Introduction to Computer Science
Lesson 4

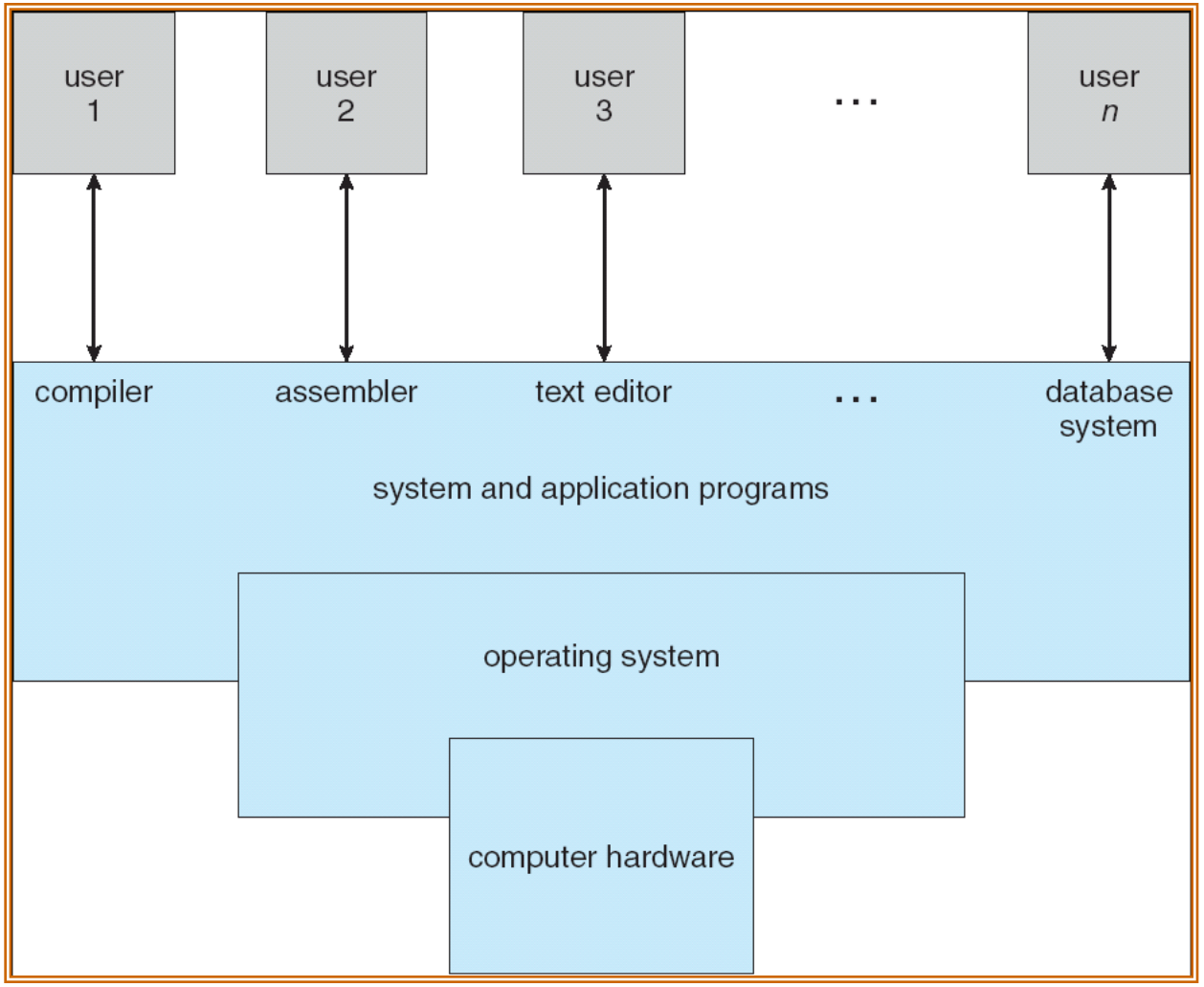
BSc in Computer Science
University of New York, Tirana

Assoc. Prof. Marenglen Biba

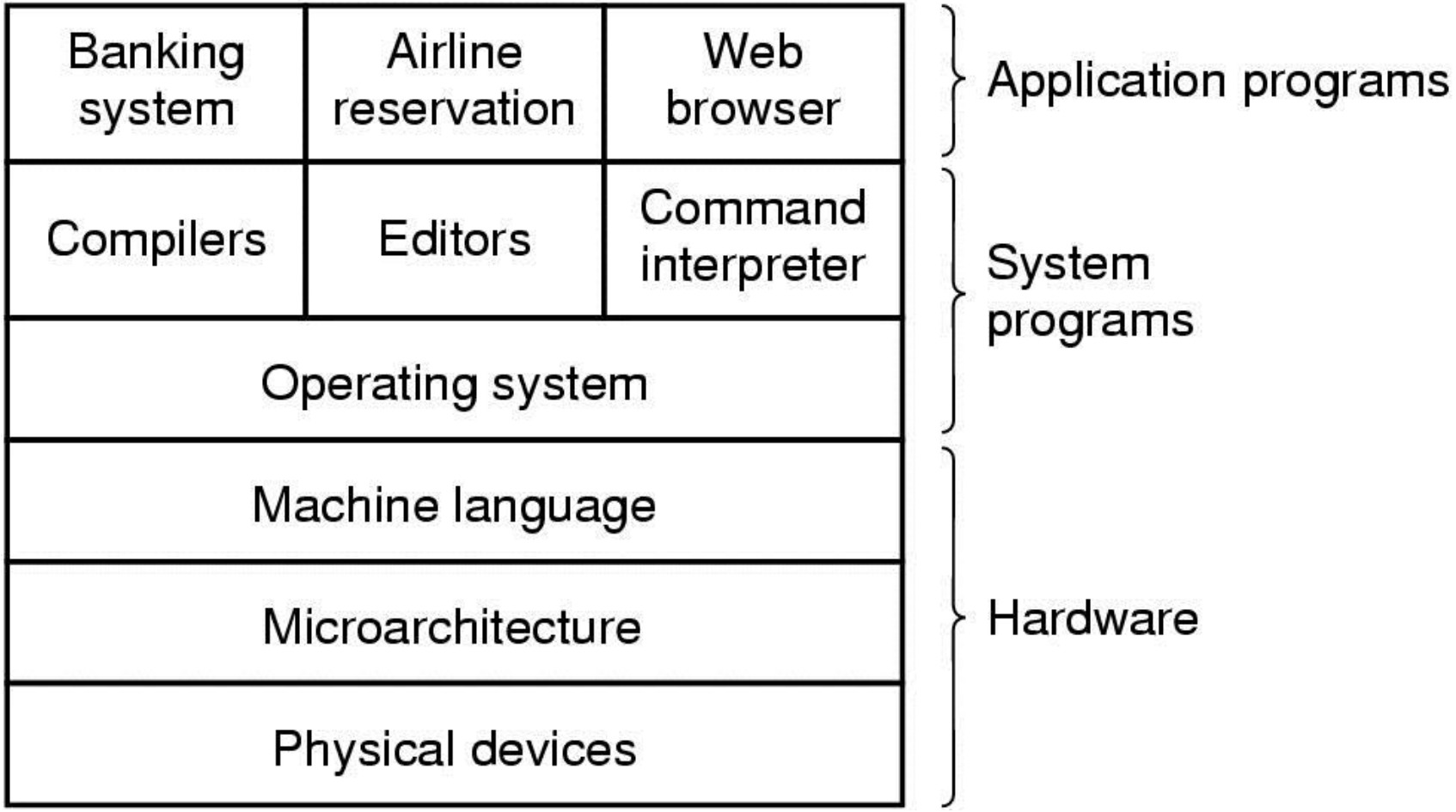
Data Manipulation

- 2.1 Computer Architecture
- 2.2 Machine Language
- 2.3 Program Execution
- 2.4 Arithmetic/Logic Instructions
- 2.5 Communicating with Other Devices
- 2.6 Other Architectures

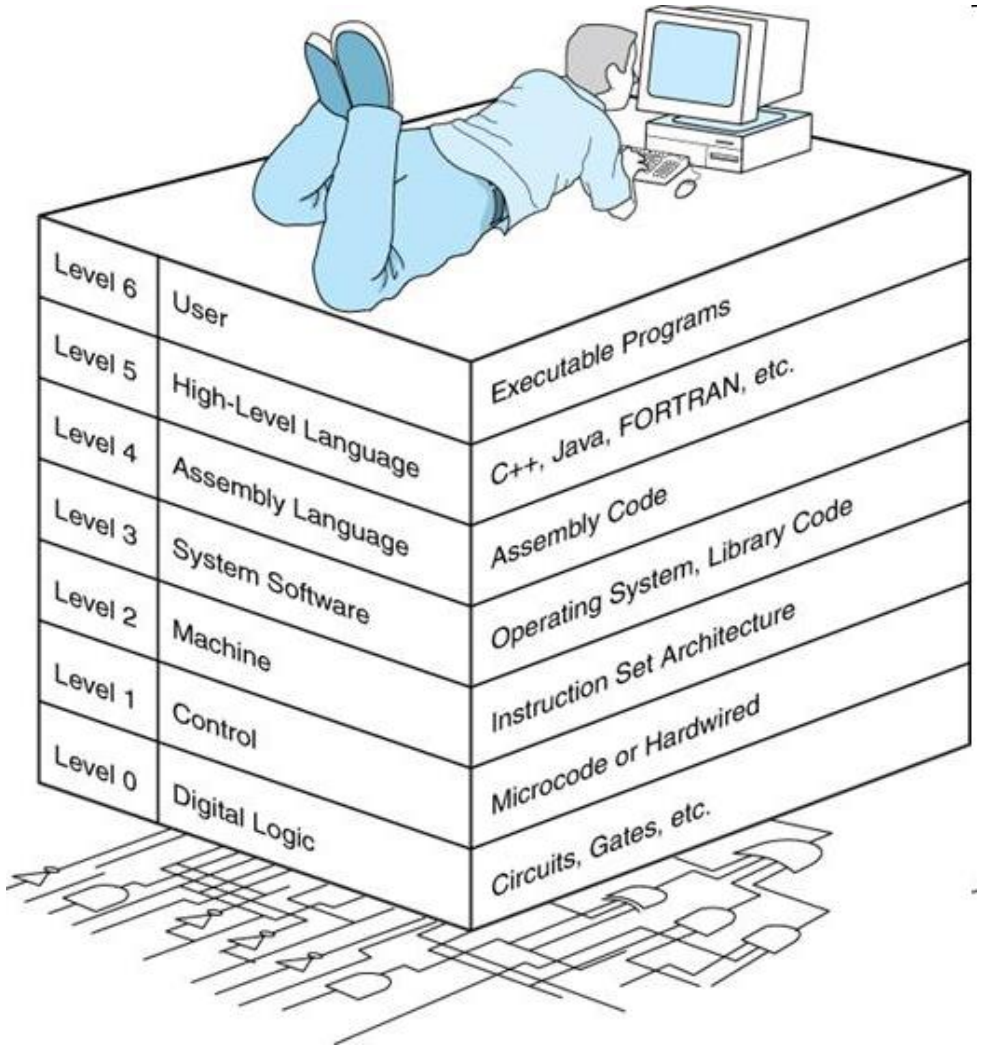
Computer Hierarchy



Computer Hierarchy



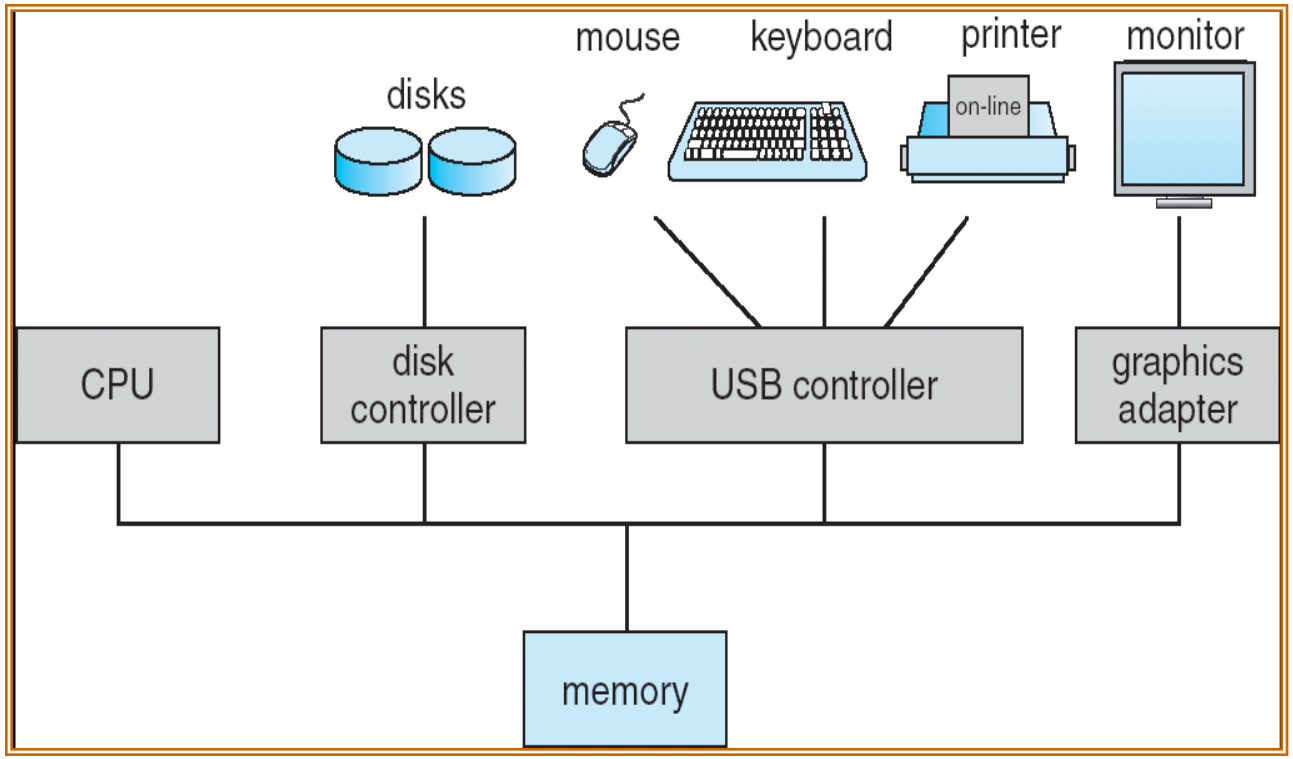
Where are you?



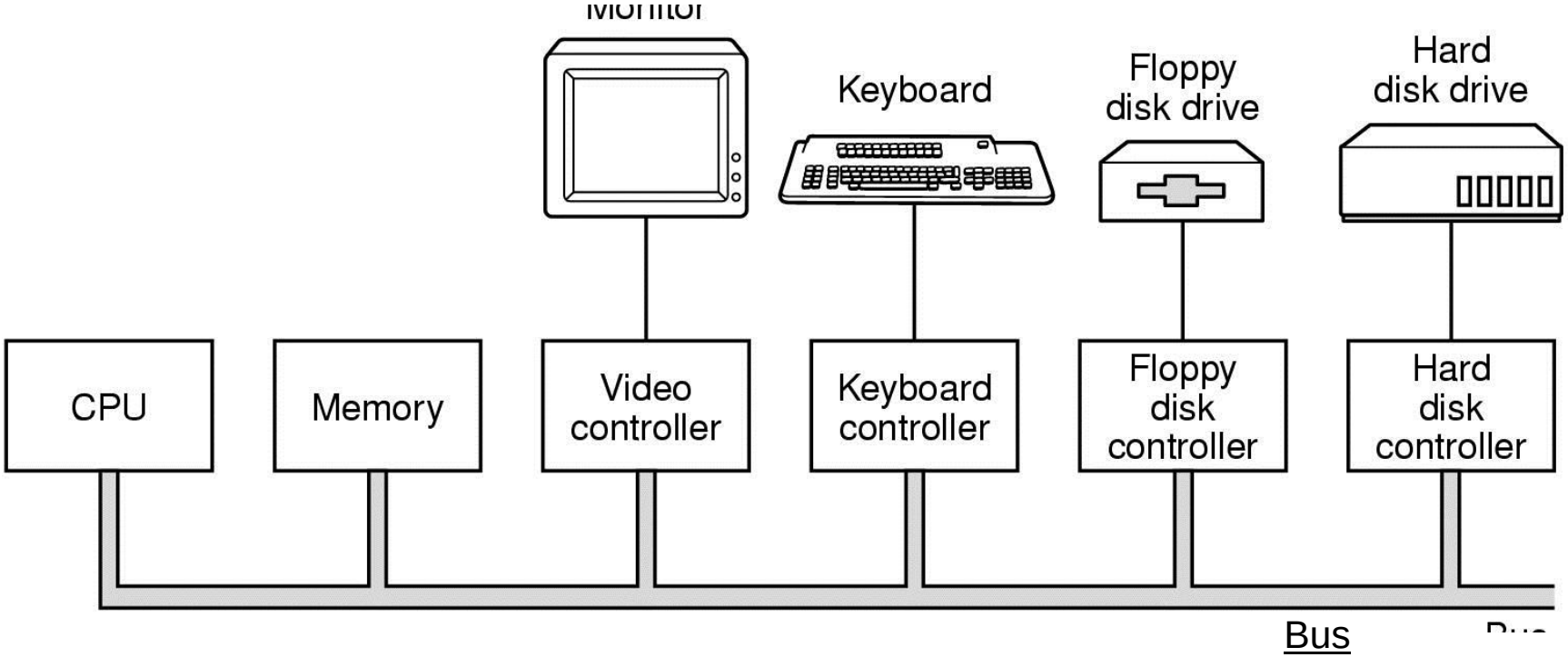
Computer Architecture

- Central Processing Unit (CPU) or processor
  - Arithmetic/Logic unit versus Control unit
  - Registers
    - General purpose
    - Special purpose
- Bus
- Motherboard

Computer Architecture



Controllers and Bus



CPU

- The circuitry in a computer that controls the manipulation of data is called the central processing unit, or **CPU** (often referred to as merely the processor).

