Computers in Engineering COMP 208

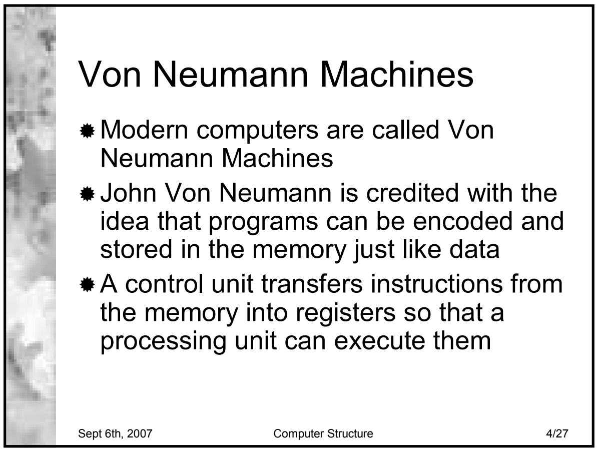
Computer Structure
Michael A. Hawker

Computer Structure

- We will briefly look at the structure of a modern computer
- That will help us understand some of the concepts that occur in Fortran and C

Computer Architecture

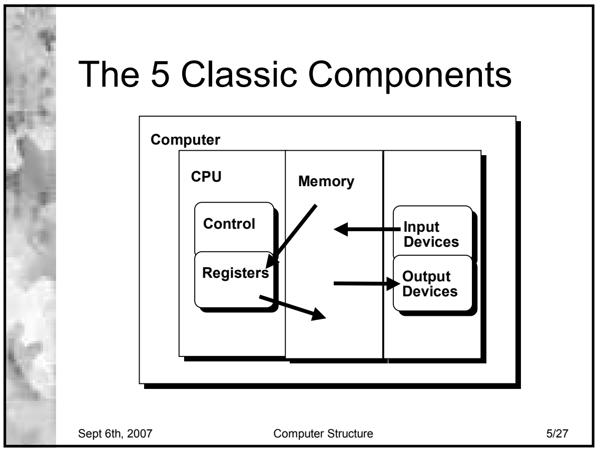
- At the lowest level a computer is just a collection of switches that can be on or off (representing 1 and 0).
- The circuitry is organized into components that serve different functions such as decoding bit sequences, carrying out simple arithmetic operations, etc.



Von Neumann Machines

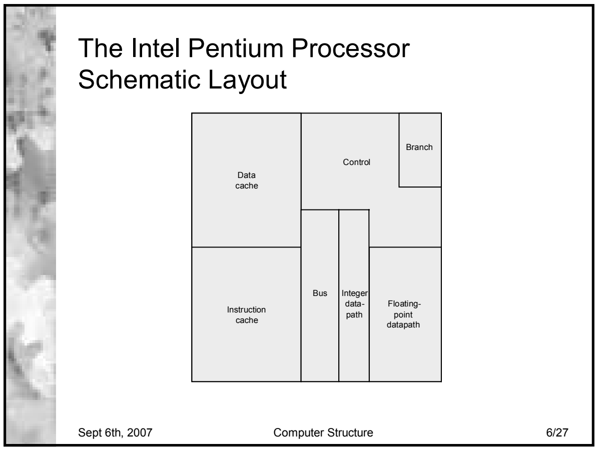
- Modern computers are called Von Neumann Machines
- John Von Neumann is credited with the idea that programs can be encoded and stored in the memory just like data
- A control unit transfers instructions from the memory into registers so that a processing unit can execute them

