15-213 Recitation 4

Introduction to Computer Systems

Fahim Dalvi

19 September, 2013

Today

- Bomblab!
- Assembly
  - Control flow
    - Loops
  - Procedures

Bomblab

- Due: Monday, 23rd September
- Questions?
- Some hints
  - sscanf → just like scanf, but reads from a string rather than stdin
    - The function returns the number of input items successfully matched and assigned, which can be fewer than provided for, or even zero in the event of an early matching failure
  - Difference between rax/eax

Assembly – Reminder!

- Registers
  - *eip* (x86), *rip* (x86-64)
  - esp and ebp (x86)
  - *eax, ebx, ecx, edx, esi, edi* (x86)
  - *rax, rbx, rcx, rdx, rsi, rdi, r8, r9, r10, r11, r12, r13, r14, r15* and sometimes *rbp* (x86-64)

- Instructions
  - mov, lea
  - add, sub, imull …
  - or, and …
  - test, cmp
  - jmp, set

Lets trace!

```
push    %rbp
mov     %rsp,%rbp
sub     $0x10,%rsp
movl    $0x0,-0x4(%rbp)
mov     $0x400614,%edi
callq   400410 <puts@plt>
addl    $0x1,-0x4(%rbp)
cmpl    $0x9,-0x4(%rbp)
jle     40053b <secret+0xf>
leaveq
retq
```

Line: 40053b

Lets trace!

```
void secret ( ) {


  int i=0;
  do {
    printf("Hello\n");
    i++;
  }while(i < 10);

}
```

```
push    %rbp
mov     %rsp,%rbp
sub     $0x10,%rsp
movl    $0x0,-0x4(%rbp)
mov     $0x400614,%edi
callq   400410 <puts@plt>
addl    $0x1,-0x4(%rbp)
cmpl    $0x9,-0x4(%rbp)
jle     40053b <secret+0xf>
leaveq
retq
```

Line: 40053b

Some more tracing?

```
push    %rbp
mov     %rsp,%rbp
sub     $0x10,%rsp
movl    $0x0,-0x4(%rbp)
jmp     400570 <supersecret+0x1f>
mov     $0x400634,%edi      Line: 400562
callq   400410 <puts@plt>
addl    $0x1,-0x4(%rbp)
cmpl    $0x9,-0x4(%rbp)     Line: 400570
jle     400562 <supersecret+0x11>
leaveq
retq
```

Some more tracing?

```
void supersecret() {


 int i;
 for(i=0; i<10; i++) {
    printf("Hello\n");
 }




}
```

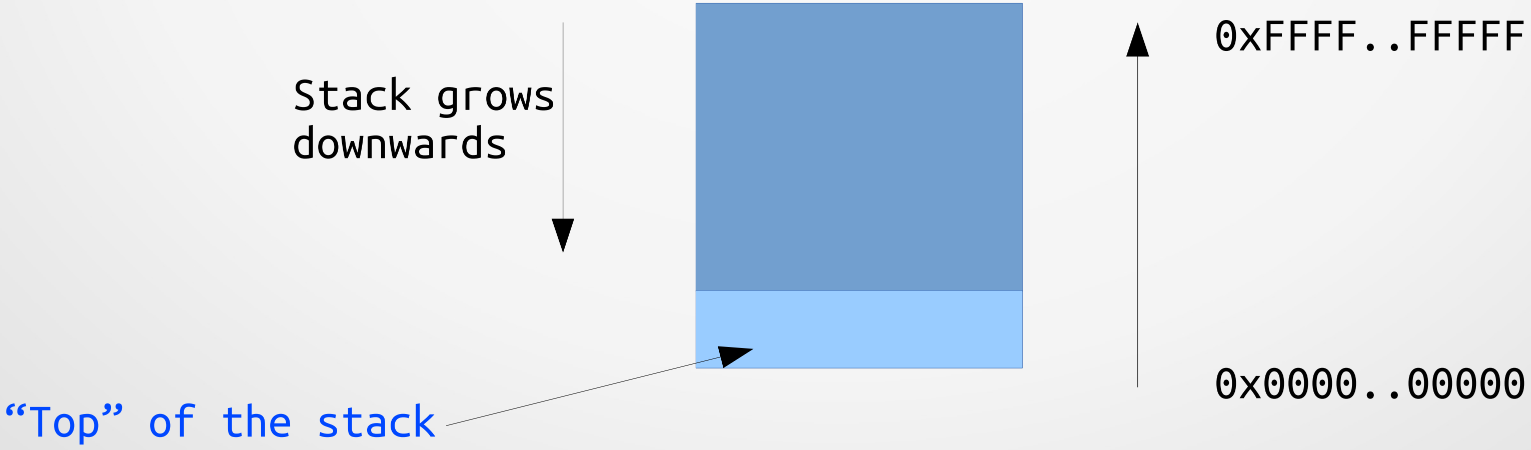
```
push    %rbp
mov     %rsp,%rbp
sub     $0x10,%rsp
movl    $0x0,-0x4(%rbp)
jmp     400570 <supersecret+0x1f>
mov     $0x400634,%edi    Line: 400562
callq   400410 <puts@plt>
addl    $0x1,-0x4(%rbp)
cmpl    $0x9,-0x4(%rbp)   Line: 400570
jle     400562 <supersecret+0x11>
leaveq
retq
```

