

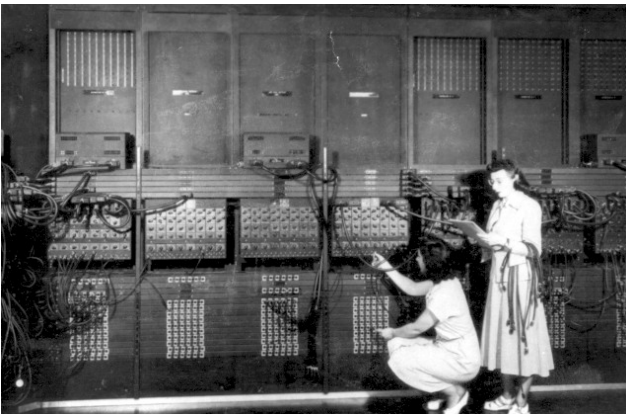
how computers work (4)

systems software - the bridge between application software and computer hardware

motivation

- imagine a computer without systems software – only applications software and hardware
- the earliest computers were this way

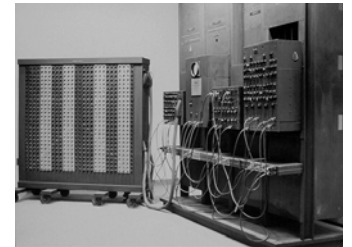
programming without an operating system :-)



<http://www.berkeleyprep.org/webhouse/bh/html/histpic.htm>

ENIAC (Electronic Numerical Integrator and Computer) 1946

“The procedure for instructing the ENIAC in its routine, then, consists of setting program switches on the units so that, when stimulated by a program input pulse, the program controls will cause the units to carry out a set of specific operations.”



- Adele Goldstine, 1946

<http://www.seas.upenn.edu/~museum/>

what is systems software?

- *application software* refers to Word, Firefox, Ultimate Paint, e-mail clients, programs you write yourself, etc.
- *systems software* provides a high-level environment in which we can run applications on the computer
- **note:** boundary between application and systems software is not always clear

what systems services can you think of?

some services of modern systems software

- organizing files (folders etc.)
- downloading software
- scheduling programs on the processor
- interfacing with the internet
- managing communication with peripherals
- providing security
- ...
- and lots more (GUI, windows, ...)

early operating systems

- advances following the ENIAC allowed programs to be stored in computer memory, paving the way for *batch* operating systems
- users submitted programs (one instruction per punch card) and retrieve the results later
- a stack of jobs could be scheduled on the computer, hence the name "batch"
- the computer processed one job at a time
- often all cards needed to be read before a program could be started – slow!

early operating systems

as technology advanced, several ideas improved on the batch operating systems:

- *multiprogramming* allowed one program to run while another paused, e.g., while waiting for a printer to finish
- *time-sharing* allowed several users to run programs on a machine, providing the opportunity for interactivity

the unix operating system

- developed in early 1970s at Bell Labs
- innovations include:
 - hierarchical file system
 - ability to run several programs simultaneously (*time-sharing*)
 - ability to simultaneously support several users on one computer
 - ability for user base to add and share tools that run as part of the system
- command line interface meant high learning curve initially, but provides lots of flexibility to a knowledgeable user

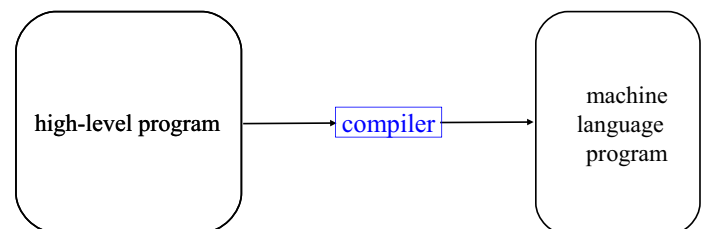
windows operating system

- inspired by early GUIs developed at Xerox PARC (Palo Alto Research Center) in early 1980s
- The Apple Macintosh operating system offered the first commercial window-based GUI
- Microsoft's earliest windows system was released in 1985, built on top of a command line operating system (MS DOS); Windows became popular in 1990 with introduction of Windows 3.0
- Modern Unix operating systems also have window-based GUIs (XWindows)

more system software: translating between programming languages

compilers translate programs written in a high-level programming language (e.g., Java) into the low-level machine language understood by a computer processor

a compiler is a program whose input and output data are programs!



