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*

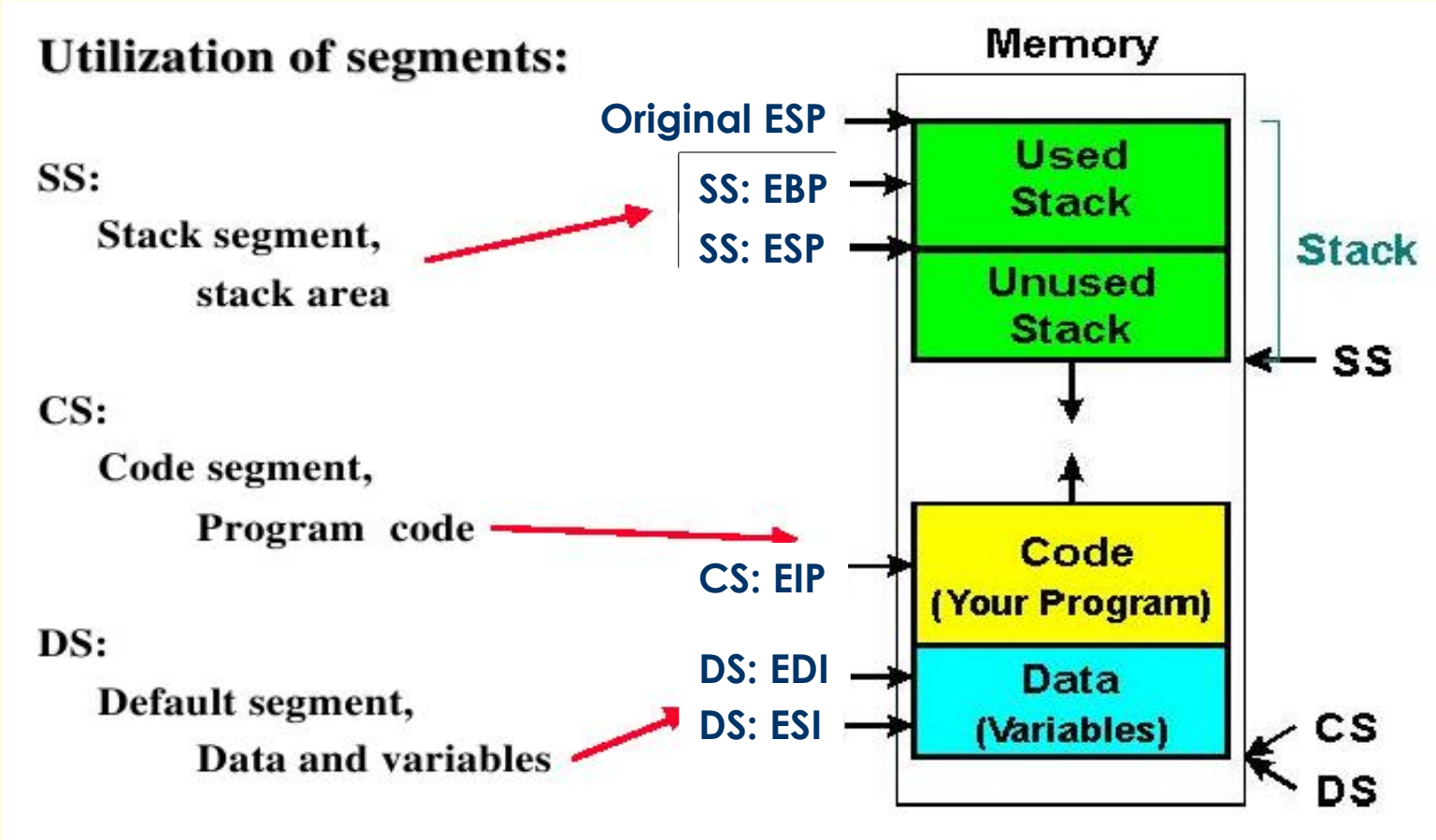
- PC is saved to allow the computer to return to the calling subprogram

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
Memory
Original ESP
SS: EBP
SS: ESP
Used Stack
Unused Stack
Stack
SS
CS: EIP
Code (Your Program)
DS: EDI
DS: ESI
Data (Variables)
CS
DS

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();
```

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);
```

(Assembly)