- In the machines of the mid-twentieth century, CPUs were large units composed of perhaps several racks of electronic circuitry that reflected the significance of the unit.
- However, technology has shrunk these devices drastically.

- The CPUs found in today's PCs (such processors made by Intel and processors made by AMD) are packaged as small flat squares (approximately two inches by two inches) whose connecting pins plug into a socket mounted on the machine's main circuit board (called the **motherboard**).

Example of processor: Pentium

CPU

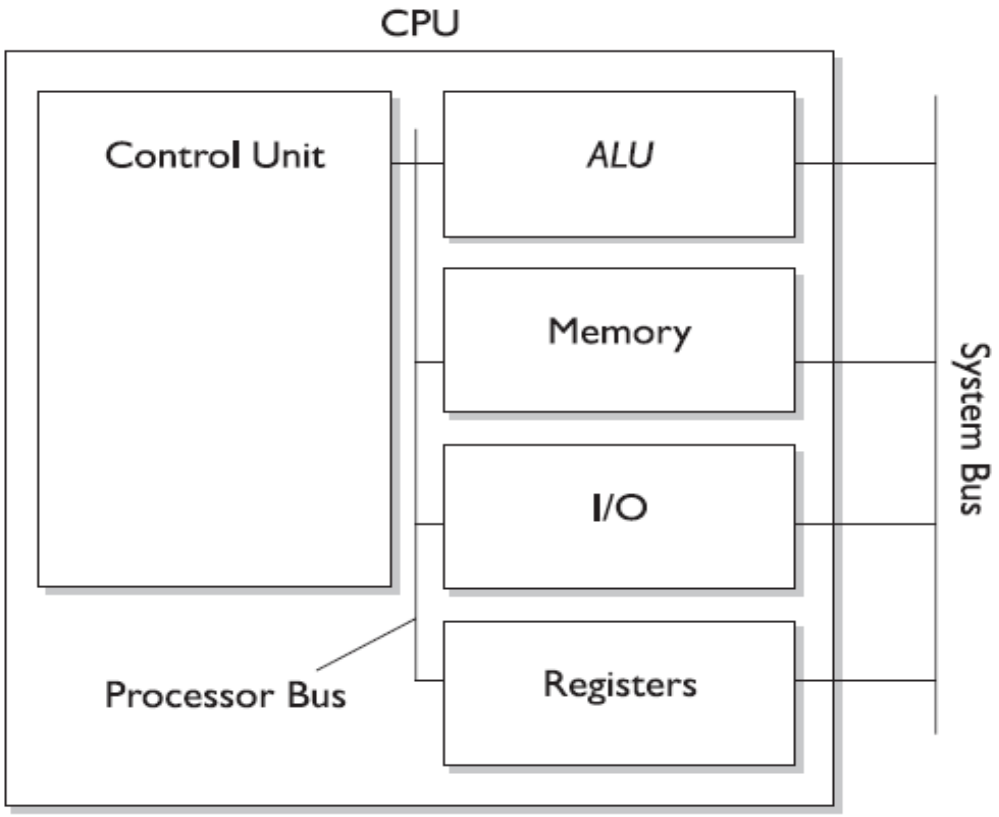
- In a PC, the **central processing unit (CPU)** is the primary control device for the entire computer system.
- The CPU is technically a **set of components** that manages all the activities and does much of the “heavy lifting” in a computer system.
- The CPU interfaces, or is connected, to all of the components such as memory, storage, and input/output (I/O) through communications channels called ***busses***.
- The CPU performs a number of individual or discrete functions that must work in harmony in order for the system to function.
- Additionally, the CPU is responsible for managing the activities of the entire system.
- The CPU takes direction from internal commands that are stored in the CPU as well as external commands that come from the operating system and other programs.
- It is important to note that these functions occur in all CPUs regardless of manufacturer.

CPU: ALU and the Control Unit

- A CPU consists of two parts:
  - the **ALU (arithmetic/logic unit),** which contains the circuitry that performs operations on data (such as addition and subtraction),
  - the **Control Unit**, which contains the circuitry for coordinating the machine's activities.

- For temporary storage of information, the CPU contains cells, or registers, that are conceptually similar to main memory cells.
  - These registers can be classified as either general-purpose registers or special-purpose registers.

CPU - Central Processing Unit



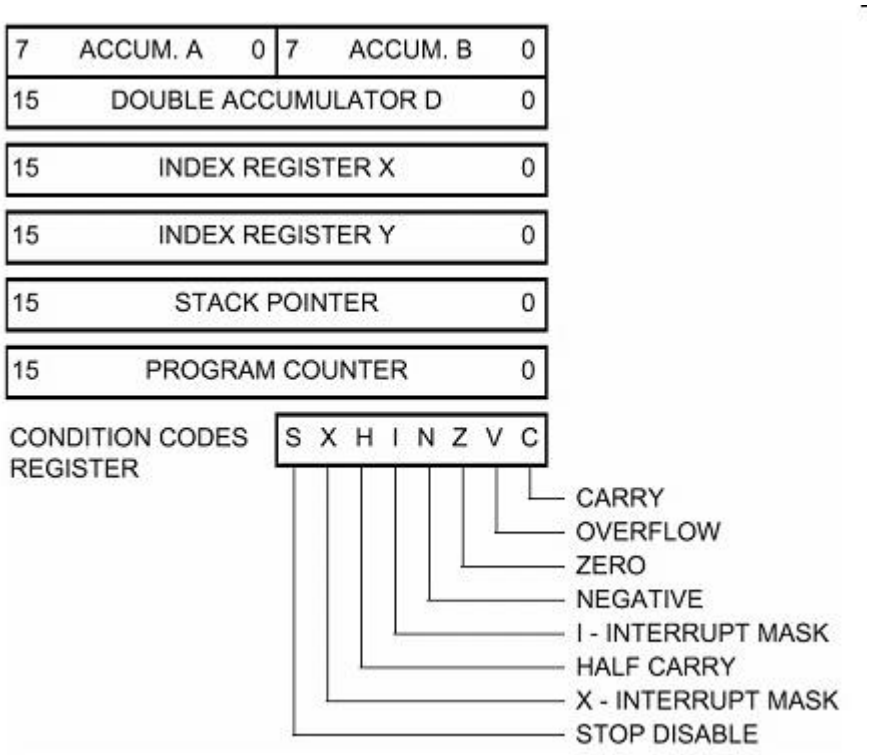
Registers

- **General-purpose registers** serve as temporary holding places for data being manipulated by the CPU.

- These registers hold the inputs to the arithmetic/logic unit's circuitry and provide storage space for results produced by that unit.

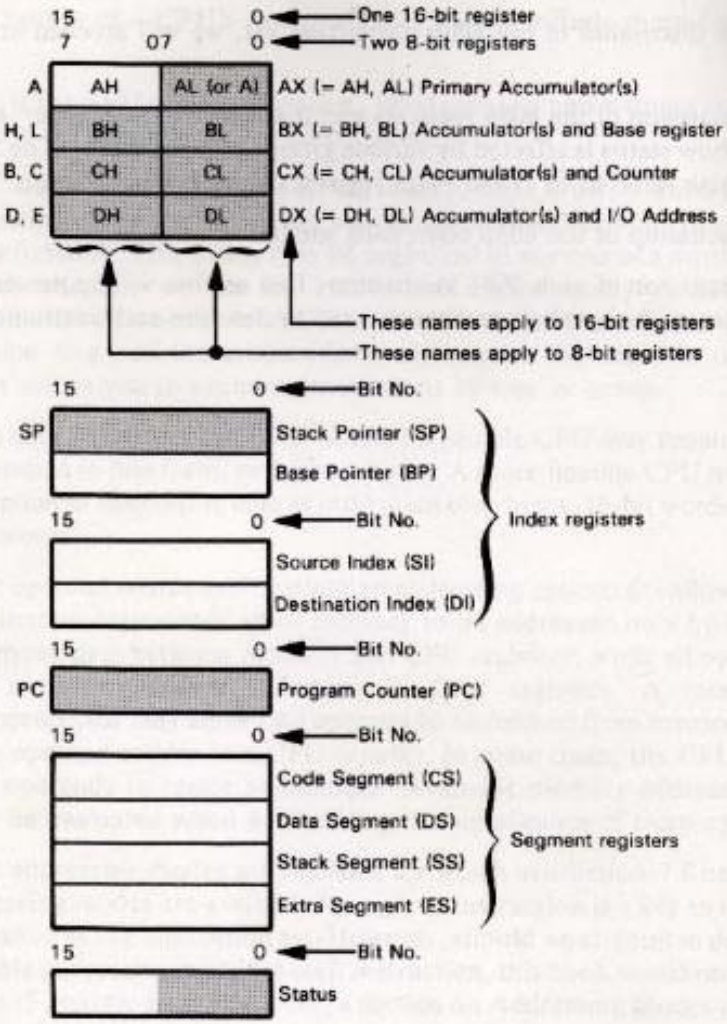
- To perform an operation on data stored in main memory, the control unit:
  - transfers the data from memory into the general-purpose registers,
  - informs the arithmetic/logic unit which registers hold the data,
  - activates the appropriate circuitry within the arithmetic/logic unit, and
  - tells the arithmetic/logic unit which register should receive the result.

Registers, 16 bit

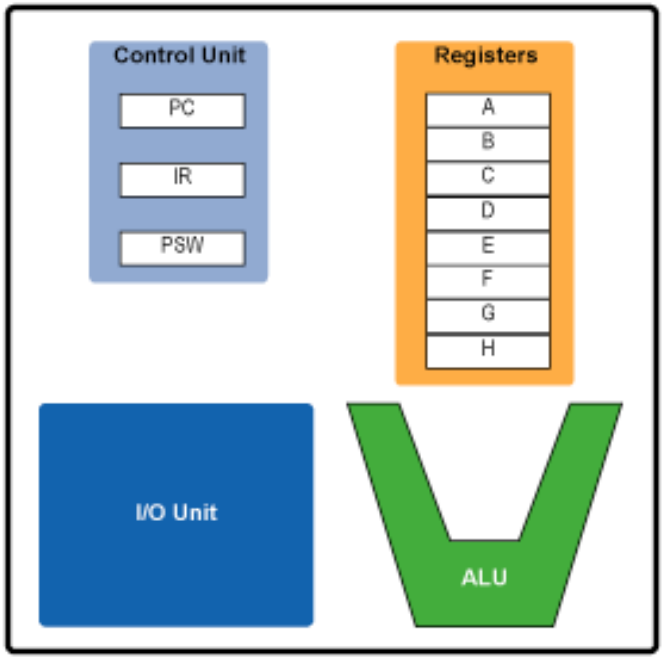


8086 REGISTERS AND FLAGS

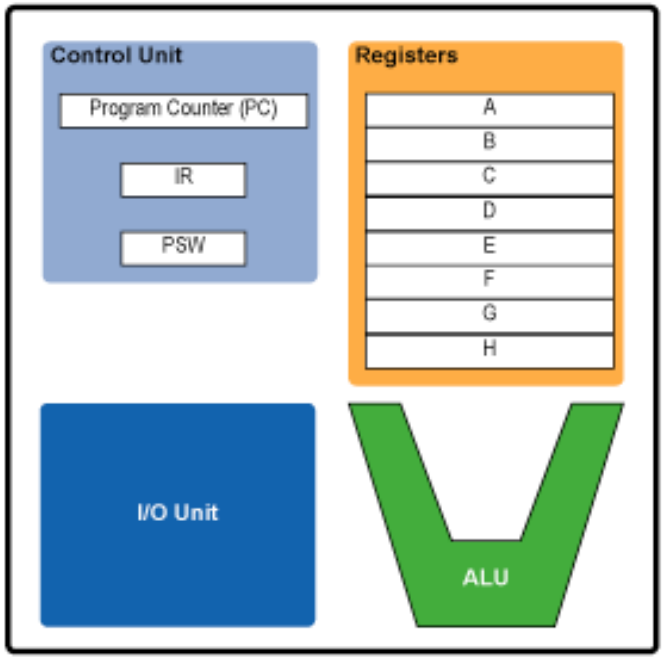
The 8086 has four 16-bit general purpose registers, two 16-bit Pointer registers, two 16-bit Index registers, one 16-bit program counter, four 16-bit Segment registers and one 16-bit Flags register. These registers may be illustrated as follows:



32 and 64 bit CPUs

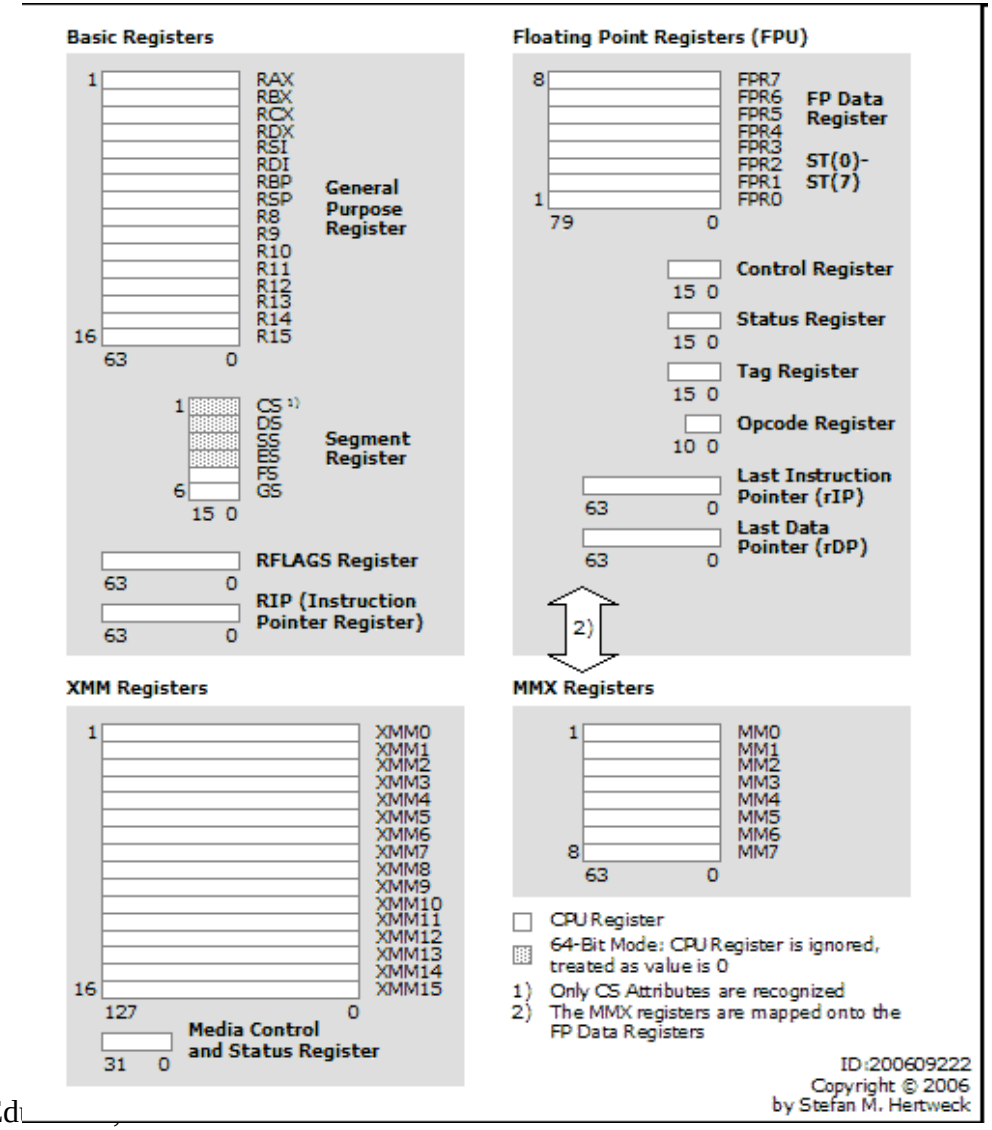


32-bit

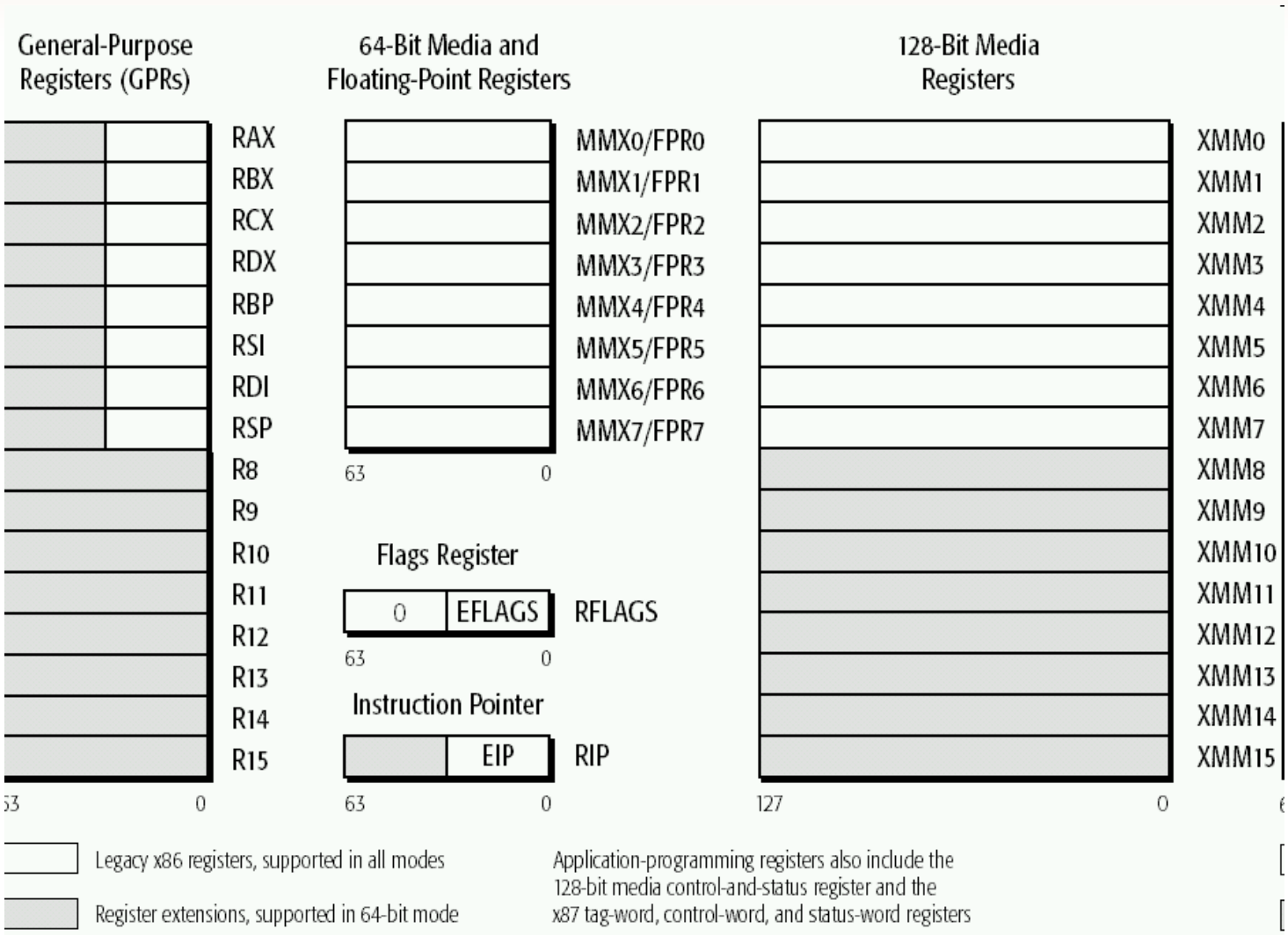


64-bit

Floating point registers (AMD)



