Course overview

Introduction to Computer

Yung-Yu Chuang

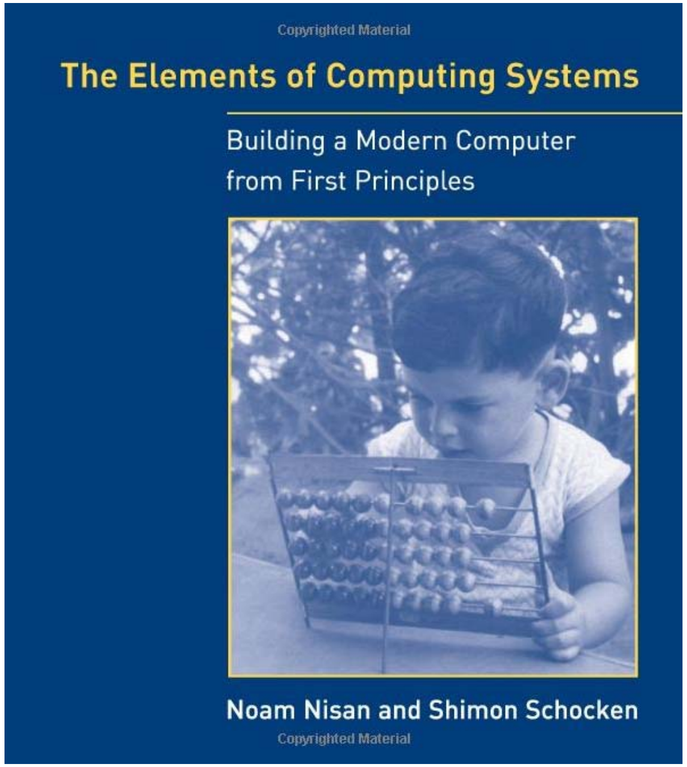
*with slides by Nisan & Schocken (*www.nand2tetris.org*)*

Logistics

- **Meeting time**: 2:20pm-5:20pm, Tuesday
- **Instructor**: 莊永裕 Yung-Yu Chuang
- **Webpage**:

  http://www.csie.ntu.edu.tw/~cyy/introcs

Textbook

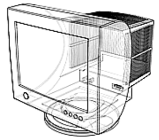


The Elements of Computing Systems, Noam Nisan, Shimon Schocken, MIT Press

Nand2Tetris on coursera

Nand2Tetris2 on coursera

References (TOY)

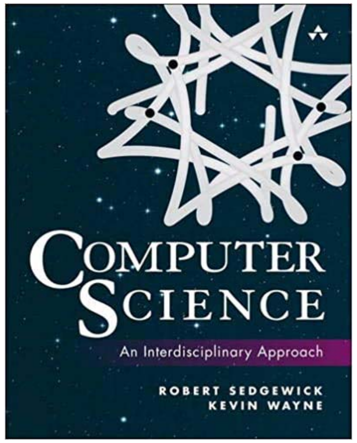


Princeton's Introduction to CS,

http://www.cs.princeton.edu/introcs/java/60machine/

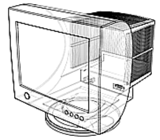
http://www.cs.princeton.edu/introcs/java/70circuits/

Coursera course



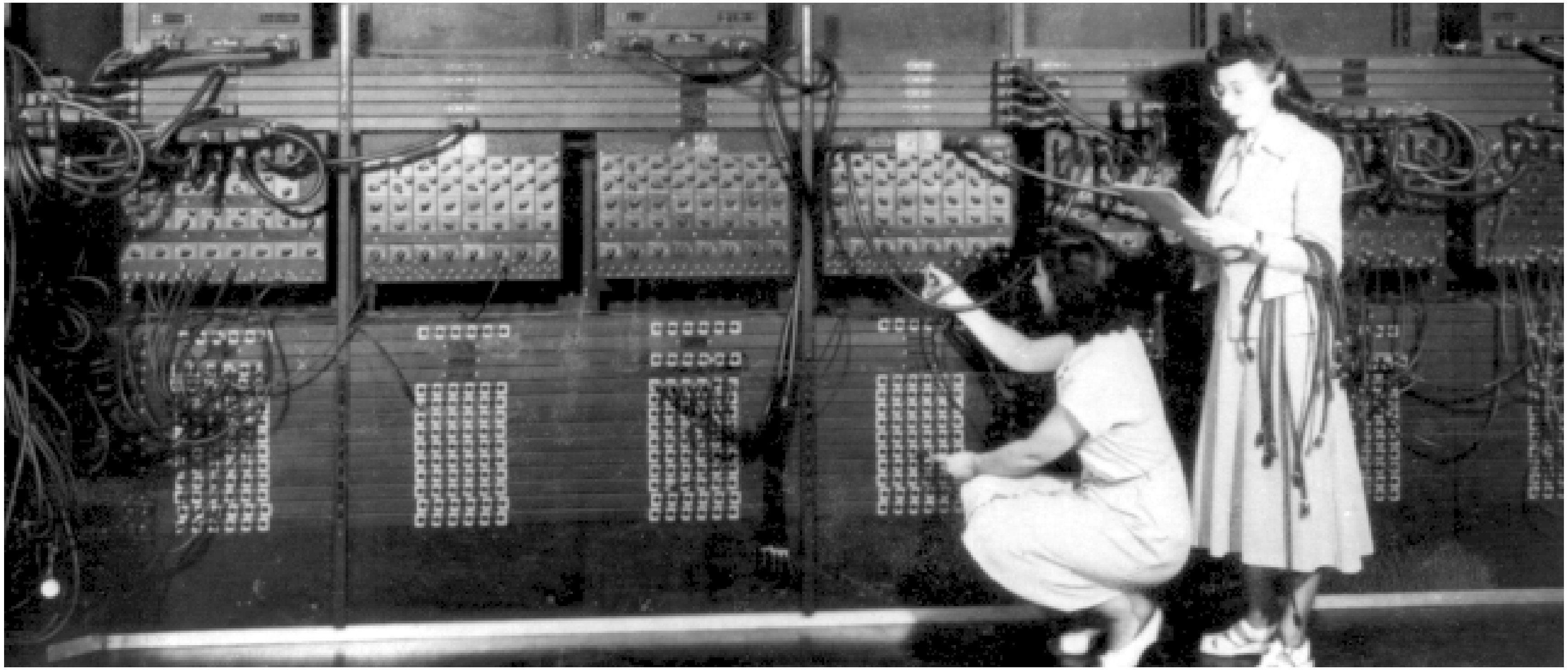
Computer Science: An Interdisciplinary Approach. Robert Sedgewick, Kevin Wayne

Grading (subject to change)

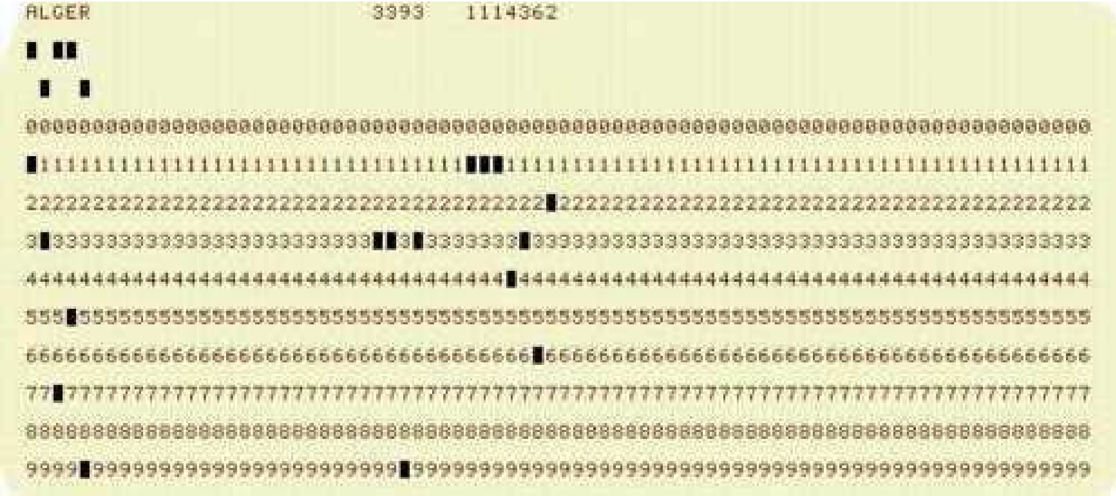


- Assignments (5 projects+1 homework, 50%) from the accompanying website
- Class participation (5%)
- Midterm quiz (20%)
- Final project (25%)

