CMSC 313
COMPUTER ORGANIZATION
&
ASSEMBLY LANGUAGE
PROGRAMMING

Lecture 1, Fall 2014

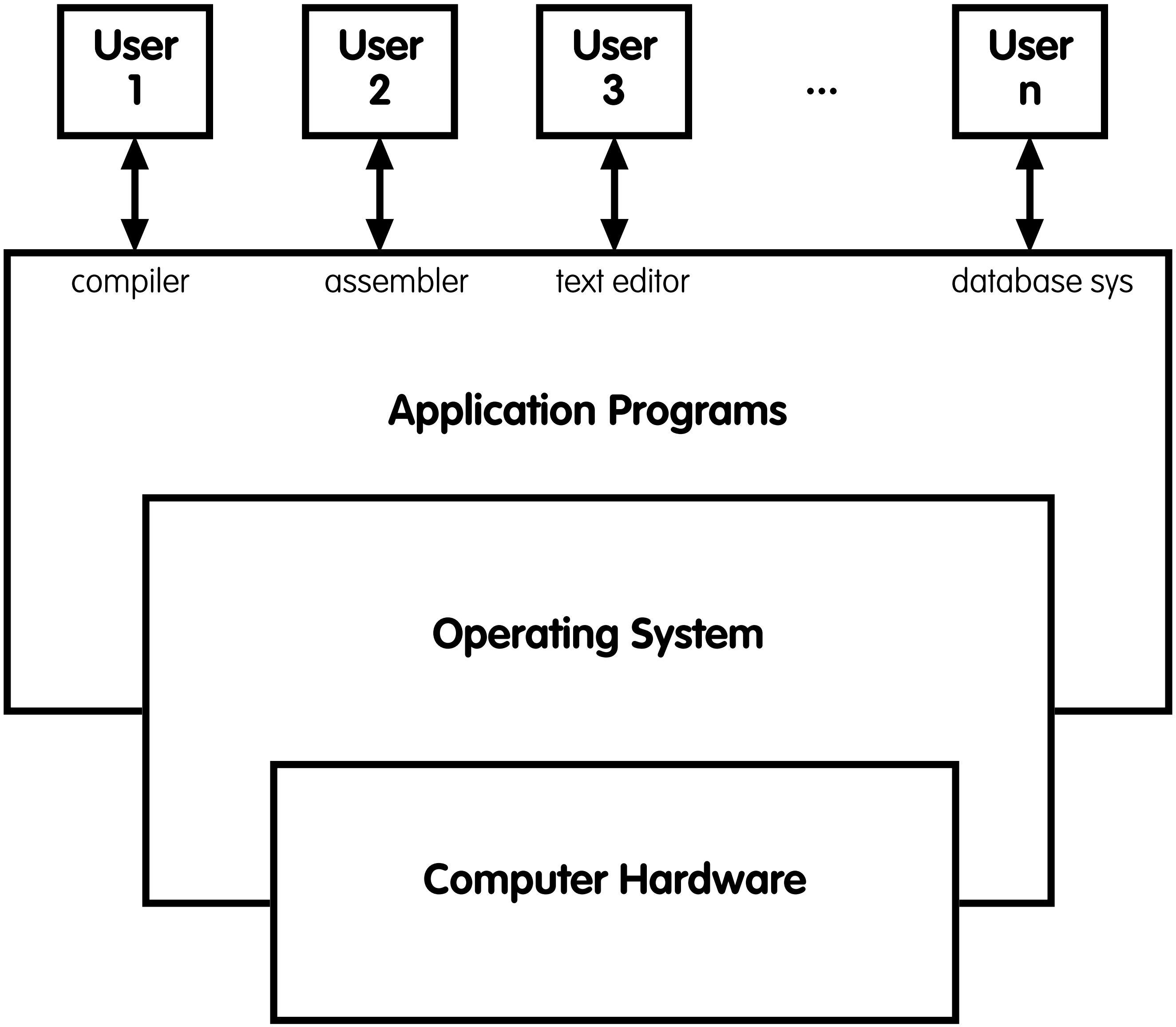
TOPICS TODAY

- Course overview
- Levels of machines
- Machine models: von Neumann & System Bus
- Fetch-Execute Cycle
- Base Conversion

