

Arrays

000004ed <mystery>:

```
4ed: 55          push  %ebp
4ee: 89 e5        mov   %esp,%ebp
4f0: 83 ec 10      sub   $0x10,%esp
4f3: c7 45 f8 00 00 00 00 movl  $0x0, -0x8(%ebp)
4fa: c7 45 fc 00 00 00 00 movl  $0x0, -0x4(%ebp)
501: eb 18        jmp   51b <mystery+0x2e>
503: 8b 45 fc      mov   -0x4(%ebp),%eax
506: 8d 14 85 00 00 00 00 lea   0x0(,%eax,4),%edx
50d: 8b 45 08      mov   0x8(%ebp),%eax
510: 01 d0        add   %edx,%eax
512: 8b 00        mov   (%eax),%eax
514: 01 45 f8      add   %eax, -0x8(%ebp)
517: 83 45 fc 01   addl  $0x1, -0x4(%ebp)
51b: 8b 45 fc      mov   -0x4(%ebp),%eax
51e: 3b 45 0c      cmp   0xc(%ebp),%eax
521: 7c e0        jl   503 <mystery+0x16>
523: 8b 45 f8      mov   -0x8(%ebp),%eax
526: c9          leave
527: c3          ret
```

00000528 <main>:

```
528: 55          push  %ebp
529: 89 e5        mov   %esp,%ebp
52b: 83 ec 10      sub   $0x10,%esp
52e: c7 45 f0 01 00 00 00 movl  $0x1, -0x10(%ebp)
535: c7 45 f4 02 00 00 00 movl  $0x2, -0xc(%ebp)
53c: c7 45 f8 03 00 00 00 movl  $0x3, -0x8(%ebp)
543: 6a 03        push  $0x3
545: 8d 45 f0      lea   -0x10(%ebp),%eax
548: 50          push  %eax
549: e8 9f ff ff ff call  4ed <mystery>
54e: 83 c4 08      add   $0x8,%esp
551: 89 45 fc      mov   %eax, -0x4(%ebp)
554: b8 00 00 00 00 mov   $0x0,%eax
559: c9          leave
55a: c3          ret
```

Structures

What is the size (in bytes) for the following structures on a 32-bit machine installed with the Linux Operating System?

sizeof(int) = 4 bytes
sizeof(short) = 2 bytes
sizeof(char) = 1 byte

Structure	Size (in bytes)
<pre>struct foo { int d1; char c1; int d2; }</pre>	
<pre>struct foo { int d1; char c1; int d2; char c2; short s; };</pre>	
<pre>struct foo { int d1; int d2; char c1; char c2; short s; };</pre>	
<pre>struct foo { char c1; int d1; short s; int d2; char c2; };</pre>	

Pointers

Assume variables a and b are stored at `-0x8(%ebp)` and `-0x4(%ebp)` respectively. Write the equivalent C expressions for the following assembly snippets.

S.No.	Assembly Instruction	Corresponding C code
1	<code>movl -0x4(%ebp),%eax</code> <code>movl (%eax),%eax</code> <code>movl %eax,-0x8(%ebp)</code>	
2	<code>leal -0x4(%ebp),%eax</code> <code>movl %eax,-0x8(%ebp)</code>	
3	<code>movl -0x4(%ebp),%eax</code> <code>movl %eax,-0x8(%ebp)</code>	
4	<code>movl -0x4(%ebp),%eax</code> <code>movl (%eax),%edx</code> <code>movl -0x8(%ebp),%eax</code> <code>movl %edx,(%eax)</code>	
5	<code>movl -0x4(%ebp), %eax</code> <code>movl (%eax), %edx</code> <code>movl -0x8(%ebp), %eax</code> <code>movl (%eax), %eax</code> <code>addl %eax, %edx</code> <code>movl -0x4(%ebp), %eax</code> <code>movl %edx, (%eax)</code>	

Jumps

In the disassembled version of the machine code shown below, instructions at addresses 0x66, 0x6c, and 0x73 use relative encoding to encode jump targets. All the values in the disassembled code below (including the address column) are represented using hexadecimal numbers.

Address	Opcode	Assembly Code
0x62:	83 7d 08 00	cmpl \$0x0,0x8(%ebp)
0x66:	74 0d	je X
0x68:	83 7d 0c 00	cmpl \$0x0,0xc(%ebp)
0x6c:	74 07	je Y
0x6e:	b8 01 00 00 00	mov \$0x1,%eax
0x73:	eb 05	jmp Z
0x75:	b8 00 00 00 00	mov \$0x0,%eax
0x7a:	85 c0	test %eax,%eax
0x7c:	0f 9e c0	setle %al
0x7f:	88 45 ff	mov %al,-0x1(%ebp)

What is the value of the addresses X, Y and Z? The values of X, Y, and Z should be written in hexadecimal notation. Remember the addresses X, Y, and Z, in assembly code will refer to actual addresses (e.g. 0x75) even though relative encoding is used in the machine code (opcode).

ANSWER:

X = 0x_____

Y = 0x_____

Z = 0x_____