x86 Assembly Cont.

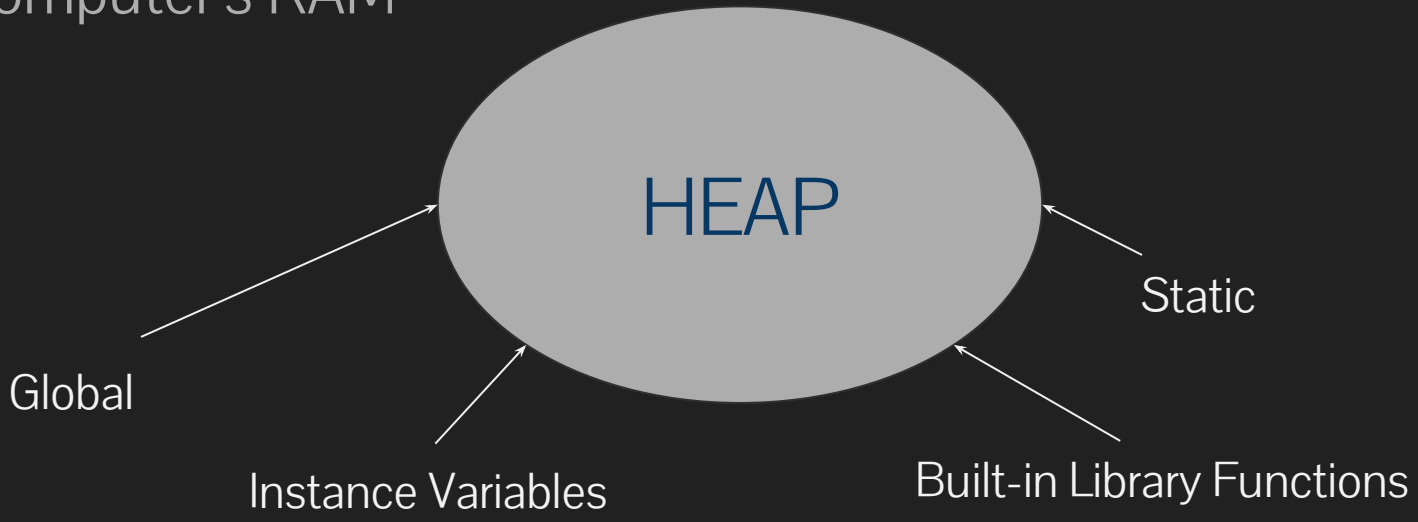
SSTCTF - by Tamir Enkhjargal

Inside a C Program and in Assembly

- Heap
- Stack
- Registers
- Instructions

The Heap

- The Heap is used for memory allocation, found in the computer's RAM



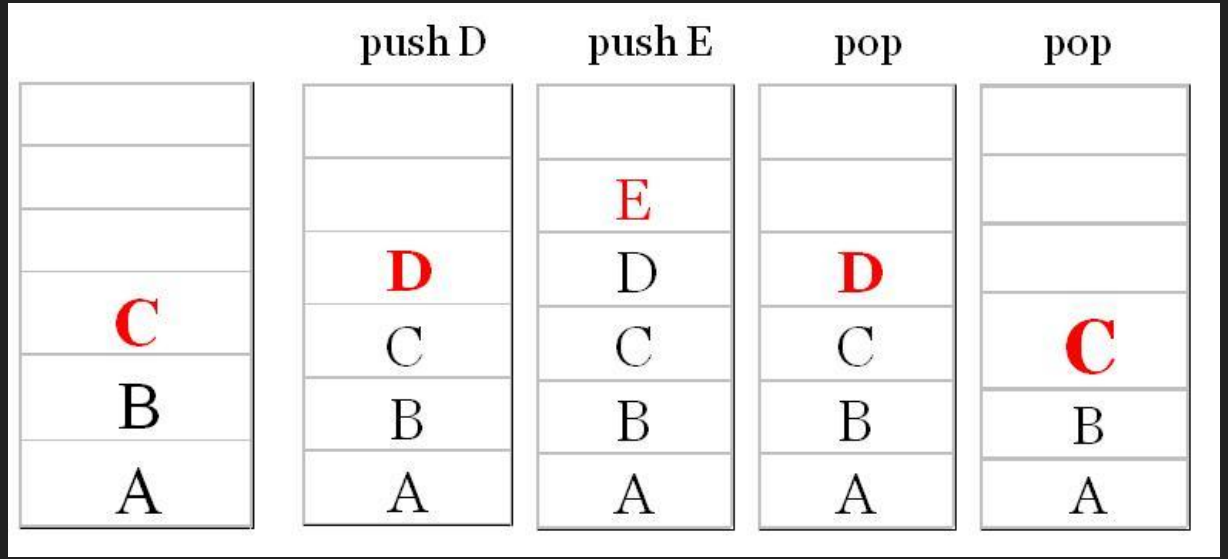
Registers

- Small storage containment that holds memory addresses and values
