

X86 Assembly Language Programming: Part 5

Procedures

EECE416 uC

Charles Kim
Howard University

www.mwftr.com/416F15.html

Procedures with Value Parameters

- Main program **call**(s) a procedure
- Main Program transfers the parameter values
- Procedure receives (retrieves) them
- Procedure may do a task or it may return a value
 - value-returning procedure is sometimes called a *function*

Diagram illustrating the interaction between a Main program and a Procedure:

Main

```
call gcd
mov eax, 0
```

Procedure

```
gcd PROC
    (x, y, → z)
ret
```

The diagram shows the Main program calling the gcd procedure. The procedure is defined with parameters x, y and returns a value z. The Main program uses the call instruction to invoke the gcd procedure, and the Procedure block shows the gcd procedure's definition, including its parameters and the return instruction (ret).

Procedure Calling and Stack

- 3 concepts:
 - How to **transfer** control from a calling [main] program to a procedure and **back**
 - How to **pass parameter values** to a procedure and results **back** from the procedure
 - How to write **procedure code** that is **independent of the calling program**.
- Hardware **stack** is used to accomplish each of the above jobs.

80x86 Stack

- Hardware Stack
 - ESP holds the address of the “**first byte above (or higher)**” of the stack pointer
 - Most access is indirect, through the stack point register ESP
 - Operating system initializes ESP to point to **byte above stack pointer**
 - As program executes, it points to the last item pushed on the stack
 - “Top” of stack is at the highest address
 - Stack grows toward lower address

How Call/Ret Works

- **call**

- The address of the instruction EIP following the `call` is pushed on the stack (**so ESP has grown by 4 --- ESP address is lowered by 4**) [**Equivalent to Push EIP**]
- The instruction pointer register EIP is loaded with the address of the first instruction in the procedure

- **ret**

- The doubleword on the **top of the stack** is popped into the instruction pointer register EIP (**so ESP has decreased by 4 ---- ESP address is increased by 4**) [**Equivalent to Pop EIP**]
- this is the address of the instruction following the call, that instruction will be executed next [**Return Address**]
- If the stack has been used for other values after the call, these must be removed before the `ret` instruction is executed

Alternative Ret Format

- **ret *n***
 - After the returned address is popped to EIP from the stack, *n* is added to ESP.
 - This is most often used to logically remove procedure parameters that have been pushed onto the stack
 - Used in **Stdcall** Protocol
- Protocol?
 - Transfer of control from calling program to procedure and back
 - Passing parameter values to procedure and results back from the procedure
 - Having procedure code that is independent of the calling program

Procedure protocols for Stack Clean-Up

- 2 Protocols for Procedure handling
 - Cdecl (“C Declaration”) --- Caller Clean-Up
 - Stdcall (“Standard Call”) --- Callee Clean-Up

“Clean-up” means move Stack Pointer back to the original position

Cdecl (“C Declaration”)

- Caller Clean-up convention
- used by many **C systems** for the x86 architecture.
- Default in Visual Studio
- **Function parameters are pushed on the stack.**
- **Function return values are returned in the EAX register**
- Registers EAX, ECX, and EDX are available for use in the function.
- The **calling program cleans the stack** after the function call returns

```
/* example of __cdecl */  
push arg1  
push arg2  
push arg3  
call function  
add esp,12      // effectively "pop; pop; pop"
```

```
:_MyFunction1  
push ebp  
mov ebp, esp  
mov eax, [ebp + 8]  
mov edx, [ebp + 12]  
add eax, edx  
pop ebp  
ret
```


Stdcall --- we use this in class

- Callee Clean-up Convention
- A variation on the **Pascal calling convention**
- Callee is responsible for cleaning up the stack
 - Ret N
 - N is added to ESP
- Parameters are pushed to the stack
- **Registers EAX, ECX, and EDX are designated for use within the function.**
- Return values are stored in the EAX register.
- Standard calling convention for the Microsoft Win32 API.

```
/* example of __stdcall */  
push arg1  
push arg2  
call function  
// no stack cleanup - callee does this
```

```
:_MyFunction@8  
push ebp  
mov ebp, esp  
mov eax, [ebp + 8]  
mov edx, [ebp + 12]  
add eax, edx  
pop ebp  
ret 8
```

Structure of Procedure in Coding (Stdcall)

.code

main PROC

```
mov EAX,22220000h
mov AX,word1
push EAX
mov AX,word2
push EAX
call AddTwo ;Call
exit
```

main ENDP

Main Code:
CALLER

AddTwo PROC

```
push EBP
mov EBP,ESP
mov EAX,[EBP + 12]
add EAX,[EBP + 8]
pop EBP
ret 8
```

