LIFE OF A HELLO PROGRAM

% CSAPP-CHAPTER 1

TRANSLATION

SOURCE FILE

- interpretation of source file
  - sequence of bits (as file)
  - text characters (as text file)
  - program of C statements, syntax (as C source file)
- encoding: context decides representation

TRANSLATION

```
gcc hello.c
```

1. from: C statement @source file
2. to: machine-language instructions @executable object file

COMPILATION SYSTEMS

Four phases

1. Preprocessor (cpp): directives (#), `.c`,`.h` -> `.i` (text file)
2. compiler (`cc1`): `.i`->`.s` (text file)
    - `.s` is assembly-lang. program, each statement is `1`-to-`1` mapped to a machine-lang. instruction
3. assembler (as): `.s`->`.o`
    - relocatable object program, binary file: encode instructions not characters
4. linker (ld): `.o`, `.o`->`.o`.
    - merge multiple relocatable objects to a single executable object

EXECUTION

in shell: command-line interpreter

`./a.out`

How does my screen know what to do, when I only tell my keyboard an object file name?

This is a process of three steps.

1. typing string `a.out`: from keyboard to main memory
2. load executable file `a.out`: from disk to main memory
3. execute object `a.out`: OS and HW

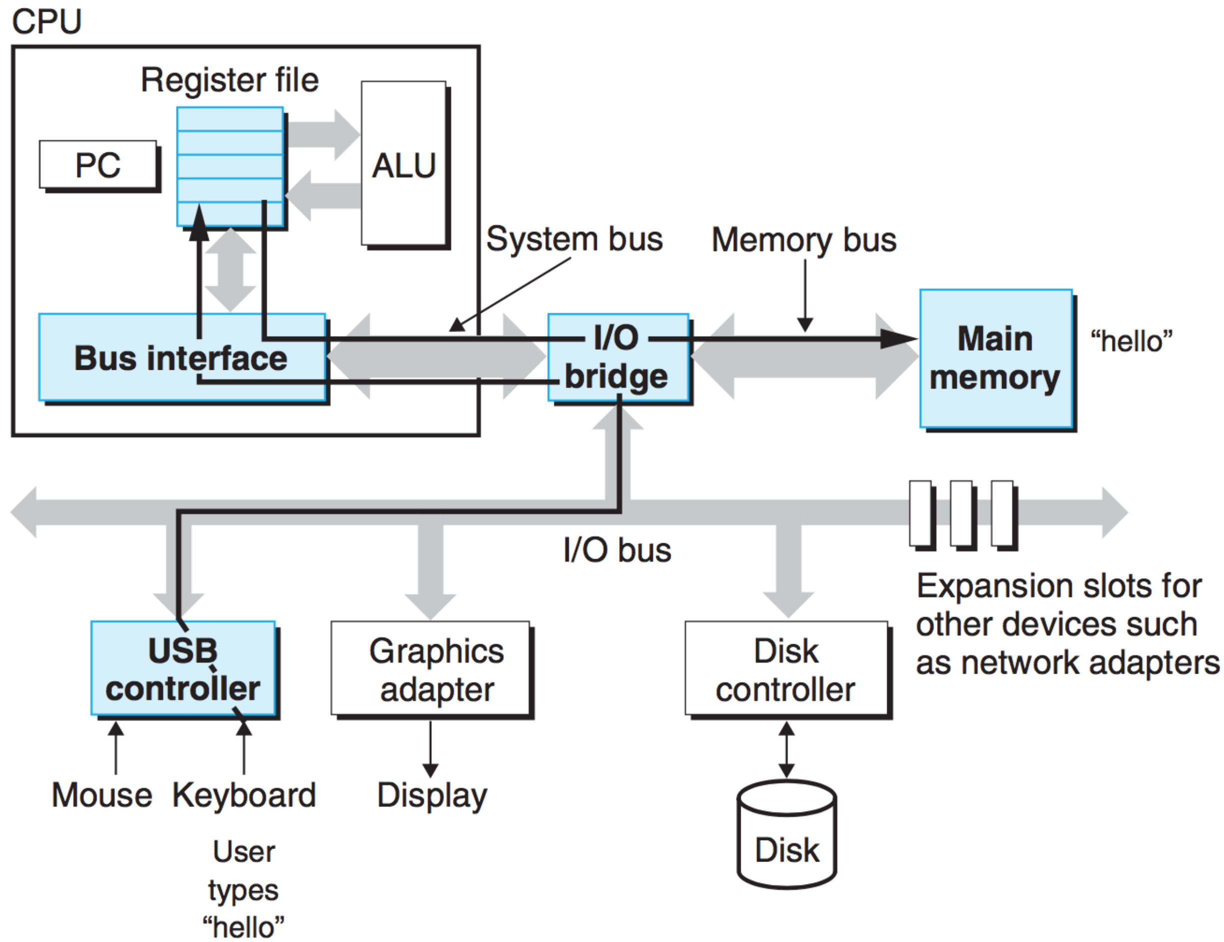
HW OVERVIEW

- Bus: words between HW components
- IO devices: keyboard, mouse, display, disk, network,
  - controller: transfer info. btwn IO bus and devices
  - DMA: copy data in disk device to memory without CPU
- Main memory (not virtual memory!):
  - DRAM chips (physically)
  - linear array of bytes (logically)
  - stores: 1. machine instructions, 2. C program variables
- Processor:
  - register: word-size storage
  - register PC: where to load instruction

- Processor (continued)
  - instruction execution model
    - appear to execute in sequence (actually pipelined, out-of-order)
    - instruction read by PC
    - Turing machine
  - internal:
    - ALU: arithmetic/logic unit
    - register file: a bunch of named registers
  - ISA
    1. Load/Store: a word from main memory to a register
    2. Operate: read register content to ALU, arithmetic op on two words, store result to a register
