Course overview

Introduction to Computer

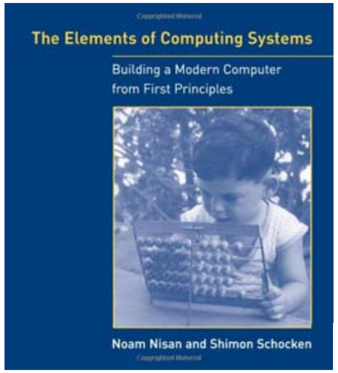
Yung-Yu Chuang

*with slides by Nisan & Schocken (*www.nand2tetris.org*)*

Logistics

- **Meeting time**: 2:20pm-5:20pm, Tuesday
- **Classroom**: CSIE Room 104
- **Instructor**: 莊永裕 Yung-Yu Chuang
- **Teaching assistant**: TBD
- **Webpage**:
  http://www.csie.ntu.edu.tw/~cyy/introcs
  id / password
- **Mailing list**: introcs@cmlab.csie.ntu.edu.tw
  Please subscribe via
  https://cmlmail.csie.ntu.edu.tw/mailman/listinfo/introcs/

Textbook

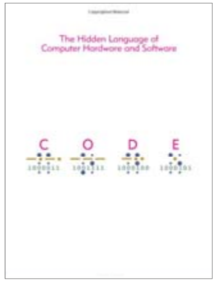


The Elements of Computing Systems, Noam Nisan, Shimon Schocken, MIT Press

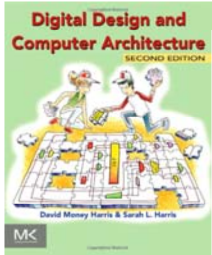
References (TOY)

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

References



CODE: The Hidden Language of Computer Hardware and Software, Charles Petzold, Microsoft Press.

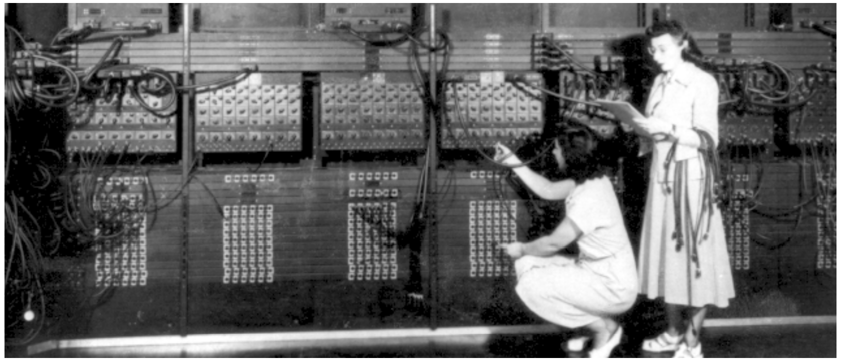


Digital Design and Computer Architecture, 2nd Edition, David Harris and Sarah Harris, Morgan Kaufmann.

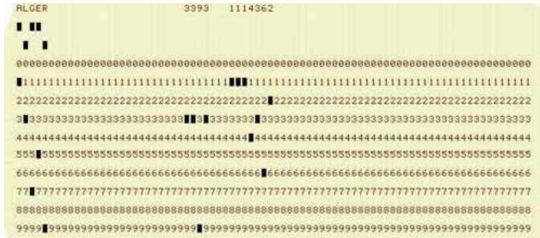
Grading (subject to change)

- Assignments (*n* projects, 60%) from the accompanying website
- Class participation (4%)
- Midterm exam (16%)
- Final project (20%)