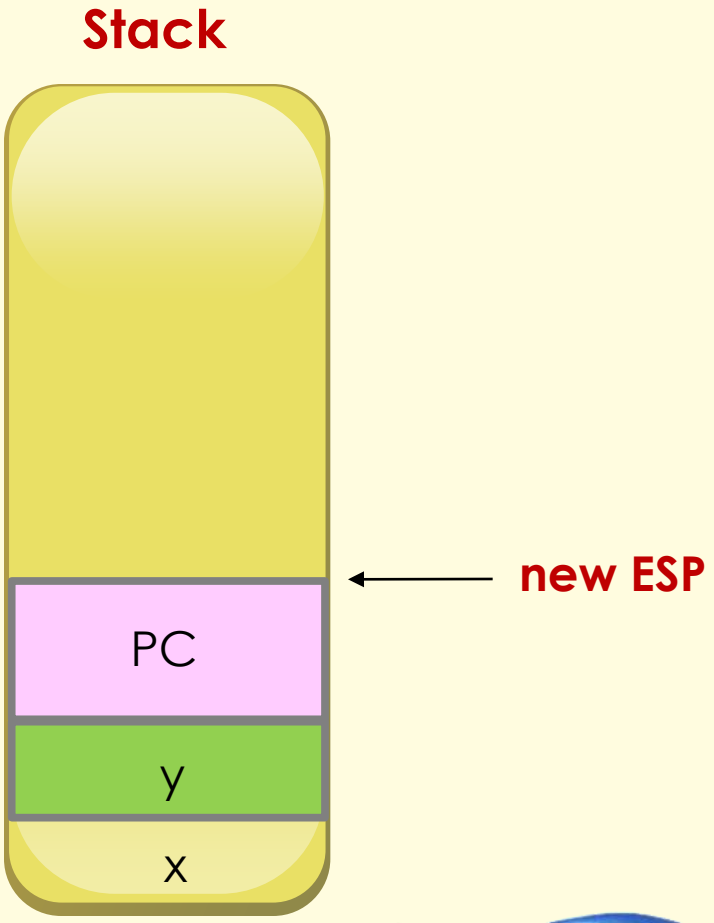
subprogram call:

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

Stack

subprogram call:

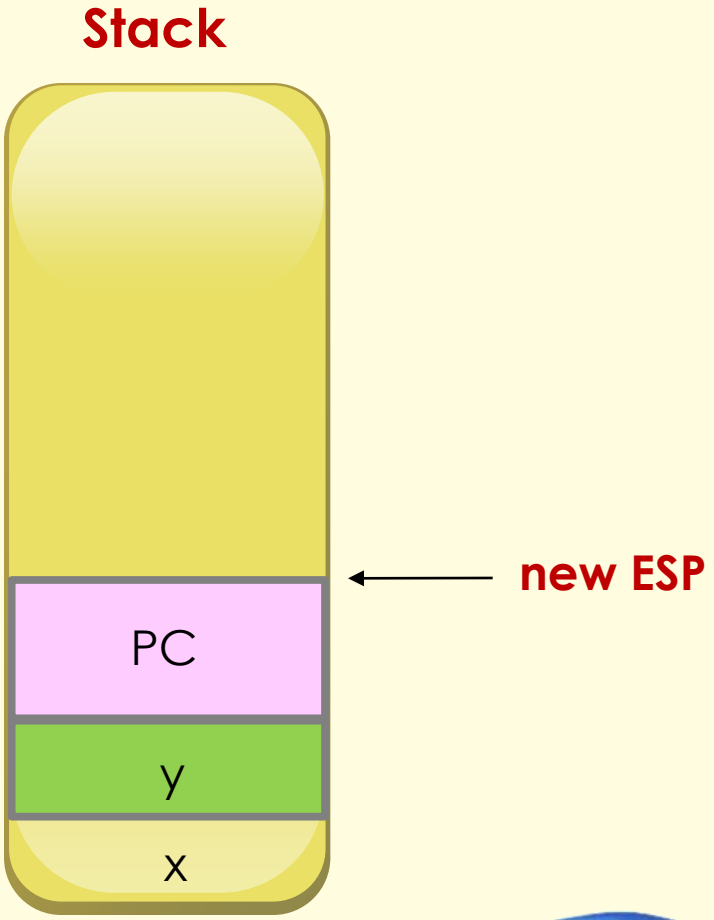
```
push  word [x]
push  word [y]
call    sum
```



