Efi Arazi School of Computer Science
Introduction to Computer Science

Lecture 13-2

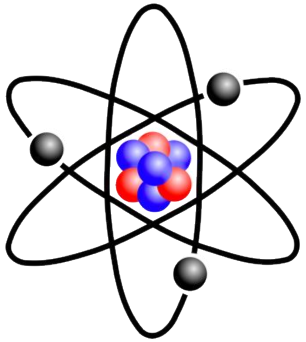
Building a Modern Computer

From First Principles

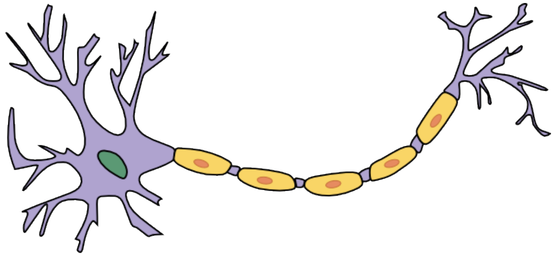
BANG

... 1001010100101101010
010010100101001010101
110011010001010010010
010011110010001000111
1111011010100101101 ...

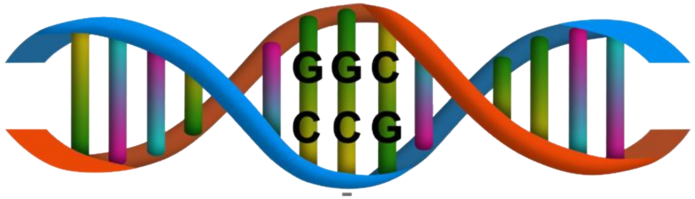
Bits ✓



Atoms



Neurons



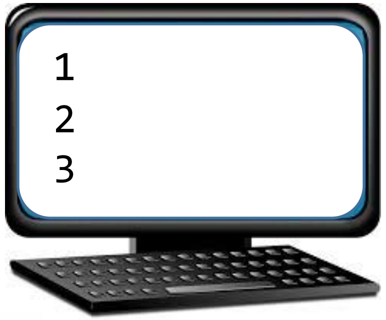
Genes

Hello World

Java / Python

```
// Prints some numbers
i = 1
while (i < 4) {
   print(i);
   i = i + 1;
}
...
```

Hello World



```
1
2
3
```

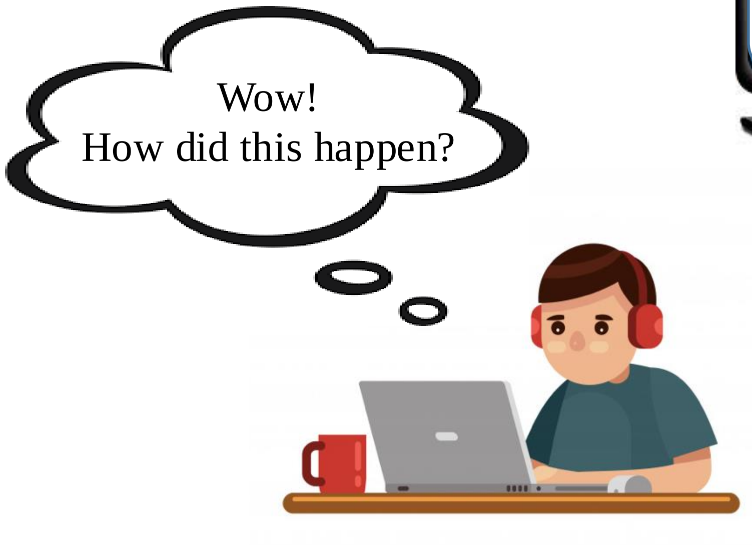
Java / Python

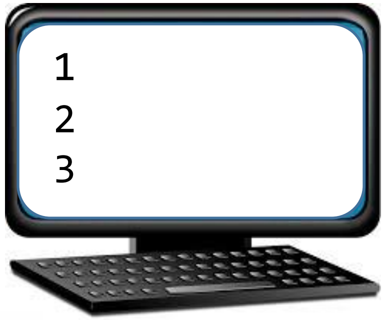
```
// Prints some numbers
i = 1
while (i < 4) {
   print(i);
   i = i + 1;
}
...
```

Hello World

Java / Python

```
// Prints some numbers
i = 1
while (i < 4) {
   print(i);
   i = i + 1;
}
...
```



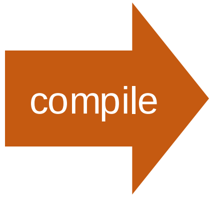


```
1
2
3
```

Hello, World Below

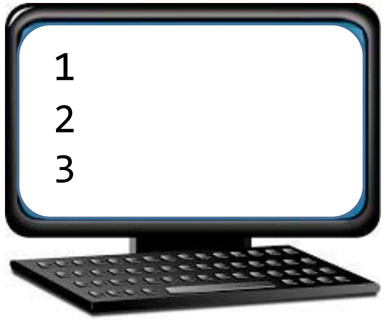
Java / Python

```
// Prints some numbers
i = 1
while (i < 4) {
   print(i);
   i = i + 1;
}
...
```


