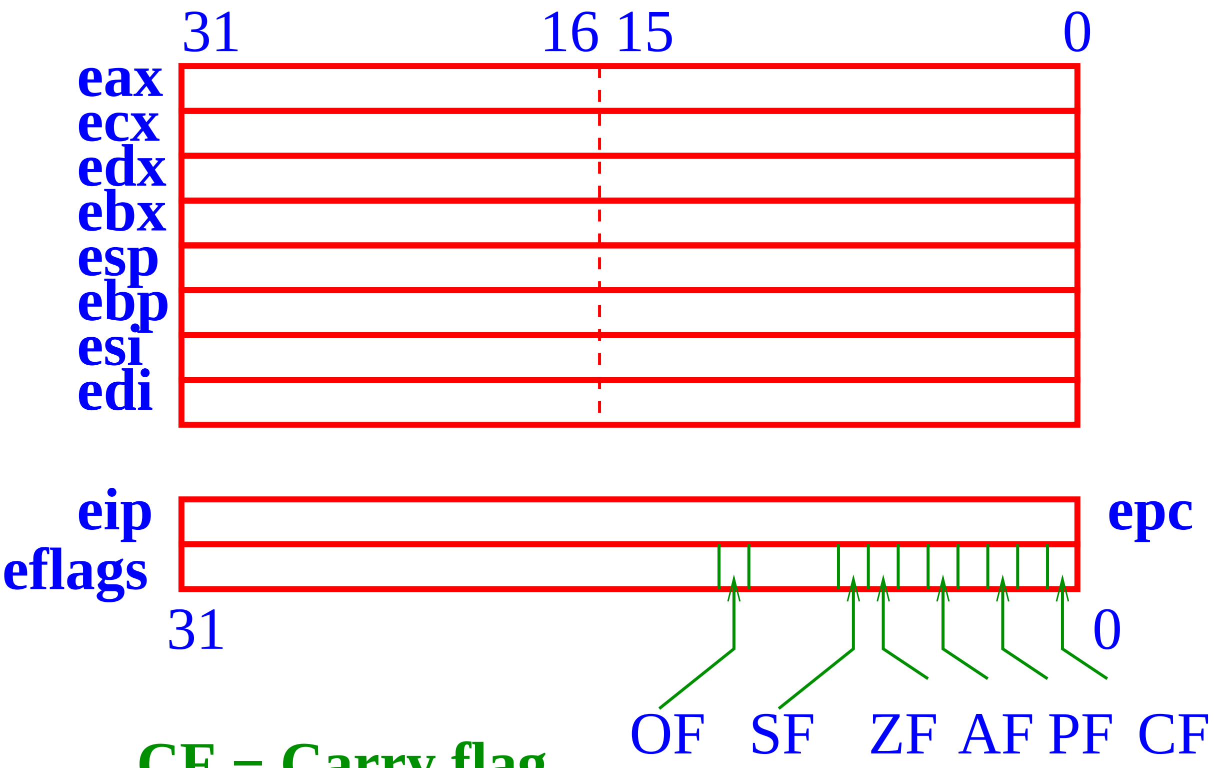
Introduction to Pentium ISA

ISA - Instruction Set Architecture - the systems programmer's view of a computer.



CF − Carry flag
ZF − Zero flag
SF − Sign flag
OF − Overflow flag

Figure 1: Pentium Integer Registers

A Sample C Program: main.c

```
int fib[10], n = 4, arg[10] = {5, 7, 10, 13} ;
int main()
{
 int fibonacci(int), i ;

 for(i=0; i<n; ++i) {
    fib[i] = fibonacci(arg[i]);
    printf("fib(%d) = %d\n", arg[i], fib[i]);
 }
}
```

C Program to Assembly Language Program

```
$ cc -S main.c
$ ls -l main.*
-rw-r--r-- 1 goutam users 191 Jul 16 11:04 main.c
-rw-r--r-- 1 goutam users 945 Jul 16 11:04 main.s
$
```

Assembly Language Program: main.s

```
  .file  "main.c"
  .globl n            # Global Object
  .data               # In the .data section
  .align 4            # Align with 4B Boundary(?)
  .type  n,@object
  .size  n,4          # Size 4B
n:                    # Label "n"
  .long  4            # Initial value is 4
  .globl arg          # Global Object
  .align 32           # Align with 32B Boundary
  .type  arg,@object
```