Value Parameters

subprogram:

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

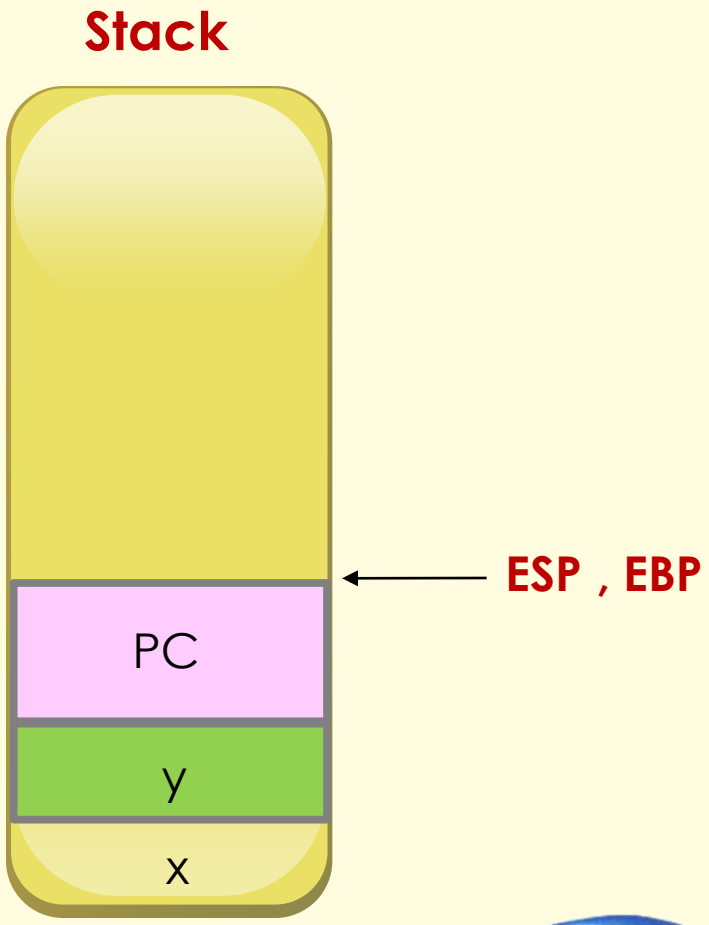


Value Parameters

subprogram:

