Stack Frame for Median Procedure

Caller

```
push len
push OFFSET array
call median
mov  med, EAX
```

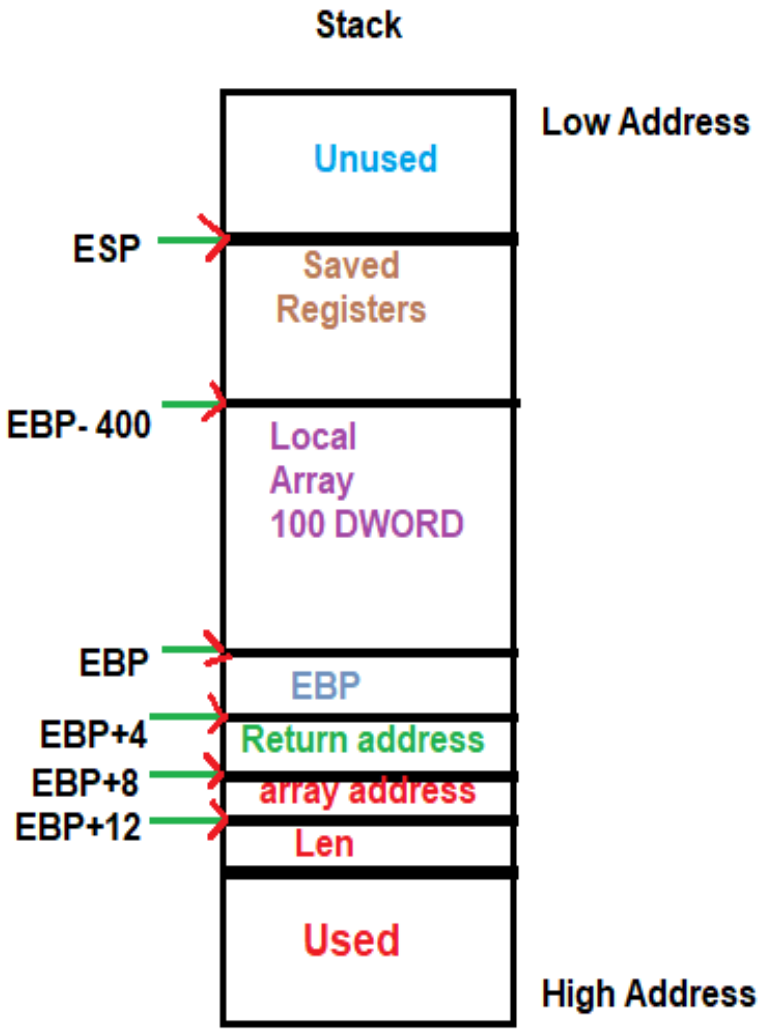
```
median PROC
  push EBP
  mov  EBP, ESP
  sub  ESP, 400
  push {regs}
  . . .
  pop  {regs}
  mov  ESP, EBP
  pop  EBP
  ret  8
median ENDP
```

