Computer Organization and Assembly Language (EL 229)

Outline

- **Advance Procedures**
- **Stack Frames**
- **INVOKE, ADDR, PROC, and PROTO**

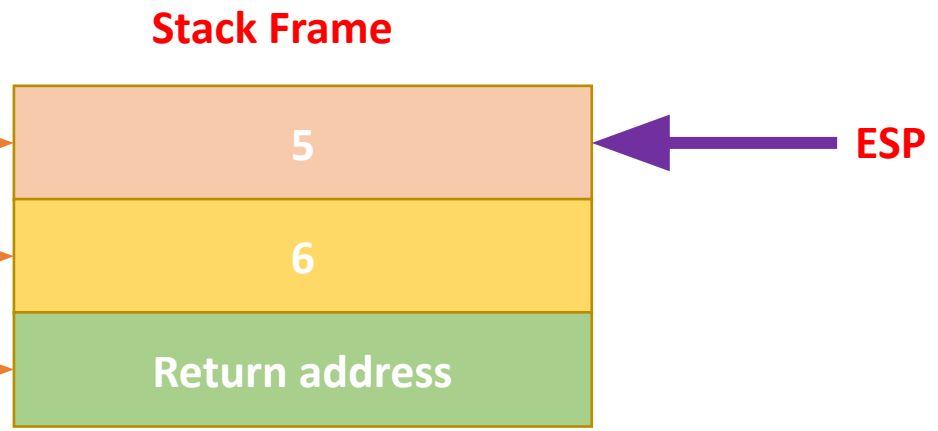
Types of Arguments

- Two general types of arguments are **pushed on the stack** during subroutine calls:
  - **Value arguments** (values of variables and constants)
  - **Reference arguments** (addresses of variables)

Pass By Value

- When an argument is passed by value , **a copy of the value is pushed on the stack.**
- Suppose we call a subroutine named **AddTwo**, passing it **two 32-bit integers**

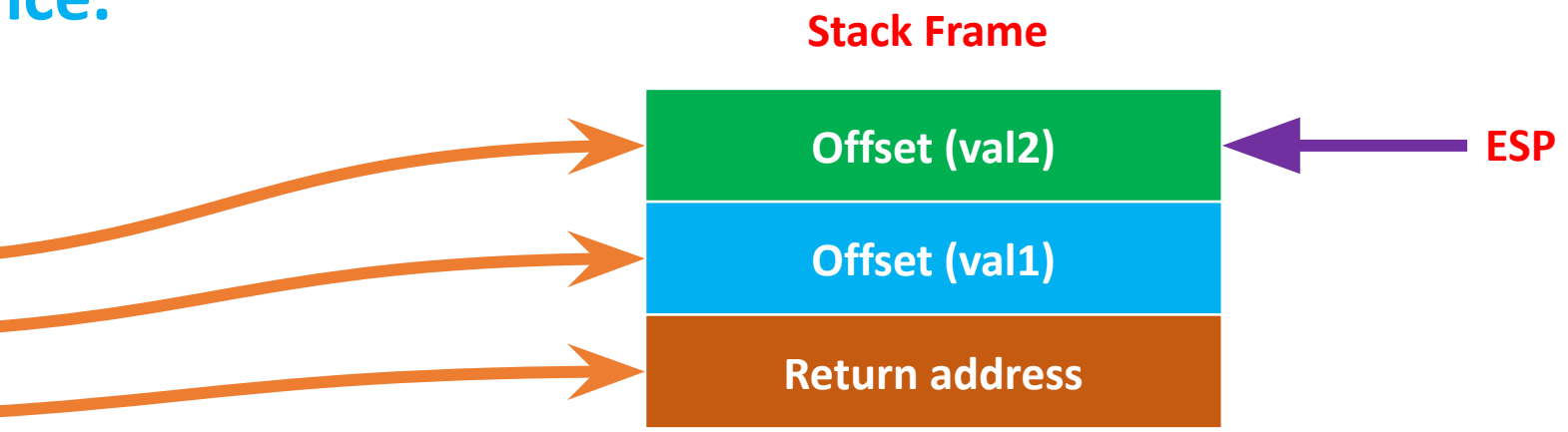
```
val1 dword 5
val2 dword 6

push val1
push val2

call AddTwo
```



Pass By Reference

- An argument passed by reference consists of **the address (offset) of an object.**
- Suppose we call a subroutine named **Swap**, passing it **two 32-bit integers by reference.**

```
val1 dword 5
val2 dword 6

push OFFSET val2
push OFFSET val1
call Swap
```