implications of compilation

- high level programming language must be automatically translatable to low-level code
- this constrains the expressiveness of programming languages (since computers are not very good at language processing)
- in particular, programming languages have to be very precise

summary

- **operating system:** software that coordinates and manages resources on the computer
- **compiler:** software that translates programs written in a high level language into a low-level language

putting it all together

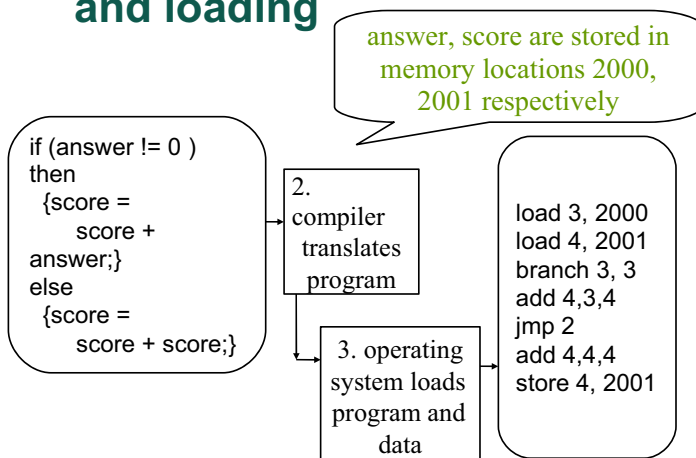
1. an application is written in a high-level programming language (e.g., Java)
2. the code is translated to machine language (e.g., by a compiler)
3. when you want to run the application, the operating system loads the code into RAM (random access memory)
4. the fetch/execute cycle is performed

let's look at an example

1. application code

```
if (answer != 0 )
    {score = score + answer;}
else
    {score = score + score;}
```

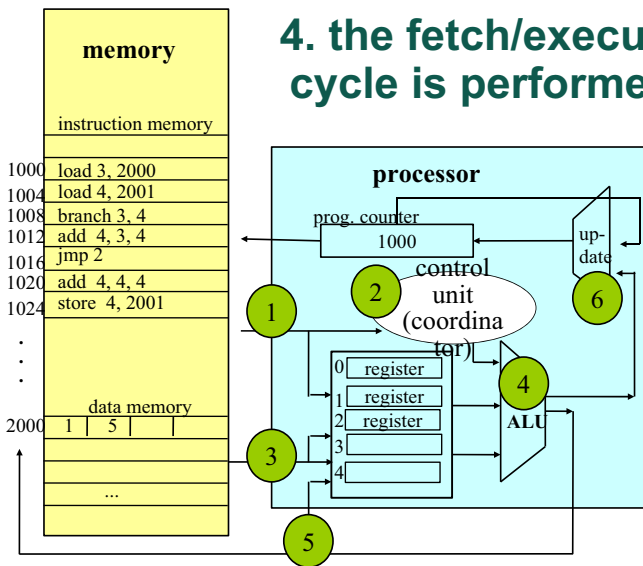
2, 3: translating and loading



(recall our low-level instructions)

- **add 3, 4, 3**
add the contents of registers 3 and 4, and store the answer in register 3
- **load 3, 7000**
load into register 3 the contents of memory location 7000
- **store 2, 2040**
store into register 2 the contents of memory location 2040
- **jmp 7**
move ahead by 7 (low level) instructions
- **branch 2, 5**
if content of register 2 is 0, then move ahead by 5 instructions, otherwise continue to the next instruction

4. the fetch/execute cycle is performed



recall the fetch/execute cycle:

- 1 fetch instruction specified by program counter
- 2 decode instruction
- 3 fetch data from memory and store in registers
- 4 perform operation and send result to data memory, a register, or program counter
- 5 update data memory
- 6 update program counter

resources

- overview of operating systems:
http://en.wikipedia.org/wiki/Operating_system
- the original operations manual for the ENIAC, by Adele Goldstine:
<http://ftp.arl.mil/~mike/comphist/46geniac-report/>