CSCU9A1: Systems 3

Leslie Smith

Last updated Oct 2016

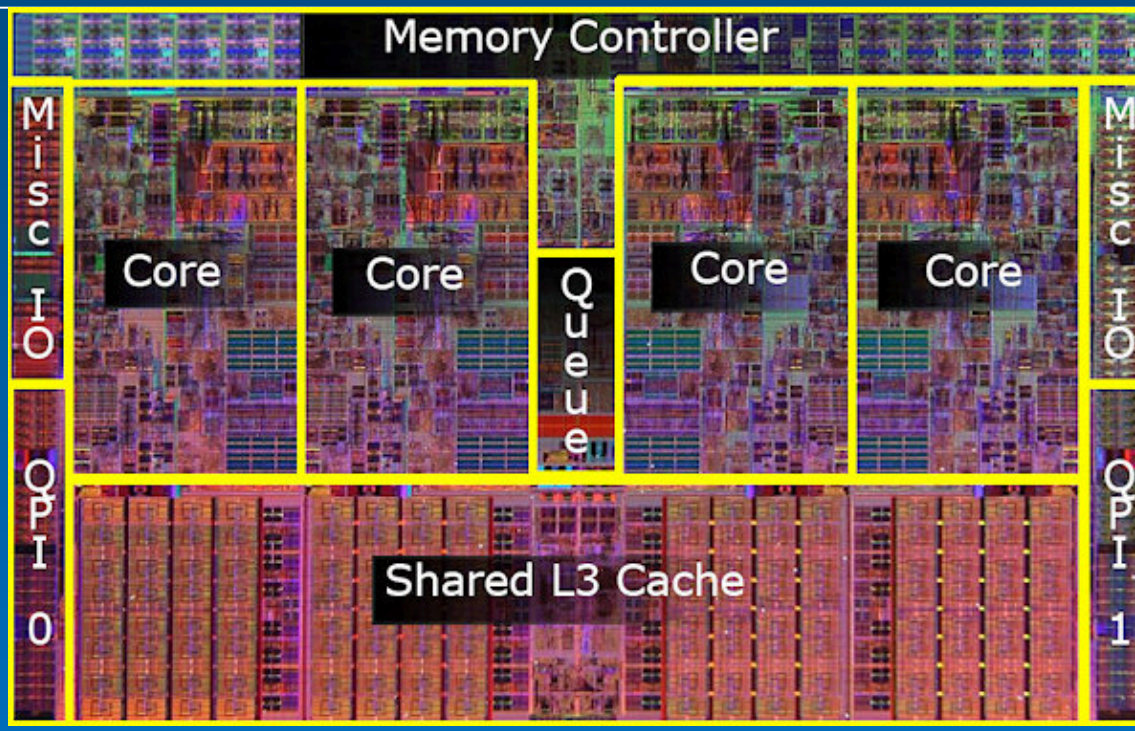
UNIVERSITY OF STIRLING

Content

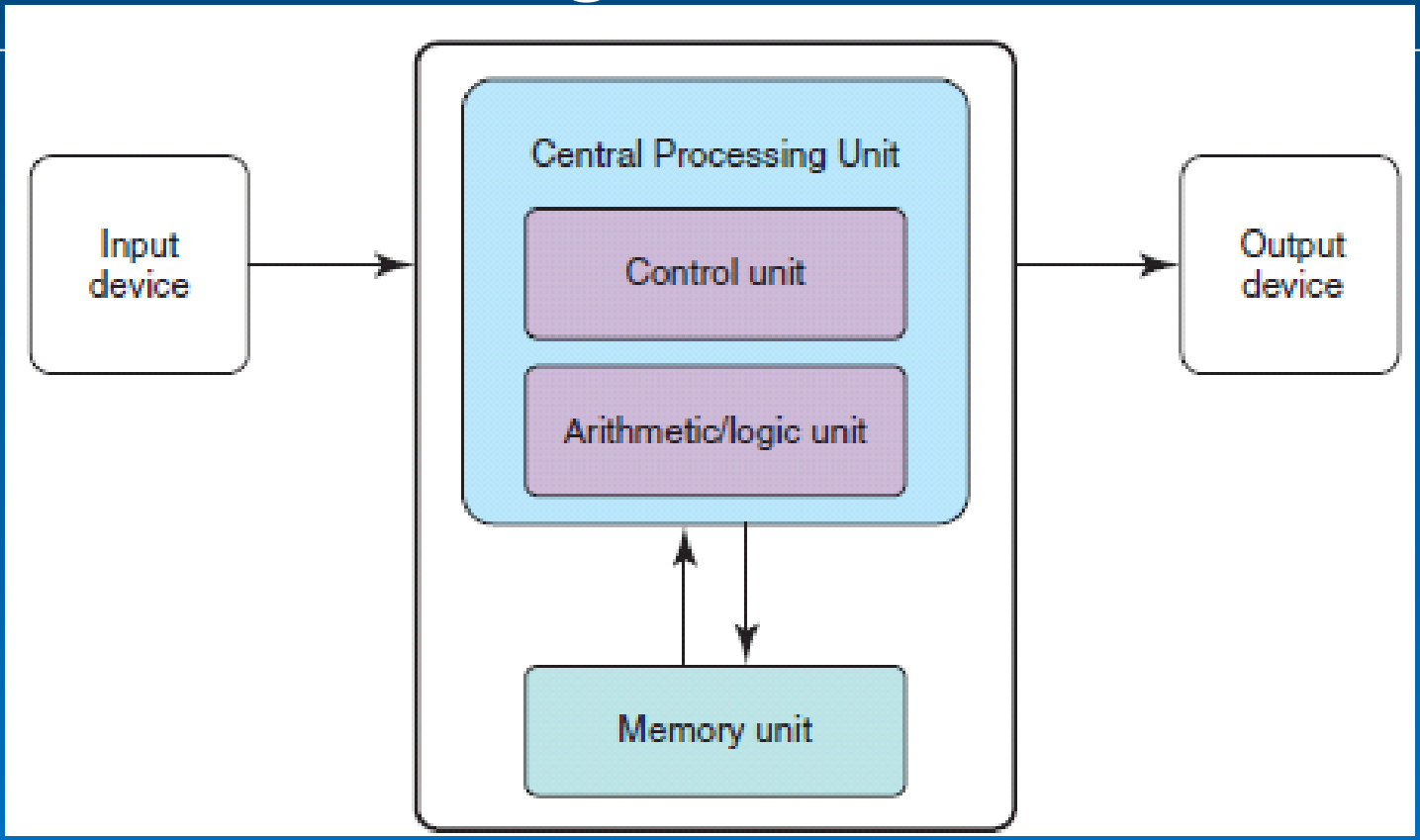
- Back to basics: Processors
- Instructions
  - Language types: low and high level
- Instruction operation
  - Fetch
    - and Cache hierarchy
  - Execute
- Relating this to Java and the Java Virtual machine

What does a processor do?

- Picture shows a current processor silicon die
- (Intel Nehalem)
- The actual CPU …
  - There's 4 of them
- … is the part labelled Core.



To begin with …



- From Lecture 1: there's a control unit and an ALU …
- A sequence of *instructions* gets executed

What is an instruction?

