Chapter 6: The Virtual Machine

Wen-Hung Liao, 12/28/2001
Introduction

- The computer model, known as Von Neumann machine described previously is capable of executing programs written in machine language.
- However, it lacks “support tools” to make the problem-solving task easy.
- Humorously called a naked machine: hardware without help user-oriented features.
Naked Machines

- Need to write the program in 0s and 1s.
- Need to represented data in binary form.
- Need to manually store programs into memory before executing.
- Need to instruct the program to start running...
User Interface

To make a Von Neumann computer usable, we must create a user-interface between user and hardware.

The interface would do things such as:

- Hide from the user messy and unnecessary details of the underlying hardware.
- Present information about what is happening in a way that does not require in-depth knowledge of the internal structure of the hardware.
- Allow easy user access to the resources available on this computer.
- Prevent accidental or intentional damage to hardware, program and data.
System Software

A collection of computer programs that manage the resources of a computer and facilitate access to those resources.
Types of System Software

- Operating system, might contain:
  - Language translators: assemblers, compilers.
  - Memory managers
  - File system
  - Scheduler
  - Utilities
Writing a program

1. Use a text editor to create a program $P$ written in high-level language.

2. Use the file system to store program $P$ on the hard disk.

3. Use a language translator to translate program $P$ from a high-level language to a machine language program $M$.

4. Use loader to allocate sufficient memory to hold program and load its instructions into memory.

5. Use the scheduler to schedule and run $M$.

6. Use a file system to store the output of program $M$ into data file $D$.

7. If the program did not complete successfully, use a debugger to help locate the error.
What we will study

- Assemblers and assembly language
- Operating system (next lecture)
Machine Language

- Designed from a machine’s point of view, complicated and difficult to understand:
  - Uses binary: no English-like words, mathematical symbols
  - Allows only numeric memory addresses.
  - Difficult to change.
  - Difficult to create data.
Assembly Language

- Second-generation language
- More properly viewed as low-level programming language.
- Contrast with languages like BASIC, C, C++, Java, which are high-level programming languages.
- Refer to Figure 6.3: the continuum of programming language
Language Translators

Translation of assembly language into machine language (object program) is done using an assembler.

Translators for high-level programming language are called compilers.
Advantages of Assembly Language

- Use of symbolic operation codes rather than numeric ones.
- Use of symbolic memory addresses rather than numeric ones.
- Pseudo-operations that provide useful user-oriented services such as data generation.
Assembly Language Format

Format:

- **label**: op code mnemonic  address field  --comment

- **Comment** is ignored during translation and execution.

- **Op code mnemonic** specifies the type of operation (see page 6.5)

- Can use symbolic addresses instead of binary addresses (label, address field) to enhance program clarity and maintainability.
Data Generation using pseudo-op

- A pseudo-op invokes a service of the assembler.
- Can be used to generate data:
  - DATA +1
- Can be used for program construction:
  - BEGIN
  - END
Example of assembly language code

LOAD    ONE
STORE   I
.
.
INCREMENT I
.
.
I:    .DATA 0
ONE:  .DATA 1
More Examples

- Conditional operation (p.249)
- Looping (p.250-251)
- Compute sum of numbers (p.253)
Translation and Loading

- The job of an assembler is to translate a symbolic assembly language program into machine language.
- The task of a loader is to read instructions from the object file and store them into memory for execution.
Assembler

- Convert symbolic op codes to binary
- Convert symbolic addresses to binary
- Perform the assembler services requested by the pseudo-op
- Put the translated instructions into a file for future use.
Op Code Table

- An alphabetized list of all legal assembly language op codes and their binary equivalents.
- Use binary search to find correspondence.
Symbolic Addresses Conversion

Translation is a two-pass process:

- First pass: Build the symbol table in the first pass (Figure 6.10, 6.11)
- Second pass: translate the source program into machine language. (plus handle data generation, produce object file...). See Figure 6.12.