Assembly Language Program: main.s

```
  .size  arg,40       # Size 4*10 = 40B
arg:                  # Label "arg"
  .long  5            # Initial value of arg[0]
  .long  7            # Initial value of arg[1]
  .long  10           # Initial value of arg[2]
  .long  13           # Initial value of arg[3]
  .zero  24           # 24B filled with 0s
  .section  .rodata   # Read-only data
.LC0:                 # Label ".LC0"
  .string  "fib(%d) = %d\n"  # Read-only data
```

Assembly Language Program: main.s

```
  .text             # .text or Code section
  .align 2          # Align with 4B boundary(?)
  .globl main       # Global function
  .type  main,@function
main:               # Label
  pushl  %ebp       # Push(save) ebp on user Stack
  movl  %esp, %ebp  # Copy esp to ebp
  pushl  %ebx       # Push(save) ebx on Stack
  subl  $4, %esp    # esp = esp - 4
```

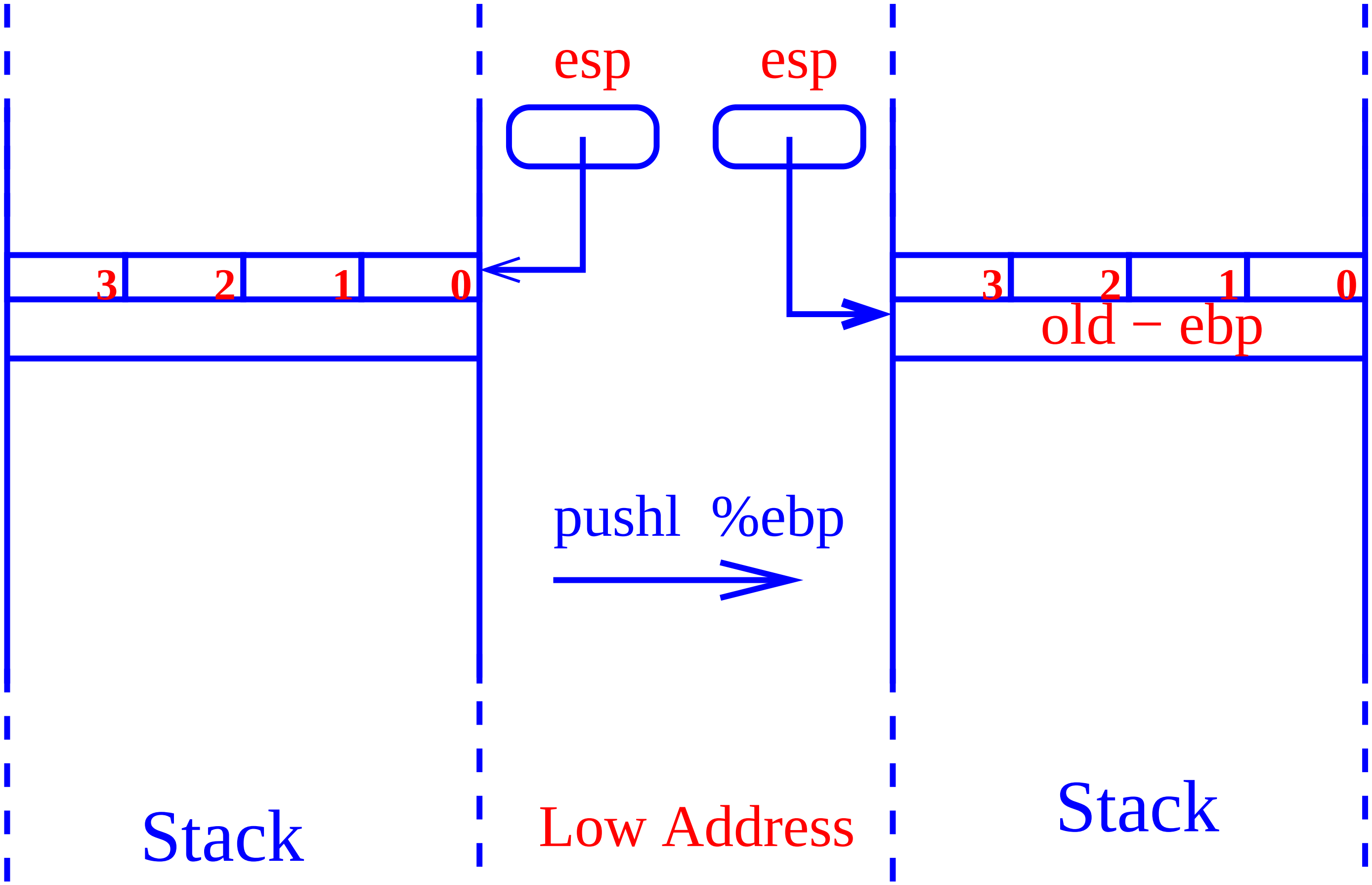
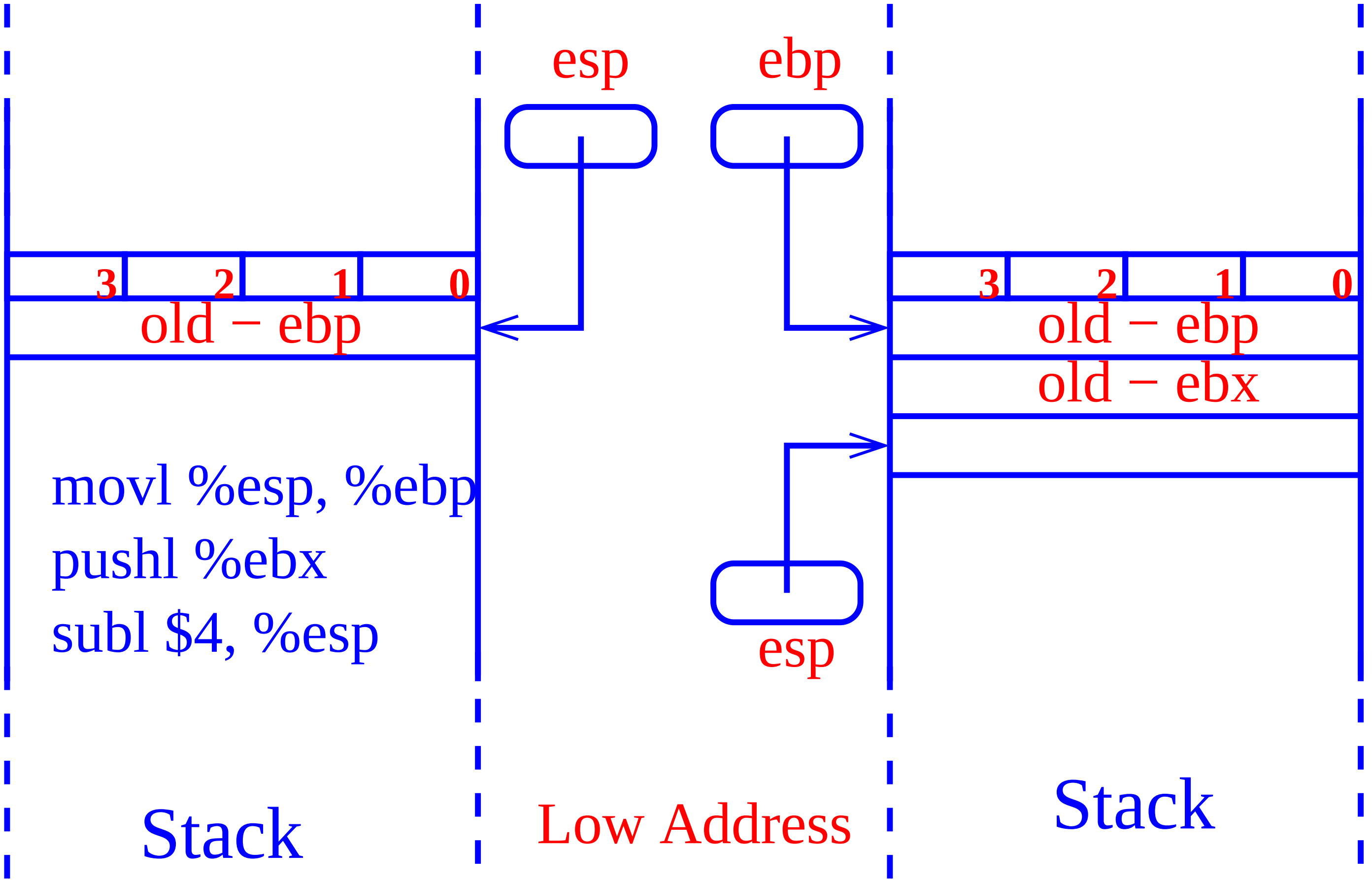


