Recitation 3: Procedures and the Stack

Andrew Faulring

15213 Section A

23 September 2002

Andrew Faulring

- faulring@cs.cmu.edu
- Office hours:
  - NSH 2504 (lab) / 2507 (conference room)
  - Thursday 5–6

- Lab 2 due Thursday, 11:59pm
- Look for Lab 3 soon—possible by Tuesday

Today's Plan

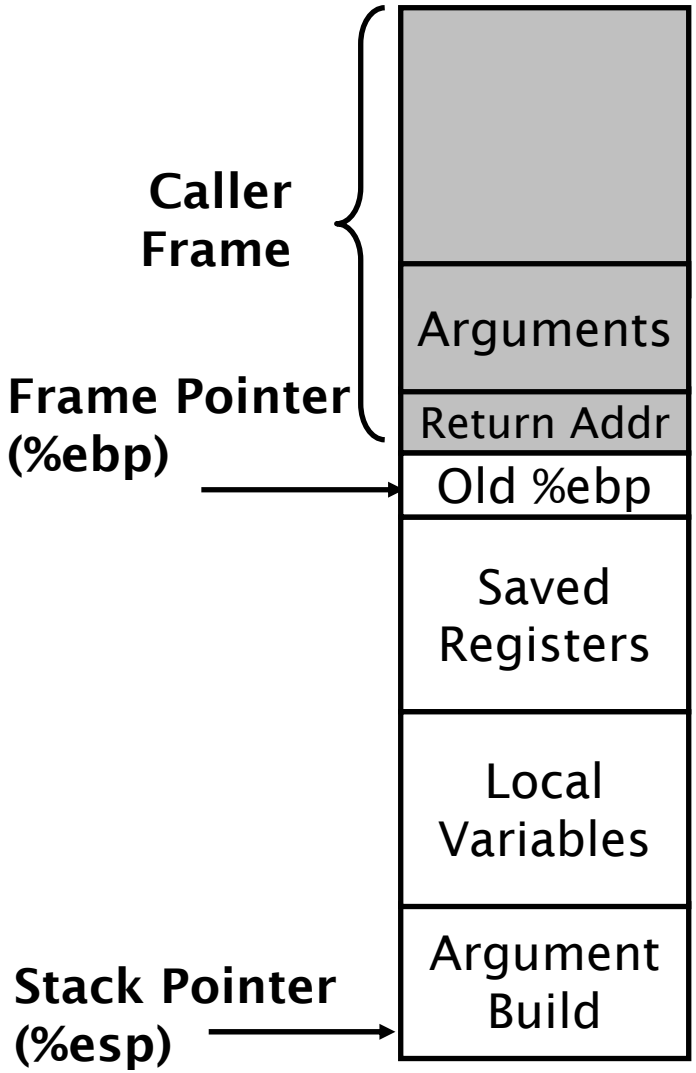
- Is everyone okay with GDB and Lab2?
- Procedures and the Stack
- Homogenous Data: Arrays

Stacks

- Grows down in memory
- Stores local variables that cannot fit in registers
- Stores arguments and return addresses
  - **`%esp`**: points to the top value on the stack
  - **`%ebp`**: points to a function's stack frame
  - **`pushl`**: decrements, then places value
  - **`popl`**: 'returns' value, then increments