compile

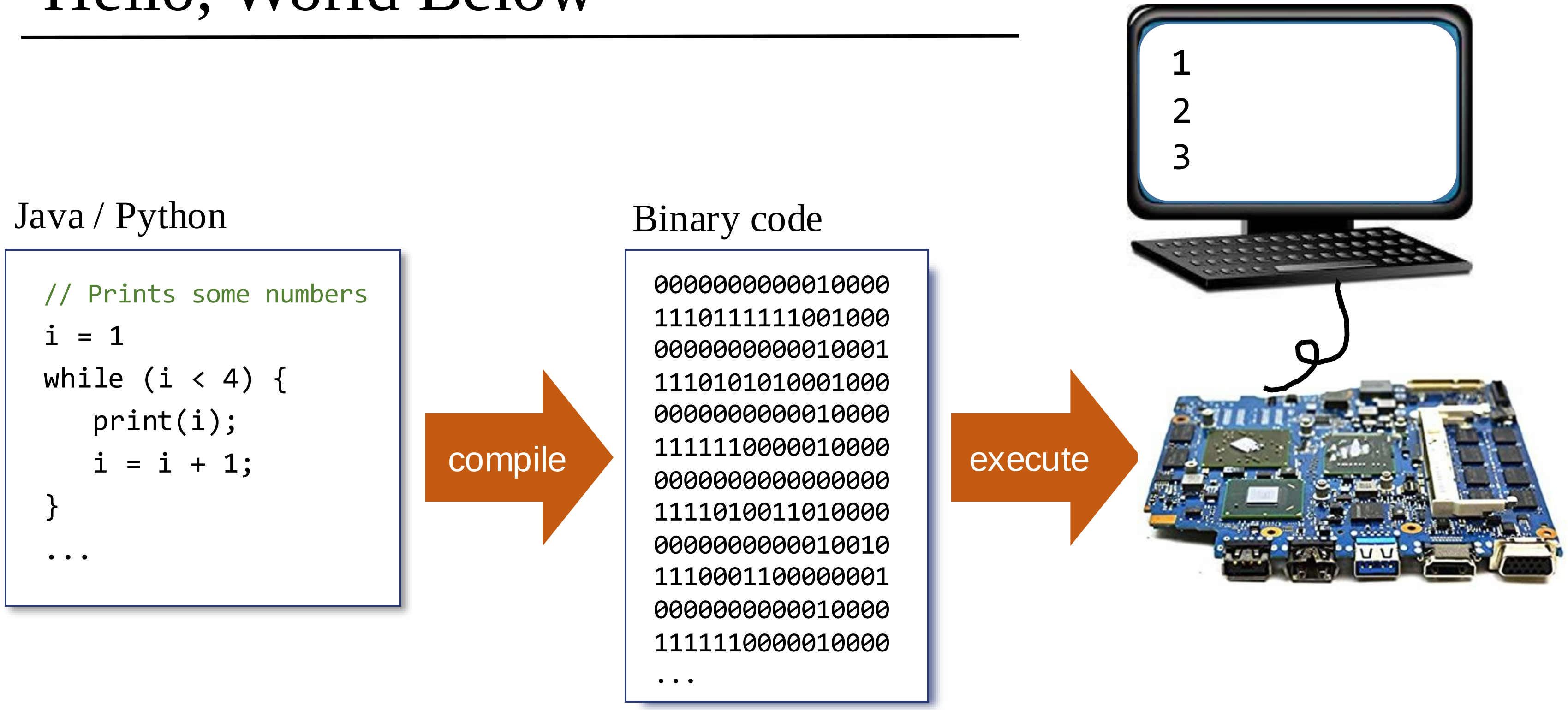
Binary code

```
0000000000010000
1110111111001000
0000000000010001
1110101010001000
0000000000010000
1111110000010000
0000000000000000
1111010011010000
0000000000010010
1110001100000001
0000000000010000
1111110000010000
...
```



```
1
2
3
```

Hello, World Below



Java / Python

```
// Prints some numbers
i = 1
while (i < 4) {
   print(i);
   i = i + 1;
}
...
```

compile

Binary code

```
0000000000010000
1110111111001000
0000000000010001
1110101010001000
0000000000010000
1111110000010000
0000000000000000
1111010011010000
0000000000010010
1110001100000001
0000000000010000
1111110000010000
...
```

execute

```
1
2
3
```

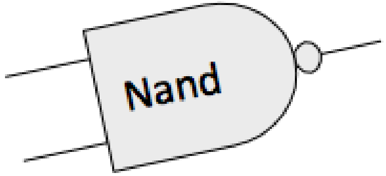
Software issues

- Compile?
- Execute?
- Runtime?
- …

Hardware issues

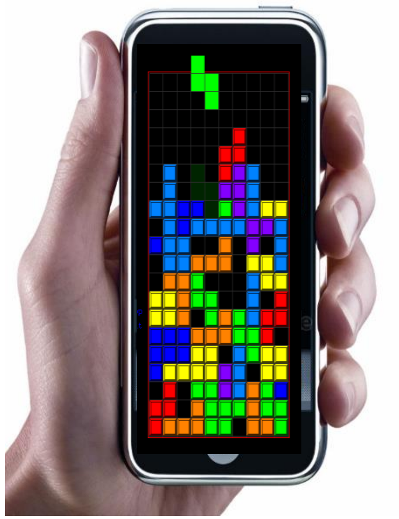
- Binary code?
- Chips?
- Screen?
- …

Nand to Tetris



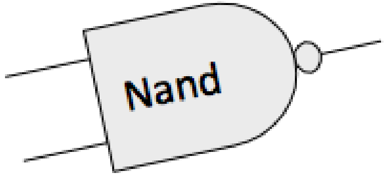
a	b	Nand
0	0	1
0	1	1
1	0	1
1	1	0

building a modern computer system from first principles



“What I hear I forget,
what I see I remember,
what I *do* I understand”

Nand to Tetris



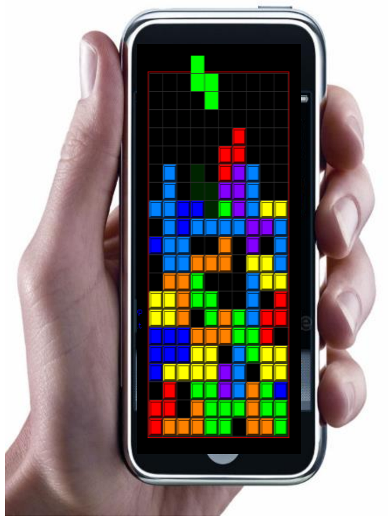
a	b	Nand
0	0	1
0	1	1
1	0	1
1	1	0

hardware platform

Projects 1-6

software hierarchy

Projects 7-12

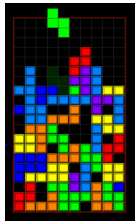


Using HDL and a hardware simulator

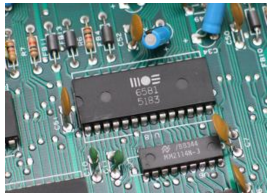
Using Java / Python
+ supplied specs and test programs

