CHAPTER 1

INFORMATION IS BITS + CONTEXT

- Back to the basics:

HelloWorld.c begins as a source program (source file) - human readable

```
#include <stdio.h>
int main( )
{
  printf(“hello, world\n”);
  return 0;
}
```

Text is represented using ascii values (0-255) each represents 8 bits or 1 byte of data per character

HELLOWORLD.C IN ASCII

#	i	n	c	l	u	d	e	Sp	<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46
h	>	\n	\n	i	n	t	Sp	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123
\n	Sp	Sp	Sp	Sp	p	r	i	n	t	f	(	“	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108
l	o	,	Sp	w	o	r	l	d	\	n	“	)	;	\n	Sp
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	32
Sp	Sp	Sp	r	e	t	u	r	n	Sp	0	;	\n	}	\n	
32	32	32	114	101	116	117	114	110	32	48	59	10	125	10	

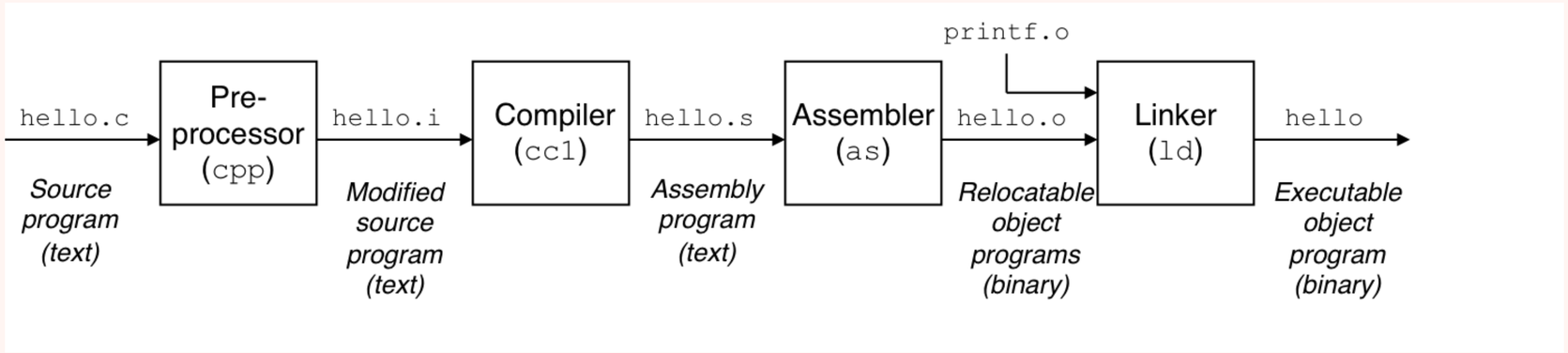
ANOTHER VIEW

CHARACTER	#	i	n	c	l	u
ASCII VALUE	35	105	110	99	108	117
BINARY VALUE	0010 0011	0110 1001	0110 1110	0110 0011	0110 1100	0111 0101
HEXADECIMAL VALUE	2 3	6 9	6 E	6 3	6 C	7 5

1.1 INFORMATION IS BITS + CONTEXT

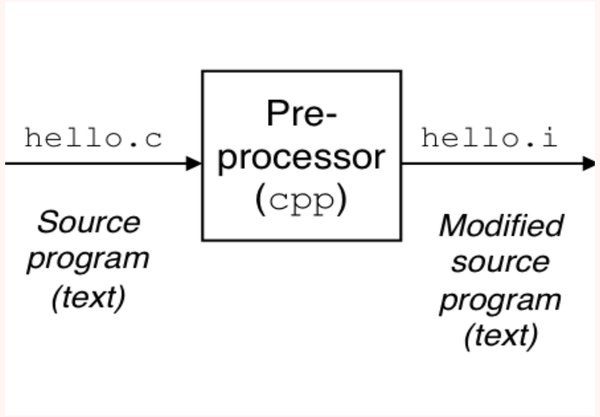
- All information in a system is represented as a bunch of bits
- The thing that distinguishes different data is the CONTEXT in which we view the data. This is what is important.
  - Example: A particular sequence of bytes could represent a range of different type of data. The context of the byte is what determines what it actually is.
- We will learn more of this in chapter 2.

1.2 PROGRAMS ARE TRANSLATED BY OTHER PROGRAMS INTO DIFFERENT FORMS



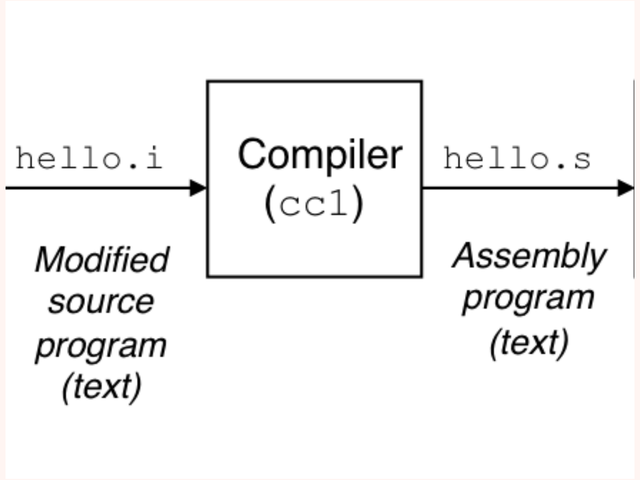
- 4 phases to the compilation system
  - Pre-processor
  - Compiler
  - Assembler
  - Linker
- We can see the files produced at each stage (gcc -save-temps hello.c -o hello)
- **Hello.c example**

