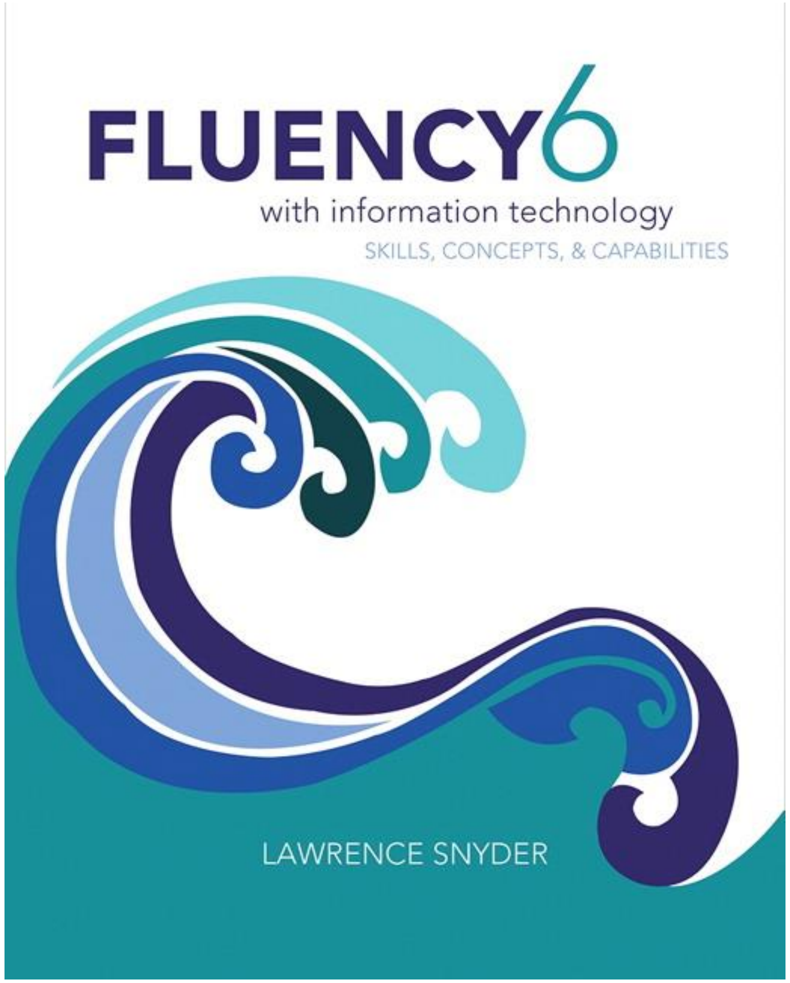


Chapter 9

Principles of Computer Operations

Table of Contents

Learning Objectives

- Explain what a software stack represents and how it is used
- Describe how the Fetch/Execute Cycle works, listing the 5 steps
- Understand the computer architecture: the function of memory, control unit, arithmetic/logic unit (ALU), input unit and output unit, and a program counter
- Discuss the purpose of an operating system
- Explain the purpose of a compiler
- Describe how large tasks are performed with simple instructions
- Explain why integration and photolithography are important in integrated circuits

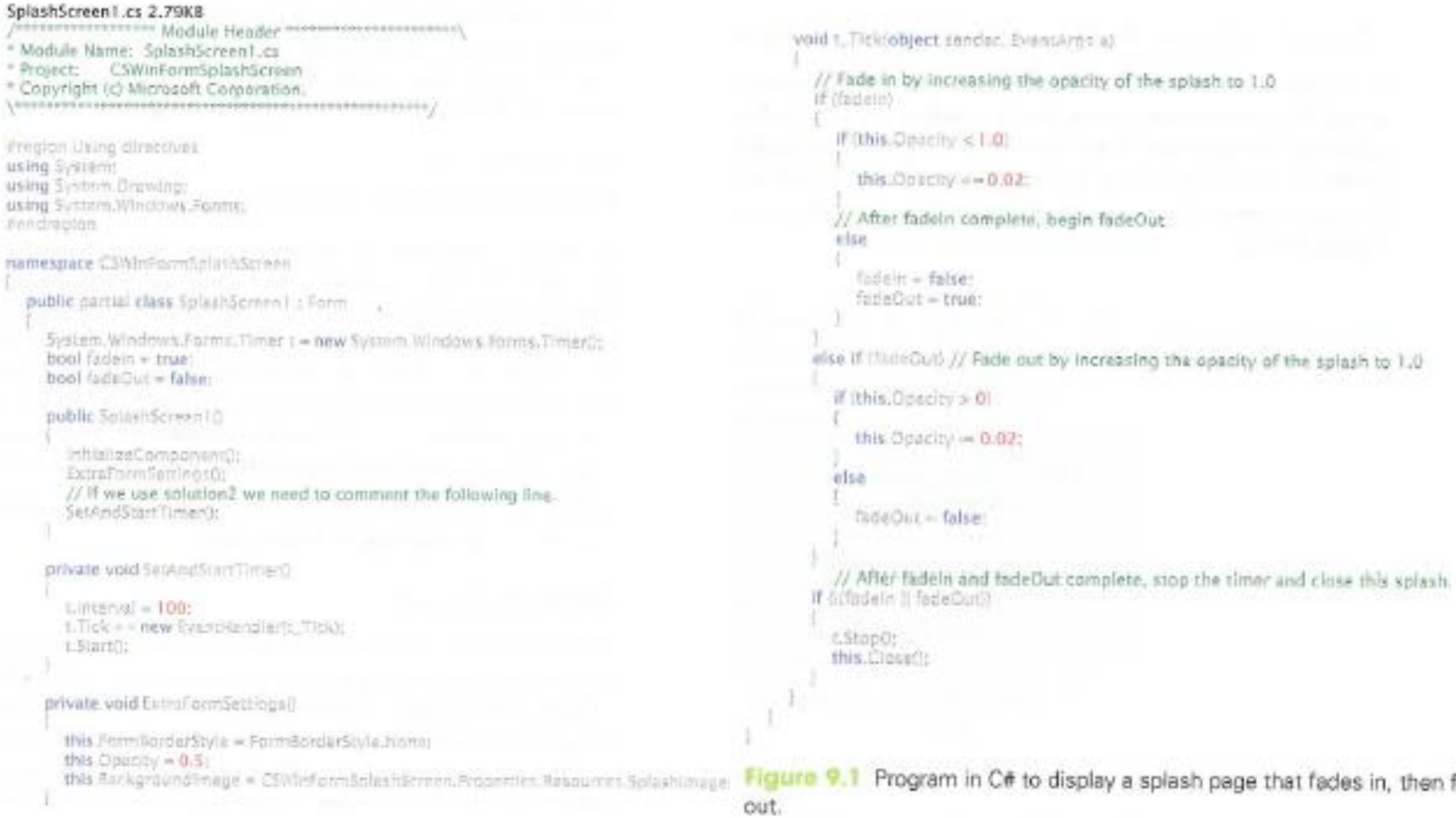
Computer Overview

- Computers are used throughout your day
  - We are continually using computation
  - They are laptops, cameras, tablets, iPods, desktops, GPS navigators, TV's and all the other electronic devices that we use
- Components of The Computer System
  - Processor: follows the program's instructions
  - Operating System: program that performs common operations, and makes your computer a useful device
  - Software: programs
  - Instructions: tell the processor what to do
  - Fetch/Execute Cycle: executes the instructions
  - Memory: stores the data
  - Hardware: the physical parts of the computer

Software

- When you get a new app it is really a long sequence of bits (millions …. !)
  - A team of programmers created the bits but did not type them one by one
- Software characteristics (C#: a typical app programming language)
  - Each line only has a few symbols on it
  - The English words are blue: public, if, this, true, false etc
  - Programmer’s chosen words run together, like SetAndStartTimer()
- Make one small typo, and the program will have a bug
- Software Layers
  - Many softwares are available in the operating system: ex. timer
  - Sharing libraries of SWs that has already been developed has lots of advantages
    - Reusing code can reduce effort and leverage other programmer’s knowledge
    - Updating a single instance is easier than changing it in lots of places

C# Sample Program



SplashScreen1.cs 2.79KB

```
/******************** Module Header ************************\
* Module Name:  SplashScreen1.cs
* Project:      CSWinFormSplashScreen
* Copyright (c) Microsoft Corporation.
\**********************************************************/

#region Using directives
using System;
using System.Drawing;
using System.Windows.Forms;
#endregion

namespace CSWinFormSplashScreen
{
   public partial class SplashScreen1 : Form
   {
      System.Windows.Forms.Timer t = new System.Windows.Forms.Timer();
      bool fadein = true;
      bool fadeOut = false;

      public SplashScreen1()
      {
         InitializeComponent();
         ExtraFormSettings();
         // If we use solution2 we need to comment the following line.
         SetAndStartTimer();
      }

      private void SetAndStartTimer()
      {
         t.Interval = 100;
         t.Tick += new EventHandler(t_Tick);
         t.Start();
      }

      private void ExtraFormSettings()
      {
         this.FormBorderStyle = FormBorderStyle.None;
         this.Opacity = 0.5;
         this.BackgroundImage = CSWinFormSplashScreen.Properties.Resources.SplashImage;
      }

      void t_Tick(object sender, EventArgs e)
      {
         // Fade in by increasing the opacity of the splash to 1.0
         if (fadein)
         {
            if (this.Opacity < 1.0)
            {
               this.Opacity += 0.02;
            }
            // After fadeIn complete, begin fadeOut
            else
            {
               fadein = false;
               fadeOut = true;
            }
         }
         else if (fadeOut) // Fade out by increasing the opacity of the splash to 1.0
         {
            if (this.Opacity > 0)
            {
               this.Opacity -= 0.02;
            }
            else
            {
               fadeOut = false;
            }
         }

            // After fadeIn and fadeOut complete, stop the timer and close this splash.
         if (!(fadein || fadeOut))
         {
            t.Stop();
            this.Close();
         }
      }
   }
}
```

Figure 9.1 Program in C# to display a splash page that fades in, then fades out.