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

sum:
    mov  ebp, esp
```

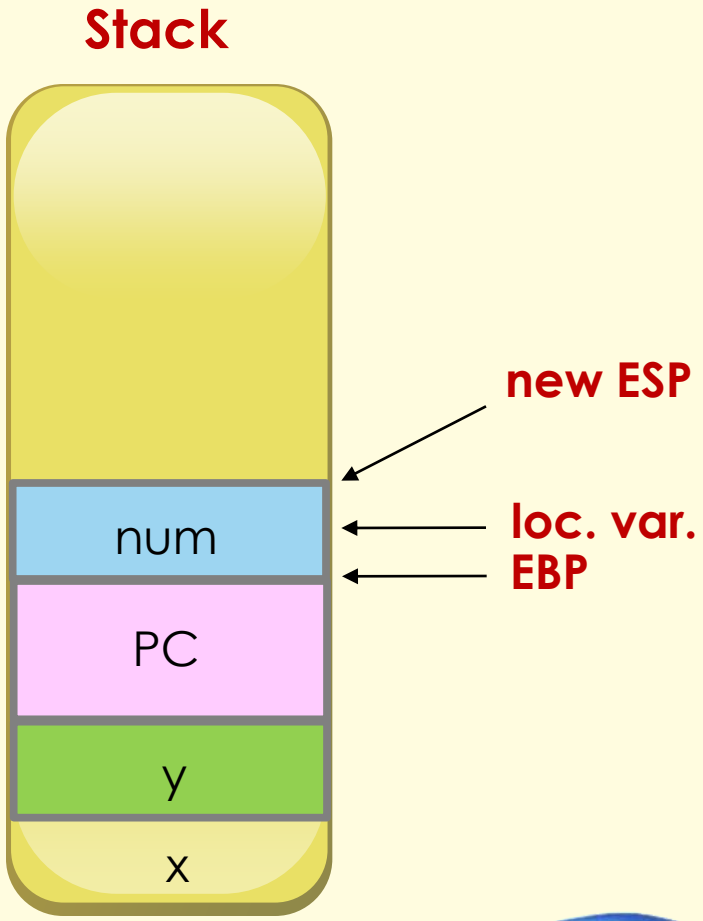


Value Parameters

subprogram:

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

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

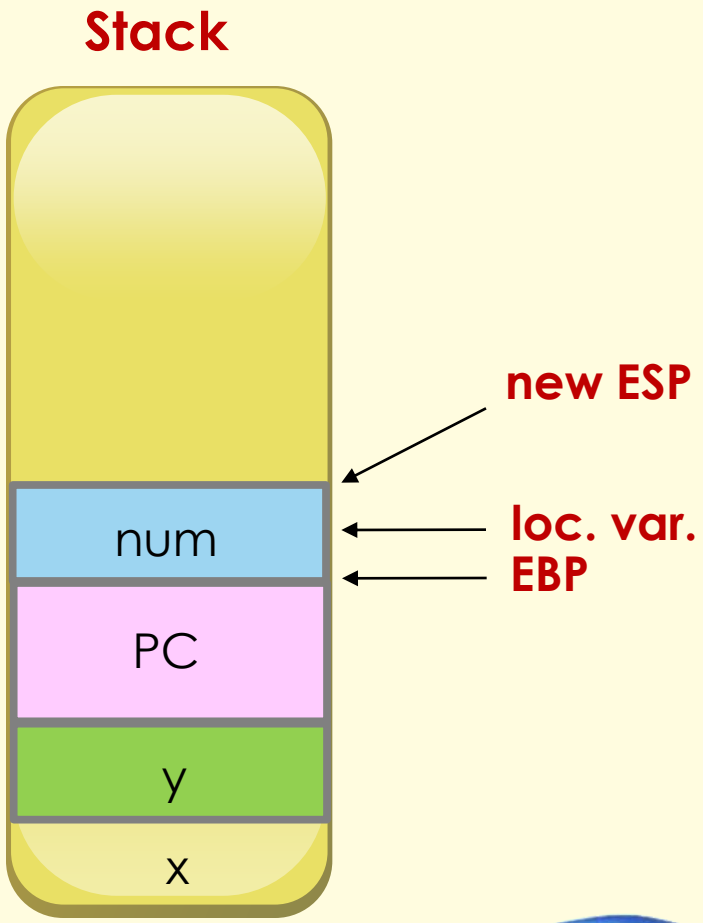


Value Parameters

subprogram:

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

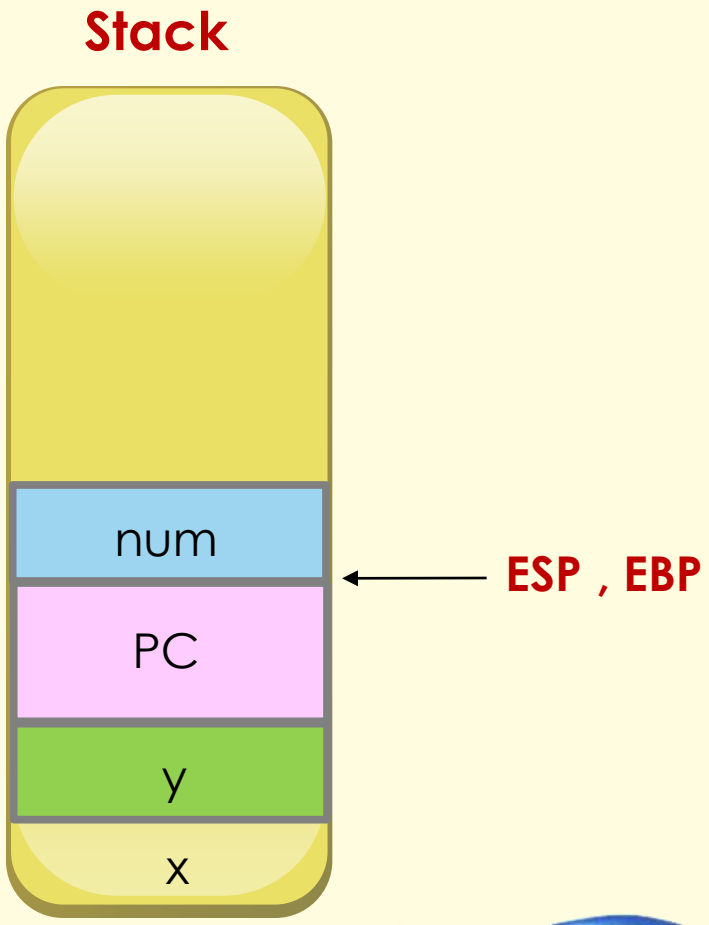
```
sum:
    mov  ebp, esp
    sub   esp, 2
    mov  ax, [ebp + 6]
    add   ax, [ebp + 4]
    mov  [ebp – 2], ax
```



Value Parameters

subprogram:

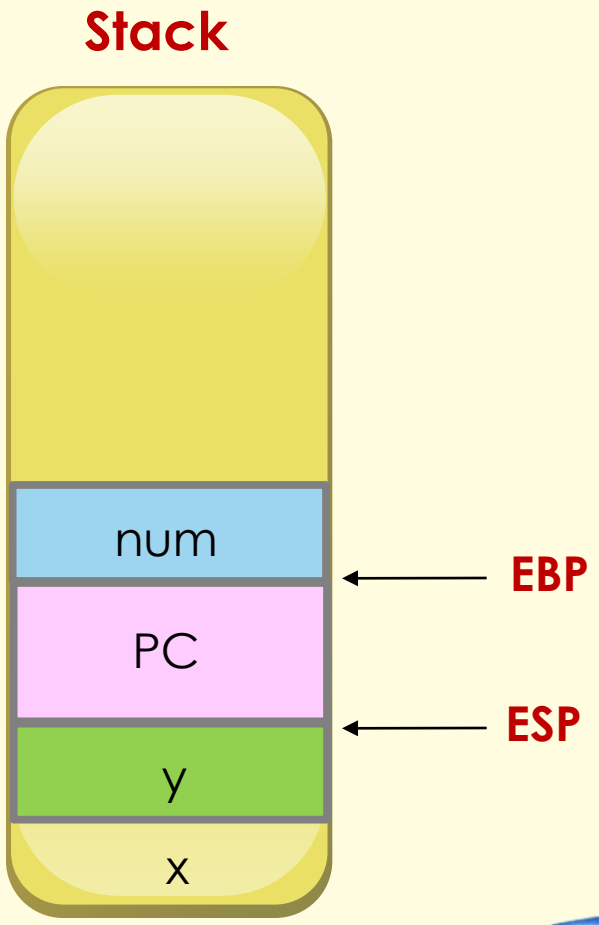
```
void sum (int a, int b){
        int num;
        num = a + b;
}
sum:
      mov  ebp, esp
      sub   esp, 2
      mov  ax, [ebp + 6]
      add   ax, [ebp + 4]
      mov  [ebp – 2], ax
      add   esp, 2
```



Value Parameters

subprogram:

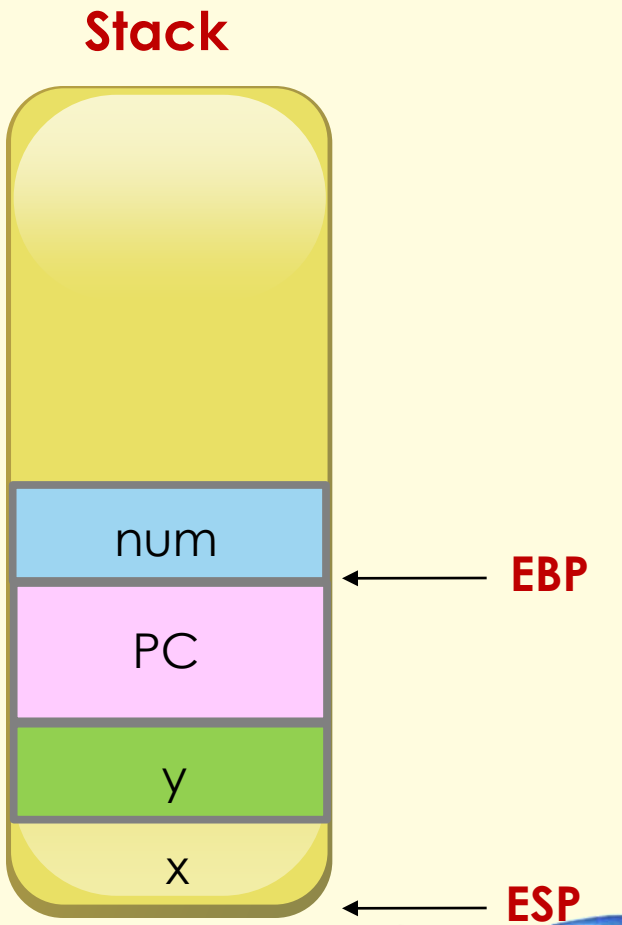
```
void sum (int a, int b){
        int num;
        num = a + b;
}
sum:
      mov  ebp, esp
      sub   esp, 2
      mov  ax, [ebp + 6]
      add   ax, [ebp + 4]
      mov  [ebp – 2], ax
      add   esp, 2
      ret    4
```



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);
```