Sept 6th, 2007 Computer Structure 4/27



The 5 Classic Components

Sept 6th, 2007 Computer Structure 5/27



The Intel Pentium Processor Schematic Layout

Sept 6th, 2007 Computer Structure 6/27

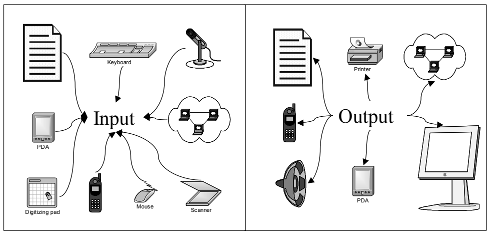
The Von Neumann Model

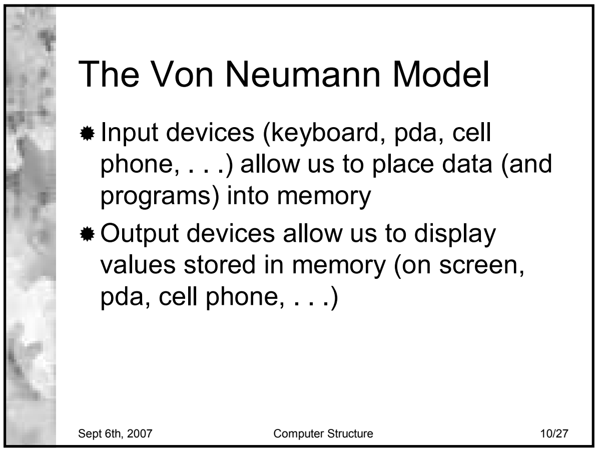
- Programs and data are both stored in the main memory of the machine
- There is one CPU (Central Processing Unit)
- The CPU has a control unit that fetches program instructions from memory, decodes them and executes them

The Von Neumann Model

- Data is loaded from memory into registers in the CPU
- The CPU contains circuitry to perform basic arithmetic operations and to compare values, placing the results into registers
- The values in the registers can be stored in main memory

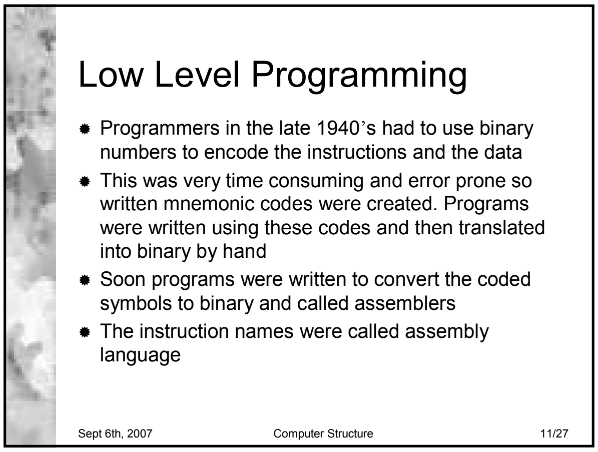
Input / Ouput





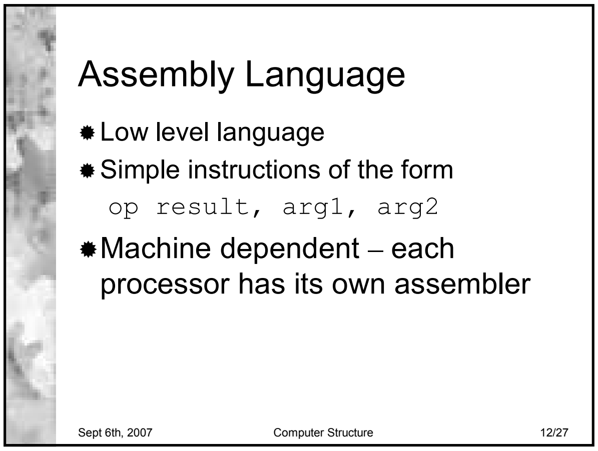
The Von Neumann Model

- Input devices (keyboard, pda, cell phone, . . .) allow us to place data (and programs) into memory
- Output devices allow us to display values stored in memory (on screen, pda, cell phone, . . .)



Low Level Programming

- Programmers in the late 1940's had to use binary numbers to encode the instructions and the data
- This was very time consuming and error prone so written mnemonic codes were created. Programs were written using these codes and then translated into binary by hand
- Soon programs were written to convert the coded symbols to binary and called assemblers
- The instruction names were called assembly language

