Computer Organization & Assembly Languages

Introduction

Pu-Jen Cheng

2008/09/15

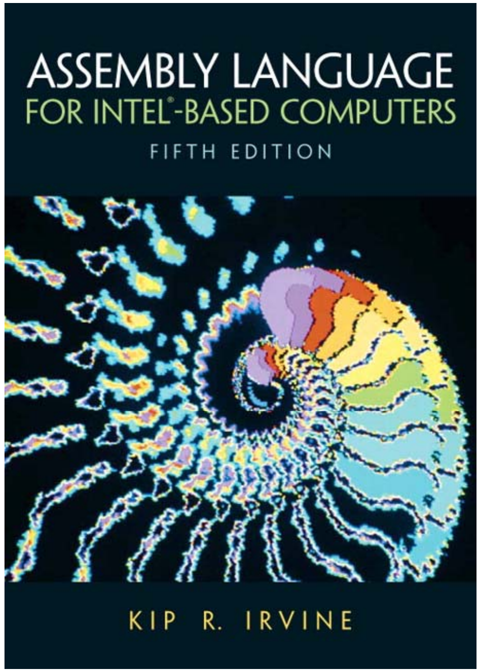
Course Administration

- **Instructor**: Pu-Jen Cheng (CSIE R323)
  pjcheng@csie.ntu.edu.tw
  http://www.csie.ntu.edu.tw/~pjcheng
- **Class Hours**: 2:00pm-5:00pm, Monday
- **Classroom**: CSIE R102
- **TA(s)**: 戴瑋彥 *b93705014@ntu.edu.tw*
- **Course Information**:

  Announce: *http://www.csie.ntu.edu.tw/~pjcheng/course/asm2008/*

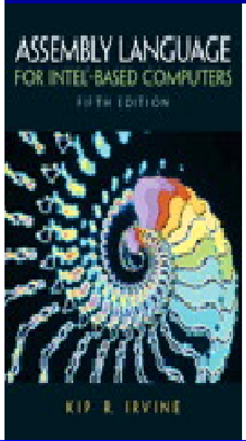
  Q&A: *bbs://ptt.cc → CSIE_ASM*

Textbook



- *Assembly Language for Intel-Based Computers*, 5th Edition, by Kip Irvine, Prentice-Hall, 2006
- *http://www.asmirvine.com*

Assembly Language for Intel-Based Computers, 5th Edition



by Kip Irvine, Florida International University

ISBN: 0-13-238310 - 1

Published by: Prentice-Hall

Visit the Web site for the Fourth Edition...

Printable Chapters

- TOC and Preface
- Chapter 1
- Chapter 2
- Chapter 3

Buy the book at Amazon.com

Find it on Bookfinder.com

Official Prentice-Hall Web site

Microsoft's MASM Reference

Intel IA-32 Architecture Manuals

Information and Help

Getting started

Bug reports

Link libraries and example programs

Where is the CDROM?

Supplemental files

Chapter objectives

Assembly language workbook

Supplemental articles

Help file for the book's link library

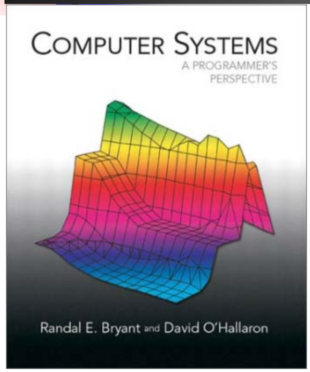
Contacts

Instructor resources

Discussion group (Y

International reader

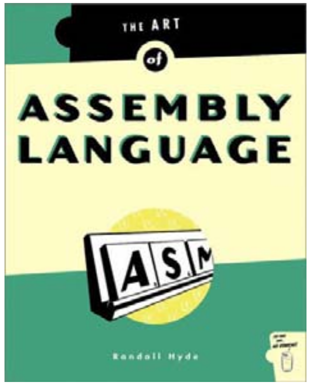
References



<u>Computer Systems: A Programmer's Perspective</u>

By Randal E. Bryant and David R. O'Hallaron, Prentice Hall

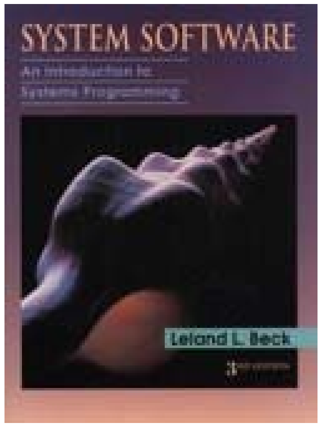
http://csapp.cs.cmu.edu/



<u>The Art of Assembly Language</u>

By Randy Hyde,

http://webster.cs.ucr.edu/AoA/Windows/PDFs/0_PDFIndexWin.html



<u>System Software: An Introduction to Systems Programming</u>

By Leland L. Beck

Addison-Wesley

Pre-requisite

- Experiences in writing programs in a high-level language such as C, C++, and Java

Course Grading (tentative)

- Assignments (55%)
- Class participation (5%)
- Midterm exam (20%)
- Final exam (20%)

Materials

- Some materials used in this course are adapted from
  - The slides prepared by Kip Irvine for the book, Assembly Language for Intel-Based Computers, 5th Ed.
  - The slides prepared by S. Dandamudi for the book, Introduction to Assembly Language Programming, 2nd Ed.
  - Introduction to Computer Systems, CMU
    (http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15213-f05/www/)
  - Assembly Language & Computer Organization, NTU
    (http://www.csie.ntu.edu.tw/~cyy/courses/assembly/06fall/news//)
    (http://www.csie.ntu.edu.tw/~acpang/course/asm_2004)

