2 pages per horizontal
Both sides, flip short
Landscape, Auto-rotate

CDA 3103
Summer 2016: Computer Organization

Chapter 1: Introduction

LoveKumar Sah

Computer Science and Engineering

University of South Florida

Materials adopted from Prof. Hao Zheng & Prof. Yan Zhang

OBJECTIVES

- Know the difference between computer organization and computer architecture.
- Understand units of measure common to computer systems.
- Generations of Computer
- Understand the computer as a layered system.
- Be able to explain the von Neumann architecture and the function of basic computer components.

OVERVIEW

- Why study computer organization and architecture?
  - Design better programs, including system software such as compilers, operating systems, and device drivers.
  - Optimize program behavior.
  - Evaluate (benchmark) computer system performance.
  - Understand time, space, and price tradeoffs.

COMPUTER ORGANIZATION Vs ARCHITECTURE

- Computer organization – implementing Architecture. Physical implementation
  - Encompasses all physical aspects of computer systems.
  - E.g., circuit design, control signals, memory types. physical Aspects
  - *How does a computer work?*
- Computer architecture – Logic & component design
  - Logical aspects of system implementation as seen by the programmer.
  - It is a combination of computer's hardware and Instruction Set Architecture (ISA).
  - ISA is an agreed upon interface between software and hardware.
  - E.g., instruction sets, instruction formats, data types, addressing modes.
  - *How do I design a computer?*

Struct & Behavior

- Computer Organization -

Comp Org - How to implement Comp Architecture work. All about physical implementation. While Archit. is about logic design

ISA - Instruction Set Architecture

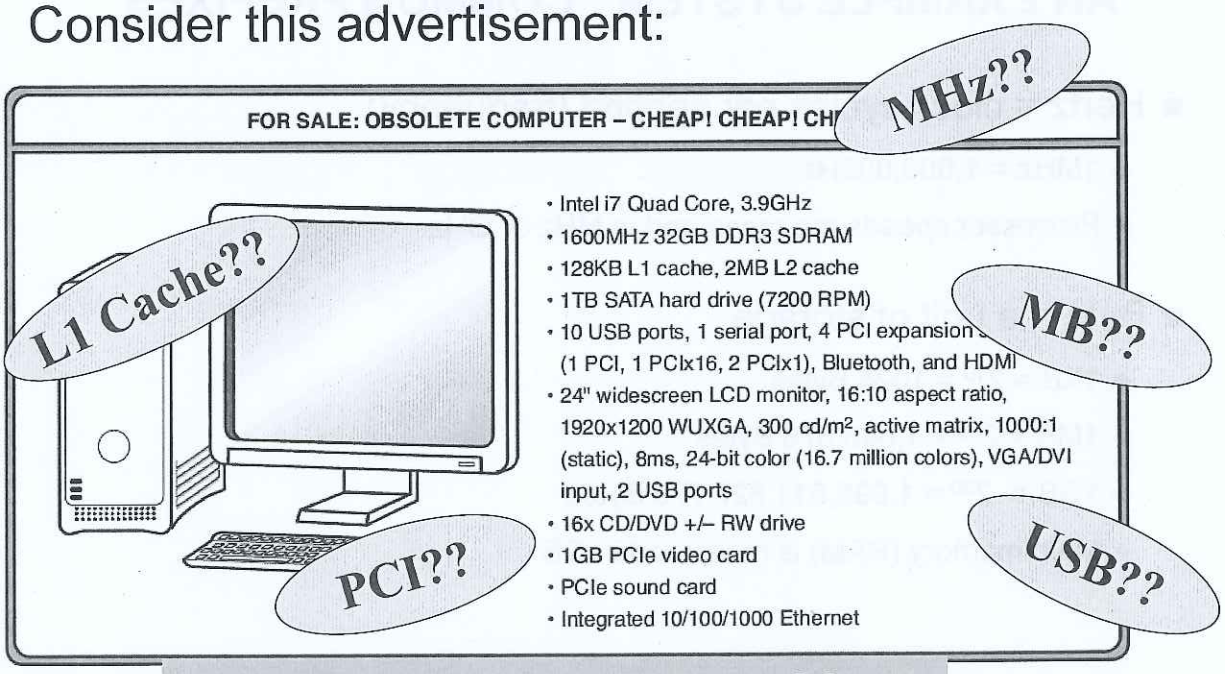
	Decimal	Byte
Kilo	10^3	2^{10}
Mega	10^6	2^{20}
Giga	10^9	2^{30}
Tera	10^{12}	2^{40}

MAIN COMPONENTS OF COMPUTER

- At the most basic level, a computer is a device consisting of three pieces:
  - A processor to interpret and execute programs
  - A memory to store both data and programs
  - A mechanism for transferring data to and from the outside world.
  - Processor -> Student's brain, Memory -> Notes taken, Pen/Pencil -> I/O devices.

An Example System

Consider this advertisement:



What does it all mean??

AN EXAMPLE SYSTEM : COMMON PREFIXES

Measures of capacity and speed:

- Kilo- (K) = 1 thousand = 10^3 and 2^{10}
- Mega- (M) = 1 million = 10^6 and 2^{20}
- Giga- (G) = 1 billion = 10^9 and 2^{30}
- Tera- (T) = 1 trillion = 10^{12} and 2^{40}
- Peta- (P) = 1 quadrillion = 10^{15} and 2^{50}

Measures of time and space:

- Milli- (m) = 1 thousandth = 10^{-3}
- Micro- (μ) = 1 millionth = 10^{-6}
- Nano- (n) = 1 billionth = 10^{-9}
- Pico- (p) = 1 trillionth = 10^{-12}
- Femto- (f) = 1 quadrillionth = 10^{-15}

AN EXAMPLE SYSTEM : COMMON PREFIXES

- Hertz = clock cycles per second (frequency)
  - 1MHz = 1,000,000Hz
  - Processor speeds are measured in MHz or GHz.
- Byte = a unit of storage
  - 1KB = 2^{10} = 1024 Bytes
  - 1MB = 2^{20} = 1,048,576 Bytes
  - 1GB = 2^{30} = 1,099,511,627,776 Bytes
  - Main memory (RAM) is measured in GB

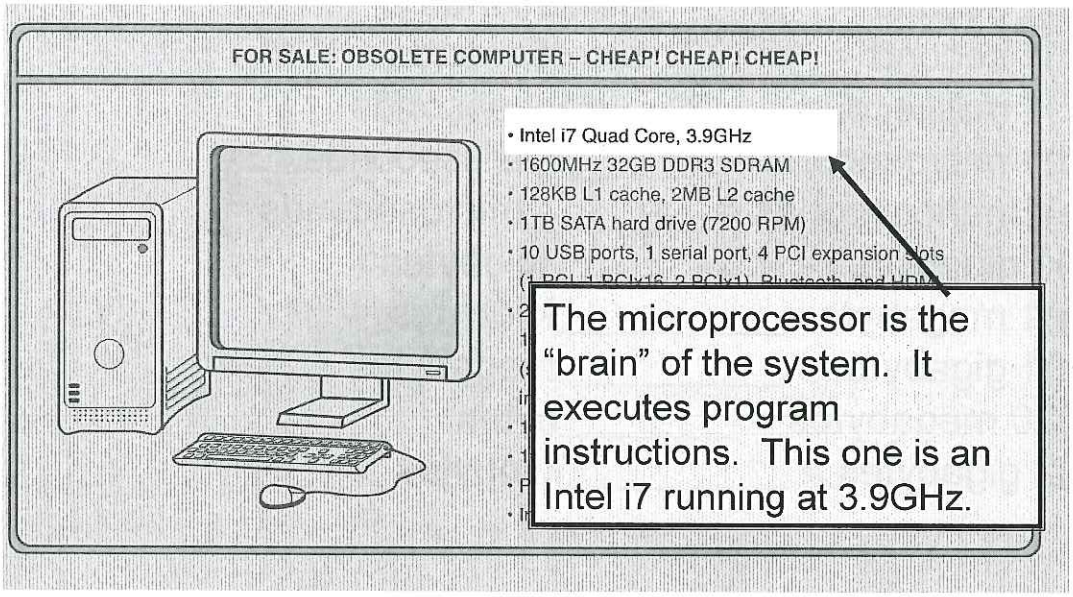
Exercises

a)1 second = 10^3 milliseconds
b)1 second = 10^6 microseconds
c)1 millisecond = 10^6 nanoseconds
d)1 microsecond = _____ milliseconds
e)1 nanoseconds = _____ microseconds
f)1 gigabyte = _____ kilobytes
g)1 megabyte = _____ kilobytes
h)1 gigabyte = _____ megabytes
i)20 megabytes = _____ bytes
j)2 gigabytes = _____ kilobytes

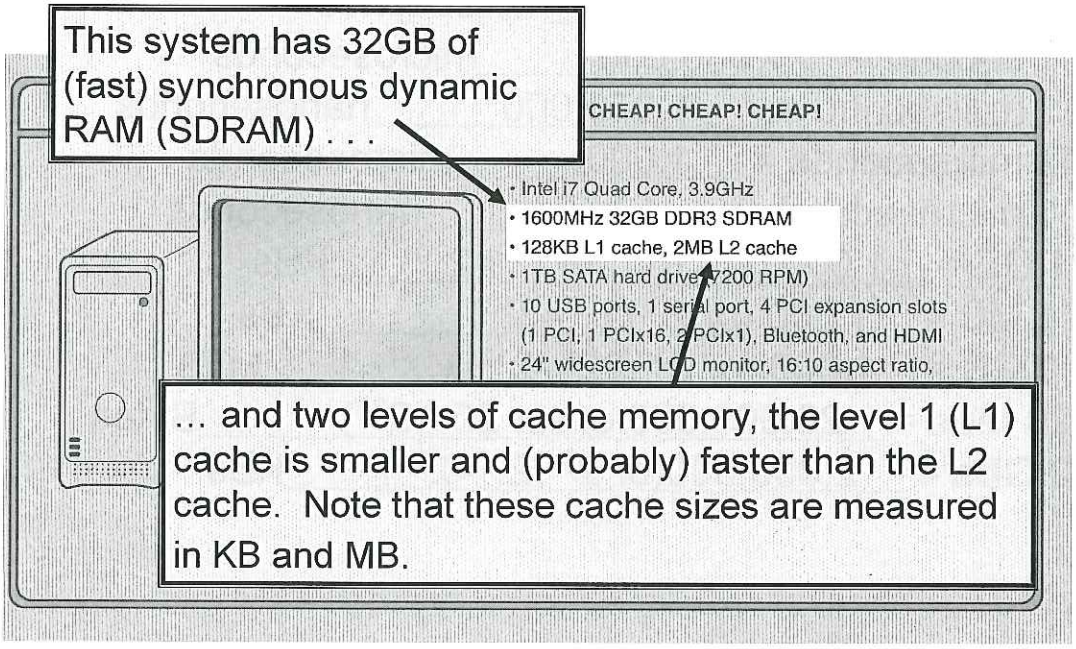
Exercises Solution

a)1 second = 1000 milliseconds
b)1 second = 1,000,000 microseconds
c)1 millisecond = 1,000,000 nanoseconds
d)1 milliseconds = 1,000 microseconds
e)1 microseconds = 1,000 nanoseconds
f)1 GB = 1,000,000 (or $2^{30}/2^{10}=2^{20}$) KBs
g)1 MB = 1,000 (or $2^{20}/2^{10}=2^{10}$) KBs
h)1 GB = 1,000 (or $2^{30}/22^{0}=2^{10}$) MBs
i)20 MBs = 20,000,000 or (or $20 * 2^{20}$) Bytes
j)2 GBs = 2,000,000(or $2^{31}/2^{10}=2^{21}$) KBs