AddTwo ENDP

Procedure,
Subroutine,
function:
CALLEE

END main

← End of the Code

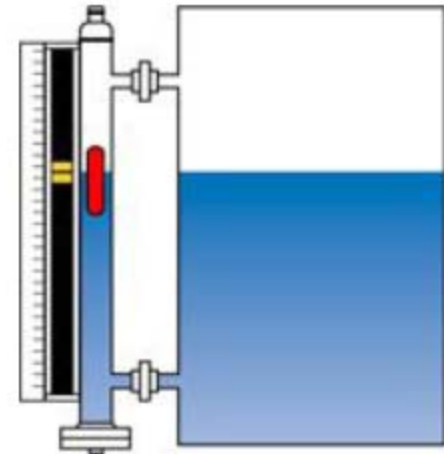
Push Instruction

- Usual format: **push** source
 - source can be memory, register or immediate
 - **doubleword** or **word** pushed on the stack
- ESP decremented by size of operand
- Operand stored in stack where ESP points after being decremented
- Flags not changed
- **By Push, stack point goes lower (“grows”) in address (ESP)**
- Push/Pop from the Stack Pointer (ESP register)

Push Instruction

		Stack size of 16	
		Stack Point = 0010	
		Stack filling starts from 000F (1 below the pointer)	
Stack		Instruction	ESP
			0010
		push 01020304	000C
		push 0A0B0C0D	0008
0000			
0001			
0002			
0003			
0004			
0005			
0006			
0007			
0008	0D		
0009	0C		
000A	0B		
000B	0A		
000C	04		
000D	03		
000E	02		
000F	01		

0010 <----- Stack Pointer



Stack push (filling) starts from 1 lower the ESP, and stays there
Stack pops from the ESP and moves 1 higher

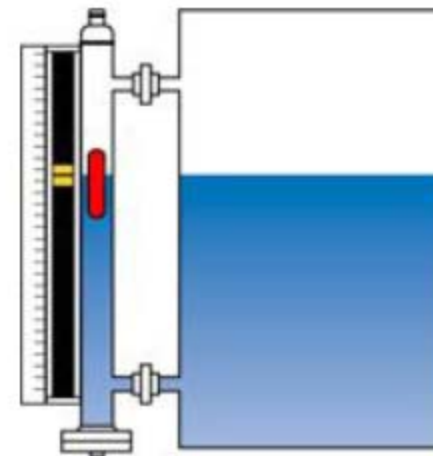
Push Example

- Pushd --- DWORD size operand

- 240d → F0h → 0000 00 F0
~~FFFFFFFF0F~~ +1
 (FFFFFFFF10)

Stack push (filling) starts from 1 lower the ESP, and stays there
 Stack pops from the ESP and moves 1 higher

Stack	Data	Instruction	ESP	EAX
			00600200	83B547A2
		push EAX	006001FC	
		pushd -240	006001F8	
006001F0				
006001F1				
006001F2				
006001F3				
006001F4				
006001F5				
006001F6				
006001F7				
006001F8	10			
006001F9	FF			
006001FA	FF			
006001FB	FF			
006001FC	A2			
006001FD	47			
006001FE	B5			
006001FF	83			
00600200				



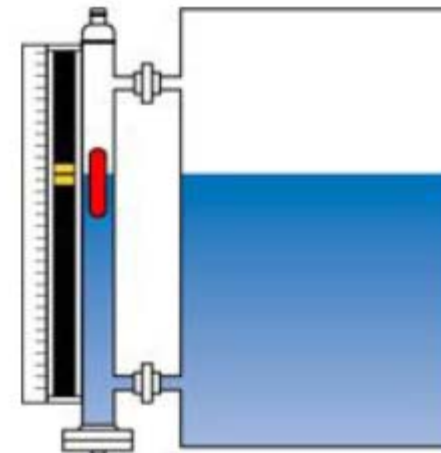
pop Instruction and Execution

- Usual format: `pop destination`
 - **doubleword** *destination* can be **memory** or **register**
- Operand stored in stack where ESP point is copied to destination
- ESP **incremented by size of operand** after the value is copied

pop Instruction and Execution

Stack push (filling) starts from 1 lower the ESP, and stays there
Stack pops from the ESP and moves 1 higher

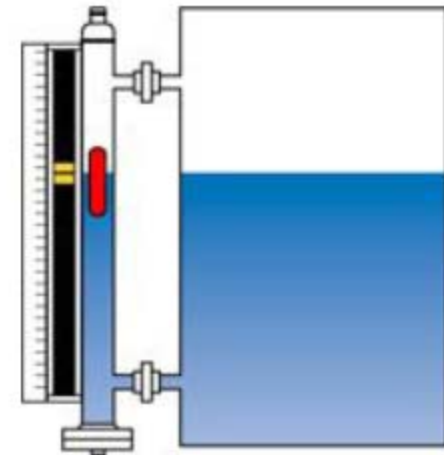
Stack	Data	Instruction	ESP	EAX
			00600200	83B547A2
		push EAX	006001FC	
		pushd -240	006001F8	
		pop EAX	006001FC	FFFFFF10
006001F0				
006001F1				
006001F2				
006001F3				
006001F4				
006001F5				
006001F6				
006001F7				
006001F8	10			
006001F9	FF			
006001FA	FF			
006001FB	FF			
006001FC	A2			
006001FD	47			
006001FE	B5			
006001FF	83			
00600200				



Pop Example [pop CX]

Stack push (filling) starts from 1 below the ESP, and stays there
Stack pops from the ESP and moves 1 above

Stack	Data	Instruction	ESP	ECX
			00600200	83B547A2
		push ECX	006001FC	
		pushd -240	006001F8	
		pop ECX	006001FC	FFFFFF10
		pop CX	006001FE	FFFF47A2
006001F0				
006001F1				
006001F2				
006001F3				
006001F4				
006001F5				
006001F6				
006001F7				
006001F8	10			
006001F9	FF			
006001FA	FF			
006001FB	FF			
006001FC	A2			
006001FD	47			
006001FE	B5			
006001FF	83			
00600200				



Push Exercise

- Before
 - [ESP]=06 00 10 00
 - [ECX]=01 A2 5B 74
- After **push ECX**
- After **pushd 10**
 - [STACK]= ?