Software Layers

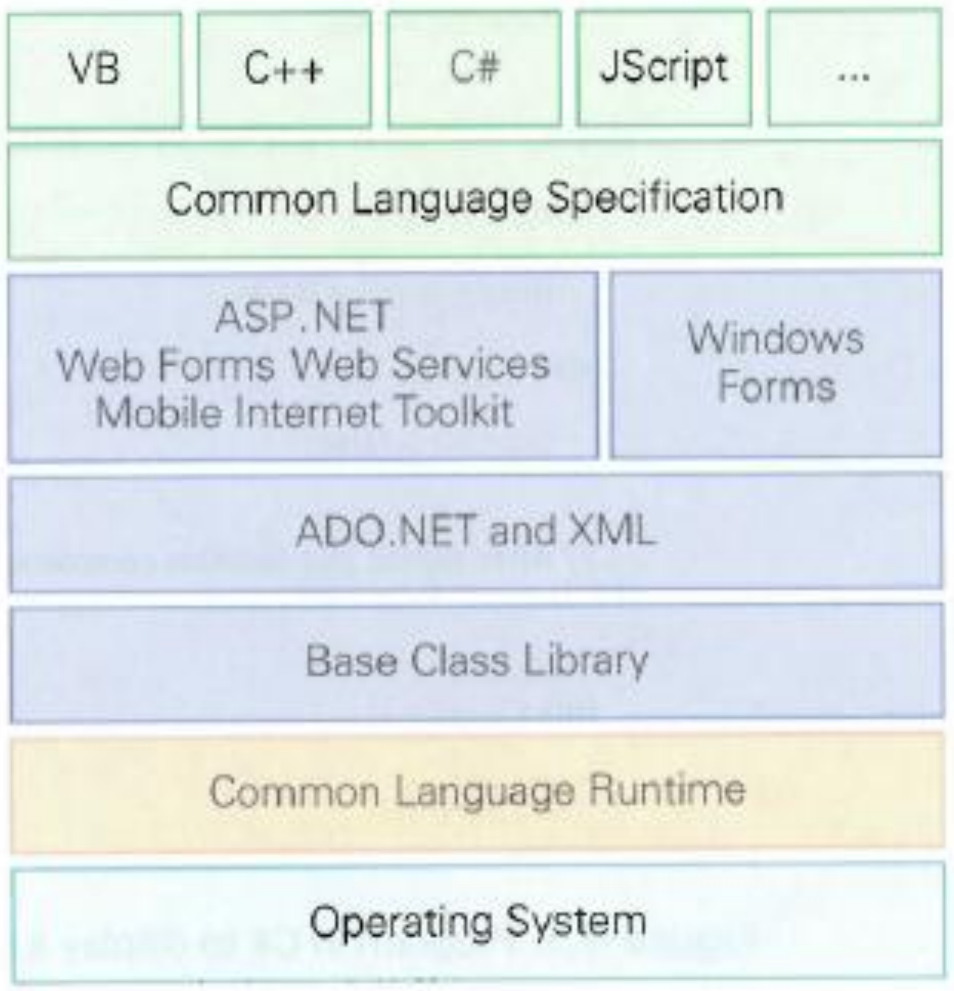


Figure 9.2 Software stack for Windows.NET.

Instruction Execution Engine

- The Fetch/Execute Cycle
  - Get the next instruction
  - Figure out what to do
  - Gathering the data needed to do it
  - Do it
  - Save the result

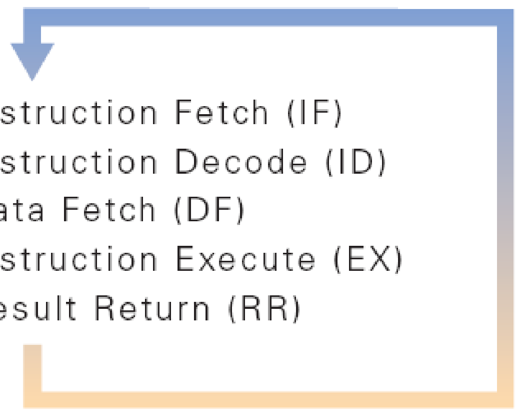


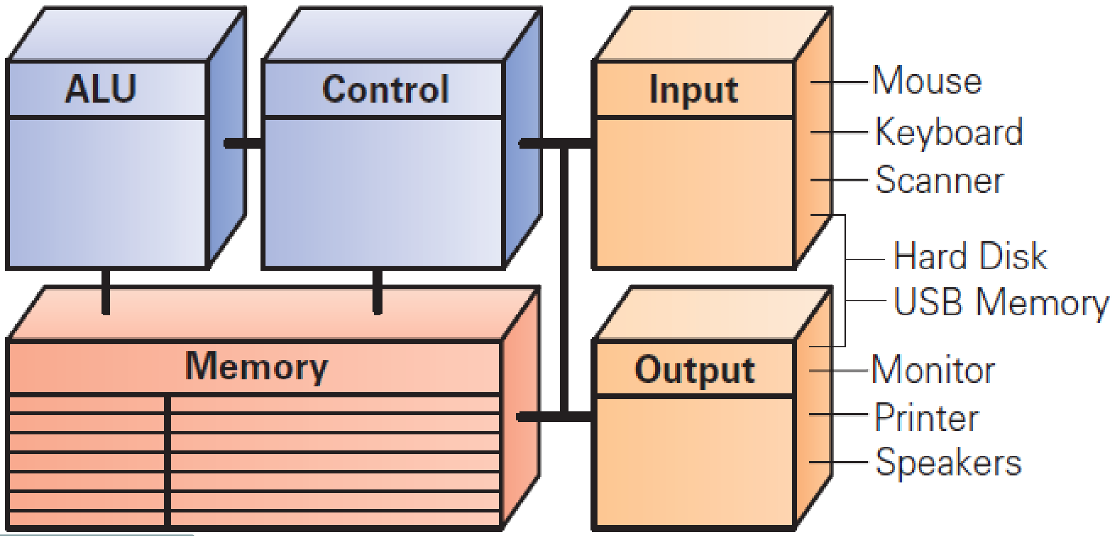
Figure 9.1 The Fetch/Execute Cycle.

  - These operations are repeated in a never-ending sequence (billions / sec)!

- The step names suggest the operations described in the Figure 9.1

Anatomy of a Computer

- All computers, regardless of their implementing technology, have five basic parts or subsystems:
  - Memory
  - Control unit
  - Arithmetic/logic unit (ALU)
  - Input unit
  - Output unit



The principal subsystems of a computer.

Closer Look of Computer Inside

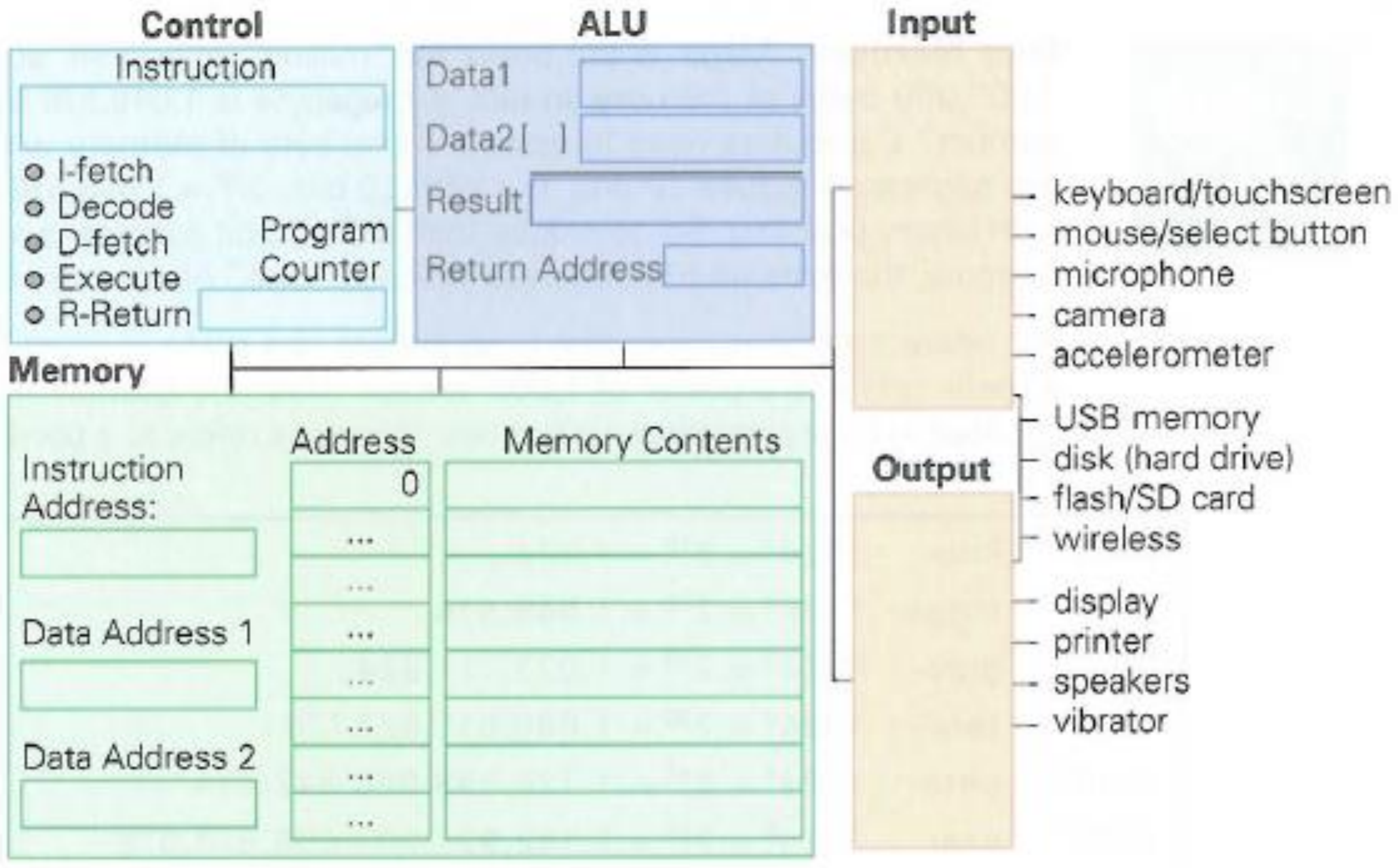


Figure 9.4 The five principal subsystems of a computer—the control unit, memory, ALU, input unit, and output unit—with typical input and output devices shown.

1. Memory: Structure

- Memory stores both the program while it is running and the data on which the program operates
- Properties of memory: Discrete locations
  - Memory is organized as a sequence of discrete locations
  - In modern memory, each location is composed of 1 byte (8 bits)
- Addresses: Every memory location has an address starting at 0
- Values: Memory locations record or store values
- Finite capacity
  - Memory locations have a finite capacity (limited size),
  - Program and Data may not "fit" in the memory location

1. Memory: Byte-Size Memory Location

- Common visualization of computer memory
- Discrete locations are shown as boxes holding 1-byte each

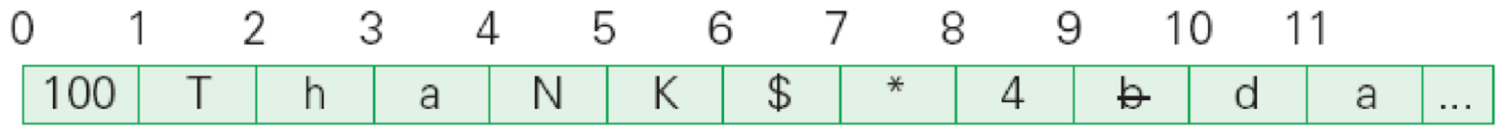


Figure 9.3 Diagram of computer memory illustrating its key properties.

