Computer organization and architecture
module 4

By
Soumya Das
Asst prof. Dept. of CSE
GCE kalahandi

Pipelining

- **Pipelining :** Pipelining is a process of arrangement of hardware elements of the CPU such that its overall performance is increased. Simultaneous execution of more than one instruction takes place in a pipelined processor.

Stages of pipelining

Pipeline Stages

RISC processor has 5 stage instruction pipeline to execute all the instructions in the RISC instruction set. Following are the 5 stages of RISC pipeline with their respective operations:

Stage 1 (Instruction Fetch)
In this stage the CPU reads instructions from the address in the memory whose value is present in the program counter.

Stage 2 (Instruction Decode)
In this stage, instruction is decoded and the register file is accessed to get the values from the registers used in the instruction.

Stage 3 (Instruction Execute)
In this stage, ALU operations are performed.

Stage 4 (Memory Access)
In this stage, memory operands are read and written from/to the memory that is present in the instruction.

Stage 5 (Write Back)
In this stage, computed/fetched value is written back to the register present in the instructions.

Performance

Performance of a pipelined processor

Consider a 'k' segment pipeline with clock cycle time as 'Tp'. Let there be 'n' tasks to be completed in the pipelined processor. Now, the first instruction is going to take 'k' cycles to come out of the pipeline but the other 'n – 1' instructions will take only '1' cycle each, i.e, a total of 'n – 1' cycles. So, time taken to execute 'n' instructions in a pipelined processor:

$ET_{pipeline}$ = k + n – 1 cycles = (k + n – 1) Tp In the same case, for a non-pipelined processor, execution time of 'n' instructions will be:

$ET_{non\text{-}pipeline}$ = n * k * Tp So, speedup (S) of the pipelined processor over non-pipelined processor, when 'n' tasks are executed on the same processor is:

S = Performance of pipelined processor / Performance of Non-pipelined processor As the performance of a processor is inversely proportional to the execution time, we have,

S = $ET_{non\text{-}pipeline}$ / $ET_{pipeline}$ => S = [n * k * Tp] / [(k + n – 1) * Tp] S = [n * k] / [k + n – 1] When the number of tasks 'n' are significantly larger than k, that is, n >> k

S = n * k / n S = k where 'k' are the number of stages in the pipeline.

Also, **Efficiency** = Given speed up / Max speed up = S / S_{max}

We know that, Smax = k

So, **Efficiency** = S / k

Throughput = Number of instructions / Total time to complete the instructions

So, **Throughput** = n / (k + n – 1) * Tp

Note: The cycles per instruction (CPI) value of an ideal pipelined processor is 1.

Pipeline hazards

Dependencies in a pipelined processor

There are mainly three types of dependencies possible in a pipelined processor. These are :

1) Structural Dependency
2) Control Dependency
3) Data Dependency

These dependencies may introduce stalls in the pipeline.

Pipelining hazards contd.

- **Stall :** A stall is a cycle in the pipeline without new input.

  Structural dependency
- This dependency arises due to the resource conflict in the pipeline. A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle. A resource can be a register, memory, or ALU.
- Example:
  Solution for structural dependency
  To minimize structural dependency stalls in the pipeline, we use a hardware mechanism called Renaming.
  Renaming : According to renaming, we divide the memory into two independent modules used to store the instruction and data separately called Code memory(CM) and Data memory(DM) respectively. CM will contain all the instructions and DM will contain all the operands that are required for the instructions.

Pipelining hazrads contd.

Operand Forwarding : In operand forwarding, we use the interface registers present between the stages to hold intermediate output so that dependent instruction can access new value from the interface register directly.

Considering the same example:

I1 : ADD R1, R2, R3
I2 : SUB R4, R1, R2

INSTRUCTION / CYCLE1234I_1IFIDEXDMI_2IFIDEX

Data Hazards

Data hazards occur when instructions that exhibit data dependence, modify data in different stages of a pipeline. Hazard cause delays in the pipeline. There are mainly three types of data hazards:

1) RAW (Read after Write) [Flow/True data dependency]
2) WAR (Write after Read) [Anti-Data dependency]
3) WAW (Write after Write) [Output data dependency]

Let there be two instructions I and J, such that J follow I. Then,

RAW hazard occurs when instruction J tries to read data before instruction I writes it.

Eg:
I: R2 <- R1 + R3
J: R4 <- R2 + R3

WAR hazard occurs when instruction J tries to write data before instruction I reads it.

Eg:
I: R2 <- R1 + R3
J: R3 <- R4 + R5

WAW hazard occurs when instruction J tries to write output before instruction I writes it.

Eg:
I: R2 <- R1 + R3
J: R2 <- R4 + R5

WAR and WAW hazards occur during the out-of-order execution of the instructions.