Stack push (filling) starts from 1 lower the ESP, and stays there
Stack pops from the ESP and moves 1 higher

Stack	Data	Instruction	ESP	ECX
			06001000	01A25B74
		push ECX	06000FFC	
		pushd 10	06000FF8	
06000FF7				
06000FF8	0A			
06000FF9	00			
06000FFA	00			
06000FFB	00			
06000FFC	74			
06000FFD	5B			
06000FFE	A2			
06000FFF	01			
06001000				

Push – Practice

- Before:
 - [ESP]=02 00 0B 7C
 - [EBX]=12 34 56 78
- Stack Diagram and [ESP]
 - After `pushd 20`
 - After `push EBX`

Stack push (filling) starts from 1 lower the ESP, and stays there Stack pops from the ESP and moves 1 higher				
Stack	Data	Instruction	ESP	EBX
			02000B7C	12345678
		<code>pushd 20</code>		
		<code>push EBX</code>		
02000B73				
02000B74				
02000B75				
02000B76				
02000B77				
02000B78				
02000B79				
02000B7A				
02000B7B				
02000B7C				
02000B7D				

Push-Pop Practice

- Before:
 - [ESP]=00 10 F8 3A
 - [EAX]=12 34 56 78
- Stack Diagram, [EAX], [EBX], & [ESP]

– After

- Push EAX
- Pushd 30
- Pop EAX
- Pop EBX

Stack push (filling) starts from 1 lower the ESP, and stays there Stack pops from the ESP and moves 1 higher					
Stack	Data	Instruction	ESP	EAX	EBX
			0010F83A	12345678	????????
		push EAX			
		pushd 30			
		pop EAX			
		pop EBX			
0010F82F					
0010F830					
0010F831					
0010F832					
0010F833					
0010F834					
0010F835					
0010F836					
0010F837					
0010F838					
0010F839					
0010F83A					

Push/Pop example code: Proc1.asm

```
TITLE Procedure Example      (Proc1.asm)
```

```
INCLUDE Irvine32.inc
```

```
.stack 4096
```

```
.data
```

```
.code
```

```
main PROC
```

```
    mov EAX,0
```

```
    mov EBX,0
```

```
    mov ECX,0
```

```
    mov EAX, 83B547A2h
```

```
    push EAX
```

```
    pushd -240      ;double word
```

```
    pushw 5; WORD Size
```

```
    pop EAX
```

```
    pop AX
```

```
    pop EBX
```

```
    exit
```

```
main ENDP
```

```
END main
```

- Pushw --- WORD size operand

Push/Pop example code: Proc1.asm

Proc1.asm x

```
TITLE Procedure Example      (Proc1.asm)

INCLUDE Irvine32.inc
.stack 4096

.data

.code
main PROC
    mov EAX,0
    mov EBX,0
    mov ECX,0
    mov EAX, 83B547A2h
    push EAX
    pushd -240      ;double word
    pushw 5; WORD Size
    pop EAX
    pop AX
    pop EBX
    exit
main FNDP
```

100 %

Memory 1

Address: 0x0018FF80

0x0018FF80	00 00 00 00 00 00 00 00
0x0018FF88	00 00 00 00 8a 33 56 76Š3Vv
0x0018FF90	00 e0 fd 7e d4 ff 18 00	.àý~Ôÿ..
0x0018FF98	72 9f 44 77 00 e0 fd 7e	rŸDw.àý~
0x0018FFA0	38 f3 32 76 00 00 00 00	8ó2v....
0x0018FFA8	00 00 00 00 00 e0 fd 7eàý~
0x0018FFB0	00 00 00 00 00 00 00 00
0x0018FFB8	00 00 00 00 a0 ff 18 00ÿ..

Registers

EAX = 76563378	EBX = 7EFDE000
ECX = 00000000	EDX = 00401005
ESI = 00000000	EDI = 00000000
EIP = 00401010	ESP = 0018FF8C
EBP = 0018FF94	EFL = 00000246

- Pushw --- WORD size operand

Push/Pop example code: Proc1.asm

- push EAX

```

mov ECX,0
mov EAX, 83B547A2h
push EAX
pushd -240 ;double
pushw 5; WORD Size
pop EAX
pop AX
pop EBX
exit
main FNDP

```

100 %

Memory1

Address: 0x0018FF80

```

0x0018FF80  00 00 00 00 00 00 00 00  ....
0x0018FF88  a2 47 b5 83 8a 33 56 76  4GµfŠ3Vv
0x0018FF90  00 e0 fd 7e d4 ff 18 00  .ây~Ôÿ..
0x0018FF98  72 9f 44 77 00 e0 fd 7e  rŸDw.ây~
0x0018FFA0  78 f3 33 7c 00 00 00 00  853...