COURSE OVERVIEW

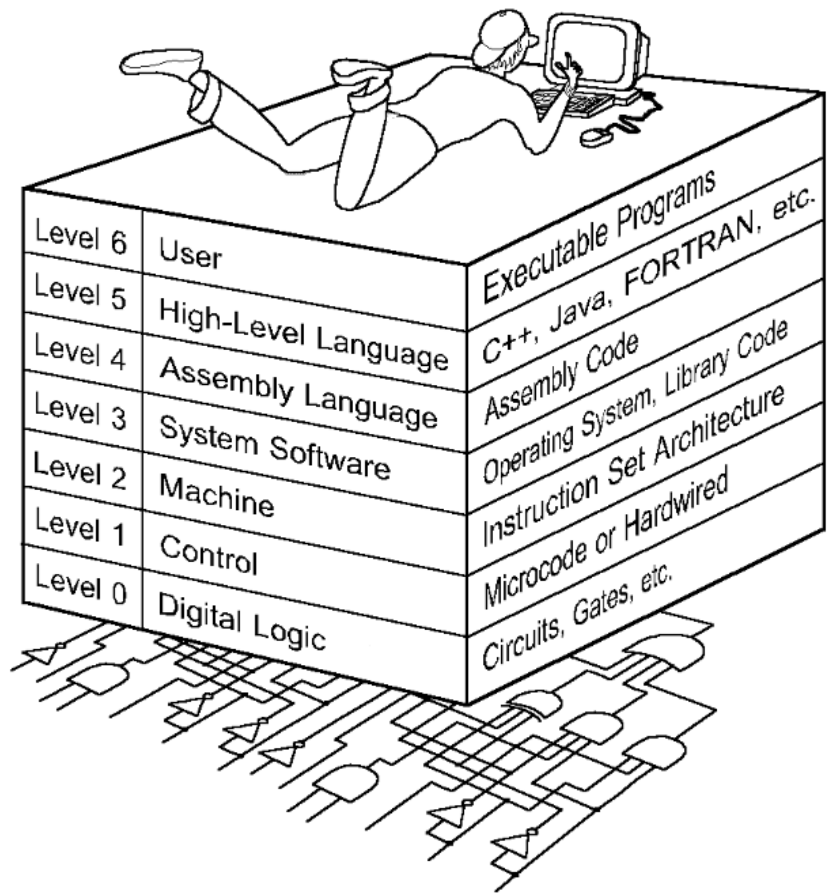
LEVELS OF MACHINES

Computer Science View of the World



1.6 The Computer Level Hierarchy

- **Each virtual machine layer is an abstraction of the level below it.**
- **The machines at each level execute their own particular instructions, calling upon machines at lower levels to perform tasks as required.**
- **Computer circuits ultimately carry out the work.**



1.6 The Computer Level Hierarchy

- Level 6: The User Level
  - Program execution and user interface level.
  - The level with which we are most familiar.
- Level 5: High-Level Language Level
  - The level with which we interact when we write programs in languages such as C, Pascal, Lisp, and Java.

1.6 The Computer Level Hierarchy

- Level 4: Assembly Language Level
  - Acts upon assembly language produced from Level 5, as well as instructions programmed directly at this level.
- Level 3: System Software Level
  - Controls executing processes on the system.
  - Protects system resources.
  - Assembly language instructions often pass through Level 3 without modification.

1.6 The Computer Level Hierarchy

- Level 2: Machine Level
  - Also known as the Instruction Set Architecture (ISA) Level.
  - Consists of instructions that are particular to the architecture of the machine.
  - Programs written in machine language need no compilers, interpreters, or assemblers.

1.6 The Computer Level Hierarchy

- Level 1: Control Level
  - A *control unit* decodes and executes instructions and moves data through the system.
  - Control units can be *microprogrammed* or *hardwired*.
  - A microprogram is a program written in a low-level language that is implemented by the hardware.
  - Hardwired control units consist of hardware that directly executes machine instructions.

1.6 The Computer Level Hierarchy

- Level 0: Digital Logic Level
  - This level is where we find digital circuits (the chips).
  - Digital circuits consist of gates and wires.
  - These components implement the mathematical logic of all other levels.

MACHINE MODELS

1.7 The von Neumann Model

- On the ENIAC, all programming was done at the digital logic level.
- Programming the computer involved moving plugs and wires.
- A different hardware configuration was needed to solve every unique problem type.

Configuring the ENIAC to solve a "simple" problem required many days labor by skilled technicians.

