**PRACTICAL # 09**

**OBJECT:**

Instructions pipeline to improve performance

**THEORY:**

Most of the instructions in program code execute in multiple clock cycles. Different basic instructions take different number of clock cycles to execute. For instance, a multiplication operation requires more number of cycles to execute than addition operation.

In this case, instruction pipeline can optimize instructions execution time.

Example operations:

*int i=0, j=0;*

*i++;*

*i++;*

*i++;*

*j++;*

*j++;*

*j++;*

Suppose three increments would not be optimized away by the compiler into one +=3. Thus the instructions above are dependent on the results of previous instruction. But, you can end up having a higher processor-pipeline throughput if you reordered the operations as:

*i++;*

*j++;*

*i++;*

*j++;*

*i++;*

*j++;*

Since j++ doesn't have to wait for the result of i++ while in the previous case, most of the instructions had a data dependency on the previous instruction. In more complicated computations, where there isn't an easy way to reducing the number of instructions to be performed, the compiler can still look at data dependencies and reorder instructions so that an instruction depending on the result of an earlier instruction is as far away from it as possible.

**REVIEW QUESTIONS**

1. What is instruction pipeline?

1. Why is it important to understand the dependencies between instructions?
2. How can you improve pipeline throughput?
3. What happens if you rearrange the dependent instructions incorrectly?