Media registers (AMD)



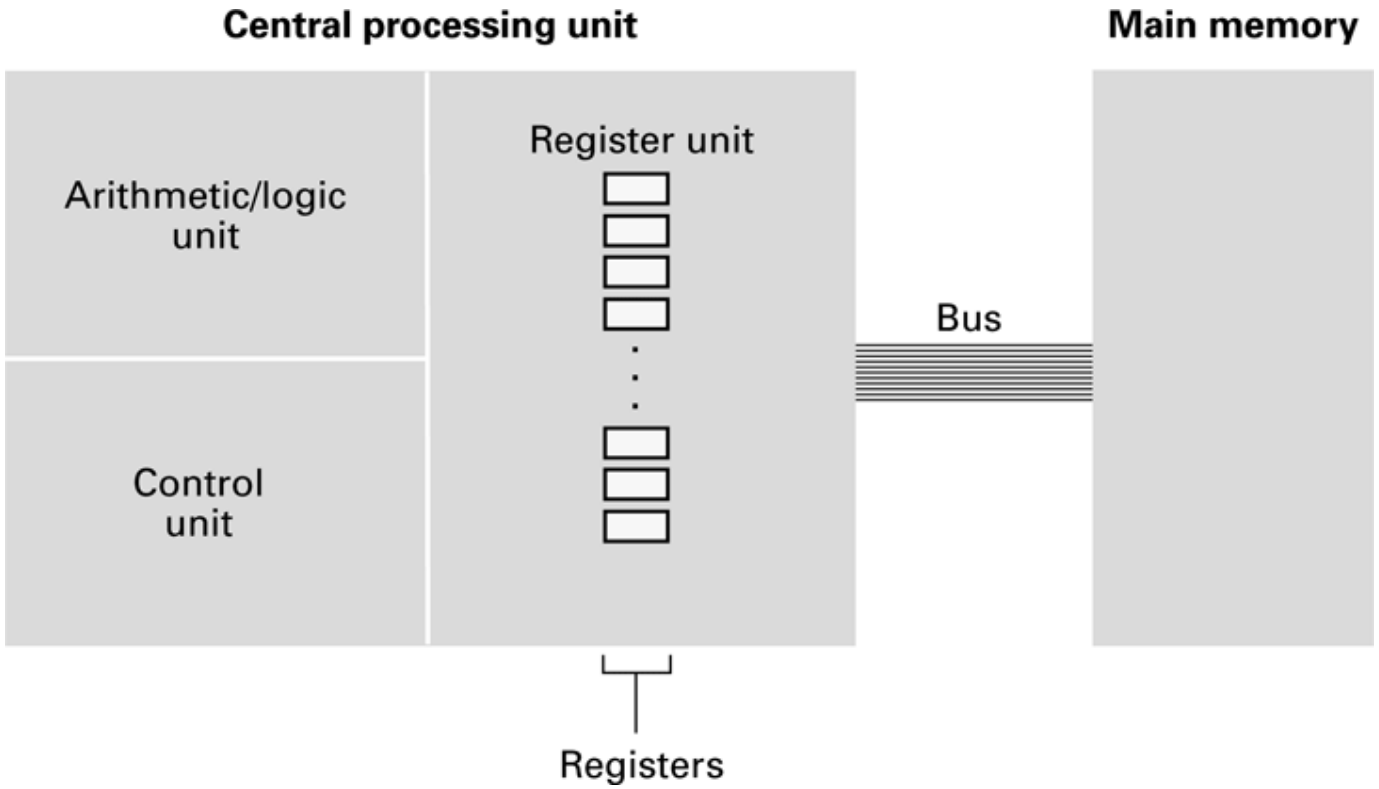
Bus

- For the purpose of transferring bit patterns, a machine's CPU and main memory are connected by a collection of wires called a bus.

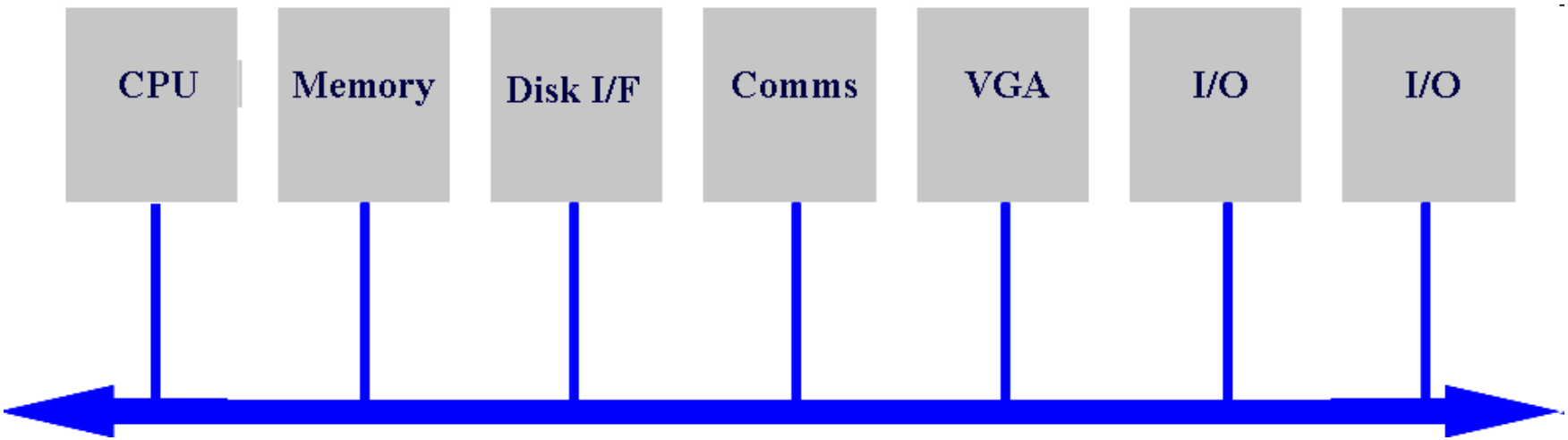
- Through this bus, the CPU extracts (**reads**) data from main memory by supplying the address of the pertinent memory cell along with an electronic signal telling the memory circuitry that it is supposed to retrieve the data in the indicated cell.

- In a similar manner, the CPU places (**writes**) data in memory by providing the address of the destination cell and the data to be stored together with the appropriate electronic signal telling main memory that it is supposed to store the data being sent to it.

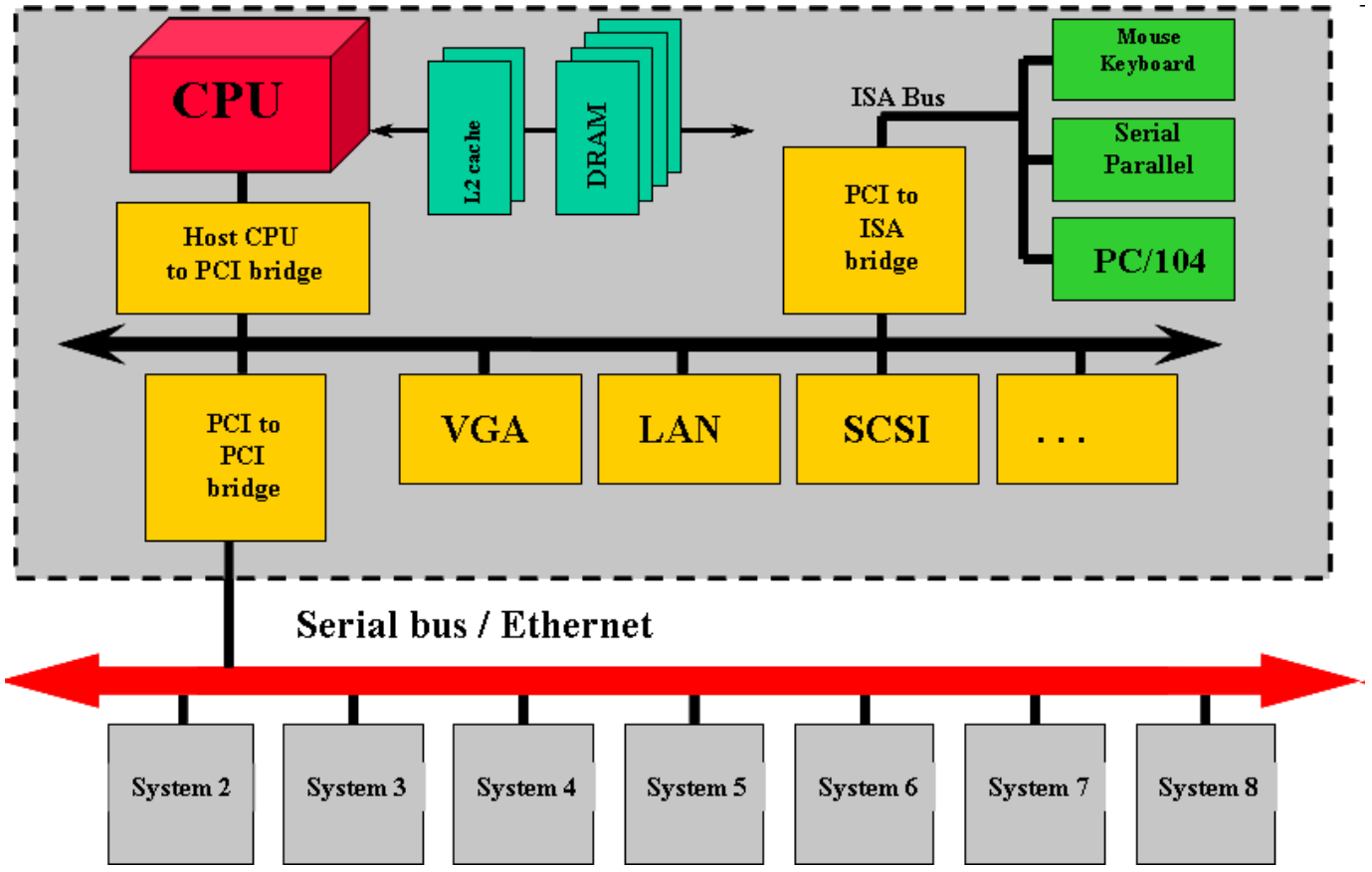
Figure 2.1 CPU and main memory connected via a bus



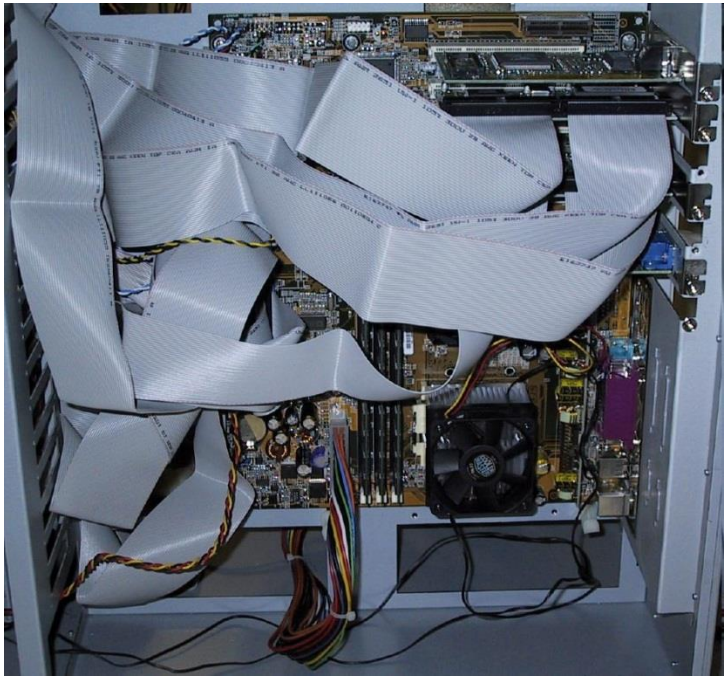
Bus within single computer



Bus among systems



Bus/Cable



Stored Program Concept

- A program can be encoded as bit patterns and stored in main memory.
- From there, the CPU can then extract the instructions and execute them.

- In turn, the program to be executed can be altered easily.

John Von Neumann

- John von Neumann (1903 –1957) was an Austro-Hungarian-born American mathematician who made major contributions to a vast range of fields, including set theory, functional analysis, quantum mechanics, ergodic theory, continuous geometry, economics and game theory, computer science, numerical analysis, hydrodynamics (of explosions), and statistics, as well as many other mathematical fields.
- He is generally regarded as one of the foremost mathematicians of the 20th century.

- The IEEE John von Neumann Medal is awarded annually by the IEEE "for outstanding achievements in computer-related science and technology.“
- On February 15, 1956, Neumann was presented with the Presidential Medal of Freedom by President Dwight Eisenhower

Von Neumann Architecture

- A breakthrough came with the realization that a program, just like data, can be **encoded and stored in main memory.**
- The control unit is designed:
  - to extract the program from memory,
  - decode the instructions,
  - and execute them,
- The program that the machine follows can be changed merely by changing the contents of the computer's memory instead of rewiring the control unit.

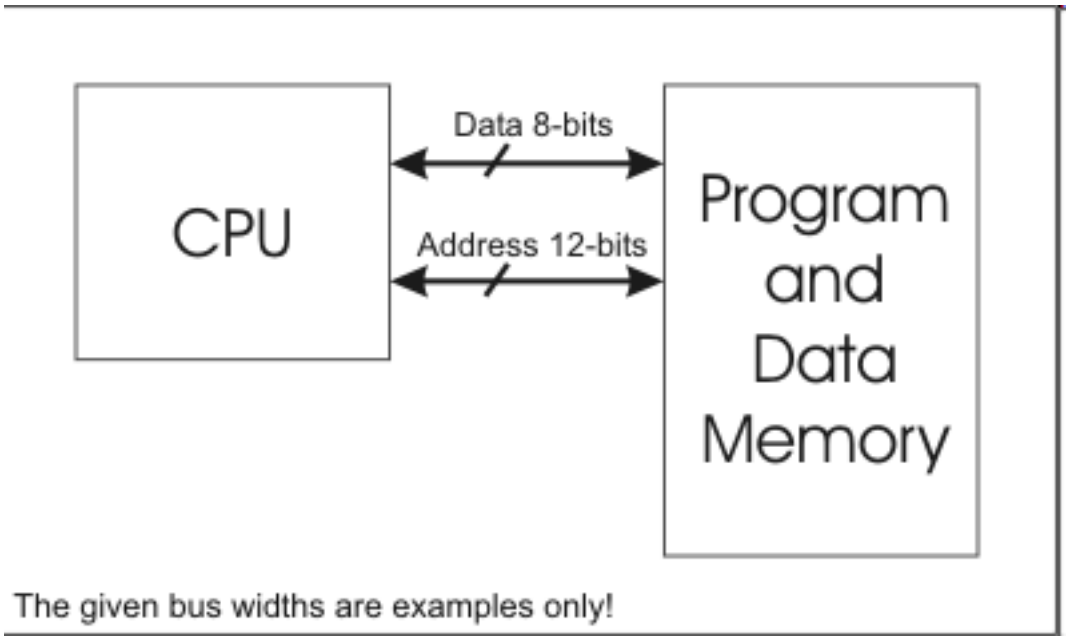
Single-memory computer architecture

- While consulting for the Moore School of Electrical Engineering on the EDVAC project, von Neumann wrote an incomplete set of notes titled the First Draft of a Report on the EDVAC.
  - The paper, which was widely distributed, described a computer architecture in which data and program memory are mapped into the same address space.
  - This architecture became the **de facto standard** and can be contrasted with a so-called **Harvard architecture**, which has separate program and data memories on a separate bus.