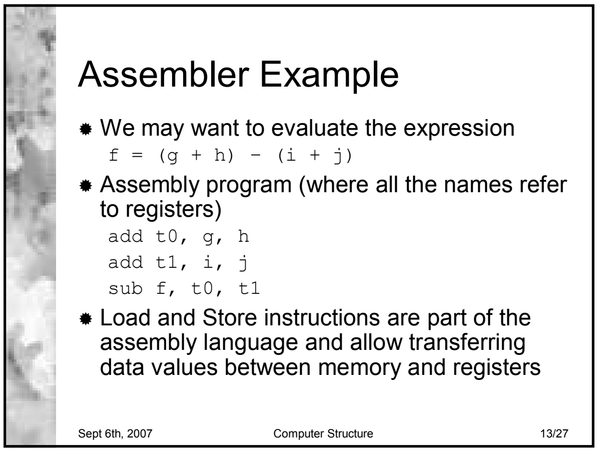


Assembly Language

- Low level language
- Simple instructions of the form

```
op result, arg1, arg2
```

- Machine dependent – each processor has its own assembler



Assembler Example

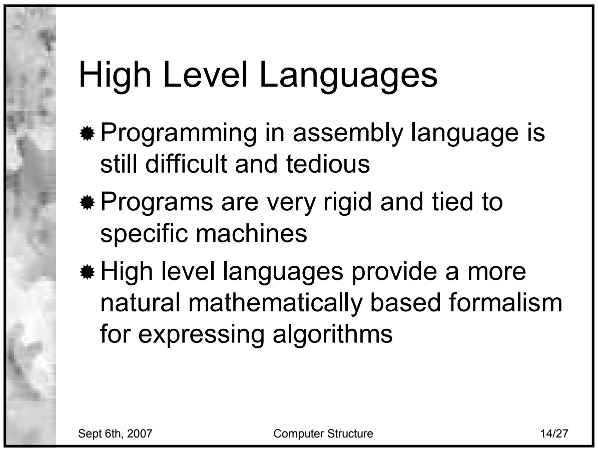
- We may want to evaluate the expression

```
f = (g + h) – (i + j)
```

- Assembly program (where all the names refer to registers)

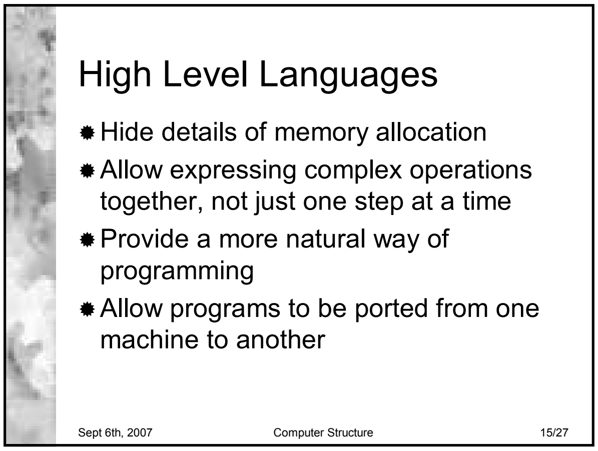
```
add t0, g, h
add t1, i, j
sub f, t0, t1
```

- Load and Store instructions are part of the assembly language and allow transferring data values between memory and registers



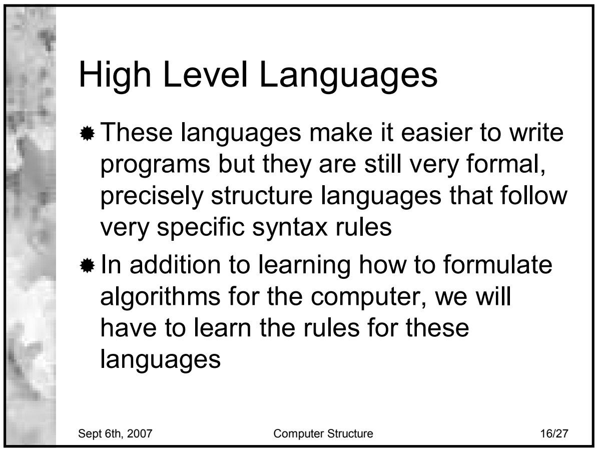
High Level Languages

- Programming in assembly language is still difficult and tedious
- Programs are very rigid and tied to specific machines
- High level languages provide a more natural mathematically based formalism for expressing algorithms