```

	A	B	C	D	E	F
1		Stack push (filling) starts from 1 lower the ESP, and stays there				
2		Stack pops from the ESP and moves 1 higher				
3	Stack	Data	Instruction	ESP	EAX	EBX
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82		pushw 5	0018FF82		
8	0018FF83		pop EAX	0018FF86	FF100005	
9	0018FF84		pop AX	0018FF88	FF10FFFF	
10	0018FF85		pop EBX	0010FF8C		83B547A2
11	0018FF86					
12	0018FF87					
13	0010FF88	A2				
14	0010FF89	47				
15	0010FF8A	B5				
16	0010FF8B	83				
17	0010FF8C					
18	0010FF8D					

Registers

```

EAX = 83B547A2 EBX = 00000000
ECX = 00000000 EDX = 00401005
ESI = 00000000 EDI = 00000000
EIP = 00401025 ESP = 0018FF88
EBP = 0018FF94 EFL = 00000246

```

Push/Pop example code: Proc1.asm

- pushd -240

```

push EAX
pushd -240      ;double
pushw 5; WORD Size
pop  EAX
pop  AX
pop  EBX
exit
main FNDP

```

	A	B	C	D	E	F
1		Stack push (filling) starts from 1 lower the ESP, and stays there				
2		Stack pops from the ESP and moves 1 higher				
3	Stack	Data	Instruction	ESP	EAX	EBX
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82		pushw 5	0018FF82		
8	0018FF83		pop EAX	0018FF86	FF100005	
9	0018FF84	10	pop AX	0018FF88	FF10FFFF	
10	0018FF85	FF	pop EBX	0010FF8C		83B547A2
11	0018FF86	FF				
12	0018FF87	FF				
13	0010FF88	A2				
14	0010FF89	47				
15	0010FF8A	B5				
16	0010FF8B	83				
17	0010FF8C					
18	0010FF8D					

Memory1

Address: 0x0018FF80

```

0x0018FF80  00 00 00 00 10 ff ff ff  ....ÿÿÿ
0x0018FF88  a2 47 b5 83 8a 33 56 76  ¢GµfŠ3Vv
0x0018FF90  00 e0 fd 7e d4 ff 18 00  .àý~Ôÿ..
0x0018FF98  72 9f 44 77 00 e0 fd 7e  rŸDw.àý~
0x0018FFA0  38 f3 32 76 00 00 00 00  8ó2v....

```

Registers

```

EAX = 83B547A2  EBX = 00000000
ECX = 00000000  EDX = 00401005
ESI = 00000000  EDI = 00000000
EIP = 0040102A  ESP = 0018FF84
EBP = 0018FF94  EFL = 00000246

```

Push/Pop example code: Proc1.asm

- Pushw --- WORD size operand

- pushw 5

```

push EAX
pushd -240      ;double word
pushw 5; WORD Size
pop  EAX
pop  AX
pop  EBX
exit
main FNDP
    
```

100 %

	A	B	C	D	E	F
1			Stack push (filling) starts from 1 lower the ESP, and stays there			
2			Stack pops from the ESP and moves 1 higher			
3	Stack	Data	Instruction	ESP	EAX	EBX
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82	05	pushw 5	0018FF82		
8	0018FF83	00	pop EAX			
9	0018FF84	10	pop AX			
10	0018FF85	FF	pop EBX			
11	0018FF86	FF				
12	0018FF87	FF				
13	0018FF88	A2				
14	0018FF89	47				
15	0018FF8A	B5				
16	0018FF8B	83				
17	0018FF8C					
18	0018FF8D					

Memory1

Address: 0x0018FF80

```

3x0018FF80  00 00 05 00 10 ff ff ff  ....ÿÿÿ
3x0018FF88  a2 47 b5 83 8a 33 56 76  ¢GµfŠ3Vv
3x0018FF90  00 e0 fd 7e d4 ff 18 00  .àý~Ôÿ..
3x0018FF98  72 9f 44 77 00 e0 fd 7e  rŸDw.àý~
3x0018FFA0  38 f3 32 76 00 00 00 00  8ó2v....
    
```

Registers

```

EAX = 83B547A2  EBX = 00000000
ECX = 00000000  EDX = 00401005
ESI = 00000000  EDI = 00000000
EIP = 0040102D  ESP = 0018FF82
EBP = 0018FF94  EFL = 00000246
    
```


Push/Pop example code: Proc1.asm

- pop EAX

```

push EAX
pushd -240 ; c
pushw 5; WORD Siz
pop EAX
pop AX
pop EBX
exit
main FNDP

```

100 %

	A	B	C	D	E	F
1		Stack push (filling) starts from 1 lower the ESP, and stays there				
2		Stack pops from the ESP and moves 1 higher				
3	Stack	Data	Instruction	ESP	EAX	EBX
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82	05	pushw 5	0018FF82		
8	0018FF83	00	pop EAX	0018FF86	FF100005	
9	0018FF84	10	pop AX			
10	0018FF85	FF	pop EBX			
11	0018FF86	FF				
12	0018FF87	FF				
13	0018FF88	A2				
14	0018FF89	47				
15	0018FF8A	B5				
16	0018FF8B	83				
17	0018FF8C					
18	0018FF8D					

Memory 1

Address: 0x0018FF80

```

0x0018FF80  00 00 05 00 10 ff ff ff  ....ÿÿÿ
0x0018FF88  a2 47 b5 83 8a 33 56 76  ¢GµfŠ3Vv
0x0018FF90  00 e0 fd 7e d4 ff 18 00  .àý~Ôÿ..
0x0018FF98  72 9f 44 77 00 e0 fd 7e  rŸDw.àý~
0x0018FFA0  38 f3 32 76 00 00 00 00  8ó2v....

