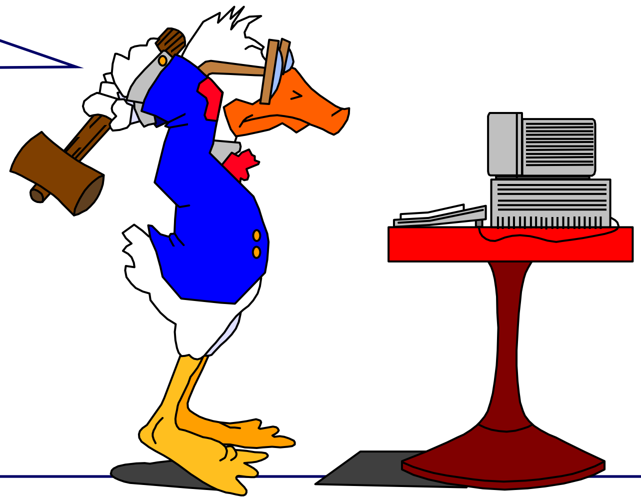
Chapter 1 -
Let’s Get Started…

What’s inside
this thing...



Computers…

- There is no magic to computing.
- Computers do not have minds of their own. You have to tell them *exactly* what to do.
- Computers follow instructions *exactly*.
- Computers are made of very simple parts… albeit, fast parts and a whole lot of them!
- We will study a simple computer, the LC-3, that has all of the important characteristics of today's commercial computers.

Two Recurring Themes…

- Abstraction
  - To manage Complexity
  - To enhance Productivity
  - To hide unnecessary details when everything works
  - To allow us to focus on what is important
- There is no Hardware vs. Software
  - Just names for different parts of a computing system
  - You will be much more capable if you master both
  - Computing problems are best solved when you understand both parts.

The Concept of Abstraction

- We abstract naturally–
  - focusing on the essential aspects of an entity, and
  - hiding the underlying complexity.
- One doesn’t want to get bogged down in unnecessary details (when everything is working fine).
- Its more efficient to think about something at the highest possible level of abstraction.
- Without abstraction, you could not design a computer. You would be overwhelmed by the complexity.
- <u>But if it doesn’t work</u>, then the abstraction fails, and you have to look at the details.

Hardware versus Software

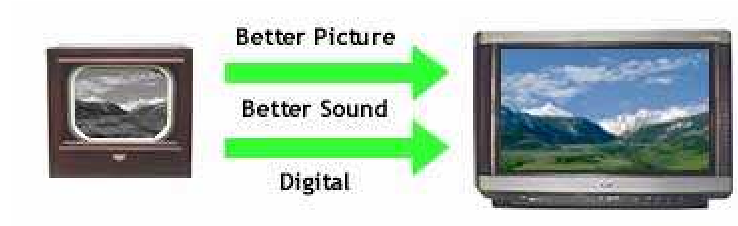
- Some people think that its OK for–
  - Software engineers to be clueless about hardware, or
  - Hardware engineers to be clueless about software.
- ***Don't believe it!*** Hardware and software are two parts of a computing system that work best when they are designed by someone who understands both parts.
- Hardware designers that understand programs and compilers design the best microprocessors.
- Software designers that understand the capabilities and limitations of hardware design more efficient programs.

A Computer

- A computer does two things:
  - It directs the processing of information. (Figure out what to do next– control)
  - It performs the actual processing of information. (Do the computations on the data– datapath)
- It does both in response to a computer program.
- The computer is called a
  - A <u>central processing unit</u>, or
  - A <u>CPU</u>, or simply
  - A <u>processor</u>.

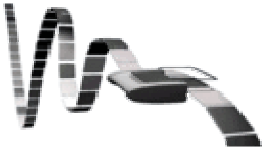
Digital vs. Analog

- Wristwatches (numbers vs. hands)
- LP's vs. CD's
- Rotary phone vs. Cell phone
- NTSC vs. HDTV
- Slide rule vs. calculator
- 737's vs. 777's



Universal Computing Devices

- Alan Turing
  - In 1937 he proposed a way to define the term "computable"
- The Turing Machine
  - Anything that *can* be computed, *can* be computed by a TM…
  - The TM is not a real machine, but an *abstract* machine



Turing Machine

- Mathematical model of a device that can perform any computation – Alan Turing (1937)
  - ability to read/write symbols on an infinite "tape"
  - "state" transitions, based on current state and symbol