- Informally:
  - Instructions are the unit of operation of a CPU. Each one might (for example)
    - Add two registers together, placing the result in a third register
      - Or multiply, or logically AND, or , …
    - Load a register from a memory location
    - Store a register's value to a memory location
      - Where that location's address might be held in another register
- An instruction is indivisible: once it starts it *will* finish
  - Instructions are held in binary
    - Generally one per word of memory, but there are many exceptions to this
  - Instructions are not usually directly written by programmers
  - Programmers write in high level languages which are compiled into instructions.
- Different CPUs have different *instruction sets.*

An aside: Language types

- Computer languages are divided into a number of types
  - Low-level: Each statement translated into a single (or small number of) machine code instructions
  - High-level: Each statement results in a large number of machine code instructions
- All the well known languages (C, C#, C++, Objective C, Prolog, … are high level languages
  - There are also other high-level-type languages called scripting languages: they are interpreted rather than directly translated or compiled
    - PHP, Javascript, Perl, R, …
  - And others which are somewhere in-between interpreted and translated
    - Java
- Low-level languages are tied to the CPU and the machine language of that CPU
  - Assembly code is the usual name for low a low-level language
    - And they are only used in systems programming
  - Generally, 1 machine instruction is generated per line of assembly code

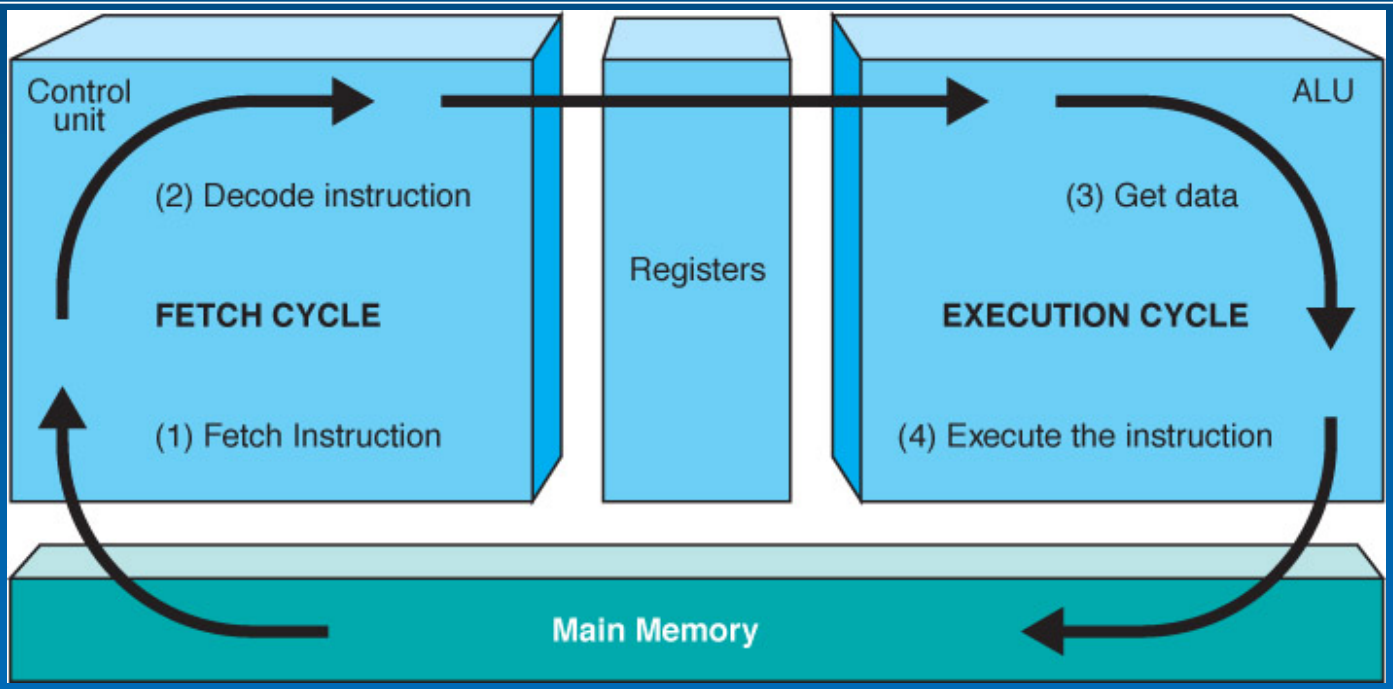
Some instructions

- We use assembly code, rather than machine code
  - Because looking at a sequence of binary digits is not educational!

```
mov eax, [esi-4]     ; Move 4 bytes at memory address ESI + (-4)
                     ; into EAX          (ESI and EXS are registers
sub eax, 2           ; Subtract 2 from register EAX
```

- The instruction is the part up to the semicolon: the rest is comment
- Each one translates into a sequence of bits (which does not concern us here).
- Assembly code is turned into binary by a program called an Assembler.