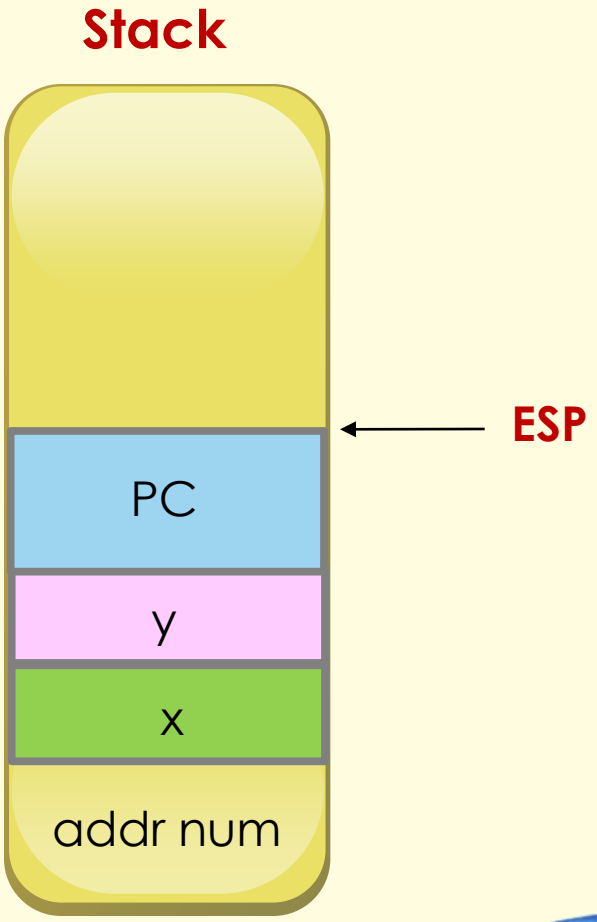
subprogram call:

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

Stack

subprogram call:

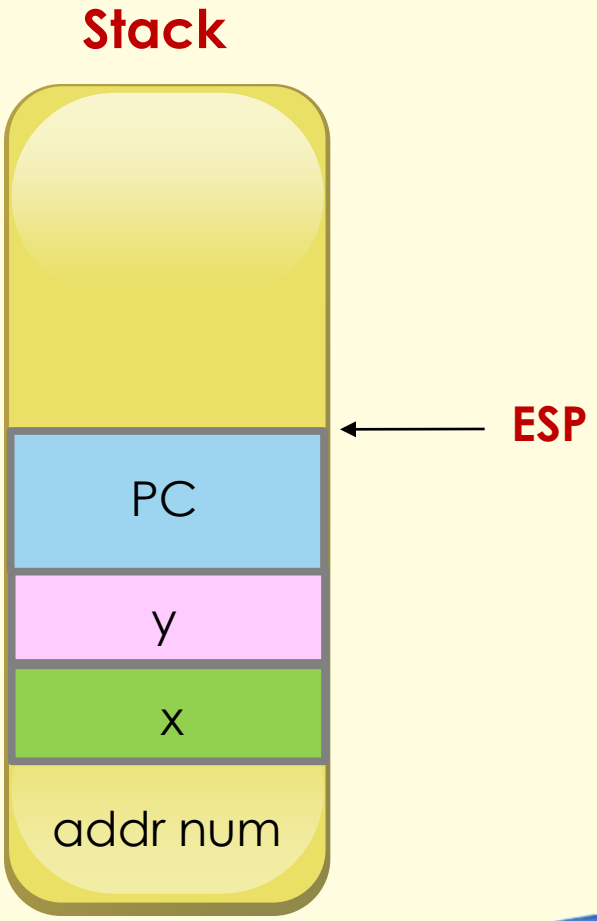
push num
push word [x]
push word [y]
call sum