- Every computation can be performed by some Turing machine. *(Turing's thesis)*

a,b → $\boxed{T_{add}}$ → a+b

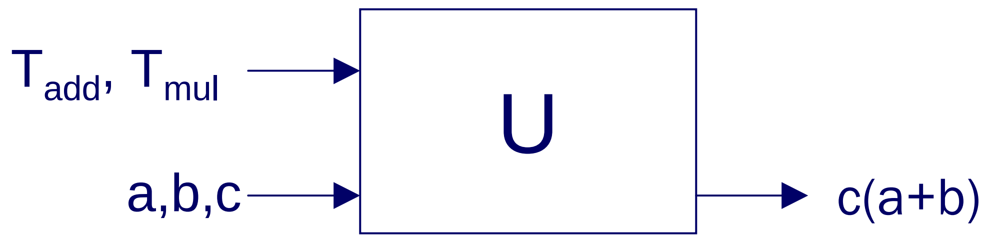
Turing machine that adds

a,b → $\boxed{T_{mul}}$ → ab

Turing machine that multiplies

Universal Turing Machine

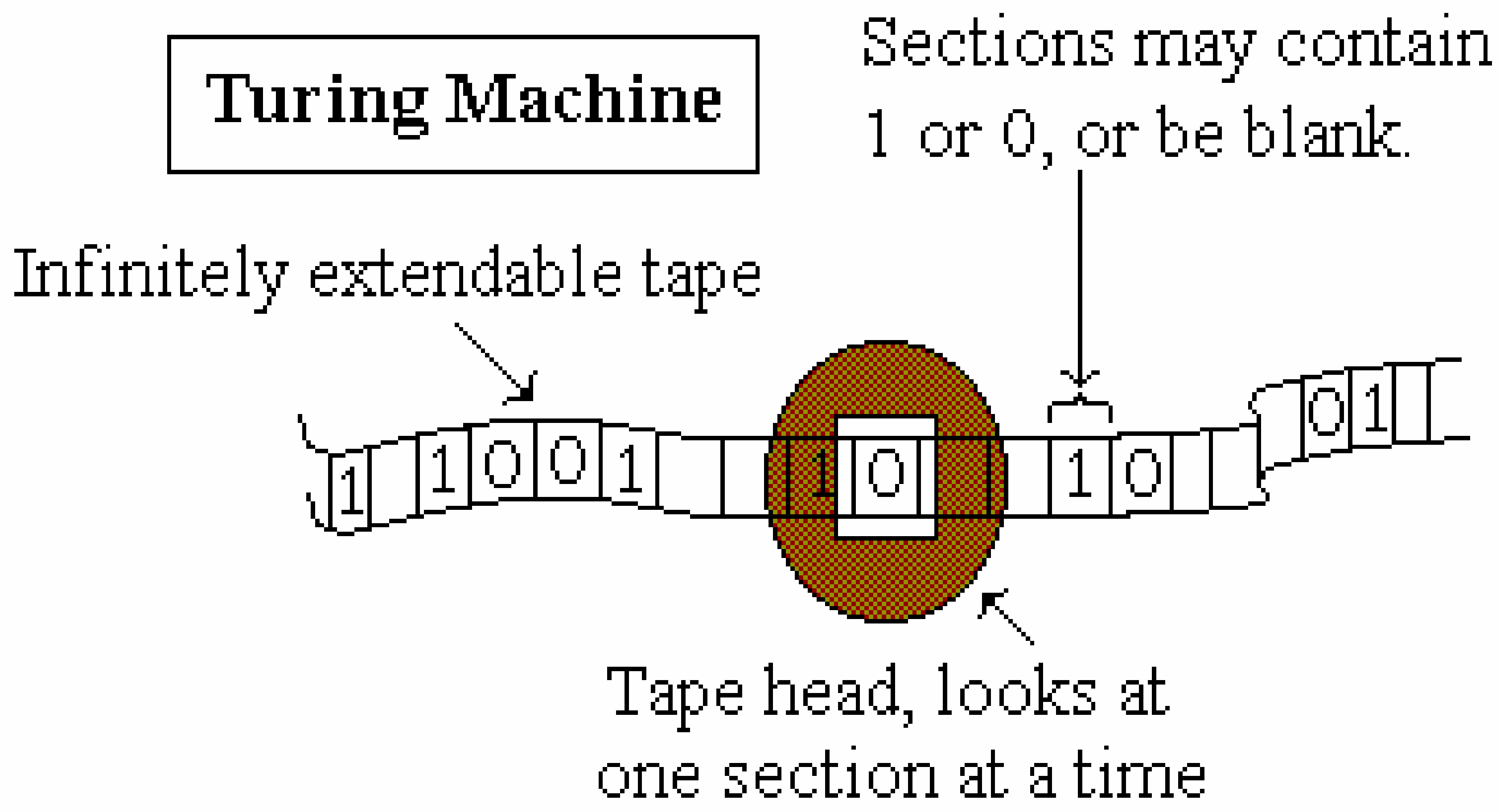
- A Universal Turing Machine is a theoretical device
  - that can implement all Turing machines
  - that accepts both input data and instructions as to how to operate on the data
  - that is also a Turing Machine!