sub ESP, 400

Allocates 400 bytes on the stack for the local array

Puts a limit on the maximum array length = 100 int

[EBP – 400] is the address of local array on the stack

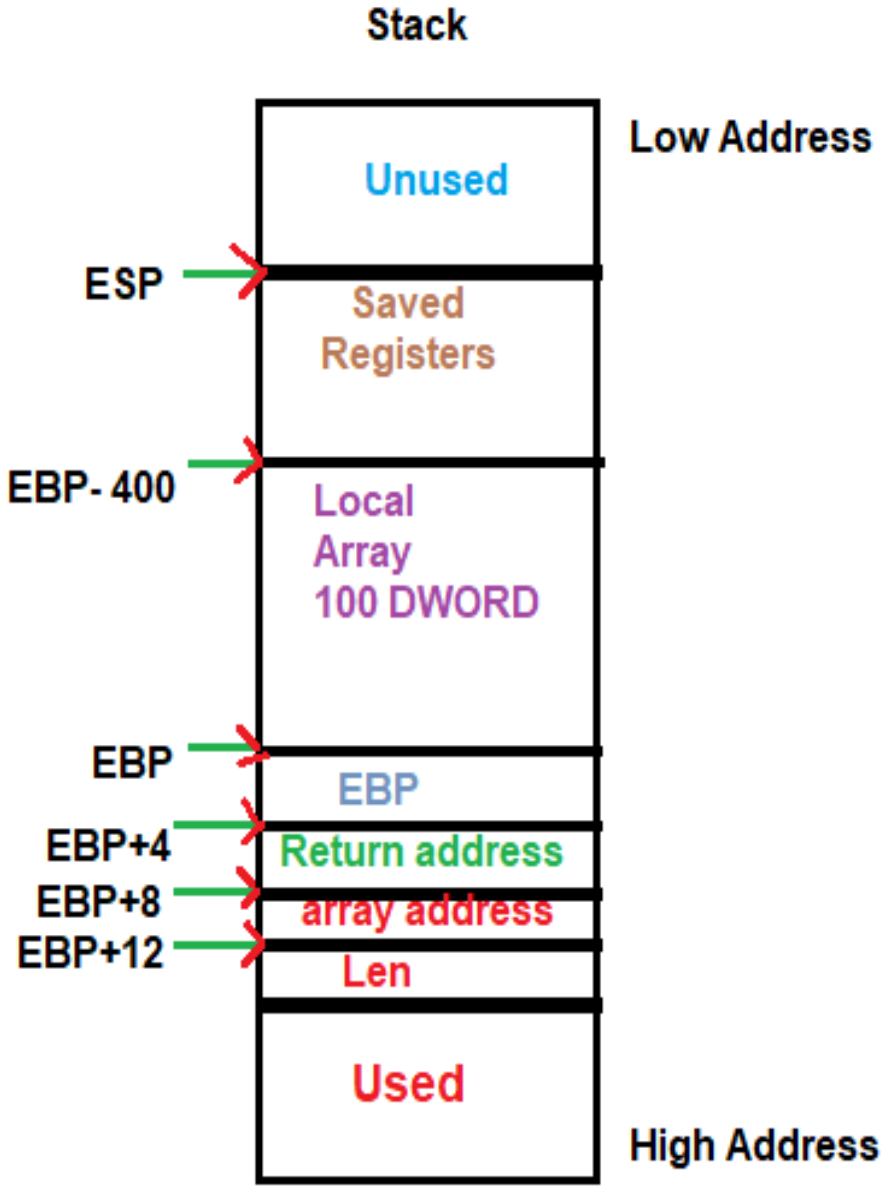


Median Procedure – slide 1 of 2

```
median PROC
  push  EBP              ; save EBP
  mov   EBP, ESP         ; EBP = ESP
  sub   ESP, 400         ; local array
  push  ECX
  push  ESI
  push  EDI
  mov   ECX, [EBP+12]    ; len
  mov   ESI, [EBP+8]     ; array addr
  lea   EDI, [EBP-400]   ; local addr
L1:
  mov   EAX, [ESI]
  mov   [EDI], EAX
  add   ESI, 4
  add   EDI, 4
  loop  L1
```

Save Regs

Copy array



Median Procedure – slide 2 of 2

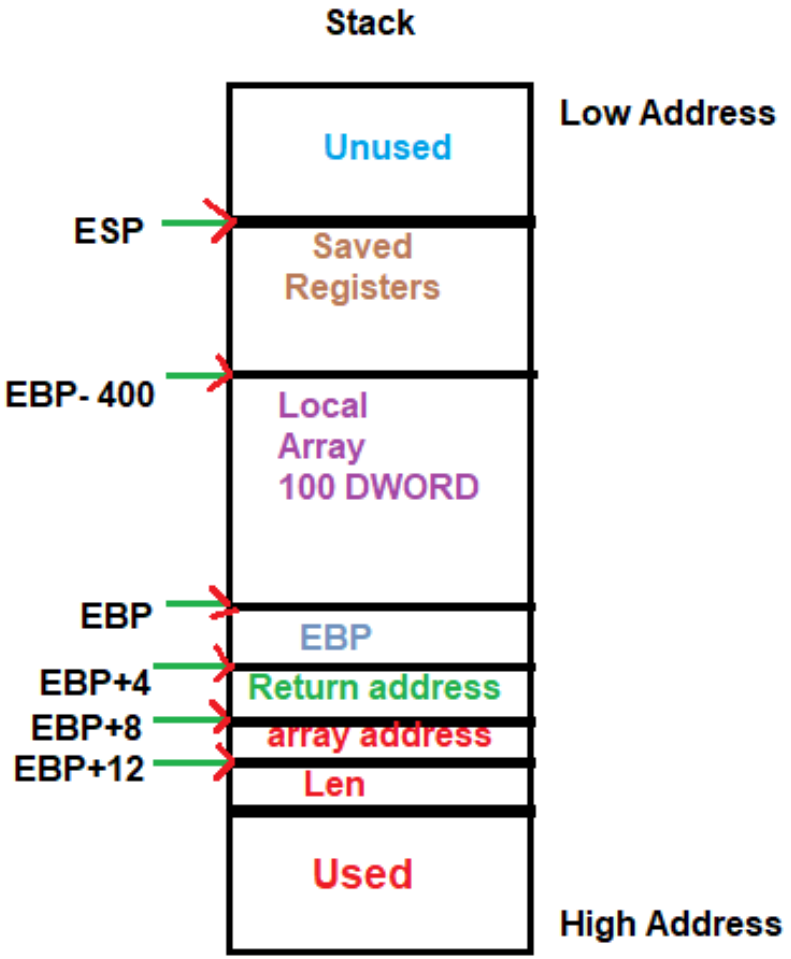
```
; Call sort procedure to sort local array
; Parameters are passed on the stack
  push DWORD PTR [EBP+12]
  lea  EDI, [EBP-400]
  push EDI   ; local array address
  call sort  ; sort local array

  mov  ESI, [EBP+12]     ; len
  shr  ESI, 1            ; ESI = len/2
  mov  EAX, [EDI+ESI*4] ; local[len/2]

  pop  EDI
  pop  ESI
  pop  ECX

  mov  ESP, EBP          ; free local
  pop  EBP               ; restore EBP
  ret  8                 ; return & cleanup
median ENDP
```