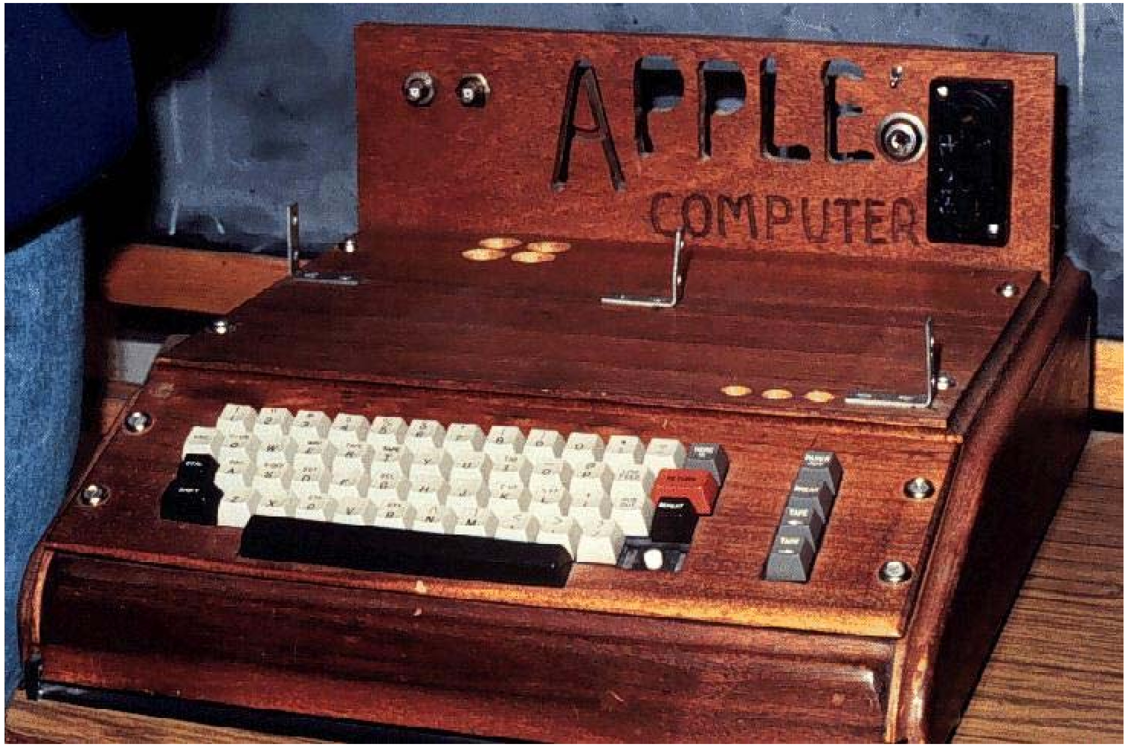
Early computers



Early programming tools



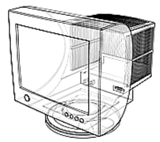
First popular PCs

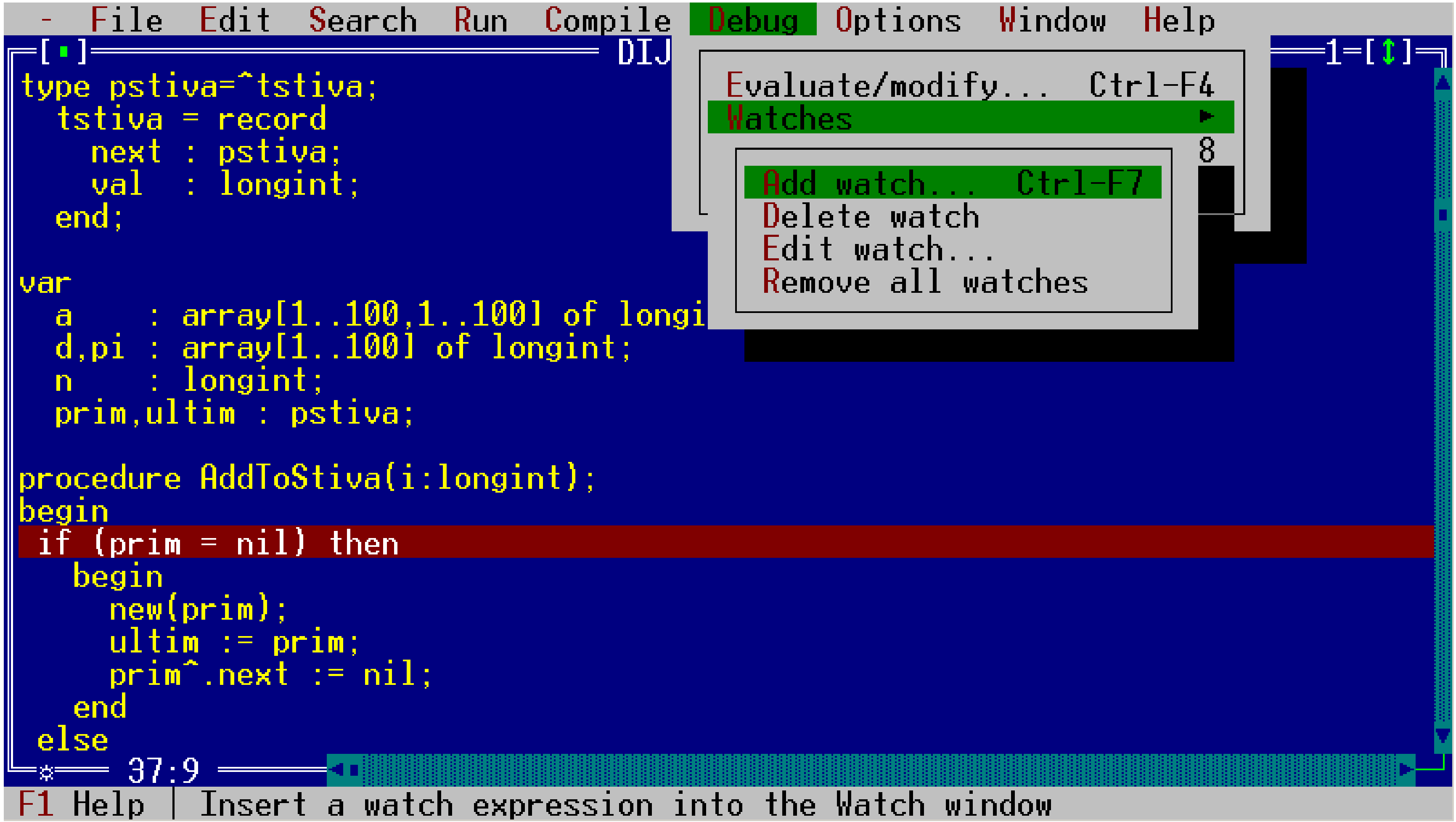


Early PCs

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

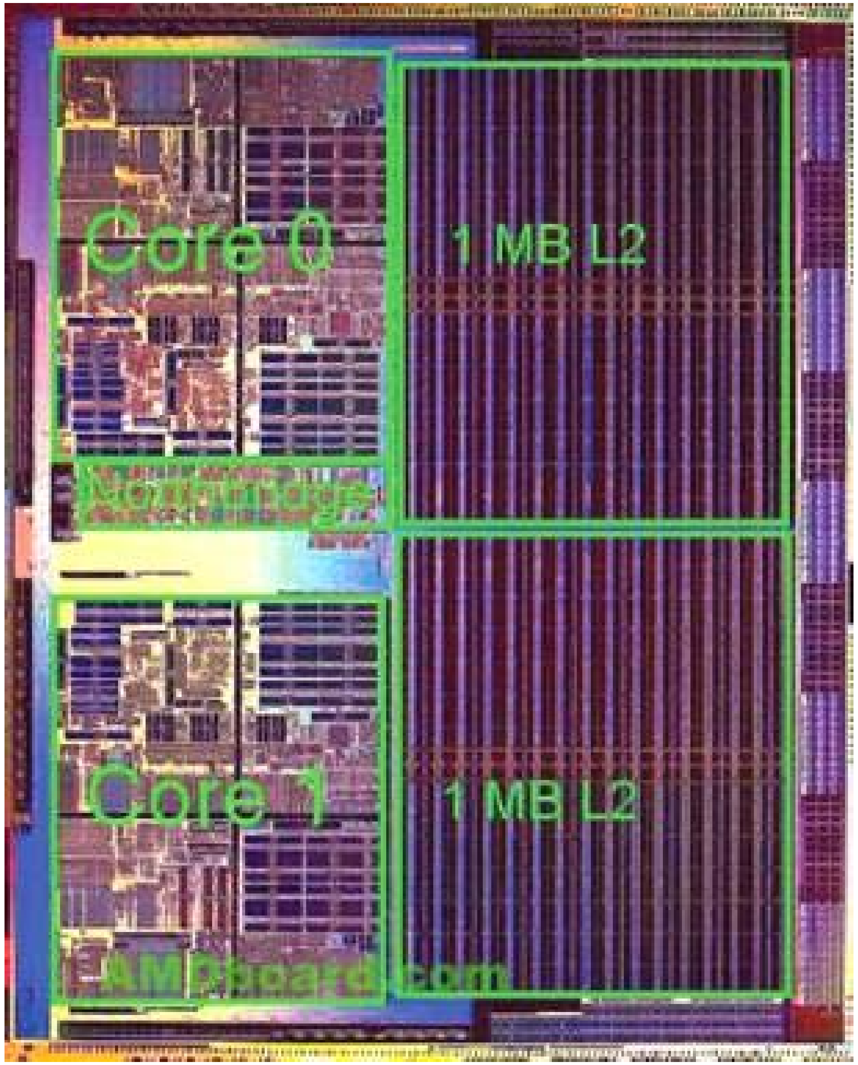
GUI/IDE





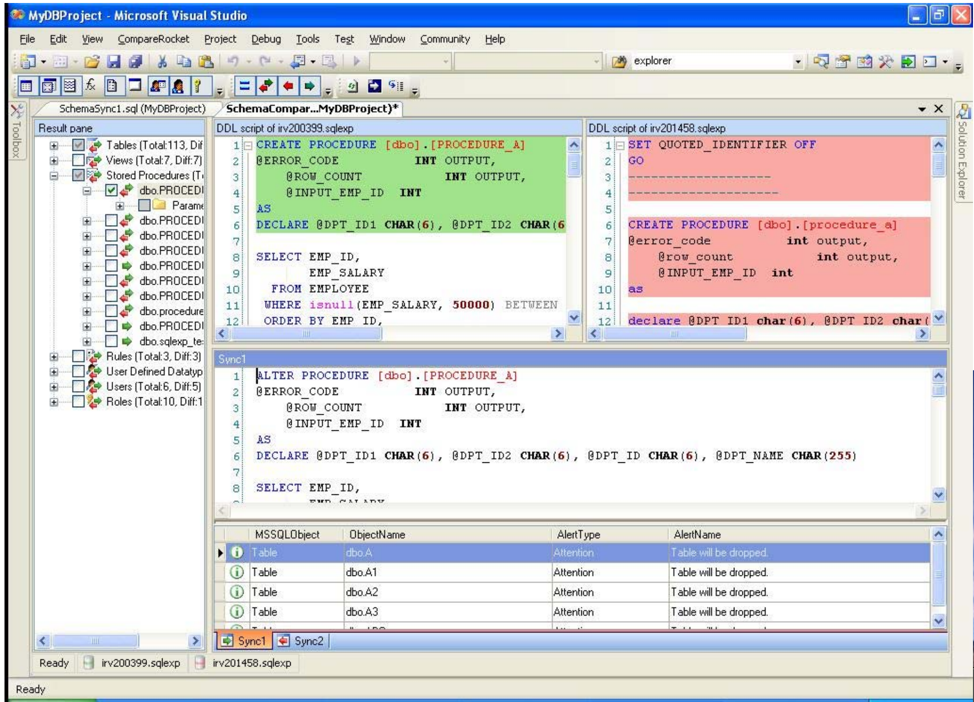
- File Edit Search Run Compile Debug Options Window Help
DIJ
1
type pstiva=^tstiva;
tstiva = record
next : pstiva;
val : longint;
end;
var
a : array[1..100,1..100] of longi
d,pi : array[1..100] of longint;
n : longint;
prim,ultim : pstiva;
procedure AddToStiva(i:longint);
begin
if (prim = nil) then
begin
new(prim);
ultim := prim;
prim^.next := nil;
end
else
37:9
Evaluate/modify... Ctrl-F4
Watches
8
Add watch... Ctrl-F7
Delete watch
Edit watch...
Remove all watches
F1 Help | Insert a watch expression into the Watch window

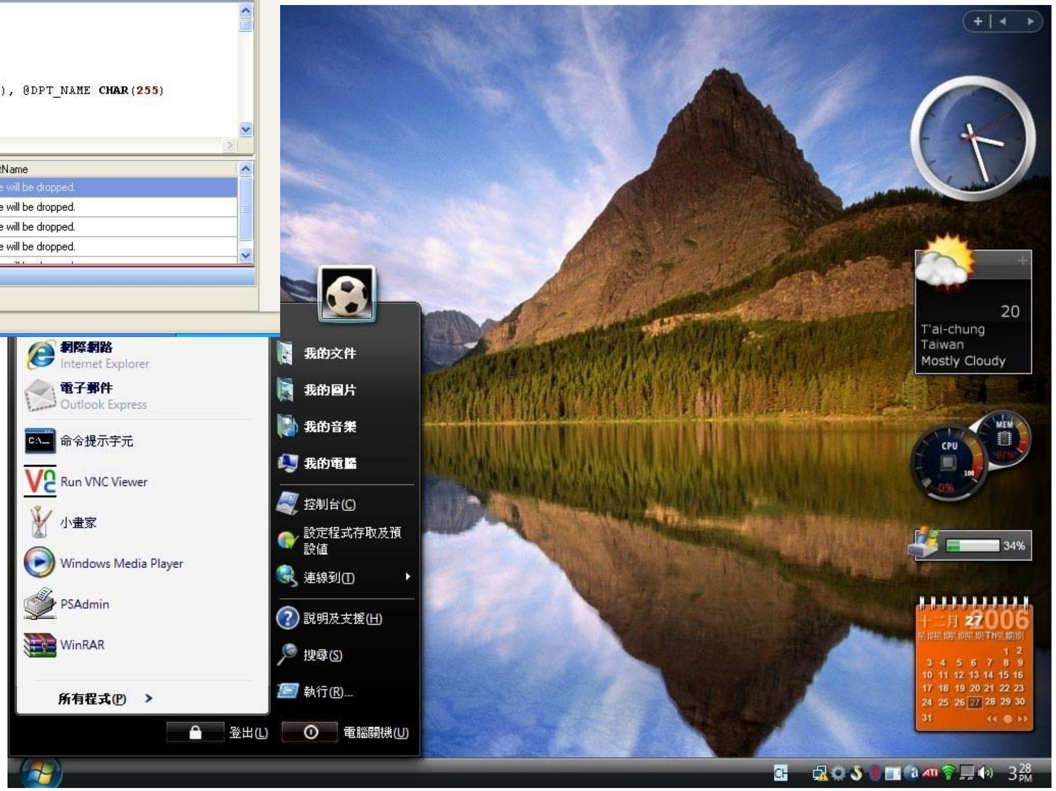
More advanced architectures



- Pipeline
- SIMD
- Multi-core
- Cache

More advanced software





More “computers” around us

My computers

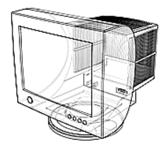
Desktop
(Intel Core i7-6700
3.4GHz, GTX960)

MacBook Pro
(Intel Core i5, 2.3GHz)

Surface Pro 4
(Intel i5-6300 2.4GHz)

iPhone 11
Pro (A13,
ARMv8.3-A)

The downside



- *"Once upon a time, every computer specialist had a gestalt understanding of how computers worked*. ... As modern computer technologies have become increasingly more complex, this clarity is all but lost." Quoted from the textbook

How is it done?

```
// First Example in Programming 101
class Main {
  function void main () {
    do Output.printString("Hello World");
    do Output.println(); // New line
    return;
  }
}
```

Main secret of computer science

implementation

Don't worry about the "how"

Only about the "what"

abstraction

what our programming
language promises to do

- Extremely complicated system
- Information hiding

Main secret of computer science

Don't worry about the "how"

But, someone has to, for example, you.

Goal of the course

"The best way to understand how computers work is to build one from scratch." Quoted from the textbook

The course at a glance

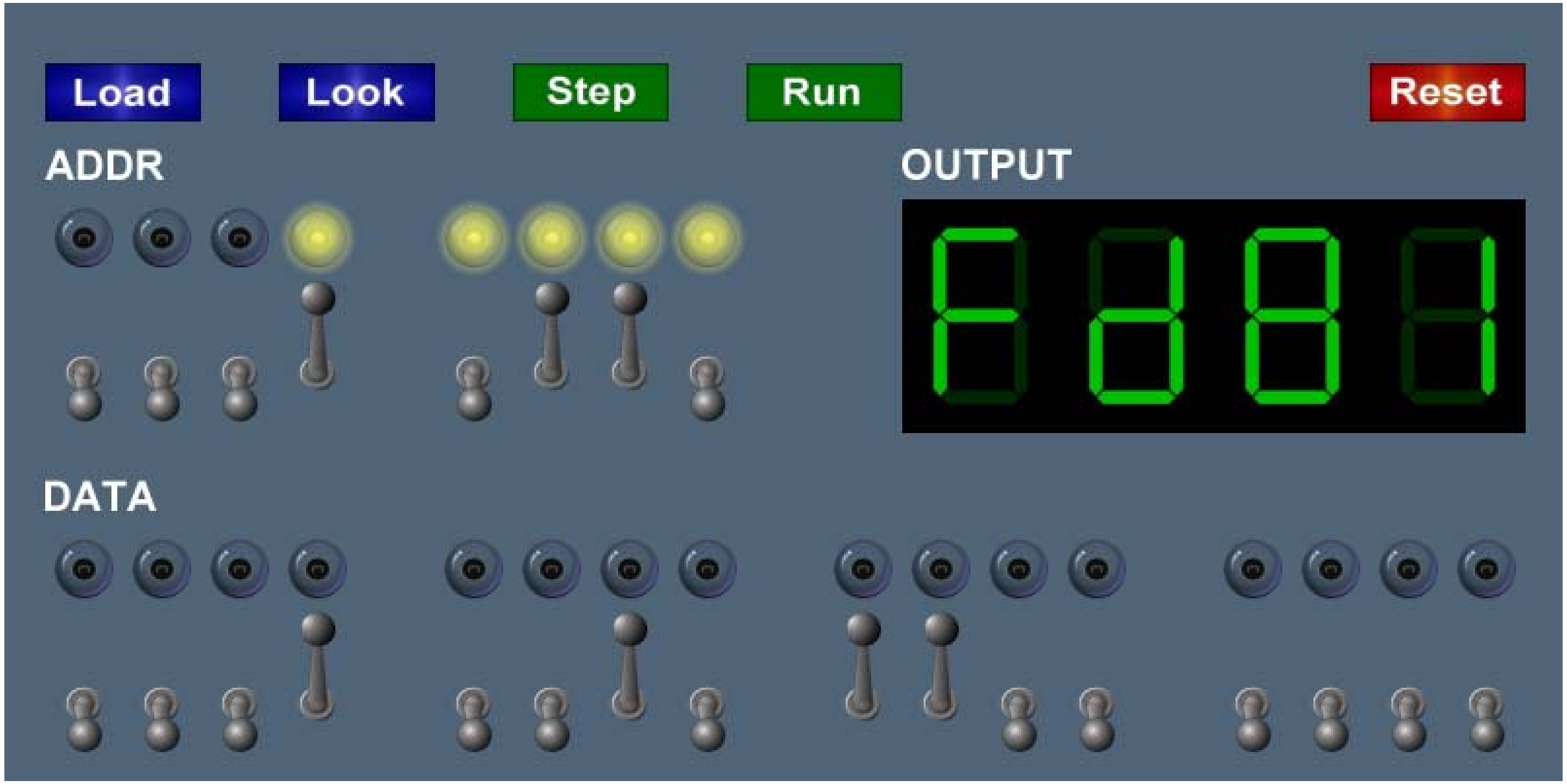
Objectives:

- 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:

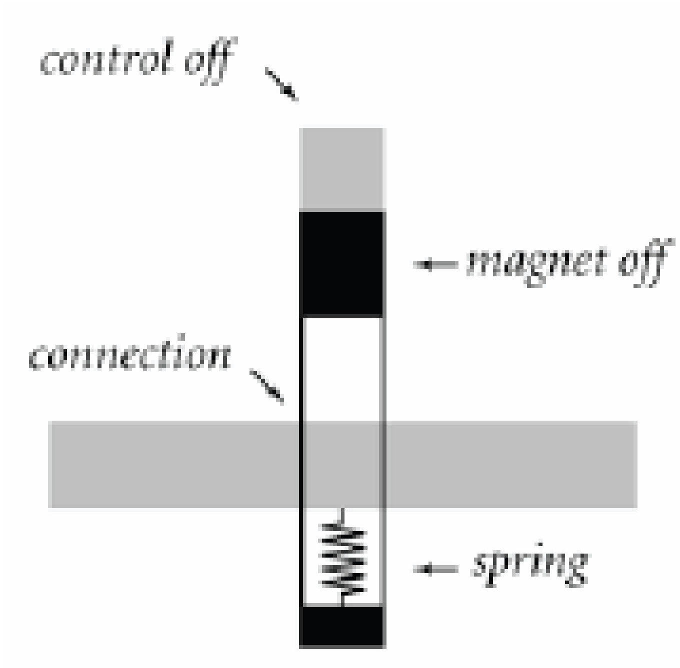
- Build a complete, general-purpose and working computer system
- Play and experiment with this computer, at any level of interest

TOY machine

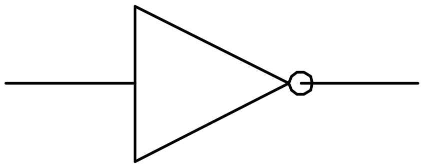


TOY machine

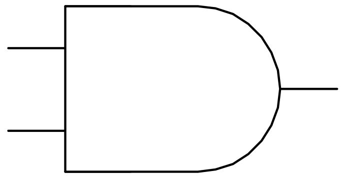
- Starting from a simple construct



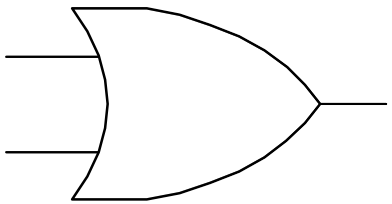
Logic gates



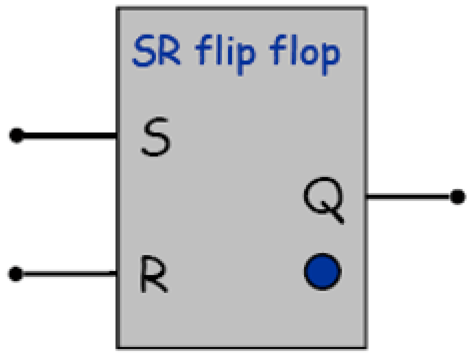
NOT



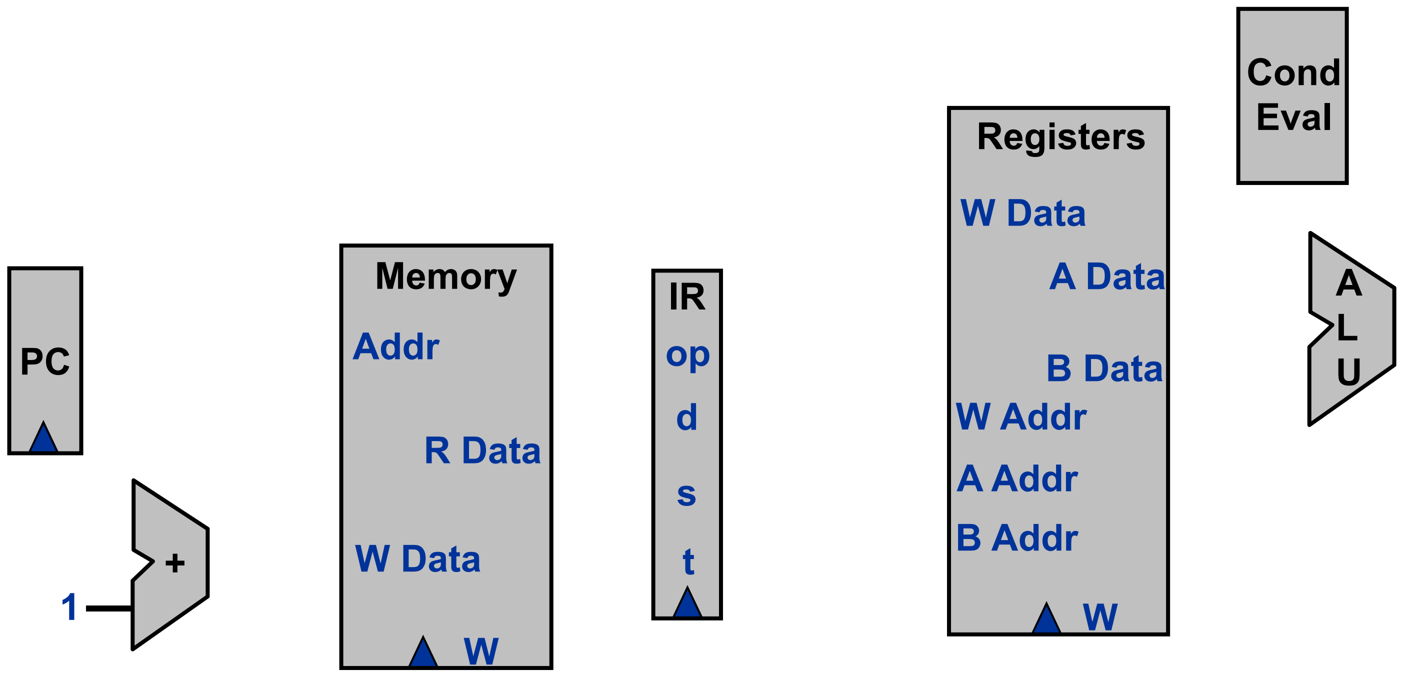
AND



OR

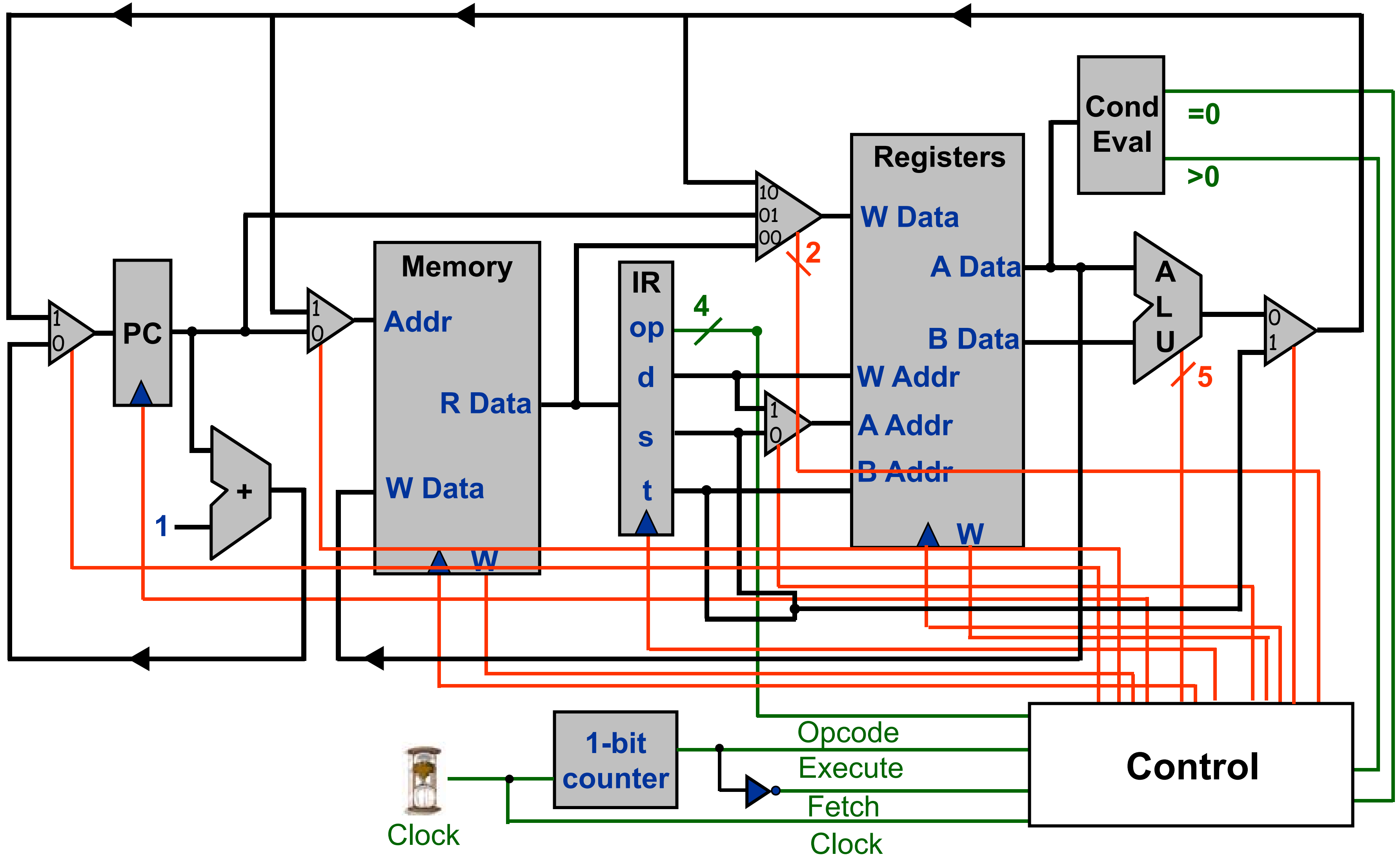


Components



PC
1
+
Memory
Addr
R Data
W Data
W
IR
op
d
s
t
Registers
W Data
A Data
B Data
W Addr
A Addr
B Addr
W
Cond
Eval
A
L
U

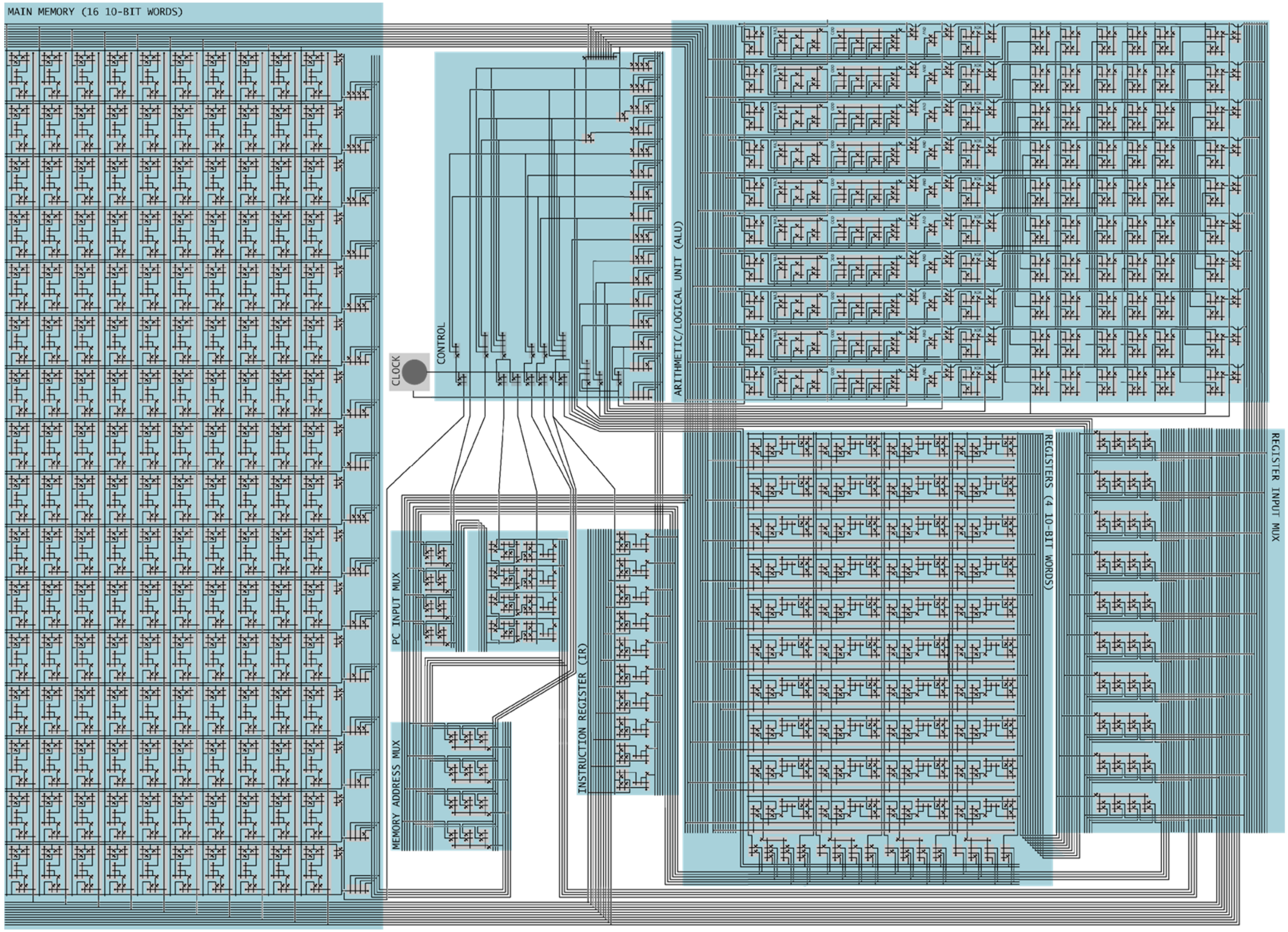
Toy machine



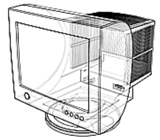
PC
1
0
1
+
Memory
Addr
R Data
W Data
W
1
0
IR
op
d
s
t
4
10
01
00
2
Registers
W Data
A Data
B Data
W Addr
A Addr
B Addr
W
1
0
Cond
Eval
=0
>0
A
L
U
5
0
1
1-bit
counter
Clock
Opcode
Execute
Fetch
Clock
Control