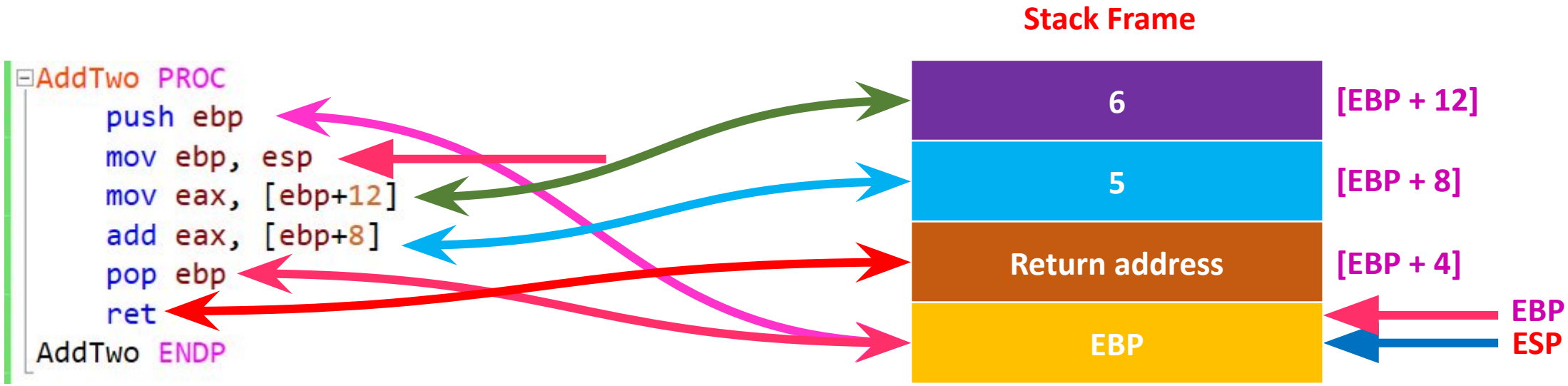
Passing Arrays

- High-level languages always **pass arrays to subroutines by reference**. That is, they push the address of an array on the stack. The subroutine can then **get the address from the stack** and **use it to access the array**.

```
arr dword 1, 2, 3, 4, 5

push OFFSET arr
call SumArr
```

Accessing Stack Parameters

- Use **base-offset addressing** to access stack parameters.
- **EBP is the base register** and the **offset** is a constant.



Cleaning Up the Stack

- There must be a **way for parameters** to be **removed from the stack** when a **subroutine returns**. Otherwise, a **memory leak** would result, and the **stack would become corrupted**.

STDCALL Calling Convention

```
push val2
push val1
call AddTwo
add esp, 8
```

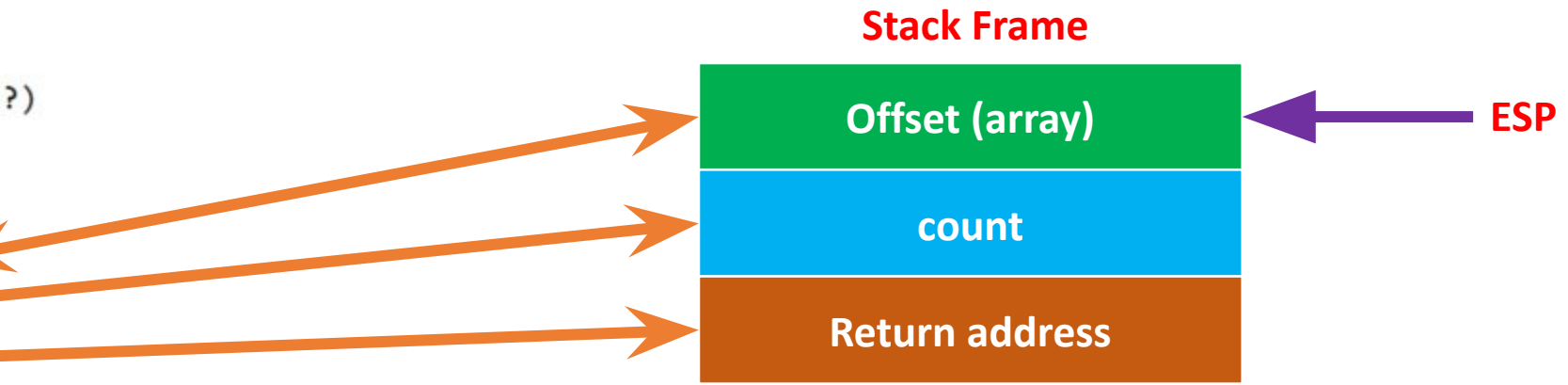

```
AddTwo PROC
    push ebp
    mov ebp, esp
    mov eax, [ebp+12]
    add eax, [ebp+8]
    pop ebp
    ret 8
AddTwo ENDP
```

