CSCI170 Lecture 1:
Analysis of Programming Languages

John Magee
1 September 2011
Some material copyright Jones and Bartlett
Overview/Questions

– How can we control the computer’s circuits?
– How does the von Neumann Computer work?
– Programming in machine language code is a lot more fun than wiring up the basic logic gates.
– How could we use abstraction to make it easier to translate algorithms into computer instructions?
– What is a programming language?
Early History of Computing

Joseph Jacquard (1801)
Jacquard’s Loom, the punched card
Programmability

What tricks does your computer do?

– Web browsing, email, instant messenger
– Play games
– Watch movies, organize photos
– Word processing, spreadsheets, database

Programmability is the ability to give a general-purpose computer *instructions* so that it can perform new tasks.
Harvard Mark I
Early Digital Computers

Harvard Mark I (1944)
First fully automatic digital computer to be completed
• 51 feet wide, 8 feet high, 2 feet deep
• Built out of switches, relays, and rotating mechanical shafts/clutches
• Storage for 72 numbers, each 23 decimal digits in length
• Read instructions from paper tape, one at a time
Early Digital Computers

ENIAC (1946)
Electronic Numerical Integrator And Computer
• first general-purpose electronic computer
• 80 feet wide, 8.5 feet high, 3 feet deep
• No moving parts
• Ability to conditional branch – do the next operation based on the result of the previous operation.
• 17,468 vacuum tubes, 7,200 crystal diodes, 1,500 relays, 70,000 resistors, 10,000 capacitors and around 5 million hand-soldered joints.
• 5,000 Hz, 324 multiplications/second
First Computer Bug

Log of first computer bug, discovered by Grace Hopper, 1945
A Computer Science Pioneer, she later wrote the first Compiler.
Architecture

• Harvard Architecture:
  – Instructions separate from data.

• Von Neumann Architecture:
  – Instructions = data.
  – All stored in the same memory.
  – A program can write a program! (Compilers)
  – All modern computers based on this.
Von Neumann Architecture

Figure 5.1  The von Neumann architecture
Central Processing Unit (CPU)

Refers to the combination of the Arithmetic/Logic Unit and the Control Unit.
Arithmetic/Logic Unit

The ALU performs basic arithmetic operations and logic operations. Examples:

- Arithmetic: addition, subtraction, multiplication
- Logical operations: AND, OR, NOT, XOR

Most modern ALUs have a small amount of special storage units called registers

- The accumulator is the “main” register
Arithmetic/Logic Unit

How addition and subtraction happen (at the level of the CPU):

Addition:
- take a value from memory
- send it to the adder
- add to the value in the accumulator

Subtraction:
- take a value from memory,
- invert the bits, send to the adder, set carry-in to 1
- add to the value in the accumulator
Arithmetic/Logic Unit

Given that:

– instructions are carried out by sending data through circuits
– circuits can be controlled by flipping switches....

A binary code can be used to control which operation the CPU will do at a given time.
Control Unit

The control unit coordinates the flow of operations and data in the computer.
– coordinates the ALU and memory

Instruction register (IR)
Contains the binary code indicating which instruction is being executed.

Program counter (PC)
Contains the address (in memory) of the next instruction to be executed.
Von Neumann Architecture

Figure 5.1 The von Neumann architecture
Flow of Information

Bus

A set of wires that connect all major components of the computer.

**Figure 5.2** Data flow through a von Neumann architecture
The Computer Clock

The computer has oscillator generates an endless rapid stream of off/on pulses (0,1,0,1...).

Each change from 0 to 1 is called a clock cycle.

- cycles per second is one measure of CPU speed
- measured in Hertz (MHz, GHz) – cycles per second

The clock cycle is used to coordinate the tasks of the various circuits.
The Fetch-Execute Cycle

**Figure 5.3** The Fetch-Execute Cycle
Fetch Execute Cycle

The previous diagram shows how it takes 4 steps to execute each instruction.

*Is cycles per second a valid measure of CPU performance?*
Stored Program Computer

Data and instructions to manipulate the data are logically the same (binary code) and can be stored in the same place (RAM).

The machine language of a particular computer is the set of binary coded instructions built into its hardware.
Machine Language

Characteristics of machine language:

– Every processor type has its own set of specific machine instructions.
– The relationship between the processor type and the instructions it can carry out is completely integrated.
– Each machine-language instruction does only one very low-level task.
A **Virtual Computer** is a hypothetical machine designed to contain the important features of a real computer that we want to illustrate.

Our examples will use the **Pep/7 virtual computer** designed by Stanley Warford that has 32 machine-language instructions.