- Address of location is displayed above the box and the contents of location is shown in the box
- That 1-byte memory location can store one ASCII character or a number less than 256 (= 2^8)
- Blocks of four bytes are used as a unit so often that they are called memory words

1. Memory: Random Access Memory

- Computer memory is called random access memory (RAM)
  - The computer can refer to the memory locations in any order
  - "Main memory" is used interchangeably
- RAM is measured in megabytes (MB) or gigabytes (GB)
- Lots of memory is need to handle the space required of programs and data
- RAM is volatile

- Hard disk does not have random access property! (cheaper than RAM)
- Flash Memory: Non-volatile Memory with random access property
  - USB memory (= USB flash drive)
  - SD card (secure disk)
  - SSD (solid state disk): 움직이는 부품이 없는 고정된 상태의 디스크

2. Control Unit

- The control unit of a computer is where the Fetch/Execute Cycle occurs

- A typical machine instruction (Assembly language) has the form:

ADD 4000, 2000, 2080

  - Looks like those three numbers should be added together!
  - But what it really means is that whatever numbers are stored in memory locations 2000 and 2080 be added together, and the result be stored in location 4000

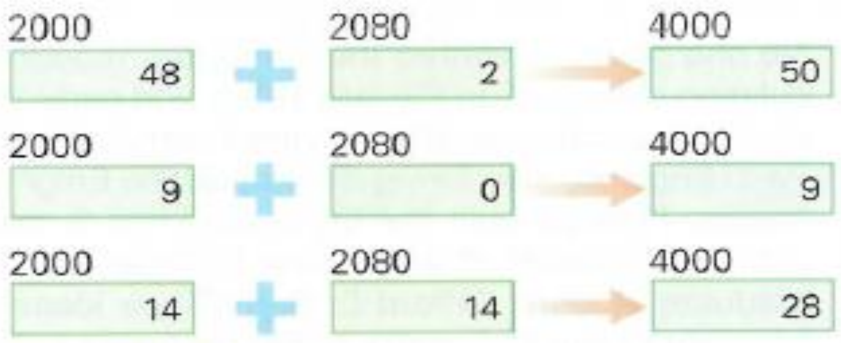


Figure 9.7 Illustration of a single ADD instruction producing different results depending on the contents of the memory locations referenced in the instruction.

3. Arithmetic/Logic Unit (ALU)

- “Does the basic math operations” during the Fetch/Execute cycle
- ALU has circuits for adding, multiplying, for comparing two numbers, etc.
- The math circuits uses logic gates or simpler circuits that implement operations like AND and OR
- The ALU carries out each machine instruction with a separate circuit

4. & 5. Input and Output Units

- These two components are the wires and circuits through which information moves into and out of a computer
- A computer without input or output is useless
- The Input unit has input port & the Output unit is output port

The Peripherals

- Peripherals are not considered part of the computer
  - Keyboard, monitor, mouse, printer, etc….
- Specialized gadgets that encode or decode information between the computer and the physical world (including human)
  - The keyboard encodes human’s keystrokes into binary form for the computer
  - The monitor decodes information from the computer’s memory and displays it on a screen
- Peripherals connect to the computer input/output (I/O) ports
- The peripherals handle the physical part of the operation

Hard Drives & Portable Memory

- Some storage device peripherals are used by computers for both input and output:
  - USB memory (= USB flash drive)
  - Hard disks/drives
- The hard disk (HD) is the alpha-peripheral, being the most tightly linked device to the computer
- HD can be replaced with SSD (solid state disk) which is a kind of flash memory

A Device Driver for Every Peripheral

- Most peripheral devices are “dumb”
  - –They provide only basic physical translation to or from binary signals.
- Additional information from the computer is needed to make it operate “intelligently”
- Added processing by software called a device driver gives the peripheral its standard meaning and behavior
- Every device needs a device driver!

Machine Instructions

- Machine instructions are more primitive than what programmers type
  - ADD 4000, 2000, 2080
  - Commands the computer to add the numbers stored in memory locations 2000 and 2080 and then store that in the memory location 4000
  - Computer instructions encode the memory addresses, not the numbers themselves
  - Indirect reference: referring to a value by referring to the address in memory

The Program Counter: The PC's PC

- How does the computer determine which instruction it should execute next?
- Address of the Next Instruction
  - The instruction is stored in memory and the computer has its address
  - Computers use the address (known as the program counter or PC) to keep track of the next instruction
- It assumes that the next instruction is the next instruction in sequence
- Because instructions use 4 bytes of memory, the next instruction must be at the memory address PC + 4 or 4 bytes further along the sequence

Branch and Jump Instructions

- Not all instructions are in a strict sequence
- The instruction may include a memory location (address) to go to next
- This changes the PC, so instead of going to PC+4 automatically, the computer "jumps" or "branches" to the specified location to continue execution

The Fetch/Execute Cycle inside the Computer

- A five-step cycle:
  - –Instruction Fetch (IF)
  - –Instruction Decode (ID)
  - –Data Fetch (DF) / Operand Fetch (OF)
  - –Instruction Execution (EX)
  - –Result Return (RR) / Store (ST)
- ADD 800, 428, 884 : ADD the values found in memory locations 428 and 884 and store the result in location 800

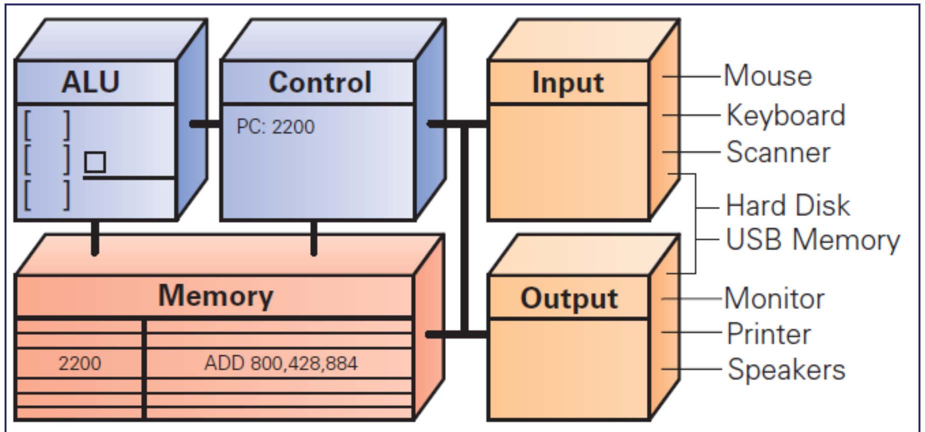


Figure 9.5 Computer before executing an ADD instruction.

Instruction Fetch (IF) Step

- Execution begins by moving the instruction at the address given by the PC (PC 2200) from memory to the control unit
- Bits of instruction are placed into the decoder circuit of the CU
- Once instruction is fetched, the PC can be readied for fetching the next instruction

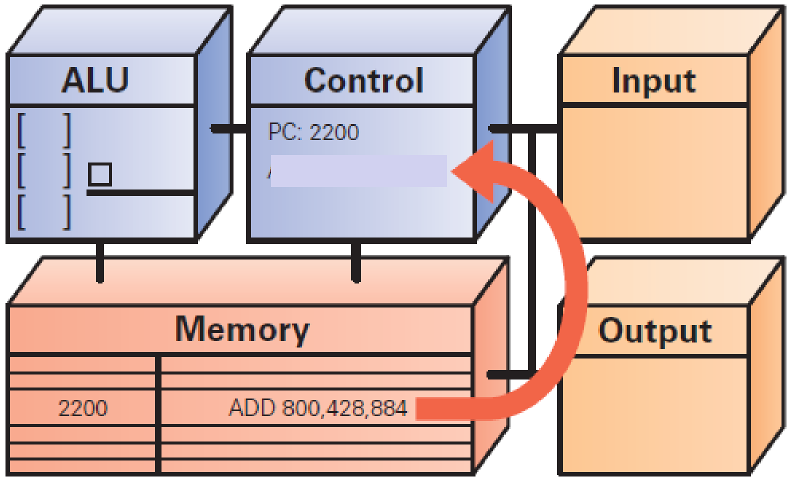


Figure 9.6 Instruction Fetch: Move instruction from memory to the control unit.

Instruction Decode (ID) Step [1/2]

- ALU is set up for the operation
- Decoder finds the memory address of the instruction's data (source operands)
  - Most instructions operate on two data values stored in memory (like ADD), so most instructions have addresses for two source operands
  - These addresses are passed to the circuit that fetches them from memory during the next step
- Decoder finds the destination address for the Result Return step and places the address in the RR circuit
- Decoder determines what operation the ALU will perform (ADD), and sets up the ALU

Instruction Decode (ID) Step [2/2]

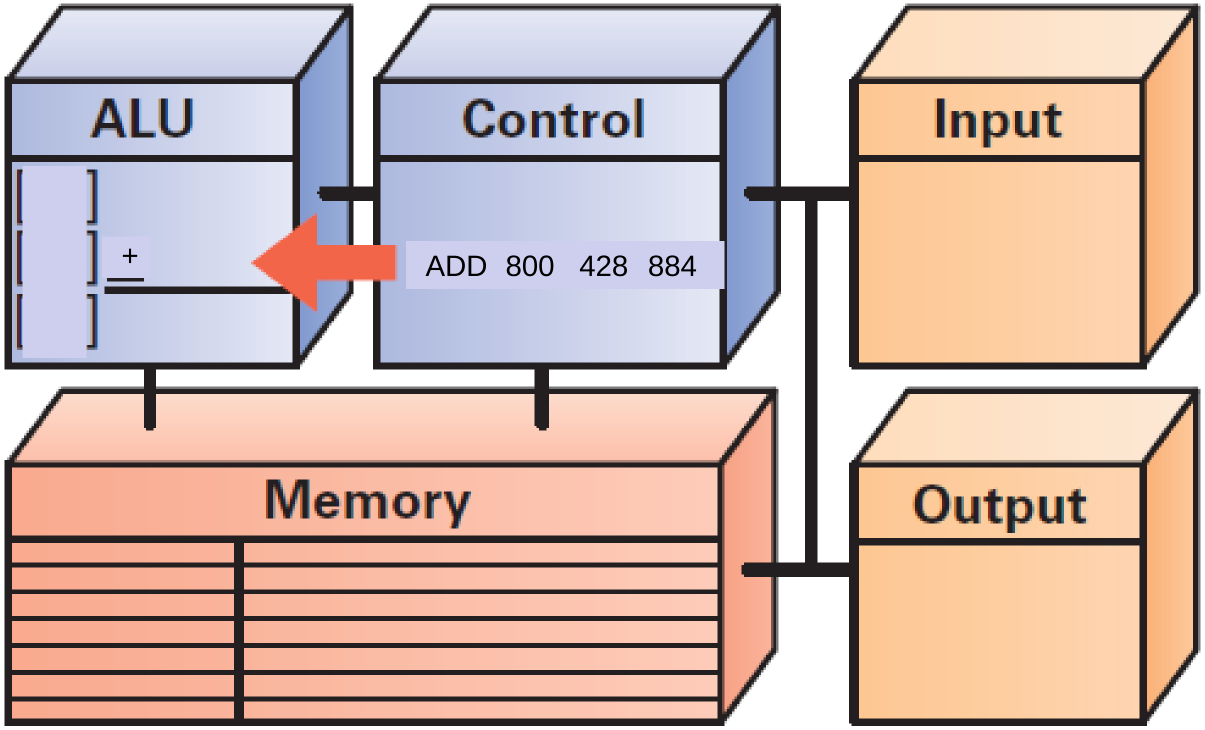


Figure 9.7 Instruction Decode: Pull apart the instruction, set up the operation in the ALU, and compute the source and destination operand addresses.

