CS 271 Lecture #1

Introduction to hardware, software, and languages

Introduction to Problem-Solving Languages

- Viewed by "levels"
  - Natural languages
    - E.G.: English, Spanish, Chinese
    - Used by humans
    - Many interpretations
    - Translated to programming languages by computer programmers
  - High-level computer programming languages
    - E.G.: Java, C++, Perl, Python
    - English-like, portable to various architectures
    - Strict rules of syntax and semantics
    - Translated to lower levels by compilers / translators
  - Low-level computer programming languages
    - E.G.: Intel assembly, Mac assembly
    - Mnemonic instructions for specific computer architectures
    - Translated to machine language by assemblers
  - Machine-level computer languages
    - E.G.: Intel machine instructions, Mac machine instructions
    - Actual binary code instructions for specific architecture

Programming tools/environments for various language levels

- Natural language
  - word processors
- High-level programming languages
  - Text editor, libraries, compiler, linker, loader, debugger
  - E.G.: Eclipse, Visual C++, etc.
- Low-level programming languages
  - Text editor, libraries, assembler, linker, loader, debugger
  - E.G.: any text editor together with MASM, Visual C++, etc.
- Machine-level computer languages
  - Some way to assign machine instructions directly into computer memory
  - E.G.: set individual bits (switches), loader

Computer languages / Computer hardware viewed by "levels" (simplified)

- Level 4: Problem solution in natural language
  - Description of algorithm, solution design
  - Programmer translates to .....
- Level 3: Computer program in high-level computer programming language
  - Source code (machine independent)
  - Compiler translates to ...
- Level 2: Program in assembly language
  - Machine specific commands to control hardware components
  - Assembler translates to ...
- Level 1: Program in machine code
  - Object code (binary)
  - Linker / loader sets up ...
- Level 0: Actual computer hardware
  - Program in electronic form

Assembly Language

- In this course …
  - Skip the “high-level language” level
  - Write programs in assembly language
  - Expand levels 1 and 0
    - Understand what happens inside the computer
  - Use an assembly language to understand a specific architecture
  - Concepts transfer to other architectures

Relationship: Assembly Language ⇔ Hardware

- Assembly language is hardware-specific
  - Assembler required to convert assembly language to machine language
- Assembly language allows direct control of hardware components
  - Most high-level languages do not
  - Provides a good way to really understand architecture and how the hardware works.

