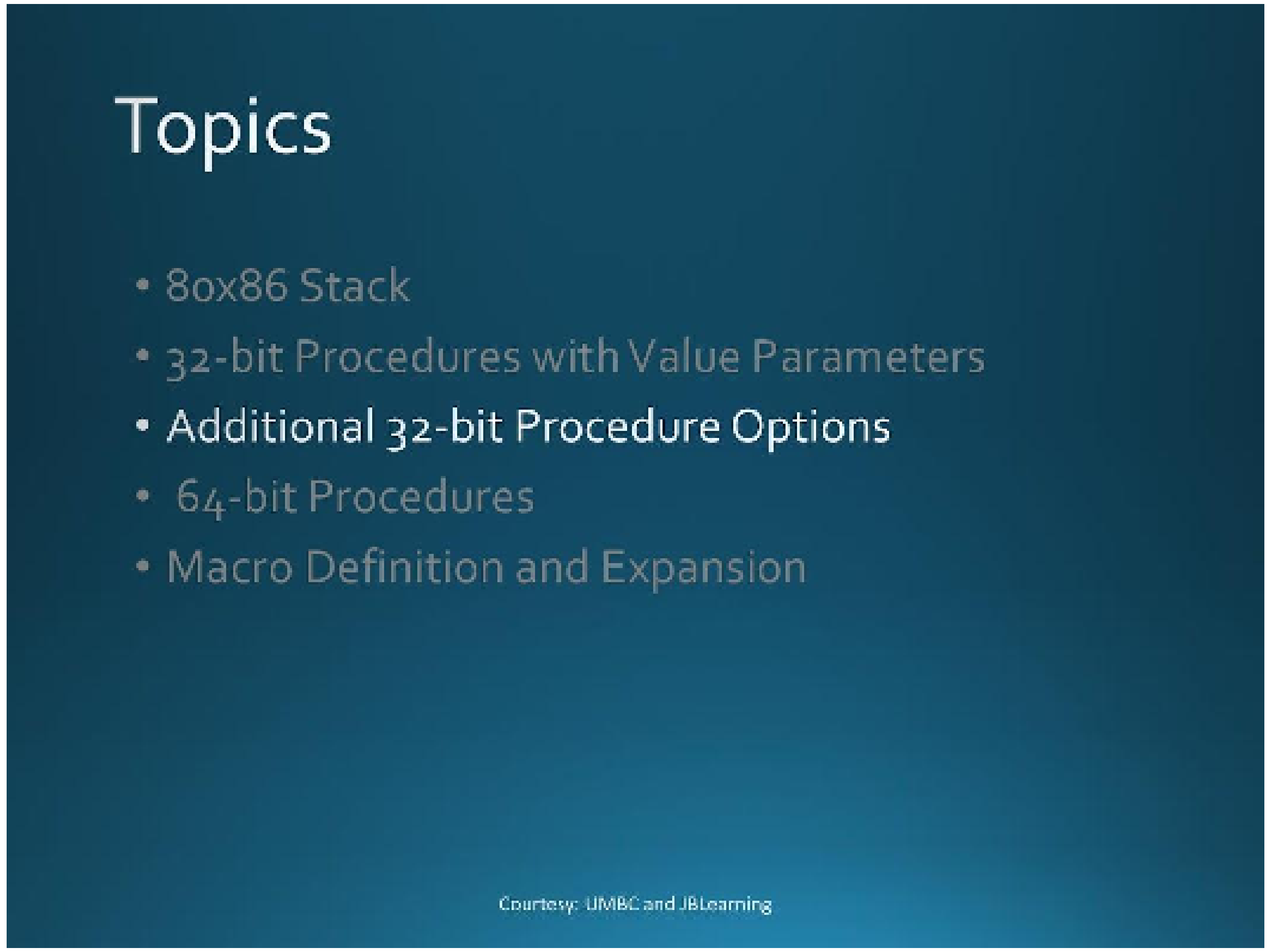
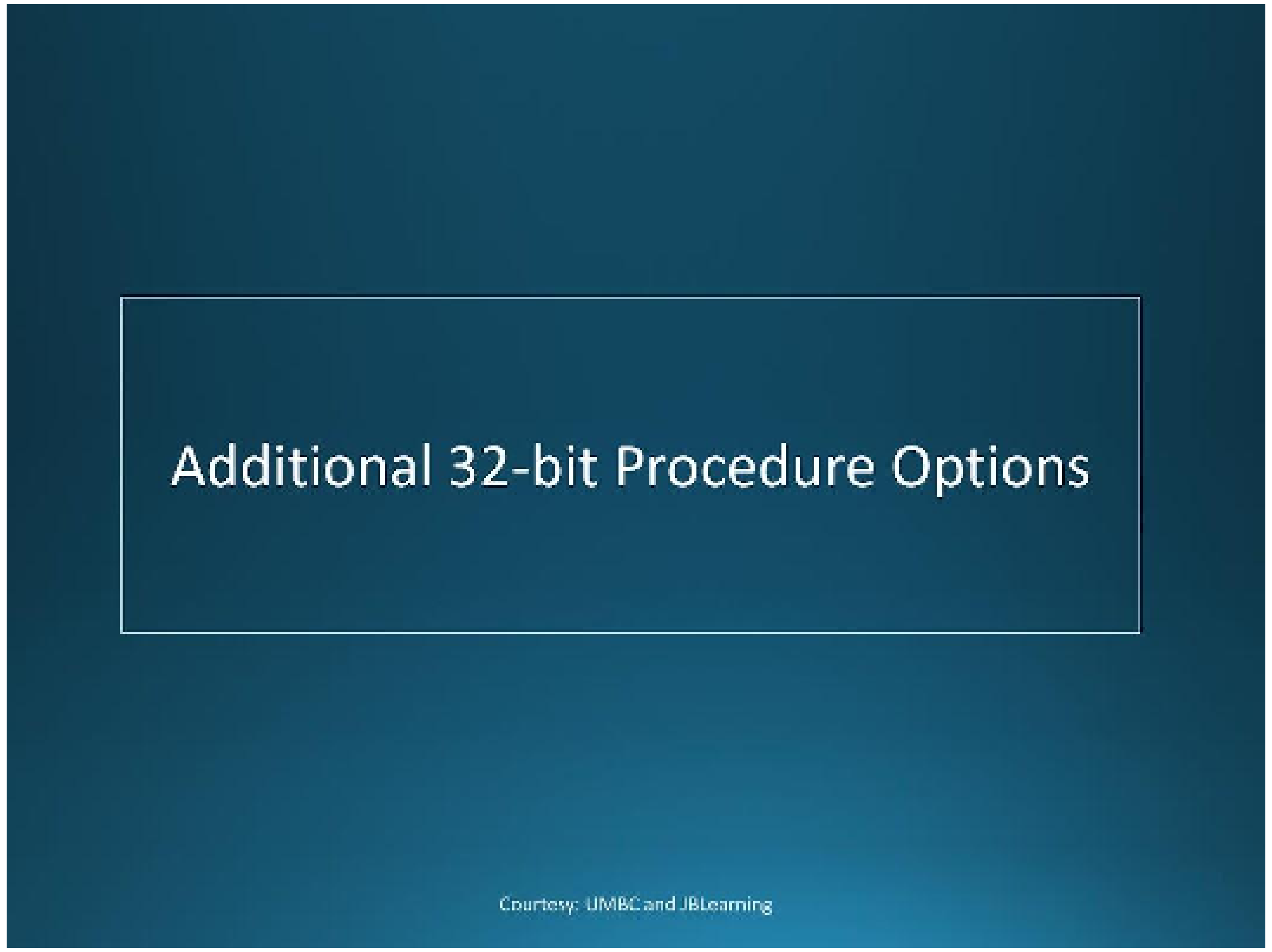


