CS 240
Laboratory 8 Notes
Disassembly and Reverse Engineering

- Positive displacements from %ebp are parameters

- Negative Displacements from $esp are local variables

- IDIV val

  o takes a 64-bit value (high 32 bits in %edx, low 32 bits in %eax), and divides by val
  o puts result in %eax
  o puts remainder in %edx

Assume the function has been invoked with values of 2 and 4.

Frame set-up: push %ebp on the stack (so it can be restored later), and decrements %esp to create the stack frame for the function

```
0x08048414 <+0>:  push   %ebp
```

Starting address of **analyze**

```
0x08048415 <+1>:  mov    %esp,%ebp
0x08048417 <+3>:  sub    $0x28,%esp
```

Initialize local variable to 1

```
0x0804841a <+6>:  movl   $0x1,-0x10(%ebp)
```

LOCAL1 = 1 must be a local var/ uses a negative offset from the %ebp

Jump to the while part of the do-while loop to test the condition before you execute the body of the loop

```
0x08048421 <+13>:  jmp    0x804844f <analyze+59>
```

If PARAM1 and PARAM2 are both divisible by LOCAL1, LOCAL2 = LOCAL1; otherwise, increment LOCAL1 and repeat the loop

LOOP:

```
0x08048423 <+15>:   mov    0x8(%ebp),%eax          params always positive displacement from the %ebp
                                                   PARAM1  (assume = 2)
```

Sign-extend PARAM1 to 64 bits, integer divide by LOCAL1, put remainder in %eax

```
0x08048426 <+18>:mov    %eax,%edx
0x08048428 <+20>:sar    $0x1f,%edx
0x0804842b <+23>:idivl  -0x10(%ebp)
0x0804842e <+26>:mov    %edx,%eax
```

If remainder is not 0 (if PARAM1 is not divisible by LOCAL1), skip the rest of the body of the loop and jump to INCREMENT-LOCAL1

```
0x08048430 <+28>:test   %eax,%eax
0x08048432 <+30>:jne    0x804844b <analyze+55>
```

Repeat for param 2 (sign-extend PARAM2 to 64 bits, integer divide by LOCAL1, if remainder is not 0 (if PARAM1 is not divisible by LOCAL1), skip the rest of the body of the loop, go to INCR-LOCAL1

```
0x08048434 <+32>:mov    0xc(%ebp),%eax
0x08048437 <+35>:mov    %eax,%edx
0x08048439 <+37>:sar    $0x1f,%edx
0x0804843c <+40>:idivl  -0x10(%ebp)
0x0804843f <+43>: mov    %edx,%eax
0x08048441 <+45>:test   %eax,%eax
0x08048443 <+47>:jne    0x804844b <analyze+55>
0x08048445 <+49>:mov    -0x10(%ebp),%eax
0x08048448 <+52>:mov    %eax,-0xc(%ebp)
```

add 1 to first local variable (acts as a loop counter)

INCR-LOCAL 1

```
 0x0804844b <+55>: addl   $0x1,-0x10(%ebp)
```

if both PARAM1 and PARM2 > LOCAL1, repeat the LOOP

WHILE of DO-WHILE:

```
0x0804844f <+59>: mov    -0x10(%ebp),%eax
0x08048452 <+62>:cmp    0x8(%ebp),%eax
0x08048455 <+65>:jg     0x804845f <analyze+75>    if  LOCAL1 > PARAM1, jump to DONE
0x08048457 <+67>:mov    -0x10(%ebp),%eax
0x0804845a <+70>:cmp    0xc(%ebp),%eax
0x0804845d <+73>:jle    0x8048423 <analyze+15>   if LOCAL1 <= PARAM2, jump to LOOP
```

Put parameters for printf on stack, and call printf , LOCAL2 (greatest common denominator)

DONE:

```
0x0804845f <+75>: mov    $0x8048594,%eax     address of formatting string for printf
0x08048464 <+80>:mov    -0xc(%ebp),%edx      LOCAL2 (result) to be printed
0x08048467 <+83>:mov    %edx,0x4(%esp)       put parameters on stack
0x0804846b <+87>:mov    %eax,(%esp)
0x0804846e <+90>:call   0x8048338 <printf@plt>
0x08048473 <+95>:leave                       clean up stack
0x08048474 <+96>:ret
```