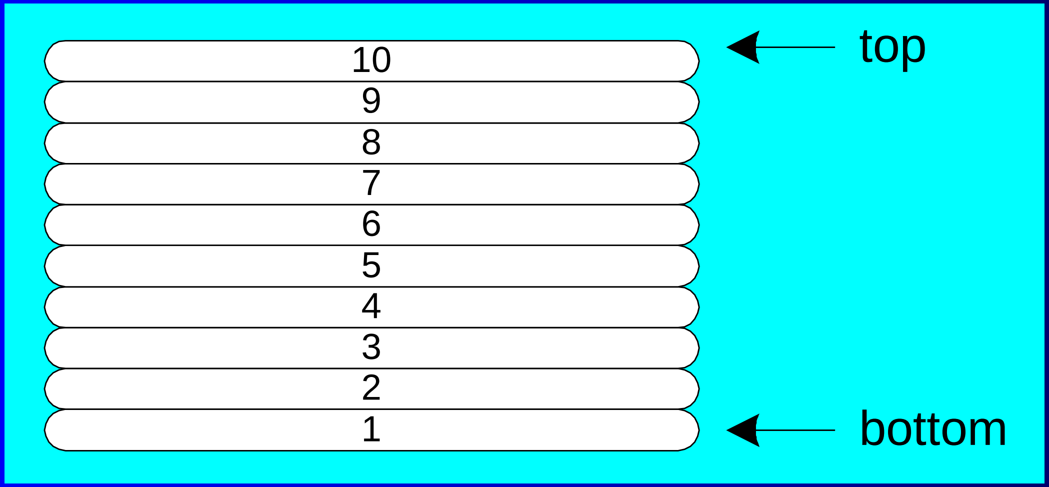
Procedures

Irvine, Kip R. Assembly Language for x86 Processors 7/e, 2015.

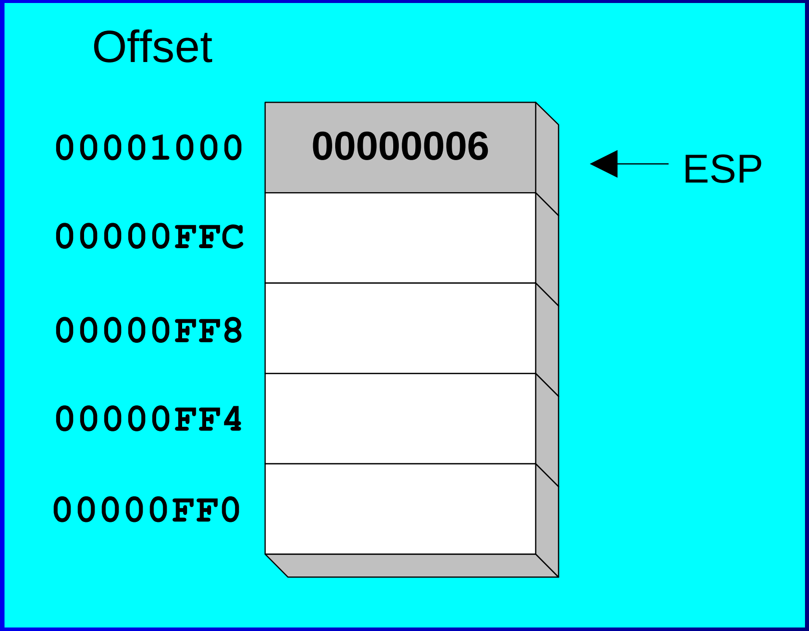
Runtime Stack

- Imagine a stack of plates . . .
  - plates are only added to the top
  - plates are only removed from the top
  - LIFO structure