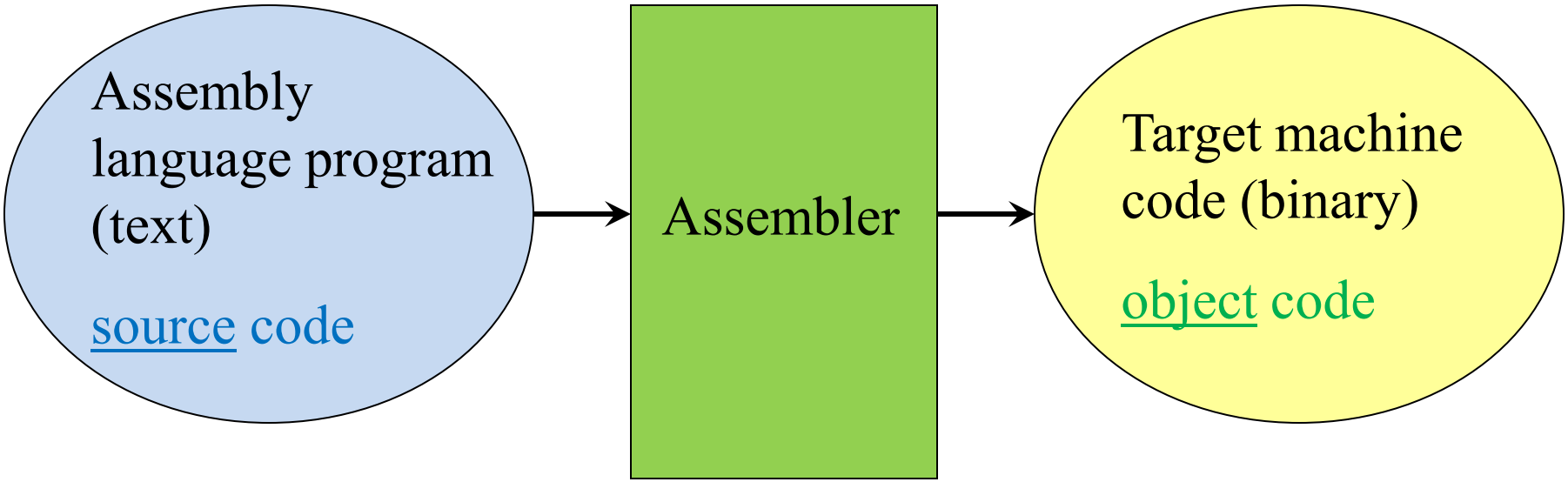
Assembly language

Assembly language provides:

1. Set of mnemonics for machine instructions
   - Opcodes and addressing modes
2. Mechanism for naming memory addresses and other constants.
   - **Note**: a named memory address is usually called a "variable“
3. Other "conveniences" for developing source code for a particular machine architecture

Assembler and assembly

An assembler is a software system that takes assembly language as input and produces machine language as output



Operating Systems

- Operating systems provide interfaces among users, programs, and devices (including the host computer itself).
- Implemented for specific architecture (in the host computer's machine language).

Low-level programming

- Level 2: Program in assembly language
  - Assembler translates to ...
- Level 1: Program in machine code
  - Operating system does partial translation
  - The hardware's instruction set architecture (ISA) provides a micro-program for each machine instruction (CISC*) or direct execution (RISC*)
- Level 0: Actual computer hardware
  - Digital logic (circuits)
  - Micro-architecture circuits control computer components

*More later on CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer)

Relationship: Instruction Set ⇔ Architecture

- A computer's instruction set is defined by the computer's architecture.
  - I.E.: each computer architecture has its own machine language.
    - E.G.: Sun machine instructions will not work on an Intel architecture.

- Cross-assemblers (software) can be used to convert a machine language to another machine language.
- Virtual machines (software) can be used to simulate another computer's architecture.

Relationship: Architecture ⇔ Software

- Hardware: Physical devices
  - E.G.: circuits, chips, disk drives, printers, etc.
- Software: Instructions that control hardware
  - E.G.: games, word processors, compilers, operating systems, etc.
- Sometimes the line between hardware and software is not clear
  - E.G.: Parts of an operating system might be implemented in hardware
- Anything that can be implemented in software could be implemented in hardware … and it would execute much faster
  - Discussion question #1:
    - If it's so much faster, why isn't everything implemented in hardware ?

System Architectures

- Super-computer
- Mainframe
- Multiprocessor/Parallel (multi-core)
- Server
- Distributed (Collection of Workstations)
  - Network
- Personal computer
  - Desktop, laptop, netbook, etc.
- Micro-controller (Real-time/Embedded system)
  - Phone, car, appliance, watch, etc.
- etc.

Two architecture development tracks

- Build more powerful machines
  - Multi-core, etc.
- Build same machine smaller/cheaper
  - Nanotech
    - Discussion question #2:
      - Try to explain: How big is a nanometer?
- Moore's Law
  - Look it up
  - Discussion question #3:
    - What does Moore's Law mean ?

Why use assembly language?

- Easier than machine code
- Access to all features of target machine
- Performance (maybe)
- Using mixed languages

- Note that assembly language tends to evolve toward a high-level language
  - advanced features (“auto” loop control, etc.)
  - libraries

Common uses of assembly language

- Embedded systems
  - Efficiency is critical
- Real-time applications
  - Timing is critical
- Interactive games
  - Speed is critical
- Low-level tasks
  - Direct control is critical
- Device drivers
  - Direct control is critical

Lecture #1: Main concepts

- Hardware / Software
- Languages (high-level, assembly, machine)
- How statements are translated from higher to lower levels
- Variety of architectures
  - Each has its own instruction set
- Applications of assembly language

- Do the “self-check” exercise
- Participate in “discussions”

- Next topic: How hardware works
- See also: Setting up
  - Visual C++ (or Visual Studio)
  - MASM
  - Irvine’s library