AN EXAMPLE SYSTEM

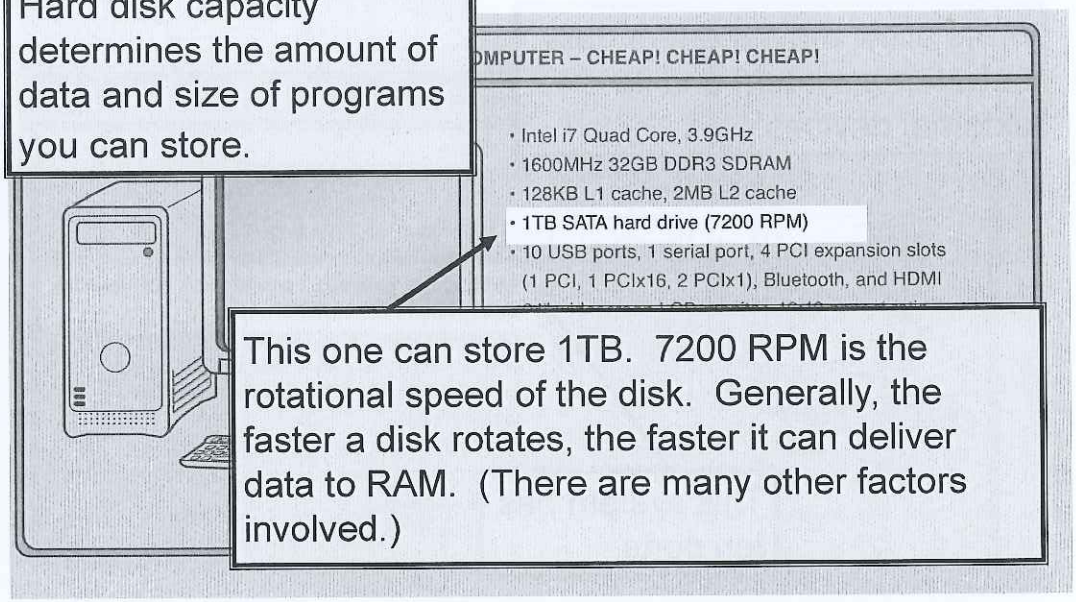


AN EXAMPLE SYSTEM



AN EXAMPLE SYSTEM

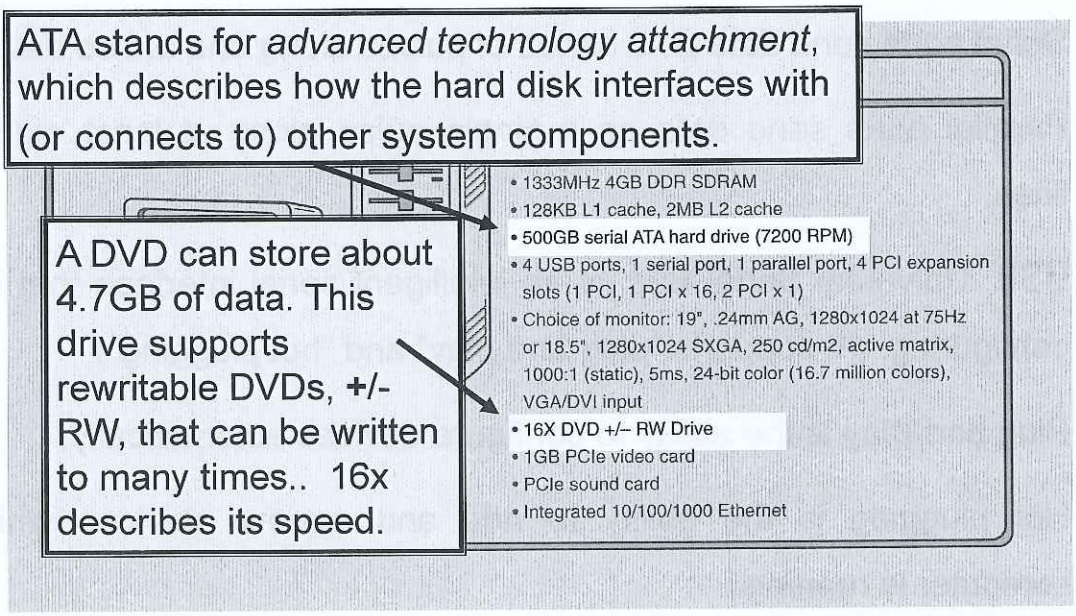
Hard disk capacity determines the amount of data and size of programs you can store.



This one can store 1TB. 7200 RPM is the rotational speed of the disk. Generally, the faster a disk rotates, the faster it can deliver data to RAM. (There are many other factors involved.)

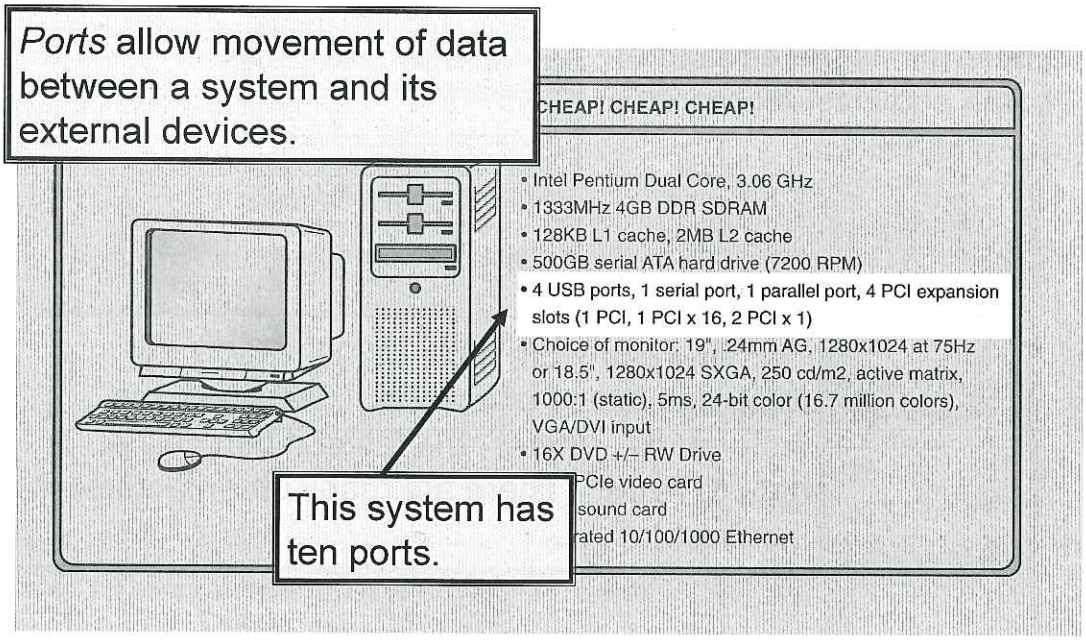
AN EXAMPLE SYSTEM

ATA stands for *advanced technology attachment*, which describes how the hard disk interfaces with (or connects to) other system components.



A DVD can store about 4.7GB of data. This drive supports rewritable DVDs, +/- RW, that can be written to many times.. 16x describes its speed.

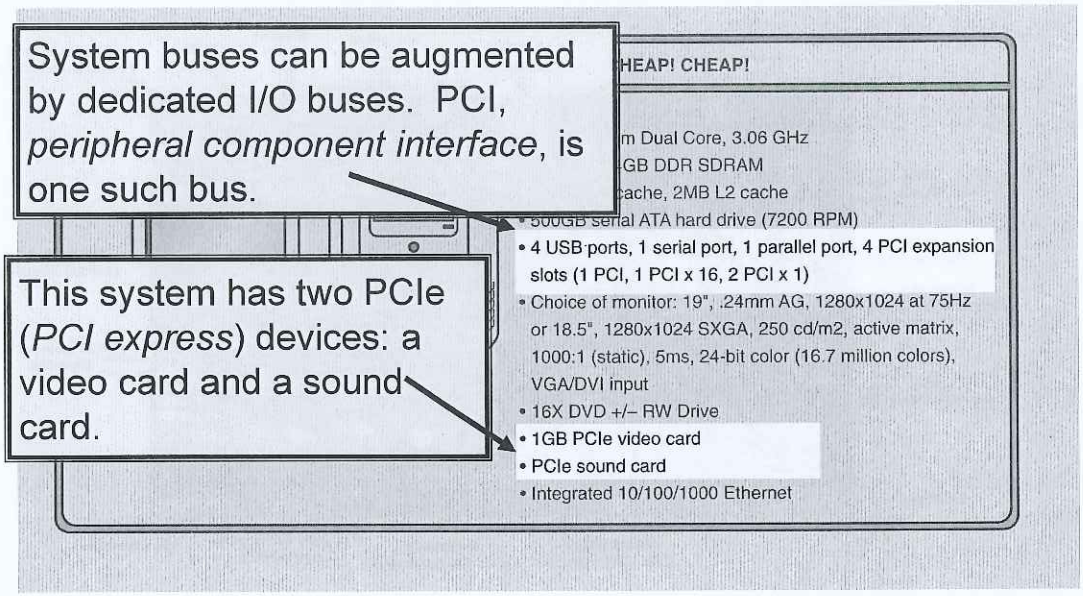
AN EXAMPLE SYSTEM



AN EXAMPLE SYSTEM

- Serial ports send data as a series of pulses along one or two data lines.
- Parallel ports send data as a single pulse along at least eight data lines.
- USB, Universal Serial Bus, is an intelligent serial interface that is self-configuring. (It supports "plug and play" and "hot plugging").
- Plug and Play is the ability to configure devices automatically.
- Hot plugging is the ability to add and remove devices while the computer is running.

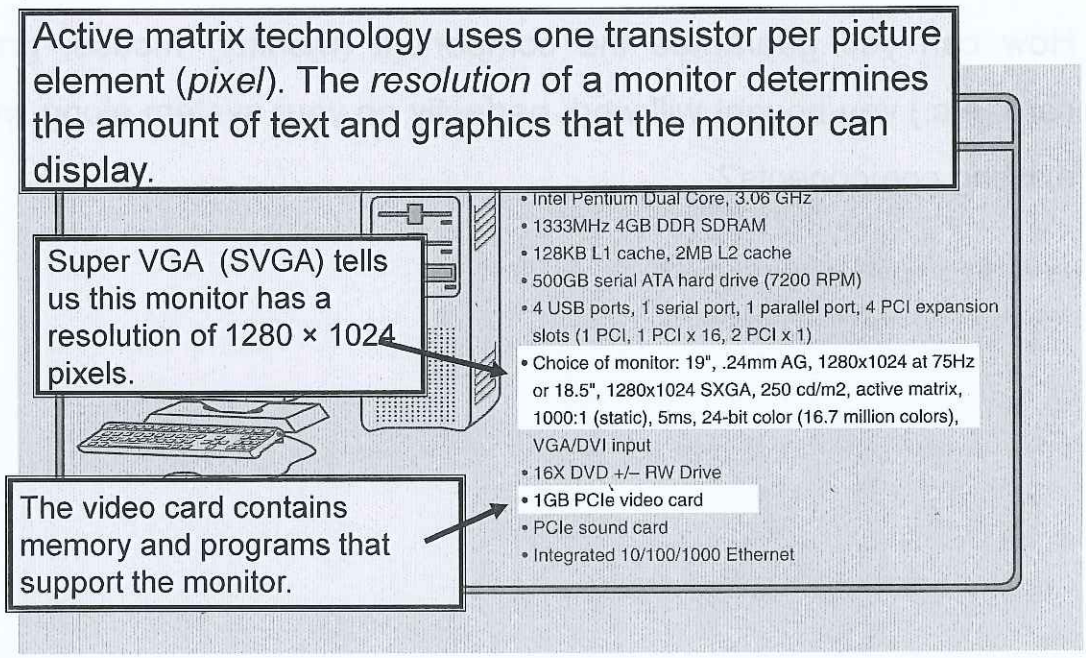
AN EXAMPLE SYSTEM



System buses can be augmented by dedicated I/O buses. PCI, *peripheral component interface*, is one such bus.

This system has two PCIe (*PCI express*) devices: a video card and a sound card.