Fetching and executing instructions



- Use the *program counter* to fetch the instruction
- Put the instruction into the *instruction register*
- *Execute the instruction*

Fetch…

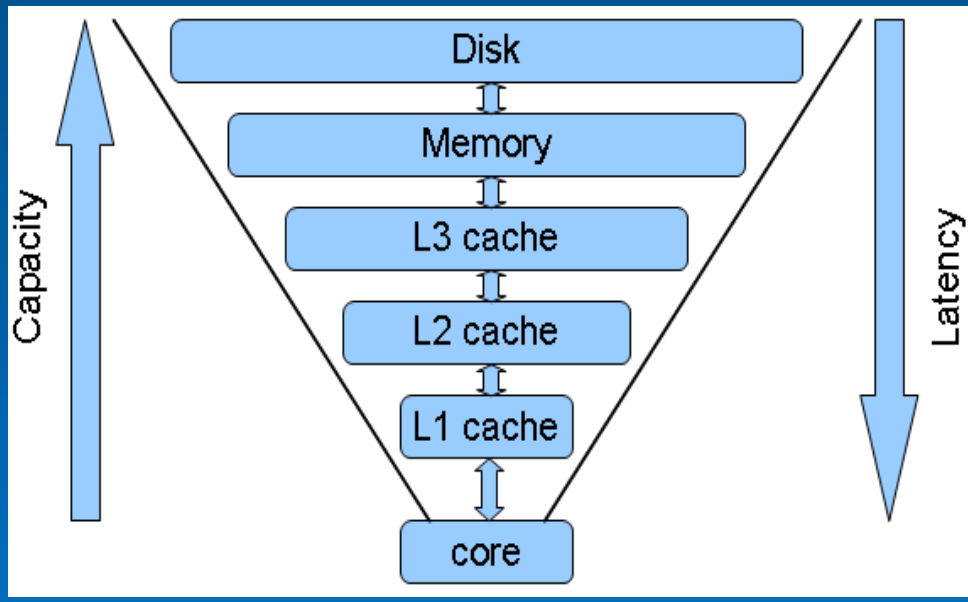
- Fast CPUs can execute more than 2,000,000,000 instructions/second
- So fetching instructions needs to be extremely fast
- The main memory cannot operate at this speed
  - (why not? Arguments from physics and resistance, capacitance and inductance in electronics)
- But on-chip cache can
  - (why? Distances, capacitances, …)

Cache concept

- Although CPUs run huge numbers of instructions/second
  - They tend to repeat many instructions.
  - Most of the instructions executed are inside loops
    - So they have already been done recently
  - So:
    - We load the instruction from the main memory into the IR *and* into the cache the first time it's executed
    - We load the instruction from the cache subsequently
- Using the cache we can feed the CPU with instructions as fast as required
  - (mostly: there's issues here, because the cache is much smaller then the main memory, so when and where do we over-write cache data, and how many instructions do we load at a time, and …)

Multi-level caches

- In fact modern processors often have more than 1 level of cache
  - And have caches for both instructions and data
- On Nehalem:
  - Level 1 and level 2 are on each core
    - Separate Level 1 cache for program and data
  - Level 3 is shared between cores
- Cache memory is critical for performance.