```

Registers

```

EAX = FF100005 EBX = 00000000
ECX = 00000000 EDX = 00401005
ESI = 00000000 EDI = 00000000
EIP = 0040102E ESP = 0018FF86
EBP = 0018FF94 EFL = 00000246

```

Push/Pop example code: Proc1.asm

- pop AX

	A	B	C	D	E	F
1		Stack push (filling) starts from 1 lower the ESP, and stays there				
2		Stack pops from the ESP and moves 1 higher				
3	Stack	Data	Instruction	ESP	EAX	EBX
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82	05	pushw 5	0018FF82		
8	0018FF83	00	pop EAX	0018FF86	FF100005	
9	0018FF84	10	pop AX	0018FF88	FF10FFFF	
10	0018FF85	FF	pop EBX			
11	0018FF86	FF				
12	0018FF87	FF				
13	0018FF88	A2				
14	0018FF89	47				
15	0018FF8A	B5				
16	0018FF8B	83				
17	0018FF8C					
18	0018FF8D					


```

push EAX
pushd -240 ;
pushw 5; WORD Si
pop EAX
pop AX
pop EBX
exit
main FNDP
  
```

100 %

Memory 1		Registers	
Address: 0x0018FF80		EAX = FF10FFFF EBX = 00000000	
0x0018FF80	00 00 05 00 10 ff ff ffÿÿÿ	ECX = 00000000	EDX = 00401005
0x0018FF88	a2 47 b5 83 8a 33 56 76 ¢GµfŠ3Vv	ESI = 00000000	EDI = 00000000
0x0018FF90	00 e0 fd 7e d4 ff 18 00 .àý~Ôÿ..	EIP = 00401030	ESP = 0018FF88
0x0018FF98	72 9f 44 77 00 e0 fd 7e rŸDw.àý~	EBP = 0018FF94	EFL = 00000246
0x0018FFA0	38 f3 32 76 00 00 00 00 8ó2v....		

Push/Pop example code: Proc1.asm

- pop EBX

```

mov EAX, 83B547A2h
push EAX
pushd -240 ;doub
pushw 5; WORD Size
pop EAX
pop AX
pop EBX
exit

```

main FNDP

	A	B	C	D	E	F
1		Stack push (filling) starts from 1 lower the ESP, and stays there				
2		Stack pops from the ESP and moves 1 higher				
3	Stack	Data	Instruction	ESP	EAX	EBX
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82	05	pushw 5	0018FF82		
8	0018FF83	00	pop EAX	0018FF86	FF100005	
9	0018FF84	10	pop AX	0018FF88	FF10FFFF	
10	0018FF85	FF	pop EBX	0010FF8C		83B547A2
11	0018FF86	FF				
12	0018FF87	FF				
13	0018FF88	A2				
14	0018FF89	47				
15	0018FF8A	B5				
16	0018FF8B	83				
17	0018FF8C					
18	0018FF8D					

Memory

Address: 0x0018FF80

```

0018FF80  00 00 05 00 10 ff ff ff  ....ÿÿÿ
0018FF88  a2 47 b5 83 8a 33 56 76  çGµfŠ3Vv
0018FF90  00 e0 fd 7e d4 ff 18 00  .àý~Ôÿ..
0018FF98  72 9f 44 77 00 e0 fd 7e  rŸDw.àý~
-----

```

Registers

```

EAX = FF10FFFF EBX = 83B547A2
ECX = 00000000 EDX = 00401005
ESI = 00000000 EDI = 00000000
EIP = 00401031 ESP = 0018FF8C
EBP = 0018FF94 EFL = 00000246

```

Procedure Example – CallAddTwo.asm (“Stdcall”)

```
TITLE Demonstrate the Calling AddTwo Procedure      (CallAddTwo.asm)
INCLUDE Irvine32.inc

.data
word1 WORD 1234h
word2 WORD 4111h

.code
main PROC
    mov EAX,22220000h
    mov AX,word1
    push EAX
    mov AX,word2
    push EAX
    call AddTwo      ;Call the STDCALL version
    call DumpRegs
    exit
main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov EBP,ESP
    mov EAX,[EBP + 12]      ; first parameter
    add EAX,[EBP + 8]       ; second parameter
    pop EBP
    ret 8                  ; clean up the stack
AddTwo ENDP

END main
```

Debugging

CallAddTwo.asm

```
word1 WORD 1234h
word2 WORD 4111h
.code
main PROC
    mov EAX,22220000h
    mov AX,word1
    push EAX
    mov AX,word2
    push EAX
    call AddTwo    ;Call the STDCALL version
    exit
main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov EBP,ESP
    mov EAX,[EBP + 12]    ; first parameter
    add EAX,[EBP + 8]     ; second parameter
    pop EBP
    ret 8                ; clean up the stack
```