  - All 8 Bytes or less

- Standard Registers in x86
  - Eax, ebx, ecx, edx ← General registers
  - Ebp, esp, eip ← Reserved registers

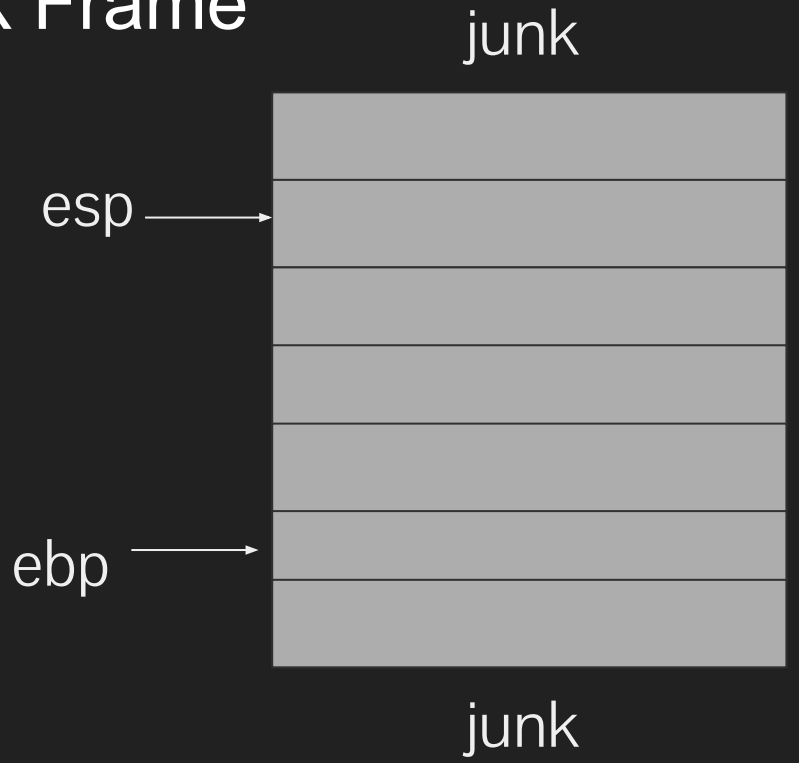
The Stack

- Two overall important functions, push and pop
- Each stack has an address. The higher you go on the stack, the lower the memory address