1.7 The von Neumann Model

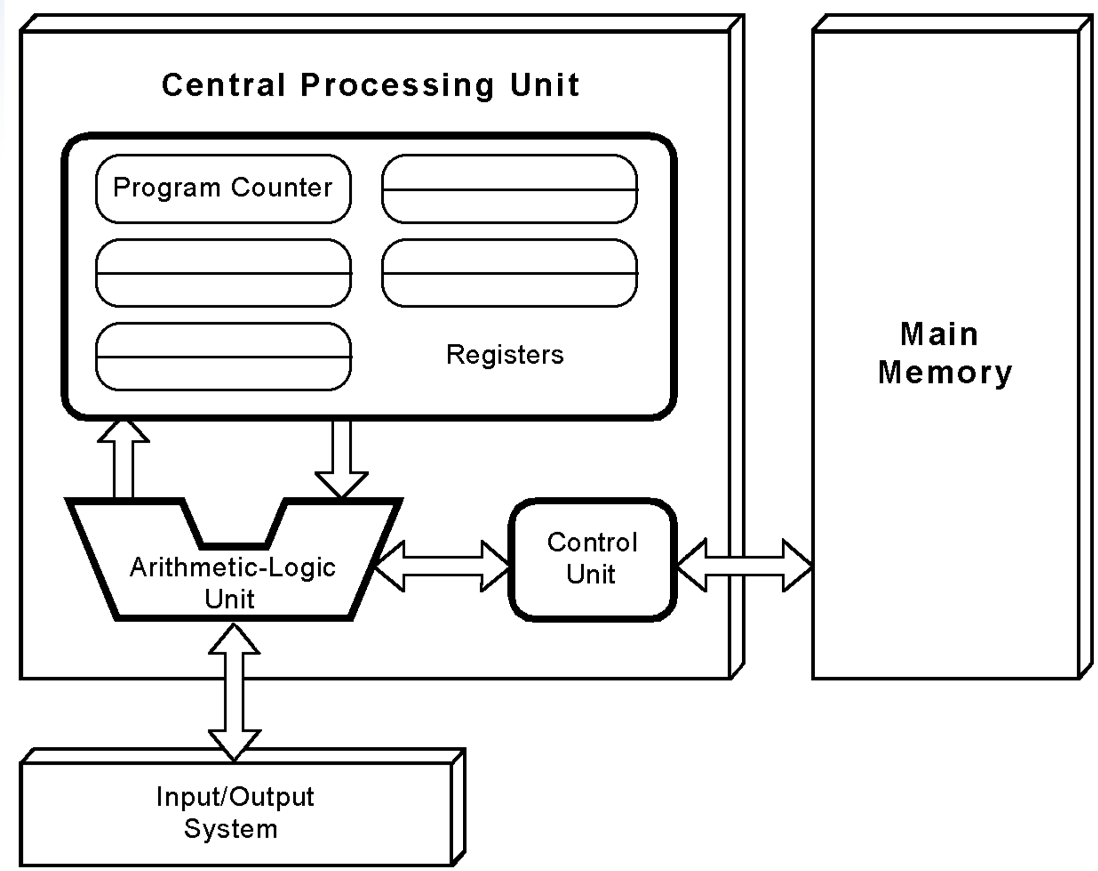
- Inventors of the ENIAC, John Mauchley and J. Presper Eckert, conceived of a computer that could store instructions in memory.
- The invention of this idea has since been ascribed to a mathematician, John von Neumann, who was a contemporary of Mauchley and Eckert.
- Stored-program computers have become known as von Neumann Architecture systems.

1.7 The von Neumann Model

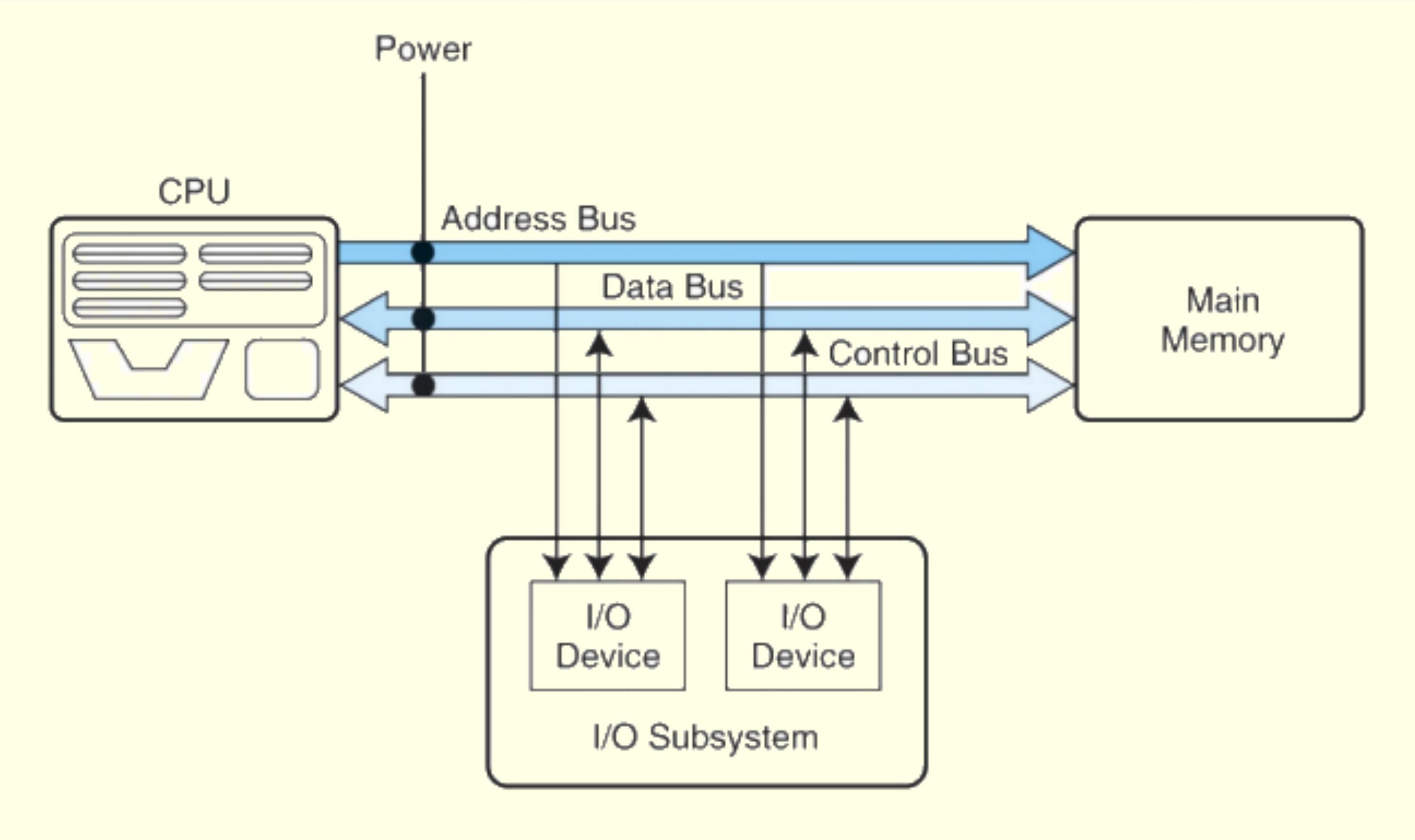
- Today's stored-program computers have the following characteristics:
  - Three hardware systems:
    - A central processing unit (CPU)
    - A main memory system
    - An I/O system
  - The capacity to carry out sequential instruction processing.
  - A single data path between the CPU and main memory.
    - This single path is known as the *von Neumann bottleneck*.