- [http://www.rsu.edu/Faculty/PMacpherson/Programs/pep7.html](http://www.rsu.edu/Faculty/PMacpherson/Programs/pep7.html)
- [http://www.rsu.edu/faculty/PMacpherson/Programs/tutorial.htm](http://www.rsu.edu/faculty/PMacpherson/Programs/tutorial.htm)
Instruction Format

The machine language of a particular computer is the set of binary coded instructions built into its hardware.

This machine language has a 3-byte instruction format.

Figure 7.2  The Pep/7 instruction format
Instruction Format

Operation Code
Specifies which instruction is to be carried out.

Register Specifier
Specifies which register is to be used.

Addressing-mode Specifier
How to interpret the operand part of the instruction.
  - 00 for immediate (value given in operand)
  - 01 for direct (memory address given in operand)
Some Sample Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Meaning of Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>Stop execution</td>
</tr>
<tr>
<td>00001</td>
<td>Load the operand into the A register</td>
</tr>
<tr>
<td>00010</td>
<td>Store the contents of the A register into operand</td>
</tr>
<tr>
<td>00011</td>
<td>Add the operand to the A register</td>
</tr>
<tr>
<td>00100</td>
<td>Subtract the operand from the A register</td>
</tr>
<tr>
<td>11011</td>
<td>Character input to the operand</td>
</tr>
<tr>
<td>11100</td>
<td>Character output from the operand</td>
</tr>
</tbody>
</table>

**Figure 7.3** Subset of Pep/7 instructions
Sample Instructions

What does this instruction mean?

Instruction specifier

Operand specifier

"Load immediate value 7 into the accumulator"
Computer Programming

The CPU does some low-level tasks, and each one is specified as a machine-language instruction.

– How do we get the CPU to do stuff for us?

Computer programming is the process of analyzing a problem, designing a solution, and expressing that solution as a series of computer instructions.

Computer programs must be written in the language the computer understands.
### Writing a Computer Program

<table>
<thead>
<tr>
<th>Module</th>
<th>Binary Instruction</th>
<th>Hex Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write &quot;H&quot;</td>
<td>111000000 00000000001001000</td>
<td>E0 0048</td>
</tr>
<tr>
<td>Write &quot;e&quot;</td>
<td>111000000 00000000001100101</td>
<td>E0 0065</td>
</tr>
<tr>
<td>Write &quot;l&quot;</td>
<td>111000000 00000000001101100</td>
<td>E0 006C</td>
</tr>
<tr>
<td>Write &quot;l&quot;</td>
<td>111000000 00000000001101100</td>
<td>E0 006C</td>
</tr>
<tr>
<td>Write &quot;o&quot;</td>
<td>111000000 00000000001101111</td>
<td>E0 006F</td>
</tr>
<tr>
<td>Stop</td>
<td>0000000000</td>
<td>00</td>
</tr>
</tbody>
</table>
Algorithms

Algorithm

A set of unambiguous instructions for solving a problem or subproblem in a finite amount of time, and using a finite amount of data.

For a given set of inputs, the algorithm will always produce the same outputs.
Grandpa Ron’s Chocolate Kahlua Mudslides

Ingredient List:

• ½ cup vodka
• ½ cup Kahlua
• ½ cup Bailey’s Irish Cream
• ¼ cup Hershey’s chocolate syrup
• 1 blender full of ice

Process:

Fill blender with ice.
Pours other ingredients onto ice.
Blend until smooth consistency.
Pour into glasses and enjoy.
Partial Summary

– Even the modern computer is still a moronic number-crunching, data moving device.

– Algorithms describe the explicit steps the computer must follow to solve a problem.

– Pseudocode is a tool to help us express an algorithm, but it is not understood by the computer.

– Why would anyone want to write a program in machine language?
Assembly Language

Assembly language

A language that uses mnemonic codes to represent machine-language instructions

Mnemonic code examples:

– LOADA means “load value into accumulator”
– ADDA means “add value into accumulator”
– STOREA means “store value from accumulator”
The Assembly Process

**Assembler**

A program that reads each of the instructions in mnemonic form and translates it into the machine-language equivalent.
# Pep/7 Assembly Language