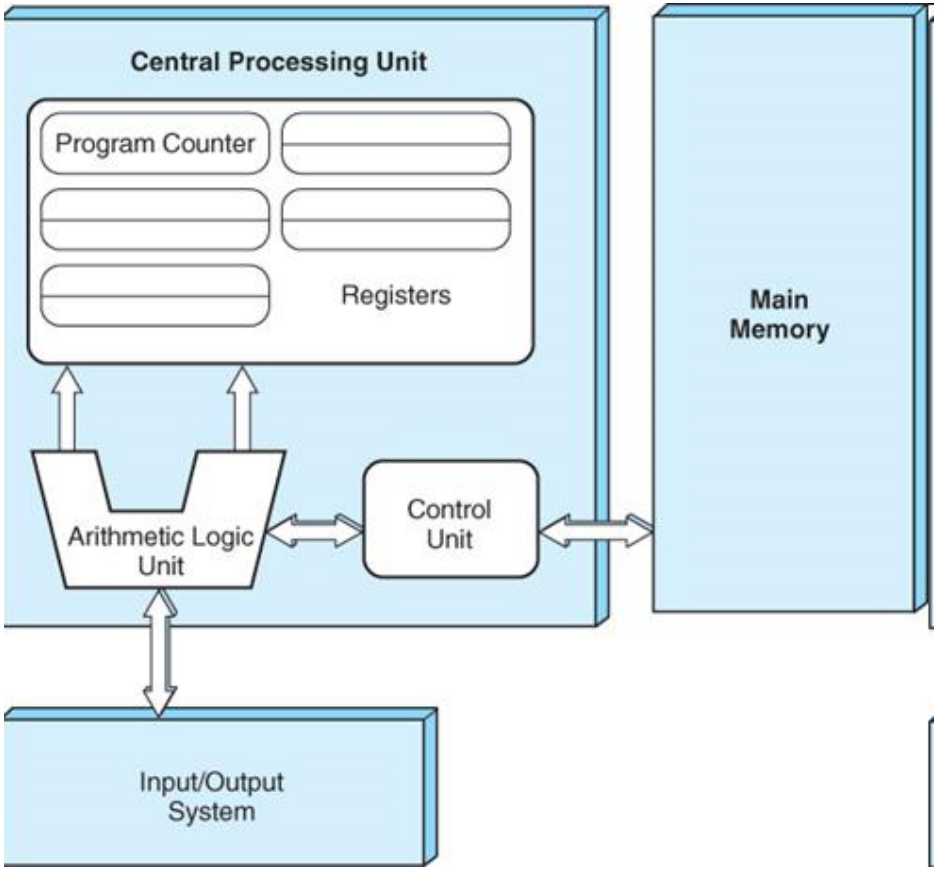
Who was the inventor?

- Although the single-memory architecture became commonly known by the name von Neumann architecture as a result of von Neumann's paper, the architecture's description was based on the work of J. Presper Eckert and John William Mauchly, inventors of the ENIAC at the University of Pennsylvania.
  - With very few exceptions, all present-day home computers, microcomputers, minicomputers and mainframe computers use this single-memory computer architecture.

Programs and data together



Von Neumann Architecture



Von Neumann bottleneck

- The separation between the CPU and memory leads to the ***Von Neumann bottleneck***, the limited throughput (data transfer rate) between the CPU and memory compared to the amount of memory.

- In most modern computers, throughput is much smaller than the rate at which the CPU can work.
  - This seriously limits the effective processing speed when the CPU is required to perform minimal processing on large amounts of data.
  - The CPU is continuously forced to wait for needed data to be transferred to or from memory.
  - Since CPU speed and memory size have increased much faster than the throughput between them, the bottleneck has become more of a problem.

- The performance problem is reduced by a **cache** between the CPU and the main memory.

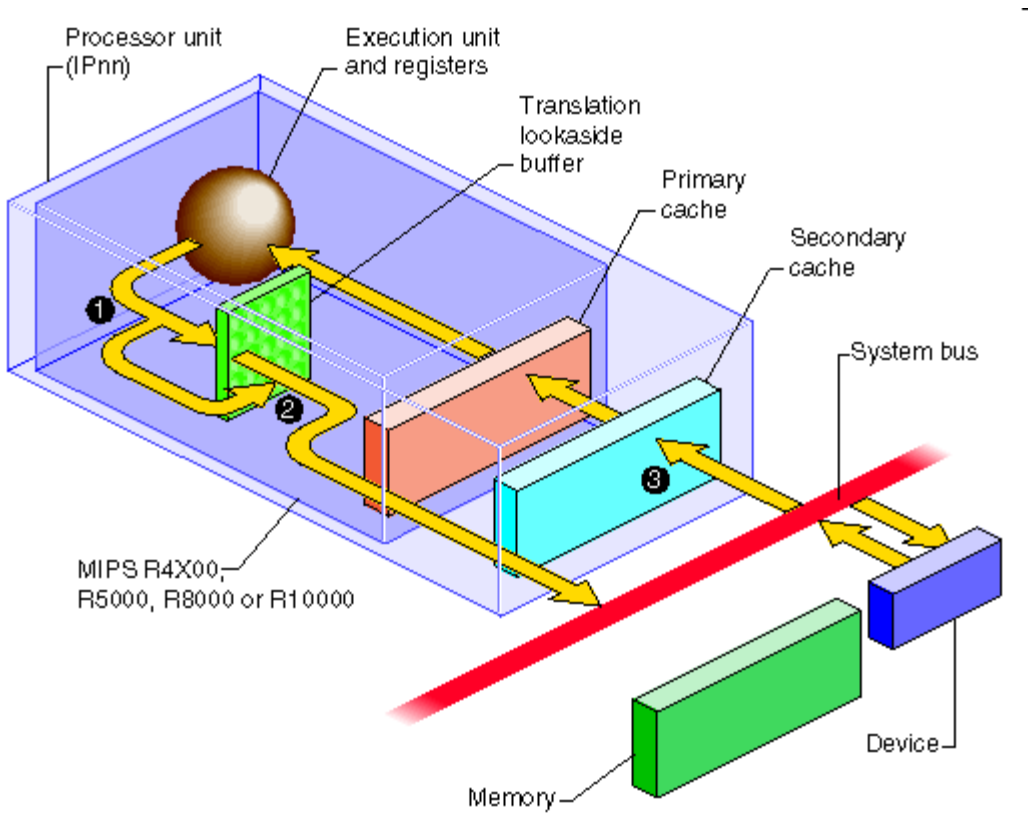
Cache memory

- It is instructive to compare the memory facilities within a computer in relation to their functionality.
  - Registers are used to hold the data immediately applicable to the operation at hand;
  - main memory is used to hold data that will be needed in the near future;
  - mass storage is used to hold data that will likely not be needed in the immediate future.
- Many machines are designed with an additional memory level, called cache memory.
- Cache memory is a portion (perhaps several hundred MB) of high-speed memory located within the CPU itself.

Cache memory

- In the cache, the machine attempts to keep a copy of that portion of main memory that is of current interest.
- In this setting, data transfers that normally would be made between registers and main memory are made between registers and cache memory.
- Any changes made to cache memory are then transferred collectively to main memory at a more opportune time.
- The result is a CPU that can execute its machine cycle more rapidly because it is not delayed by main memory communication.

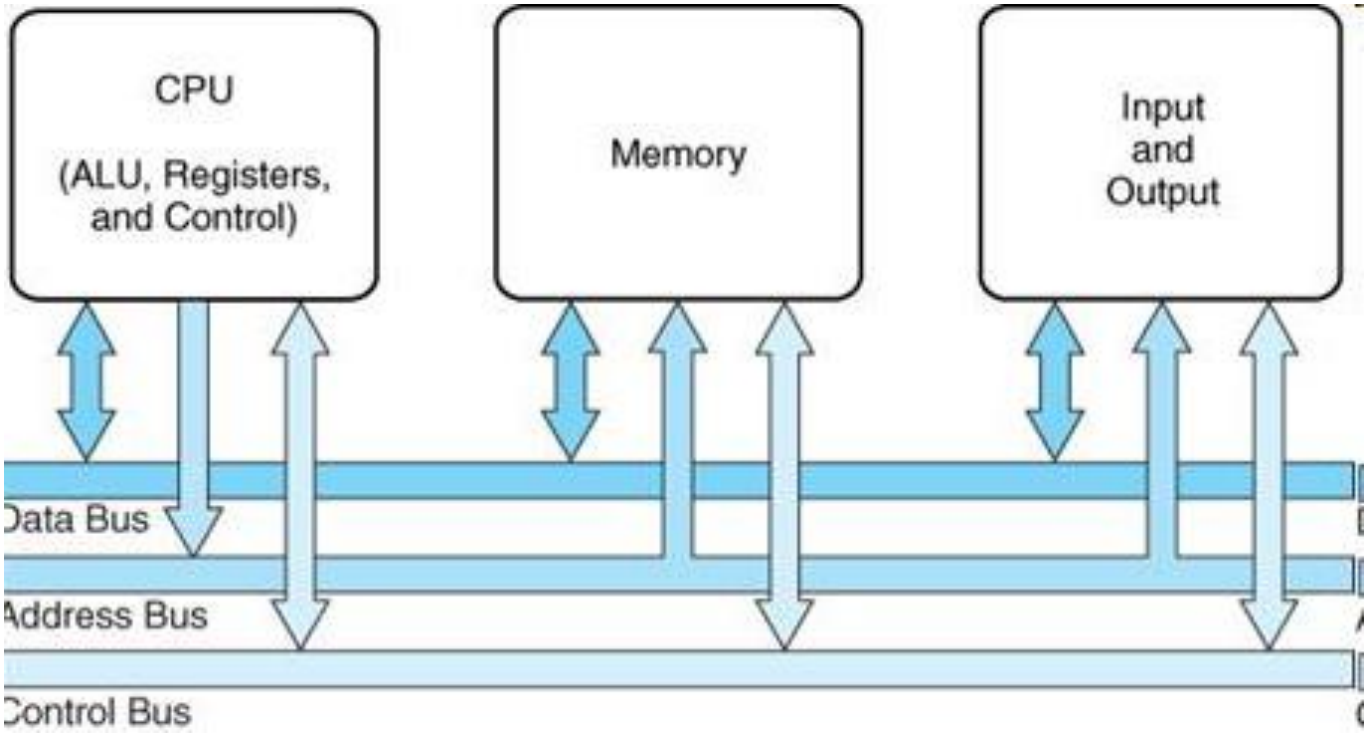
Caches and bus: where data stand and go



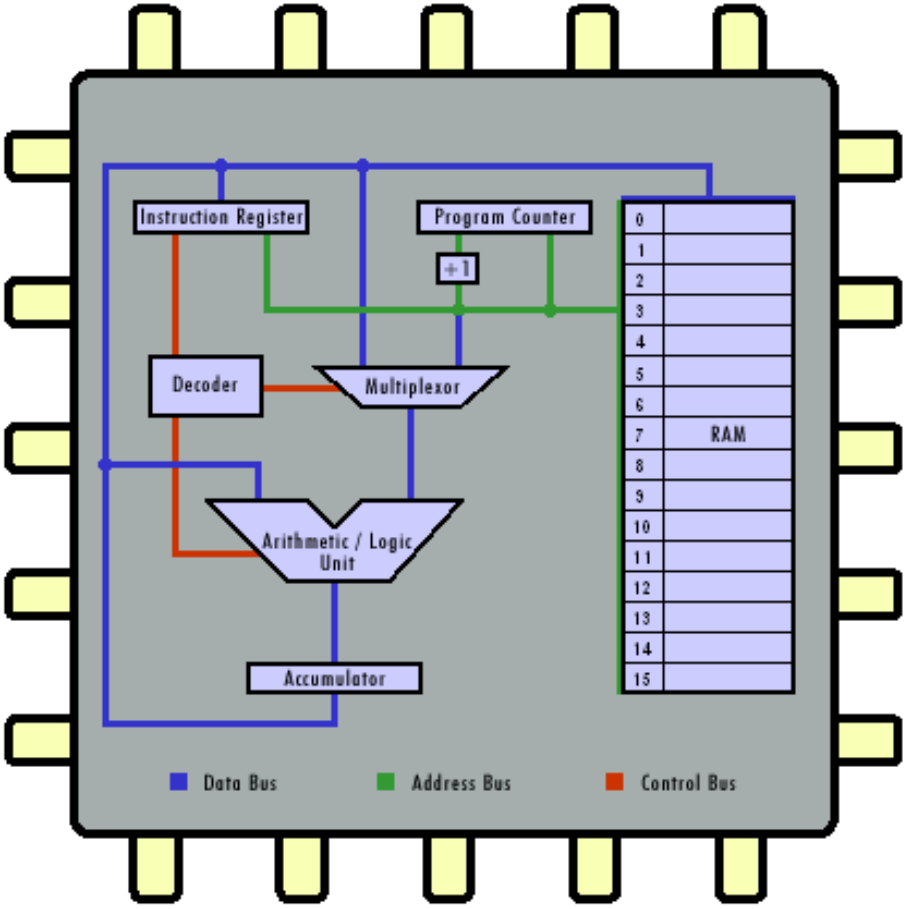
Extensions to Von Neumann architecture

- The ideas present in von Neumann architecture have been extended. (**von Neumann bottleneck**).
- The **data bus** moves data from main memory to the CPU registers (and vice versa).
- The **address bus** holds the address of the data that the data bus is currently accessing.
- The **control bus** carries the necessary control signals that specify how the information transfer is to take place.