Early computers



Early programming tools

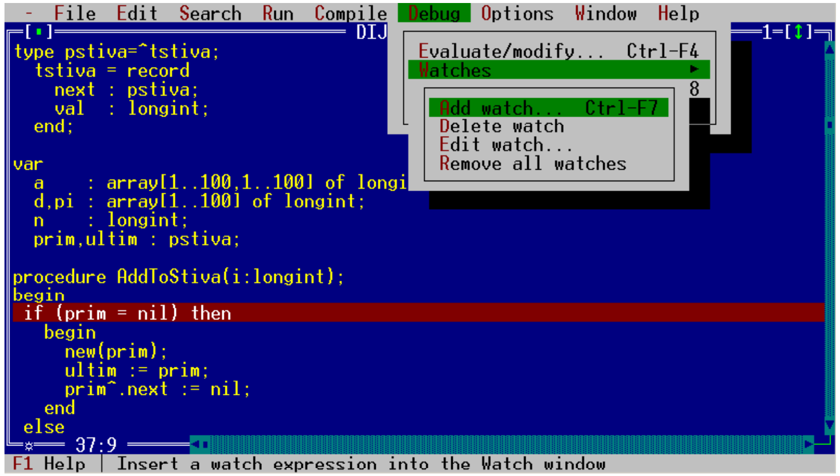


First popular PCs

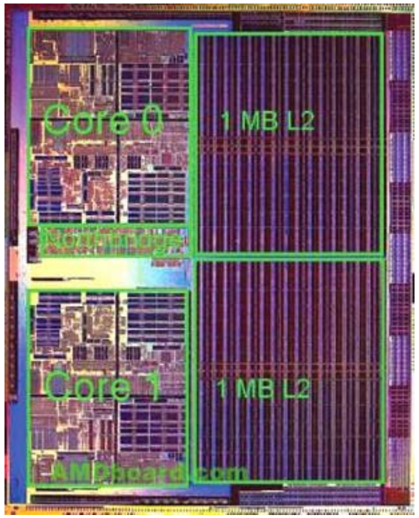
Early PCs

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

GUI/IDE

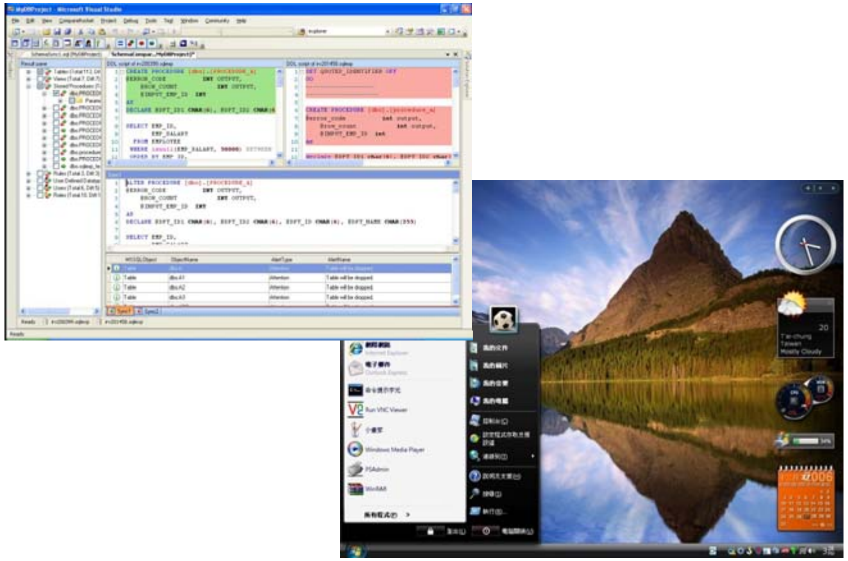


More advanced architectures



- 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)

iPad 2
(dual-core A5 1GHz)

iPhone 5
(A6,
ARM Cortex-A15?)

Goal of the course

The course at a glance

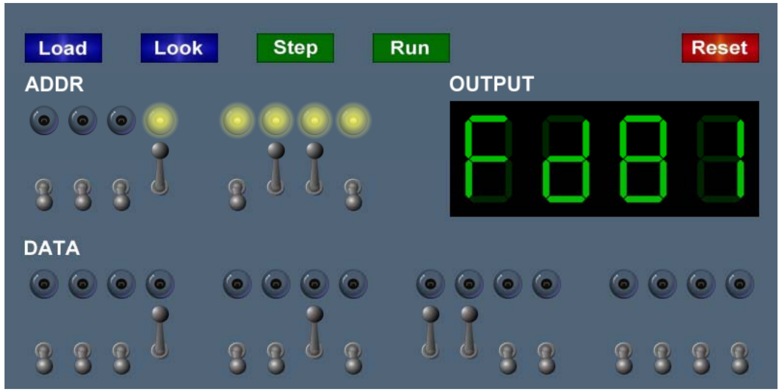
Objectives:

- 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:

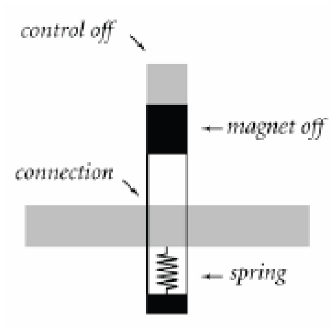
- Build a complete, general-purpose and working computer system
- Play and experiment with this computer, at any level of interest

TOY machine



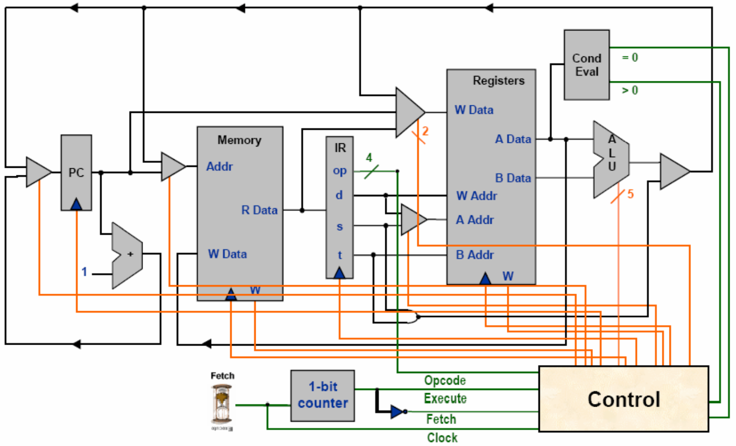
TOY machine

- Starting from a simple construct



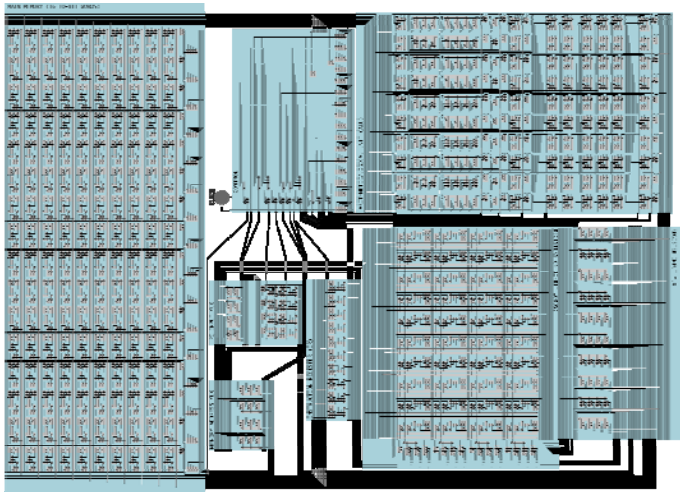
TOY machine

- Build several components and connect them together



TOY machine

- Almost as good as any computers

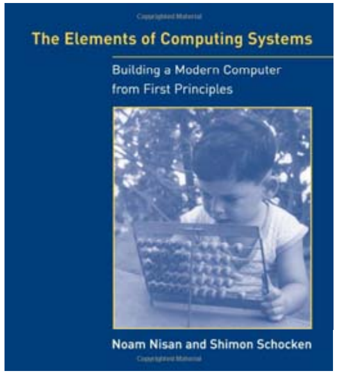


TOY machine

int A[32];	A	DUP	32	10: C020
		lda	R1, 1	20: 7101
		lda	RA, A	21: 7A00
i=0;		lda	RC, 0	22: 7C00
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);		bz	R0, read	28: C023
printr();	exit	jl	RF, printr	29: FF2B
		hlt		2A: 0000