Procedures!

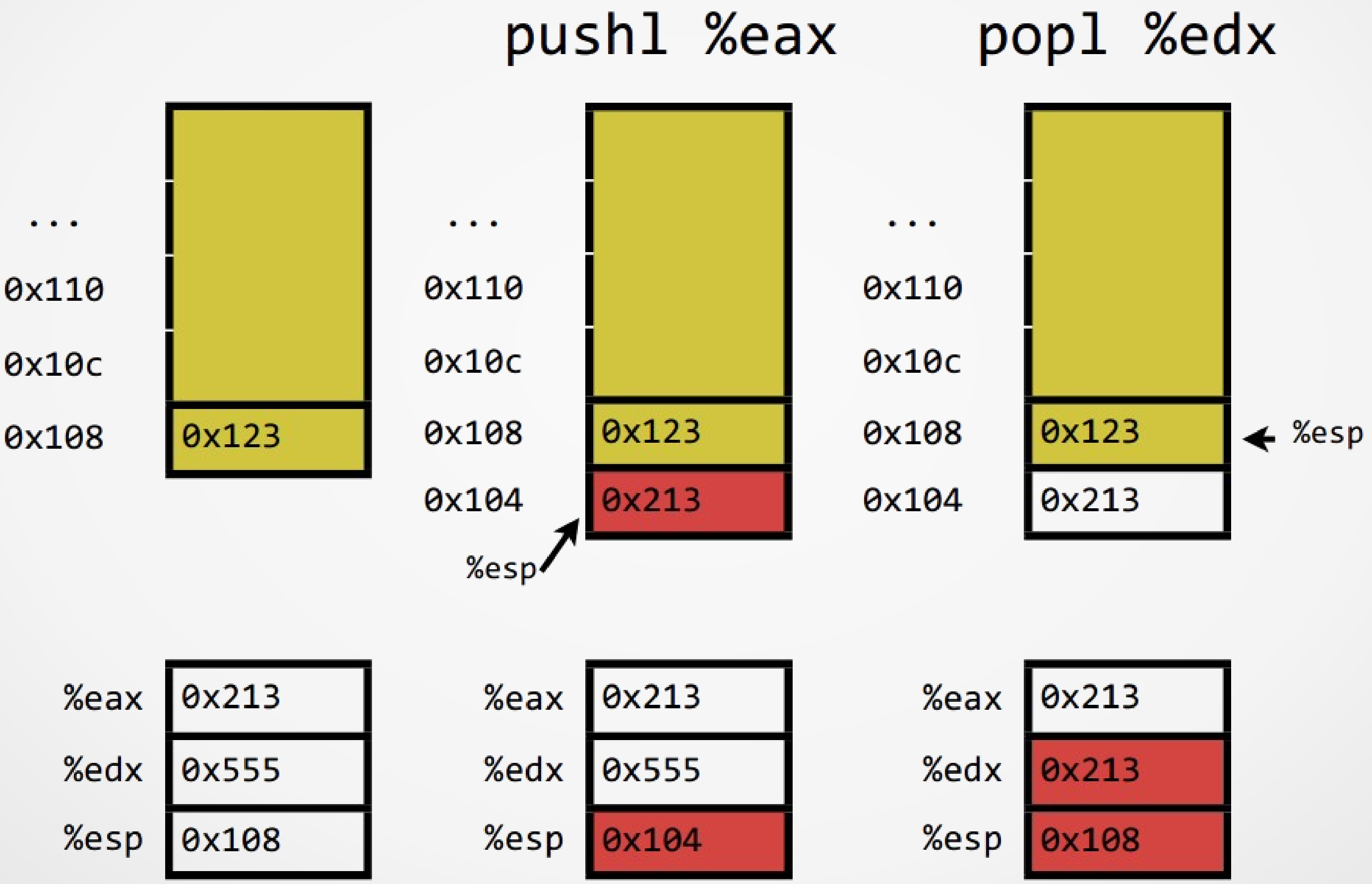
- `call` → You might have seen this a lot
- Remember the lines we always ignored at the beginning and end of functions?
- Lets look at the stack first!

