Assembly Language Programming:

Procedures

EECE416 uC

Charles Kim
Howard University

Fall 2013

www.mwftr.com

Before we start

- Schedule of the next few weeks
  - T Nov 19: Procedure and Calls (continued)
  - R Nov 21: Coding Project Due
  - R Nov 21: Intel Atom/FPGA Board
  - T Dec 03: Presentation + Demo (for Microcontrollers)
  - R Dec 05: Final Exam
    - Subjects related to all class activities since Exam01
      - Mul/imul, Div/idiv, Branching, Loop, and Procedure
    - Code reading → Flowchart → Description
    - Description →flowchart → Code writing
    - Coding similar to the GCD practice
    - Terminologies of Intel Atom/FPGA board

Procedure Calling and Stack

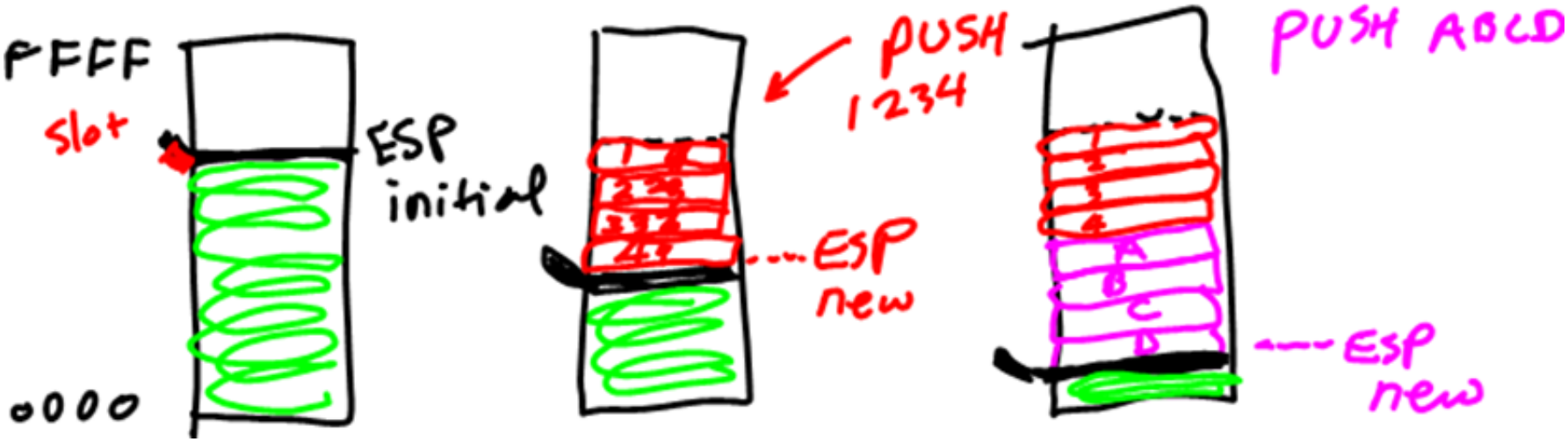
- 3 concepts:
  - How to transfer control from a calling [main] program to a procedure and back
  - How to pass parameter values to a procedure and results back from the procedure
  - How to write procedure code that is independent of the calling program.
- Hardware stack is used to accomplish each of the above jobs.
- Focus on 32-bit mode only

80x86 Stack

- Hardware Stack
  - Allocated with directive, for example
    `.STACK 4096`
    allocates 4096 uninitialized memory bytes
  - ESP holds the address of the "**first byte above"** the 4096 bytes in the stack
  - Most access is indirect, through the stack point register ESP
    - Operating system initializes ESP to point to byte above stack
    - 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

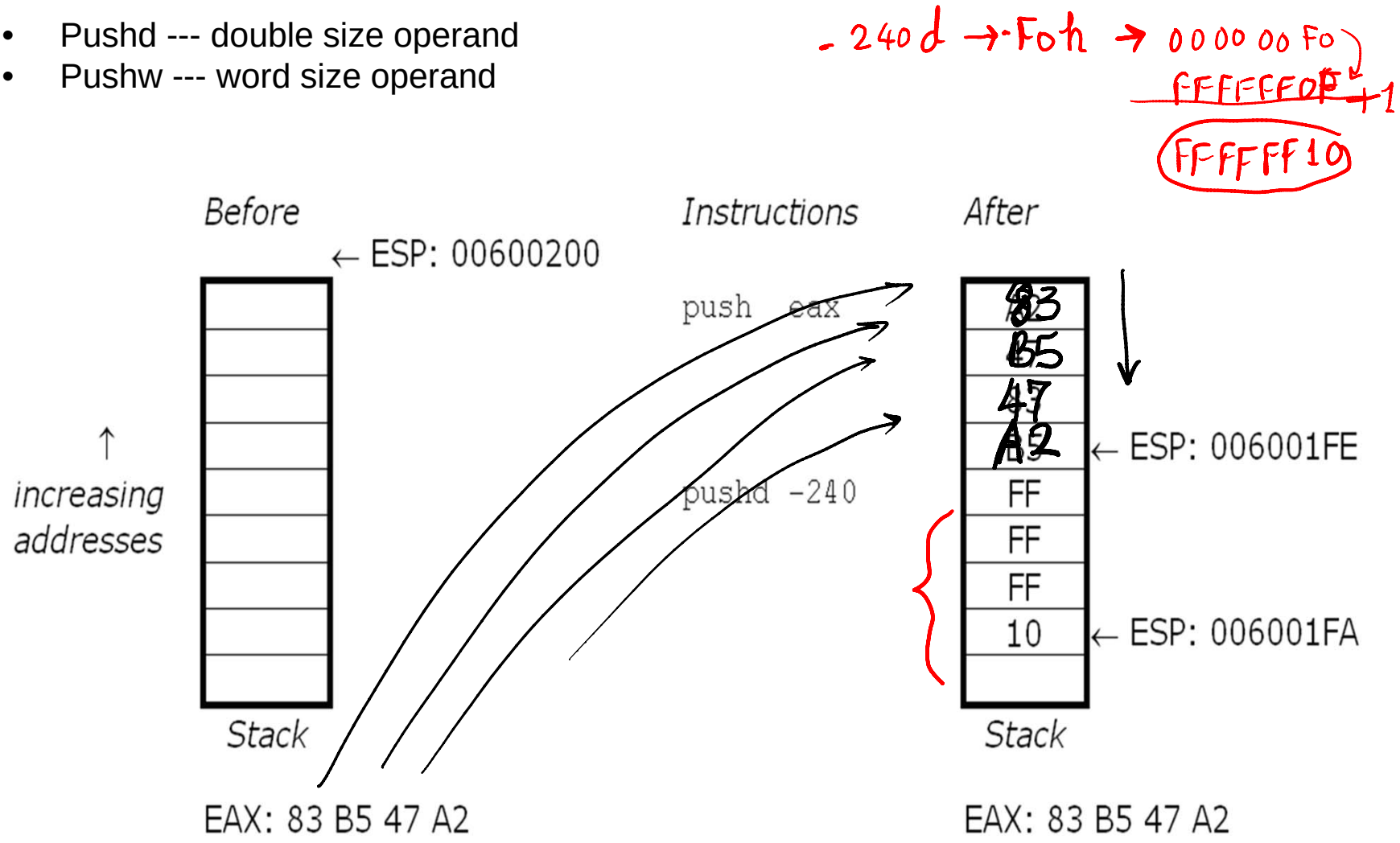
Push Instruction

- 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)