High Level Languages

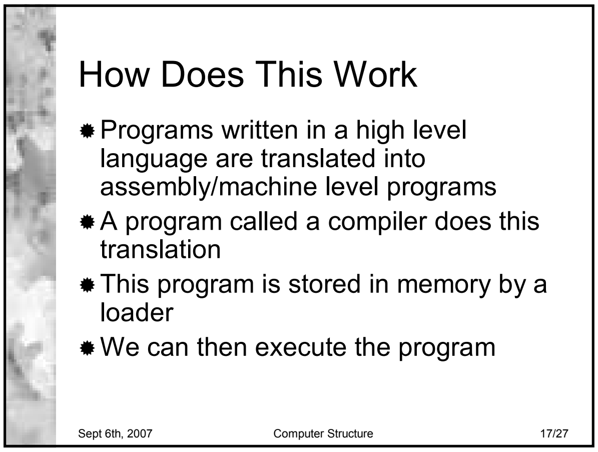
- Hide details of memory allocation
- Allow expressing complex operations together, not just one step at a time
- Provide a more natural way of programming
- Allow programs to be ported from one machine to another



High Level Languages

- These languages make it easier to write programs but they are still very formal, precisely structure languages that follow very specific syntax rules
- In addition to learning how to formulate algorithms for the computer, we will have to learn the rules for these languages

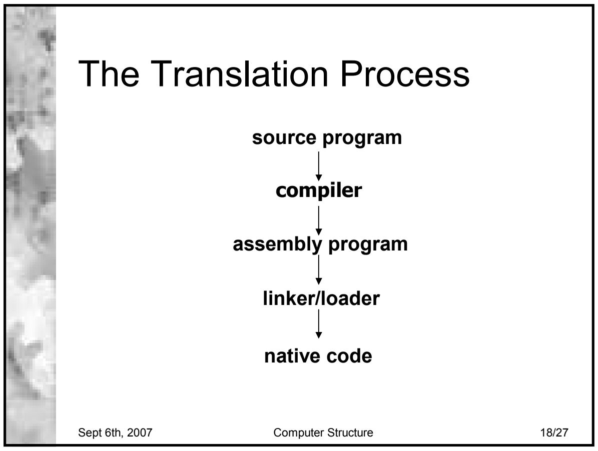
Sept 6th, 2007 Computer Structure 16/27



How Does This Work

- Programs written in a high level language are translated into assembly/machine level programs
- A program called a compiler does this translation
- This program is stored in memory by a loader
- We can then execute the program

