Assembly Language Programming - III

GDB Debugger

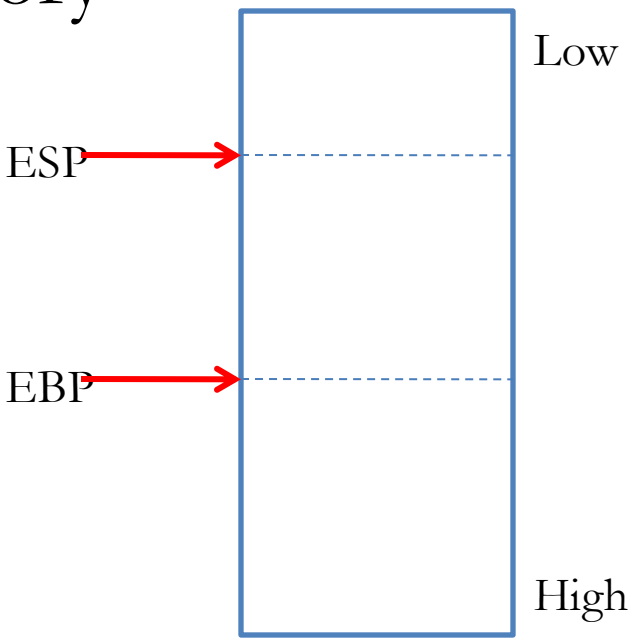
- Please refer to the handout
- New GDB commands (power.s)
  - `info registers` (prints all register values)
  - `print/d $eax` (prints individual register value. Note '$')
  - `print/d iBase` (prints iBase value in decimal)
  - `print/c cPrompt1`(prints 1st byte of mem starting @ cPrompt1)
  - `x/d &iBase` (prints iBase address and dereferenced value)
  - `x/c &cPrompt1`
  - `x/s &cPrompt1`

Local variables

- Some facts
  - Stack consists of stack frames (activation records)
  - Each stack frame holds some local data for a function
- Special registers
  - ESP (extended stack pointer) => Top of the current stack frame
  - EBP (extended base pointer) => Bottom of the current stack frame

Local variables

- Stack grows toward low memory
- Push operation
  - subtracts from ESP
  - does not affect EBP
- Pop operation
  - add to ESP
  - does not affect EBP
- EBP retains its value throughout function



Example: powerfunction.c

- Same basic functionality but:
  - uses local variables
  - calls power() function

```c
#include <stdio.h>

static int power(int iBase, int iExp)
{
   int iPower = 1;
   int iIndex;

   for (iIndex = 1; iIndex <= iExp; iIndex++)
      iPower *= iBase;

   return iPower;
}

int main(int argc, char *argv[])
{
   int iBase;
   int iExp;
   int iPower;

   printf("Enter the base:  ");
   scanf("%d", &iBase);

   printf("Enter the exponent:  ");
   scanf("%d", &iExp);

   iPower = power(iBase, iExp);

   printf("%d to the %d power is %d.\n", iBase, iExp, iPower);

   return 0;
}
```

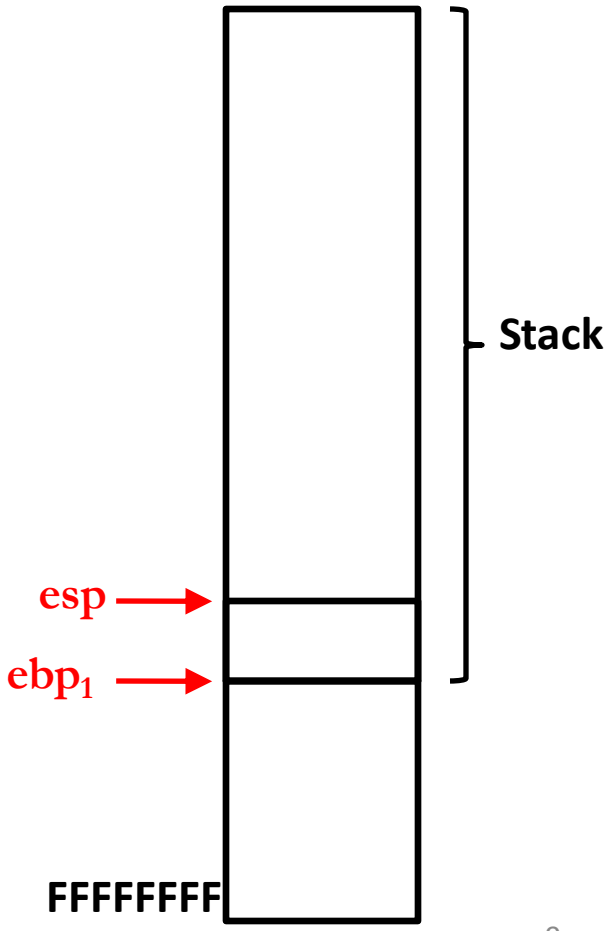
FLATTEN IT

```
### ------------------------------------------------------------------
### powerfunction.s
### Author: Bob Dondero
### Functions
### ------------------------------------------------------------------

        .section ".rodata"

cPrompt1:
        .asciz  "Enter the base:  "

cPrompt2:
        .asciz  "Enter the exponent:  "

cScanfFormat:
        .asciz  "%d"

cResult:
        .asciz  "%d raised to the %d power is %d.\n"
```

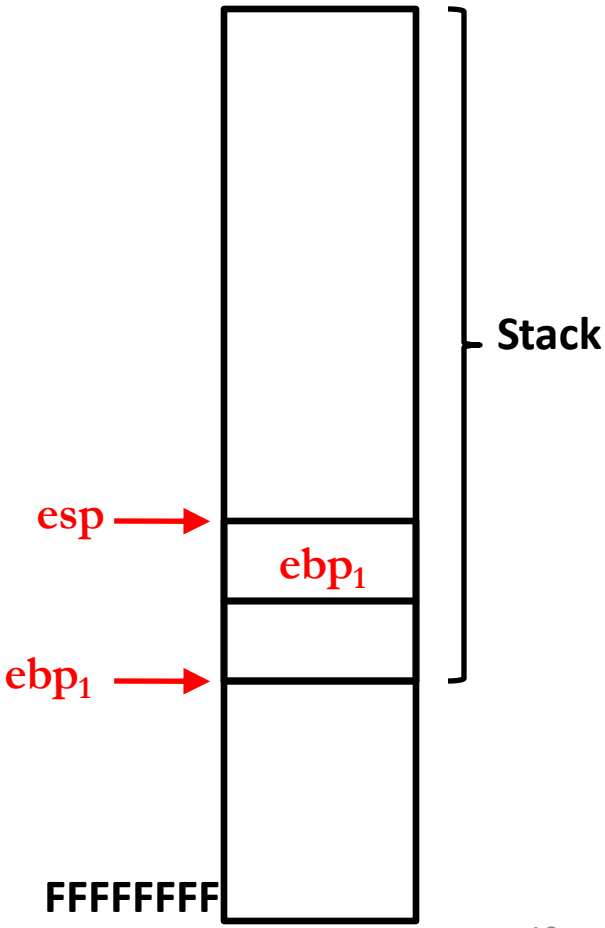
```
### ------------------------------------------------------------------

        .section ".data"

### ------------------------------------------------------------------

        .section ".bss"

### ------------------------------------------------------------------
```

```
    ## ------------------------------------------------------------
    ## int main(int argc, char *argv[])
    ## Read a non-negative base and exponent from stdin.  Write
    ## base raised to the exponent power to stdout.  Return 0.
    ## ------------------------------------------------------------


    ## Local variable offsets:
    .equ IBASE,    -4
    .equ IEXP,     -8
    .equ IPOWER, -12

    .globl  main
    .type   main,@function
main:

        pushl   %ebp
        movl    %esp, %ebp

        ## int iBase
        subl    $4, %esp

        ## int iExp
        subl    $4, %esp

        ## int iPower
        subl    $4, %esp
```