From NAND to Tetris

- The elements of computing systems
- Courses
- Software
- Cool stuffs

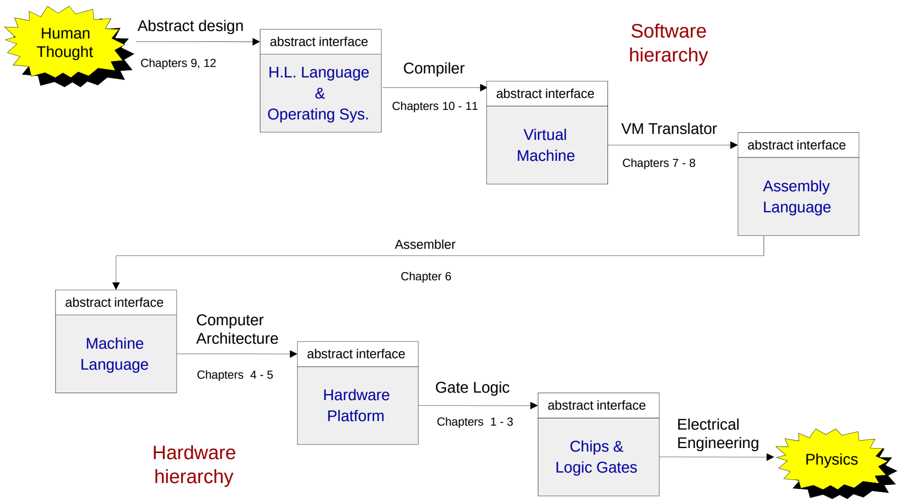


Pong on the Hack computer

Pong, 1985

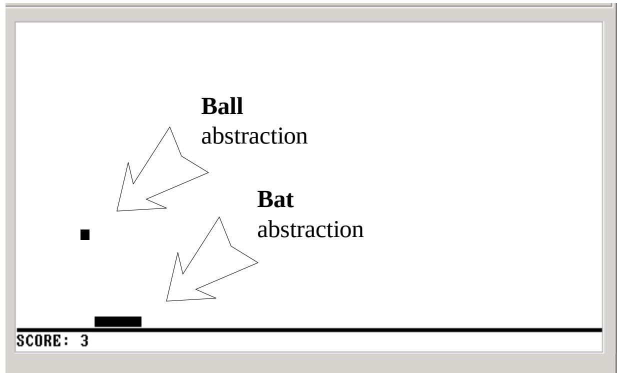
Pong, 2011

Theme and structure of the book

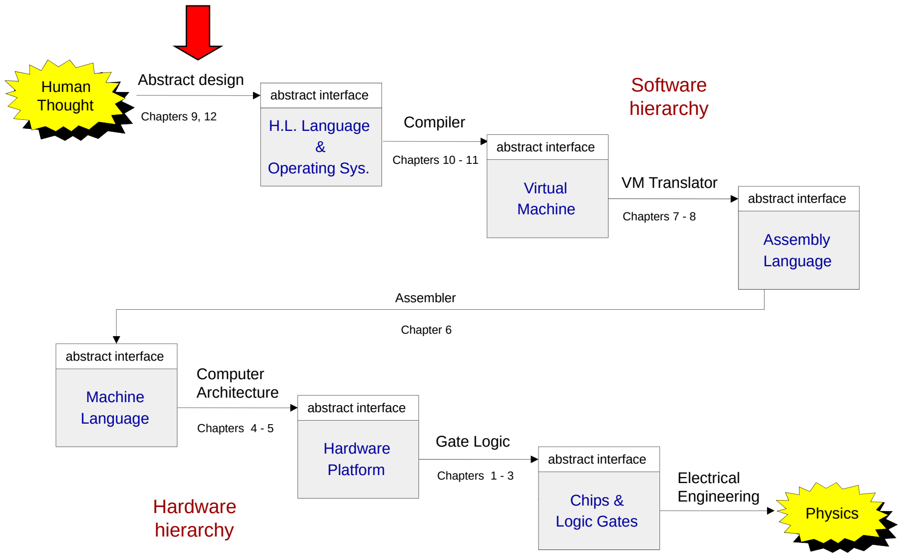


(Abstraction–implementation paradigm)

Application level: Pong (an example)



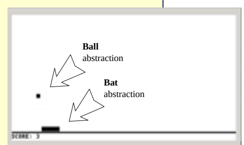
The big picture



High-level programming (Jack language)