Accessing Reference Parameters

- Reference parameters are usually accessed by subroutines using **base-offset addressing (from EBP).**
- Because each **reference parameter is a pointer**, it is usually loaded into a register for use as an **indirect operand**.
- Suppose, for example, that **a pointer to an array is located at stack address [ebp+12]**. The following statement **copies the pointer into ESI:**
- mov **esi,[ebp+12]** ; points to the array

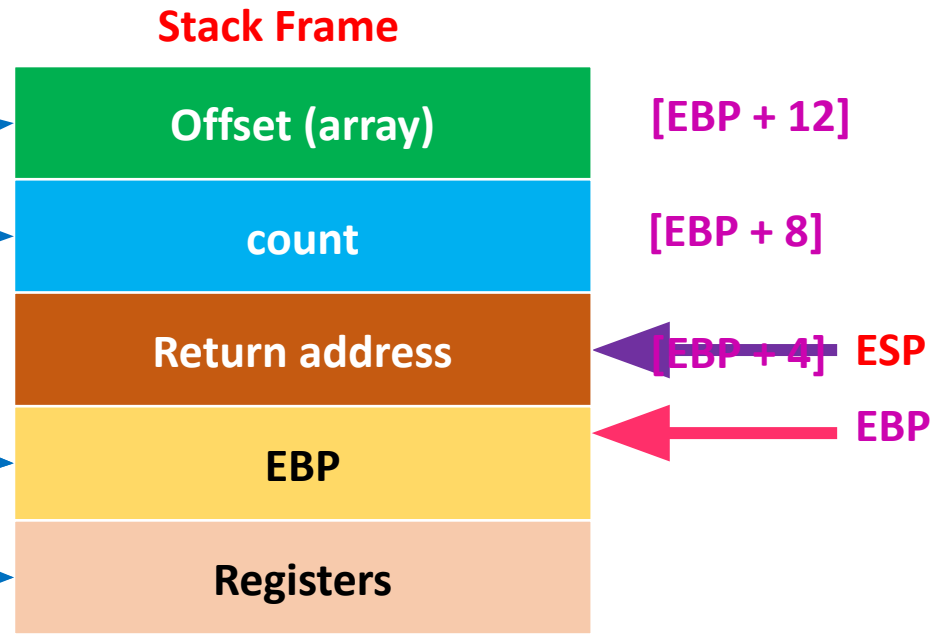
Accessing Reference Parameters

```
INCLUDE Irvine32.inc
.data
    count = 100
    array word count DUP(?)
.code
main PROC

    push offset array
    push count
    call ArrayFill

call crlf
call waitmsg
exit
main ENDP
```



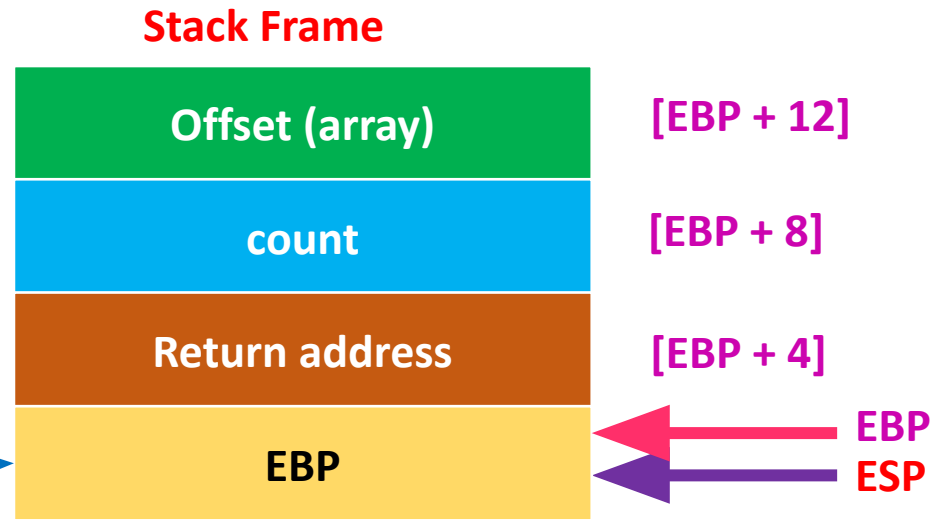
Accessing Reference Parameters

```
ArrayFill PROC
    push ebp
    mov ebp, esp
    pushad
    mov esi, [ebp + 12]
    mov ecx, [ebp + 8]
    cmp ecx, 0
    je L2
    L1:
        mov eax, 1000h
        call RandomRange
        mov [esi], ax
        add esi, TYPE WORD
        loop L1
    L2:
        popad
        pop ebp
        ret 8
ArrayFill ENDP
```



