Computer Systems Architecture

James Zhang*

September 4, 2024

These are my notes for UMD's MATH411: Computer Systems Architecture. These notes are taken live in class ("live-TEX"-ed). This course is taught by Professor Ilchul Yoon, whose notes are inspired by *Computer Architecture: a Quantitative Approach* by Hennessy and Patterson.

Contents

*Email: jzhang72@terpmail.umd.edu

Computer Architecture Landscape Changes

Here are some of the following things we will cover in the course:

- How processors exploit instruction/thread/data parallelism
- How processors/memory work
- A great deal of jargon in the system area
- How computer architecture affects programming style
- How programming style affects computer architecture

§1 Chapter 1

Definition 1.1 (Dependability)**.** Infrastructure providers started to offer **service level agreements** to guarantee how dependable their service will be. Failure is a transition from working to not working, and restoration is a transitiion from not working to working.

Definition 1.2. **Mean time to Failure (MTTF)** measures reliability, and **Failures in Time (FIT)** $= \frac{1}{\text{MTTF}}$, reported as failures per billion hours.
Mean time to repair (MTTR) measures service interruption
Mean time between Failures (MTBF) = MTTF + MTTR
Module Availability = MTTF / MTBF = MTTF / (MTTF + MTTR)

Note 1.3. If modules have exponentially distributed lifetimes, the age of the module does not affect its probability of failure, the overall failure rate (FIT) is the sum of failure rates of the modules.

Example 1.4
See slide 8 in lecture 3

Definition 1.5 (Performance)**.** Performance is the inverse of execution time

$$\text{Performance} = \frac{1}{\text{Execution Time}}$$

Example 1.6

- Unix time command example:
  - 90.7u 12.9s 2:39 65%
  - The user used the CPU for 90.7 seconds (user CPU time)
  - The system used it for 12.9 seconds (system CPU time)
  - Elapsed time (wall clock time) from the user's request to completion of the task was 2 minutes, 39 seconds (159 seconds)
  - And (90.7 + 12.9)/159 = 65%
    - » the rest of the time was spent waiting for I/O or running other programs

Note 1.7. To make computers faster, make common cases faster - make addition faster, instead of optimizing the square root.

Definition 1.8 (Amdahl's Law)**.** **Speedup** as the time the task took originally divided by the time the task takes after improvement
Suppose the original task takes 1 second, f seconds in the critical piece and $1-f$ in the non-critical. Then the improved task will only take f/s in the critical piece, but still $1-f$ in the noncritical. Therefore, the

$$\text{Speedup} = \frac{\text{Old Time}}{\text{New Time}} = \frac{1}{(1-f)+\frac{f}{s}}$$

Here s is the speedup factor, note that we want to target operations that have large f to maximize speedup.

Example 1.9

Suppose we get a speedup of 2 after we replace the GPU card. If the new card can process GPU instructions 5 times faster, what is the percentage of instructions affected by the new GPU?

Solution. We want to find f and $s = 5$

$$2 = \frac{1}{(1-f)+\frac{f}{5}} \implies 2 - 2f + \frac{2f}{5} = 1 \implies \frac{-8f}{5} = -1 \implies f = \frac{5}{8} = 0.625$$

□

§2 Appendix A

Definition 2.1 (Instruction Sets)**.** Instruction sets sit in between software and hardware, and a good ISA defines how the software controls the CPU in a computer. Some properties of a good abstraction are

- Lasts through many generations (portability)
- Generality
- Convenient functionality to higher levels
- Efficient lower level implementations

There are a few types of ISA classes.

- Stack - a block of memory in RAM, LIFO structure, pushing and popping
- Accumulator - places one operand in the accumulator (implicit) and one in memory (explicit memory address). The operand in the accumulator is loaded from memory using LOAD and the result is stored in memory from the accumulator using STORE.
- Register (register-memory)
- Regsiter (load-store) - this is what MIPS uses, this ISA divides instructions into two categories: memory access and (load and store between memory and registers) and ALU operations which only occur between registers.

Note 2.2 (Addressing Modes)**.** See slide 57 in lecture 2.