Introduction:
From Nand to Tetris

Building a Modern Computer From First Principles

www.nand2tetris.org
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 nand2tetris details

- 12 projects (We’ll probably do 5 or 6)
- Hardware projects are done and simulated in HDL (Hardware Description Language)
- Software projects can be done in any 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
Demo

Pong, 1985

Pong, 2011

Pong, on our computer
Course theme and structure

Human Thought

Abstract design

Chapter 6

H.L. Language & Operating Sys.

Chapter 10 - 11

Virtual Machine

Chapter 7 - 8

Assembly Language

Compiler

Chapter 11

VM Translator

Assembler

Chapter 6

Computer Architecture

Chapter 4 - 5

Machine Language

Chapter 3

Hardware Platform

Chapter 1 - 3

Chips & Logic Gates

Electrical Engineering

Physics

Hardware hierarchy

Abstract design

Machine Language

Abstract interface

Hardware Platform

Abstract interface

Gate Logic

Abstract interface

Chips & Logic Gates

Abstract interface

VM Translator

Abstract interface

Software hierarchy

(Habstraction–implementation paradigm)
Application level: Pong (example app)
The big picture

Human Thought

Abstract design

H.L. Language & Operating Sys.

Compiler

Virtual Machine

VM Translator

Assembly Language

Hardware hierarchy

Machine Language

Computer Architecture

Hardware Platform

Gate Logic

Chips & Logic Gates

Electrical Engineering

Physics

Software hierarchy

Abstract interface

Assembly

Operating Sys.

abstract interface

Chapters 4 - 5

Chapters 6

Chapters 7 - 8

Chapters 10 - 11

Chapters 9, 12

High-level programming (our very own 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;
    }
}
```

Typical call to an OS method
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

Some language \rightarrow Some compiler
Some Other language \rightarrow Some Other compiler
Jack language \rightarrow Jack compiler

\begin{center}
\begin{itemize}
\item VM language
\item VM implementation over CISC platforms
\item VM imp. over RISC platforms
\item VM emulator
\item VM imp. over the Hack platform
\item CISC machine language
\item RISC machine language
\item written in a high-level language
\item Hack machine language
\end{itemize}
\end{center}

\begin{center}
\begin{itemize}
\item CISC machine
\item RISC machine
\item other digital platforms, each equipped with its VM implementation
\item Any computer
\item Hack computer
\end{itemize}
\end{center}

Projects 7-8
Projects 10-11
Projects 1-6

Proj. 9: building an app.
Proj. 12: building the OS
Compilation 101

Observations:

- Modularity
- Abstraction / implementation interplay
- The implementation uses abstract services from the level below.
The Virtual Machine (our very own VM, modeled after 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></td>
<td></td>
<td></td>
</tr>
<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>
```
The big picture

Human Thought

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

Hardware hierarchy

Electrical Engineering

Physics
Low-level programming (on the Hack computer)

Virtual machine program

```
...  
push x
push width
add
push 511
gt
if-goto L1
goto L2
L1:
push 511
push width
sub
pop x
L2:
...
```

For now, ignore all details!
Low-level programming (on the Hack computer)

Virtual machine program

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

Assembly program

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

For now, ignore all details!
Low-level programming (on the Hack computer)

Virtual machine program

...  
push x  
push width  
add  
push 511  
gt  
if-goto L1  
goto L2  
L1:  
push 511  
push width  
sub  
pop x  
L2:  
...

Assembly program

// push 511  
@511  
goto L2  
L1:  
push 511  
push width  
sub  
pop x  
L2:  
...

For now, ignore all details!
The big picture

Human Thought

Abstract design

H.L. Language & Operating Sys.

Compiler

Virtual Machine

VM Translator

Assembly Language

Human Thought

Machine Language

Computer Architecture

Hardware Platform

Gate Logic

Chips & Logic Gates

Electrical Engineering

Physics

Hardware hierarchy

Software hierarchy

Abstract design

Chapters 9, 12

Compiler

Chapters 10 - 11

VM Translator

Chapters 7 - 8

Chapters 4 - 5

Chapters 1 - 3

Chapters 6
Machine language semantics (our very own Hack platform)

Code syntax

```
0000000000000000
1111110111001000
```

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!
Computer architecture (Hack platform, approx.)

- A typical Von Neumann machine

For now, ignore all details!
The big picture

**Human Thought**

- Abstract design
  - Chapters 9, 12

**Abstract interface**

- Compiler
  - Chapters 10 - 11

**Virtual Machine**

- VM Translator
  - Chapters 7 - 8

**Operating Sys.**

- Abstract interface

**Assembly Language**

**Hardware hierarchy**

- Machine Language
  - Abstract interface
  - Chapters 4 - 5

- Computer Architecture
  - Abstract interface
  - Chapter 6

- Hardware Platform
  - Abstract interface
  - Chapters 4 - 5

- Gate Logic
  - Abstract interface
  - Chapters 1 - 3

- Chips & Logic Gates
  - Abstract interface

**Software hierarchy**

- Assembler
  - Chapter 6

- Electrical Engineering

- Physics
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 elementary 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.

<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>

**Interface**

**Implementation**

```plaintext
Gate logic

- Hardware platform = inter-connected set of chips
- Chips are made of simpler chips, all the way down to elementary 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.

<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>

**Interface**

**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);
    }

\[
\begin{align*}
&a \\
&\text{Not} \rightarrow \text{And} \rightarrow \text{Or} \\
&b \\
&\text{Not} \rightarrow \text{And} \rightarrow \text{Or}
\end{align*}
\]
The tour ends:

Interface

<table>
<thead>
<tr>
<th></th>
<th></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
Human Thought
Chapters 9, 12

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

Compiler

Virtual Machine
abstract interface
Chapters 7 - 8

VM Translator

Assembly Language

Assembler
Chapter 6

Computer Architecture
abstract interface
Chapters 4 - 5

Machine Language

Hardware Platform
abstract interface
Chapters 1 - 3

Gate Logic

Chips & Logic Gates
abstract interface

Electrical Engineering

Physics

Hardware hierarchy

Software hierarchy