

Course Overview and background material

- Some terminology and concepts
- A peek at what we will spend the semester talking about.
- A chance to talk about things at a high-level
 - we will spend the course worrying about details!

Programs and Processes

Program: a file (an executable *image* or set of instructions that can be interpreted).

Process: an instance of a program that is running.

A process is *alive*, a program is just a file.

Compiled vs. Interpreted

Some languages required a translator (called a compiler) that converts a text file in to an executable image (machine language).

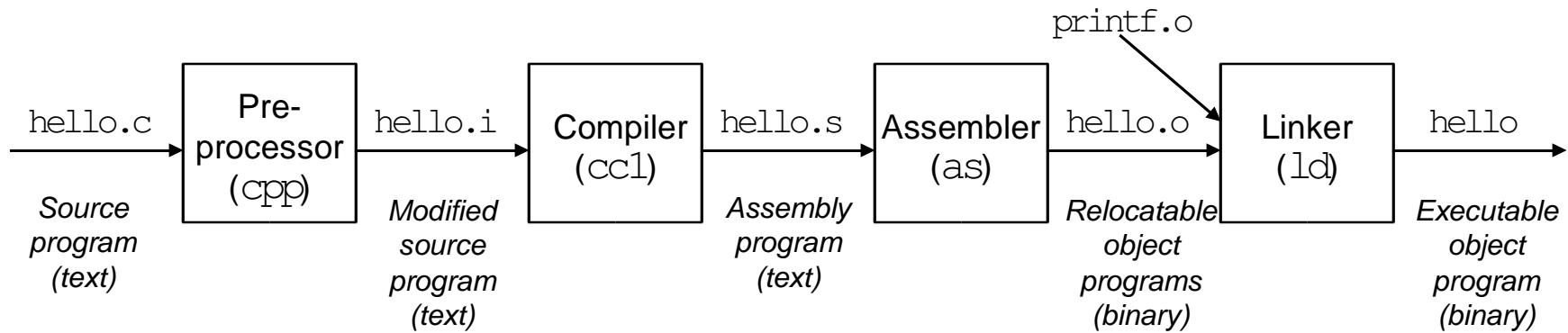
Interpreted languages are text files that are read by another program (an interpreter) and executed *on the fly*.

A C Program: hello.c

```
#include <stdio.h>

int main() {
    printf("Hello World");
    return(0);
}
```

Compilation of hello.c



Text vs. Binary

- The original C program (and some of the intermediate representations) are *text*
 - Typically ASCII encoded.
- The object code and executable are *binary*.
 - anything that is not text is binary!
 - The actual program representation depends on the *machine language* (instruction set) supported by the computer.
 - also depends somewhat on the Operating System

Machine Language

- A sequence of bits.
- Organized in to *words*.
 - different architectures use different length words – typical sizes are 16, 32 or 64 bits.
- The bits are an encoding of some operations that should take place in the CPU.

Instructions

- Each *instruction* does something relatively simple.
 - move some bits around
 - treat some bits as base 2 numbers and apply arithmetic operations.
 - send/read some bits to/from I/O devices.
 - select the group of bits that will make up the next *instruction*.

Example Machine Language

Each instruction is 4 bits long.

There are $2^4=16$ possible unique instructions.

We can organize the bits to make it easy to design a machine that can execute the instructions.

first 2 bits represent an operation.

last 2 bits represent what to operate on.

Machine Language for Life

Operations (2 bits = 4 possible operations):

00: Eat

01: Play

10: Study

11: Watch MTV

00xy

Eat Instructions

The last 2 bits specify what we eat:

<u>xy</u>	<u>What we eat</u>
00	Pizza
01	Burger
10	Macaroni and Cheese
11	Softshell Crab

We have 4 different eat instructions.

01xy

Play Instructions

The last 2 bits specify what we play:

<u>xy</u>	<u>What we play</u>
00	Surf the Web
01	Doom
10	Music
11	Pin-the-tail-on-the-donkey

We have 4 different play instructions

10xy

Study Instructions

The 3rd bit specifies what we study:

<u>x</u>	<u>What we study</u>
0	Comp. Org.
1	American History

The 4th bit specifies how hard we study:

<u>x</u>	<u>How Hard</u>
0	Holding book, but eyes closed
1	Cram session

We have 4 different study instructions

11??