AN EXAMPLE SYSTEM

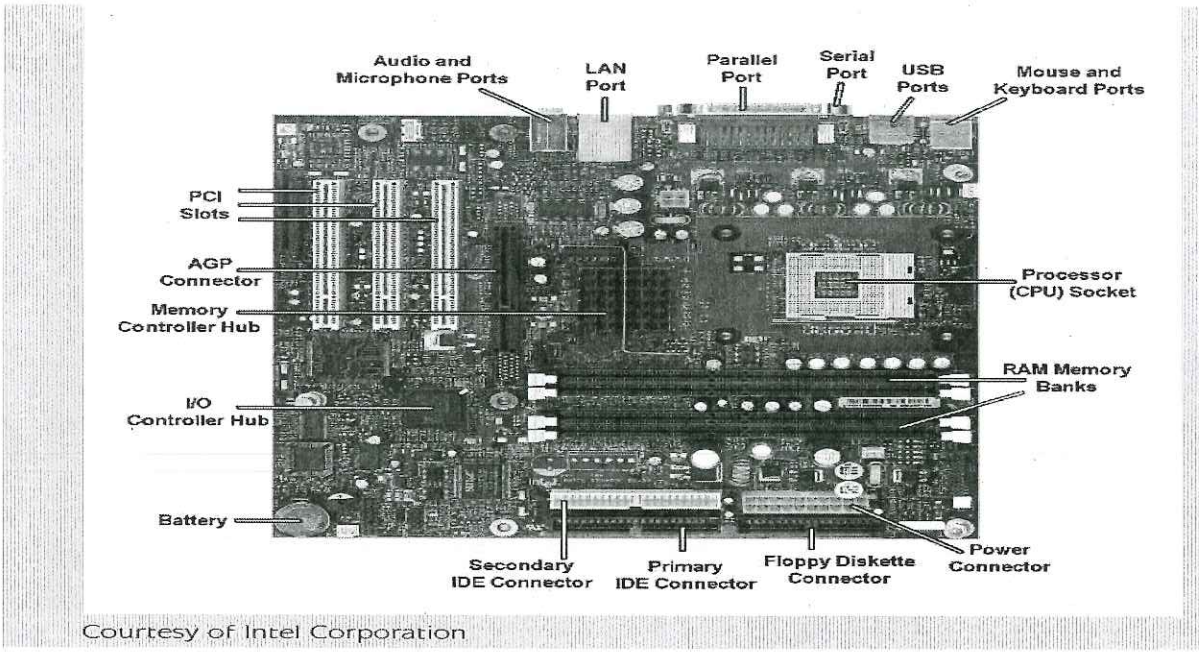


Active matrix technology uses one transistor per picture element (*pixel*). The *resolution* of a monitor determines the amount of text and graphics that the monitor can display.

Super VGA (SVGA) tells us this monitor has a resolution of 1280 × 1024 pixels.

The video card contains memory and programs that support the monitor.

AN EXAMPLE SYSTEM



THE BIG QUESTION

- How can you guarantee the component (monitor, mouse, graphics card, etc.) you bought will work perfectly on your system along with the existing components?

STANDARDS ORGANIZATION

- There are many organizations that set computer hardware standards--to include the interoperability of computer components.
- Some of the most important standards-setting groups are:
  - The Institute of Electrical and Electronic Engineers (IEEE)
  - The International Organization for Standardization (ISO)
  - The International Telecommunications Union (ITU)
- National groups establish standards within their respective countries:
  - The American National Standards Institute (ANSI)

HISTORICAL DEVELOPEMENT

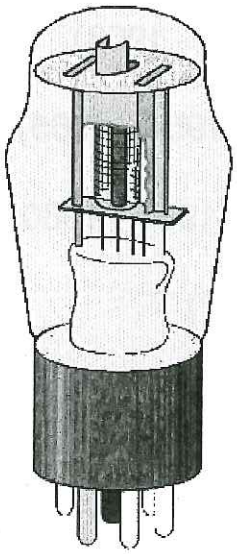
- Generation Zero: Mechanical Calculating Machines (1642 - 1945)
- The First Generation: Vacuum Tube Computers (1945 - 1953)
- The Second Generation: Transistorized Computers (1954 - 1965)
- The Third Generation: Integrated Circuit Computers (1965 - 1980)
- The Fourth Generation: VLSI Computers (1980 - ????)

Generation Zero: Mechanical Calculating Machines (1642 - 1945)

- Calculating Clock - Wilhelm Schickard (1592 - 1635).
  - First mechanical calculator, add and subtract numbers with as many as six digits.
- Pascaline - Blaise Pascal (1623 - 1662).
  - Addition with carry and subtraction.
- Difference Engine - Charles Babbage (1791 - 1871), also designed but never built the Analytical Engine.
  - Based on a calculating technique called the method of difference
  - The Analytical Engine included several components associated with modern computers: an arithmetic processing unit to perform calculations, a memory, and input and output devices.
  - "the father of computing" — Charles Babbage
- Punched card tabulating machines - Herman Hollerith (1860 - 1929).
  - Hollerith cards were commonly used for computer input well into the 1970s.

Created IBM

The First Generation: Vacuum Tube Computers (1945 - 1953)



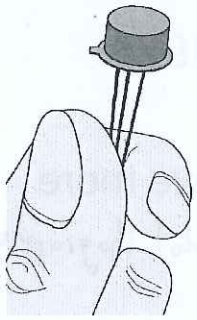
- Atanasoff Berry Computer (ABC) (1937 - 1938) solved systems of linear equations.
- The first completely electronic computer
- John Atanasoff and Clifford Berry of Iowa State University.

Thermionic emission: the follow of electrons from negatively charged cathode to the positively charged anode.

The First Generation: Vacuum Tube Computers (1945 - 1953)

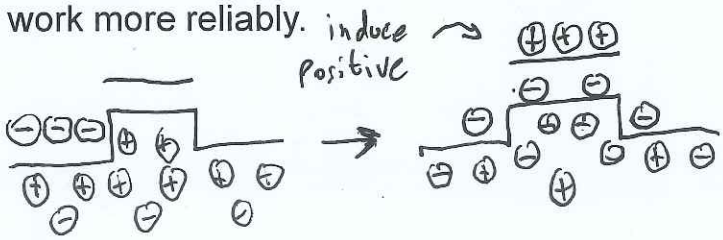
- The ENIAC was the first all-electronic, general-purpose digital computer.
  - Electronic Numerical Integrator and Computer (ENIAC)
  - John Mauchly and J. Presper Eckert
  - University of Pennsylvania, 1946
- The IBM 650 first mass-produced computer. (1955)
  - It was phased out in 1969.

The Second Generation: Transistorized Computers (1954 - 1965)



- IBM 7094 (scientific) and 1401 (business)
- Digital Equipment Corporation (DEC) PDP-1
- Univac 1100
- Control Data Corporation 1604.
- . . . and many others.

- Transistor: the follow of electrons in the solid medium.
- The vacuum tubes are not very dependable.
- Transistors consume less power than vacuum tubes, are smaller, and work more reliably.



The Third Generation: Integrated Circuit Computers (1965 - 1980)

- Multiple transistors were integrated onto one chip.
- The Third Generation: Integrated Circuit Computers (1965 - 1980)
  - IBM 360
  - DEC PDP-8 and PDP-11
  - Cray-1 supercomputer
  - . . . and many others.
- By this time, IBM had gained overwhelming dominance in the industry.

The Fourth Generation: VLSI Computers (1980 - ????)

- The Fourth Generation: VLSI Computers (1980 - ????)
  - Very large scale integrated circuits (VLSI) have more than 10,000 components per chip. done via photolithography
  - Enabled the creation of microprocessors.
  - The first was the 4-bit Intel 4004.
  - Later versions, such as the 8080, 8086, and 8088 spawned the idea of "personal computing."

Computer Evolution

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost

Year	Technology	Relative performance/cost	
1951	Vacuum tube	1	
1965	Transistor	35	
1975	Integrated circuit (IC)	900	
1995	Very large-scale Integrated Circuit	2,400,000	
2013	Ultra large-scale Integrated Circuit	250,000,000,000	

Moore's Law

- Moore's Law (1965)
  - Gordon Moore, Intel founder
  - "The density of transistors in an integrated circuit will double every year."
- Contemporary version:
  - "The density of silicon chips doubles every 18 months."

But this "law" cannot hold forever ...

Rock's Law

- Rock's Law
  - Arthur Rock, Intel financier
  - "The cost of capital equipment to build semiconductors will double every four years."
  - In 1968, a new chip plant cost about $12,000.

At the time, $12,000 would buy a nice home in the suburbs. An executive earning $12,000 per year was "making a very comfortable living."

Rock's Law

- Rock's Law
  - In 2012, a chip plants under construction cost well over $5 billion.

$5 billion is more than the gross domestic product of some small countries, including Barbados, Mauritania, and Rwanda.

- For Moore's Law to hold, Rock's Law must fall, or vice versa. But no one can say which will give out first.

Moore's Law Exercise

It was found that the number of transistors on various chips made by Intel could be approximated by $y = 2336(1.39)^x$, where $x = 0$ corresponds to 1971.

- What is the doubling time for the function $y = 2336(1.39)^x$?
- Suppose a function of the form $y = y_0a^x$, where x is in years, describes a quantity that doubles every 18 months. What is the value of a?

1.39 = 39% increase in capacity for every 2.1 yrs

if we want to accomplish the same in 18 mths, the capacity needs to be increased to 1.59

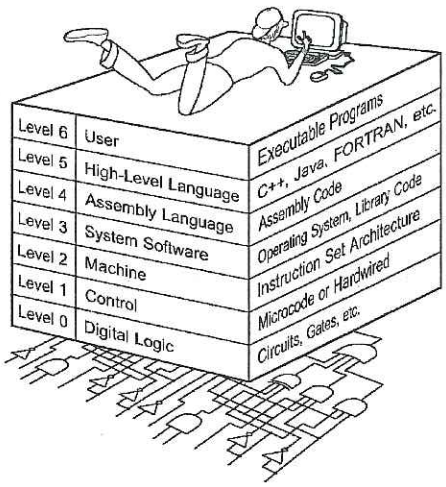
Moore's Law – Exercise Solution

It was found that the number of transistors on various chips made by Intel could be approximated by $y = 2336(1.39)^x$, where $x = 0$ corresponds to 1971.

- What is the doubling time for the function $y = 2336(1.39)^x$?
  - Hint: $(1.39)^x = 2 \Rightarrow x = 2.1$ years
- Suppose a function of the form y = y0ax, , where x is in years, describes a quantity that doubles every 18 months. What is the value of a?
  - Hint: 1.59

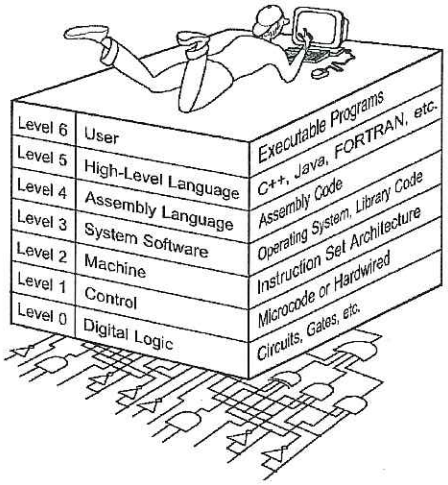
THE COMPUTER LEVEL HIERARCHY

- The hypothetical computer at each level is called a virtual machine.
- 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.



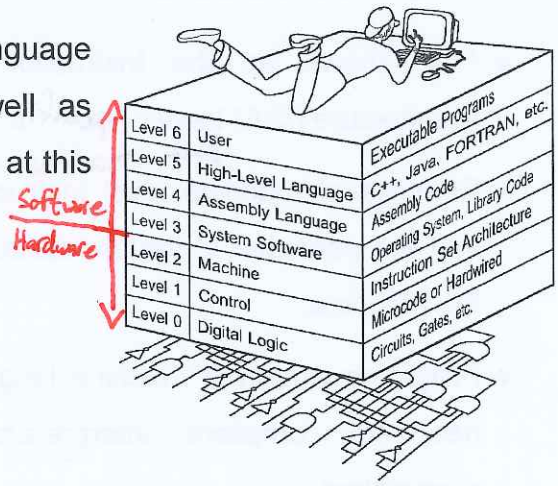
THE COMPUTER LEVEL HIERARCHY

- Level 6: The User Level
  - Program execution (word processor, games, etc.) 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.



