ARQCP Course

Arquitetura de Computadores
Licenciatura em Engenharia Informática

2024/25

Paulo Baltarejo Sousa
pbs@isep.ipp.pt

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Disclaimer

Material and Slides

Some of the material/slides are adapted from various:

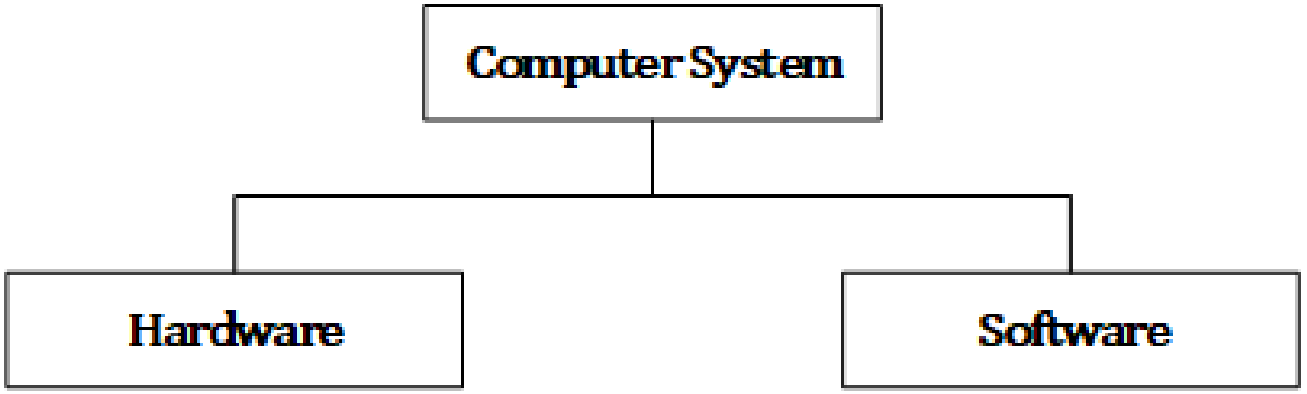
- Presentations found on the internet;
- Books;
- Web sites;
- ...

Outline

Introduction

What is?

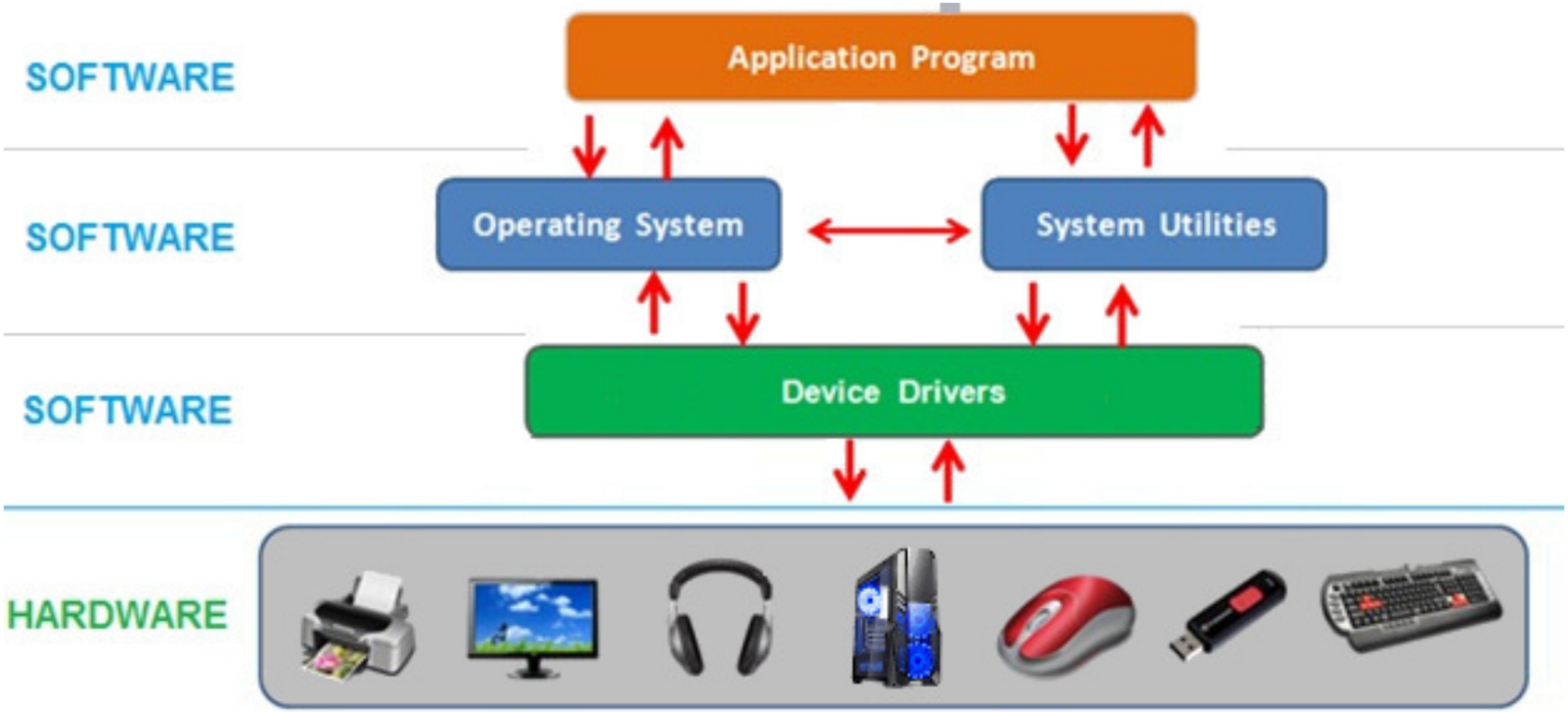
- A **computer system** consists of hardware and system software that work together to run application programs.



  - All physical components that forms computer system are known as computer **hardware**.
  - **Software** is basically collection of different programs that tells computer's hardware what to do.
- The way these components work and interact with each other affects the **correctness and performance of application programs**.

