III. STRUCTURED ASSEMBLY LANGUAGE PROGRAMMING TECHNIQUES

Modular Programming

Modular Programming

- smaller program
  - own set of variables (scope)
  - called by another subprogram
  - returns to calling subprogram after it executes
- procedures
  - no return value
- functions
  - returns value(s)

Modular Programming

- **call** *label*
  - **push** PC
  - **jmp** *label*

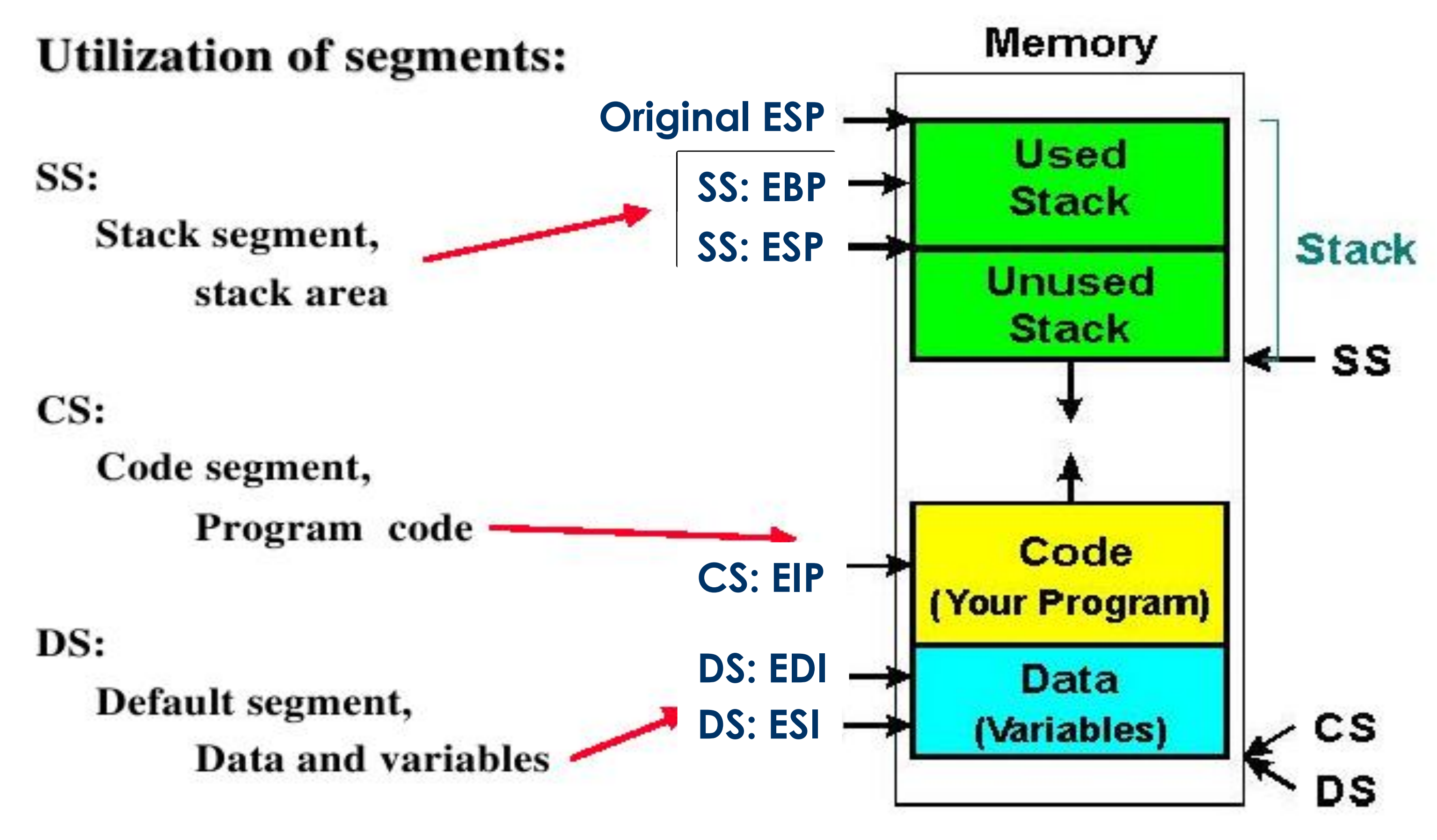
- PC is saved to allow the computer to return to the calling subprogram

Modular Programming

- **ret** *source*
  - **pop** PC
  - **add** SP, source

- PC is restored.
- Space used in stack by parameters are removed.
- Source is an immediate operand.

The Stack



Utilization of segments:

SS:
Stack segment,
stack area

CS:
Code segment,
Program code

DS:
Default segment,
Data and variables

The Stack

- Data Structure
  - insert and delete an element from a single point: the top of stack
  - push
    - insert element
  - pop
    - retrieve element

The Program Stack

- Stack Segment
  - SS: stack segment
    - Starting address of stack
  - ESP: stack pointer
    - Top of Stack pointer

- Machine Instructions
  - **push** *source*
  - **pop** *destination*

- We can only insert to and retrieve from the stack 16-bit and 32-bit values.

- The instruction operands can be registers or memory operands.

Example

subprogram:

```
void sample ()
{
    // body
}
```

subprogram call:

```
sample();
```

Example

subprogram:

```
void sample ()
{
    // body
}
```

subprogram call:

```
sample();
```

subprogram:

```
sample:
      ; body
      ret
```

subprogram call:

```
call sample
```

Parameter Passing

- Call by Value
  - only the value of the parameter is passed on to the subprogram

- Call by Reference (Variable Parameters)
  - used when the parameter is to be changed within the called subprogram
  - the address of the parameter is called

Value Parameters

subprogram:

```
void sum (int a, int b)
{
      int num;
      num = a + b;
}
```

subprogram call:

```
sum(x, y);
```

Value Parameters

subprogram:

```
void sum (int a, int b)
{
    int num;
    num = a + b;
}
```

subprogram call:

```
sum(x, y);
```

(Assembly)

subprogram call:

```
push    word [x]
```

Value Parameters

subprogram:

```
void sum (int a, int b)
{
    int num;
    num = a + b;
}
```

subprogram call:

```
sum(x, y);
```

(Assembly)

subprogram call:

```
push    word [x]
push    word [y]
```

Value Parameters

subprogram:

```
void sum (int a, int b)
{
    int num;
    num = a + b;
}
```

subprogram call:

```
sum(x, y);
```

(Assembly)

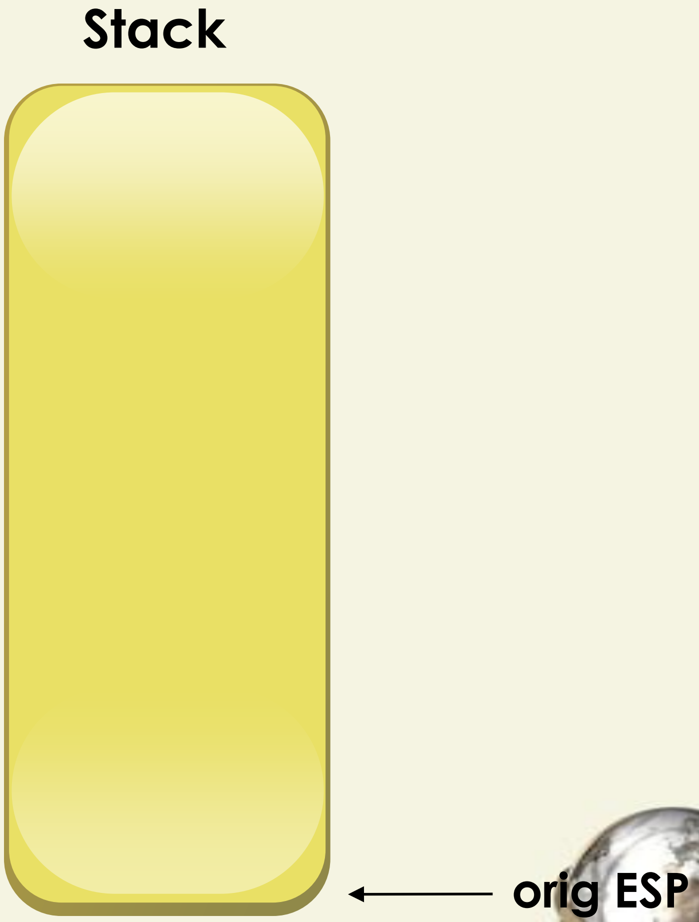
subprogram call:

```
push    word [x]
push    word [y]
call sum
```

Stack

subprogram call:

```
push     word [x]
push     word [y]
call   sum
```



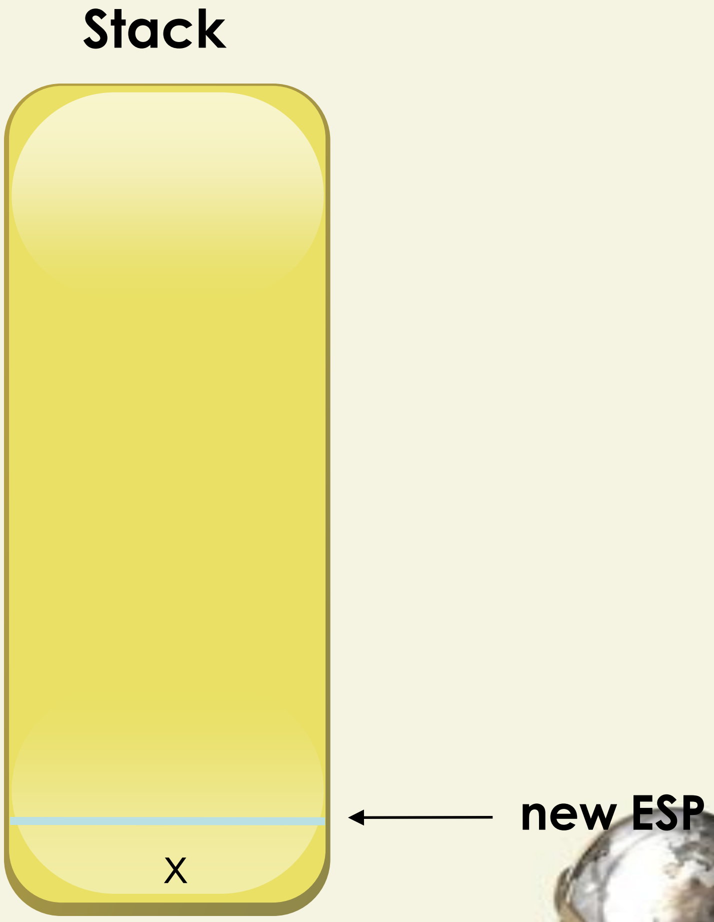
Stack

orig ESP

Stack

subprogram call:

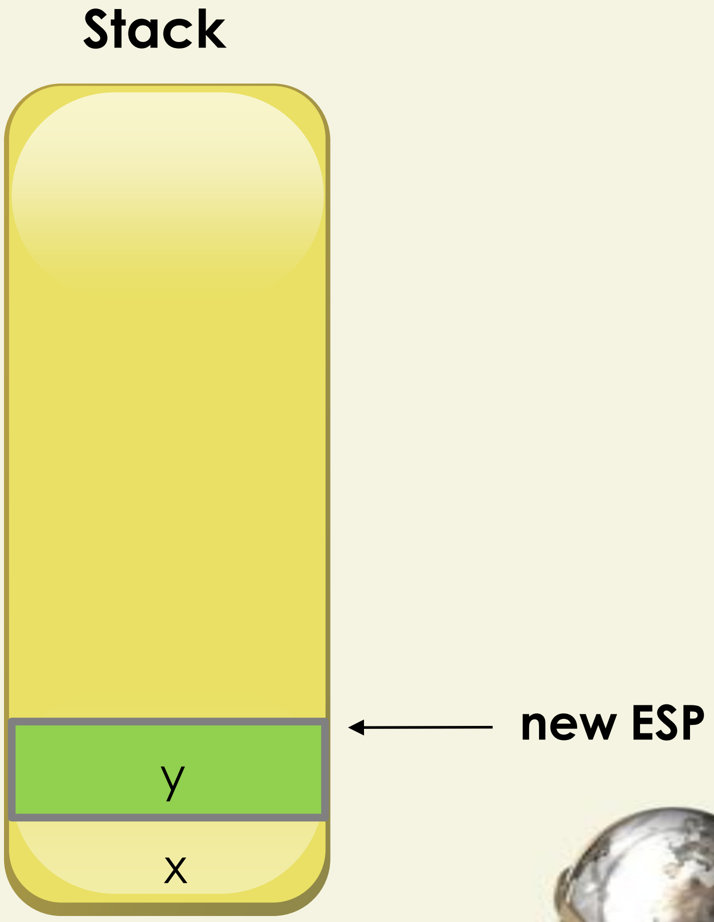
```
push     word [x]
push     word [y]
call   sum
```



Stack

subprogram call:

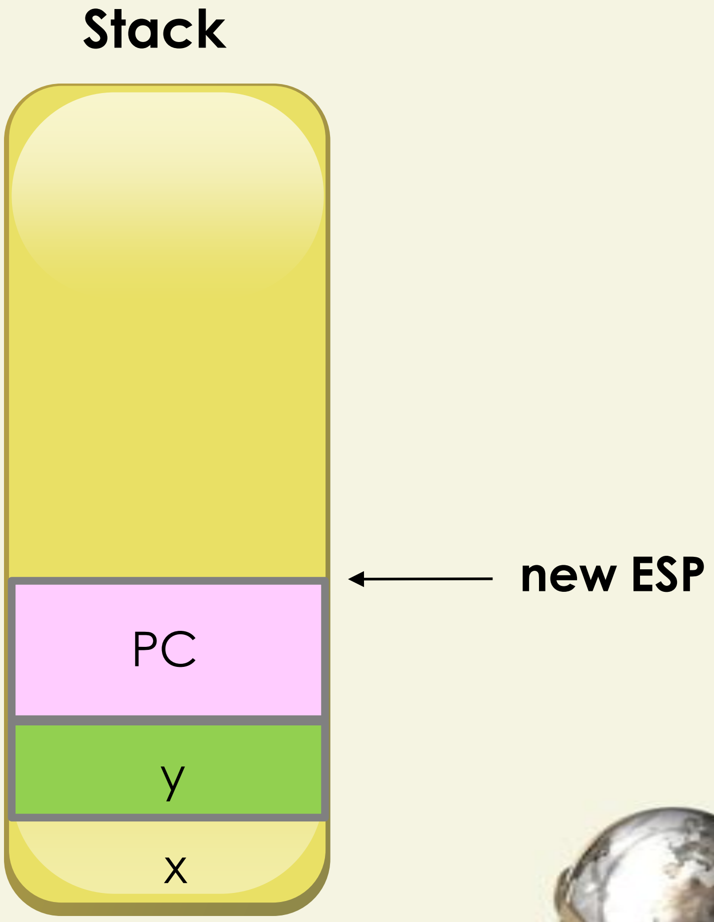
```
push     word [x]
push     word [y]
call   sum
```



Stack

subprogram call:

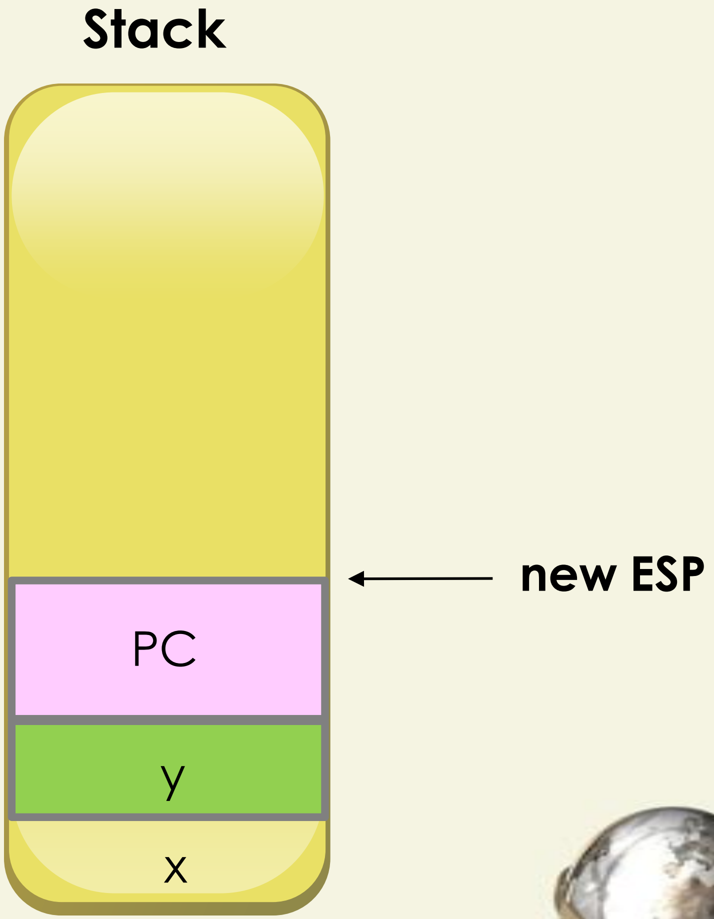
```
push	word [x]
push	word [y]
call	sum
```