Stack

- Vital role in handling procedure calls
- Somewhat like the “stack” datastructure
  - First In Last Out
  - But we will mend this definition a lot



Pushing and Popping – Simple Example

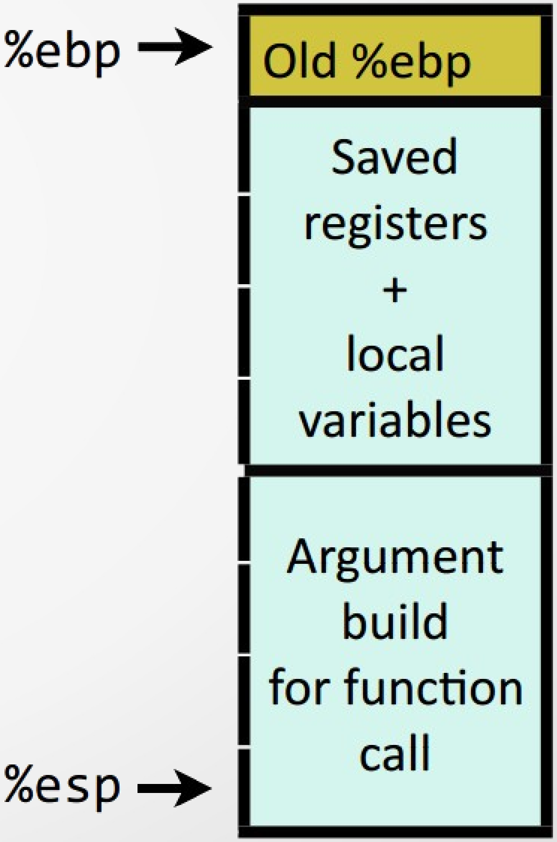


		pushl %eax	popl %edx
...			
0x110			
0x10c			
0x108	0x123	0x123	0x123 ← %esp
0x104		%esp → 0x213	0x213

%eax	0x213	0x213	0x213
%edx	0x555	0x555	0x213
%esp	0x108	0x104	0x108

Frames

- Every function call is given a stack frame
- What does a C function need?
  - Local Variables
  - Space to save callee saved registers
  - Space to put computations
  - A way to give arguments and call other functions