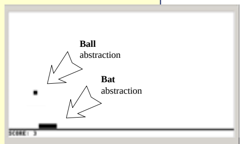
```
/** A Graphic Bat for a Pong Game */
class Bat {
    field int x, y;            // screen location of the bat's top-left corner
    field int width, height;   // bat's width & height

    // 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

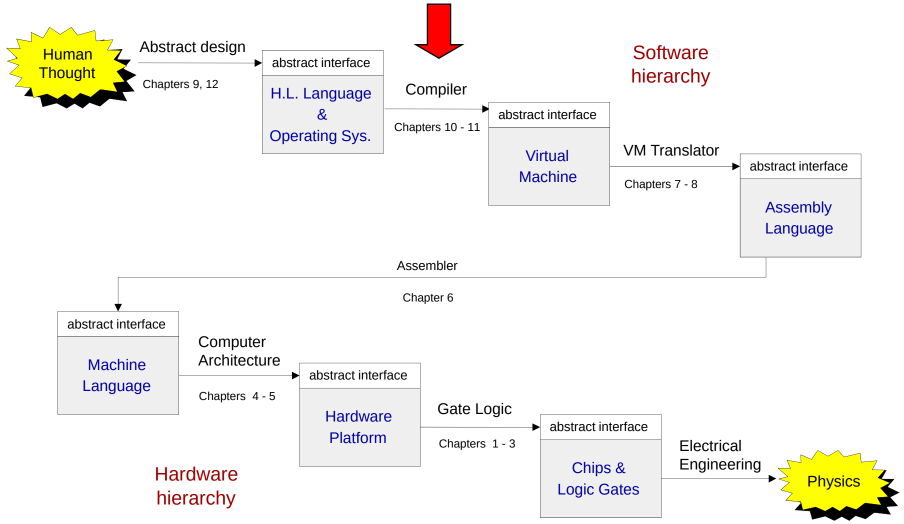


Operating system level (Jack OS)

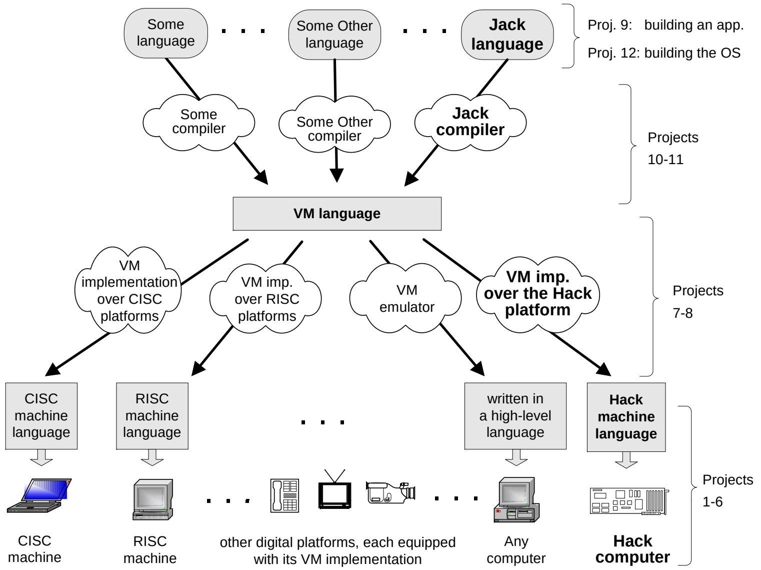
```
/** An OS-level screen driver that abstracts the computer's physical screen */
class Screen {
     static boolean currentColor;  // the current color