Nand to Tetris

abstract
concept

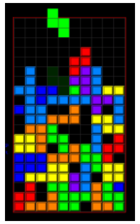


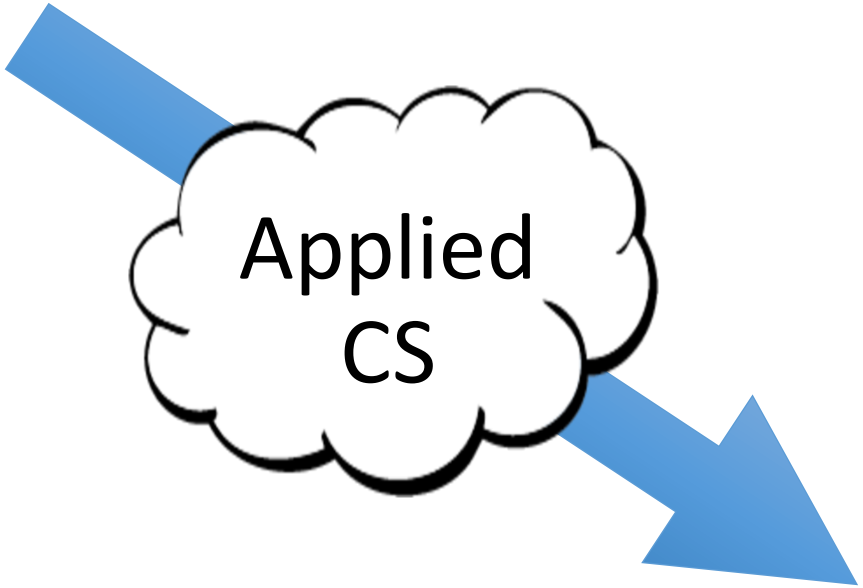
... 1001010100101101010
010010100101001010101
110011010001010010010
010011110010001000111
1111011010100101101 ...



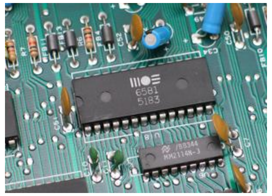
Nand to Tetris

abstract concept



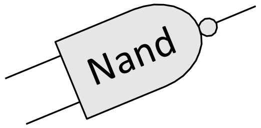

Applied CS

... 1001010100101101010
0100101001010010101
110011010001010010010
010011110010001000111
1111011010100101101 ...