Value Parameters

subprogram:

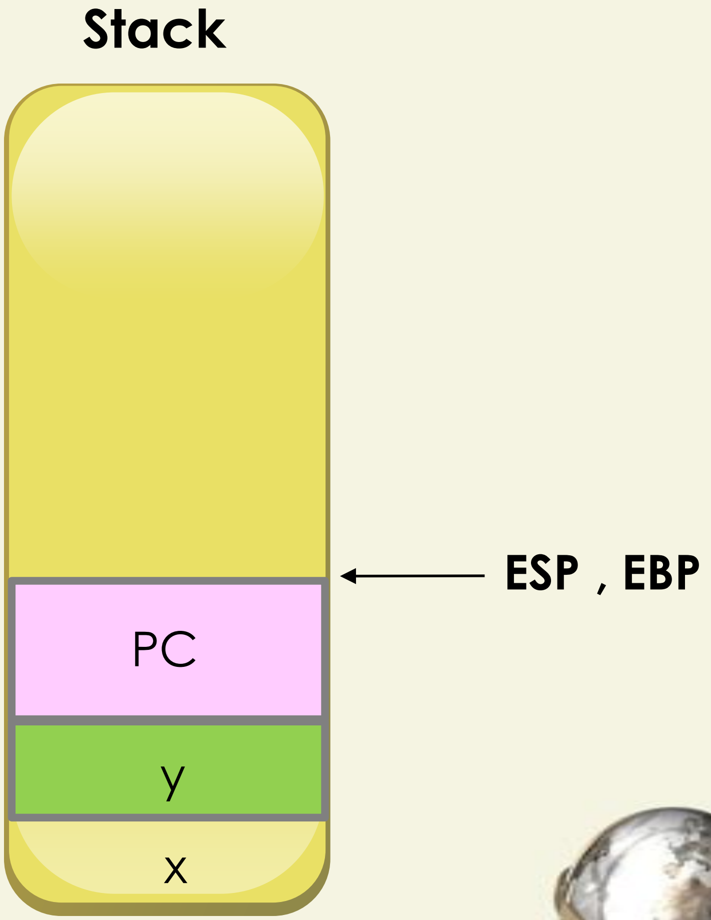
```
void sum (int a, int b)
{
    int num;
        num = a + b;
}
```



Value Parameters

subprogram:

```
void sum (int a, int b)
{
    int num;
    num = a + b;
}

sum:
   mov  ebp, esp
```



Stack

ESP , EBP

PC

y

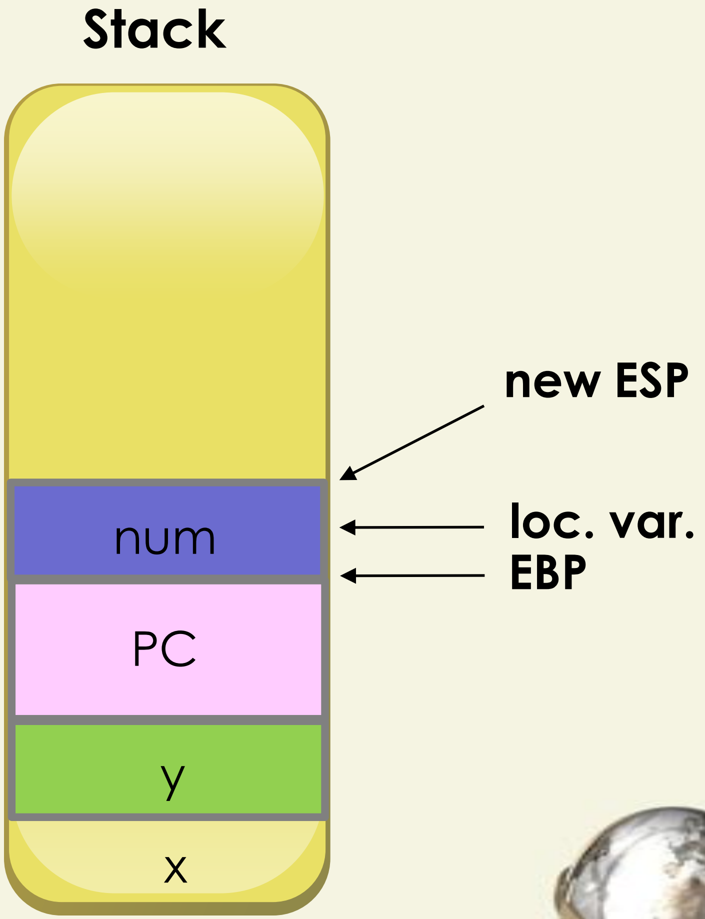
x

Value Parameters

subprogram:

```
void sum (int a, int b)
{
     int num;
          num = a + b;
}
```

```
sum:
    mov  ebp, esp
    sub   esp, 2
```

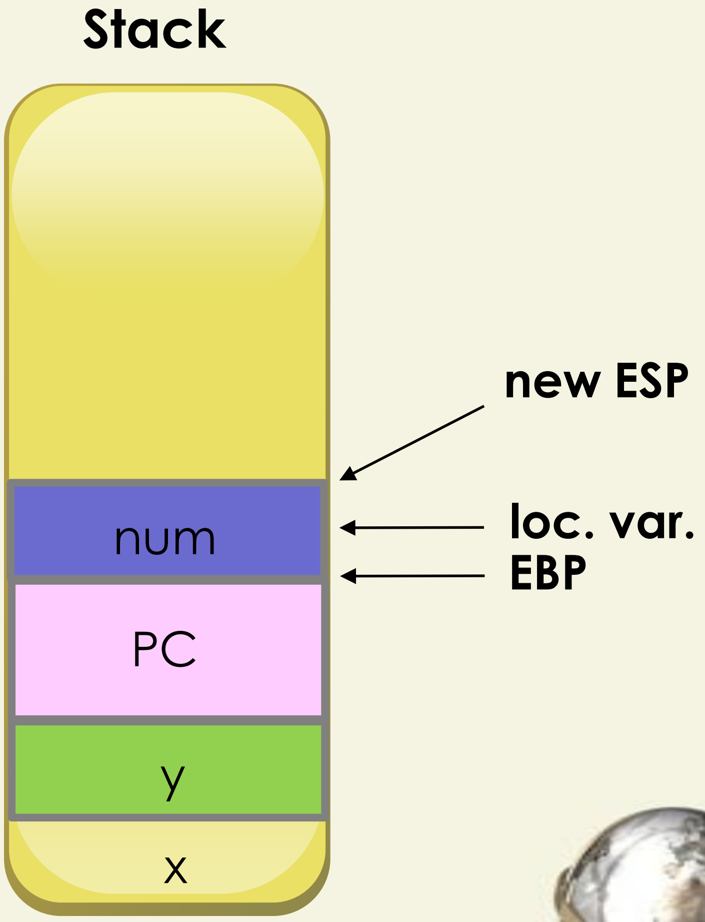


Value Parameters

subprogram:

```
void sum (int a, int b)
{
      int num;
      num = a + b;
}
```

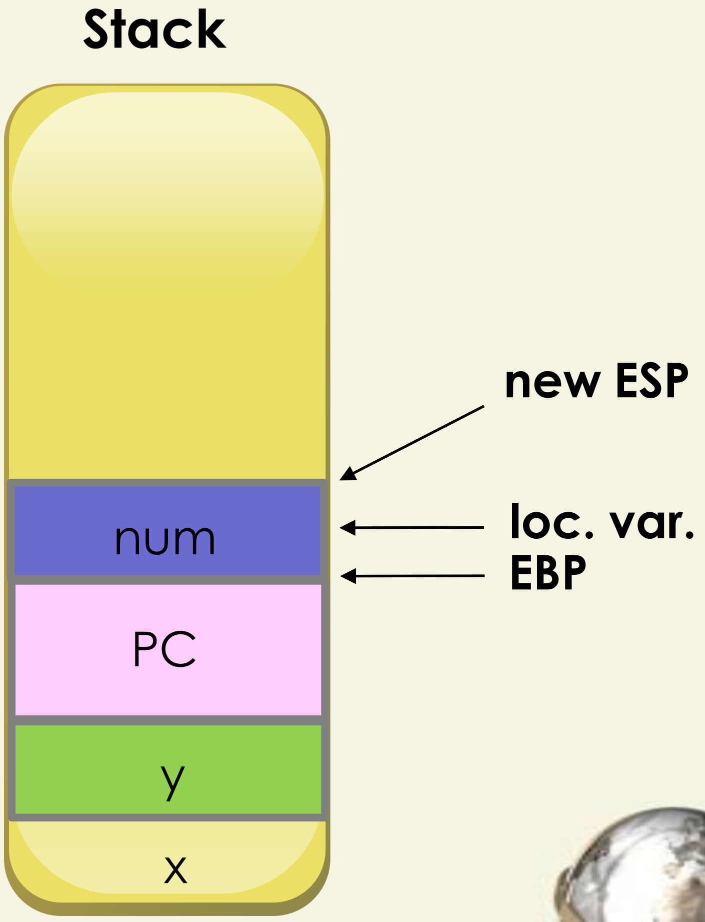
```
sum:
    mov  ebp, esp
    sub  esp, 2
    mov  ax, [ebp + 6]
```



Value Parameters

subprogram:

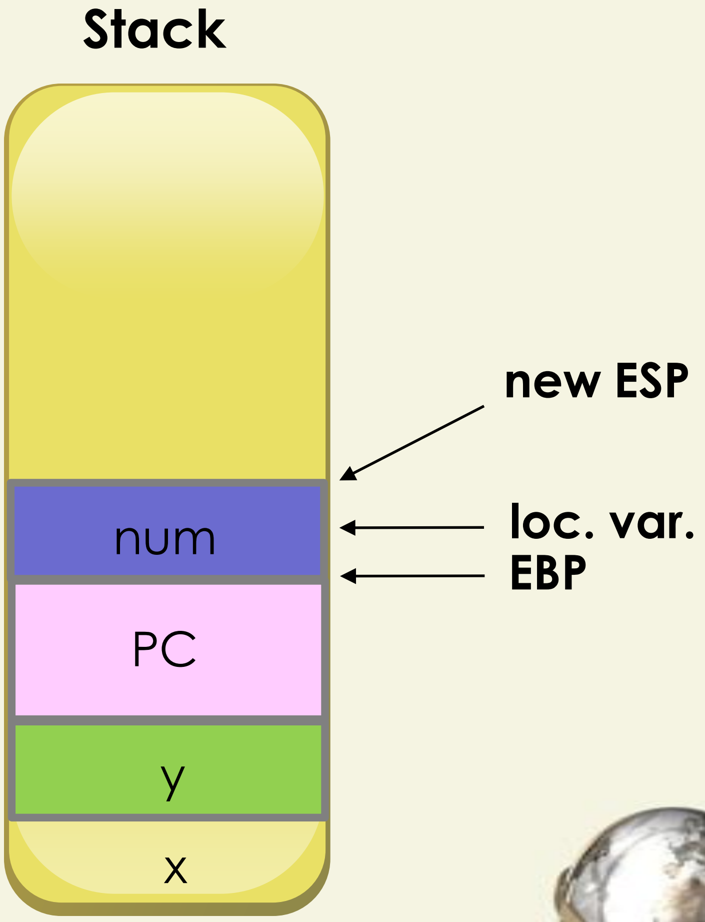
```
void sum (int a, int b)
{
     int num;
     num = a + b;
}

sum:
    mov  ebp, esp
    sub   esp, 2
    mov  ax, [ebp + 6]
    add   ax, [ebp + 4]
```



Value Parameters

subprogram:

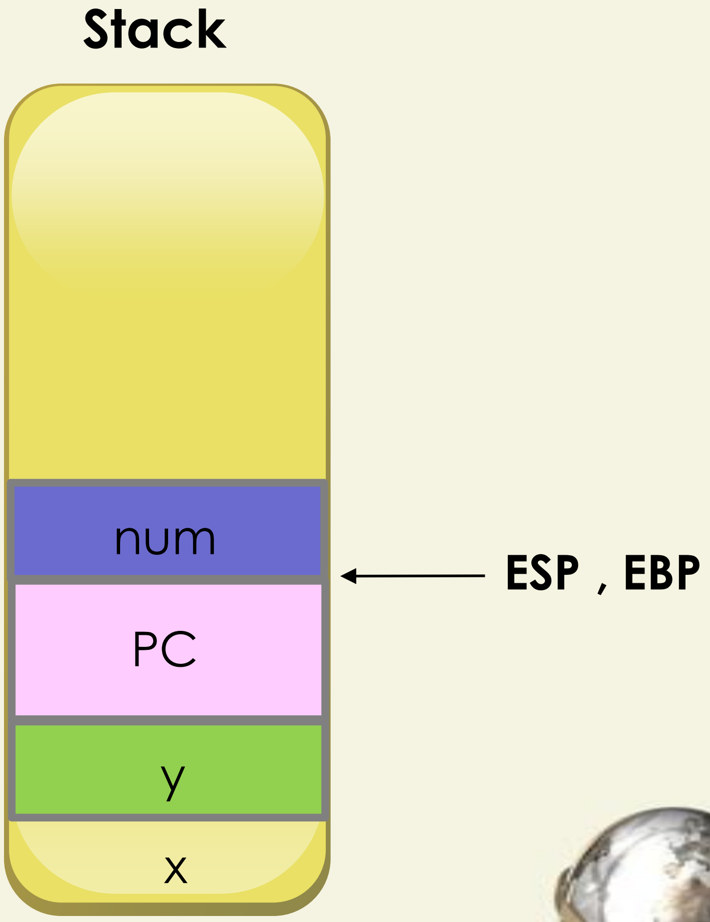
```
void sum (int a, int b)
{
      int num;
      num = a + b;
}

sum:
    mov  ebp, esp
    sub   esp, 2
    mov  ax, [ebp + 6]
    add   ax, [ebp + 4]
    mov  [ebp – 2], ax
```



Value Parameters

subprogram:

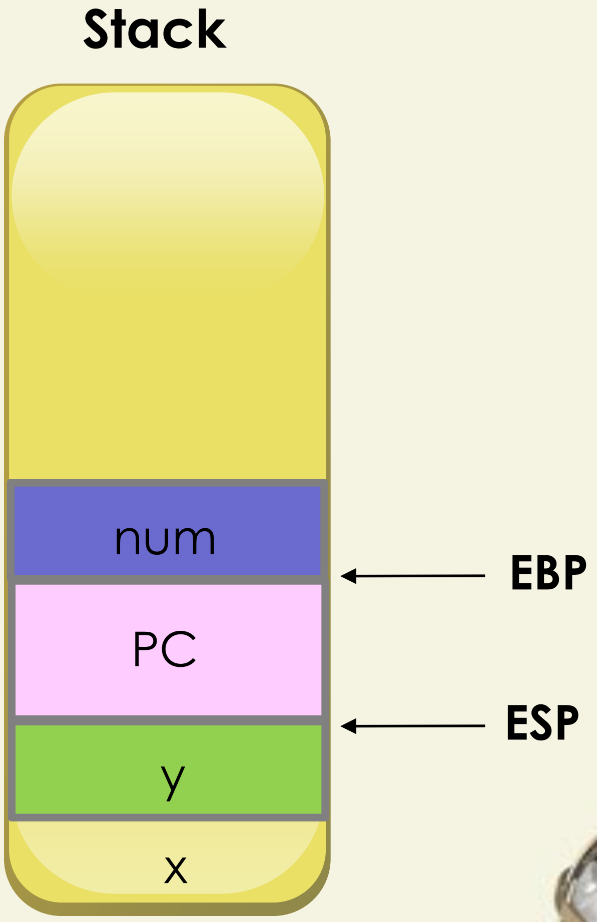
```
void sum (int a, int b){
      int num;
      num = a + b;
}
sum:
    mov  ebp, esp
    sub   esp, 2
    mov  ax, [ebp + 6]
    add   ax, [ebp + 4]
    mov  [ebp – 2], ax
    add   esp, 2
```



