Introduction: Hello, World Below

Usage and Copyright Notice:

Copyright 2005 © Noam Nisan and Shimon Schocken


We provide both PPT and PDF versions.

The book web site, www.idc.ac.il/tecs, features 13 such presentations, one for each book chapter. Each presentation is designed to support about 3 hours of classroom or self-study instruction.

You are welcome to use or edit this presentation as you see fit for instructional and non-commercial purposes.

If you use our materials, we will appreciate it if you will include in them a reference to the book’s web site.

If you have any questions or comments, you can reach us at tecs.ta@gmail.com
The course at a glance

Objectives:

- Understand how hardware and software systems are built, and how they work together
- Learn how to break complex problems into simpler ones
- Learn how large scale development projects are planned and executed
- Have fun

Methodology:

- Build a complete, general-purpose, and working computer system
- Play and experiment with this computer, at any level of interest.
Some course details

- 12 projects, can be done by pairs
- Hardware projects are done and simulated in HDL (Hardware Description Language)
- Software projects can be done in any programming language of your choice (we recommend Java)
- Projects methodology:
  - Design (API) + test materials are given
  - Implementation done by students
- Tools: simulators, tutorials, test scripts
- Book
- Q&A policy
- Exam.
Demo

Pong, 1985

Pong, 2011

Pong, on our computer
Course theme and structure

(Human Thought)

Abstract design
- Abstract interface
  - Chapters 9, 12

H.L. Language & Operating Sys.
- Chapters 10 - 11

Compiler
- Virtual Machine
  - Chapters 7 - 8

VM Translator
- Assembly Language

Assembler
- Chapter 6

Machine Language
- Abstract interface
  - Chapters 4 - 5

Computer Architecture
- Hardware Platform
  - Abstract interface
  - Chapters 1 - 3

Gate Logic
- Chips & Logic Gates
  - Abstract interface

Chips & Logic Gates
- Electrical Engineering

Physics

Hardware hierarchy

Software hierarchy

(Abstraction–implementation paradigm)
Application level: (say,) Pong
The big picture

- **Human Thought**
  - Abstract design
      - Abstract interface
        - Compiler
          - Abstract interface
            - Virtual Machine
              - Abstract interface
                - VM Translator
                  - Abstract interface
                    - Assembly Language
                      - Abstract interface
            - Assembler
              - Abstract interface
                - Machine Language
                  - Abstract interface
                    - Computer Architecture
                      - Abstract interface
                        - Hardware Platform
                          - Abstract interface
                            - Gate Logic
                              - Abstract interface
                                - Chips & Logic Gates
                                  - Abstract interface
                                    - Electrical Engineering
                                      - Physics
High-level programming (Jack language)

```java
/** A Graphic Bat for a Pong Game */
class Bat {
    field int x, y;            // screen location of the bat's top-left corner
    field int width, height;   // bat's width & height

    // The class constructor and most of the class methods are omitted

/** Draws (color=true) or erases (color=false) the bat */
method void draw(boolean color) {
    do Screen.setColor(color);
    do Screen.drawRectangle(x,y,x+width,y+height);
    return;
}

/** Moves the bat one step (4 pixels) to the right. */
method void moveR() {
    do draw(false);  // erase the bat at the current location
    let x = x + 4;   // change the bat's X-location
    // but don't go beyond the screen's right border
    if ((x + width) > 511) {
        let x = 511 - width;
    }
    do draw(true);  // re-draw the bat in the new location
    return;
}
}
```
/** An OS-level screen driver that abstracts the computer's physical screen */
class Screen {
    static boolean currentColor; // the current color

    // The Screen class is a collection of methods, each implementing one
    // abstract screen-oriented operation. Most of this code is omitted.

    /** Draws a rectangle in the current color. */
    // the rectangle's top left corner is anchored at screen location (x0,y0)
    // and its width and length are x1 and y1, respectively.
    function void drawRectangle(int x0, int y0, int x1, int y1) {
        var int x, y;
        let x = x0;
        while (x < x1) {
            let y = y0;
            while(y < y1) {
                do Screen.drawPixel(x,y);
                let y = y+1;
            }
            let x = x+1;
        }
    }
}
A modern compilation model

RISC
VM language

CISC machine

RISC machine

other digital platforms, each equipped with its VM implementation

Any computer

Projects 1-6

Projects 7-8

Projects 10-11

VM imp. over the Hack platform

VM imp. over RISC platforms

VM imp. over CISC platforms

VM language

Some language

Some compiler

Some Other language

Some Other compiler

Jack language

Jack compiler

Some Other language

Some compiler

Jack language

Projects 9: building an app.

