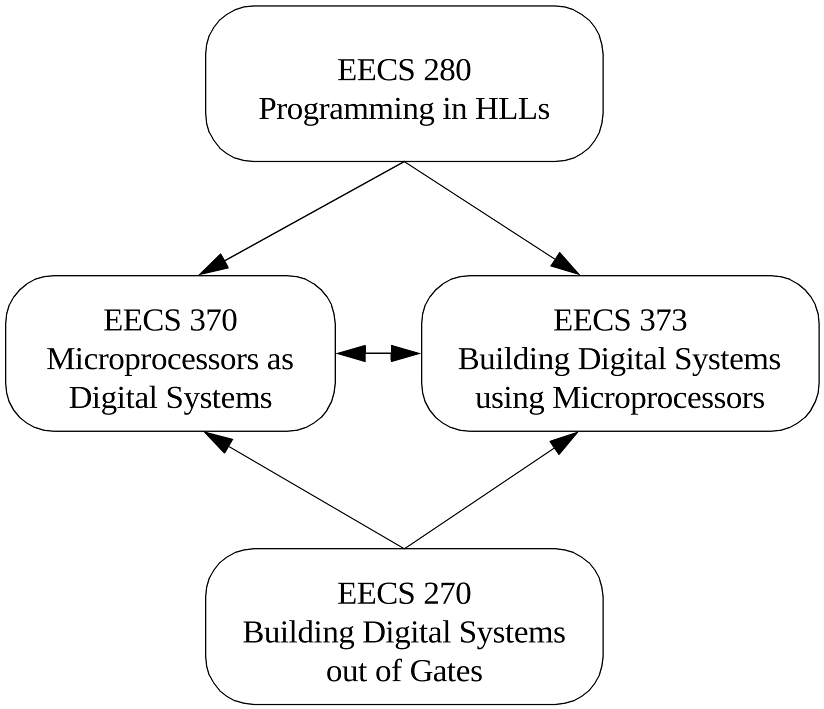
EECS 373

Design of Microprocessor-based Systems

Fall 1999 Course Notes

Prof. Steven K. Reinhardt

Where does this course fit in?



Overview

What is this course about?

- how to design and build systems using microprocessors

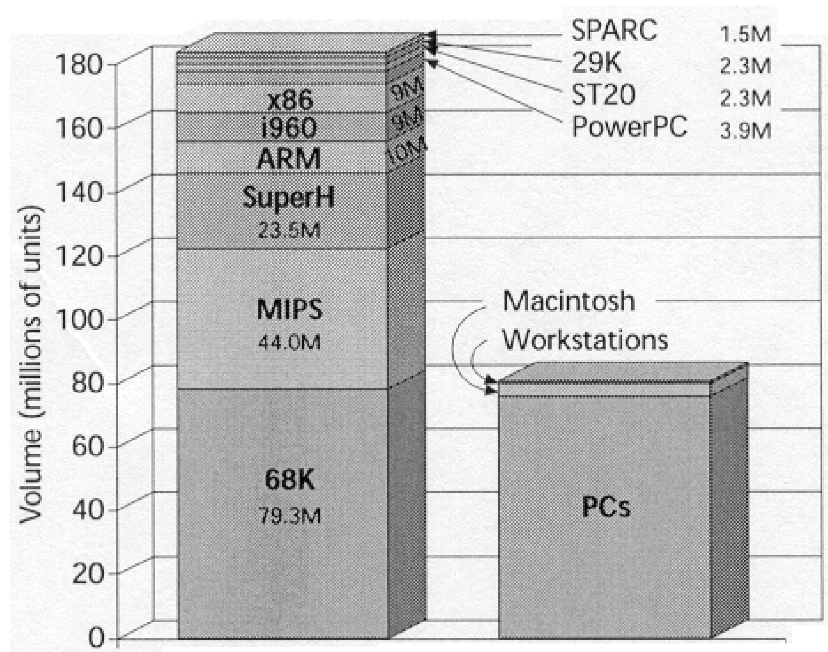
What are some examples of microprocessor-based systems?

Why are more and more systems using microprocessors?