What is Assembly Language

- First Glance at Assembly Language

Translating Languages

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

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

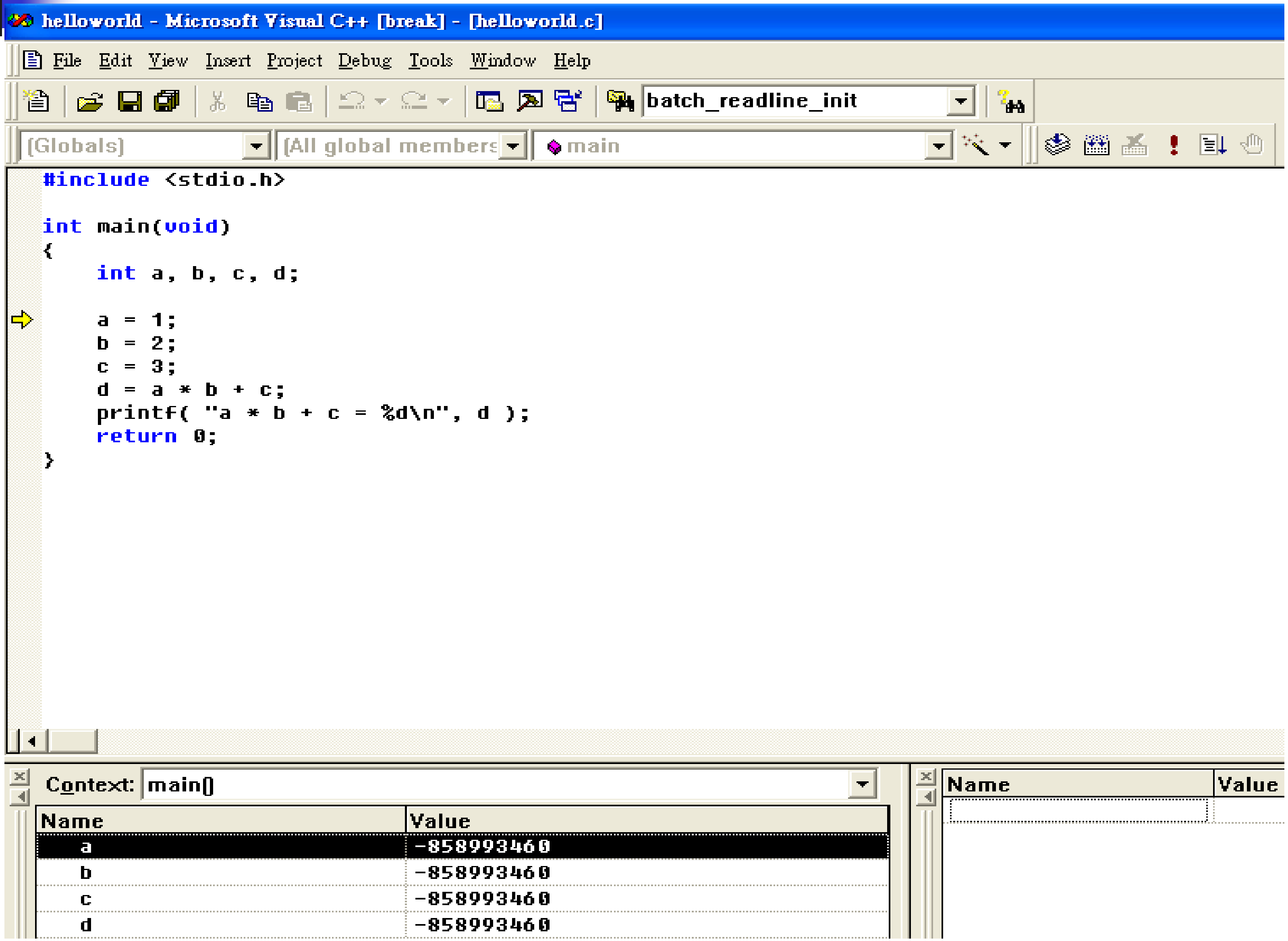
Assembly Language:

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

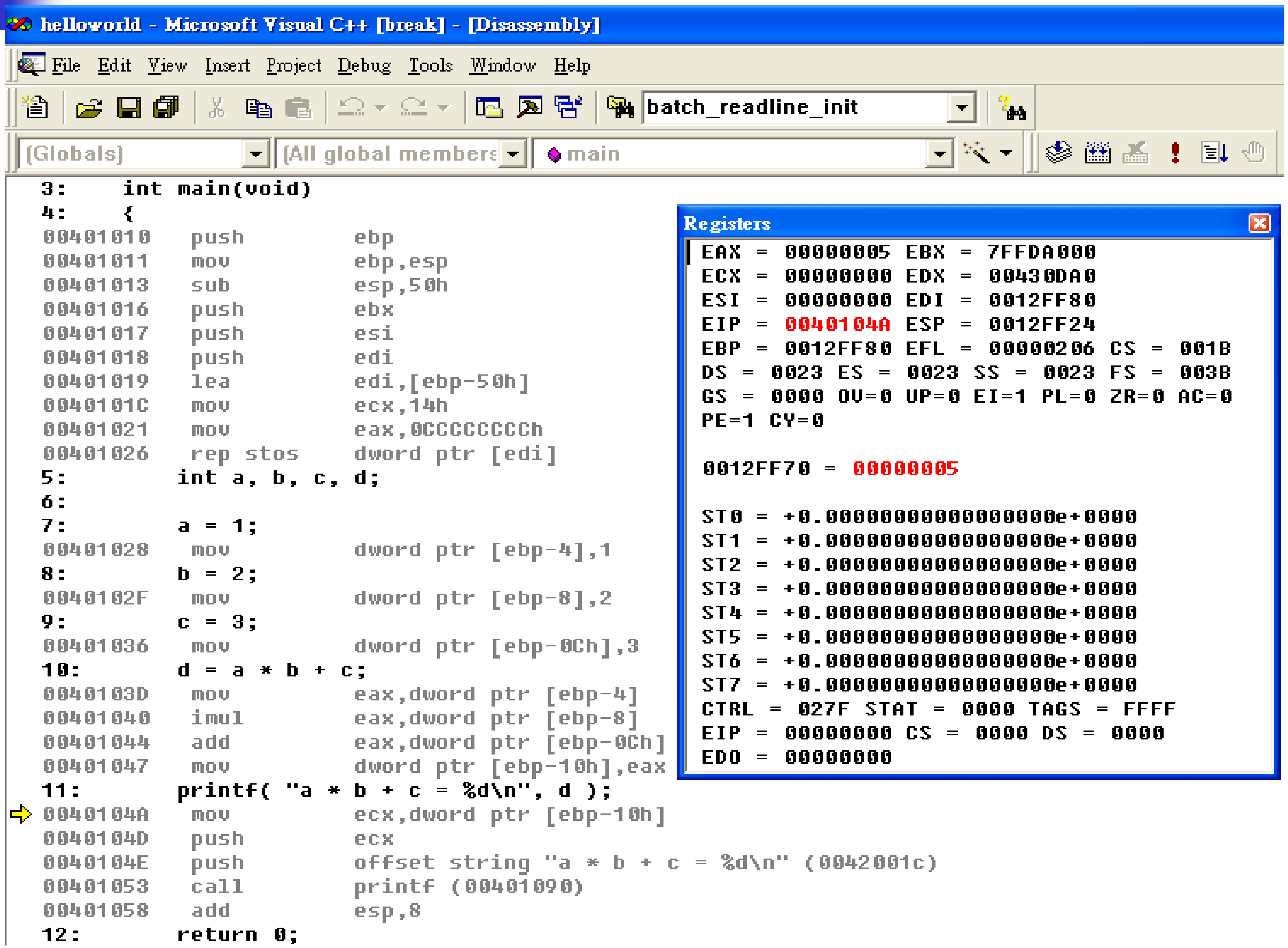
Intel Machine Language:

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

A Simple Example in VC++



View/Debug Windows/Disassembly

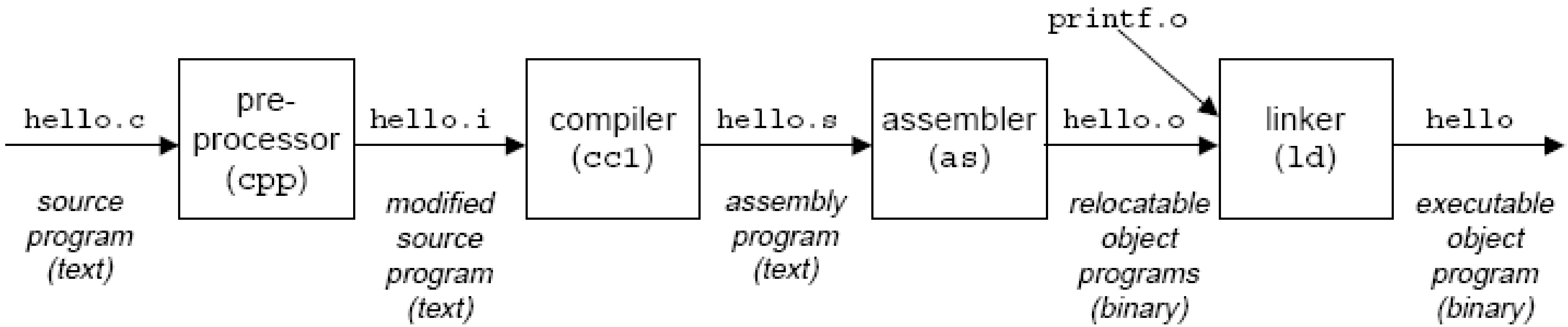


gcc -s prog.c

ken.csie.ntu.edu.tw - PuTTY

```
        .file   "prog1.c"
        .section        .rodata
.LC0:
        .string "a * b + c = %d\n"
        .text
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $36, %esp
        movl    $1, -20(%ebp)
        movl    $2, -16(%ebp)
        movl    $3, -12(%ebp)
        movl    -20(%ebp), %eax
        imull   -16(%ebp), %eax
        addl    -12(%ebp), %eax
        movl    %eax, -8(%ebp)
        movl    -8(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    printf
        movl    $0, %eax
        addl    $36, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (GNU) 4.1.2 20060901 (prerelease) (Debian 4.1.1-13)"
        .section        .note.GNU-stack,"",@progbits
```

The Compilation System



First Glance at Assembly Language

- Low-level language
  - Each instruction performs a much lower-level task compared to a high-level language instruction
  - Most high-level language instructions need more than one assembly instruction
- One-to-one correspondence between assembly language and machine language instructions
  - For most assembly language instructions, there is a machine language equivalent
- Directly influenced by the instruction set and architecture of the processor (CPU)

Comparisons with High-level Languages

- Advantages of Assembly Languages
  - Space-efficiency

    (e.g. hand-held device softwares, etc)
  - Time-efficiency

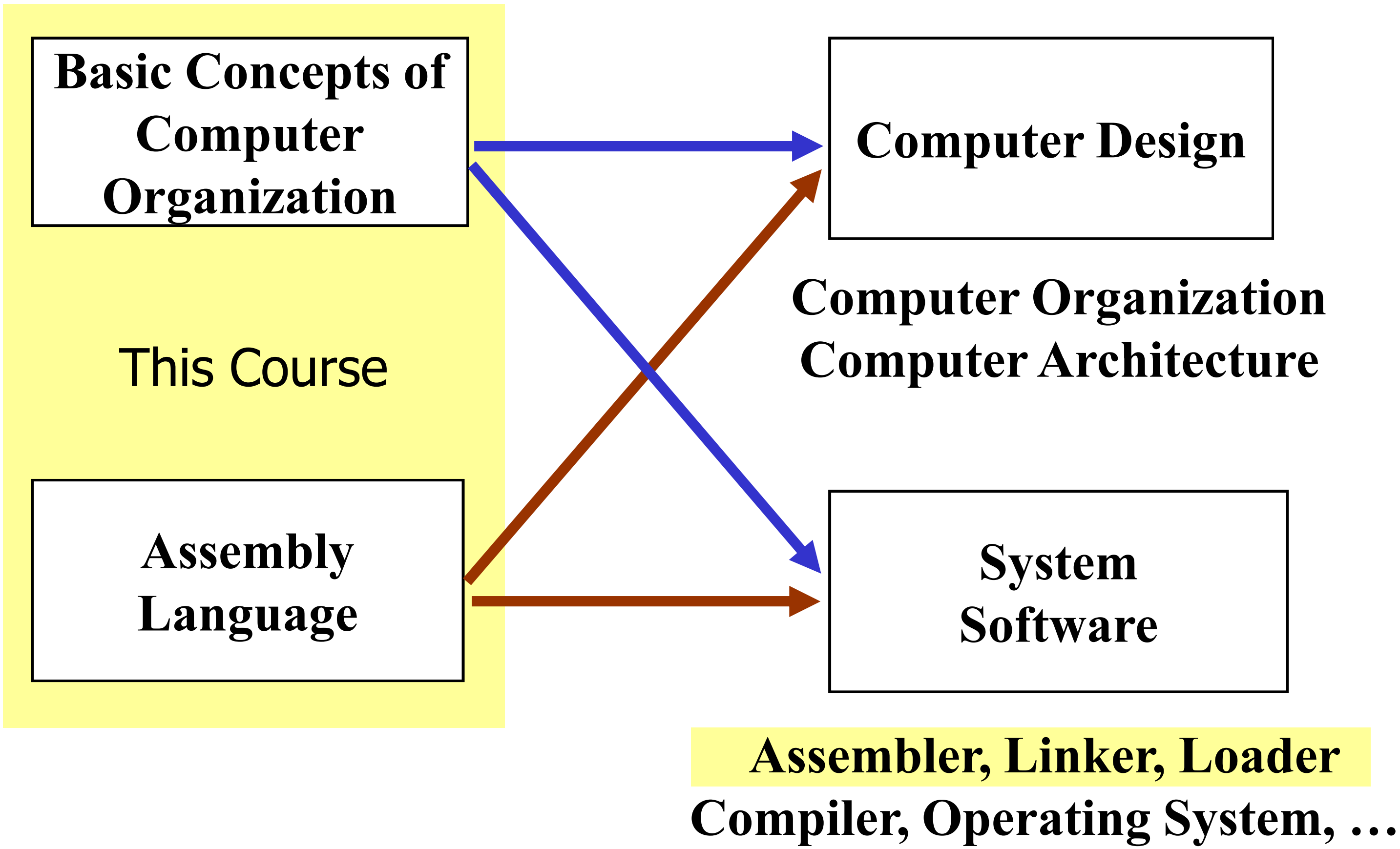
    (e.g. Real-time applications, etc)
  - Accessibility to system hardwares

    (e.g., Network interfaces, device drivers, video games, etc)
- Advantages of High-level Languages
  - Development
  - Maintenance (Readability)
  - Portability (compiler, virtual machine)

Comparisons with High-level Languages (cont.)