Projects 12: building the OS
Compilation 101

Observations:
- Modularity
- Abstraction / implementation interplay
- The implementation uses abstract services from the level below.
The Virtual Machine (our stack-based VM, quite similar to Java’s JVM)

if ((x+width)>511) {
    let x=511-width;
}

// VM implementation
push x       // s1
push width   // s2
add          // s3
push 511     // s4
gt           // s5
if-goto L1   // s6
goto L2      // s7
L1:
    push 511   // s8
    push width // s9
    sub         // s10
    pop x       // s11
L2:
    ...

memory (before)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>450</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

memory (after)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>450</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Low-level programming (on the Hack computer)

Virtual machine program

\[
\ldots \begin{align*}
&\text{push } x \\
&\text{push width} \\
&\text{add} \\
&\text{push 511} \\
&\text{gt} \\
&\text{if-goto L1} \\
&\text{goto L2} \\
L1: &\begin{align*}
&\text{push 511} \\
&\text{push width} \\
&\text{sub} \\
&\text{pop } x \\
L2: &\ldots
\end{align*}
\end{align*}
\]

Assembly program

\[
\begin{align*}
// &\text{ push 511} \\
&@511 \\
&D=A // D=511 \\
&@SP \\
&A=M \\
&M=D // *SP=D \\
&@SP \\
&M=M+1 // SP++
\end{align*}
\]

VM translator

Assembler

Executable

For now, ignore all details!
The big picture

**Human Thought**

- Abstract design
  - Chapters 9, 12
- Machine Language
  - Chapters 4 - 5
- Computer Architecture
  - Chapter 6
- Hardware Platform
- Gate Logic
  - Chapters 1 - 3

**Hardware hierarchy**

**Software hierarchy**

  - Chapters 10 - 11
- Compiler
- Virtual Machine
  - Chapters 7 - 8
- VM Translator
- Assembly Language
- Assembler
  - Chapter 6

**Electrical Engineering**

- Chips & Logic Gates
  - Abstract interface
- Electrical Engineering

**Physics**
Machine language semantics (Hack platform)

We need a hardware architecture that realizes this semantics

The hardware platform should be designed to:

• Parse instructions, and
• Execute them.

For now, ignore all details!
A typical Von Neumann machine

For now, ignore all details!
The big picture

Abstract design

Human Thought
Chapters 9, 12

H.L. Language & Operating Sys.
Chapters 10 - 11

Compiler

VM Translator
Chapters 7 - 8

Virtual Machine

Assembly Language

Assembler
Chapter 6

Machine Language
Chapters 4 - 5

Computer Architecture

Gate Logic
Chapters 1 - 3

Hardware Platform

Chips & Logic Gates

Electrical Engineering

Physics

Hardware hierarchy

Software hierarchy
Logic design

- **Combinational logic** (leading to an **ALU**)

- **Sequential logic** (leading to a **RAM**)

- **Putting the whole thing together** (leading to a **Computer**)

Using ... *gate logic.*
Gate logic

- Hardware platform = inter-connected set of chips
- Chips are made of simpler chips, all the way down to logic gates
- Logic gate = hardware element that implements a certain Boolean function
- Every chip and gate has an interface, specifying WHAT it is doing, and an implementation, specifying HOW it is doing it.

**Interface**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Implementation**
Hardware Description Language (HDL)

CHIP Xor {
    IN a,b;
    OUT out;
    PARTS:
    Not (in=a, out=Nota);
    Not (in=b, out=Notb);
    And (a=a, b=Notb, out=w1);
    And (a=Nota, b=b, out=w2);
    Or (a=w1, b=w2, out=out);
}
The tour ends:

Interface

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

One implementation option (CMOS)
The tour map, revisited

Course overview: Building this world, from the ground up

Abstract design
H.L. Language & Operating Sys.
Compiler
Virtual Machine
VM Translator
Assembly Language
Assembler
Computer Architecture
Machine Language
Hardware Platform
Gate Logic
Chips & Logic Gates
Electric Engineering
Physics

Hardware hierarchy

Software hierarchy

Human Thought

Abstract interface

Chapters 9, 12

Chapters 10 - 11

Chapters 7 - 8

Chapter 6

Chapters 4 - 5

Chapters 1 - 3