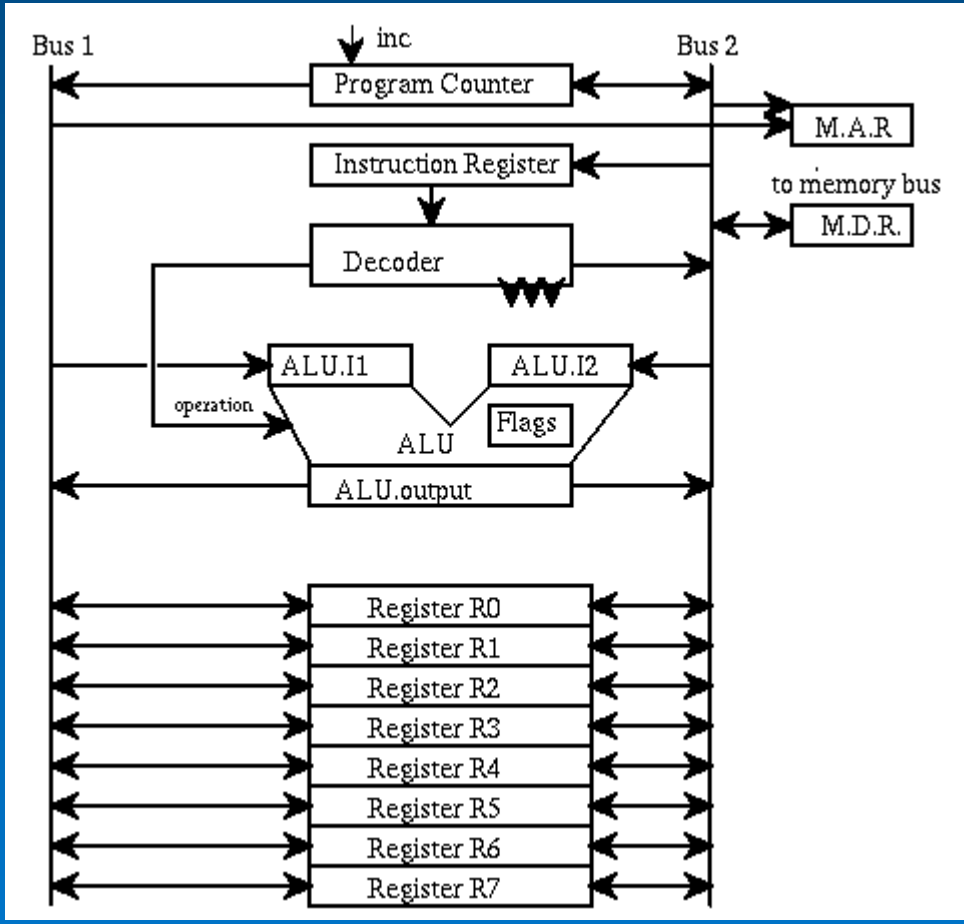


Cache hit and miss

- A cache hit occurs when the datum
  - whether program or data
- … is actually found in the cache
- And a cache miss occurs when the datum is *not* in the cache
- Cache misses slow down processing
  - The data has to be fetched (at the very least) from the next level of cache.
- Cache size (and locality of processing) can have a major effect on performance.

