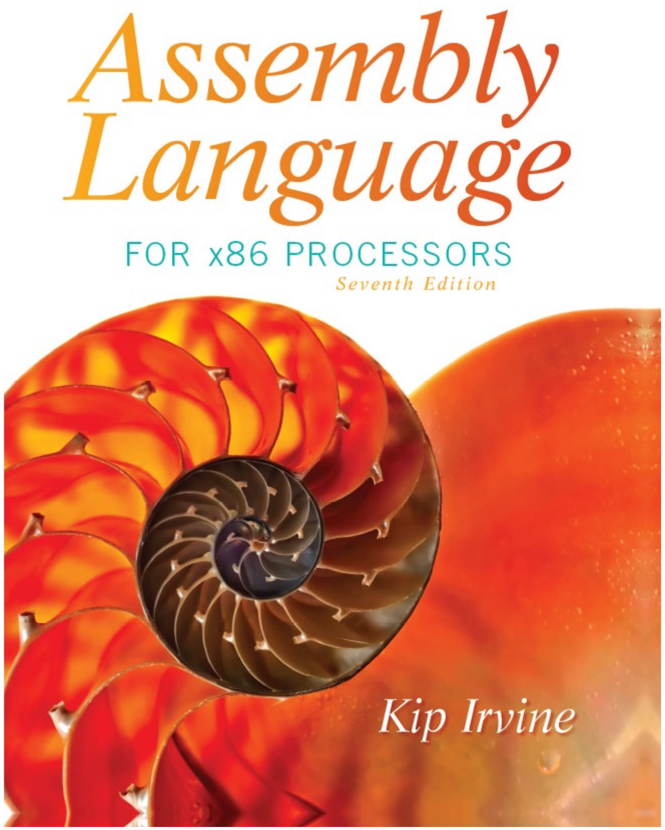
Assembly Language for x86 Processors

Seventh Edition



Chapter 1

Basic Concepts

Chapter Overview

- Virtual Machine Concept
- Boolean Operations
- Data Representation

Virtual Machines

- Tanenbaum: Virtual machine concept
- Programming Language analogy:
  - Each computer has a native machine language (language L0) that runs directly on its hardware
  - A more human-friendly language is usually constructed above machine language, called Language L1
- Programs written in L1 can run two different ways:
  - Interpretation - L0 program interprets and executes L1 instructions one by one
  - Translation - L1 program is completely translated into an L0 program, which then runs on the computer hardware

Translating Languages

English: Display the sum of A times B plus C.

C++: cout << (A * B + C);

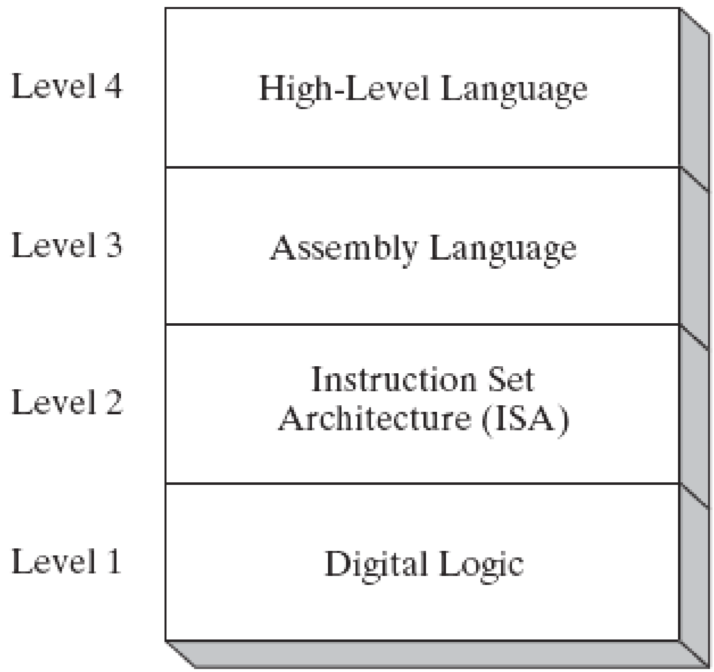
Assembly Language:

```
mov eax,A
mul B
add eax,C
call WriteInt
```

Intel Machine Language:

```
A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000
```

Specific Machine Levels



High-Level Language

- Level 4
- Application-oriented languages
  - C++, Java, Pascal, Visual Basic . . .
- Programs compile into assembly language (Level 4)

Assembly Language

- Level 3
- Instruction mnemonics that have a one-to-one correspondence to machine language
- Programs are translated into Instruction Set Architecture Level - machine language (Level 2)

Instruction Set Architecture (ISA)

- Level 2
- Also known as conventional machine language
- Executed by Level 1 (Digital Logic)

Digital Logic

- Level 1
- CPU, constructed from digital logic gates
- System bus
- Memory
- Implemented using bipolar transistors