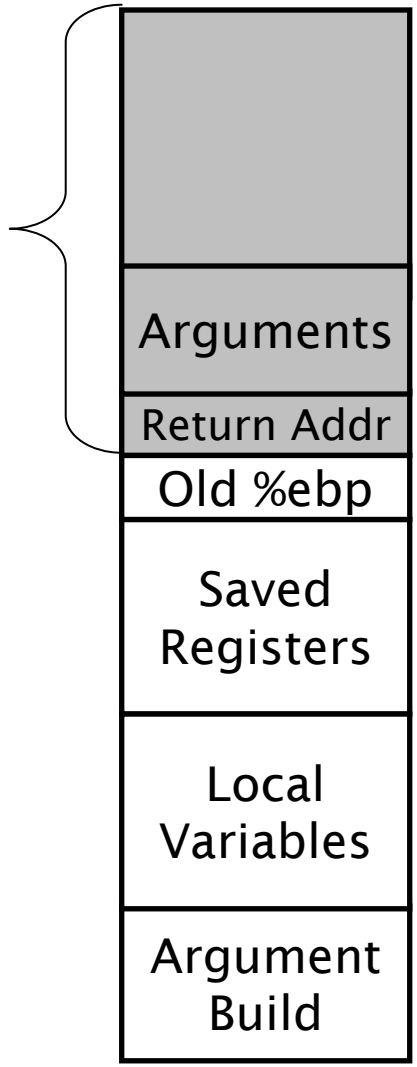
Stack Frames

- Abstract partitioning of the stack
- Each Stack Frame contains the state for a single function instance
  - Recursive functions have multiple Stack Frames—one for each invocation



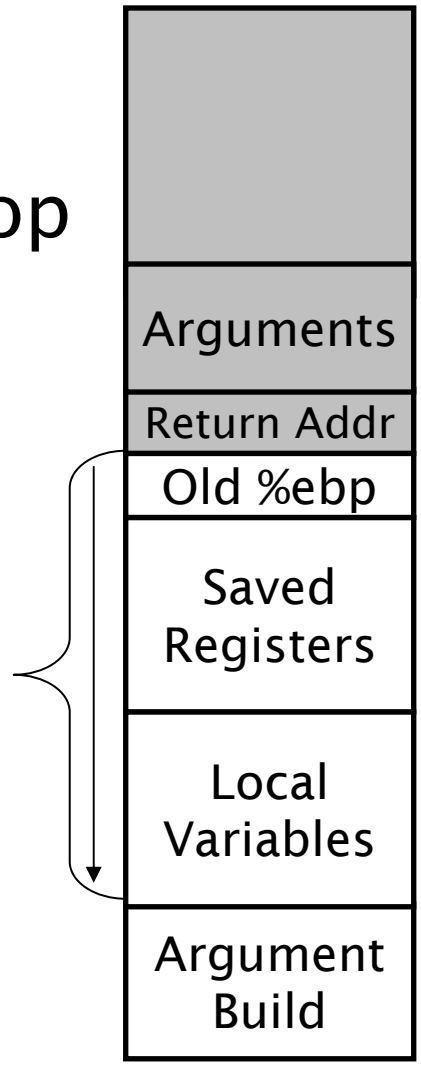
Procedures: Caller Responsibilities

- Save *caller save* registers (if necessary)
  - **`%eax`**, **`%ecx`**, **`%edx`**
- Arguments (**`pushl`**)
  - Pushed onto stack
  - In what order?
- Execute the **`call`** instruction
  - Pushes return address (that of the next instruction) onto the stack



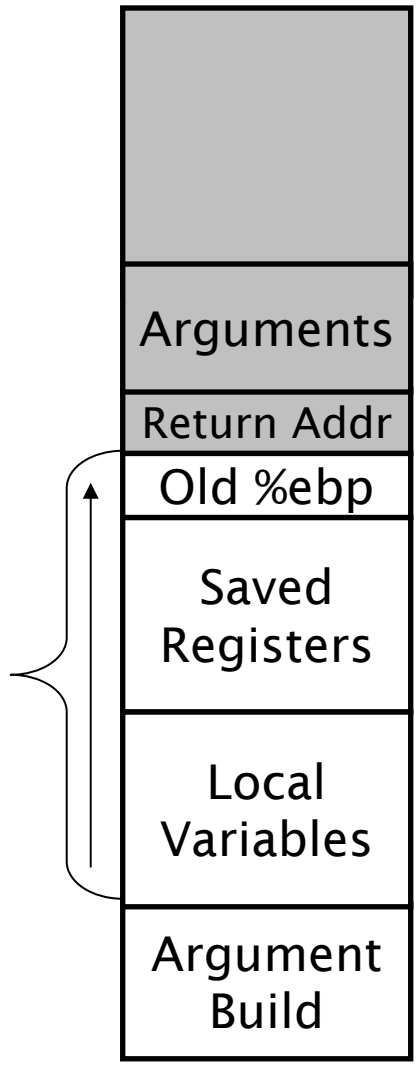
Procedures: Callee Responsibilities (prolog)