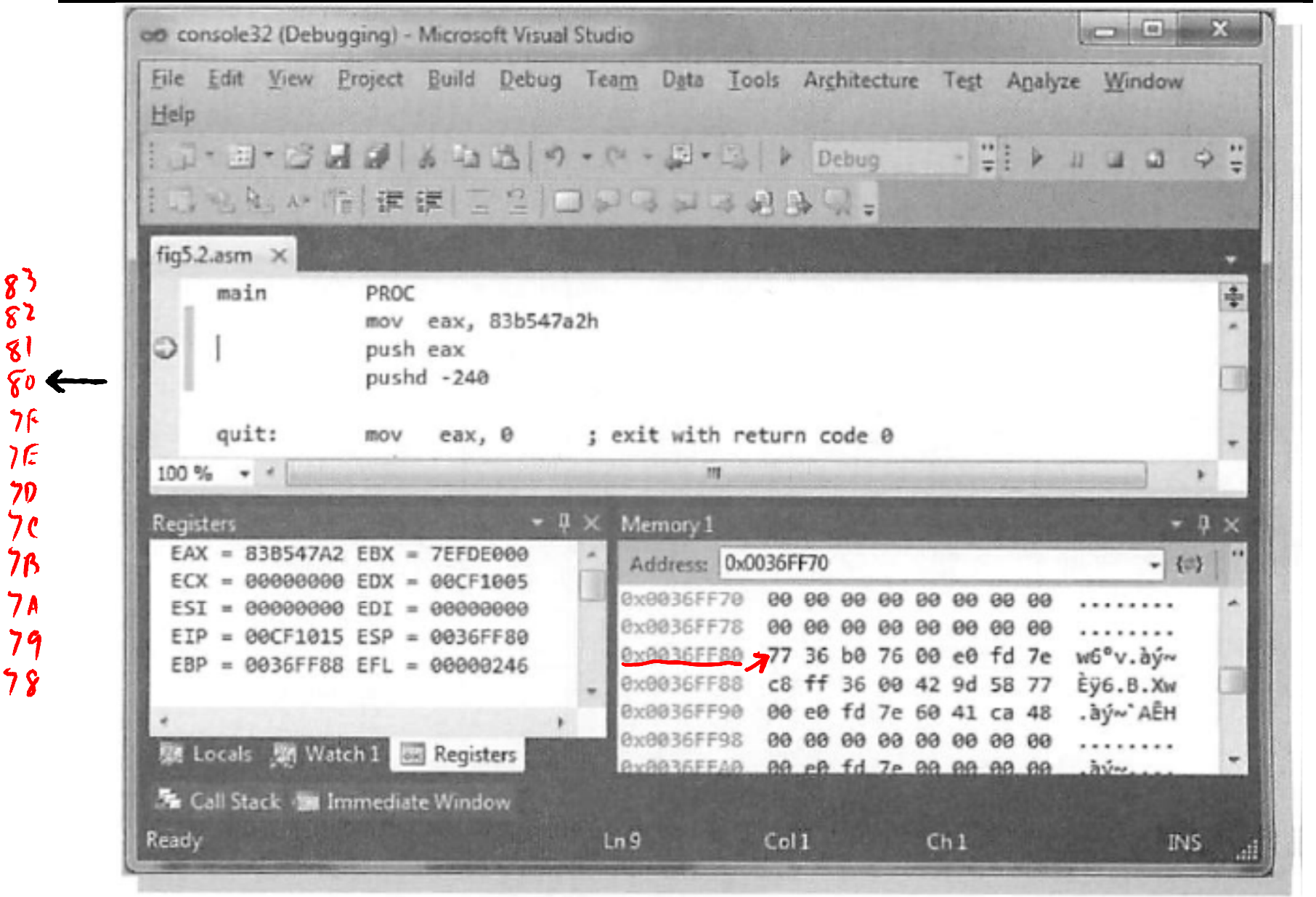


Push Example

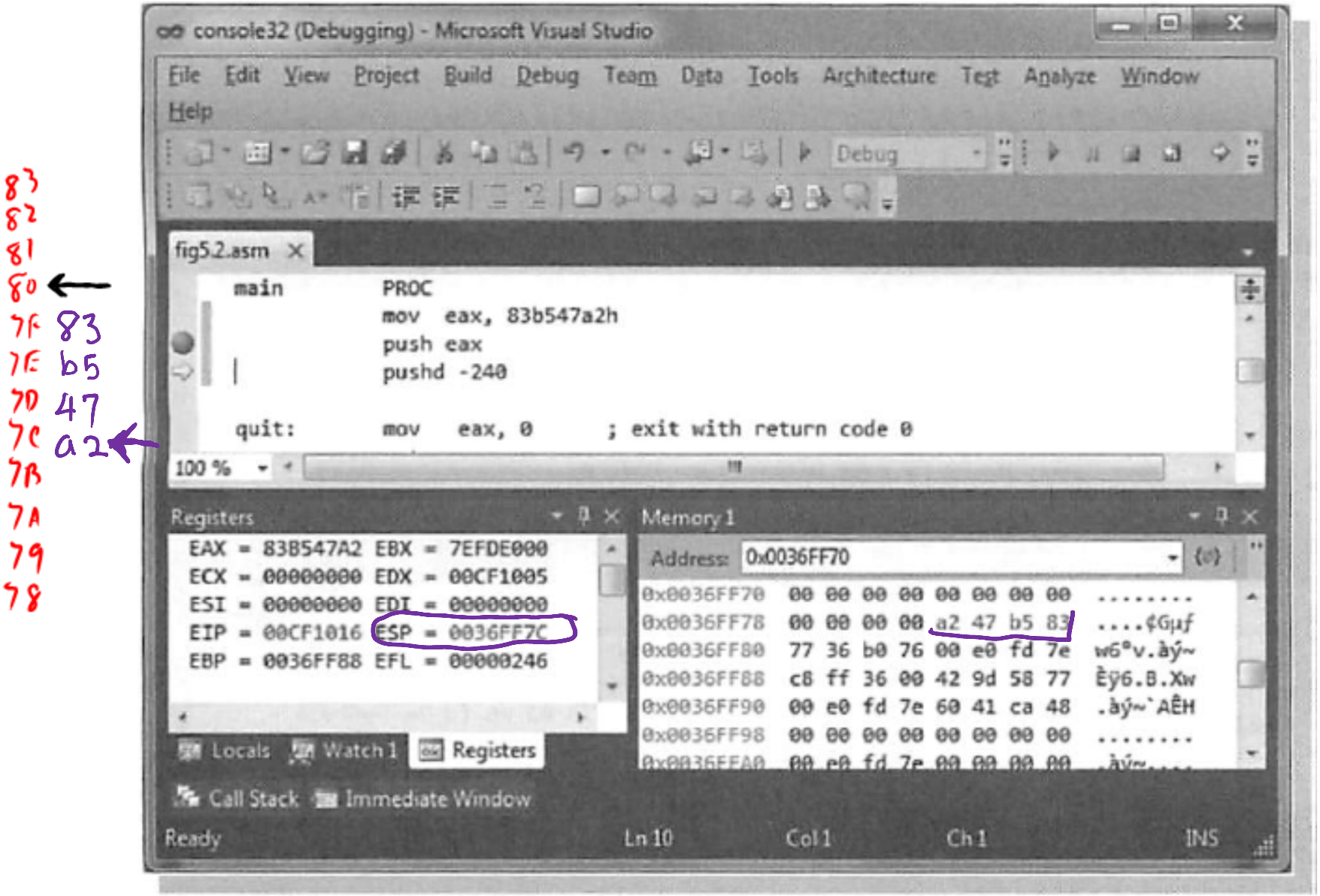
- Pushd --- double size operand
- Pushw --- word size operand