Sept 6th, 2007 Computer Structure 17/27



The Translation Process

source program

↓

compiler

↓

assembly program

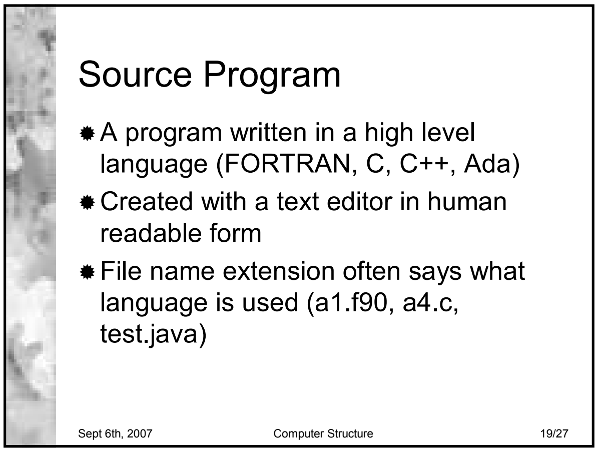
↓

linker/loader

↓

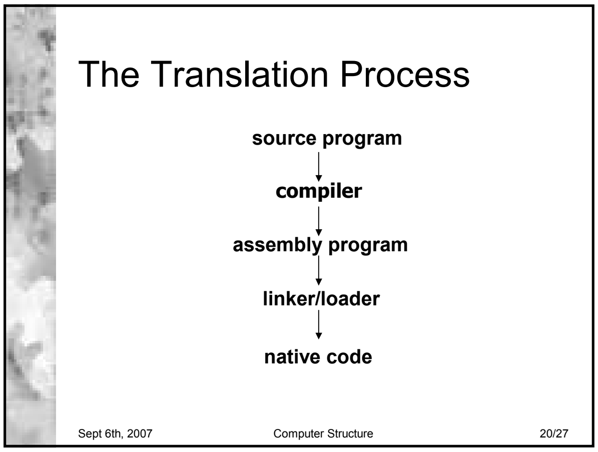
native code

Sept 6th, 2007 Computer Structure 18/27



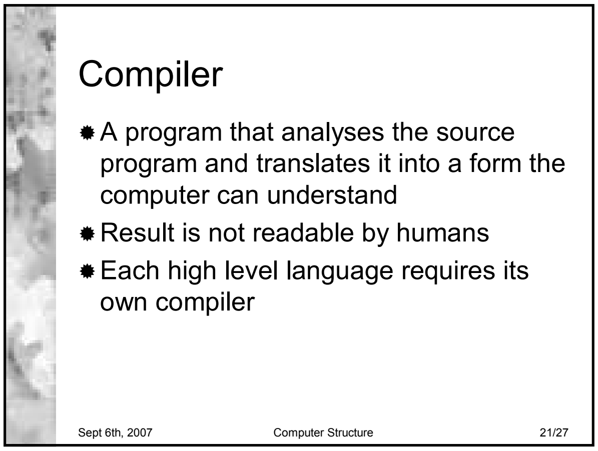
Source Program

- A program written in a high level language (FORTRAN, C, C++, Ada)
- Created with a text editor in human readable form
- File name extension often says what language is used (a1.f90, a4.c, test.java)



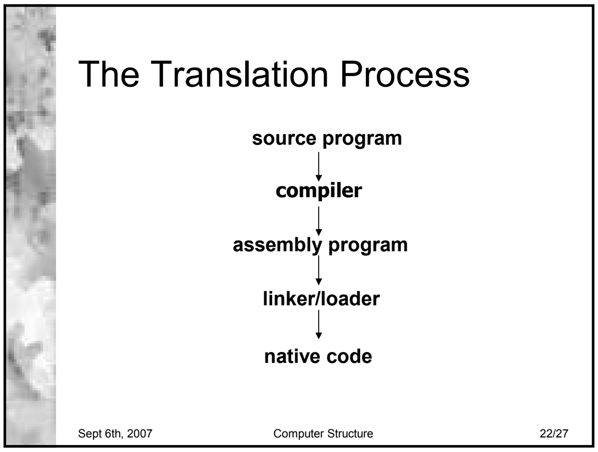
The Translation Process

source program
↓
compiler
↓
assembly program
↓
linker/loader
↓
native code



Compiler

- A program that analyses the source program and translates it into a form the computer can understand
- Result is not readable by humans
- Each high level language requires its own compiler