1.7 The von Neumann Model

- **This is a general depiction of a von Neumann system:**
- **These computers employ a fetch-decode-execute cycle to run programs as follows . . .**

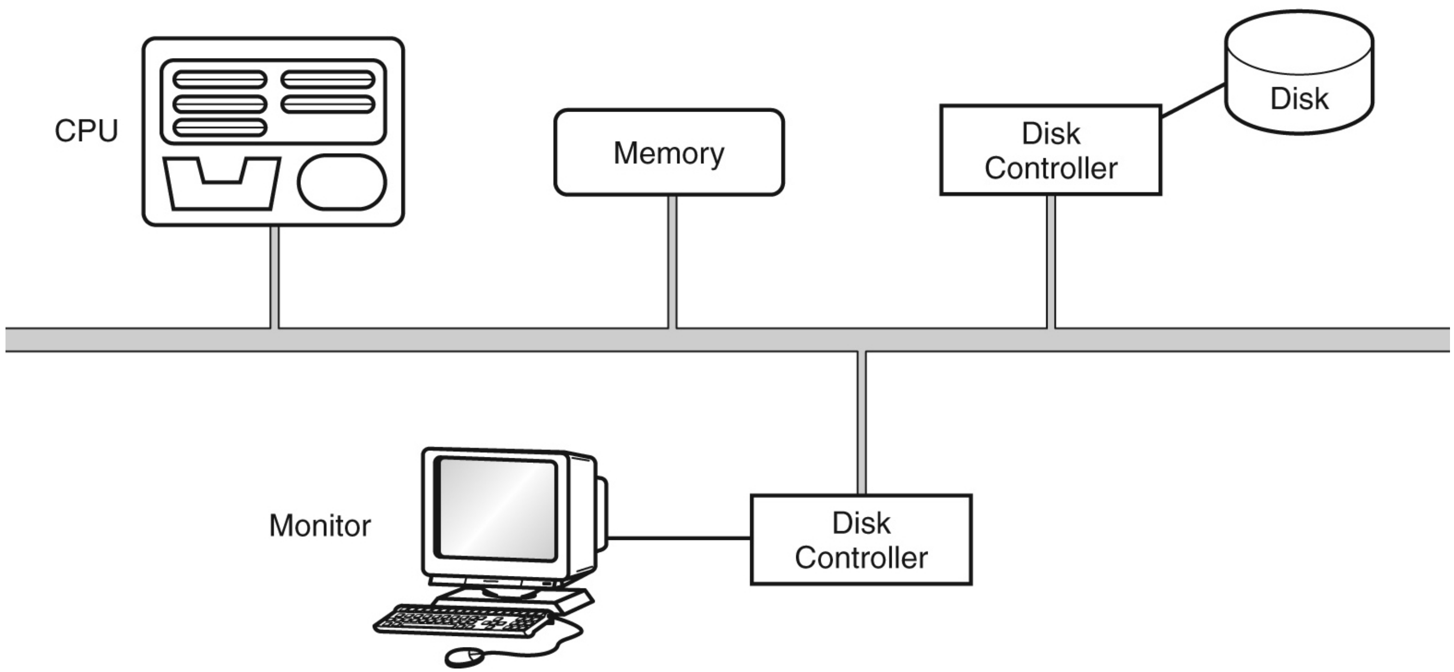


4.3 The Bus


Power
CPU
Address Bus
Data Bus
Control Bus
Main Memory
I/O Device
I/O Device
I/O Subsystem