Universal Turing Machine

A computer is a universal computing device.

Turing Machine Details



Turing Machine Example

Old St	Read Sym		Write Sym	New Mv	New St
s1	1	→	0	R	s2
s2	1	→	1	R	s2
s2	0	→	0	R	s3
s3	0	→	1	L	s4
s3	1	→	1	R	s3
s4	1	→	1	L	s4
s4	0	→	0	L	s5
s5	1	→	1	L	s5
s5	0	→	1	R	s1

Step	State	Tape
1	s1	**1**1
2	s2	0**1**
3	s2	01**0**
4	s3	010**0**
5	s4	01**0**1
6	s5	0**1**01
7	s5	**0**101
8	s1	1**1**01
9	s2	10**0**1
10	s3	100**1**
11	s3	1001**0**
12	s4	100**1**1
13	s4	10**0**11
14	s5	1**0**011
15	s1	11**0**11

--halt--

All Computers Are Created Equal…

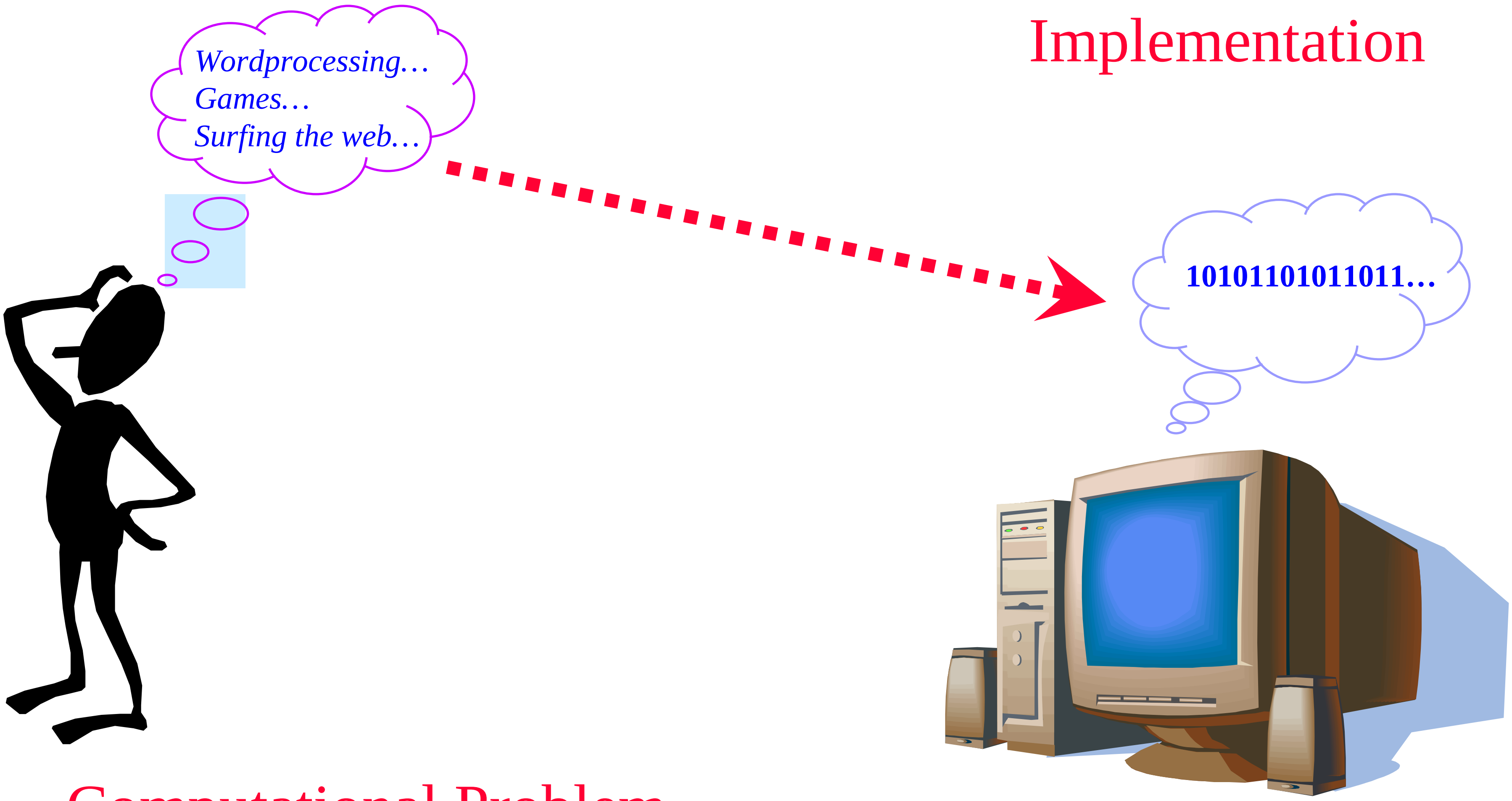
- In theory, a computer can ***compute*** anything that's possible to compute
  - given enough *memory* and
  - given enough *time*
- In practice, ***solving problems*** involves computing under constraints.
  - time
    - weather forecast, next frame of animation, ...
  - cost
    - cell phone, automotive engine controller, ...
  - power
    - cell phone, handheld video game, ...

