Course overview

Computer Organization and Assembly Languages
Yung-Yu Chuang

with slides by Kip Irvine

Logistics

- Meeting time: 2:20pm-5:20pm, Wednesday
- Classroom: CSIE Room 111
- Instructor: 莊永裕 Yung-Yu Chuang
- Teaching assistant: 黃子桓
- Webpage: http://www.csie.ntu.edu.tw/~cyy/asm
  id / password
- Mailing list: assembly@cmlab.csie.ntu.edu.tw
  Please subscribe via https://cmlmail.csie.ntu.edu.tw/mailman/listinfo/assembly/

Caveats

- It is a course from the old curriculum.
- It is not for you if you have taken or are taking computer architecture.
- It is not tested in your graduate school entrance exam, and not listed as a required course anymore.
- It is a fundamental course, not a geek-level one.
- It is more like advanced introduction to CS, better suited to freshman or sophomore.

Prerequisites

- Better to have programming experience with some high-level languages such C, C++, Java ...
Textbook

- Readings and slides

References (TOY)

Princeton’s Introduction to CS,
http://www.cs.princeton.edu/introcs/50machine/
http://www.cs.princeton.edu/introcs/60circuits/

References (ARM)

ARM Assembly Language Programming, Peter Knaggs and Stephen Welsh

ARM System Developer’s Guide, Andrew Sloss, Dominic Symes and Chris Wright

Whirlwind Tour of ARM Assembly, TONC, Jasper Vijn.

ARM System-on-chip Architecture, Steve Furber.
References (IA32)

Assembly Language for Intel-Based Computers, 5th Edition, Kip Irvine

The Art of Assembly Language, Randy Hyde

References (IA32)

Michael Abrash' s Graphics Programming Black Book

Computer Systems: A Programmer's Perspective, Randal E. Bryant and David R. O'Hallaron

Grading (subject to change)

- Assignments (4 projects, 56%), most graded by performance
- Class participation (4%)
- Midterm exam (16%)
- Final project (24%)
  - Examples from previous years

Computer Organization and Assembly language.

- It is not only about assembly but also about “computer organization”.
Early computers

First popular PCs

Early programming tools

Early PCs

- Intel 8086 processor
- 768KB memory
- 20MB disk
- Dot-Matrix printer (9-pin)
GUI/IDE

More advanced architectures
- Pipeline
- SIMD
- Multi-core
- Cache

More advanced software

More “computers” around us
My computers

- Desktop (Intel Pentium D 3GHz, Nvidia 7900)
- VAIO Z46TD (Intel Core 2 Duo P9700 2.8GHz)
- GBA SP (ARM7 16.78MHz)
- iPhone 3GS (ARM Cortex-A8 833MHz)

Computer Organization and Assembly language

- It is not only about assembly but also about “computer organization”.
- It will cover
  - Basic concept of computer systems and architecture
  - ARM architecture and assembly language
  - x86 architecture and assembly language

TOY machine

- Starting from a simple construct
TOY machine

- Build several components and connect them together

TOY machine

- Almost as good as any computers

```
int A[32];

A   DUP  32  10: C020

lda R1, 1  20: 7101
lda RA, A   21: 7A00
lda RC, 0   22: 7C00

i=0;
Do {
  RD=stdin;
  read ld RD, 0xFF   23: 8DFF
  if (RD==0) break;
  bz RD, exit 24: CD29
  add R2, RA, RC  25: 12AC
  A[i]=RD;
  sti RD, R2 26: BD02
  i=i+1;
  add RC, RC, R1 27: 1CC1
} while (1);

printr();
  exit jl RF, printr 29: FF2B
  hlt 2A: 0000
```

ARM

- ARM architecture
- ARM assembly programming
IA32

- IA-32 Processor Architecture
- Data Transfers, Addressing, and Arithmetic
- Procedures
- Conditional Processing
- Integer Arithmetic
- Advanced Procedures
- Strings and Arrays
- High-Level Language Interface
- Real Arithmetic (FPU)
- SIMD
- Code Optimization

What you will learn

- Basic principle of computer architecture
- How your computer works
- How your C programs work
- Assembly basics
- ARM assembly programming
- IA-32 assembly programming
- Specific components, FPU/MMX
- Code optimization
- Interface between assembly to high-level language

Why taking this course?

- Does anyone really program in assembly nowadays?
  - Yes, at times, you do need to write assembly code.
  - It is foundation for computer architecture and compilers. It is related to electronics, logic design and operating system.

CSIE courses

- Hardware: electronics, digital system, architecture
- Software: operating system, compiler
Today, assembly language is used primarily for direct hardware manipulation, access to specialized processor instructions, or to address critical performance issues. Typical uses are device drivers, low-level embedded systems, and real-time systems.

