

**CPS104
Computer Organization
Lecture 1**

Robert Wagner

Slides available on:
<http://www.cs.duke.edu/~raw/cps104/Lectures>

cps104 L01OV 1
©RW Fall 2000

CPS104: Computer Organization

Instructor: Robert Wagner
Office: LSRC D336, 660-6536 raw@cs.duke.edu
Office Hours: Tue. & Thur., 3:30-4:30 or by appt.

TA:
TA Office:
TA Office Phone:
TA Office Hours:

UTAs:
Text: Computer Organization & Design: The Hardware /
Software Interface (2nd printing).
Web page: <http://www.cs.duke.edu/~raw/cps104/>
Newsgroup: duke.cs.cps104

cps104 L01OV 2
©RW Fall 2000

Meat of the Course

★ What is a Stored Program Digital Computer?

- What are the actions it can REALLY perform?
- How can these actions be used?
- How can hardware to perform them be designed?

★ How do these “internal details” affect programs you write?

- Computer arithmetic vs. human
- Performance (execution-time speed)

★ Why learn how hardware is built?

- Underlying technology changes over time
- Be better prepared for changes
- Understand why some operations are slow
 - Decide for yourself if a new technology would make them more usable

cps104 L01OV 3
©RW Fall 2000

Course Outline:

- ★ Introduction to Computer organization and brief history.
- ★ C and C++
- ★ Instruction Set Architecture.
- ★ Assembly level programming.
 - Relation to HLL programming.
 - Instruction and data type representations.
 - Addressing, procedure calls and Exceptions.
- ★ Digital Logic:
 - Digital Gates and Boolean Algebra.
 - Arithmetic and Logic circuits

Course Outline (continue):

- ★ The Central Processing Unit (CPU).
 - The ALU.
 - The data path.
 - Controlling the CPU.
- ★ The Memory Hierarchy.
 - Cache Memory.
 - Virtual Memory and Paging.
- ★ Interrupts.
- ★ Pipelining (if there is time).
- ★ I/O Devices and Networks.
- ★ Advanced Computer Architecture (if there is time)

Grading

- ★ Grade breakdown
 - Midterm Exam: 15%
 - Final Exam: 35%
 - Homework Assignments 50%
- ★ Late homework policy:
 - No sad stories!
 - No "cooperation" on homework (Unless specified in the assignment).
 - 3% reduction for each day late.
- ★ Grades e-mailed:
 - Written/email request for changes to grades.

Homework-0

★ Send me (raw@cs.duke.edu) email message with: your name, year, major and a short description of your computer science / Engineering background.

★ Readings: Chapter-1, next time we start on Chapter-3.

Course Organization

★ All programs in this course will be written in "C" -- not C++ -- unless otherwise specified. The programs must compile and run using "gcc", without use of any libraries.

★ Programs in this course will be graded on correctness, understandability, speed, and elegance

- Correctness requires that you understand what the problem requires, and build a program to produce the required results -- and nothing else
- Elegance = minimum number of C "words" -- comments don't count

★ Most projects done in teams

- Every team member is responsible for understanding the entire project
- Quizzes will be given in class to test this
- Consider splitting your team into 2 competitive parts, each working somewhat independently. The 2 versions can be tested against one another for correctness (same answers), speed, and elegance

Course Problems

★ Can't make midterm

- Tell us early and we will schedule alternate time

★ Late homework problems:

- As a result of feedback, going to grade almost immediately so that can give results back quickly => late homework is a hassle

★ What is cheating?

- Studying together in groups is encouraged
- All written work must be your own. Programs that are substantially the same as others will receive a grade of 0!
- Common examples of cheating: running out of time on a assignment and then pick up someone else's output, person asks to borrow solution "just to take a look", copying an exam question, ...

CPS104: So what's in it for me?

- ★ In-depth understanding of the inner-workings of modern computers, their evolution, and tradeoffs present at the hardware/software boundary.
 - Understanding of the reality underlying theoretical computer models
 - Insight into fast/slow operations that are easy/hard to implement
 - Better understanding of operating System and compiler functions
- ★ Designer's "Intellectual" toolbox.

Things We Hope You Will Learn from 104

- ★ A computer is NOT "magic in a box".
 - It is important to understand principles.
 - All computers work in similar ways.
 - The more you understand how it works, the better you can use the tool.
- ★ Keep it simple and make it work (applies to both programs and hardware designs)
 - Fully test everything individually and then together
 - Retest everything whenever you make any changes
 - Last minute changes are big "no no's"
- ★ Planning is very important:
 - Murphy's Law: things *DO* break at the last minute
 - Don't make your plan based on the best case scenarios
 - plan to look at more than one design -- START EARLY
- ★ Never give up! It is not over until you give up.