- **Types of pipeline**
- Uniform delay pipeline
  In this type of pipeline, all the stages will take same time to complete an operation.
  In uniform delay pipeline, **Cycle Time (Tp) = Stage Delay**If buffers are included between the stages then, **Cycle Time (Tp) = Stage Delay + Buffer Delay**

Pipelining contd.

Non-Uniform delay pipeline

In this type of pipeline, different stages take different time to complete an operation.

In this type of pipeline, Cycle Time (Tp) = Maximum(Stage Delay)For example, if there are 4 stages with delays, 1 ns, 2 ns, 3 ns, and 4 ns, then

Tp = Maximum(1 ns, 2 ns, 3 ns, 4 ns) = 4 ns

If buffers are included between the stages,

Tp = Maximum(Stage delay + Buffer delay)

Example : Consider a 4 segment pipeline with stage delays (2 ns, 8 ns, 3 ns, 10 ns). Find the time taken to execute 100 tasks in the above pipeline.

Solution : As the above pipeline is a non-linear pipeline,

Tp = max(2, 8, 3, 10) = 10 ns

We know that $ET_{pipeline}$ = (k + n – 1) Tp = (4 + 100 – 1) 10 ns = 1030 ns

NOTE: MIPS = Million instructions per second

Pipelining contd.

- **Performance of pipeline with stalls**
- Speed Up (S) = Performance$_{pipeline}$ / Performance$_{non\text{-}pipeline}$ => S = Average Execution Time$_{non\text{-}pipeline}$ / Average Execution Time$_{pipeline}$ => S = CPI$_{non\text{-}pipeline}$ * Cycle Time$_{non\text{-}pipeline}$ / CPI$_{pipeline}$ * Cycle Time$_{pipeline}$Ideal CPI of the pipelined processor is ‘1’. But due to stalls, it becomes greater than ‘1’.
  =>
- S = CPI$_{non\text{-}pipeline}$ * Cycle Time$_{non\text{-}pipeline}$ / (1 + Number of stalls per Instruction) * Cycle Time$_{pipeline}$ As Cycle Time$_{non\text{-}pipeline}$ = Cycle Time$_{pipeline}$,
- **Speed Up (S) = CPI$_{non\text{-}pipeline}$ / (1 + Number of stalls per instruction)**

Parallel computing

Why parallel computing?

The whole real world runs in dynamic nature i.e. many things happen at a certain time but at different places concurrently. This data is extensively huge to manage.

Real world data needs more dynamic simulation and modeling, and for achieving the same, parallel computing is the key.

Parallel computing provides concurrency and saves time and money.

Complex, large datasets, and their management can be organized only and only using parallel computing's approach.

Ensures the effective utilization of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of hardware was used and the rest rendered idle.

Also, it is impractical to implement real-time systems using serial computing.

Applications of Parallel Computing:

Data bases and Data mining.

Real time simulation of systems.

Science and Engineering.

Advanced graphics, augmented reality and virtual reality.

Limitations of Parallel Computing:

It addresses such as communication and synchronization between multiple sub-tasks and processes which is difficult to achieve.

The algorithms must be managed in such a way that they can be handled in the parallel mechanism.

The algorithms or program must have low coupling and high cohesion. But it's difficult to create such programs.

More technically skilled and expert programmers can code a parallelism based program well.

Future of Parallel Computing: The computational graph has undergone a great transition from serial computing to parallel computing. Tech giant such as Intel has already taken a step towards parallel computing by employing multicore processors. Parallel computation will revolutionize the way computers work in the future, for the better good. With all the world connecting to each other even more than before, Parallel Computing does a better role in helping us stay that way. With faster networks, distributed systems, and multi-processor computers, it becomes even more necessary.

Parallel computing contd.

Parallel Computing –
It is the use of multiple processing elements simultaneously for solving any problem. Problems are broken down into instructions and are solved concurrently as each resource which has been applied to work is working at the same time.

Advantages of Parallel Computing over Serial Computing are as follows:

It saves time and money as many resources working together will reduce the time and cut potential costs.

It can be impractical to solve larger problems on Serial Computing.

It can take advantage of non-local resources when the local resources are finite.

Serial Computing 'wastes' the potential computing power, thus Parallel Computing makes better work of hardware.

Types of Parallelism:

Bit-level parallelism: It is the form of parallel computing which is based on the increasing processor's size. It reduces the number of instructions that the system must execute in order to perform a task on large-sized data.
Example: Consider a scenario where an 8-bit processor must compute the sum of two 16-bit integers. It must first sum up the 8 lower-order bits, then add the 8 higher-order bits, thus requiring two instructions to perform the operation. A 16-bit processor can perform the operation with just one instruction.

Instruction-level parallelism: A processor can only address less than one instruction for each clock cycle phase. These instructions can be re-ordered and grouped which are later on executed concurrently without affecting the result of the program. This is called instruction-level parallelism.

Task Parallelism: Task parallelism employs the decomposition of a task into subtasks and then allocating each of the subtasks for execution. The processors perform execution of sub tasks concurrently.