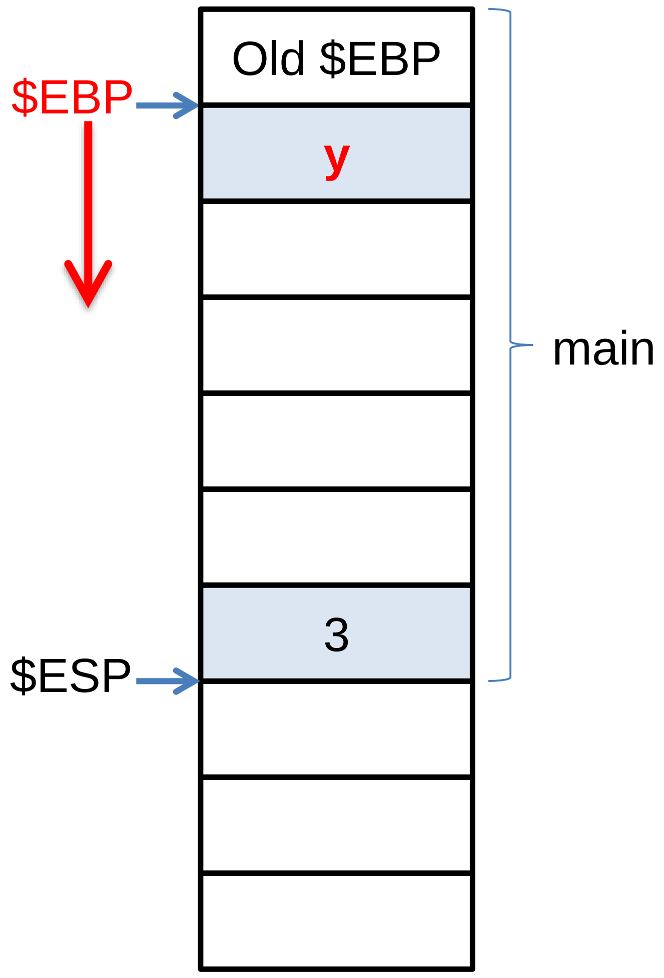
Stack Activation Records

CS449 Fall 2018

Local Variables: Negative Offset from $EBP

```
#include <stdio.h>

int f(int x)
{
  return x;
}

int main()
{
  int y;

  y = f(3);

  return 0;
}
```

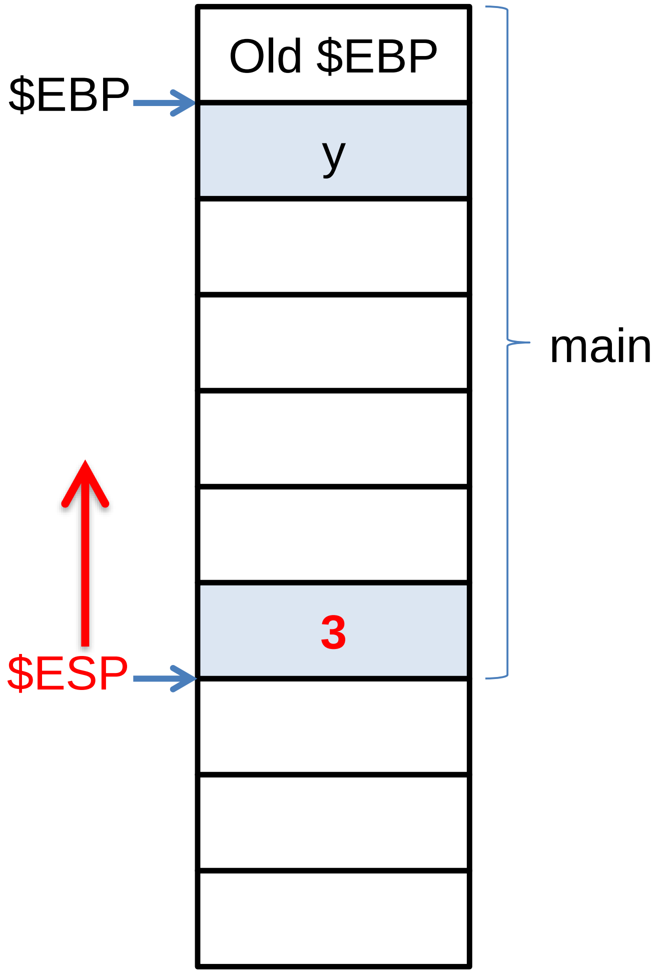
```
f:
   pushl    %ebp
   movl     %esp, %ebp
   movl     8(%ebp), %eax
   leave
   ret

main:
   pushl    %ebp
   movl     %esp, %ebp
   subl     $8, %esp
   andl     $-16, %esp
   subl     $16, %esp
   movl     $3, (%esp)
   call     f
   movl     %eax, -4(%ebp)
   movl     $0, %eax
   leave
   ret
```



Arguments: (Non-negative) Offset from $ESP

```c
#include <stdio.h>

int f(int x)
{
  return x;
}

int main()
{
  int y;

  y = f(3);

  return 0;
}
```