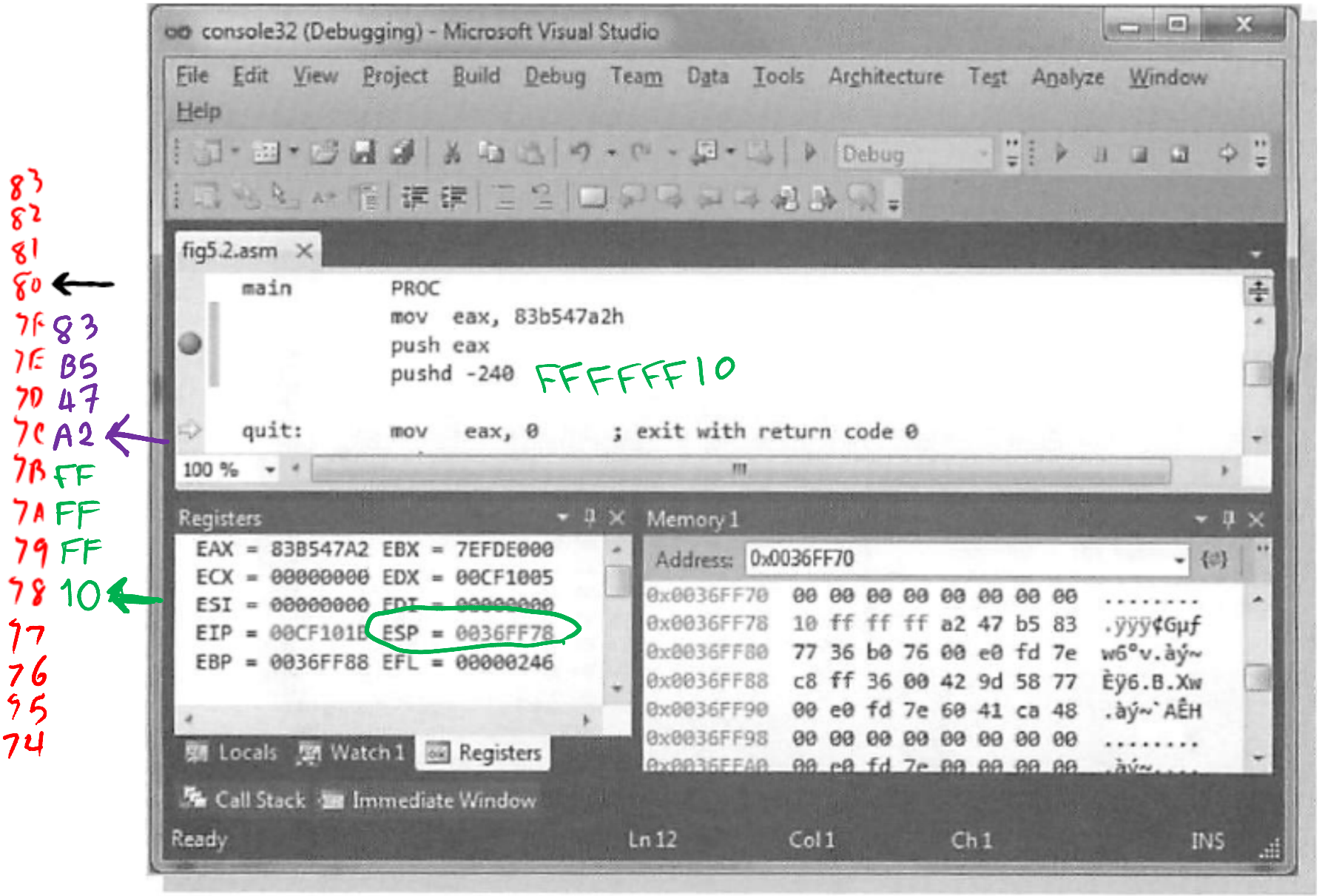
EAX and ESP contents



After PUSH


console32 (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window
Help
Debug
fig5_2.asm
main PROC
mov eax, 83b547a2h
push eax
pushd -240
quit: mov eax, 0 ; exit with return code 0
100 %
Registers
EAX = 83B547A2 EBX = 7EFDE000
ECX = 00000000 EDX = 00CF1005
ESI = 00000000 EDI = 00000000
EIP = 00CF1016 ESP = 0036FF7C
EBP = 0036FF88 EFL = 00000246
Memory 1
Address: 0x0036FF70
0x0036FF70 00 00 00 00 00 00 00 00 ....
0x0036FF78 00 00 00 00 a2 47 b5 83 ....¢Gµƒ
0x0036FF80 77 36 b0 76 00 e0 fd 7e w6°v.àý~
0x0036FF88 c8 ff 36 00 42 9d 58 77 Èÿ6.B.Xw
0x0036FF90 00 e0 fd 7e 60 41 ca 48 .àý~`AÊH
0x0036FF98 00 00 00 00 00 00 00 00 ....
Locals Watch 1 Registers
Call Stack Immediate Window
Ready Ln 10 Col 1 Ch 1 INS
83
82
81
80
7F 83
7E b5
7D 47
7C a2
7B
7A
79
78