TOY machine

- Almost as good as any computers

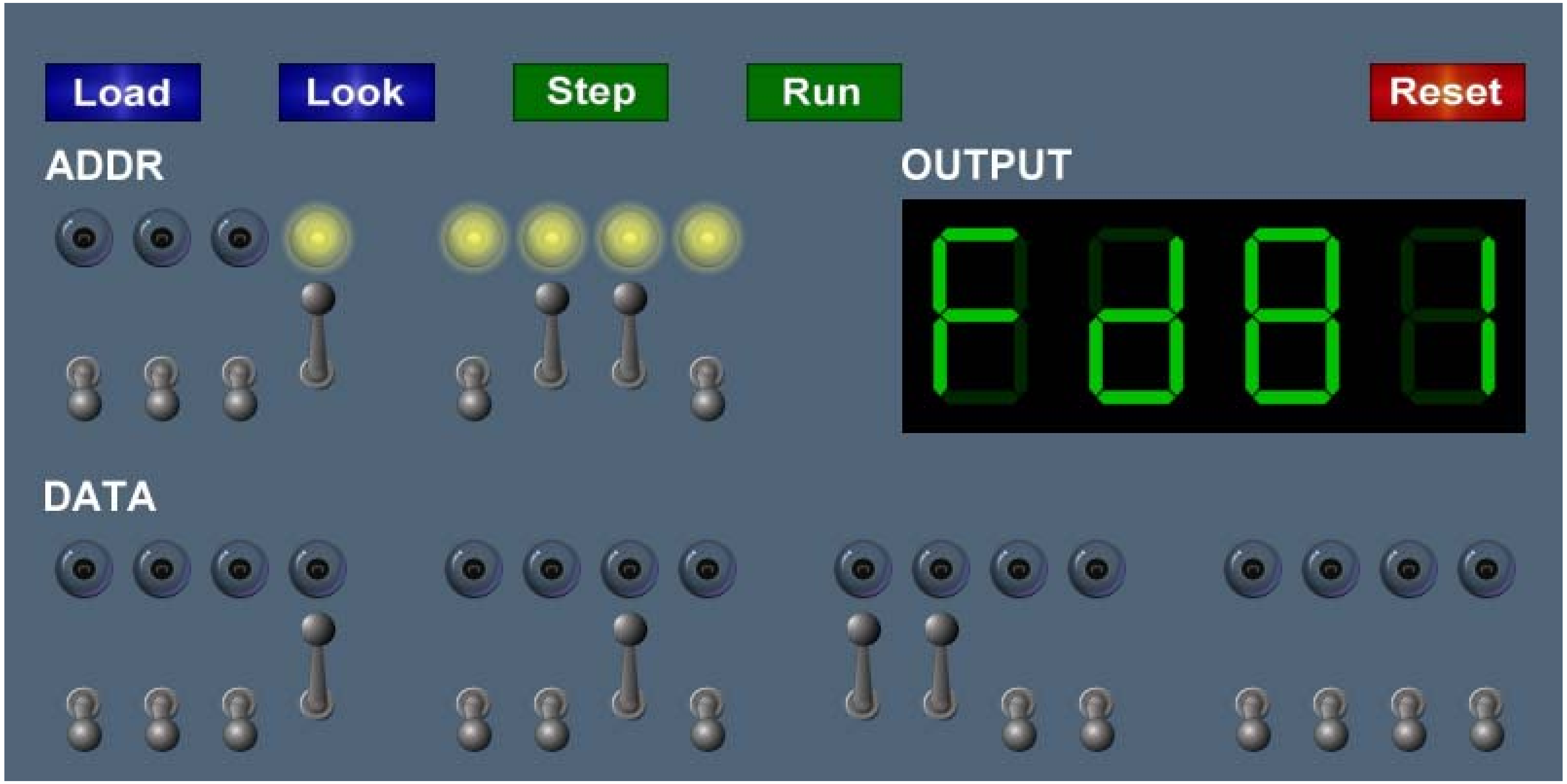


TOY machine



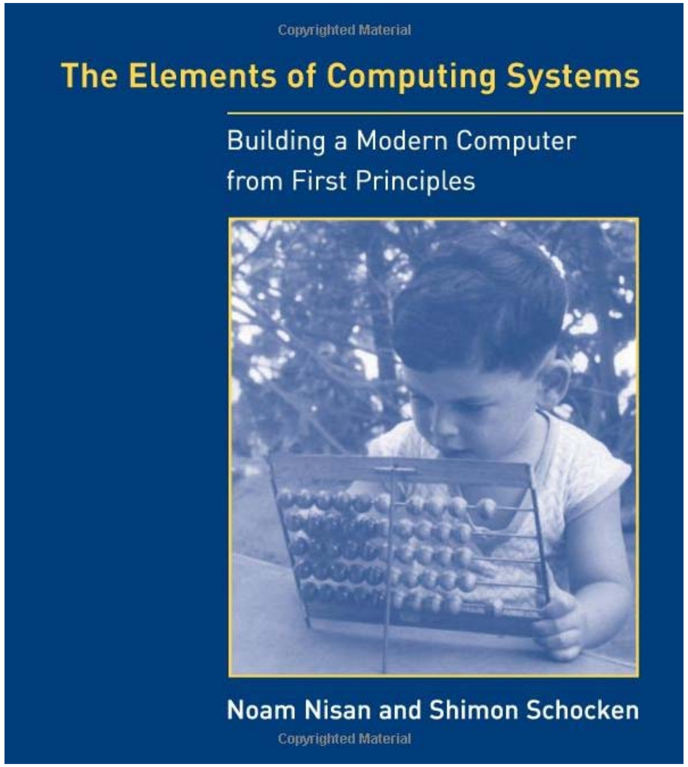
int A[32];	A	DUP	32	10: C020
		lda	R1, 1	20: 7101
		lda	RA, A	21: 7A00
i=0;		lda	RC, 0	22: 7C00
Do {				
RD=stdin;	read	ld	RD, 0xFF	23: 8DFF
if (RD==0) break;		bz	RD, exit	24: CD29
		add	R2, RA, RC	25: 12AC
A[i]=RD;		sti	RD, R2	26: BD02
i=i+1;		add	RC, RC, R1	27: 1CC1
} while (1);		bz	R0, read	28: C023
printr();	exit	jl	RF, printr	29: FF2B
		hlt		2A: 0000

