**Pipelining**

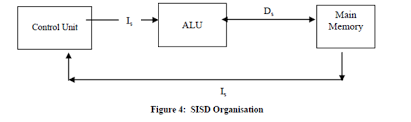
It is the process of dividing sequencial process and parallelizing it in a processor.

**Parallel Processing:**

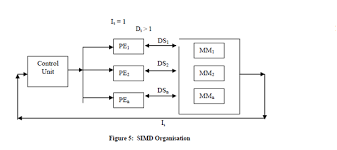
Flyn’s classification of parallel processing:

1. Single instruction stream single data stream
2. Single instruction stream multiple data stream
3. Multiple instruction stream single data stream
4. Multiple instruction stream muliple data stream

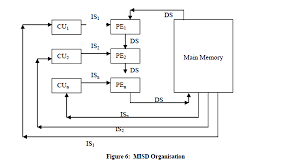
Single instruction stream single data stream



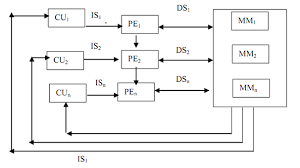
Single instruction stream multiple data stream



Multiple instruction stream single data stream



Multiple instruction stream muliple data stream



**Speed-up Eqn:**

Lets suppose there are “n” operations each operation takes “tn” time to complete. For n operations to complete:

total time = n \* tn

Consider “K” segment pipeline. Each segment takes tp time to operate. For 1st operation to complete it will take k\*tp time to complete. For remaining n-1 tasks, it will take

(n-1)\*tp time.

So total time = k\*tp + (n-1)\*tp

= (k+n-1)\*tp

Speed-up = time for non-pipeline/time for pipelined system

= n\*tn/(k+n-1)tp

**Arithmetic Pipeline:**

Floating point addition and subtraction:

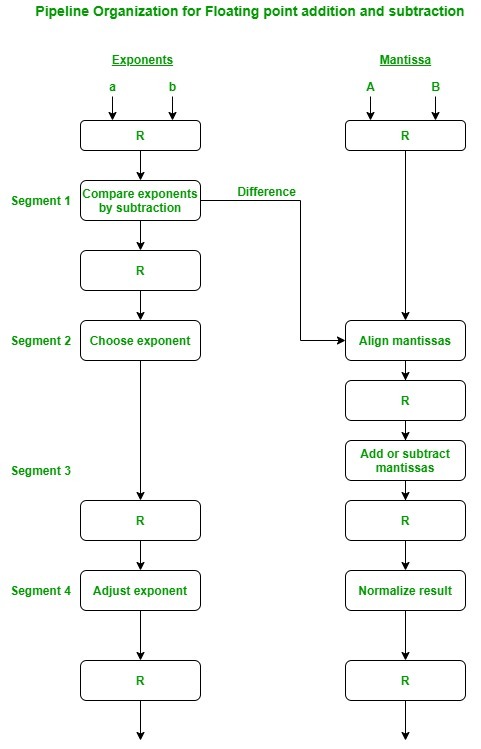
x = 2.5 x107

+ y = 3.36x106

2.836x107

Steps:

* 1. Compare exponent
  2. Align mantissa (choosing larger exponent)
  3. Add / Subtract mantissa
  4. Normalize mantissa



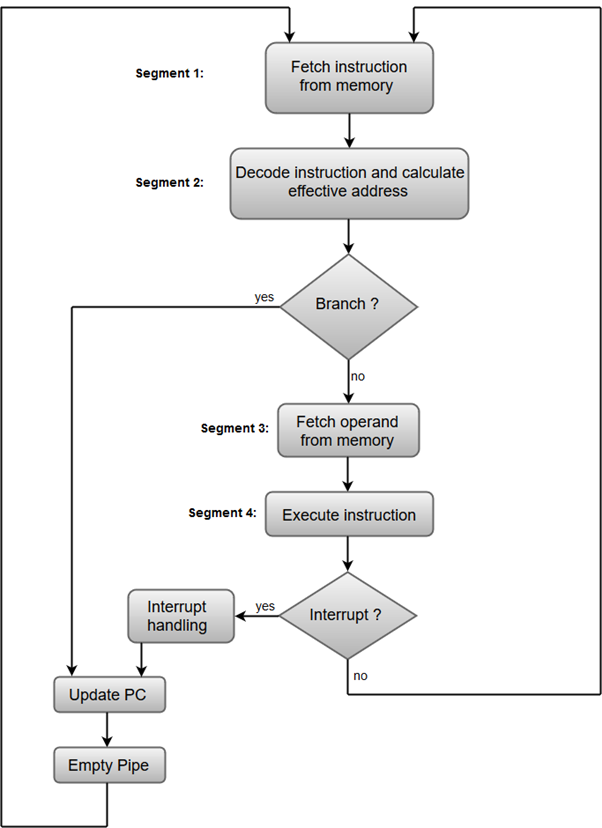
**Instruction Pipeline:**

Steps:

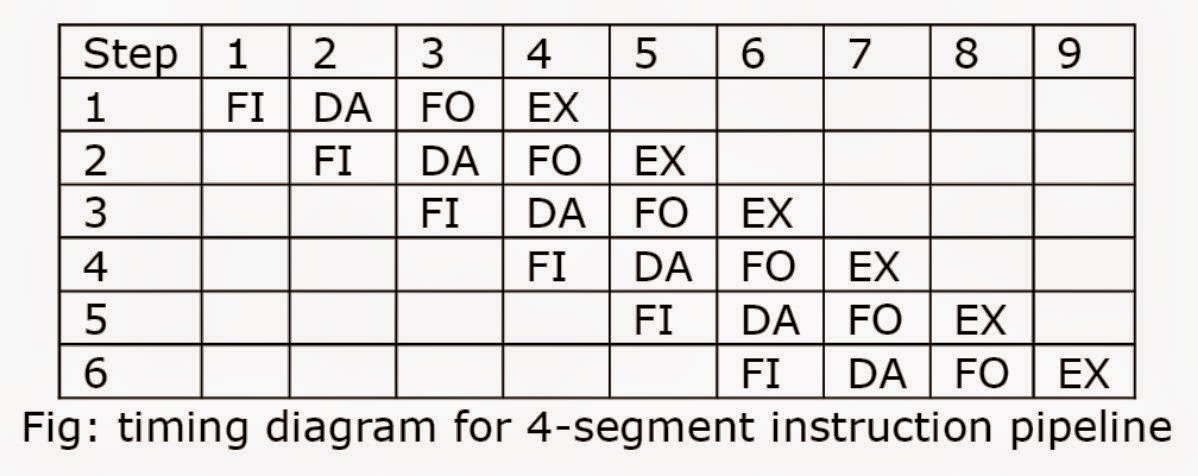
* + 1. Fetch instruction (opcode) from memory
    2. Decode instruction
    3. Calculate effective address of operand
    4. Fetch operand from memory
    5. Execute instruction
    6. Store result

Pipelined steps:

* + 1. Fetch instruction (opcode) from memory (FI)
    2. Decode instruction and Calculate effective address of operand (DA)
    3. Fetch operand from memory (FO)
    4. Execute instruction and Store result (EX)



Space Time diagram:



**Pipelining Conflicts:**

When two or more segments cannot run simultaneously , pipeline cannot function properly. This is called pipeline conflict or hazard. The types of hazards are:

1. Resource conflict (Structural hazard)
2. Data dependency conflict
3. Branch conflict

Resource conflict → When two or more segments access resources such as memory at the same time this conflict occurs.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| clock | 1 | 2 | 3 | 4 | 5 | 6 |
| Inst 1 | FI | DA | FO | EX |  |  |
| Inst 2 |  | FI | DA | FO | EX |  |
| Inst 3 |  |  | FI | DA | FO | EX |

In clock 3, 2 simultanious fetch operations are occuring which creates resource conflict.

Solution: We can use seperate memory for instruction and data.

Data dependency conflict → When two or more segments cannot run simultaniously . Pipelinr cannot function properly.

Ex:

R3 ← R1 + R2

R4 ← R3 + R1

Here, second instruction has to wait for first instruction to finish which is data dependency conflit.

Branch conflict →

Ex:

Inst 1

Inst 2

Jump Target

Target Inst

Inst 3

inst 4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| clock | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Inst 1 | FI | DA | FO | EX |  |  |  |
| Inst 2 |  | FI | DA | FO | EX |  |  |
| Inst 3 |  |  | FI | DA | FO | EX |  |
| Inst 4 |  |  |  | FI | - | - | FI |

In clock 5 and 6 the 4th instruction couldnot be run paralellly due to branching in instruction 3. This is branching conflict.

Solution:

* Hardware Solution
  + Hardware Interlocking → hardware detects data dependency introducing delay in pipeline to solve data dependency.
  + Operand forwarding → we forward operand to next instruction by changing the data path of operand to next instruction.
* Software Solution
  + Delayed load → compiler detects data dependency in complile time. If there is data dependency, it introduces delay by inserting NOPE instruction.

**Branch difficulties:**

Solutions:

1. Hardware solution
   1. Prefetch target instruction → both next instruction and target instruction are fetched
   2. Branch target buffer → small memory inside fetch segment at pipeline, target instruction and some other instructions previously run are stored in this memory
   3. Loop buffer → just like branch target buffer but for loops
   4. Branch prediction → a digital circuit that tries to guess which way a branch (e.g., an if–then–else structure) will go before this is known definitively
2. Software solution
   1. Delayed branch → compiler introduces NOP instruction or rearranges instructions

Before delayed branch

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| clock | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Inst 1 | FI | DA | EX |  |  |  |  |
| Inst 2 |  | FI | DA | EX |  |  |  |
| Branch |  |  | FI | DA | EX |  |  |
| Inst 3 |  |  |  | FI | - | - |  |
| Traget inst |  |  |  |  |  | FI | DA |
|  |  |  |  |  |  |  | FI |

After delayed branch

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| clock | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Inst 1 | FI | DA | EX |  |  |  |  |
| Inst 2 |  | FI | DA | EX |  |  |  |
| Branch |  |  | FI | DA | EX |  |  |
| NOP |  |  |  | FI | DA | EX |  |
| Traget inst |  |  |  |  | FI | DA | EX |
| Inst - |  |  |  |  |  | FI | DA |

*H/W:* short notes on vector processing and array proceessing