The Translation Process

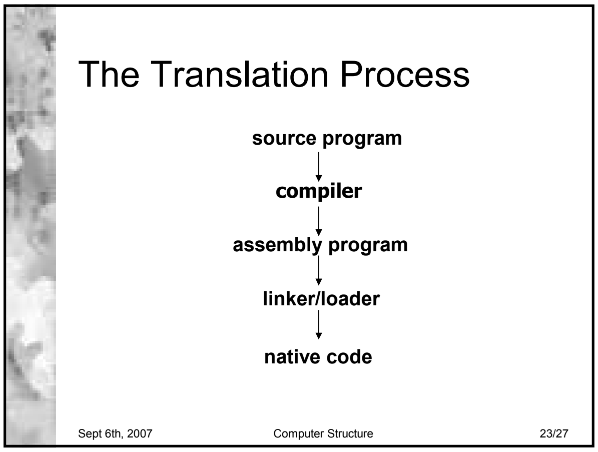
source program

compiler

assembly program

linker/loader

native code



The Translation Process

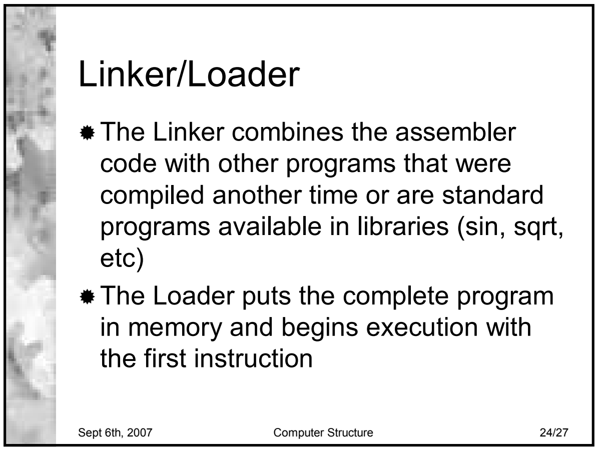
source program

compiler

assembly program

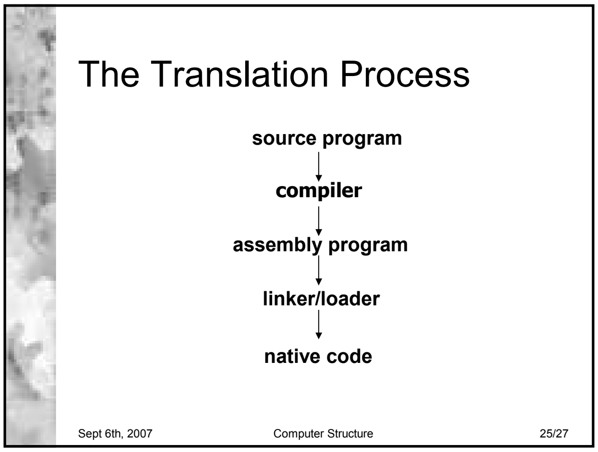
linker/loader

native code



Linker/Loader

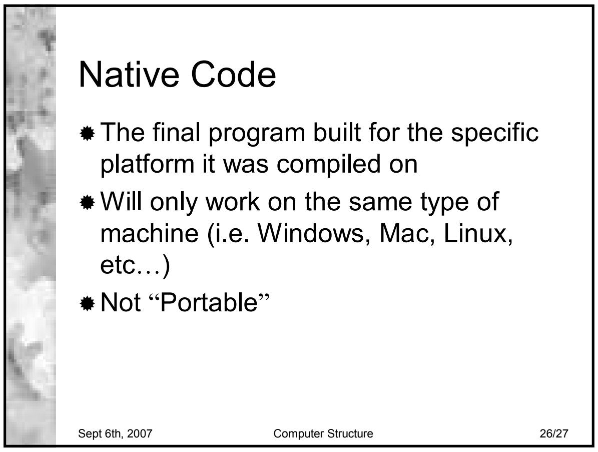
- The Linker combines the assembler code with other programs that were compiled another time or are standard programs available in libraries (sin, sqrt, etc)
- The Loader puts the complete program in memory and begins execution with the first instruction



The Translation Process

source program
↓
compiler
↓
assembly program
↓
linker/loader
↓
native code

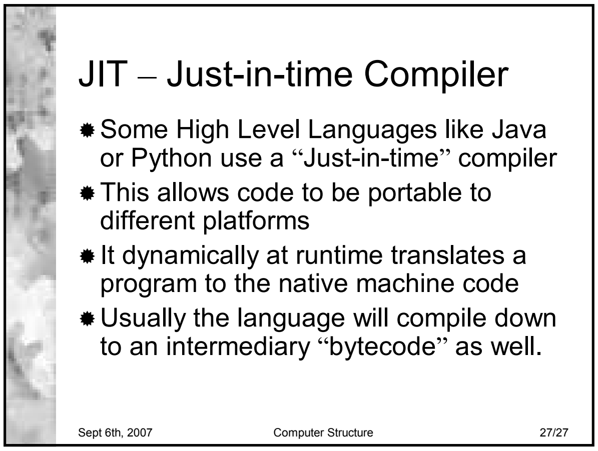
Sept 6th, 2007 Computer Structure 25/27



Native Code

- The final program built for the specific platform it was compiled on
- Will only work on the same type of machine (i.e. Windows, Mac, Linux, etc…)
- Not “Portable”

Sept 6th, 2007 Computer Structure 26/27



JIT – Just-in-time Compiler

- Some High Level Languages like Java or Python use a “Just-in-time” compiler
- This allows code to be portable to different platforms
- It dynamically at runtime translates a program to the native machine code
- Usually the language will compile down to an intermediary “bytecode” as well.

Sept 6th, 2007 Computer Structure 27/27