Data Fetch (DF) Step

- The data values to be operated on are retrieved from memory
- Bits at specified memory locations are copied into locations in the ALU circuitry
- Data values remain in memory (they are not destroyed)

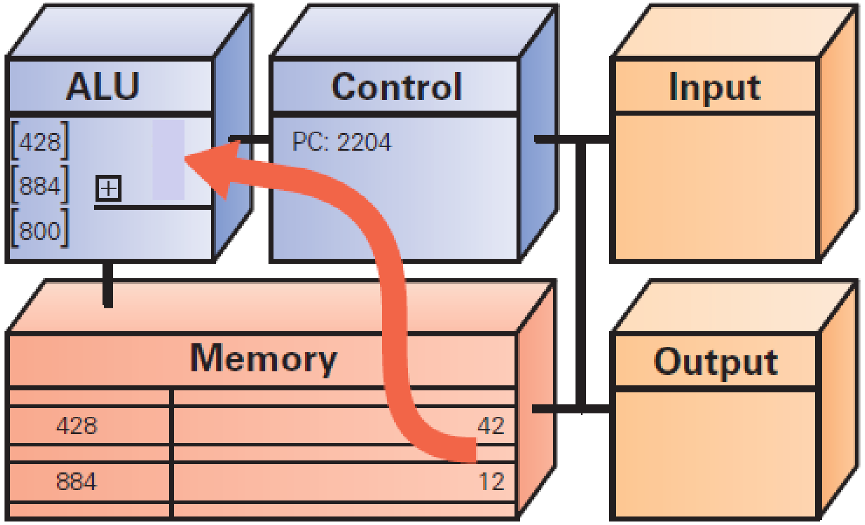


Figure 9.8 Data Fetch: Move the operands from memory to the ALU.

Instruction Execution (EX) Step

- For this ADD instruction, the addition circuit adds the two source operands together to produce their sum
- Sum is held in the ALU circuitry
- This is the actual computation

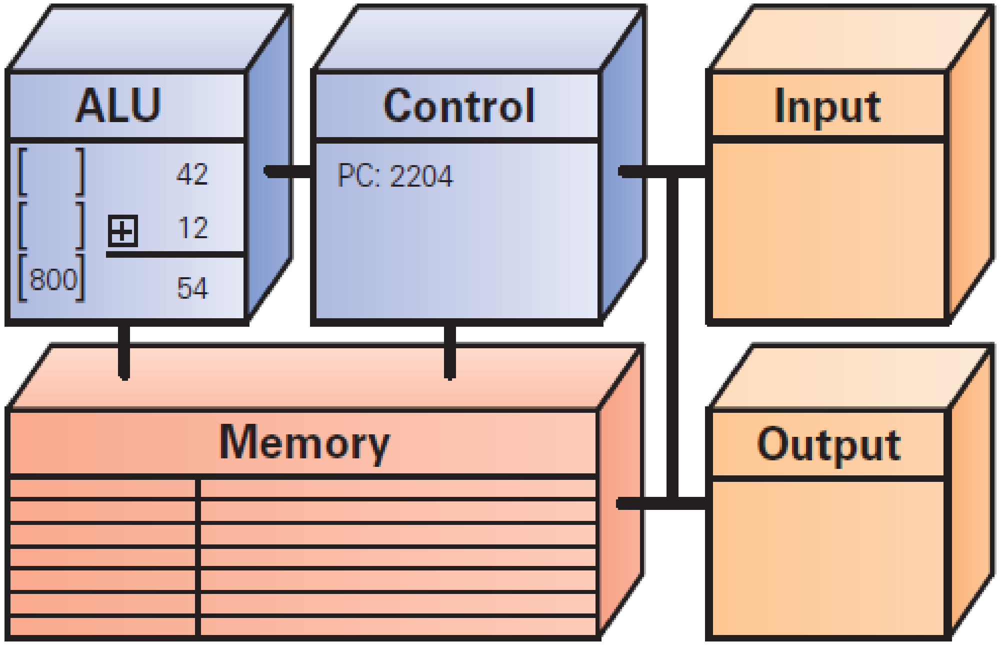


Figure 9.9 Instruction Execute: Compute the result of the operation in the ALU.

Return Result (RR) Step

- RR returns the result of EX to the memory location specified by the destination address.
- Once the result is stored, the cycle begins again

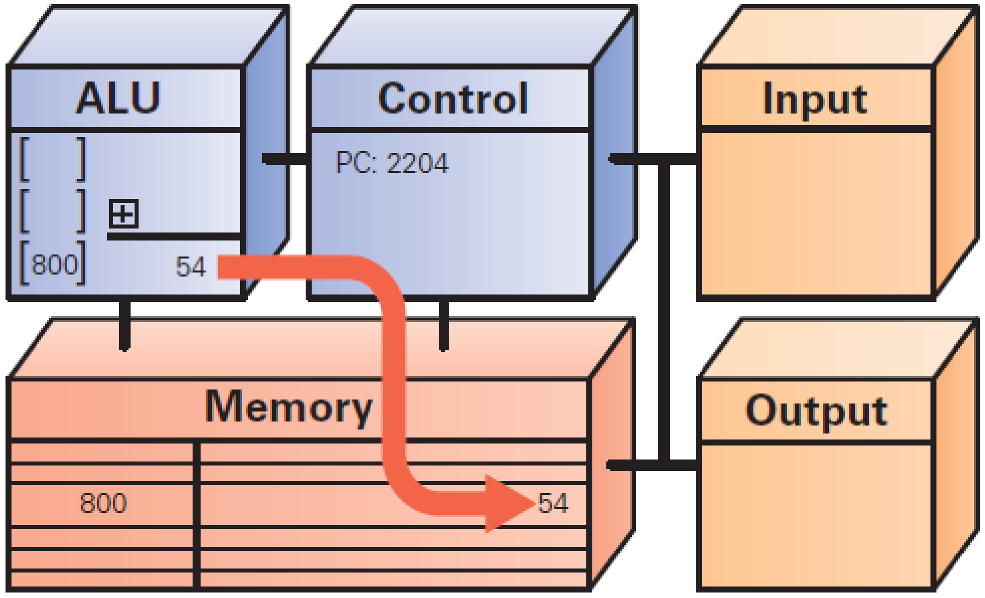


Figure 9.10 Result Return: Store the result from the ALU into the memory at the destination address.

Closer Look: ADD 4000, 2000, 2080 [1/6]

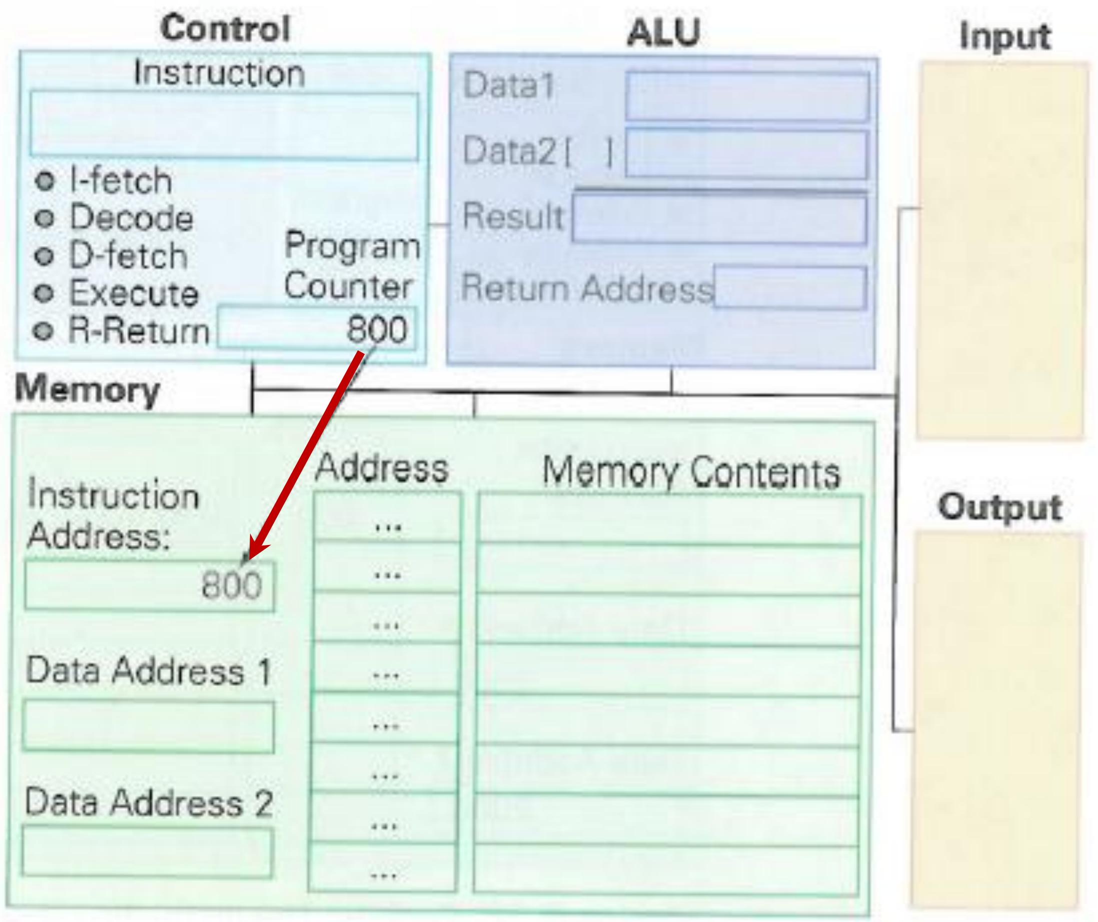


Figure 9.8 The processor before executing the instruction in memory location 800.

