CSCI 212: Computer Organization & Assembly Language

Palomar College

Chapter 1: Basic Concepts

Based upon the book: Assembly Language for x86 Processors, 7th Edition

Some slides were prepared by the book author, Kip Irvine.

Those slides are covered by the following copyright.

Chapter Overview

- **Welcome to Assembly Language**
- Virtual Machine Concept
- Data Representation
- Boolean Operations

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

2

Welcome to Assembly Language

- Some Good Questions to Ask
- Assembly Language Applications

Questions to Ask

- Why am I learning Assembly Language?
- What background should I have?
- What is an assembler?
- What hardware/software do I need?
- What will I learn?
- How does assembly language (AL) relate to machine language?
- Is AL portable?

Why am I learning Assembly Language?

- Embedded systems software development
- Device driver development
- RT applications such as simulation or H/W monitoring
- Optimizing computer games
- Low level bit manipulation can be easier than with HLL
- Promotes understanding of the interaction between hardware, operating systems and application programs

What background should I have?

- Experience with at least one HLL such as Java, C, Python or C++

- You should know how to use IF statements, arrays and functions to solve programming problems

What Are Assemblers and Linkers?

- An assembler is a utility program that converts source code programs from assembly language into machine language

- A linker is a utility program that combines individual files created by an assembler into machine language

What hardware/software do I need?

- You need a computer that runs a 32-bit or 64-bit version of Microsoft Windows, along with one of the recent versions of Microsoft Visual Studio

What will I learn?

- Basic principles of computer architecture as applied to x86 processors
- Basic boolean logic
- How x86 processors manage memory
- How arithmetic expressions, loops and logical structures are implemented at the machine level
- Data representation, including integers, real numbers and character data
- How to debug programs at the machine level
- How to create assembly language application programs

How does assembly language (AL) relate to machine language?

- Machine language is a numeric language specifically understood by a computer's CPU.

- All x86 processors understand a common machine language.

- Assembly language has a one-to-one relationship with machine language

- Each assembly language instruction corresponds to a single machine-language instruction

Is Assembly Language portable?

- A language whose source programs can be compiled and run on a wide variety of computer systems is said to be portable.

- Assembly language is not portable because it is designed for a specific processor family.

Assembly Language Applications

- Some representative types of applications:
  - Business application for single platform
  - Hardware device driver
  - Business application for multiple platforms
  - Embedded systems & computer games

(see next panel)

Comparing ASM to High-Level Languages

Type of Application	High-Level Languages	Assembly Language
Business application software, written for single platform, medium to large size.	Formal structures make it easy to organize and maintain large sections of code.	Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.
Hardware device driver.	Language may not provide for direct hardware access. Even if it does, awkward coding techniques must often be used, resulting in maintenance difficulties.	Hardware access is straightforward and simple. Easy to maintain when programs are short and well documented.
Business application written for multiple platforms (different operating systems).	Usually very portable. The source code can be recompiled on each target operating system with minimal changes.	Must be recoded separately for each platform, often using an assembler with a different syntax. Difficult to maintain.
Embedded systems and computer games requiring direct hardware access.	Produces too much executable code, and may not run efficiently.	Ideal, because the executable code is small and runs quickly.

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

13

What's Next

- Welcome to Assembly Language
- **Virtual Machine Concept**
- Data Representation
- Boolean Operations

Virtual Machine Concept

- Virtual Machines
- Specific Machine Levels

Virtual Machines

- Tanenbaum: Virtual machine concept
- Programming Language analogy:
  - Each computer has a native machine language that runs directly on its hardware
  - A more human-friendly language is usually constructed above machine language, called Language X

- Programs written in X can run two different ways:
  - Interpretation – Machine language program interprets and executes language X instructions one by one
  - Translation – Language X program is completely translated into a machine language 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

