EE 4720—Computer Architecture

Call Number 1380 (Spring 1998)

URL: http://www.ee.lsu.edu/ee4720

Offered by:
David M. Koppelman
349 EE Building
388-5482, koppel@ee.lsu.edu, http://www.ee.lsu.edu/koppel
Tentative office hours: Monday, Thursday 14:00–16:30

Teaching Assistant:
Khalid Al-dajani
aldajani@ee.lsu.edu

Should already know:
How to design a computer.

Will learn:
How to design a *good* computer.

Prerequisites By Course:

EE 3755, Computer Organization. (Current name.)

EE 4730, Structure and design of digital computers. (Old name.)

Prerequisites By Topic:

- Logic design.
- Computer organization.
- Assembly-language programming.

Text

"Computer architecture, a quantitative approach," John L. Hennessy & David A. Patterson, Second Edition.

Course Content

- Importance of *instruction set architecture (ISA).*
- Using cost and performance to guide design.
- Instruction set design.
- Pipelining.
- Multiple-issue techniques.
- Memory.

Course content will closely follow text.

Lecture material not in book will be marked: (NIB).

Graded Material

Midterm Exam, 40%

Fifty minutes, closed book.

Final Exam, 40%

Two hours, closed book.

Homework, 20%

Lowest grade or unsubmitted assignment dropped.

ISA and Implementation Distinction

What is a computer?

A machine that executes instructions which read and write memory.

What a computer engineer does:

- Develops an *instruction set architecture* (ISA).
- Designs hardware to execute, *implement,* the instruction set.

Definitions

Instruction Set Architecture (ISA):
Precise definition of computer's instructions and their effects.

- It's all programmer needs to program machine.
- It's all hardware designer needs to design machine.

Implementation [of an ISA] (noun):
Hardware that executes instructions defined by ISA.

Instruction Set Architecture

ISA and Implementation Examples

ISA: SPARC V8. (Developed by Sun for its workstations.)

Impl: Cypress CY7C601 and Fujitsu MB86900/1A.

Who ISA Developed For

- Compiler writers.
- Compute-intensive library writers.
  E.g., graphics and scientific libraries.

Instruction set requirements don't change very much over time.

Scope of ISA Specification

Describes instruction codings, and what they should do.

Should specify action of all codings, used or not . . .

Why an ISA should specify behavior of unused codings:

- Reserve future instructions.
- Avoid differences in behavior of implementations, intentional or not.

Consequences of Not Specifying Behavior of Some Codings

Programmers may use unspecified coding . . .

. . . producing code that may not run on new implementations . . .

. . . or forcing all implementations to act the same way . . .

. . . so the unspecified behavior is a de-facto instruction.

Implementation

Two aspects of implementation: *organization* and *hardware*.

Organization:
Details of functional units, data paths, control, etc.

Also called *microarchitecture*. (NIB[1]).

This includes memory system, bus, and CPU.

Hardware:
Logic design and packaging.

Course focus: ISA and organization, not hardware.

[1] Not in book.

Technological Change

Technological Change and Computer Designer

Technology determines "raw materials" for designer.

ISA lifetime can be decades.

Raw materials greatly change over this time.

So, design ISA for now and future.

How technological advancement affects processor.

Transistor Speed, Clock Rate
No changes to organization or ISA.

Number of Transistors Available
Changes to organization and possible changes to ISA.

Memory Size
Change ISA to use larger address space.
Can use ISA having larger instruction codings.

Memory Speed Compared to Processor Speed
Include more sophisticated caching in organization.

Summary

What a computer engineer does:

- Develops an *instruction set* (ISA).
- Designs hardware to execute instruction set.

If instruction set *poorly* designed . . .

. . . many instructions will not be used (wasting silicon) . . .

. . . and instructions will execute slowly.

Why ISA design is surprisingly difficult:

- Hard to predict which instructions useful . . .
  . . . without writing and running software using instructions.
- Hard to predict which instructions fast . . .
  . . . in current *and future* technologies.