PIPELINING

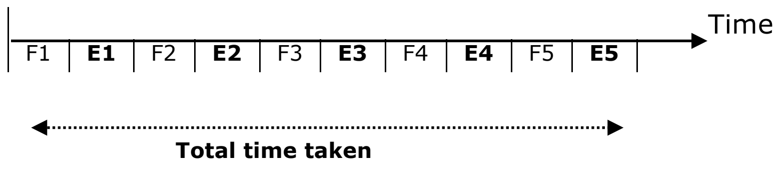
Overlapping different stages of an instruction is called pipelining.

An instruction requires several steps which mainly involve fetching, decoding and execution.
If these steps are performed one after the other, they will take a long time.
As processors became faster, several of these steps started to get overlapped, resulting in faster processing. This is done by a mechanism called pipelining.

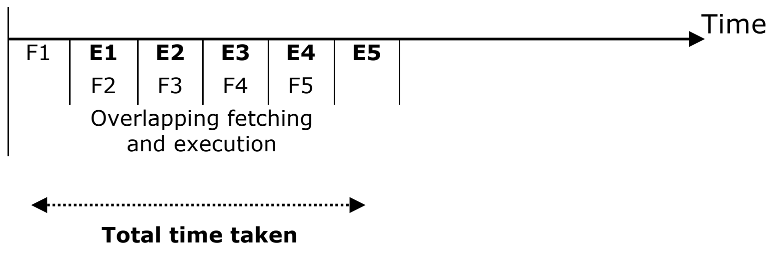
2 STAGE PIPELINING - 8086

Here the instruction process in divided into two stages of fetching and execution.
Fetching of the next instruction takes place while the current instruction is being executed. Hence two instructions are being processed at any point of time.