4.3 The Bus

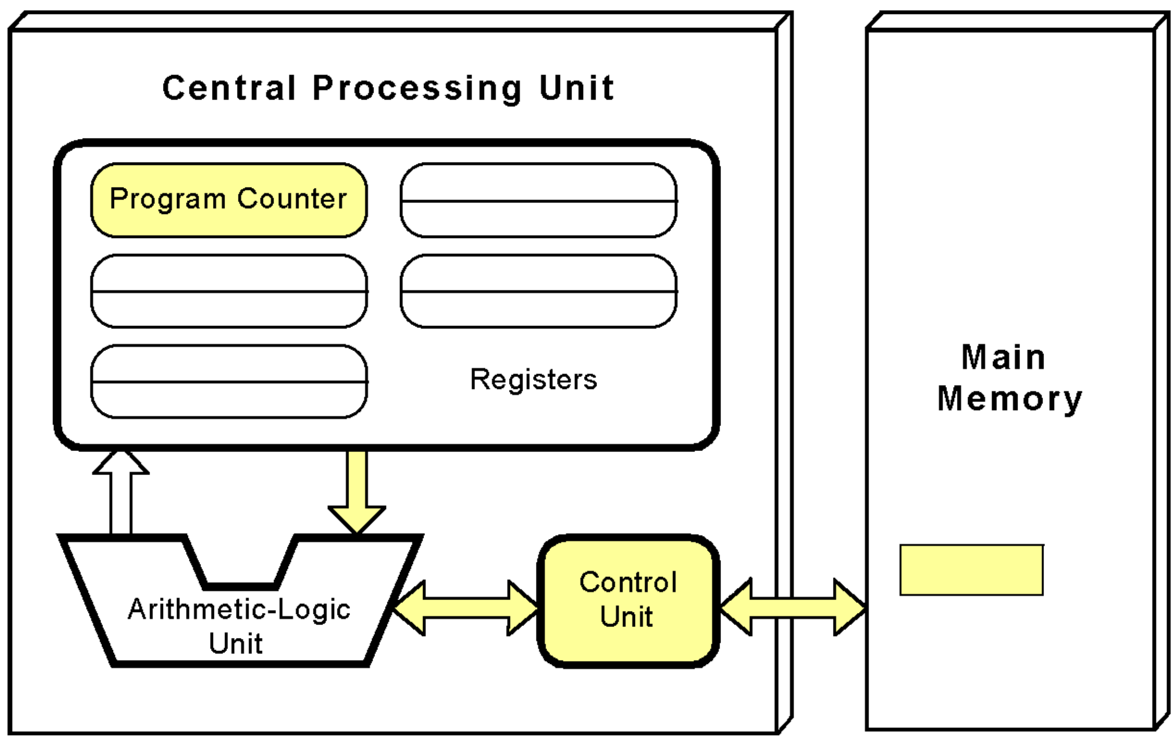
- A multipoint bus is shown below.
- Because a multipoint bus is a shared resource, access to it is controlled through protocols, which are built into the hardware.