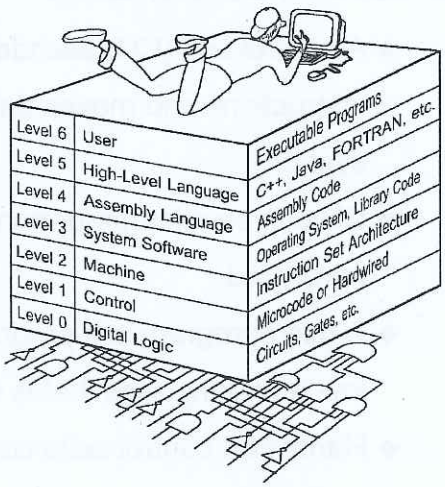
THE COMPUTER LEVEL HIERARCHY

- Level 4: Assembly Language Level
  - Acts upon assembly language produced from Level 5, as well as instructions programmed directly at this level.



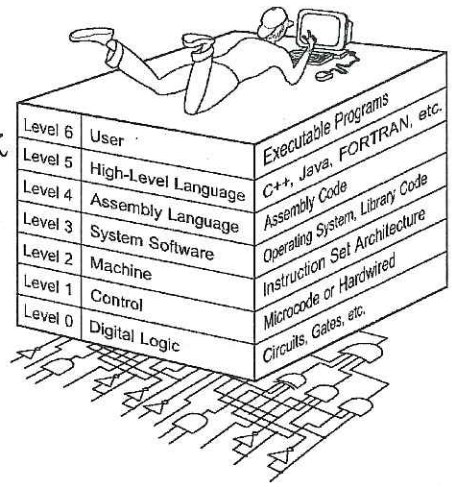
THE COMPUTER LEVEL HIERARCHY

- Level 3: System Software Level
  - Deals with Operating System instructions.
  - Responsible for multiprogramming, protecting memory, synchronization.
  - Controls executing processes on the system.



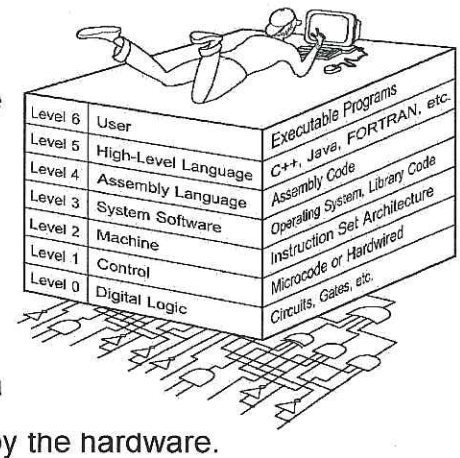
THE COMPUTER LEVEL HIERARCHY

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



THE COMPUTER LEVEL HIERARCHY

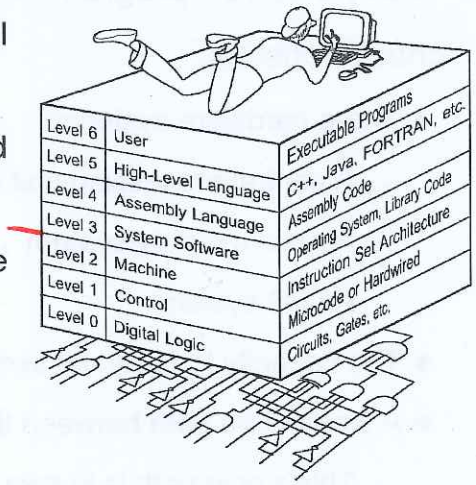
- Level 1: Control Level
  - A control unit (CU) decodes and execute instructions and moves data through the system.
  - CU can be microprogrammed or hardwired.
  - A microprogram is a program written in a low-level language that is implemented by the hardware.
  - Hardwired control units consist of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



orderly opening & closing transistor gates to perform operations in ALU. Allows data to move from Registers, to ALU, to memory

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.



THE VON NEUMANN MODEL

- Inventors of the ENIAC, John Mauchley and J. Presper Eckert, conceived of a computer that could store instructions in memory.

- Inventors of the ENIAC, John Mauchley and J. Presper Eckert, conceived of a computer that could store instructions in memory.

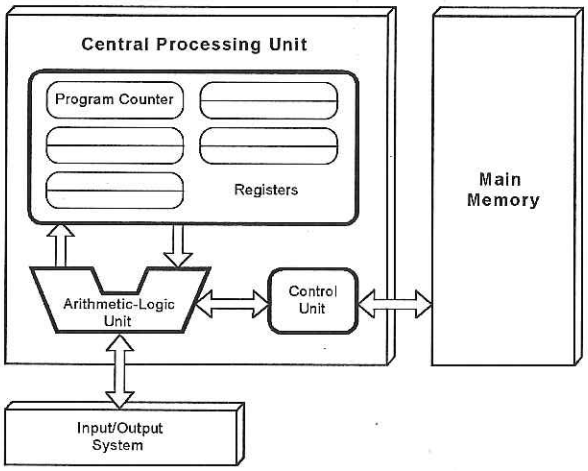
- **Stored-program computers** have become known as von Neumann Architecture systems.

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.

THE VON NEUMANN MODEL

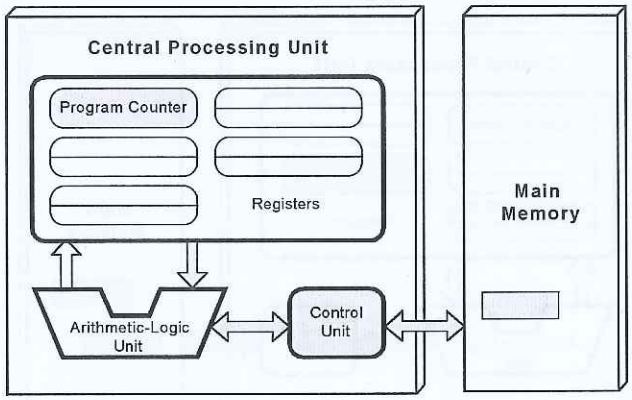
- These computers employ a fetch-decode-execute cycle to run programs.



program counter helps to grab the correct data from memory. Counter tells control what line to grab next

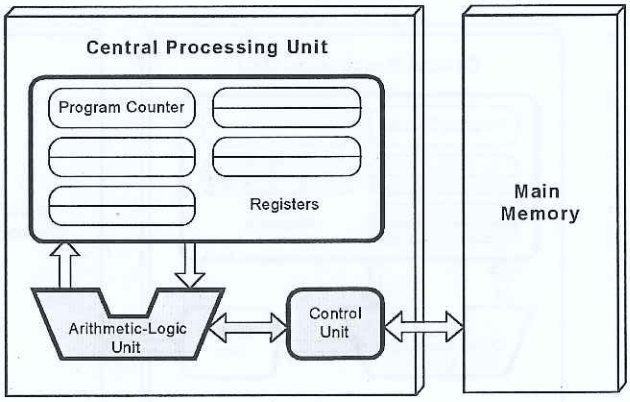
THE VON NEUMANN MODEL

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



THE VON NEUMANN MODEL

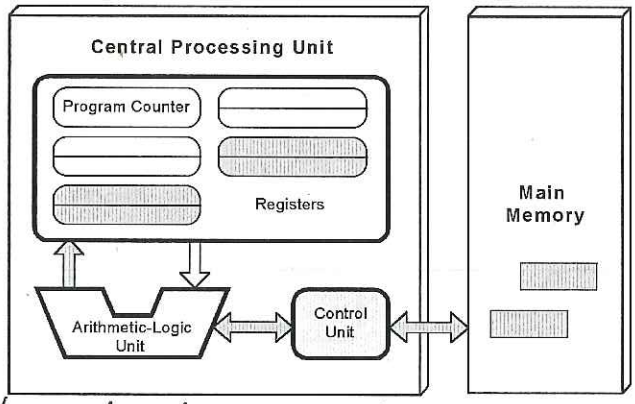
- The instruction is decoded into a language that the ALU can understand.



Control receives data to add, so it decodes the information first

THE VON NEUMANN MODEL

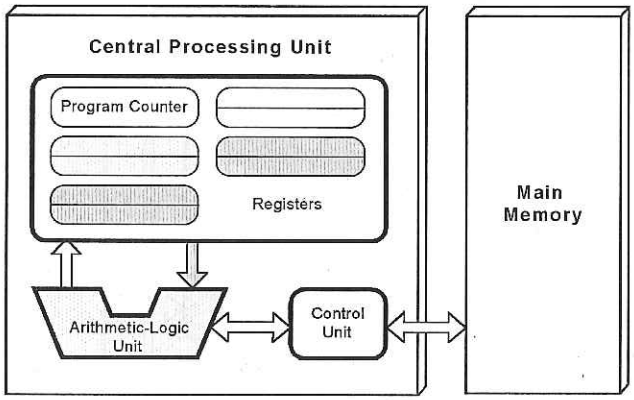
- Any data operands required to execute the instruction are fetched from memory and placed into registers within the CPU.



Control knows it needs two operands to add. It fetches & sends both to ALU so it can add

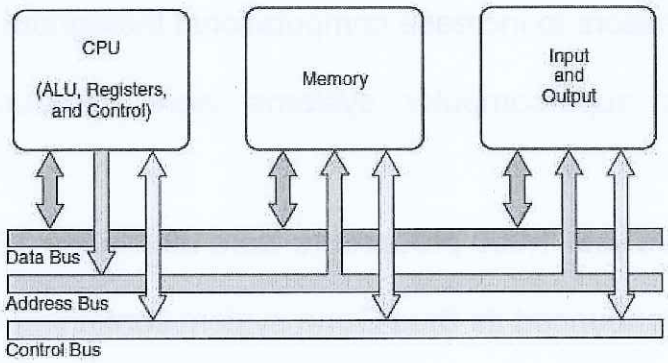
THE VON NEUMANN MODEL

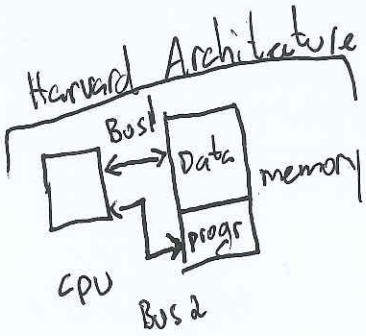
- The ALU executes the instruction and places results in registers or memory.



ALU adds, then stores info into register if it needs something else, or it sends it back to memory

THE VON NEUMANN MODEL (SYSTEM BUS)





NON VON NEUMANN MODEL

- Conventional stored-program computers have undergone many incremental improvements over the years.
- These improvements include adding specialized buses, floating-point units, and cache memories, more processors.
- Some of today's systems have separate buses for data and instructions - **harvard architecture**
- Other non-von Neumann systems provide special-purpose processors to offload work from the main CPU.
- More radical departures include quantum computing, parallel computing.

PARALLEL COMPUTING

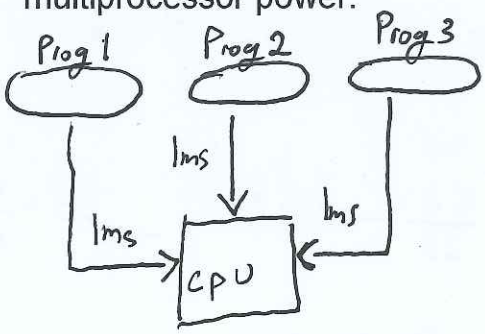
- In the late 1960s, high-performance computer systems were equipped with dual processors to increase computational throughput.
- In the 1970s supercomputer systems were introduced with 32 processors.
- Supercomputers with 1,000 processors were built in the 1980s.
- In 1999, IBM announced its Blue Gene system containing over 1 million processors.

PARALLEL COMPUTING

- Parallel processing allows a computer to simultaneously work on subparts of a problem.
- Multicore processors have two or more processor cores sharing a single die.
- Each core has its own ALU and set of registers, but all processors share memory and other resources.

PARALLEL COMPUTING

- Multi-core systems provide the ability to multitask
  - E.g., browse the Web while burning a CD
- Multithreaded applications spread mini-processes, **threads**, across one or more processors for increased throughput.
- New programming languages are necessary to fully exploit multiprocessor power.



process each program, or part of program for a few seconds. switches b/w every ms or ns.