Depth of Knowledge Expected

- ★ We expect you to learn the material well enough to solve problems
- ★ Exams and homework will test your ability to APPLY knowledge to examples that differ from those presented in class
- ★ We expect you to learn how to write simple programs in C, and Assembly Language, to design gate-level circuits, and to be able to modify hardware designs made up of circuits for useful building blocks
- ★ We will also expect you to be able to understand and simulate the performance of major components like the CPU, the cache, the virtual memory system, and I/O devices – either mentally, or by writing a computer program which simulates the device

Big Ideas Underlying Computer Hardware

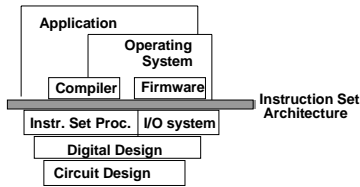
- ★ Represent EVERYTHING in the same way: as sequences of 1's and 0's
 - Interpret as binary integers, or printable characters, or bit patterns
- ★ Provide a few fundamental operations on bit patterns
 - Add, negate : definitions based on binary integer interpretation
 - If (n<0) then goto L : tests one bit for value 1
 - Load and Store: move bit-sequences between locations
 - Shift : Move bits around within one bit-sequence
- ★ Build "useful" operations from combinations of the fundamental ones
 - "for" and "while" loops
 - Multiplication and Division
 - Array indexing
- ★ Concept of "hierarchical design" replicated when useful software is developed: "useful operations" become "methods"

cps104 L01OV

13
©RW Fall 2000

What is "Computer Architecture"

◦ Coordination of *levels of abstraction*

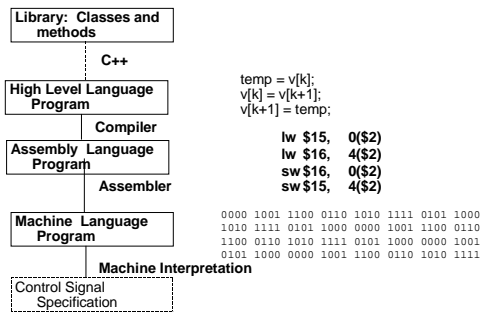


◦ Under a set of rapidly changing *Forces*

cps104 L01OV

14
©RW Fall 2000

Review: Levels of Representation



cps104 L01OV

15
©RW Fall 2000

Instructions

- ★ The most elementary order a programmer can give to a computer
- ★ Logical form: A "subroutine call"
 - Built in to the hardware
 - performs very specific operations
 - uses "arguments" which locate its operands within the computer
 - Operands are stored in memory
 - Located by the number (Address or Location) which identifies the "pigeonhole" where they are stored
- ★ Instructions are themselves stored in memory
- ★ Are represented in one memory location (or Word), as a sequence of "fields", each of which holds a logically separate part of the instruction:
 - | | | | |
|----|------|------|------|
| OP | DEST | SRCA | SRCB |
|----|------|------|------|
 - Each field holds a number, which is used when the hardware interprets the instruction – according to that instruction's definition

cps104 L010V

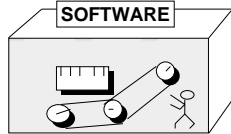
16
©RW Fall 2000

Instruction Set Architecture

... the attributes of a [computing] system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and control, the logic design, and the physical implementation.

Amdahl, Blaaw, and Brooks, 1964

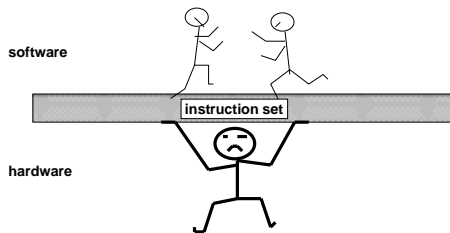
- Organization of Programmable Storage
- Data Types & Data Structures: Encoding & Representations
- Instruction Formats
- Instruction (or Operation Code) Set
- Modes of Addressing and Accessing Data Items and Instructions
- Exceptional Conditions



cps104 L010V

17
©RW Fall 2000

Instruction Set Interface



cps104 L010V

18
©RW Fall 2000

MIPS I Instruction Set Architecture

★ Instruction Categories

- Load/Store
- Computational
- Jump and Branch
- Floating Point
- Memory Management
- Special

R0 - R31

PC

HI

LO

3 Instruction Formats: all 32 bits wide

OP	rs	rt	rd	sa	funct
OP	rs	rt	immediate		
OP	jump_target				

cps104 L01OV 19 ©RW Fall 2000

Organization

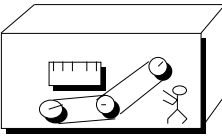
ISA Level

FUs & Interconnect

Logic Designer's View

- Capabilities & Performance Characteristics of Principal Functional Units
(e.g., Registers, ALU, Shifters, Logic Units, ...)
- Ways in which these components are interconnected
- nature of information flows between components
- logic and means by which such information flow is controlled.