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.

Why Taking the Course?



"I really don't think that you can write a book for serious computer programmers unless you are able to discuss low-level details."

Donald Knuth (高德納)

The Art of Computer Programming

http://en.wikipedia.org/wiki/Donald_Knuth

Course Coverage

- Basic Concepts
- IA-32 Processor Architecture
- Assembly Language Fundamentals
- Data Transfers, Addressing, and Arithmetic
- Procedures
- Conditional Processing
- Integer Arithmetic
- Advanced Procedures
- Strings and Arrays
- Structures and Macros
- High-Level Language Interface
- Assembler, Linker, and Loader
- Other Advanced Topics (optional)

What You Will Learn

- Basic principles of computer architecture
- IA-32 processors and memory management
- Basic assembly programming skills
- How high-level language is translated to assembly
- How assembly is translated to machine code
- How application program communicates with OS
- Interface between assembly to high-level language

Performance: Multiword Arithmetic

- Longhand multiplication
  - Final 128-bit result in P:A

```
    0 1 0 1
    1 1 0 1
-----------
    0 1 0 1
  0 1 0 1
0 1 0 1
```

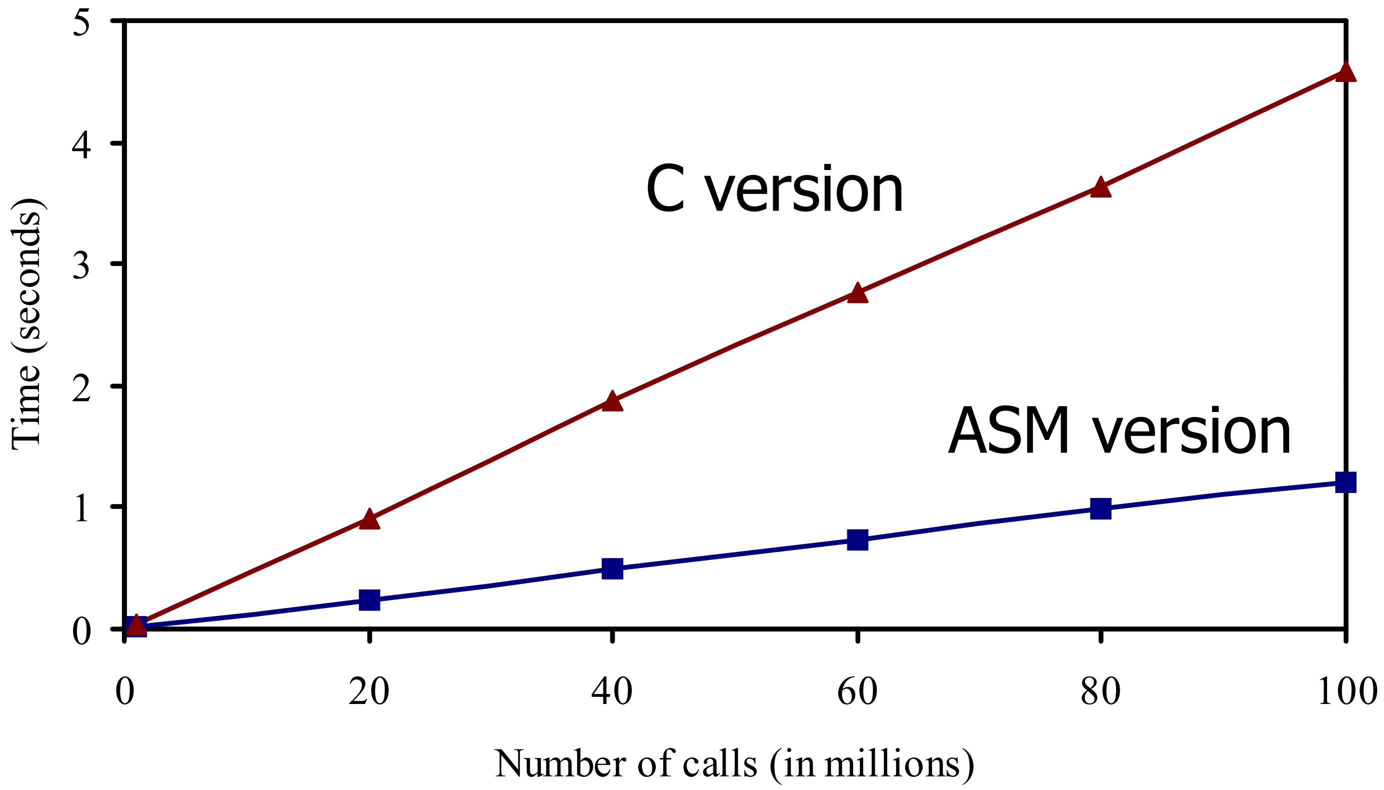
- P := 0; count := 64
- A := multiplier; B := multiplicand
- <u>while</u> (count > 0)
-     <u>if</u> (LSB of A = 1)
-     <u>then</u> P := P+B
-           CF := carry generated by P+B
-     <u>else</u> CF := 0
-     <u>end if</u>
-     shift right CF:P:A by one bit position
-     count := count-1
- <u>end while</u>

Example

- $A = 1101_2$ (13)
- $B = 0101_2$ (5)

	After P+B			After the shift		
	CF	P	A	CF	P	A
Initial state	?	0000	1101	--	----	----
Iteration 1	0	0101	1101	?	0010	1110
Iteration 2	0	0010	1110	?	0001	0111
Iteration 3	0	0110	0111	?	0011	0011
Iteration 4	0	1000	0011	?	0100	0001

Time Comparison



Multiplication time comparison on a 2.4-GHz Pentium 4 system

Chapter 1: Basic Concept

- Virtual Machine Concept
- Data Representation
- Boolean Operations

Translating Languages

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

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

Assembly Language:

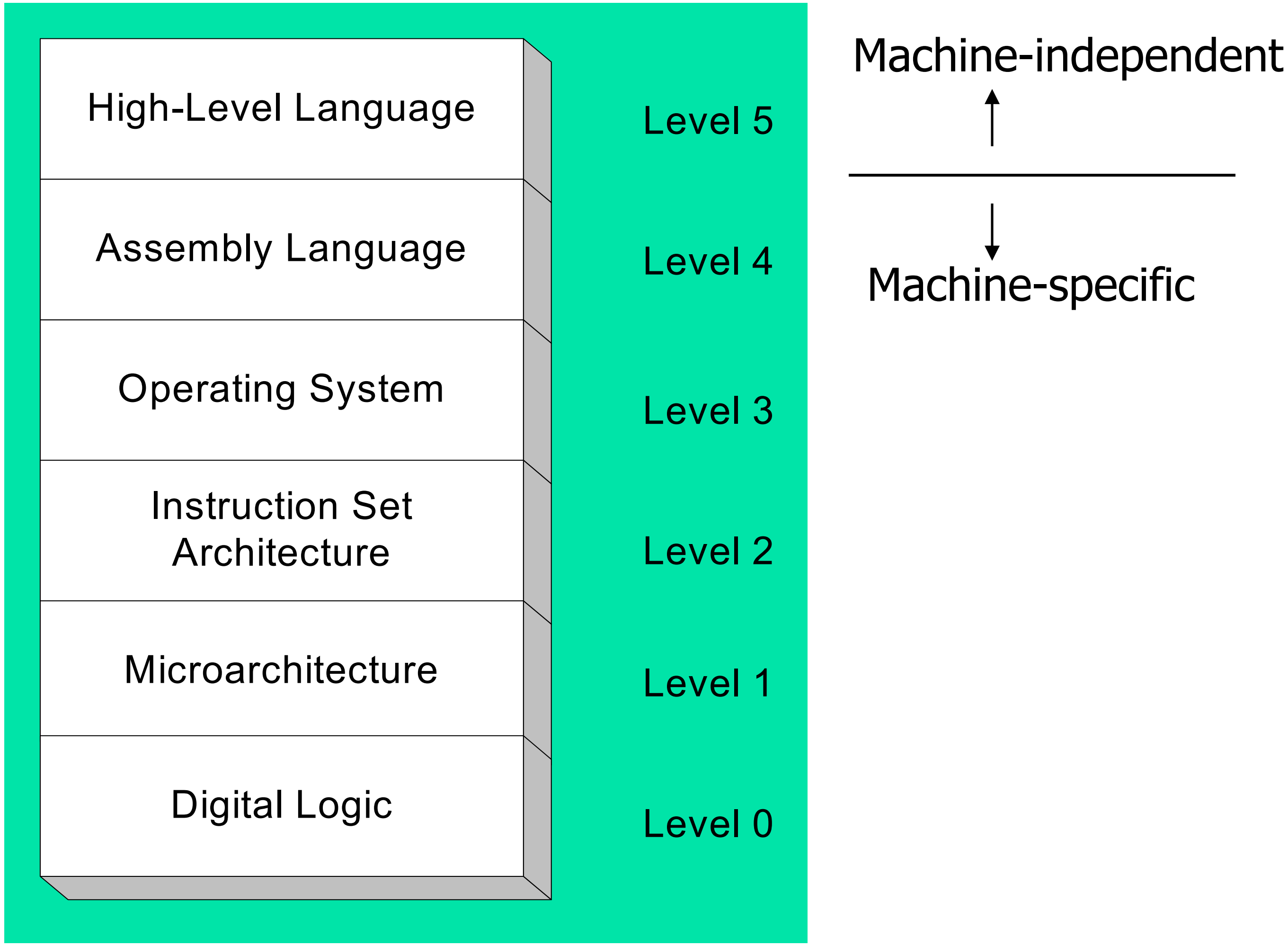
```
mov eax,A
mul B
add eax,C
call WriteInt
```

Intel Machine Language:

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

Virtual Machines

Abstractions for computers



High-Level Language

- Level 5
- Application-oriented languages
  - C++, Java, Pascal, Visual Basic . . .
- Programs compile into assembly language (Level 4)

Assembly Language

- Level 4
- Instruction mnemonics that have a one-to-one correspondence to machine language
- Calls functions written at the operating system level (Level 3)
- Programs are translated into machine language (Level 2)

Operating System

- Level 3
- Provides services to Level 4 programs
- Translated and run at the instruction set architecture level (Level 2)

Instruction Set Architecture

- Level 2
- Also known as conventional machine language
- Executed by Level 1 (microarchitecture) program

Microarchitecture

- Level 1
- Interprets conventional machine instructions (Level 2)
- Executed by digital hardware (Level 0)

Digital Logic

- Level 0
- CPU, constructed from digital logic gates
- System bus
- Memory

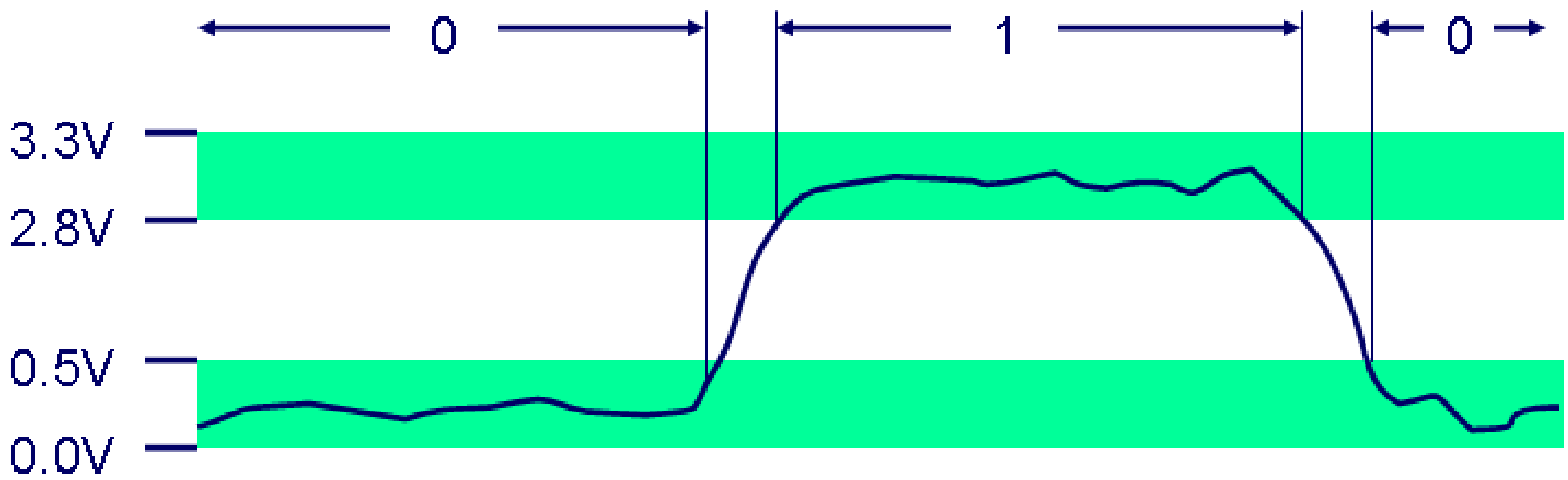
next: Data Representation

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
- Fractional Binary Numbers
- Character Storage
- Machine Words

Binary Representation

- Electronic Implementation
  - Easy to store with bistable elements
  - Reliably transmitted on noisy and inaccurate wires



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:

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

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

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	

Integer Storage Sizes

Standard sizes:

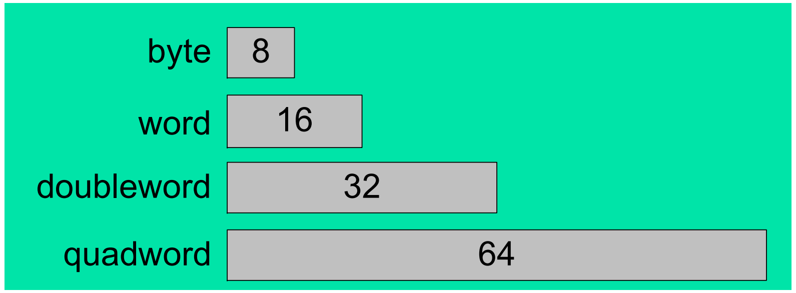


Table 1-4 Ranges of Unsigned Integers.