Figure 2: **User Stack**



```
movl %esp, %ebp
pushl %ebx
subl $4, %esp
```

Figure 3: **User Stack**

Assembly Language Program: main.s

```
andl  $-16, %esp        # LS 4-bits are 0s,
                        # 16B Boundary
movl  $0, %eax          # eax = 0
subl  %eax, %esp        # ???
movl  $0, -8(%ebp)      # Memory[ebp - 8] = 0
                        # Local variable i
                        # 32-bit 2's complement
```

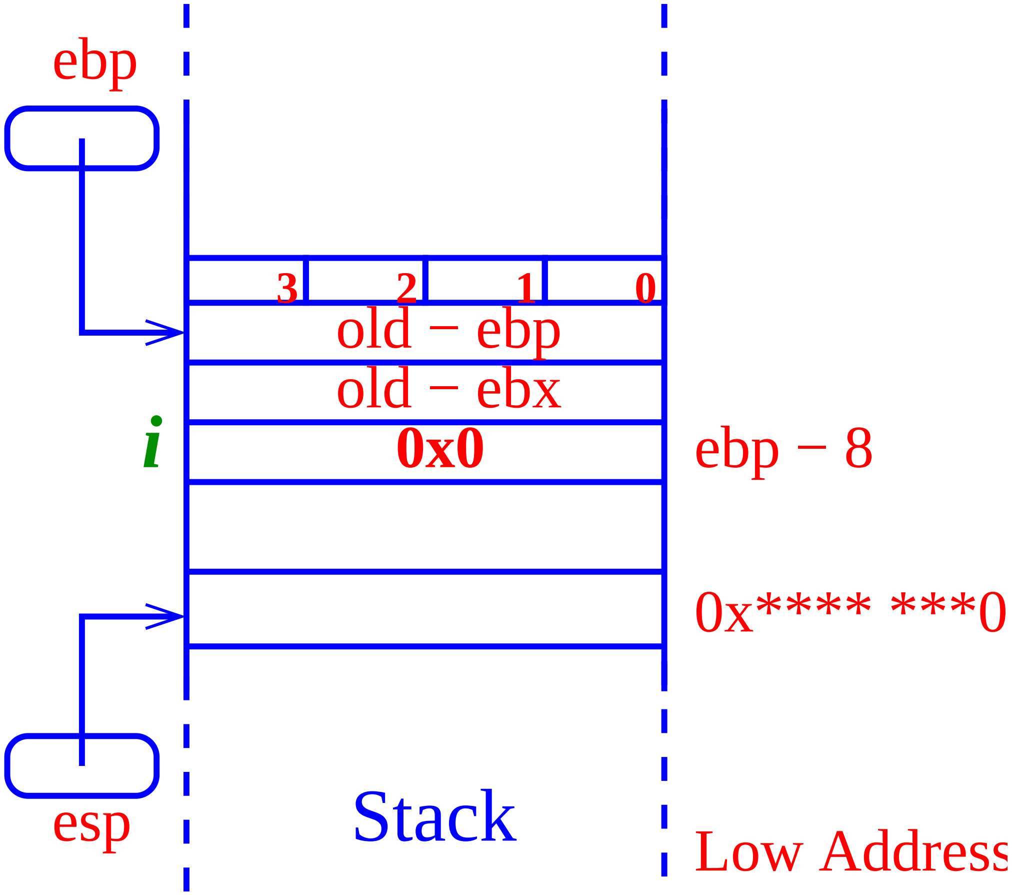


Figure 4: **User Stack**

Assembly Language Program: main.s

```
.L2:
  movl  -8(%ebp), %eax    # eax = i
  cmpl  n, %eax           # compare n and i
  jl  .L5                 # if i < n goto .L5
  jmp  .L3                # goto .L3
.L5:                      # Label
  movl  -8(%ebp), %ebx    # ebx = i
  subl  $12, %esp         # esp = esp - 12
```

