Pipelining in computer architecture

Team code: 7033

2019

0.1 DESCRIPTION OF THE TOPIC:

0.1.1 Introduction

Most current computer architectures are based on a sequential model of program execution in which an architectural program counter sequences through instructions one-by-one. In contrast, a high performance implementation may be pipelined,"Pipelining" is the process of accumulating , storing and executing instructions in an orderly process from the processor where multiple instructions are overlapped during execution through a pipeline.

0.1.2 History of pipelining

The first general-purpose pipelined machine is considered to be Stretch, the IBM 7030 (1959). Stretch followed the IBM 704 and had a goal of being 100 times faster than the 704. The goal was a stretch from the state of the art that time - hence the nickname. The plan was to obtain a factor of 1.6 from overlapping fetch, code and execute, using a four-stage pipeline. The CDC 6600, developed in the early 1960s, also introduced several enhancements in pipelining. A series of general pipelining descriptions that appeared in the late 1970s and early 1980s provided most of the terminology and described most of the basic techniques used in simple pipelines. The RISC machines were originally designed with ease of implementation and pipelining in mind. Several of the early RISC papers, published in the early 1980s, attempt to quantify the performance advantages of the simplification in instruction set. The best analysis, however, is a comparison of a VAX and a MIPS implementation published by Bhandarkar and Clark in 1991, 10 years after the first published RISC papers. After 17 years of arguments about the implementation benefits of RISC, this paper convinced even the most skeptical designers of the advantages of a RISC instruction set architecture. The RISC machines refined the notion of compiler. The concepts of delayed branches and delayed loads were extended into the high-level architecture. The Stanford MIPS architecture made the pipeline structure purposely visible to the compiler and allowed multiple operations per instruction. Simple schemes for scheduling the pipeline in the compiler were described by Sites (1979), by Hennessy and Gross (1983) and by Gibbons and Muchnik (1969). Rymarczyk (1982) in his paper showed the complex interaction between pipelining and an instruction set not designed to be pipelined. Static branch prediction by profiling

has been explored by McFarling and Hennessy (1986) and by Fisher Frendenberger (1992). J.E. Smith and his colleagues have written a number of papers examining instruction issue, exception, handling and pipeline depth for high-speed scalar machines. Kunkel and Smith (1986) evaluated the impact of pipeline overhead and dependences on the choice of optimal pipeline depth. Smith and Pleszkun (1988) evaluated a variety of techniques for preserving precise exceptions. Weiss and Smith (1984) evaluared a variety of hardware pipeline scheduling and instruction-issue techniques.

0.2 IMPORTANCE OF PIPELINING

Pipelining plays an important role in making CPU fast, pipeline stages increases the number of instructions executed simultaneously and therefore increases system's throughput.However this pipelining is complex and the most interesting part lies with detecting and avoiding the different types of hazards like Structural Hazards,Control Hazards,Data hazards.

0.2.1 Advantages and Disadvantages of Pipelining

0.2.2 Advantages

- Instruction throughput increases.
- Increase in the number of pipeline stages increases the number of instructions executed simultaneously.
- Faster ALU can be designed when pipelining is used.
- Pipelined CPU's works at higher clock frequencies than the RAM.
- Pipelining increases the overall performance of the CPU.

0.2.3 Disadvantages

- Designing of the pipelined processor is complex.
- Instruction latency increases in pipelined processors.

- The throughput of a pipelined processor is difficult to predict.
- The longer the pipeline, worse the problem of hazard for branch instructions.

0.2.4 Pipeline Hazards and Solutions

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.

0.2.5 Structural Hazards

When hardware units are being used by instructions already in the pipeline, these units will not be available for use by other instructions. Any situation where there is not enough hardware is a structural hazard. A structural hazard would for example result from memory access of instruction fetch and memory access of data, were it not for separate data and instruction caches. There are two main reasons why a designer would allow structural hazards. The first one is to reduce the cost. For example, machines that support both an instruction and a cache access at least twice as much total memory. The second one is to reduce the latency of the unit. The shorter latency comes from the lack of pipeline registers that introduce overhead. A good way of avoiding structural hazards is using the Harvard architecture, instead of the von Neumann architecture. In the von Neumann architecture, program and data are stored in the same memory and managed by the same information – handling subsystem. In the Harvard architecture, program and data are stored and handled by different subsystems. In this way, instruction fetching and memory accessing of data are happening in different units without creating stalls.

0.2.6 Data hazards

They occur when instructions that exhibit data dependence modify data in different stages of a pipeline. There are three situations in which a data

hazard can occur:
1. Read after write (RAW), a true dependency
2. Write after read (WAR), an anti – dependency
3. Write after write (WAW), an output dependency

1.READ AFTER WRITE (RAW)

A read after write hazard refers to a situation where an instruction refers to a result that has not yet been calculated or retrieved. This can occur because even though an instruction is executed after a previous instruction, the previous one has not been completely processes through the pipeline.

2. WRITE AFTER READ (WAR)

A write after read data hazard represent a problem with concurrent execution.

3. WRITE AFTER WRITE (WAW)

A write after write data hazard may occur in a concurrent execution environment.
There are three ways of handling data hazards. From the software aspect, we can insert independent instructions (or no operation instructions). From the hardware aspect, we can insert bubbles, so that we can stall the pipeline or we can forward the data.

0.2.7 Control hazards

In the ideal pipeline, we fetch instructions one after another in order. As long as location of the next instruction is known, this process can go forward. When a branch instruction is fetched, the next instruction location is not known until the branch instruction finishes execution. Thus, we may have to wait until the correct location of the next instruction is known before fetching more instructions. This is a control hazard. There are two ways in order to avoid control hazards. Either insert a pipeline bubble, guaranteed to increase latency or use branch prediction and essentially make educated guesses about which instructions to insert, in which case a pipeline bubble will only be needed in the case of an incorrect prediction. In the event that

a branch causes a pipeline bubble after incorrect instructions have entered the pipeline, care must be taken to prevent any of the wrongly – loaded instructions from having any effect on the processor state excluding energy wasted processing them before they were discovered to be loaded incorrectly.

0.3 PREVIOUS RESEARCH

MIPS pipelining implementation codes were written in C, C++ and many other programming languages and there have been a lot of papers on this subject. Data dependencies have been handled in many different and interesting ways. However, it is always astonishing studying and observing how pipelining can change a machine's performance, and at the same time how hazards can destroy this "magic". Few reference papers references in this aspect are:

- https://ieeexplore.ieee.org/document/6578243
- https://www.elprocus.com/pipelining-architecture-hazards-advantages-disadvantages/
- https://ieeexplore.ieee.org/abstract/document/5403912

0.4 PROJECTS WITH ORIGINAL RESEARCH

- https://www.researchgate.net/profile/Norman_Jouppi/publication/234795328_MIPS_A_micr
  A-microprocessor-architecture.pdf
- https://ieeexplore.ieee.org/abstract/document/5403912