Accessing Reference Parameters

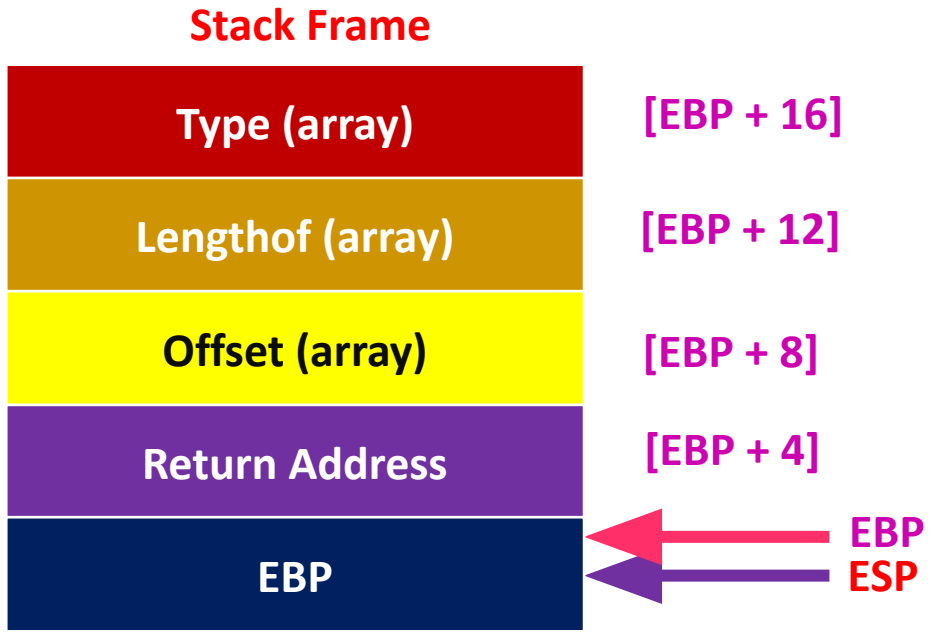
```
ArrayFill PROC
    push ebp
    mov ebp, esp
    pushad
    mov esi, [ebp + 12]
    mov ecx, [ebp + 8]
    cmp ecx, 0
    je L2
    L1:
        mov eax, 1000h
        call RandomRange
        mov [esi], ax
        add esi, TYPE WORD
        loop L1
    L2:
        popad
        pop ebp
        ret 8
ArrayFill ENDP
```



DumpArray Procedure

```
array dword 2, 4, 5, 8, 9

push TYPE array
push LENGTHOF array
push OFFSET array
call DumpArray
```