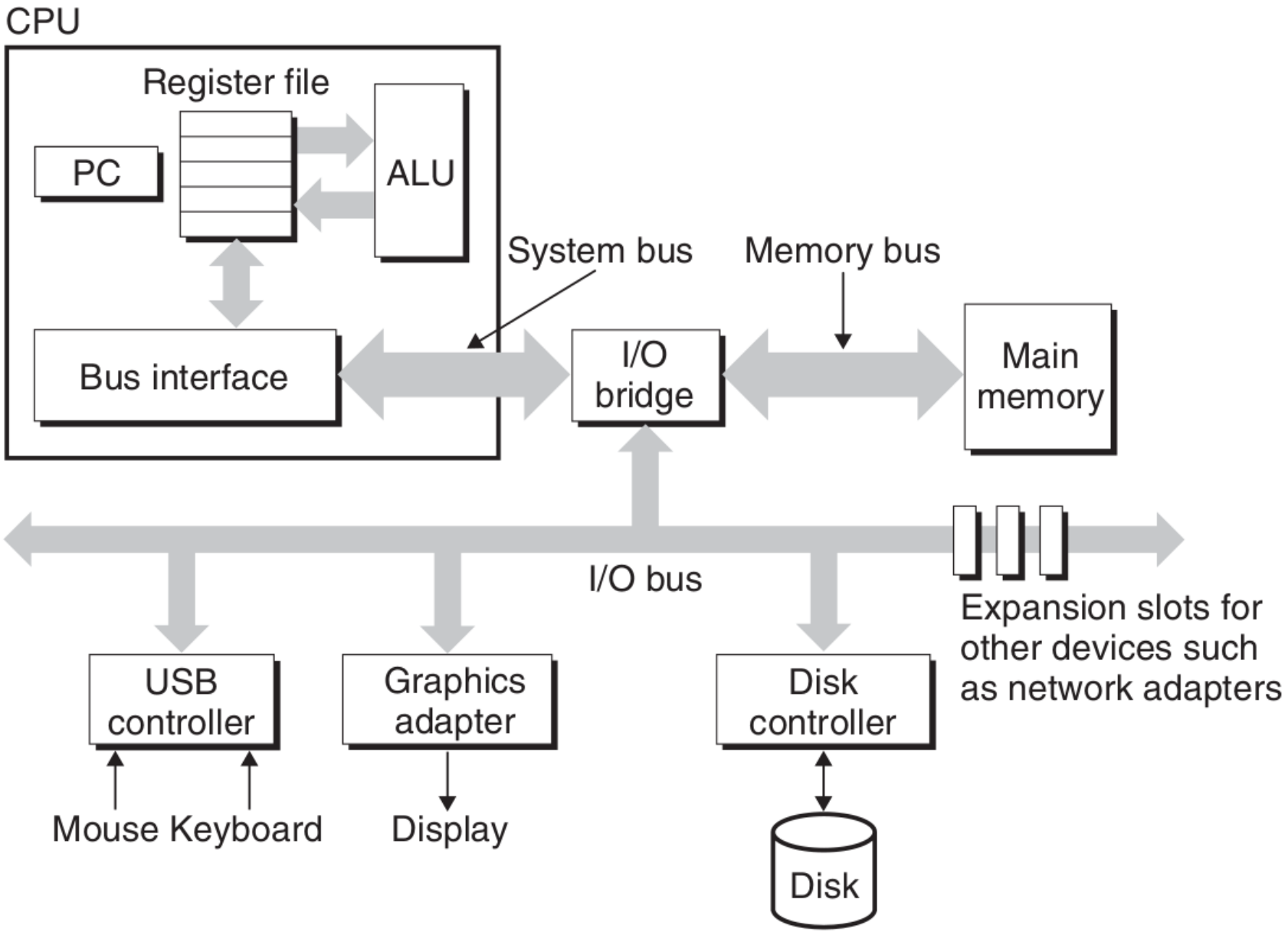
How is it structured?

- Typically, it follows a **layer architecture**.



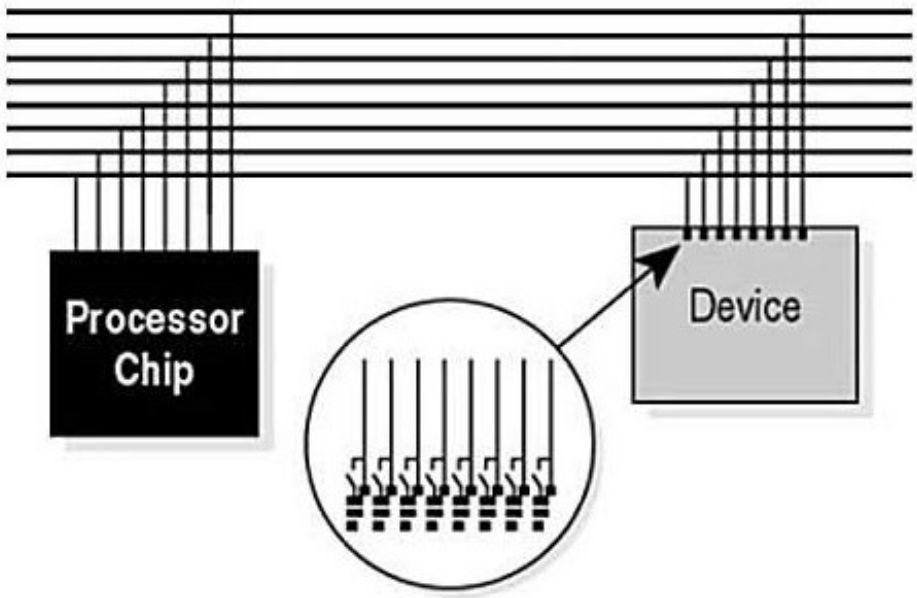
- Operating System, System Utilities, and Device Drivers are **out of scope of this course**.

Hardware Organization



Buses

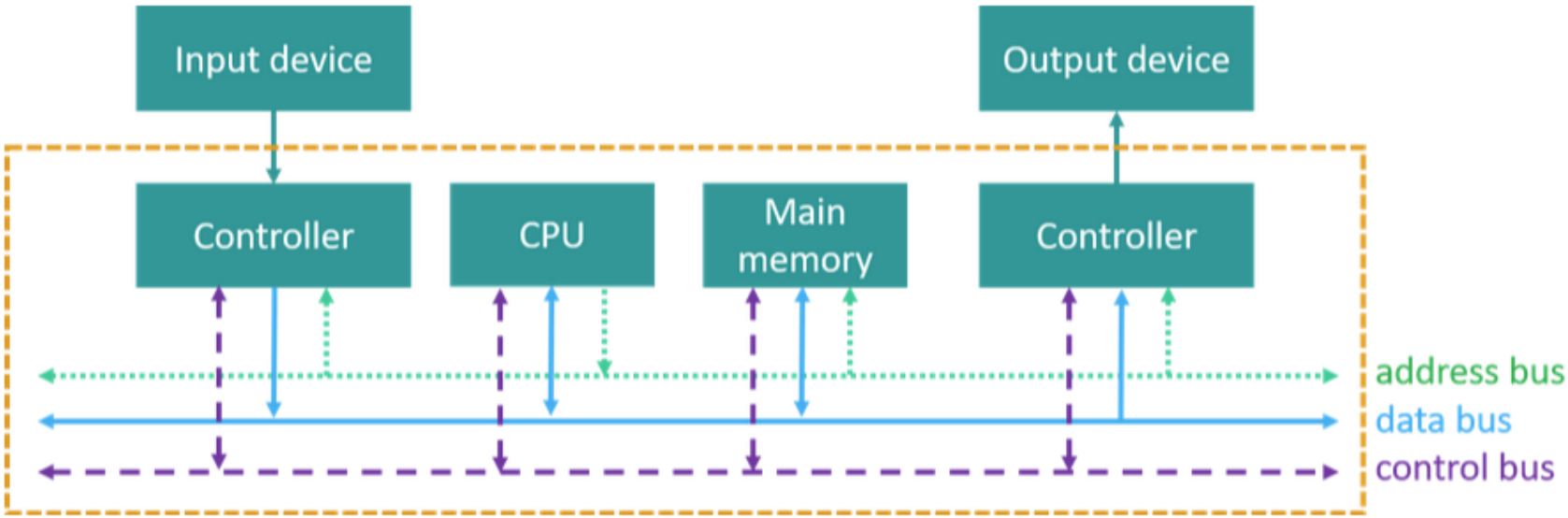
- Running throughout the system is a collection of electrical circuits that **carry bytes of information back and forth between the components**.