Variable Parameters

subprogram:

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

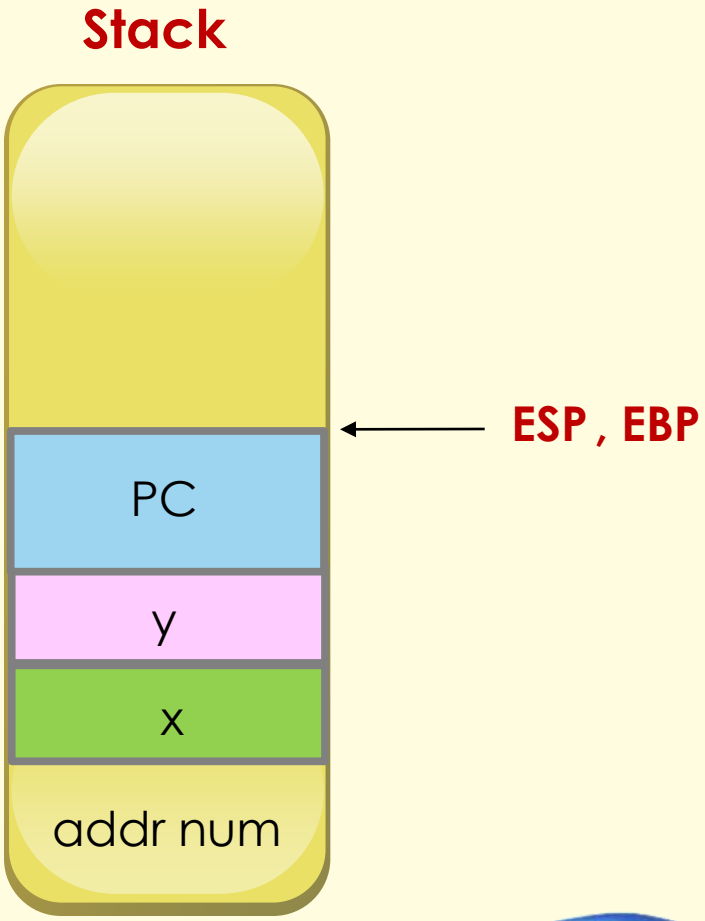


Variable Parameters - Stack

subprogram:

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

sum:
    mov  ebp, esp
```

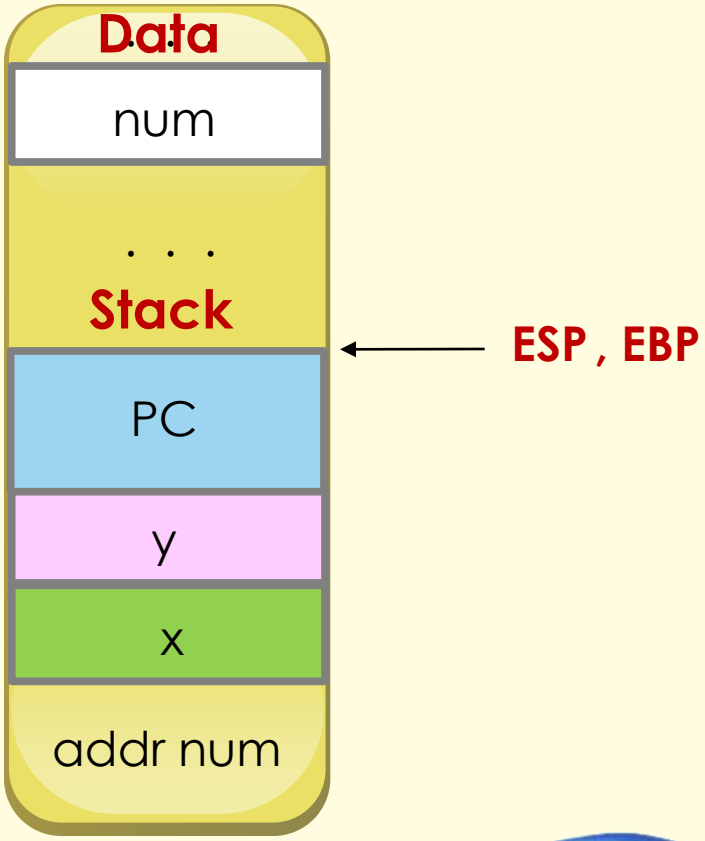


Variable Parameters - Stack

subprogram:

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

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

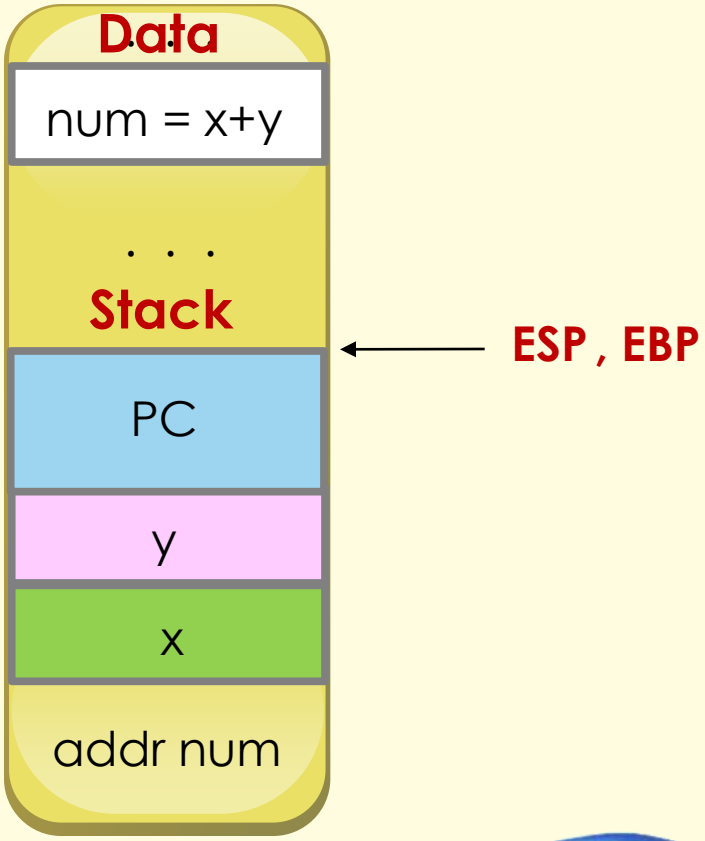


Variable Parameters - Stack

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
```

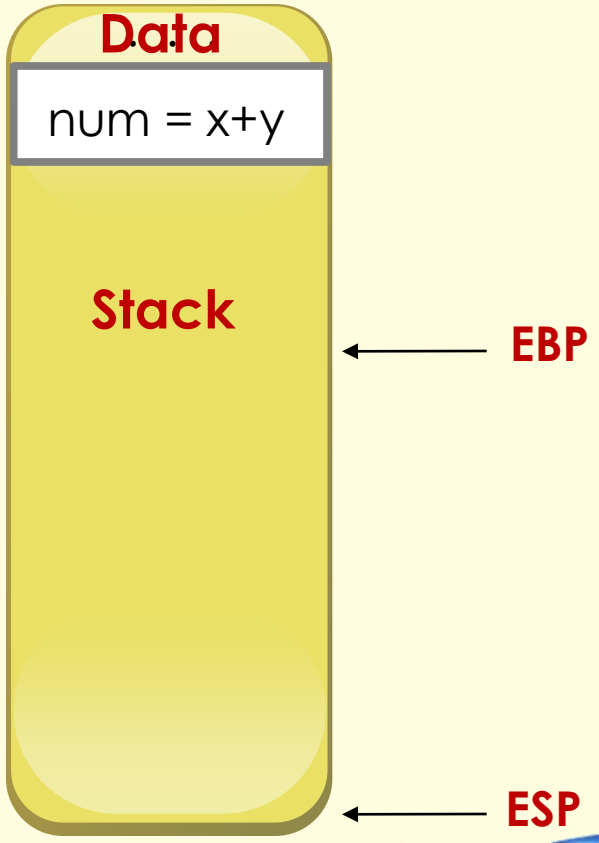


Variable Parameters - Stack

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
```