Fundamentals of Quantitative Design and Analysis

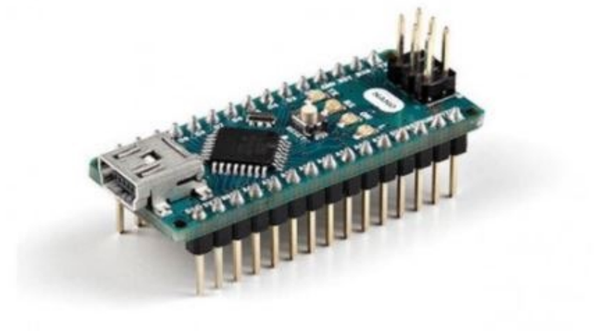
(Chapter 1)

Outline

- Classes of Computers
- Defining Computer Architecture
- Quantitative Principles of Computer Design

Classes of Computers

- Internet of Things/Embedded Computers
- Personal Mobile Devices
- Desktop Computing
- Servers
- Clusters/Warehouse-Scale Computers



Internet-of-thing computer

Clusters

Defining Computer Architecture

- A computer architecture covers three important aspects:
  - Instruction Set Architecture
  - Computer Organization
  - Hardware Technology

Instruction Set Architecture (ISA)

We use the term ISA to refer to the actual programmer-visible instruction set.

ISA serves as the boundary between software and hardware.

Popular ISAs include 80x86, ARMv8 and RISC V.

RISC V ISA is the major focus in our study.

◆Computer Organization and Hardware Technology

Computer Organization: High level aspect of computer design, including the design memory system and Central Processing Unit (CPU).

Hardware Technology: Logic circuit design and packaging technology of the computer.

Quantitative Principles of Computer Design

Here we explore some of the guidelines and principles that are useful in design and analysis of computers.

- Make the common case fast

In making a design trade-off, favor the frequent case over the infrequent case.

◆Amdahl's Law

The performance gain can be calculated using Amdahl's Law.

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

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

$$\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

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

Example:

Suppose that we are considering an enhancement to the processor of a server system used for WEB serving.

The new CPU is 10 times faster on computation in the WEB serving application than the original processor.

Assuming that the original CPU is busy with computation 40% of the time, and is waiting for I/O 60% of the time.

What is the overall speedup gained by incorporating the enhancement?

Sol:

$\text{Fraction}_{\text{enhanced}}=0.4$

$\text{Speedup}_{\text{enhanced}}=10$

Therefore, $\text{Speedup}_{\text{overall}}=\dfrac{1}{0.6+\dfrac{0.4}{10}}=1.56$

Example:

Our target now is to improve the performance of a graphic engine.

A common transformation required in graphics engines is square root. Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark.

There are two proposals for the performance improvement.

One proposal is to enhance the FPSQR hardware and speedup this operation by a factor of 10.

The other alternative is just try to make all FP instructions in the graphics processor run faster by a factor of 1.6; FP instructions are responsible for a total of 50% of the execution time for the application.

Compare these two design alternatives.

Sol:

$$\text{Speedup}_{FPSQR} = \frac{1}{1-0.2+\frac{0.2}{10}} = 1.22$$

$$\text{Speedup}_{FP} = \frac{1}{1-0.5+\frac{0.5}{1.6}} = 1.23$$

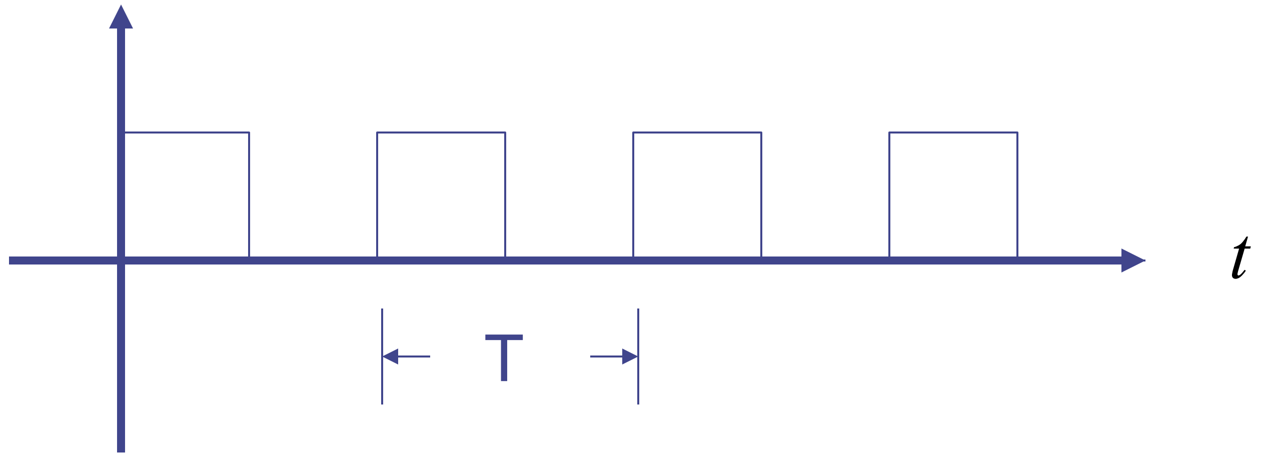
Improving the performance of the FP operations overall is slightly better because of the higher frequency.

The CPU Performance Equation

CPU performance =1/ CPU time

$$\text{CPU Time} = \frac{\text{Seconds}}{\text{Programs}}$$

All computers are constructed using a clock running at a constant rate.



T= clock cycle time (i.e., 1ns)

T^{-1}= clock rate (i.e., 1GHz)

We can then rewrite the CPU time as

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Programs}}$$

$$= \text{clock cycles per program} \times \text{clock cycle time}$$

Clock cycles per program can be divided further into two terms:

clock cycles per program =

$$\frac{\text{Instruction Count}}{\text{Programs}} \times \frac{\text{Clock Cycles}}{\text{Instruction Count}}$$

CPU time therefore can be rewritten as

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Programs}}$$

➔

$$\text{CPU time} = \underbrace{\frac{\text{Instruction Count}}{\text{Programs}}}_{\text{Instruction Count (IC) per Program}} \times \underbrace{\frac{\text{Clock Cycles}}{\text{Instruction Count}}}_{\text{Cycles per Instruction (CPI)}} \times \underbrace{\frac{\text{Seconds}}{\text{Clock Cycles}}}_{\text{Clock Cycle Time}}$$

As this formula demonstrates, CPU performance is dependent upon three characteristics:

(1) Instruction Count (IC),
(2) Cycles per instruction (CPI)
(3) Clock cycle time.

Unfortunately, it is difficult to change one parameter in complete isolation from others because the basic technologies involved in changing each characteristic are interdependent:

Clock cycle time → Hardware technology and computer organization,
CPI → Computer organization and instruction set architecture,
IC → Instruction set architecture and compiler technology

Example:

Complex instruction set computers (CISC) and
Reduced instruction set computers (RISC)

Target of CISC (such as VAX): provides powerful instructions for reducing instruction count (IC).

Target of RISC (such as RISC V): provides simple instructions for efficient hardware implementations for reducing CPI.

In general, performance of RISC is better than CISC.

Principle of Locality

Programs tend to reuse data and instructions they have used recently. An implication of locality is that we can predict with reasonable accuracy what instructions and data a program will use in the near future based on its accesses in the recent past.