Computing Machines

- Ubiquitous (= everywhere)
  - General purpose: servers, desktops, laptops, PDAs, etc.
  - Special purpose: cash registers, ATMs, games, telephone switches, etc.
  - Embedded: cars, hotel doors, printers, VCRs, industrial machinery, medical equipment, etc.
- Distinguishing Characteristics
  - Speed
  - Cost
  - Ease of use, software support & interface
  - Scalability
  - Reliability

The “Gap”

Implementation



Computational Problem

Overcoming the Gap

Problems

Algorithms

Language (Program) — *Programmable*

Machine (ISA) Architecture — ***Computer Specific***

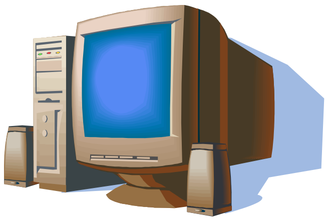
Microarchitecture — *Manufacturer Specific*

Circuits

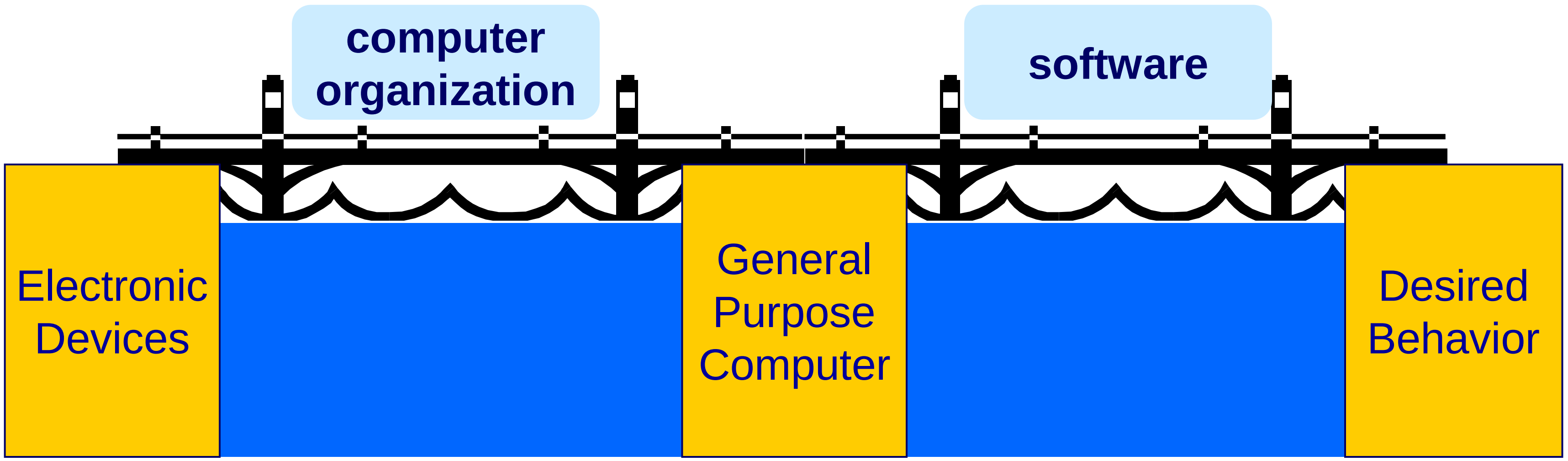
Devices

As we move down, many choices must be made.

This is how we make our living!