TOY machine



From NAND to Tetris

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



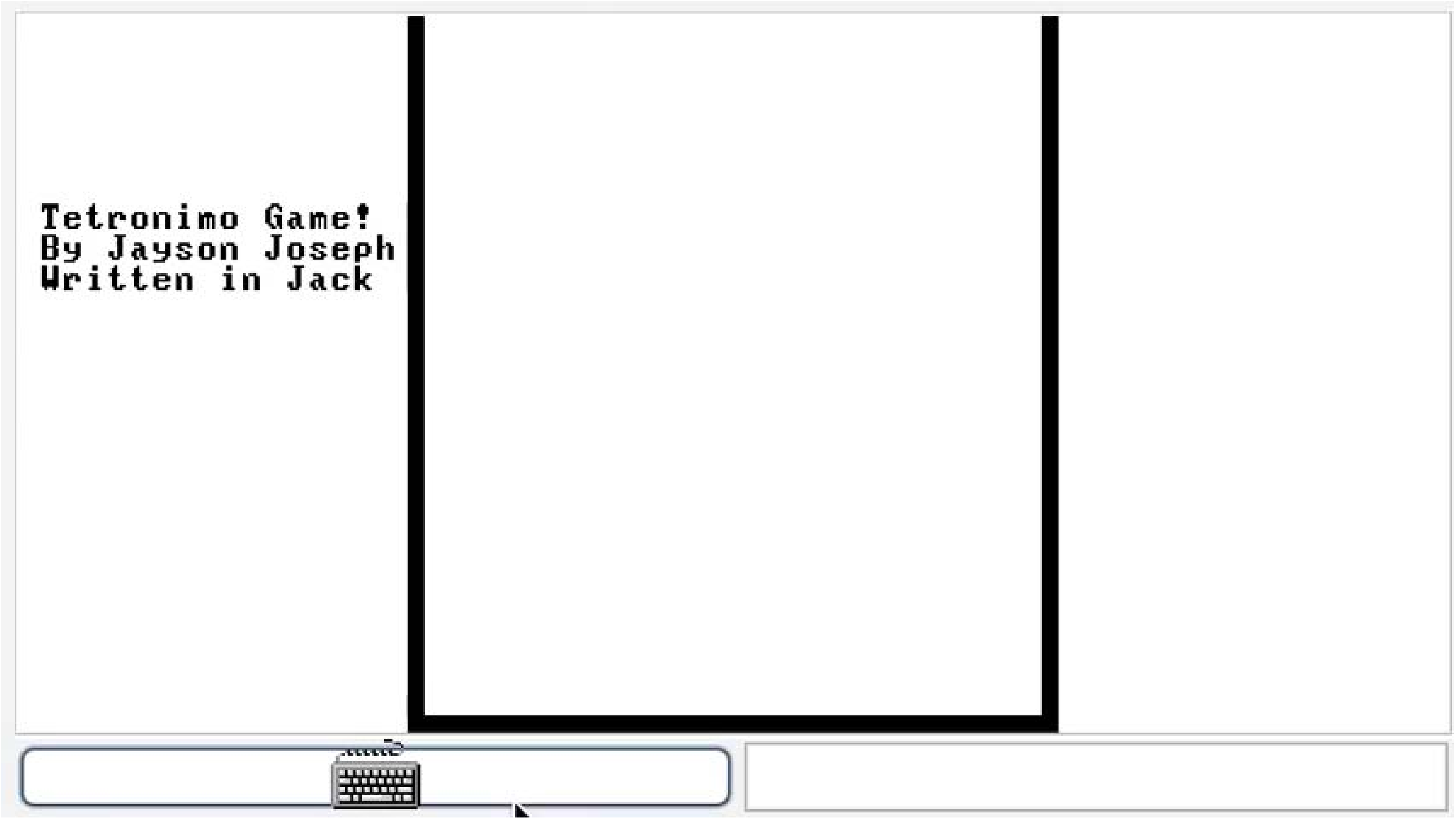
Pong on the Hack computer

Pong, 1985

Pong, 2011

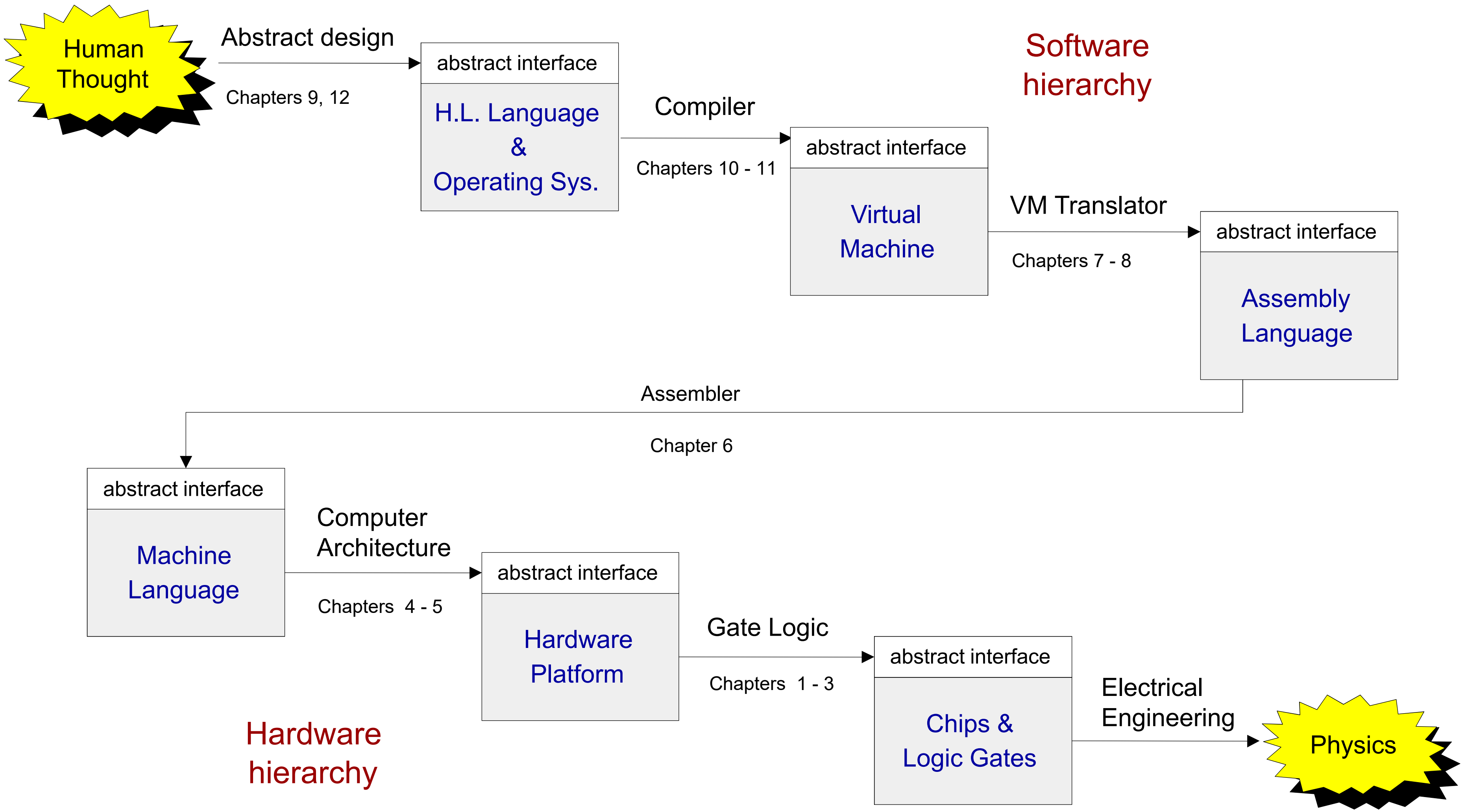
Pong, on our computer

Sample projects



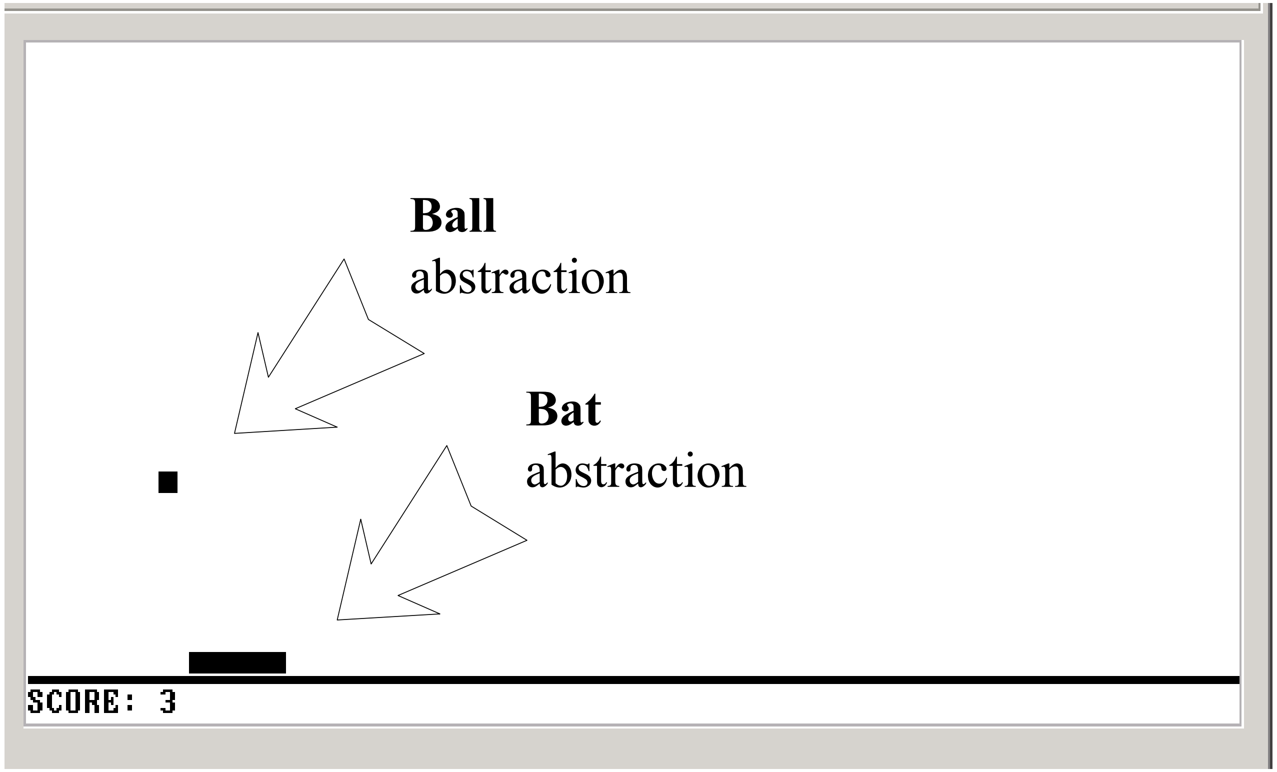
Sample projects

Theme and structure of the book

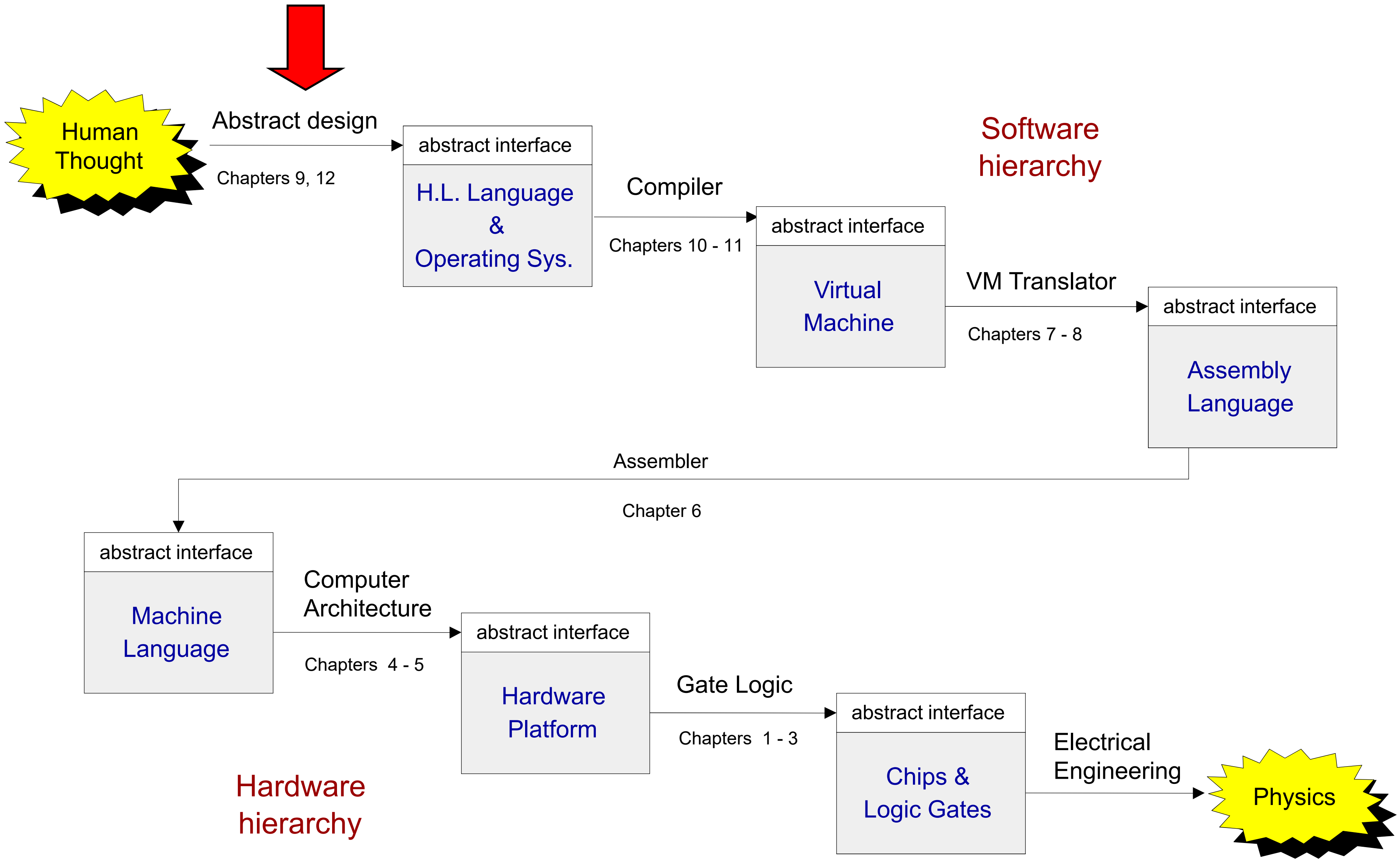


(Abstraction–implementation paradigm)

Application level: Pong (an example)



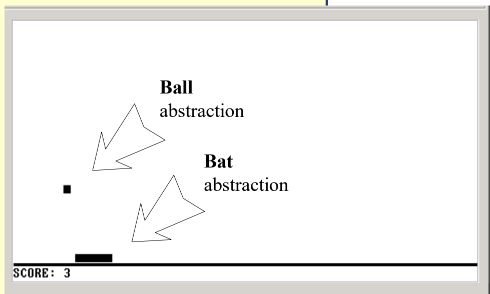
The big picture



High-level programming (Jack language)