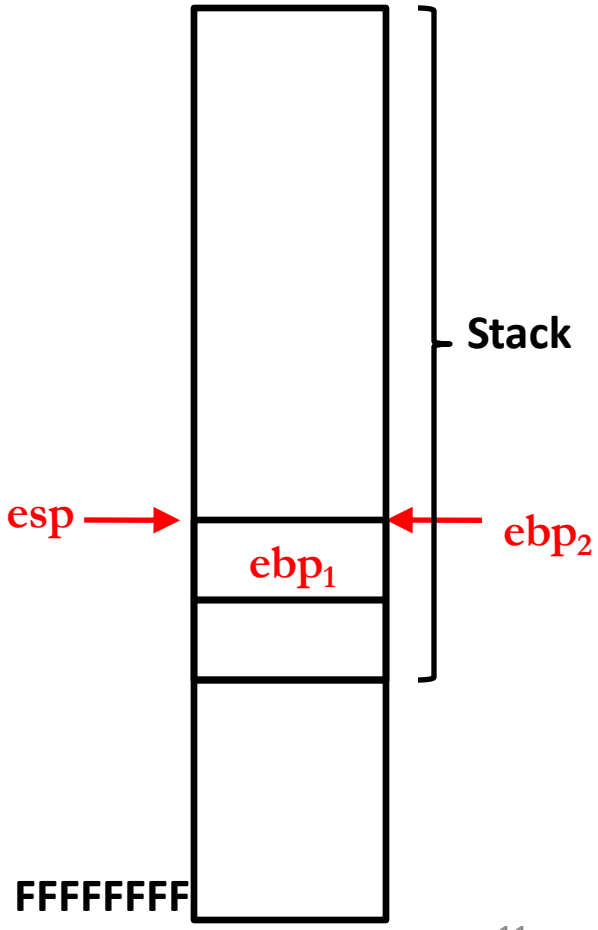
aliases



```
    ## ------------------------------------------------------------
    ## int main(int argc, char *argv[])
    ## Read a non-negative base and exponent from stdin.  Write
    ## base raised to the exponent power to stdout.  Return 0.
    ## ------------------------------------------------------------


    ## Local variable offsets:
    .equ IBASE,   -4
    .equ IEXP,    -8
    .equ IPOWER, -12

    .globl  main
    .type   main,@function
main:

        pushl   %ebp
        movl    %esp, %ebp

        ## int iBase
        subl    $4, %esp
        ## int iExp
        subl    $4, %esp

        ## int iPower
        subl    $4, %esp
```

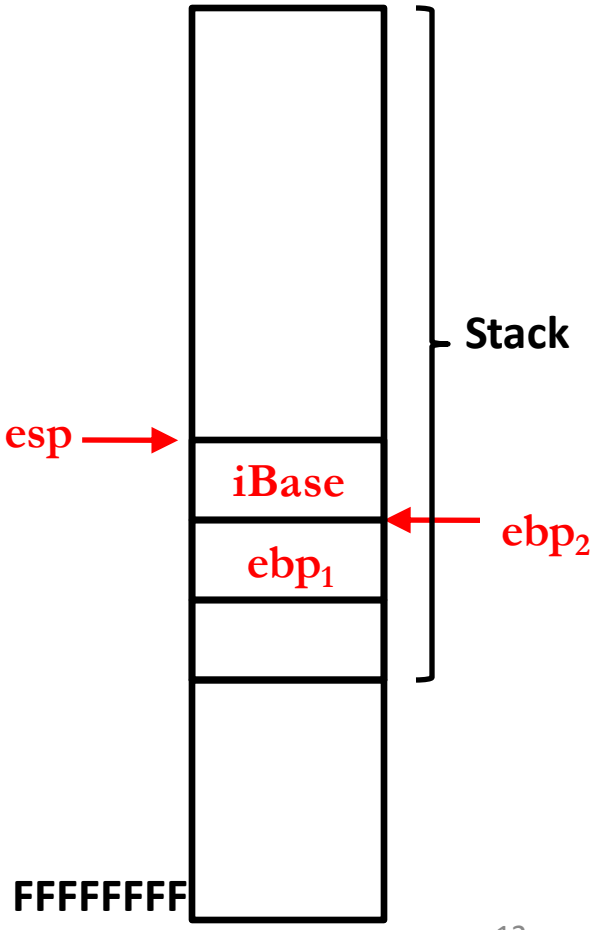


```
    ## ------------------------------------------------------------
    ## int main(int argc, char *argv[])
    ## Read a non-negative base and exponent from stdin.  Write
    ## base raised to the exponent power to stdout.  Return 0.
    ## ------------------------------------------------------------


    ## Local variable offsets:
    .equ IBASE,   -4
    .equ IEXP,    -8
    .equ IPOWER, -12

    .globl  main
    .type   main,@function
main:

        pushl   %ebp
        movl    %esp, %ebp

        ## int iBase
        subl    $4, %esp
        ## int iExp
        subl    $4, %esp

        ## int iPower
        subl    $4, %esp
```

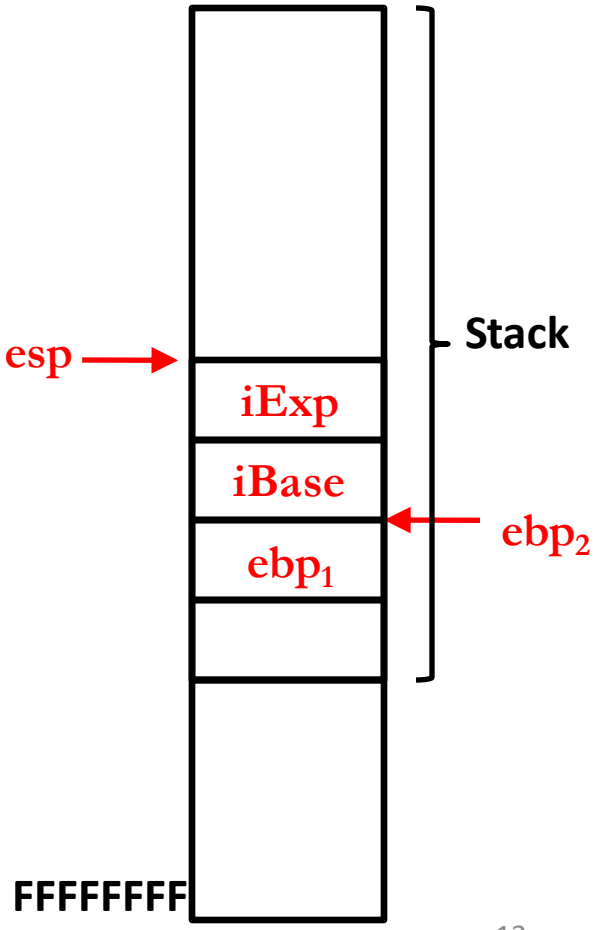


```
    ## ------------------------------------------------------------
    ## int main(int argc, char *argv[])
    ## Read a non-negative base and exponent from stdin.  Write
    ## base raised to the exponent power to stdout.  Return 0.
    ## ------------------------------------------------------------
```

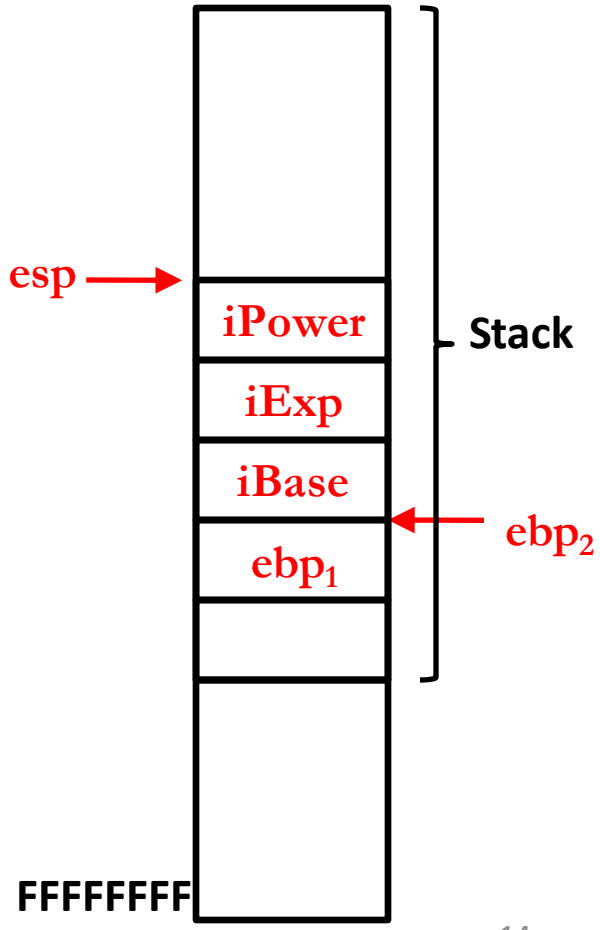
```
    ## Local variable offsets:
    .equ IBASE,   -4
    .equ IEXP,    -8
    .equ IPOWER, -12

    .globl  main
    .type   main,@function
main:

        pushl   %ebp
        movl    %esp, %ebp

        ## int iBase
        subl    $4, %esp
        ## int iExp
        subl    $4, %esp

        ## int iPower
        subl    $4, %esp
```