Value Parameters

subprogram:

```
void sum (int a, int b){
      int num;
      num = a + b;
}
sum:
    mov  ebp, esp
    sub   esp, 2
    mov  ax, [ebp + 6]
    add   ax, [ebp + 4]
    mov  [ebp – 2], ax
    add   esp, 2
    ret     4
```



Stack

num

PC

y

x

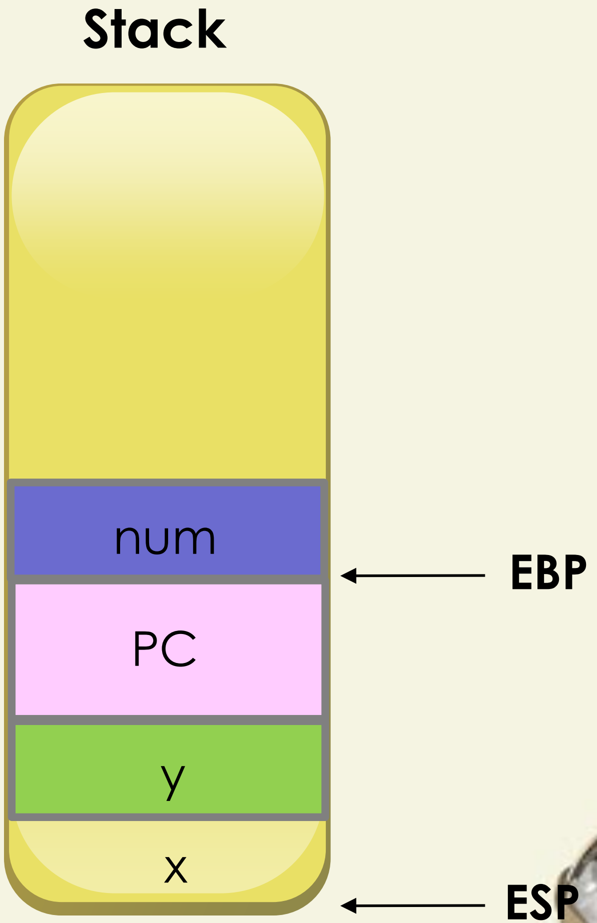
EBP

ESP

Value Parameters

subprogram:

```
void sum (int a, int b){
      int num;
      num = a + b;
}
sum:
    mov  ebp, esp
    sub  esp, 2
    mov  ax, [ebp + 6]
    add  ax, [ebp + 4]
    mov  [ebp – 2], ax
    add  esp, 2
    ret  4
```



Value Parameters

```
sum:
    mov  ebp, esp           ; create stack frame
    sub   esp, 2            ; reserve local variable
    mov  ax, [ebp + 6]      ; retrieve parameter a
    add  ax, [ebp + 4]      ; retrieve parameter b
    mov  [ebp – 2], ax      ; num = a + b
    add   esp, 2            ; release local variable
    ret     4                   ; return to caller and
                                  clear stack
```

Variable Parameters

subprogram:

```
void sum
(int *n, int a, int b) {
    *n = a + b;
}
```

subprogram call:

```
sum(&num, x, y);
```

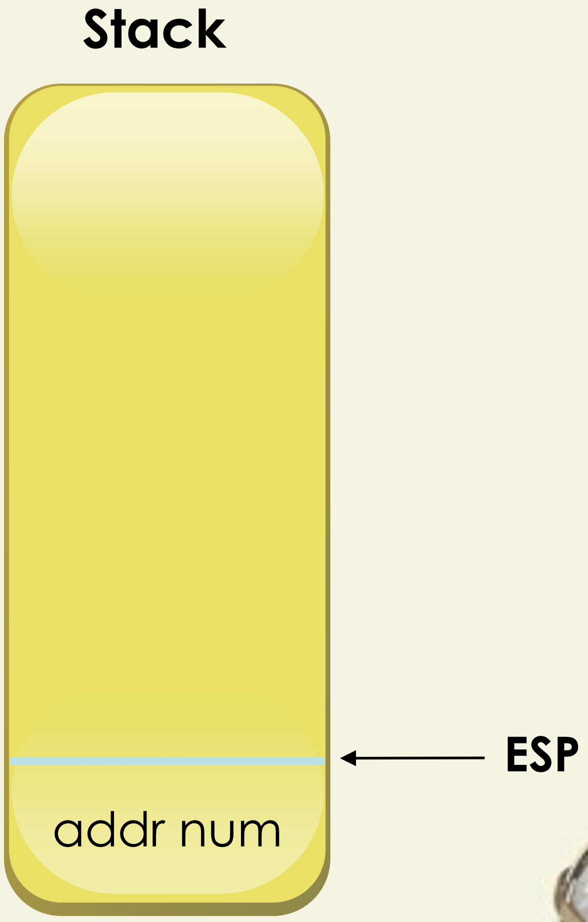
subprogram call:

```
push num
push word [x]
push word [y]
call sum
```

Stack

subprogram call:

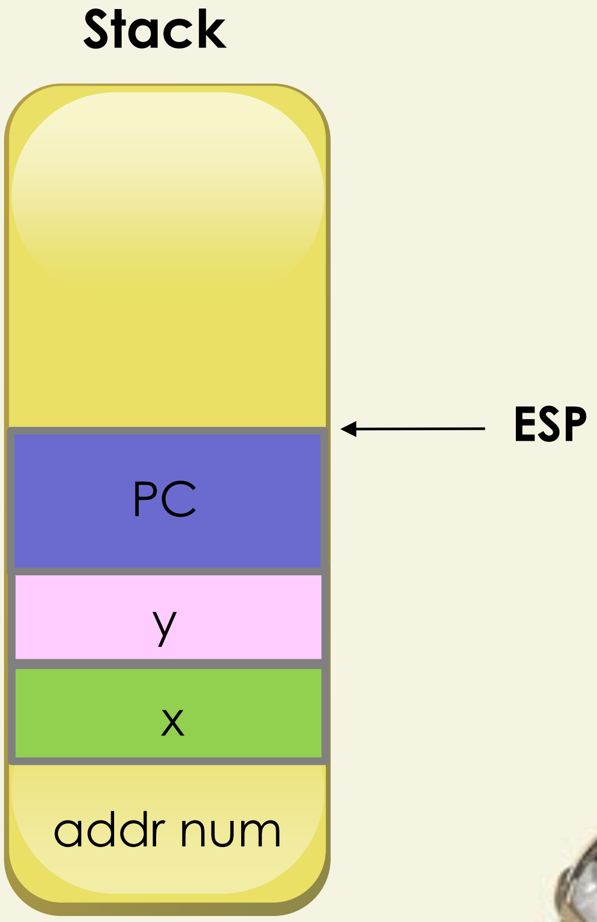
```
push num
push word [x]
push word [y]
call sum
```



Stack

subprogram call:

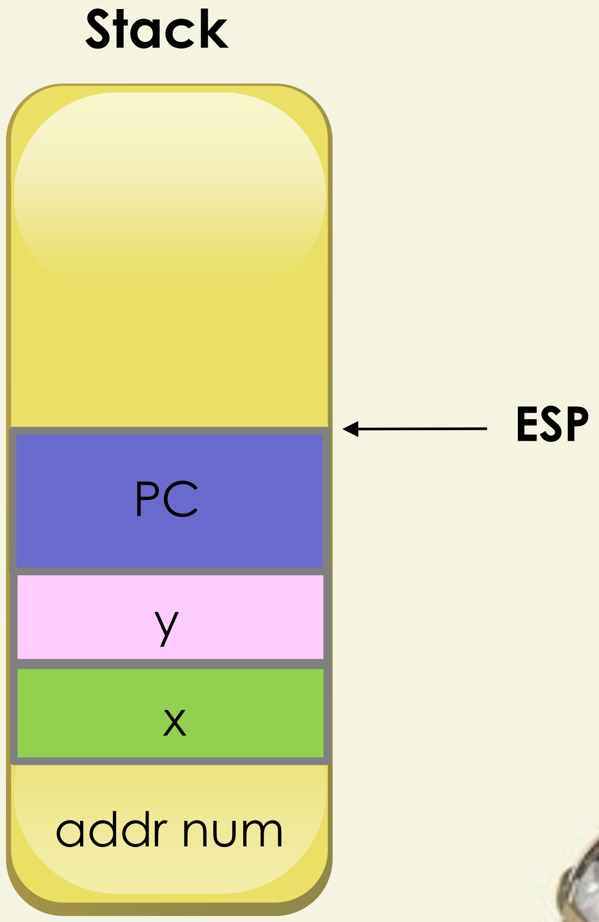
```
push num
push word [x]
push word [y]
call sum
```



Variable Parameters

subprogram:

```
void sum
(int *n, int a, int b) {
    *n = a + b;
}
```

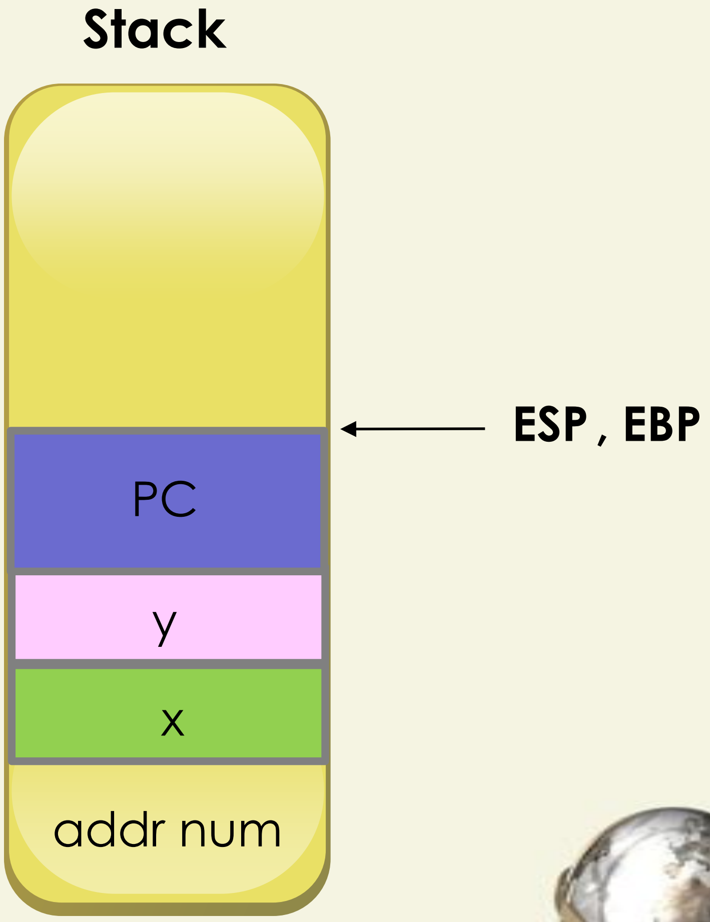


Variable Parameters - Stack

subprogram:

```
void sum(int *n, int a, int b) {
    *n = a + b;
}
```

```
sum:
    mov  ebp, esp
```

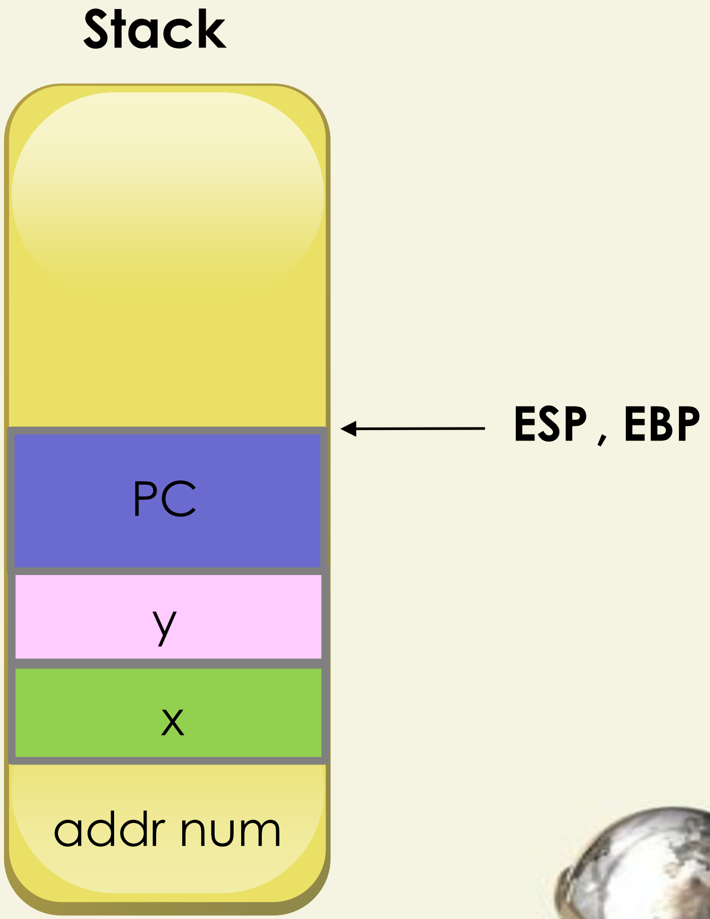


Variable Parameters - Stack

subprogram:

```
void sum(int *n, int a, int b) {
    *n = a + b;
}
```

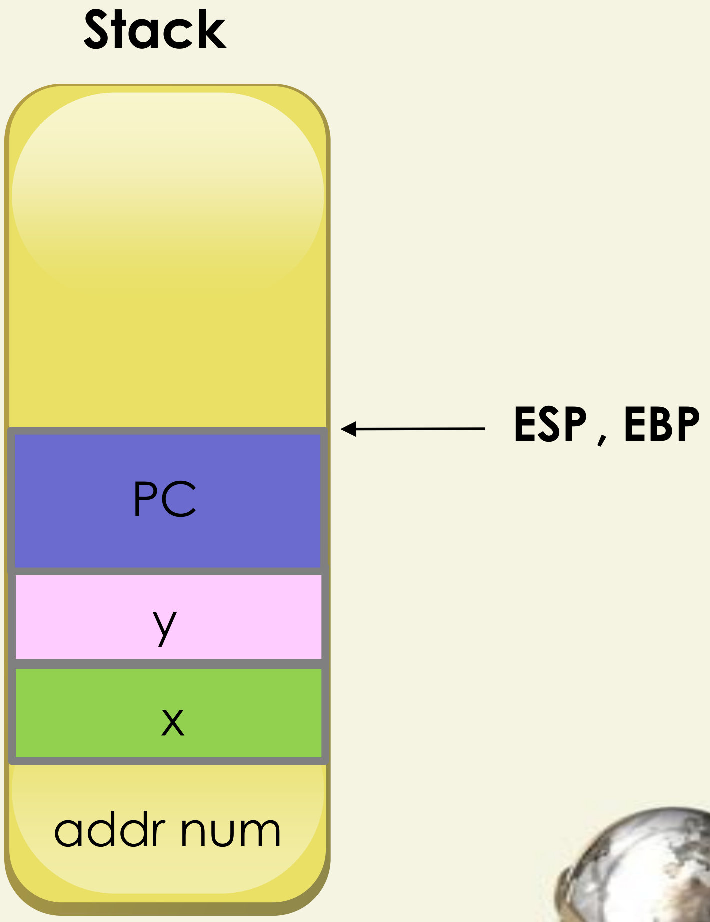
```
sum:
    mov  ebp, esp
    mov  ax, [ebp + 6]
    add  ax, [ebp + 4]
```



Variable Parameters - Stack

subprogram:

```
void sum(int *n, int a, int b) {
    *n = a + b;
}

sum:
    mov  ebp, esp
    mov  ax, [ebp + 6]
    add  ax, [ebp + 4]
    mov  ebx, [ebp + 8]
```