COMPILATION SYSTEM - PHASE 1 PRE-PROCESSING



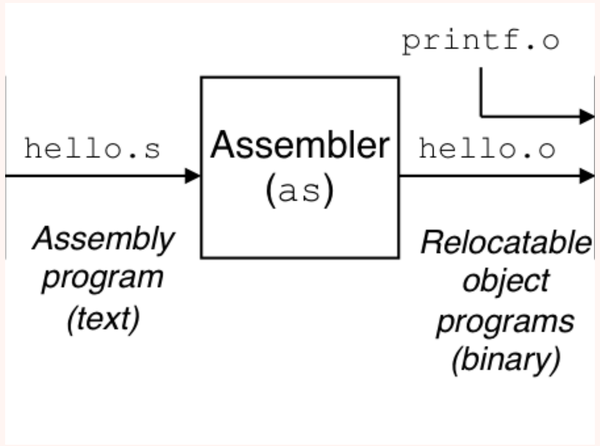
- Preprocessing (cpp) phase (not to be confused with C++)
  - Modifies the original C program as per the preprocessing directives (starts with #)

COMPILATION SYSTEM - PHASE 2 COMPILATION



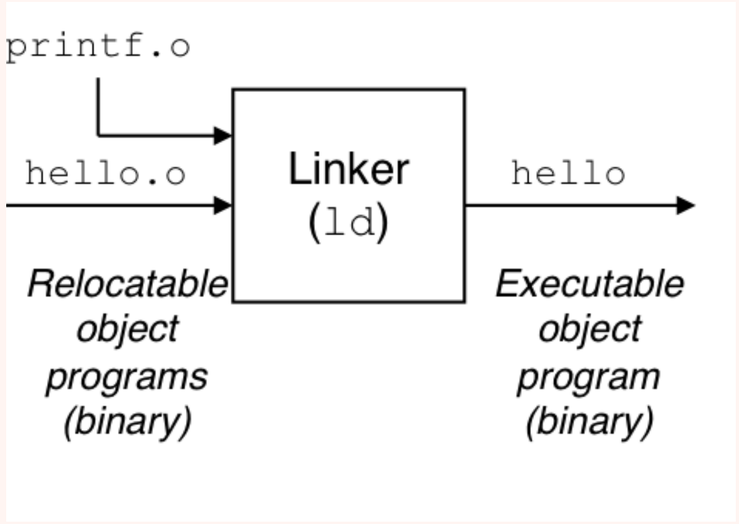
- Compilation (ccl) phase
  - Translates the text (hello.i) file into an assembly-language program
  - Produces a hello.s file
  - Each line of assembly defines one low-level machine-language instruction in textual form
  - Assembly produces a common output for different compilers for different high-level languages (C and Fortran produce output in the same assembly language)

COMPILATION SYSTEM - PHASE 3 ASSEMBLY



- Assembly (as) Phase
  - The assembler translates the assembly code (.s) into machine language instructions
  - Then packages the instructions in a form known as a relocatable object program
  - Stores the result in the object file in a (.o) file
  - Object files appear to be gibberish if opened

COMPILATION SYSTEM - PHASE 4 LINKER



- Linker (ld) Phase
  - When you call external functions (printf) in your program, these functions have object files that need to be merged with your program. This is what the linker does.
  - This phase produces the actual executable file
  - IMHO, errors produced from this section of the compiler process is the most irritating errors to debug (other than segfaults)

1.3 IT PAYS TO UNDERSTAND HOW COMPILATION SYSTEM WORK

- Optimizing program performance
  - Compilers usually produce optimal code
  - While we do not need to know all of the inner workings of a compiler it is advised that you have a basic understanding of machine-level code. This understanding could help you make good programming decisions. Ex. Is a switch statement better than if/else, while loop better than for loop.
  - Some of this material we may cover in the following chapters.

1.4 PROCESSORS READ AND INTERPRET INSTRUCTIONS STORED IN MEMORY

- Now that we have the program written, compiled, and an executable created, we are ready to execute it (run it)

