Topics for today:

Overview of pipelining

Extended aside: What's the big deal about Java, anyway?

Pipelining

Pipelining is a method which carries out steps in the fetch/decode/execute cycle for several different (macro)instructions simultaneously.

Pipeline Hazards

• Structural hazards
• Data hazards
• Control hazards

Jump penalty

The biggest problem with pipelining is that when a jump is executed, instructions already in the pipeline are rendered irrelevant and a new instruction must be fetched. This is called *flushing the pipeline* or *squashing*. The cycles lost while fetching the new instruction are the *jump penalty*.

Suppose an instruction takes four cycles to execute in a nonpipelined CPU: one cycle to fetch the instruction, one cycle to decode the instruction, one cycle to perform the ALU operation, and one cycle to store the result. In a CPU with a 4-stage pipeline, that instruction still takes four cycles to execute, so how can we say the pipeline speeds up the execution of the program?

Java

Java is a high-level programming language, similar to C++ (and many other languages: C, Pascal, Algol, …).

It was designed by researchers at Sun Microsystems in the mid-1990s for use with the Internet.

```
static void computeAverage(double[] list, int arraySize )
      {
              double sum;
              int n;
              double average;
              sum = 0.0;
              for (n = 0; n <= arraySize-1; n++)
                      sum = sum + list[n];
              average = sum / arraySize;

              scr.putln("The total is " + sum);
              scr.putln("The average is " + average);
      }
```

Recall

Typically a program is written in a high-level language (<u>source</u> <u>code</u>) and is translated by software into machine language (<u>object</u> <u>code</u>) for a particular processor – or more precisely, for a particular instruction set architecture.

There are two main ways of doing this translation:

Compilation – A *compiler* translates an entire program to machine language all at once. The program can be run *after* it is compiled.

Interpretation – An *interpreter* translates a program one line at a time, and then executes the instruction(s) before moving on to the next line.

Note

If you want to distribute a program (for free or for sale) without sharing the source code, the program must be compiled.

BUT– *how do you compile a program when you don't know what machine is going to run it?*