Course overview

Computer Organization and Assembly Languages
Yung-Yu Chuang
2005/09/22

with slides by Kip Irvine

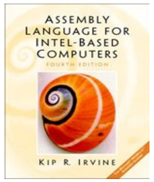
Logistics

- **Meeting time**: 9:10am-12:10pm, Thursday
- **Classroom**: CSIE Room 103
- **Instructor**: Yung-Yu Chuang
- **Teaching assistants**: 徐士璿/楊善詠
- **Webpage**:
  http://www.csie.ntu.edu.tw/~cyy/assembly
  id / password
- **Forum**:
  http://www.cmlab.csie.ntu.edu.tw/~cyy/forum/viewforum.php?f=4
- **Mailing list**: assembly@cmlab.csie.ntu.edu.tw
  Please subscribe via
  https://cmlmail.csie.ntu.edu.tw/mailman/listinfo/assembly/

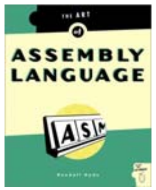
Prerequisites

- Programming experience with some high-level language such C, C ++,Java …

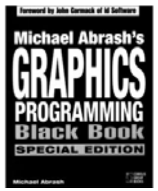
Books



Textbook
Assembly Language for Intel-Based Computers, 4th Edition, Kip Irvine



Reference
The Art of Assembly Language, Randy Hyde



Michael Abrash's Graphics Programming Black Book, chap 1-22

Grading (subject to change)

- Assignments (50%)
- Class participation (5%)
- Midterm exam (20%)
- Final project (25%)

Why learning assembly?

- It is required.
- It is foundation for computer architecture and compilers.
- At times, you do need to write assembly code.

"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

Why programming in assembly?

- It is all about lack of smart compilers

- Faster code, compiler is not good enough
- Smaller code , compiler is not good enough, e.g. mobile devices, embedded devices, also Smaller code → better cache performance → faster code
- Unusual architecture , there isn't even a compiler or compiler quality is bad, eg GPU, DSP chips, even MMX.

Syllabus (topics we might cover)

- 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
- BIOS Level Programming
- Real Arithmetic
- MMX
- Code Optimization

What you will learn

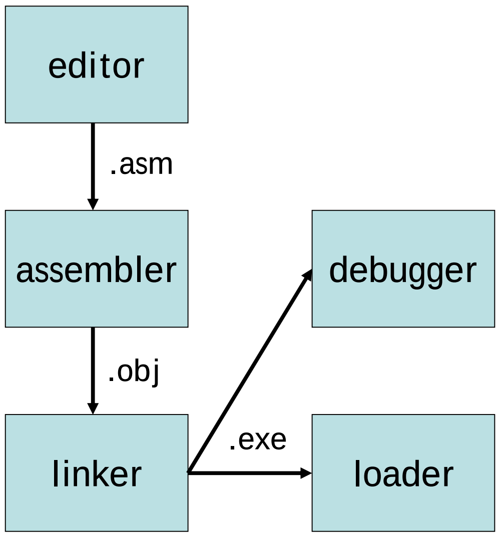
- Basic principle of computer architecture
- IA-32 modes and memory management
- Assembly basics
- How high-level language is translated to assembly
- How to communicate with OS
- Specific components, FPU/MMX
- Code optimization
- Interface between assembly to high-level language

Chapter.1 Overview

- Virtual Machine Concept
- Data Representation
- Boolean Operations

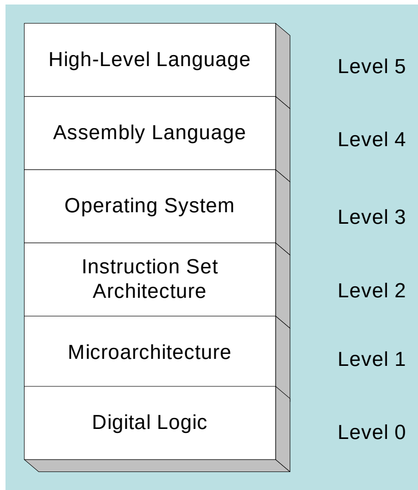
Assembly programming

```
mov eax, Y
add eax, 4
mov ebx, 3
imul ebx
mov X, eax
```