100 %

Memory 1		Registers	
Address:	0x0018FF70	EAX =	75C23378
0x0018FF70	00 00 00 00 00 00 00 00	EBX =	7EFDE000
0x0018FF78	00 00 00 00 00 00 00 00	ECX =	00000000
0x0018FF80	00 00 00 00 00 00 00 00	EDX =	0040100A
0x0018FF88	00 00 00 00 8a 33 c2 75	ESI =	00000000
0x0018FF90	00 e0 fd 7e d4 ff 18 00	EDI =	00000000
		EIP =	00401020
		ESP =	0018FF8C
		EBP =	0018FF94
		EFL =	00000246

push EAX
;for word1

CallAddTwo.asm X

```
word1 WORD 1234h
word2 WORD 4111h
.code
main PROC
    mov EAX,22220000h
    mov AX,word1
    push EAX
    mov AX,word2
    push EAX
    call AddTwo    ;Call the STDCALL version
    exit
main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov EBP,ESP
    mov EAX,[EBP + 12]    ; first parameter
    add EAX,[EBP + 8]     ; second parameter
    pop EBP
    ret 8                ; clean up the stack
```

100 %

Memory1

Address: 0x0018FF70

0x0018FF70	00 00 00 00 00 00 00 00
0x0018FF78	00 00 00 00 00 00 00 00
0x0018FF80	00 00 00 00 00 00 00 00
0x0018FF88	34 12 22 22 8a 33 c2 75
0x0018FF90	00 e0 fd 7e d4 ff 18 00
0x0018FF98	72 9f 92 77 00 e0 fd 7e

Registers

EAX = 22221234	EBX = 7EFDE000
ECX = 00000000	EDX = 0040100A
ESI = 00000000	EDI = 00000000
EIP = 0040102C	ESP = 0018FF88
EBP = 0018FF94	EFL = 00000246
00404002 = 4111	

push EAX
; for
word2

```
CallAddTwo.asm
word1 WORD 1234h
word2 WORD 4111h
.code
main PROC
    mov EAX,22220000h
    mov AX,word1
    push EAX
    mov AX,word2
    push EAX
    call AddTwo    ;Call the STDCALL version
    exit
main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov EBP,ESP
    mov EAX,[EBP + 12]    ; first parameter
    add EAX,[EBP + 8]     ; second parameter
    pop EBP
    ret 8                ; clean up the stack
```

Memory1

Address: 0x0018FF70

0x0018FF70	00 00 00 00 00 00 00 00
0x0018FF78	00 00 00 00 00 00 00 00
0x0018FF80	00 00 00 00 11 41 22 22
0x0018FF88	34 12 22 22 8a 33 c2 75
0x0018FF90	00 e0 fd 7e d4 ff 18 00
0x0018FF98	72 9f 92 77 00 e0 fd 7e
0x0018FFA0	14 11 d8 77 00 00 00 00

Registers

EAX = 22224111	EBX = 7EFDE000
ECX = 00000000	EDX = 0040100A
ESI = 00000000	EDI = 00000000
EIP = 00401033	ESP = 0018FF84
EBP = 0018FF94	EFL = 00000246

call AddTwo

- Note that EIP (for return address) is stored in the stack

The screenshot shows a debugger window with the title bar 'CallAddTwo.asm'. The assembly code is as follows:

```
word1 WORD 1234h
word2 WORD 4111h
.code
main PROC
    mov EAX,22220000h
    mov AX,word1
    push EAX
    mov AX,word2
    push EAX
    call AddTwo    ;Call the STDCALL version
    exit
main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov EBP,ESP
    mov EAX,[EBP + 12]    ; first parameter
    add EAX,[EBP + 8]     ; second parameter
    pop EBP
    ret 8                ; clean up the stack
```

The Memory1 window shows the stack contents at address 0x0018FF70:

Address	0x0018FF70	0x0018FF78	0x0018FF80	0x0018FF88	0x0018FF90	0x0018FF98
00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	38 10 40 00 11 41 22 22	34 12 22 22 8a 33 c2 75	00 e0 fd 7e d4 ff 18 00	72 9f 92 77 00 e0 fd 7e	

The Registers window shows the following values:

Register	Value
EAX	22224111
EBX	7EFDE000
ECX	00000000
EDX	0040100A
ESI	00000000
EDI	00000000
EIP	0040103F
ESP	0018FF80
EBP	0018FF94
EFL	00000246

push EBP

```
CallAddTwo.asm X
word1 WORD 1234h
word2 WORD 4111h
.code
main PROC
    mov EAX,22220000h
    mov AX,word1
    push EAX
    mov AX,word2
    push EAX
    call AddTwo ;Call the STDCALL version
    exit
main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov EBP,ESP
    mov EAX,[EBP + 12] ; first parameter
    add EAX,[EBP + 8] ; second parameter
    pop EBP
    ret 8 ; clean up the stack