- microprocessors are very cheap (< $1 at the low end)
- an off-the-shelf microprocessor + software can replace a lot of application-specific logic
- software enables flexible, sophisticated features that would be difficult or impossible otherwise
- software is typically easier to debug & fix than hardware
- more and more information is being stored and transmitted in digital form

Embedded vs. general-purpose systems

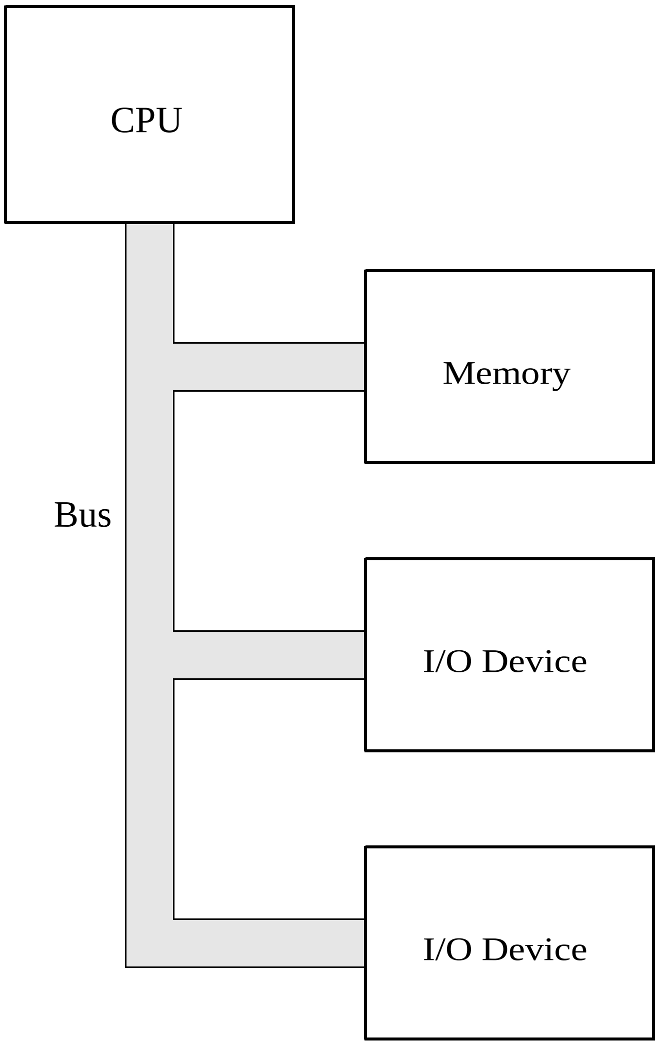
A microprocessor-based system that is dedicated to a specific task or tasks as part of a larger product is referred to as an *embedded system*. Although embedded systems get much less publicity, they vastly outnumber general-purpose systems in the market.



Source: *Microprocessor Report*

This chart shows the number of 32-bit microprocessors shipped in 1997. On the left are processors shipped in embedded systems; on the right are processors shipped in general-purpose systems. Note that this chart does not include hundreds of millions of 8-bit and 16-bit embedded processors.

Generic microprocessor-based system



Microprocessor-based System Features

Unlike the digital systems you built in EECS 270, the physical interconnection of components in a microprocessor-based system doesn't indicate its function. The system's function is primarily determined by the software (instructions) executed by the processor.

Instead, all the primary system components are connected to a common *bus*. A bus is simply a group of signals (wires) that communicates among several devices. Two devices communicate when one device (typically the processor) sends data to or requests data from another device over the bus.

A typical microprocessor system bus is composed of three smaller busses:

1. address bus: indicates the device and location within the device that is being accessed
2. data bus: carries the data value that is being communicated
3. control bus: control signals that indicate what's going on on the address and data busses

The *central processing unit (CPU)* is the core of the processor where instructions get executed. Instructions are encoded in *machine language*, which maps every instruction to a binary value. We will primarily use *assembly language* in this class, which is a human-readable encoding of the same set of instructions.

high-level language: a = b + c
assembly language: add r1, r2, r3
machine language: 01111100001000100001101000010100

Memory

- Stores instructions (programs) & data
- Organized as an array of locations (addresses), each storing one byte (8 bits). Reading (retrieving data from) a particular location always returns the last value stored (sent to) that location.

I/O devices

- Let system interact with the world
- *Device interface* (a.k.a. *controller* or *adapter*) is digital logic that connects actual device to bus
- examples?
- *I/O device registers* look just like memory to the CPU: a bunch of locations that can be accessed over the bus. However, I/O device registers are connected to other things (external wires, device control logic, etc.)
- Result:
  1. reads usually don't return last value written (e.g., may be value of last key pressed on keyboard instead)
  2. writes usually have side effects (e.g., display character on screen)

Course topics (not in order)

- PowerPC architecture & assembly language programming
- Bus designs, protocols, and interfacing
- Standard busses (PCI, USB, etc) & bus bridging
- Common I/O devices:
  - Timers
  - Analog-to-digital, digital-to-analog conversion
  - Serial I/O
  - Video
- Interrupts: I/O devices demand attention
- DMA: I/O devices talk directly to memory
- Memory technologies: SRAM, DRAM (page mode, EDO, synchronous, Rambus), Flash, etc.