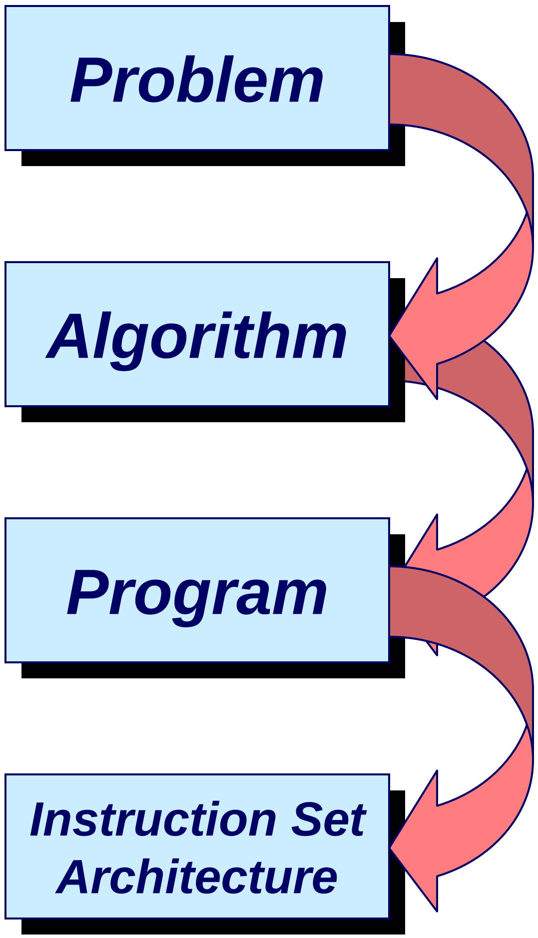


Role of General Purpose Computers



A general purpose computer is like an island that helps span the gap between the desired behavior (application) and the basic building blocks (electronic devices).

How do we solve a problem using a computer?

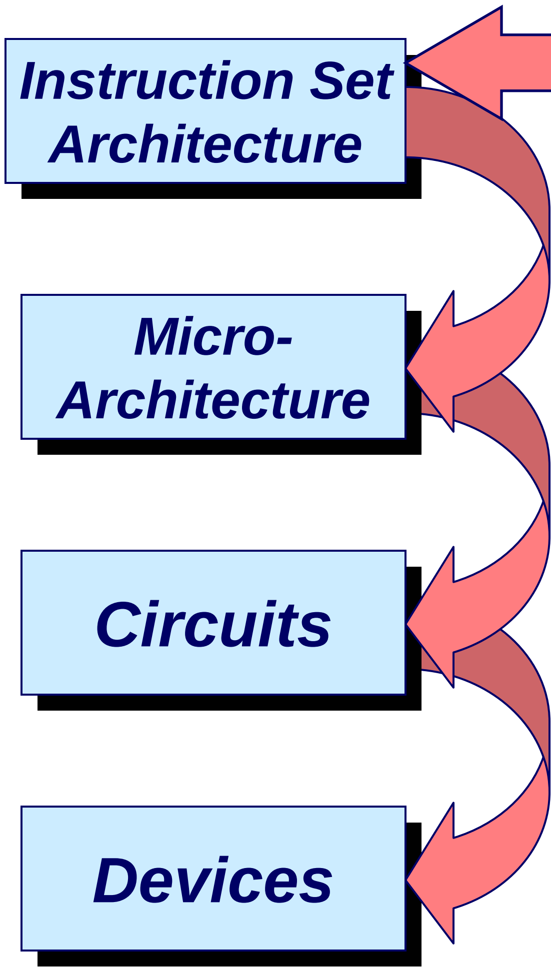


Software Design:
choose algorithms and data structures

Programming:
use language to express design

Compiling/Interpreting:
convert language to
machine instructions

Deeper and Deeper…



Instruction Set Design:
Design ISA that enables efficient problem solving

Processor Design:
choose structures to implement ISA

Logic/Circuit Design:
gates and low-level circuits to implement components

Process Engineering & Fabrication:
develop and manufacture lowest-level components

Which levels are programmable?