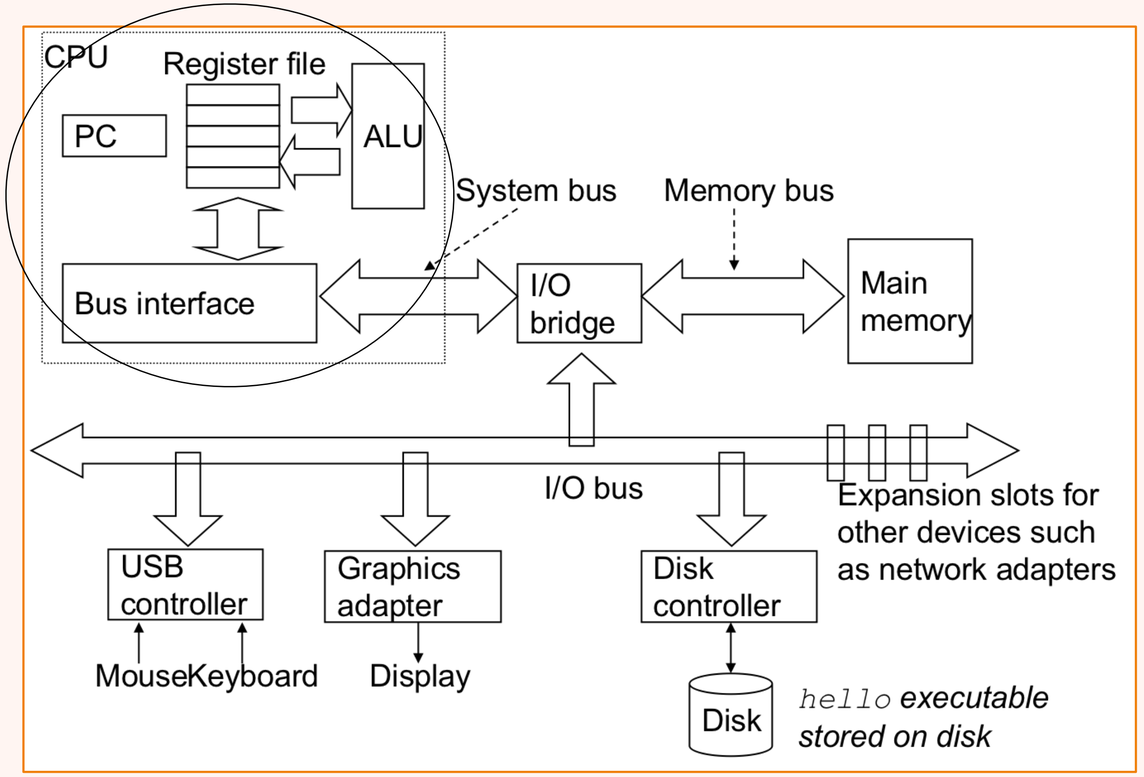
- We go to a terminal - which is basically an application shell - command-line interpreter that prints something like:

    yfeaste@cerf15: ~[1]

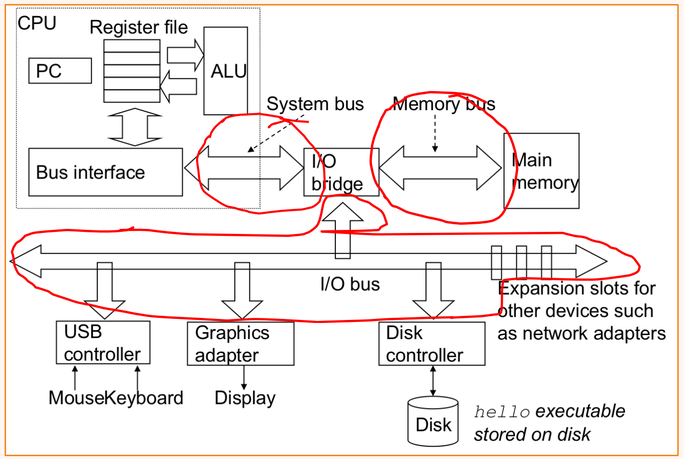
- If the first argument on the command line is not some built-in shell command (ls, mkdir, rm, cd, etc.) then it assumes it is an executable and attempts to load and run the program

1.4.1 HARDWARE ORGANIZATION OF A SYSTEM

- CPU - central processing unit
  - ALU - arithmetic/logical unit
  - PC - program counter
  - USB - universal serial bus



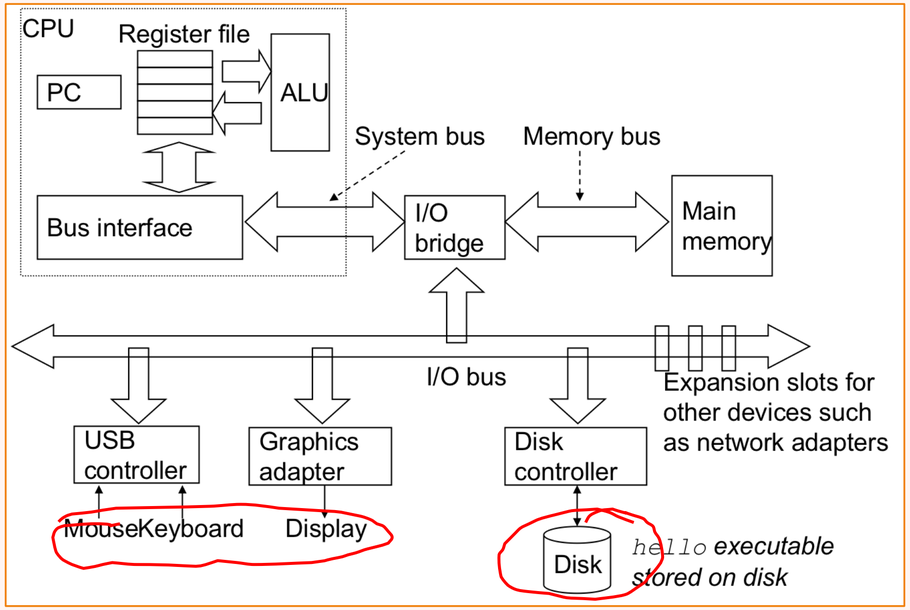
SYSTEM BUSES



- Buses - Electrical conduits(channels) that carry bytes of information between components.
- They transfer fixed-sized chunks of bytes known as words (32 bits - 4 bytes or 64 bits - 8 bytes). The size of a word varies depending on the system. We will talk more about words later.
- Not too many years ago buses used cables - shown in the pictures above.
- Now Buses are built into the motherboards. Devices connect directly to the board.