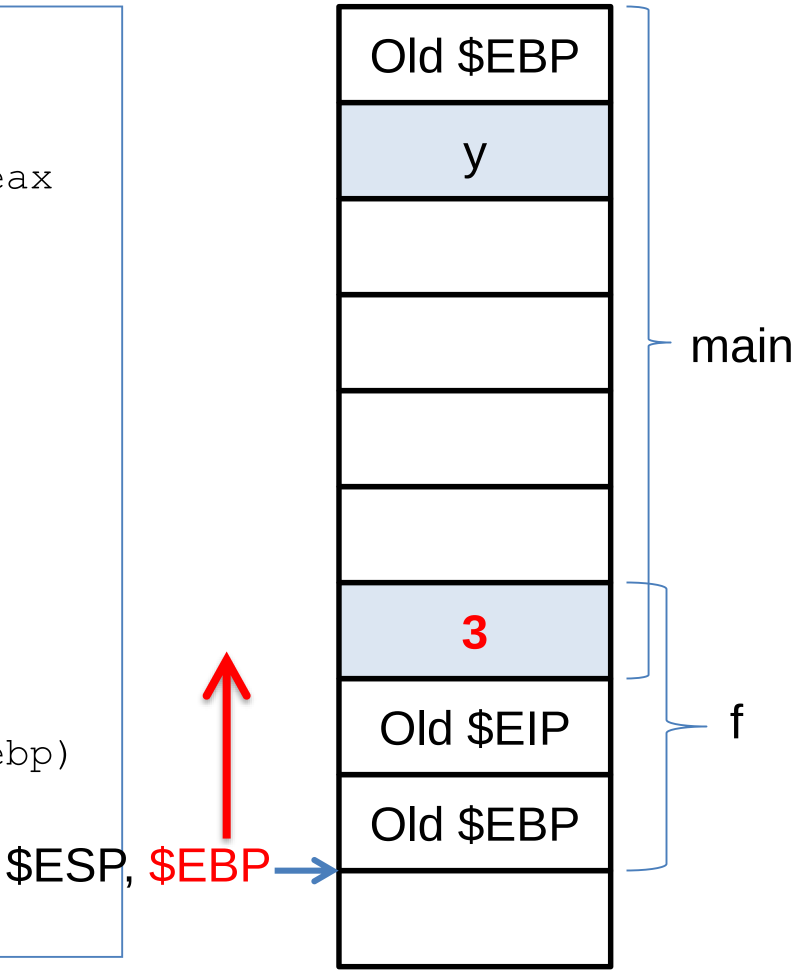
```
f:
   pushl    %ebp
   movl     %esp, %ebp
   movl     8(%ebp), %eax
   leave
   ret

main:
   pushl    %ebp
   movl     %esp, %ebp
   subl     $8, %esp
   andl     $-16, %esp
   subl     $16, %esp
   movl     $3, (%esp)
   call     f
   movl     %eax, -4(%ebp)
   movl     $0, %eax
   leave
   ret
```



Parameters: Positive Offset from $EBP

```c
#include <stdio.h>

int f(int x)
{
  return x;
}

int main()
{
  int y;

  y = f(3);

  return 0;
}
```

```
f:
  pushl   %ebp
  movl    %esp, %ebp
  movl    8(%ebp), %eax
  leave
  ret

main:
  pushl   %ebp
  movl    %esp, %ebp
  subl    $8, %esp
  andl    $-16, %esp
  subl    $16, %esp
  movl    $3, (%esp)
  call    f
  movl    %eax, -4(%ebp)
  movl    $0, %eax
  leave
  ret
```



Objdump

- Can dump various sections of an object file
  - The –D option disassembles entire file
  - Detailed usage in project 2 FAQ
- Do the following to disassemble your puzzle:
  - `objdump -D recitation > rec.dump`
- Then open with your favorite editor:
  - `nano rec.dump`
- Next, we will interpret snippets of assembly code from above file, using above knowledge

Example 1: strlen() call

```
804826c: lea    -0x88(%ebp),%eax
8048272: mov    %eax,(%esp)
8048275: call   804fa08 <strlen>
```

- Line 1: load address -0x88(%ebp) to %eax
  - -0x88(%ebp) is a negative offset from %ebp
  - What comes after below %ebp? Local variables.
  - So this stores the address of a local variable to %eax
- Line 2: move address in %eax to (%esp)
  - (%esp) is a non-negative offset from %esp
  - What gets pushed on the top of the stack? Arguments.
  - So this stores the local variable address as an argument
- Line 3: call `int strlen(const char*)` with above argument
  - What then would that address be that we passed?
  - Probably the starting address of a char array local variable.

Example 2: while loop

```
804826c: lea    -0x88(%ebp),%eax
8048272: %eax,(%esp)
8048275: call   804fa08 <strlen>
804827a: cmp    %eax,-0xc(%ebp)
804827d: jae    8048291 <main+0x7f>
804827f: lea    -0x88(%ebp),%eax
8048285: add    -0xc(%ebp),%eax
8048288: incb   (%eax)
804828a: lea    -0xc(%ebp),%eax
804828d: incl   (%eax)
804828f: jmp    804826c <main+0x5a>
```

- Notice the jump instruction marked in red
  - Note that it jumps backwards to the address marked in red
  - Can only mean that the code in between is a loop (for, while, do/while)
- Inside the loop is the `strlen()` call we saw previously