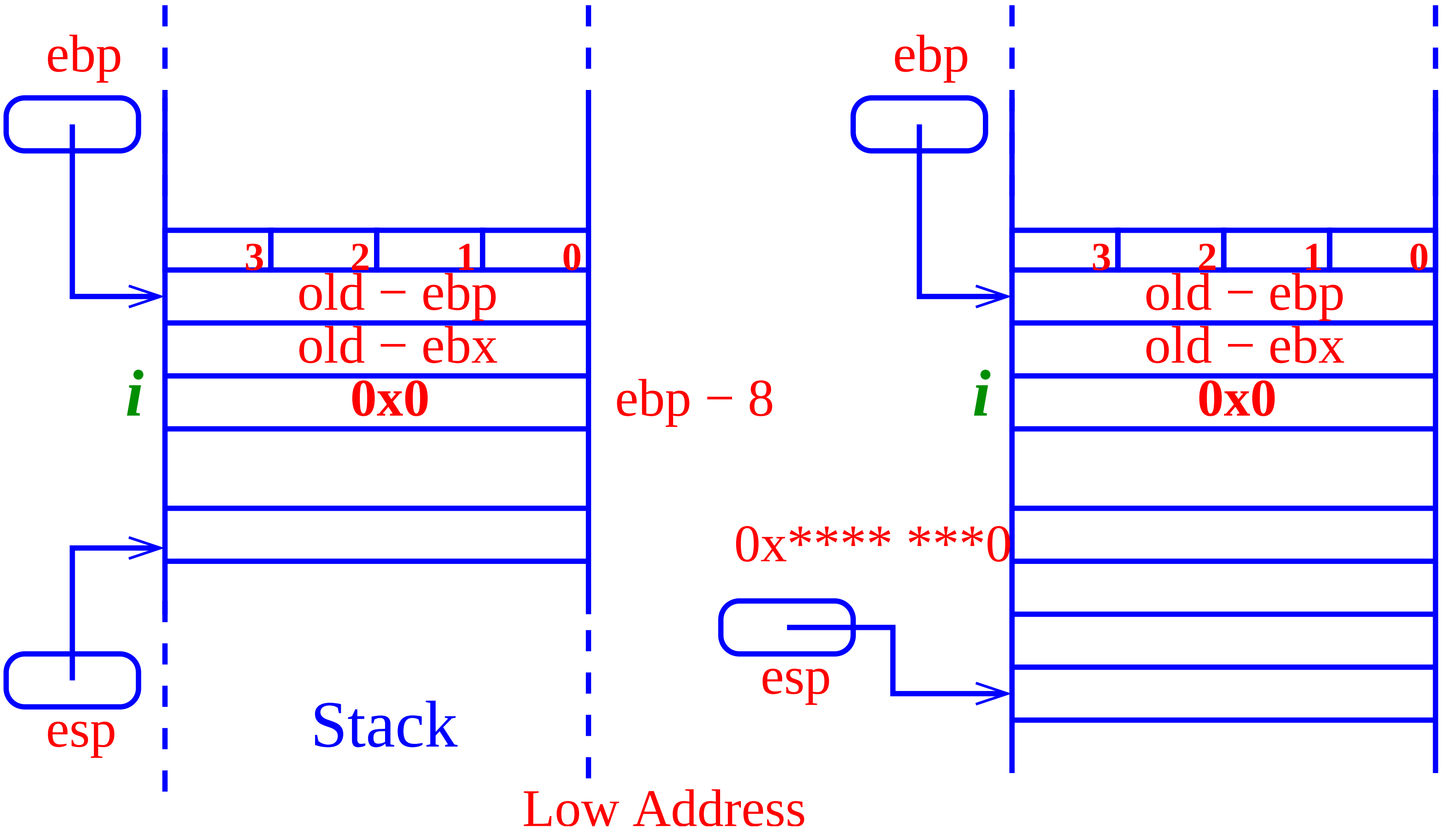


Figure 5: **User Stack**

Assembly Language Program: main.s

```
movl  -8(%ebp), %eax  # eax = i
pushl  arg(,%eax,4)   # Push Memory[arg + 4*eax]
                      # i.e. push arg[0]
call  fibonacci       # Call fibonacci
addl  $16, %esp       # esp = esp + 16
```

