Introduction

Lecture 1
CS 212 – Fall 2007

Mundane Details

- Staff
  - Instructor: Paul Chew
  - Course Administrator: Kelly Patwell
  - TAs: Etan Bukiet, Jeff Chadwick, Zoe Chiang, Jimmy Hartzell, Anthony Jawad, Ken Kruger, Cangming (Geoff) Liu, Dan Perelman, Chuck Sakoda, Ozzie Smith
  - Consultants: none (but the 211 consultants can help with general Java questions)
- Text
  - None required, but some that might be helpful are listed on the website
- Lecture
  - W 3:35 – 4:25, Hollister B14
- Sections (beginning Sept 3)
  - Monday, 12:20 - 1:10 in Hollister 306
  - Monday, 7:30 - 8:20 in Upson 205
  - Wednesday, 7:30 - 8:20 in Upson 205
- Website:
  - cs.cornell.edu/courses/212/
- Software (see CS 211 website)
  - JDK (Java Development Kit) 5 or JDK 6
  - IDE (Interactive Development Environment): DrJava or Eclipse are recommended

Announcements

- Sections start this next week (beginning Sept 3)
- We use CMS (Course Management System) for maintaining grade information
  - Make sure you're on CMS
  - Notify the course administrator (see website) if you're not
- The first assignment (Part 1) will appear on the website later this week

The Course

- Description

  "A project course that introduces students to the ways of software engineering using the Java programming language. The course requires the design and implementation of several large programs."
- Objectives
  - Improve your programming skills
  - Learn something about software engineering
    - Top-down and bottom-up design
    - Software reuse
    - Abstraction
    - Testing
  - Develop project management skills
  - Learn about computer science

When to Take CS212

- At same time as CS211
  - Coordination of topics
  - Coordination of assignment due dates
- After CS211
  - You'll have more experience
  - But possibly less connection with *your* CS211
- Before CS211
  - No!

Course Topics

- Introduction, computer architecture, JVM
- Compilers, syntax, context-free grammars
- Recursive descent parsing, abstract syntax trees (ASTs)
- Programming in a group
- Software engineering
- Software tools
- Software testing
- Programming languages
- Runtime stack, implementing functions
- Recursion
- Pointers, the heap
- Implementing objects

- No exams
- But there is a Project

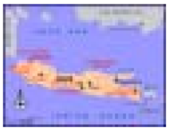
The Project

- Build a compiler for a Java-like language called Bali

An island of southern Indonesia in the Lesser Sundas just east of Java

- Compiled code: sam-code
  - Resembles (sort of) Java Byte Code (JBC)
  - Runs on SaM (Stack Machine)
    - a simplified substitute for the JVM (Java Virtual Machine)

- Four parts
  - Part 1
    - Introduction to SaM, simple expressions
  - Part 2
    - Compiling expressions, control structures
  - Part 3
    - Compiling functions
  - Part 4
    - Compiling (simple) classes



Working in Groups

- Work individually on first assignment (Part 1)
- After that, partners are allowed/encouraged
  - Good practice for group-projects in later courses
  - Groups of 1, 2, or 3

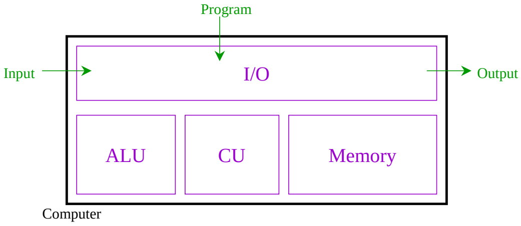
- Partnership rules
  - You choose group
  - For a given assignment, once you start with a group, you must continue
  - You may not work with different partners for different parts of the same assignment
  - Can change groups for each assignment
  - More details on course website

Computer Architecture: Memory

- A computer contains a large collection of circuits that can be used to store *bits* (a bit is a 0 or a 1)
  - Bits are grouped into *bytes* (8 bits)
  - Bytes are grouped into words or *cells*

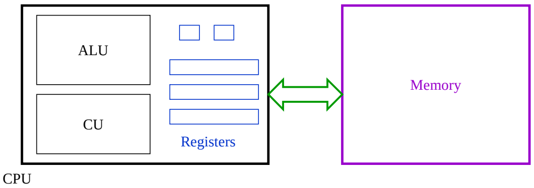
- *Memory* consists of a large collection of cells
  - Each memory cell has an *address* (usually from 0 to numCells–1)
  - Cells can be accessed in any order
  - Computer memory is called
    - *Main memory* or
    - *RAM* (Random Access Memory) or
    - (obsolete) *core memory*

Von Neumann Model



- Memory: holds both data and program
- Arithmetic Logic Unit: handles arithmetic and logic calculations
- Control Unit: interprets instructions; controls ALU, Memory, I/O
- I/O: storage, input, output

Central Processing Unit (CPU)



