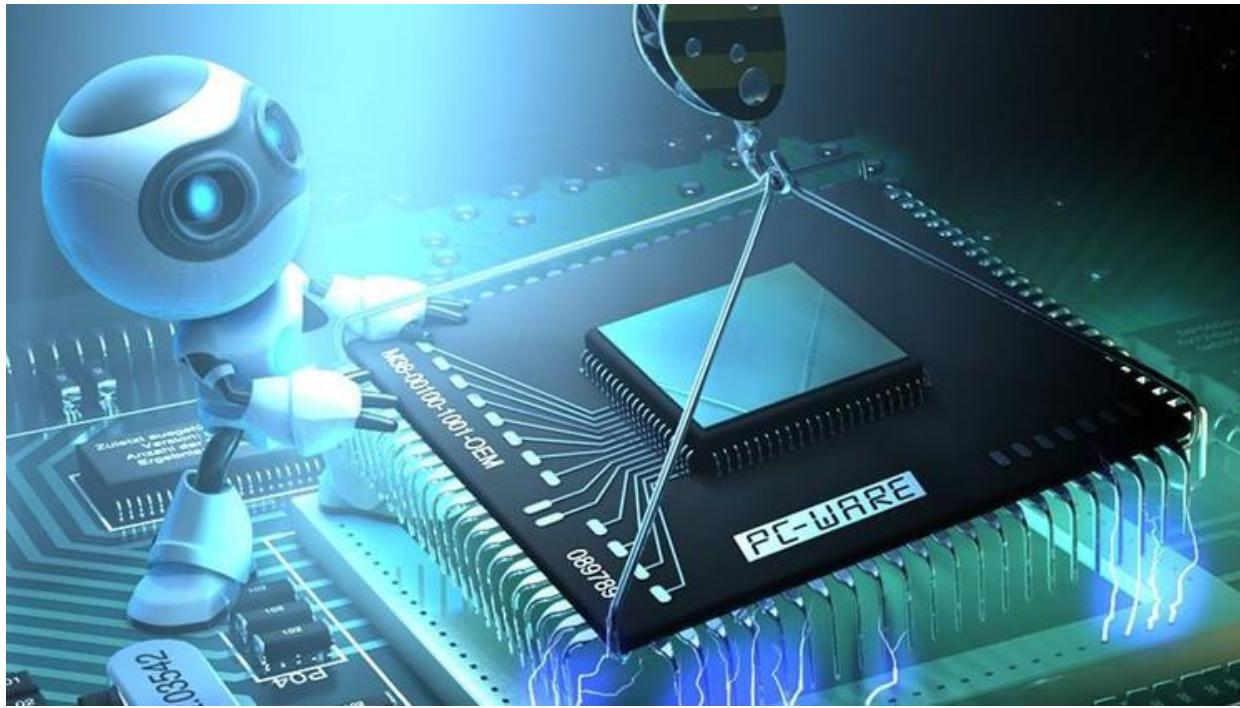
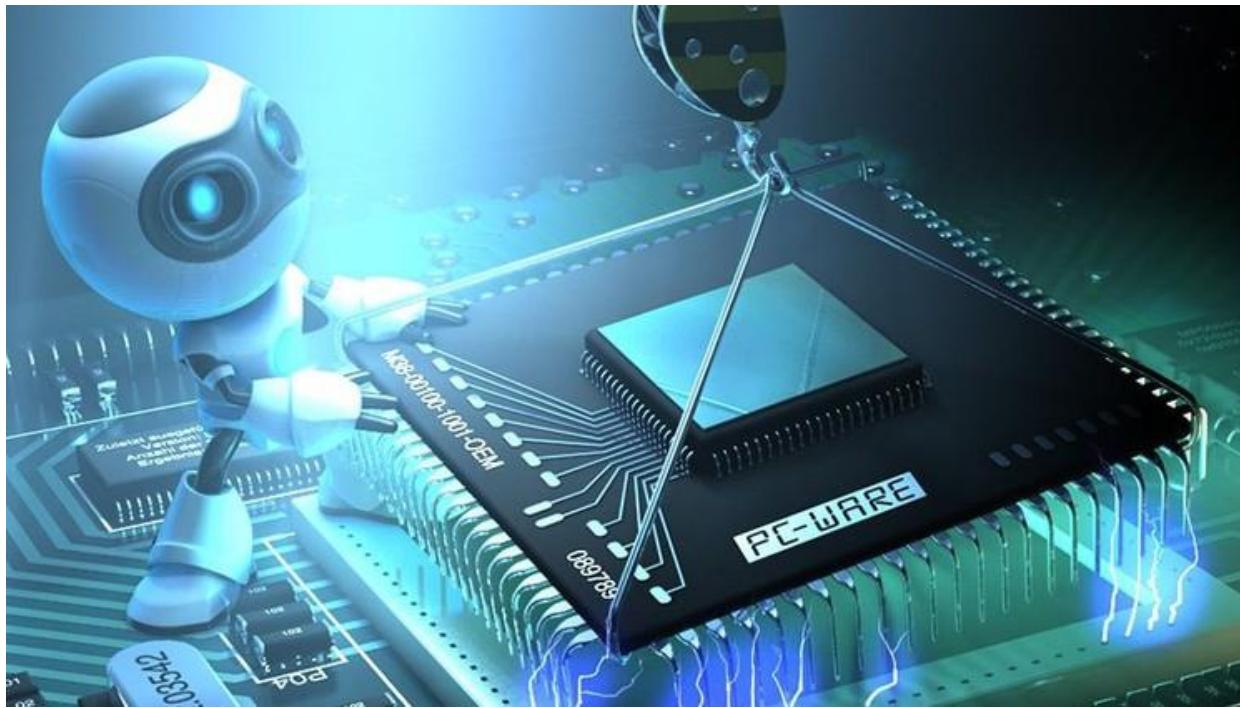