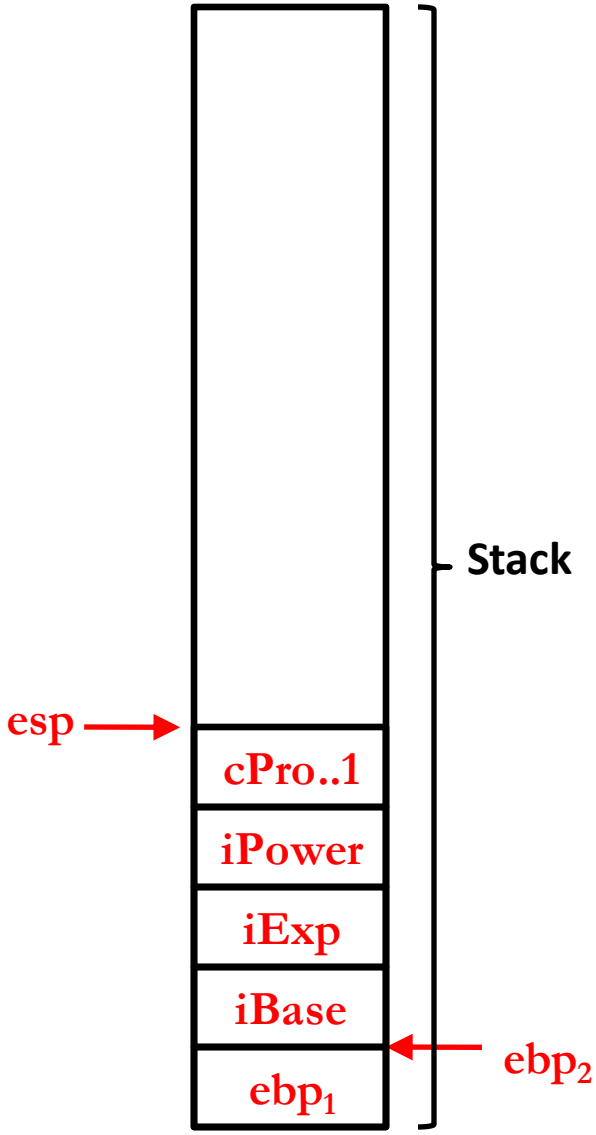
```
    ## ------------------------------------------------------------
    ## int main(int argc, char *argv[])
    ## Read a non-negative base and exponent from stdin.  Write
    ## base raised to the exponent power to stdout.  Return 0.
    ## ------------------------------------------------------------

    ## Local variable offsets:
    .equ IBASE,   -4
    .equ IEXP,    -8
    .equ IPOWER, -12

    .globl  main
    .type   main,@function
main:

        pushl   %ebp
        movl    %esp, %ebp

        ## int iBase
        subl    $4, %esp
        ## int iExp
        subl    $4, %esp

        ## int iPower
        subl    $4, %esp
```



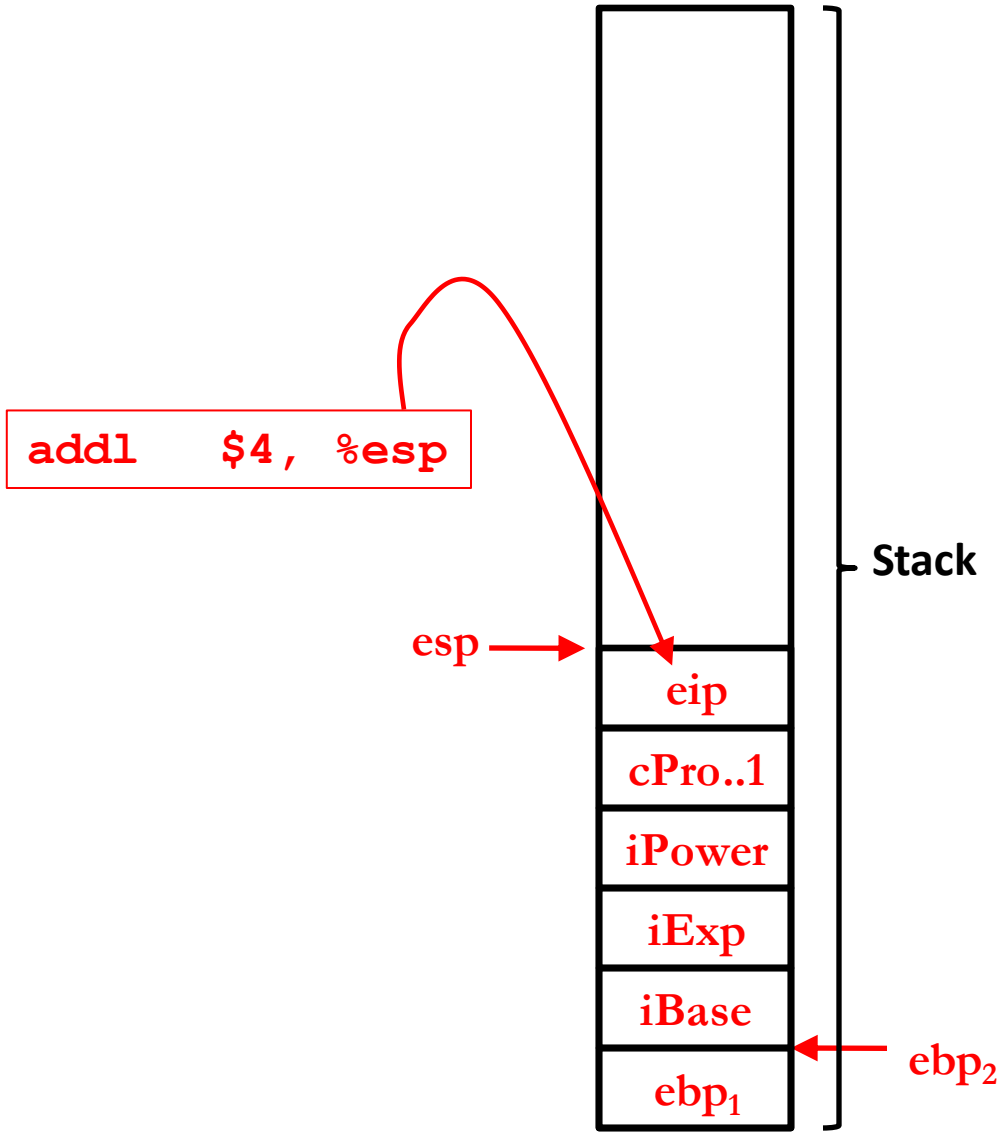
```
    ## ------------------------------------------------------------
    ## int main(int argc, char *argv[])
    ## Read a non-negative base and exponent from stdin.  Write
    ## base raised to the exponent power to stdout.  Return 0.
    ## ------------------------------------------------------------


    ## Local variable offsets:
    .equ IBASE,   -4
    .equ IEXP,    -8
    .equ IPOWER, -12

    .globl  main
    .type   main,@function
main:

        pushl   %ebp
        movl    %esp, %ebp

        ## int iBase
        subl    $4, %esp
        ## int iExp
        subl    $4, %esp

        ## int iPower
        subl    $4, %esp
```