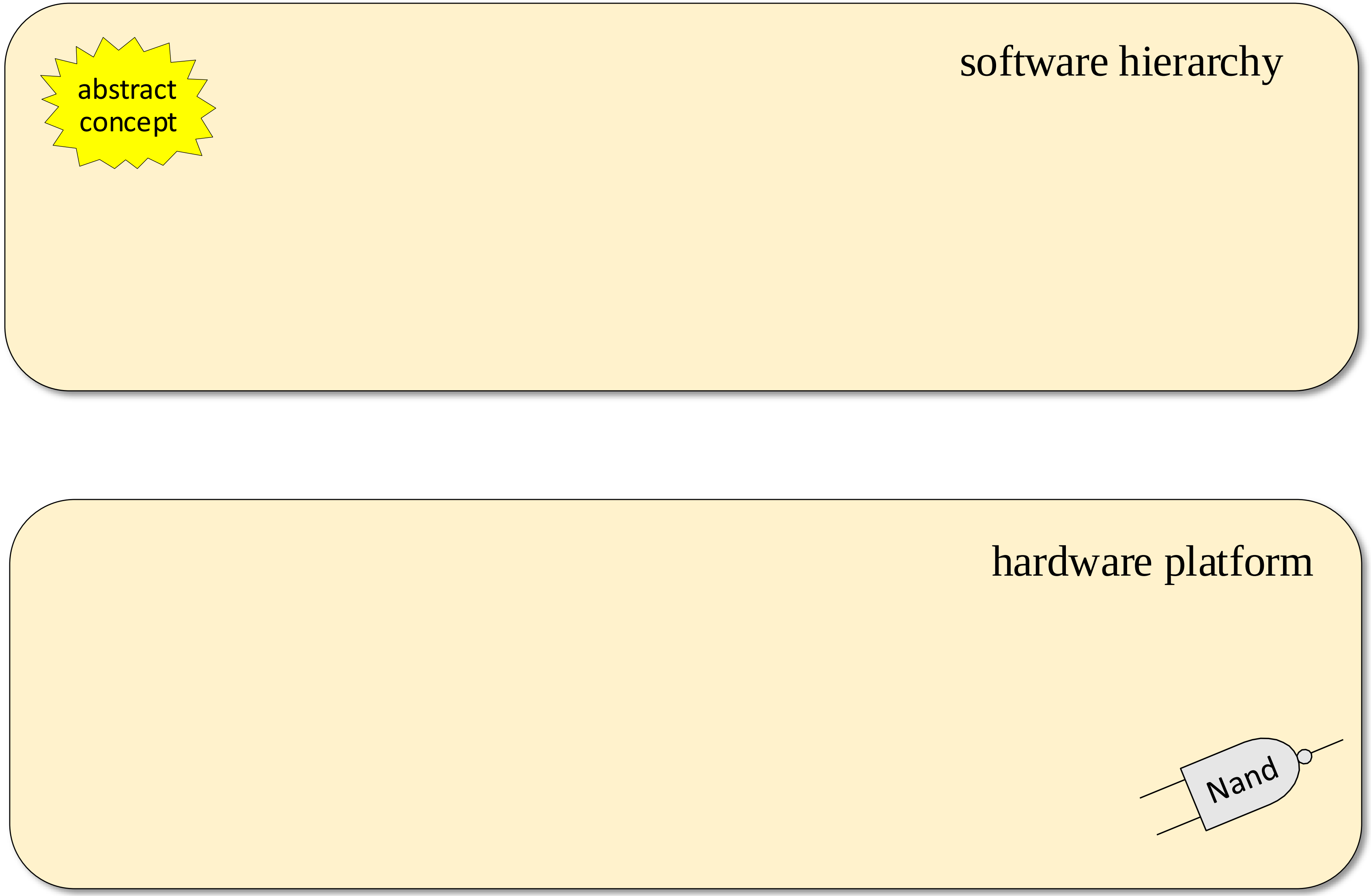


Nand to Tetris

abstract concept



Nand to Tetris

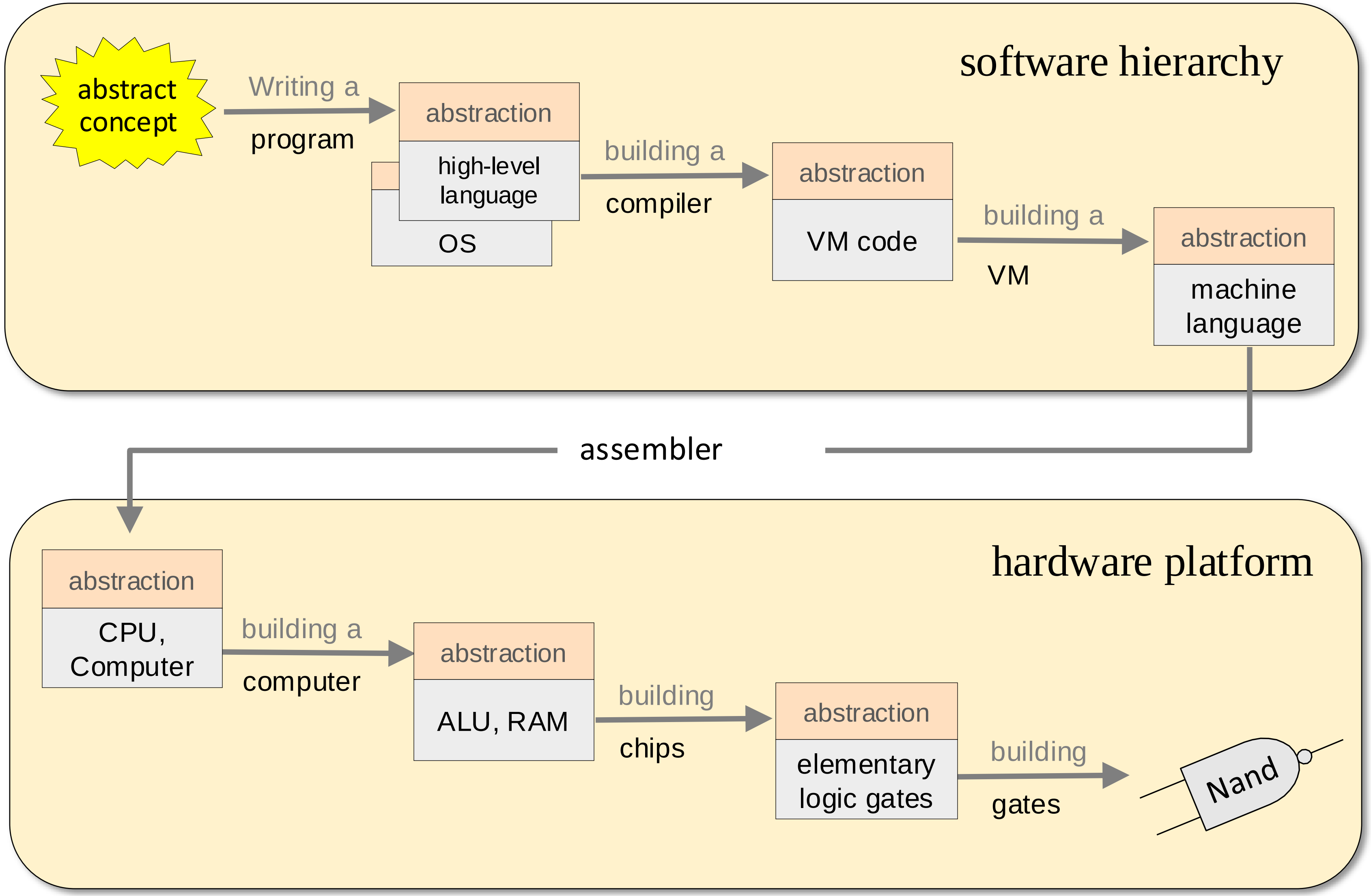


abstract concept

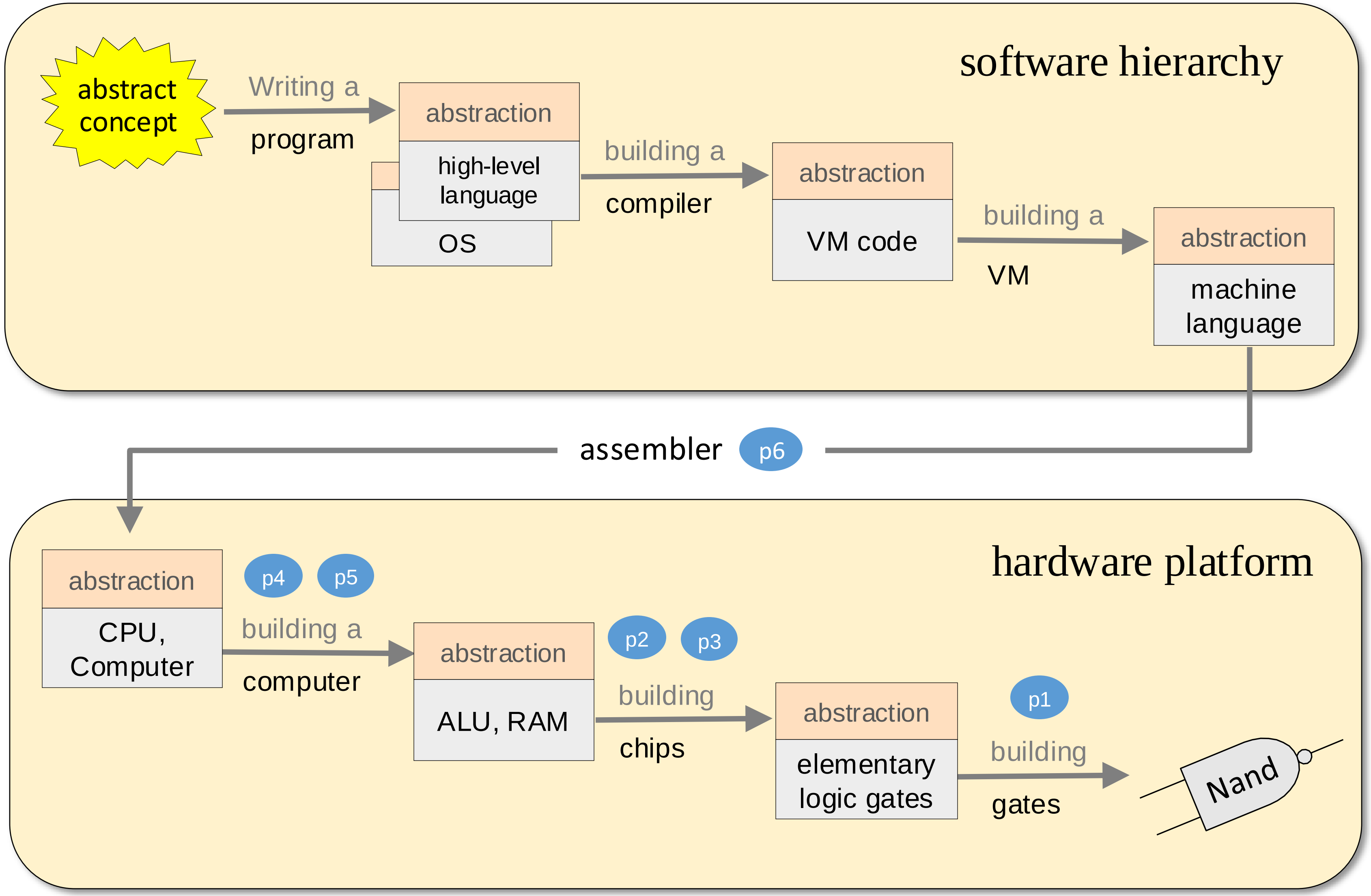
software hierarchy

hardware platform

Nand to Tetris



Nand to Tetris

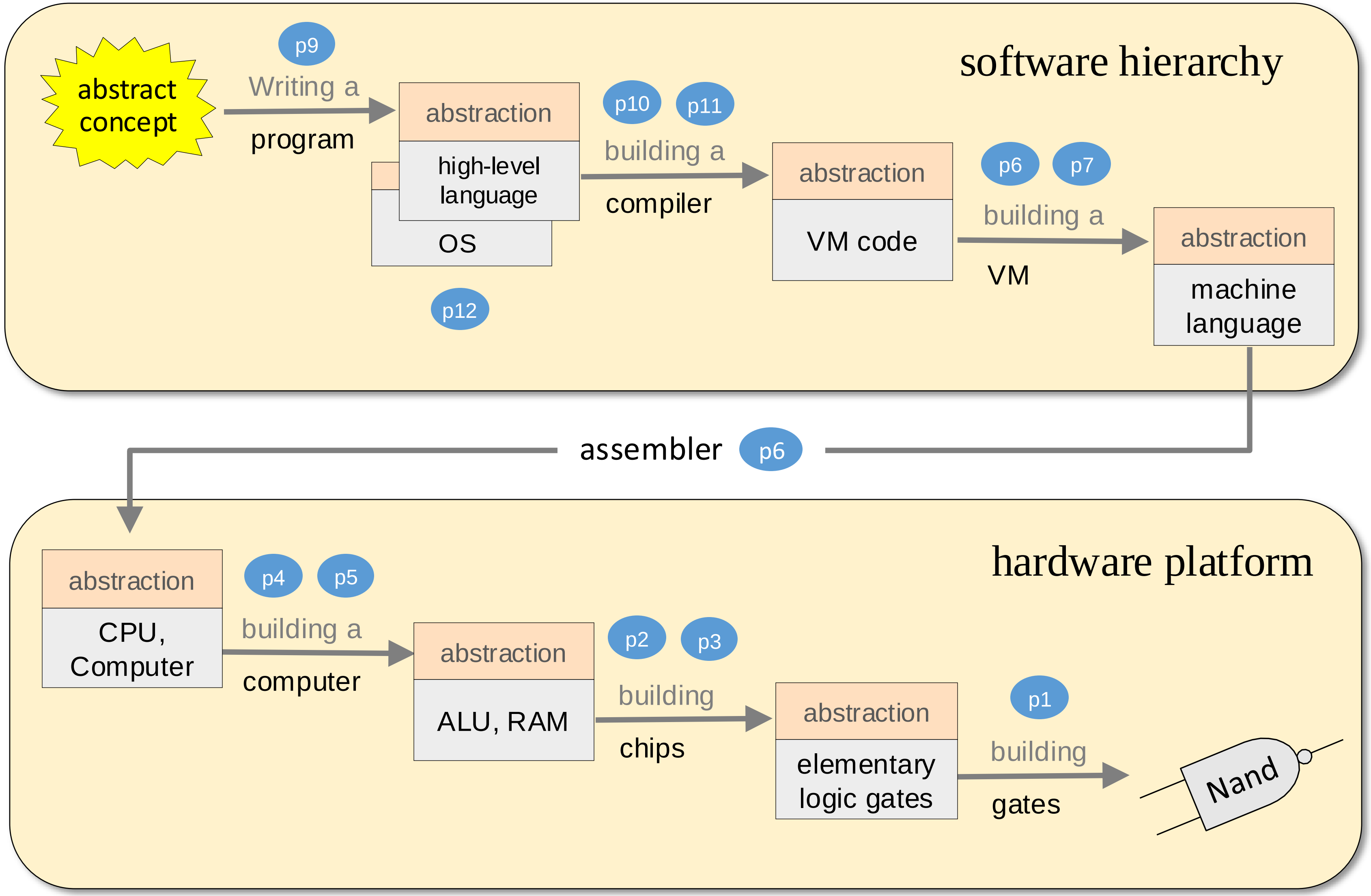


Part I

Building a general-purpose computer, capable of executing programs in machine language.

p = project,
lecture,
book chapter

Nand to Tetris



Part II

Building a software hierarchy, capable of compiling programs written in a high-level language

Part I

Building a general-purpose computer, capable of executing programs in machine language.

p = project,
lecture,
book chapter

Part I: Hardware

Given: `Nand(a,b)`

a	b	Nand(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

Part I: Hardware

Given: `Nand(a,b)`

Build:

```
Not(a)

And(a,b)

Or(a,b)

Xor(a,b)

Mux(s,a,b)

...

Adder

ALU

RAM

CPU

Computer
```

30 chips, leading up to a general-purpose computer platform

a	b	Nand(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

Part I: Hardware

Given: `Nand(a,b)`

`Not(a) = ?`

`And(a,b)`

`Or(a,b)`

`Xor(a,b)`

`Mux(s,a,b)`

`...`

`Adder`

`ALU`

`RAM`

`CPU`

`Computer`

a	b	Nand(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

Part I: Hardware

Given: `Nand(a,b)`

a	b	Nand(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

```
Not(a) =   Nand(a,a)

And(a,b)

Or(a,b)

Xor(a,b)

Mux(s,a,b)

...

Adder

ALU

RAM

CPU

Computer
```