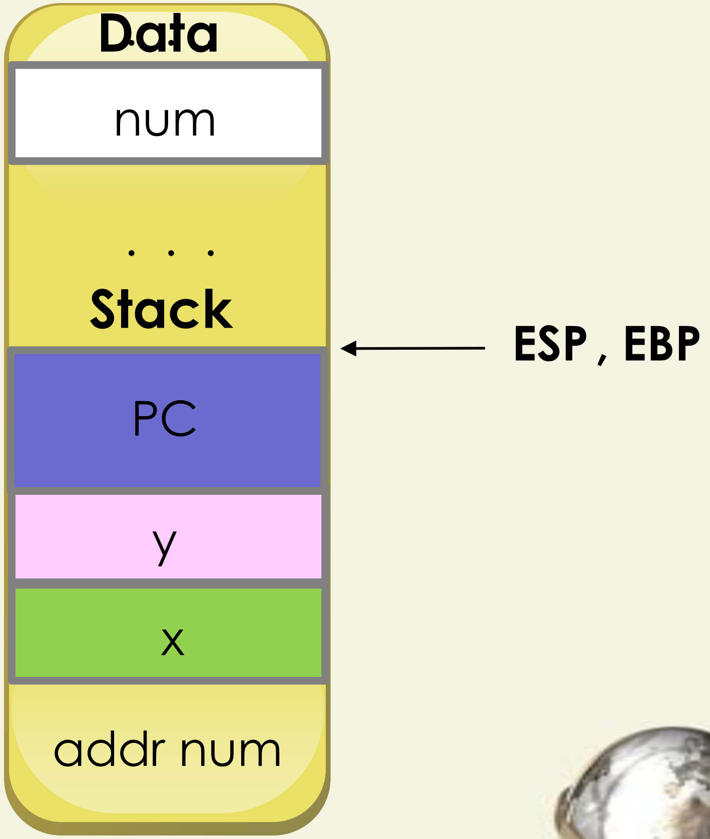


Variable Parameters - Stack

subprogram:

```
void sum(int *n, int a, int b) {
    *n = a + b;
}
```

```
sum:
    mov  ebp, esp
    mov  ax, [ebp + 6]
    add  ax, [ebp + 4]
    mov  ebx, [ebp + 8]
    mov [ebx], ax
```

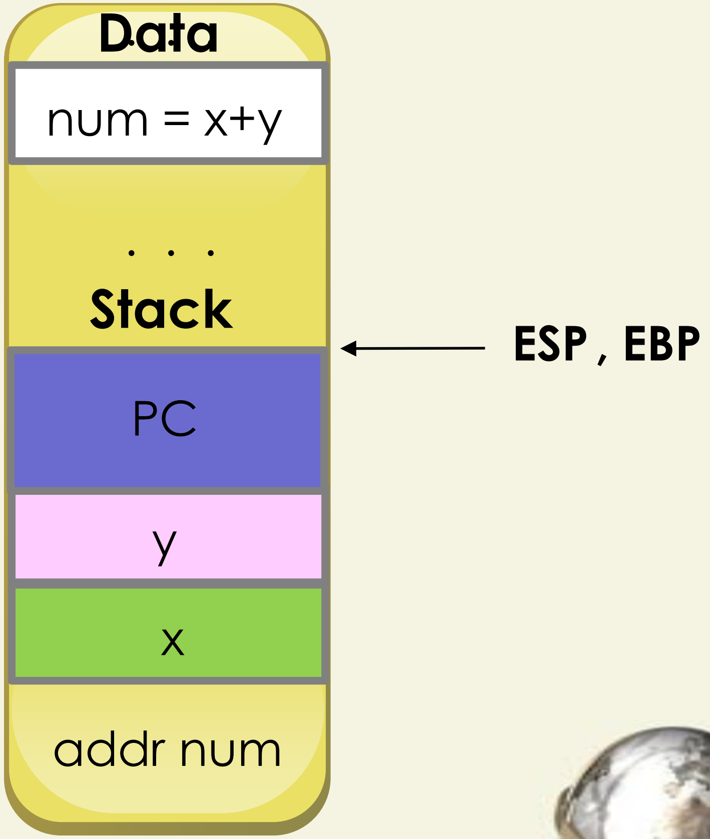


Variable Parameters - Stack

subprogram:

```
void sum(int *n, int a, int b) {
    *n = a + b;
}
```

```
sum:
    mov  ebp, esp
    mov  ax, [ebp + 6]
    add  ax, [ebp + 4]
    mov  ebx, [ebp + 8]
    mov [ebx], ax
```



Data

num = x+y

. . .

Stack

ESP , EBP

PC

y

x

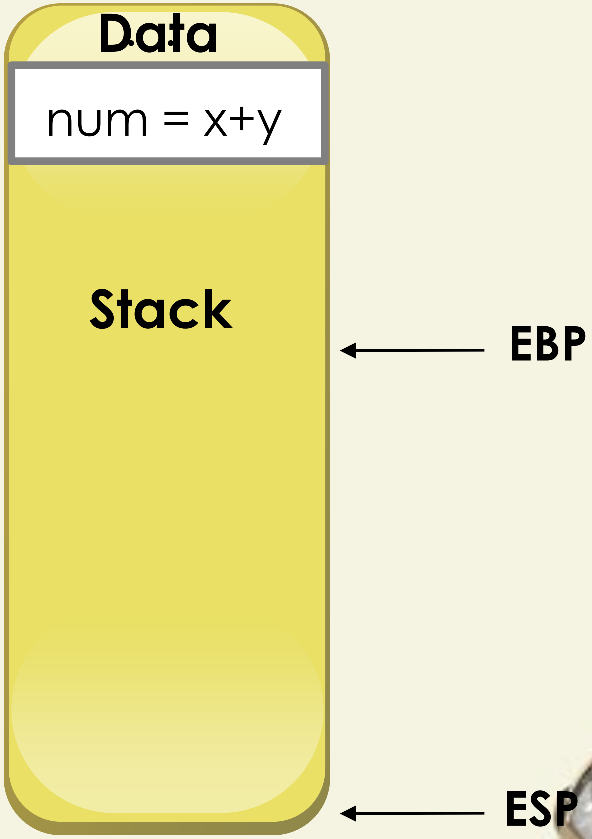
addr num

Variable Parameters - Stack

subprogram:

```
void sum(int *n, int a, int b) {
    *n = a + b;
}
```

```
sum:
    mov  ebp, esp
    mov  ax, [ebp + 6]
    add  ax, [ebp + 4]
    mov  ebx, [ebp + 8]
    mov [ebx], ax
    ret     8
```



Variable Parameters

```
sum:
   mov  ebp, esp                ; create stack frame
   mov  ax, [ebp + 6]               ; retrieve parameter a
   add  ax, [ebp + 4]               ; retrieve parameter b
   mov  ebx, [ebp + 8]          ; BX = &num
   mov [ebx], ax                ; *BX = a + b
   ret     8                    ; return to caller and
                                    clear stack
```

Functions: Return Value

High-level PL

subprogram:

```
int sum (int a, int b) {
    return(a + b);
}
```

subprogram call:

```
num = sum(x, y);
```

Functions: Return Value

High-level PL

subprogram:

```
int sum (int a, int b) {
    return(a + b);
}
```

subprogram call:

```
num = sum(x, y);
```

Assembly

; **subprogram call**

```
sub esp, 2
```

Functions: Return Value

High-level PL

subprogram:

```
int sum (int a, int b) {
    return(a + b);
}
```

subprogram call:

```
num = sum(x, y);
```

Assembly

; subprogram call

```
sub esp, 2
push word [x]
push word [y]
call    sum
```

Functions: Return Value

High-level PL

subprogram:

```
int sum (int a, int b) {
    return(a + b);
}
```

subprogram call:

```
num = sum(x, y);
```

Assembly

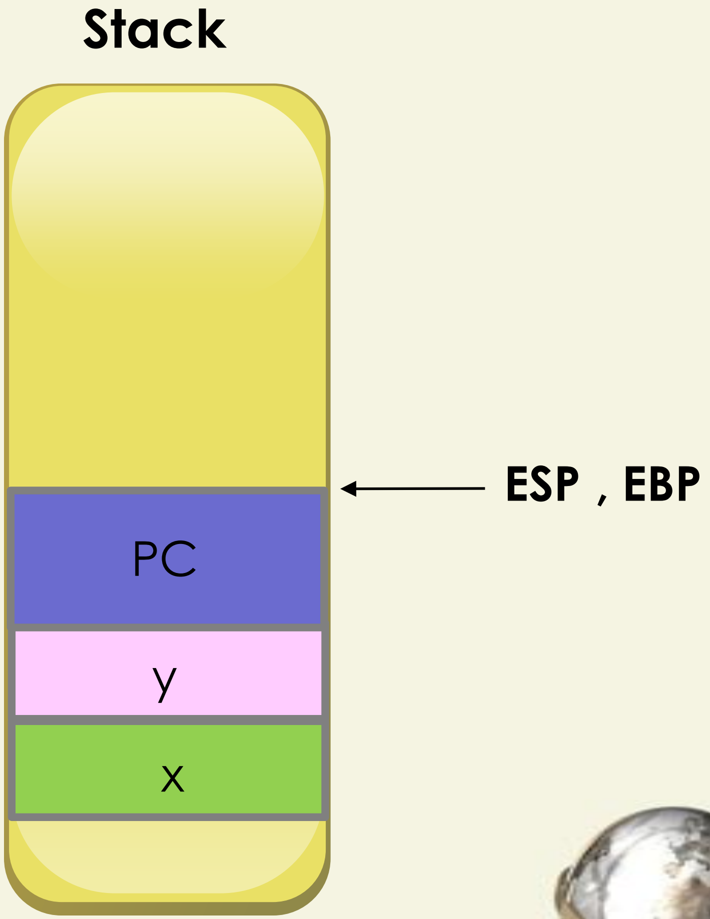
; subprogram call

```
sub esp, 2
push word [x]
push word [y]
call     sum
pop word[num]
```

Functions: Return Value

subprogram:

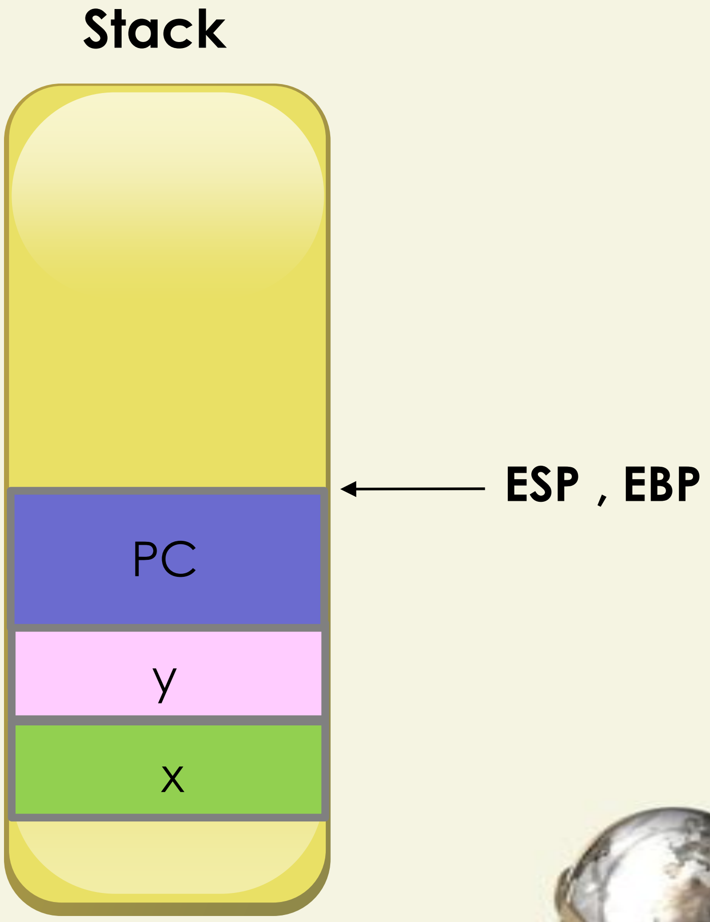
```
int sum (int a, int b) {
    return(a + b);
}

sum:
    mov  ebp, esp
```



Functions: Return Value

subprogram:

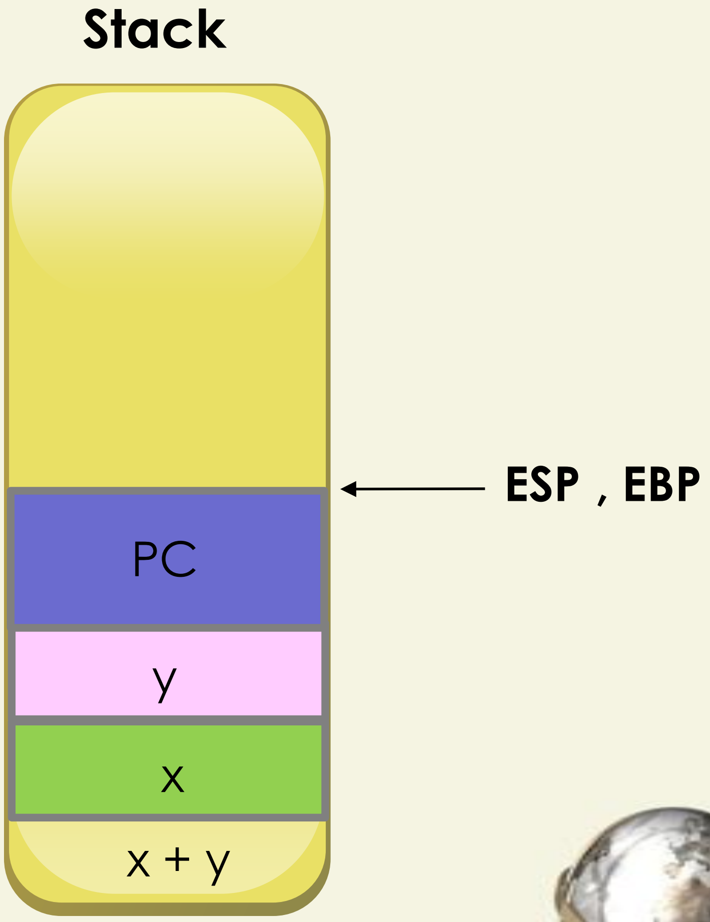
```
int sum (int a, int b) {
    return(a + b);
}

sum:
    mov  ebp, esp
    mov  ax, [ebp + 6]
    add  ax, [ebp + 4]
```



Functions: Return Value

subprogram:

```
int sum (int a, int b) {
    return(a + b);
}

sum:
    mov  ebp, esp
    mov  ax, [ebp + 6]
    add  ax, [ebp + 4]
    mov  [ebp + 8], ax
```

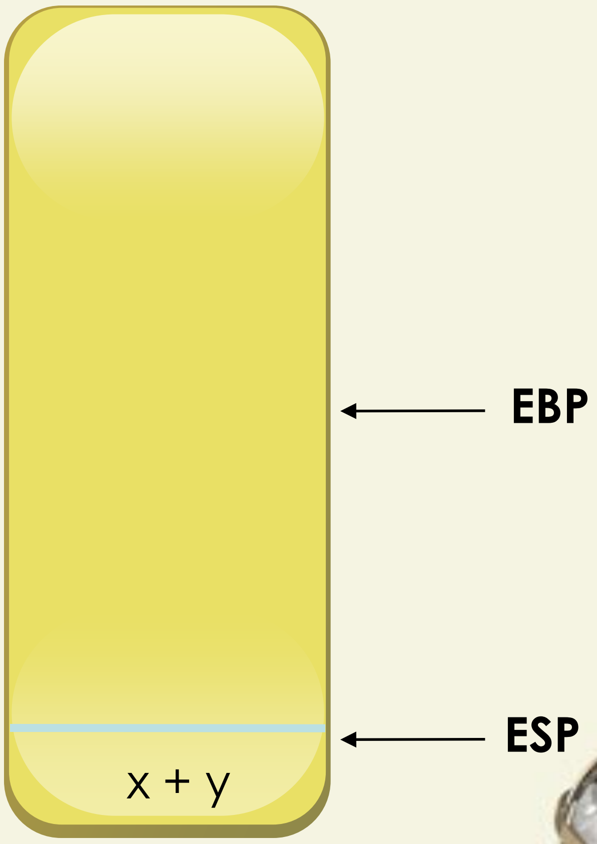


Functions: Return Value

subprogram:

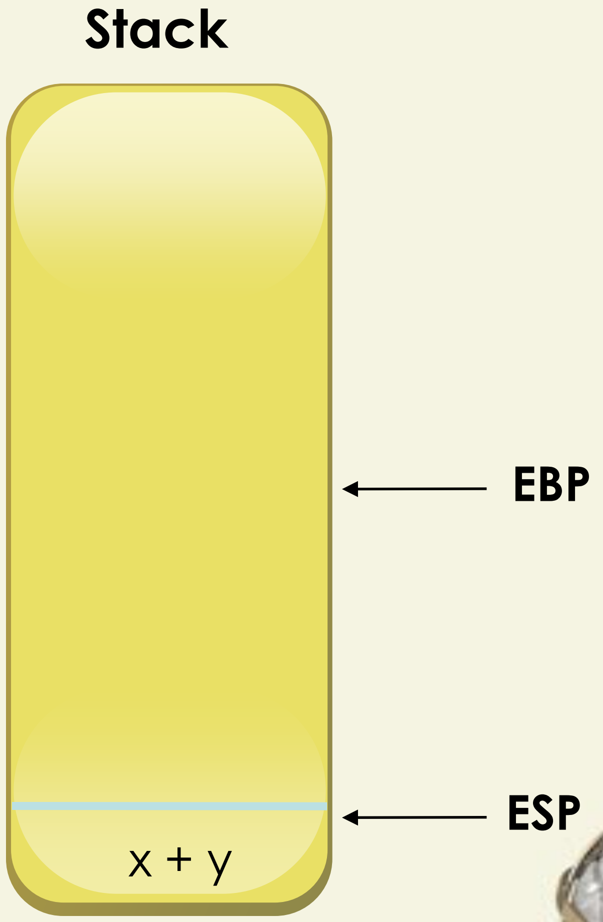
```
int sum (int a, int b) {
    return(a + b);
}

sum:
    mov  ebp, esp
    mov  ax, [ebp + 6]
    add  ax, [ebp + 4]
    mov  [ebp + 8], ax
    ret     4
```