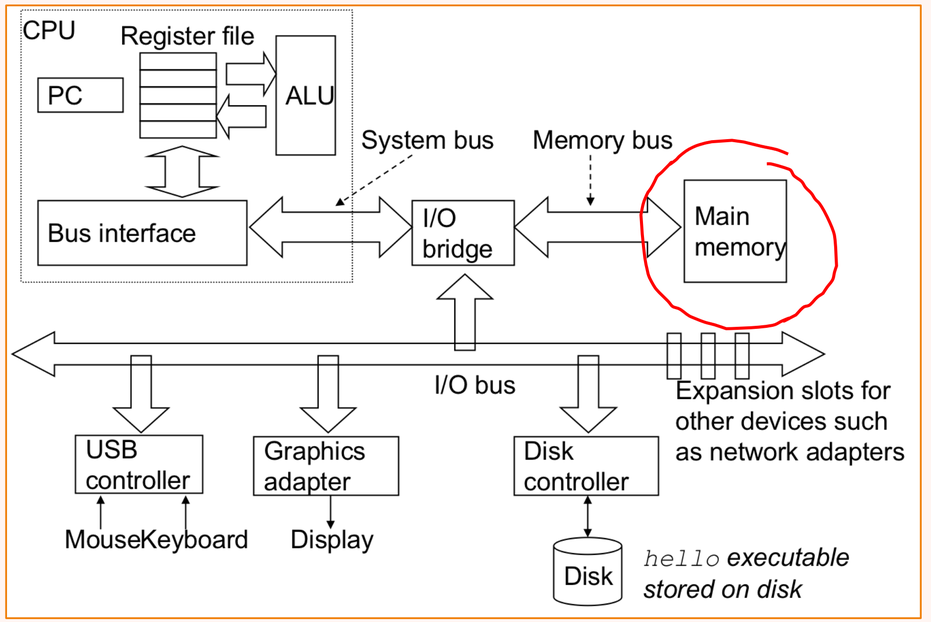
SYSTEM I/O DEVICES

- I/O devices - input/output devices are connected to the system from the outside world
- This example has 4
  - Mouse
  - Keyboard
  - Display
  - Disk drive



SYSTEM MAIN MEMORY

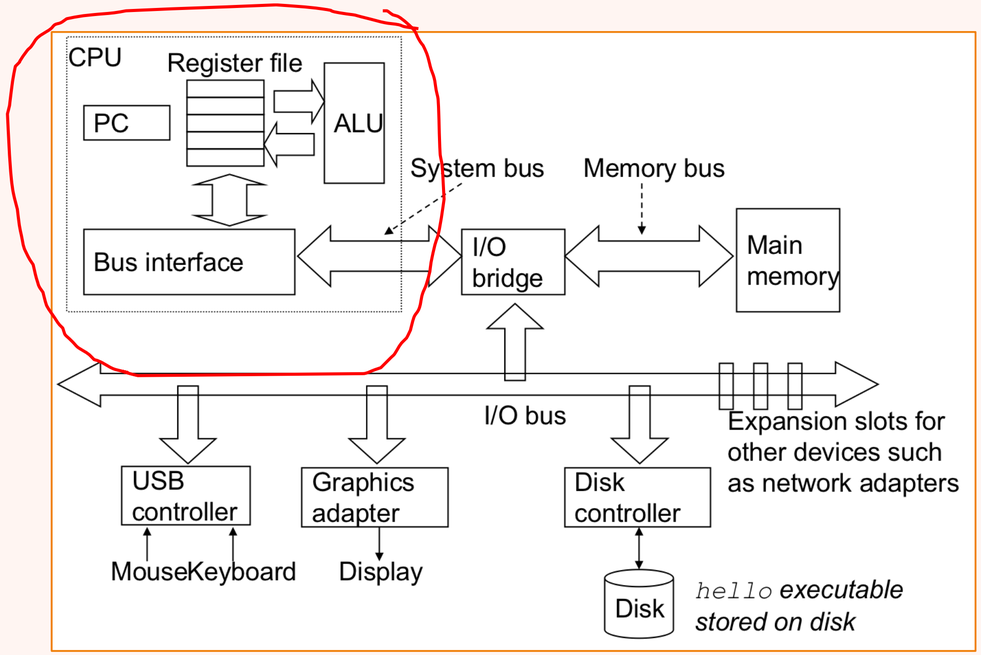
- Main Memory - temporary storage device that holds a program and the data it manipulates while the processor is executing the program
- Physical - main memory consist of a collection of dynamically random access memory (DRAM) chips
- Logical - memory is organized as a linear array of bytes. The Size of the logical memory depends on the type of data being stored: char, int, float, etc



SYSTEM PROCESSORS

- CPU - Central processing unit. This is the engine that executes the instructions stored in the main memory. It has a word size register called the PC (program counter) that contains the address of some instruction in main memory.

- Executes the instruction being pointed to, upon completion, the program counter moves to the next instruction in memory and repeats the process.




```
PC —->  subq   $16,    %rsp
        movl   $0,     -4(%rbp)
        movl   $10,    -8(%rbp)
        movl   $10,    -12(%rbp)
```

