Principles of Assembly and Compilation
汇编与编译原理
44100593

王朝坤
ISE-SS@Tsinghua

教学内容

- 针对软件工程学科对汇编语言与编译原理的培养需求，围绕计算机高级程序设计语言的具体实现过程，《汇编与编译原理》讲解的主要知识点包括：
    - 汇编级机器组织
    - 程序设计语言概述
    - 语言翻译系统
    - …
- 教学目标：
    - 汇编语言程序设计的基本方法
    - 编译器的基本原理及组织
- 为学生掌握设计与实现高级程序设计语言的能力打下良好基础。

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
    - should be converted to programs run in L0.

- How to construct the converters?

Translating Languages

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

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

one to many

Assembly Language:

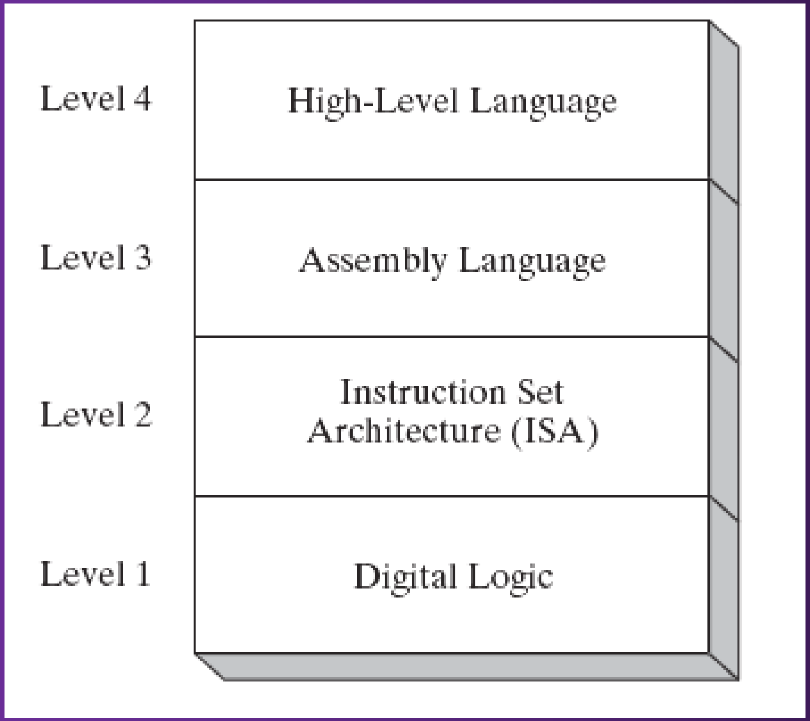
mov eax,A
mul B
add eax,C
call WriteInt

one to one

Intel Machine Language:

A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000

Specific Machine Levels



(descriptions of individual levels follow . . .)

Digital Logic

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

Instruction Set Architecture (ISA)

