COMP 3370 – Computer Organization

Course Description

Calendar entry

Principles of computer systems architecture, organization, and design. Performance, instruction sets, processors, input/output, memory hierarchies. Prerequisite: COMP 2280 or ECE 3610.

General Course Description

This course builds on the memory and CPU organizations introduced in second year by investigating advanced computer architectures that go beyond a simple Von Neumann machine. Rather than being able to design computer hardware, this course aims to make students better software developers through a deeper understanding of how computer organization affects code. Students can expect to use assembly language and/or C to deepen their understanding of architectural design choices and the impact those choices have on the code they write.

In this course we will look at how we keep the CPU actively executing instructions. We will look at techniques that try to maximize the amount of concurrent and parallel work on a computer can do within a CPU, across multiple levels of memory, and through independent input/output operations. All of these techniques not only have a computer doing multiple things at the same time, they all change how and *when* code is executed. What you expect a computer to do with code you write is *not* what the computer will do with those instructions. How does this happen in a way that guarantees correct code execution? How does this change what is meant by good code? Welcome to COMP 3370.

Detailed Prerequisites

Before entering this course, a student should be able to:

- Demonstrate how the code they write gets interpreted and executed by a computer.
- Convert high-level code, including code representing data structures and recursive operations, into assembly.
- Implement common assembly provided by a compiler: including the run-time stack, parameter passing, local variables, and dynamic data structures.
- Explain how a simple set of circuits leads to the creation of Memory, the Central Processing Unit, and everything in between.
- Demonstrate how a state machine controls the execution of instructions.

Course Goals

By the end of this course students will:

- See how memory organization can both positively and negatively impact the performance of their code.
- Gain an understanding of modern CPU design and how those design choices lead to our ability to implement advanced operating systems and sophisticated applications.
- Be prepared to apply a greater understanding of CPU/device interactions to the implementation of device drivers.

Learning Outcomes

Performance

Students should be able to:

1. Calculate the performance of a computer in terms execution time.
2. Compare the performance of two computers running the same code.
3. Analyze attempts to improve computer performance using cycles per instruction relative to the number of instructions and CPU cycle time.

Memory

Students should be able to:

1. Compare cache memory implementation strategies and their effectiveness at different levels of the memory hierarchy.
2. Analyze the effect of a caching strategy on a memory intensive algorithm.
3. Explain the architectural designs needed to support the virtual memory algorithms used by operating systems.

I/O

Students should be able to:

1. Use timing diagrams to describe synchronous bus transfer operations.
2. Use interrupts and memory mapped I/O to implement the transfer of blocks of data between a storage device and memory independent of the CPU.
3. Demonstrate the impact of multiple units accessing memory in parallel.

CPU Pipelining

Students should be able to:

1. Show how pipelining affects the execution of a sequence of instructions.

2. Implement simple branch prediction algorithms and show how each algorithm works to minimize control hazards for a given piece of code.
3. Show how a given piece of code would have its instructions reordered to minimize data hazards.
4. Explain the software and hardware design choices used to minimize structural hazards.