NON-PIPELINED PROCESSOR EG: 8085

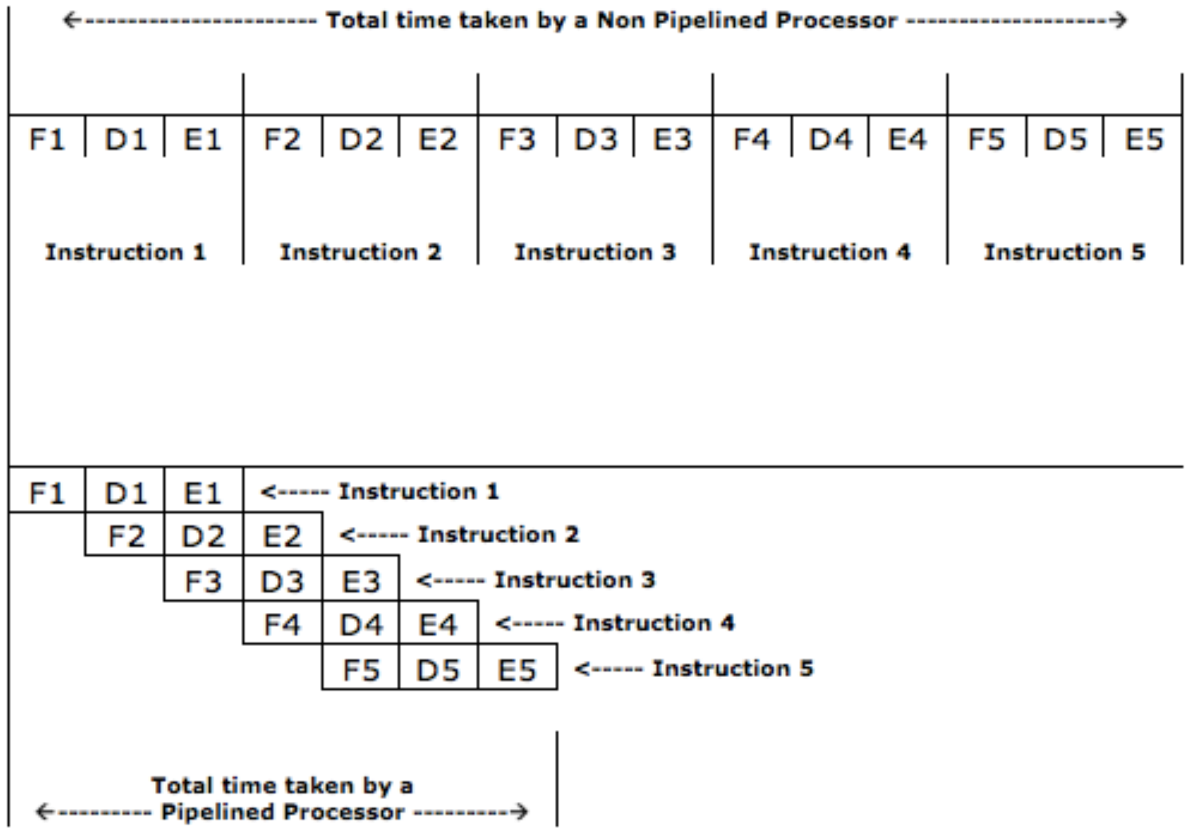


PIPELINED PROCESSOR EG: 8086



3 STAGE PIPELINING – 80386/ ARM 7

Here the instruction process in divided into three stages of **fetching, decoding and execution** and are overlapped. Hence **three instructions** are being processed at any point of time.


Total time taken by a Non Pipelined Processor
F1 D1 E1 F2 D2 E2 F3 D3 E3 F4 D4 E4 F5 D5 E5
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
F1 D1 E1 <----- Instruction 1
F2 D2 E2 <----- Instruction 2
F3 D3 E3 <----- Instruction 3
F4 D4 E4 <----- Instruction 4
F5 D5 E5 <----- Instruction 5
Total time taken by a Pipelined Processor