Virtual machines

Abstractions for computers



High-Level Language

- Level 5
- Application-oriented languages
- Programs compile into assembly language (Level 4)

```
X:=(Y+4)*3
```

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)

```
mov eax, Y
add eax, 4
mov ebx, 3
imul ebx
mov X, eax
```

Operating System

- Level 3
- Provides services
- 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 program (microarchitecture, Level 1)

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

Data representation

- Computer is a construction of digital circuits with two states: *on* and *off*
- You need to have the ability to translate between different representations to examine the content of the machine
- Common number systems: binary, octal, decimal and hexadecimal

Binary numbers

- Digits are 1 and 0
  (a binary digit is called a bit)
  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

- A bit string could have different interpretations

Unsigned binary integers

- 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

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:

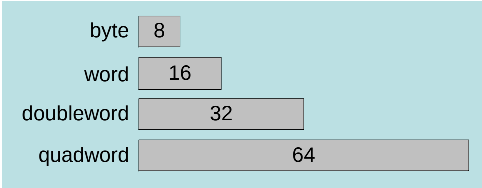


Table 1-4 Ranges of Unsigned Integers.

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 $(2^{64} - 1)$

Practice: 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
- Exabyte
- Zettabyte
- Yottabyte

Hexadecimal integers

All values in memory are stored in binary. Because long binary numbers are hard to read, we use hexadecimal representation.

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:

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

- 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

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

Hexadecimal subtraction

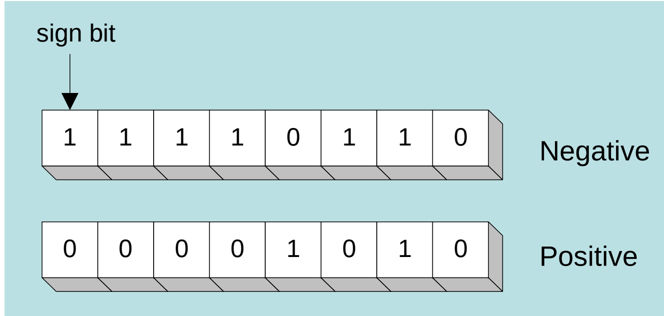
When a borrow is required from the digit to the left, add 10h to the current digit's value:

```
      -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?

Signed integers

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



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

Two's complement notation

Steps:

- Complement (reverse) each bit
- 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

Binary subtraction

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

```
  1 1 0 0   ------>    1 1 0 0
– 0 0 1 1              1 1 0 1
---------            ---------
                       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 $(2^{63} - 1)$

Character

- 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

Boolean algebra

- Boolean expressions created from:
  - NOT, AND, OR

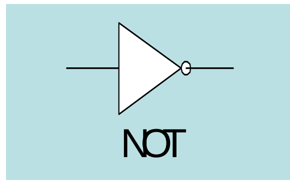
Expression	Description
$\neg X$	NOT X
$X \wedge Y$	X AND Y
$X \vee Y$	X OR Y
$\neg X \vee Y$	(NOT X) OR Y
$\neg(X \wedge Y)$	NOT (X AND Y)
$X \wedge \neg Y$	X AND (NOT Y)

NOT

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

X	¬X
F	T
T	F

Digital gate diagram for NOT:

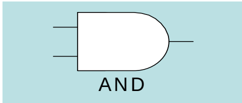


AND

- Truth if both are true
- 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:

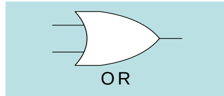


OR

- True if either is true
- 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
¬X ∨ Y	NOT, then OR
¬(X ∨ Y)	OR, then NOT
X ∨ (Y ∧ 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: ¬X ∨ Y

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

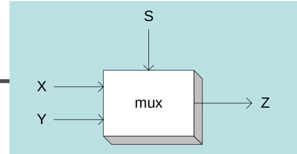
Truth Tables (2 of 3)

- 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

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