```

100 %

Memory1		Registers
Address:	0x0018FF70	EAX = 22224111 EBX = 7EFDE000
0x0018FF70	00 00 00 00 00 00 00 00	ECX = 00000000 EDX = 0040100A
0x0018FF78	00 00 00 00 94 ff 18 00	ESI = 00000000 EDI = 00000000
0x0018FF80	38 10 40 00 11 41 22 22	EIP = 00401040 ESP = 0018FF7C
0x0018FF88	34 12 22 22 8a 33 c2 75	EBP = 0018FF94 EFL = 00000246
0x0018FF90	00 e0 fd 7e d4 ff 18 00	
0x0018FF98	72 9f 92 77 00 e0 fd 7e	

mov EBP, ESP
;to save ESP

```
CallAddTwo.asm X
word1 WORD 1234h
word2 WORD 4111h
.code
main PROC
    mov EAX,22220000h
    mov AX,word1
    push EAX
    mov AX,word2
    push EAX
    call AddTwo    ;Call the STDCALL version
    exit
main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov EBP,ESP
    mov EAX,[EBP + 12]    ; first parameter
    add EAX,[EBP + 8]     ; second parameter
    pop EBP
    ret 8                ; clean up the stack

```

Memory1		Registers
Address:	0x0018FF70	EAX = 22224111 EBX = 7EFDE000
0x0018FF70	00 00 00 00 00 00 00 00	ECX = 00000000 EDX = 0040100A
0x0018FF78	00 00 00 00 94 ff 18 00	ESI = 00000000 EDI = 00000000
0x0018FF80	38 10 40 00 11 41 22 22	EIP = 00401042 ESP = 0018FF7C
0x0018FF88	34 12 22 22 8a 33 c2 75	EBP = 0018FF7C EFL = 00000246
0x0018FF90	00 e0 fd 7e d4 ff 18 00	
0x0018FF98	72 9f 92 77 00 e0 fd 7e	0018FF88 = 22221234
0x0018FFA0	11 11 48 77 00 00 00 00	

CallAddTwo.asm

```
word1 WORD 1234h
word2 WORD 4111h
.code
main PROC
    mov EAX,22220000h
    mov AX,word1
    push EAX
    mov AX,word2
    push EAX
    call AddTwo    ;Call the STDCALL version
    exit
main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov EBP,ESP
    mov EAX,[EBP + 12]    ; first parameter
    add EAX,[EBP + 8]     ; second parameter
    pop EBP
    ret 8                ; clean up the stack
```

mov EAX,[EBP+12]

Memory1

Address: 0x0018FF70

0x0018FF70	00 00 00 00 00 00 00 00
0x0018FF78	00 00 00 00 94 ff 18 00
0x0018FF80	38 10 40 00 11 41 22 22
0x0018FF88	34 12 22 22 8a 33 c2 75
0x0018FF90	00 e0 fd 7e d4 ff 18 00
0x0018FF98	72 9f 92 77 00 e0 fd 7e

Registers

EAX = 22221234	EBX = 7EFDE000
ECX = 00000000	EDX = 0040100A
ESI = 00000000	EDI = 00000000
EIP = 00401045	ESP = 0018FF7C
EBP = 0018FF7C	EFL = 00000246
0018FF84 = 22224111	

CallAddTwo.asm X

```
word1 WORD 1234h
word2 WORD 4111h
.code
main PROC
    mov EAX,22220000h
    mov AX,word1
    push EAX
    mov AX,word2
    push EAX
    call AddTwo    ;Call the STDCALL version
    exit
main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov EBP,ESP
    mov EAX,[EBP + 12]    ; first parameter
    add EAX,[EBP + 8]     ; second parameter
    pop EBP
    ret 8                ; clean up the stack
```

add EAX, [EBP+8]

100 %

Memory1

Address: 0x0018FF70

0x0018FF70	00 00 00 00 00 00 00 00
0x0018FF78	00 00 00 00 94 ff 18 00
0x0018FF80	38 10 40 00 11 41 22 22
0x0018FF88	34 12 22 22 8a 33 c2 75
0x0018FF90	00 e0 fd 7e d4 ff 18 00
0x0018FF98	72 9f 92 77 00 e0 fd 7e

Registers

EAX = 44445345	EBX = 7EFDE000
ECX = 00000000	EDX = 0040100A
ESI = 00000000	EDI = 00000000
EIP = 00401048	ESP = 0018FF7C
EBP = 0018FF7C	EFL = 00000202

pop EBP

CallAddTwo.asm X

```
word1 WORD 1234h
word2 WORD 4111h
.code
main PROC
    mov EAX,22220000h
    mov AX,word1
    push EAX
    mov AX,word2
    push EAX
    call AddTwo ;Call the STDCALL version
    exit
main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov EBP,ESP
    mov EAX,[EBP + 12] ; first parameter
    add EAX,[EBP + 8] ; second parameter
    pop EBP
    ret 8 ; clean up the stack
```