FETCH EXECUTE CYCLE

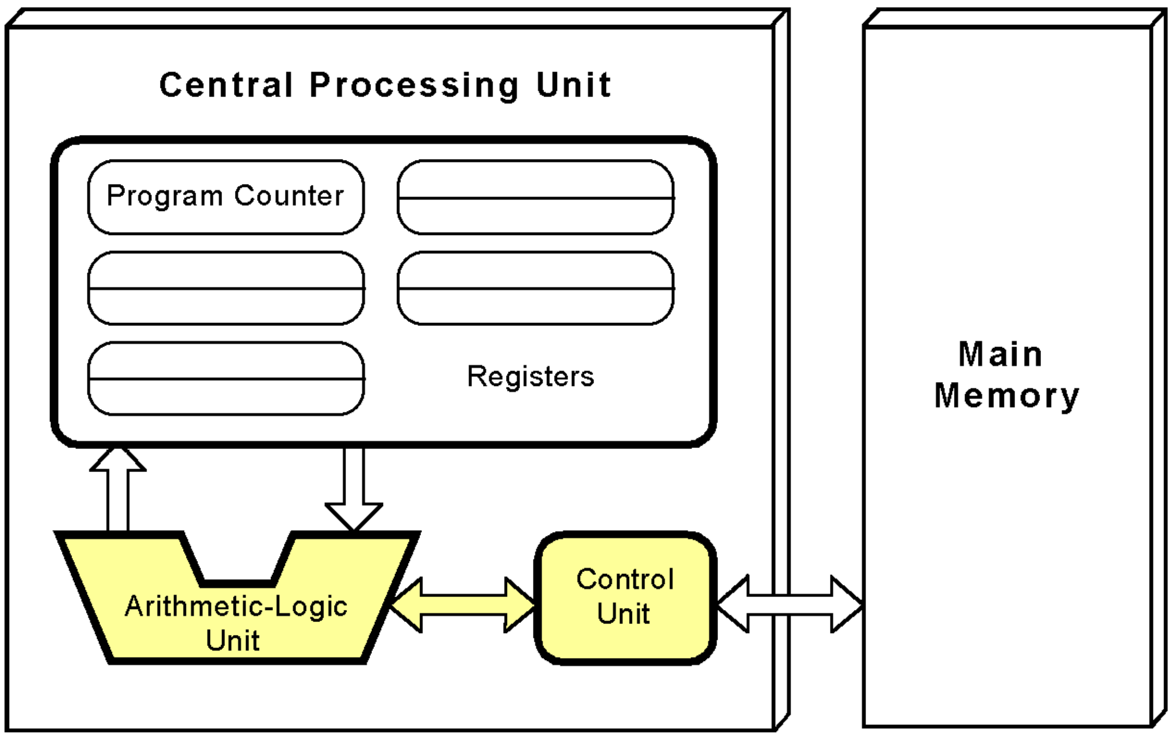
1.7 The von Neumann Model

- The control unit fetches the next instruction from memory using the program counter to determine where the instruction is located.



