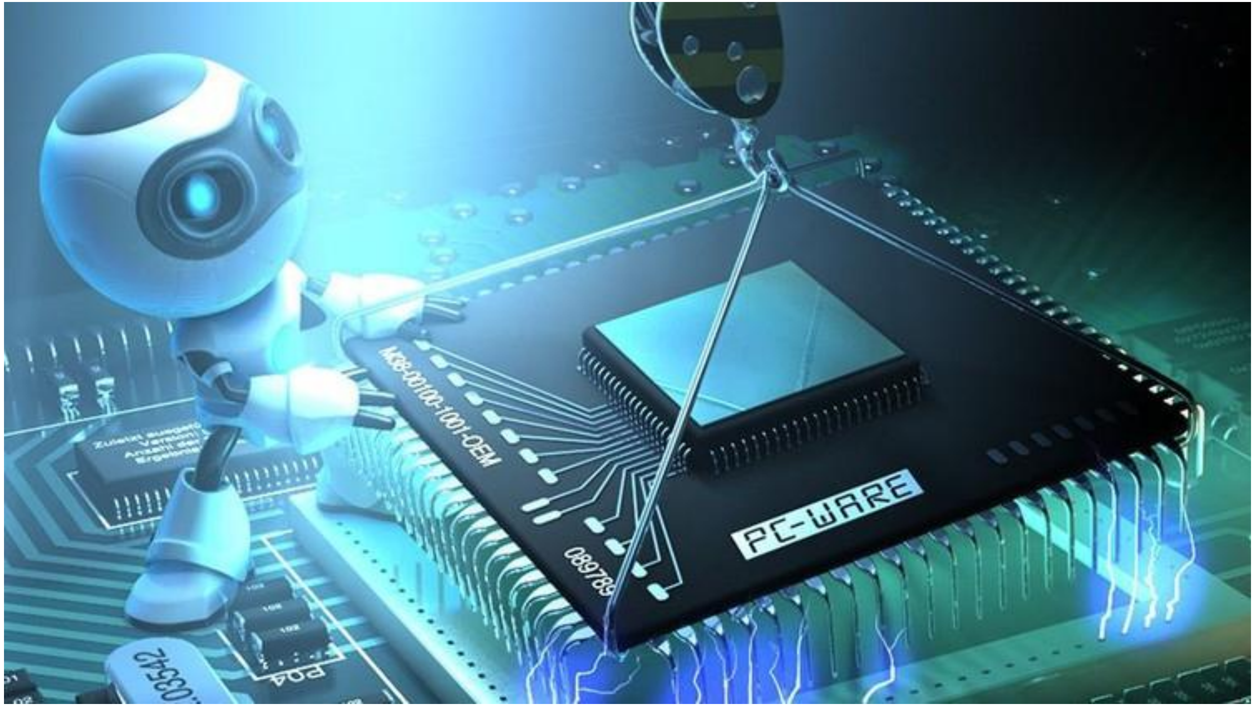
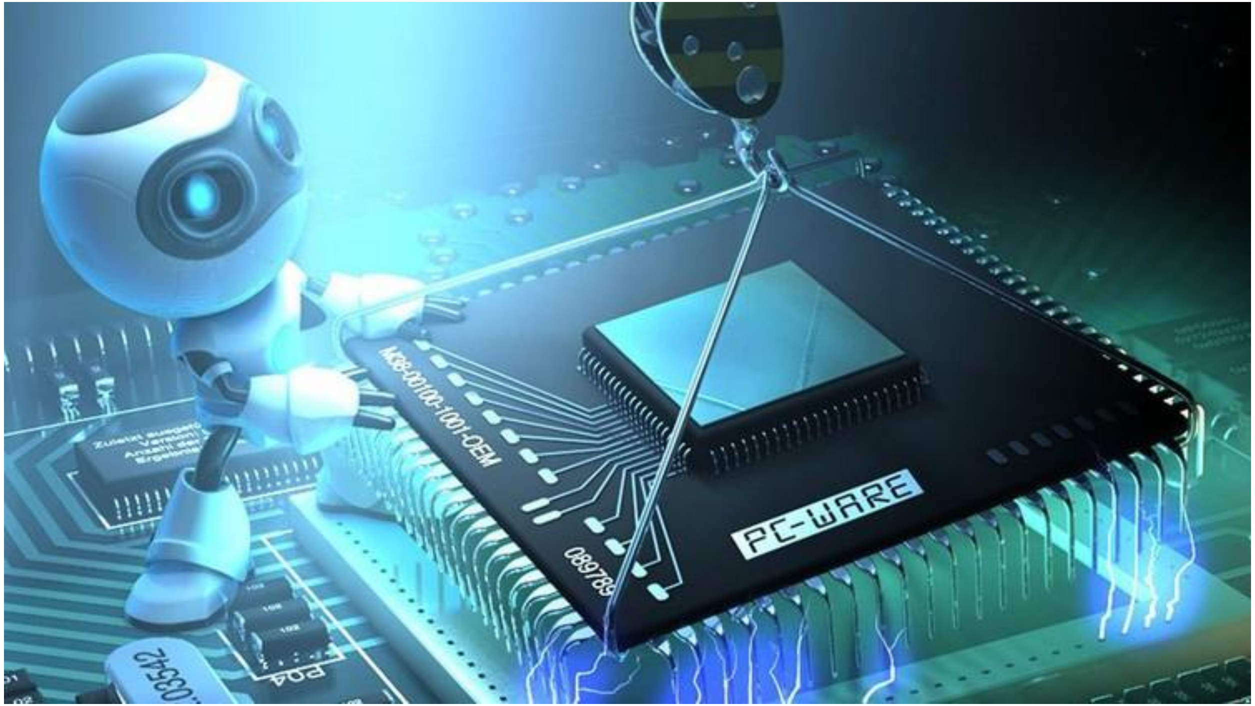
Exam 1