- Save base pointer on stack
  - **`push %ebp`**
- Set base pointer to point to the top of the stack
  - **`mov  %esp,%ebp`**
- Save *callee save* registers
  - **`%ebx`**, **`%esi`**, **`%edi`**
  - **`push %esi`**
- Allocate space for local variables
  - Decrement the stack pointer
  - **`add 0xFFFFFFF8, %esp`**



Procedures: Callee Responsibilities (epilog)

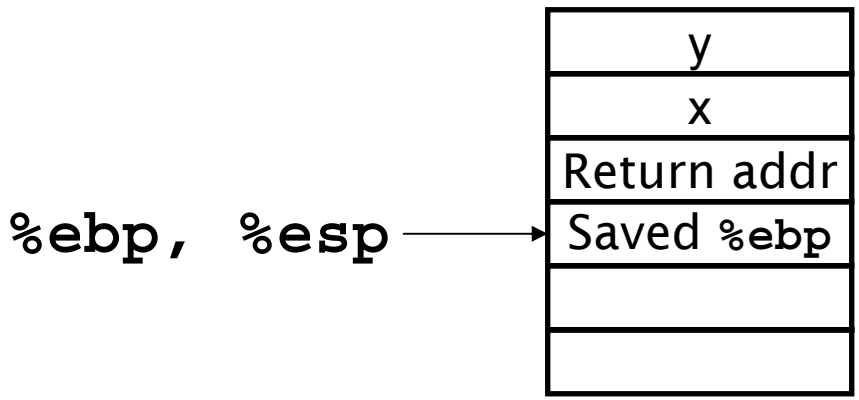
- Move return value into **`%eax`**
- Restore stack pointer
  - **`mov %ebp,%esp`**
- Restore the base pointer to value for calling function
  - **`pop %ebp`**
- Execute the **`ret`** instruction
  - Pops a value from the stack
  - Jumps to the address



Example 1: add

```
int add (int x, int y)
{
   return x+y;
}
```

```
push   %ebp
mov    %esp,%ebp
mov    0xc(%ebp),%eax
add    0x8(%ebp),%eax
mov    %ebp,%esp
pop    %ebp
ret
```

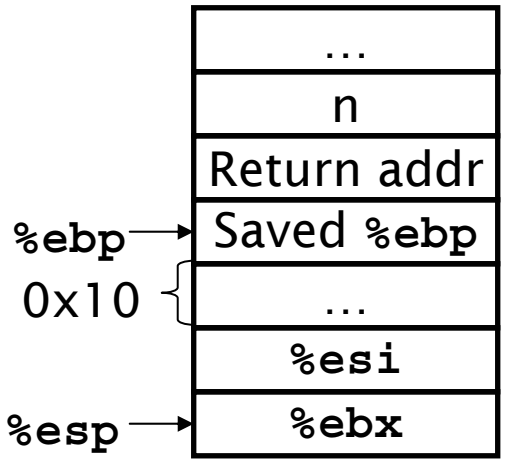


Example 2: fib

```
int fib(int n)
{
  int result;

  if(n <= 2)
    result = 1;
  else
    result = fib(n-2) + fib(n-1);

  return result;
}
```