Execute

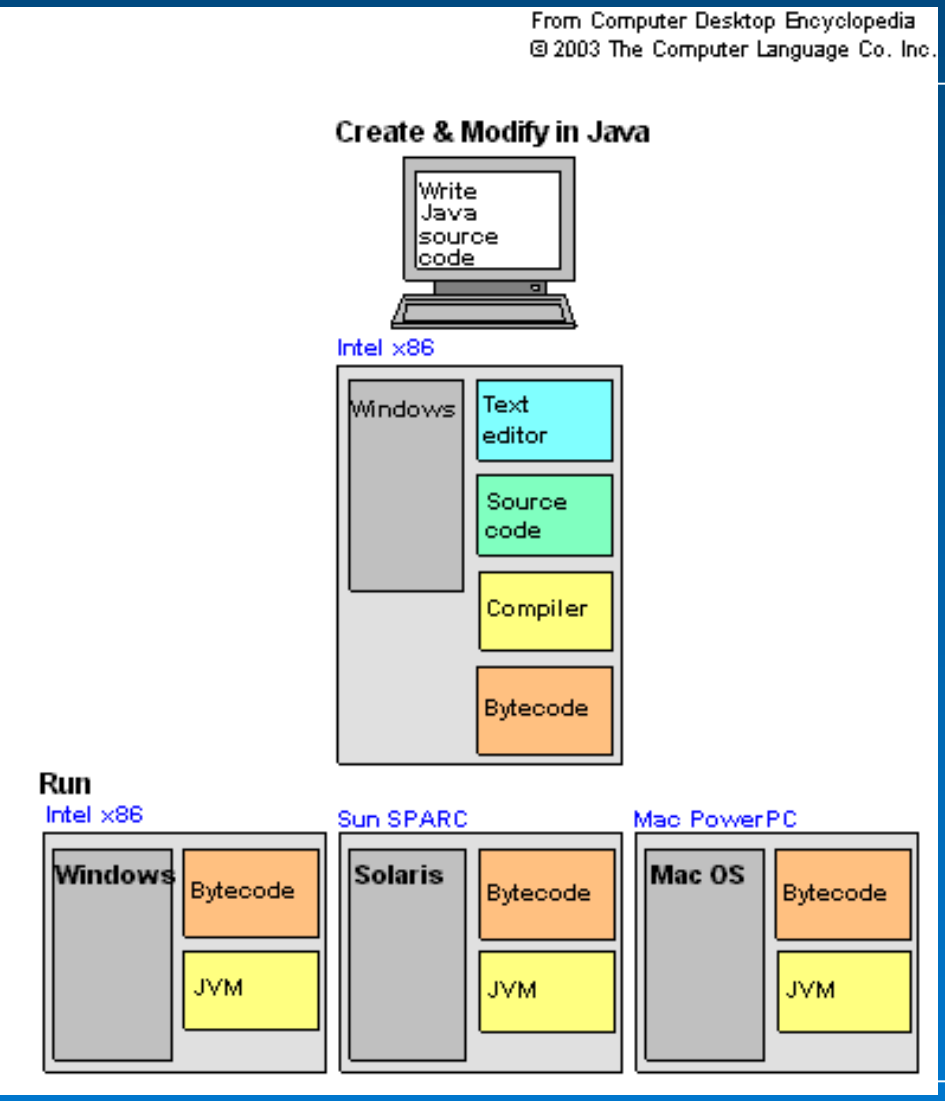
- Once the instruction is fetched
  - And is placed in the Instruction Register
- Execution of the instruction can start
  - The instruction bit pattern is used to …
    - Open and close data paths
    - Provide the ALU with the appropriate code to select an operation (if relevant)
  - … so that the sequence of steps leads to the execution of the instruction.
  - The sequence is different for different instructions.