- Buses are typically designed to transfer fixed-sized chunks of bytes known as **words**.
  - The number of bytes in a word (the **word size**) is a fundamental system parameter that varies across systems.
  - Most machines today have word sizes of either 4 bytes (32 bits) or 8 bytes (64 bits).

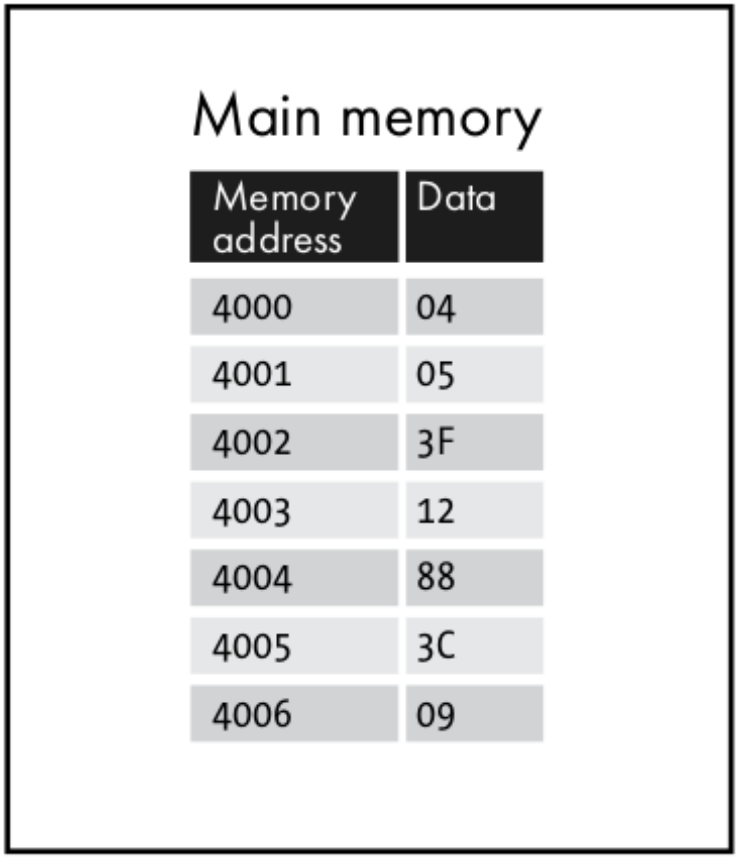
I/O Devices

- Input/output (I/O) devices are the system's connection to the external world.
- Each I/O device is connected to the I/O bus by either a **controller** or an **adapter**
  - The purpose of each is to transfer information back and forth between the I/O bus and an I/O device.



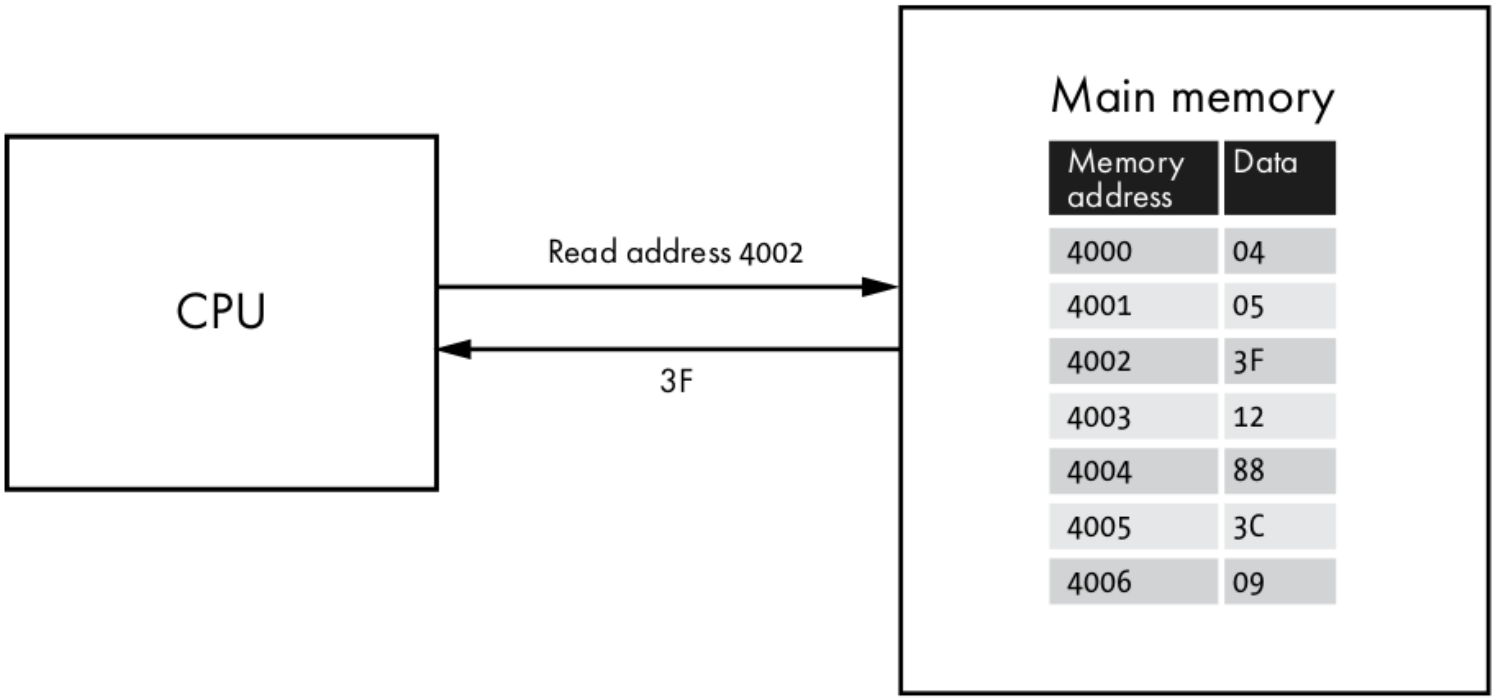
Main Memory

- The main memory is a **temporary storage device that holds both a program (instructions) and the data**.
- Logically, **memory is organized as a linear array of bytes**, each with **its own unique address** (array index) starting at zero.