    3. Jump: overwriting PC

- Processor (continued 2)
  - Cache
    - process-memory gap: register >100 faster than memory
    - L1 size: 10KB
    - L2 size: 1MB
    - SRAM chip
    - Locality: program to access code/data in localized regions

Figure 1.5
Reading the `hello` command from the keyboard.



EXECUTION IN STEP 1,2 FROM HW PERSPECTIVE

1. Step 1: the shell read through keyboard IO characters (`./a.out`) into register, and then store them to main memory
2. Step 2: the shell read character `ENTER`, starts to load `a.out`
   - load.o: load executable file `a.out` by copy program code/data from file to main memory.
3. Step 3: What happens when executing `a.out`?
   - `a.out` is executed in a **process**, has its own address space (in **virtual memory**), and can print out string "helloworld" on screen by writing them to a **file**.

SYSTEMS/OS OVERVIEW

- OS goal:
  1. protecting HW from misuse by runaway app
  2. providing app easy way to manipulate HW
- fundamental abstractions: **process**, **virtual memory**, **file**

```
OS:process      |-----------------------------------|
   virt. mem                    |-----------------|
   file                                        |---|

HW               processor       main mem.       disk
```

1. Process
    - look like exclusive use of HW (no interrupt, only obj in memory)
    - actually, run concurrently
        - instructions of different process interleaved
        - context switch: OS as mediator when switching from one process to another
2. Virtual memory
    - virtual address space
        1. top-most region: kernel
        2. lower region: user
3. file: sequence of bytes
    - read/write file through syscall (Unix IO)

```
+------------+
|  kernel    |
+------------+  +
| user stack |  |
+------------+  |
| shared lib |  dynamic sized
+------------+  |
|   heap     |  |
+------------+  +
|   program  |  |
|  code/data |  fixed size
+------------+  +
```

EXECUTION FROM SYSTEMS PERSPECTIVE

- Shell and `./hello` are two processes, run concurrently

SUMMARY (IN-CLASS EVALUATION)