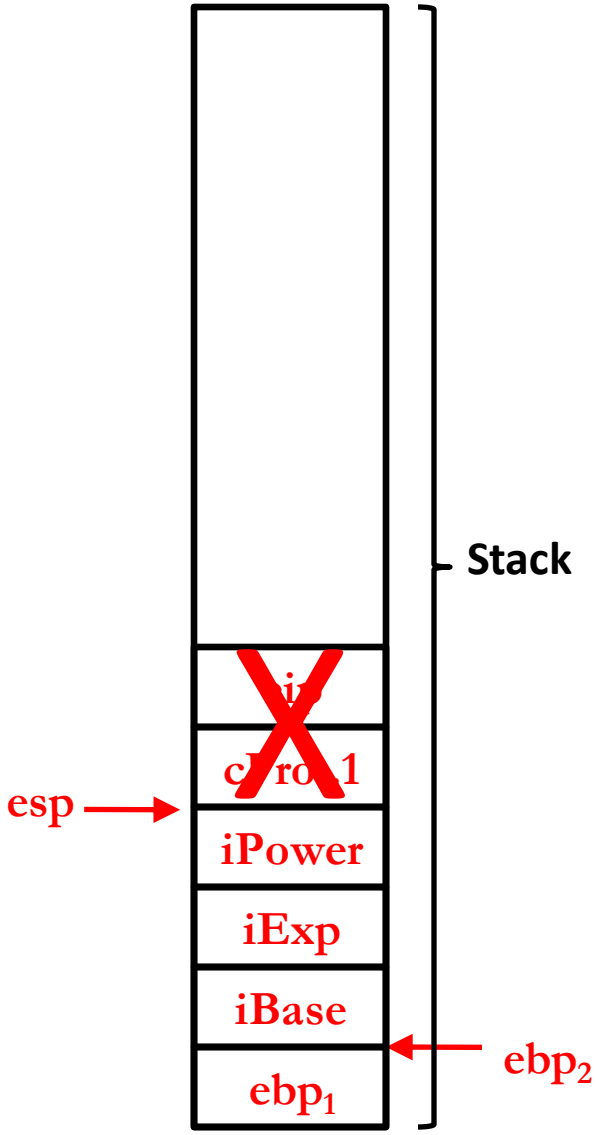
```
## printf("Enter the base:  ")
pushl   $cPrompt1
call    printf
addl    $4, %esp

## scanf("%d", &iBase)
leal    IBASE(%ebp), %eax
## Alternative to leal:
##    movl %ebp,%eax
##    addl $IBASE,%eax
pushl   %eax
pushl   $cScanfFormat
call    scanf
addl    $8, %esp
```



```
## printf("Enter the base:  ")
pushl   $cPrompt1
call    printf
addl    $4, %esp

## scanf("%d", &iBase)
leal    IBASE(%ebp), %eax
## Alternative to leal:
##    movl %ebp,%eax
##    addl $IBASE,%eax
pushl   %eax
pushl   $cScanfFormat
call    scanf
addl    $8, %esp
```

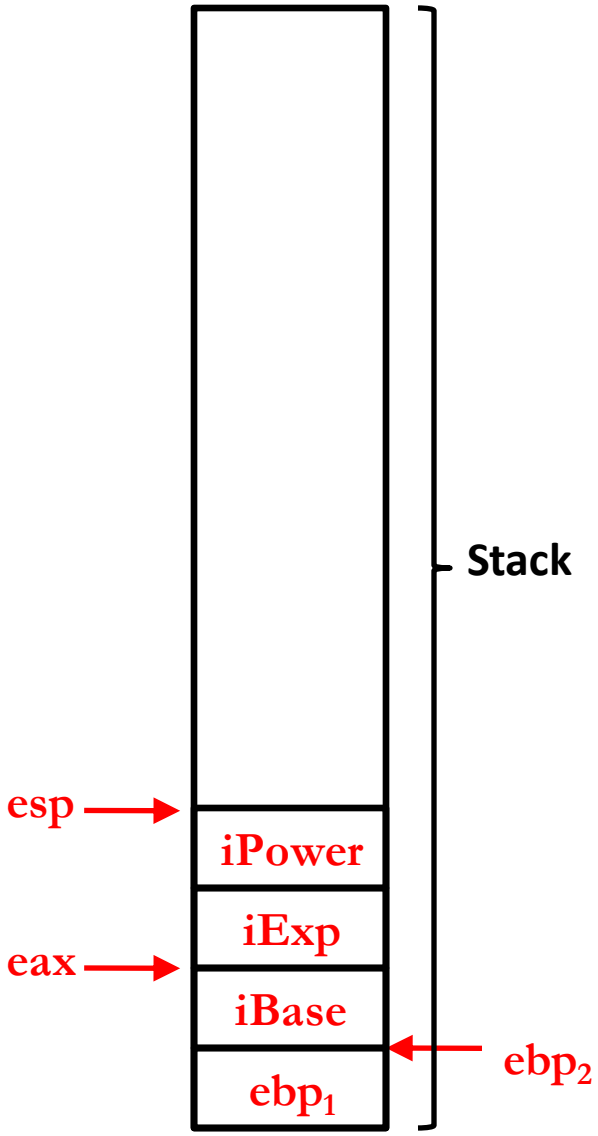


```
## printf("Enter the base:  ")
pushl   $cPrompt1
call    printf
addl    $4, %esp

## scanf("%d", &iBase)
leal    IBASE(%ebp), %eax
## Alternative to leal:
##    movl %ebp,%eax
##    addl $IBASE,%eax
pushl   %eax
pushl   $cScanfFormat
call    scanf
addl    $8, %esp
```



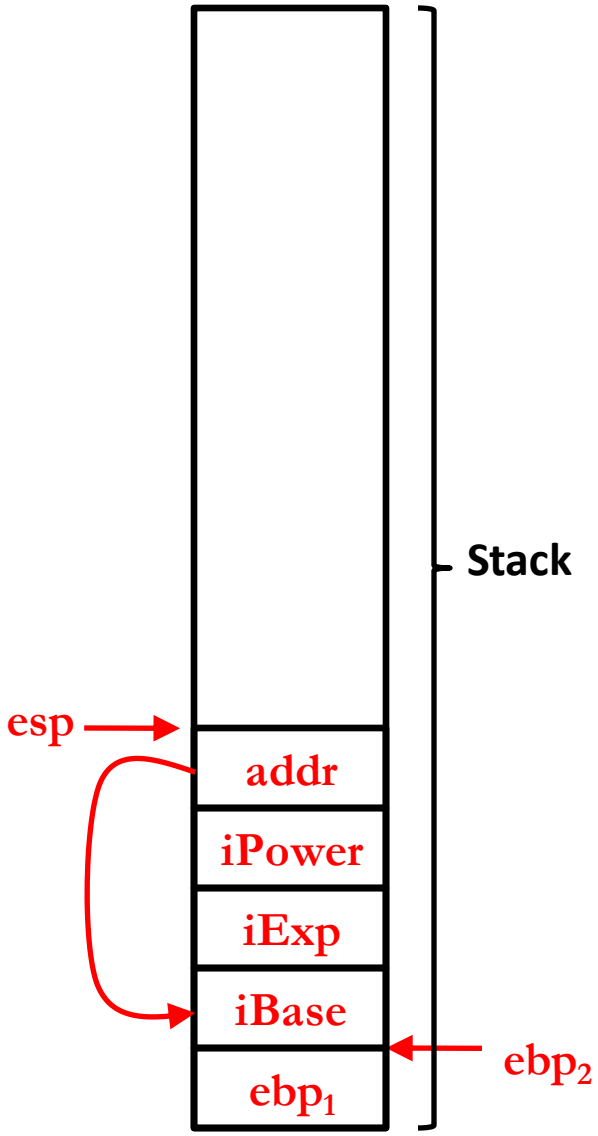
```
## printf("Enter the base:  ")
pushl   $cPrompt1
call    printf
addl    $4, %esp

## scanf("%d", &iBase)
leal    IBASE(%ebp), %eax
## Alternative to leal:
##    movl %ebp,%eax
##    addl $IBASE,%eax
pushl   %eax
pushl   $cScanfFormat
call    scanf
addl    $8, %esp
```

