CS 2340

Syllabus

Course Description

This course introduces the concepts of computer architecture by going through multiple levels of abstraction, the numbering systems and their basic computations. It focuses on the instruction-set architecture of the MIPS machine, including MIPS assembly programming, translation between C and MIPS, and between MIPS and machine code. General topics include performance calculations, processor datapath, pipelining, instruction level parallelism, and memory hierarchy, including cache memories.

Required Textbook

Computer Organization and Design 5th Edition

Course Materials

Assembler/Simulator: MARS MIPS

Syllabus: Syllabus

Grading Criteria

1. Exams (10%, 20%, 30%): open book, 2 midterms and 1 final
2. Assignments (20%): 8, around one per week
3. Projects (20%): details TBA

Assignment grading policy:

1. Code Development (30%): compiles without errors
2. Program Execution (20%): runs successfully
3. Program Design (25%): conforms to spec
4. Documentation (15%): program comments
5. Coding Style (10%): clear and efficient

No late homework or assignments.

Day 1: Jan 19

General

Reference notes file: **Assembler.pdf** (on eLearning)

Course will be using **MIPS**.

Computer Architecture:

ISA: Instruction Set Architecture

- **RAM** (Random Access Memory) communicates with the **CPU** as well as the storage disks
- **CPU** contains **registers** and **data path**
- **Von Neumann Principle** states that instructions and data resides in RAM (invented by Alan Turing)
- The larger the size, the slower the speed

MIPS RAM Memory Layout

- Reserved
- Stack $\Longrightarrow$ Dynamic Data $\Longleftarrow$ Heap
- Static data
- Text
- Reserved

Instruction Execution

- **Data**: flows between Memory and CPU (both ways)
- **Instructions**: only flows from **Memory** to CPU
- MIPS uses 32-bit integer registers $\Longrightarrow$ 16 Double long floating point
- Loop process (infinite loop, Instruction Fetch Cycle)
    1. Fetch instruction from RAM (Program Counter)
    2. Increment PC
    3. Execute instruction (may change PC)
    4. **ONLY** terminated by interrupts from outside source

Registers

There are 32 registers, and each of them are 32 bits long. All ALU instructions have 3 addresses and each is 5 bits, for example:

```
ADD $1, $2, $3
```

There are many reserved registers so be careful when choosing them.

Types of Instructions

1. **R-Type** (ALU functions): op, rs, rt, rd (result), shamt, funct
    1. Shift, add, multiply, divide, etc.
2. **I-Type** (loads and stores): op, rs, rt (result), imm
    1. Must load data before operating on them
    2. Store result back in store operation
    3. Around 20% of operations are I-Type
3. **J-Type** (unconditional branch): op, addr
    1. Jump somewhere

Clock Cycle

All RISC instructions take one **clock cycle** to execute.

Execution example: is

(Note: `Mem`=read from memory, `WB`=Write Back to memory)

```
IF -> ID -> IX -> Mem -> WB (I300)
      IF -> ID -> IX -> Mem -> WB (I301)
            IF -> ID -> IX -> Mem -> WB (I302)
```

Split instructions into smaller chunks so each can execute in one clock cycle. However, there are issues relating to dependencies between different instructions which causes delays. Goal is to get maximum throughput through the MIPS pipeline.

Coming up next: performance evaluation and some Law.

Day 2: Jan 21

In today's class, we're going to keep talking about classification, but we'll introduce a way of thinking about algorithms with the following basic underlying idea: come up with machinery that lets us derive machine learning algorithms for arbitrarily propagated problems. This will be the foundation of much more complicated learning methods later on in this class!

Basically, we'll turn machine learning problems into optimization problems: taking a function and using computational methods to find its minimum or maximum.

Definition

The **objective function**, often denoted $J(\Theta)$, is a measurement of "how well we're doing:" it's the quantity we're trying to optimize with respect to Θ