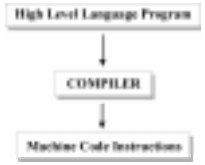
High Level Languages

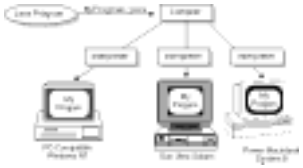
- C, C++, Java are examples of HIGH LEVEL languages
- HLL code must first be converted to an machine code version before the CPU can execute the program
- This is known as compilation. (vs. assembling)
- Compilation is performed automaticly by a piece of software called a compiler



Java

- Java is a high-level programming language that is all of the following:
  - Architecture-neutral
  - Object-oriented
  - Portable
  - Distributed
  - High-performance
  - Interpreted
  - Multithreaded
  - Robust
  - Dynamic
  - Secure

Java Program Languages



http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

Java Program Languages

- Java compiler translate a program into an intermediate language
- It is called Java **bytecode**
- The platform-independent codes is interpreted by the Java interpreter

http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

High Level Languages -> Machine Code

Machine Code	Assembly Language	High Level Language

303C
0004
323
0005
9240
4E74
	move #4, D0
	
	move #5, D1
	
	add D0, D1
	end
		a := 4 + 5;

303C [Hex] = 0011 0000 0011 1100 [Bin]

Questions

- How do we write programs?
- How do we execute the programs?
- How do we supply Input Data?
- How do we obtain the Output Data?

Operating Systems

- A computer without software cannot:
  - Load and run programs from disk
  - Perform I/O (Input/Output)
  - Handle situations where the software to run requires more memory than is physically available on the computer
- An Operating System = A Program which insulates the user from the technical detail of the machine.

More Operating Systems

- It provides an environment/tool on the computer with which the user can exexcute and manage other software resources, e.g.:
  - Editor
  - Compiler
  - etc.
- Advanced operating systems (such as UNIX) allow many users to access the same machine simultaneously (multi-tasking).

The MONITOR

A *MONITOR* is the simplest form of operating system. It is designed with a minimal specification and allows execution of low level assembly language programs.

The monitor is a program (written in assembly language) which provides an *interface* to the CPU.

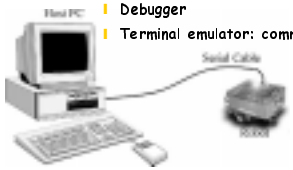
Using knowledge from this course, you will write your own monitor in 2BA4 for a computer system (similar to the Robot) that you will design and build.

Monitor provides functions to allow:

- Writing and editing assembly language code
- Assembling of code to machine language
- Execution of software
- Reading/Writing memory and CPU registers
- Connecting to external computer systems for saving and loading of code
- Debugging

Robot Development System

- The RDS contains an
  - Editor: used for writing and saving assembly code
  - Assembler
  - Downloader: loads code onto robot
  - Debugger
  - Terminal emulator: communicates with robot for I/O

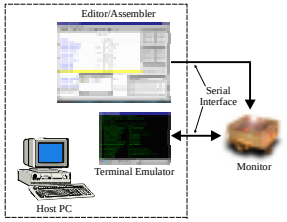


Robot Development System

- All software is *written/saved* on the PC.
- Programs are *downloaded* and *executed* on the Robot.
- The Robot monitor does **not** allow editing/assembling or saving/printing.
- The monitor is saved in ROM ⇒ monitor remains even if the power is off.
- User programs saved in RAM ⇒ program lost when power off.

Terminal Emulator

The TE handles direct communication with the Robot.

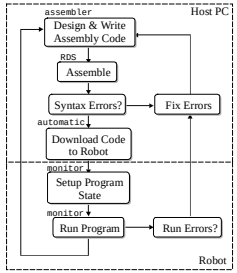


Every key hit in the TE is sent to the robot.

Robot responds with messages which are printed in the TE window.

Development System Software

Development Cycle



How can the monitor and my program run at the same time?

- Answer: they don't.
  - Only one program can run at any given time on the CPU
  - Monitor stops when user program is run.
  - Monitor starts again when user program terminates (hopefully!).

Program Execution

- How does the CPU execute code?
  - The program will exist as numbers in memory
  - The CPU must be told which program to execute (i.e. where code exists in memory -> memory location)

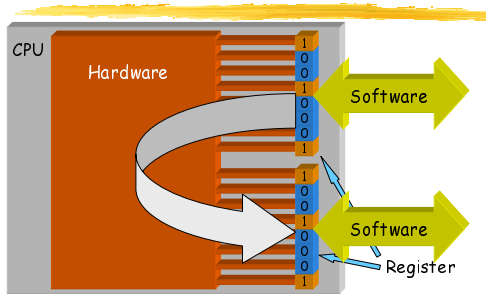
Fetch -> Decode -> Execute

- FETCH:
  - Read the next memory location to retrieve the next instruction.
- DECODE:
  - Determine what the instruction requires of the CPU and read any extra data need.
- EXECUTE:
  - Perform the required operation.

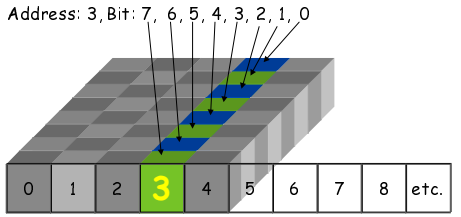
Registers

- To operate efficiently with data, the CPU has, built into it, a number of storage areas of fixed size.
- These areas are known as accumulators or data-register.
- Before operating on data, it is loaded from memory into the required registers.
- The CPU must know where the program to be executed is located in memory and be able to keep track of where it is at any given time.

Registers -> Hardware



Memory - Address - Bit



CPU - RAM Read & Write