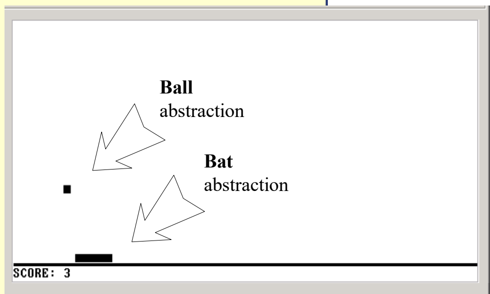
```
/** A Graphic Bat for a Pong Game */
class Bat {
    field int x, y;            // screen location of the bat's top-left corner
    field int width, height;   // bat's width & height

    // 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

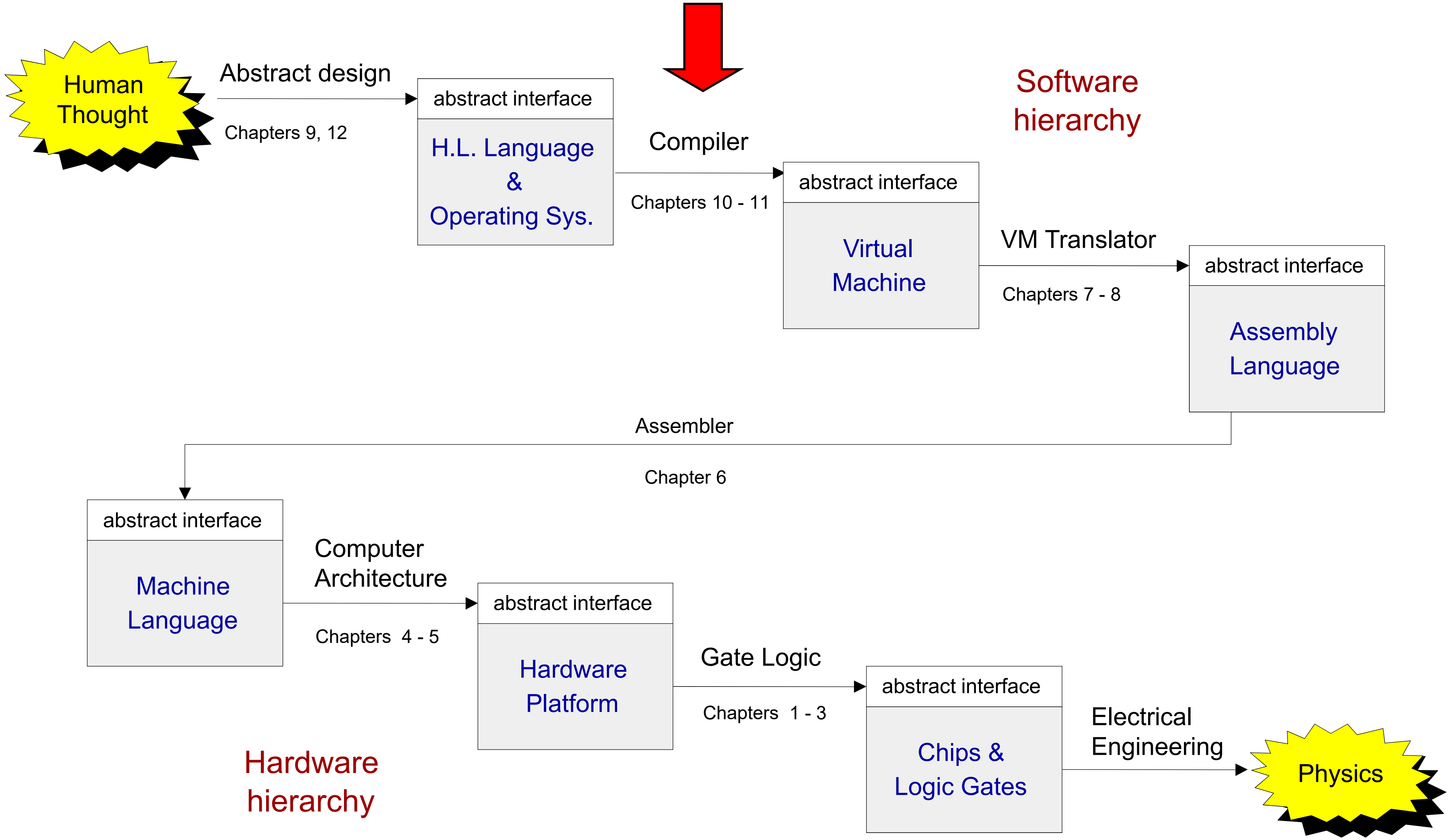


Operating system level (Jack OS)

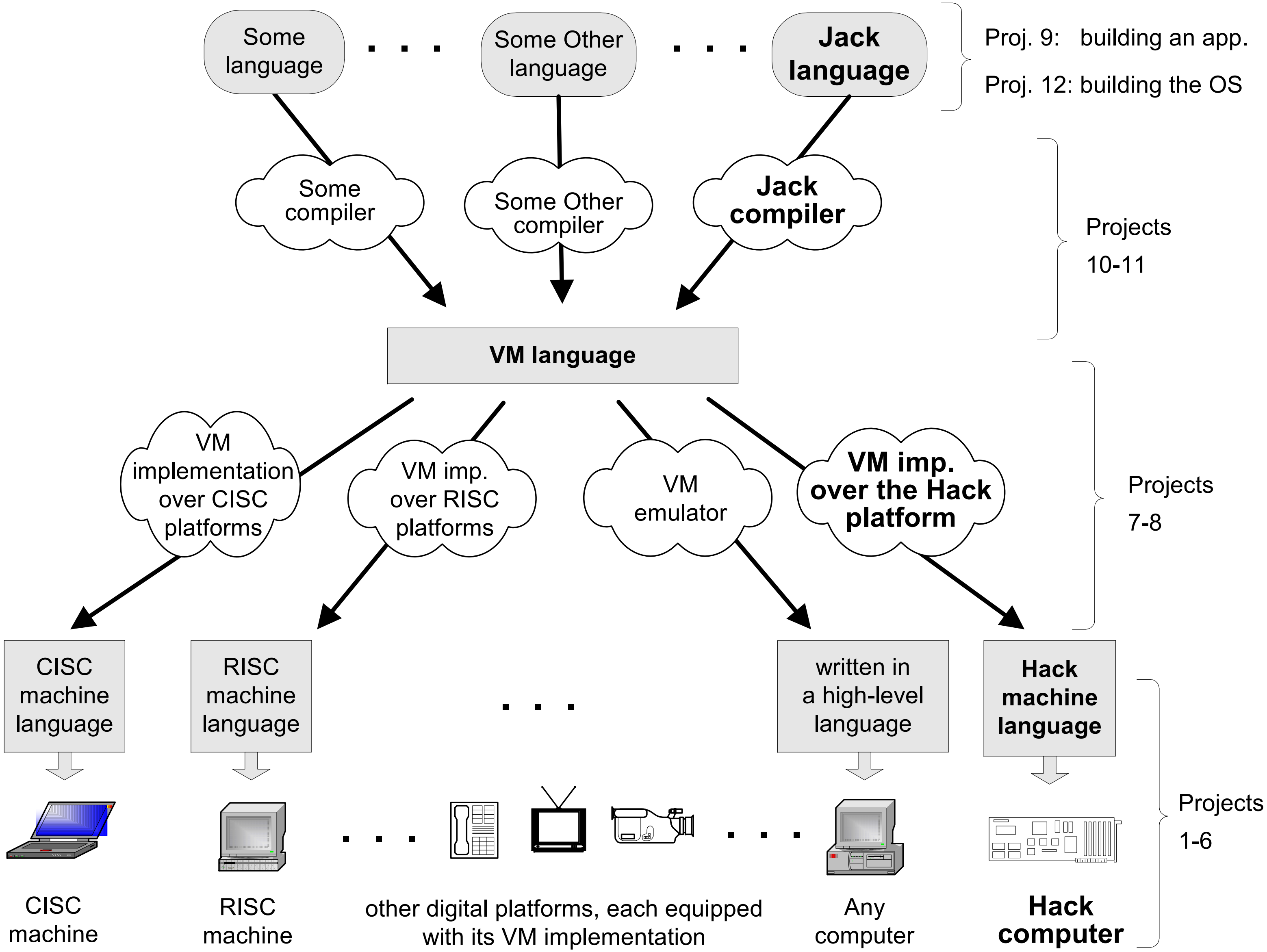
```
/** An OS-level screen driver that abstracts the computer's physical screen */
class Screen {
     static boolean currentColor;  // the current color

     // 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;
         }
     }
}
```



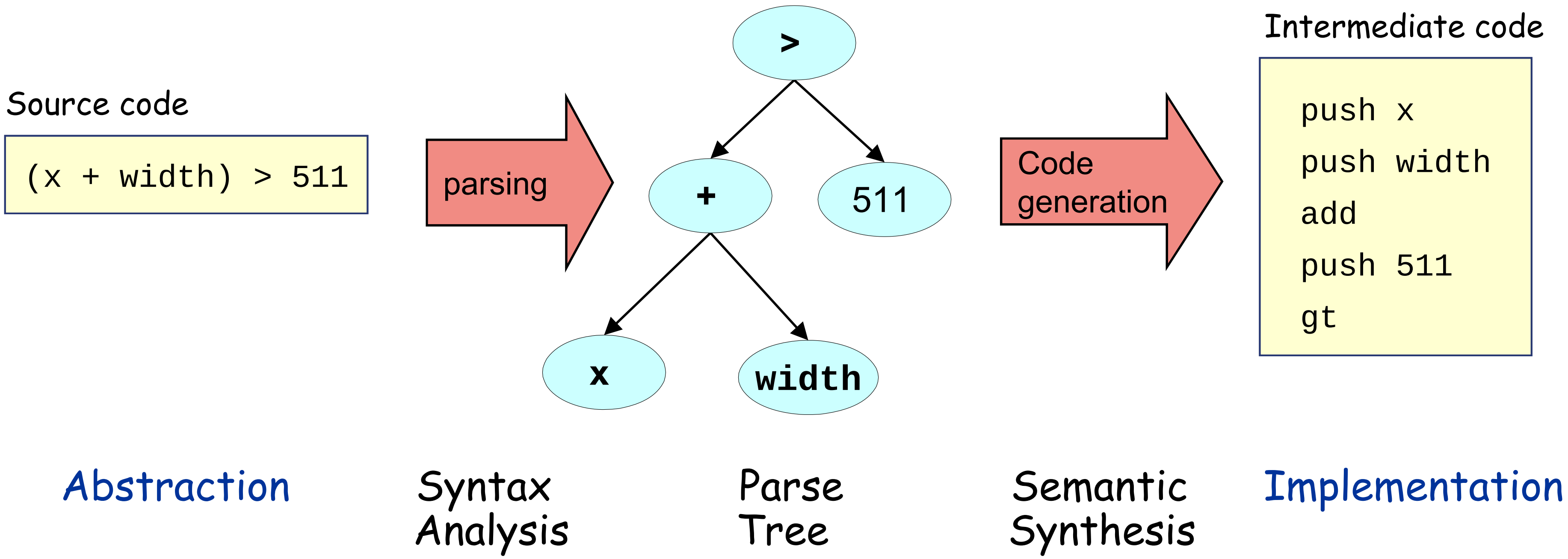
The big picture



A modern compilation model



Compilation 101



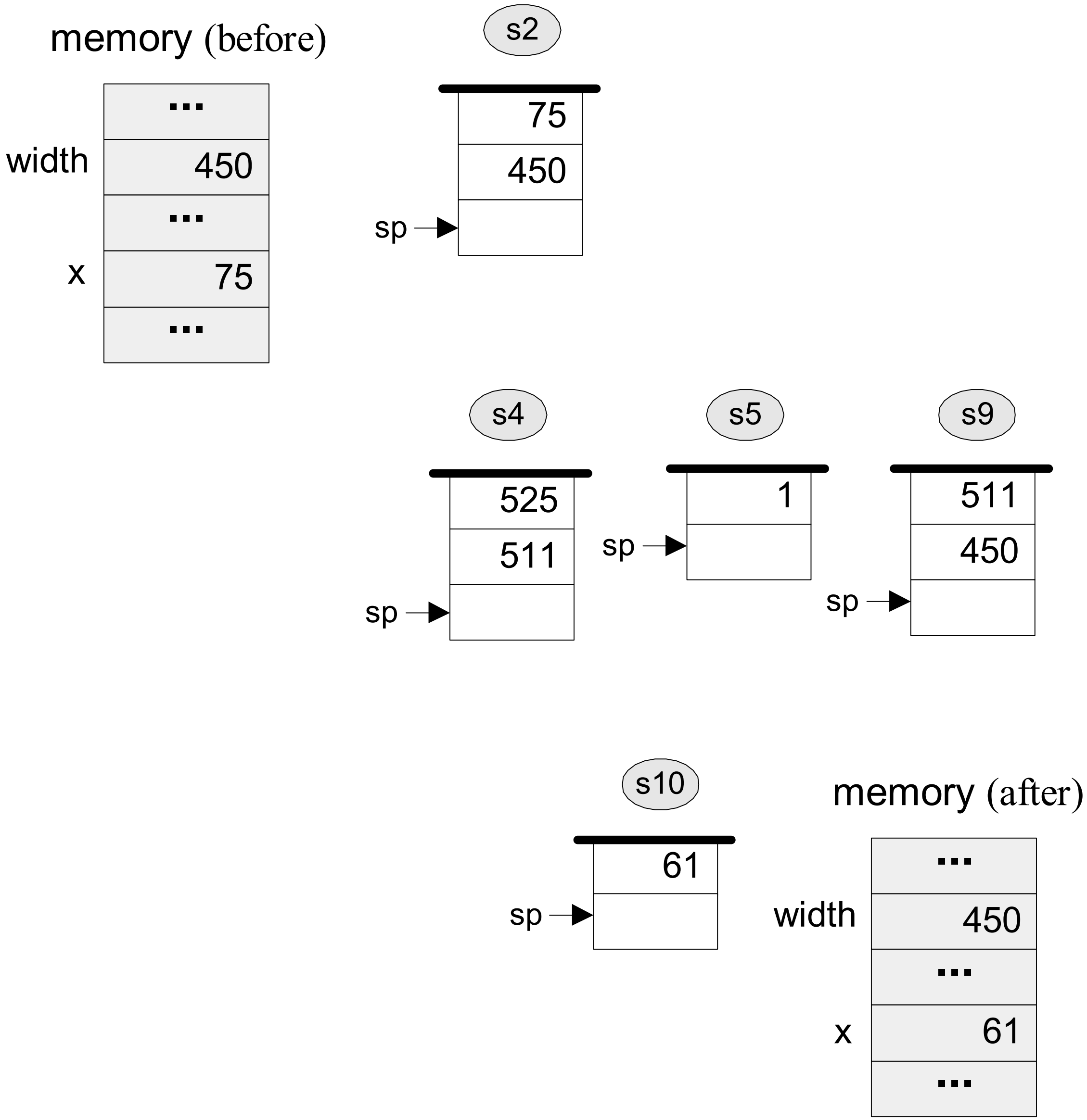
Observations:

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

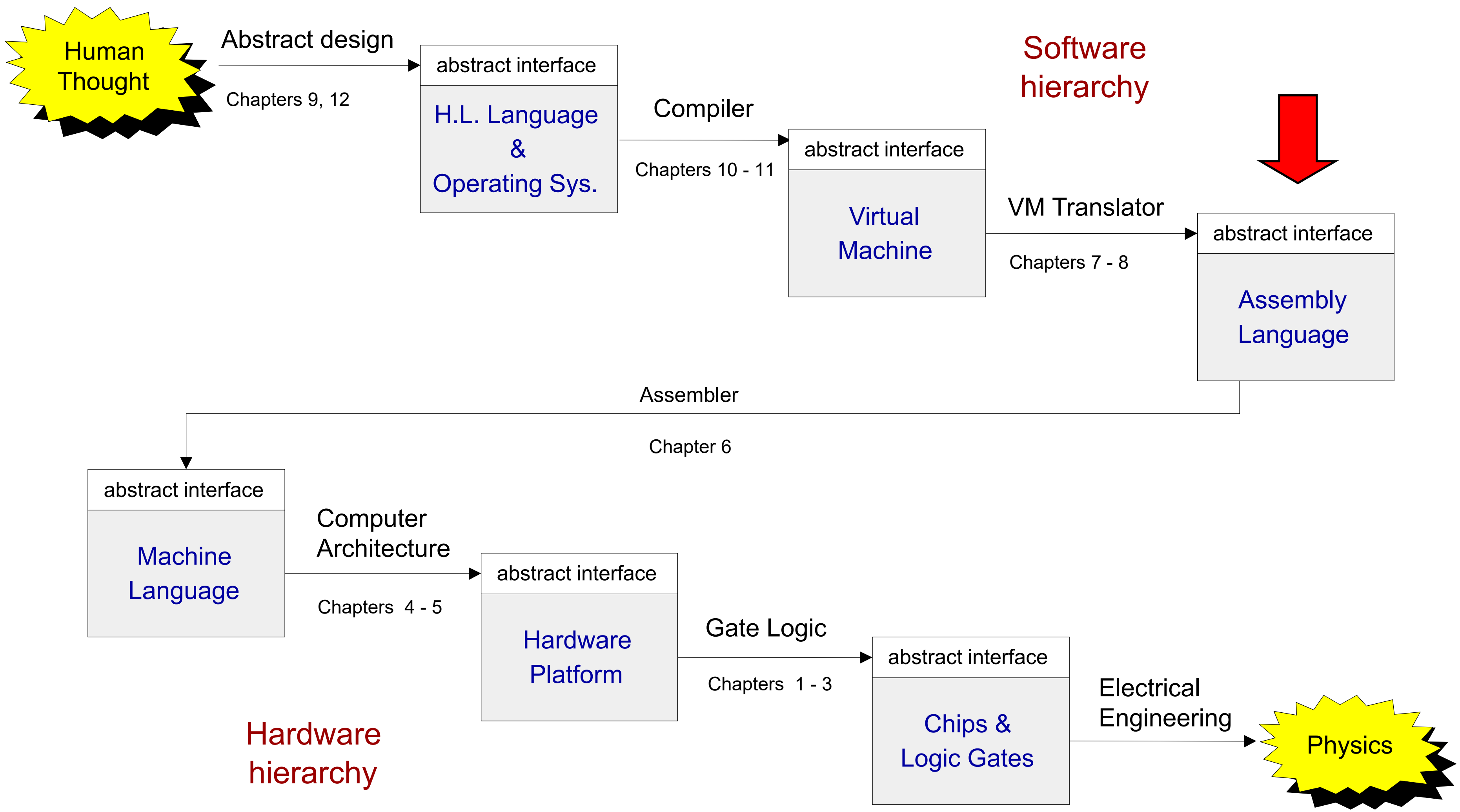
The virtual machine (VM modeled after JVM)

