CSE11 Fall 2013

Lecture 1

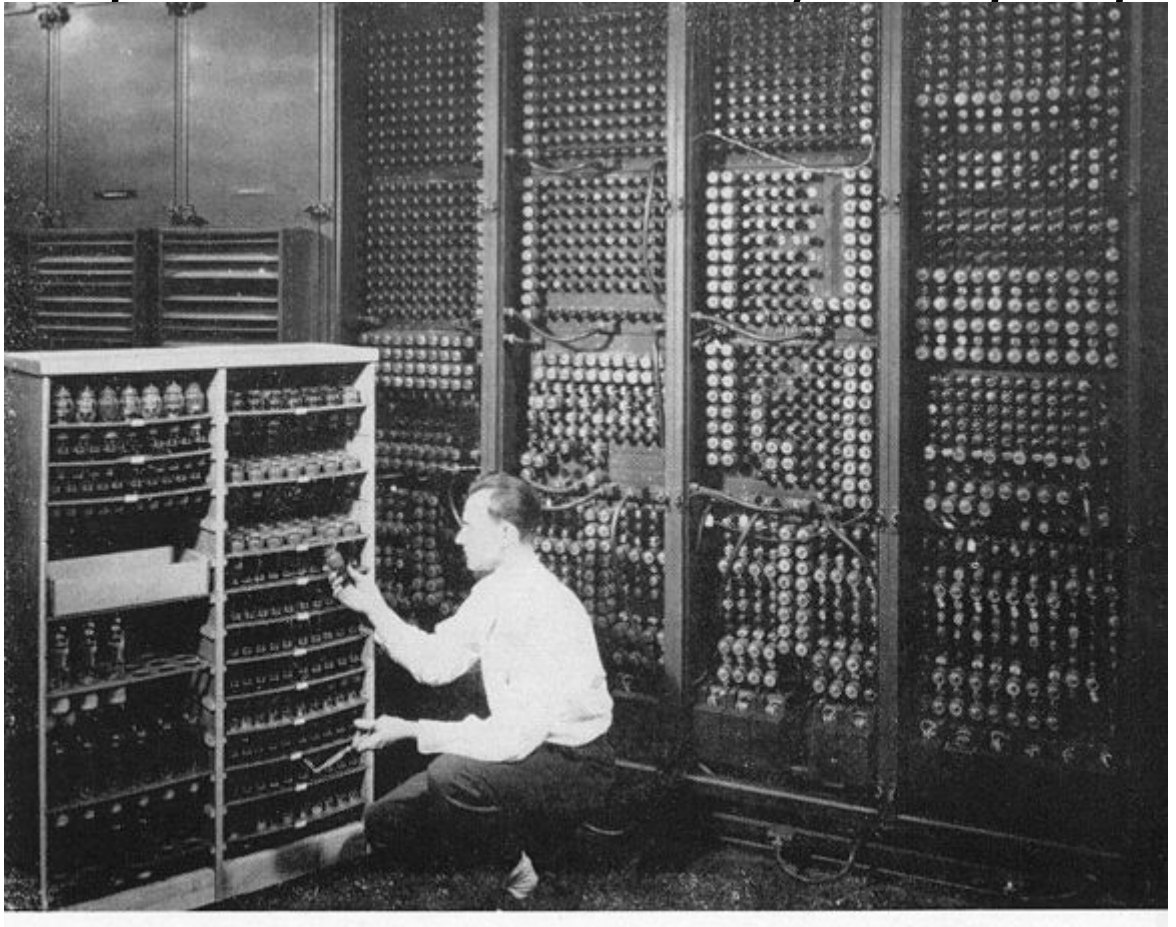
Topics Covered

- Class handout
- Is CSE11 the right class for you?

- A short history of programming languages
- Computer organization
- What is a procedural language?
- What is an object-oriented language?
- What is an object, a class, an instance

A Short History of Programming

- 1943 – The ENIAC. University of Pennsylvania
- Used for Artillery Projectile Calculations 19,000 Vacuum Tubes
- Programmed in Assembly Language (Machine Code).



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Assembly Language

- This is what CPU “understands”
- Here's an Intel x86 code snippet

```
1   pushl %ebp
2   movl %esp, %ebp
3   subl $4, %esp
4   movl $10, -4(%ebp)
5   leal -4(%ebp), %eax
6   addl $66, (%eax)
7   leave
8   ret
```

- Not readable without comments
- Specific to brand particular hardware (Intel, ARM (what's in many smartphones), PowerPC all different)
- Prone to errors. Slow to program. Essential Today.