     // 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;
         }
     }
}
```



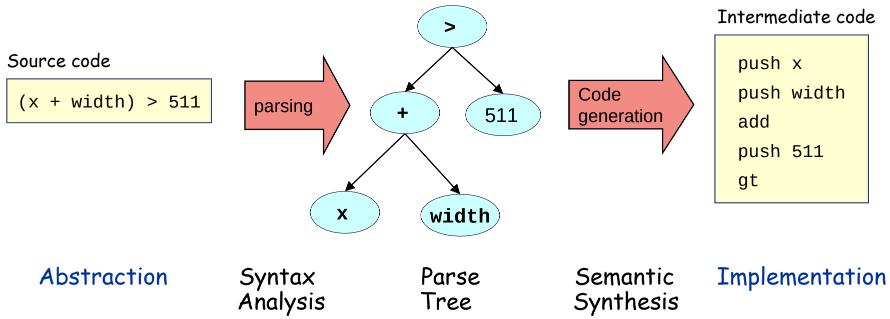
The big picture



A modern compilation model



Compilation 101



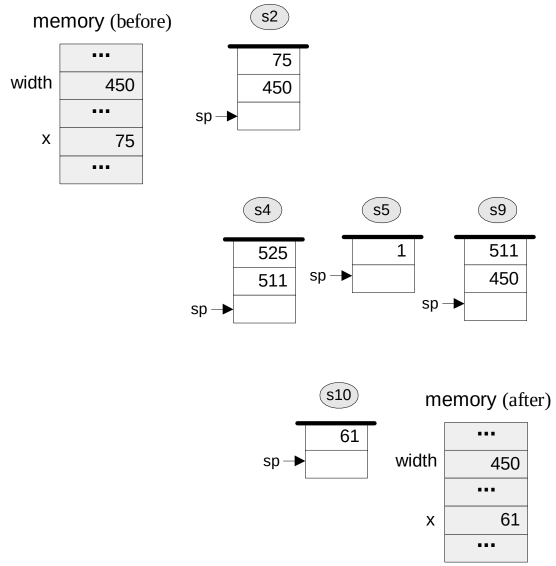
Observations:

- Modularity
- Abstraction / implementation interplay
- The implementation uses abstract services from the level below.

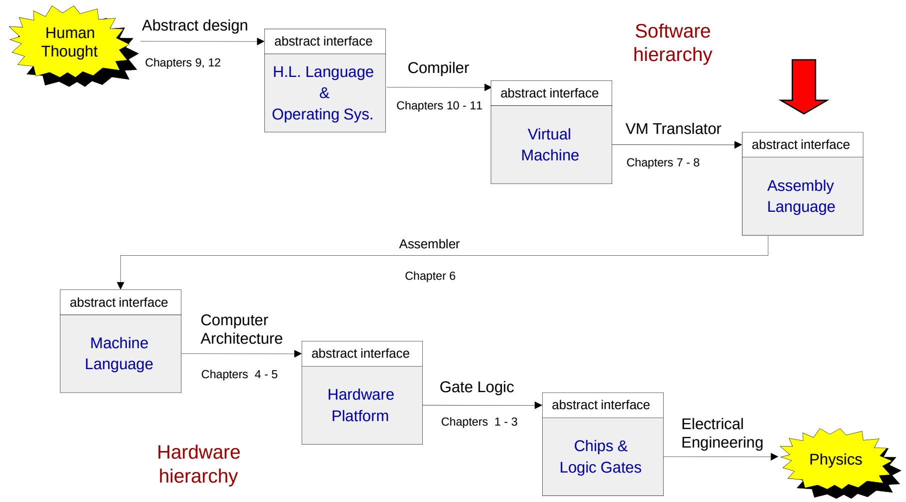
The virtual machine (VM modeled after JVM)

```
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:
...
```



The big picture



Low-level programming (on Hack)

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:
...
```

Low-level programming (on Hack)

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
D=A   // D=511
@SP
A=M
M=D   // *SP=D
@SP
M=M+1 // SP++
```

Low-level programming (on Hack)

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

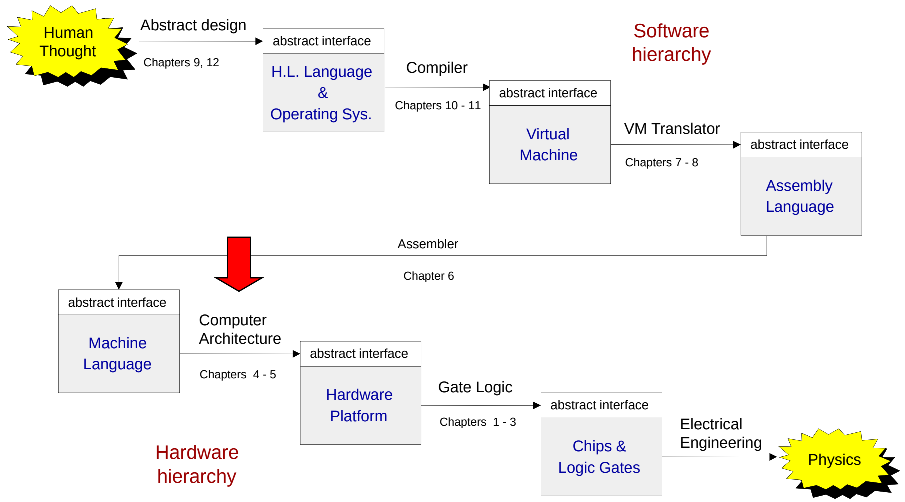
```
// push 511
@511
D=A   // D=511
@SP
A=M
M=D   // *SP=D
@SP
M=M+1 // SP++
```

Assembler

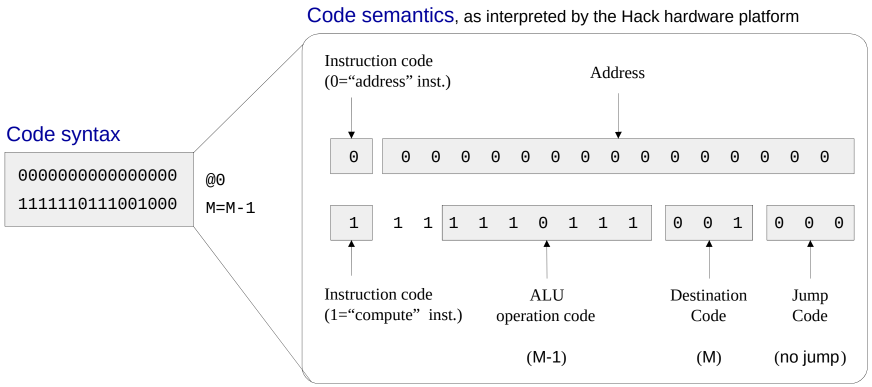
Executable