CONCLUSION

- This chapter has given you an overview of the subject of computer architecture.
  - Difference between computer organization and arch.
  - Principle of Equivalence of Hardware and Software
  - Main components of a computer
  - Measures of quantities
  - Computer level Hierarchy
  - Von Neumann Model of computers

positional # system

Base 10 - express $\underset{3}{1}\underset{2}{0}\underset{1}{2}\underset{0}{4}$: $1\times10^3 + 0\times10^2 + 2\times10^1 + 4\times10^0 = 1024$

Binary $(10011101)_2$ - express in radix 10

$1\times2^7 + 1\times2^4 + 1\times2^3 + 1\times2^2 + 1\times2^0 = 157$

Binary $(0.1011)_2 = 2^{-1} + 2^{-3} + 2^{-4} = 0.6875$

Octal $(7204)_8 = 7\times8^3 + 2\times8^2 + 0\times8^1 + 4\times8^0 = 3716$

Hex $(3BA4)_{16} = 3\times16^3 + 11\times16^2 + 10\times16^1 + 4\times16^0 = 15268$

Convert 190 to base 3 $= 2\times3^4 =$ 190 − 162 = 28 $\rightarrow 1\times3^3 =$ 28 − 27 = 1 $\rightarrow 1\times3^0 =$ 1 − 1 = 0

$\underline{2}\times3^4 + \underline{1}\times3^3 + \underline{0}\times3^2 + \underline{0}\times3^1 + \underline{1}\times3^0 = 2\ 1\ 0\ 0\ 1$

Now using division

3 | 190	remainder 1	
3 | 63	remainder 0	
3 | 21	remainder 0	$= 2\ 1\ 0\ 0\ 1$
3 | 7	remainder 1	
3 | 2	remainder 2	

Convert to binary

2 | 190	r = 0
2 | 95	r = 1
2 | 47	r = 1
2 | 23	r = 1
2 | 11	r = 1
2 | 5	r = 1
2 | 4	r = 1
2 | 2	r = 0
2 | 1	r = 1

or

190	
−128	$2^7\times1$
62	
−32	$2^5\times1$
30	
−16	
14	
−8	
6	
−4	
2	

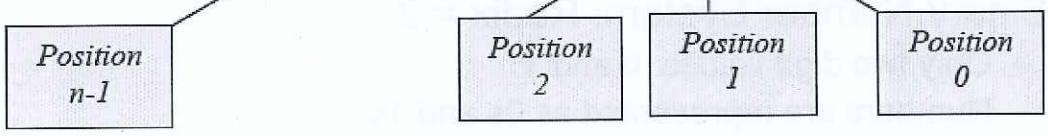
Bit, Nibble, Byte & Word

- Bit: the most basic unit of information in a computer.
  - Binary Digit
  - 1/0, True/False, closed/open, off/on, high/low;
- Nibble: A group of four bits
  - 1 nibble = 4 bits
- Byte: a group of eight bits.
  - 1 byte = 8 bits
- Word:
  - a contiguous group of bytes.
  - Word sizes of 16, 32, or 64 bits are most common.

Positional Number Systems

- A Number D consists of n digits with each digit has a particular *position*.

$$\mathbf{D} = \mathbf{d}_{n-1}\ \mathbf{d}_{n-2}\ \ldots\ldots\ \mathbf{d}_2\ \mathbf{d}_1\ \mathbf{d}_0$$



- Every digit *position* is associated with a fixed weight.
- Weight w_i for digit position i: powers of some constant value, called radix or the base, such that $w_i = r^i$.

$$D = d_{n-1}r^{n-1} + d_{n-1}r^{n-2} + \cdots + d_2r^2 + d_1r^1 + d_0r^0$$

- A number system of radix r, typically has a set of r **allowed digits** $\{0, 1, \cdots, r-1\}$.
- Most Significant Digit (MSD): the leftmost digit with the highest weight
- Least significant Digit (LSD): the rightmost digit with the lowest weight

Representing Fractions

- A number N_r in *radix* r can also have a fraction part:

$$N_r = \underbrace{d_{n-1}d_{n-2} \ldots d_1 d_0}_{\text{Integer Part}} \,.\, \underbrace{d_{-1}\, d_{-2} \ldots d_{-m+1}\, d_{-m}}_{\text{Fraction Part}} \qquad 0 \le d_i < r$$

MSD

Radix Point

LSD

- The number N_r represents the value:

$$N_r = d_{n-1} \times r^{n-1} + \ldots + d_1 \times r + d_0 + \qquad \textbf{(Integer Part)}$$

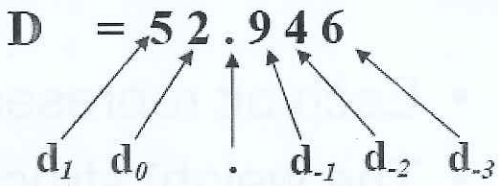
$$d_{-1} \times r^{-1} + d_{-2} \times r^{-2} \ldots + d_{-m} \times r^{-m} \qquad \textbf{(Fraction Part)}$$

Popular Number Systems

- Decimal Number System: Radix = 10
  - Ten digit values: 0, 1, 2, …, 9
- Binary Number System: Radix = 2
  - Only two digit values: 0 and 1
  - Numbers are represented as 0s and 1s
- Octal Number System: Radix = 8
  - Eight digit values: 0, 1, 2, …, 7
- Hexadecimal Number Systems: Radix = 16
  - Sixteen digit values: 0, 1, 2, …, 9, A, B, …, F
  - A = 10, B = 11, …, F = 15
- Octal and Hexadecimal numbers can be converted easily to Binary and vice versa

Decimal Numbers

- Radix $r = 10$
- 10 possible digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Weight of position i: $w_i = 10^i$

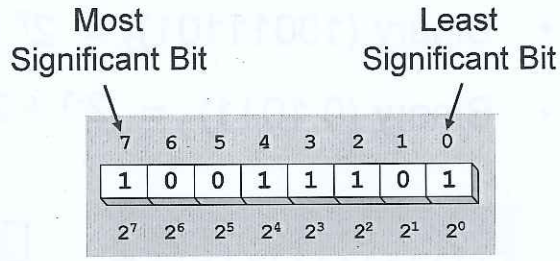


Number	5	2	.	9	4	6
Position	1	0	.	-1	-2	-3
Weight	10^1 = 10	10^0 = 1	.	10^{-1} = 0.1	10^{-2} = 0.01	10^{-3} = 0.001
Value	5 x 10	2 x 1	.	9 x 0.1	4 x 0.01	6 x 0.001
Value	50 +	2 +		0.9 +	0.04 +	0.006

$$D = 5x10^1 + 2x10^0 + 9x10^{-1} + 4x10^{-2} + 6x10^{-3}$$

Binary Numbers

- Each binary digit (called a bit) is either 1 or 0
- Bits have no inherent meaning, they can represent …
  - Unsigned and signed integers
  - Fractions
  - Characters
  - Images, sound, etc.



- Bit Numbering
  - Least significant bit (LSB) is rightmost (bit 0)
  - Most significant bit (MSB) is leftmost (bit 7 in an 8-bit number)
- The largest n bits binary number is $2^n - 1$ in decimal number

Weight Structure of Binary Numbers

- Each bit represents a power of 2
- The weight structure of a binary number is:

$$2^{n-1}2^{n-2}\ldots 2^3 2^2 2^1 2^0 . 2^{-1}2^{-2}\ldots 2^{-m}$$

Binary Point

Binary-to-Decimal Conversion

- Every binary number is a sum of powers of 2
- Decimal Value = $(d_{n-1} \times 2^{n-1}) + \ldots + (d_1 \times 2^1) + (d_0 \times 2^0)$
- Binary $(10011101)_2 = 2^7 + 2^4 + 2^3 + 2^2 + 1 = 157$
- Binary $(0.1011)_2 = 2^{-1} + 2^{-3} + 2^{-4} = 0.6875$

7	6	5	4	3	2	1	0
1	0	0	1	1	1	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

-1	-2	-3	-4
1	0	1	1
2^{-1}	2^{-2}	2^{-3}	2^{-4}

Some common powers of 2

2^n	Decimal Value	2^n	Decimal Value
2^0	1	2^8	256
2^1	2	2^9	512
2^2	4	2^{10}	1024
2^3	8	2^{11}	2048
2^4	16	2^{12}	4096
2^5	32	2^{13}	8192
2^6	64	2^{14}	16384
2^7	128	2^{15}	32768

Binary-to-Decimal Conversion Exercise

Convert the binary number 100101.01 to decimal

Solution:

1 0 0 1 0 1. 0 1

2^5 2^4 2^3 2^2 2^1 2^0. 2^{-1} 2^{-2}

32 16 8 4 2 1 . ½ ¼

32 +4 +1 +¼ = 37¼

Octal and Hexadecimal Numbers

- Octal = Radix 8
- Only eight digits: 0 to 7
- Digits 8 and 9 not used
- Hexadecimal = Radix 16
- 16 digits: 0 to 9, A to F
- A=10, B=11, …, F=15
- First 16 decimal values (0 to15) and their values in binary, octal and hex. Memorize table

Decimal Radix 10	Binary Radix 2	Octal Radix 8	Hex Radix 16
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

Octal/Hex-to-Decimal Conversion

- Octal to Decimal: $N_8 = (d_{n-1} \times 8^{n-1}) + \ldots + (d_1 \times 8) + d_0$
- Hex to Decimal: $N_{16} = (d_{n-1} \times 16^{n-1}) + \ldots + (d_1 \times 16) + d_0$
- Examples:

$(7204)_8 = (7 \times 8^3) + (2 \times 8^2) + (0 \times 8) + 4 = 3716$

$(3BA4)_{16} = (3 \times 16^3) + (11 \times 16^2) + (10 \times 16) + 4 = 15268$

Important Properties

- How many possible digits can we have in Radix r ?
  r digits: 0 to $r - 1$
- What is the result of adding 1 to the largest digit in Radix r?
  Since digit r is not represented, result is $(10)_r$ in Radix r
  Examples: $1_2 + 1 = (10)_2$ $\quad$ $7_8 + 1 = (10)_8$
  $9_{10} + 1 = (10)_{10}$ $\quad$ $F_{16} + 1 = (10)_{16}$

OCTAL System

$$\begin{array}{r} 7 \\ + \\ 1 \\ \hline \cancel{8} \end{array}$$

illegal octal digit

$\Downarrow$

$10 = 0 \times 8^0 + 1 \times 8^1$

HEX System

$$\begin{array}{r} F \\ + \\ 1 \\ \hline (16)_{10} \end{array}$$

$\Downarrow$ convert to HEX

$(10)_{16} = 0 \times 16^0 + 1 \times 16^1$

- Adding 1 to the largest digit in any number system always has a result of (10) in that number system.

Important Properties – cont'd

- What is the largest value using 3 digits in Radix r?

  In binary: $(111)_2 = 2^3 - 1$

  In octal: $(777)_8 = 8^3 - 1$

  In decimal: $(999)_{10} = 10^3 - 1$

  In Radix r:

  largest value = $r^3 - 1$

- How many possible values can be represented …

Using n binary digits?	2^n values: 0 to $2^n - 1$
Using n octal digits	8^n values: 0 to $8^n - 1$
Using n decimal digits?	10^n values: 0 to $10^n - 1$
Using n hexadecimal digits	16^n values: 0 to $16^n - 1$
Using n digits in Radix r ?	r^n values: 0 to $r^n - 1$

- Total number of values (patterns) representable in n digits with radix r?

$$r^n$$

Important Properties of Fractions

- How many fractional values exist with m fraction bits?

  2^m fractions, because each fraction bit can be 0 or 1

- What is the largest fraction value if m bits are used?

  Largest fraction value = $2^{-1} + 2^{-2} + \ldots + 2^{-m} = 1 - 2^{-m}$

  Because if you add 2^{-m} to largest fraction you obtain 1

- In general, what is the largest fraction value if m fraction digits are used in radix r?