1950s, 1960s

- 1955: FORTRAN (Formula Translation)
- 1958: LISP (List Processing)
- 1959: COBOL (Common Business Oriented Language)
- 1964: BASIC (Beginner's All Purpose Symbolic Instruction Code)
- These were critical advances in programmability of computers.
  - English-like constructions
    - Compare, Test, Jump → If
  - Compiler/Interpreters translated automatically to machine code

1970s,1980s

- 1970 – Pascal
  - UCSD Pascal, and the p-System in 1978 made Pascal portable
- 1972 – C
  - This made the UNIX operating System practical
  - Most of Linux kernel is in C
- 1978 – SQL (Structured Query Language)
  - Program databases
- 1980 - C++ (C with “classes”)
- 1984 – MATLAB (Matrix Linear Algebra)
- 1987 – PERL (Practical Extraction and Reporting Language)
- Proliferation of Specialized Languages, Greatly improved the ease with which complex algorithms could be expressed.

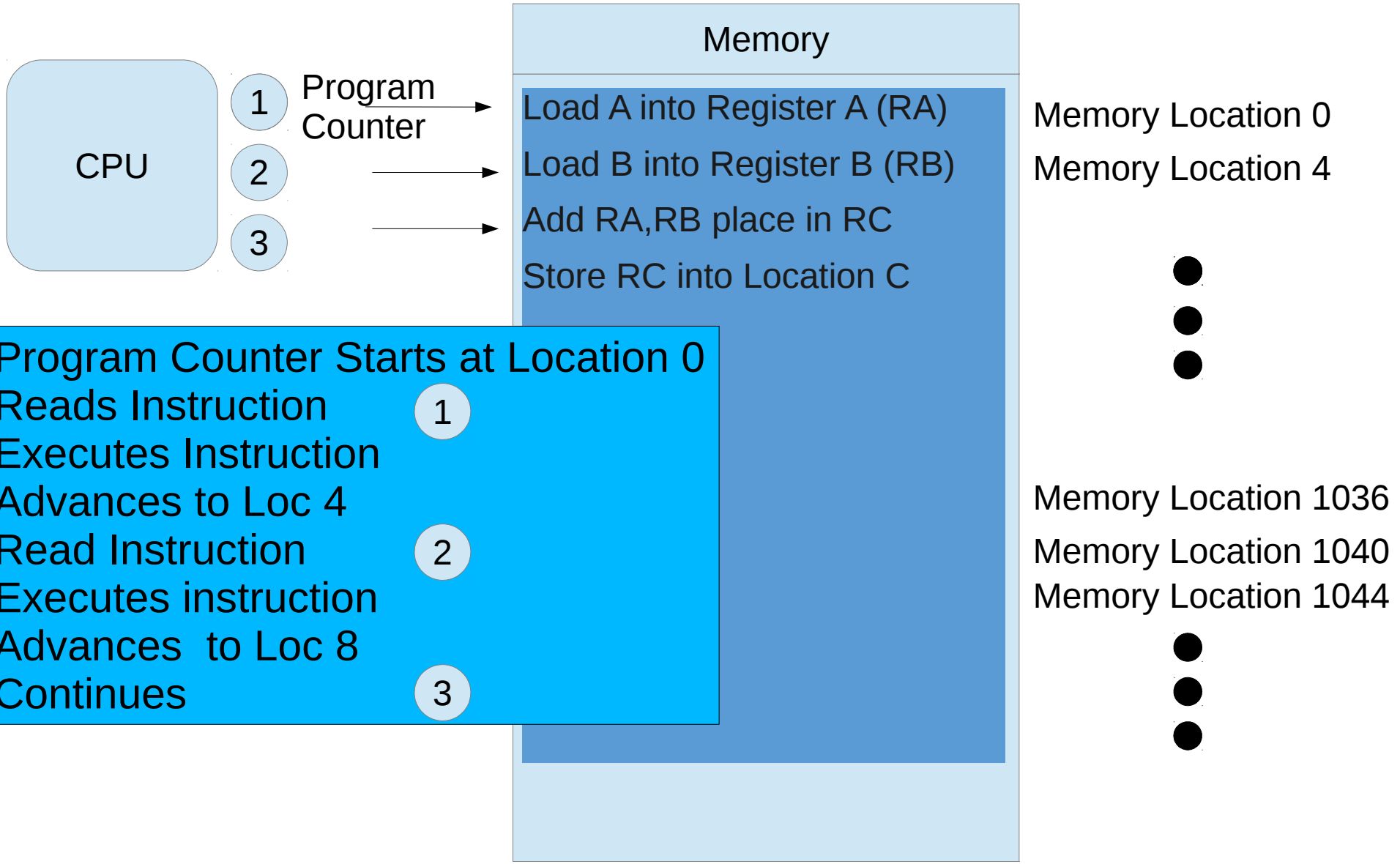
1990s

- 1991 – Python, Visual Basic
- 1991 – HTML (Hypertext Markup language)
  - Websites
- 1995 – Java
  - Rapid Application Development
  - Object Oriented
  - Loosely based upon C
  - Simplified inheritance versus C++
  - Portable (more on this later)
- Programmers uses the Language that is most suitable the problem at hand.
- Even “Old” languages persist. e.g., FORTRAN is used on all modern supercomputers. C is critical for *NIX operating systems