Stack Frame

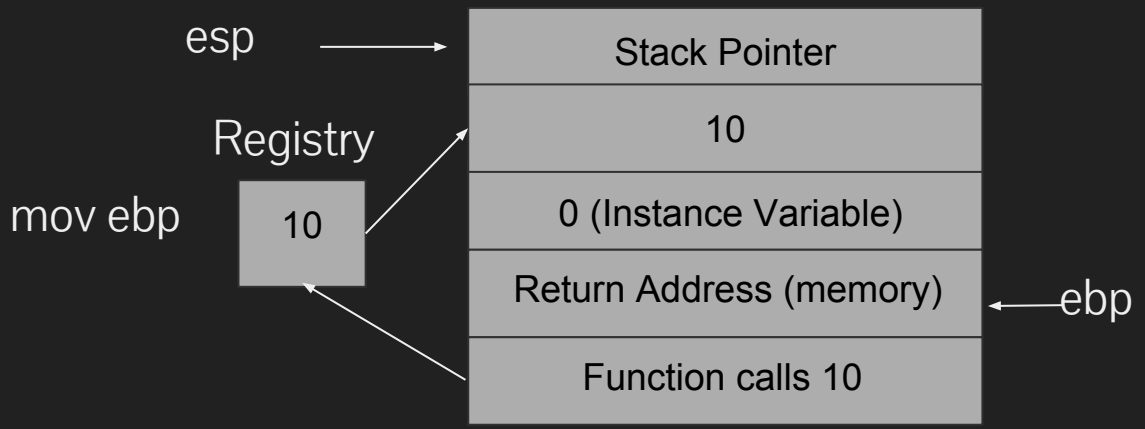
- Each stack has a frame
- esp = stack pointer ← Calls beginning
- ebp = base pointer ← Calls end



Stack Frame Example

In the C program:

- #include <stdio.h>
- void func(int x) {
  - int a = 0;
  - int b = x; }
- int main() {
  - func(15); }

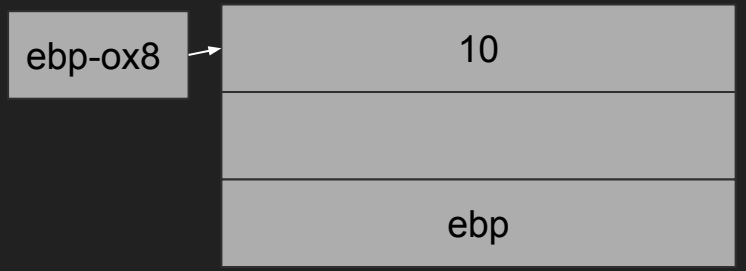


The Instruction in Assembly

- This is where you have your operation and arguments
- Such as mov arg1, arg2 (Move arg2 to arg1)
- Or move from registries
  - mov eax, ebp, mov ea… etc.

Instructions Example

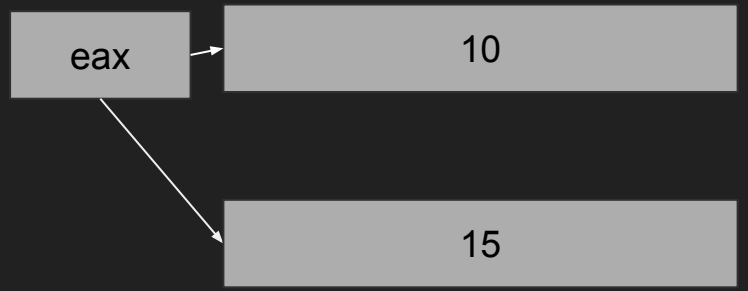
- mov arg1, arg2
- mov eax, ebp-ox8
- mov eax, [ebp-ox8]
- Square brackets ‘dereference’ the memory address and return the value it's pointing to



The Add Instruction

- add arg1, arg2

(arg1 + arg2)



- add eax, 0x5
- eax = 15
- This is the exact same as the sub instruction (subtract)

More Instructions

- push arg
- pop arg
- lea reg, addr (load effective address, used for getting a pointer from a memory addr)
- cmp arg1, arg2 (compare is similar to subtract, but returns a flag 0, <0, or >0)
- jmp addr (this is the follow up of compare, and just checks the state of the flag)
  - je, jne, jg, etc.
- call <func> = push eip, jmp <func> (calling makes the eip go to the top of stack, then jumps to it)