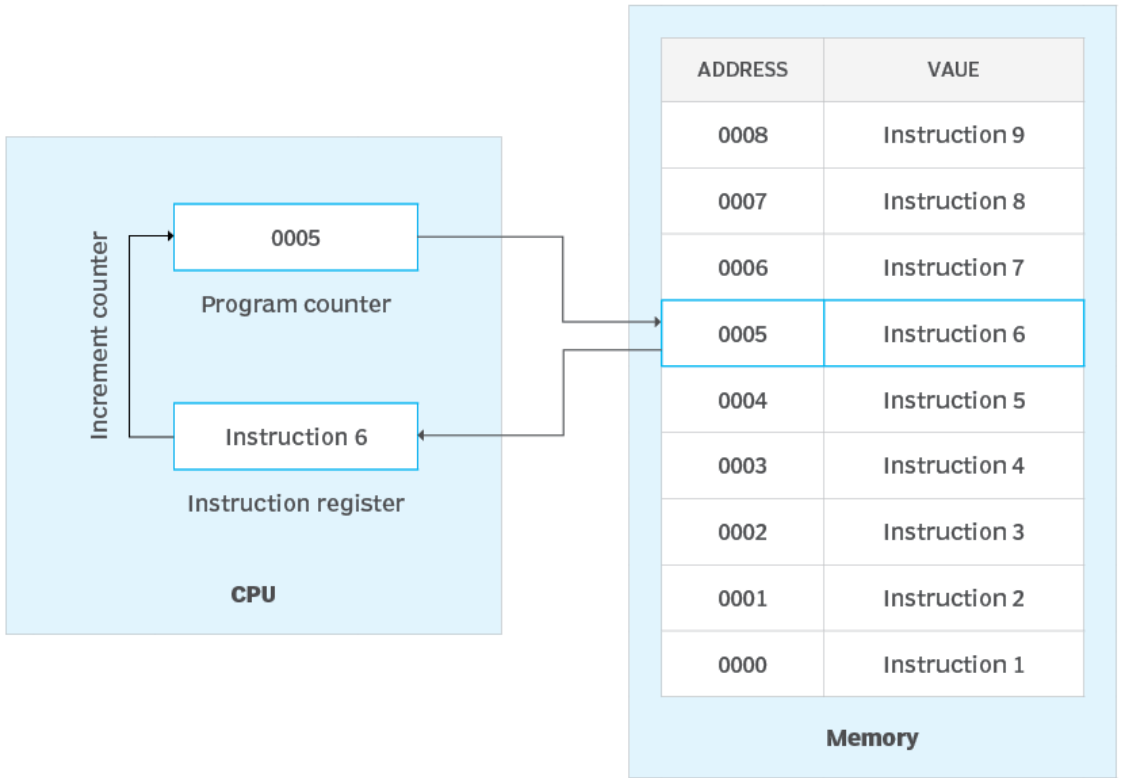
Processor (I)

- The **central processing unit** (CPU), or simply processor, is the **engine that interprets (or executes) instructions stored in main memory**.



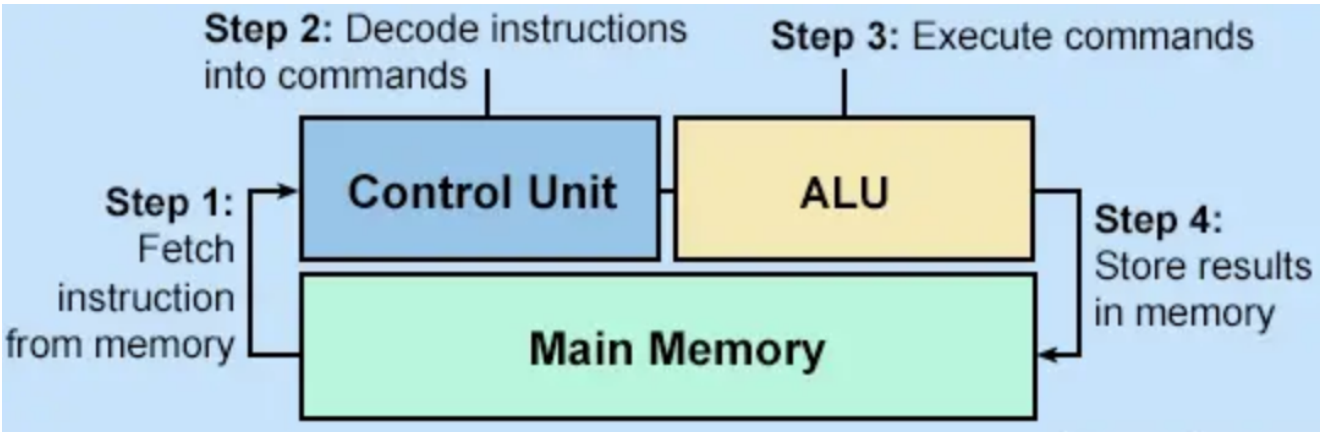
Processor (II)

- At its core is a word-sized storage device (or register) called the **program counter (PC)**.
  - PC contains the memory address (location) of the next program instruction to be executed.
  - After the CPU fetches the instruction, it increases the PC by 1 so it points to the next instruction in the program's sequence.



Processor (III)

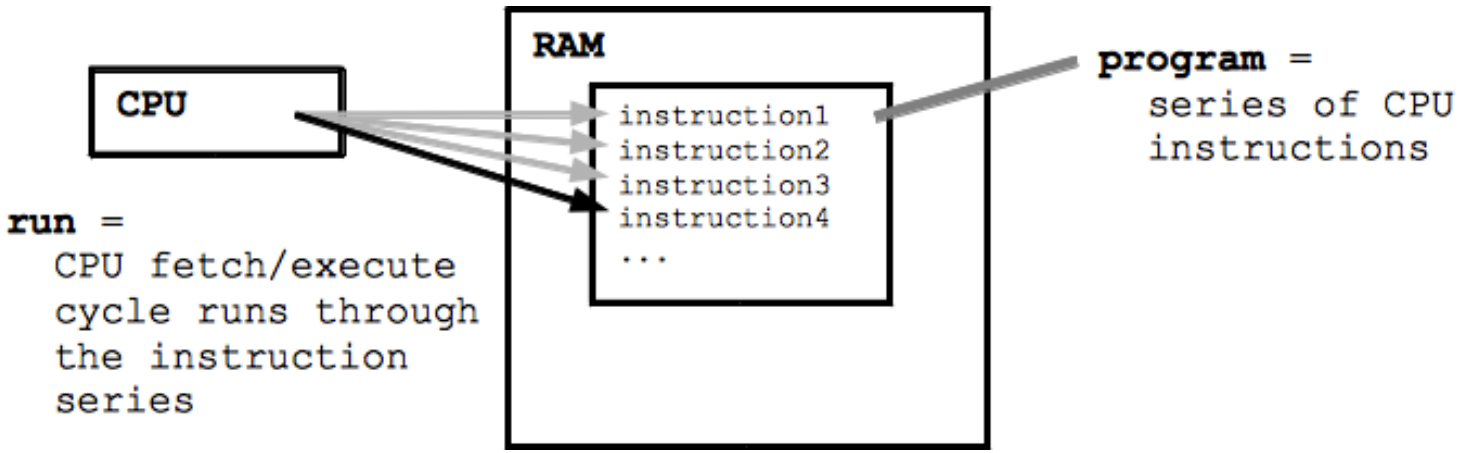
- Instruction cycle
  - The **instruction cycle is the time required by the CPU to execute one single instruction**.
  - The **instruction cycle is the basic operation** of the CPU which consists on four steps:
    - **Fetch** the next instruction from memory
    - **Decode** the instruction just fetched
    - **Execute** this instruction as decoded
    - **Store** the result



  - At the end of each instruction cycle **CPU advances PC register**.