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?

Large Measurements

- Kilobyte (KB), 2^{10} bytes
- Megabyte (MB), 2^{20} bytes
- Gigabyte (GB), 2^{30} bytes
- Terabyte (TB), 2^{40} bytes
- Petabyte, 2^{50} bytes
- Exabyte, 2^{60} bytes
- Zettabyte, 2^{70} bytes
- Yottabyte, 2^{80} bytes
- Googol, 10^{100}

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

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.

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

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:

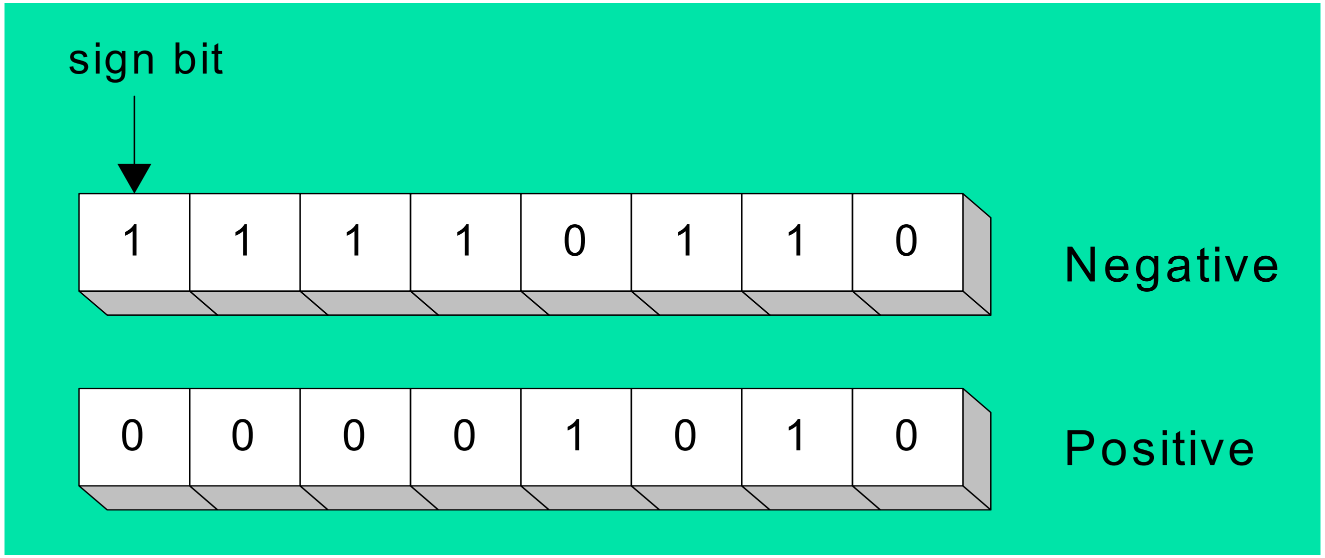
$\boxed{16 + 5 = 21}$

$$
\begin{array}{r}
{\scriptstyle -1} \\
\text{C6 75} \\
\text{A2 47} \\
\hline
\text{24 2E}
\end{array}
$$

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

- Bitwise NOT of the number and add 1

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

8-bit Two's Complement Integers

sign bit									
0	1	1	1	1	1	1	1	=	127
0	0	0	0	0	0	1	0	=	2
0	0	0	0	0	0	0	1	=	1
0	0	0	0	0	0	0	0	=	0
1	1	1	1	1	1	1	1	=	−1
1	1	1	1	1	1	1	0	=	−2
1	0	0	0	0	0	0	1	=	−127
1	0	0	0	0	0	0	0	=	−128

8-bit two's complement integers

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

Advantages for 2's complement:

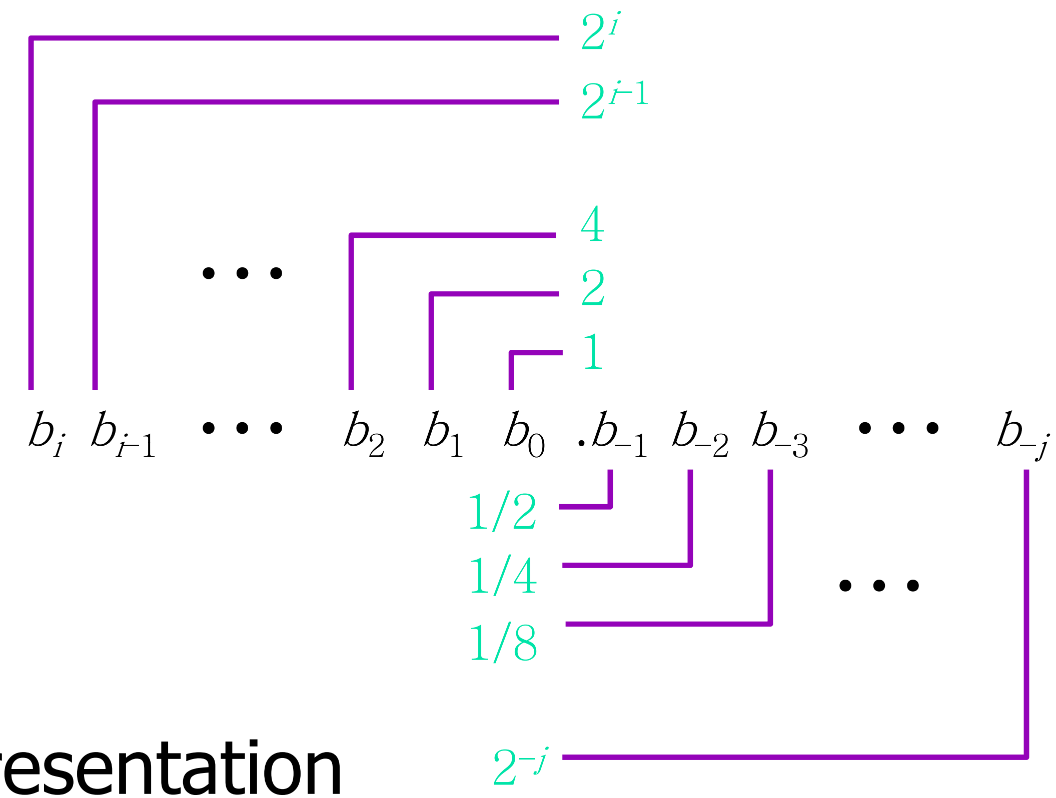
- No two 0's
- Sign bit
- Remove the need for separate circuits for add and sub