Extensions to Von Neumann architecture



CPU: Data bus and address bus



Improved architecture with more registers

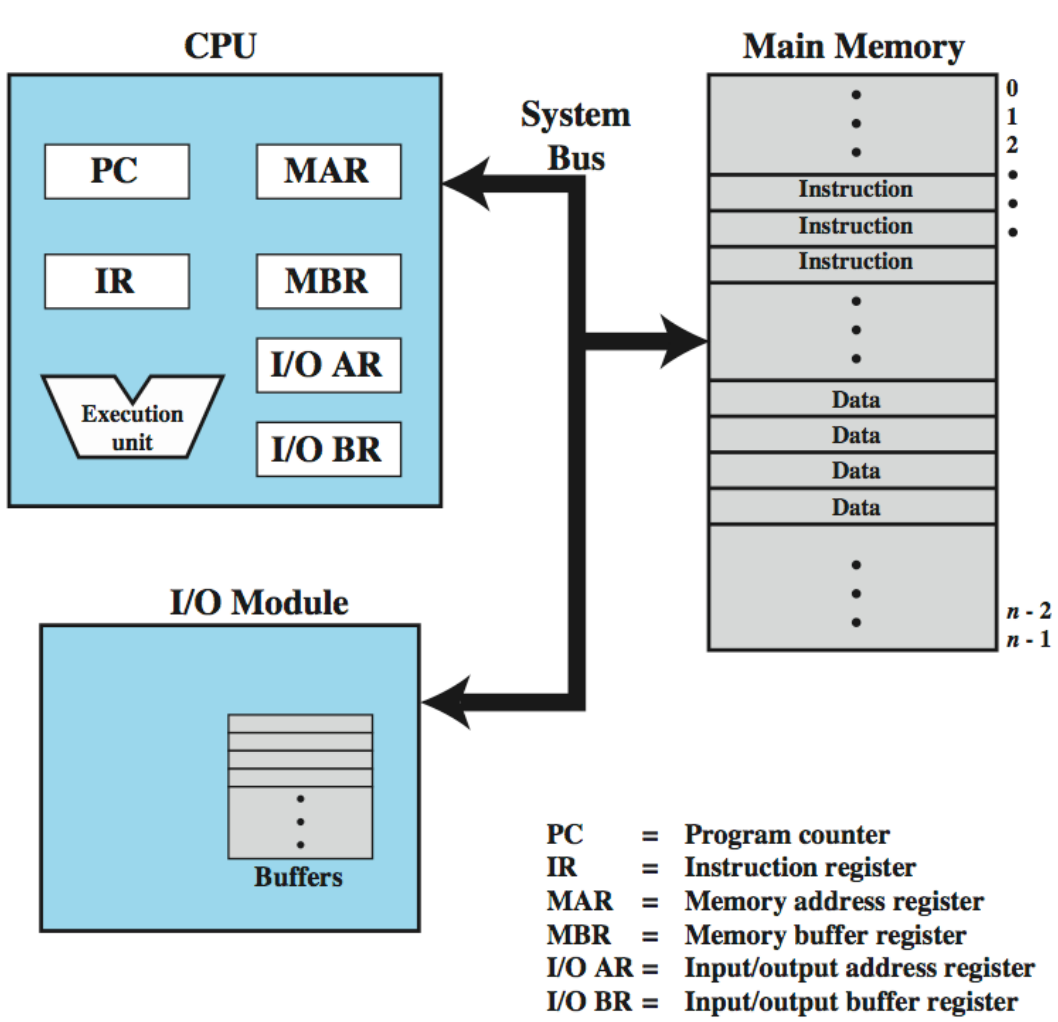


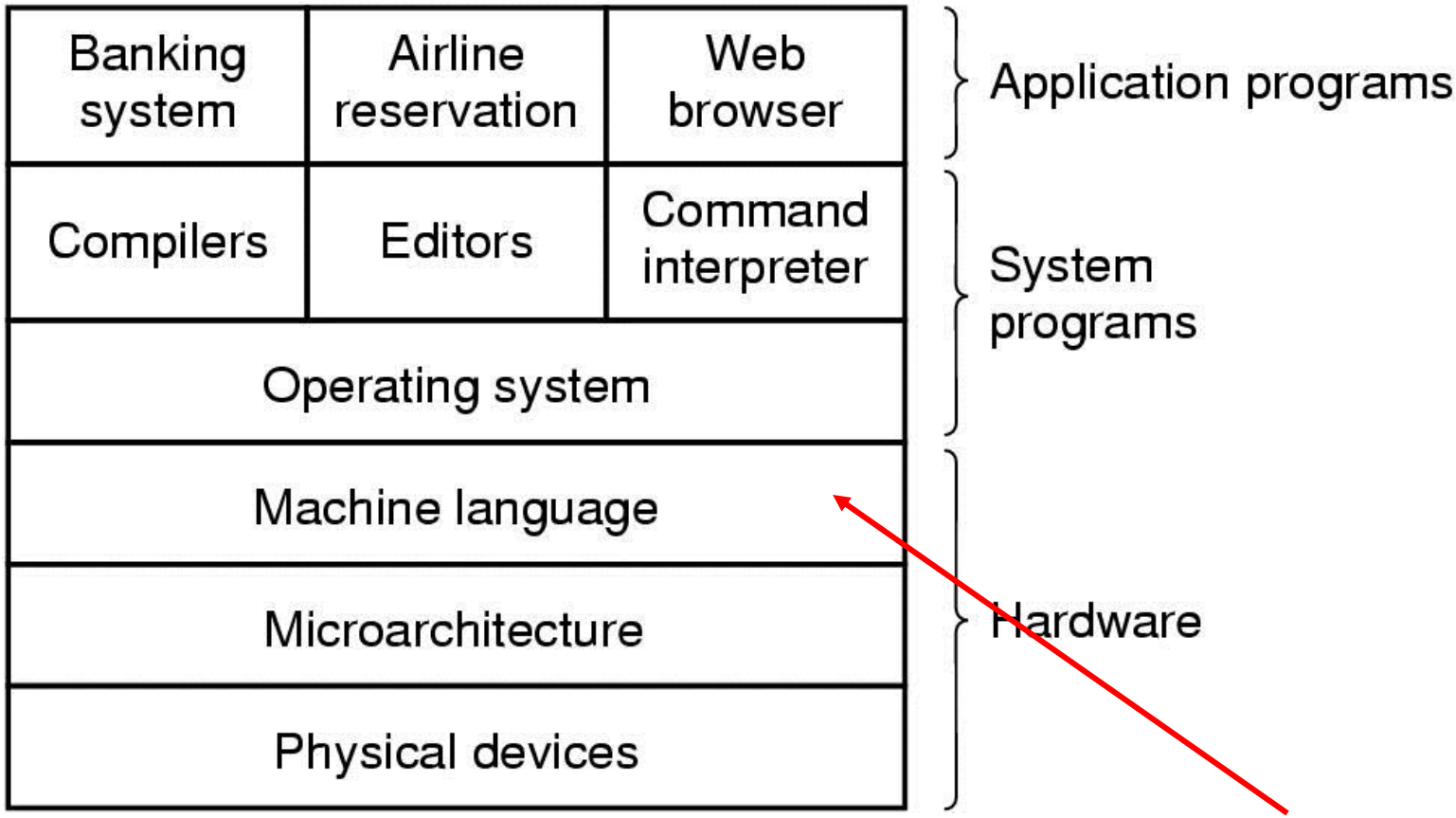
Figure 1.1 Computer Components: Top-Level View

Other enhancements to the von Neumann architecture include using index registers for addressing, adding floating point data, using interrupts and asynchronous I/O, adding virtual memory, and adding general registers.

Data Manipulation

- 2.1 Computer Architecture
- 2.2 Machine Language
- 2.3 Program Execution
- 2.4 Arithmetic/Logic Instructions
- 2.5 Communicating with Other Devices
- 2.6 Other Architectures

Where is machine language?



Instructions to the CPU

- To apply the stored-program concept, CPUs are designed to recognize instructions encoded as bit patterns.
- This collection of instructions along with the encoding system is called the machine language.
- An instruction expressed in this language is called a machine-level instruction or, more commonly, a machine instruction.

Terminology

- Machine instruction: An instruction (or command) encoded as a bit pattern recognizable by the CPU

- Machine language: The set of all instructions recognized by a machine

Machine capability

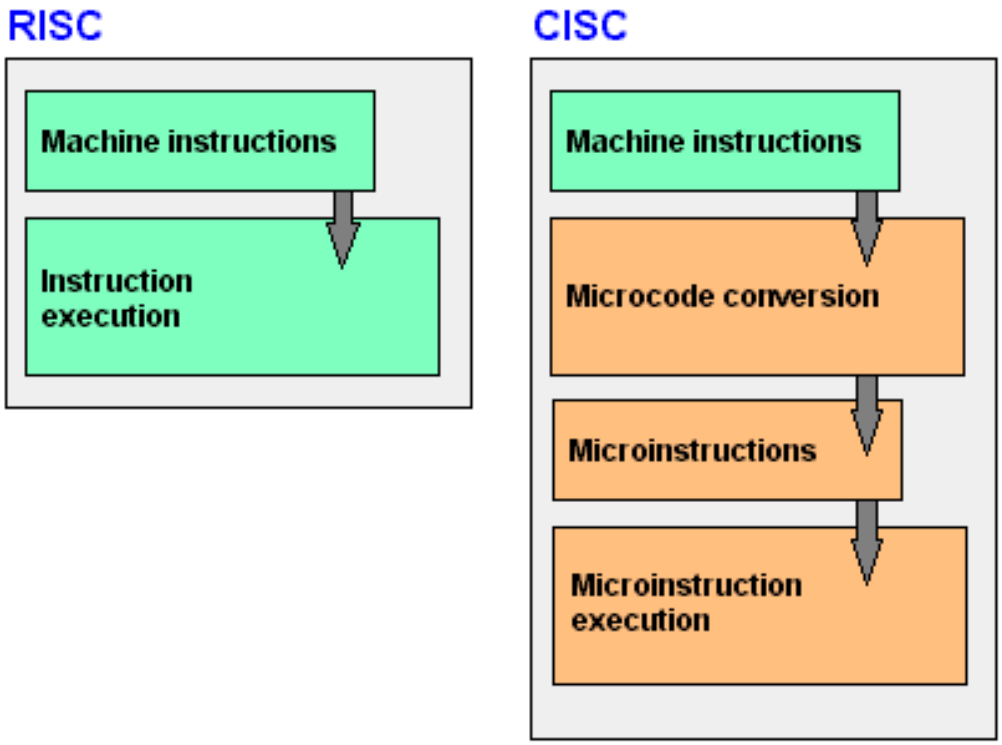
- The list of machine instructions that a typical CPU must be able to decode and execute is quite short.
- Indeed, once a machine can perform certain elementary but well-chosen tasks, adding more features does not increase the machine's theoretical capabilities.
- In other words, beyond a certain point, additional features may increase such things as convenience but add nothing to the machine's fundamental capabilities.

Machine Language Philosophies

- Reduced Instruction Set Computing (RISC)
  - Few, simple, efficient, and fast instructions
  - Examples: PowerPC from Apple/IBM/Motorola and SPARK from Sun Microsystems
- Complex Instruction Set Computing (CISC)
  - Many, convenient, and powerful instructions
  - Example: Pentium from Intel

RISC vs. CISC

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.



RISC - Reduced Instruction Set Computer

- A computer architecture that reduces chip complexity by using simpler instructions.
- RISC compilers have to generate software routines to perform the equivalent processing performed by more comprehensive instructions in "complex instruction set computers" (CISC computers).

- RISC - No Microcode
  - In a RISC computer, there is no microcode layer, and the associated overhead of that translation is eliminated.
  - RISC keeps instruction size constant
  - It retains only those instructions that can be overlapped and made to execute in one machine cycle or less.

Machine Instruction Types

- Data Transfer: copy data from one location to another
- Arithmetic/Logic: use existing bit patterns to compute a new bit patterns
- Control: direct the execution of the program

Data transfer

- The data transfer group consists of instructions that request the movement of data from one location to another.
  - We should note that using terms such as transfer or move to identify this group of instructions is actually a misnomer.
  - It is rare that the data being transferred is erased from its original location.
- The process involved in a transfer instruction is more like copying the data rather than moving it.
- Thus terms such as copy or clone better describe the actions of this group of instructions.

Figure 2.2 Adding values stored in memory

Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

I/O instructions

- An important group of instructions within the data transfer category consists of the commands for communicating with devices outside the CPU - main memory context (printers, keyboards, monitors, disk drives, etc.).
- Since these instructions handle the input/output (I/O) activities of the machine, they are called I/O instructions and are sometimes considered as a category in their own right.
- The I/O activities can be handled by the same instructions that request data transfers between the CPU and main memory.
- Thus, we shall consider the I/O instructions to be a part of the data transfer group.

Arithmetic/logic Instructions

- The arithmetic/logic group consists of the instructions that tell the control unit to request an activity within the arithmetic/logic unit.
- The arithmetic/logic unit is capable of performing operations other than the basic arithmetic operations.
- Some of these additional operations are the Boolean operations AND, OR, and XOR.
- Another collection of operations available within most arithmetic/logic units allows the contents of registers to be moved to the right or the left within the register.
  - These operations are known as either SHIFT or ROTATE operations, depending on whether the bits that "fall off the end" of the register are merely discarded (SHIFT) or are used to fill the holes left at the other end (ROTATE).

Figure 2.3 Dividing values stored in memory

Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

Step 3. If this second value is zero, JUMP to Step 6.

Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

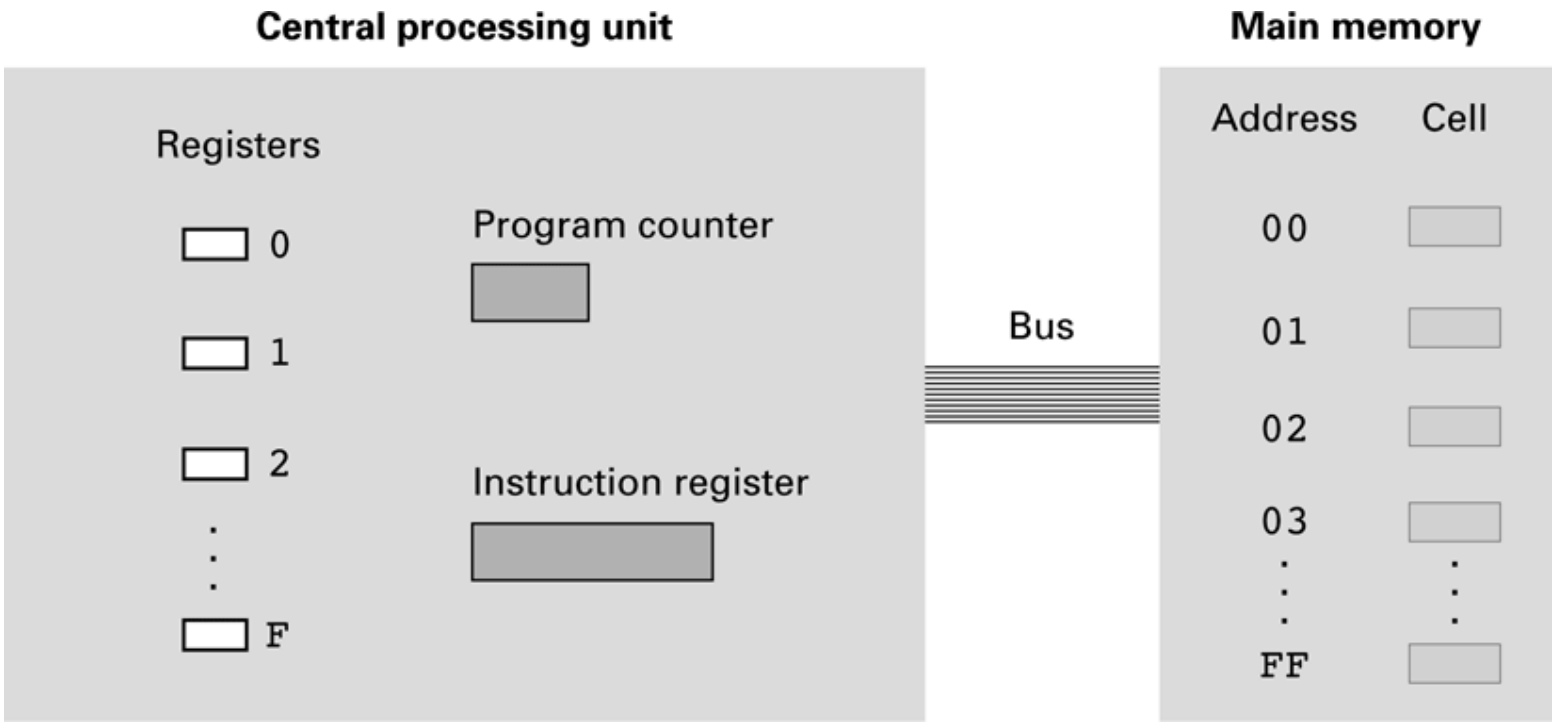
Step 5. STORE the contents of the third register in memory.

Step 6. STOP.

Control Instructions

- The control group consists of those instructions that direct the execution of the program rather than the manipulation of data.
- This group contains many of the more interesting instructions in a machine's repertoire, such as the family of JUMP (or BRANCH) instructions used to direct the control unit to execute an instruction other than the next one in the list.
- These JUMP instructions appear in two varieties: unconditional jumps and conditional jumps.
- The distinction is that a conditional jump results in a "change of venue" only if a certain condition is satisfied.
- As an example, the sequence of instructions in Figure 2.3 represents an algorithm for dividing two values where Step 3 is a conditional jump that protects against the possibility of division by zero.

Figure 2.4 The architecture of the machine described in Appendix C



Parts of a Machine Instruction

- **Op-code:** Specifies which operation to execute
- **Operand:** Gives more detailed information about the operation
  - Interpretation of operand varies depending on op-code

Instruction fields

- The bit pattern appearing in the op-code field indicates which of the elementary operations, such as STORE, SHIFT, XOR, and JUMP, is requested by the instruction.
- The bit patterns found in the operand field provide more detailed information.
  - For example, in the case of a STORE operation, the information in the operand field indicates which register contains the data to be stored and which memory cell is to receive the data.

Figure 2.6 Decoding the instruction 35A7

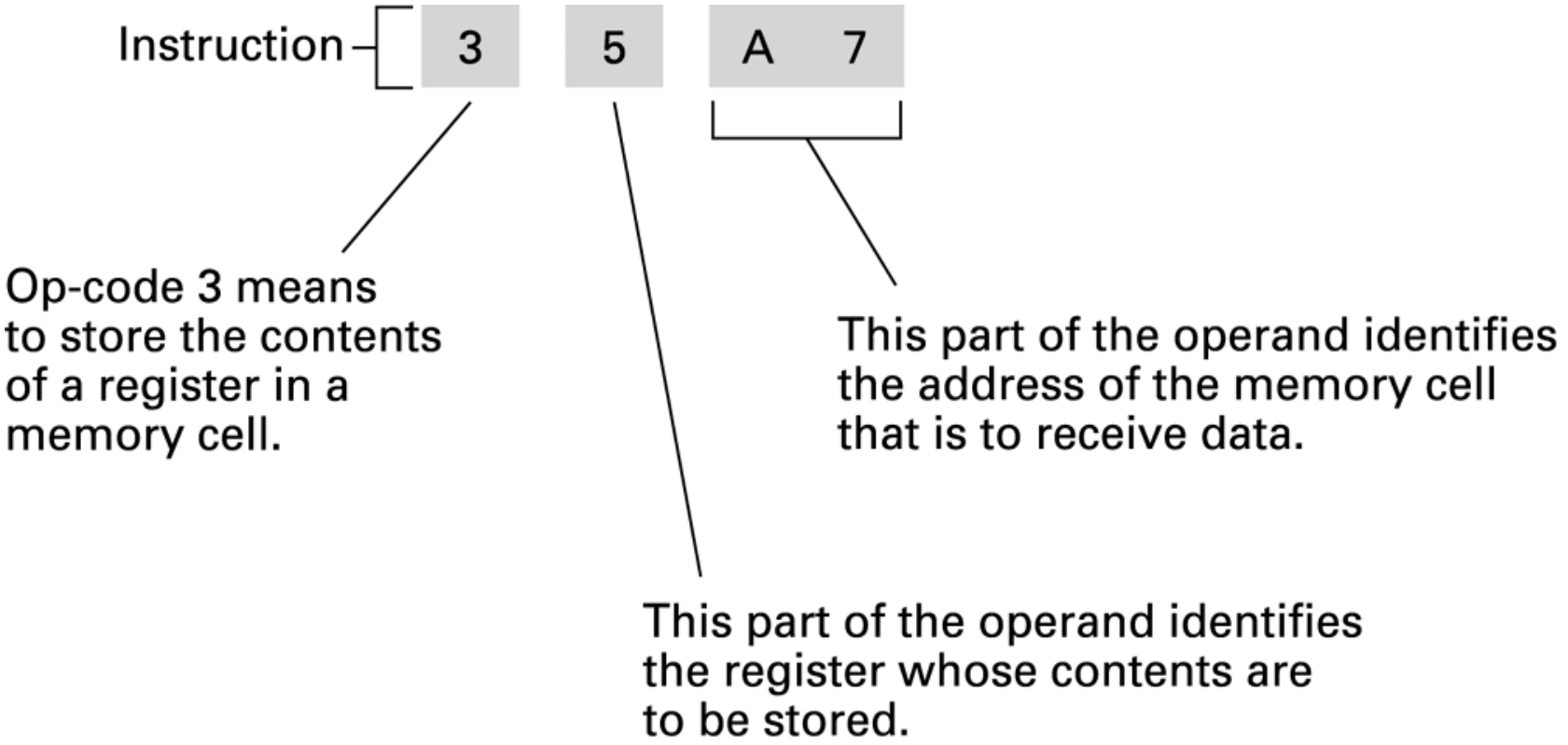
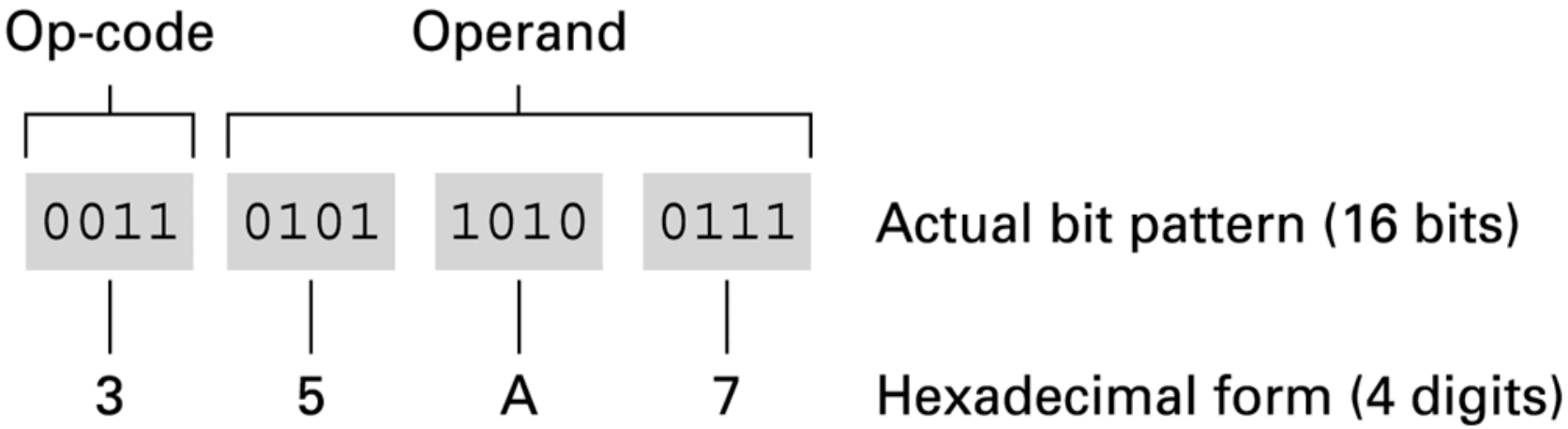