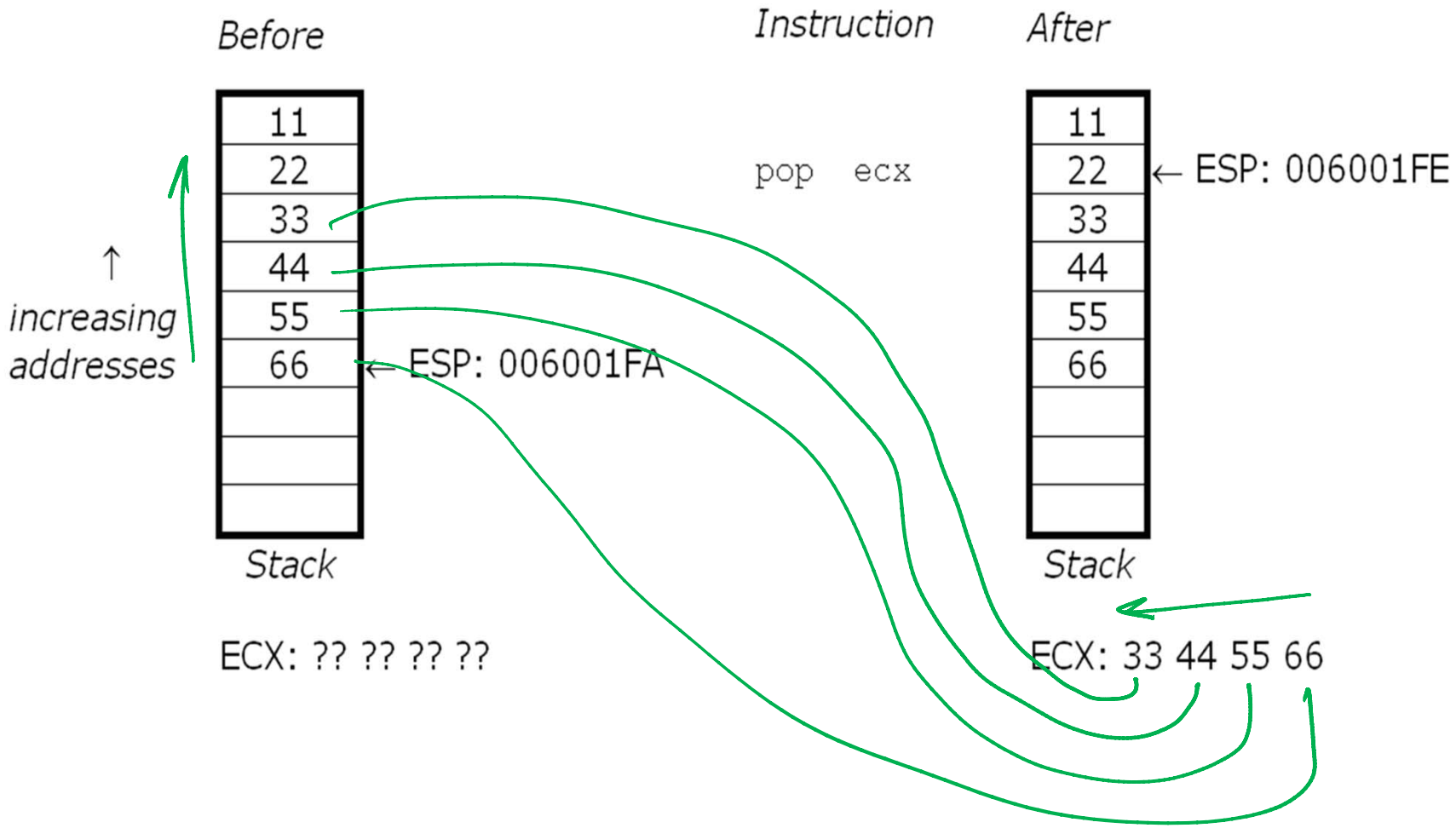
After pushd


console32 (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window
Help
Debug
fig5.2.asm
main PROC
mov eax, 83b547a2h
push eax
pushd -240
FFFFFF10
quit: mov eax, 0 ; exit with return code 0
100 %
Registers
EAX = 83B547A2 EBX = 7EFDE000
ECX = 00000000 EDX = 00CF1005
ESI = 00000000 EDI = 00000000
EIP = 00CF101E ESP = 0036FF78
EBP = 0036FF88 EFL = 00000246
Memory 1
Address: 0x0036FF70
0x0036FF70 00 00 00 00 00 00 00 00 ....
0x0036FF78 10 ff ff ff a2 47 b5 83 .ÿÿÿ¢Gµf
0x0036FF80 77 36 b0 76 00 e0 fd 7e w6°v.àý~
0x0036FF88 c8 ff 36 00 42 9d 58 77 Èÿ6.B.Xw
0x0036FF90 00 e0 fd 7e 60 41 ca 48 .àý~`AÊH
0x0036FF98 00 00 00 00 00 00 00 00 ....
Locals Watch 1 Registers
Call Stack Immediate Window
Ready Ln 12 Col 1 Ch 1 INS
83
82
81
80
7F 83
7E B5
7D 47
7C A2
7B FF
7A FF
79 FF
78 10
77
76
75
74

pop Instruction and Execution

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

Pop Example

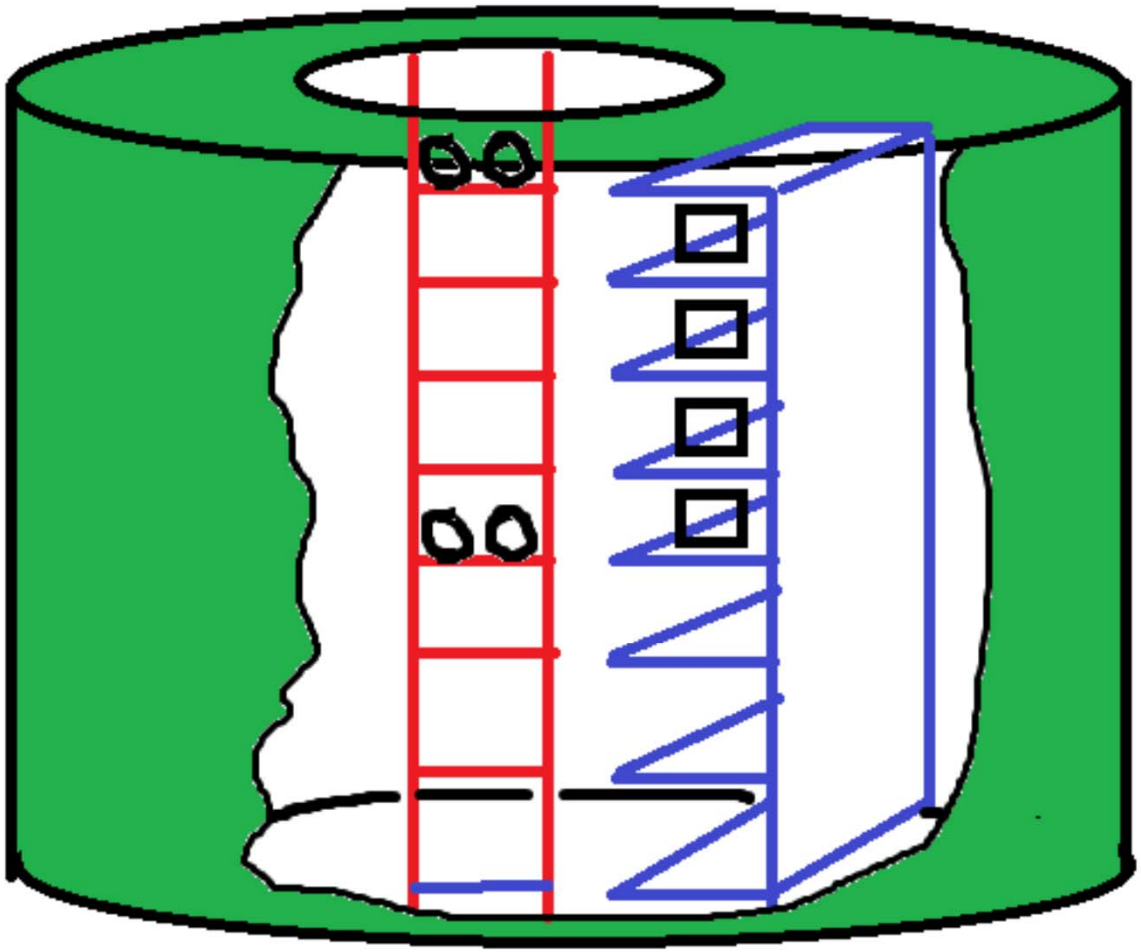

Before
Instruction
After
11
22
33
44
55
66
pop ecx
11
22
← ESP: 006001FE
33
44
55
66
↑
increasing
addresses
← ESP: 006001FA
Stack
Stack
ECX: ?? ?? ?? ??
ECX: 33 44 55 66