MTV Instructions

4 possible codes: 1100, 1101, 1110, 1111

All four mean the same thing (the last 2 bits don't matter): **WATCH MTV**

It's OK to waste 2 bits, after all – we're already wasting our time with MTV, right?

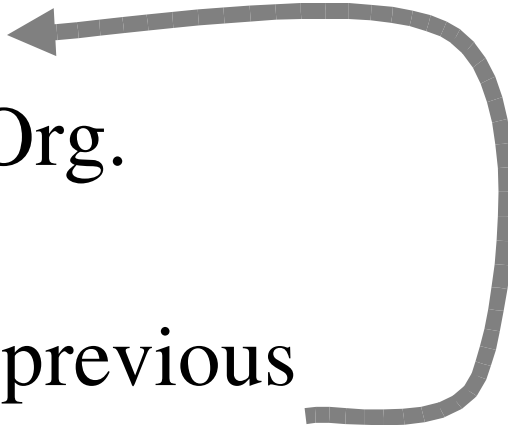
A program

0011	Eat soft shell crab.
1010	Study history with eyes closed.
0100	Surf the web.
1101	Watch MTV.
1001	Cram for Comp. Org.
0101	Play Doom

Lose the MTV Instruction

- Let's make things more interesting:
- *Go back* instruction instead of MTV.
- Last 2 bits determine how far back we go:
 - 00 Repeat previous instruction
 - 01 Go back 1 instruction (before previous)
 - 10 back 2 instructions
 - 11 back 3 instructions

A new plan for tomorrow

- 0011 Eat soft shell crab.
 - 1010 Study history with eyes closed.
 - 0100 Surf the web.
 - 1001 Cram for Comp. Org.
 - 0101 Play Doom
 - 1110 Go back 2 before previous
- 

Stored-program Computer (VonNeumann Computer)

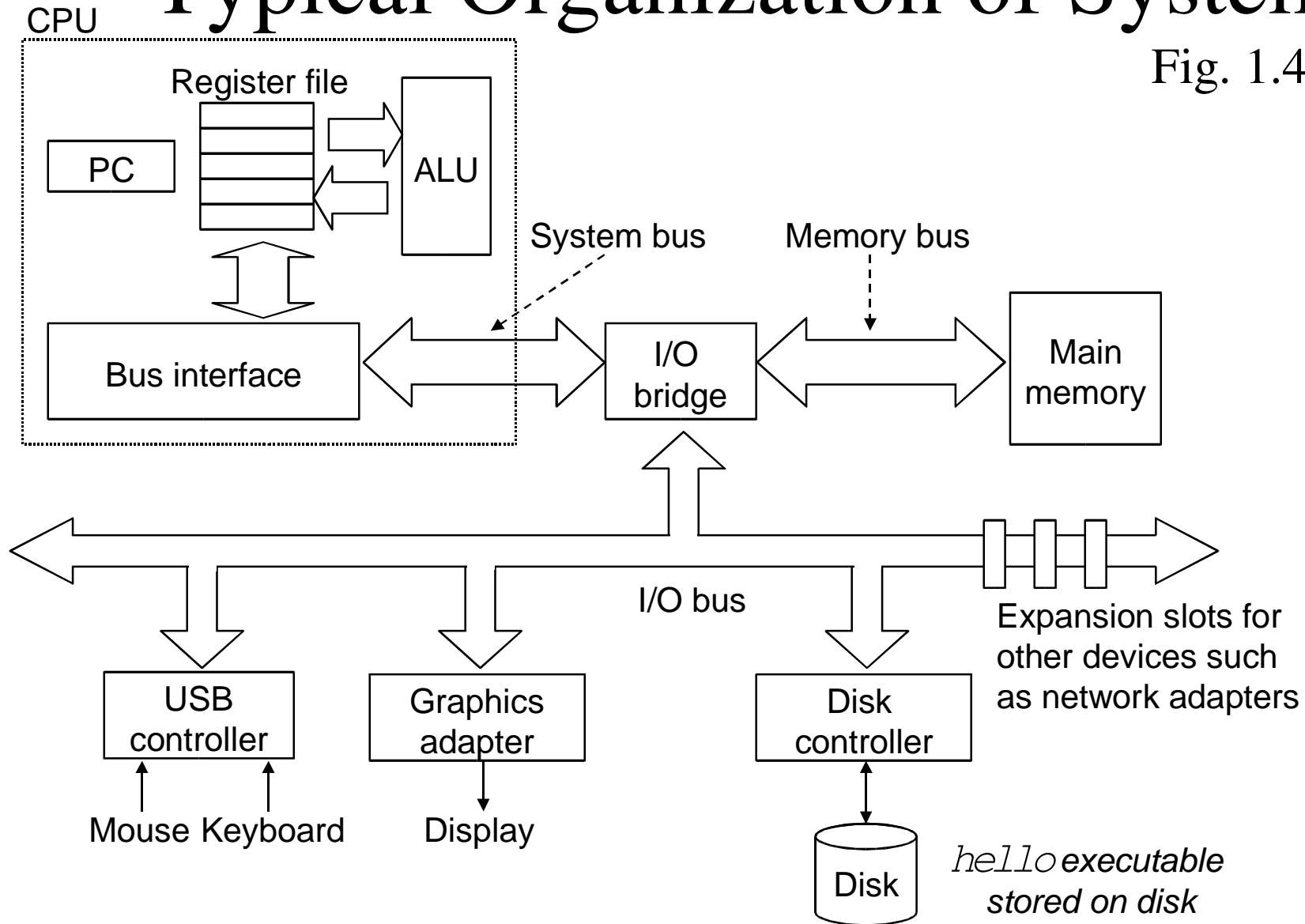
- Somehow put sequence of instructions in a memory.
- A control unit sequences through the memory (instructions) one at a time.
- It is also possible to put *other stuff* in the memory (data).