IBASE = -4

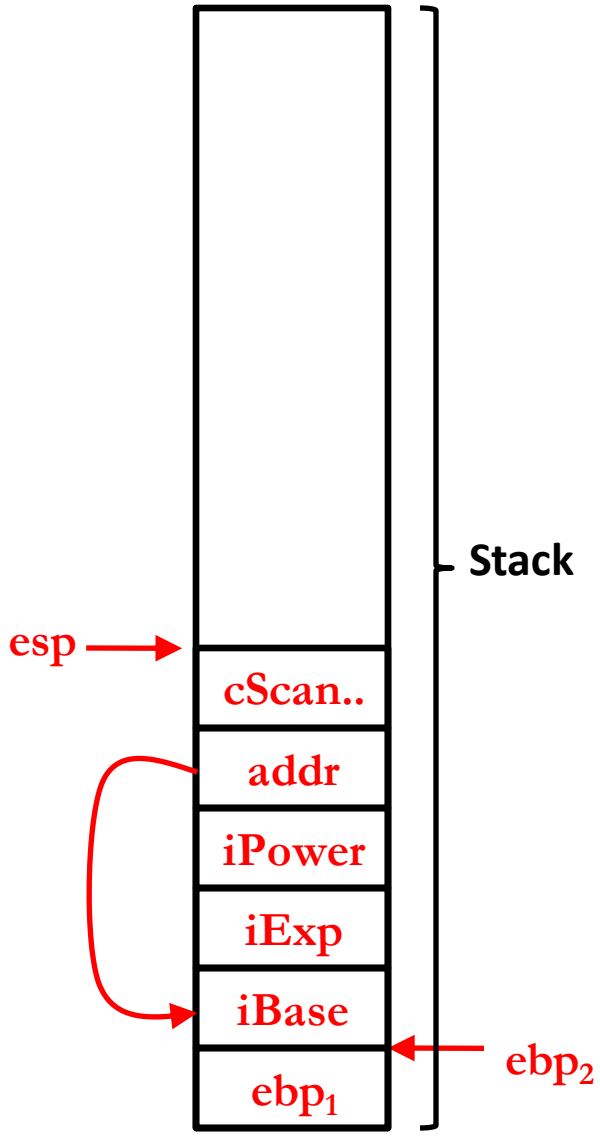


```
## printf("Enter the base:  ")
pushl   $cPrompt1
call    printf
addl    $4, %esp

## scanf("%d", &iBase)
leal    IBASE(%ebp), %eax
## Alternative to leal:
##    movl %ebp,%eax
##    addl $IBASE,%eax
pushl   %eax
pushl   $cScanfFormat
call    scanf
addl    $8, %esp
```

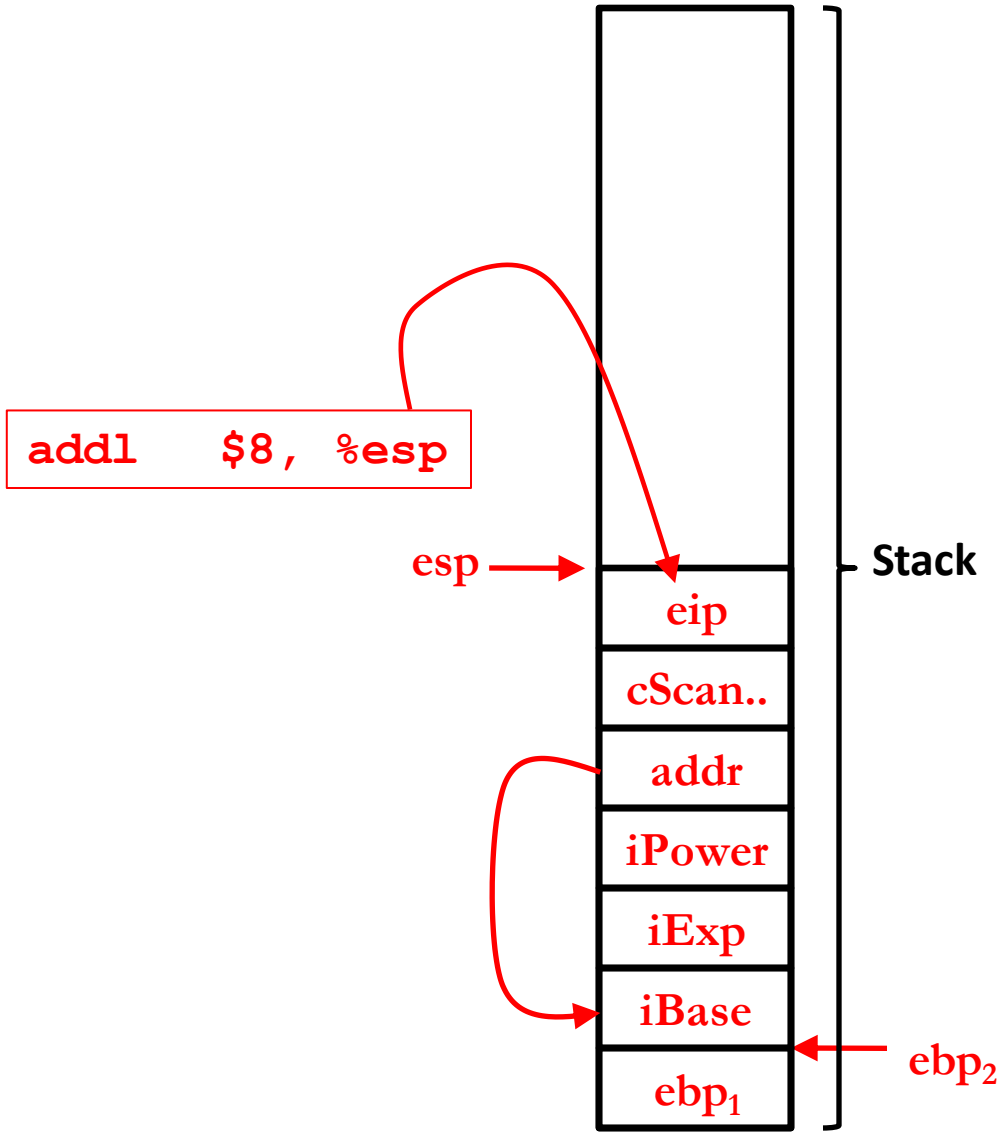
COPY BY REFERENCE



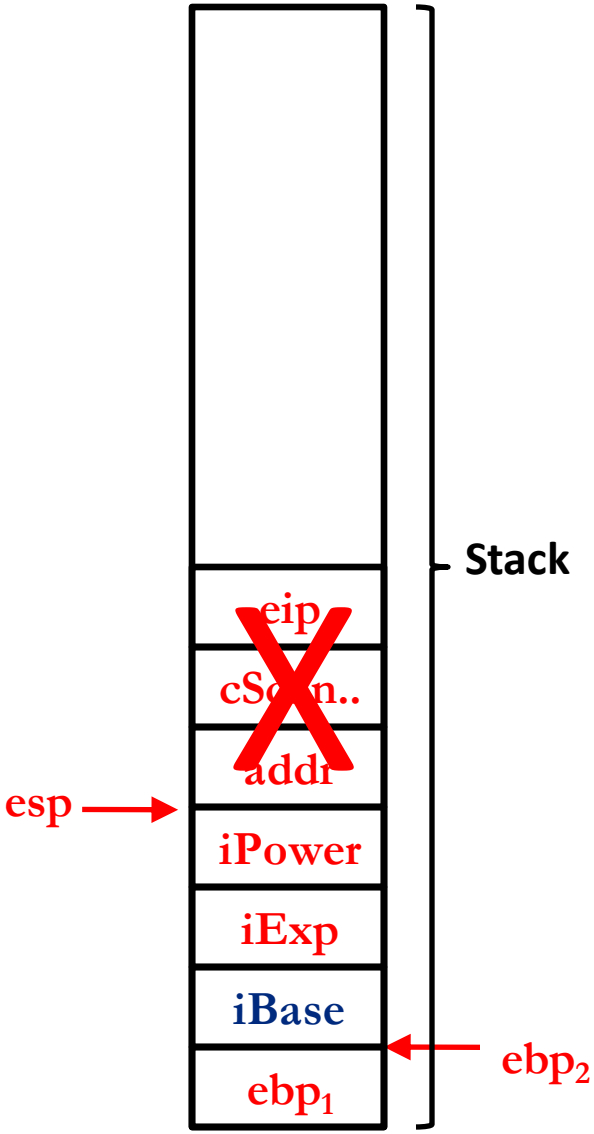
```
## printf("Enter the base:  ")
pushl   $cPrompt1
call    printf
addl    $4, %esp

## scanf("%d", &iBase)
leal    IBASE(%ebp), %eax
## Alternative to leal:
##    movl %ebp,%eax
##    addl $IBASE,%eax
pushl   %eax
pushl   $cScanfFormat
call    scanf
addl    $8, %esp
```



```
## printf("Enter the base:  ")
pushl   $cPrompt1
call    printf
addl    $4, %esp

## scanf("%d", &iBase)
leal    IBASE(%ebp), %eax
## Alternative to leal:
##    movl %ebp,%eax
##    addl $IBASE,%eax
pushl   %eax
pushl   $cScanfFormat
call    scanf
addl    $8, %esp
```



```
## printf("Enter the base:  ")
pushl   $cPrompt1
call    printf
addl    $4, %esp

## scanf("%d", &iBase)
leal    IBASE(%ebp), %eax
## Alternative to leal:
##    movl %ebp,%eax
##    addl $IBASE,%eax
pushl   %eax
pushl   $cScanfFormat
call    scanf
addl    $8, %esp
```