Reasons for not using assembly
- Development time: it takes much longer to develop in assembly. Harder to debug, no type checking, side effects...
- Maintainability: unstructured, dirty tricks
- Portability: platform-dependent

Reasons for using assembly
- Educational reasons: to understand how CPUs and compilers work. Better understanding to efficiency issues of various constructs.
- Developing compilers, debuggers and other development tools.
- Hardware drivers and system code
- Embedded systems
- Developing libraries.
- Accessing instructions that are not available through high-level languages.
- Optimizing for speed or space

To sum up
- 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, e.g. GPU, DSP chips, even MMX.
Overview

- Virtual Machine Concept
- Data Representation
- Boolean Operations

Translating languages

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

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

Intel Machine Language:
A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000

Assembly Language:
mov eax,A
mul B
add eax,C
call WriteInt

Virtual machines

Abstractions for computers

High-level language

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

cout << (A * B + C);
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, A
  mul B
  add eax, C
  call WriteInt
  ```

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 representations

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

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:

```
1 0 1 1 0 0 1 0 1 0 0 1 1 1 0 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:

<table>
<thead>
<tr>
<th>$2^n$</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^0$</td>
<td>1</td>
</tr>
<tr>
<td>$2^1$</td>
<td>2</td>
</tr>
<tr>
<td>$2^2$</td>
<td>4</td>
</tr>
<tr>
<td>$2^3$</td>
<td>8</td>
</tr>
<tr>
<td>$2^4$</td>
<td>16</td>
</tr>
<tr>
<td>$2^5$</td>
<td>32</td>
</tr>
<tr>
<td>$2^6$</td>
<td>64</td>
</tr>
<tr>
<td>$2^7$</td>
<td>128</td>
</tr>
</tbody>
</table>

Every binary number is a sum of powers of 2

---

**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:

<table>
<thead>
<tr>
<th>Division</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>37 / 2</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>18 / 2</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>9 / 2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4 / 2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2 / 2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 / 2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

37 = 100101

---

**Binary addition**

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

```
| 0 | 0 | 0 | 0 | 1 | 0 | 0 |
+ 0 | 0 | 0 | 0 | 1 | 1 | 1 |
-----------------------------
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
```

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

Standard sizes:

- **byte**: 8
- **word**: 16
- **doubleword**: 32
- **quadword**: 64

<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?

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>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Binary</th>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
<td>1001</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
<td>1010</td>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
<td>1011</td>
<td>11</td>
<td>B</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
<td>1100</td>
<td>12</td>
<td>C</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
<td>1101</td>
<td>13</td>
<td>D</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
<td>1110</td>
<td>14</td>
<td>E</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
<td>1111</td>
<td>15</td>
<td>F</td>
</tr>
</tbody>
</table>

Translating binary to hexadecimal

- Each hexadecimal digit corresponds to 4 binary bits.
- Example: Translate the binary integer 00010110101011110010100 to hexadecimal:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>1</td>
<td>0110</td>
</tr>
<tr>
<td>1010</td>
<td>6</td>
<td>1010</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>0101</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>0100</td>
</tr>
</tbody>
</table>
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(^3)</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.

\[
\begin{array}{cccc}
36 & 28 & 28 & 6A \\
42 & 45 & 58 & 4B \\
78 & 6D & 80 & B5 \\
\end{array}
\]

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 \texttt{var1} is 00400020. The address of the next variable after \texttt{var1} is 0040006A. How many bytes are used by \texttt{var1}?

Signed integers

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

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 +00000001</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 \((-B)\)

\[
\begin{array}{c|c|c}
01010 & 01010 \\
- 01011 & 10100 \\
\hline & 11111
\end{array}
\]

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

Representing Instructions

```c
int sum(int x, int y)
{
    return x+y;
}
```
- For this example, Alpha & Sun use two 4-byte instructions
  - Use differing numbers of instructions in other cases
- PC uses 7 instructions with lengths 1, 2, and 3 bytes
  - Same for NT and for Linux
  - NT / Linux not fully binary compatible

```plaintext
Different machines use totally different instructions and encodings
```

Boolean algebra

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>¬X</td>
<td>NOT X</td>
</tr>
<tr>
<td>X ∧ Y</td>
<td>X AND Y</td>
</tr>
<tr>
<td>X ∨ Y</td>
<td>X OR Y</td>
</tr>
<tr>
<td>¬(X ∨ Y)</td>
<td>( NOT X ) OR Y</td>
</tr>
<tr>
<td>¬(X ∧ Y)</td>
<td>NOT ( X AND Y )</td>
</tr>
<tr>
<td>X ∧ ¬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>¬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>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Digital gate diagram for OR:

### Implementation of gates

- Fluid switch [link](http://www.cs.princeton.edu/introcs/lectures/fluid-computer.swf)
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: \( \neg X \lor Y \)

<table>
<thead>
<tr>
<th>( X )</th>
<th>( \neg X )</th>
<th>( Y )</th>
<th>( \neg X \lor 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>

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>