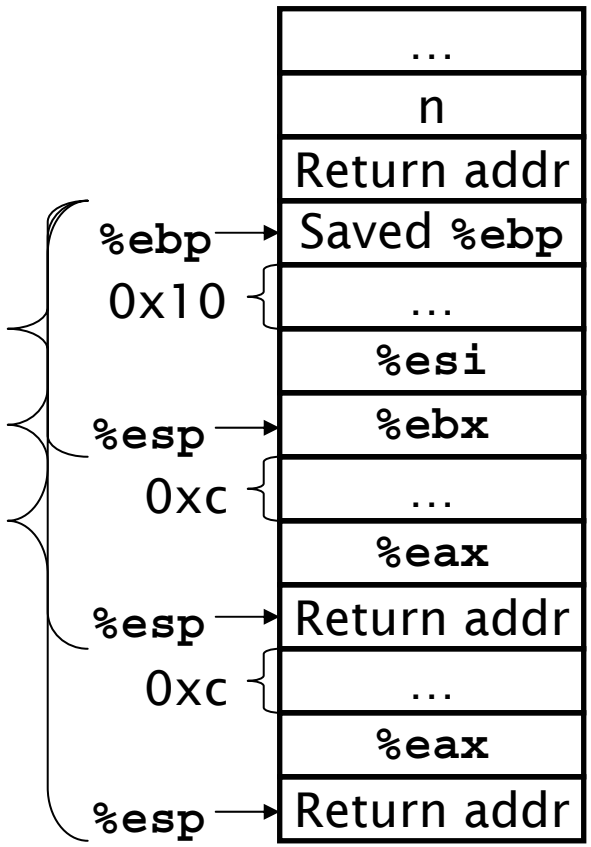
fib: prolog

```
0x8048420 <fib>:
  push   %ebp
  mov    %esp,%ebp
  sub    $0x10,%esp
  push   %esi
  push   %ebx

# first part of body
  mov    0x8(%ebp),%ebx
  cmp    $0x2,%ebx
  jg     0x8048437 <fib+23>
  mov    $0x1,%eax
  jmp    0x8048453 <fib+51>
```



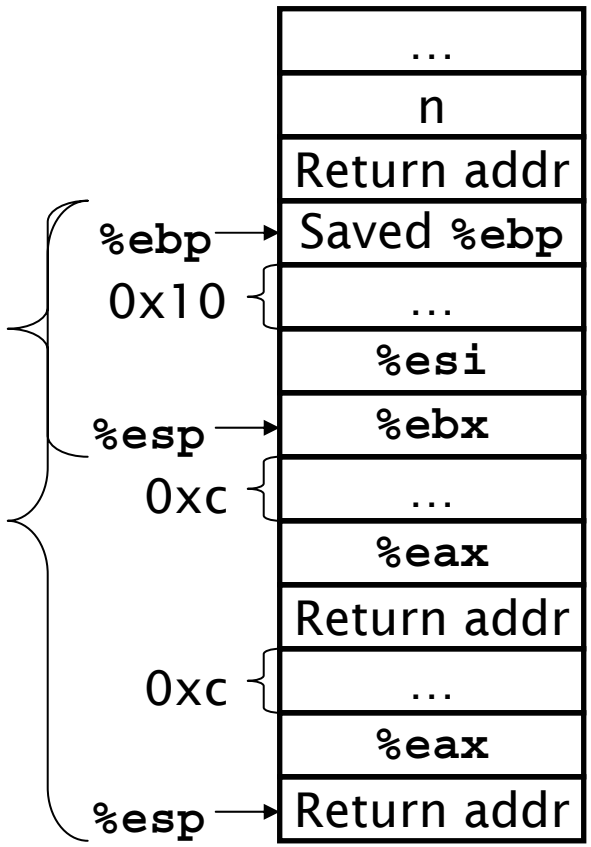
fib: body

```
0x8048437 <fib+23>
  add    $0xfffffff4,%esp
  lea    0xfffffffe(%ebx),%eax
  push   %eax
  call   0x8048420 <fib>
  mov    %eax,%esi
  add    $0xfffffff4,%esp
  lea    0xffffffff(%ebx),%eax
  push   %eax
  call   0x8048420 <fib>
  add    %esi,%eax
```



fib: epilog

```
0x8048453 <fib+51>:
  lea    0xfffffff8(%ebp),%esp
  pop    %ebx
  pop    %esi
  mov    %ebp,%esp
  pop    %ebp
  ret
```