Topics

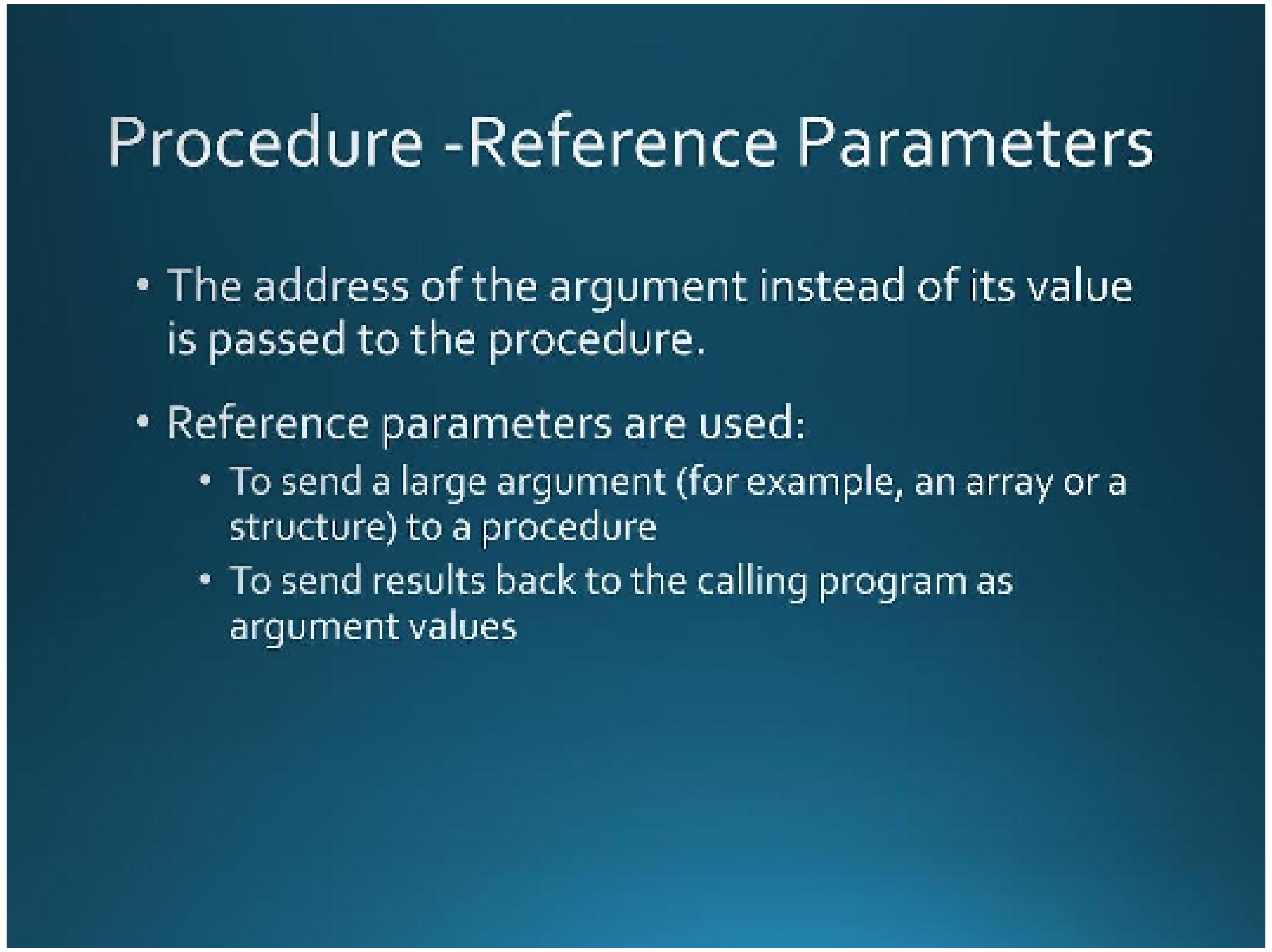
- 80x86 Stack
- 32-bit Procedures with Value Parameters
- Additional 32-bit Procedure Options
- 64-bit Procedures
- Macro Definition and Expansion

Courtesy: UMBC and JBLearning



Additional 32-bit Procedure Options

Courtesy: UMBC and JBLearning



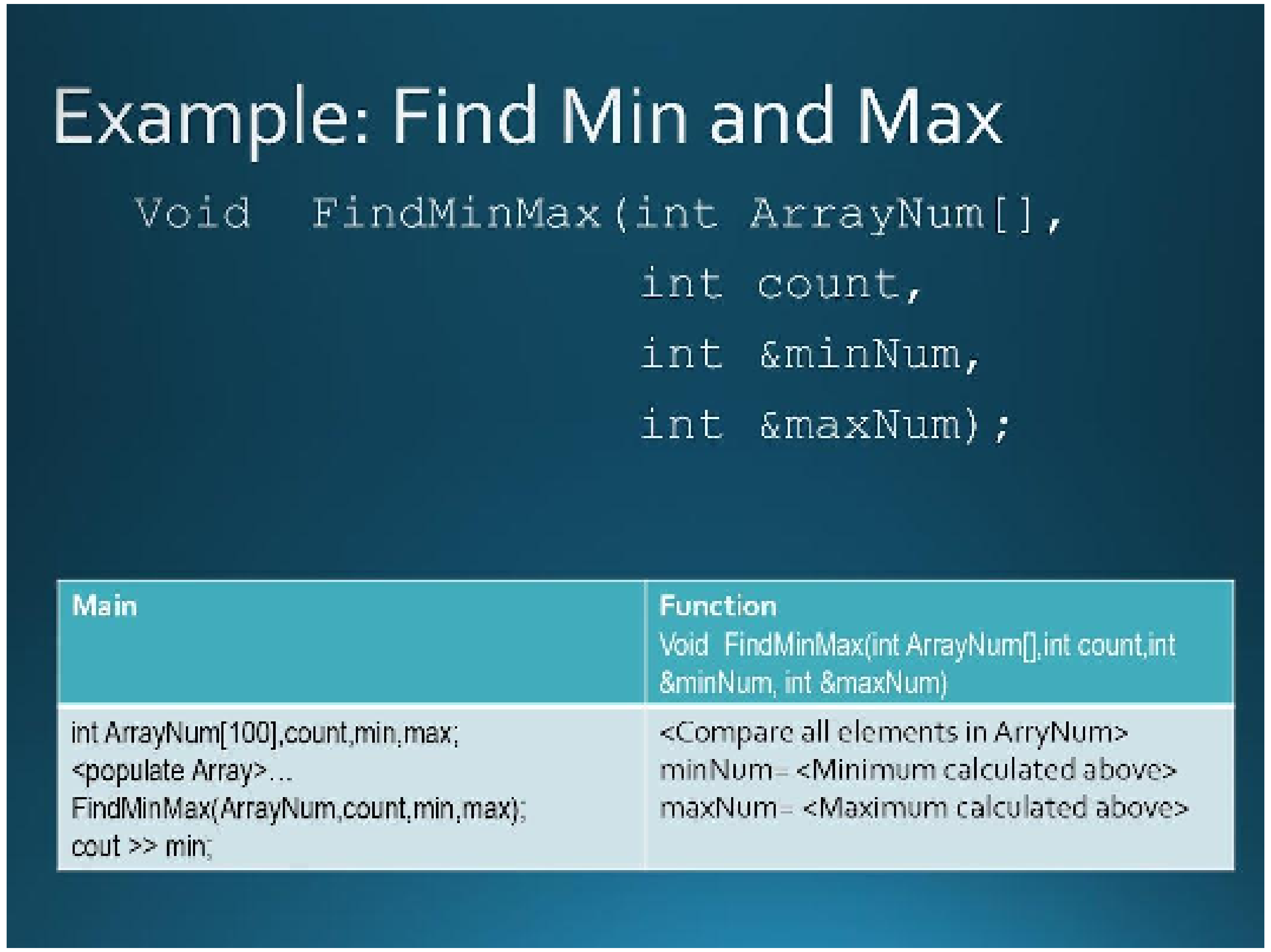
Procedure -Reference Parameters

- The address of the argument instead of its value is passed to the procedure.
- Reference parameters are used:
  - To send a large argument (for example, an array or a structure) to a procedure
  - To send results back to the calling program as argument values