# Laboratory Session 1



# Practise - Problem 2



# Practise - Problem 2



2. Translate this subroutine that has the following high-level code:

```
#define N 3
int OperationMat(int Matrix[N][N], int jump) {
    // The @ of Matrix is in @ 8[ebp] and the
    // value of the variable jump in @ 12[ebp]
    int j;    // j is in @ -12[ebp]
    int i;    // i is in @ -8[ebp]
    int res; // res is in @ -4[ebp]
    res=0;
    for (i=0; i <3; i+=jump) {
        for (j=0; j <3; j++) {
            res -= Matrix[i][i]+j; }
    }
    return res;
}
```



# Practise - Problem 2



2.

Part 1/3

```
.text  
.align 4  
.globl OperationMat  
.type OperationMat, @function
```

OperationMat:

```
    pushl    %ebp  
    movl %esp, %ebp  
    subl $12, %esp  
    pushl    %ebx  
    pushl    %esi  
    pushl    %edi
```

```
    movl $0, -4(%ebp)      # res = 0  
    movl $0, -8(%ebp)      # i = 0
```

A blue square containing the white text "x86", representing the x86 processor architecture.

2.

# Practise - Problem 2



Part 2/3

```
for1: cmpl $3, -8(%ebp)          # comp 3, i  
      jge endfor1
```

```
      movl $0, -12(%ebp)          # j = 0
```

```
for2: cmpl $3, -12(%ebp)          # comp 3, j  
      jge endfor2
```

```
# @Matrix + (i * 3 + i) * 4 = @Matrix + (i * 16)  
      movl -8(%ebp), %eax          # %eax = i  
      movl 8(%ebp, %eax, 16), %eax    #eax = Matrix[i][i]  
      addl -12(%ebp), %eax          # %eax = Matrix[i][i] + j  
      subl %eax, -4(%ebp)           # res -= %eax
```

```
# increment for2  
      incl -12(%ebp)
```



2.

# Practise - Problem 2



Part 3/3

endfor2:

```
# increment for1  
movl 12(%ebp), %edx    # %edx ← jump  
addl %edx, -8(%ebp)    # i += jump
```

endfor1:

```
movl -4(%ebp), %eax    # %eax ← res  
popl %edi  
popl %esi  
popl %ebx  
movl %ebp,%esp  
popl %ebp  
ret
```