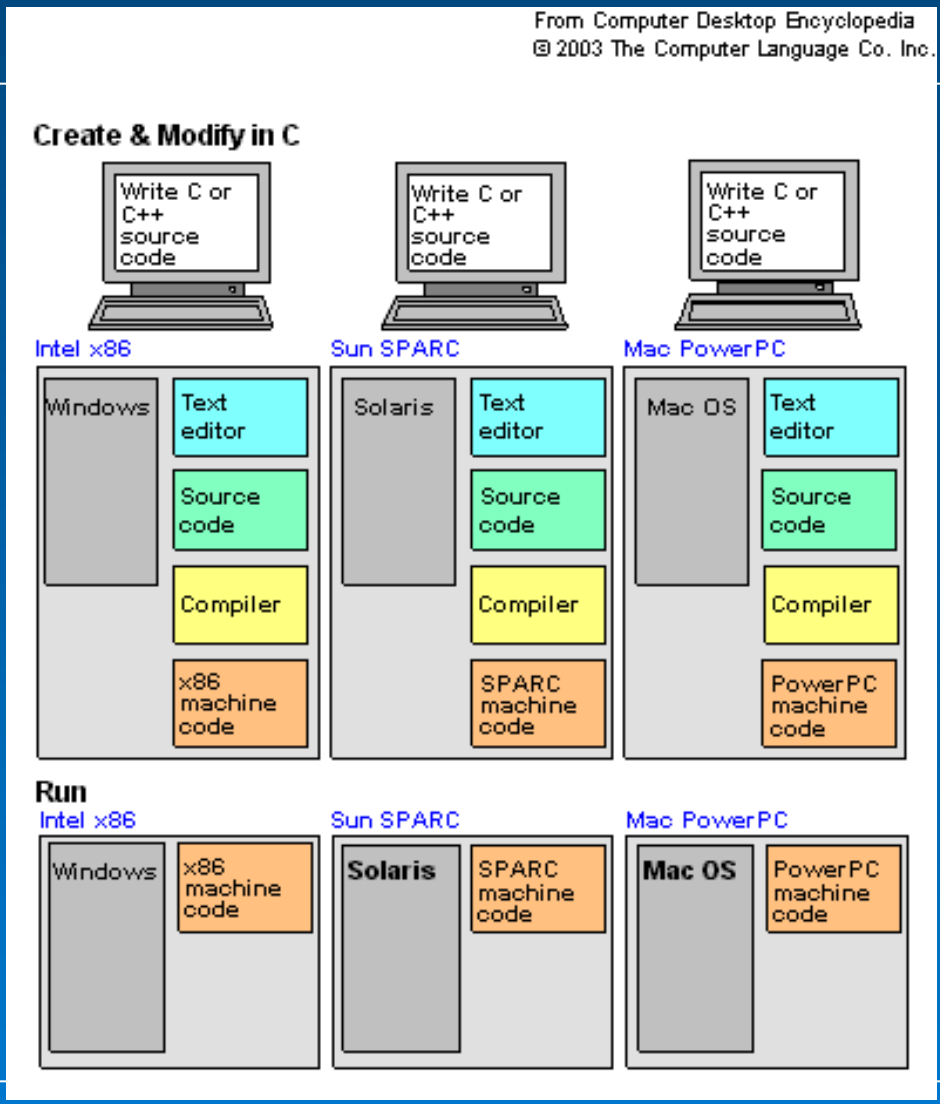


Relating this to programming Java

- Java is a *portable* high level programming language
- Conceptually, it runs on a Java Virtual Machine
  - JVM
- Using an assembly code called bytecode.
- Bytecode is interpreted by the JVM
  - The JVM itself is simply a program running on your machine
  - JVM implementations exist for virtually every sophisticated processor
    - They all run the same bytecode, so compiled programs in java are portable
- Other high level languages are portable only at the high-level-language level
  - They need to be compiled for each different CPU type.

Compilation and running: Java vs C





Why JVM and bytecode?

- There are many high level languages
  - And they are usually compiled down to machine code
  - This is highly efficient
  - But: each CPU has a different machine code
    - So compiled programs are not portable
    - Code needs compiled for each different CPU (and each Operating System too)
- Some high level languages are directly interpreted
  - Scripting languages, for example
  - So long as the machine has an interpreter, these languages may be run on that machine
    - Generally less efficient than full compilation
- Java uses a program (the JVM) which interprets compiled Java programs
  - Compilation is into bytecode
  - JVM's are simpler to write than full high level language interpreters
  - Interpreting bytecode is relatively efficient.
  - Provides reasonably efficient portability!

Java and web browsers

- A web browser can include a JVM
  - So it can run compiled java programs
- And this JVM can be set up so that it cannot access other machine resources
  - Apart from those belonging to the browser
- Making running Java programs inside a web browser safe
  - Whatever the bytecode might attempt to do
    - Safe against both accidental and intentional attempts to breach security
- That was the idea: however, it has proven difficult to ensure security of Java inside browsers.
  - More often the functionality intended to be provided by java is now provided by JavaScript.