Restore Regs



Practice Question 1:

Create a procedure named **Swap** that exchanges the contents of two 32-bit integers. Swap receives two input–output parameters named pValX and pValY, which contain the addresses of data to be exchanged. The two parameters in the Swap procedure, pValX and pValY, are input–output parameters. Their existing values are input to the procedure, and their new values are also output from the procedure. Write a test program that displays the array before the exchange then calls the procedure Swap and finally displays the array after exchange.

Data to be exchanged: **Array DWORD 10000h, 20000h**

One possible Solution:

```
INCLUDE Irvine32.inc
.data
Array DWORD 10000h,20000h
.code
main PROC
 ; Display the array before the exchange:
 mov esi, OFFSET Array
 mov ecx, 2                    ; count = 2
 mov ebx, TYPE Array
 call DumpMem                  ; dump the array values
 PUSH OFFSET [Array +4]        ;pValY
 PUSH OFFSET Array             ;pValX
 CALL Swap
 ; Display the array after the exchange:
 mov esi, OFFSET Array
 mov ecx, 2                    ; count = 2
 mov ebx, TYPE Array
 call DumpMem
 exit
main ENDP
```

```
;-----------------------------------------------------
Swap PROC
;-----------------------------------------------------
push ebp
mov ebp, esp
 mov esi, [ebp+8]     ; get pointer pValX
 mov edi, [ebp +12]  ; get pointer pValY
 mov eax, [esi]          ; get first integer
 xchg eax, [edi]         ; exchange with second
 mov [esi], eax          ; replace first integer
 mov esp,  ebp
 pop ebp
 ret 8                       ; Clean the stack
Swap ENDP
END main
```

Practice Question 2:

Create a procedure named **ArraySum** that sums the element of a doubleword array. ArraySum receives two parameters: a pointer to a unsigned doubleword array, and a count of the array's length. ArraySum returns the result in EAX. Write a test program that calls the procedure ArraySum and then stores the result in a doubleword variable named theSum.

Sample array : Array DWORD 10, 20, 30, 40, 50
theSum DWORD ?

One possible Solution

```
INCLUDE Irvine32.inc
.data
array DWORD 10,20,30,40,50
theSum DWORD ?
.code
main PROC
 mov ebx, LENGTHOF array
 PUSH ebx
 PUSH OFFSET array
 CALL ArraySum
 mov theSum, eax            ; store the sum
 call DumpRegs
 call WriteDec
 exit
 main ENDP
 ArraySum PROC
 push ebp
 mov ebp,esp
 push ecx
 push esi
```

```
mov esi, [ebp + 8]          ; address of the array
 mov ecx, [ebp +12]         ; size of the array
 mov eax,0                  ; set the sum to zero
 cmp ecx,0                  ; length = zero?
 je L2                      ; yes: quit
L1: add eax, [esi]          ; add each integer to sum
 add esi, 4                 ; point to next integer
 loop L1                    ; repeat for array size
L2:
pop esi
pop ecx
mov esp, ebp
pop ebp
ret 8
ArraySum ENDP
END main
```

Practice Question 3:

Create the **ArrayFill** procedure, which fills an array with a pseudorandom sequence of numbers. It receives four arguments: a pointer to the array, the array length, the max possible random value (assuming 0 is the minimum) and the array type. The first is passed by reference and the others are passed by value. Write a test program that calls ArrayFill once and fills an array.