How Computer Executes a Program (I)

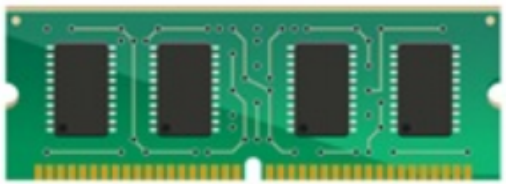
- A computer **program is a file**, in which its content is a **set of CPU instructions**.
- The program **instructions are loaded into the main memory** (RAM).
- The CPU **initiates the program execution by fetching the instructions one by one from memory to registers**.
- The CPU executes these instructions **by repetitively performing an instruction cycle**.
- **At the end of each instruction cycle it increments the PC register**



How Computer Executes a Program (II)

START

Disk Memory
Secondary Memory
Operating System Loads
Program Into RAM

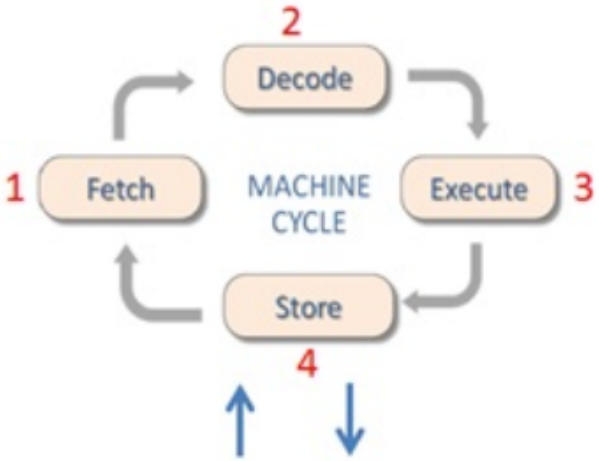


Primary Memory
Main Memory
RAM

PROGRAM

1 Instruction.
2 Instruction.
3 Instruction.
4 Instruction.
5 Instruction.
6 Instruction.

Program Is Set Of
Instructions Stored In The
Main Memory
RAM



Processor CPU

Instruction Set Architecture (ISA)

- An ISA is part of the abstract model of a computer that defines how the CPU is controlled by the software.
- The ISA acts as an interface between the hardware and the software, specifying both **what the CPU is capable of doing** as well as **how it gets done**.
  - The ISA defines the **set of commands that the CPU can perform to execute the program instructions**.
    - Instructions are **bit-patterns**.
- Different “families” of processors, such as Intel x86-64, IBM/Freescale PowerPC, and the ARM processor family have **different ISAs**.
  - A program **compiled for one type of machine will not run on another**.

Creating the `hello` Program

Source code file (I)

- When you create a program, you tell the **computer what to do**.
- Using a text editor, you create what is called a **source code** file.
  - The only special thing about this file is that it has to contain **statements according to the programming language rules**.
    - In this case, the statements are written using the C programming language, so the source code file must be saved with ".c" extension (in this case `hello.c`)

```c
#include <stdio.h>

int main()
{
   printf("hello, world\n");
   return 0;
}
```

Source code file (II)

- The source code file is a **text file**, which **consist exclusively of American Standard Code for Information Interchange (ASCII) codes**.
  - The ASCII code **associates a bit representation for each symbol in the character set**, such as letters, digits, punctuation marks, special characters, and control characters.

Control Characters				Graphic Symbols											
Name	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex
NUL	0	0000000	00	space	32	0100000	20	@	64	1000000	40	‘	96	1100000	60
SOH	1	0000001	01	!	33	0100001	21	A	65	1000001	41	a	97	1100001	61
STX	2	0000010	02	"	34	0100010	22	B	66	1000010	42	b	98	1100010	62
ETX	3	0000011	03	#	35	0100011	23	C	67	1000011	43	c	99	1100011	63
EOT	4	0000100	04	$	36	0100100	24	D	68	1000100	44	d	100	1100100	64
ENQ	5	0000101	05	%	37	0100101	25	E	69	1000101	45	e	101	1100101	65
ACK	6	0000110	06	&	38	0100110	26	F	70	1000110	46	f	102	1100110	66
BEL	7	0000111	07	'	39	0100111	27	G	71	1000111	47	g	103	1100111	67
BS	8	0001000	08	(	40	0101000	28	H	72	1001000	48	h	104	1101000	68
HT	9	0001001	09	)	41	0101001	29	I	73	1001001	49	i	105	1101001	69
LF	10	0001010	0A	*	42	0101010	2A	J	74	1001010	4A	j	106	1101010	6A
VT	11	0001011	0B	+	43	0101011	2B	K	75	1001011	4B	k	107	1101011	6B
FF	12	0001100	0C	,	44	0101100	2C	L	76	1001100	4C	l	108	1101100	6C
CR	13	0001101	0D	−	45	0101101	2D	M	77	1001101	4D	m	109	1101101	6D
SO	14	0001110	0E	.	46	0101110	2E	N	78	1001110	4E	n	110	1101110	6E
SI	15	0001111	0F	/	47	0101111	2F	O	79	1001111	4F	o	111	1101111	6F
DLE	16	0010000	10	0	48	0110000	30	P	80	1010000	50	p	112	1110000	70
DC1	17	0010001	11	1	49	0110001	31	Q	81	1010001	51	q	113	1110001	71
DC2	18	0010010	12	2	50	0110010	32	R	82	1010010	52	r	114	1110010	72
DC3	19	0010011	13	3	51	0110011	33	S	83	1010011	53	s	115	1110011	73
DC4	20	0010100	14	4	52	0110100	34	T	84	1010100	54	t	116	1110100	74
NAK	21	0010101	15	5	53	0110101	35	U	85	1010101	55	u	117	1110101	75
SYN	22	0010110	16	6	54	0110110	36	V	86	1010110	56	v	118	1110110	76
ETB	23	0010111	17	7	55	0110111	37	W	87	1010111	57	w	119	1110111	77
CAN	24	0011000	18	8	56	0111000	38	X	88	1011000	58	x	120	1111000	78
EM	25	0011001	19	9	57	0111001	39	Y	89	1011001	59	y	121	1111001	79
SUB	26	0011010	1A	:	58	0111010	3A	Z	90	1011010	5A	z	122	1111010	7A
ESC	27	0011011	1B	;	59	0111011	3B	[	91	1011011	5B	{	123	1111011	7B
FS	28	0011100	1C	<	60	0111100	3C	\	92	1011100	5C	|	124	1111100	7C
GS	29	0011101	1D	=	61	0111101	3D	]	93	1011101	5D	}	125	1111101	7D
RS	30	0011110	1E	>	62	0111110	3E	^	94	1011110	5E	~	126	1111110	7E
US	31	0011111	1F	?	63	0111111	3F	_	95	1011111	5F	Del	127	1111111	7F