Pushfd and popfd

- pushfd pushes EFLAGS register contents onto stack
- popfd pops doubleword from top of stack into EFLAGS

Push Pop Illustration (Analogy)

- For the usual double-word operation
- Top is higher in address than bottom

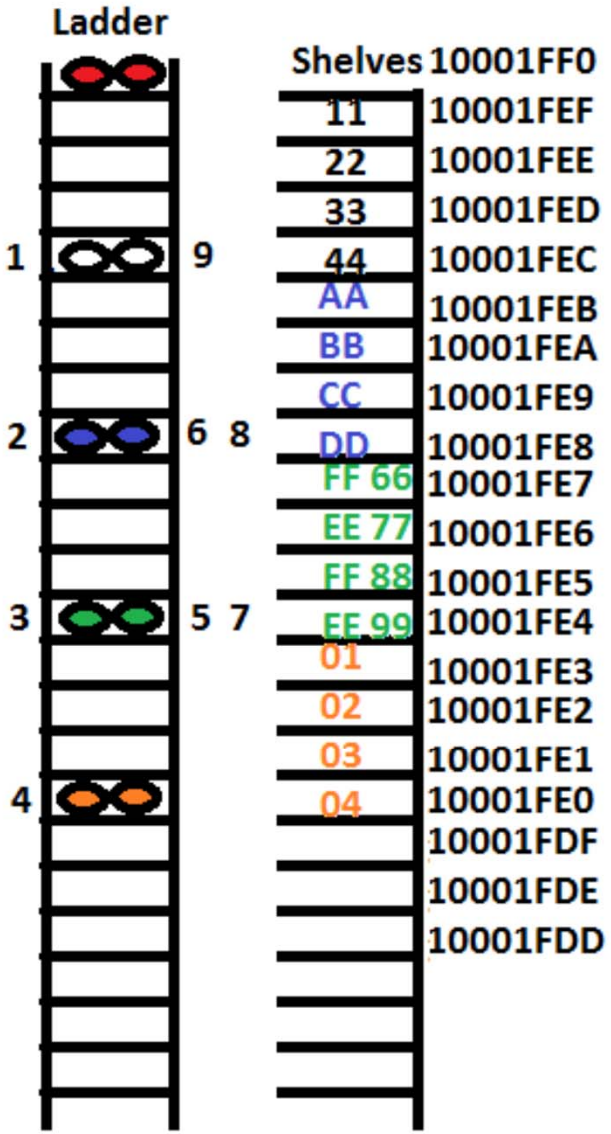


PUSH:
Climb down 4 rungs from where you are, while putting a BYTE at each shelf.

POP:
Climb up 4 rungs from where you are, while taking a BYTE from each shelf.

ESP:
Your current feet location.

Example (with original ESP=10001FF0)



ESP=10001FF0

1. PUSH 11223344h — ESP=10001FEC

2. PUSH AABBCCDDh — ESP=10001FE8

3. PUSH FFEEFFEEh — ESP=10001FE4

4. PUSH 01020304h — ESP=10001FE0

5. POP EAX — EAX=01020304 — ESP=10001FE4

6. POP EBX — EBX=FFEEFFEE — ESP=10001FE8

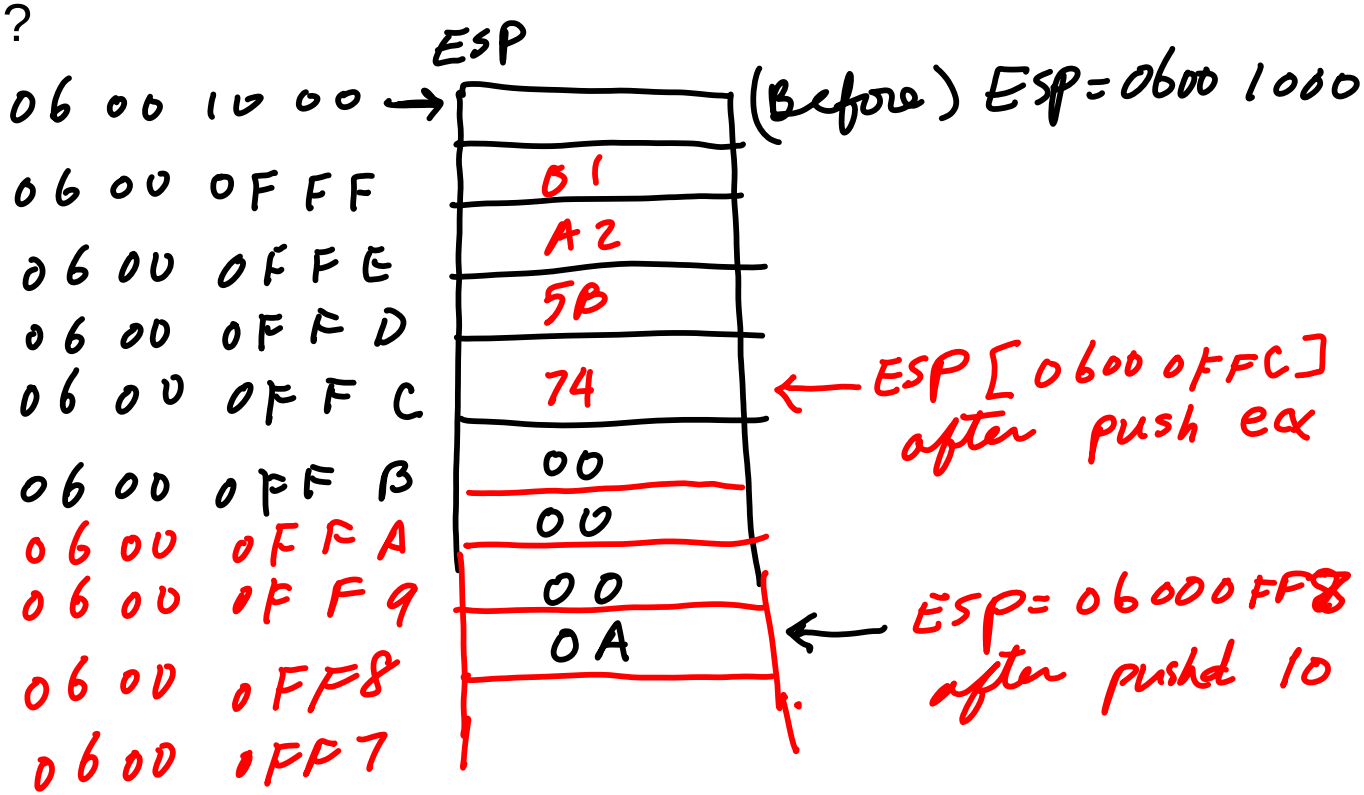
7. PUSH 66778899h — ESP=10001FE4

8. POP ECX — ECX=66778899 — ESP=10001FE8

9. POP EDX — EDX=AABBCCDD — ESP=10001FEC

Push Exercise

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



Push – Practice (sub 1)

