Bryan Melanson

How to Not Fail

Computer Architecture

While never going to class

Computer Engineering 2020 $\mathcal{BM}$

Contents

1 Introduction

1.1 RISC

RISC (Reduced Instruction Set Computer) is an architecture which allowed for the exploitation of instruction level parallelism and the use of caches. This increased performance couldn't be matched by other architectures, and Intel 80x86 was forced to translate its instructions into RISC-like form internally to adopt RISC innovations. In low-end applications, this has lead to RISC architecture like ARM becoming dominant.

1.2 Dennard Scaling

Power density was a constant for a given area of silicon, even as the number of transistors increased, due to their smalll dimension. This held up until a certain level in 2004, when the current and voltage levels could no longer ensure the dependability of integrated circuits.

1.3 Multiple Processors

The end of Dennard Scaling meant that improve performance more processors would be implemented, since single core chips would be inefficient with the increasing number of transistors. This meant that designs would no longer rely on **Instruction Level Parallelism**.

1.4 Classes of Parallelism

- **Data Level Parallelism** - data items operated on at the same time
- **Task Level Parallelism** - tasks performed independently in parallel

1.5 Application Parallelism

- **Instruction Level Parallelism** - exploits data-level using pipelining and speculative execution
- **Vector Archictectures** - exploits data-level parallelism using single instructions on a collection of data in parallel
- **Thread Level Parallelism** - data or task-level parallelism allowing for interaction between parallel threads
- **Request Level Parallelism** - parallelism between decoupled tasks

1.6 Flynn's Taxonomy

- **Single Instruction Stream, Single Data Stream** - a single processor operates on sequential computation but can exploit ILP
- **Single Instruction Stream, Single Data Stream** - the same instruction executed by multiple processors, uses data level parallelism
- **Multiple Instruction Streams, Single Data Stream** - no implementation to date
- **Multiple Instruction Streams, Multiple Data Stream** - processesors have their own instructions and own data operated on, using task level parallelism. Used in warehouse level computing, exploiting request level parallelism where many independent tasks can proceed in parallel

1.7 Warehouse Scale Computers

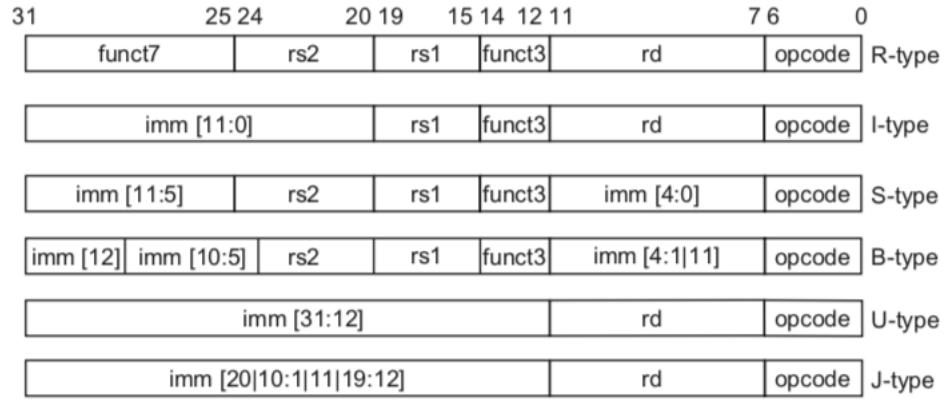
Warehouse Scale Computers **WSC**s are collections of servers or **clusters** which Software as a Service (SaaS) uses as a single larger computer, capable of considerable processing power. Price-performance and power are critical.

1.8 Instruction Set Architecture

The old view of computer architecture focused on **Instruction Set Architecture** (ISA) design involves decisions regarding registers, memory addressing, addressing modes, instruction operands, available operations, control instructions and instruction encoding.

Computer architecture now involves specific requirementgs of the target machine, designs meant to maximize performance within constraints, ISA, microarchitecture, and requires consideration of how compilers work.

1.9 RISC-V Instruction Formats



1.10 Latency and Bandwidth

1.11 Dynamic State Energy and Power

1.12 Die Yield

$$\text{Die Yield} = (1 + (\frac{\text{Defects/Area} \cdot \text{Area of Die}}{N}))^{-N} \tag{1}$$

Die Yield is a value which defines how likely it is that a die will be non-defective during the fabrication process. This value is a probability < 1, and the probability that a die will be defective is $1-$Die Yield.
When converting between units of area (mm^2, cm^2), be sure to repeat conversion between units twice.

$$1 \text{ mm}^2 = \frac{1}{10\text{cm}} \cdot \frac{1}{10\text{cm}} = \frac{1}{100} \text{ cm}^2$$

1.13 Wafer Yield

Wafer Yield is a value which describes how many dies (chips) can be created from a single wafer. This calculation is done using the size of a chip, and the diameter of each wafer. It takes into effect the square or rectangle shape of a chip and losses due to rounded sides of a wafer.

$$\text{Dies per wafer} = \frac{\pi \cdot (\text{Wafer Diameter}/2)^2}{\text{Die area}} - \frac{\pi \cdot \text{Wafer diameter}}{\sqrt{2 \cdot \text{Die area}}} \quad (2)$$

These values can be used to calculate the cost of each die:

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \cdot \text{Die yield}}$$

1.14 Dependability

- **Mean Time to Failure (MTTF)** - How long is continuous service delivered, the reciprocal is the rate of failure. One failure for every length of time.
- **Mean Time to Repair (MTTR)** - How long is service interruption
- **Availability = MTTF / (MTTF + MTTR)** - Service is continuous for x amount of time for every y length, MTTF + MTTR.

1.15 Performance

Performance metrics can differ between computers and their purposes - WSCs will want to increase bandwidth, and mobile devices will want to increase the speed with which commands are executed.

$$\frac{\text{Execution time}_Y}{\text{Execution time}_X} = n = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

1.15.1 Execution Time

Amdahl's Law states that the performance improvement to be gained from using some fasted mode of execution is limited by the fraction of time the faster mode can be used.

$$\text{Speedup} = \frac{\text{Performance for entire task using enhancement when possible}}{\text{Performance for entire task without using the enhancement}}$$

The speedup depends on two factors:

1. The fraction of computation time in the original computer that can be converted to take advantage of the enhancement - $\text{Fraction}_{\text{enhanced}}$
2. How much faster the task would run if the entire program were enhanced - $\text{Speedup}_{\text{enhanced}}$

$$\text{Exec Time}_{\text{new}} = \text{Exec Time}_{\text{old}} \cdot ((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}})$$

The above equation cancels in several points to be equal to

$$\text{Execution Time}_{\text{old}} \cdot \text{Old Code} + \text{Execution Time}_{\text{new}} \cdot \text{New Code}$$

$$\text{Speedup}_{\text{new}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}}$$

$$\text{Speedup}_{\text{new}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

1.15.2 Processor Performance Equation

$$\text{CPU Time} = \text{CPU Clock Cycles for a Program} \cdot \text{Clock Cycle Time}$$

Clock cycles can also be defined as **Instruction Count** · **Clock Cycle Time**

$$\text{CPU Time} = \text{Instruction Count} \cdot \text{Cycles per Instruction} \cdot \text{Clock Cycle Time}$$

2 Instruction Set Architecture