

Parameter Passing Through Stack

- ❖ Parameters can be saved on the stack before a procedure is called.
- ❖ The called procedure can easily access the parameters using either the ESP or EBP registers without altering ESP register.
- ❖ Example

Suppose you want to implement the following pseudo-code:

```
i = 25;  
j = 4;  
Test(i, j, 1);
```

Then, the assembly language code fragment looks like:

```
mov i, 25  
mov j, 4  
push 1  
push j  
push i  
call Test
```

Parameter Passing Through Stack

Example: Accessing parameters on the stack

Test PROC

mov AX, [ESP + 4] ;get i

add AX, [ESP + 8] ;add j

sub AX, [ESP + 12] ;subtract parm 3
(1) from sum

ret

Test ENDP

Lower Address

ESP

ESP+4

ESP+8

ESP+12

Higher Address

Return Address
25 (i)
4 (j)
1

Call & Return Instructions

Instruction	Operand	Note
CALL	label name	Push IP IP = IP + displacement relative to next instruction
CALL	r/m	Push IP IP = [r/m]
CALL	label name (FAR)	Push CS Push IP CS:IP = address of label name
CALL	m (FAR)	Push CS Push IP CS:IP = [m]
RET		Pop IP
RET	imm	Pop IP SP = SP + imm
RET	(FAR)	Pop IP Pop CS
RET	imm (FAR)	Pop IP Pop CS SP = SP + imm

Freeing Passed Parameters From Stack

- ❖ Use **RET N** instruction to free parameters from stack

Example: Accessing parameters on the stack

Test PROC

mov AX, [ESP + 4] ;get i

add AX, [ESP + 8] ;add j

sub AX, [ESP + 12] ;subtract parm. 3
(1) from sum

ret 12

Test ENDP

Local Variables

- ❖ Local variables are dynamic data whose values must be preserved over the lifetime of the procedure, but not beyond its termination.
- ❖ At the termination of the procedure, the current environment disappears and the previous environment must be restored.
- ❖ Space for local variables can be reserved by subtracting the required number of bytes from ESP.
- ❖ Offsets from ESP are used to address local variables.

Local Variables

Pseudo-code (Java-like)

```
void Test(int i){  
    int k;  
  
    k = i+9;  
    .....  
}
```

Assembly Language

```
Test PROC  
    push EBP  
    mov EBP, ESP  
    sub ESP, 4  
    push EAX  
    mov DWORD PTR [EBP-4], 9  
    mov EAX, [EBP + 8]  
    add [EBP-4], EAX  
    .....  
    pop EAX  
    mov ESP, EBP  
    pop EBP  
    ret 4  
Test ENDP
```

Summary

- ❖ Procedure – Named block of executable code
 - ✧ CALL: call a procedure, push return address on top of stack
 - ✧ RET: pop the return address and return from procedure
 - ✧ Preserve registers across procedure calls
- ❖ Runtime stack – LIFO structure – Grows downwards
 - ✧ Holds return addresses, saved registers, etc.
 - ✧ PUSH – insert value on top of stack, decrement ESP
 - ✧ POP – remove top value of stack, increment ESP
- ❖ Use the Irvine32.lib library for standard I/O
 - ✧ Include Irvine32.inc to make procedure prototypes visible
 - ✧ You can learn more by studying Irvine32.asm code