Stack



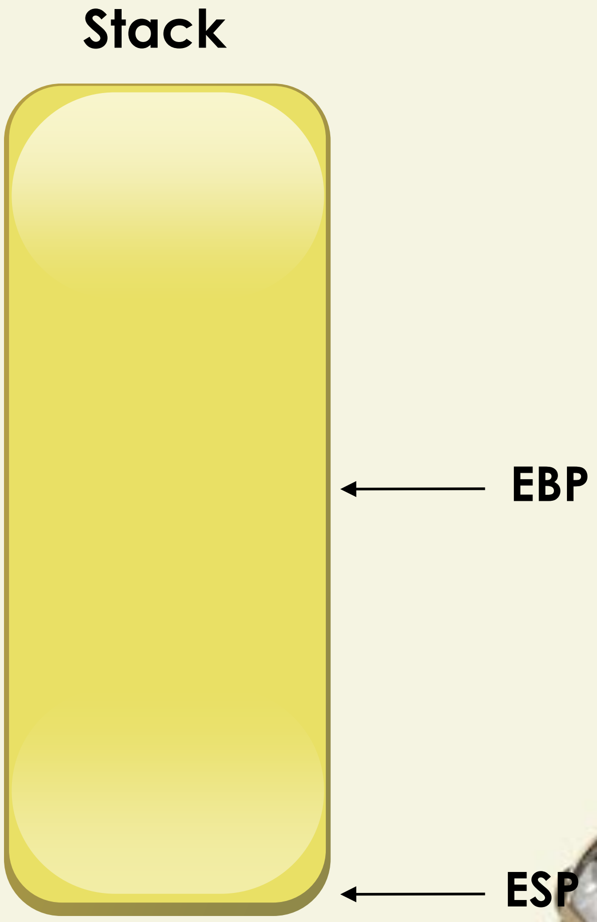
Functions: Return Value

```
; subprogram call

sub esp, 2
push word [x]
push word [y]
call   sum
pop word [num]
```



Functions: Return Value

```
; subprogram call

sub esp, 2
push word [x]
push word [y]
call   sum
pop word [num]
```



Functions: Return Value

```
sum:
    mov  ebp, esp              ; create stack frame
    mov  ax, [ebp + 6]            ; retrieve parameter a
    add   ax, [ebp + 4]           ; retrieve parameter b
    mov  [ebp + 8], ax            ; return a + b
    ret     4                  ; return to caller and
                                  clear stack
```

More Examples

```
int abc (int n) {
    int result=n;
    while(n>1) {
          n--;
          result=result*n;
    }
    return result;
}

r = abc (a);
```

More Examples

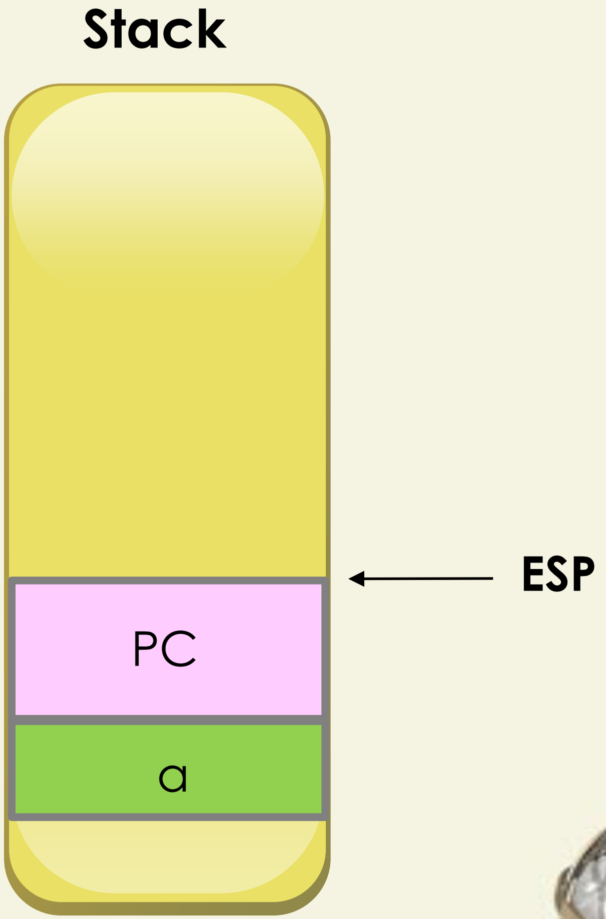
```
int abc (int n) {
    int result=n;
    while(n>1) {
          n--;
          result=result*n;
    }
    return result;
}

r = abc (a);
```

```
;subprogram call

sub esp, 2
push word [a]
call abc
pop word [r]
```

Stack

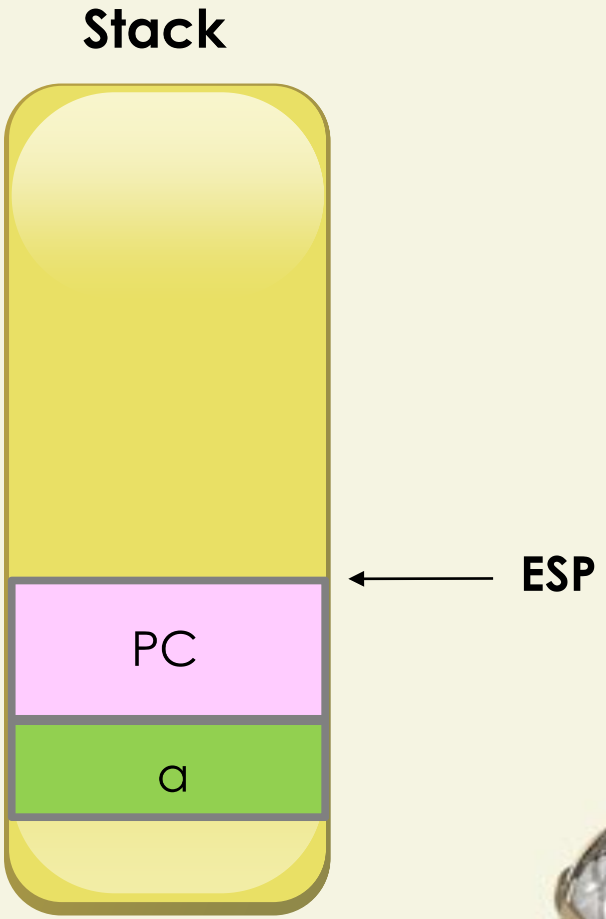
subprogram call:

```
sub esp, 2
push word [a]
call abc
pop word [r]
```



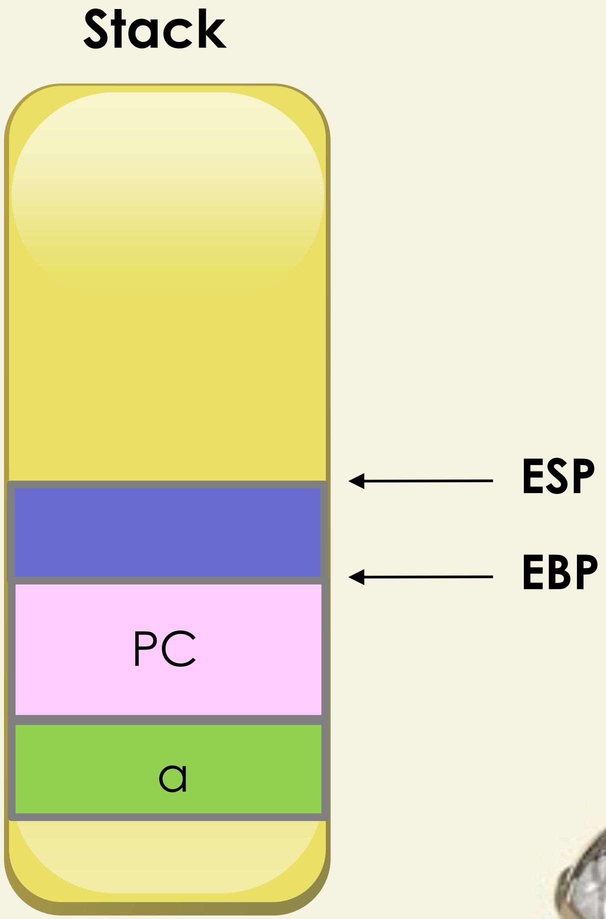
More Examples

```
int abc (int n) {
   int result=n;
   while(n>1) {
      n--;
      result=result*n;
   }
   return result;
}
```



More Examples

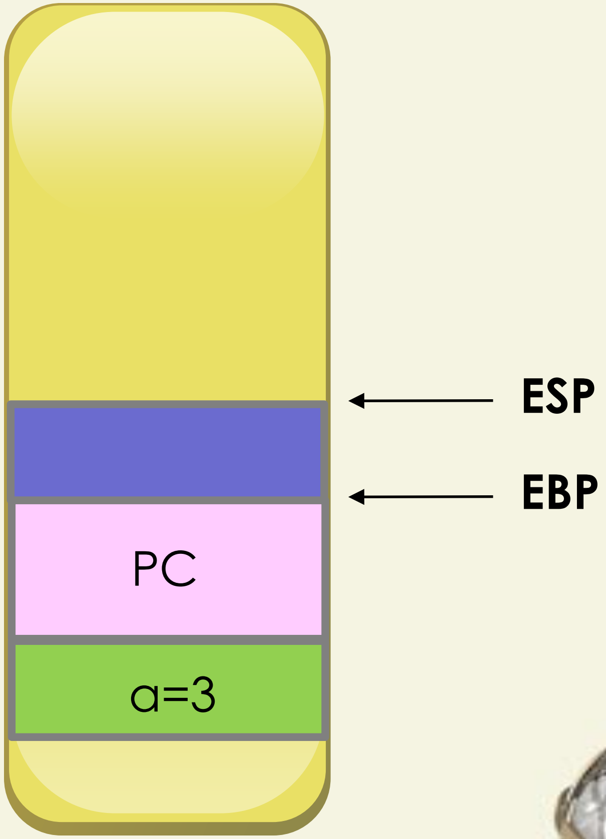
```
int abc (int n) {
    int result=n;
    while(n>1) {
          n--;
          result=result*n;
    }
    return result;
}

;subprogram
   abc:
    mov ebp, esp
    sub esp, 2
```



More Examples

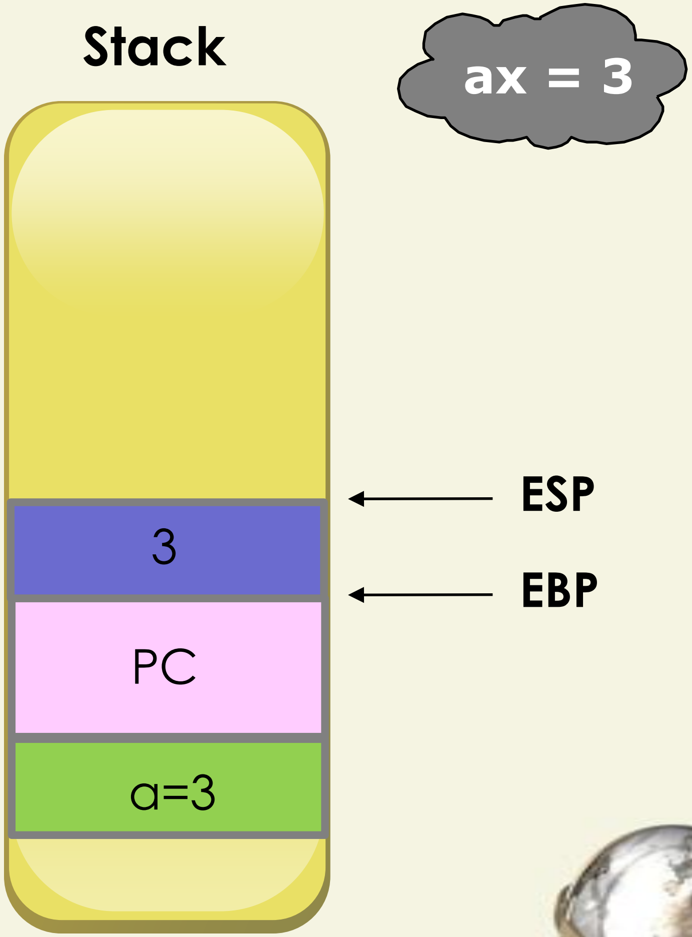
```
int abc (int n) {
     int result=n;
     while(n>1) {
            n--;
            result=result*n;
     }
     return result;
}

;subprogram
   abc:
     mov ebp, esp
     sub esp, 2
```

Stack



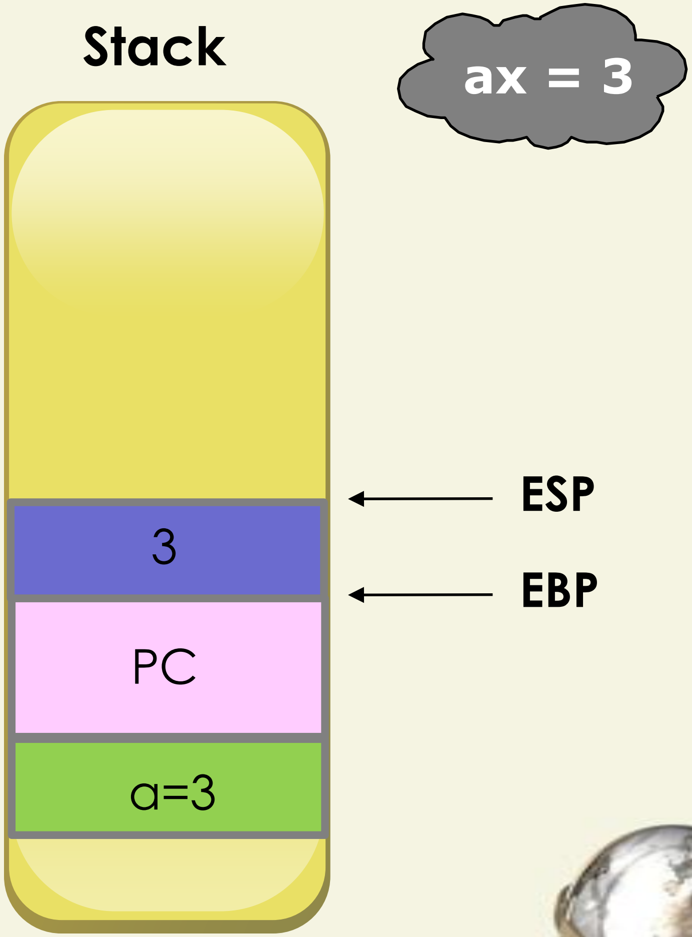
More Examples

```
int abc (int n) {
     int result=n;
     . . .
     return result;
}

;subprogram
   abc:
     mov ebp, esp
     sub esp, 2
      mov ax, [ebp+4]
      mov word[ebp-2], ax
```