Boolean Operations

- NOT
- AND
- OR
- Operator Precedence
- Truth Tables

Boolean Algebra

- Based on symbolic logic, designed by George Boole
- Boolean expressions created from:
  - NOT, AND, OR

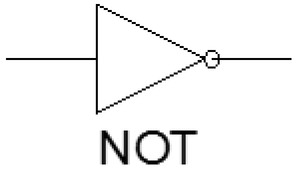
Expression	Description
$\neg X$	NOT X
$X \wedge Y$	X AND Y
$X \vee Y$	X OR Y
$\neg X \vee Y$	(NOT X) OR Y
$\neg(X \wedge Y)$	NOT (X AND Y)
$X \wedge \neg Y$	X AND (NOT Y)

NOT

- Inverts (reverses) a boolean value
- Truth table for Boolean NOT operator:

X	¬X
F	T
T	F

Digital gate diagram for NOT:

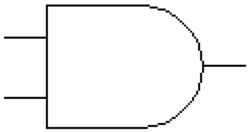


AND

- Truth table for Boolean AND operator:

X	Y	X ∧ Y
F	F	F
F	T	F
T	F	F
T	T	T

Digital gate diagram for AND:

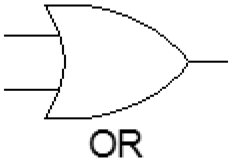


OR

- Truth table for Boolean OR operator:

X	Y	X ∨ Y
F	F	F
F	T	T
T	F	T
T	T	T

Digital gate diagram for OR:



Operator Precedence

- Examples showing the order of operations:

Expression	Order of Operations
$\neg X \vee Y$	NOT, then OR
$\neg (X \vee Y)$	OR, then NOT
$X \vee (Y \wedge Z)$	AND, then OR

Truth Tables (1 of 3)

- A Boolean function has one or more Boolean inputs, and returns a single Boolean output.
- A truth table shows all the inputs and outputs of a Boolean function

Example: $\neg X \vee Y$

X	¬X	Y	¬X ∨ Y
F	T	F	T
F	T	T	T
T	F	F	F
T	F	T	T

Truth Tables (2 of 3)

- Example: $X \wedge \neg Y$

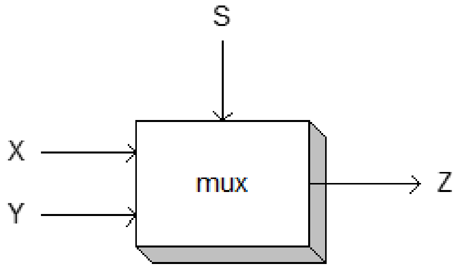
X	Y	$\neg$Y	X $\wedge \neg$Y
F	F	T	F
F	T	F	F
T	F	T	T
T	T	F	F

Pearson

Truth Tables (3 of 3)

- Example: $(Y \wedge S) \vee (X \wedge \neg S)$

X	Y	S	Y ∧ S	¬S	X ∧ ¬S	(Y ∧ S) ∨ (X ∧ ¬S)
F	F	F	F	T	F	F
F	T	F	F	T	F	F
T	F	F	F	T	T	T
T	T	F	F	T	T	T
F	F	T	F	F	F	F
F	T	T	T	F	F	T
T	F	T	F	F	F	F
T	T	T	T	F	F	T



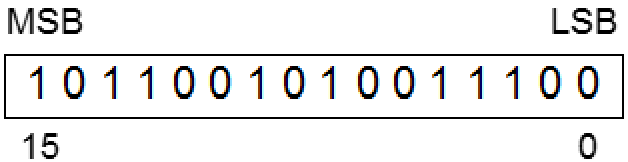
Two-input multiplexer

Data Representation

- Binary Numbers
  - Translating between binary and decimal
- Binary Addition
- Integer Storage Sizes
- Hexadecimal Integers
  - Translating between decimal and hexadecimal
  - Hexadecimal subtraction
- Signed Integers
  - Binary subtraction
- Character Storage

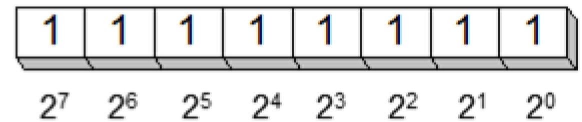
Binary Numbers (1 of 2)

- Digits are 1 and 0
  - 1 = true
  - 0 = false
- MSB -most significant bit
- LSB least significant bit
- Bit numbering:



Binary Numbers (2 of 2)

- Each digit (bit) is either 1 or 0
- Each bit represents a power of 2:



Every binary number is a sum of powers of 2

Table 1-3 Binary Bit Position Values.