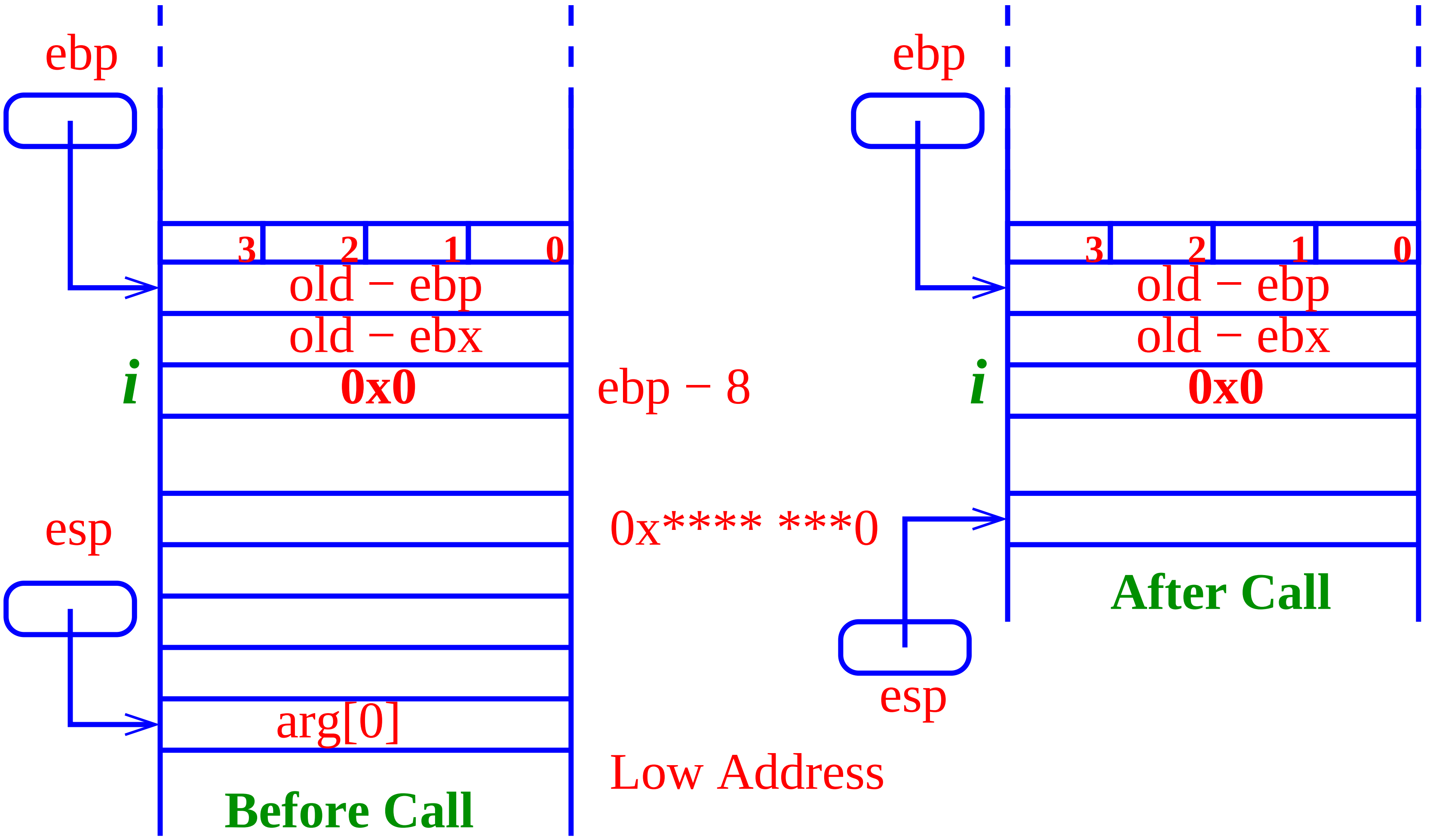


Figure 6: **User Stack**

Assembly Language Program: main.s

```
movl  %eax, fib(,%ebx,4) # Memory[fib+4*i]=eax
                         # returned value in eax
                         # fib[i] = eax
subl  $4, %esp           # esp = esp - 4
movl  -8(%ebp), %eax     # eax = Memory[i]
pushl  fib(,%eax,4)      # Push fib[i]
movl  -8(%ebp), %eax     # eax = i
pushl  arg(,%eax,4)      # Push arg[i]
pushl  $.LC0             # Push the address of
                         # format string
```