Hardware to execute instructions

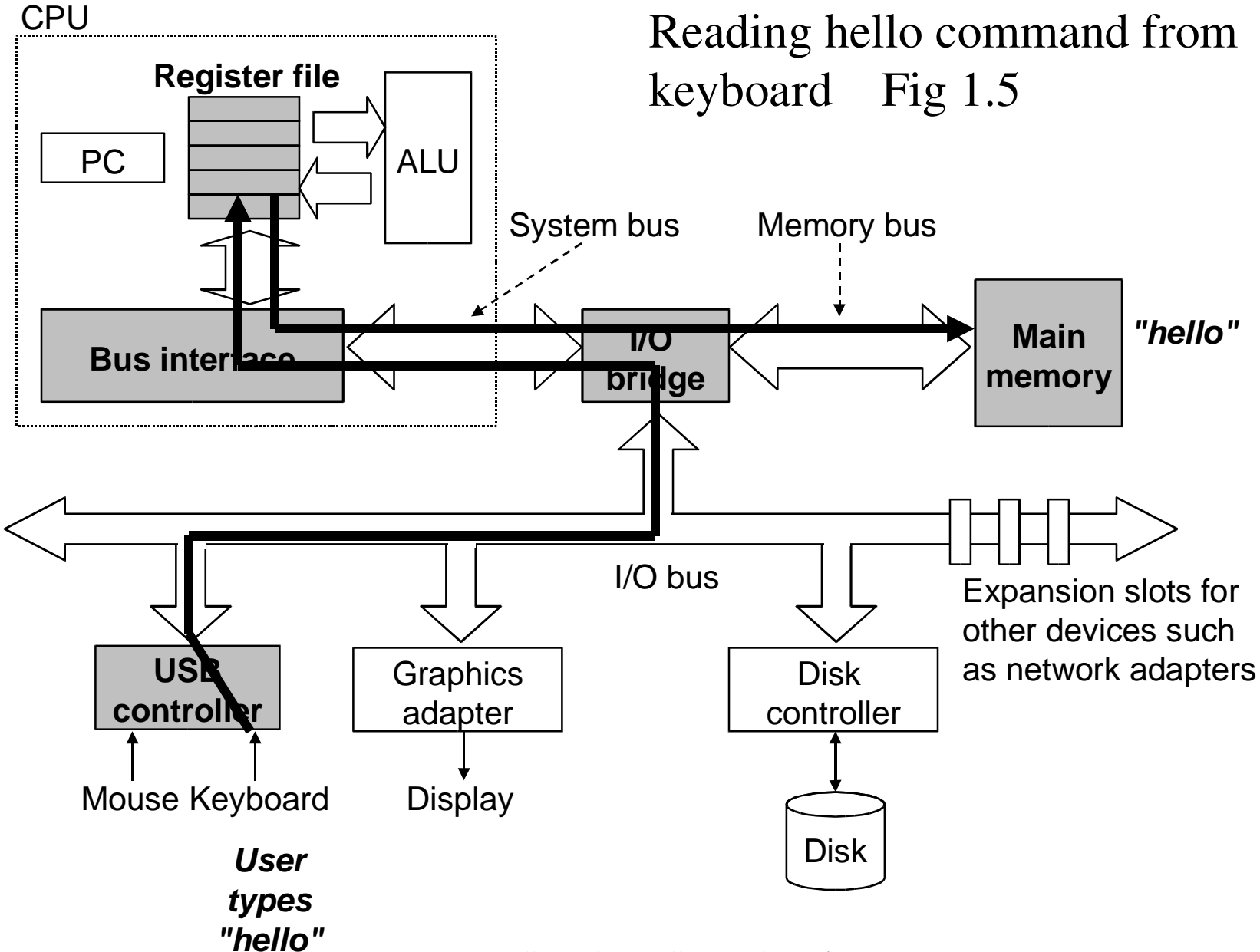
- Need memory to hold the sequence of instructions.
- Something to sequence through the instructions.
- Something to execute one instruction.
- Interface to the *outside world* (I/O).