100 %

Memory1		Registers	
Address:	0x0018FF70	EAX =	44445345
0x0018FF70	00 00 00 00 00 00 00 00	EBX =	7EFDE000
0x0018FF78	00 00 00 00 94 ff 18 00	ECX =	00000000
0x0018FF80	38 10 40 00 11 41 22 22	EDX =	0040100A
0x0018FF88	34 12 22 22 8a 33 c2 75	ESI =	00000000
0x0018FF90	00 e0 fd 7e d4 ff 18 00	EDI =	00000000
		EIP =	00401049
		ESP =	0018FF80
		EBP =	0018FF94
		EFL =	00000202

```

CallAddTwo.asm
word1 WORD 1234h
word2 WORD 4111h
.code
main PROC
    mov EAX,22220000h
    mov AX,word1
    push EAX
    mov AX,word2
    push EAX
    call AddTwo    ;Call the STDCALL version
    exit
main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov EBP,ESP
    mov EAX,[EBP + 12]    ; first parameter
    add EAX,[EBP + 8]     ; second parameter
    pop EBP
    ret 8                ; clean up the stack

```

100 %

Memory1

Address: 0x0018FF70

0x0018FF70	00 00 00 00 00 00 00 00
0x0018FF78	00 00 00 00 94 ff 18 00
0x0018FF80	38 10 40 00 11 41 22 22
0x0018FF88	34 12 22 22 8a 33 c2 75
0x0018FF90	00 e0 fd 7e d4 ff 18 00
0x0018FF98	72 9f 92 77 00 e0 fd 7e

Registers

EAX = 44445345	EBX = 7EFDE000
ECX = 00000000	EDX = 0040100A
ESI = 00000000	EDI = 00000000
EIP = 00401038	ESP = 0018FF8C
EBP = 0018FF94	EFL = 00000202

ret 8

;Add 8
to ESP

.LST file

```
00000000
00000000 1234
00000002 4111
00000000
00000000
00000000 B8 22220000
00000005 66| A1
00000000 00000000 R
0000000B 50
0000000C 66| A1
00000000 00000002 R
00000012 50
00000013 E8 00000007
0000001F
0000001F
0000001F 55
00000020 8B EC
00000022 8B 45 0C
00000025 03 45 08
00000028 5D
00000029 C2 0008
0000002C

.data
word1 WORD 1234h
word2 WORD 4111h
.code
main PROC
    mov     EAX,22220000h
    mov     AX,word1

    push    EAX
    mov     AX,word2

    push    EAX
    call    AddTwo ;Call the STDCALL version
    exit

main ENDP

AddTwo PROC
; Adds two integers, returns sum in EAX.
; The RET instruction cleans up the stack.
    push EBP
    mov  EBP,ESP
    mov  EAX,[EBP + 12] ; first parameter
    add  EAX,[EBP + 8]  ; second parameter
    pop  EBP
    ret  8 ; clean up the stack
AddTwo ENDP

END main
```

Code with actual address (and stack)

ADDRESS	Machine Code (or Sta	Instruction
		Main PROC
00401020	B8 22220000	mov EAX, 22220000h
00401025	66 A1 00000000 R	mov AX, word1
0040102B	50	push EAX
0040102C	66 A1 00000002 R	mov AX, word2
00401032	50	push EAX
00401033	E8 00000007	call AddTwo
00401038		exit
0040103F	55	push EBP
00401040	8B EC	mov EBP, ESP
00401042	8B 45 0C	mov EAX, [EBP+12]
00401045	03 45 08	add EAX, [EBP+8]
00401048	5D	pop EBP
00401049	C2 0008	ret 8
0040104C		ENDP
00404000	1234	word1
00404002	4111	word2

- Stack

0018FF70		
0018FF74		
0018FF78		
0018FF7C	94 FF 18 00	[EBP saved]
0018FF80	38 10 40 00	[Return address]
0018FF84	11 41 22 22	[word 2]
0018FF88	34 12 22 22	[word 1]
0018FF8C		

ADDRESS	Machine Code (or Statement)	Instruction	EIP	EAX	EBP	ESP
		Main PROC		75C23378	0018FF94	0018FF8C
00401020	B8 22220000	mov EAX, 22220000h				
00401025	66 A1 00000000 R	mov AX, word1				
0040102B	50	push EAX				
0040102C	66 A1 00000002 R	mov AX, word2				
00401032	50	push EAX				
00401033	E8 00000007	call AddTwo				
00401038		exit				
0040103F	55	push EBP				
00401040	8B EC	mov EBP, ESP				
00401042	8B 45 0C	mov EAX, [EBP+12]				
00401045	03 45 08	add EAX, [EBP+8]				
00401048	5D	pop EBP				
00401049	C2 0008	ret 8				
0040104C		ENDP				
00404000	1234	word1				
00404002	4111	word2				
0018FF70						
0018FF74						
0018FF78						
0018FF7C						
0018FF80						
0018FF84						
0018FF88						
0018FF8C						

Stack push (filling) starts from 1 lower the ESP, and stays there
Stack pops from the ESP and moves 1 higher

Summary for Stdcall

MAIN CODE

1. Parameter values passed on the stack
2. Call a procedure (this pushes the return address in EIP to the stack)

PROCEDURE

1. **Push EBP** and **Copy ESP to EBP** (EBP becomes the reference for retrieving the parameter values) – fixed location on the stack while ESP may vary.
2. Push Register(s) if necessary
3. Retrieve Parameter values referenced to EBP
4. Do the functions
5. Pop the Register(s) if pushed
6. Pop EBP
7. Ret N (First, this pops the return address to EIP. And, second, N, which is the number of bytes pushed in the MAIN CODE, is added to ESP)

MAIN CODE

1. Continue for the next step.