  Largest fraction value = $(r-1)(r^{-1} + r^{-2} + \ldots + r^{-m}) = 1 - r^{-m}$

  For decimal, largest fraction value = $1 - 10^{-m}$

  For hexadecimal, largest fraction value = $1 - 16^{-m}$

What is the largest value that can be expressed in n integral digits and m fractional digits? $r^n - r^{-m}$

Summary of Number Systems Properties

	Decimal 10	**Octal** 8	**Binary** 2	**Hexadecimal** 16	**General** r
Allowed Digits	{0-9}	{0-7}	{0-1}	{0-9, A-F}	{ 0 - R} where R = (r-1)
Value of $a_{n-1}\ldots a_2 a_1 a_0 . a_{-1}a_{-2}\ldots a_{-m}$	$a_{n-1}x10^{n-1} + a_{n-2}x10^{n-2} + \ldots + a_2x10^2 + a_1x10^1 + a_0 \times 10^0 + a_{-1} \times 10^{-1} + a_{-2} x10^{-2} + \ldots + a_{-m}x10^{-m}$ $a_i \in \{0\text{-}9\}$ $i=-m,\ldots, 0, 1, \ldots, n-1$	$a_{n-1}8^{n-1}+\ldots+ a_2 8^2+a_1 8^1+a_0 8^0+a_{-1}8^{-1}+ a_{-2}8^{-2}+\ldots+a_{-m}8^{-m}$ $a_i \in \{0\text{-}7\}$	$a_{n-1}2^{n-1}+\ldots+ a_2 2^2+a_1 2^1+a_0 2^0+a_{-1}2^{-1}+ a_{-2}2^{-2}+\ldots+a_{-m}2^{-m}$ $a_i \in \{0,1\}$		$a_{n-1}r^{n-1}+\ldots+ a_2 r^2 + a_1 r^1 + a_0 r^0 + a_{-1}r^{-1} + a_{-2}r^{-2} +\ldots+a_{-m}r^{-m}$ $a_i \in \{0 - (r-1)\}$
Smallest n-digit number	000.....0	000.....0	000.....0	000.....0	000.....0
Largest n-digit number	999.....9 = 10^n-1	77.....7 = 8^n-1	11.....1 = 2^n-1	FF.....F = 16^n-1	RR.....R = r^n-1
Range of n-digit integers	$0 - (10^n-1)$	$0- (8^n-1)$	$0- (2^n-1)$	$0- (16^n -1)$	$0 - (r^n-1)$
# of Possible Combinations of n-digits	10^n	8^n	2^n	16^n	r^n
Max Value of m Fractional Digits	$1\text{-}10^{-m}$	$1\text{-}8^{-m}$	$1\text{-}2^{-m}$	$1\text{-}16^{-m}$	$1\text{-}r^{-m}$

Lecture 3

- Today's Topics
  - Definitions: bit, nibble, byte, word
  - Positional Numbering Systems
  - Converting Between Bases
    - Converting Integer Numbers
    - Converting Fractional Numbers

SUBTRACTION METHOD

- **Suppose we want to convert the decimal number 190 to base 3.**
  - We know that $3^5 = 243$ so our result will be less than six digits wide. The largest power of 3 that we need is therefore $3^4 = 81$, and $81 \times 2 = 162$.
  - Write down the 2 and subtract 162 from 190, giving 28.

$$\begin{array}{r} 190 \\ -\ 162 \\ \hline 28 \end{array} = 3^4 \times 2$$

SUBTRACTION METHOD

- **Converting 190 to base 3...**
  - The next power of 3 is $3^3 = 27$. We'll need one of these, so we subtract 27 and write down the numeral 1 in our result.
  - The next power of 3, $3^2 = 9$, is too large, but we have to assign a placeholder of zero and carry down the 1.

$$\begin{array}{r} 190 \\ -\ 162 \\ \hline 28 \\ -\ 27 \\ \hline 1 \\ -\ 0 \\ \hline 1 \end{array} \begin{array}{l} \\ = 3^4 \times 2 \\ \\ = 3^3 \times 1 \\ \\ = 3^2 \times 0 \\ \\ \end{array}$$

SUBTRACTION METHOD

- **Converting 190 to base 3...**
  - $3^1 = 3$ is again too large, so we assign a zero placeholder.
  - The last power of 3, $3^0 = 1$, is our last choice, and it gives us a difference of zero.
  - Our result, reading from top to bottom is:

    $190_{10} = 21001_3$

$$
\begin{array}{rl}
190 & \\
-\ 162 & = 3^4 \times 2 \\
\hline
28 & \\
-\ 27 & = 3^3 \times 1 \\
\hline
1 & \\
-\ 0 & = 3^2 \times 0 \\
\hline
1 & \\
-\ 0 & = 3^1 \times 0 \\
\hline
1 & \\
-\ 1 & = 3^0 \times 1 \\
\hline
0 &
\end{array}
$$

CONVERTING UNSIGNED WHOLE NUMBERS

- Another method of converting integers from decimal to some other radix uses division.
- This method is mechanical and easy.
- It employs the idea that successive division by a base is equivalent to successive subtraction by powers of the base.
- Let's use the division remainder method to again convert 190 in decimal to base 3.

DIVISON REMAINDER METHOD

- **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
```

DIVISON REMAINDER METHOD

- **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
```