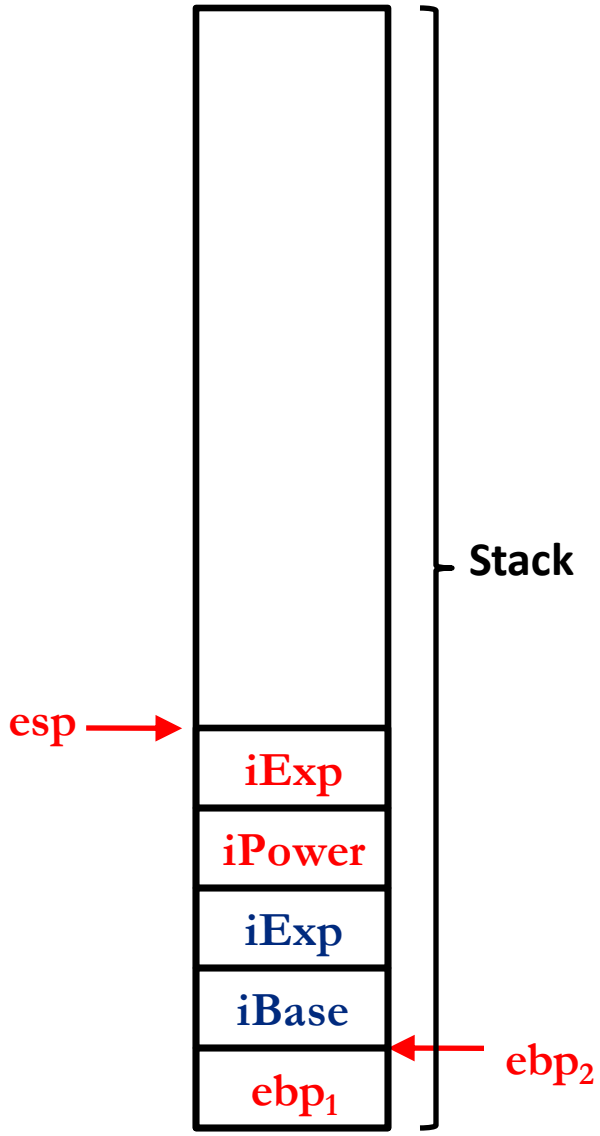
```
 ## printf("Enter the exponent:  ")
pushl   $cPrompt2
call    printf
addl    $4, %esp

## scanf("%d", &iExp)
leal    IEXP(%ebp), %eax
## Alternative to leal:
##    movl %ebp,%eax
##    addl $IEXP,%eax
pushl   %eax
pushl   $cScanfFormat
call    scanf
addl    $8, %esp
```

SAME SEQUENCE OF STEPS FOR THIS BLOCK AS WELL

```
## iPower = power(iBase, iExp)
pushl   IEXP(%ebp)
pushl   IBASE(%ebp)
call    power
addl    $8, %esp
movl    %eax, IPOWER(%ebp)
```

IEXP = -8

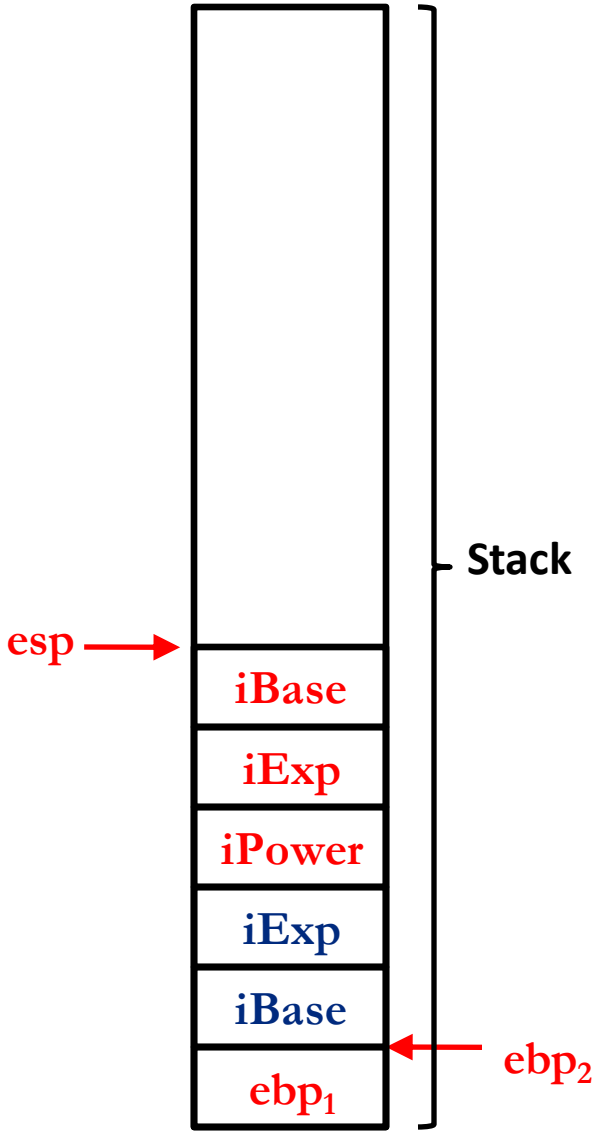
COPY BY VALUE



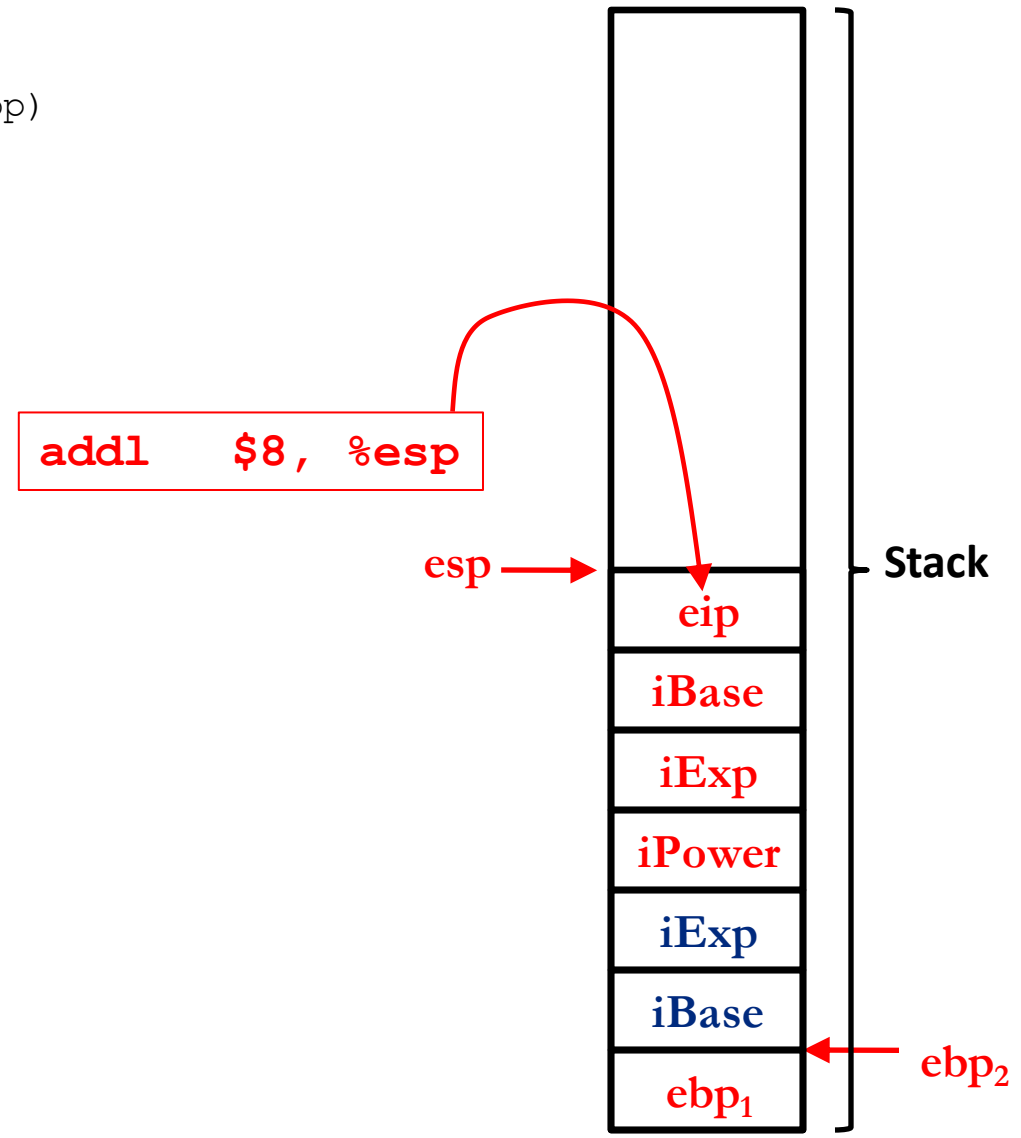
```
## iPower = power(iBase, iExp)
pushl   IEXP(%ebp)
pushl   IBASE(%ebp)
call    power
addl    $8, %esp
movl    %eax, IPOWER(%ebp)
```