Pass by Value

Array=[1,2,3,4]

→ Address

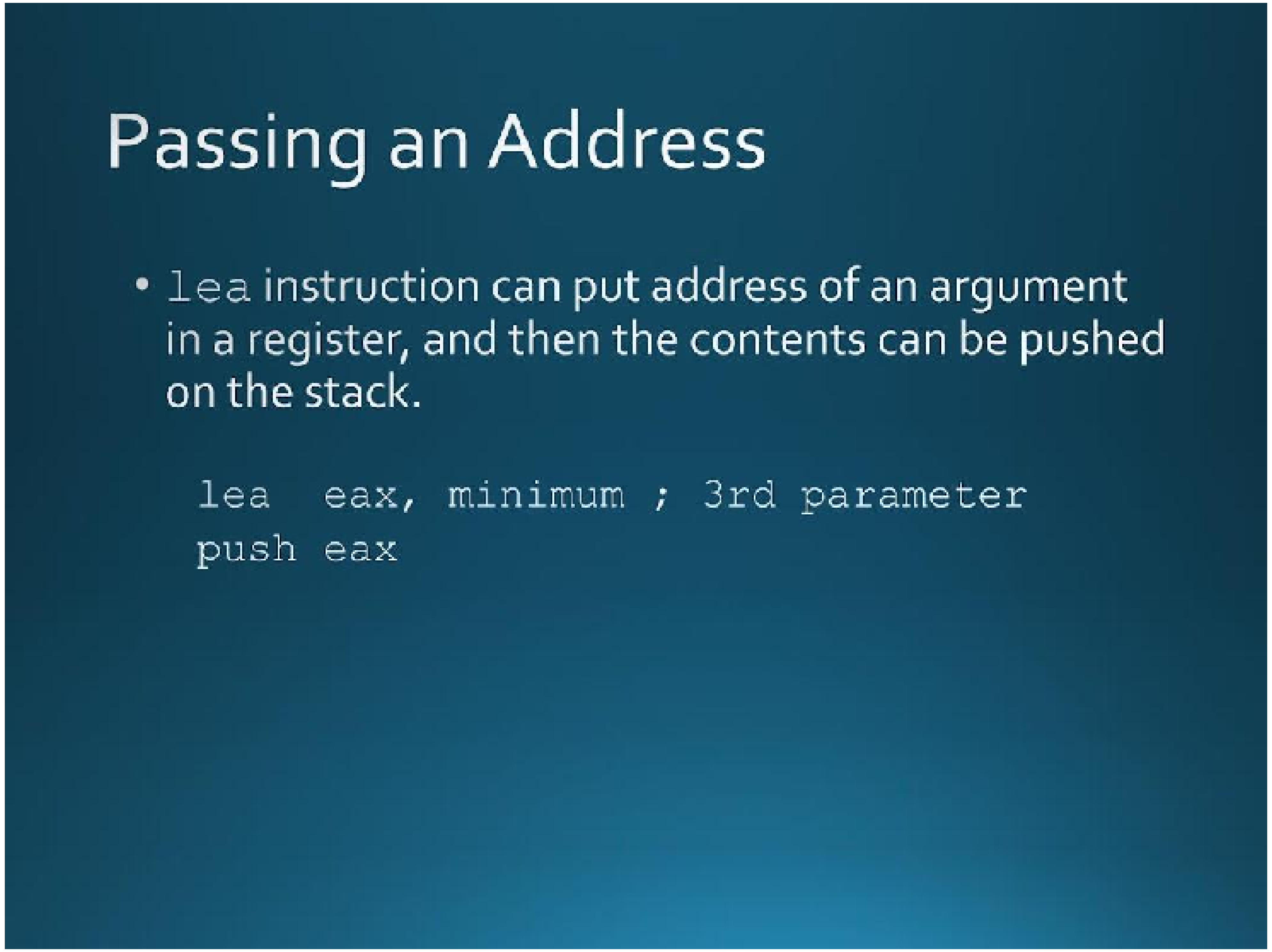


Example: Find Min and Max

```
Void  FindMinMax(int ArrayNum[],
                 int count,
                 int &minNum,
                 int &maxNum);
```

Main	Function Void FindMinMax(int ArrayNum[],int count,int &minNum, int &maxNum)

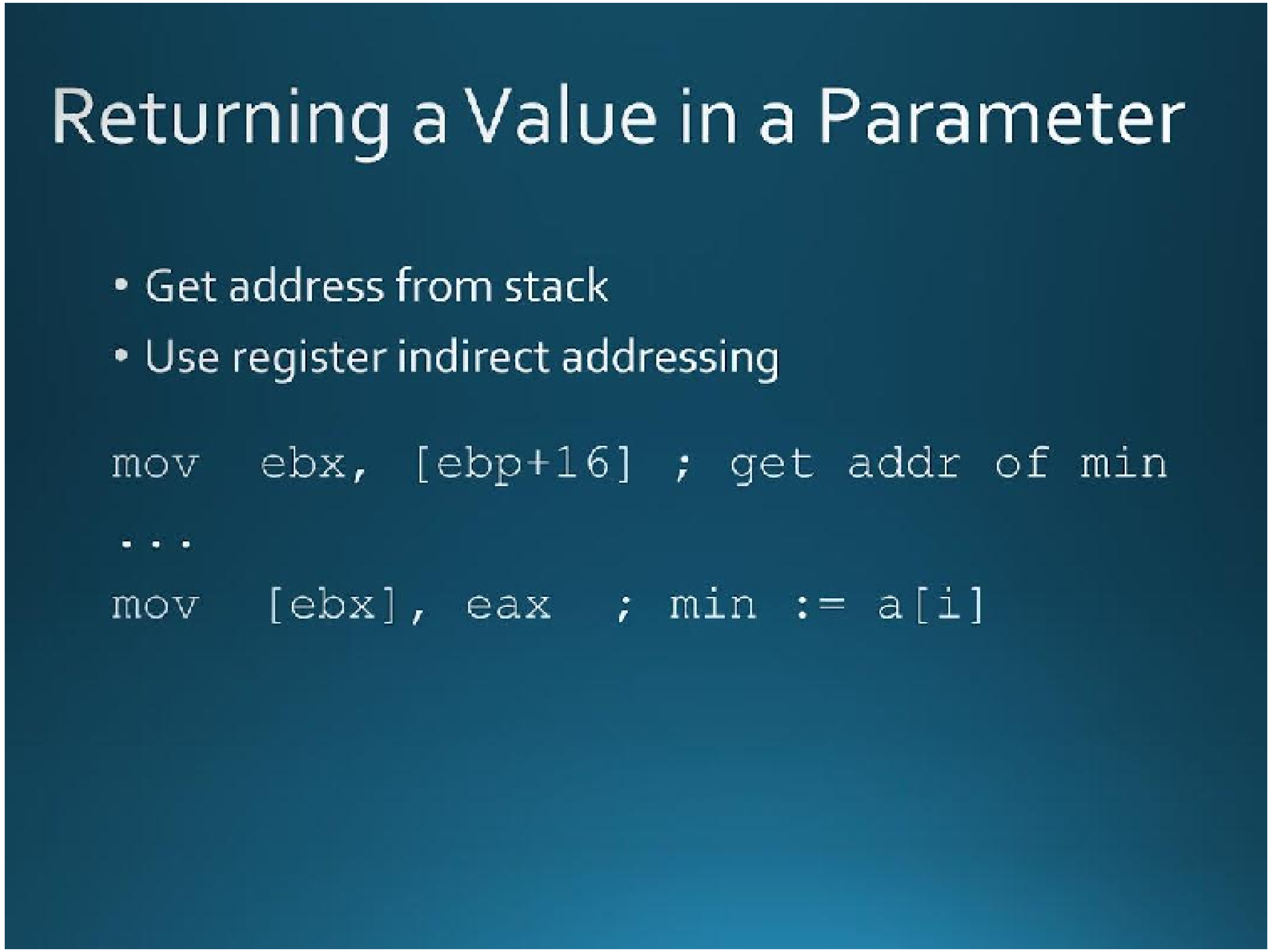
int ArrayNum[100],count,min,max;
<populate Array>…
FindMinMax(ArrayNum,count,min,max);
cout >> min;
	<Compare all elements in ArryNum>
	minNum= <Minimum calculated above>
	maxNum= <Maximum calculated above>