One possible Solution:

```
INCLUDE Irvine32.inc
.data
count = 10
array WORD count DUP(0)
.code
main PROC
call Randomize
push type array
push  100
push  count
push  OFFSET array
call ArrayFill
mov esi, OFFSET array
mov ecx, LENGTHOF array
mov ebx, TYPE array
call DumpMem
exit
main ENDP
```

```
ArrayFill PROC
 push ebp
 mov ebp, esp
 pushad                    ; save registers
 mov esi, [ebp+8]          ; offset of array
 mov ecx, [ebp+12]         ; array length
L1:
 mov eax, [ebp+16]         ; get random number from 0-99(n-1)
 call RandomRange          ; from the link library
 mov [esi], ax             ; insert value in array
 add esi, TYPE word        ; move to next element
 loop L1
L2: popad                  ;  restore registers
mov esp,ebp
 pop ebp
 ret 16                    ; clean up the stack
ArrayFill ENDP
END main
```

Practice Question 4:

Create the **ArrayFillandSum** procedure, which fills a local variable that is an array of 5 words(16-bit integer) with a pseudorandom sequence of integers, displays them and then adds those numbers and returns the result in EAX. It receives one argument: the max possible random value (assuming 0 is the minimum). Write a test program that calls ArrayFillandSum once and saves the sum in a Dword variable Sum.

One possible Solution:

```
INCLUDE Irvine32.inc
.data
Maxrange = 100
Sum Dword ?
.code
main PROC
call Randomize
push Maxrange
call ArrayFillandSum
mov Sum, eax
call DumpRegs
exit
main ENDP
```

```
ArrayFillandSum PROC
 push ebp
 mov ebp,esp
 sub esp, 12              ; ESP decremented by 12 to align with doubleword boundary
 lea esi, [ebp-10]        ; load address of array
 mov ecx, 5               ; array length
L1:
 mov eax,[ebp+8]          ; get random number from 0 to (n-1)
 call RandomRange         ; from the link library
 mov [esi],ax             ; insert value in array
```

```
add esi,TYPE word       ; move to next element
 loop L1
  lea esi, [ebp-10]
   mov edx,0
   mov ecx, 5
   mov eax, 0
L2:
 mov ax, [esi]
 add edx, eax
 call WriteDec
 call crlf
 add esi, TYPE word     ; move to next element
 loop L2
 mov eax, edx
 mov esp, ebp           ; clean the locals
 pop ebp
 ret 4                  ; clean up the stack
 ArrayFillandSum ENDP
 END main
```

Practice Question 5:

Write a procedure named **Sumarrayelements** that receives pointers to three arrays of unsigned byte, word and doubleword respectively, and a fourth parameter that indicates the length of the three arrays. The procedure adds each element xi in the first array(byte type) to the corresponding yi in the second array(word type) and store the result in zi which is the ith element of third array(dword). Write a test program that calls your procedure and passes the pointers to three different arrays and length of those arrays.

Sample Variables:

- **Arr1 byte 2, 23, 45, 75, 23**
- **Arr2 word 3, 100, 720, 350, 6**
- **Arr3 Dword LENGTHOF Arr1 Dup(?)**

One possible solution:

```
INCLUDE Irvine32.inc
.data
Arr1 byte   2, 23, 45, 75, 23
Arr2 word   3, 100, 720, 350, 6
Arr3 Dword LENGTHOF Arr1 Dup(?)
.code
main PROC
push LENGTHOF Arr1
push OFFSET Arr3
push OFFSET Arr2
push OFFSET Arr1
call SumArrayElements
mov esi, OFFSET Arr3
mov ecx, LENGTHOF Arr3
mov ebx, TYPE Arr3
call DumpMem
```

```
exit
main ENDP
SumArrayElements PROC
 push ebp
 mov ebp,esp
pushad
mov esi, [ebp +8]
mov edi, [ebp +12]
mov ebx, [ebp+ 16]
mov ecx, [ebp+20]
mov eax, 0
L1:
movzx eax, byte ptr [esi]
add ax, [edi]
mov [ebx], eax
add esi, type byte
add edi, type word
add ebx, type dword
Loop L1
popad
 mov esp, ebp                    ; clean the locals
 pop ebp
 ret 16                           ;  clean up the stack
SumArrayElements ENDP
END main
```