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 $(2^{63} - 1)$

Fractional Binary Numbers



- Representation
  - Bits to right of “binary point” represent fractional powers of 2
  - Represents rational number: $\sum_{k=-j}^{i} b_k \cdot 2^k$

Examples of Fractional Binary Numbers

- Value / Representation

Value	Representation
5-3/4	101.11_2
2-7/8	10.111_2
63/64	0.111111_2

- Observations
  - Divide by 2 by shifting right
  - Multiply by 2 by shifting left
  - Numbers of form $0.111111\ldots_2$ just below 1.0
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - Use notation $1.0 - \varepsilon$

Representable Numbers

- Limitation
  - Can only exactly represent numbers of the form $x \times 2^y$
  - Other numbers have repeating bit representations
- Value Representation

Value	Representation
1/3	0.0101010101[01]...$_2$
1/5	0.001100110011[0011]...$_2$
1/10	0.0001100110011[0011]...$_2$

Converting Real Numbers

- Binary real to decimal real

$$110.011_2 = 4 + 2 + 0.25 + 0.125 = 6.375$$

- Decimal real to binary real

$$\begin{aligned} 0.5625 \times 2 &= 1.125 & \text{first bit} &= 1 \\ 0.125 \times 2 &= 0.25 & \text{second bit} &= 0 \\ 0.25 \times 2 &= 0.5 & \text{third bit} &= 0 \\ 0.5 \times 2 &= 1.0 & \text{fourth bit} &= 1 \end{aligned}$$

$$4.5625 = 100.1001_2$$

True or False

- If x > 0 then x + 1 > 0
- If x < 0 then x * 2 < 0
- If x > y then -x < -y
- If x >= 0 then -x <= 0
- If x < 0 then -x > 0
- If x >= 0 then ((!x – 1) & x) == x
- If x < 0 && y > 0 then x * y < 0
- If x < 0 then ((x ^ x >> 31) + 1) > 0

Character Storage

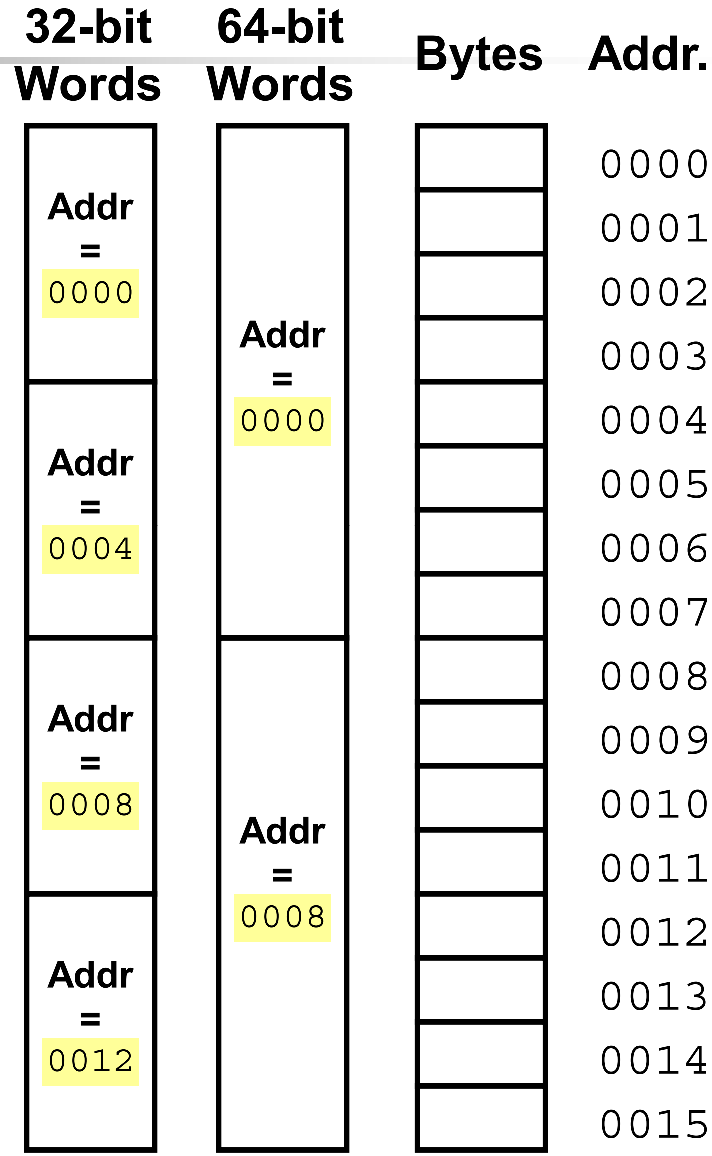
- Character sets
  - Standard ASCII (0 – 127)
  - Extended ASCII (0 – 255)
  - ANSI (0 – 255)
  - Unicode (0 – 65,535)
- Null-terminated String
  - Array of characters followed by a *null byte*
- Using the ASCII table
  - back inside cover of book

Machine Words

- Machine Has “Word Size”
  - Nominal size of integer-valued data
    - Including addresses
  - Most current machines use 32 bits (4 bytes) words
    - Limits addresses to 4GB
      - Users can access 3GB
    - Becoming too small for memory-intensive applications
  - High-end systems use 64 bits (8 bytes) words
    - Potential address space $\approx 1.8 \times 10^{19}$ bytes
    - x86-64 machines support 48-bit addresses: 256 Terabytes
  - Machines support multiple data formats
    - Fractions or multiples of word size
    - Always integral number of bytes

Word-Oriented Memory Organization

- Addresses Specify Byte Locations
  - Address of first byte in word
  - Addresses of successive words differ by 4 (32-bit) or 8 (64-bit)



Data Representations

- Sizes of C Objects (in Bytes)

C Data Type	Typical 32-bit	Intel IA32	x86-64
unsigned	4	4	4
int	4	4	4
long int	4	4	4
char	1	1	1
short	2	2	2
float	4	4	4
double	8	8	8
char *	4	4	8

  - Or any other pointer

Byte Ordering

- How should bytes within multi-byte word be ordered in memory?
- Conventions
  - Big Endian: Sun, PPC Mac
    - Least significant byte has highest address
  - Little Endian: x86
    - Least significant byte has lowest address

Byte Ordering Example

- Big Endian
  - Least significant byte has highest address
- Little Endian
  - Least significant byte has lowest address
- Example
  - Variable `x` has 4-byte representation `0x01234567`
  - Address given by `&x` is `0x100`