Passing an Address

- `lea` instruction can put address of an argument in a register, and then the contents can be pushed on the stack.

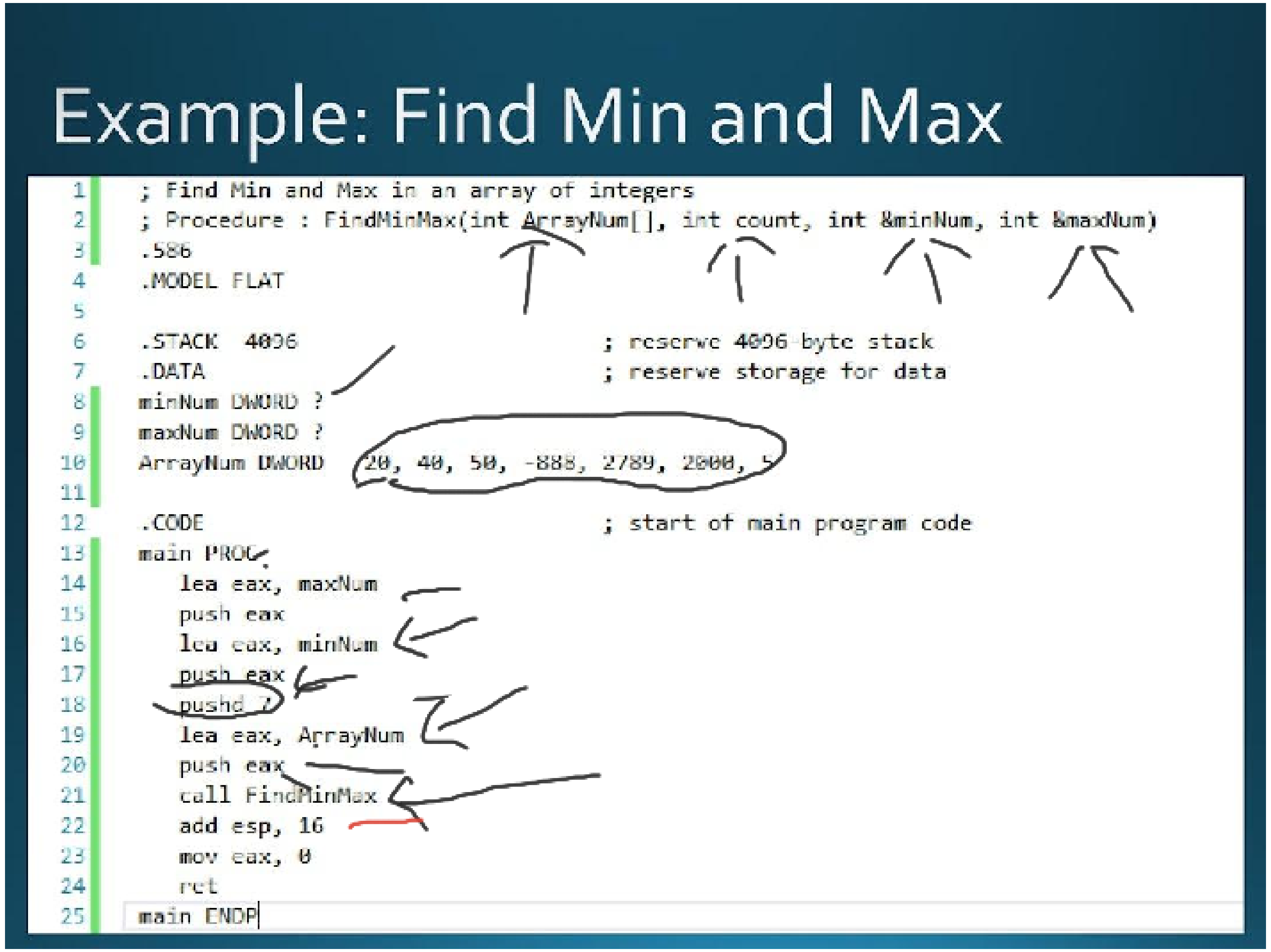
```
lea  eax, minimum ; 3rd parameter
push eax
```