Computer Organization

- Basic Computer operation is very straightforward
- CPU – Central Processing Unit
- Memory
  - Instructions
  - Data
- Input/Output Devices (files, display)

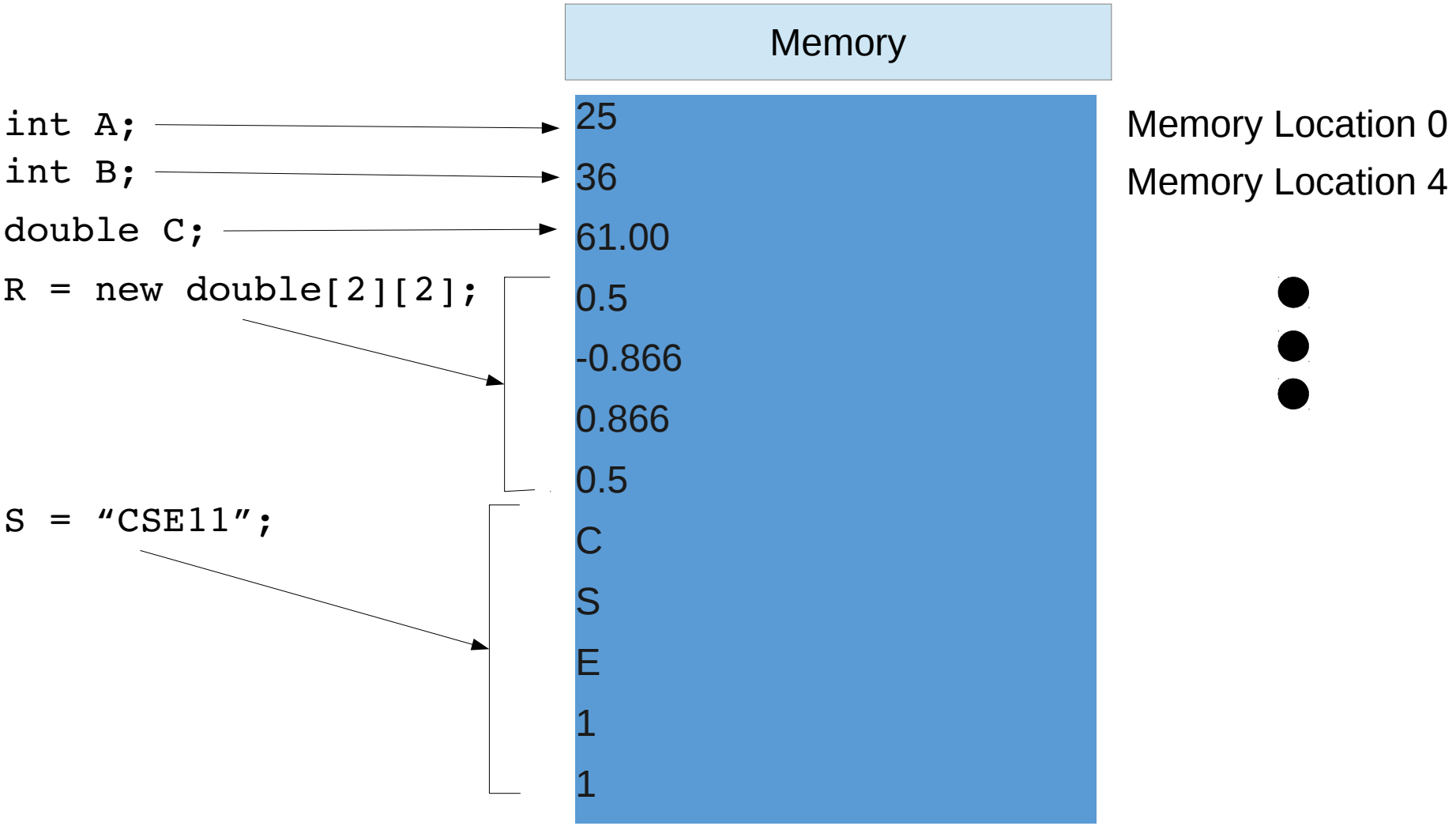
The basic CPU “Loop“



What are the important items

- Memory is Linear. Starts at 0
- Instructions (what the CPU is told to do) and Data (what it is supposed to operate on) are both in memory
  - Memory can either be CPU instructions OR Data
  - The compiler usually separates the memory into two logical pieces
    - Memory for instructions (sometimes called the code segment)
    - Memory for data (sometimes called the memory segment)
- Assembly language enables the programmer to deal directly with the Hardware (code and data)
- Eventually, what we program must end up into this structure. This is the only format that the Computer understands
- This basic structure (and understanding it) can help you “demystify” what is going on when you program