IBASE = -4

COPY BY VALUE

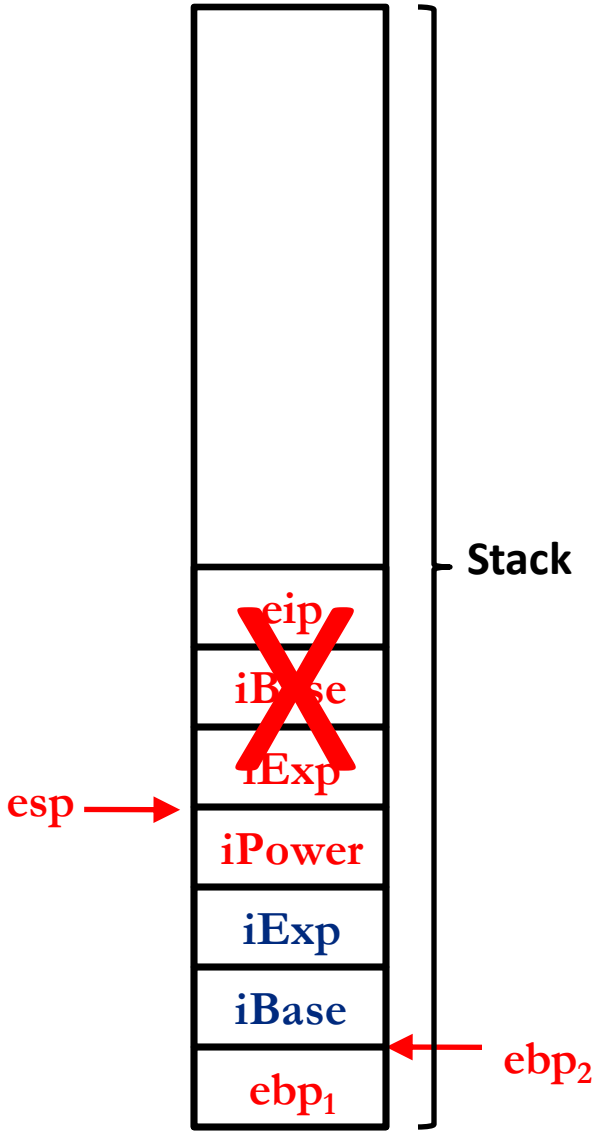


```
## iPower = power(iBase, iExp)
pushl   IEXP(%ebp)
pushl   IBASE(%ebp)
call    power
addl    $8, %esp
movl    %eax, IPOWER(%ebp)
```



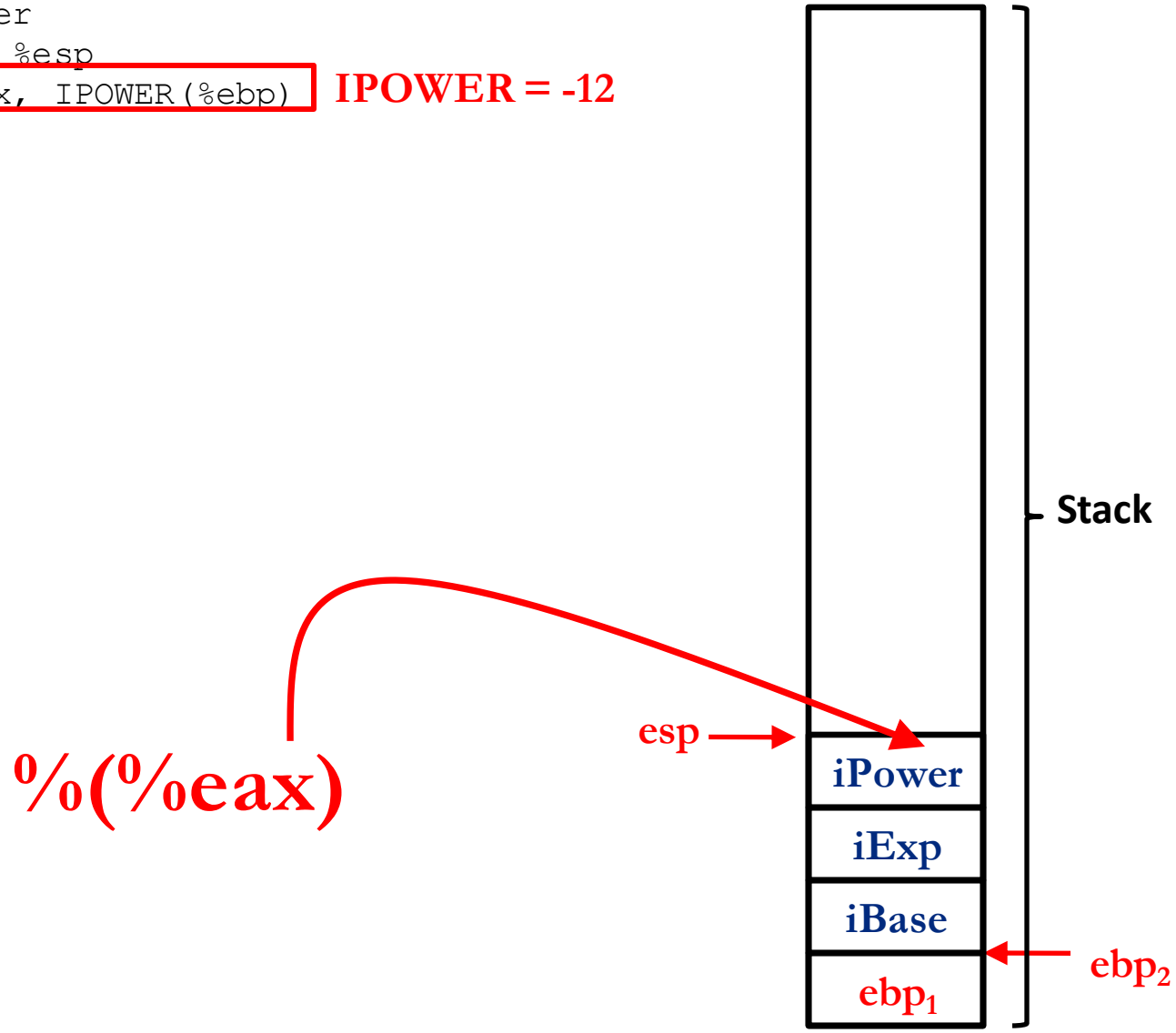
```
## iPower = power(iBase, iExp)
pushl   IEXP(%ebp)
pushl   IBASE(%ebp)
call    power
addl    $8, %esp
movl    %eax, IPOWER(%ebp)
```