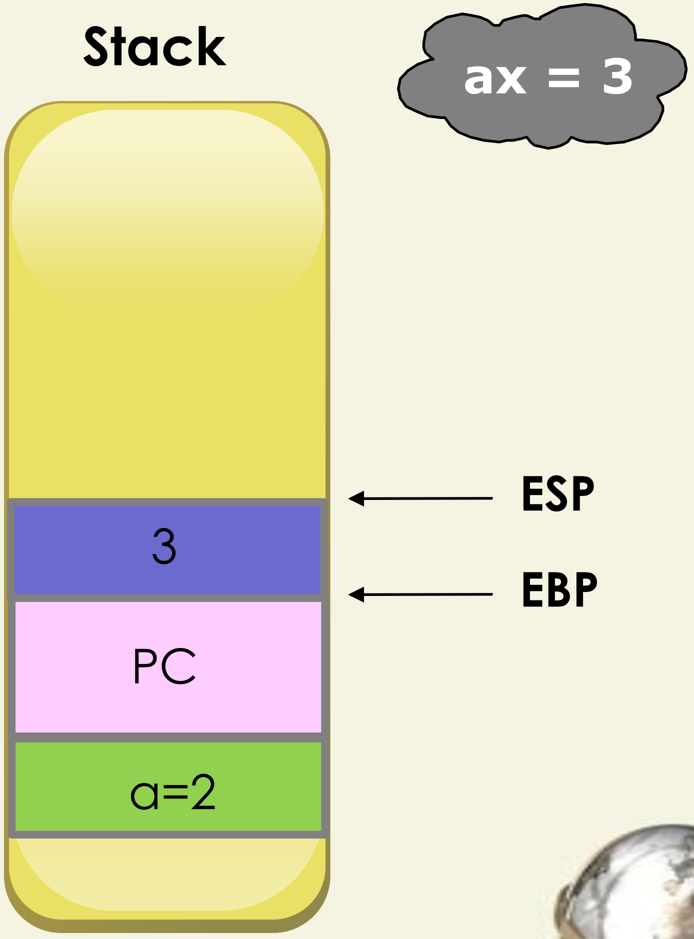
More Examples

```
int abc (int n) {
    int result=n;
    while(n>1) {
        n--;
        result=result*n;
    }
    return result;
}
```



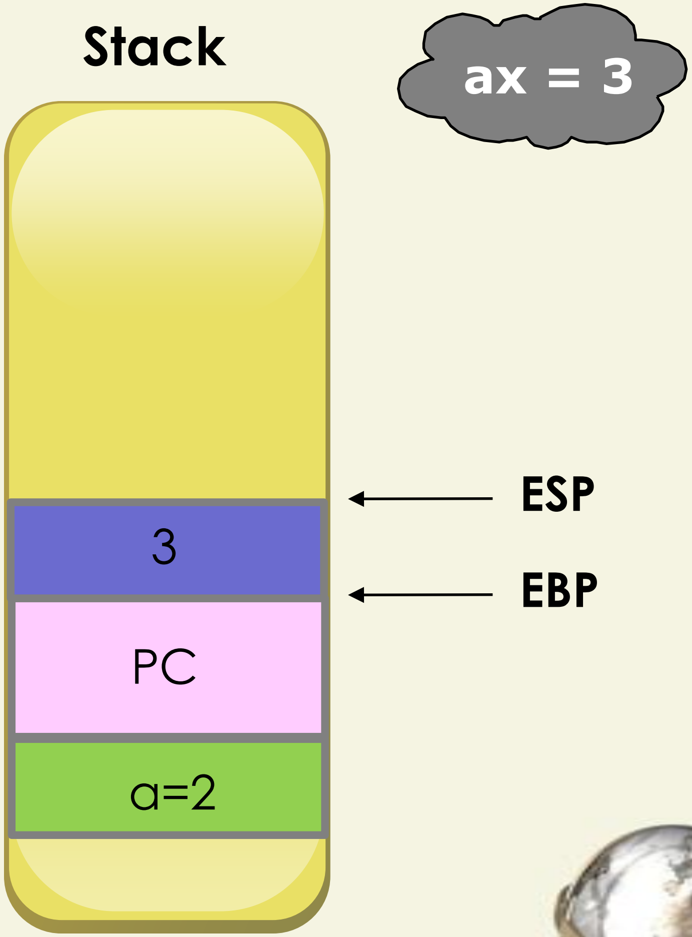
More Examples

```
    . . .
    while(n>1) {
          n--;
          result=result*n;
    }
    . . .

;subprogram
while:
      cmp word[ebp+4], 1
      jng exit
      dec word[ebp+4]
```



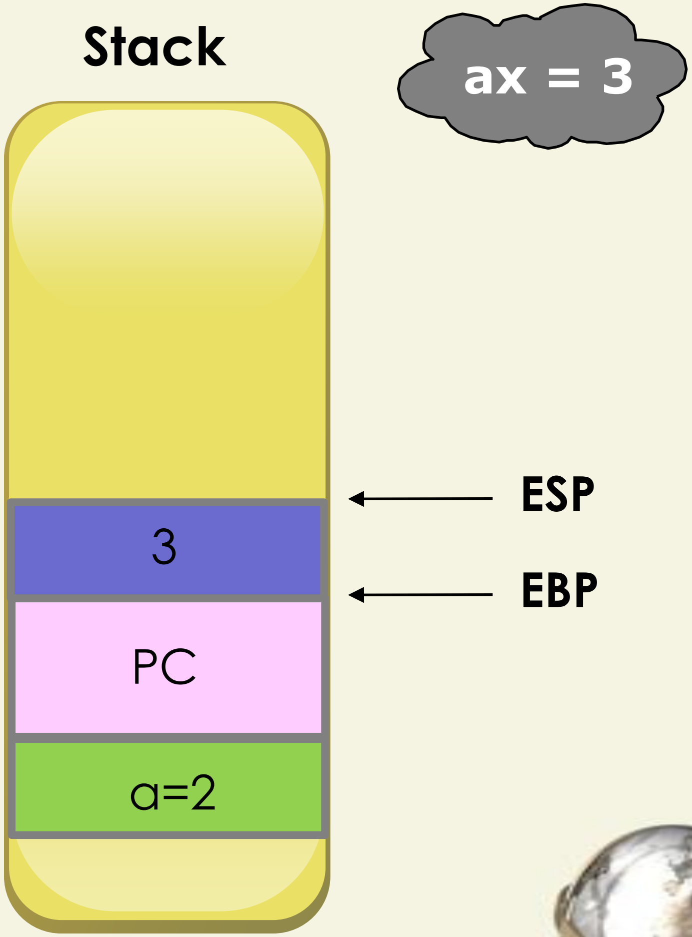
More Examples

```
  . . .
       result=result*n;
  . . .

;subprogram
while:
       cmp word[ebp+4], 1
       jng exit
       dec word[ebp+4]
       mov ax, [ebp-2]
```



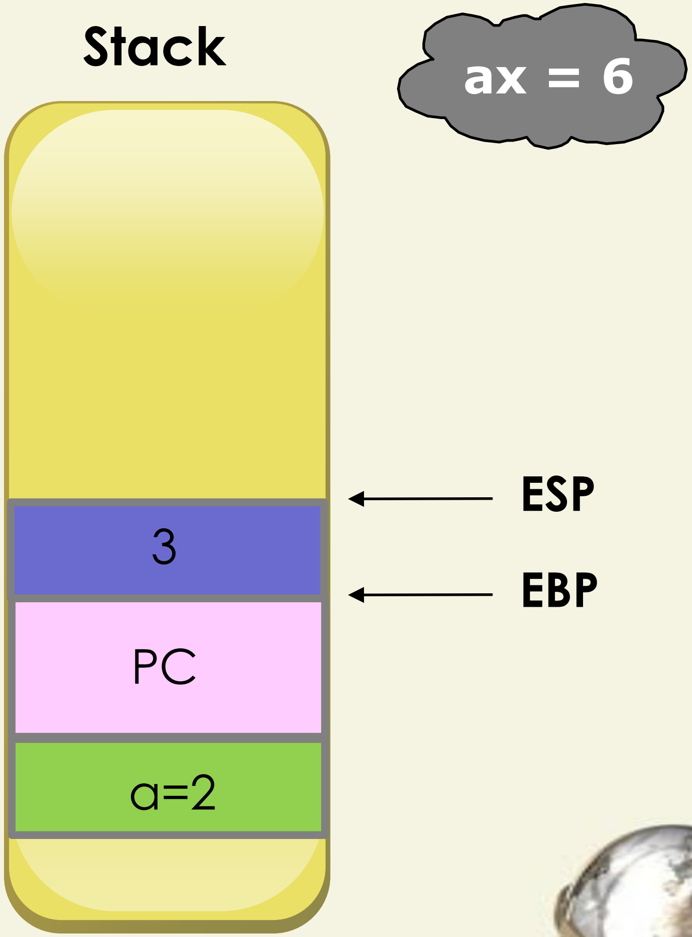
More Examples

```
  . . .
        result=result*n;
  . . .

;subprogram
while:
       cmp word[ebp+4], 1
       jng exit
       dec word[ebp+4]
       mov ax, [ebp-2]
       mul word[ebp+4]
```



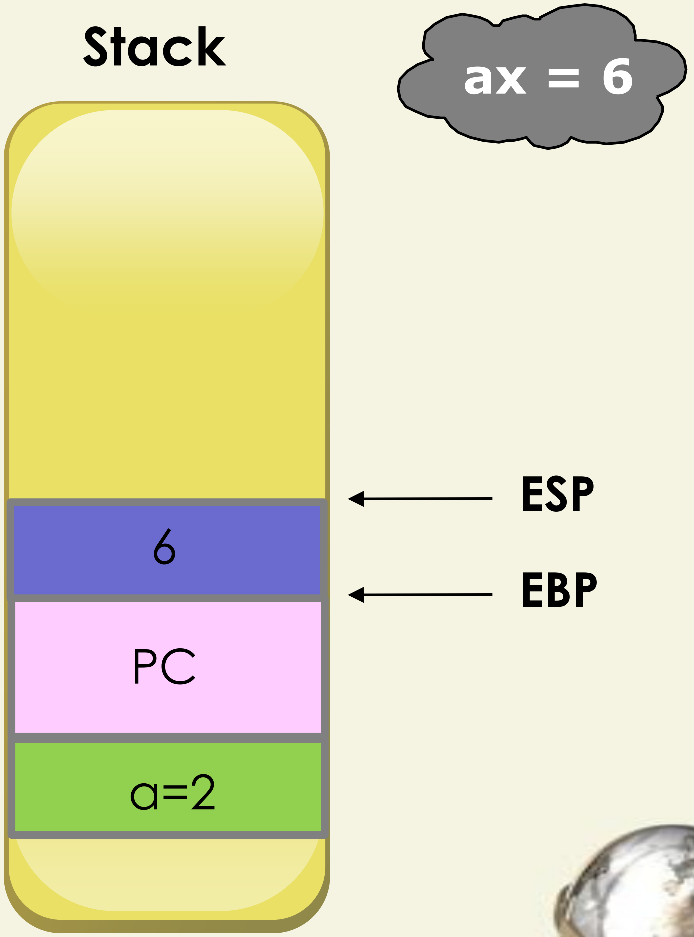
More Examples

```
    . . .
        result=result*n;
    . . .

;subprogram
while:
    cmp word[ebp+4], 1
    jng exit
    dec word[ebp+4]
    mov ax, [ebp-2]
    mul word[ebp+4]
```



More Examples

```
    . . .
        result=result*n;
    . . .

;subprogram
while:
    cmp word[ebp+4], 1
    jng exit
    dec word[ebp+4]
    mov ax, [ebp-2]
    mul word[ebp+4]
    mov word[ebp-2], ax
```



Stack

ax = 6

6

PC

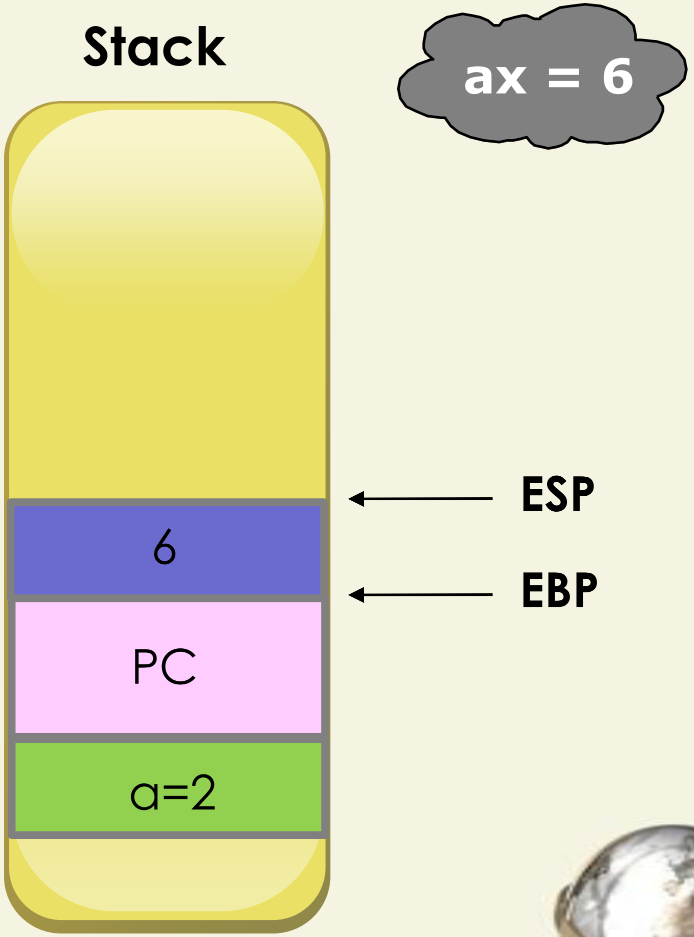
a=2

ESP

EBP

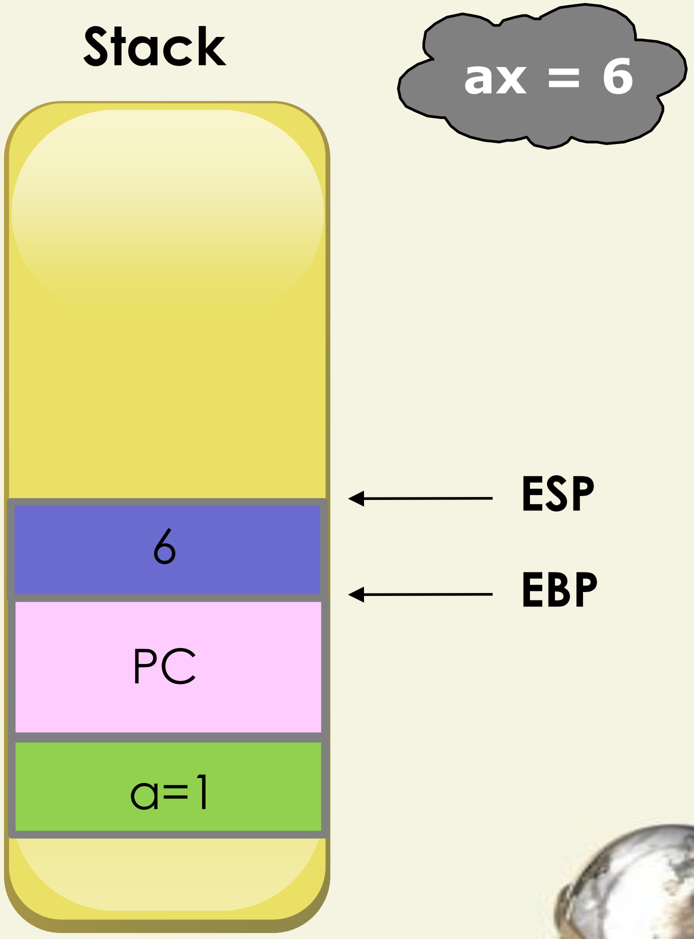
More Examples

```
  . . .
        result=result*n;
  . . .

;subprogram
while:
      cmp word[ebp+4], 1
      jng exit
      dec word[ebp+4]
      mov ax, [ebp-2]
      mul word[ebp+4]
      mov word[ebp-2], ax
      jmp while
```