DIVISON REMAINDER METHOD

- **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
```

EXERCISE

- Convert decimal number 190 to base 2.

CONVERTING FRACTIONS

- Fractional decimal values have nonzero digits to the right of the decimal point.
- Fractional values of other radix systems have nonzero digits to the right of the radix point.
- Numerals to the right of a radix point represent negative powers of the radix:

$$
\begin{aligned}
0.47_{10} &= 4 \times 10^{-1} + 7 \times 10^{-2} \\
0.11_2 &= 1 \times 2^{-1} + 1 \times 2^{-2} \\
&= \tfrac{1}{2} + \tfrac{1}{4} \\
&= 0.5 + 0.25 = 0.75
\end{aligned}
$$

CONVERTING FRACTIONS

- As with whole-number conversions, you can use either of two methods:
  - subtraction method or
  - multiplication method.
- The subtraction method for fractions is identical to the subtraction method for whole numbers. Instead of subtracting positive powers of the target radix, we subtract negative powers of the radix.
- We always start with the largest value first, r -1, where r is our radix, and work our way along using larger negative exponents.

SUBTRACTION METHOD

- **Suppose we want to convert the decimal 0.8125 to base 2.**
  - Our result, reading from <u>top to bottom</u> is:

    $0.8125_{10} = 0.1101_2$
  - Of course, this method works with any base, not just binary.

$$
\begin{array}{rl}
0.8125 & \\
-\ 0.5000 & = 2^{-1} \times 1 \\
\hline
0.3125 & \\
-\ 0.2500 & = 2^{-2} \times 1 \\
\hline
0.0625 & \\
-\qquad 0 & = 2^{-3} \times 0 \\
\hline
0.0625 & \\
-\ 0.0625 & = 2^{-4} \times 1 \\
\hline
0 &
\end{array}
$$

MULTIPLICATION METHOD

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

$$
\begin{array}{r}
.8125 \\
\times\ \ 2 \\
\hline
1.6250
\end{array}
\qquad
\begin{array}{r}
.6250 \\
\times\ \ 2 \\
\hline
1.2500
\end{array}
\qquad
\begin{array}{r}
.2500 \\
\times\ \ 2 \\
\hline
0.5000
\end{array}
\qquad
\begin{array}{r}
.5000 \\
\times\ \ 2 \\
\hline
1.0000
\end{array}
$$

Converting Integer Numbers

$$X_B = \overbrace{(b_{n-1}\ldots\ldots\ldots b_2\, b_1\, b_0)_B}^{\textit{Knowns}} \Rightarrow \underbrace{(a_{m-1}\ldots\ldots\ldots a_2\, a_1\, a_0)_A}_{\textit{Unknowns}}$$

$$X_B = \underbrace{a_{m-1}*A^{m-1}+\ldots\ldots+ a_2*A^2+ a_1*A^1}_{\text{Divisible by A}} + \underbrace{a_0*A^0}_{\text{Not Divisible by A}}$$

Dividing X_B by A, the remainder will be a_0.

$$X_B = Q_0.A+a_0$$

$$\text{Where, } Q_0 = \underbrace{a_{m-1}*A^{m-2} +\ldots+ a_2*A^1}_{\text{Divisible by A}} + \underbrace{a_1*A^0}_{\text{Not Divisible by A}}$$

Converting Integer Numbers

$$X_B = a_{m-1}*A^{m-1}+\ldots\ldots+ a_2*A^2+ a_1*A^1 + a_0*A^0$$

$$Q_0 = Q_1A + a_0$$

$$Q_1 = Q_2A + a_1$$

$$\vdots$$

$$Q_{m-1} = Q_mA + a_{m-1}$$

where $Q_m = 0$

- This division procedure can be used for number base conversion.

Decimal-To-Binary Conversion - 1

- Repeated Division-by-2 Method
  - Repeatedly divide the decimal integer by 2
- Each remainder is a binary digit in the translated value
- Example: Convert 37_{10} to Binary

Division	Quotient	Remainder
37 / 2	18	1
18 / 2	9	0
9 / 2	4	1
4 / 2	2	0
2 / 2	1	0
1 / 2	0	1

least significant bit

$37 = (100101)_2$

most significant bit

stop when quotient is zero

Decimal to Binary Conversion - 2

- $N = (d_{n-1} \times 2^{n-1}) + \ldots + (d_1 \times 2^1) + (d_0 \times 2^0)$
- Dividing N by 2 we first obtain
  - $\text{Quotient}_1 = (d_{n-1} \times 2^{n-2}) + \ldots + (d_2 \times 2) + d_1$
  - $\text{Remainder}_1 = d_0$
  - Therefore, first remainder is least significant bit of binary number
- Dividing first quotient by 2 we first obtain
  - $\text{Quotient}_2 = (d_{n-1} \times 2^{n-3}) + \ldots + (d_3 \times 2) + d_2$
  - $\text{Remainder}_2 = d_1$
- Repeat dividing quotient by 2
  - Stop when new quotient is equal to zero
  - Remainders are the bits from least to most significant bit

Exercise

- Convert $(45)_{10}$ to binary using repeated division-by-2 method.

Decimal-To-Binary Integer Conversion - 3

- Sum-of-Weights Methods
  - Determine the set of binary weights whose sum is equal to the decimal number.
  - Place 1s in the appropriate weight positions.

$9 = 8 + 1 = 2^3 + 2^0$			
2^3	2^2	2^1	2^0
1	**0**	**0**	**1**

- Exercise: $(58)_{10} = (\quad ? \quad)_2$

Decimal-to-Hexadecimal Conversion

- Repeatedly divide the decimal integer by 16
- Each remainder is a hex digit in the translated value
- Example: convert 422 to hexadecimal

Division	Quotient	Remainder
422 / 16	26	6
26 / 16	1	A
1 / 16	0	1

least significant digit

most significant digit

stop when quotient is zero

$422 = (1A6)_{16}$

- To convert decimal to octal divide by 8 instead of 16

Decimal-to-Octal Conversion

- Repeatedly divide the decimal integer by 8
- Each remainder is a hex digit in the translated value
- Example: convert 359 to octal

Division	Quotient	Remainder
359/8	44	7
44/8	5	4
5/8	0	5

least significant digit

most significant digit

stop when quotient is zero

$359 = (547)_8$

Converting Fractions

$$X_B = \overbrace{(0.b_{-1}\, b_{-2}\, b_{-3} \ldots\ldots b_{-n})_B}^{\textit{Knowns}} \Rightarrow \underbrace{(0.a_{-1}\, a_{-2}\, a_{-3} \ldots\ \ldots a_{-m})_A}_{\textit{Unknowns}}$$

$$X_B = a_{-1}*A^{-1} + a_{-2}*A^{-2} + \ldots\ldots\ldots a_{-m}*A^{-m}$$

Integer → $X_B*A = a_{-1} + X_{B1}$ ← Fraction

Converting Fractions

$$X_B = a_{-1}*A^{-1} + a_{-2}*A^{-2} + \ldots\ldots\ldots a_{-m}*A^{-m}$$

$$X_B*A = a_{-1} + X_{B1}$$

$$X_{B1}*A = a_{-2} + X_{B2}$$

$$\vdots$$

$$X_{Bm-2}*A = a_{-m-1} + X_{Bm-1}$$

$$X_{Bm-1}*A = a_{-m}$$

Converting Fraction to any Radix r

- To convert fraction N to any radix r

  $N_r = (0.d_{-1}\, d_{-2} \ldots d_{-m})_r = d_{-1} \times r^{-1} + d_{-2} \times r^{-2} \ldots + d_{-m} \times r^{-m}$

- Multiply N_r by r to obtain d_{-1}

  $N_r \times r = d_{-1} + d_{-2} \times r^{-1} \ldots + d_{-m} \times r^{-m+1}$

- The integer part is the digit d_{-1} in radix r
- The new fraction is $d_{-2} \times r^{-1} \ldots + d_{-m} \times r^{-m+1}$
- Repeat multiplying the new fractions by r to obtain $d_{-2}\, d_{-3} \ldots$
- Stop when new fraction becomes 0.0 or enough fraction digits are obtained

Converting Decimal Fractions to Binary

- Sum-of-weights

  $$0.625 = 0.5 + 0.125 = 2^{-1} + 2^{-3} = 0.101$$

- Repeated Multiplication by 2
  - Repeatedly multiplying the fractional results of successive multiplications by 2.
  - Stop when new fraction = 0.0, or when enough fraction bits are obtained
  - The carries form the binary number.

Converting Decimal Fraction to Binary - 2

- Repeated Multiplication by 2 example
  - Convert $N = 0.6875$ to Radix 2

Multiplication	New Fraction	Bit	
$0.6875 \times 2 = 1.375$	0.375	1	→ First fraction bit
$0.375 \times 2 = 0.75$	0.75	0	
$0.75 \times 2 = 1.5$	0.5	1	
$0.5 \times 2 = 1.0$	0.0	1	→ Last fraction bit

- Therefore, $N = 0.6875 = (0.1011)_2$
- Check $(0.1011)_2 = 2^{-1} + 2^{-3} + 2^{-4} = 0.6875$

Converting Decimal Fractions to Octal

Convert $(0.731)_{10} \longrightarrow (?)_8$

$\bullet \longleftarrow$ **Octal Point**

8*0.731 =5.848

8*0.848 = 6.784

8*0.784 = 6.272

8*0.272 = 2.176

$(0.731)_{10} = (0.5662)_8$

Conversion Procedure to Radix *r*

- To convert decimal number *N* (with fraction) to radix *r*
- Convert the Integer Part
  - Repeatedly divide the integer part of number *N* by the radix *r* and save the remainders. The integer digits in radix *r* are the remainders in reverse order of their computation. If radix $r > 10$, then convert all remainders > 10 to digits A, B, … etc.
- Convert the Fractional Part
  - Repeatedly multiply the fraction of *N* by the radix *r* and save the integer digits that result. The fraction digits in radix *r* are the integer digits in order of their computation. If the radix $r > 10$, then convert all digits > 10 to A, B, … etc.
- Join the result together with the radix point

Examples

- Convert $N = 139.6875$ to Octal (Radix 8)
- Solution: $N = 139 + 0.6875$ (split integer from fraction)
- The integer and fraction parts are converted separately

Division	Quotient	Remainder
139 / 8	17	3
17 / 8	2	1
2 / 8	0	2

Multiplication	New Fraction	Digit
$0.6875 \times 8 = 5.5$	0.5	5
$0.5 \times 8 = 4.0$	0.0	4

- Therefore, $139 = (213)_8$ and $0.6875 = (0.54)_8$
- Now, join the integer and fraction parts with radix point
  $N = 139.6875 = (213.54)_8$

Binary-to-Octal/Hex Conversion

- Binary, Octal, and Hexadecimal are related:

  Radix 16 = 2^4 and Radix 8 = 2^3

- Hexadecimal digit = 4 bits and Octal digit = 3 bits

- Starting from least-significant bit, group each 4 bits into a hex digit or each 3 bits into an octal digit

- Example: Convert 32-bit number into octal and hex

3	5	3	0	5	5	2	3	6	2	4	Octal - use 3 bits
1110	1011	0001	0110	1010	0111	1001	0100				32-bit binary
E	B	1	6	A	7	9	4				Hexadecimal - use 4 bits

3-bits = 0 – 7
4-bits = 0 – 15

Octal/Hex-to-Binary Conversion

- Hex-to-Binary Conversion
  - Replace each hexadecimal symbol with the appropriate four bits.
- Octal-to-Binary Conversion
  - Replace each octal symbol with the appropriate three bits.

10A4

$1\times16^3 + 0\times16^2 + A\times16^1 + 4\times16^0$

- Example:
  - $(10A4)_{16} = (0001\ 0000\ 1010\ 0100)_2$
  - $(7526)_8 = (111\ 101\ 010\ 110)_2$

Simplified Conversions

- Converting fractions between Binary, Octal, and Hexadecimal can be simplified
- Starting at the radix pointing, the integer part is converted from right to left and the fractional part is converted from left to right
- Group 4 bits into a hex digit or 3 bits into an octal digit

← integer: right to left — — fraction: left to right →

7	2	6	1	3	.	2	4	7	4	5	2	Octal
111	010	110	001	011	.	010	100	111	100	101	010	Binary
7	5	8	B		.	5	3	C	A		8	Hexadecimal

000

4bits

- Use binary to convert between octal and hexadecimal

Binary Addition Rules

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0, with a carry bit to the next more significant bit, Sum = 0 , carry = 1

$$1 + 1 = \begin{array}{r} {}^{1} \\ +1 \\ \hline 10 \end{array}$$

carry one

Binary Subtraction Rules

- 0 - 0 = 0
- 1 - 0 = 1
- 1 - 1 = 0
- 0 - 1 = 1, with a borrow bit from the next more significant bit

$$\begin{array}{r} 0 \\ -1 \\ \hline ? \end{array} \rightarrow \text{borrow } 2 \rightarrow \begin{array}{r} 2 \\ -1 \\ \hline \boxed{1} \end{array}$$

-1

Borrow

Single Bit Binary Addition

- Given two binary digits (X and Y) and a carry in, we get the sum (S) and the carry out (C)

Carry in = 0

X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

Carry in = 1

1	1	1	1	1
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

Multiple Bit Binary Addition

- Start with the least significant bit (rightmost bit)
- Add each pair of bits
- Include the carry in the addition, if present

carry		1	1	1	1				
	0	0	1	1	0	1	1	0	(54)
+	0	0	0	1	1	1	0	1	(29)
	0	1	0	1	0	0	1	1	(83)
bit position:	7	6	5	4	3	2	1	0	

Single Bit Binary Subtraction

- Given two binary digits (X and Y), and a borrow in we get the difference (D) and the borrow out (B) shown as -1

Borrow in = 0

X	0	0	1	1
– Y	– 0	– 1	– 0	– 1
B D	0 0	-1 1	0 1	0 0

Borrow in = -1

-1	-1	-1	-1	-1
X	0	0	1	1
– Y	– 0	– 1	– 0	– 1
B D	-1 1	-1 0	0 0	-1 1

Multiple Bit Binary Subtraction

- Start with the least significant bit (rightmost bit)
- Subtract each pair of bits
- Include the borrow in the subtraction, if present

borrow			-1	-1			-1		
	0	0	1	1	0	1	1	0	(54)
–	0	0	0	1	1	1	0	1	(29)
	0	0	0	1	1	0	0	1	(25)
bit position:	7	6	5	4	3	2	1	0	

-1 Borrow → 0 → need to borrow again

-1

-1

-2

-1

1

Hexadecimal Addition

- Start with the least significant hexadecimal digits
- Let Sum = summation of two hex digits
- If Sum is greater than or equal to 16
  - Sum = Sum – 16 and Carry = 1
- Example:

```
carry:        1 1   1
      1 C 3 7 2 8 6 A
    + 9 3 9 5 E 8 4 B
    -----------------
      A F C D 1 0 B 5
```

A + B = 10 + 11 = 21
Since 21 ≥ 16
Sum = 21 – 16 = 5
Carry = 1

Hexadecimal Subtraction

- Start with the least significant hexadecimal digits
- Let Difference = subtraction of two hex digits
- If Difference is negative
  - Difference = 16 + Difference and Borrow = -1
- Example:

```
borrow  -1   -1     -1
      9 C 3 7 2 8 6 5
    - 1 3 9 5 E 8 4 B
    -----------------
      8 8 A 1 4 0 1 A
```

Since 5 < B, Difference < 0
Difference = 16+5–11 = 10
Borrow = -1

Signed Numbers

- Several ways to represent a signed number
  - Sign-Magnitude
  - 1's complement – flip bits
  - 2's complement – flip bits & add 1
- Divide the range of values into 2 equal parts
  - First part corresponds to the positive numbers (≥ 0)
  - Second part correspond to the negative numbers (< 0)
- The 2's complement representation is widely used
  - Has many advantages over other representations

Sign-Magnitude Representation

Sign Bit	bit n-2	. . .	bit 2	bit 1	bit 0

Magnitude = n – 1 bits

- Independent representation of the sign and magnitude
- Leftmost bit is the sign bit: 0 is positive and 1 is negative
- Using n bits, largest represented magnitude = $2^{n-1} - 1$

Sign-magnitude representation of +45 using 8-bit register

0	0	1	0	1	1	0	1

pos

32 + 8+4 + 1 = +45

Sign-magnitude representation of -45 using 8-bit register

1	0	1	0	1	1	0	1

neg

Signed Magnitude

- Ex. 4-bit signed magnitude
  - 1 bit for sign
  - 3 bits for magnitude
  - $N = 2^{n-1} - 1$

	$+N$	$-N$
0	0000	1000
1	0001	1001
2	0010	1010
3	0011	1011
4	0100	1100
5	0101	1101
6	0110	1110
7	0111	1111

Note: signed magnitude allows two different representations for zero: positive zero and negative zero.

Properties of Sign-Magnitude

- Two representations for zero: +0 and -0
- Symmetric range of represented values:
  For n-bit register, range is from $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$
  For example using 8-bit register, range is -127 to +127
- Hard to implement addition and subtraction
  - Sign and magnitude parts have to processed independently
  - Sign bit should be examined to determine addition or subtraction
    Addition is converted into subtraction when adding numbers of different signs
  - Need a different circuit to perform addition and subtraction
    Increases the cost of the logic circuit

Convert 0.8125 to base 5

$$0.8125 \times 5 = 4.0625 \qquad .0625 \times 5 = 0.3125 \qquad .3125 \times 5 = 1.5625$$

$$(401)_5$$

Adding binary

Remember 4 rules

$0+0=0$, $0+1=1$, $1+0=1$, $1+1=10$

One's compliment - flip the bits

$$\begin{array}{r} 00000001 \ (+1) \\ +\ 11111110 \ (-1) \\ \hline = 11111111 \end{array}$$

now add 1 $\rightarrow$ 00000000 = (0) answer

Two's compliment - flip the bits & add 1

Starting value = 00100100 = +36

flip bits = 11011011

add one = + 1

Sum = 2's compliment representation = 11011100 = -36

One's Complement Representation

no subtraction ALU needed

- Positive numbers
  - Signed value = Unsigned value
- Negative numbers
  - Signed value = Unsigned value – (2^n-1)
  - n = number of bits
- Negative weight for MSB
  - Another way to obtain the signed value is to assign a negative weight to MSB

1	0	1	1	0	1	0	0
-127	64	32	16	8	4	2	1

= -127 + 32 + 16 + 4 = -75

- Forming the One's Complement
  - Bit flipping for negative numbers
- One's complement range:

$$[-(2^{n-1}-1), 2^{n-1}-1]$$

8-bit Binary value	Unsigned value	Signed value
00000000	0	0
00000001	1	+1
00000010	2	+2
. . .	. . .	. . .
01111110	126	+126
01111111	127	+127
10000000	128	-127
10000001	129	-126
. . .	. . .	. . .
11111110	254	-1
11111111	255	-0

Two's Complement Representation

no subtraction ALU needed

- Positive numbers
  - Signed value = Unsigned value
- Negative numbers
  - Signed value = Unsigned value – 2^n
  - n = number of bits
- Negative weight for MSB
  - Another way to obtain the signed value is to assign a negative weight to MSB

1	0	1	1	0	1	0	0
-128	64	32	16	8	4	2	1

= -128 + 32 + 16 + 4 = -76

- Two's complement range:

$$[-2^{n-1}, 2^{n-1}-1]$$

8-bit Binary value	Unsigned value	Signed value
00000000	0	0
00000001	1	+1
00000010	2	+2
. . .	. . .	. . .
01111110	126	+126
01111111	127	+127
10000000	128	-128
10000001	129	-127
. . .	. . .	. . .
11111110	254	-2
11111111	255	-1

Forming the Two's Complement

starting value	00100100 = +36
step1: reverse the bits (1's complement)	11011011
step 2: add 1 to the value from step 1	+ 1
sum = 2's complement representation	11011100 = -36

Sum of an integer and its 2's complement must be zero:

00100100 + 11011100 = 00000000 (8-bit sum) ⇒ Ignore Carry

Another way to obtain the 2's complement:

Start at the least significant 1
Leave all the 0s to its right unchanged
Complement all the bits to its left

Binary Value

= 00100$\boxed{1}$00 least significant 1

2's Complement

= 11011$\boxed{1}$00 (flip / keep)

find the first 1 appearing right to left

Properties of the 2's Complement

- The 2's complement of N is the negative of N
- The sum of N and 2's complement of N must be zero
  The final carry is ignored
- Consider the 8-bit number $N = 00101100_2 = +44$
  -44 = 2's complement of $N = 11010100_2$
  $00101100_2 + 11010100_2 = \mathbf{1}\ 00000000_2$ (8-bit sum is 0)
  Ignore final carry
- In general: Sum of N + 2's complement of $N = 2^n$
  where 2^n is the final carry (1 followed by n 0's)
- There is only one zero: 2's complement of 0 = 0

Comparison

* Good Table

	Signed Magnitude	1's Complement	2's Complement
No. of 0's	2 (± 0)	2 (± 0)	1 (+ 0)
Symmetric	yes	yes	no
Largest +ive value	$+(2^{n-1}-1)$	$+(2^{n-1}-1)$	$+(2^{n-1}-1)$
Smallest –ive Value	$-(2^{n-1}-1)$	$-(2^{n-1}-1)$	-2^{n-1}

Exercise

Show how the numbers +53 and -53 are represented in 8-bit registers using signed-magnitude, 1's complement and 2's complement representations.

	+53	-53
Signed Magnitude	00110101	10110101
1's Complement	00110101	11001010
2's Complement	00110101	11001011

Exercise: Fill in the following table to indicate what each binary pattern represents using the various formats.

Unsigned integer	4-bit Binary Value	Signed Magnitude	1's Complement (Neg: flip)	2's Complement (Neg)
0	0000	0000	1111	0000
1	0001	0001	1110	1111
2	0010	0010	1101	1110
3	0011	0011	1100	1101
4	0100	0100	1011	1100
5	0101	0101	1010	1011
6	0110	0110	1001	1010
7	0111	0111	1000	1001
8	1000	1000	0111	1000
9	1001	1001	0110	0111
10	1010	1010	0101	0110
11	1011	1011	0100	0101
12	1100	1100	0011	0100
13	1101	1101	0010	0011
14	1110	1110	0001	0010
15	1111	1111	0000	0001

for Neg replace 0 with 1

Signed-Magnitude Arithmetic

Examples

add 01001011_2 to 00101110_2 using signed-magnitude arithmetic.