Level	
Level 4	High-Level Language
Level 3	Assembly Language
Level 2	Instruction Set Architecture (ISA)
Level 1	Digital Logic
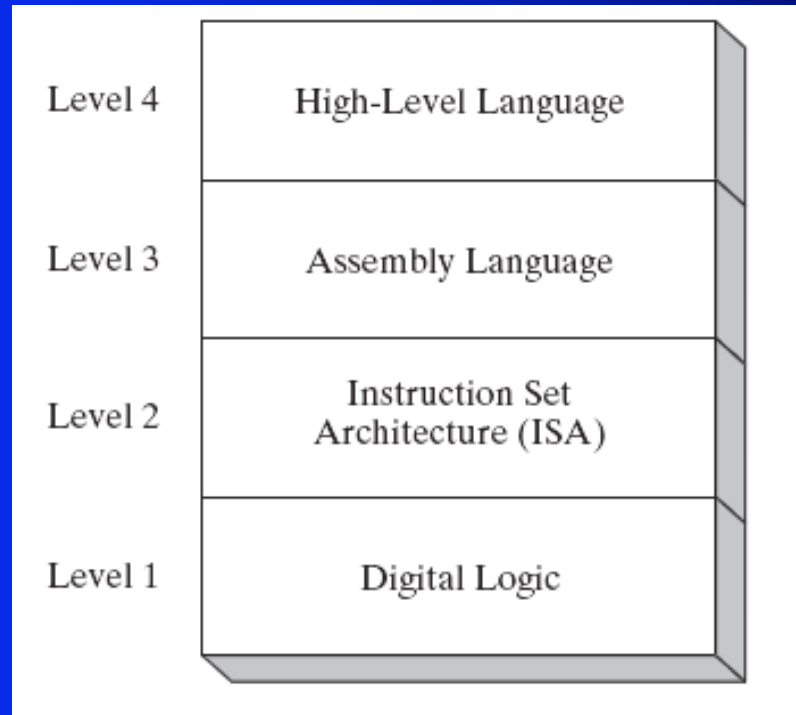
(descriptions of individual levels follow . . .)

High-Level Language

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

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

19

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)

20

Instruction Set Architecture (ISA)

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

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

21

Digital Logic

- Level 1
- CPU, constructed from digital logic gates
- System bus
- Memory

What's Next

- Welcome to Assembly Language
- Virtual Machine Concept
- **Data Representation**
- Boolean Operations

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

23

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

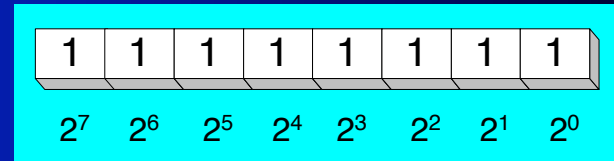- Digits are 1 and 0
    - 1 = true
    - 0 = false
- MSB – most significant bit
- LSB – least significant bit

- Bit numbering:

MSB	LSB
1 0 1 1 0 0 1 0 1 0 0 1 1 1 0 0	
15	0

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

25

Binary Numbers

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

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

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

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

26

Translating Binary to Decimal

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

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

D = binary digit

binary 00001001 = decimal 9:

$$(1 \times 2^3) + (1 \times 2^0) = 9$$

Translating Unsigned Decimal to Binary

- Repeatedly divide the decimal integer by 2. Each remainder is a binary digit in the translated value:

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

37 = 100101

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

Binary Addition

- Starting with the LSB, add each pair of digits, include the carry if present.

carry:					1				
	0	0	0	0	0	1	0	0	(4)
+	0	0	0	0	0	1	1	1	(7)
	0	0	0	0	1	0	1	1	(11)
bit position:	7	6	5	4	3	2	1	0	

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

29

Integer Storage Sizes

Standard sizes:

byte	8
word	16
doubleword	32
quadword	64
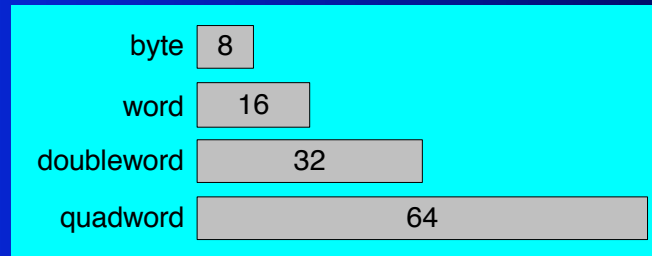
Table 1-4 Ranges of Unsigned Integers.