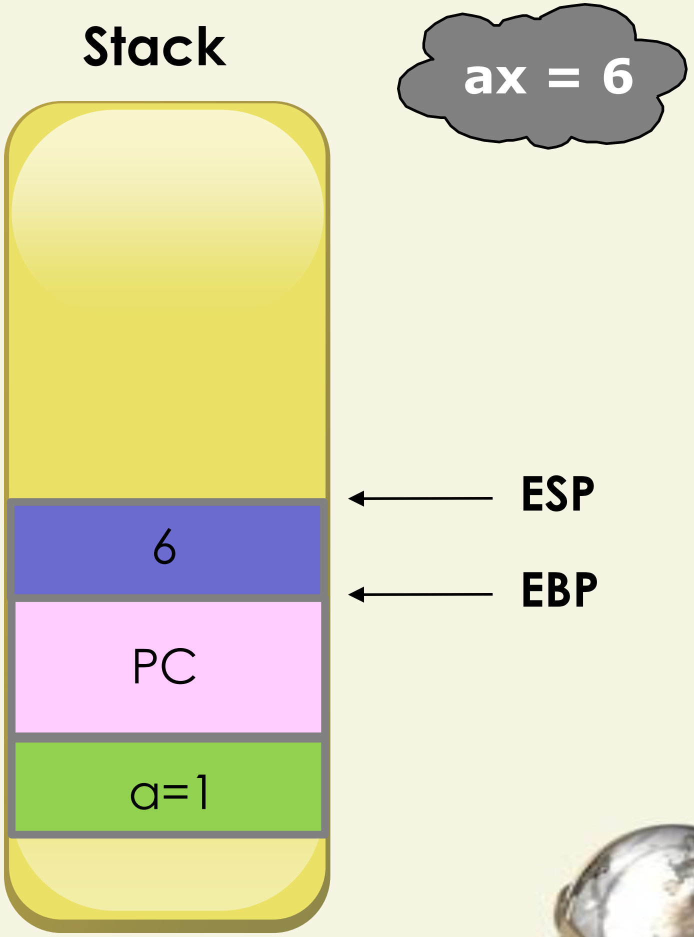
More Examples

```
    . . .
          result=result*n;
    . . .

;subprogram
while:
      cmp word[ebp+4], 1
      jng exit
      dec word[ebp+4]
      mov ax, [ebp-2]
      mul word[ebp+4]
      mov word[ebp-2], ax
      jmp while
```



More Examples

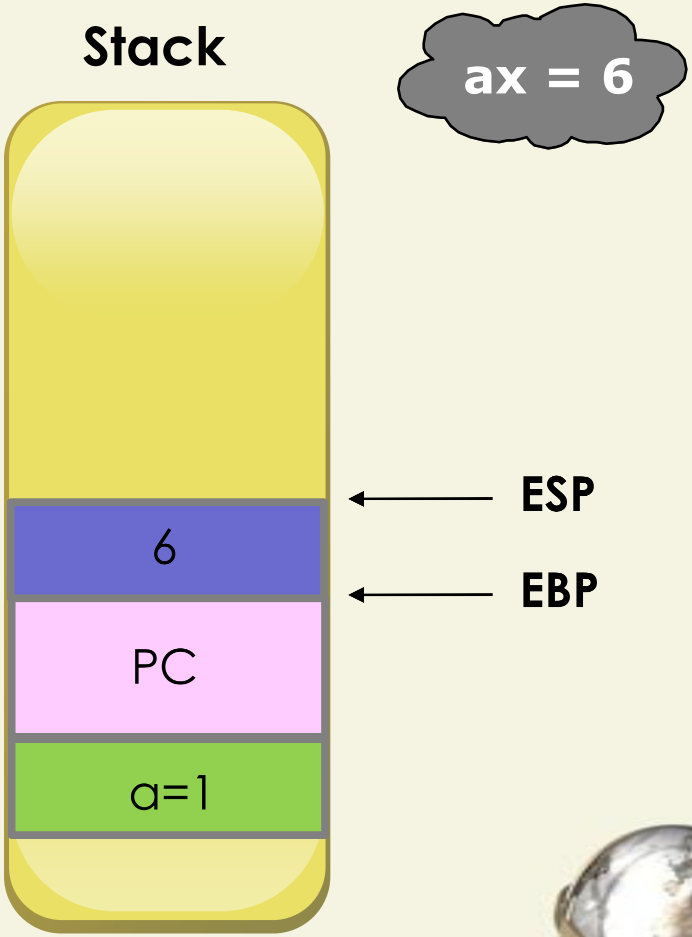
```
int abc (int n) {
     int result=n;
     while(n>1) {
            n--;
            result=result*n;
     }
     return result;
}
```



More Examples

;subprogram

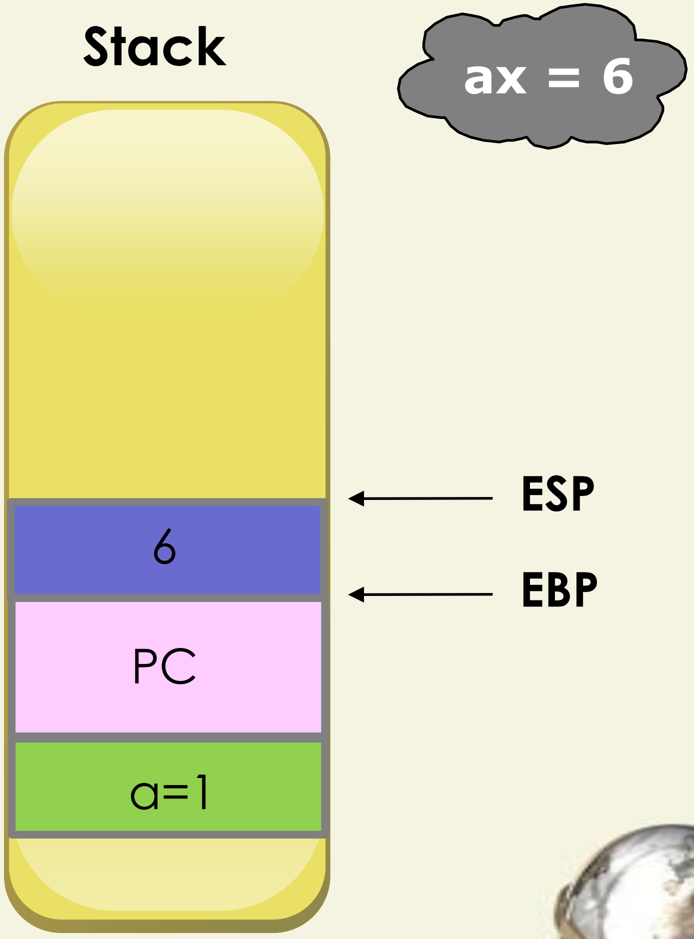
```
abc:
    ...
    mov word[ebp-2], ax
    while:
      cmp word[ebp+4], 1
      jng exit
      dec word[ebp+4]
      mov ax, [ebp-2]
      mul word[ebp+4]
      mov word[ebp-2], ax
      jmp while
```



More Examples

;subprogram

```
abc:
    ...
    mov word[ebp-2], ax
    while:
      . . .
      jmp while
    exit:
```



Stack

ax = 6

6

PC

a=1

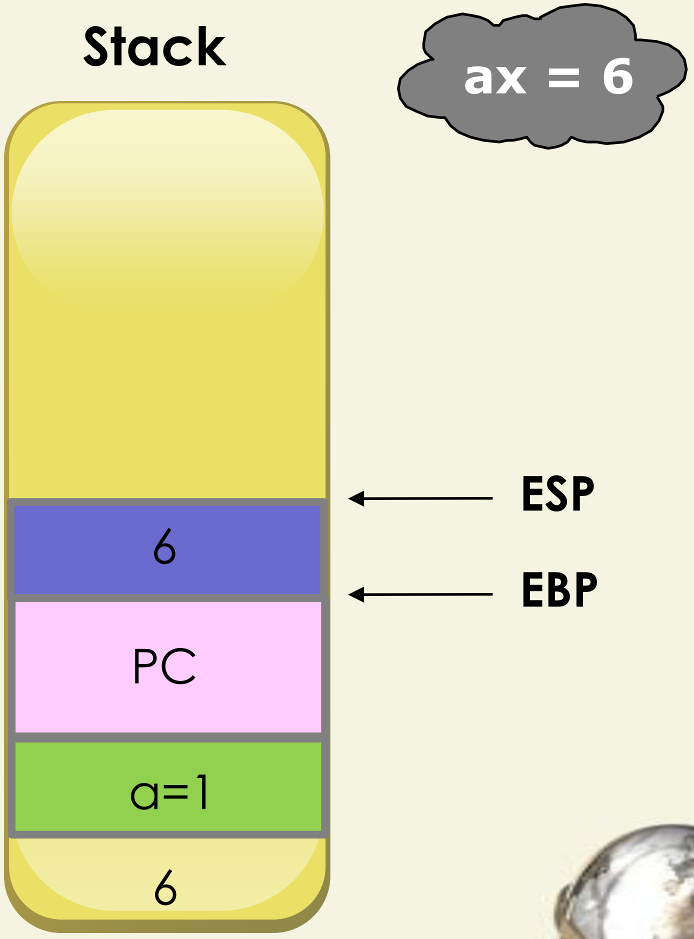
ESP

EBP

More Examples

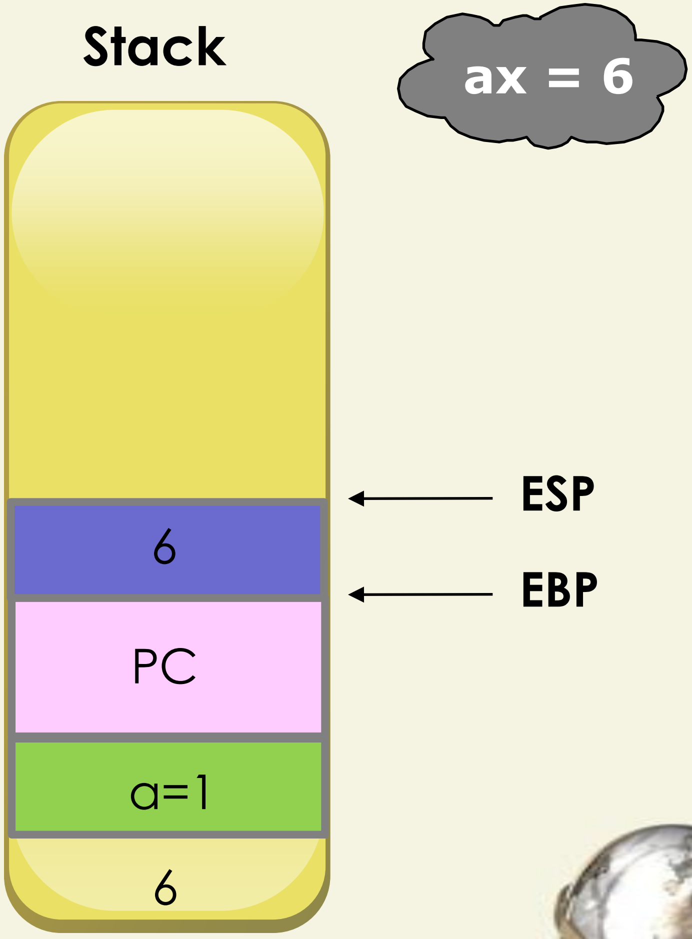
;subprogram

```
abc:
    ...
    mov word[ebp-2], ax
    while:
      . . .
      jmp while
    exit:
```



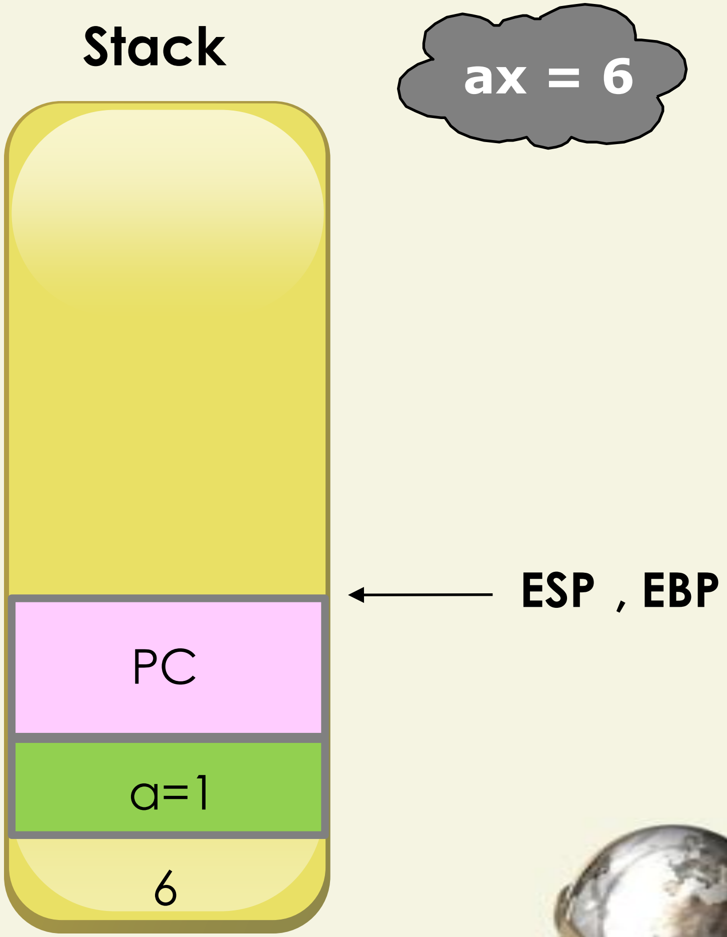
More Examples

```
;subprogram

abc:
    ...
    mov word[ebp-2], ax
    while:
      . . .
      jmp while
    exit:
      mov ax, [ebp-2]
     mov word[ebp+6], ax
```



More Examples

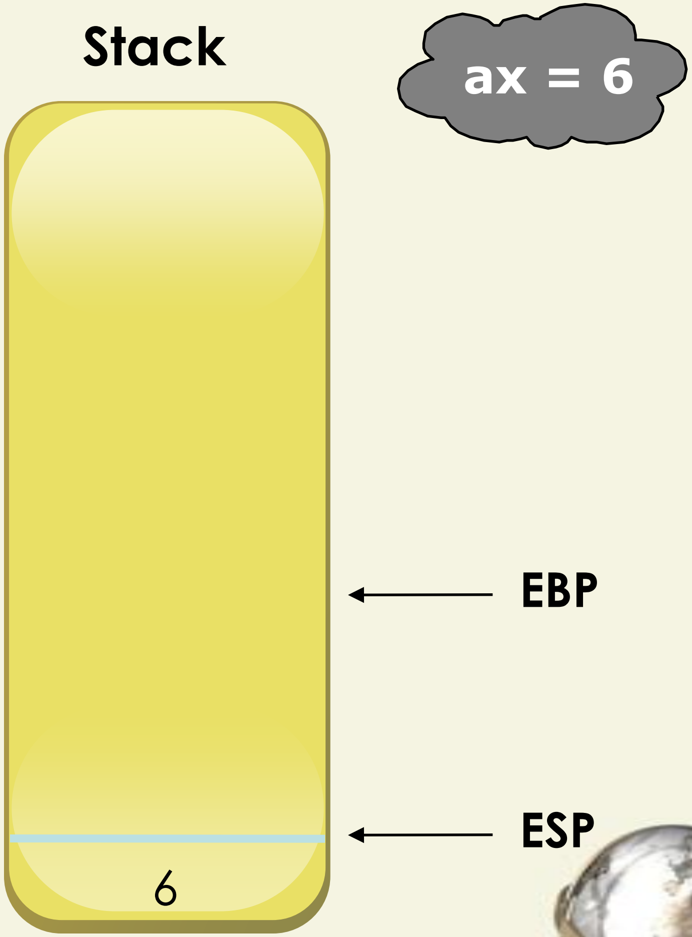
```
;subprogram

abc:
    ...
    mov word[ebp-2], ax
    while:
      . . .
      jmp while
    exit:
      mov ax, [ebp-2]
     mov word[ebp+6], ax
     add esp, 2
```



More Examples

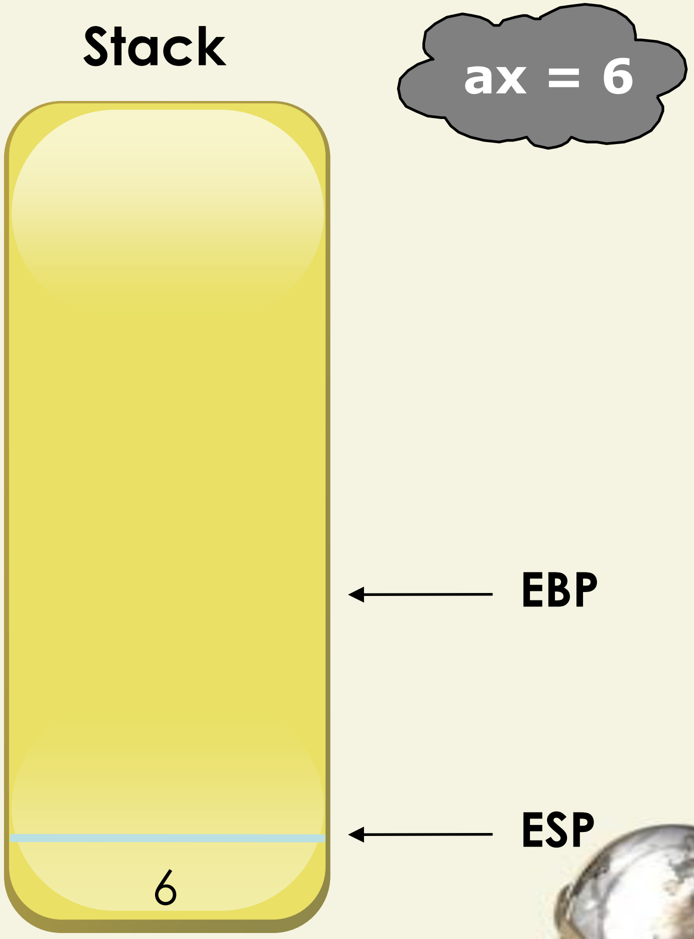
;subprogram

```
abc:
    ...
    mov word[ebp-2], ax
    while:
      . . .
      jmp while
    exit:
      mov ax, [ebp-2]
     mov word[ebp+6], ax
     add esp, 2
     ret 2
```