```
0000000000000000
1110110010001000
```

The big picture

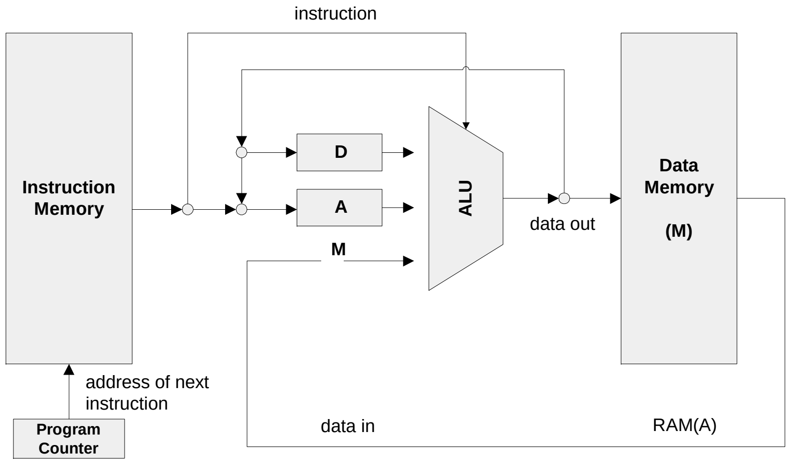


Machine language semantics (Hack)



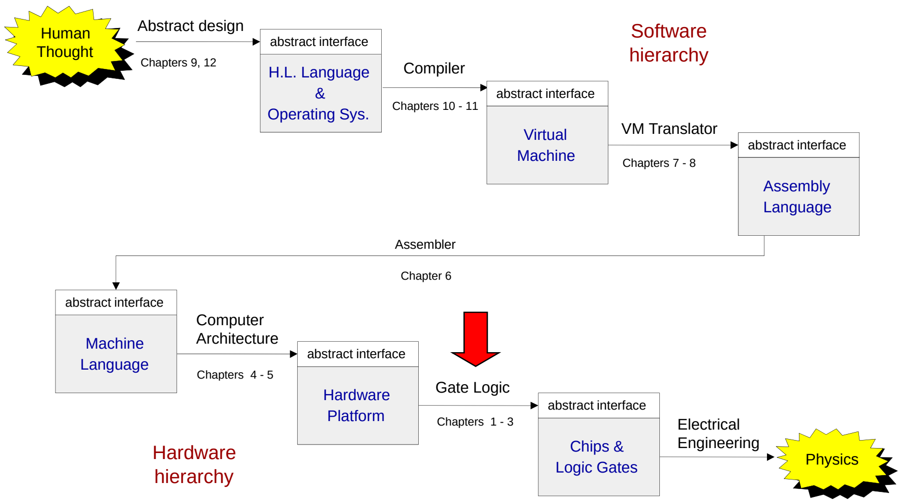
- We need a hardware architecture that realizes this semantics
- The hardware platform should be designed to:
  - Parse instructions, and
  - Execute them.

Computer architecture (Hack)



- A typical Von Neumann machine

The big picture



Logic design

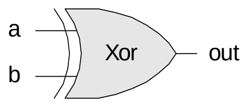
- Combinational logic (leading to an ALU)
- Sequential logic (leading to a RAM)
- Putting the whole thing together (leading to a computer)

Using … gate logic

Gate logic

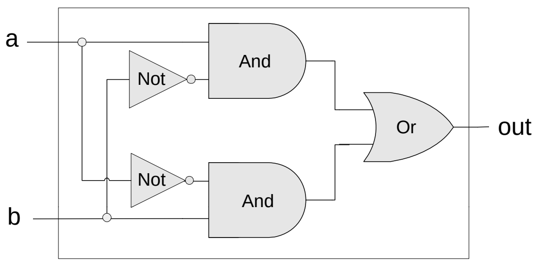
- Hardware platform = inter-connected set of chips
- Chips are made of simpler chips, all the way down to elemantary logic gates
- Logic gate = hardware element that implements a certain Boolean function
- Every chip and gate has an *interface*, specifying WHAT it is doing, and an *implementation*, specifying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

Interface

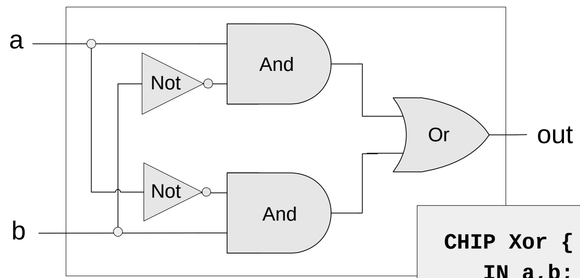


a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Implementation



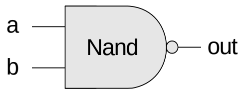
Hardware description language (HDL)