```
        111
0   1001011
0 + 0101110
0   1111001
```

Using signed magnitude binary arithmetic, find the sum of - 46 and - 25.

```
        11
1   0101110
1 + 0011001
1   1000111
```

Signed-Magnitude Arithmetic

- The sign of the result gets the sign of the number that is larger.
- Example: Using signed magnitude binary arithmetic, find the sum of 46 and - 25.

```
        0 2     0 2
0     0~1011~0
1   + 0011001
0     0010101
```

Note the "borrows" from the second and sixth bits.

One's Complement Arithmetic

- With one's complement addition, the carry bit is "carried around" and added to the sum.
  - Example: Using one's complement binary arithmetic, find the sum of 48 and - 19

```
1,  1 1
    00110000
    11101100 -> 00010011
    --------    actually 19 before flipped
    00011100
          + 1
    --------
    00011101
```

16+8+4+1 = 29

48
-19
29

We note that 19 in binary is `00010011`,
so -19 in one's complement is: `11101100`.

Two's Compliment

Arithmetic Operations with Signed Numbers

- Using the signed number notation with negative numbers in 2's complement form simplifies addition and subtraction of signed numbers.
- Rules for **addition**: Add the two signed numbers. Discard any final carries. The result is in signed form.
- Examples:

00011110 = +30	00001110 = +14	11111111 = –1
00001111 = +15	11101111 = –17	11111000 = –8
00101101 = +45	11111101 = –3	~~1~~ 11110111 = –9

Discard carry

in 2's Compliment

Arithmetic Operations with Signed Numbers

- Rules for **subtraction**: 2's complement the subtrahend and add the numbers. Discard any final carries. The result is in signed form.

change second # into 2's Compliment

- Examples:

00011110 (+30)	00001110 (+14)	11111111 (–1)
– 00001111 –(+15)	– 11101111 –(–17)	– 11111000 –(–8)

Converted to 2's compliment

2's complement subtrahend and add:

00011110 = +30	00001110 = +14	11111111 = –1
11110001 = –15	00010001 = +17	00001000 = +8
~~1~~ 00001111 = +15	00011111 = +31	~~1~~ 00000111 = +7

Discard carry

Discard carry

Overflow

when you exceed the word size = overflow

- If the number of bits required for the answer is exceeded, overflow will occur.
- An overflow can occur only when both numbers are positive or both numbers are negative.
- The overflow will be indicated by an incorrect sign bit.

01000000 = +128	10000001 = –127
01000001 = +129	10000001 = –127
10000001 = ~~–126~~	Discard carry → ~~1~~00000010 = ~~+2~~

Wrong! The answer is incorrect and the sign bit has changed.

Overflow Detection in Signed Numbers

- Rule for detecting an overflow condition in signed numbers: When the "carry in" and the "carry out" of the sign bit differ, overflow has occurred.

Carry 0 Carry 1

		0	1	1	1					⇐ carries
Discard last			0	1	1	1	1	1	1	0	(126)
carry.	+		0	0	0	0	1	0	0	0	+(8)
			1	0	0	0	0	1	1	0	(–122???)

Carry and Overflow

- Carry is important when …
  - Adding or subtracting unsigned integers
  - Indicates that the unsigned sum is out of range
  - Either < 0 or >maximum unsigned *n*-bit value
- Overflow is important when …
  - Adding or subtracting signed integers
  - Indicates that the signed sum is out of range
- Overflow occurs when
  - Adding two positive numbers and the sum is negative
  - Adding two negative numbers and the sum is positive
  - Can happen because of the fixed number of sum bits

Carry and Overflow Examples

- We can have carry without overflow and vice-versa
- Four cases are possible (Examples are 8-bit numbers)

				1					
	0	0	0	0	1	1	1	1	15
+	0	0	0	0	1	0	0	0	8
	0	0	0	1	0	1	1	1	23

Carry = 0 Overflow = 0

1	1	1	1	1					
	0	0	0	0	1	1	1	1	15
+	1	1	1	1	1	0	0	0	248 (-8)
	0	0	0	0	0	1	1	1	7

Carry = 1 Overflow = 0

0	1								
	0	1	0	0	1	1	1	1	79
+	0	1	0	0	0	0	0	0	64
	1	0	0	0	1	1	1	1	143 (-113)

Carry = 0 Overflow = 1

1	0		1	1					
	1	1	0	1	1	0	1	0	218 (-38)
+	1	0	0	1	1	1	0	1	157 (-99)
	0	1	1	1	0	1	1	1	119

Carry = 1 Overflow = 1

Binary Multiplication and Division Using Shifting

- We can do binary multiplication and division by 2 very easily using an *arithmetic shift* operation
- A *left arithmetic shift* inserts a 0 in for the rightmost bit and shifts everything else left one bit; in effect, it multiplies by 2
- A *right arithmetic shift* shifts everything one bit to the right, but copies the sign bit; it divides by 2

Binary Multiplication and Division Using Shifting

Example:

Multiply the value 11 (expressed using 8-bit signed two's complement representation) by 2. → 2^1 → shift one place

We start with the binary value for 11:

00001011 (+11)

We shift left one place, resulting in:

00010110 (+22) add a zero = shift left

The sign bit has not changed, so the value is valid.

To multiply 11 by 4, we simply perform a left shift twice.

→ 2^2 → shift two places

Ex: multiply by 8 = 2^3 → shift 3 places

Binary Multiplication and Division Using Shifting

Example:

Multiply the value 66 (expressed using 8-bit signed two's complement representation) by 2.

We start with the binary value for 66:

01000010 (+66)

We shift left one place, resulting in:

10000100 (-4)

= overflow

The sign bit has changed, so overflow has occurred.

Binary Multiplication and Division Using Shifting

Example:

Divide the value 12 (expressed using 8-bit signed two's complement representation) by 2.

We start with the binary value for 12:

00001100 (+12)

We shift right one place, resulting in:

add a zero → 00000110 (+6)

(Remember, we carry the sign bit to the left as we shift.)

To divide 12 by 4, we right shift twice.

Binary Multiplication and Division Using Shifting

Example:

Divide the value -14 (expressed using 8-bit signed two's complement representation) by 2.

We start with the two's complement representation for -14:

11110010 (-14)

We shift right one place, resulting in:

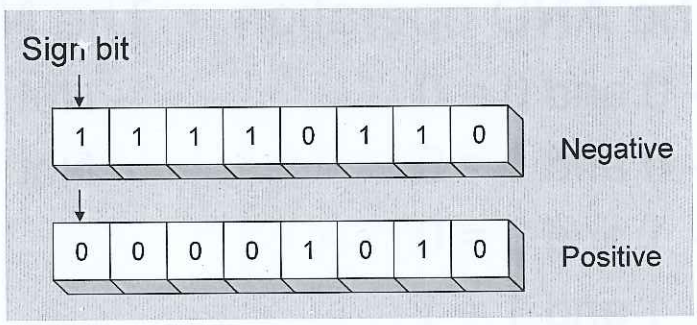
add a one → 11111001 (-7)

(Remember, we carry the sign bit to the left as we shift.)

Consider the fact that this is only truncated to 8-bit format. The extra one is coming from a neighboring 8-bit in a 16-bit or higher configuration

Sign Bit

- Highest bit indicates the sign
- 1 = negative
- 0 = positive



For Hexadecimal Numbers, check most significant digit

If highest digit is > 7, then value is negative

Examples: 8A and C5 are negative bytes

B1C42A00 is a negative word (32-bit signed integer)

Sign Extension

Step 1: Move the number into the lower-significant bits

Step 2: Fill all the remaining higher bits with the sign bit

- This will ensure that both magnitude and sign are correct
- Examples
  - Sign-Extend 10110011 to 16 bits

    `10110011 = -77` ⇨ `11111111 10110011 = -77`
  - Sign-Extend 01100010 to 16 bits

    `01100010 = +98` ⇨ `00000000 01100010 = +98`
- Infinite 0s can be added to the left of a positive number
- Infinite 1s can be added to the left of a negative number

Logic Operations on Bits

- The AND operation
  - 0 and 0 = 0
  - 0 and 1 = 0
  - 1 and 0 = 0
  - 1 and 1 = 1
- The NOT operation
  - Not 0 = 1
  - Not 1 = 0
- The OR operation
  - 0 or 0 = 0
  - 0 or 1 = 1
  - 1 or 0 = 1
  - 1 or 1 = 1
- The XOR operation
  - 0 xor 0 = 0
  - 0 xor 1 = 1
  - 1 xor 0 = 1
  - 1 xor 1 = 0