Return value is saved in eax register



```
## iPower = power(iBase, iExp)
pushl   IEXP(%ebp)
pushl   IBASE(%ebp)
call    power
addl    $8, %esp
movl    %eax, IPOWER(%ebp)
```

IPOWER = -12

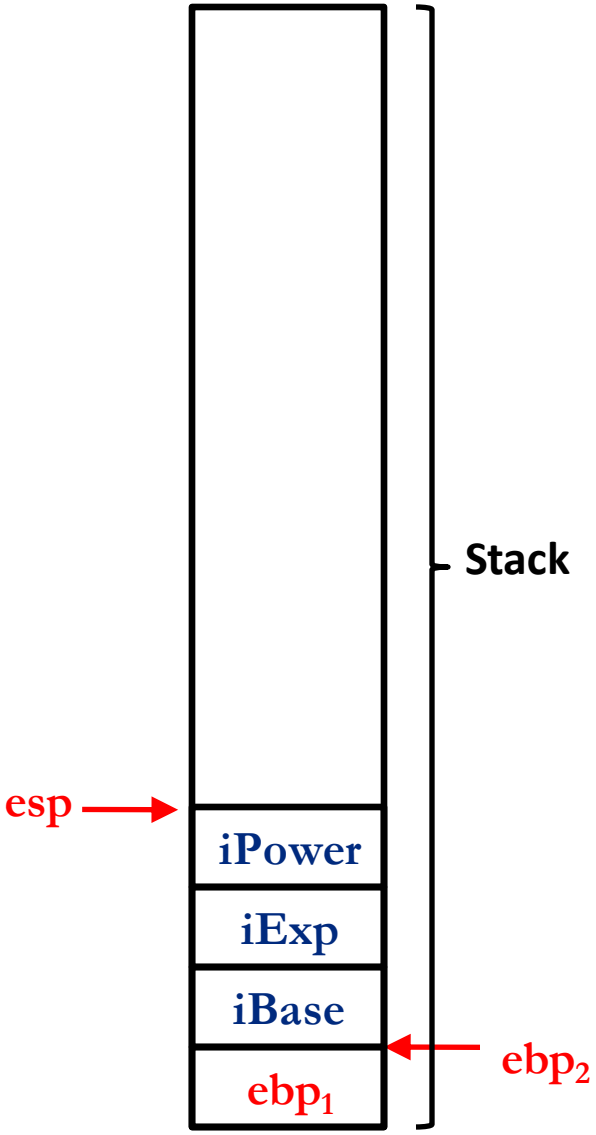


```
## printf("%d to the %d power is %d.\n", iBase, iExp, iPower)
pushl   IPOWER(%ebp)
pushl   IEXP(%ebp)
pushl   IBASE(%ebp)
pushl   $cResult
call    printf
addl    $16, %esp
```

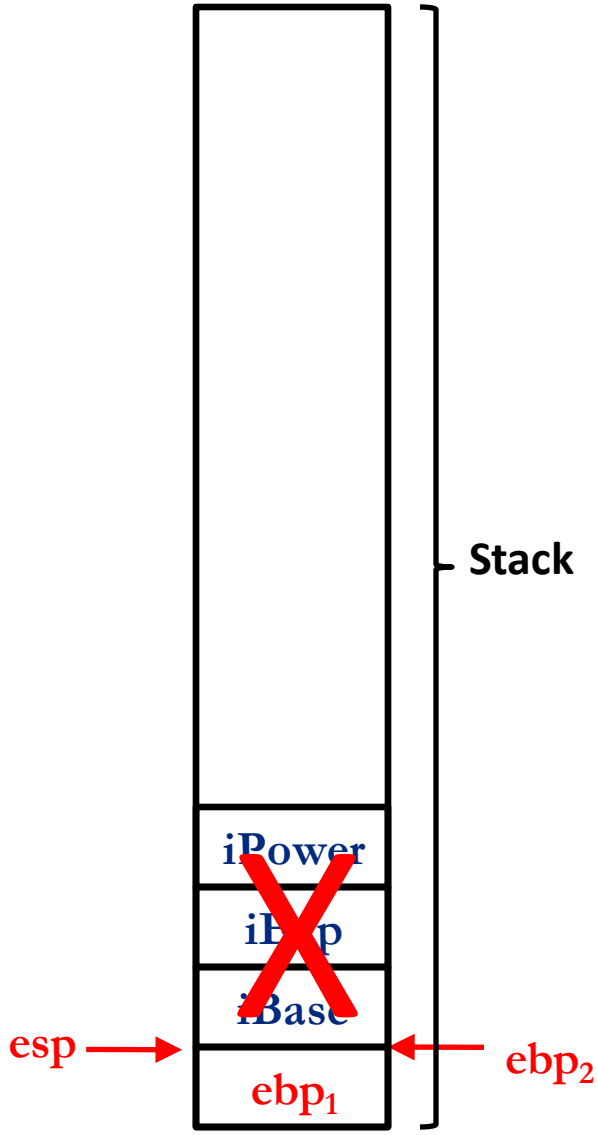
Can you figure out how these instructions will be executed?

```
## return 0
movl    $0, %eax
movl    %ebp, %esp
popl    %ebp
ret
```

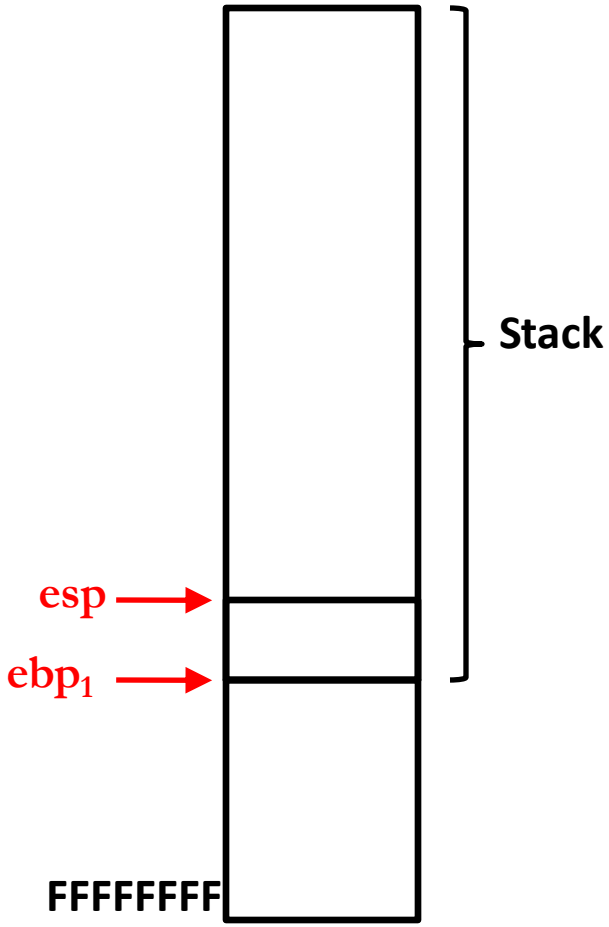
eax register set to 0



```
## return 0
movl    $0, %eax
movl    %ebp, %esp
popl    %ebp
ret
```



```
## return 0
movl    $0, %eax
movl    %ebp, %esp
popl    %ebp
ret
```



```
.section ".text"

    ## ------------------------------------------------------------
    ## int power(int iBase, int iExp)
    ## Return iBase raised to the iExp power, where iBase and iExp
    ## are non-negative.
    ## ------------------------------------------------------------

    ## Formal parameter offsets:
    .equ IBASE,     8
    .equ IEXP,      12

    ## Local variable offsets:
    .equ IPOWER,   -4
    .equ IINDEX,   -8

    .type   power,@function
```

You can define aliases for the same name

```
power:
        pushl   %ebp
        movl    %esp, %ebp

        ## int iPower = 1
        pushl   $1

        ## int iIndex
        subl    $4, %esp

        ## iIndex = 1
        movl    $1, IINDEX(%ebp)

    ...
        ## return iPower
        movl    IPOWER(%ebp), %eax
        movl    %ebp, %esp
        popl    %ebp
        ret
```

Init stack

Push local variables

Set return value to EAX

Restore stack

Return

Good Luck!

- Assignment 4 is due this Wednesday