Returning a Value in a Parameter

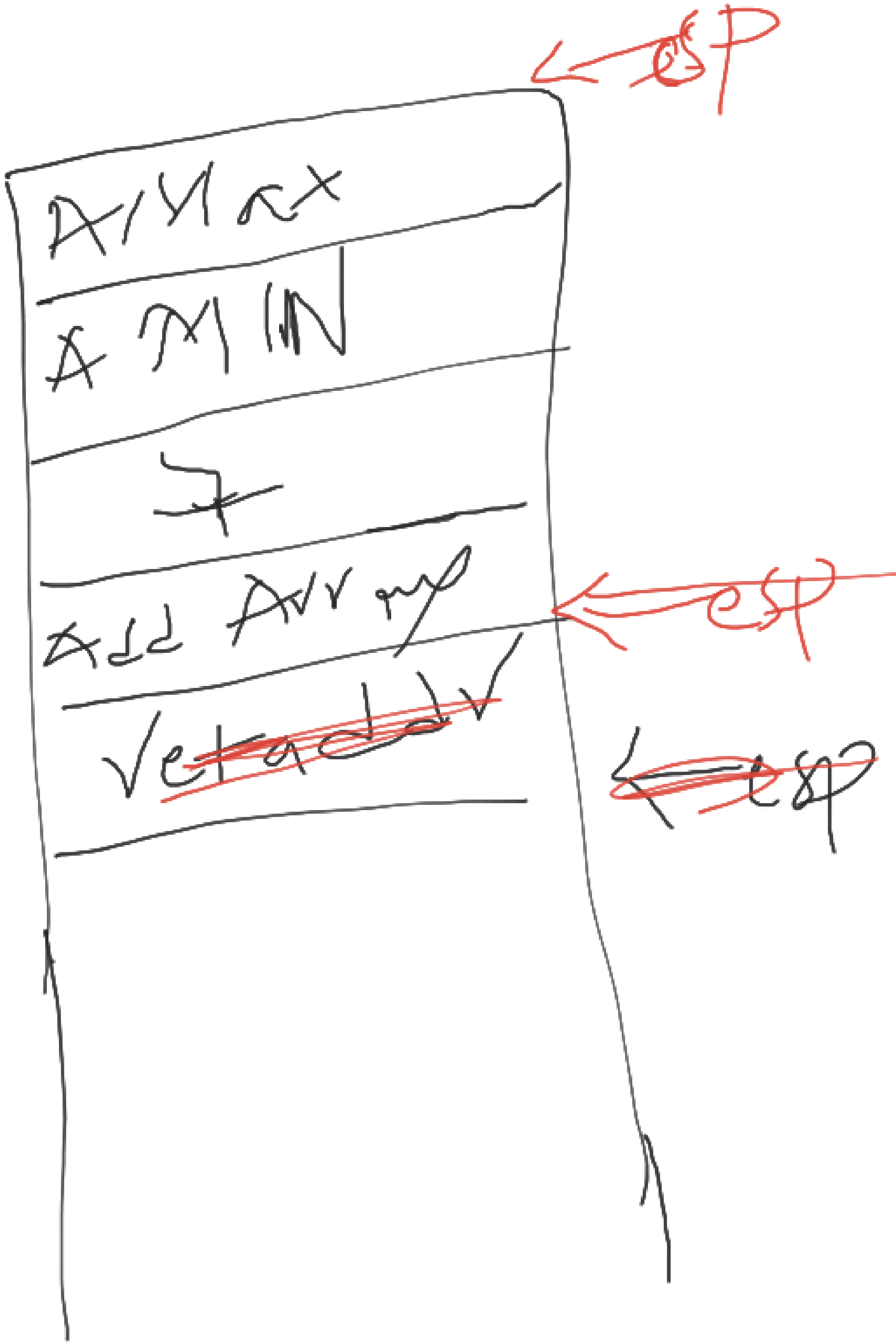
- Get address from stack
- Use register indirect addressing

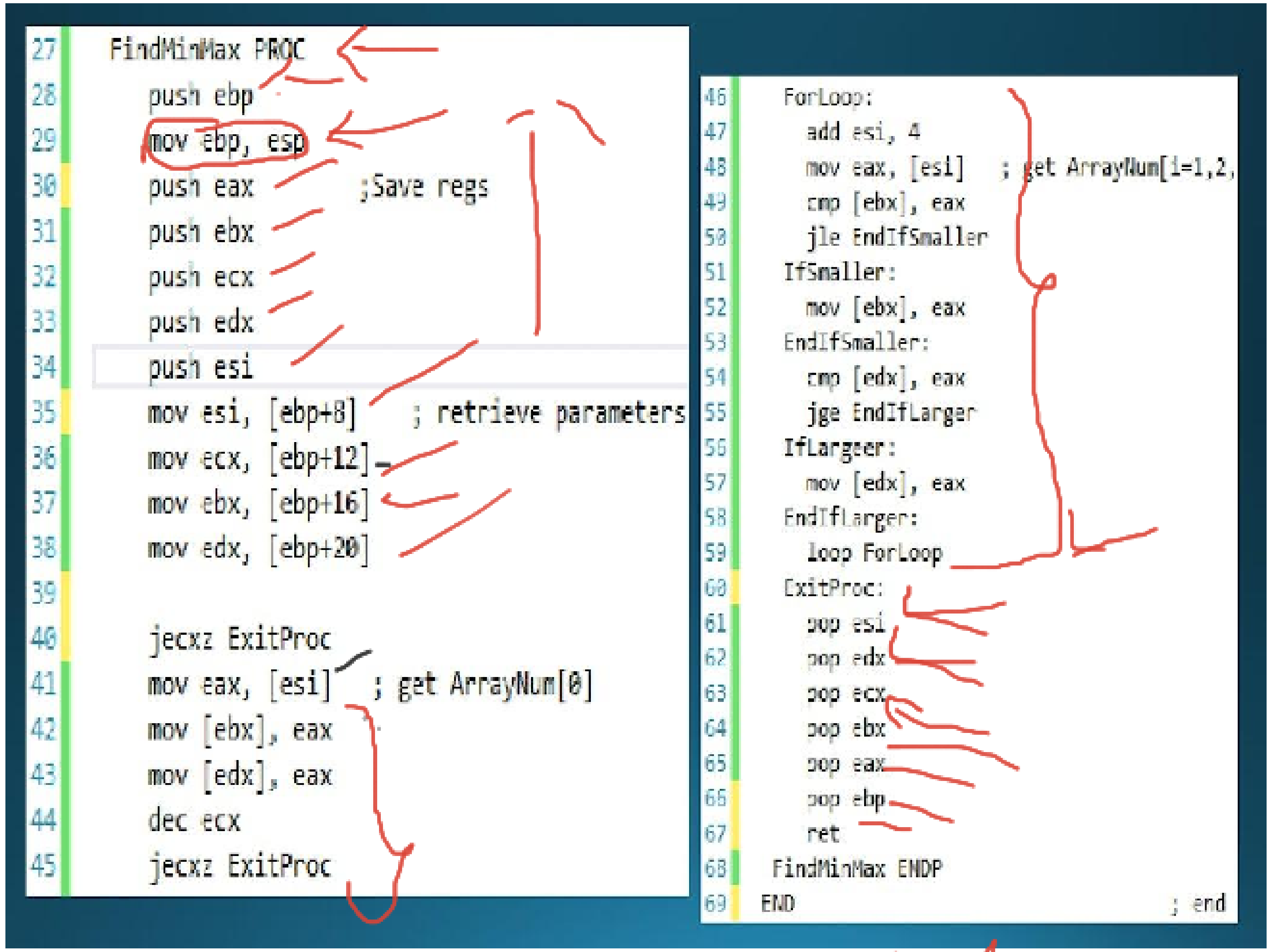
```
mov  ebx, [ebp+16] ; get addr of min
...
mov  [ebx], eax  ; min := a[i]
```



Example: Find Min and Max

```
; Find Min and Max in an array of integers
; Procedure : FindMinMax(int ArrayNum[], int count, int &minNum, int &maxNum)
.586
.MODEL FLAT

.STACK  4096                    ; reserve 4096-byte stack
.DATA                           ; reserve storage for data
minNum DWORD ?
maxNum DWORD ?
ArrayNum DWORD  20, 40, 50, -888, 2789, 2000, 5

.CODE                           ; start of main program code
main PROC
   lea eax, maxNum
   push eax
   lea eax, minNum
   push eax
   pushd 7
   lea eax, ArrayNum
   push eax
   call FindMinMax
   add esp, 16
   mov eax, 0
   ret
main ENDP
```





```
FindMinMax PROC
    push ebp
    mov ebp, esp
    push eax        ;Save regs
    push ebx
    push ecx
    push edx
    push esi
    mov esi, [ebp+8]     ; retrieve parameters
    mov ecx, [ebp+12]
    mov ebx, [ebp+16]
    mov edx, [ebp+20]

    jecxz ExitProc
    mov eax, [esi]    ; get ArrayNum[0]
    mov [ebx], eax
    mov [edx], eax
    dec ecx
    jecxz ExitProc
```