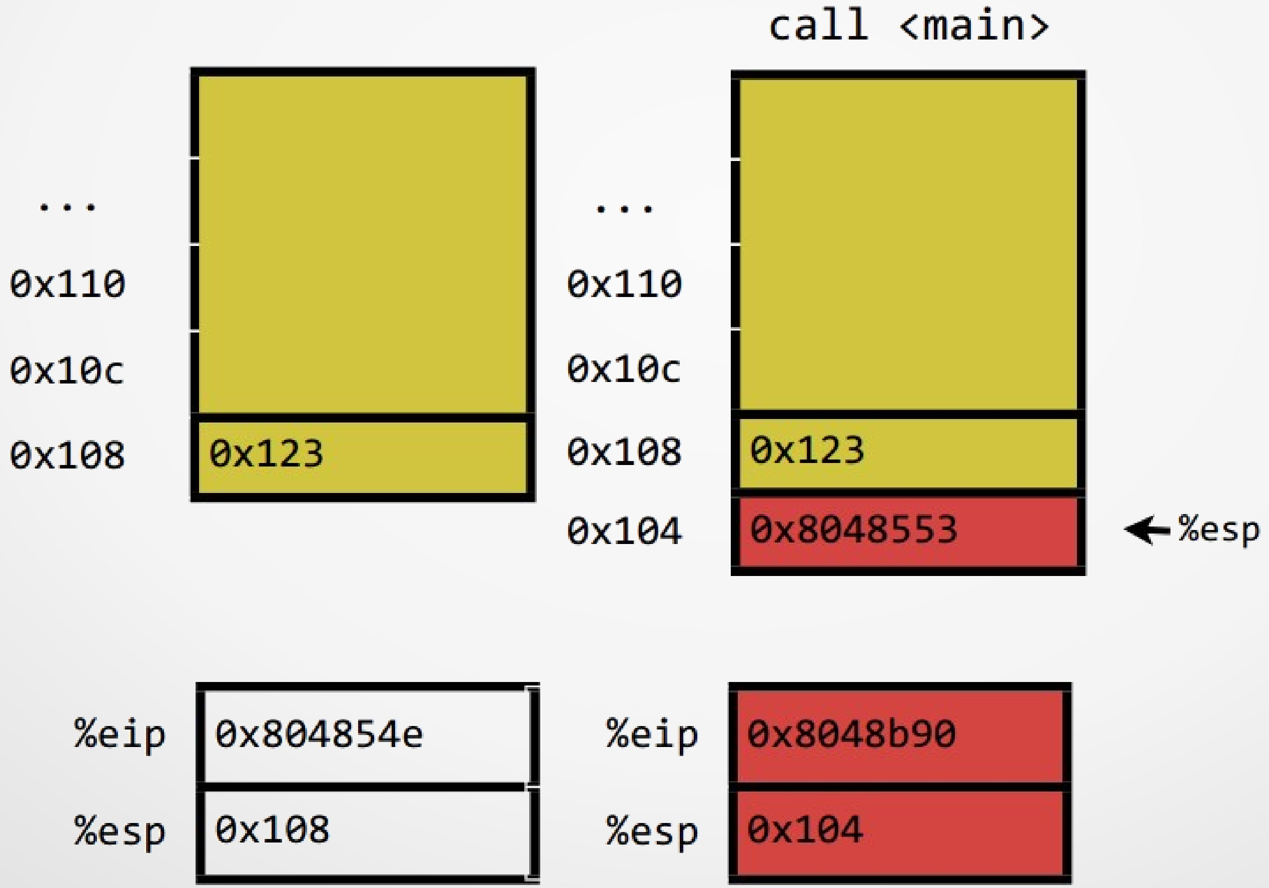

%ebp
Old %ebp
Saved registers + local variables
Argument build for function call
%esp

Function calls

- `call label` → Push "return address" on stack, jump to label
- Return address
  - Address of the instruction immediately after the call
  - Example from disassembly:
    - `804854e: e8 3d 06 00 00 call  8048b90  <main>`
    - `8048553: 50             pushl %eax`
  - Return address is `0x8048553`
- Returning from function call
  - `ret` → Pop return address [(%esp)] into %eip, keep running
  - Remember that the function's actual return value must be in %eax