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

Translating Binary to Decimal

Weighted positional notation shows how to calculate the decimal value of each binary bit:

$$dec = (D_{n-1} \times 2^{n-1}) + \left(D_{n-2} \times 2^{n-2}\right) + \ldots + \left(D_1 \times 2^1\right) + \left(D_0 \times 2^0\right)$$

Binary 10001001 = ?D (D: decimal)

Integer Storage Sizes

Standard sizes:

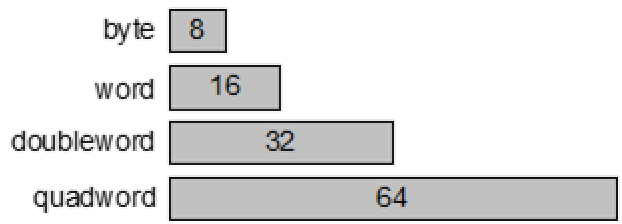


Table 1-4 Ranges of Unsigned Integers.

Storage Type	Range (low–high)	Powers of 2
Unsigned byte	0 to 255	0 to $(2^8 - 1)$
Unsigned word	0 to 65,535	0 to $(2^{16} - 1)$
Unsigned doubleword	0 to 4,294,967,295	0 to $(2^{32} - 1)$
Unsigned quadword	0 to 18,446,744,073,709,551,615	0 to $(2^{64} - 1)$

Hexadecimal Integers

Binary values are represented in hexadecimal.

Table 1-5 Binary, Decimal, and Hexadecimal Equivalents.

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

Powers of 16

Used when calculating hexadecimal values up to 8 digits long:

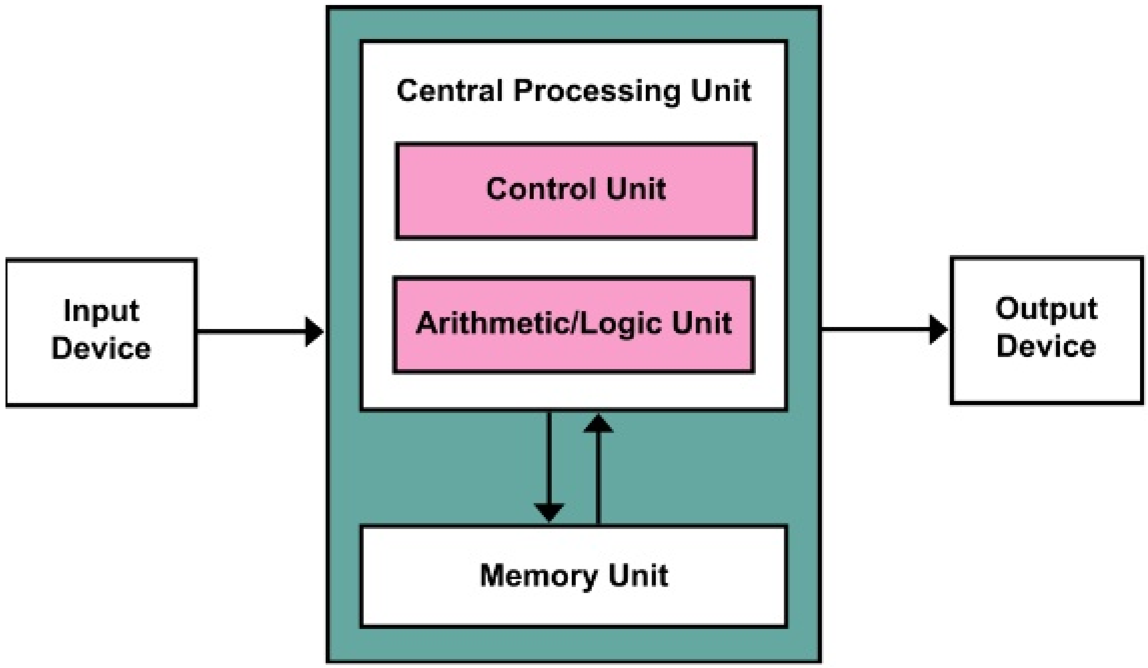
16^n	Decimal Value	16^n	Decimal Value
16^0	1	16^4	65,536
16^1	16	16^5	1,048,576
16^2	256	16^6	16,777,216
16^3	4096	16^7	268,435,456

Characters Recognition

- People can easily recognize human-readable characters
  - +1, +2, -2…….
  - A, a…..
- Unfortunately, in digital circuits there is no provision made to put a “+” or “-” symbol thanks to desperate numbers: “0’s” and “1’s”