Runtime Stack

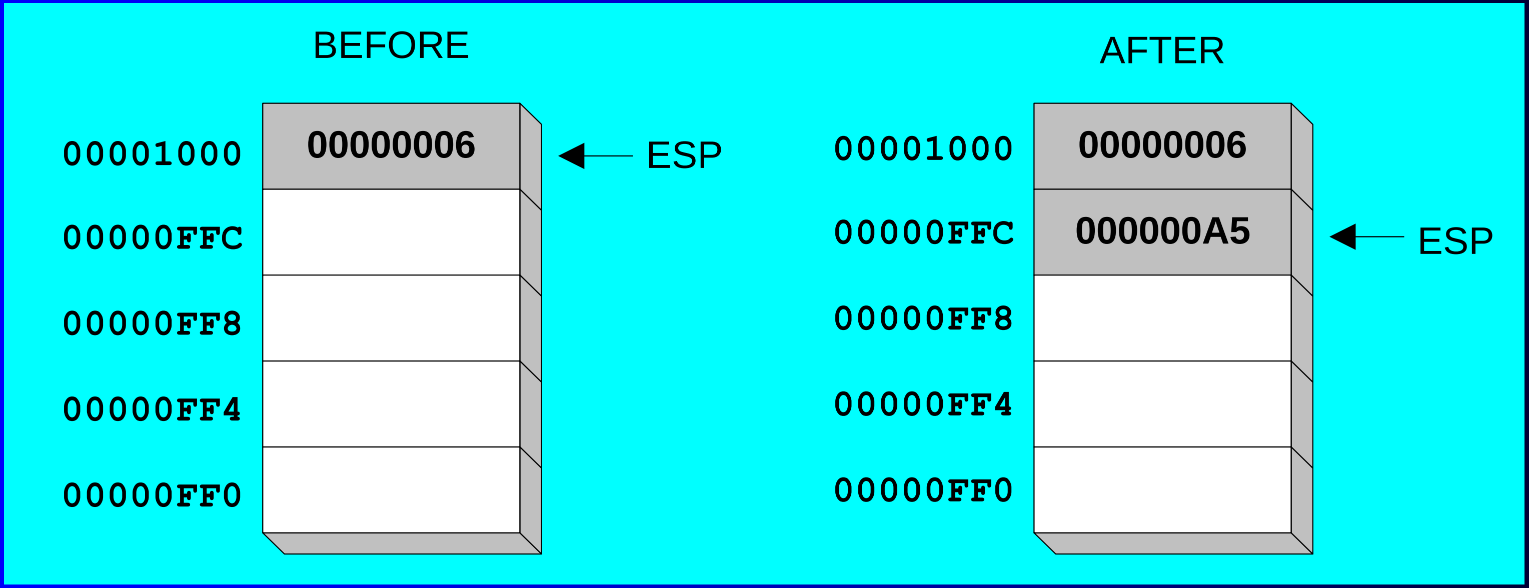
- Managed by the CPU, using two registers
  - SS (stack segment)
  - ESP (stack pointer) *



* SP in Real-address mode

PUSH Operation (1 of 2)

- A 32-bit push operation decrements the stack pointer by 4 and copies a value into the location pointed to by the stack pointer.



BEFORE

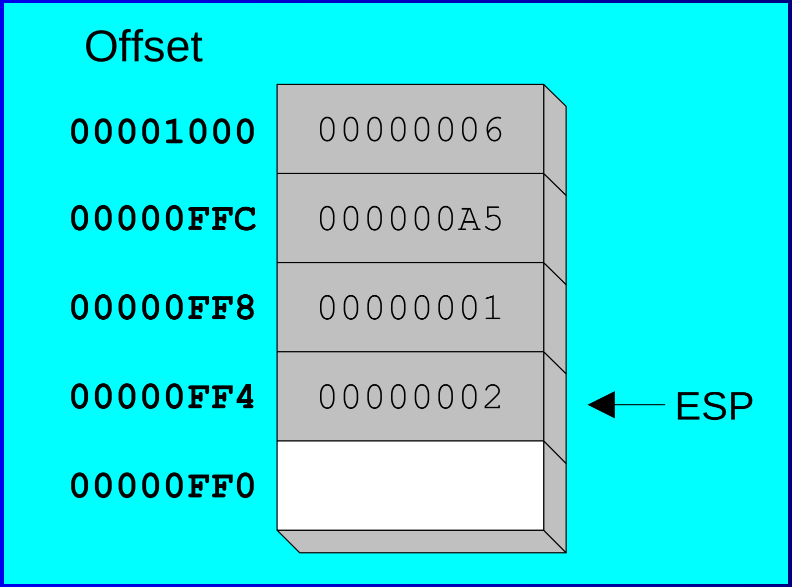
00001000	00000006	ESP
00000FFC		
00000FF8		
00000FF4		
00000FF0		

AFTER

00001000	00000006	
00000FFC	000000A5	ESP
00000FF8		
00000FF4		
00000FF0		

PUSH Operation (2 of 2)

- Same stack after pushing two more integers:

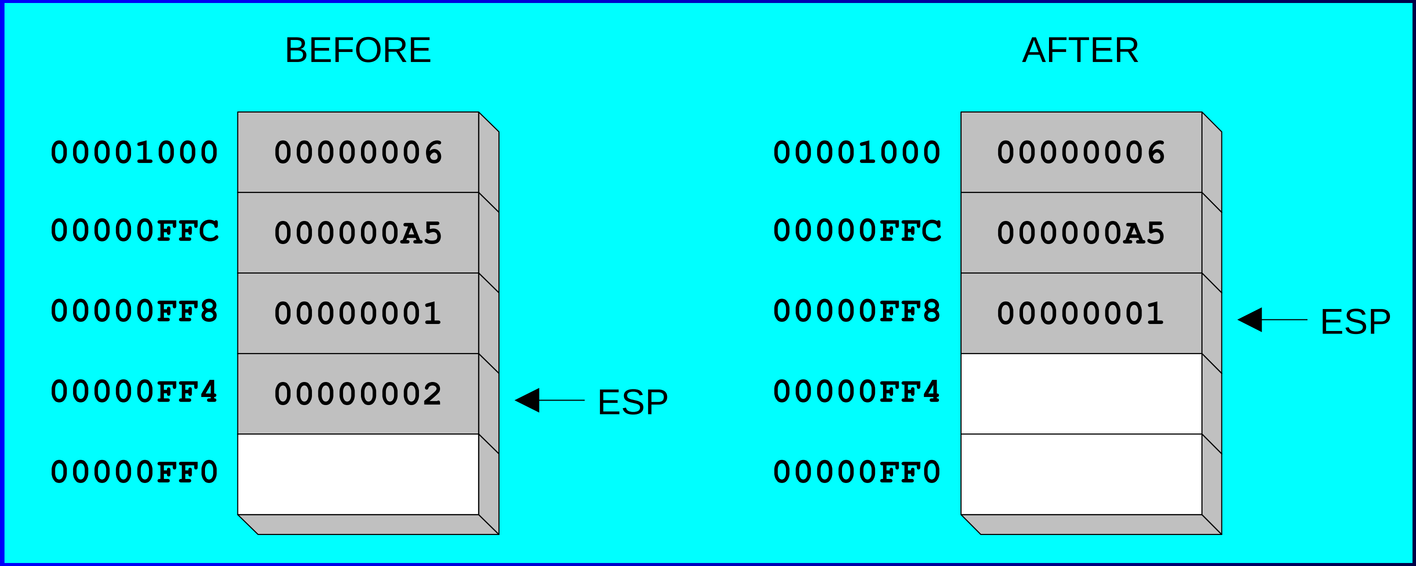


Offset		
00001000	00000006	
00000FFC	000000A5	
00000FF8	00000001	
00000FF4	00000002	ESP
00000FF0		

The stack grows downward. The area below ESP is always available (unless the stack has overflowed).

POP Operation

- Copies value at stack[ESP] into a register or variable.
- Adds *n* to ESP, where *n* is either 2 or 4.
  - value of *n* depends on the attribute of the operand receiving the data



BEFORE

00001000	00000006	
00000FFC	000000A5	
00000FF8	00000001	
00000FF4	00000002	← ESP
00000FF0		

AFTER

00001000	00000006	
00000FFC	000000A5	
00000FF8	00000001	← ESP
00000FF4		
00000FF0		

Irvine, Kip R. Assembly Language for x86 Processors 7/e, 2015.

PUSH and POP Instructions

- PUSH syntax:
  - PUSH *r/m16*
  - PUSH *r/m32*
  - PUSH *imm32*
- POP syntax:
  - POP *r/m16*
  - POP *r/m32*

Irvine, Kip R. Assembly Language for x86 Processors 7/e, 2015.

Using PUSH and POP

Save and restore registers when they contain important values. PUSH and POP instructions occur in the opposite order.

```
push esi                          ; push registers
push ecx
push ebx

mov  esi,OFFSET dwordVal          ; display some memory
mov  ecx,LENGTHOF dwordVal
mov  ebx,TYPE dwordVal
call DumpMem

pop  ebx                          ; restore registers
pop  ecx
pop  esi
```

Example: Nested Loop

When creating a nested loop, push the outer loop counter before entering the inner loop:

```
    mov ecx,100             ; set outer loop count
L1:                         ; begin the outer loop
    push ecx                ; save outer loop count

    mov ecx,20              ; set inner loop count
L2:                         ; begin the inner loop
    ;
    ;
    loop L2                 ; repeat the inner loop

    pop ecx                 ; restore outer loop count
    loop L1                 ; repeat the outer loop
```

Example: Reversing a String

- Use a loop with indexed addressing
- Push each character on the stack
- Start at the beginning of the string, pop the stack in reverse order, insert each character back into the string
- Source code

- Q: Why must each character be put in EAX before it is pushed?

Because only word (16-bit) or doubleword (32-bit) values can be pushed on the stack.

Related Instructions

- PUSHFD and POPFD
  - push and pop the EFLAGS register
- PUSHAD pushes the 32-bit general-purpose registers on the stack
  - order: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI
- POPAD pops the same registers off the stack in reverse order
  - PUSHA and POPA do the same for 16-bit registers

Your Turn . . .