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

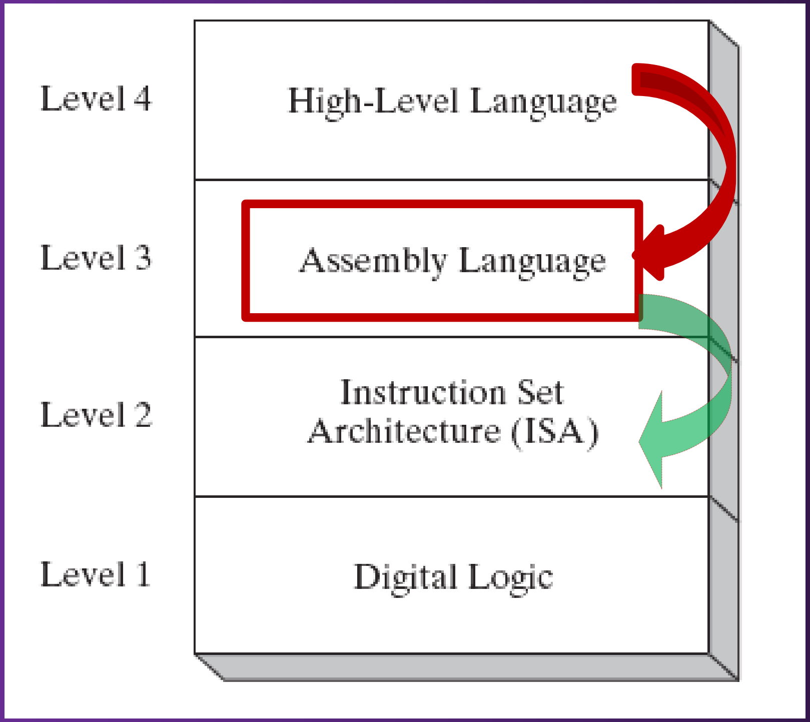
Assembly Language

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

High-Level Language

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

Aim of this Course

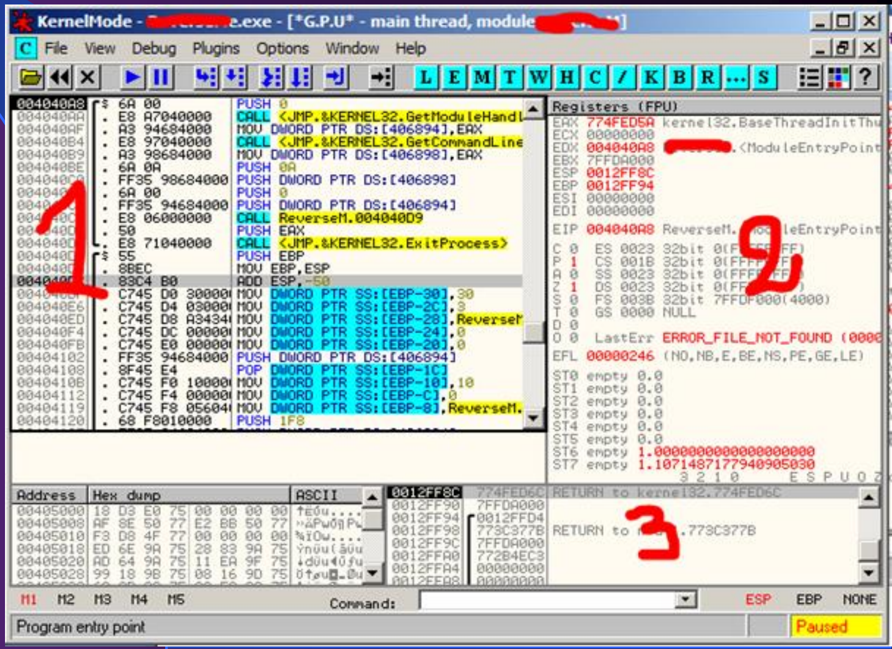


汇编语言程序设计

重点讲述汇编语言程序设计的基本方法。

讲授内容：

1. 汇编语言的基本构成；
2. 汇编程序设计的基本方法；
3. 汇编语言程序设计专题；
4. 开发实例：游戏设计等。



语言翻译系统

重点讲述编译器的设计原理和常用实现技术。

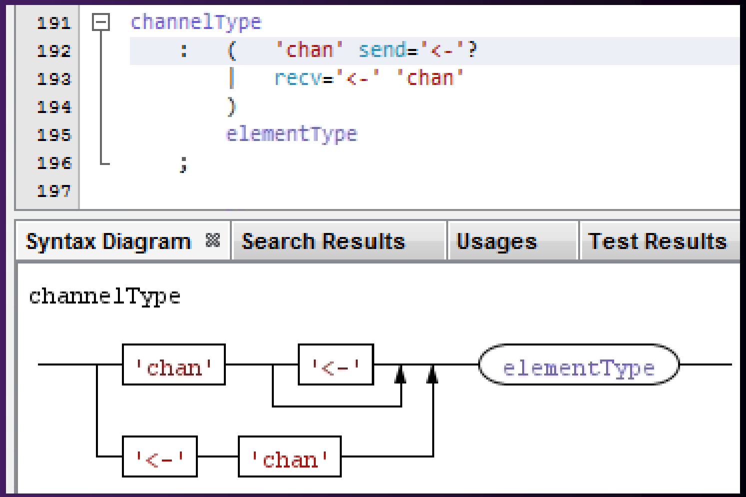
讲授内容：

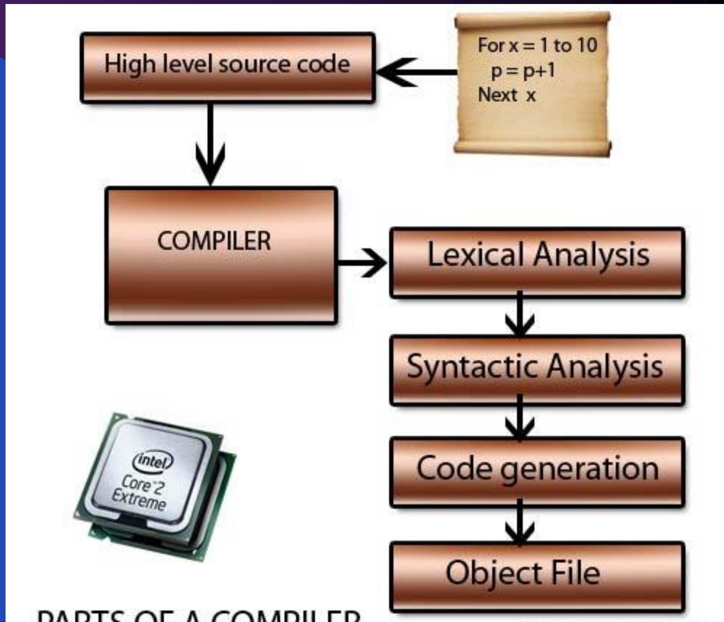
1. 编译器的理论框架；

2. 编译方法及核心算法；

3. 常用编译开发工具；

4. 开发实例：设计与实现一个较为完整的编译器。





Part I: Assembly Language Programming

王朝坤
IISE@Tsinghua

CHAPTER 1: BASIC CONCEPTS

Chapter Overview

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

Questions to Ask

- What is Assembly Language?
  - e.g.~