Part I: Hardware

Given: `Nand(a,b)`

```
  Not(a) =   Nand(a,a)
➡ And(a,b) = ?
  Or(a,b)
  Xor(a,b)
  Mux(s,a,b)
  ...
  Adder
  ALU
  RAM
  CPU
  Computer
```

a	b	Nand(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

Part I: Hardware

Given: `Nand(a,b)`

```
Not(a) =   Nand(a,a)

And(a,b) = Not(Nand(a,b))

Or(a,b)

Xor(a,b)

Mux(s,a,b)

...

Adder

ALU

RAM

CPU

Computer
```

a	b	Nand(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

Part I: Hardware

Given: `Nand(a,b)`

```
  Not(a) =   Nand(a,a)

  And(a,b) = Not(Nand(a,b))

➡ Or(a,b) =  ?

  Xor(a,b)

  Mux(s,a,b)

  ...

  Adder

  ALU

  RAM

  CPU

  Computer
```

a	b	Nand(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

Part I: Hardware

Given: `Nand(a,b)`

```
Not(a) =   Nand(a,a)

And(a,b) = Not(Nand(a,b))

Or(a,b) =  Not(And(Not(a),Not(b)))

Xor(a,b)

Mux(s,a,b)

...

Adder

ALU

RAM

CPU

Computer
```

a	b	Nand(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

Part I: Hardware

Given: `Nand(a,b)`

```
Not(a) =   Nand(a,a)

And(a,b) = Not(Nand(a,b))

Or(a,b) =  Not(And(Not(a),Not(b)))

Xor(a,b) = ?

Mux(s,a,b)

...

Adder

ALU

RAM

CPU

Computer
```

a	b	Nand(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

Part I: Hardware

Given: `Nand(a,b)`

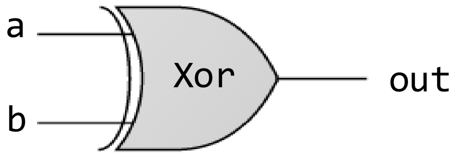
```
Not(a) =   Nand(a,a)

And(a,b) = Not(Nand(a,b))

Or(a,b) =  Not(And(Not(a),Not(b)))

Xor(a,b) = Or(And(a,Not(b)),And(Not(a),b))

Mux(s,a,b)

...

Adder

ALU

RAM

CPU

Computer
```

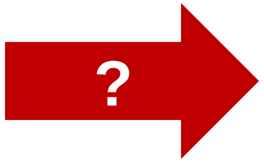
a	b	Nand(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

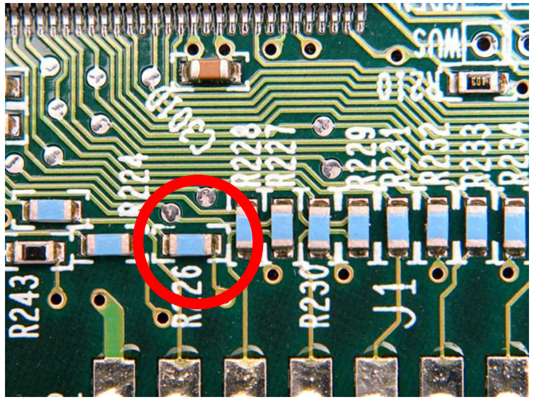
<u>Tools</u>

- Chip specifications / test scripts
- Hardware Description Language
- Hardware simulator.

Building a chip





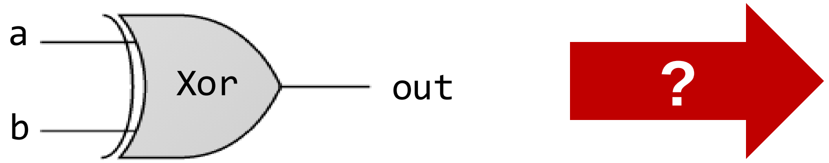


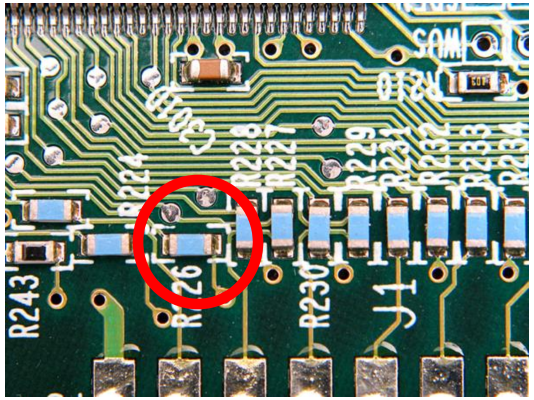
```
if ((a == 0 and b == 1) or (a == 1 and b == 0))
    out = 1
else
    out = 0
```

The process

- Design the chip architecture
- Specify the architecture in HDL
- Test the chip in a hardware simulator
- Optimize the design
- Realize the optimized design in silicon.

Building a chip



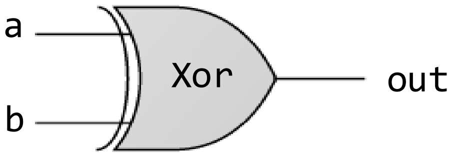


```
if ((a == 0 and b == 1) or (a == 1 and b == 0))
    out = 1
else
    out = 0
```

<u>The process</u>

- ✓ Design the chip architecture
- ✓ Specify the architecture in HDL
- ✓ Test the chip in a hardware simulator
- Optimize the design
- Realize the optimized design in silicon.

Chip design



```
if ((a == 0 and b == 1) or (a == 1 and b == 0))
    out = 1
else
    out = 0
```

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

Requirement

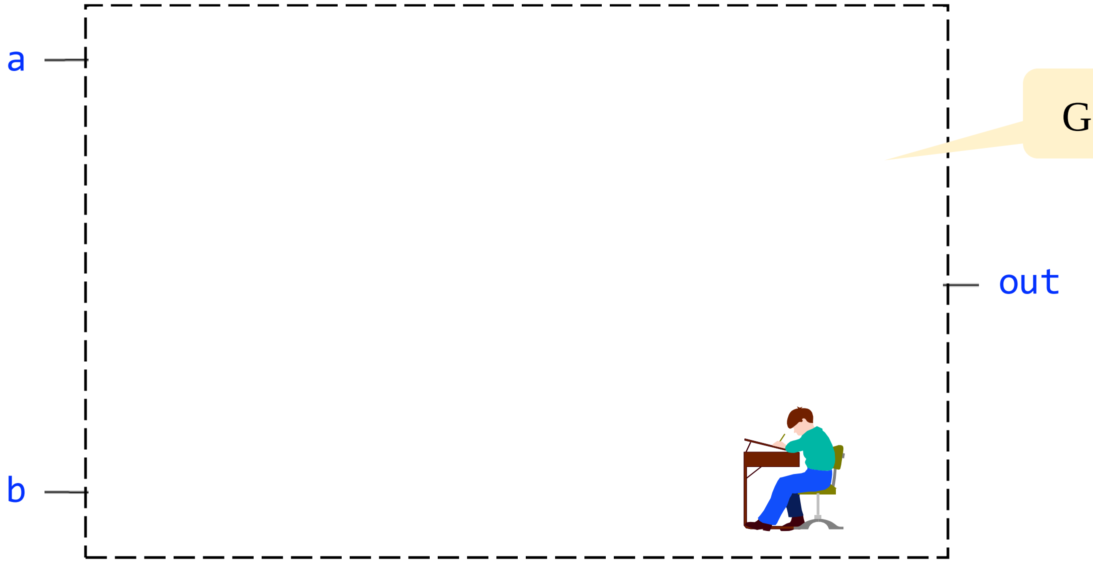
Build a chip that delivers this functionality

```
/** out = (a And Not(b)) Or (Not(a) And b)) */
CHIP Xor {
    IN a, b;
    OUT out;
    PARTS:
    // Missing implementation
}
```

