# CS 271
# Computer Architecture & Assembly Language

Lecture 12

2/10/22, Thursday


Oregon State University

# Odds and Ends

- Due Sunday 2/13 11:59 pm:
  - Assignment 4

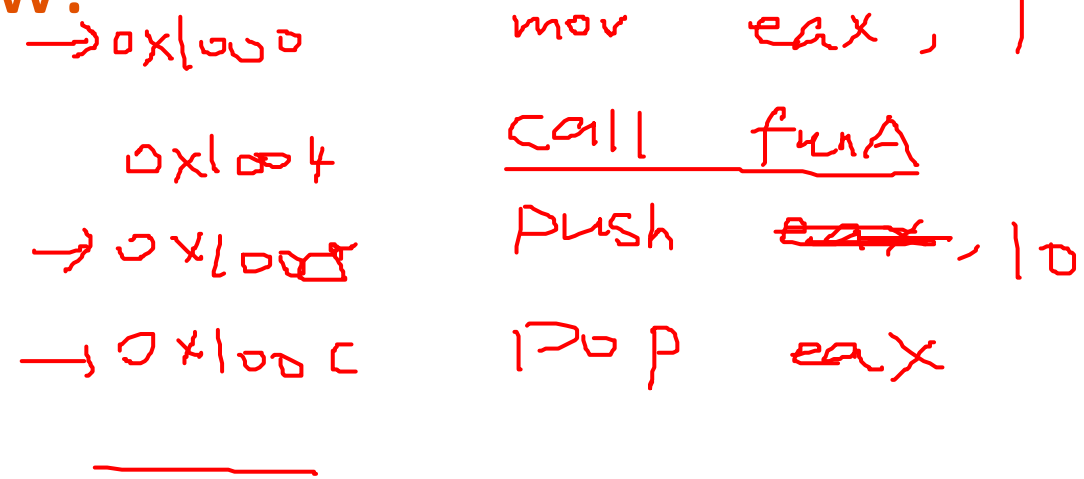
# Lecture Topics:

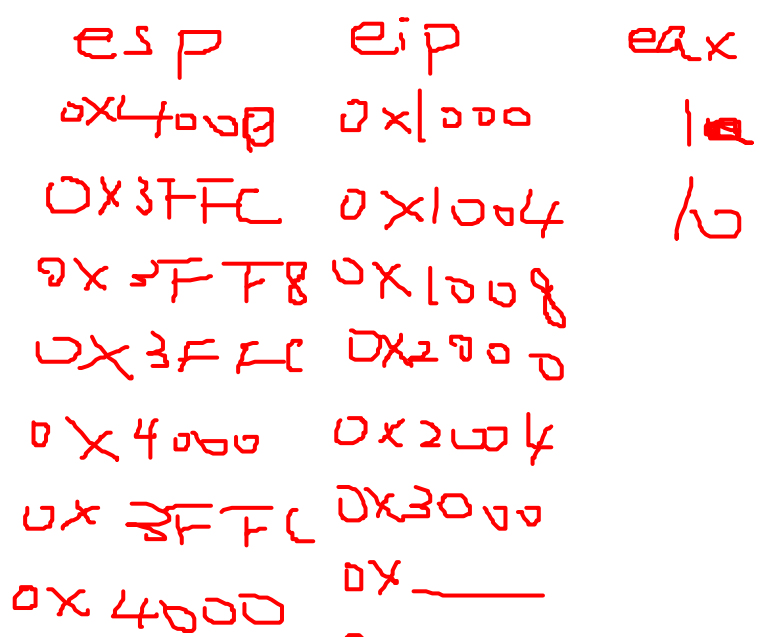
- Passing Parameters on the System Stack

# Passing Parameters on the System Stack

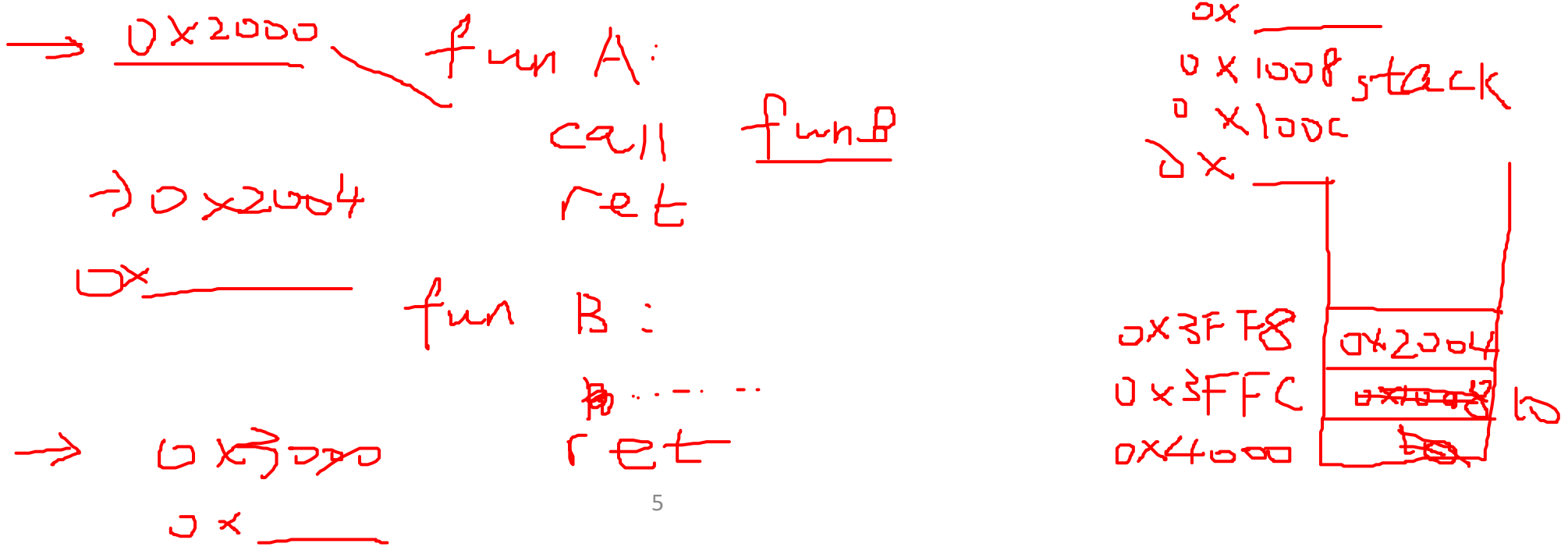
# Quick Review:

- Push
- Pop




- Call
- Ret



# RET Instruction

ret <=> pop EIP

- Pops stack into the instruction pointer (EIP)
- Syntax:
  - **RET**
  - **RET n**
- Optional operand *n* causes *n* to be added to the stack pointer after EIP is assigned a value
  - Equivalent to popping the return address **and n additional bytes** off the stack

# Stack Frame

- Also known as an activation record
- Area of the stack used for a procedure's return address, passed parameters, saved registers, and local variables

- Created by the following steps:
  - Calling program pushes arguments onto the stack and calls the procedure
  - The called procedure pushes EBP onto the stack, and sets EBP to ESP

base pointer

# Addressing Modes

- Immediate — Constants, literal, absolute address
- Direct — Contents of referenced memory address
- Register — Contents of register

- Register indirect — Access memory through address in a register

- Indexed — Array name using element “distance” in register
- Base-indexed — Start address in one register; offset in another, add and access memory

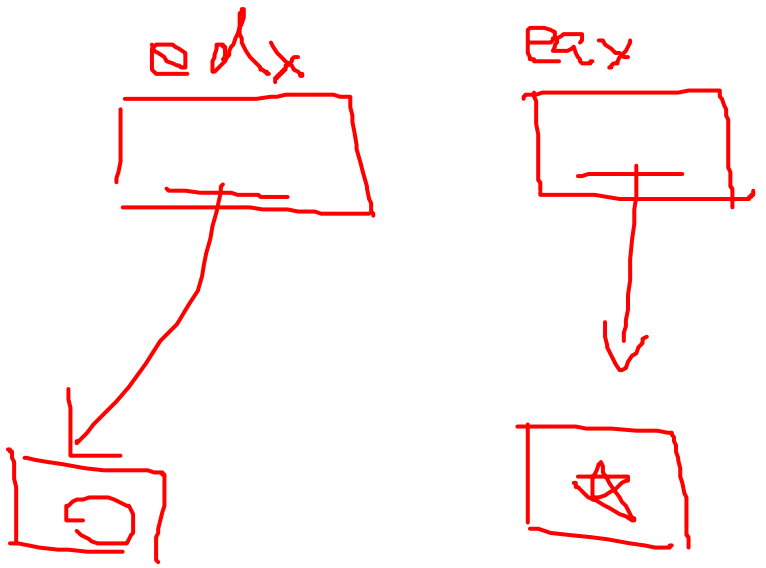
- Stack — Memory area specified and maintained as a stack; Stack pointer in ESP
- Offset — Memory address; may be computed

# Register Indirect Mode

- [reg] means "contents of memory at the address in *reg*"
- It is OK to add a constant (named or literal)
  - Example: mov [edx+12], eax

- We have used register indirect with **esp** to reference the value at the top of the system stack

- Note: register indirect is a **memory reference**
  - There are no memory-memory instruction
  - E.g., mov [edx], [eax] is WRONG!



# Explicit Access to Stack Parameters

- A procedure can explicitly access stack parameters using constant offsets from EBP.
  - Example: [ebp + 8]
- EBP is often called the base pointer or frame pointer because it is (should be) set to the base address of the stack frame
- EBP should not change value during the procedure
- EBP must be restored to its original value when the procedure returns
- Remember that the return address is pushed onto the stack **after** the parameters are pushed
- Programmer is responsible for managing the stack.

# Stack Frame Example

SumTwo ( x , y , &z ) ;

```
.data
x DWORD    175
y DWORD    37
Z DWORD ?
.code
main    PROC
  push x
  push y
  push OFFSET z
  call SumTwo
  ...
```

| SYSTEM STACK (after call sumTwo) | |
|---|---|
| | |
| [ESP] | *return @* |
| [ESP+4] | @ z |
| [ESP+8] | 37 |
| [ESP+12] | 175 |

Note: @ means "address of"

# Stack Frame Example

```
SumTwo PROC
   push ebp
   mov  ebp,esp
   mov   eax,[ebp+16]
        ;175 in eax
   add   eax,[ebp+12]
        ;175+37 = 212 in eax
   mov   ebx,[ebp+8]
        ;@z in ebx
   mov   [ebx],eax
        ;store 212 in z
   pop   ebp
   ret   12
SumTwo  ENDP
```

| SYSTEM STACK (after mov ebp, esp) | |
|---|---|
| [EBP] | *old EBP* |
| [EBP+4] | *return @* |
| [EBP+8] | @ z |
| [EBP+12] | 37 |
| [EBP+16] | 175 |

- Why don’t we just use ESP instead of EBP?
  - Pushes and pops inside the procedure might cause us to lose the base of the stack frame.

# Trouble-Avoidance Tips

- Save and restore registers when they are modified by a procedure.
  - Exception: a register that returns a function result

- Do not pass an immediate value or variable contents to a procedure that expects a reference pointer.
  - Dereferencing it as an address will likely cause a general-protection fault.

# Demo