Course overview

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

Logistics
- Meeting time: 9:10am-12:10pm, Thursday
- Classroom: CSIE Room 103
- Instructor: Yung-Yu Chuang
- Teaching assistants: 徐士瑤/楊善詠
- Webpage: http://www.csie.ntu.edu.tw/~cyy/assembly
  id / password
- 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 $\rightarrow$ better cache performance $\rightarrow$ faster code
- Unusual architecture, there isn’t even a compiler or compiler quality is bad, e.g. 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

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:

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 0 0</td>
<td>1 0 1 0 0 1 1 1 0 0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

- 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:

\[
\begin{array}{cccccccc}
2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}
\]

Every binary number is a sum of powers of 2

<table>
<thead>
<tr>
<th>$2^n$</th>
<th>Decimal Value</th>
<th>$2^n$</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^0$</td>
<td>1</td>
<td>$2^8$</td>
<td>256</td>
</tr>
<tr>
<td>$2^1$</td>
<td>2</td>
<td>$2^9$</td>
<td>512</td>
</tr>
<tr>
<td>$2^2$</td>
<td>4</td>
<td>$2^{10}$</td>
<td>1024</td>
</tr>
<tr>
<td>$2^3$</td>
<td>8</td>
<td>$2^{11}$</td>
<td>2048</td>
</tr>
<tr>
<td>$2^4$</td>
<td>16</td>
<td>$2^{12}$</td>
<td>4096</td>
</tr>
<tr>
<td>$2^5$</td>
<td>32</td>
<td>$2^{13}$</td>
<td>8192</td>
</tr>
<tr>
<td>$2^6$</td>
<td>64</td>
<td>$2^{14}$</td>
<td>16384</td>
</tr>
<tr>
<td>$2^7$</td>
<td>128</td>
<td>$2^{15}$</td>
<td>32768</td>
</tr>
</tbody>
</table>

Table 1-3: Binary Bit Position Values.

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:

\[
\begin{array}{|c|c|c|}
\hline
\text{Division} & \text{Quotient} & \text{Remainder} \\
\hline
37 / 2 & 18 & 1 \\
18 / 2 & 9 & 0 \\
9 / 2 & 4 & 1 \\
4 / 2 & 2 & 0 \\
2 / 2 & 1 & 0 \\
1 / 2 & 0 & 1 \\
\hline
\end{array}
\]

37 = 100101

Binary addition

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

\[
\begin{array}{c}
\text{carry: 1} \\
0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \\
+ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \\
\hline
0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \\
\end{array}
\]

bit position: 7 6 5 4 3 2 1 0
### Integer storage sizes

**Standard sizes:**

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

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

Table 1-4 Ranges of Unsigned Integers.

### Hexadecimal integers

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

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

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

### 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

### Translating binary to hexadecimal

- Each hexadecimal digit corresponds to 4 binary bits.
- Example: Translate the binary integer 000101101010011110010100 to hexadecimal:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0110</td>
</tr>
<tr>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>0101</td>
<td>1001</td>
</tr>
<tr>
<td>0010</td>
<td>1111</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

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 \((3 \times 16^3) + (11 \times 16^2) + (10 \times 16^1) + (4 \times 16^0)\), or decimal 15,268.

Powers of 16

Used when calculating hexadecimal values up to 8 digits long:

<table>
<thead>
<tr>
<th>16^n</th>
<th>Decimal Value</th>
<th>16^n</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>16^0</td>
<td>1</td>
<td>16^4</td>
<td>65,536</td>
</tr>
<tr>
<td>16^1</td>
<td>16</td>
<td>16^5</td>
<td>1,048,576</td>
</tr>
<tr>
<td>16^2</td>
<td>256</td>
<td>16^6</td>
<td>16,777,216</td>
</tr>
<tr>
<td>16^3</td>
<td>4096</td>
<td>16^7</td>
<td>268,435,456</td>
</tr>
</tbody>
</table>

Converting decimal to hexadecimal

<table>
<thead>
<tr>
<th>Division</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>422 / 16</td>
<td>26</td>
<td>6</td>
</tr>
<tr>
<td>26 / 16</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>1 / 16</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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.