Closer Look: ADD 4000, 2000, 2080 [2/6]

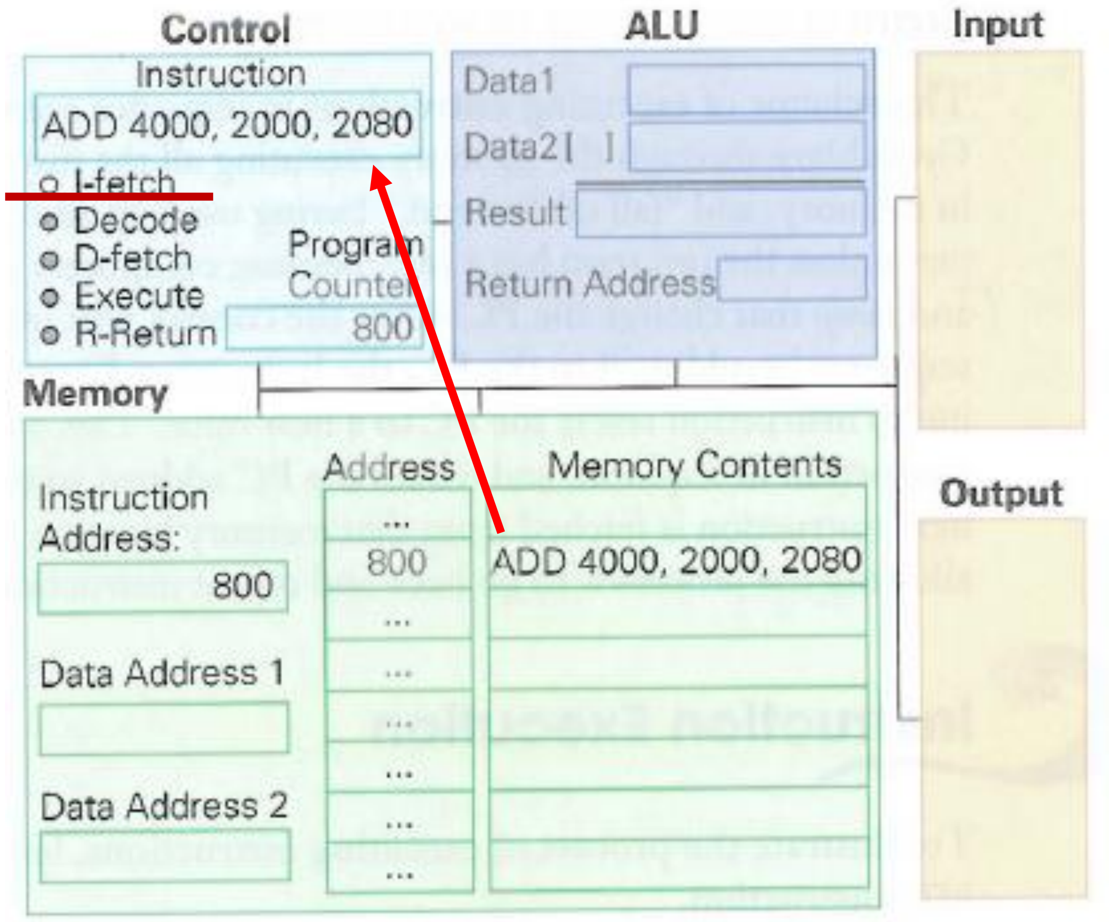


Figure 9.9 Instruction Fetch: The instruction addressed by the PC is moved from memory to the control unit.

Instruction Fetch

Closer Look: ADD 4000, 2000, 2080 [3/6]

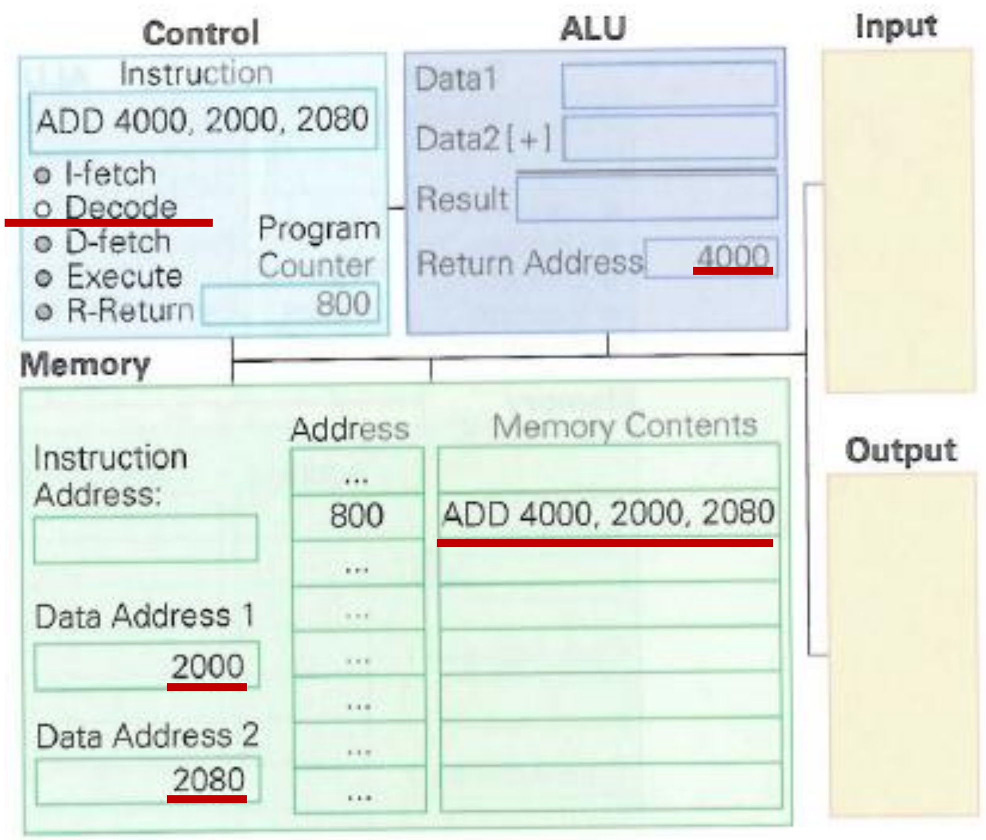


Figure 9.10 Decode: The instruction is analyzed and the processor is configured for later steps: the data addresses are sent to the Memory, the operation (+) is set in the ALU, and the result return address is set.

Instruction Decode

Closer Look: ADD 4000, 2000, 2080 [4/6]

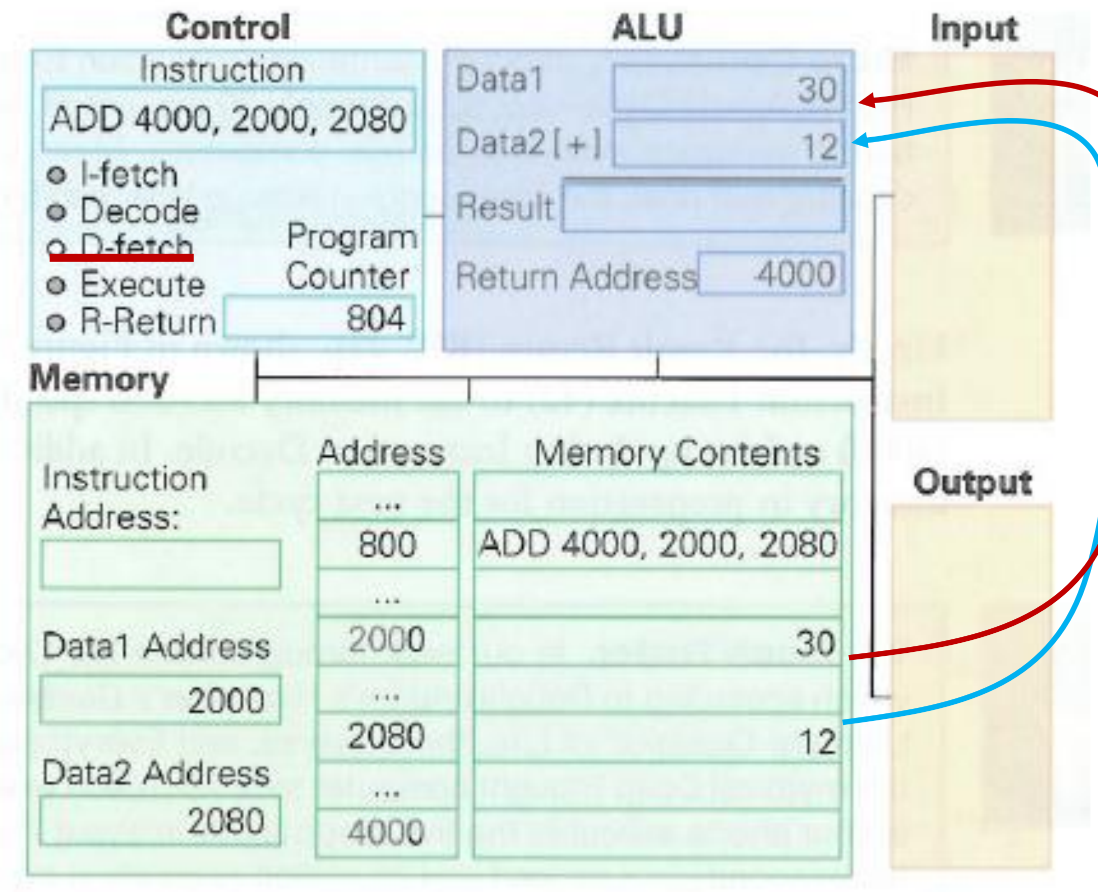


Figure 9.11 Data Fetch: The values for the two operands are fetched from memory and stored in the ALU.