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

Push-Pop Practice (sub 2)

- Before:
  - [ESP]=00 10 F8 3A
  - [EAX]=12 34 56 78
- Stack Diagram, [EAX], [EBX], & [ESP]
  - After
    - `Push eax`
    - `Pushd 30`
    - `Pop ebx`
    - `Pop ecx`

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*

Procedure in Coding

- In a code segment with body statements bracketed by PROC and ENDP directives giving procedure name

```
.CODE
procName   PROC
; procedure body
           ...
procName   ENDP
```

- Transfer Control to a Procedure
  - In the “main” program, use

    call *procName*
  - The next instruction executed will be the first one in the procedure
- Returning from a Procedure
  - In the procedure, use

    ret
  - The next instruction executed will be the one following the call in the “main” program

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
  - 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*
- *n* is added to ESP after the return address is popped
- 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 --- Our Default !!
- 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  sp,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

- 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
push arg3
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 12
```

Procedure Example – CallEX1.asm

CallEx1.asm - Notepad

File Edit Format View Help

```
; CallEx1.Asm

.586
.MODEL FLAT
.STACK 4096

.DATA
number1 DWORD   12
number2 DWORD   5

.CODE
main    PROC
        push    number2         ; 2nd parameter
        push    number1         ; 1st parameter
        call    fctn1           ; fctn1(number1, number2)
        add     esp, 8          ; remove parameters from stack

        mov     eax, 0          ; exit with return code 0
        ret
main    ENDP

; int fctn1(int x, int y)
; returns 3*x+7*y
fctn1   PROC
        push    ebp             ; save base pointer
        mov     ebp, esp        ; establish stack frame
        push    ebx             ; save EBX

        mov     eax, [ebp+8]    ; x
        imul    eax, 3          ; 3*x
        mov     ebx, [ebp+12]   ; y
        imul    ebx, 7          ; 7*y
        add     eax, ebx        ; 3*x + 7*y

        pop     ebx             ; restore EBX
        pop     ebp             ; restore EBP
        ret                     ; return
fctn1   ENDP
```

LST file

CallEx1_ADDR.lst - Notepad

File Edit Format View Help

```
                                .586
                                .MODEL FLAT
                                .STACK 4096

00000000                        .DATA
00000000 0000000C               number1 DWORD   12
00000004 00000005               number2 DWORD   5

00401020                        .CODE
00401020                        main    PROC
00401020  FF 35 00000004 R              push    number2         ; 2nd parameter
00401026  FF 35 00000000 R              push    number1         ; 1st parameter
0040102C  E8 00000009                   call    fctn1           ; fctn1(number1, number2)
00401031  83 C4 08                      add     esp, 8          ; remove parameters from stack

00401034  B8 00000000                   mov     eax, 0  ; exit with return code 0
00401039  C3                            ret
0040103A                        main    ENDP

                                ; int fctn1(int x, int y)
                                ; returns 3*x+7*y
0040103A                        fctn1   PROC
0040103A  55                            push    ebp             ; save base pointer
0040103B  8B EC                         mov     ebp, esp        ; establish stack frame
0040103D  53                            push    ebx             ; save EBX

0040103E  8B 45 08                      mov     eax, [ebp+8]    ; x
00401041  6B C0 03                      imul    eax, 3          ; 3*x
00401044  8B 5D 0C                      mov     ebx, [ebp+12]   ; y
00401047  6B DB 07                      imul    ebx, 7          ; 7*y
0040104A  03 C3                         add     eax, ebx        ; 3*x + 7*y

0040104C  5B                            pop     ebx             ; restore EBX
0040104D  5D                            pop     ebp             ; restore EBP
0040104E  C3                            ret                     ; return
0040104F                        fctn1   ENDP