4 STAGE PIPELINING

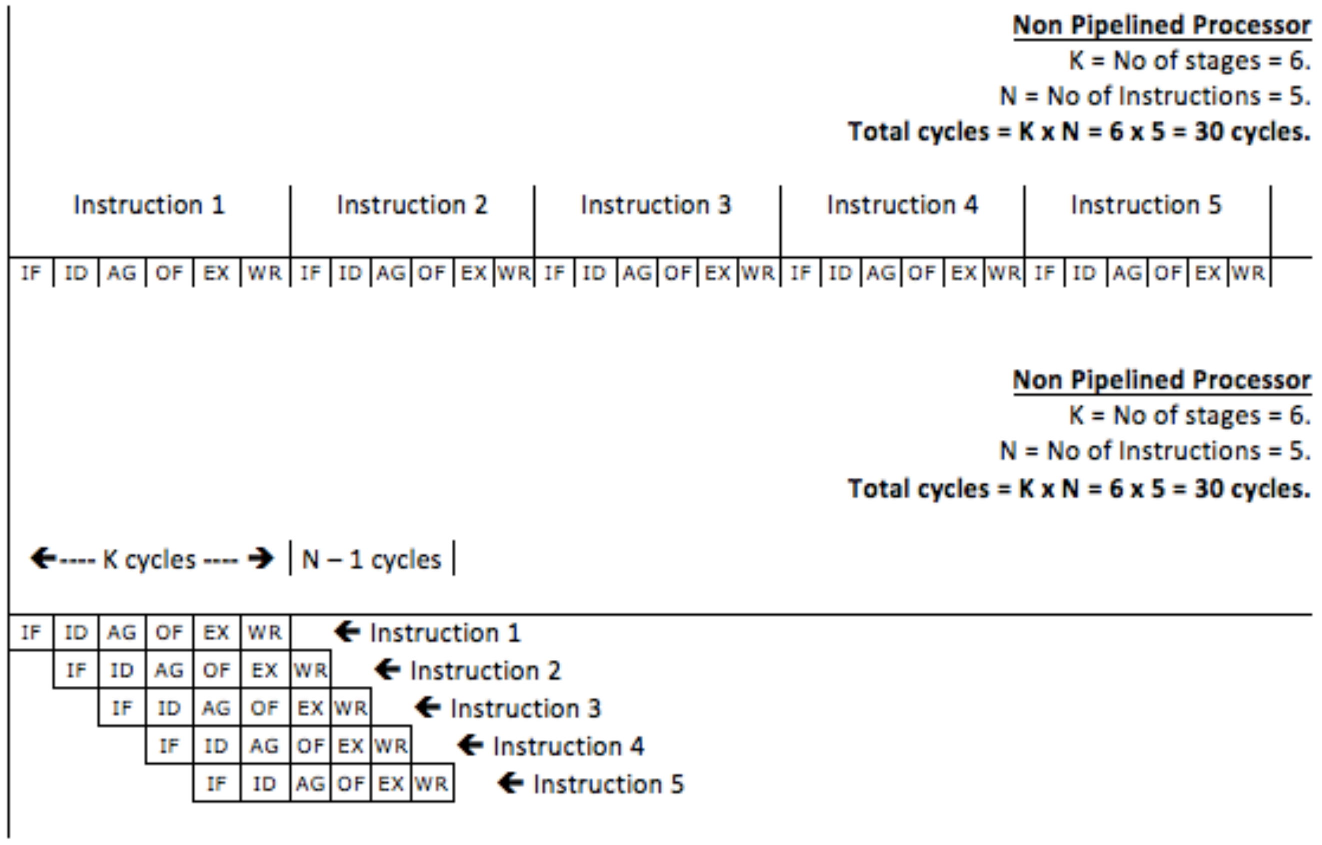
Fetch, Decode, Execute, Store

5 STAGE PIPELINING - PENTIUM

Fetch, Decode, Address Generation, Execute, Store

6 STAGE PIPELINING – PENTIUM PRO

Instruction Fetch (IF):	Fetch the instruction
Instruction Decode (ID):	Decode the instruction.
Address Generation (AG):	Calculate address of Memory operand
Operand Fetch (OF):	Fetch memory operand
Execute (EX):	Execute the operation
Write Back (WR):	Write back/ Store the result



Modern processors have a very deep pipeline.
Ex: Pentium 4 (Net burst Architecture) has a 20 stage pipeline.

ADVANTAGE OF PIPELINING

The obvious advantage of pipelining is that it increases the performance. As shown by the various examples above, deeper the pipelining, more is the level of parallelism, and hence the processor becomes much faster.

DRAWBACKS/ HAZARDS OF PIPELINING

There are various hazards of pipelining, which **cause a dip** in the performance of the processor. These hazards become even **more prominent** as the **number of pipeline stages increase**. They may occur due to the following reasons.

1) DATA HAZARD/ DATA DEPENDENCY HAZARD

Data Hazard is caused when **the result (destination) of one instruction becomes the operand (source) of the next instruction**.

Consider two instructions I1 and I2 (I1 being the first).

Assume I1: INC [4000H]
Assume I2: MOV BL , [4000H]

Clearly in I2, BL should get the incremented value of location [4000H].
But this can only happen once I1 has completely finished execution and also written back the result at [4000H].
In a multistage pipeline, I2 may reach execution stage before I1 has finished storing the result at location [4000H], and hence get a wrong value of data.
This is called **data dependency hazard**.
It is solved by inserting NOP (No operation) instructions between such data dependent instructions.

2) CONTROL HAZARD/ CODE HAZARD

Pipelining assumes that the program will always flow in a sequential manner.
Hence, it performs various stages of the forthcoming instructions before-hand, while the current instruction is still being executed.
While programs are sequential most of the times, it is not true always.
Sometimes, branches do occur in programs.
In such an event, all the forthcoming instructions that have been fetched/ decoded etc have to be flushed/ discarded, and the process has to start all over again, from the branch address. This causes pipeline bubbles, which simply means time of the processor is wasted.

Consider the following set of instructions:

```
Start:
        JMP Down
        INC BL
        MOV CL, DL
        ADD AL, BL
        …
        …
        …

Down:   DEC CH
```

JMP Down is a branch instruction.
After this instruction, program should jump to the location "Down" and continue with DEC CH instruction.

But, in a multistage pipeline processor, the sequentially next instructions after JMP Down have already been fetched and decoded. These instructions will now have to be discarded and fetching will begin all over again from DEC CH. This will keep several units of the architecture idle for some time. This is called a pipeline bubble.

The **problem of branching is solved** in higher processors by a method called "**Branch Prediction Algorithm**". It was introduced by **Pentium** processor. It relies on the **previous history** of the instruction as most programs are repetitive in nature. It then **makes a prediction** whether branch will be **taken or not** and hence puts the correct instructions in the pipelines.

3) STRUCTURAL HAZARD

Structural hazards are caused by **physical constraints in the architecture like the buses**. Even in the most basic form of pipelining, we want to execute one instruction and fetch the next one. Now as long as execution only involves registers, pipelining is possible. But **if execution requires to read/ write data from the memory, then it will make use of the buses, which means fetching cannot take place at the same time**. So the fetching unit will have to wait and hence a pipeline bubble is caused. This problem is solved in complex Harvard architecture processors, which use separate memories and separate buses for programs and data. This means fetching and execution can actually happen at the same time without any interference with each other.
E.g.: PIC 18 Microcontroller.