Data Fetch

Closer Look: ADD 4000, 2000, 2080 [5/6]

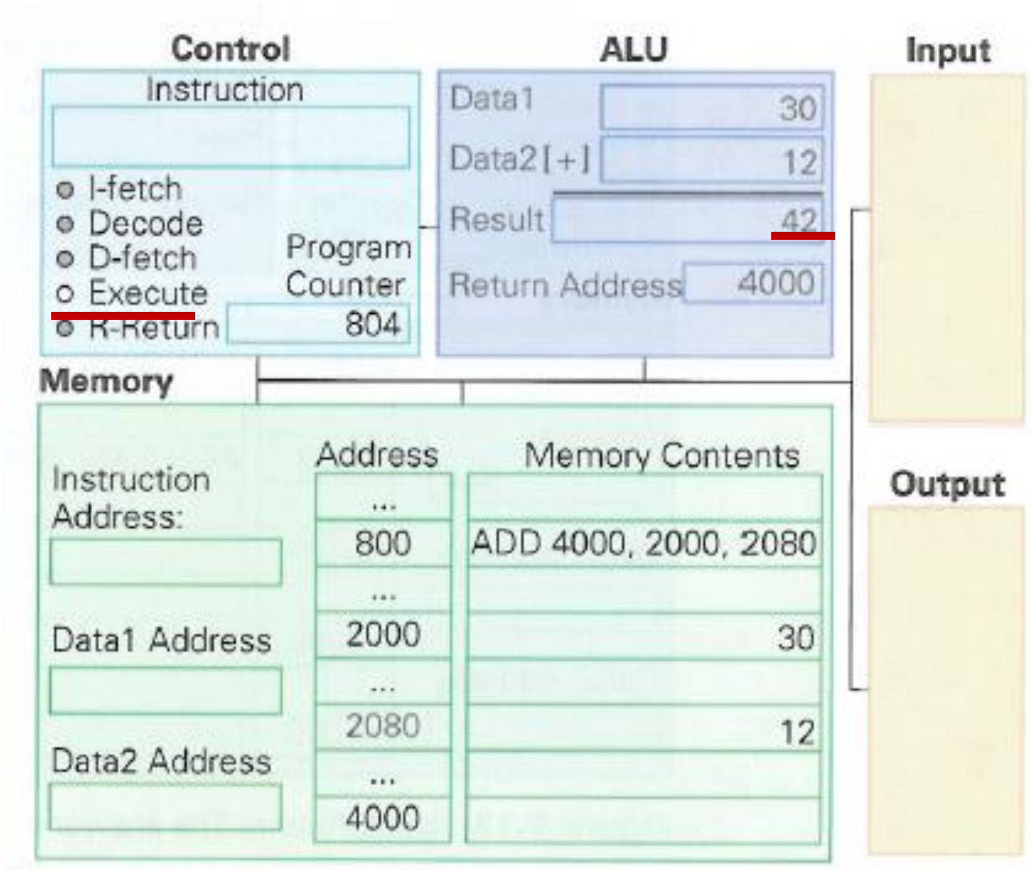


Figure 9.12 Instruction Execute: The addition operation is performed.

Instruction Execute

Closer Look: ADD 4000, 2000, 2080 [6/6]

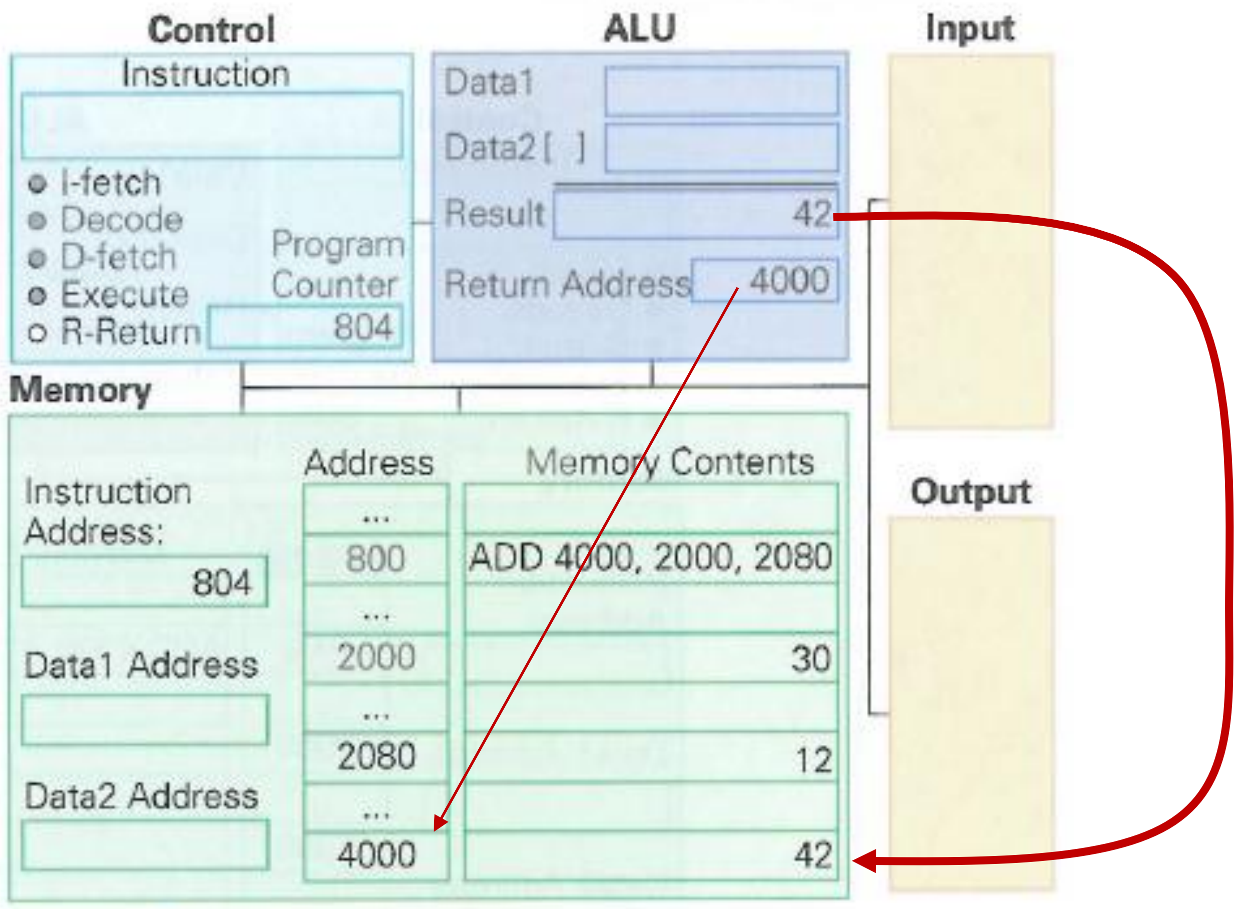


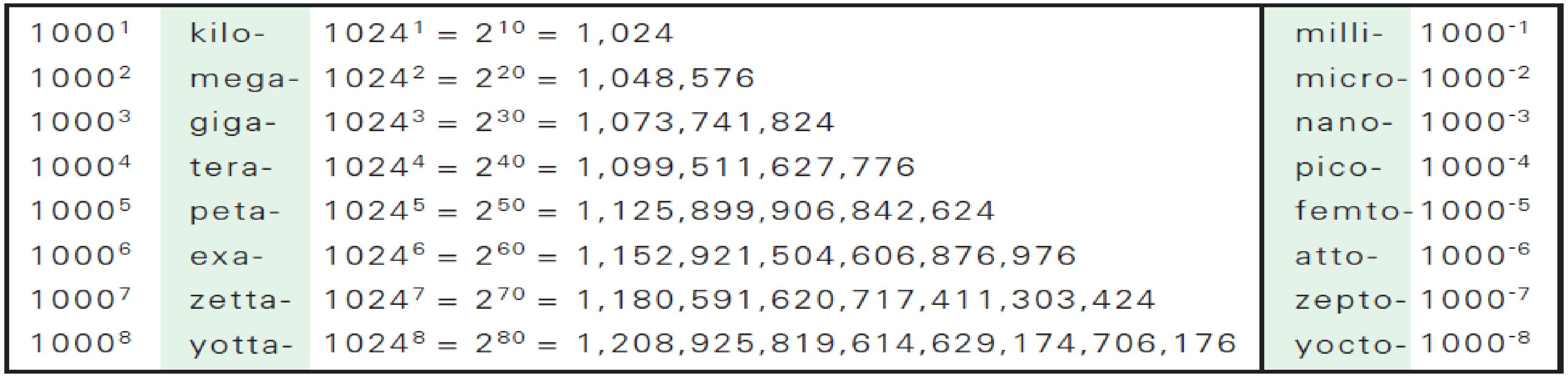
Figure 9.13 Result Return: The answer is returned to the memory, and the program counter's updated value is sent to the memory in preparation for the next fetch.

Result Return

The Computer Clock's Ticking

- Computers are instruction execution engines
- Since the computer does one instruction per cycle in principle, the speed of a computer depends on the number of Fetch/Execute Cycles it completes per second.
- The rate of the Fetch/Execute Cycle is determined by the computer's clock, and it is measured in megahertz, or millions (mega) of cycles per second (hertz).
- A 1,000 MHz clock ticks a billion (in American English) times per second, which is one gigahertz (1 GHz)

1 clock tick = 1 step of F/E cycle이 걸리는 시간, But…



1000^1	kilo-	$1024^1 = 2^{10} = 1,024$	milli-	1000^{-1}
1000^2	mega-	$1024^2 = 2^{20} = 1,048,576$	micro-	1000^{-2}
1000^3	giga-	$1024^3 = 2^{30} = 1,073,741,824$	nano-	1000^{-3}
1000^4	tera-	$1024^4 = 2^{40} = 1,099,511,627,776$	pico-	1000^{-4}
1000^5	peta-	$1024^5 = 2^{50} = 1,125,899,906,842,624$	femto-	1000^{-5}
1000^6	exa-	$1024^6 = 2^{60} = 1,152,921,504,606,876,976$	atto-	1000^{-6}
1000^7	zetta-	$1024^7 = 2^{70} = 1,180,591,620,717,411,303,424$	zepto-	1000^{-7}
1000^8	yotta-	$1024^8 = 2^{80} = 1,208,925,819,614,629,174,706,176$	yocto-	1000^{-8}