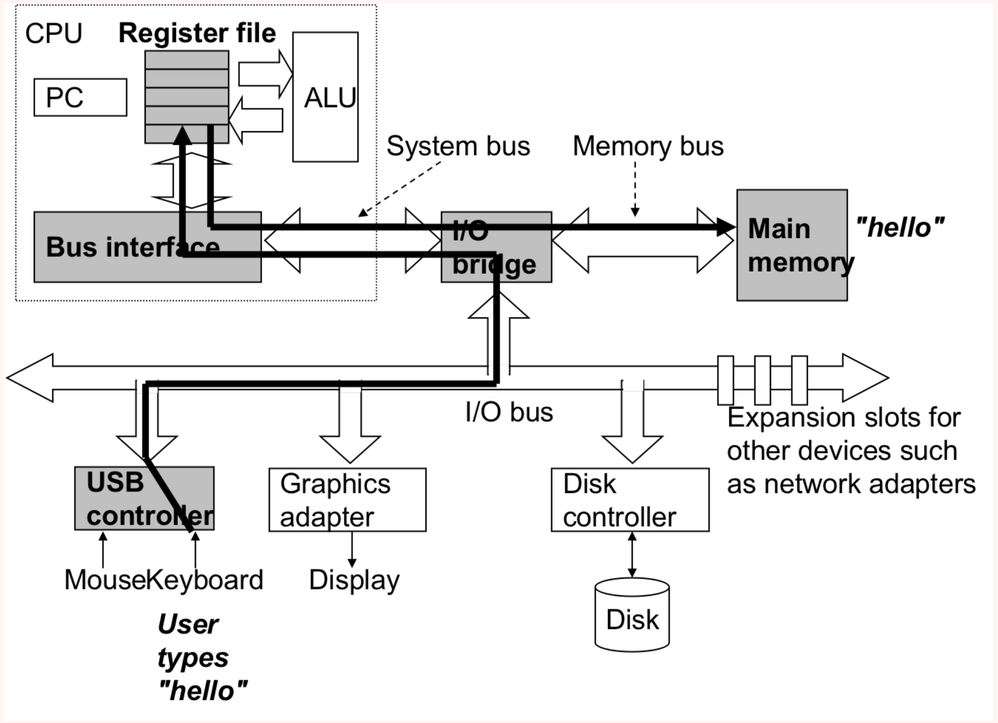
SYSTEM PROCESSORS

- Some simple operations that the CPU might carry out at the request of an instruction:
  - Load - copy a byte or word from main memory into a register, overwriting the previous content of the register
    - movl -8(%rbp), %eax
  - Store - copy a byte or a word from a register to a location in main memory, overwriting the previous contents of that location
    - movl %eax, -8(%rbp)
  - Operate - Copy the contents of two registers to the ALU, perform an arithmetic operation on the two words, and store the results in a register, overwriting the previous data
    - imulq %rdi, %rax
  - Jump - Extract a word from the instruction itself and copy that word into the PC, overwriting the previous value. Causes the execution to switch to a completely new position in the program.
    - cmpq %1, %rdi
    - jg .L2

RUNNING THE HELLO WORLD PROGRAM

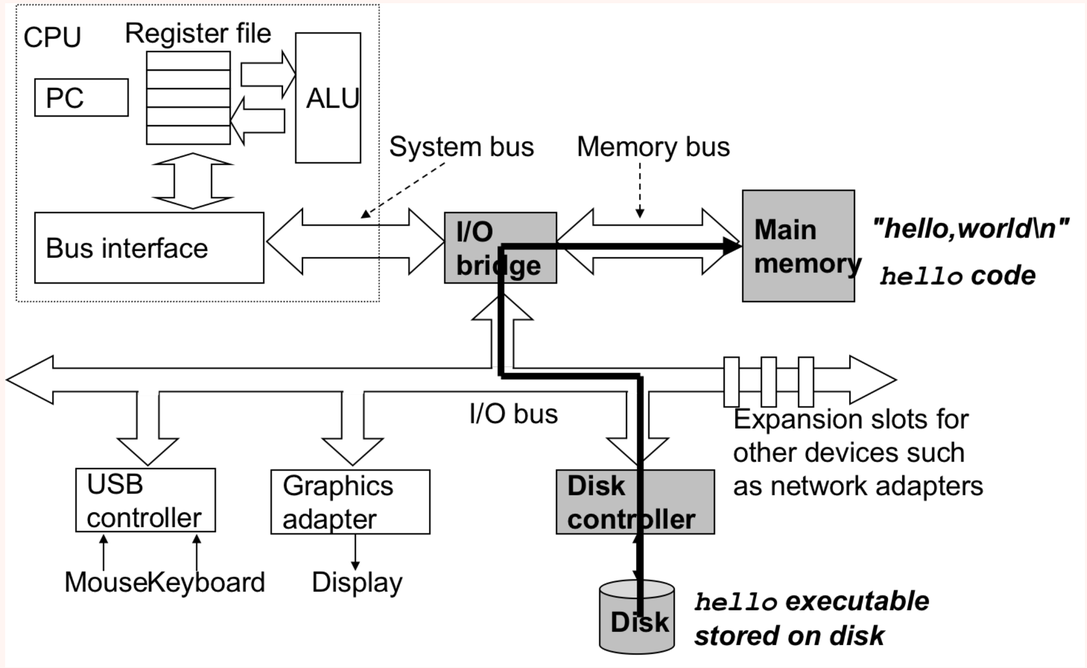
- Now we will explore what happens when we run the hello world program:

- Remember we are using the command line and a shell program is running. As you type characters on the command line the characters are being read and saved in a register and then copied to main memory.