Big Endian

		0x100	0x101	0x102	0x103		
		01	23	45	67		

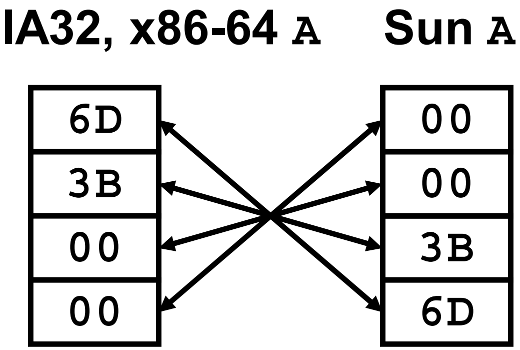
Little Endian

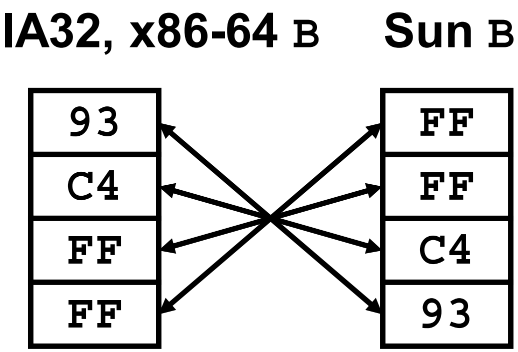
		0x100	0x101	0x102	0x103		
		67	45	23	01		

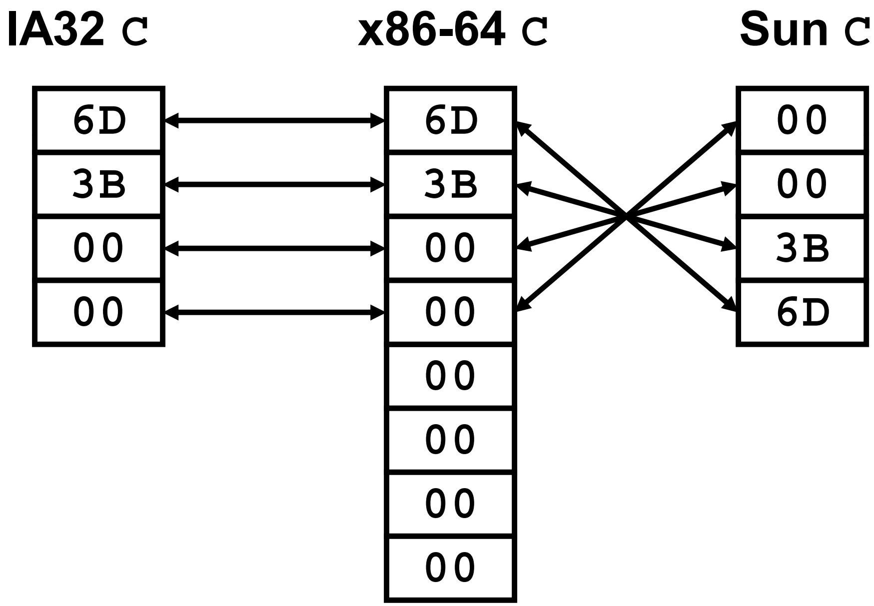
Representing Integers

- `int A = 15213;`
- `int B = -15213;`
- `long int C = 15213;`

Decimal:	15213			
Binary:	0011	1011	0110	1101
Hex:	3	B	6	D





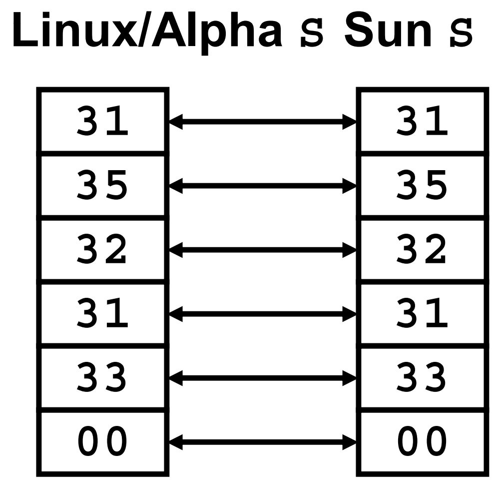


Two's complement representation

Representing Strings

- Strings in C
  - Represented by array of characters
  - Each character encoded in ASCII format
    - Standard 7-bit encoding of character set
    - Character "0" has code `0x30`
      - Digit *i* has code `0x30+`*i*
  - String should be null-terminated
    - Final character = 0
- Compatibility
  - Byte ordering not an issue

- `char S[6] = "15213";`



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

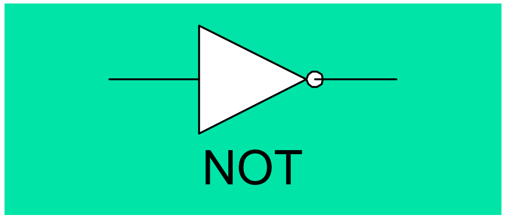
Expression	Description
$\neg X$	NOT X
$X \wedge Y$	X AND Y
$X \vee Y$	X OR Y
$\neg X \vee Y$	(NOT X) OR Y
$\neg(X \wedge Y)$	NOT (X AND Y)
$X \wedge \neg Y$	X AND (NOT Y)

NOT

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

X	¬X
F	T
T	F

Digital gate diagram for NOT:

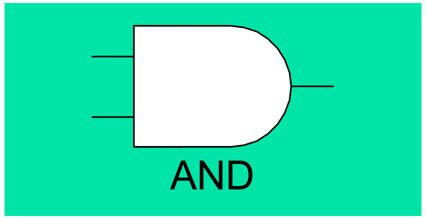


AND

- 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:

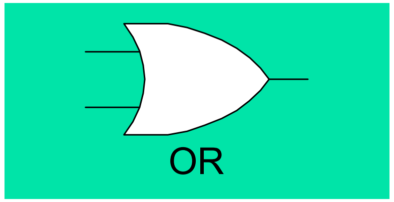


OR

- 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

- NOT > AND > OR
- 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

- Use parentheses to avoid ambiguity

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

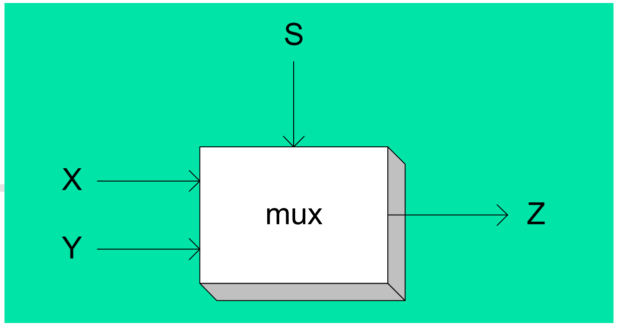
Truth Tables (2 of 3)

- Example: $X \wedge \neg Y$

X	Y	$\neg \mathbf{Y}$	$\mathbf{X} \wedge \neg \mathbf{Y}$
F	F	T	F
F	T	F	F
T	F	T	T
T	T	F	F

Truth Tables (3 of 3)

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



Two-input multiplexer

X	Y	S	$Y \wedge S$	$\neg S$	$X \wedge \neg S$	$(Y \wedge S) \vee (X \wedge \neg S)$
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