroutine uses the Dos service call ah= 0ah, int 21h which requires DX to point to some buffer space (memory area)



Comments on *mecho* example

- When calling the DOS int 21h, ah=0ah function, we need to make sure that the DATA SEGMENT is the same as the Stack segment since the data buffer is on the stack (and the DOS function assumes that the data buffer is pointed to by data segment
- Must be sure to restore the data segment to original value before returning from subroutine.