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 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 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

- Course grade.
Demo

Pong, 1985  Pong, 2011

Pong, on our computer
Course theme and structure

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

Human Thought

Abstract design

Chapters 9, 12

H.L. Language & Operating Sys.

Compiler

Chapters 10 - 11

Virtual Machine

VM Translator

Chapters 7 - 8

Assembly Language

Assembler

Chapter 6

Machine Language

Computer Architecture

Chapters 4 - 5

Hardware Platform

Gate Logic

Chapters 1 - 3

Chips & Logic Gates

Electrical Engineering

Physics

Abstract design

hardware hierarchy

Software hierarchy

Hardware hierarchy
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;
    }
}
```
Operating system level (our very own Jack OS)

```java
/** 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;
        }
    }
}
```
The big picture

Human Thought

Abstract design

H.L. Language & Operating Sys.

 Compiler

Virtual Machine

Assembly Language

VM Translator

Assembler

Computer Architecture

Machine Language

Hardware Platform

Gate Logic

Chips & Logic Gates

Electrical Engineering

Physics

Hardware hierarchy

Software hierarchy
A modern compilation model

- Some language
- Some Other language
- Jack language
  - Some compiler
  - Some Other compiler
  - Jack compiler
  - VM language
    - VM implementation over CISC platforms
    - VM imp. over RISC platforms
    - VM imp. over the Hack platform
      - CISC machine language
      - RISC machine language
      - written in a high-level language
      - Hack machine language
        - Hack computer

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

```java
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):

- `x`: 75
- `width`: 450
- `...`

Memory (after):

- `x`: 61
- `width`: 450
- `...`
The big picture

Human Thought
- Chapters 9, 12

Abstract design

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

Compiler

Virtual Machine
- Chapters 7 - 8

VM Translator

Assembly Language

Assembler
- Chapter 6

Computer Architecture
- Chapters 4 - 5

Hardware Platform

Gate Logic
- Chapters 1 - 3

Chips & Logic Gates

Electrical Engineering

Physics

Hardware hierarchy

Machine Language
- Chapters 4 - 5

Electrical Engineering

Software hierarchy

Virtual Machine

H.L. Language & Operating Sys.

Compiler

VM Translator

Assembly Language

Assembler

Computer Architecture

Hardware Platform

Gate Logic

Chips & Logic Gates

Electrical Engineering

Physics

Human Thought

Abstract design

H.L. Language & Operating Sys.

Compiler

Virtual Machine

VM Translator

Assembly Language

Assembler

Computer Architecture

Hardware Platform

Gate Logic

Chips & Logic Gates

Electrical Engineering

Physics
Low-level programming (on the Hack computer)

Virtual machine program

```plaintext
... 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!
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:  
...

VM translator

Assembly program

// push 511  
@511  
goto L2  
L1:  
push 511  
@511  
D=A  // D=511  
@SP  
A=M  
M=D  // *SP=D  
@SP  
M=M+1  // SP++  

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
D=A
@SP
A=M
M=D
// *SP=D @SP
M=M+1 // SP++
Machine language semantics (our very own Hack platform)

**Code semantics**, as interpreted by the Hack hardware platform

- **Code syntax**

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **Instruction code**
  - 0 = “address” inst.
  - 1 = “compute” inst.

- **Address**

- **ALU operation code**

- **Destination Code**

- **Jump Code**
  - (M-1)
  - (M)
  - (no jump)

- **@0**

- **M=M-1**

---

- **We need a hardware architecture that realizes this semantics**
- **The hardware platform should be designed to:**
  - Parse instructions, and
  - Execute them.
Computer architecture (Hack platform, approx.)

A typical Von Neumann machine

For now, ignore all details!
The big picture

Abstract design
Chapters 9, 12

Hardware hierarchy

Human Thought

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

Compiler

Virtual Machine
Chapters 7 - 8

Virtual Machine

VM Translator
Chapters 7 - 8

Assembly Language

Software hierarchy

Computer Architecture
Chapters 4 - 5

Machine Language

abstract interface

abstract interface

abstract interface

Gate Logic
Chapters 1 - 3

Chips & Logic Gates

abstract interface

abstract interface

Hardware Platform

abstract interface

abstract interface

Assembler
Chapter 6

Electrical Engineering

Physics

Human Thought

Introduction
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.

### 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);
}

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

<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>

**Interface**

**One implementation option (CMOS)**
The tour map, revisited

Course overview: Building this world, from the ground up