                                END
```

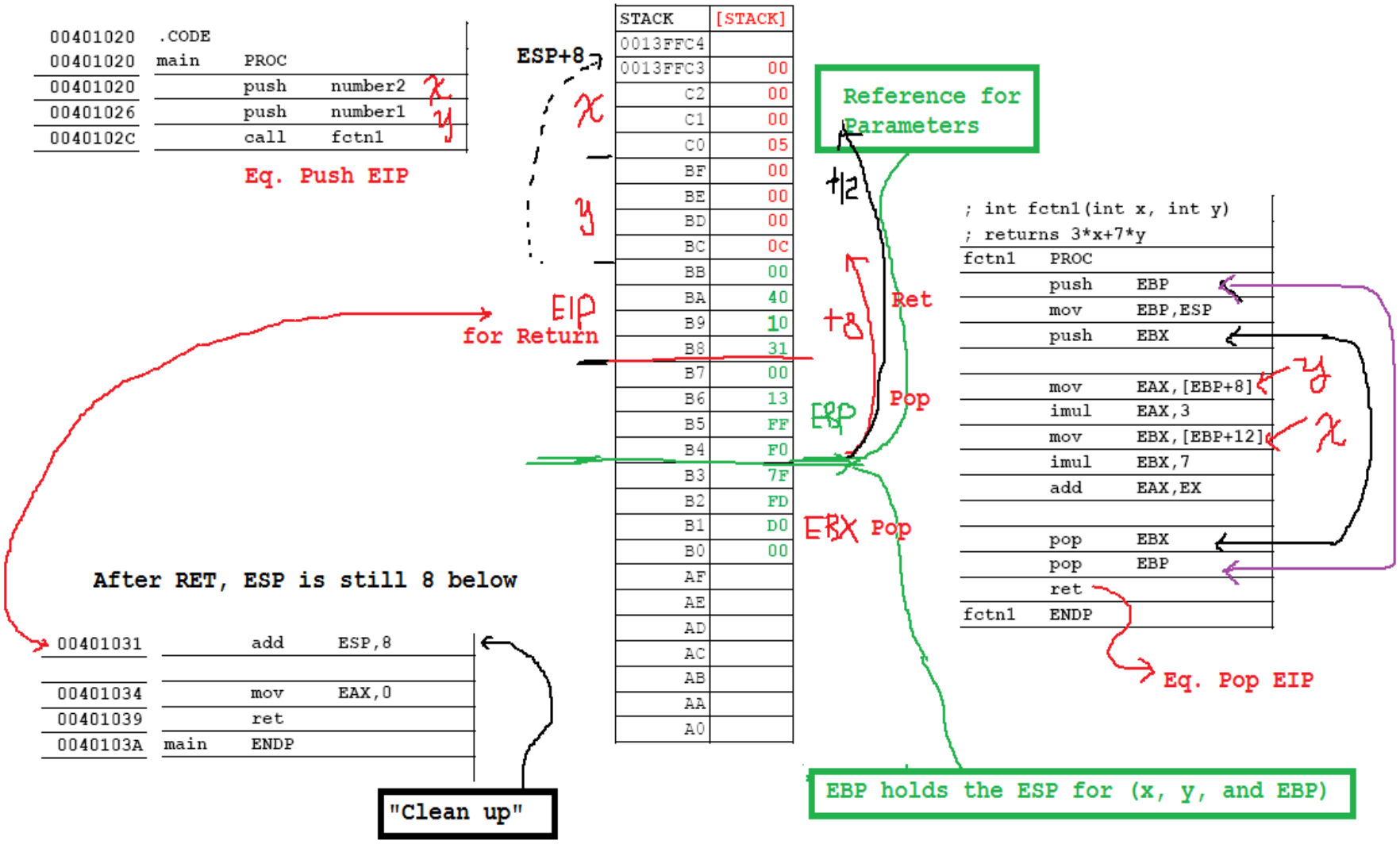
Follow EIP, ESP, and Result

ADDRESS	MACHINE CODE	LABEL	INSTRUCTION		EAX	EBX	EIP	ESP	EBP
		.DATA							
00000000	0000000C	number1	DWORD	12					
00000004	00000005	number2	DWORD	5					
00401020		.CODE			EAX	EBX	EIP	ESP	EBP
00401020		main	PROC		00000000	7FFDE000	00401020	0013FFC4	0013FFF0
00401020	FF 35 00000004 R		push	number2					
00401026	FF 35 00000000 R		push	number1					
0040102C	E8 00000009		call	fctn1					
00401031	83 C4 08		add	ESP,8					
00401034	B8 00000000		mov	EAX,0					
00401039	C3		ret						
0040103A		main	ENDP						
		; int fctn1(int x, int y)							
		; returns 3*x+7*y							
0040103A		fctn1	PROC						
0040103A	55		push	EBP					
0040103B	8B EC		mov	EBP,ESP					
0040103D	53		push	EBX					
0040103E	8B 45 08		mov	EAX,[EBP+8]					
00401041	6B C0 03		imul	EAX,3					
00401044	8B 5D 0C		mov	EBX,[EBP+12]					
00401047	6B DB 07		imul	EBX,7					
0040104A	03 C3		add	EAX,EBX					
0040104C	5B		pop	EBX					
0040104D	5D		pop	EBP					
0040104E	C3		ret						
0040104F		fctn1	ENDP						
		END							

STACK	[STACK]
0013FFC4	
0013FFC3	
C2	
C1	
C0	
BF	
BE	
BD	
BC	
BB	
BA	
B9	
B8	
B7	
B6	
B5	
B4	
B3	
B2	
B1	
B0	
AF	
AE	
AD	
AC	
AB	
AA	
A0	

STACK

Post-Mortem:


00401020 .CODE
00401020 main PROC
00401020 push number2
00401026 push number1
0040102C call fctn1
Eq. Push EIP
ESP+8
x
y
EIP for Return
STACK [STACK]
0013FFC4
0013FFC3 00
C2 00
C1 00
C0 05
BF 00
BE 00
BD 00
BC 0C
BB 00
BA 40
B9 10
B8 31
B7 00
B6 13
B5 FF
B4 F0
B3 7F
B2 FD
B1 D0
B0 00
AF
AE
AD
AC
AB
AA
A0
Reference for Parameters
+12
Ret
+8
EBP Pop
EBX Pop
; int fctn1(int x, int y)
; returns 3*x+7*y
fctn1 PROC
push EBP
mov EBP,ESP
push EBX
mov EAX,[EBP+8]
imul EAX,3
mov EBX,[EBP+12]
imul EBX,7
add EAX,EX
pop EBX
pop EBP
ret
fctn1 ENDP
Eq. Pop EIP
After RET, ESP is still 8 below
00401031 add ESP,8
00401034 mov EAX,0
00401039 ret
0040103A main ENDP
"Clean up"
EBP holds the ESP for (x, y, and EBP)

Summary

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 (this pops the return address to EIP)

MAIN CODE

1. Clean-up process (Add 4*N to ESP) --- N is the number of parameters pushed before Call.

Alternative 32-bit Procedure Options

- 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
- Passing an Address
  - **`lea`** instruction can put address of an argument in a register, and then the contents can be pushed on the stack (**Load Effective Address**)

```
lea  eax, minimum ;
push eax
```

PTR operator

- Register indirect mode:
  - The register contains the location of the data to be used in the instruction (not the data itself)
  - Example: `add eax, [edx]` ; when source and desiination is known as doublelword
    - » CF: `add eax DOWRD PTR [edx]`
  - Example: `mov [ebx], 0` ; ambiguous size. Source - byte, word, etc?
    - `Mov BYTE PTR [ebx], 0`

Procedure using Address Parameter - CallEx2.asm

CallEx2.asm - Notepad

File Edit Format View Help

```
;CallEx2.asm

.586
.MODEL FLAT
.STACK  4096

.DATA
minimum      DWORD    ?
maximum      DWORD    ?
nbrArray     DWORD    25, 47, 95, 50, 16, 84 DUP (?)

.CODE
main     PROC
         lea    eax, maximum   ; 4th parameter
         push   eax
         lea    eax, minimum   ; 3rd parameter
         push   eax
         pushd  5              ; 2nd parameter (number of elements)
         lea    eax, nbrArray  ; 1st parameter
         push   eax
         call   minMax         ; minMax(nbrArray, 5, minimum, maximum)
         add    esp, 16        ; remove parameters from stack

quit:    mov    eax, 0         ; exit with return code 0
         ret
main     ENDP
```

DUP: DUP directive tells the assembler to duplicate an expression a given number of times

ARR1 Byte 10 DUP (?); 10 uninitialized bytes

ARR2 DWORD 100 dup (0); 100 Dwords initialized as 0

Procedure using Address Parameter - CallEx2.asm

```
; void minMax(int arr[], int count, int& min, int& max);
; Set min to smallest value in arr[0],..., arr[count-1]
; Set max to largest value in arr[0],..., arr[count-1]
minMax  PROC
        push  ebp             ; save base pointer
        mov   ebp,esp         ; establish stack frame
        push  eax             ; save registers
        push  ebx
        push  ecx
        push  edx
        push  esi

        mov   esi,[ebp+8]     ; get address of array arr
        mov   ecx,[ebp+12]    ; get value of count
        mov   ebx, [ebp+16]   ; get address of min
        mov   edx, [ebp+20]   ; get address of max

        mov   DWORD PTR [ebx], 7fffffffh  ; largest possible integer
        mov   DWORD PTR [edx], 80000000h  ; smallest possible integer
        jecxz exitCode        ; exit if there are no elements

forLoop:
        mov   eax, [esi]      ; a[i]
        cmp   eax, [ebx]      ; a[i] < min?
        jnl   endIfSmaller    ; skip if not
        mov   [ebx], eax      ; min := a[i]
endIfSmaller:
        cmp   eax, [edx]      ; a[i] > max?
        jng   endIfLarger     ; skip if not
        mov   [edx], eax      ; max := a[i]
endIfLarger:
        add   esi, 4          ; point at next array element
        loop  forLoop         ; repeat for each element of array
```

Procedure using Address Parameter - CallEx2.asm