Choreography of FUs to realize the ISA
Register Transfer Level Description



cps104 L01OV 20 ©RW Fall 2000

Execution Cycle

Instruction
Fetch

Obtain instruction from program storage

Instruction
Decode

Determine required actions and instruction size

Operand
Fetch

Locate and obtain operand data

Execute

Compute result value or status

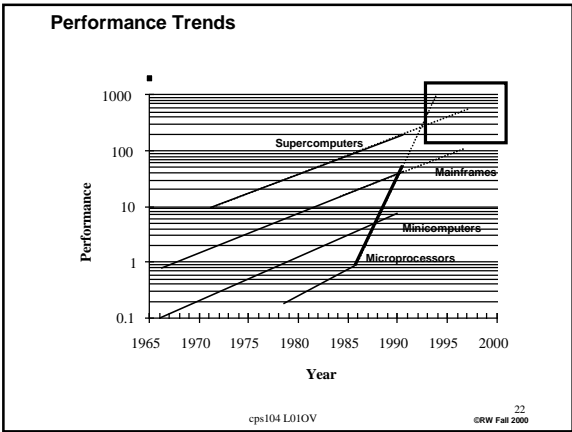
Result
Store

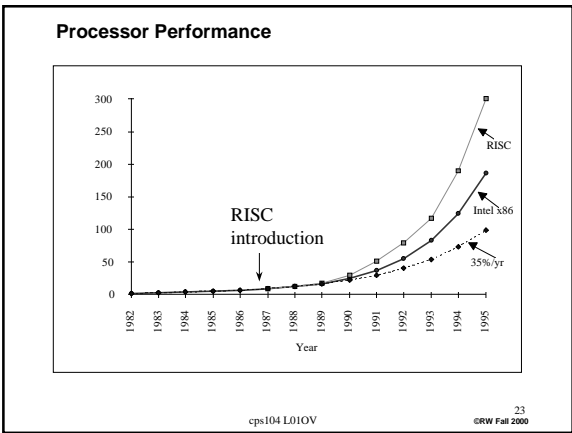
Deposit results in storage for later use

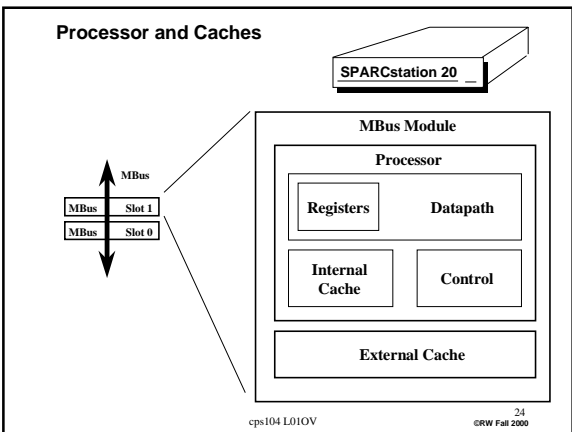
Next
Instruction

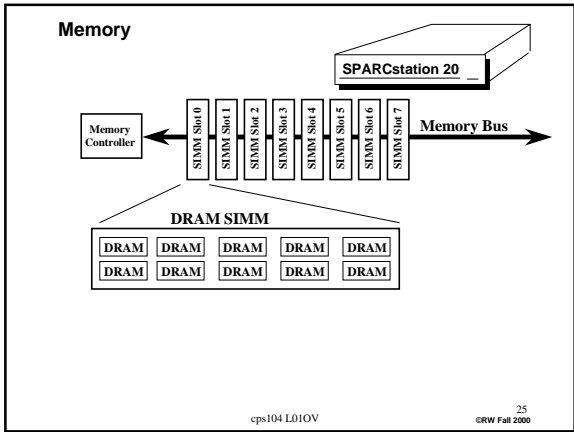
Determine successor instruction

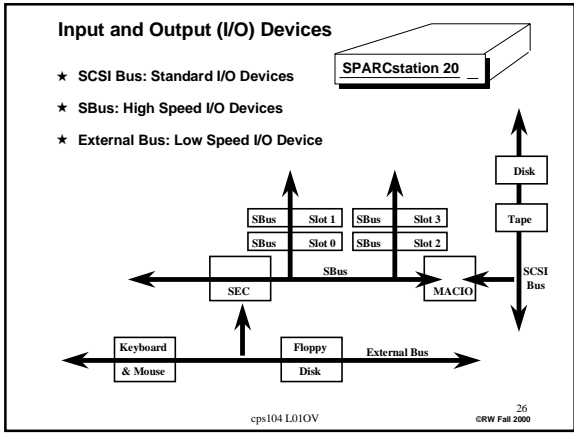
cps104 L01OV 21 ©RW Fall 2000











Summary

- ★ All computers consist of five components
 - Processor: (1) datapath and (2) control
 - (3) Memory
 - (4) Input devices and (5) Output devices
- ★ Not all "memory" is created equally
 - Cache: fast (expensive) memory is placed closer to the processor
 - Main memory: inexpensive, slower memory--we can have more
- ★ Input and output (I/O) devices have the messiest organization
 - Wide range of speed: graphics vs. keyboard
 - Wide range of requirements: speed, standard, cost ... etc.
 - Least amount of research (so far)

cps104 L01OV 27
©RW Fall 2000