HDL program

```
/** Chips set (APIs): */
...
Not (in= , out= );
And (a= , b= , out= );
Or  (a= , b= , out= );
Xor (a= , b= , out= );
...
```

Chip design



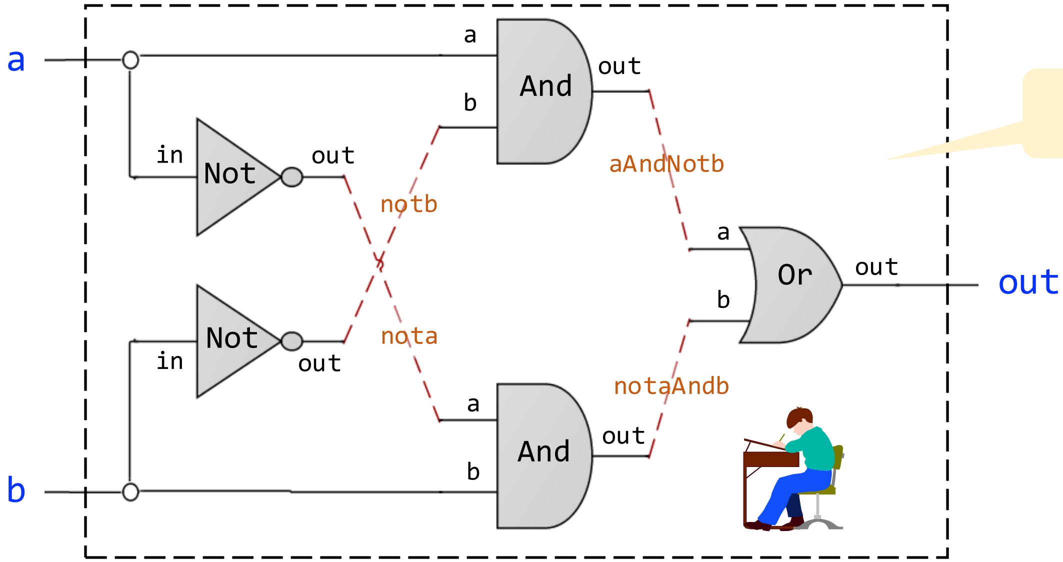
Gate diagram

```
/** out = (a And Not(b)) Or (Not(a) And b))  */
CHIP Xor {
    IN a, b;
    OUT out;
    PARTS:
    // Missing implementation
}
```

HDL program

```
/** Chips set (APIs): */
...
Not (in= , out= );
And (a= , b= , out= );
Or  (a= , b= , out= );
Xor (a= , b= , out= );
...
```

Chip design



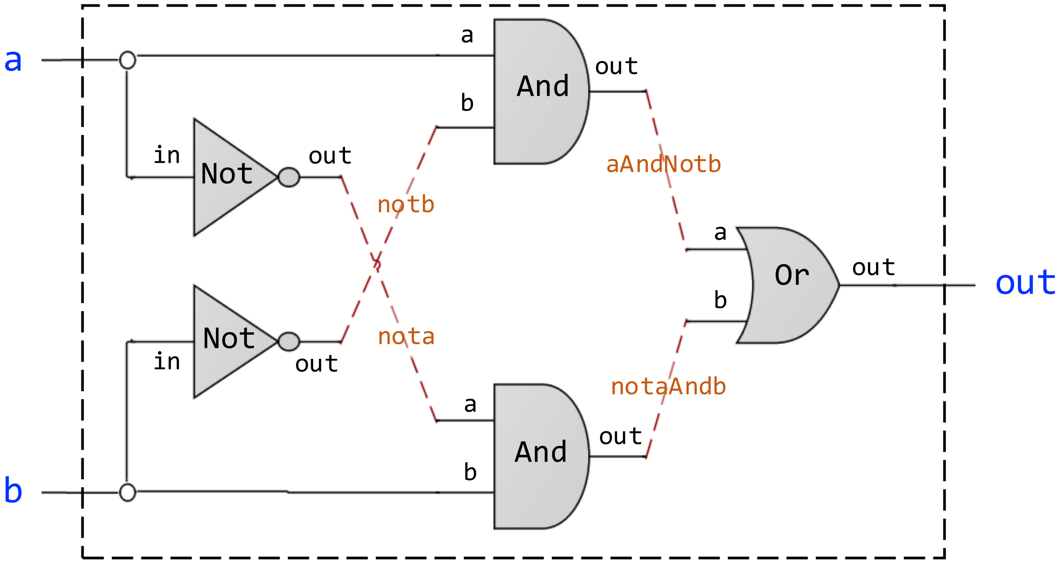
Gate diagram

```
/** out = (a And Not(b)) Or (Not(a) And b))  */
CHIP Xor {
    IN a, b;
    OUT out;
    PARTS:
    // Missing implementation
}
```

HDL program

```
/** Chips set (APIs): */
...
Not (in= , out= );
And (a= , b= , out= );
Or  (a= , b= , out= );
Xor (a= , b= , out= );
...
```