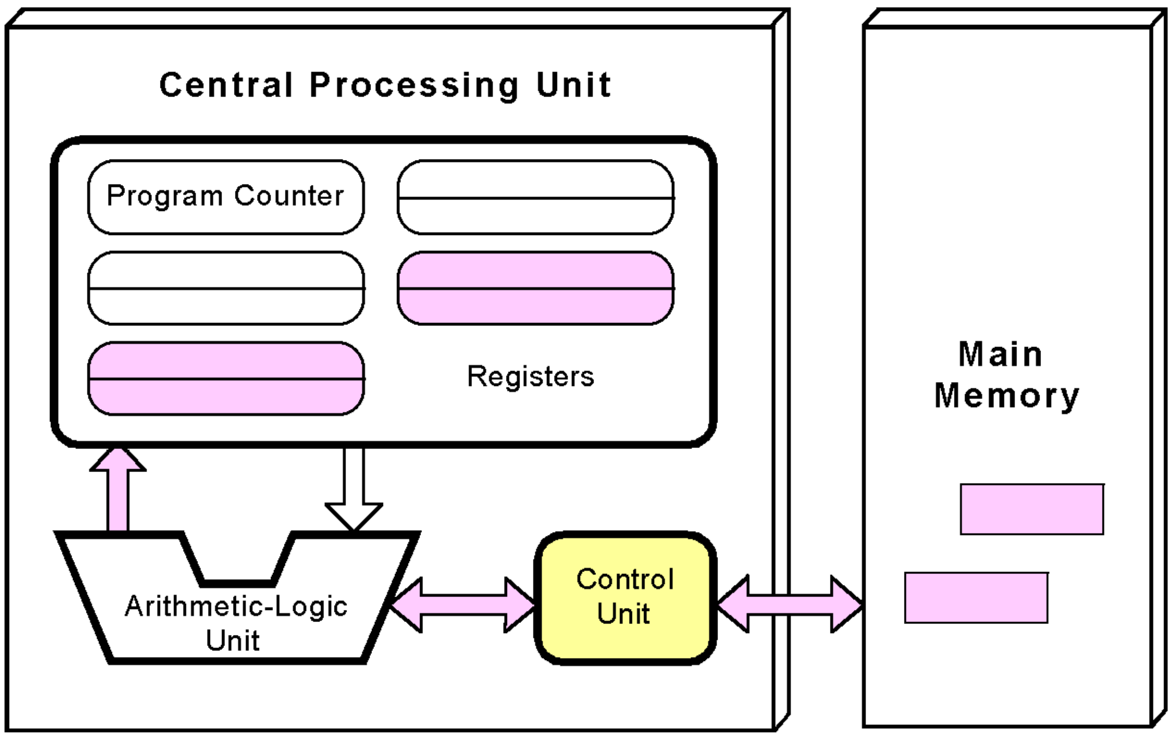
1.7 The von Neumann Model

- **The instruction is decoded into a language that the ALU can understand.**



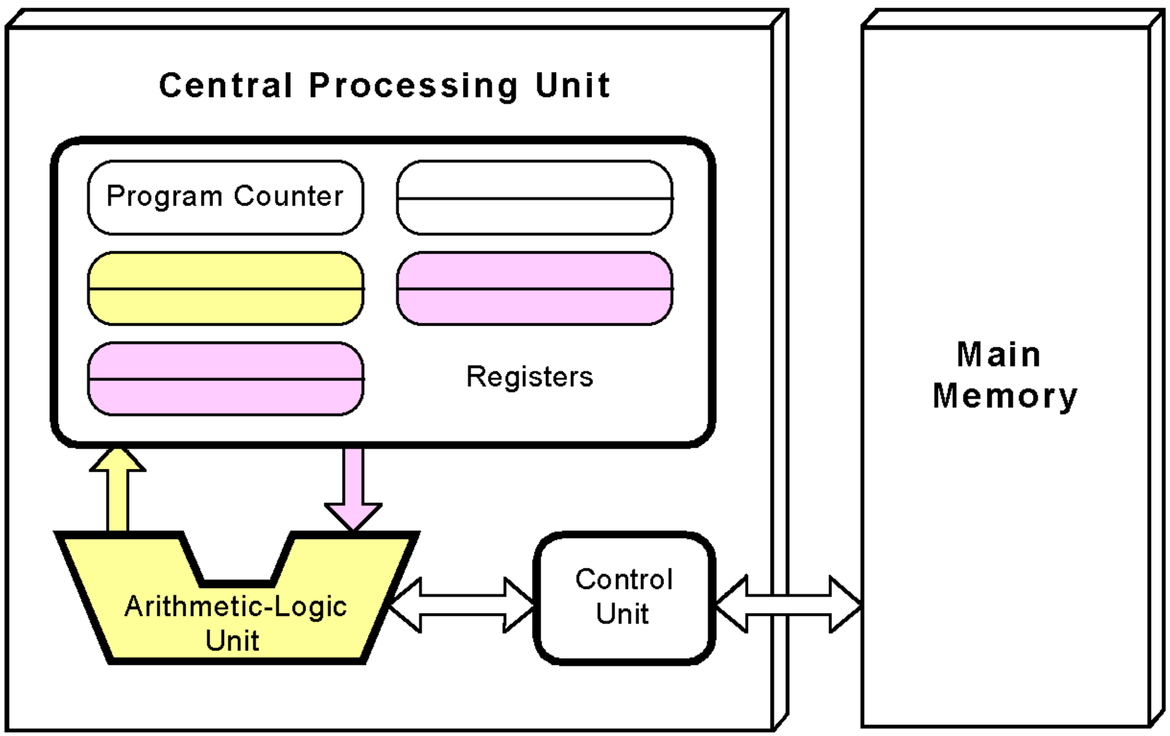
1.7 The von Neumann Model

- **Any data operands required to execute the instruction are fetched from memory and placed into registers within the CPU.**



1.7 The von Neumann Model

- **The ALU executes the instruction and places results in registers or memory.**



BASE CONVERSION

2.1 Introduction

- A *bit* is the most basic unit of information in a computer.
  - It is a state of “on” or “off” in a digital circuit.
  - Sometimes these states are “high” or “low” voltage instead of “on” or “off..”
- A *byte* is a group of eight bits.
  - A byte is the smallest possible *addressable* unit of computer storage.
  - The term, “addressable,” means that a particular byte can be retrieved according to its location in memory.

2.1 Introduction

- A *word* is a contiguous group of bytes.
  - Words can be any number of bits or bytes.
  - Word sizes of 16, 32, or 64 bits are most common.
  - In a word-addressable system, a word is the smallest addressable unit of storage.
- A group of four bits is called a *nibble*.
  - Bytes, therefore, consist of two nibbles: a “high-order nibble,” and a “low-order” nibble.

2.2 Positional Numbering Systems

- Bytes store numbers using the position of each bit to represent a power of 2.
  - The binary system is also called the base-2 system.
  - Our decimal system is the base-10 system. It uses powers of 10 for each position in a number.
  - Any integer quantity can be represented exactly using any base (or *radix*).

2.2 Positional Numbering Systems

- The decimal number 947 in powers of 10 is:

$$9 \times 10^{2} + 4 \times 10^{1} + 7 \times 10^{0}$$

- The decimal number 5836.47 in powers of 10 is:

$$5 \times 10^{3} + 8 \times 10^{2} + 3 \times 10^{1} + 6 \times 10^{0}$$
$$+ 4 \times 10^{-1} + 7 \times 10^{-2}$$

2.2 Positional Numbering Systems

- The binary number 11001 in powers of 2 is:

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 16 + 8 + 0 + 0 + 1 = 25$$

- When the radix of a number is something other than 10, the base is denoted by a subscript.
  - Sometimes, the subscript 10 is added for emphasis:

$$11001_2 = 25_{10}$$

2.3 Converting Between Bases

- **Converting 190 to base 3...**
  - First we take the number that we wish to convert and divide it by the radix in which we want to express our result.
  - In this case, 3 divides 190 63 times, with a remainder of 1.
  - Record the quotient and the remainder.

```
3 | 190   1
  ------
     63
```

2.3 Converting Between Bases

- **Converting 190 to base 3...**
  - 63 is evenly divisible by 3.
  - Our remainder is zero, and the quotient is 21.

```
3 | 190   1
  ----------
   3 | 63   0
     -------
       21
```