More Examples

```
;subprogram call

sub sp, 2
push word [a]
call abc
pop word [r]
```



Pointers Review (C Programming)

- Pointers
  - variables which hold the address of other variables
  - tell user where a variable resides in memory
  - can access a variable indirectly

```
int *p;          int x;
x = 10;
p = &x;
*p = 100;
```

Variables and their Addresses

- Variables are just locations in memory.
- Variable name == Human readable location of variable in memory.
- Variable location in memory
  - Offset from the start of the Data Segment
- num
  - address of variable
- [num]
  - value at memory address DS+num

Machine Equivalent of Pointers

```
int *EBX;
int x, y;

x = 100;
EBX = &x;
y = *EBX – 90
```

```
x  dw 0
y dw 0

mov word [x], 100
mov ebx, x
mov ax, [ebx]
sub ax, 90
mov word[y], ax
```

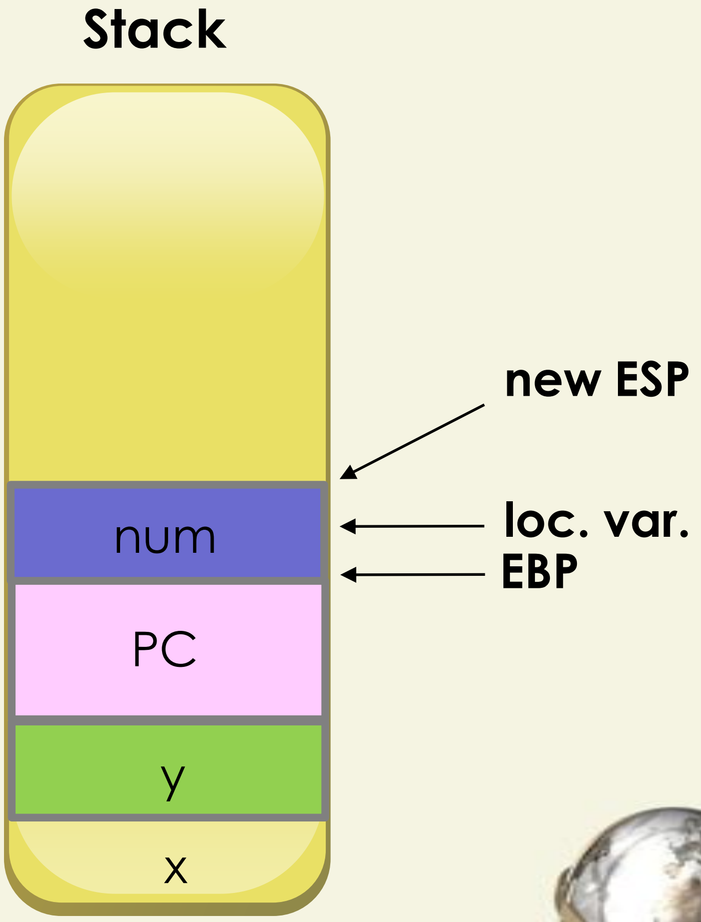
Caution

- Registers and globally declared variables may be changed within any subprogram.

- Whatever the final values of registers and globally declared variables are at the end of the subprogram will be the value they hold when they return to the calling subprogram.

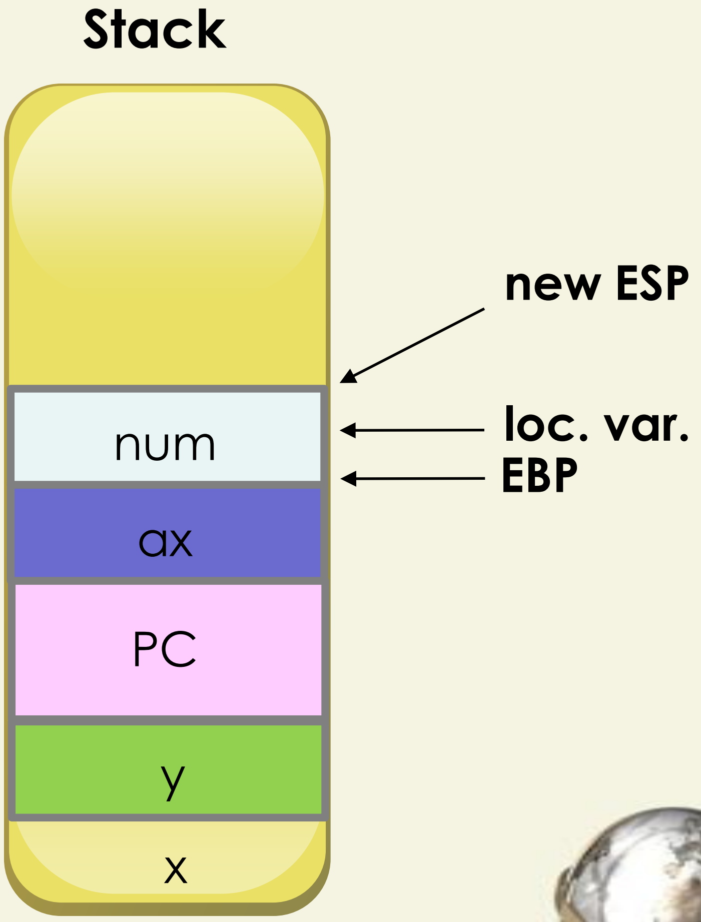
Recall: Stack

```
sum:
    mov  ebp, esp
    sub  esp, 2
    mov  ax, [ebp + 6]
    add  ax, [ebp + 4]
    mov  [ebp – 2], ax
    add  esp, 2
    ret  4
```



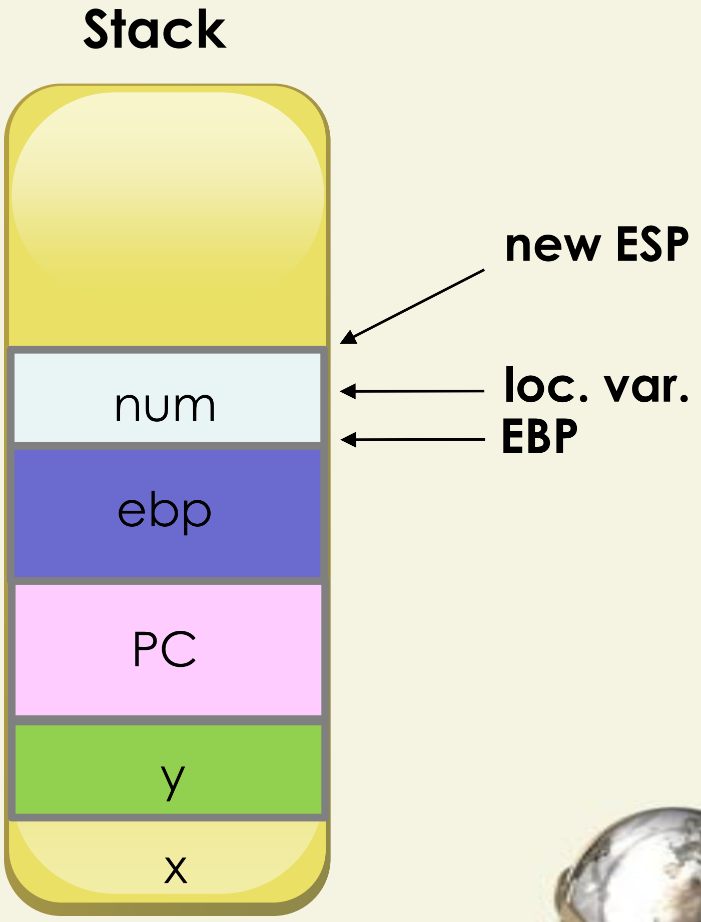
Recall: Stack

```
sum:
    push ax
    mov  ebp, esp
    sub  esp, 2
    mov  ax, [ebp + 8]
    add  ax, [ebp + 6]
    mov  [ebp – 2], ax
    add  esp, 2
    pop ax
    ret   4
```



Recall: Stack

```
sum:
    push ebp
    mov  ebp, esp
    sub  esp, 2
    mov  ax, [ebp + 10]
    add  ax, [ebp + 8]
    mov  [ebp – 2], ax
    add  esp, 2
    pop ebp
    ret    4
```



Saving Registers

- At the start of the subprogram, save all registers not just EBP.
- At the end of the subprogram, restore all registers not just EBP.
- **pusha**
  - EAX ECX EDX EBX ESP EBP ESI EDI
- **popa**

Recall

;subprogram call

```
void more
(int *x, int y, int z){
   *x = y * z ;
}

more (&d, e, f);
```

Recall

```
void more
(int *x, int y, int z){
   *x = y * z ;
}


more (&d, e, f);
```

```
;subprogram call

push d
push word[e]
push word[f]
call more
```

Recall

```
void more
(int *x, int y, int z){
   *x = y * z;
}


more (&d, e, f);
```

```
;subprogram

more:
```

Recall

```
void more
(int *x, int y, int z){
   *x = y * z;
}


more (&d, e, f);
```

```
;subprogram

more:
    mov ebp, esp





    ret 8
```

Recall

```
void more
(int *x, int y, int z){
   *x = y * z;
}


more (&d, e, f);
```

```
;subprogram

more:
    mov ebp, esp
    mov ax, [ebp+6]
    mul word[ebp+4]



    ret 8
```

Recall

```
void more
(int *x, int y, int z){
   *x = y * z;
}


more (&d, e, f);
```

```
;subprogram

more:
    mov ebp, esp
    mov ax, [ebp+6]
    mul word[ebp+4]
    mov ebx, [ebp+8]
    mov [ebx], ax
    ret 8
```

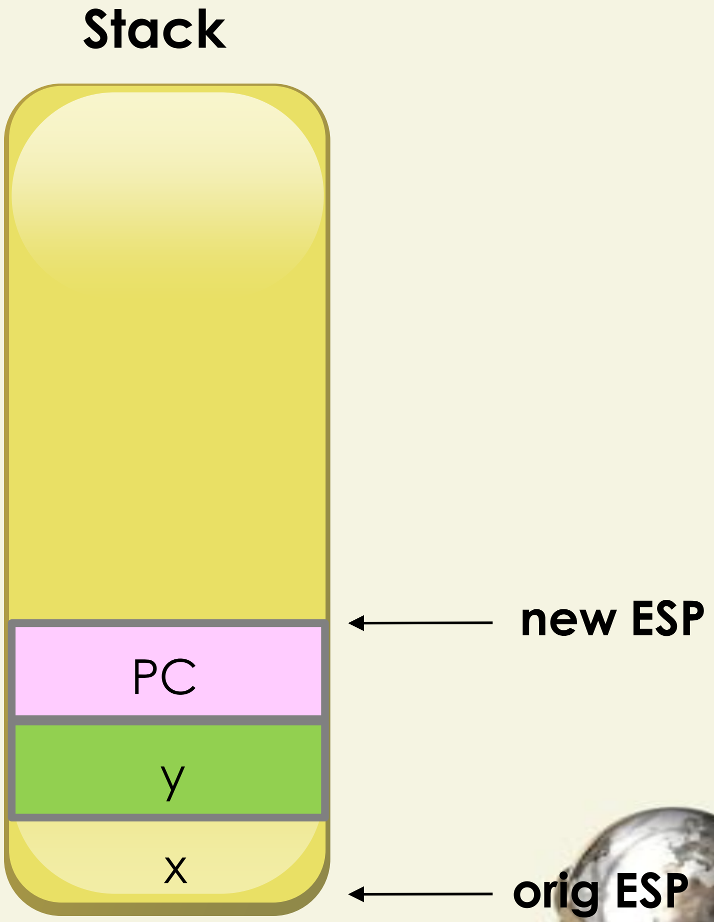
Bonus

```
bonus:
    mov ebp, esp
    mov ax, [ebp+6]
    while:
        cmp word[ebp+4], 1
        jng exit
        add ax, [ebp+6]
        dec word[ebp+4]
        jmp while
    exit:
        mov word[ebp+8], ax
        ret 4
```

Stack

subprogram call:

```
push     word [x]
push     word [y]
call   sum
```



Value Parameters

subprogram:

```
void sum (int a, int b)
{
    int num;
    num = a + b;
}
```

Value Parameters

```
sum:
   mov  ebp, esp
   sub   esp, 2
   mov  ax, [ebp + 4]
   add   ax, [ebp + 2]
   mov  [ebp – 2], ax
   add   esp, 2
   ret     4
```

subprogram:

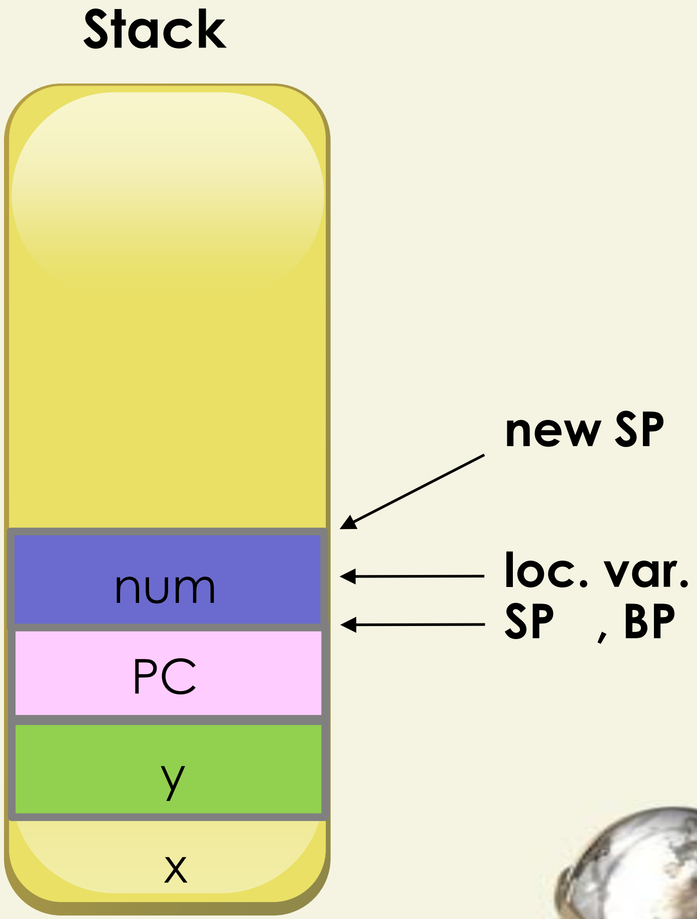
```
void sum (int a, int b)
{
      int num;
      num = a + b;
}
```

Value Parameters

```
sum:
   mov  ebp, esp              ; create stack frame
   sub   esp, 2               ; reserve local
   variable
   mov  ax, [ebp + 4] ; retrieve parameter a
   add  ax, [ebp + 2]  ; retrieve parameter b
   mov  [ebp – 2], ax ; num = a + b
   add   esp, 2               ; release local
   variable
   ret     4                ; return to caller and
                                clear stack
```

Stack

```
sum:
    mov  ebp, esp
    sub   esp, 2
    mov  ax, [ebp + 4]
    add  ax, [ebp + 2]
    mov  [ebp – 2], ax
    add   esp, 2
    ret     4
```



Variable Parameters

subprogram:

```
void sum
(int *n, int a, int b) {
    *n = a + b;
}
```

subprogram call:

```
sum(&num, x, y);
```

Variable Parameters

subprogram:

```
void sum
(int *n, int a, int b) {
    *n = a + b;
}
```

subprogram call:

```
sum(&num, x, y);
```

subprogram call:

```
push num
push word [x]
push word [y]
call    sum
```

Quiz

```
sum:
    mov  bp, sp          ; create stack frame
    mov  ax, _____       ; retrieve parameter a
    add  ax, _____       ; retrieve parameter b
    mov  bx, _____       ; BX = &num
    mov [bx], ax         ; *BX = a + b
    ret     6            ; return to caller and
                           clear stack
```