Typical Organization of System

Fig. 1.4

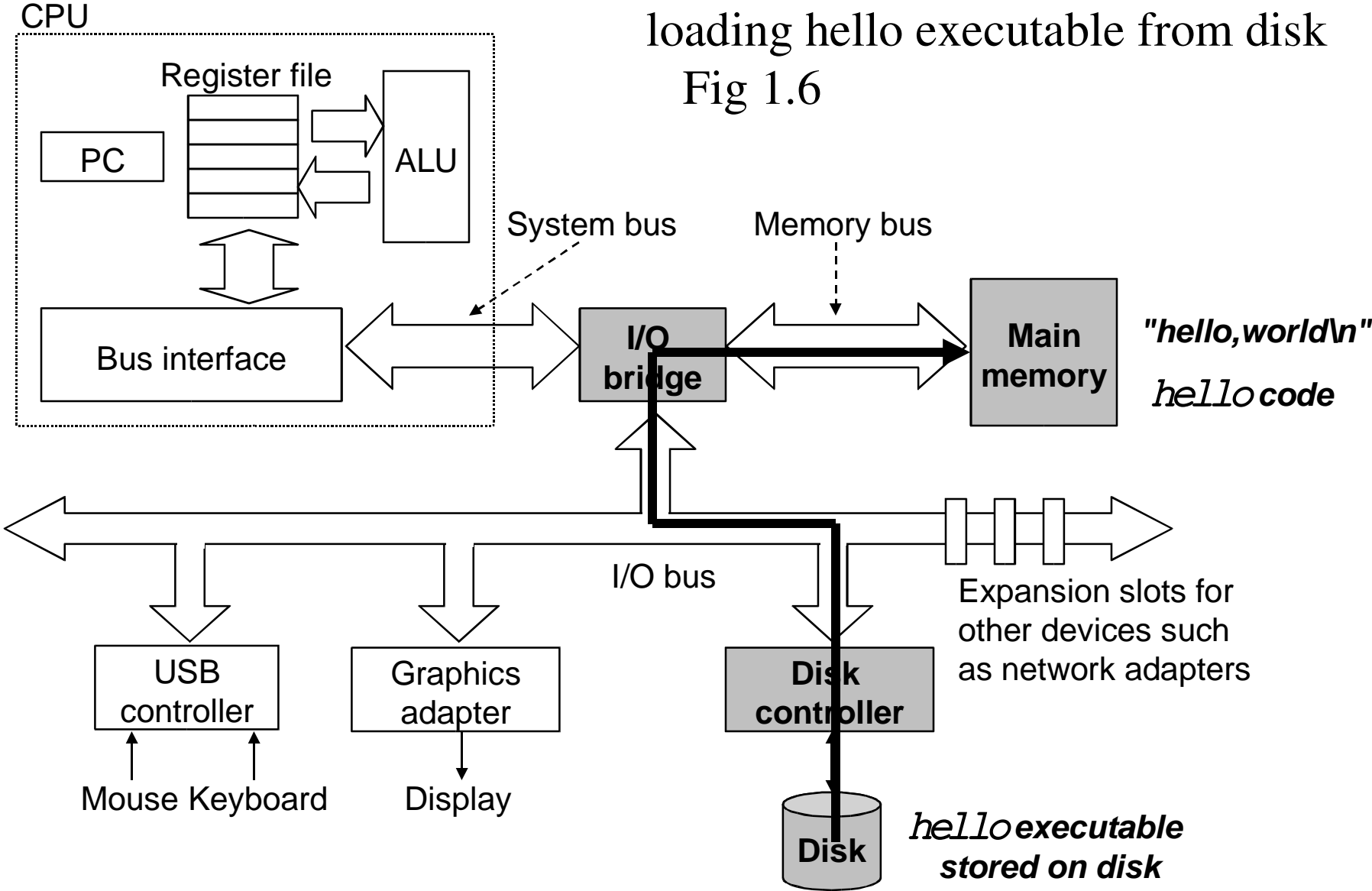


Reading hello command from keyboard Fig 1.5



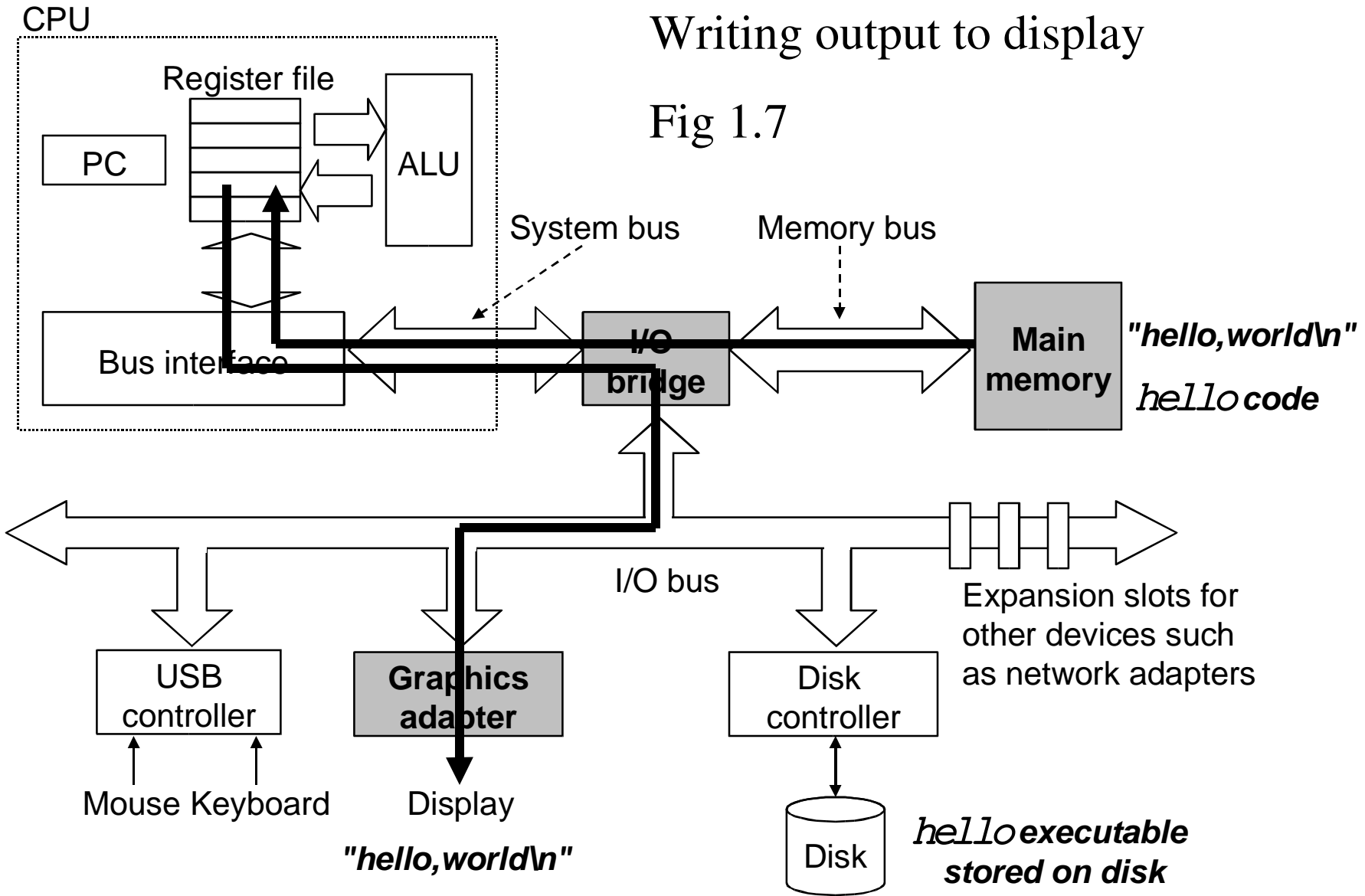
loading hello executable from disk

Fig 1.6



Writing output to display

Fig 1.7



Memory: the key to competence