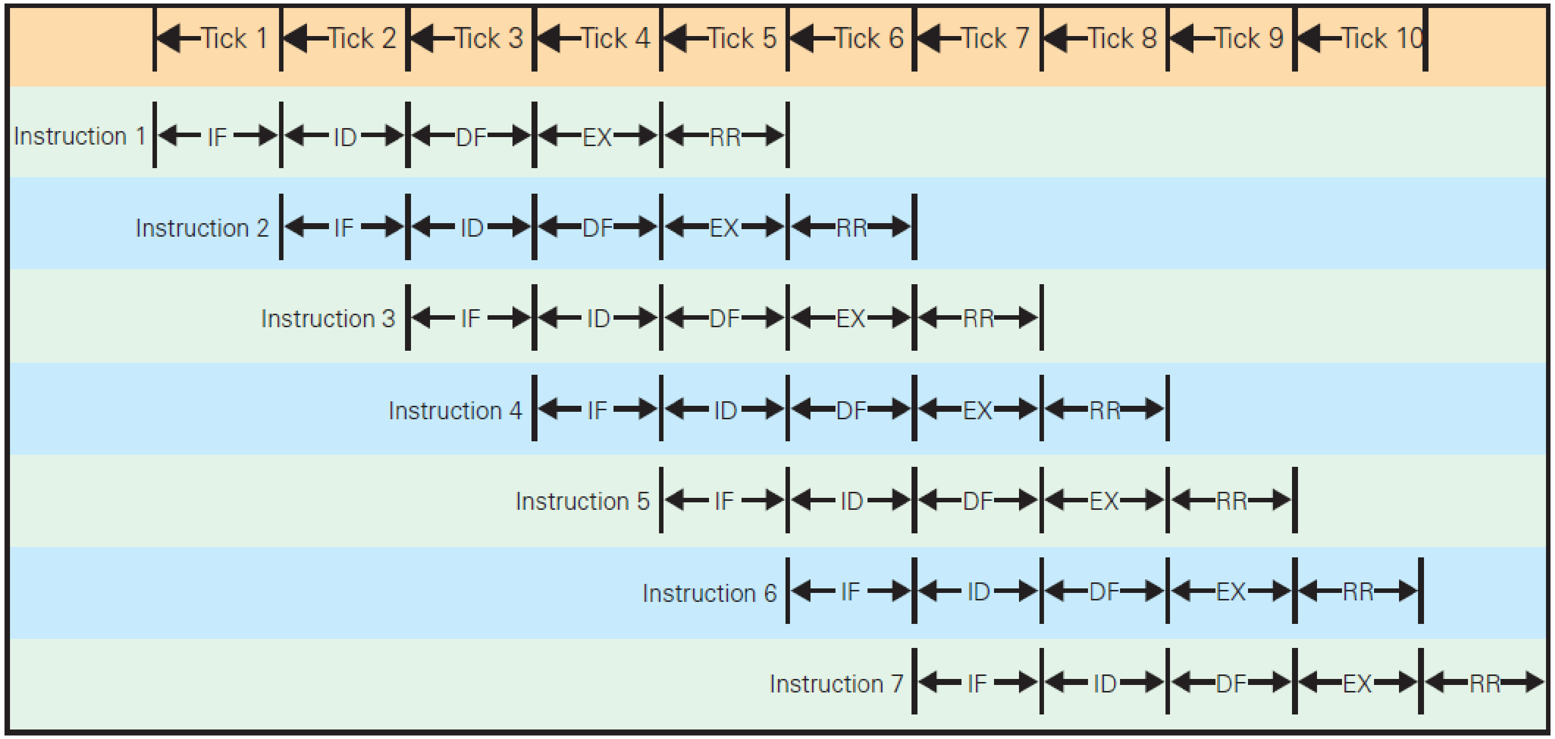
Figure 9.11 Standard prefixes from the Système International (SI) convention on scientific measurements. Generally a prefix refers to a power of 1000, except when the quantity (for example, memory) is counted in binary; for binary quantities the prefix refers to a power of 1024, which is 2^{10}.

One Cycle per Clock Tick

- A computer with a 1 GHz clock has one billionth of a second ($1/10^9$: one nanosecond) between clock ticks to run the Fetch/Execute Cycle.
- In that amount of time, light travels about one foot (~30 cm)
- Modern computers try to start an instruction on each clock tick
- They pass off completing the instruction to other circuitry
- This process is called pipelining and frees the fetch unit to start the next instruction before the last one is done
- It is not quite true that 1,000 instructions are executed in 1,000 ticks

5 steps of one instruction F/E cycle need 5 clock ticks,
but owing to pipelining, 5 steps of one instruction F/E cycle needs only 1 clock tick.
Therefore, it looks like one instruction F/E cycle needs only 1 clock tick!

Schematic Diagram of a Pipelined Fetch/Execute Cycle



Schematic diagram of a pipelined Fetch/Execute Cycle. On each tick, the IF (Instruction Fetch) circuit starts a new instruction, and then passes it along to the ID (Instruction Decode) unit; the ID unit works on the instruction it receives, and when it finishes, it passes it along to the DF (Data Fetch) circuit, and so on. When the pipeline is filled, five instructions are in process at once, and one instruction is finished on each clock tick, making the computer appear to be running at one instruction per tick.

Many Simple Machine-Level Operations

- Computers "know" and "perform" very few machine-level instructions
- In fact, there are only about 20 different kinds of machine-level operations
    - Instructions such as ADD integers, ADD real numbers, ADD floating numbers are all based on ADD operation
    - MOVE, COPY, MUL, DIV, ……
- Everything that computers do must be reduced to some combination of these primitive, hardwired instructions
- Computers achieve success at what they can do with speed
    - by executing many simple instructions per second

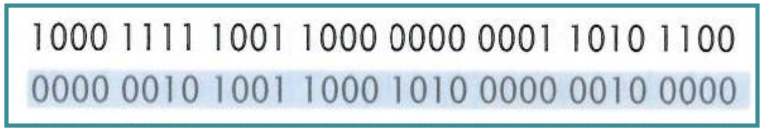
Assembly language

- Source code ➔ Assembly code ➔ Object code
- Compiler converts source code into assembly code

`Opacity = Opacity + 0.02` $\longrightarrow$ `ADD Opacity, TwoCths, Opacity`

- Assembler converts assembly code into object code

`ADD Opacity, TwoCths, Opacity` $\longrightarrow$



```
1000 1111 1001 1000 0000 0001 1010 1100
0000 0010 1001 1000 1010 0000 0010 0000
```

- The bits the processor needs are known as object code (= binary code consisting of 1’s and 0’s = machine code)

Sample Assembly Language: P88 Instruction Set

• COPY AX, mem	// AX $\leftarrow$ mem (AX: Computation Register)
• COPY mem, AX	// mem $\leftarrow$ AX
• ADD AX, mem	// AX $\leftarrow$ AX + mem
• SUB AX, mem	// AX $\leftarrow$ AX - mem
• MUL AX, mem	// AX $\leftarrow$ AX + mem
• DIV AX, mem	// AX $\leftarrow$ AX / mem
• CMP AX, mem	// If AX < mem ** CF : Condition Flag Then CF = B (below) Else CF = NB (not below)
• JMP lab1	// Go to instruction with label lab1
• JNB lab1	// Go to lab1 if CF = NB (Jump if not below)
• JB lab1	// Go to lab1 if CF = B (Jump if below)
• IN AX	// Read an integer from screen and Input it into AX
• OUT AX	// Read an integer from AX and Output it to screen

Compilation From High-Level PL to P88 Instruction Set

보조자료

```
C-like Programming Language
---------------------
{
   n = a.getInt();
   i = 1;
   fact = 1;
   while (i < (n+1))
   {
           fact = (fact * i);
           i = (i + 1);
   }
   b.setInt(fact);
}
---------------------
```

```
1            in        ax
2            copy      n, ax
3            copy      ax, #c1
4            copy      i, ax
5            copy      fact, ax
6  #l0       copy      ax, n
7            add       ax, #c1
8            copy      _e0, ax
9            copy      ax, i
10           cmp       ax, _e0
11           jnb       #l1
12           copy      ax, fact
13           mult      ax, i
14           copy      fact, ax
15           copy      ax, i
16           add       ax, #c1
17           copy      i, ax
18           jmp       #l0
19 #l1       copy      ax, fact
20           out       ax
21           halt
40 n                   0
41 i                   0
42 #c1                 1
43 fact                0
44 _e0                 0
note:
#l0                    6
#l1                    19
```

Compile 과정

- C Language

```
>>> gcc test.c   ➔   a.out (excutable binary code) 생성

>>> a.out  ➔ 실행

>>> gcc –S test.c ➔ test.s (assembly code) 생성
```

- Java Language

```
>>> javac test.java   ➔  test.class (java byte code) 생성

>>> java test  ➔ 실행
```

Integrated Circuits (ICs): Miniaturization [1/3]

- Computer clocks run at GHz rates because their processor chips are so tiny
- Electrical signals can travel one foot (30 cm) in a nanosecond
- Early computers (the size of whole rooms) could never have run as fast because their components were farther apart than one foot
- Making everything smaller has made computers faster!

Integrated Circuits (ICs): Integration [2/3]

- Early computers were made from separate parts (discrete components) wired together by hand
  - There were 3 wires coming out of each transistor, the 2 wires from each resistor, the 2 wires from each capacitor, and so on
  - Each had to be connected to the wires of another transistor, resistor, or capacitor
- Active components and the wires that connect them are manufactured from similar materials by a single (multistep) process
- IC technology places two transistors side by side in the silicon, and a wire connecting the two is placed in position

Integrated Circuits (ICs): Photolithography [3/3]

- ICs are fabricated with a printing process called photolithography:
  - Begin by depositing a layer of material (like aluminum) on the silicon
  - Cover that layer with a light-sensitive material called photoresist, and place a mask over it
  - The mask has a pattern corresponding to the features being constructed
  - Exposure to ultraviolet light causes open areas to harden
  - Unexposed areas do not harden and can be washed away leaving the pattern
  - Hot gases etch the original layer
  - When the remaining photoresist is removed, the pattern from the mask remains