- Registers hold small amounts of data
  - PC: program counter
  - IR: instruction register (current instruction)
  - SP: stack pointer
  - more…

Machine Language vs. Assembly Language

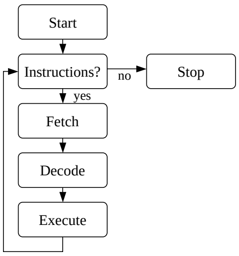
- Machine Language
  - Instructions and coding scheme used internally by computer
  - Humans do not usually write machine language
  - Typical machine language instructions have two parts
    - Op-code (operation code)
    - Operand

- Assembly Language
  - Symbolic representation of machine language
  - Use mnemonic word for op-code
    - Example: PUSHIMM 5
  - Typically provide additional features to help make code readable for humans
    - Example: names as labels instead of numbers

Machine Instruction Categories

- Data transfer
  - Copy data from one memory location to another
    - LOAD: copy data from a memory cell to a register
    - STORE: copy data from a register to a memory cell
    - I/O instructions
- Arithmetic / Logic
  - Request activity in ALU
    - Arithmetic (ADD, SUB, TIMES, ...)
    - Logic (AND, OR, NOT, XOR)
    - SHIFT, ROTATE
- Control
  - Direct execution of program
    - JUMP, JUMPC (conditional jump)

Fetch and Decode Cycle



- Control Unit (CU) fetches next instruction from memory at the address specified by Program Counter (PC)
- CU places instruction into the instruction register (IR)
- CU increments the PC to prepare for next cycle
- CU decodes instruction to see what to do
- CU activates correct circuits to execute the instruction (e.g., ALU performs an addition)

Java Byte Code (JBC)

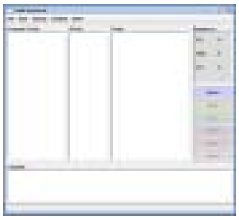
- A Java compiler creates Java Byte Code (JBC)
  - A sequence of bytes
  - Not easily readable by humans
  - JBC is machine code for a virtual (pretend) computer called the Java Virtual Machine (JVM)
  - A *byte code interpreter* reads and executes each instruction
- javap –c classfile
  - Can use this to see JBC

Java Virtual Machine (JVM)

- JBC is code for the JVM
  - No such machine really exists
  - A *JVM interpreter* must be created for each machine architecture on which JBC is to run
- The JVM is designed as an "average" computer
  - Uses features that are widely available (e.g., a stack)
- Design goals
  - Should be easy to convert Java code into JBC
  - Should be reasonably easy to create a JVM interpreter for most computer architectures

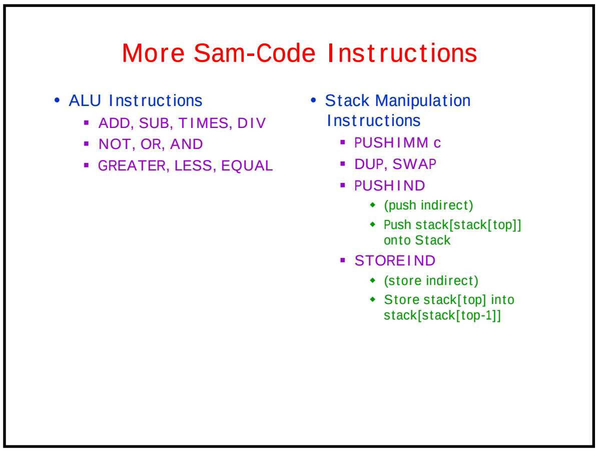
SaM (Stack Machine)

- Goals
  - Approximate the JVM
  - But simpler
- We will produce sam-code, assembly language for SaM, our own virtual machine
- We have a SaM Simulator (thanks David Levitan) that we can use to execute sam-code
- In place of
  JBC for the JVM
- We will produce
  sam-code for SaM



Some Sam-Code Instructions

- SaM's main memory is maintained as a Stack
- The SP (stack pointer) register points at the next empty position on the stack
  - The first position has address 0
  - Addresses increase as more items are pushed onto the Stack
- PUSHIMM c
  - (push immediate)
  - Push integer c onto Stack
- ADD
  - Add top two Stack items, removing those items, and pushing result onto Stack
- SUB
  - Subtract top two Stack items, removing those items, and pushing result onto Stack
  - Order is important
    - stack[top-1] – stack[top]



More Sam-Code Instructions

- ALU Instructions
  - ADD, SUB, TIMES, DIV
  - NOT, OR, AND
  - GREATER, LESS, EQUAL
- Stack Manipulation Instructions
  - PUSHIMM c
  - DUP, SWAP
  - PUSHIND
    - (push indirect)
    - Push stack[stack[top]] onto Stack
  - STOREIND
    - (store indirect)
    - Store stack[top] into stack[stack[top-1]]