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 $(2^{64} - 1)$

What is the largest unsigned integer that may be stored in 20 bits?

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

30

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

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

31

Translating Binary to Hexadecimal

- Each hexadecimal digit corresponds to 4 binary bits.

- Example: Translate the binary integer 000101101010011110010100 to hexadecimal:

1	6	A	7	9	4
0001	0110	1010	0111	1001	0100

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

32

Converting Hexadecimal to Decimal

- Multiply each digit by its corresponding power of 16:

  $$dec = (D_3 \times 16^3) + (D_2 \times 16^2) + (D_1 \times 16^1) + (D_0 \times 16^0)$$

- Hex 1234 equals $(1 \times 16^3) + (2 \times 16^2) + (3 \times 16^1) + (4 \times 16^0)$, or decimal 4,660.

- Hex 3BA4 equals $(3 \times 16^3) + (11 * 16^2) + (10 \times 16^1) + (4 \times 16^0)$, or decimal 15,268.

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

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

34

Converting Decimal to Hexadecimal

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

decimal 422 = 1A6 hexadecimal

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

35

Hexadecimal Addition

- Divide the sum of two digits by the number base (16). The quotient becomes the carry value, and the remainder is the sum digit.

		1	1
36	28	28	6A
42	45	58	4B
78	6D	80	B5

21 / 16 = 1, rem 5

Important skill: Programmers frequently add and subtract the addresses of variables and instructions.

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

36

Hexadecimal Subtraction

- When a borrow is required from the digit to the left, add 16 (decimal) to the current digit's value:
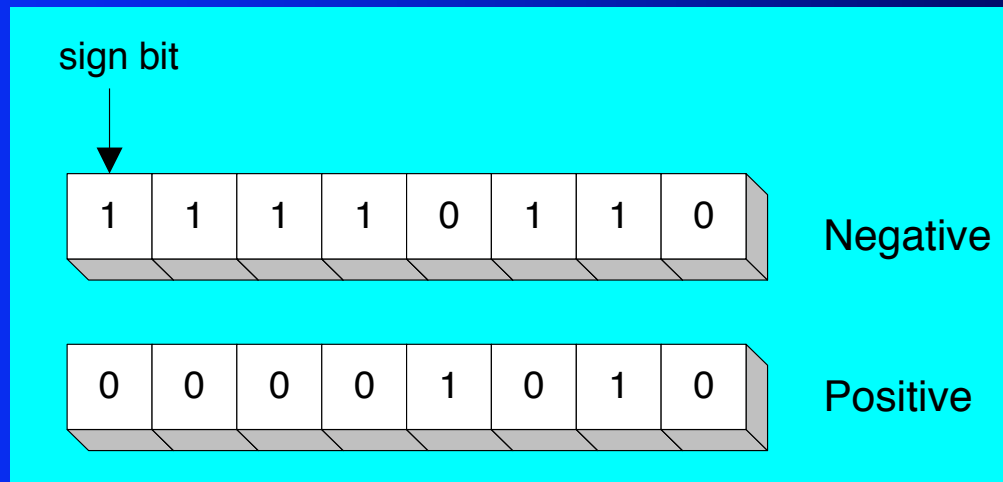
16 + 5 = 21

−1

C6   75
A2   47
―――――――――
24   2E

Practice: The address of **var1** is 00400020. The address of the next variable after var1 is 0040006A. How many bytes are used by var1?

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

37

Signed 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

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)

```
  0 0 0 0 1 1 0 0            0 0 0 0 1 1 0 0
– 0 0 0 0 0 0 1 1    ──►     1 1 1 1 1 1 0 1
  ─────────────────          ─────────────────
                             0 0 0 0 1 0 0 1
```

Practice: Subtract 0101 from 1001.

Learn How To Do the Following:

- Form the two's complement of a hexadecimal integer
- Convert signed binary to decimal
- Convert signed decimal to binary
- Convert signed decimal to hexadecimal
- Convert signed hexadecimal to decimal

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

41

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 $(2^{63} - 1)$

Practice: What is the largest positive value that may be stored in 20 bits?