```
exitCode:
        pop     esi             ; restore registers
        pop     edx
        pop     ecx
        pop     ebx
        pop     eax
        pop     ebp
        ret                     ; return
minMax  ENDP
END
```

LST File

```
00404000                        .DATA
00404000 00000000               minimum     DWORD   ?
00404004 00000000               maximum     DWORD   ?
00404008 00000019               nbrArray    DWORD   25, 47, 95, 50, 16, 84 DUP (?)
0040400C 0000002F
00404010 0000005F
00404014 00000032
00404018 00000010
0040401C 00000054 [
          00000000
         ]

00401020                        .CODE
00401020                        main    PROC
00401020  8D 05 00000004 R              lea   eax, maximum  ; 4th parameter
00401026  50                            push  eax
00401027  8D 05 00000000 R              lea   eax, minimum  ; 3rd parameter
0040102D  50                            push  eax
0040102E  6A 05                         pushd 5             ; 2nd parameter (number of elements)
00401030  8D 05 00000008 R              lea   eax, nbrArray ; 1st parameter
00401036  50                            push  eax
00401037  E8 00000009                   call  minMax        ; minMax(nbrArray, 5, minimum, maximum)
0040103C  83 C4 10                      add   esp, 16       ; remove parameters from stack

0040103F  B8 00000000           quit:   mov   eax, 0        ; exit with return code 0
00401044  C3                            ret
00401045                        main    ENDP
```

LST file --- Procedure part

```
00401045                        minMax  PROC
00401045  55                            push  ebp             ; save base pointer
00401046  8B EC                         mov   ebp,esp         ; establish stack frame
00401048  50                            push  eax             ; save registers
00401049  53                            push  ebx
0040104A  51                            push  ecx
0040104B  52                            push  edx
0040104C  56                            push  esi

0040104D  8B 75 08                      mov   esi,[ebp+8]     ; get address of array arr
00401050  8B 4D 0C                      mov   ecx,[ebp+12]    ; get value of count
00401053  8B 5D 10                      mov   ebx, [ebp+16]   ; get address of min
00401056  8B 55 14                      mov   edx, [ebp+20]   ; get address of max

00401059  C7 03 7FFFFFFF                mov   DWORD PTR [ebx], 7fffffffh  ; largest possible integer
0040105F  C7 02 80000000                mov   DWORD PTR [edx], 80000000h  ; smallest possible integer
00401065  E3 13                         jecxz exitCode        ; exit if there are no elements

00401067                        forLoop:
00401067  8B 06                         mov   eax, [esi]      ; a[i]
00401069  3B 03                         cmp   eax, [ebx]      ; a[i] < min?
0040106B  7D 02                         jnl   endIfSmaller    ; skip if not
0040106D  89 03                         mov   [ebx], eax      ; min := a[i]
0040106F                        endIfSmaller:
0040106F  3B 02                         cmp   eax, [edx]      ; a[i] > max?
00401071  7E 02                         jng   endIfLarger     ; skip if not
00401073  89 02                         mov   [edx], eax      ; max := a[i]
00401075                        endIfLarger:
00401075  83 C6 04                      add   esi, 4          ; point at next array element
00401078  E2 ED                         loop  forLoop         ; repeat for each element of array

0040107A                        exitCode:
0040107A  5E                            pop    esi            ; restore registers
0040107B  5A                            pop    edx
0040107C  59                            pop    ecx
0040107D  5B                            pop    ebx
0040107E  58                            pop    eax
0040107F  5D                            pop    ebp
00401080  C3                            ret                   ; return
00401081                        minMax  ENDP
```

Follow EIP, ESP, and Result

ADDRESS	MACHINE CODE	LABEL	INSTRUCTION		EAX	EIP	ESP
		.DATA					
00404000	00000000	minimum	DWORD	?			
00404004	00000005	maximum	DWORD	?			
00404008	00000019	nbArray	DWORD	25,47,95,50,16, 84 DUP (?)			
0040400C	0000002F						
00404010	0000005F						
00404014	00000032						
00404018	00000010						
0040401C	00000054 [00000000]						
00401020		.CODE			EAX	EIP	ESP
00401020		main	PROC		00000000	00401020	0013FFC4
00401020	8D 05 00000004 R		lea	eax,maximum			
00401026	50		push	eax			
00401027	8D 05 00000000 R		lea	eax,mimimum			
0040102D	50		push	eax			
0040102E	6A 05		pushd	5			
00401030	8D 05 00000008 R		lea	eax,nbrArray			
00401036	50		push	eax			
00401037	E8 00000009		call	mimMax			
0040103C	83 C4 10		add	esp,16			
0040103F	B8 00000000	quit:	mov	eax,0			
00401044	C3		ret				
00401045		main	ENDP				
		;void minMax (int arr[], int count, int &min, int &max) ; set min to smalles tvalue; and max to largest value					
		minMax	PROC				
00401045	55		push	ebp			
00401046	8B EC		mov	ebp,esp			
00401048	50		push	eax			
00401049	53		push	ebx			
0040104A	51		push	ecx			
0040104B	52		push	edx			
0040104C	56		push	esi			
0040104D	8B 75 08		mov	esi,[ebp+8]			

STACK

STACK	[STACK]
0013FFC4	
0013FFC3	
C2	
C1	
C0	
BF	
BE	
BD	
BC	
BB	
BA	
B9	
B8	
B7	
B6	
B5	
B4	
B3	
B2	
B1	
B0	
AF	
AE	
AD	
AC	
AB	

Follow EIP, ESP, and Result - continued

```
00401050  8B 4D 0C              mov     ecx,[ebp+12]
00401053  8B 5D 10              mov     ebx,[ebp+16]
00401056  8B 55 14              mov     edx,[ebp+20]
00401059  C7 03 7FFFFFFF        mov     DWORD PTR [ebx],7FFFFFFF
0040105F  C7 02 80000000        mov     DWORD PTR [edx],80000000
00401065  E3 13                 jecxz   exitCode
00401067  8B 06          forLoop: mov     eax,[esi]
00401069  3B 03                 cmp     eax,[ebx]
0040106B  7D 02                 jnl     endIfSmaller
0040106D  89 03                 mov     [ebx],eax
0040106F  3B 02          endIfSmaller: cmp eax,[edx]
00401071  7E 02                 jng     endIfLarger
00401073  89 02                 mov     [edx],eax
00401075  83 C6 04       endIfLarger: add  esi,4
00401078  E2 ED                 loop    forLoop
0040107A  5E             exitCode: pop    esi
0040107B  5A                    pop     edx
0040107C  59                    pop     ecx
0040107D  5B                    pop     ebx
0040107E  58                    pop     eax
0040107F  5D                    pop     edp
00401080  C3                    ret
00401081                 minMax  ENDP
                         END
```

						AA
						A9
						A8
						A7
						A6
						A5
						A4
						A3
						A2
						A1
						A0
						9F
						9E
						9D
						9C
						9B
						9A
						99
						98
						97
						96
						95
						94

	0	1	2	3	4	5	
00404000							min
00404004							max
00404008							
0040400C							
00404010							
00404014							
00404018							
0040401C							