```
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

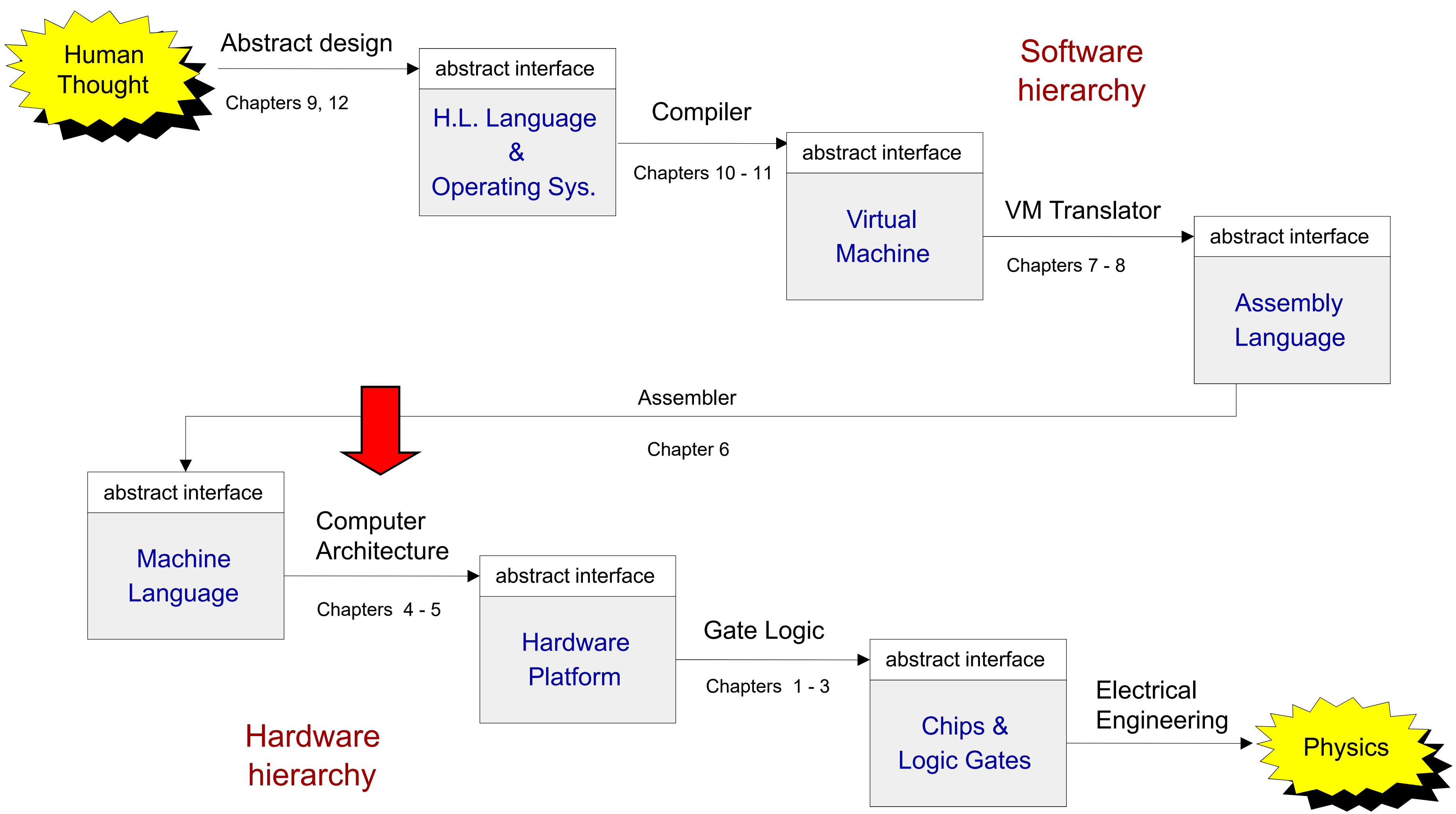
```
// push 511
@511
D=A    // D=511
@SP
A=M
M=D    // *SP=D
@SP
M=M+1  // SP++
```

Assembler

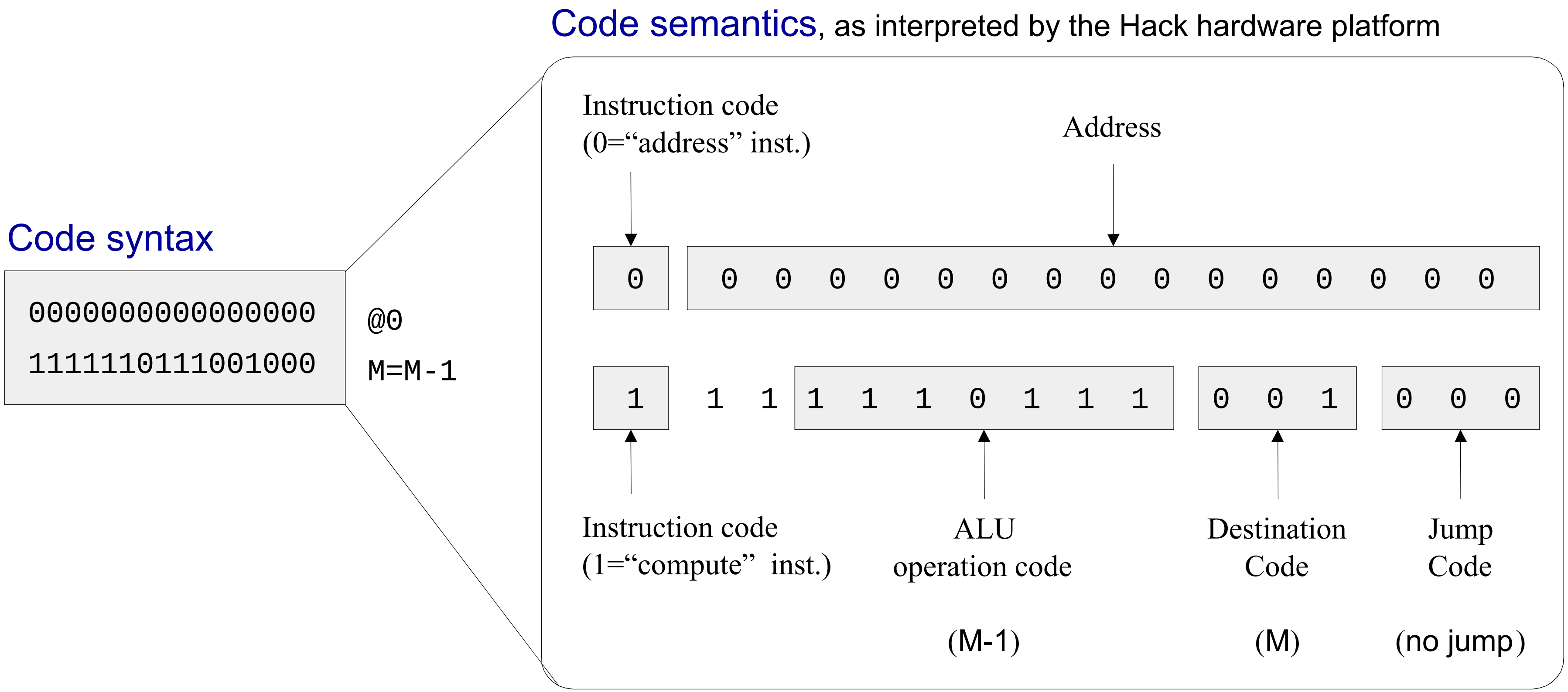
Executable

```
0000000000000000
1110110010001000
```

The big picture

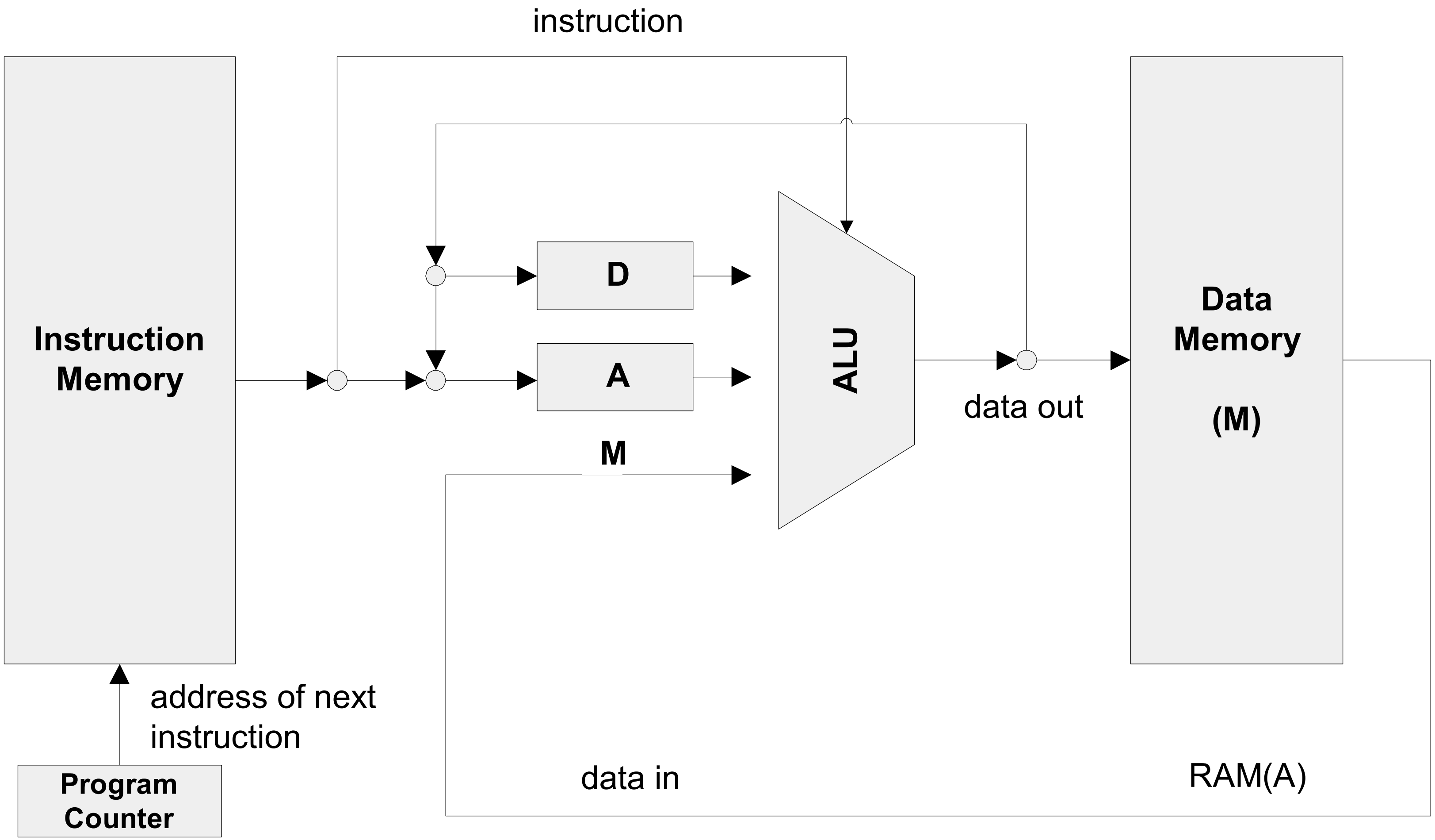


Machine language semantics (Hack)



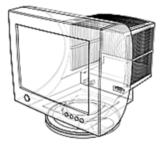
- We need a hardware architecture that realizes this semantics
- The hardware platform should be designed to:
  - o Parse instructions, and
  - o Execute them.