Homogenous Data: Arrays

- Allocated as contiguous blocks of memory
- `int array[5] = {…}`
- `array` begins at memory address 40
  - `array[0]` 40 + 4*0 = 40
  - `array[3]` 40 + 4*3 = 52
  - `array[-1]` 40 + 4*-1 = 36
  - `array[15]` 40 + 4*15 = 100

Example 3: sum_array

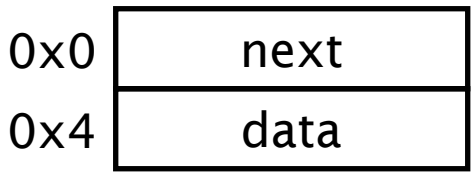
```
int sum_array(int x[], int num)
{
   int i, sum;
   sum = 0;
   for(i = 0; i < num; i++)
      sum += x[i];

   return sum;
}
```

sum_array

```
0x80483f0 <sum_array>:      push  %ebp
0x80483f1 <sum_array+1>:    mov   %esp,%ebp
0x80483f3 <sum_array+3>:    push  %ebx
0x80483f4 <sum_array+4>:    mov   0x8(%ebp),%ebx      # x
0x80483f7 <sum_array+7>:    mov   0xc(%ebp),%ecx      # num
0x80483fa <sum_array+10>:   xor   %eax,%eax           # sum
0x80483fc <sum_array+12>:   xor   %edx,%edx           # i
0x80483fe <sum_array+14>:   cmp   %ecx,%eax           # 0 >= num
0x8048400 <sum_array+16>:   jge   0x804840a <sum_array+26>
0x8048402 <sum_array+18>:   add   (%ebx,%edx,4),%eax # sum += x[i]
0x8048405 <sum_array+21>:   inc   %edx                # i++
0x8048406 <sum_array+22>:   cmp   %ecx,%edx           # i < num
0x8048408 <sum_array+24>:   jl    0x8048402 <sum_array+18>
0x804840a <sum_array+26>:   pop   %ebx
0x804840b <sum_array+27>:   mov   %ebp,%esp
0x804840d <sum_array+29>:   pop   %ebp
0x804840e <sum_array+30>:   ret
```

Example 4: linked list

```
typedef struct linked_list
{
   struct linked_list *next;
   int                 data;

} linked_list;
```



Example 4: sum_linked_list

```
int sum_linked_list(linked_list *head)
{
  int sum;

  sum = 0;
  while(head != NULL)
  {
    sum += head->data;
    head = head->next;
  }

  return sum;
}
```

sum_linked_list

```
0x8048434 <sum>:      push   %ebp
0x8048435 <sum+1>:    mov    %esp,%ebp
0x8048437 <sum+3>:    mov    0x8(%ebp),%edx      # head
0x804843a <sum+6>:    xor    %eax,%eax           # sum = 0
0x804843c <sum+8>:    test   %edx,%edx           # head == NULL
0x804843e <sum+10>:   je     0x8048449 <sum+21>
0x8048440 <sum+12>:   add    0x4(%edx),%eax      # sum += head->data
0x8048443 <sum+15>:   mov    (%edx),%edx         # head = head->next
0x8048445 <sum+17>:   test   %edx,%edx           # head == NULL
0x8048447 <sum+19>:   jne    0x8048440 <sum+12>
0x8048449 <sum+21>:   mov    %ebp,%esp
0x804844b <sum+23>:   pop    %ebp
0x804844c <sum+24>:   ret
```

- Good luck with Lab2!