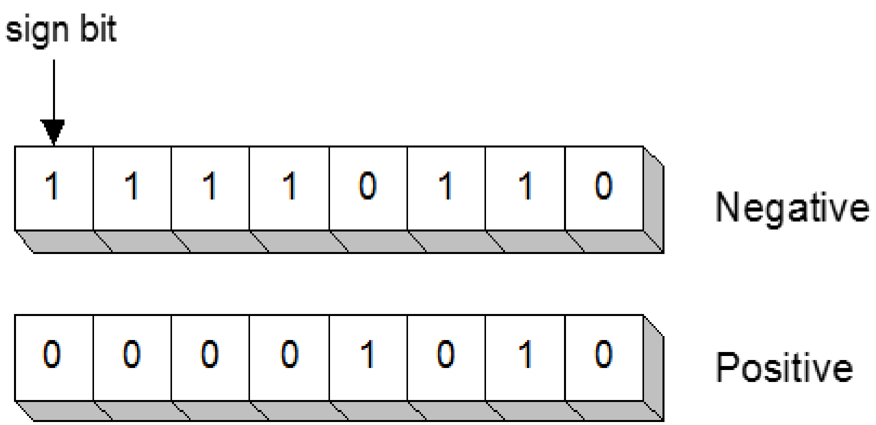
Computer Architecture

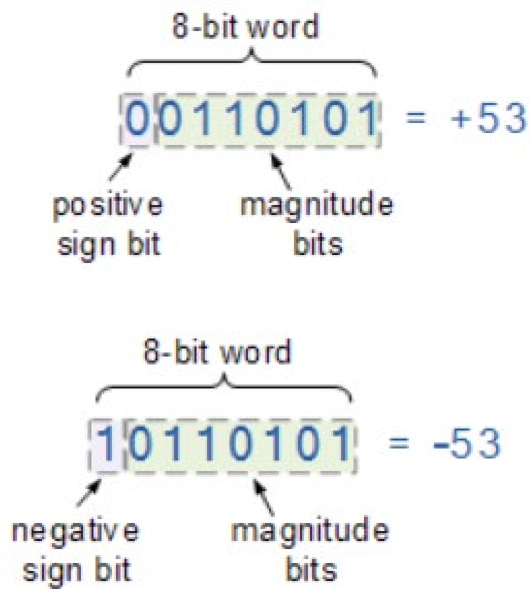
von Neumann Architecture



Signed Binary Integers

The highest bit indicates the sign. 1 = negative, 0 = positive





If the highest digit of a hexadecimal integer is > 7, the value is negative. Examples: 8A, C5, A2, 9D

Pearson

Sign-Magnitude Representation

- Left most digit can be used to indicate the sign
- The remaining digits can be used to indicate the magnitude or absolute value of the number

- Practice (4-bit): (5+5)D, (5-5)D

1's Complement

- Positive numbers: remain unchanged as before with the sign-magnitude numbers
- Negative numbers: keep a sign bit and invert remaining bits
  - $(-2)_D = (1010)_{s\_m} = (1101)_{1's}$
- Again practice (4-bit): (2-2)D, (1-2)D, (2-1)D, (-1-2)D

2’s Complement

- Positive numbers: remain unchanged as before with the sign-magnitude numbers
- Negative numbers: keep a sign bit, invert remaining bits and add 1 to LSB ➔ 1’s + 1
  - $(-2)_D = (1010)_{s\_m} = (1101)_{1's} = (1110)_{2's}$
- Again practice (4-bit): (2-2)D, (1-2)D, (2-1)D, (-1-2)D

Forming the Two's Complement

- Negative numbers are stored in two's complement notation
- Represents the additive Inverse

Starting value	00000001
Step 1: reverse the bits	11111110
Step 2: add 1 to the value from Step 1	11111110 +00000001
Sum: two's complement representation	11111111

Note that 00000001 + 11111111 = 00000000

Binary Subtraction

- When subtracting A-B, convert B to its two's complement
- Add A to (−B)

$$\begin{array}{r} 0\,0\,0\,0\,1\,1\,0\,0 \\ -\;0\,0\,0\,0\,0\,0\,1\,1 \\ \hline \end{array} \longrightarrow \begin{array}{r} 0\,0\,0\,0\,1\,1\,0\,0 \\ 1\,1\,1\,1\,1\,1\,0\,1 \\ \hline 0\,0\,0\,0\,1\,0\,0\,1 \end{array}$$

Practice: Subtract 0101 from 1001.

Ranges of Signed Integers

The highest bit is reserved for the sign. This limits the range:

Storage Type	Range (low–high)	Powers of 2
Signed byte	−128 to +127	-2^{7} to $(2^{7} - 1)$
Signed word	−32,768 to +32,767	-2^{15} to $(2^{15} - 1)$
Signed doubleword	−2,147,483,648 to 2,147,483,647	-2^{31} to $(2^{31} - 1)$
Signed quadword	−9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to $(2^{63} - 1)$

Summary

- Assembly language helps you learn how software is constructed at the lowest levels
- Assembly language has a one-to-one relationship with machine language
- Each layer in a computer's architecture is an abstraction of a machine
  - layers can be hardware or software
- Boolean expressions are essential to the design of computer hardware and software