** photolithography (광식각법, 사진석판술), etch 부식시키다, 에칭하다

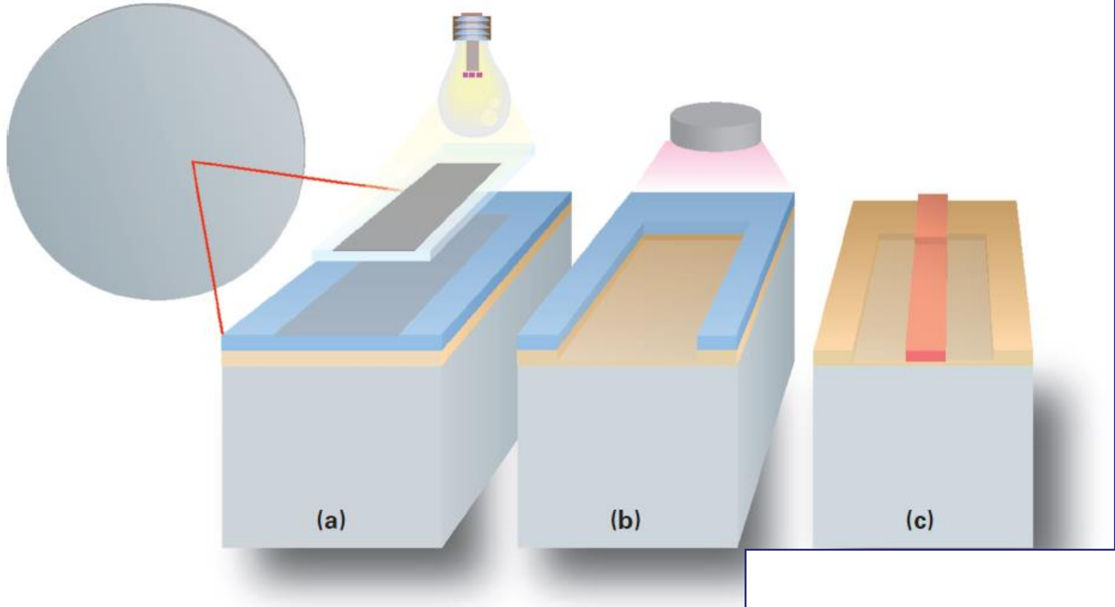
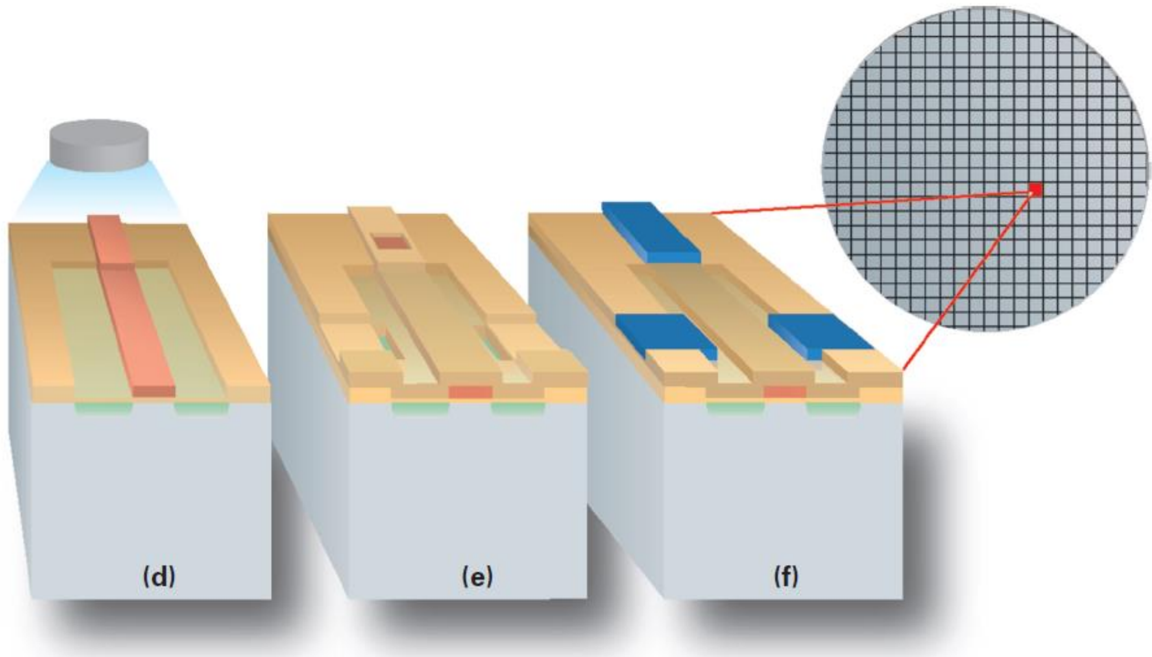


Figure 9.16 Early steps in the fabrication process. (a) A layer of p
UV light through a pattern mask (light blue), hardening the expose
the unexposed photoresist, hot gases etch away (nearly all of) the
ing resist is washed away and other layers are created by repeati
processes. In later stages of the fabrication process, (d) "impuriti



diffused into the silicon surface in a process called doping, which improves the availability of electrons in this region of the silicon. (e) After additional layering, etching exposes contact points for metal wires, and (f) a metal (dark blue) such as aluminum is deposited, creating "wires" to connect to other transistors. Millions of such transistors form a computer chip occupying a small square on the final fabricated wafer.

Semiconductor Technology: The Field Effect [1/2]

- Conductivity of a semiconductor is controlled using the field effect
  - Objects can be charged positively or negatively
  - The effect that charged objects have on each other without actually touching is called the field effect
- Field Effect Transistor

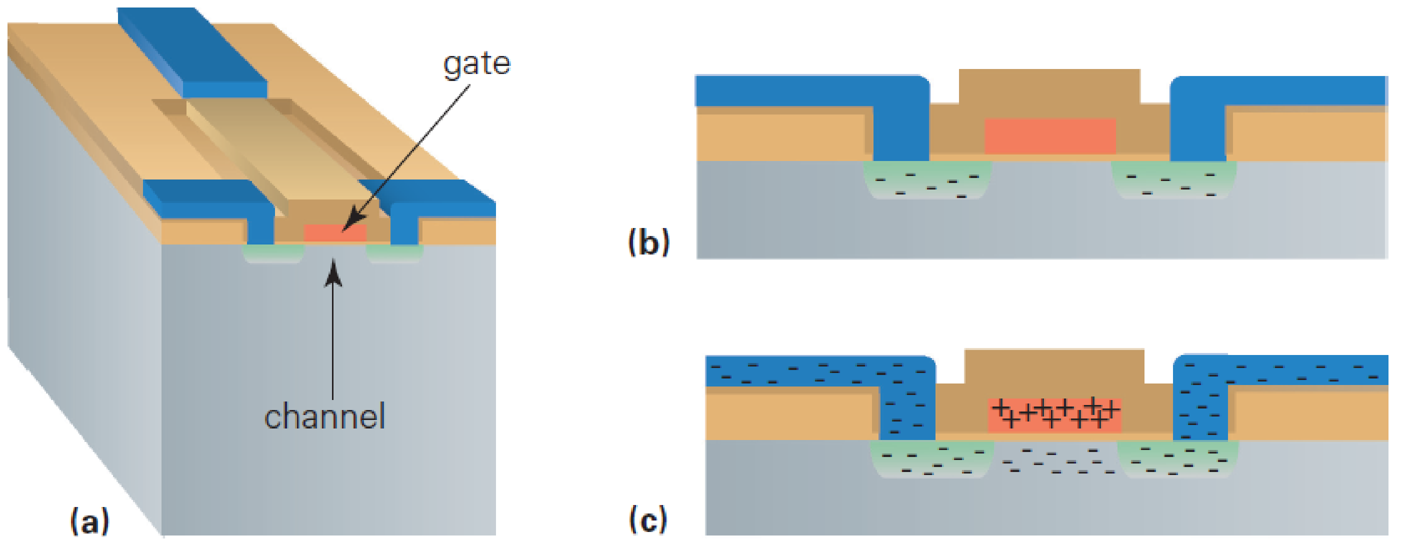


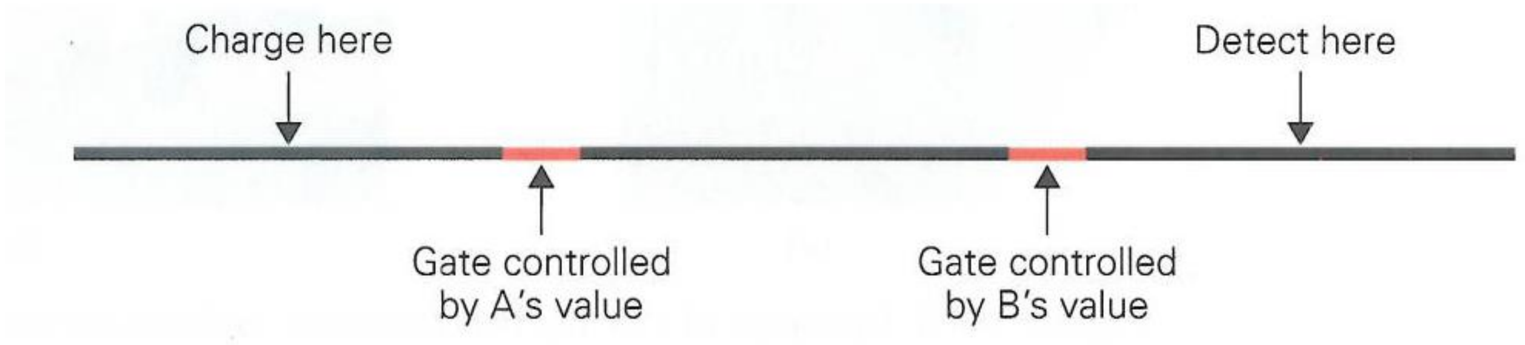
Figure 9.15 Operation of a field effect transistor. (a) Cross-section of the transistor of Figure 9.16(f). (b) The gate (red) is neutral and the channel, the region in the silicon below the gate, does not conduct, isolating the wires (blue); (c) charging the gate causes the channel to conduct, connecting the wires.

Semiconductor Technology: The Field Effect [2/2]

- A transistor is a connector between two wires that can be controlled to allow a charge to flow between the wires (conduct) or not
  - Originally MOS (Metal Oxide Semiconductor)
  - Moderndday transistors adopt CMOS technology (“complementary MOS”)
- The part between the ends is called a channel, because it creates a path for electricity to travel on
  - An insulator covers the channel
  - Passing over the insulator is a third wire called the gate
- The silicon in the channel can conduct electricity when it is in a charged field
  - Charging the gate positively creates a field over the channel
  - Electrons are then attracted from the silicon into the channel, causing it to conduct
- If the field is removed, the electrons disperse into the silicon, the channel doesn’t conduct

Implementing ALU operations

- The ability to control when semiconductors do and don't conduct electricity is the main process used in computer construction
- A simple principle of setting up a situation in which the conductivity of a wire is controlled to create a logical conclusion is needed
  - It is the basis of all the instructions and operations of a computer
  - In the ALU hardware, the circuit computes A AND B for any logical values A and B
  - Such a circuit (AND) is part of the ALU, performing the Instruction Execute step of all the AND instructions



Combining the Ideas: from Applications to Electrons

- Start with an information-processing task.
  - Task is performed by an application implemented as a large
    - The program performs the specific operations of the application
      - The program’s commands are compiled into many simple assembly language instructions (by compiler)
        - The assembly instructions are then translated into a more primitive binary form (by assembler)
          - Fetch/Execute Cycle executes the instructions
            - Instruction execution would need ALU operations
              - Transistors in ALU circuits are working

Summary [1/2]

- Modern software is written in a language using familiar terms and operations, though they are expressed very briefly; the code relies heavily on the software stack
- The repeating FETCH/EXECUTE process fetches each instruction (indicated by the PC), decodes the operation, retrieves the data, performs the operation, and stores the result back into the memory
- This process is hardwired into the control subsystem, one of the five components of a processor
- The memory, a very long sequence of bytes, each with an address, stores the program and data while the program is running
- The ALU does the actual computing

Summary [2/2]

- The input and output units are the interfaces for the peripheral devices connected to the computer
- Machine instructions do not refer to the data (operands) directly, but rather indirectly. Thus, different computations can be done with an instruction, just by changing the data in the referenced memory locations each time the instruction is executed
- Programmers use sophisticated programming languages to create operating systems as well as complex applications software
- The basic ideas of integrated circuits are integrating active and connective components, fabrication by photolithography, and controlling conductivity through the field effect