- Why learn Assembly Language?
  - Game and real-time applications
  - Optimization
  - Reverse Engineering
  - Anti-Virus

  - Device drivers and embedded programming
  - Understanding Hardware
  - Mixed language programming
  - Other courses: OS, DBS…

- What will I learn?

教学目的与重点

- 汇编语言是软件工程/计算机科学与技术专业核心内容。
- 作为理论和实践并重的学习内容，本部分将介绍汇编语言的基本构成（指令、伪指令等）和汇编程序设计的基本方法。具体包括
  - 汇编语言基础、
  - 屏幕与键盘操作、
  - 数据操作、
  - 高级输入／输出、
  - 中断与端口、
  - 运算符与指令、
  - PC指令系统等。

教学目的与重点 (II)

- 通过本部分的学习，学生应
  - 具有使用汇编语言编写程序的能力，
  - 对顺序、分支、循环三大程序结构在汇编语言中的实现方法有较好的掌握，
  - 对模块化程序设计技术有进一步的了解，
  - 了解混合程序设计
- 为深入理解计算机及其编程语言的工作原理打下基础。
- 课程实验中用汇编语言实现若干完整的程序。

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)

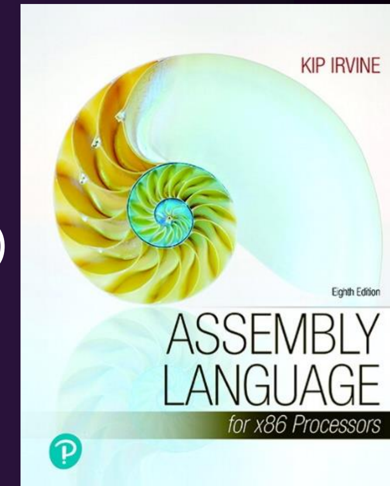
Comparing ASM to High-Level Languages

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.