Problems

Algorithms

Language (Program)

Programmable

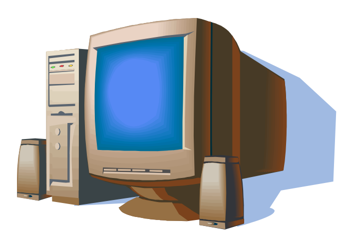
Fixed

Machine (ISA) Architecture

Microarchitecture

Circuits

Devices



Problems

- Why not use *natural languages* to program computers?
  - Incomplete

    Missing words and/or word structures for computer procedures.
  - Imprecise

    Too many words meaning the same thing that are difficult to translate into computer instructions.
  - Ambiguous – the most unacceptable attribute!

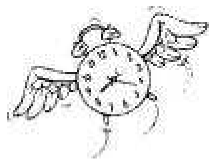
    To infer the meaning of a sentence, a listener is often helped by the tone of the speaker, or at the very least, the context of the sentence.

For example…

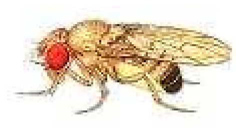
- Consider the English sentence,

  “Time flies like an arrow.”

  - One is noticing how fast time passes,



  - One is at a track meet for insects,



  - One is writing a letter to the Dear Abby of Insectville.

Problems
Algorithms
Programs
Machine (ISA) Architecture
Microarchitecture
Circuits
Devices

Algorithms

- An algorithm is a step-by-step procedure that:
  - guarantees to terminate (*finiteness*)
  - each step is precisely stated (*definiteness*)
  - each step can be carried out (*effective computability*)

- Examples
  - Starting a car
  - Computing the average of *n* integers

For any given problem, there are usually multiple algorithms that will work.

- Problems
- **Algorithms**
- Programs
- Machine (ISA) Architecture
- Microarchitecture
- Circuits
- Devices

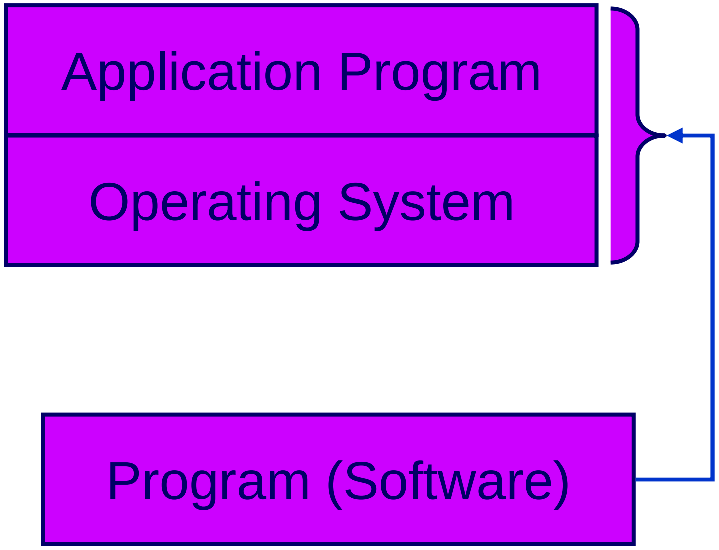
Programs

- The next step is to transform the algorithm into a computer program using a computer language.
  - computer languages communicate with the computer
  - computer languages are defined by a grammar
  - computer languages are *mechanical* rather than *natural*
  - computer languages are **not** ambiguous
- More than 1,000 programming languages
  - different languages for different purposes
    - financial processing/report generation
    - manipulating lists of symbolic data
    - natural language processing

Problems
Algorithms
Programs
Machine (ISA) Architecture
Microarchitecture
Circuits
Devices

The Program Level

- Most computers run a management program called the *operating system (OS).*



- Application programs interface to the machine architecture via the OS.