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand, Mode Specifier</th>
<th>Meaning of Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOP</td>
<td></td>
<td>Stop execution</td>
</tr>
<tr>
<td>LOAD A</td>
<td>h#008B,i</td>
<td>Load 008B into register A</td>
</tr>
<tr>
<td>LOAD A</td>
<td>h#008B,d</td>
<td>Load the contents of location 8B into register A</td>
</tr>
<tr>
<td>STORE A</td>
<td>h#008B,d</td>
<td>Store the contents of register A into location 8B</td>
</tr>
<tr>
<td>ADD A</td>
<td>h#008B,i</td>
<td>Add 008B to register A</td>
</tr>
<tr>
<td>ADD A</td>
<td>h#008B,d</td>
<td>Add the contents of location 8B to register A</td>
</tr>
<tr>
<td>SUB A</td>
<td>h#008B,i</td>
<td>Subtract 008B from register A</td>
</tr>
<tr>
<td>SUB A</td>
<td>h#008B,d</td>
<td>Subtract the contents of location 8B from register A</td>
</tr>
<tr>
<td>CHAR I</td>
<td>h#008B,d</td>
<td>Read a character and store it into byte 8B</td>
</tr>
<tr>
<td>CHAR 0</td>
<td>c#/B/,,i</td>
<td>Write the character B</td>
</tr>
<tr>
<td></td>
<td>h#008B,d</td>
<td>Write the character stored in byte 8B</td>
</tr>
<tr>
<td>DEC I</td>
<td>h#008B,d</td>
<td>Read a decimal number and store it into location 8B</td>
</tr>
<tr>
<td>DEC O</td>
<td>h#008B,i</td>
<td>Write the decimal number 139 (8B in hex)</td>
</tr>
<tr>
<td>DEC O</td>
<td>h#008B,d</td>
<td>Write the decimal number stored in 8B</td>
</tr>
</tbody>
</table>

**Mode specifier: i = immediate, d = direct (memory)**
## Pep/7 Assembly Language

An assembly language can have **pseudo operations** which do not map directly to the underlying machine language operations.

<table>
<thead>
<tr>
<th>Pseudo-op</th>
<th>Operand</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.ASCII</code></td>
<td><code>/.../</code></td>
<td>Store the characters between the `/s into memory</td>
</tr>
<tr>
<td><code>.BLOCK</code></td>
<td>d#3</td>
<td>Generate 3 bytes of storage and set each byte to zero</td>
</tr>
<tr>
<td><code>.WORD</code></td>
<td>d#5</td>
<td>Generate a word with the decimal value 5 stored in it</td>
</tr>
<tr>
<td><code>.WORD</code></td>
<td>h#0105</td>
<td>Generate a word with the hexadecimal value 5 stored in it</td>
</tr>
<tr>
<td><code>.END</code></td>
<td></td>
<td>Signals the end of the assembly language program</td>
</tr>
</tbody>
</table>
Example: Pseudo Code Program

**Reading and adding three numbers**

Set sum to 0
Read num1
Add num1 to sum
Read num2
Add num2 to sum
Read num3
Add num3 to sum
Write sum
Example: Assembly Program

BR Main ;branch to location Main
sum: .WORD d#0 ;set up word with zero as the contents
num1: .BLOCK d#2 ;set up a two byte block for num1
num2: .BLOCK d#2 ;set up a two byte block for num2
num3: .BLOCK d#2 ;set up a two byte block for num3
Main: LOADA sum,d ;load a copy of sum into accumulator
DECI num1,d ;read and store a decimal number in num1
ADDA num1,d ;add the contents of num1 to accumulator
DECI num2,d ;read and store a decimal number in num2
ADDA num2,d ;add the contents of num2 to accumulator
DECI num3,d ;read and store a decimal number in num3
ADDA num3,d ;add the contents of num3 to accumulator
STOREA sum,d ;store contents of the accumulator into sum
DECO sum,d ;output the contents of sum
STOP ;stop the processing
.END ;end of the program
Partial Summary

– Assembly language: much easier to write/read than machine language!
– Why not just use assembly language instead of machine language?
– Who wrote the assembler program, and what language was it written in?
– Why not just write programs in pseudo code, and have a program translate that into machine language instructions?
High-level Programming Languages

High-level language
A language that provides a richer (more English like) set of instructions.

Example:
```python
a = 2
b = 5
total = a + b
print total
```

*How can the computer understand this kind of language?*
Source Code and Translation

A high-level language program is written in a plain-text format called source code.

There are 2 ways to translate source code into the machine-level instructions:

– Typically, a high-level language is either compiled OR interpreted.
A **compiler** is a program that translates a high-level language program into machine code.

The “compile step” is done all at once, in advance of running the program, usually by the programmer who wrote the high-level source code.

**Figure 8.1** Compilation process
Most Popular Compiled Languages

Fortran, 1952
- IBM, Numeric and Scientific Computing
- Still in use in biology dept at BU!