What's Next

- Welcome to Assembly Language
- **Course Information (for this Part)**
- Virtual Machine Concept
- Data Representation
- Boolean Operations

Textbooks

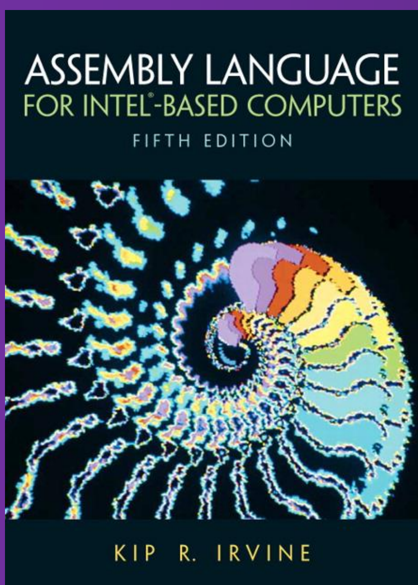


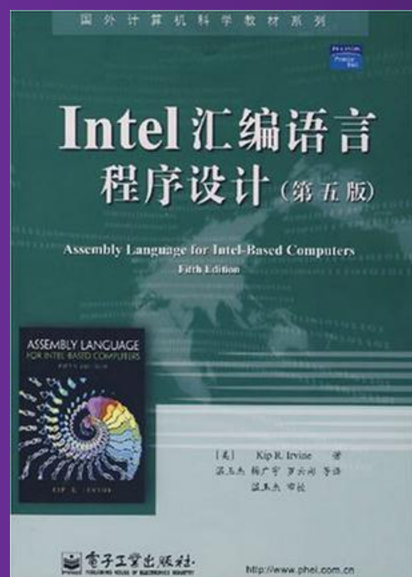
Assembly Language for x86 Processors

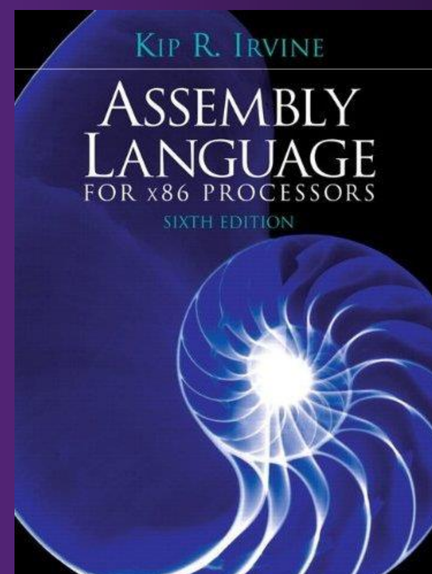
Assembly Language for Intel-Based Computers (< 6th Edition)

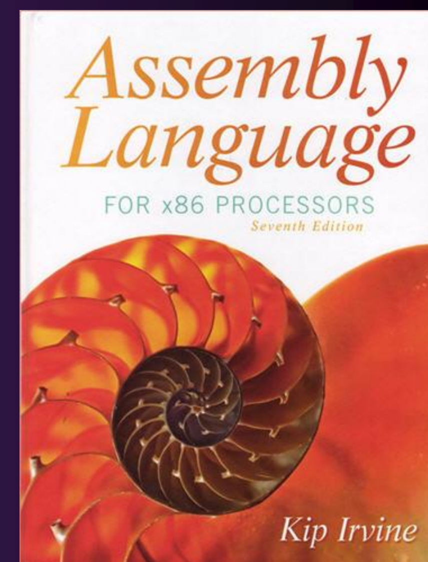
Kip Irvine

Prentice-Hall Press, 2006 (5th), 2010 (6th), 2014 (7th), 2019 (8th) ISBN 978-0135381656







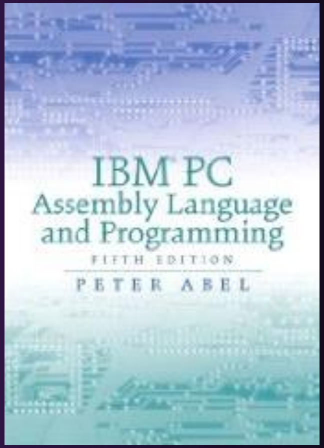


Thanks to Kip Irvine!