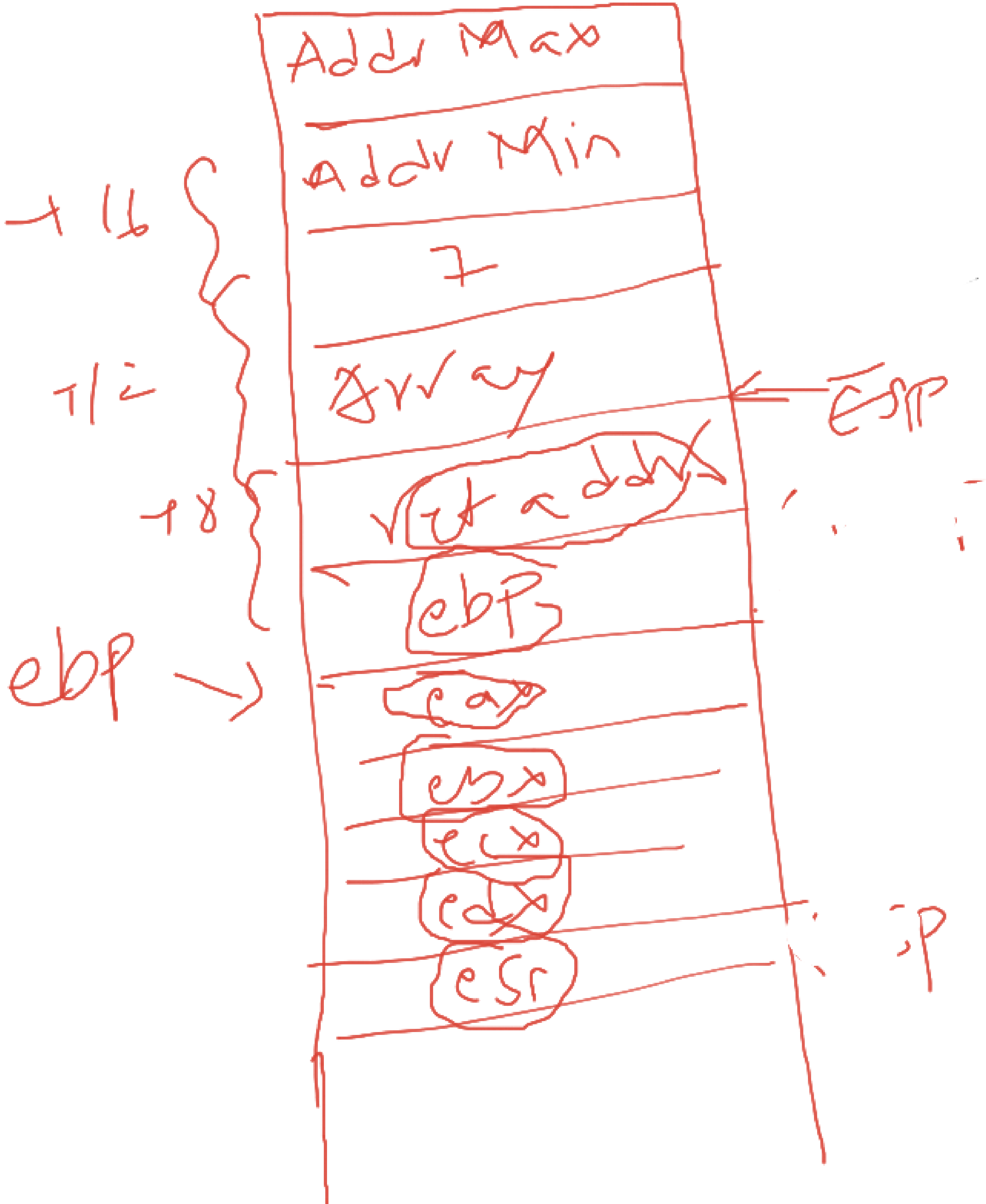
```
  ForLoop:
    add esi, 4
    mov eax, [esi]    ; get ArrayNum[i=1,2,
    cmp [ebx], eax
    jle EndIfSmaller
  IfSmaller:
    mov [ebx], eax
  EndIfSmaller:
    cmp [edx], eax
    jge EndIfLarger
  IfLargger:
    mov [edx], eax
  EndIfLarger:
    loop ForLoop
  ExitProc:
    pop esi
    pop edx
    pop ecx
    pop ebx
    pop eax
    pop ebp
    ret
 FindMinMax ENDP
END                                  ; end
```

esi = Addr of Array

eCX = 7

ebx = addr of Min

edx = addr of Max



```
-->min=arr[0]
--->max=arr[0]

 loop
increment i

if (a[i] < min)
{
min= a[i];
}
if (a[i]> max)
{
  max = a[i];
}

end loop
```

Arr={3, 2, 4, 5, 1,2,3}

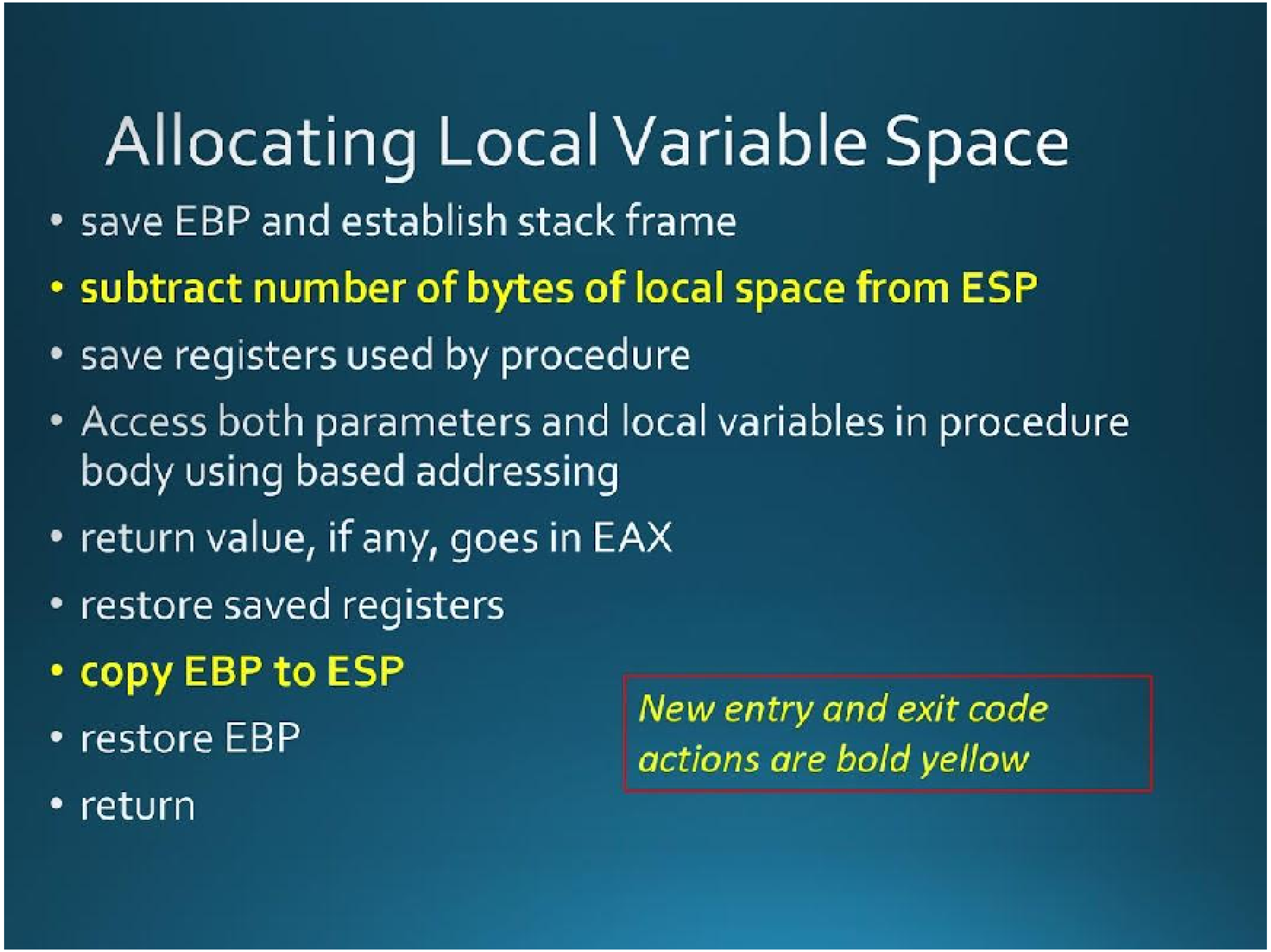
Esi = address of array, ecx=7, ebx=addr,min, edx= addr max

```
jecxz exitproc
mov eax,[esi]
mov [ebx],eax
mov [edx],eax
dec ecx

forloop:

jecxz exitproc
add esi,4
mov eax,[esi]
cmp [ebx],eax
jle Endifsmaller:
mov [ebx],eax
endifsmaller:
                    cmp [edx],eax
                     Jge Endifgreater
                      mov [edx],eax
                      Endifgreater:
                           loop forloop

Exitproc:
```