- Write a program that does the following:
  - Assigns integer values to EAX, EBX, ECX, EDX, ESI, and EDI
  - Uses PUSHAD to push the general-purpose registers on the stack
  - Using a loop, your program should pop each integer from the stack and display it on the screen

Defining and Using Procedures

- Creating Procedures
- Documenting Procedures
- Example: SumOf Procedure
- CALL and RET Instructions
- Nested Procedure Calls
- Local and Global Labels
- Procedure Parameters
- Flowchart Symbols
- USES Operator

Creating Procedures

- A procedure is the ASM equivalent of a Java or C++ method/function
- Following is an assembly language procedure named sample:

```
sample PROC
     .
     .
    ret
sample ENDP
```

Irvine, Kip R. Assembly Language for x86 Processors 7/e, 2015.

Example: SumOf Procedure

```
SumOf PROC

; Calculates and returns the sum of three 32-bit integers.
; Receives: EAX, EBX, ECX, the three integers

    add eax,ebx
    add eax,ecx
    ret
SumOf ENDP
```

Irvine, Kip R. Assembly Language for x86 Processors 7/e, 2015.

CALL and RET Instructions

- The CALL instruction calls a procedure
  - pushes offset of next instruction on the stack
  - copies the address of the called procedure into EIP
- The RET instruction returns from a procedure
  - pops top of stack into EIP

CALL-RET Example (1 of 2)

0000025 is the offset of the instruction immediately following the CALL instruction

00000040 is the offset of the first instruction inside MySub

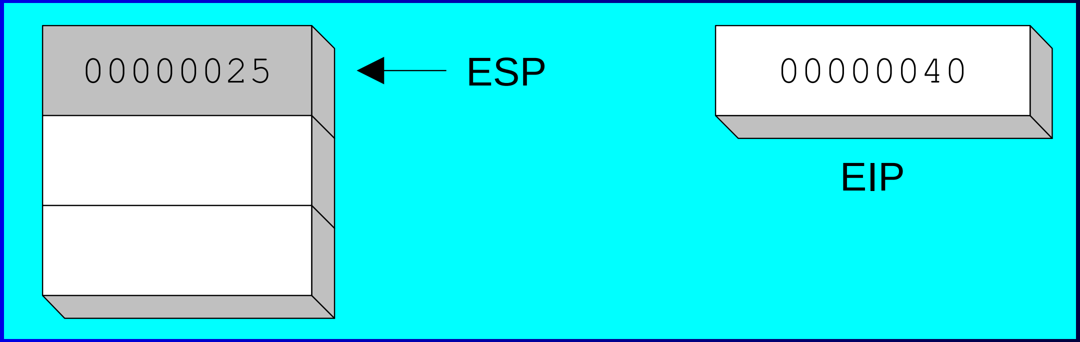
```
main PROC
    00000020 call MySub
    00000025 mov eax,ebx
    .
    .
main ENDP

MySub PROC
    00000040 mov eax,edx
    .
    .
    ret
MySub ENDP
```

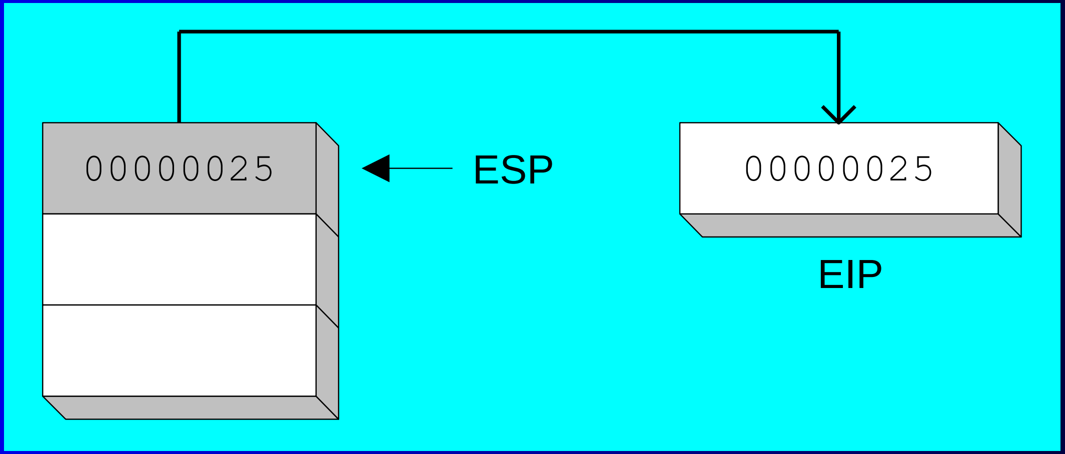
Irvine, Kip R. Assembly Language for x86 Processors 7/e, 2015.

CALL-RET Example (2 of 2)

The CALL instruction pushes 00000025 onto the stack, and loads 00000040 into EIP



The RET instruction pops 00000025 from the stack into EIP



(stack shown before RET executes)

Irvine, Kip R. Assembly Language for x86 Processors 7/e, 2015.

Thanks!