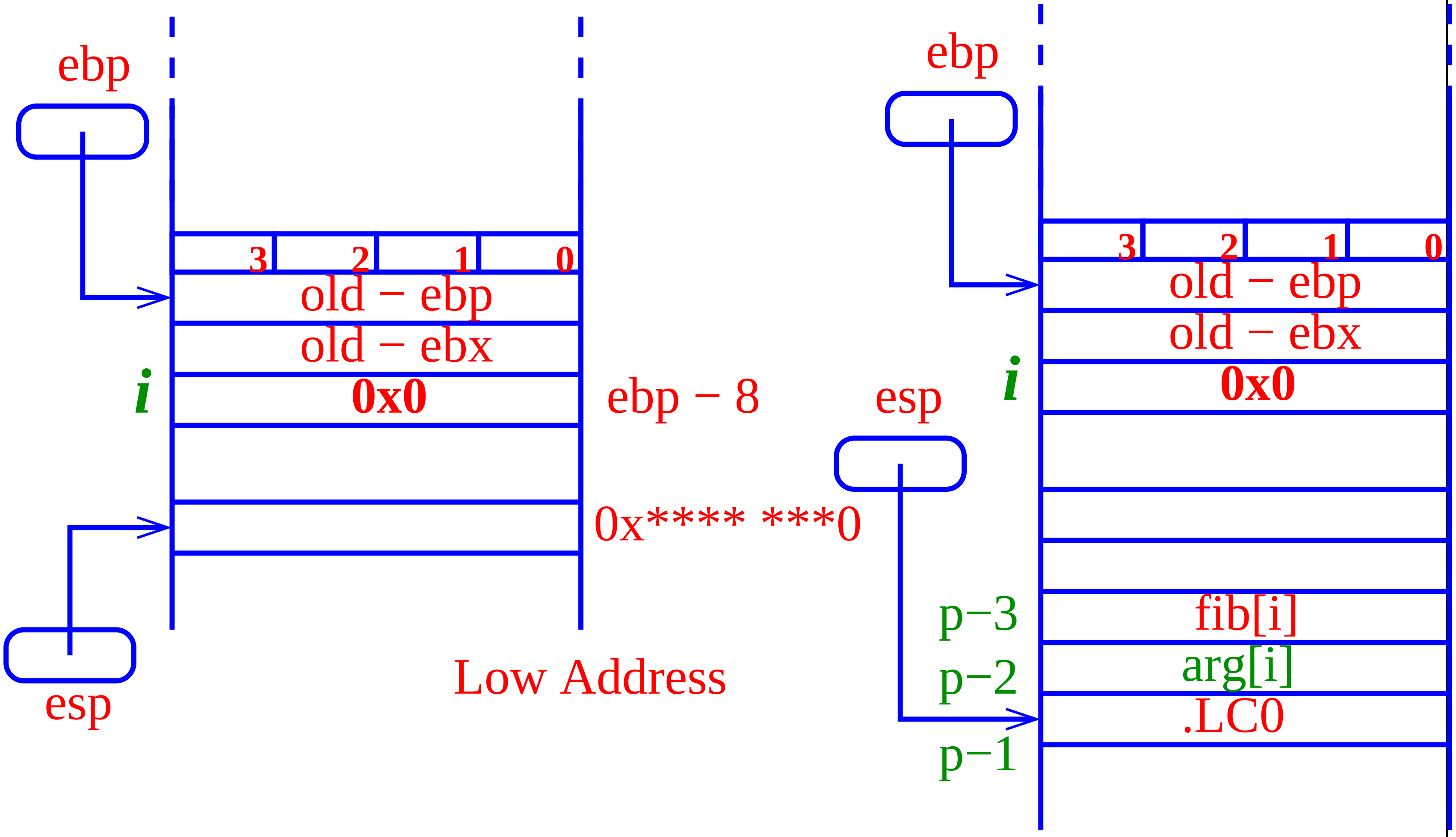


Figure 7: **User Stack**

Assembly Language Program: main.s

```
call   printf            # Call printf
addl   $16, %esp         # esp = esp + 16
leal   -8(%ebp), %eax    # eax = ebp - 8 (&i)
incl   (%eax)            # Increment Memory[eax]
                         # (++i)
jmp   .L2                # goto .L2
```

Assembly Language Program: main.s

```
.L3:
  movl  -4(%ebp), %ebx  # ebx = Memory[ebp - 4]
                        # Old value of ebx is restored
  leave                 # Clear stack
  ret                   # Return
.Lfe1:
  .size  main,.Lfe1-main # Size is (.Lfe1 - main)
  .comm  fib,40,32      # Reserve 40 bytes
                        # (32B aligned)
                        # in .common area.
  .ident  "GCC: (GNU) 3.2 20020903 (Red Hat Linux 8.0 3.2-7
```

Manuals

- **GNU Assembler (as):** **http://www.gnu.org/software/binutils/manual/gas-2.9.1/as.html**
- **Intel Pentium:** **http://developer.intel.com/design/pentium/manuals/**