Wrapup

CMPU 224 – Computer Organization
Jason Waterman
We Made It!

FINISHED 224

AW YISS!
We Made It!

IT'S OVER

IT'S DONE
End of the Semester Logistics

• Lab due: Friday May 10\textsuperscript{th} 11:59pm
  • Last day to turn in assignment (with late days): Sunday May 12\textsuperscript{th}, 11:59pm
  • No exceptions (per order of Dean of Studies)

• Final: Tuesday May 21\textsuperscript{st} 9:00am - 11:00am Olmsted Hall 266
  • Closed books and closed notes, no electronic devices
  • Two hour final
  • Comprehensive, but will focus on material since the second quiz
  • I’ll stay a bit after the final (till noon) so you won’t feel cramped for time
  • I’ll post some practice problems to the course website
In its broadest definition, computer organization is the design of the abstraction/implementation layers that allow us to execute information processing applications efficiently using manufacturing technologies.
Recap: What is Computer Organization?

Application
Algorithm
Programming Language
Operating System
Instruction Set Architecture
Microarchitecture
Register-Transfer Level
Gates
Circuits
Devices
Physics

CMPU-224 Computer Organization
The Other Purpose of this Course

- To learn how to write better programs by learning what is going on “under the hood” of a computer system
  - Data representation
  - Syntax and operations in C
  - Machine-level representation of programs X86-64
  - Y86-64 instruction set architecture
  - Y86-64 Sequential and Pipeline Implementation
  - Memory systems
  - Cache memories
  - Program optimization
  - Whew!
Review: Data Representation

- Basic C data types (char, short, int, long, float, and double)
  - Understanding the difference between the little-endian and big-endian representation of numbers

- Compound data types: arrays and structs
  - How they are laid out in memory, alignment rules
  - Relationship between pointers and arrays, pointer arithmetic
  - Code for accessing array elements and struct fields

- Converting numbers to and from decimal, binary, and hexadecimal
  - Be able to convert to and from a decimal number to any base-n system

- 2’s complement numbers
  - Conversion between unsigned and 2’s complement numbers
  - Be able to subtract two numbers using 2’s complement addition
  - How to perform 2’s complement negation

- Floating point representation of numbers

- Understand whether an operation will result in overflow
  - What the result from an overflow is
Review: C Syntax and Operations

• Bit-level operations
  • and (&), or (|), not (~)
  • Shift operations: left shift (<<), right shift (>>)
  • Difference between arithmetic and logical shifts
  • Mathematical equivalent of shifting

• Logical operations: and (&&), or (||), not (!)
  • Differences between logical and bit-level operations

• Looping: while loops, do while loops, and for loops
  • How to convert from one looping construct to another and goto representation of these loops
  • Assembly level representation of these loops

• Branching: if and switch statements
  • Assembly level representation of branches and jump tables

• Pointers: how to declare, dereference, and use them

• Functions:
  • How do declare and use functions in C
  • Difference between passing arguments by reference and passing arguments by value
Review: Machine-level Representation

- x86-64 registers and their special uses (if any)
- Understanding assembly operand forms (Fig 3.3 page 181 in the book)
- Difference between the leaq instruction and the movq instruction
- The mechanics of a function call
  - Where are the arguments, local variables, return value
  - What is done during the call and ret instructions
- Condition codes: CF,ZF, SF, and OF flags, how they are set and how other instructions use them (e.g., jne)
  - Understand how the cmpq and test instructions work
- The set, jump, and conditional move families of instructions
Review: Y86-64 Instruction Set Architecture

• Meanings of Y86-64 Instructions and how they differ from the x86-64 instructions

• The programmer-visible state of the Y86-64 ISA
  • Registers, condition codes, program counter
  • Programs status (AOK, HLT, ADR, INS) and when they occur

• Understand how Y86-64 instructions are encoded into their byte-level machine representation
  • Translate to and from Y86-64 instructions and machine code

• Given a x86-64 program, be able to translate it into a Y86-64 program
Review: Logic Design and HCL

- **Logic gates:** and, or, and not gates
- **Combinational circuits:** acyclic network of logic gates
  - Convert to and from truth tables, boolean equations, circuit diagram
    - Sum of products: a two-level representation of a truth table as a logical sum of products
  - Building blocks: equality circuit, multiplexors
  - ALU operation
- **Sequential Logic:**
  - Bistable Element, R-S Latch, D Latch, Edge-Triggered Latch
  - Registers: change on the rising edge of the clock
  - Register file: reads happen asynchronously, writes on rising edge of the clock
- **HCL:** Hardware Control Language
  - Implement logic functions using HCL
Review: Sequential Y86-64 Implementation

• The stages in the SEQ processor
  • Fetch, Decode, Execute, Memory, Write back, PC update
  • What happens in each stage, inputs/outputs for each state

• For an existing or new Y86-64 instruction
  • Be able to trace the execution of the instruction through each of the stages
  • See the asides from p388 to p395

• The timings in the SEQ implementation
  • When do the various components get updated?
Review: Pipelined Y86-64 Implementation

- The advantages and disadvantages of the pipelined implementation
- For a pipeline diagram and timing information, compute throughput and latency of the system
- The limitations of pipelining
- The five stages of the Y86-64 pipeline processor
  - Fetch, Decode, Execute, Memory, Write back
  - How are they rearranged from the sequential implementation
- The Y86-64 pipeline registers and how are they used
- Pipeline hazards and how do we avoid them
  - Data hazards, control hazards, stalling, bubbles, data forwarding
- For Y86-64 program, show a pipeline diagram of the program
  - See figure 4.56 on page 444 as an example
- Given a Y86-64 program, be able to compute the CPI for the program
Review: Memory Systems

• The difference between static (SRAM) and dynamic (DRAM)
  • The advantages and disadvantages of each
  • The relative performance between them

• Locality:
  • The difference between temporal locality and spatial locality
  • Identify programs with good locality and ones with poor locality
  • Stride-n memory references and the effect on locality
Review: Cache Memories

• Cache organization (S, E, B, m)
• For a given (S, E, B, m), the parts of m used for the block offset, set, and tag bits
  • Perform a cache lookup for a given address
• The difference between a direct mapped, n-way set associative, and fully associative caches
• The operation of LRU (least recently used) cache line replacement policy
  • The cache line modifications needed to support LRU replacement
• Given a trace of memory accesses, simulate the behavior of a cache, deciding if each access results in a hit, miss, or eviction
• The difference between write-back and write-through caches
Review: Optimization

- **Code motion**
  - Reduce the frequency with which a computation is performed

- **Strength reduction**
  - Replace a costly operation with a simpler one

- **Loop unrolling**
  - Reduce overhead by doing more work per loop

- **Optimization blockers (memory aliasing, functions)**
  - Accumulate using local variable within loops
  - Compiler treats functions as a black box

- **Cycles per element to express program performance**

- **Functional unit performance (latency, throughput)**
  - Reduce data dependencies to maximize throughput