Exam 1 - Problem 1



Exam 1 - Problem 1

Given the following code written in C:

```
typedef struct {
    int a ;
    char b;
    char c;
    double d;
} s1;

typedef struct {
    short e[5];
    s1 f;
} s2;

short F(s1 *high, int ball, char *tail);
int examine(s1 one, char two, s2 *three) {
    char vl1;
    int vl2;
}
```

Exam 1 - Problem 1

a) **Draw** how the structures **s1** and **s2** would be stored in memory, clearly indicating the displacements with respect to the beginning and the size of all the fields.

b) **Draw** the activation block of the examine function, clearly indicating the relative offsets to the EBP register needed to access the parameters and local variables.

c) **Translate** the following statement to x86 assembler, assuming it's inside the examine function:

```
vl1=two+one.b;
```

Exam 1 - Problem 1

d) **Translate** the following statement to x86 assembler, assuming it's inside the examine function:

```
three->e[1]=F(&one, vl2, &one.c);
```

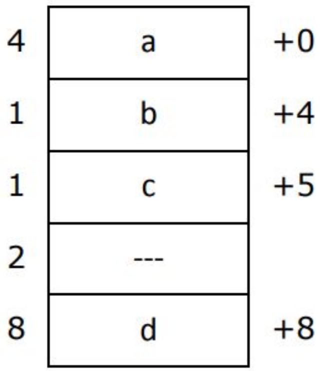
e) **Translate** the following statement to x86 assembler, assuming it's inside the examine function:

```
if (vl2 > 0)  { vl2 = three->f.a; }
```

Exam 1 - Problem 1

a) **Draw** how the structures `s1` and `s2` would be stored in memory, clearly indicating the displacements with respect to the beginning and the size of all the fields.

s1 (16 bytes)



4 a +0

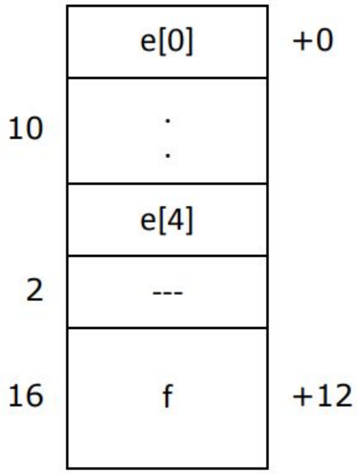
1 b +4

1 c +5

2 ---

8 d +8

s2 (28 bytes)



e[0] +0

10 .
.

e[4]

2 ---

16 f +12

Exam 1 - Problem 1

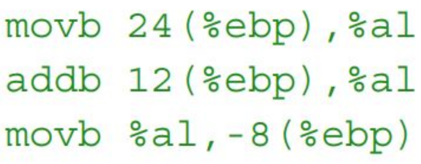
b) **Draw** the activation block of the examine function, clearly indicating the relative offsets to the EBP register needed to access the parameters and local variables.

1	vl1	-8
3	---	
4	vl2	-4
4	ebp old	<--%ebp
4	ret	
16	one	8
1	two	24
3	---	
4	three	28

Exam 1 - Problem 1

c) **Translate** the following statement to x86 assembler, assuming it's inside the `examine` function:

```
vl1=two+one.b;
```



```
movb  24(%ebp),%al
addb  12(%ebp),%al
movb  %al,-8(%ebp)
```

```
# %al = two
# %al = two + one.b
# vl1 = two + one.b
```

```
typedef struct {
        int a ;
        char b;
        char c;
        double d;
} s1;

int examine(s1 one, char two, s2 *three)
{
        char vl1;
        int vl2;
}
```

Lucas Bazilio - Udemy

Exam 1 - Problem 1

d) **Translate** the following statement to x86 assembler, assuming it's inside the examine function:

```
three->e[1]=F(&one, vl2, &one.c);
```

```
leal 13(%ebp), %eax     # %eax = &one.c
pushl %eax   # push &one.c
pushl -4(%ebp)  # push vl2
leal 8(%ebp), %eax
pushl %eax   # push &one
call F
addl $12, %esp
movl 28(%ebp), %ecx     # %ecx = &three
movw %ax, 2(%ecx)     # three->e[1] = result
```

x86

```
typedef struct {
        int a ;
        char b;
        char c;
        double d;
} s1;

typedef struct {
        short e[5];
        s1 f;
} s2;

int examine(s1 one, char two, s2 *three)
{
        char vl1;
        int vl2;
}
```

Lucas Bazilio - Udemy

Exam 1 - Problem 1

e) **Translate** the following statement to x86 assembler, assuming it's inside the examine function:

```
if (vl2 > 0)  { vl2 = three->f.a; }
```

```
cmpl  $0,-4(%ebp)
jle   end             # jump if vl2 <= 0
movl  28(%ebp),%ecx    # %ecx = &three
movl  12(%ecx),%ecx    # %ecx = three->f.a
movl  %ecx,-4(%ebp)   # vl2 = three->f.a
 end:
```

```
typedef struct {
        int a ;
        char b;
        char c;
        double d;
} s1;

typedef struct {
        short e[5];
        s1 f;
} s2;

int examine(s1 one, char two, s2 *three)
{
        char vl1;
        int vl2;
}
```