Figure 2.7 An encoded version of the instructions in Figure 2.2

Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

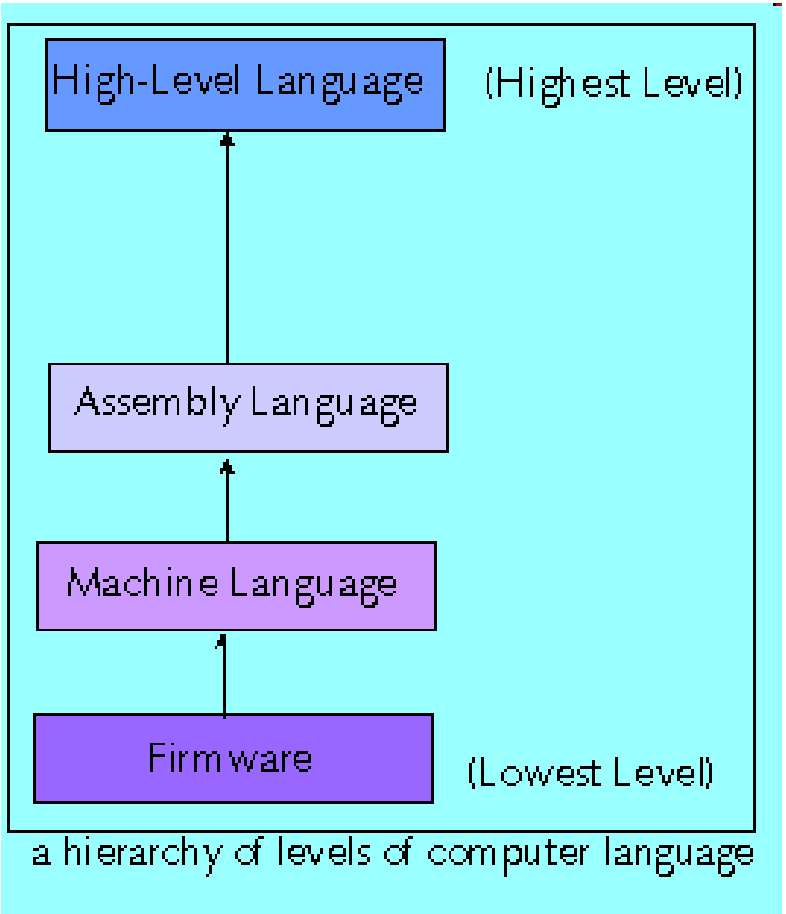
Figure 2.5 The composition of an instruction for the machine in Appendix C



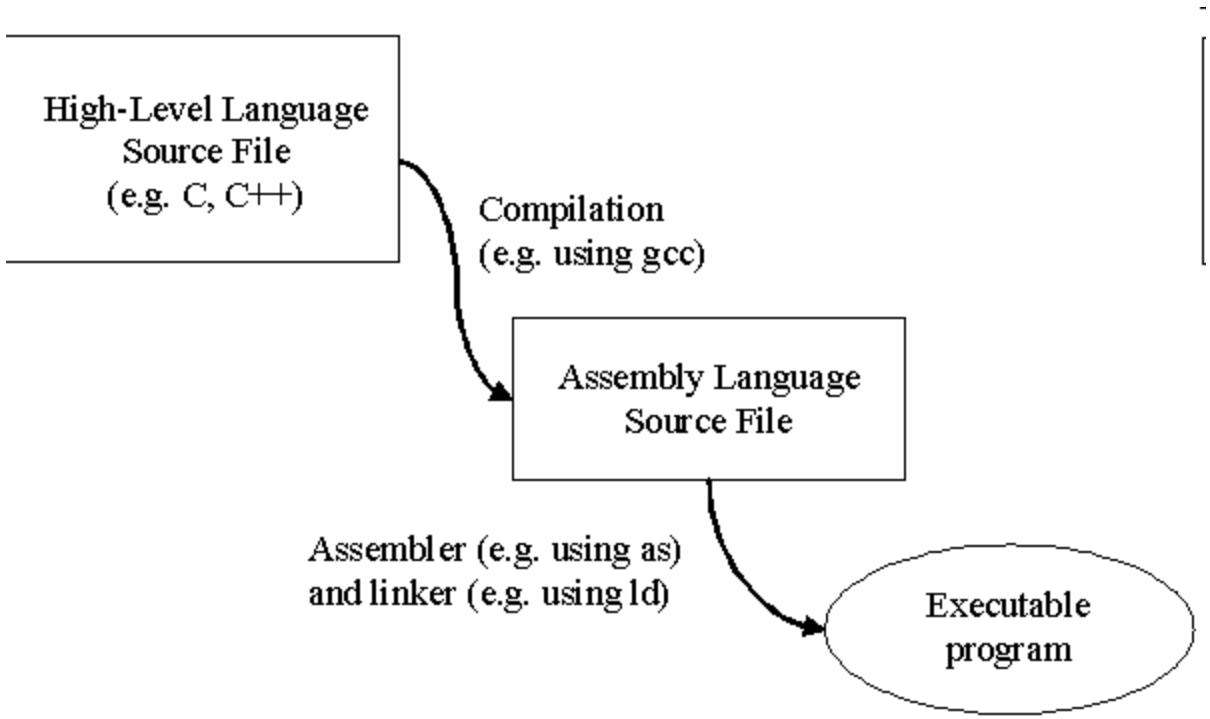
Assembly language

- Assembly languages are a family of low-level languages for programming computers, microprocessors, microcontrollers, and other (usually) integrated circuits.
- They implement a symbolic representation of the numeric machine codes and other constants needed to program a particular CPU architecture.
- This representation is usually defined by the hardware manufacturer, and is based on abbreviations (called mnemonics) that help the programmer remember individual instructions, registers, etc.
- An assembly language is thus specific to a certain physical or virtual computer architecture (as opposed to most high-level languages, which are usually portable).
- A utility program called an assembler is used to translate assembly language statements into the target computer's machine code.
- The assembler performs a more or less isomorphic translation (a one-to-one mapping) from mnemonic statements into machine instructions and data.
- This is in contrast with high-level languages, in which a single statement generally results in many machine instructions.

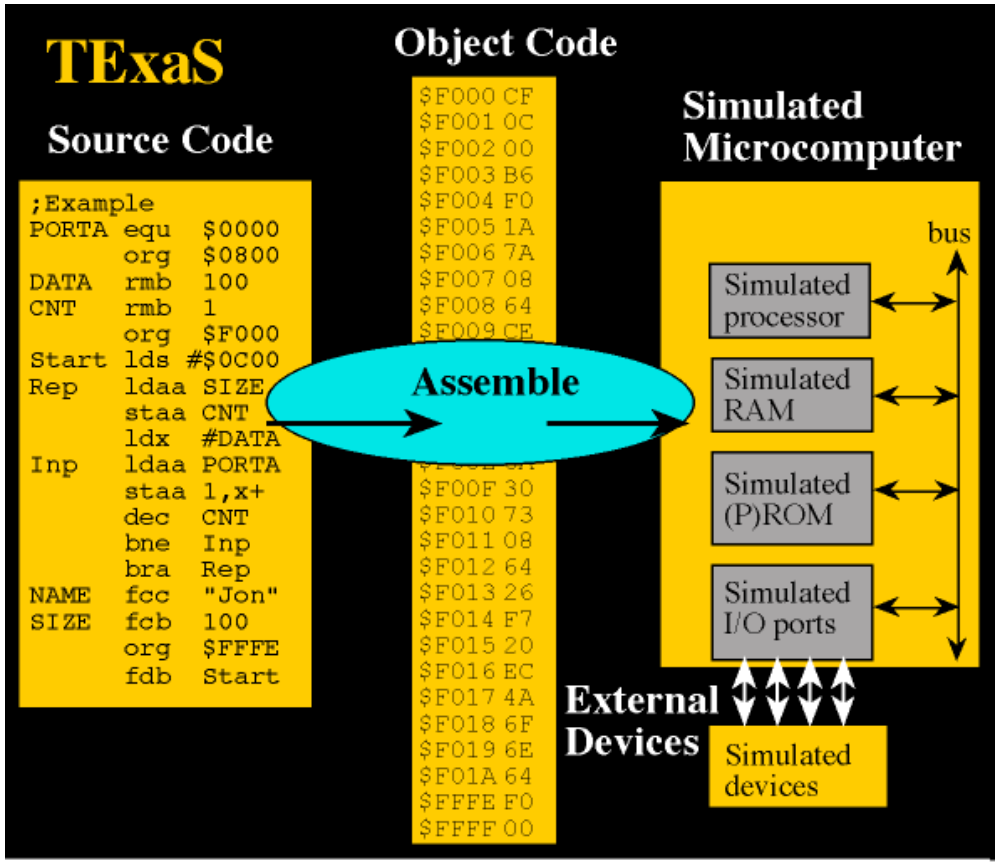
Levels of programming languages



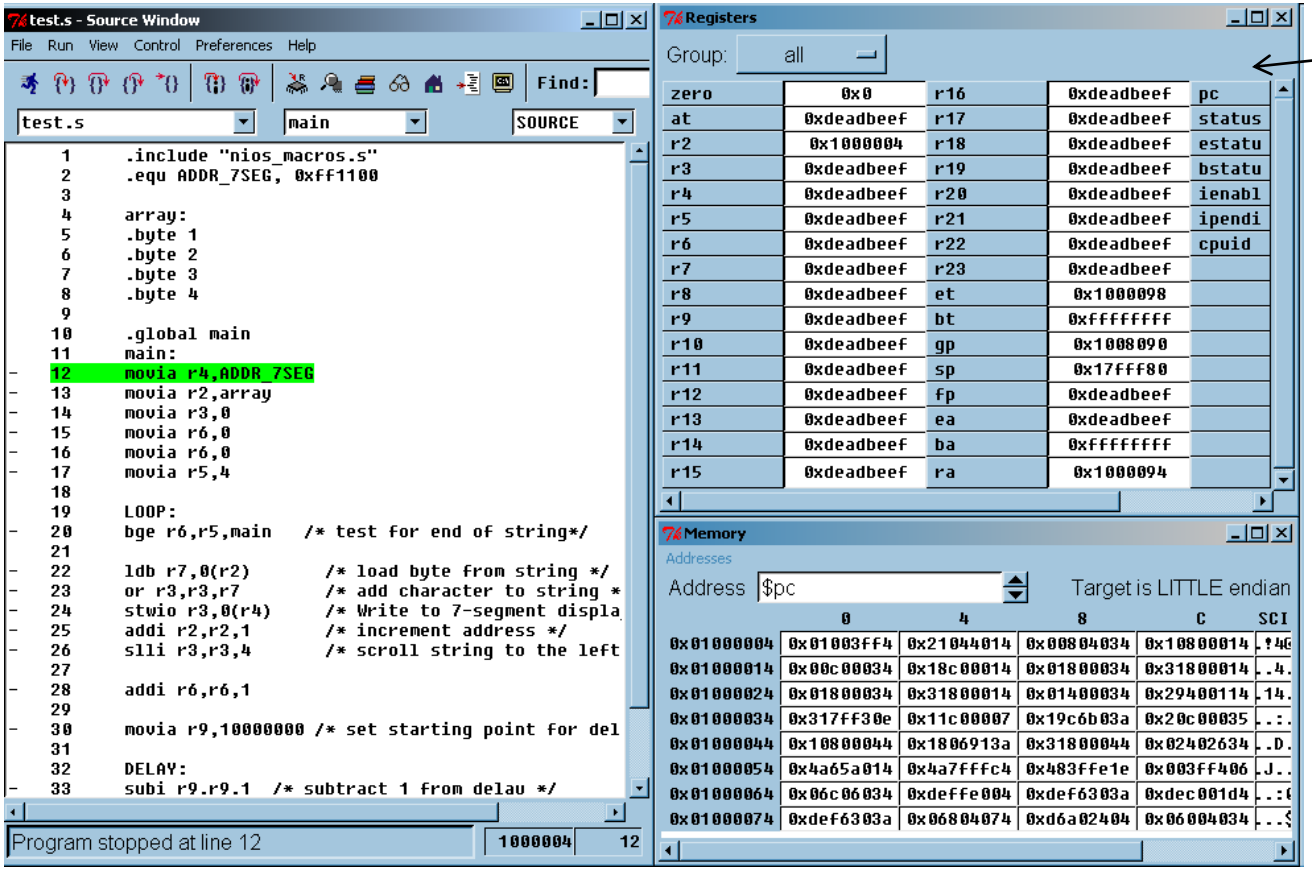
From high level to machine code



Assembler: from source code to machine code



Programming in Assembly



PC: program counter

Registers

Memory

Assembly: working with registers

```
; Example of IBM PC assembly language
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32   PROC          ; procedure begins here
        CMP   AX,97   ; compare AX to 97
        JL    DONE    ; if less, jump to DONE
        CMP   AX,122  ; compare AX to 122
        JG    DONE    ; if greater, jump to DONE
        SUB   AX,32   ; subtract 32 from AX
DONE:   RET           ; return to main program
SUB32   ENDP          ; procedure ends here
```

FIGURE 17. Assembly language

The same program in high level language:

AX = has a certain value;
If(AX < 97 OR AX > 122) return;
else
AX = AX - 32

Assembly is fast and fits in small memories ☺

MenuetOS, a new operating system that is built in assemby language, not only does this OS fit on a floppy disk but it's incredibly fast and efficient. No surprise there: assembly language is designed to work close to the hardware.

Example of Machine Language - 1

- The machine has 16 general-purpose registers numbered 0 through F (in hexadecimal).
- Each register is one byte (eight bits) long.
- For identifying registers within instructions, each register is assigned the unique four-bit pattern that represents its register number.
- Thus register 0 is identified by 0000 (hexadecimal 0), and register 4 is identified by 0100 (hexadecimal 4).

Example of Machine Language - 2

- There are 256 cells in the machine's main memory. Each cell is assigned a unique address consisting of an integer in the range of 0 to 255.

- An address can therefore be represented by a pattern of eight bits ranging from 00000000 to 11111111 (or a hexadecimal value in the range of 00 to FF).

Example of Machine Language - 3

- Each machine instruction is two byte's long.
- The first 4 bits provide the op-code
- The last 12 bits make up the operand field.
- The table in next slide lists the instructions in hexadecimal notation together with a short description of each.
- The letters R, S, and T are used in place of hexadecimal digits in those fields representing a register identifier (that varies depending on the particular application of the instruction.
- The letters X and Y are used instead of hexadecimal digits in variable fields not representing a register.

Instructions (Book - Appendix A)

Op-code	Operand	Description
1	RXY	LOAD the register R with the bit pattern found in the memory cell whose address is XY. *Example:* 14A3 would cause the contents of the memory cell located at address A3 to be placed in register 4.
2	RXY	LOAD the register R with the bit pattern XY. *Example:* 20A3 would cause the value A3 to be placed in register 0.