min=3
max=3

min<=eax
F
eax<min
min=eax

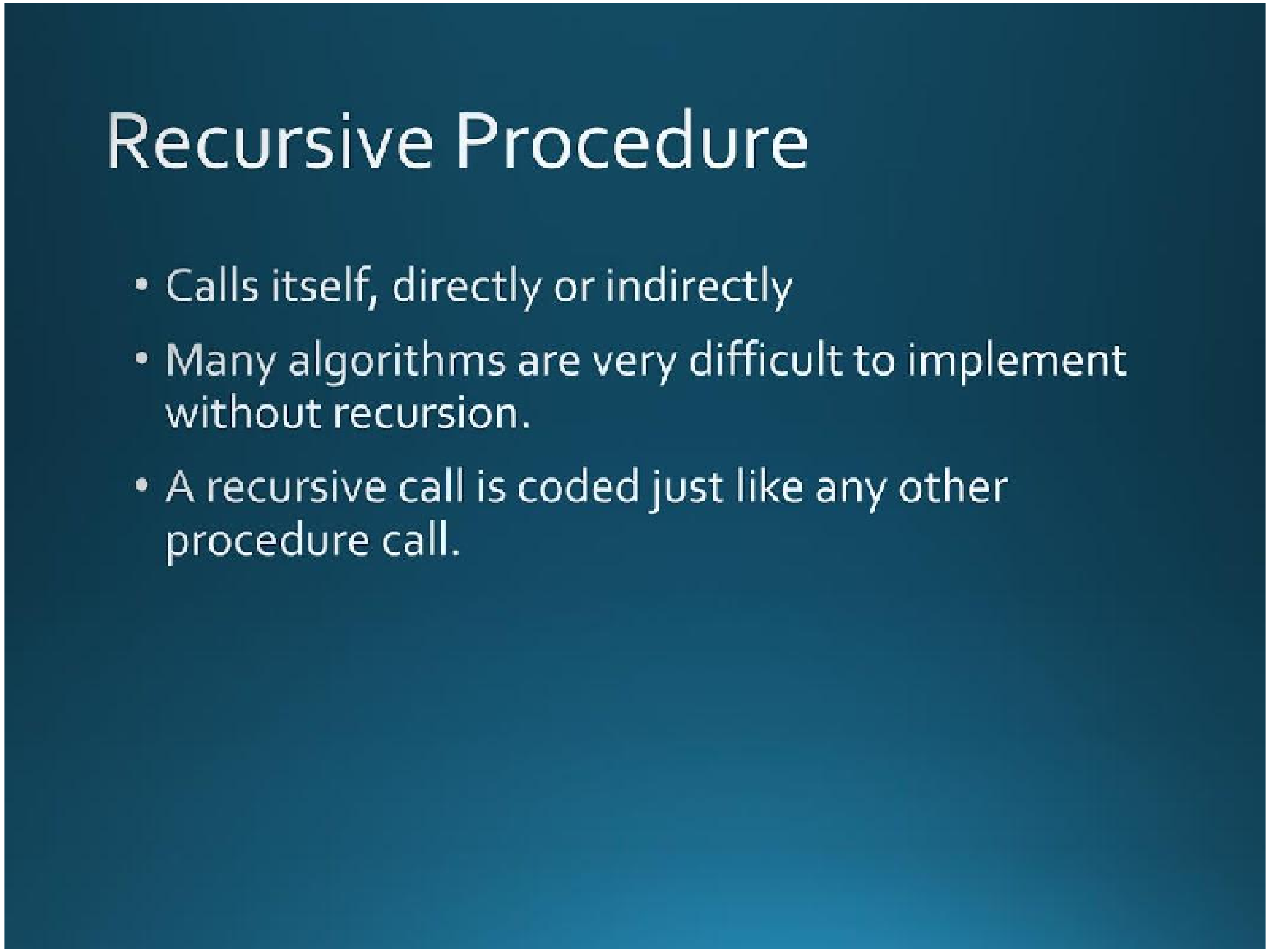
max>=eax
F
-->eax>max
max=eax



Allocating Local Variable Space

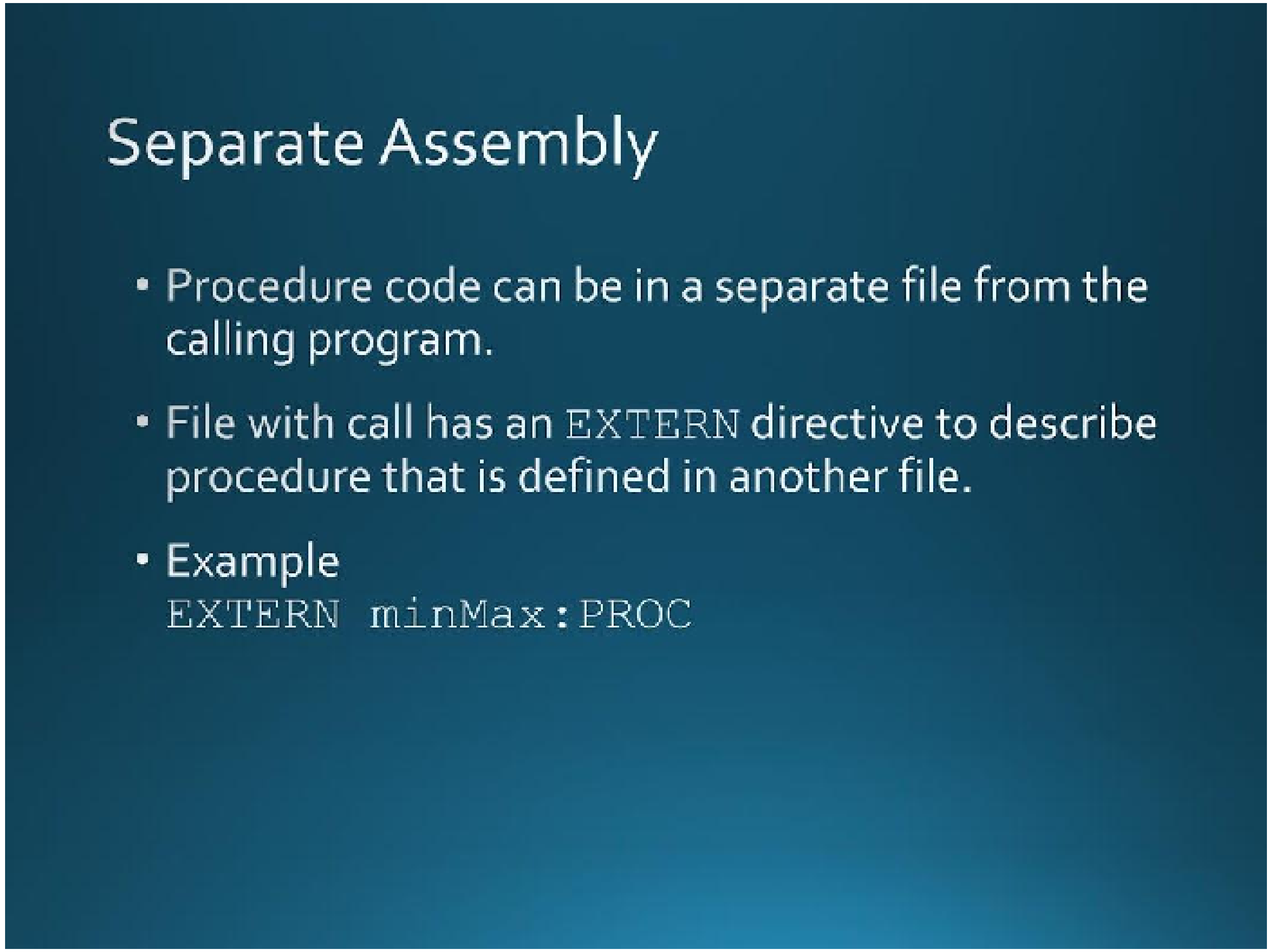
- save EBP and establish stack frame
- **subtract number of bytes of local space from ESP**
- save registers used by procedure
- Access both parameters and local variables in procedure body using based addressing
- return value, if any, goes in EAX
- restore saved registers
- **copy EBP to ESP**
- restore EBP
- return