RUNNING THE HELLO WORLD PROGRAM

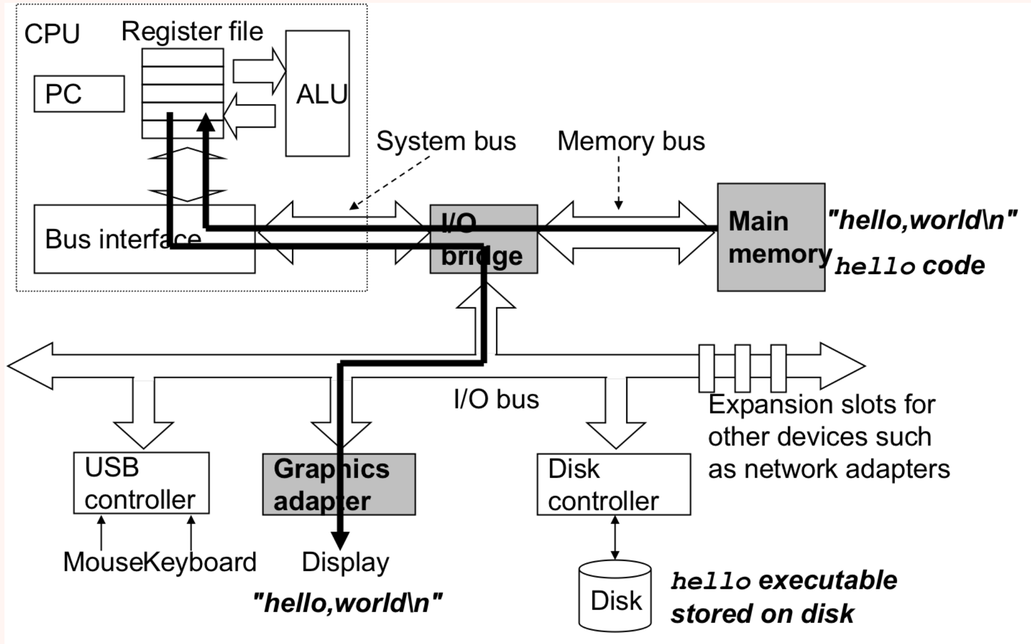
- When we hit the enter key the shell knows we are finished defining the command.
- The shell then loads the executable file by coping the code and data in the hello object file, which is stored on the disk, to the main memory.
- This step uses direct memory access (DMA) and does not need to pass through the processor.



RUNNING THE HELLO WORLD PROGRAM

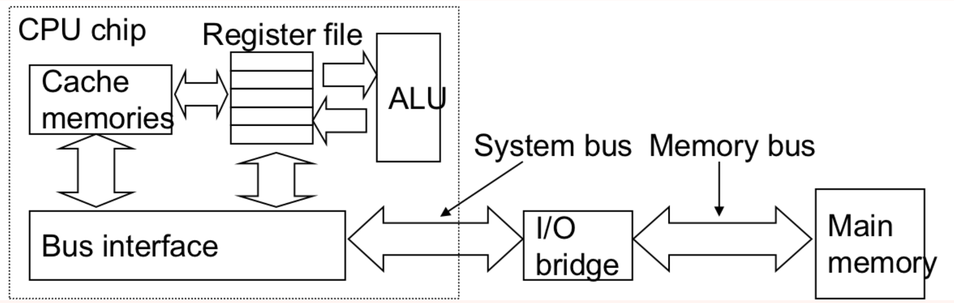
- Once the instructions from the object file are all loaded in memory, the processor begins executing the machine-language instructions in the hello program's main routine.

- In the end, the hello world string is produced and copied to the registers, then sent to the output display.