Instructions (Book - Appendix A)

3	RXY	STORE the bit pattern found in register R in the memory cell whose address is XY. *Example:* 35B1 would cause the contents of register 5 to be placed in the memory cell whose address is B1.
4	0RS	MOVE the bit pattern found in register R to register S. *Example:* 40A4 would cause the contents of register A to be copied into register 4.
5	RST	ADD the bit patterns in registers S and T as though they were two's complement representations and leave the result in register R. *Example:* 5726 would cause the binary values in registers 2 and 6 to be added and the sum placed in register 7.
6	RST	ADD the bit patterns in registers S and T as though they represented values in floating-point notation and leave the floating-point result in register R. *Example:* 634E would cause the values in registers 4 and E to be added as floating-point values and the result to be placed in register 3.
7	RST	OR the bit patterns in registers S and T and place the result in register R. *Example:* 7CB4 would cause the result of ORing the contents of registers B and 4 to be placed in register C.

Instructions (Book - Appendix A)

8	RST	AND the bit patterns in registers S and T and place the result in register R. *Example:* 8045 would cause the result of ANDing the contents of registers 4 and 5 to be placed in register 0.
9	RST	EXCLUSIVE OR the bit patterns in registers S and T and place the result in register R. *Example:* 95F3 would cause the result of EXCLUSIVE ORing the contents of registers F and 3 to be placed in register 5.
A	R0X	ROTATE the bit pattern in register R one bit to the right X times. Each time place the bit that started at the low-order end at the high-order end. *Example:* A403 would cause the contents of register 4 to be rotated 3 bits to the right in a circular fashion.
B	RXY	JUMP to the instruction located in the memory cell at address XY if the bit pattern in register R is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution. (The jump is implemented by copying XY into the program counter during the execute phase.) *Example:* B43C would first compare the contents of register 4 with the contents of register 0. If the two were equal, the pattern 3C would be placed in the program counter so that the next instruction executed would be the one located at that memory address. Otherwise, nothing would be done and program execution would continue in its normal sequence.
C	000	HALT execution. *Example:* C000 would cause program execution to stop.

Variable-Length Instructions

- To simplify explanations, the machine language described in Appendix C uses a fixed size (two bytes) for all instructions.
- Thus, to fetch an instruction, the CPU always retrieves the contents of two consecutive memory cells and increments its **program counter** by two.
- This consistency streamlines the task of fetching instructions and is characteristic of RISC machines.
- CISC machines, however, have machine languages whose instructions vary in length.

Variable-Length Instructions

- The **Pentium series**, for example, has instructions that range from single-byte instructions to multiple-byte instructions whose length depends on the exact use of the instruction.
- CPUs with such machine languages determine the length of the incoming instruction by the instruction's op-code.
- That is, the CPU first fetches the op-code of the instruction and then, based on the bit pattern received, knows how many more bytes to fetch from memory to obtain the rest of the instruction.

Data Manipulation

- 2.1 Computer Architecture
- 2.2 Machine Language
- 2.3 Program Execution
- 2.4 Arithmetic/Logic Instructions
- 2.5 Communicating with Other Devices
- 2.6 Other Architectures

Program Execution

- A computer follows a program stored in its memory by copying the instructions from memory into the control unit as needed.
- Once in the control unit, each instruction is decoded and executed.
- The order in which the instructions are fetched from memory corresponds to the order in which the instructions are stored in memory unless otherwise altered by a JUMP instruction.

Program Execution

- Controlled by two special-purpose registers
  - **Program counter**: address of next instruction
  - **Instruction register**: current instruction
- Machine Cycle
  - **Fetch**
  - **Decode**
  - **Execute**

Program counter and Instruction Register

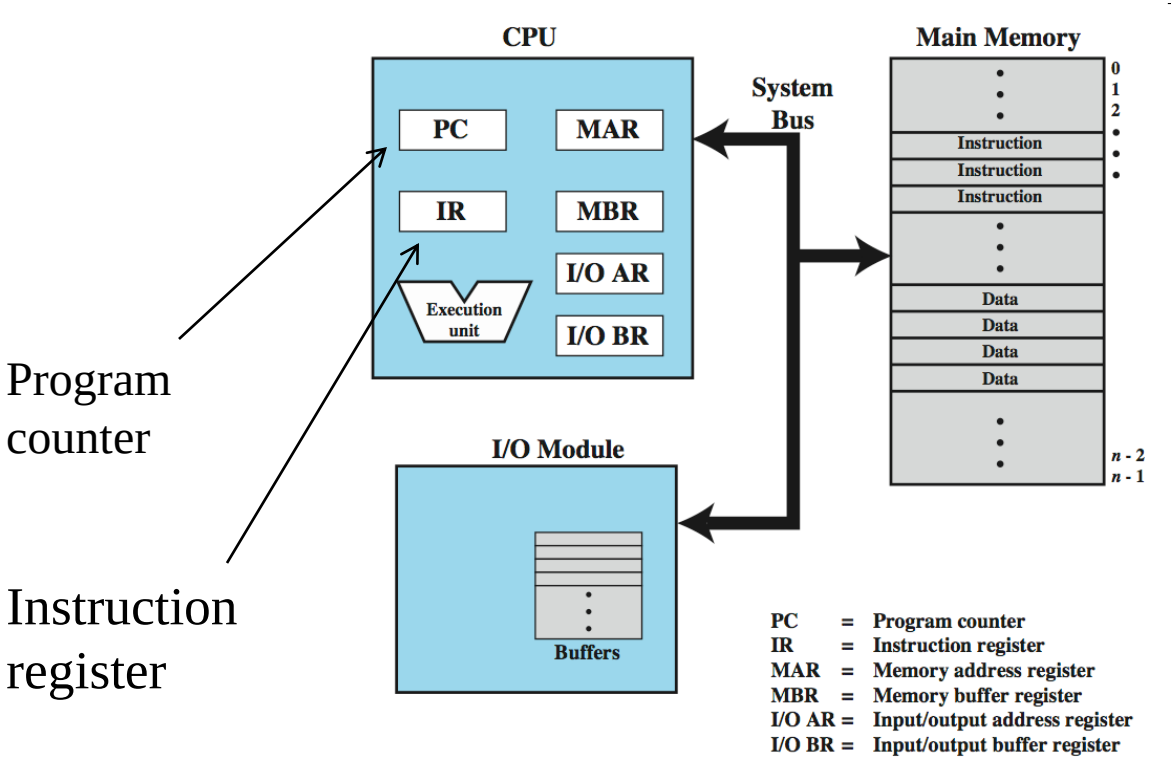
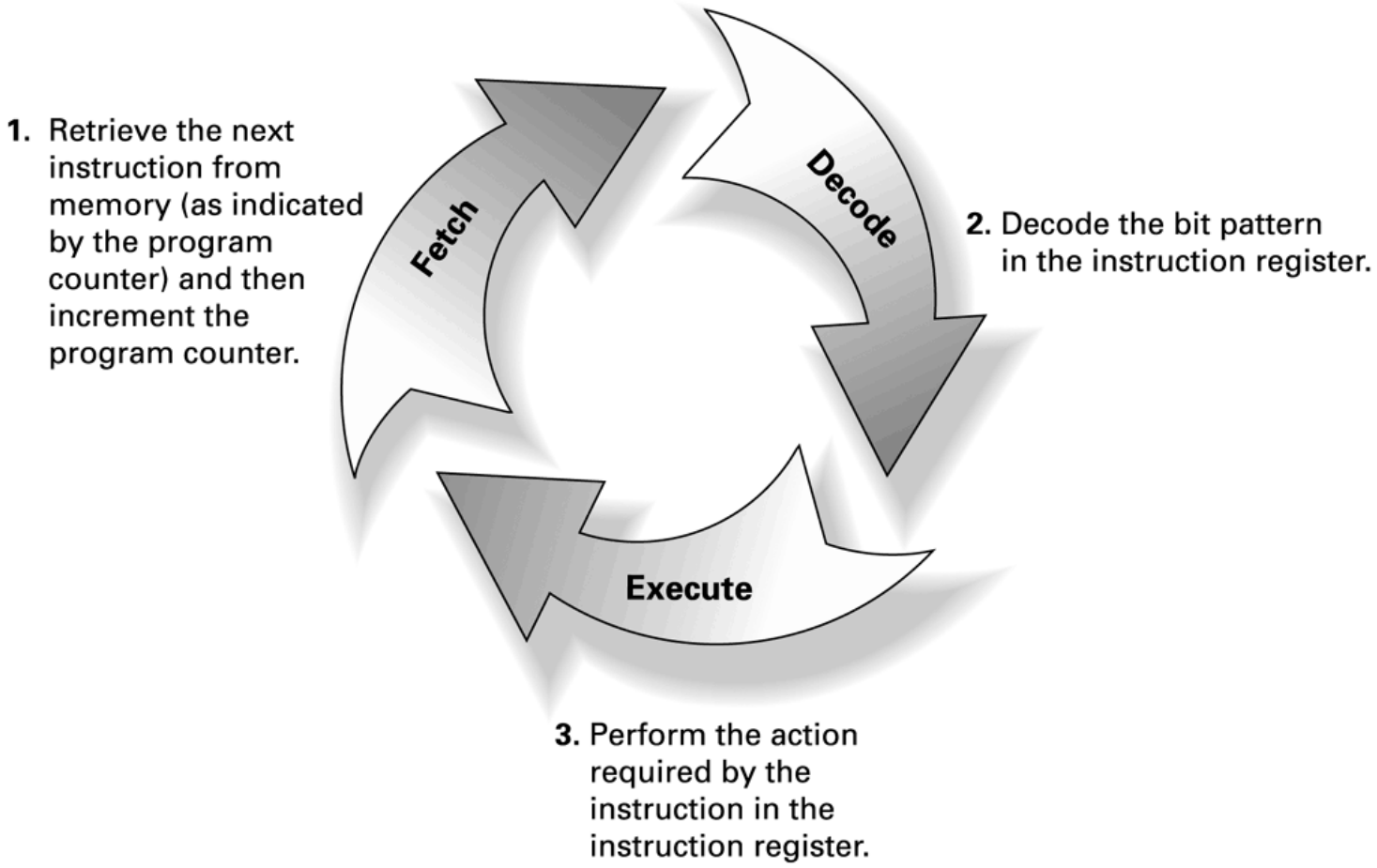


Figure 1.1 Computer Components: Top-Level View

Figure 2.8 The machine cycle



Control unit in the program execution: Fetch

- The control unit performs its job by continually repeating an algorithm that guides it through a three-step process known as the machine cycle.

- During the fetch step, the control unit requests that main memory provide it with the instruction that is stored at the address indicated by the program counter.
  - Since each instruction in our machine is two bytes long, this fetch process involves retrieving the contents of two memory cells from main memory.
  - The control unit places the instruction received from memory in its instruction register and then increments the program counter by two so that the counter contains the address of the next instruction stored in memory.
  - Thus the program counter will be ready for the next fetch.

Control unit in the program execution: Decode

- With the instruction now in the instruction register, the **control unit decodes the instruction**, which involves breaking the operand field into its proper components based on the instruction's op-code.

Control unit in the program execution: Execute

- The control unit then **executes** the instruction by activating the appropriate circuitry to perform the requested task.

- For example, if the instruction is a **load from memory**, the control unit sends the appropriate signals to main memory, waits for main memory to send the data, and then places the data in the requested register;

- If the instruction is for an arithmetic operation, the **control unit activates** the appropriate circuitry in the **arithmetic/logic unit** with the correct registers as inputs and waits for the arithmetic/logic unit to compute the answer and place it in the appropriate register.