2.3 Converting Between Bases

- **Converting 190 to base 3...**
  - Continue in this way until the quotient is zero.
  - In the final calculation, we note that 3 divides 2 zero times with a remainder of 2.
  - Our result, reading from bottom to top is:

$$190_{10} = 21001_3$$

3	190	1
3	63	0
3	21	0
3	7	1
3	2	2
	0	

2.3 Converting Between Bases

- **Using the multiplication method to convert the decimal 0.8125 to binary, we multiply by the radix 2.**
  - The first product carries into the units place.

$$\begin{array}{r} .8125 \\ \times \quad 2 \\ \hline 1.6250 \end{array}$$

2.3 Converting Between Bases

- **Converting 0.8125 to binary . . .**
  - Ignoring the value in the units place at each step, continue multiplying each fractional part by the radix.

$$\begin{array}{r} .8125 \\ \times\quad 2 \\ \hline 1.6250 \end{array}$$

$$\begin{array}{r} .6250 \\ \times\quad 2 \\ \hline 1.2500 \end{array}$$

$$\begin{array}{r} .2500 \\ \times\quad 2 \\ \hline 0.5000 \end{array}$$

2.3 Converting Between Bases

- **Converting 0.8125 to binary . . .**
  - You are finished when the product is zero, or until you have reached the desired number of binary places.
  - Our result, reading from top to bottom is:

    $0.8125_{10} = 0.1101_2$
  - This method also works with any base. Just use the target radix as the multiplier.

```
  .8125
 ×    2
 ------
 1.6250

  .6250
 ×    2
 ------
 1.2500

  .2500
 ×    2
 ------
 0.5000

  .5000
 ×    2
 ------
 1.0000
```

Converting Base 6 to Base 10

- $123.45_6 = ???.???_{10}$

$123_6 = 1 \times 36_{10} + 2 \times 6_{10} + 3 \times 1_{10} = 51_{10}$

$0.45_6 = 4 \times 1/6_{10} + 5 \times 1/36_{10} = 0.805555\ldots_{10}$

$123.45_6 = 51.805555\ldots_{10}$

Converting Base 10 to Base 6

- $754.94_{10} = 3254.5\ 35012\ 35012\ 35012..._6$

$754_{10} = 11_6 \times 244_6 + 5_6 \times 14_6 + 4_6 \times 1_6 = ???_6$

$754 \div 6 = 125$ remainder 4

$125 \div 6 = 20$ remainder 5

$20 \div 6 = 3$ remainder 2

$3 \div 6 = 0$ remainder 3

$3254_6 = 3 \times 216_{10} + 2 \times 36_{10} + 5 \times 6_{10} + 4 \times 1 = 754_{10}$

Converting Base 10 to Base 6 (cont)

- $0.94_{10} = ???.???_6$

$0.94 \times 6 = \quad 5.64 \quad \text{-->} \quad 5$

$0.64 \times 6 = \quad 3.84 \quad \text{-->} \quad 3$

$0.84 \times 6 = \quad 5.04 \quad \text{-->} \quad 5$

$0.04 \times 6 = \quad 0.24 \quad \text{-->} \quad 0$

$0.24 \times 6 = \quad 1.44 \quad \text{-->} \quad 1$

$0.44 \times 6 = \quad 2.64 \quad \text{-->} \quad 2$

$0.64 \times 6 = \quad 3.84 \quad \text{-->} \quad 3$

$0.94_{10} = 0.5\ 35012\ 35012\ 35012\ldots_6$

$5/6 + 3/36 + 5/216 + 0 + 1/6^5 + 2/6^6 = 0.939986282\ldots_{10}$

2.3 Converting Between Bases

- The binary numbering system is the most important radix system for digital computers.
- However, it is difficult to read long strings of binary numbers -- and even a modestly-sized decimal number becomes a very long binary number.
  - For example: $11010100011011_2 = 13595_{10}$
- For compactness and ease of reading, binary values are usually expressed using the hexadecimal, or base-16, numbering system.

BASES

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

2.3 Converting Between Bases

- The hexadecimal numbering system uses the numerals 0 through 9 and the letters A through F.
  - The decimal number 12 is C_{16}.
  - The decimal number 26 is $1A_{16}$.
- It is easy to convert between base 16 and base 2, because $16 = 2^4$.
- Thus, to convert from binary to hexadecimal, all we need to do is group the binary digits into groups of four.

A group of four binary digits is called a hextet

2.3 Converting Between Bases

- Using groups of hextets, the binary number 11010100011011_2 (= 13595_{10}) in hexadecimal is:

0011	0101	0001	1011
3	5	1	B

If the number of bits is not a multiple of 4, pad on the left with zeros.

- Octal (base 8) values are derived from binary by using groups of three bits ($8 = 2^3$):

011	010	100	011	011
3	2	4	3	3

Octal was very useful when computers used six-bit words.

NEXT TIME

- Representing numbers
- Representing negative numbers
- Floating point numbers (briefly)
- Characters and strings