```
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);
}
```

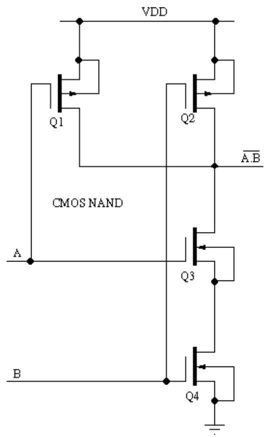
The tour ends:

Interface

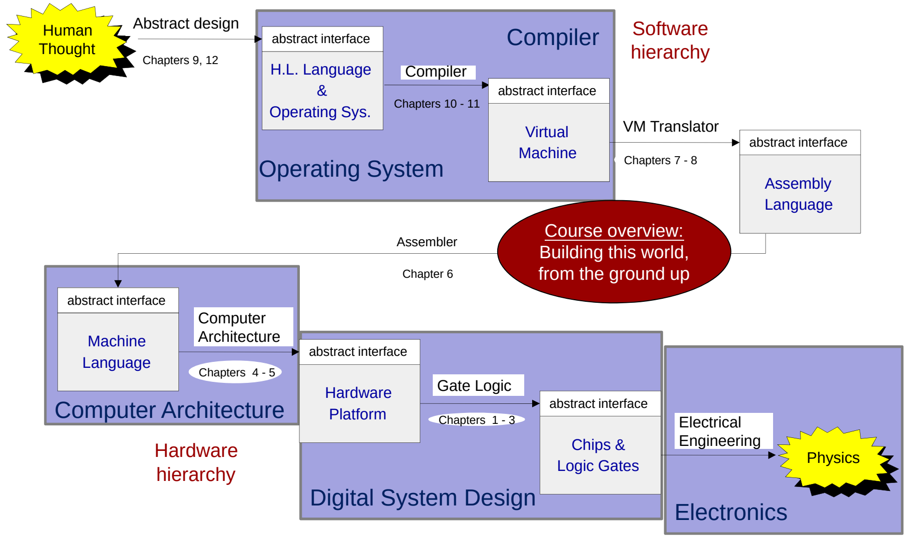


a	b	out
0	0	1
0	1	1
1	0	1
1	1	0

One implementation option (CMOS)



The tour map, revisited



What you will learn

- Number systems
- Combinational logic
- Sequential logic
- Basic principle of computer architecture
- Assembler
- Virtual machine
- High-level language
- Fundamentals of compilers
- Basic operating system
- Application programming

In short