Figure 2.9 Decoding the instruction B258

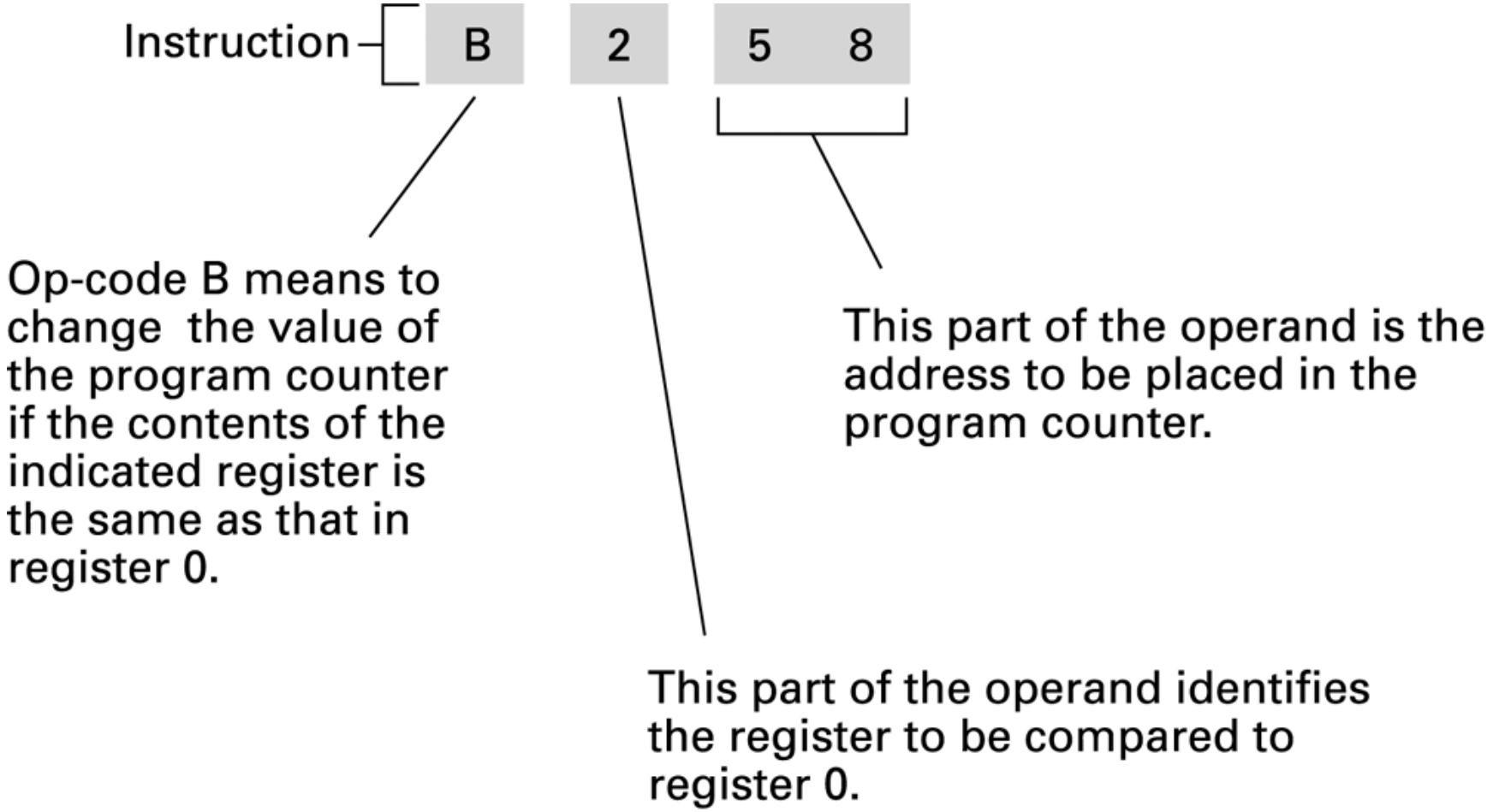


Illustration of machine cycles: Program to execute

Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution

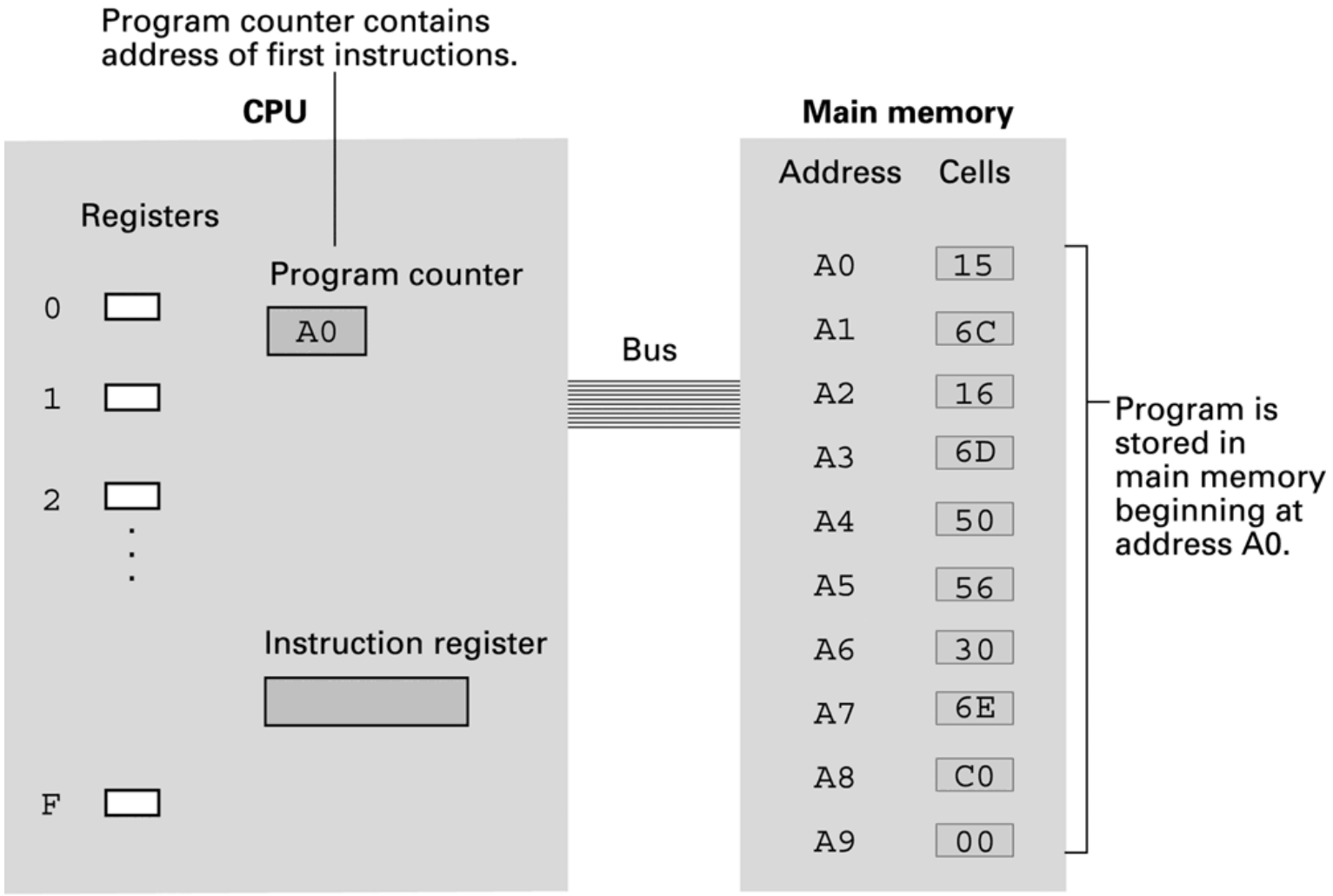
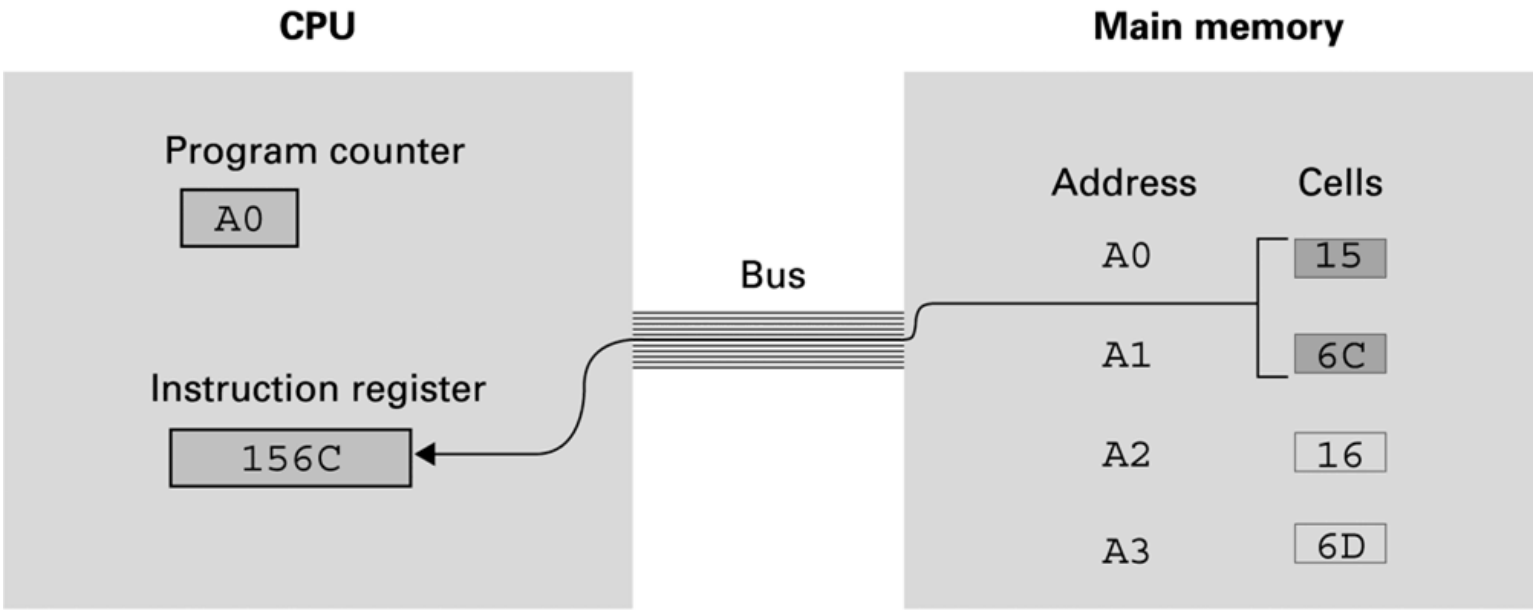
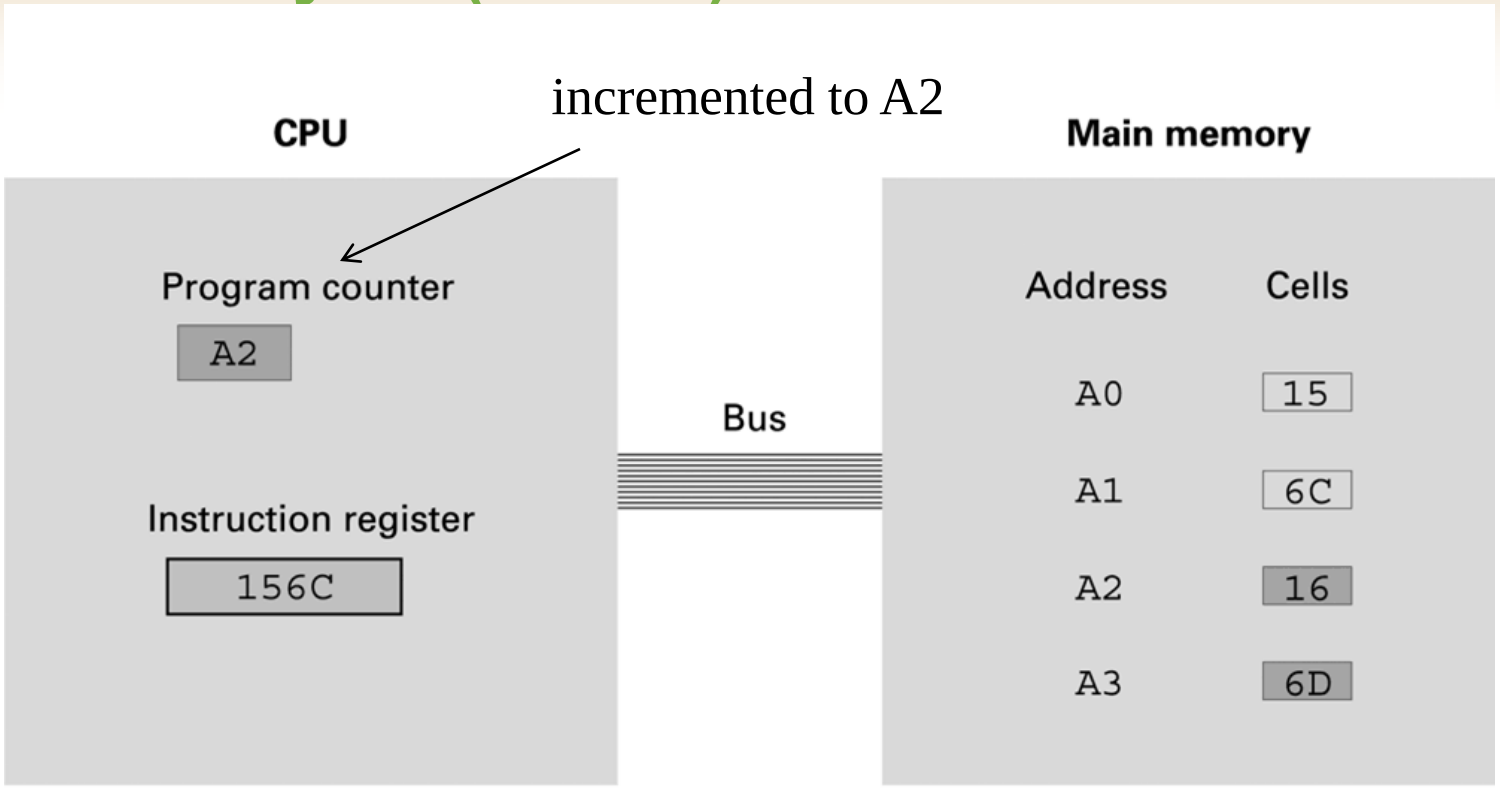


Figure 2.11 Performing the fetch step of the machine cycle



a. At the beginning of the fetch step the instruction starting at address A0 is retrieved from memory and placed in the instruction register.

Figure 2.11 Performing the fetch step of the machine cycle (cont'd)



b. Then the program counter is incremented so that it points to the next instruction.

- Next, the control unit analyzes the instruction in its instruction register and concludes that it is to load register 5 with the contents of the memory cell at address 6C**.** This load activity is performed during the execution step of the machine cycle, and the control unit then begins the next cycle.

Next steps

- Program Counter: A4
- Instruction Register: 166D

- The CPU decodes the instruction 166D and determines that it is to load register 6 with the contents of memory address 6D. It then executes the instruction.
- It is at this time that register 6 is actually loaded.

Next steps

- Since the program counter now contains A4, the CPU extracts the next instruction starting at this address.
- The result is that 5056 is placed in the instruction register, and the program counter is incremented to A6.
- The CPU now decodes the contents of its instruction register and executes it by activating the two's complement addition circuitry with inputs being registers 5 and 6.

Addition step

- During this execution step, the arithmetic/logic unit performs the requested addition, leaves the result in register 0 (as requested by the control unit), and reports to the control unit that it has finished.
- The CPU then begins another machine cycle.

Move result into memory location

- Once again, with the aid of the program counter, it fetches the next instruction (306E) from the two memory cells starting at memory location A6 and increments the program counter to A8.

- This instruction is then decoded and executed. At this point, the sum is placed in memory location 6E.

Last step: stop

- The next instruction is fetched starting from memory location A8, and the program counter is incremented to AA.
- The contents of the instruction register (C000) are now decoded as the halt instruction.
- Consequently, the machine stops during the execute step of the machine cycle, and the program is completed.

Programs versus data

- Many programs can be stored simultaneously in a computer's main memory, as long as they occupy different locations.
- Which program will be run when the machine is started can then be determined merely by setting the **program counter** appropriately.
- One must keep in mind, however, that because data are also contained in main memory and encoded in terms of 0s and 1s, the machine alone **has no way of knowing what is data and what is program.**
- If the program counter were assigned the address of data instead of the address of the desired program, the computer, not knowing any better, would extract the data bit patterns as though they were instructions and execute them.
  - The final result would depend on the data involved!!!

Data Manipulation

- 2.1 Computer Architecture
- 2.2 Machine Language
- 2.3 Program Execution
- 2.4 Arithmetic/Logic Instructions
- 2.5 Communicating with Other Devices
- 2.6 Other Architectures

Arithmetic/Logic Operations

- Logic: AND, OR, XOR
  - Masking
- Rotate and Shift: circular shift, logical shift, arithmetic shift
- Arithmetic: add, subtract, multiply, divide
  - Precise action depends on how the values are encoded (two's complement versus floating-point).

Logic Operations

```
  10011010
AND 11001001
  10001000
```

```
  10011010
OR 11001001
  11011011
```

```
  10011010
XOR 11001001
  01010011
```

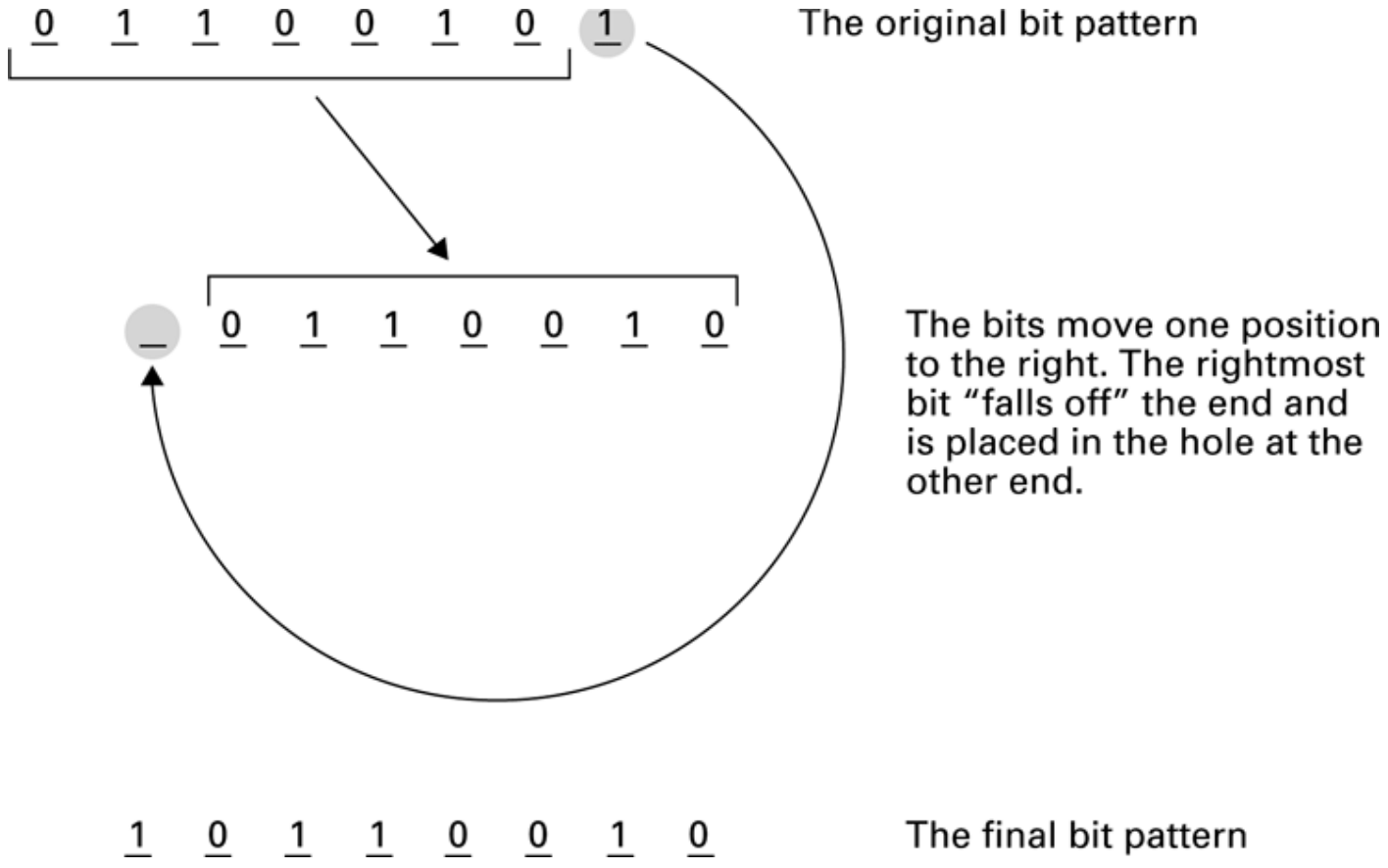
```
  00001111
AND 10101010
  00001010
```

Complementing bit strings with XOR

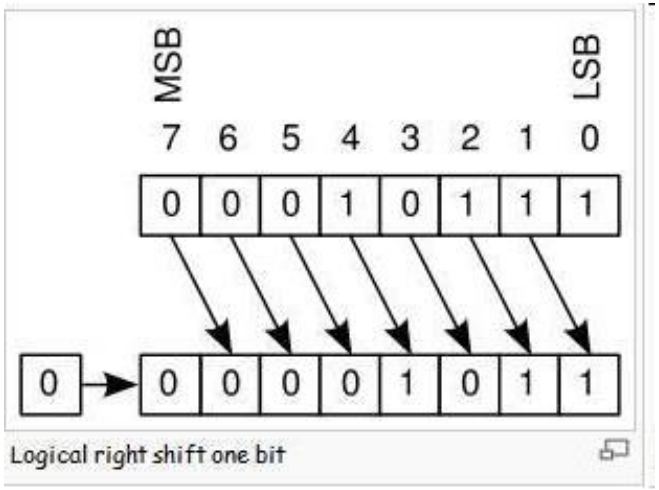
- A major use of the XOR operation is in forming the complement of a bit string.
- XORing any byte with a mask of all 1s produces the complement of the byte.
- For example, note the relationship between the second operand and the result in the following example:

```
    11111111
XOR 10101010
    01010101
```

Figure 2.12 Rotating the bit pattern 65 (hexadecimal) one bit to the right



Logical Shift



Logical shift

- Shifts to the left can be used for multiplying two's complement representations by two.
- After all, **shifting binary digits to the left corresponds to multiplication by two**, just as a similar shift of decimal digits corresponds to multiplication by ten.
- Moreover, division by two can be accomplished by shifting the binary string to the right.
- In either shift, care must be taken to preserve the sign bit when using certain notational systems.
- Shifts that leave the sign bit unchanged are sometimes called **arithmetic shifts**.

Arithmetic operations

- Subtraction can be simulated by means of addition and negation.
- Moreover, multiplication is merely repeated addition and division is repeated subtraction.
- For this reason, some small CPUs are designed with only the add or perhaps only the add and subtract instructions.

Next lesson

- 2.1 Computer Architecture
- 2.2 Machine Language
- 2.3 Program Execution
- 2.4 Arithmetic/Logic Instructions
- 2.5 Communicating with Other Devices
- 2.6 Other Architectures

End of class

- Readings
  - Book: Chapter 2