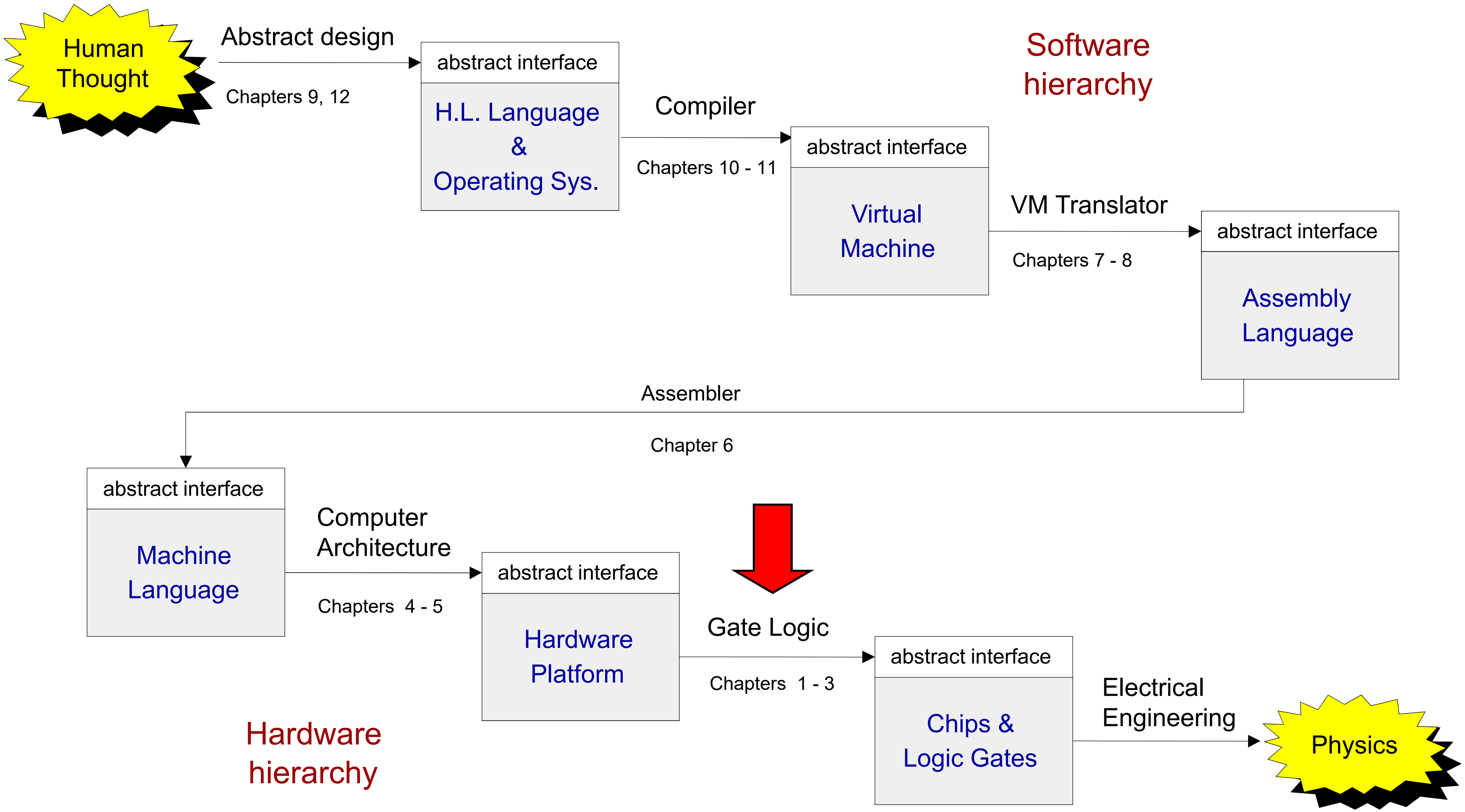
Computer architecture (Hack)



- A typical Von Neumann machine

The big picture





Logic design

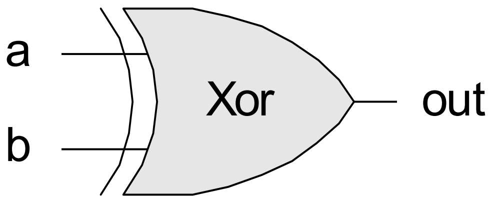
- Combinational logic (leading to an ALU)
- Sequential logic (leading to a RAM)
- Putting the whole thing together (leading to a computer)

Using ... gate logic

Gate logic

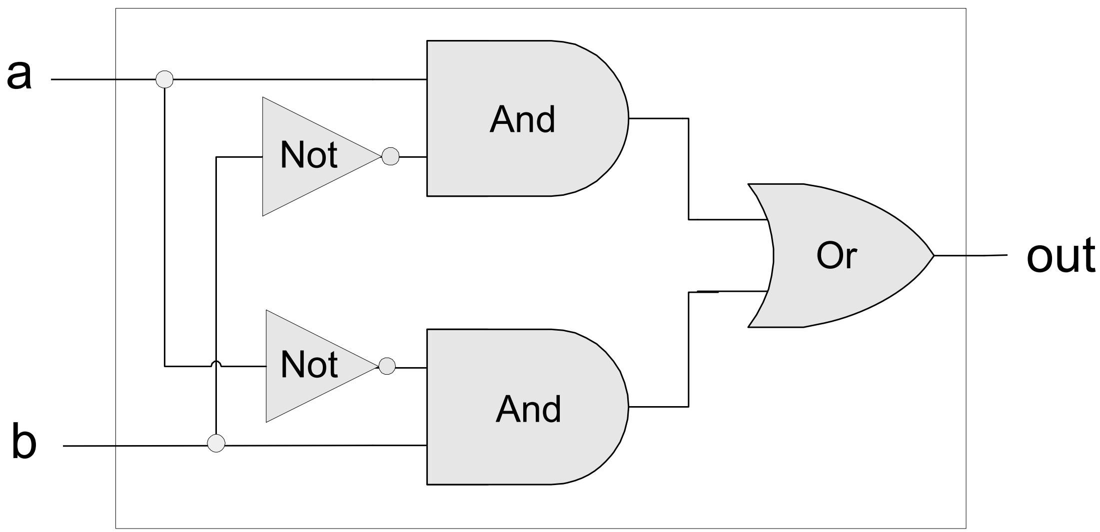
- Hardware platform = inter-connected set of chips
- Chips are made of simpler chips, all the way down to elemantary logic gates
- Logic gate = hardware element that implements a certain Boolean function
- Every chip and gate has an *interface*, specifying WHAT it is doing, and an *implementation*, specifying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

Interface

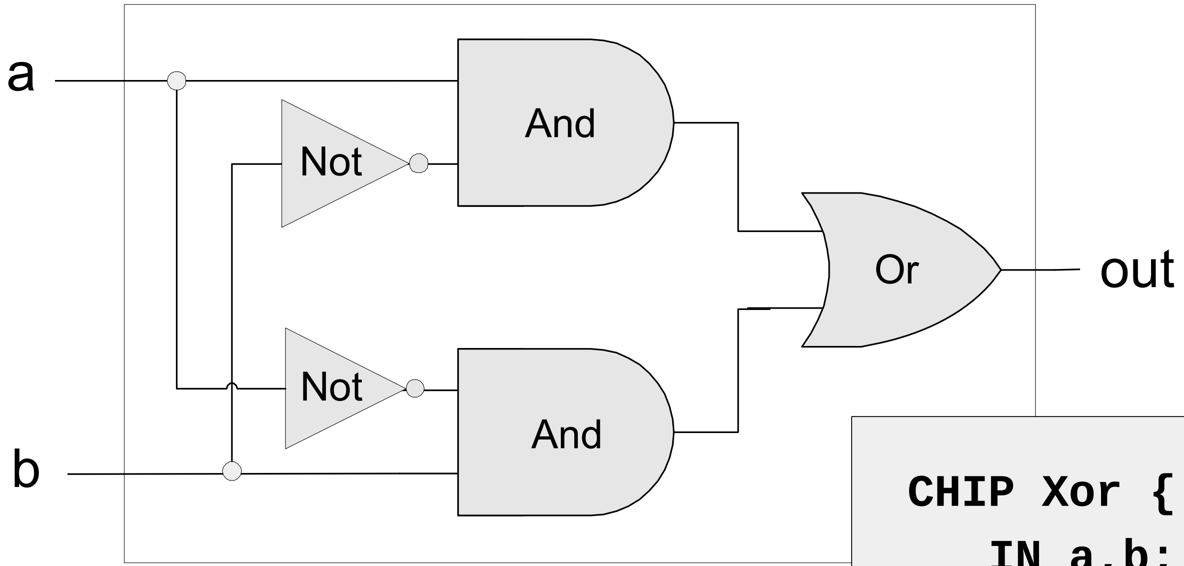


a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Implementation



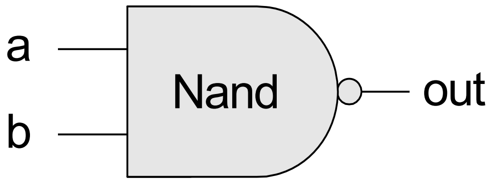
Hardware description language (HDL)




```
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);
}
```

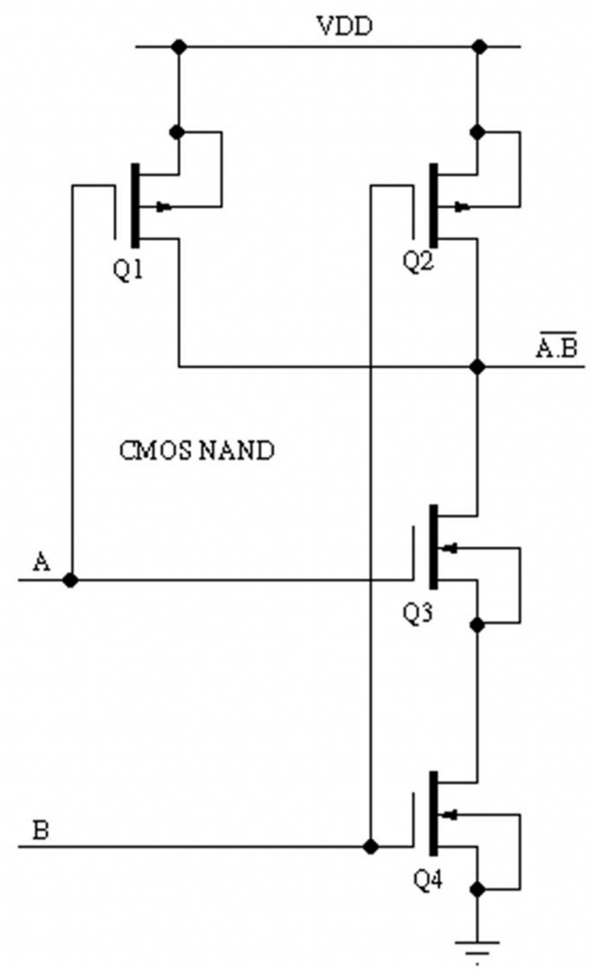
The tour ends:

Interface

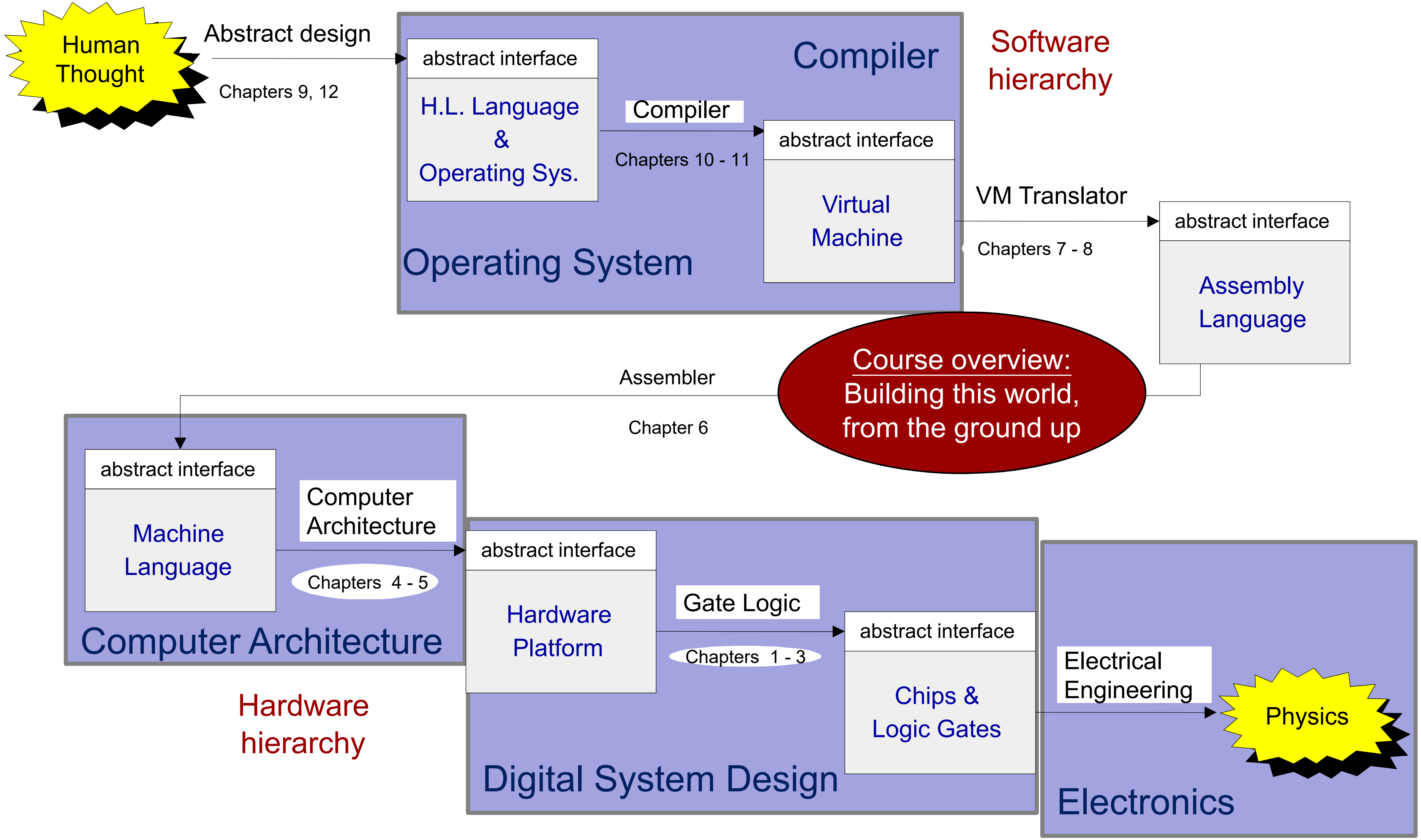


a	b	out
0	0	1
0	1	1
1	0	1
1	1	0

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