Source code file (III)

- The `hello.c` contains **a sequence of bytes and each byte has a bit representation that corresponds to some character.**
- `xxd -b hello.c`

```
00000000: 00100011 01101001 01101110 01100011 01101100 01110101 #inclu
00000006: 01100100 01100101 00100000 00111100 01110011 01110100 de <st
0000000c: 01100100 01101001 01101111 00101110 01101000 00111110 dio.h>
00000012: 00001010 00001010 01101001 01101110 01110100 00100000 ..int
00000018: 01101101 01100001 01101001 01101110 00101000 00101001 main()
0000001e: 00001010 01111011 00001010 00100000 00100000 00100000 .{.
...
```

- `hexdump -C hello.c`

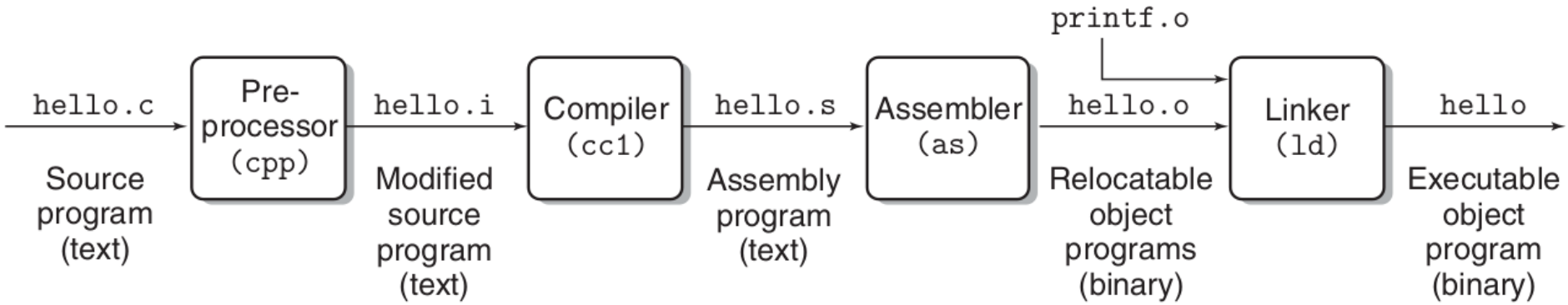
```
00000000 23 69 6e 63 6c 75 64 65 20 3c 73 74 64 69 6f 2e |#include <stdio.|
00000010 68 3e 0a 0a 69 6e 74 20 6d 61 69 6e 28 29 0a 7b |h>..int main().{|
00000020 0a 20 20 20 20 70 72 69 6e 74 66 28 22 68 65 6c |. printf("hel|
00000030 6c 6f 2c 20 77 6f 72 6c 64 5c 6e 22 29 3b 0a 20 |lo, world\n");. |
00000040 20 20 72 65 74 75 72 6e 20 30 3b 0a 7d 0a | return 0;.}.|
0000004f
```

Source code file (IV)

- The `hello.c` program is a high-level C program because it can be read and understood by human beings, but not by processor.
- **Machine language**, or **machine code**, is the most basic set of instructions that a computer can execute.
- **Each type of processor** has its own set of **machine language instructions** (defined by ISA).
- To run `hello` program on the computer, the **C statements must be translated into a sequence of low-level machine-language instructions**.
- These instructions are then packaged in a form called **an executable object program and stored as a binary disk file**.
- This translation process is designated by **compilation**.
  - To perform compilation, it is required a **compiler**
    - A compiler is **a special computer program that translates a programming language's source code into machine code** (according to correspondent ISA).

Compilation

- GNU Compiler Collection (GCC) is a **free and open source set of compilers and development tools** for C, C++ and other programming languages.
  - It is available for Linux, Windows and other operating systems.
- For compiling `hello` program
  `> gcc -o hello hello.c`
  - The gcc compiler **reads the source file `hello.c`** and **translates it into an executable object file `hello`**.
  - The translation is carried out **in a sequence of four stages**



Compilation Phases (I)

1. **Preprocessing phase.**
   - The preprocessor (`cpp`) modifies the original C program according to directives that begin with the # character.
     - For example, the `#include <stdio.h>` command in line 1 of `hello.c` tells the preprocessor to read the contents of the system header file `stdio.h` and insert it directly into the program text.
   - The result is another **text file**, source file C program, typically with the `.i` suffix.
2. **Compilation phase.**
   - The compiler (`cc1`) **translates the text file `hello.i` into the text file `hello.s`**, which contains an **assembly-language program**.
     - Each statement in an assembly-language program exactly describes one low-level machine-language instruction (according to the ISA) in a standard text form.

Compilation Phases (II)

3. **Assembly phase.**
   - The assembler (`as`) **translates `hello.s` into machine- language instructions**, packages them in a form known as a relocatable object program, and stores the result in the object file `hello.o`.
   - The `hello.o` file is **a binary file whose bytes encode machine language instructions (according to the ISA)** rather than characters.
4. **Linking phase.**
   - The linker (`ld`) links/**merges the object code with the library code to produce an executable file**.
     - Notice that our hello program calls the `printf` function, which is part of the standard C library provided by every C compiler.
     - The `printf` function resides in a separate precompiled object file called `printf.o`, which must somehow be merged with our `hello.o` program.
     - The result is the `hello` file, which is an executable object file (or simply executable) that is ready to be loaded into memory and executed by the system.