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

42

Character Storage

- Character sets
  - Standard ASCII (0 – 127)
  - Extended ASCII (0 – 255)
  - ANSI (0 – 255)
  - Unicode (0 – 65,535)
- Null-terminated String
  - Array of characters followed by a *null byte*
- Using the ASCII table
  - back inside cover of book

Numeric Data Representation

- pure binary
  - can be calculated directly
- ASCII binary
  - string of digits: "01010101"
- ASCII decimal
  - string of digits: "65"
- ASCII hexadecimal
  - string of digits: "9C"

next: Boolean Operations

What's Next

- Welcome to Assembly Language
- Virtual Machine Concept
- Data Representation
- **Boolean Operations**

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

45

Boolean Operations

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

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

46

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)

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

47

NOT

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

X	¬X
F	T
T	F

Digital gate diagram for NOT:

NOT

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

48
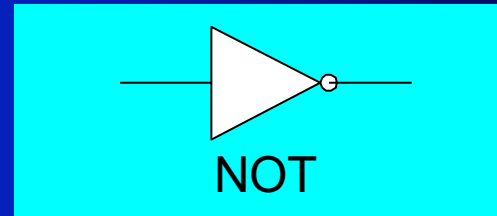
AND

- 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:

AND

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

49
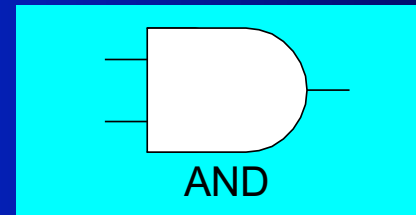
OR

- 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:

OR

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

50
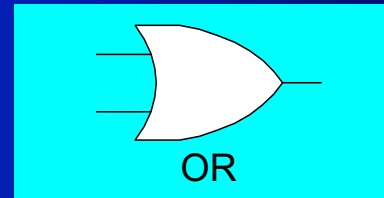
Operator Precedence

- Examples showing the order of operations:

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

- 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: ¬X ∨ Y

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

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

52

- Example: X ∧ ¬Y

X	Y	¬Y	X ∧ ¬Y
F	F	T	F
F	T	F	F
T	F	T	T
T	T	F	F

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

53

- Example: (Y ∧ S) ∨ (X ∧ ¬S)

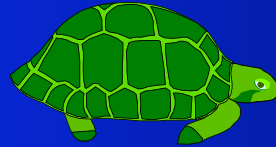| X	Y	S	**Y ∧ S**	**¬S**	**X ∧ ¬S**	**(Y ∧ S) ∨ (X ∧ ¬S)**
F	F	F	F	T	F	F
F	T	F	F	T	F	F
T	F	F	F	T	T	T
T	T	F	F	T	T	T
F	F	T	F	F	F	F
F	T	T	T	F	F	T
T	F	T	F	F	F	F
T	T	T	T	F	F	T
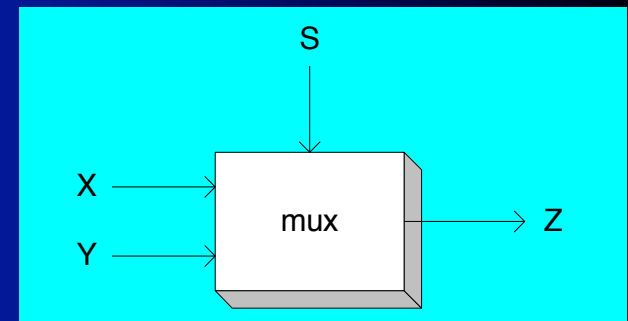
Two-input multiplexer

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

In-class Assignment

- Work on the in-class assignment for about 10-15 minutes.

- Do the work in Blackboard if at all possible.

- You can work together on this.

- We will review the answers in class after you are done. The answers must come from you.

54 68 65 20 45 6E 64

What do these numbers represent?

Irvine, Kip R. Assembly Language for Intel-Based Computers 7/e, 2015.

57