“To be a good programmer you must understand the organization of instructions and data in memory”

- Dave Matthews

from the CD Remember Two Things

Real Instructions and Data

We will spend lots of time on real instruction sets (IA32).

Data representation is determined by processor architecture.

For now – we need to understand that each piece of data is just a bunch of bits (data looks just like instructions).

Interpreting Data

- It's impossible to tell what a chunk of bits means without any other information.

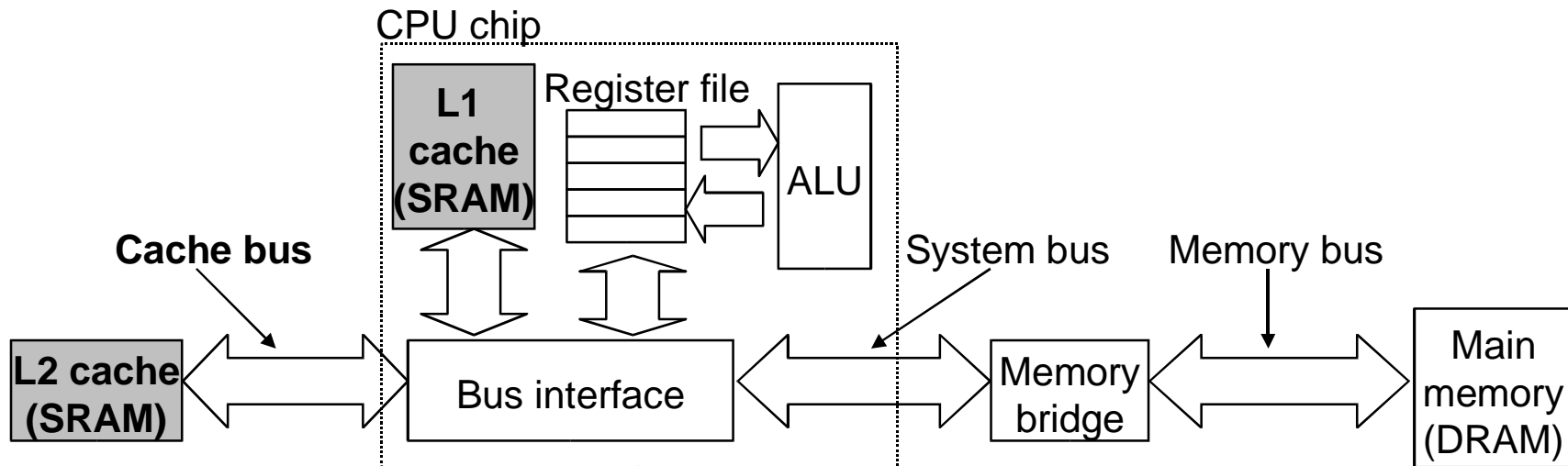
01001000

- Could be the integer 80
- Could be an instruction.
- Could be the character 'A'
- Could be a floating point number.