An example:

This lecture	Data
PowerPoint	Application Program
Windows XP	Operating System

High-Level Languages (HLL's)

- Machine independent
  - not dependent on a particular machine's organization
  - can be *compiled* to run anywhere
- Easier to write than low-level languages
- Usually result in higher programmer productivity
- Incorporate higher levels of abstraction
  - data structures (stacks, arrays)
  - control structures (loops, switch statements, …)

Problems
Algorithms
Programs
Machine (ISA) Architecture
Microarchitecture
Circuits
Devices

Low-Level Languages (LLL's)

- Machine-specific
  - internal machine organization is exposed to the programmer
  - they can access the nitty gritty details
    - numbers and types of registers
    - specific instructions
    - memory addressing modes
    - condition code flags, special purpose
    - hardware features
- Lower productivity but higher performing code

Problems
Algorithms
Programs
Machine (ISA) Architecture
Microarchitecture
Circuits
Devices

Levels of Transformation

Problems

Algorithms

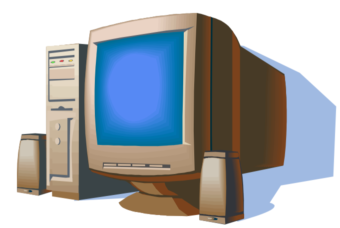
Language (Program) *Programmable*

Machine (ISA) Architecture ***Computer Specific***

Microarchitecture *Manufacturer Specific*

Circuits

Devices



Instruction Set Architecture (ISA)

- Next is to translate the computer program (language) into the instruction set of a particular computer

- Specific to a CPU
  - data types
    - What are the different representations of operands
  - operations on data
    - What functions can be done
  - addressable memory
    - Where are operands stored
  - addressing modes
    - How to find operands in memory

Machine code, or the 1's and 0's that instruct the machine.

Problems
Algorithms
Programs
Machine (ISA) Architecture
Microarchitecture
Circuits
Devices

Microarchitecture

- The microarchitecture transforms the ISA into an implementation.
  - IA-32
    - 386
    - 486
    - Pentium
    - Pentium-II, III, IV
    - Xeon

    faster and more complex
  - AMD vs. Intel vs. Transmeta Crusoe (emulation)
- *How* the operations in the ISA are implemented
  - For example, how do you add two binary numbers?
  - or, how do you access memory?

Problems
Algorithms
Programs
Machine (ISA) Architecture
Microarchitecture
Circuits
Devices

Circuits

- The next step is to implement each element of the micro-architecture with simple logic circuits.
  - Gates, adders, multiplexers
  - Flip flops, memory cells
  - Adders, subtracters, multipliers

Circuits are used to make the computer do useful things like multiply or store a result.

Problems
Algorithms
Program
Machine (ISA) Architecture
Microarchitecture
Circuits
Devices

Devices

- Finally, each basic logic circuit is implemented by a particular device technology.
  - Wires and traces
  - Types of circuits (transistors)
    - CMOS
    - NMOS
    - Gallium arsenide

Devices are the building blocks for more complex circuits.

Problems
Algorithms
Program
Machine (ISA) Architecture
Microarchitecture
Circuits
Devices

Example: Levels of Translation

Algorithm

Language

ISA

Algorithm

by hand

C-language program (HLL)

`c = a + b;`

compiler

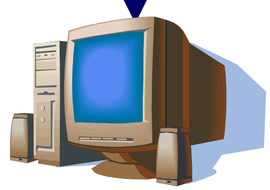
Assembly language program (LLL)

`ADD r0,r1,r2`

assembler

Machine language programs

`0010110010010001111`



program execution

Review: Descriptions of Each Level

- Problem Statement
  - Stated using "natural language“
  - Ambiguous, imprecise
- Algorithm
  - step-by-step procedure, guaranteed to finish
  - definiteness, effective computability, finiteness
- Program
  - express the algorithm using a computer language
  - high-level language, low-level language

Review of Each Level (continued…)

- Instruction Set Architecture (ISA)
  - specifies the set of instructions the computer can perform
  - data types, addressing mode
- Micro-architecture
  - detailed organization of a processor implementation
  - different implementations of a single ISA
- Logic Circuits
  - combine basic operations to realize microarchitecture
  - many different ways to implement a single function (e.g., addition)
- Devices
  - properties of materials, manufacturability