A more visual explanation - Calling

- 804854e: e8 3d 06 00 00 call 8048b90 <main>
- 8048553: 50 pushl %eax



call <main>

...
0x110
0x10c
0x108 0x123

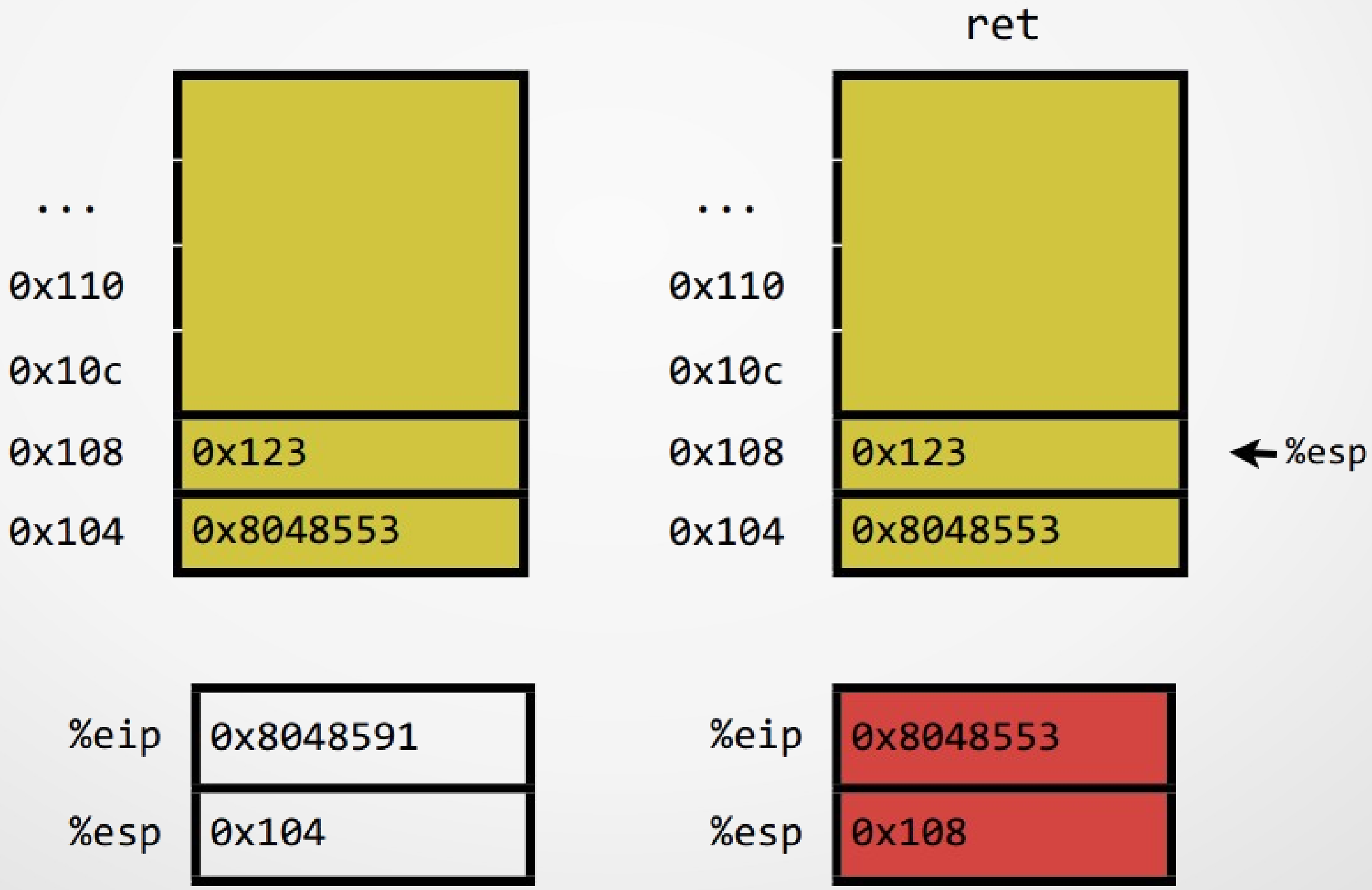
%eip 0x804854e
%esp 0x108

...
0x110
0x10c
0x108 0x123
0x104 0x8048553 ← %esp

%eip 0x8048b90
%esp 0x104

A more visual explanation - Returning

- `8048591: c3                ret`

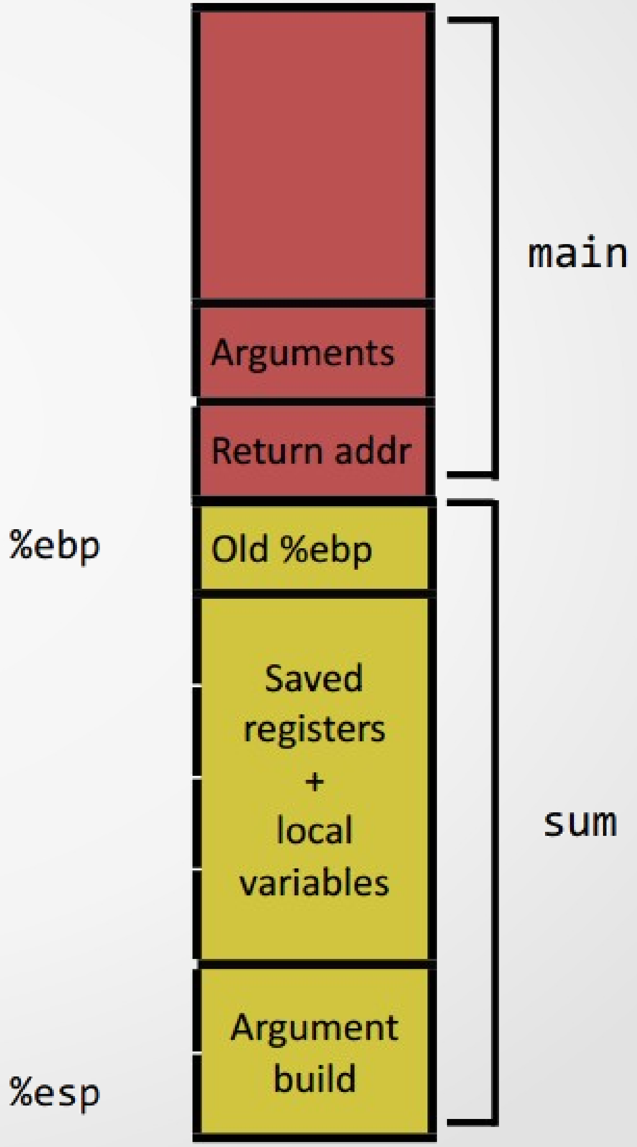


Stack frames

- Suppose you have

```
int main(void)
{
   int x = 3;
   return sum(x, 0);
}
```

- Sum grabs arguments by reaching up the callers stack frame


Arguments
Return addr
main
%ebp
Old %ebp
Saved registers + local variables
Argument build
sum
%esp