Function Call and Return

- The x86 instruction set itself only provides for transfer of control (jump) and return
- Stack is used to capture return address and recover it
- Everything else – parameter passing, stack frame organization, register usage – is a matter of convention and not defined by the hardware
call and ret Instructions

call label
- Push address of next instruction and jump
- \texttt{esp <- esp - 4; memory[esp] <- eip}
- \texttt{eip <- address of label}

ret
- Pop address from top of stack and jump
- \texttt{eip <- memory[esp]; esp <- esp + 4}
- **WARNING!** The word on the top of the stack had better be an address, not some leftover data
enter and leave

- Complex instructions for languages with nested procedures
  - enter can be slow on current CPUs – best avoided
    - i.e., don’t use it in your project
  - leave is equivalent to
    - mov esp,ebp
    - pop ebp
    - and is generated by many compilers. Fits in 1 byte, saves space. Not clear if it’s any faster.
Win 32 C Function Call Conventions

- Wintel code obeys the following conventions for C programs
  - Note: calling conventions normally designed very early in the instruction set/basic software design. Hard (e.g., basically impossible) to change later.

- C++ augments these conventions to include the "this" pointer
- We'll use these conventions in our code
Win32 C Register Conventions

- These registers must be restored to their original values before a function returns, if they are altered during execution.
  - esp, ebp, ebx, esi, edi
  - Traditional: push/pop from stack to save/restore

- A function may use the other registers (eax, ecx, edx) without having to save/restore.

- A 32-bit function result is expected to be in eax when the function returns.
Call Site

- Caller is responsible for
  - Pushing arguments on the stack from right to left (allows implementation of varargs)
  - Execute call instruction
  - Pop arguments from stack after return
    - For us, this means add 4*(# arguments) to esp after the return, since everything is either a 32-bit variable (int, bool), or a reference (pointer)
Call Example

\[ n = \text{sumOf}(17, 42) \]

- push 42
- push 17
- call \text{sumOf}

\[
\begin{align*}
\text{add} & \quad \text{esp, 8} \\
\text{mov} & \quad [\text{ebp} + \text{ offset }_n], \text{eax}
\end{align*}
\]

; push args
; jump &
; push addr
; pop args
; store result
Callee

- Called function must do the following
  - Save registers if necessary
  - Allocate stack frame for local variables
  - Execute function body
  - Ensure result of non-void function is in eax
  - Restore any required registers if necessary
  - Pop the stack frame
  - Return to caller
Win32 Function Prologue

- The code that needs to be executed before the statements in the body of the function are executed is referred to as the prologue.

- For a Win32 function \( f \), it looks like this:
  
  ```
  f:  push ebp
  mov ebp,esp
  sub esp,"# bytes needed"

  ; save old frame pointer
  ; new frame ptr is top of stack after arguments and return address are pushed
  ; allocate stack frame (size must be multiple of 4)
  ```
Win32 Function Epilogue

- The epilogue is the code that is executed for a return statement (or if execution “falls off” the bottom of a void function)

- For a Win32 function, it looks like this:
  ```
  mov eax,"function result"
  ; put result in eax if not already
  ; there (if non-void function)
  mov esp,ebp
  ; restore esp to old value
  ; before stack frame allocated
  pop ebp
  ; restore ebp to caller’s value
  ret
  ; return to caller
  ```
Example Function

Source code

```c
int sumOf(int x, int y) {
    int a, int b;
    a = x;
    b = a + y;
    return b;
}
```
int sumOf(int x, int y) {
    int a, int b;
    a = x;
    b = a + y;
    return b;
}
Assembly Language Version

```asm
;; int sumOf(int x, int y) {
;; int a, int b;
sumOf:
  push ebp ; prologue
  mov ebp,esp
  sub esp, 8

  ;; a = x;
  mov eax,[ebp+8]
  mov [ebp-4],eax

;; b = a + y;
  mov eax,[ebp-4]
  add eax,[ebp+12]
  mov [ebp-8],eax

  ;; return b;
  mov eax,[ebp-8]
  pop ebp
  ret

  ;; }
}
```

Stack Frame:
- `esp` points to `42`
- `ebp` points to `17`
- `eax` points to `59`
- `ebp` is saved as `old ebp`