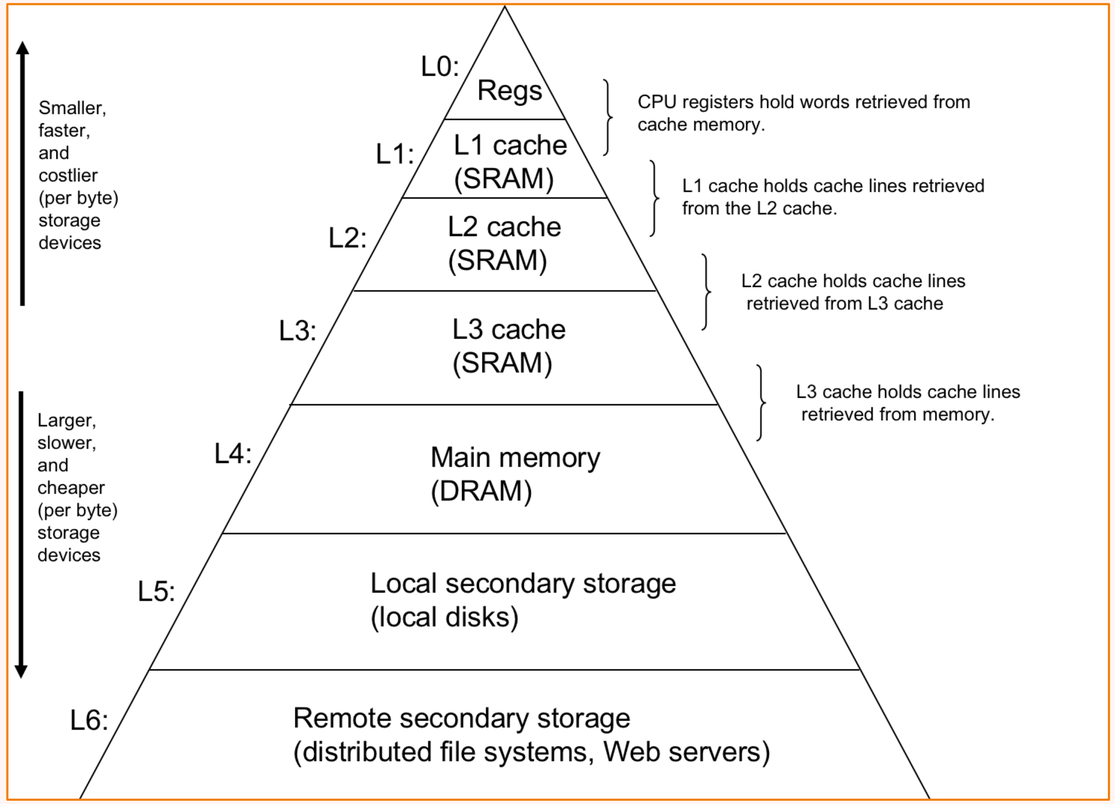
1.5 CACHES MATTER

- As we can see from previous slides a very simple example program can spend a great deal of time moving information from one part of the computer system to another.
- All of this copying is overhead that slows down the “real work” of the program.
- System designers designed smaller faster storage devices called cache memories to serve as temporary staging areas for information that the processor is likely to need in the near future.
- Algorithms try to predict what information will be needed next and stores the info on the cache memory
  - Cache hit - if the algorithm predicts correctly - more efficient
  - Cache miss - if the algorithm predicts incorrectly



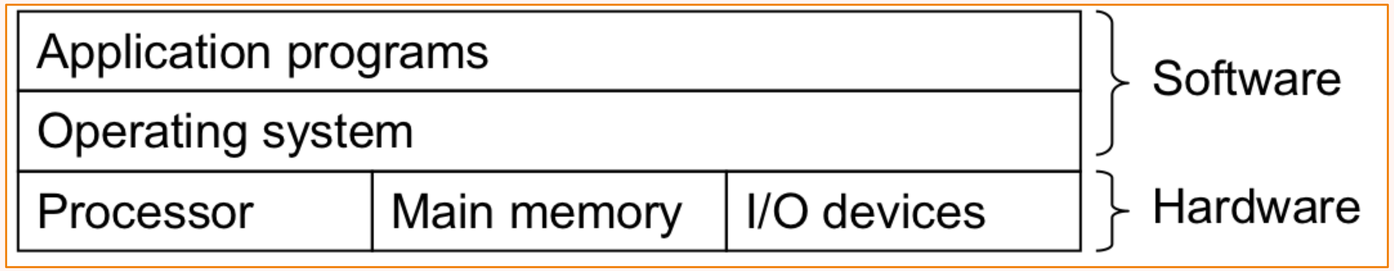
1.6 STORAGE DEVICES FROM A HIERARCHY

- As it turns out, the cache philosophy of storage worked out pretty well.
- Storage is often set up in a hierarchy manner with smaller, faster, more expensive memory at the top and larger, less expensive memory at the bottom.
- However, the main idea of memory hierarchy being that storage at one level serves as a cache for storage to the next lower level.
  - L3 holds info from main memory
  - L2 holds info from L3
  - L1 holds info from L2
  - L0 (registers) retrieves info from L1



https://www.extremetech.com/extreme/188776-how-l1-and-l2-cpu-caches-work-and-why-theyre-an-essential-part-of-modern-chips
https://hazelcast.com/glossary/memory-caching/

1.7 THE OPERATING SYSTEM MANAGES THE HARDWARE

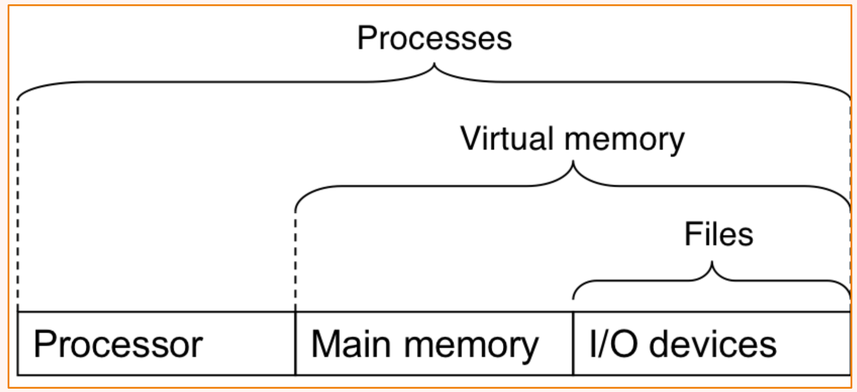


- All of this magic that happens between the shell program that prompted us to type a command, the moving of information from registers to main memory to the output could not happen without the help of operating system.

- The OS is a layer of software interposed between the application program and the hardware as depicted above.

1.7 THE OPERATING SYSTEM MANAGES THE HARDWARE

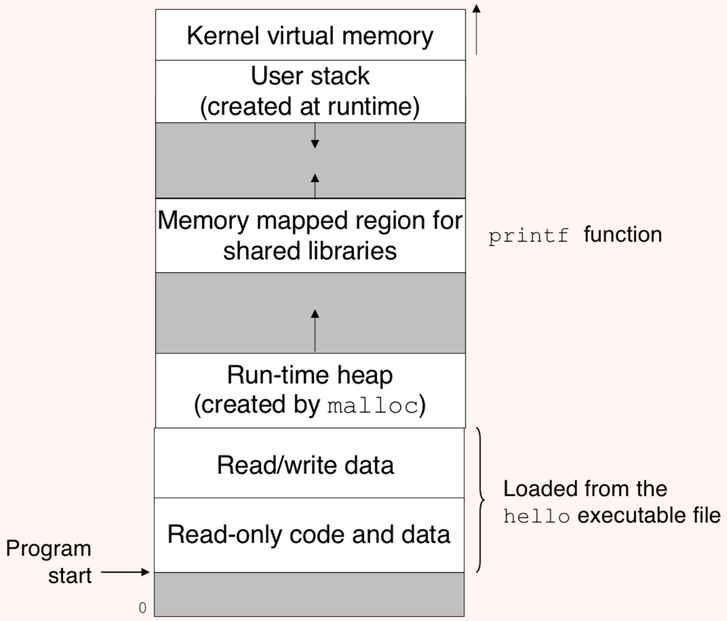
- The OS has 2 primary purposes
  - To protect the hardware from misuse by runaway applications
  - To provide applications with simple and uniform mechanisms for manipulating complicated and often wildly different low-level hardware devices (device drivers)
- It achieves its goals through the abstraction of files, virtual memory, and processes