\[
\begin{align*}
36 & \div 16 = 2 \text{ remainder } 4 \\
28 & \div 16 = 1 \text{ remainder } 12 \\
28 & \div 16 = 1 \text{ remainder } 12 \\
6A & \div 16 = 1 \text{ remainder } 10 \\
42 & \div 16 = 2 \text{ remainder } 10 \\
45 & \div 16 = 2 \text{ remainder } 11 \\
58 & \div 16 = 3 \text{ remainder } 10 \\
4B & \div 16 = 2 \text{ remainder } 11 \\
78 & \div 16 = 4 \text{ remainder } 10 \\
6D & \div 16 = 4 \text{ remainder } 13 \\
80 & \div 16 = 5 \text{ remainder } 0 \\
B5 & \div 16 = 5 \text{ remainder } 13 \\
\end{align*}
\]

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:

\[
\begin{array}{c|c}
-1 & \hline
C6 & 75 \\
A2 & 47 \\
\hline
24 & 2E
\end{array}
\]

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

Two's complement notation

Steps:
- Complement (reverse) each bit
- Add 1

<table>
<thead>
<tr>
<th>Starting value</th>
<th>00000001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: reverse the bits</td>
<td>11111110</td>
</tr>
<tr>
<td>Step 2: add 1 to the value from Step 1</td>
<td>11111110</td>
</tr>
<tr>
<td>Sum: two’s complement representation</td>
<td>11111111</td>
</tr>
</tbody>
</table>

Note that 00000001 + 11111111 = 00000000

Binary subtraction

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

\[
\begin{array}{c|c|c|c}
1100 \rightarrow & 1100 \\
-0011 & 1101 \\
\hline
& 1001
\end{array}
\]

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:

<table>
<thead>
<tr>
<th>Storage Type</th>
<th>Range (low–high)</th>
<th>Powers of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signed byte</td>
<td>−128 to +127</td>
<td>$-2^7$ to $(2^7 - 1)$</td>
</tr>
<tr>
<td>Signed word</td>
<td>−32,768 to +32,767</td>
<td>$-2^{15}$ to $(2^{15} - 1)$</td>
</tr>
<tr>
<td>Signed doubleword</td>
<td>−2,147,483,648 to 2,147,483,647</td>
<td>$-2^{31}$ to $(2^{31} - 1)$</td>
</tr>
<tr>
<td>Signed quadword</td>
<td>−2,147,483,648 to 2,147,483,647</td>
<td>$-2^{63}$ to $(2^{63} - 1)$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg X$</td>
<td>NOT X</td>
</tr>
<tr>
<td>$X \land Y$</td>
<td>X AND Y</td>
</tr>
<tr>
<td>$X \lor Y$</td>
<td>X OR Y</td>
</tr>
<tr>
<td>$\neg X \lor Y$</td>
<td>( NOT X ) OR Y</td>
</tr>
<tr>
<td>$\neg (X \land Y)$</td>
<td>NOT ( X AND Y )</td>
</tr>
<tr>
<td>$X \land \neg Y$</td>
<td>X AND ( NOT Y )</td>
</tr>
</tbody>
</table>

NOT

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

<table>
<thead>
<tr>
<th>X</th>
<th>$\neg X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

Digital gate diagram for NOT:
AND

• Truth if both are true
• Truth table for Boolean AND operator:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X ∧ Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Digital gate diagram for AND:

OR

• True if either is true
• Truth table for Boolean OR operator:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X ∨ Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Digital gate diagram for OR:

Operator precedence

• NOT > AND > OR
• Examples showing the order of operations:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Order of Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>¬X ∨ Y</td>
<td>NOT, then OR</td>
</tr>
<tr>
<td>¬(X ∨ Y)</td>
<td>OR, then NOT</td>
</tr>
<tr>
<td>X ∨ (Y ∧ Z)</td>
<td>AND, then OR</td>
</tr>
</tbody>
</table>

• 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

<table>
<thead>
<tr>
<th>X</th>
<th>¬X</th>
<th>Y</th>
<th>¬X ∨ Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
### Truth Tables (2 of 3)

- Example: $X \land \neg Y$

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>$\neg Y$</th>
<th>$X \land \neg Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

### Truth Tables (3 of 3)

- Example: $(Y \land S) \lor (X \land \neg S)$

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>S</th>
<th>Y $\land$ S</th>
<th>$\neg S$</th>
<th>X $\land \neg$ S</th>
<th>$(Y \land S) \lor (X \land \neg S)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>