New entry and exit code actions are bold yellow



Recursive Procedure

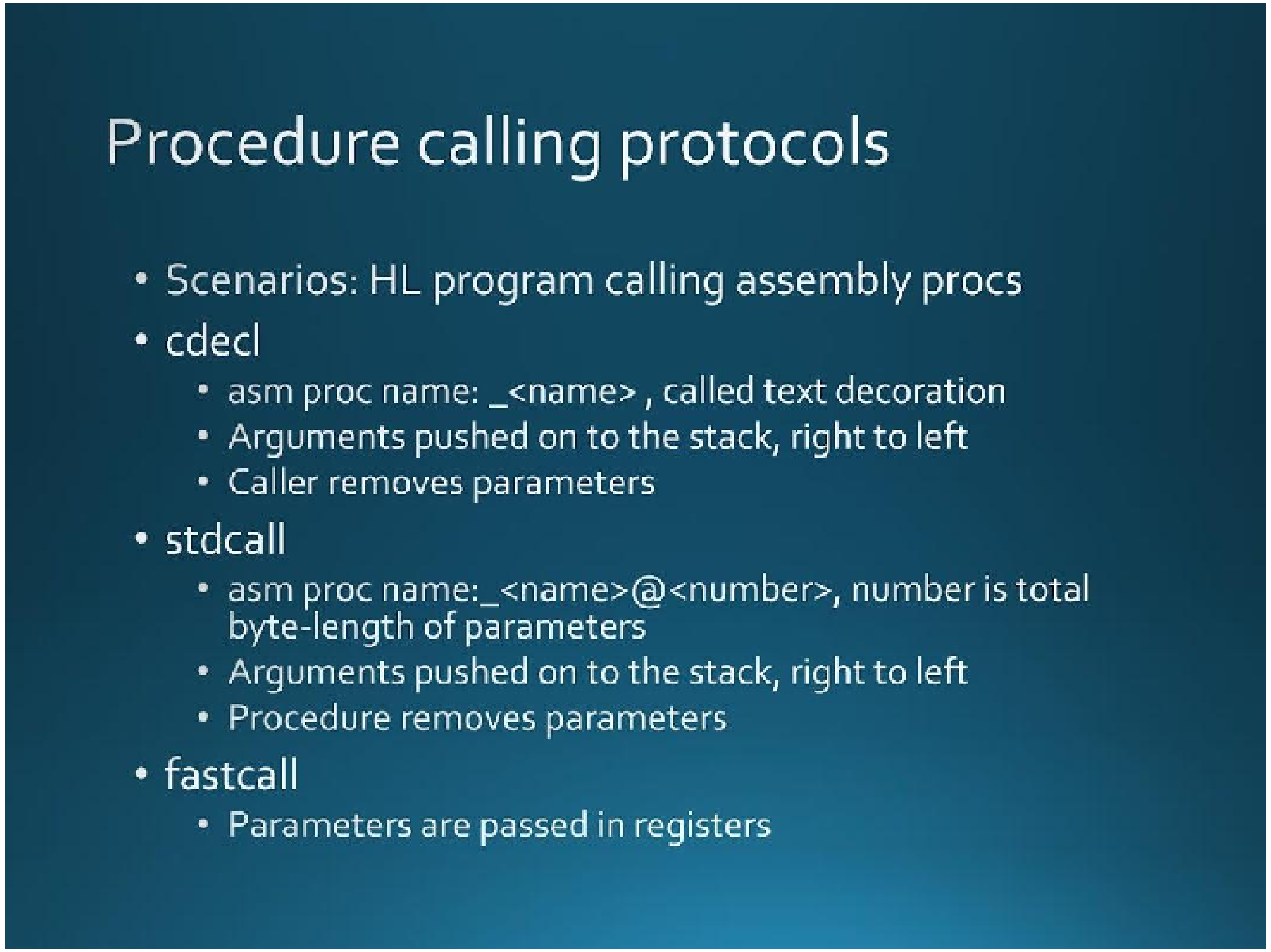
- Calls itself, directly or indirectly
- Many algorithms are very difficult to implement without recursion.
- A recursive call is coded just like any other procedure call.



Separate Assembly

- Procedure code can be in a separate file from the calling program.
- File with call has an `EXTERN` directive to describe procedure that is defined in another file.
- Example

```
EXTERN minMax:PROC
```



Procedure calling protocols

- Scenarios: HL program calling assembly procs
- cdecl
  - asm proc name: _<name> , called text decoration
  - Arguments pushed on to the stack, right to left
  - Caller removes parameters
- stdcall
  - asm proc name:_<name>@<number>, number is total byte-length of parameters
  - Arguments pushed on to the stack, right to left
  - Procedure removes parameters
- fastcall
  - Parameters are passed in registers

esi = index , ecx= 7, ebx= addr of min edx= addr of max

```
jecxz exitproc
mov eax,[esi]
mov [ebx],eax
mov [edx],eax
dec ecx

forloop:
jecxz exitproc
add esi,4
mov eax,[esi]
cmp [ebx], eax
jle endifsmaller

ifsmaller :    mov [ebx],eax
endifsmaller:
cmp [edx],eax
jge endifgreater
Ifgreater:
mov [edx],eax
Endifgreater:
loop forloop


exitproc:
```

Factorial recursive:

```
int factorial(int n)
{
   if (n == 0)
      return 1;
   return n * factorial(n - 1);
}


int main()
{
   int num = 5;
   cout << "Factorial of "
       << num << " is " << factorial(num) << endl;
   return 0;
}
```

Factorial Non recursive:

```
int factorial(int n)
{
   int res = 1, i;
   for (i = n; i >0; i--)
      res *= i;
   return res;
}

int main()
{
   int num = 5;
   cout << "Factorial of "
       << num << " is "
       << factorial(num) << endl;
   return 0;
}
```