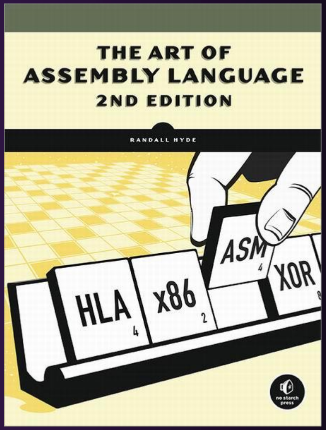
Slides of this course are based on that provided by Kip Irvine

Ref.

IBM PC Assembly Language and Programming (5th Edition)
Peter Abel
Prentice Hall Press, 2001
ISBN 013030655X



The Art of Assembly Language (2nd Edition)
Randall Hyde
No Starch Press, 2010
ISBN 1-59327-207-3



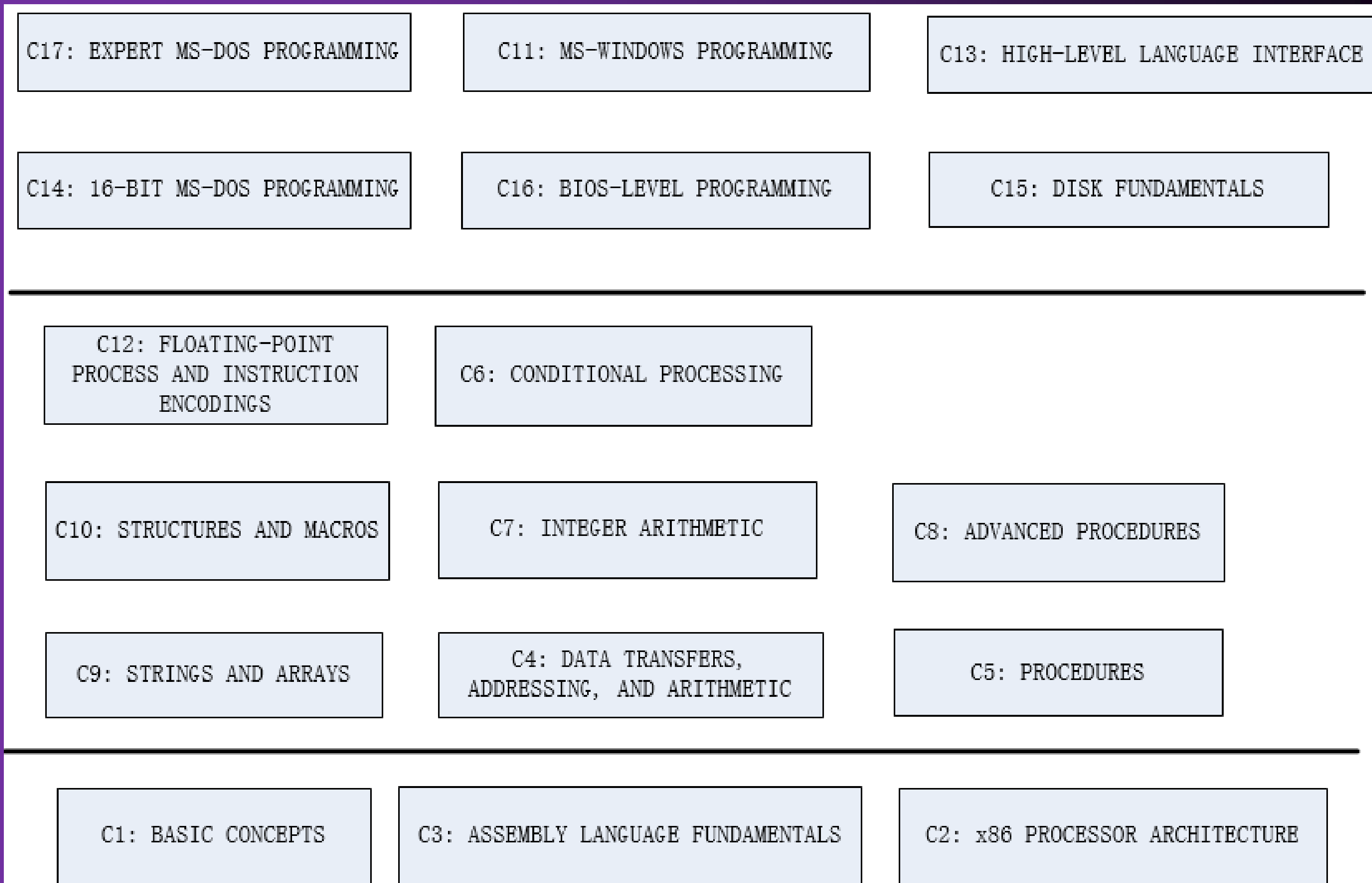
Assembly Language Step by Step: Programming with Linux. 2009.