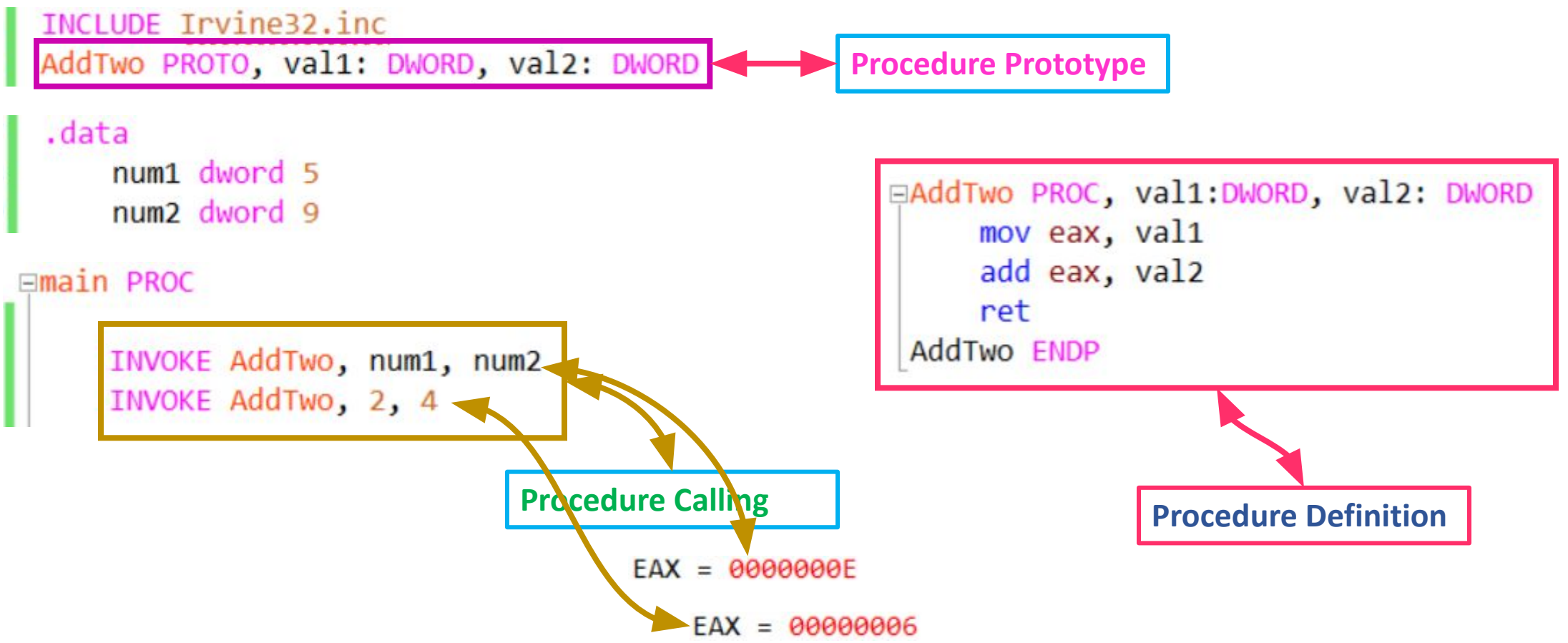
```
DumpArray PROC
    push ebp
    mov ebp, esp
    mov ecx, [ebp + 12]
    mov esi, [ebp + 8]
    L1:
        mov eax, [esi]
        call WriteDec
        call Crlf
        add esi, [ebp + 16]
        loop L1
    pop ebp
    ret 12
DumpArray ENDP
```



INVOKE, ADDR, PROC, and PROTO

- The **INVOKE**, **ADDR**, **PROC**, and **PROTO** directives provide **powerful tools for defining and calling procedures**.
- In many ways, they approach the convenience offered by **high-level programming languages**.
- **Read more about it in the book, lets see the examples.**

INVOKE, ADDR, PROC, and PROTO Example

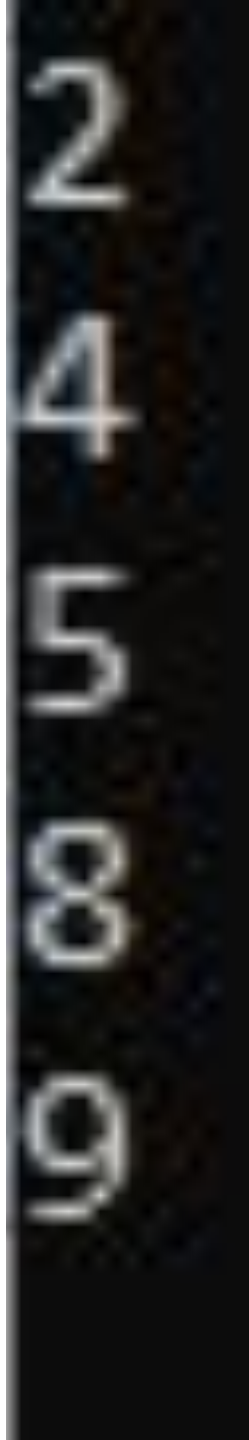


INVOKE, ADDR, PROC, and PROTO Example

```
INCLUDE Irvine32.inc
DumpArray PROTO, arrAddress: PTR DWORD, arrSize: DWORD

main PROC

    INVOKE DumpArray, ADDR array, LENGTHOF array
```

```
.data
    array dword 2, 4, 5, 8, 9
```

```
DumpArray PROC USES eax ecx esi, arrAddress: PTR DWORD, arrSize: DWORD
    mov ecx, arrSize
    mov esi, arrAddress
    L1:
        mov eax, [esi]
        call WriteDec
        call Crlf
        add esi, 4
        loop L1
    ret
DumpArray ENDP
```



References

- Assembly Language For x86 Processors 6th edition by Kip R. Irvine
  - **8.1 Introduction**
  - **8.2 Stack Frames**
  - **8.4 INVOKE, ADDR, PROC, and PROTO**