Memory that Stores Data



Memory is Universal Storage.
What is stored on Location N is different for every program

Abstraction of Memory

- Data is stored in a linear set of memory locations on the computer itself
- Languages let us declare the names and types of variables that we want to use
- The compiler and runtime systems keeps track of exactly where in memory a variable is located
  - Nothing in the computer itself protects us from storing a double in location N and then reading it back as if it were an integer
- The same ideas apply to the “code” part of the program.

Programming Languages

- Convert high-level, human-understandable instructions into a form that the computer can execute
- Different kinds of languages make it easier (or harder) to express these high-level instructions

What came before Object-Oriented Programming?

- FORTRAN introduced in 1955
- 25 years later, the first popular object-oriented language appeared (C++)
- The basic abstraction is called *procedural* programming
  - It is the bedrock of computer programming and you use elements of it all the time (even in object oriented programming)

What is procedural programming?

- A program is organized as data and a set of procedures (also called subprograms).
- For the program to do its job, the procedures are called in the correct order
- This very much mirrors how data and code are defined/organized in the computer

```
#include <stdio.h>
int square(int iA) { return iA * iA; }
int cube (int iA) { return iA * iA * iA; }
void main() {
   int A, squaredA, sixthA;
   A = 3;
   // Calculate A^6
   squaredA = square(A);
   sixthA = cube(squaredA);
   printf("%d\n", sixthA);
}
```

What are some of the “complaints” of procedural programming

- Code was separated from the actual data
- Suppose you have
  - Integer A
  - 2 x 2 Matrix B
- You use one procedure to square an integer A (it's just one multiplication)
- You use a different procedure to square matrix B (8 multiplies, 4 additions)
- In procedural programming, the author must explicitly track whether he/she is squaring a number or a matrix and the call the right piece of code to get the job done

Object-Oriented Programming

- Revisit integer A and Matrix B
- Suppose A and B knew “how to square themselves”?
- Instead of the programmer explicitly figuring out which “squaring subprogram to call”, he/she would rely on these variables (objects) to know how to perform the operation properly on themselves.
- This would be called the squaring method

Function Calls vs Method Invocations

- code: `B = square(A)`
  - A *function* called “square” has an *argument* A.
  - Whatever square does, it does it on A (the argument)
  - Whatever square does, it returns something and stores it in B
  - Square is a *procedure* or *function*

- code: `B = A.square()`
  - An object called “A” has *method* called square
  - Whatever square does, it does it to A.
    - If A and B are the same kind of objects, then B.square() is valid
  - Whatever square does, it returns and object and stores it in B
  - Square is called a *method*

An “object” version of A^6

```
public class example1
{
private int myvalue;

        public example1(int iA) { myvalue = iA; } //constructor
        private int square() { return myvalue * myvalue; }
        private int cube () { return myvalue * myvalue * myvalue; }

        public static void main(String[] args) {
                example1 A, squaredA;
                A = new example1(3);
                squaredA = new example1(A.square());
                 // Calculate and print out A^6
                System.out.println(squaredA.cube());
        }

}
```

Objects, Classes, Methods, Instances

- An object is a software construction that has both *state* and *behavior*
  - *State* is the data required to define the object
  - *Behavior* are the *methods* or procedures that can be used to manipulate the state
- Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication.
- Hiding internal state and requiring all interaction to be performed through an object's methods is known as data encapsulation — a fundamental principle of object-oriented programming

Objects

- State is memory in the computer
- Methods are procedures
- We logically link these two together to create an object
- Shorthand: Object = Data + Methods

Objects, Classes, Methods, Instances

- A class is *blueprint* from which individual objects are created
- Suppose we have a bicycle class
  - The internal state that might be used to define a particular bike in the bicycle class are
    - Color, wheel size, seat height, number of gears
  - Methods that are used to control the bike
    - pedal, brake, selectGear, turnRight, turnLeft, goStraight
-

Objects, Classes, Methods, Instances

- An *instance* of a class is a specific object with state.
- eg. The following would be two different instances of the bicycle class.
  - `RedBike = new Bicycle(RED,27);`
  - `BlueBike = new Bicycle(BLUE,24);`
- RedBike has color red and 27” wheels
- BlueBike has color blue and 24” wheels.
- Tell RedBike to brake and blueBike to turnLeft
  - `RedBike.brake()`
  - `BlueBike.turnLeft()`
-