GCC compiler

- When you invoke `gcc`, it normally does preprocessing, compilation, assembly and linking.
- The `gcc` program accepts options that could change its default behavior.
- Using the **`-save-temps`** options you could save the compilation intermediate files (`*.i`, `*.s`, and, `*.o`)

```
> gcc -o hello hello.c -save-temps
```

Executable object file

- The `hello` program is stored in a file as a sequence of bytes and each byte (or a set of bytes) has a bit representation that corresponds to some machine instruction.
- `xxd -b hello`

```
00000000: 01111111 01000101 01001100 01000110 00000010 00000001 .ELF..
00000006: 00000001 00000000 00000000 00000000 00000000 00000000 ......
0000000c: 00000000 00000000 00000000 00000000 00000011 00000000 ......
00000012: 00111110 00000000 00000001 00000000 00000000 00000000 >.....
00000018: 01100000 00010000 00000000 00000000 00000000 00000000 `.....
0000001e: 00000000 00000000 01000000 00000000 00000000 00000000 ..@...
00000024: 00000000 00000000 00000000 00000000 01111000 00111001 ....x9
0000002a: 00000000 00000000 00000000 00000000 00000000 00000000 ......
...
```

- `hexdump -C hello`

```
00000000 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00 |.ELF............|
00000010 03 00 3e 00 01 00 00 00 60 10 00 00 00 00 00 00 |..>.....`.......|
00000020 40 00 00 00 00 00 00 00 78 39 00 00 00 00 00 00 |@.......x9......|
00000030 00 00 00 00 40 00 38 00 0d 00 40 00 1f 00 1e 00 |....@.8...@.....|
00000040 06 00 00 00 04 00 00 00 40 00 00 00 00 00 00 00 |........@.......|
00000050 40 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 |@.......@.......|
...
```

Files

Types

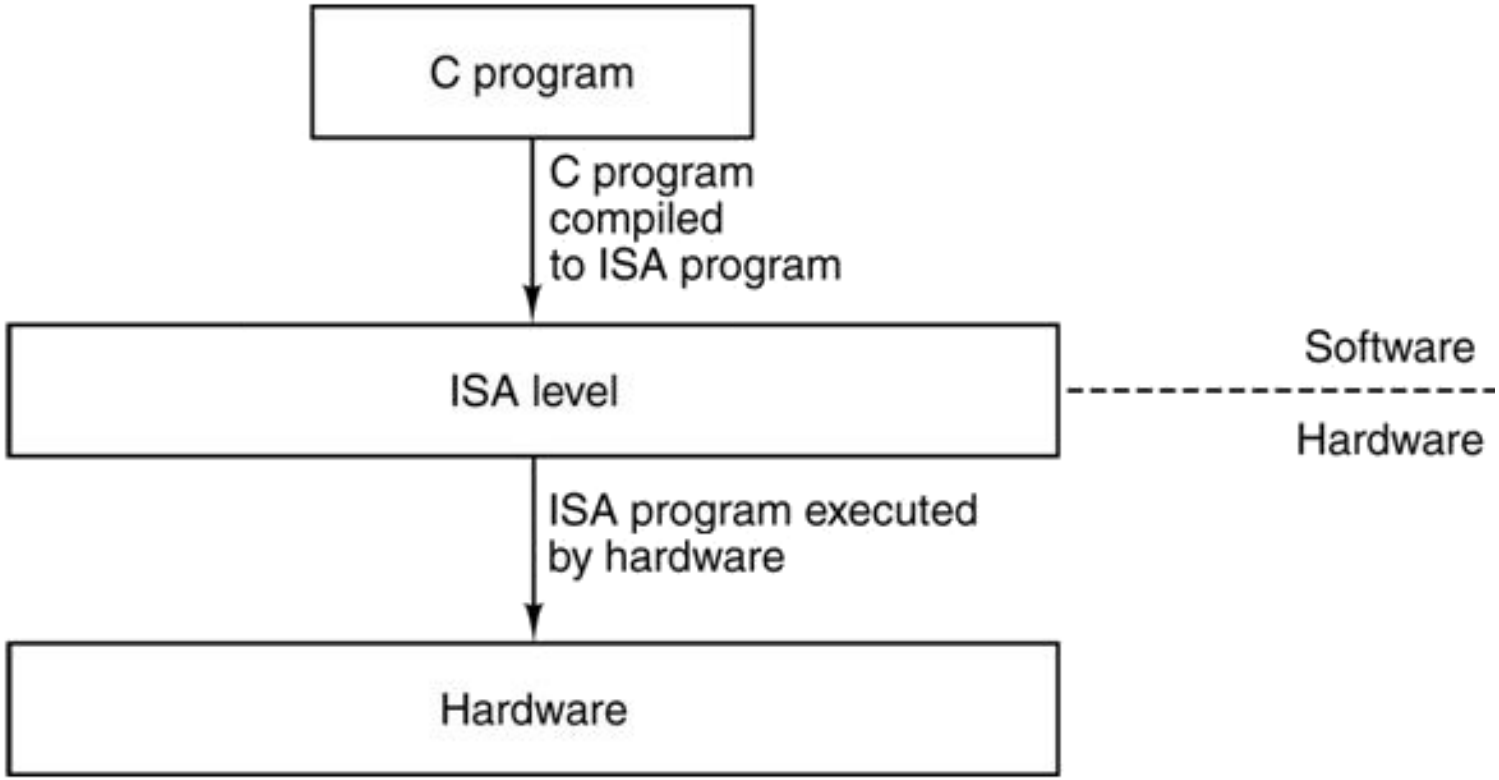
- **Text files**
  - Contain readable characters and can be opened with any text editor
- **Binary files**
  - Contain non-readable characters and require specific programs for access.

Feature	Text File	Binary File
Readability	Human-readable	Requires specific software
Data Type	ASCII or Unicode characters	Any data type (images, audio, etc.)
File Size	Generally larger due to character encoding	More compact, efficient storage
Use Cases	Configuration files, scripts, notes	Executable files, media files