Course Requirements

- Presentation (~5%).
  - Selected topics in each week
- Homework (10%).
  - Personal work: Programming exercises.
  - Different deadlines
- Project (20%).
  - Team work: Could be conducted with a team (# of members <=3)
- Easy in this part
- Hard-working …

Contents

C1: BASIC CONCEPTS



C17: EXPERT MS-DOS PROGRAMMING

What's Next

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

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

What's Next

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

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

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

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

Integer Storage Sizes

Standard sizes:

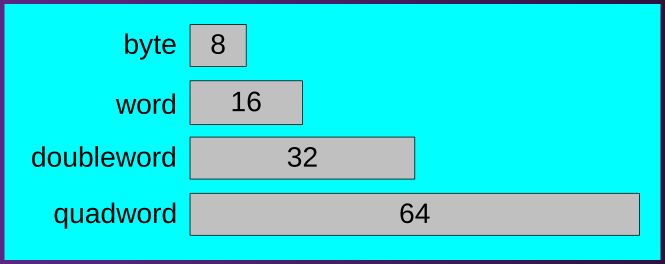


Table 1-4 Ranges of Unsigned Integers.

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 $(2^{64} - 1)$

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

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

Converting Hexadecimal to Decimal

- Multiply each digit by its corresponding power of 16:

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

- 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.

Converting Decimal to Hexadecimal

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

decimal 422 = 1A6 hexadecimal

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.

Hexadecimal Subtraction

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

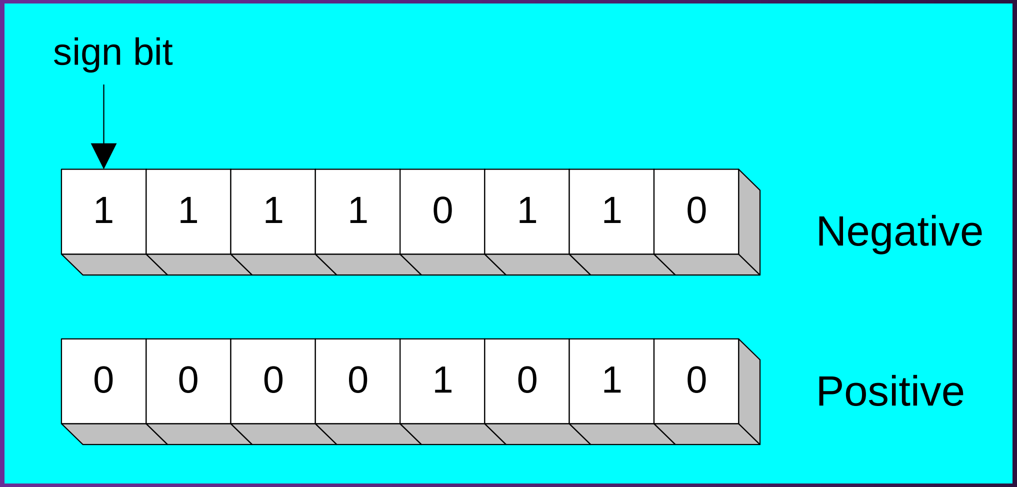
$$\boxed{16 + 5 = 21}$$

$$\begin{array}{rr} & {-1} \\ C6 & 75 \\ A2 & 47 \\ \hline 24 & 2E \end{array}$$

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

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 (补码)

- 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.

What's Next

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

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)

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

- 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	$\neg$X	Y	$\neg X \vee Y$
F	T	F	T
F	T	T	T
T	F	F	F
T	F	T	T