Pascal, 1970
- Developed for teaching students structured Programming
- Early Macintosh operating system, Apple Lisa written in Pascal

C, 1972
- Ritchie and Kernigan, AT&T Bell Laboratories
- Developed for system software, UNIX

C++, 1979
- Bjarne Stroustrup, AT&T Bell Laboratories
- Backwards compatible with C, and objects

ADA, 1983
- Department of Defense
- 1970s: concern that DoD had too many programming languages
- By 1996: down to only 37

C and C++ together have been the most popular compiled languages with the most programmers and applications.
Interpreters

An interpreter is a program that translates and executes the program’s source code statements in sequence.

– An interpreter translates a statement and then immediately executes the statement.**

– Interpreters can be viewed as simulators.

** Whereas an assembler or compiler produces machine code as output; executing the code is a separate step.
Most Popular Interpreted Languages

BASIC, 1963
- Beginner’s All-purposes Special Instruction Code
- Kemeny and Kurtz, Dartmouth College
- Invented for non-science and non-mathematics users.

Perl, 1987
- Invented by Larry Wall of Unisys Corporation
- Regular Expression engine for text processing

Visual Basic/Visual Basic .NET, 1991
- Invented at Microsoft, a GUI scripting language
- Extremely popular; huge installed business application base

Python, 1991
- Guido van Rossum, researcher at BDFL in the Netherlands
- Minimalist semantics
- Named after Monty Python
Portability

Portability is the ability of a program written on one machine (CPU) to be run on different machines.

Compiler portability
A program in a standardized language can be recompiled to run on any machine that has the appropriate compiler. (This takes time and skill.)

Source code portability
A program in an interpreted language can be distributed and run (interpreted) directly on any machine that has the appropriate interpreter.
Compiled vs. Interpreted

Compiled Languages

– Speed: tend to run faster than interpreted languages.

– Portability: virtually none. To port to new CPU architecture, must re-compile the source code and distribute new executable file.

Interpreted Languages

– Speed: tend to run more slowly than compiled languages.

– Portability: virtually seamless. Distribute source code directly, and it will run on any suitable interpreter.
Figure 1.1 A programming language timeline
Abstractions in Programming Languages

• Two types of programming language abstractions:
  – Data abstraction
  – Control abstraction

• **Data abstractions**: simplify the behavior and attributes of data for humans
  – Examples: numbers, character strings, search trees

• **Control abstractions**: simplify properties of the transfer of control
  – Examples: loops, conditional statements, procedure calls
Abstractions in Programming Languages (cont’d.)

• Abstractions can also be categorized by levels (measures of the amount of information contained or hidden in the abstraction)

• **Basic abstractions**: collect the most localized machine information

• **Structured abstractions**: collect intermediate information about the structure of a program

• **Unit abstractions**: collect large-scale information in a program
Control: Basic Abstractions

- **Basic control abstractions**: statements that combine a few machine instructions into an abstract statement that is easier to understand

- **Syntactic sugar**: a mechanism that allows you to replace a complex notation with a simpler, shorthand notation
  
  - Example: $x += 10$ instead of $x = x + 10$
Control: Structured Abstractions

• **Structured control abstractions**: divide a program into groups of instructions nested within tests that govern their execution
  – Help to express the logic of primary control structures of sequencing, selection, and iteration

• **Branch instructions**: instructions that support selection and iteration to memory locations other than the next one
Computational Paradigms

• **Imperative language**: a language with three properties
  – Sequential execution of instructions
  – Use of variables representing memory locations
  – Use of assignment to change the values of variables

• Represents one **paradigm** (pattern) for programming languages

• **von Neumann bottleneck**: requirement that a program be described as a sequence of instructions
• **Functional paradigm:**
  – Based on the abstract notion of a function in lambda calculus

• **Logic paradigm:**
  – Based on symbolic logic

• Both functional and logic paradigms correspond to mathematical foundations
  – Makes it easier to determine if a program will execute correctly
Computational Paradigms (cont’d.)

• **Object-oriented paradigm:**
  – Reusable code that operates in a way to mimic behaviors of real-world objects
Take-Away Points

- Von Neumann Architecture
- CPU: Arithmetic/Logic Unit + Control Unit
- Machine language / Assembly language
- High-level languages
  - Compiled vs. translated
- Abstraction
  - Data
  - Control
- Paradigms
  - Imperative
  - Functional
  - Logical
  - Object-Oriented Programming (OOP)
Student ToDos

• Visit the course webpage.
• Do a “Hello World” in C
  – Instructions to be posted on web.
• Readings:
  – Louden and Lambert, Chapter 1