Executing the `hello` Program

An executable object file

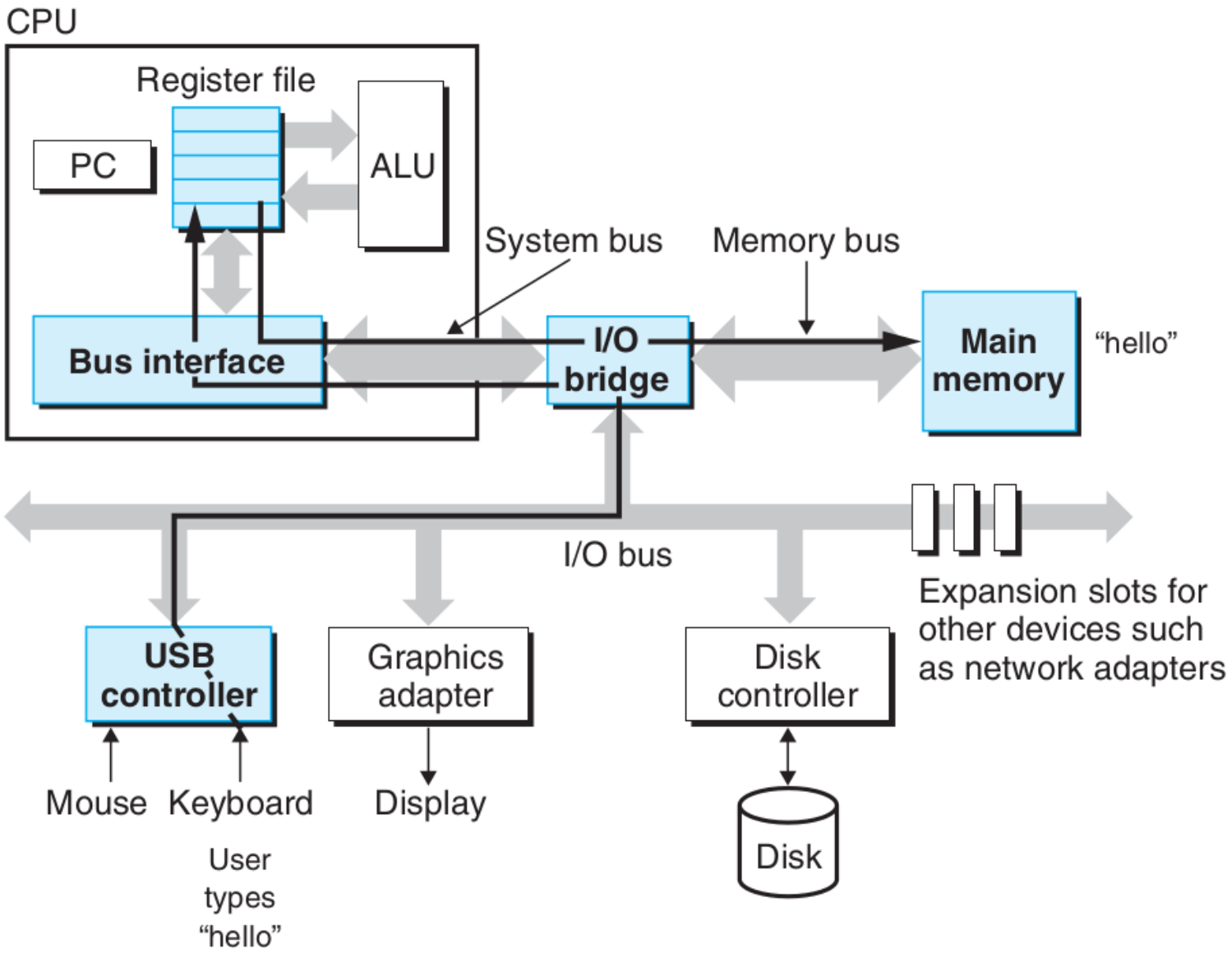
- The content of an executable object is a **set of CPU instructions** (according to ISA).



Executing (I)

- In order to run the executable file on a Unix derived system, we type its name to an application program known as a **shell**:
  `> ./hello`
  - **A shell is a program that takes commands from the keyboard and gives them to the operating system to perform**.
- As we type the characters `./hello` at the keyboard, the shell program reads each one into a register, and then stores it in memory.
- When we hit the `enter` key on the keyboard, the shell knows that we have finished typing the command.

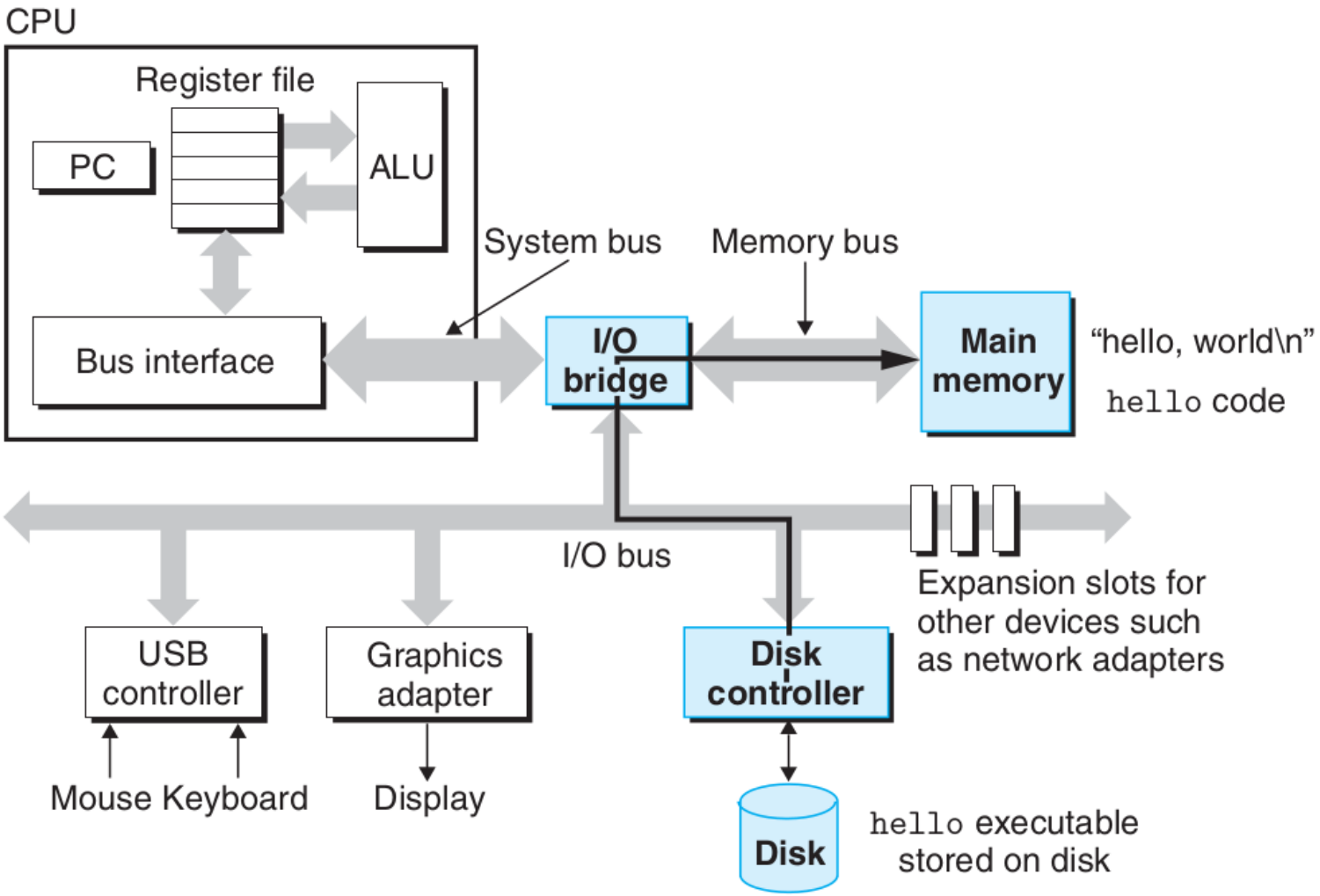
Executing (II)



Executing (III)

- A set of operations are executed by operating systems that copies the contents of the `hello` object file from disk to main memory.
  - Using a technique known as direct memory access (DMA), the data travels directly from disk to main memory, without passing through the processor.

Executing (IV)



Executing (V)

- Once the contents of the `hello` **object file are loaded into memory**, the **processor begins executing the machine-language instructions** in the `hello` program's `main` routine.
  - These instructions copy the bytes in the `hello, world\n` string from memory to the register file, and from there to the display device, where they are displayed on the screen.

Executing (VI)