VIRTUAL MEMORY

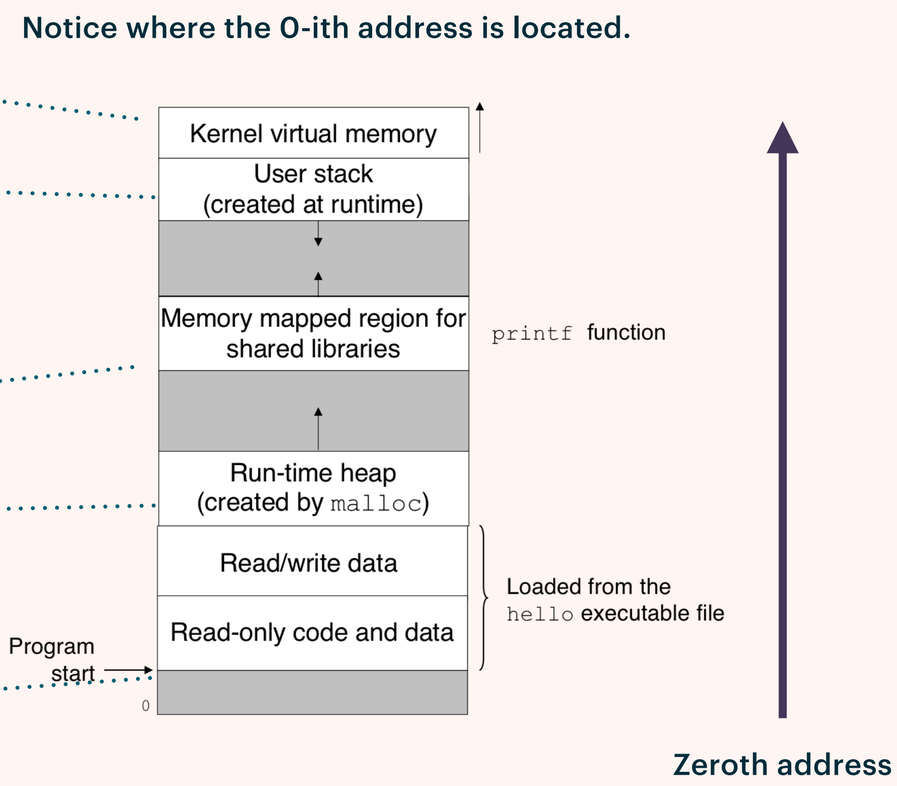
- Provides each process with the illusion that it has use of the main memory
- Uses both hardware and software to enable a computer to compensate for physical memory shortages, temporarily transferring data form random access memory to disk storage. Mapping chunks of memory to disk files enables a computer to treat secondary memory as though it were main memory. https://www.techtarget.com/searchstorage/definition/virtual-memory
- The illustration used here is the virtual address space for Linux.
- You will learn more about this in OS class
- We will do a quick overview

View of the virtual address space for Linux
Others architectures are similar



VIRTUAL MEMORY

- **Kernel Virtual Memory**
  - Reserved for the OS
- **Stack**
  - Grows and shrinks as needed - grows down
  - Used to implement function calls
- **Shared Libraries (stdio, stdlib, etc.)**
- **Heap**
  - Grows and shrinks as needed dynamic memory stored here - grows up
- **Program code and data**
  - Size does not change after programs get started



OTHER TOPICS COVERED

- The books covers several other topics we will not cover
  - Processes
  - Threads
  - System Communication
    - Locally
    - Globally
  - Concurrency and Parallelism
  - Multi-Core Processes
  - Hyper-threading

ASIDE - ORIGINS OF THE ‘C’ PROGRAMMING LANGUAGE

- C was developed from 1969 to 1973 by Dennis Ritchie of Bell Laboratories
- It was ratified by the American National Standards Institute (ANSI) in 1989 and later by the International Standards Organization (ISO)
- The C language was determined to be a set of library functions known as “C standard library”
- C was written originally for Bell Laboratories’ Unix Operating System
  - Often used for system-level programming
  - Later people found it useful for other application programming projects
- It was seen as a small, simple language
- Brian Kernighan and Dennis Ritchie wrote a book about ‘C’ that is still popular (“The C Programming Language”)

ASIDE - ORIGINS OF THE ‘C PROGRAMMING LANGUAGE

- The ISO issued an update of C in 1999 - known as “ISO C99”
  - Introduced new data types and provided support for text strings requiring characters not in the English language
- Updated again in 2011 - “ISO C11”
  - Added more data types and features
- Most new additions have been backward compatible
  - Programs written according to the earlier standards will have the same behavior when compiled using newer standards and the appropriate flags

C version	GCC command line option
GNU 89	None, -std=gnu89
ANSI, ISO C90	-ansi, -std= 89
ISOC99	-std=c99
ISO C11	-std=c11