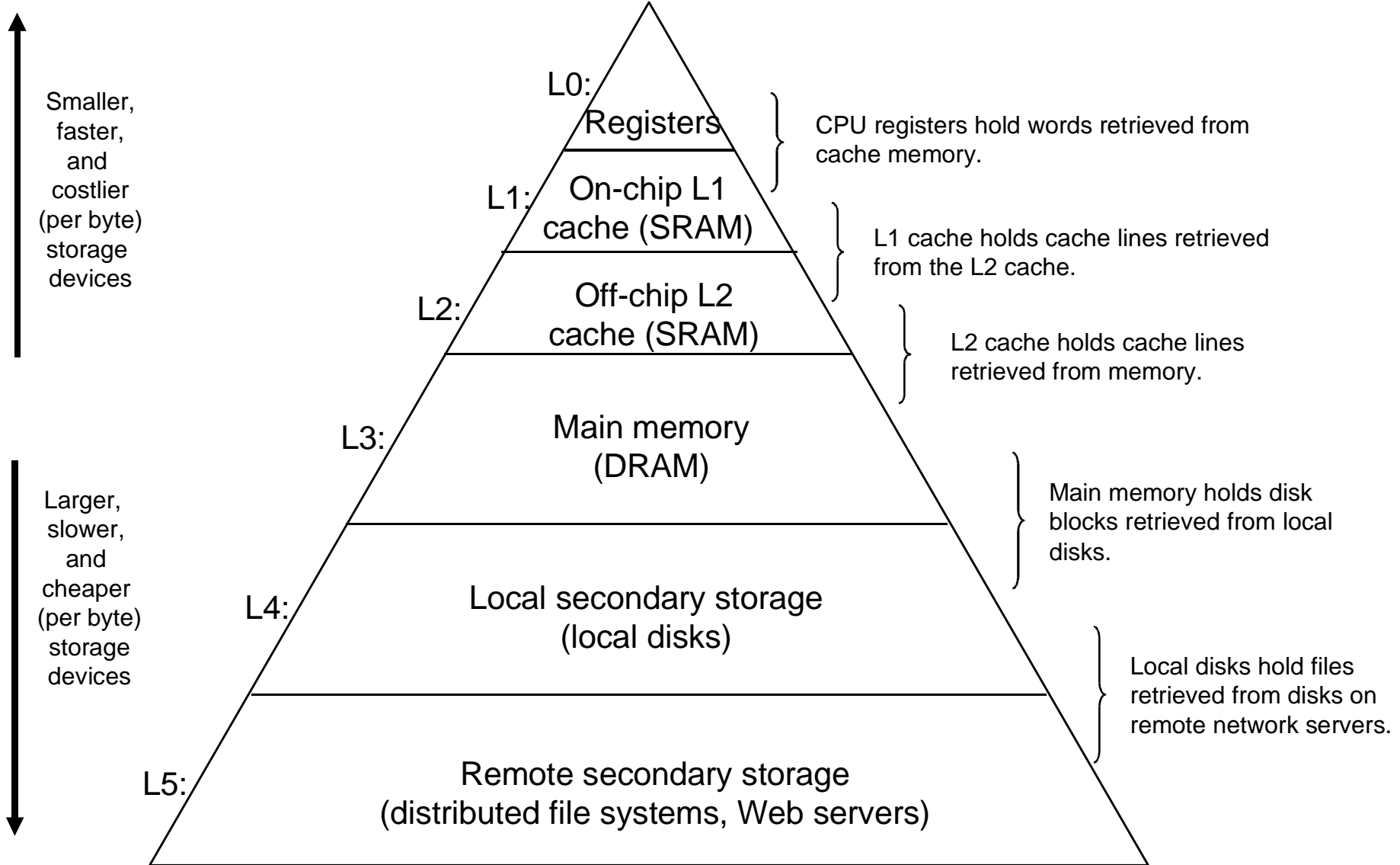
Context determines data type

- A computer executes instructions sequentially – it just grabs the bits in the next memory location and assumes it's an instruction.
- Some instructions add integers – in this case the computer assumes the bits in memory represent integer numbers.

Caches



Memory Hierarchy



Operating System

- One of the functions of the operating system (which is itself a running program) is to move programs in to memory and start them running.
- The operating system must manage the memory of the computer (the OS itself must be in memory!).

The power of two

Everything is based on powers of 2

There are special names for some powers of 2:

1 Kilo (as in Kilobyte) is 2^{10}

1 Mega is 2^{20}

1 Giga is 2^{30}

Machine Language and Humans

In the old days... humans would write programs in machine code (determine every 1 and 0).

The 1s and 0s were put in to memory with a set of switches!

Assembly Language

- An assembler is a program that reads a text file containing symbolic descriptions of instructions (much easier for humans to write).
- The output of an assembler is machine code (the 1s and 0s).

High-level languages

- Many high-level languages translate program text to assembly language (and then run through an assembler).
- C compilers often do this, we will test this out by generating assembly language from C code.