Please refer to the following code for all questions. Assume that the machine is little-endian.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  unsigned int p[2];
  unsigned int x;
  unsigned char *cp;
  unsigned char c;

  p[0] = 0x28f4a13e;
  p[1] = 0xbc70695d;
  cp = (unsigned char *) p;

  c = cp[0];
  printf("0x%02x\n", c);

  c = cp[6];
  printf("0x%02x\n", c);

  c = (p[0] >> 8);
  printf("0x%02x\n", c);

  x = ((p[0] >> 16) | (p[1] << 16));
  printf("0x%08x\n", x);

  return 0;
}
```

Question 1

What is the first line of output?
Choices are:

A. 0x28
B. 0x28f4
C. 0x28f4a13e
D. 0x3e
E. 0x3ea1f428
F. 0x5d
G. 0x82
H. 0xa13e
I. 0xbc
J. 0xe3

Question 2

What is the second line of output?
Choices are:

A. 0x07
B. 0x5d
C. 0x69
D. 0x695d
E. 0x70
F. 0x96
G. 0xbc
H. 0xbc70
I. 0xbc70695d
J. 0xd59607cb

Question 3

What is the third line of output?
Choices are:

A. 0x13
B. 0x28
C. 0x28f4
D. 0x28f4a13e
E. 0x28f4a1
F. 0x3e
G. 0xa13e
H. 0xa1
I. 0xf4
J. 0xf4a13e

Question 4

What is the last line of output?
Choices are:

A. 0x28f4
B. 0x28f4695d
C. 0x28f4bc70
D. 0x695d
E. 0x695d28f4
F. 0xa13e
G. 0xa13e695d
H. 0xa13ebc70
I. 0xbc70
J. 0xbc70a13e

Answers to the clicker questions for the CStuff-2 Lecture

You can cut/paste/compile/run to confirm these answers.

Take a look at the following picture, which shows `p` and `cp`. The ordering of `cp` is because the machine is little endian:

```
             cp[3] cp[2] cp[1] cp[0]
p[0]:       0x   28    f4    a1    3e

             cp[7] cp[6] cp[5] cp[4]
p[1]:       0x   bc    70    69    5d
```

This diagram clearly answers the first two questions:

- **Question 1**: `cp[0] is 0x3e`: D
- **Question 2**: `cp[6] is 0x70`: E

Now, `(p[0] >> 8)` will shift the least significant byte of `p[0]` off, and you are left with 0x28f4a1. When we store it as a byte, it takes the least significant byte, and jettisons the rest:

- **Question 3**: `c is 0xa1`: H

For question 4, `(p[0] >> 16)` will shift the least significant two bytes of `p[0]` off: 0x28f4.

`(p[1] << 16)` will shift the least significant two bytes into the most significant positions, and put zeros in their place: 0x695d0000. When you OR them together, you get the answer:

- **Question 4**: `x is 0x695d28f4`: E