Chip design

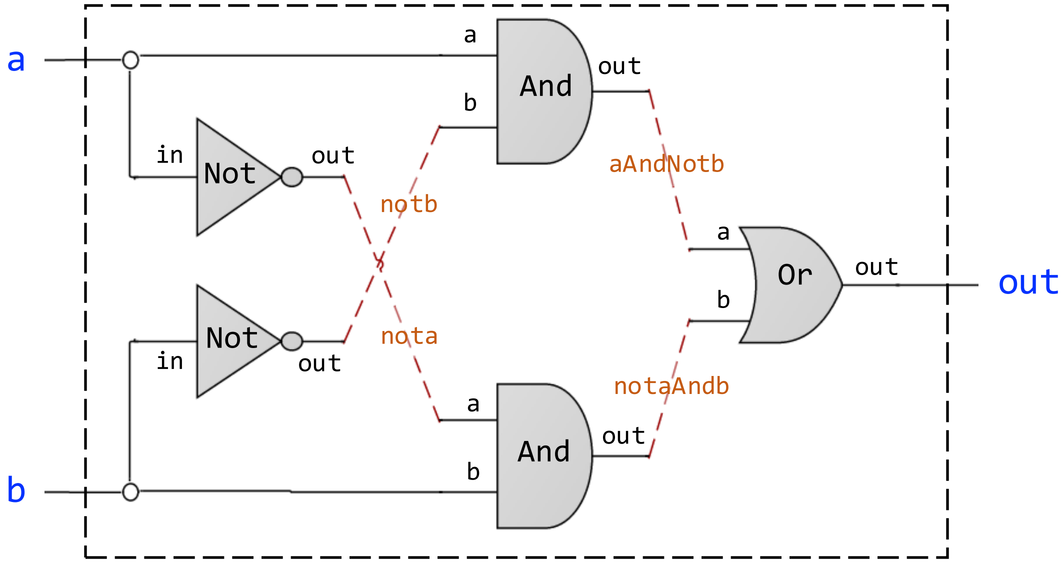


```
/** out = (a And Not(b)) Or (Not(a) And b))  */
CHIP Xor {
    IN a, b;
    OUT out;
    PARTS:
    // Missing implementation
}
```

HDL *program*

```
/** Chips set (APIs): */
...
Not (in= , out= );
And (a= , b= , out= );
Or  (a= , b= , out= );
Xor (a= , b= , out= );
...
```

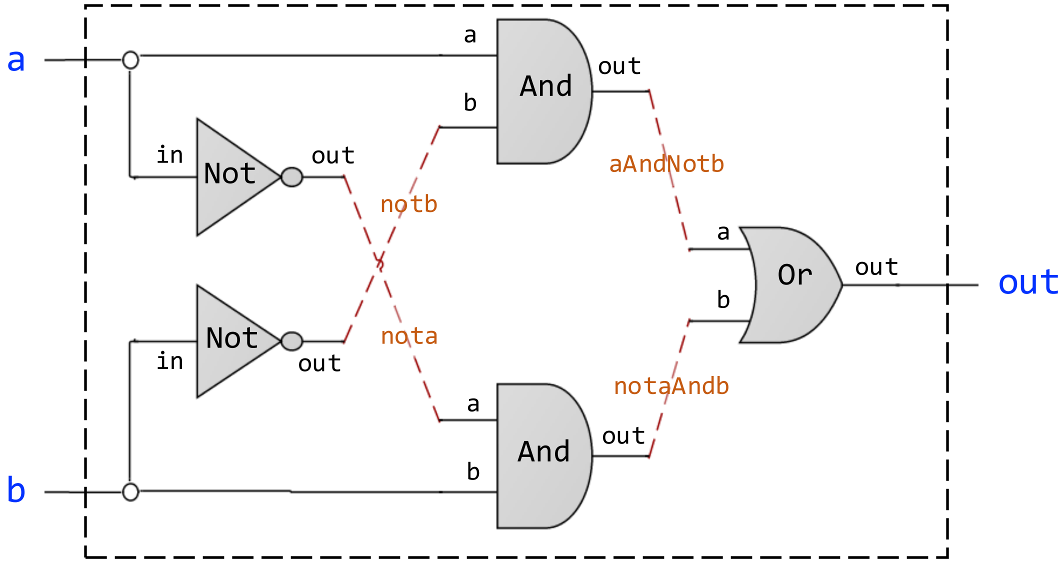
Chip design



```
/** out = (a And Not(b)) Or (Not(a) And b))  */
CHIP Xor {
    IN a, b;
    OUT out;
    PARTS:
    Not (in=a, out=nota);
    Not (in=b, out=notb);
    And (a=a, b=notb, out=aAndNotb);
    And (a=nota, b=b, out=notaAndb);
    Or  (a=aAndNotb, b=notaAndb, out=out);
}
```

```
/** Chips set (APIs): */
...
Not (in= , out= );
And (a= , b= , out= );
Or  (a= , b= , out= );
Xor (a= , b= , out= );
...
```

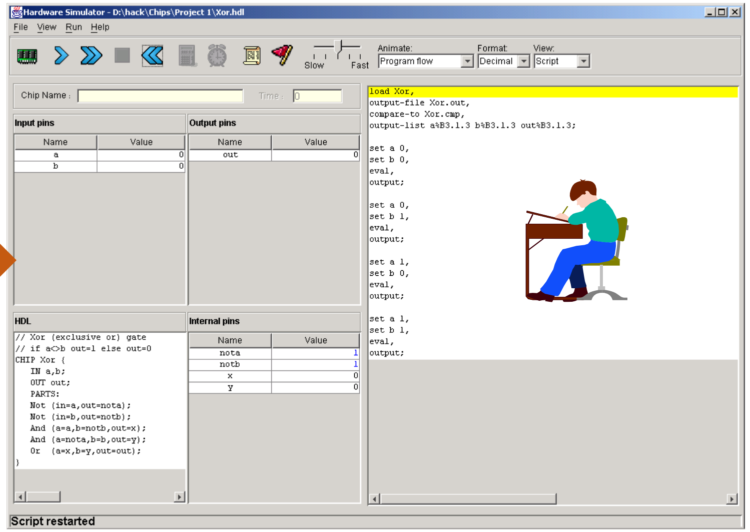
Chip design



```
/** out = (a And Not(b)) Or (Not(a) And b))  */
CHIP Xor {
    IN a, b;
    OUT out;
    PARTS:
    Not (in=a, out=nota);
    Not (in=b, out=notb);
    And (a=a, b=notb, out=aAndNotb);
    And (a=nota, b=b, out=notaAndb);
    Or  (a=aAndNotb, b=notaAndb, out=out);
}
```

simulate

Hardware simulator



Animate: Program flow
Format: Decimal
View: Script

Chip Name :
Time : 0

Input pins

Name	Value
a	0
b	0

Output pins

Name	Value
out	0

HDL

```
// Xor (exclusive or) gate
// if a<>b out=1 else out=0
CHIP Xor {
   IN a,b;
   OUT out;
   PARTS:
   Not (in=a,out=nota);
   Not (in=b,out=notb);
   And (a=a,b=notb,out=x);
   And (a=nota,b=b,out=y);
   Or  (a=x,b=y,out=out);
}
```

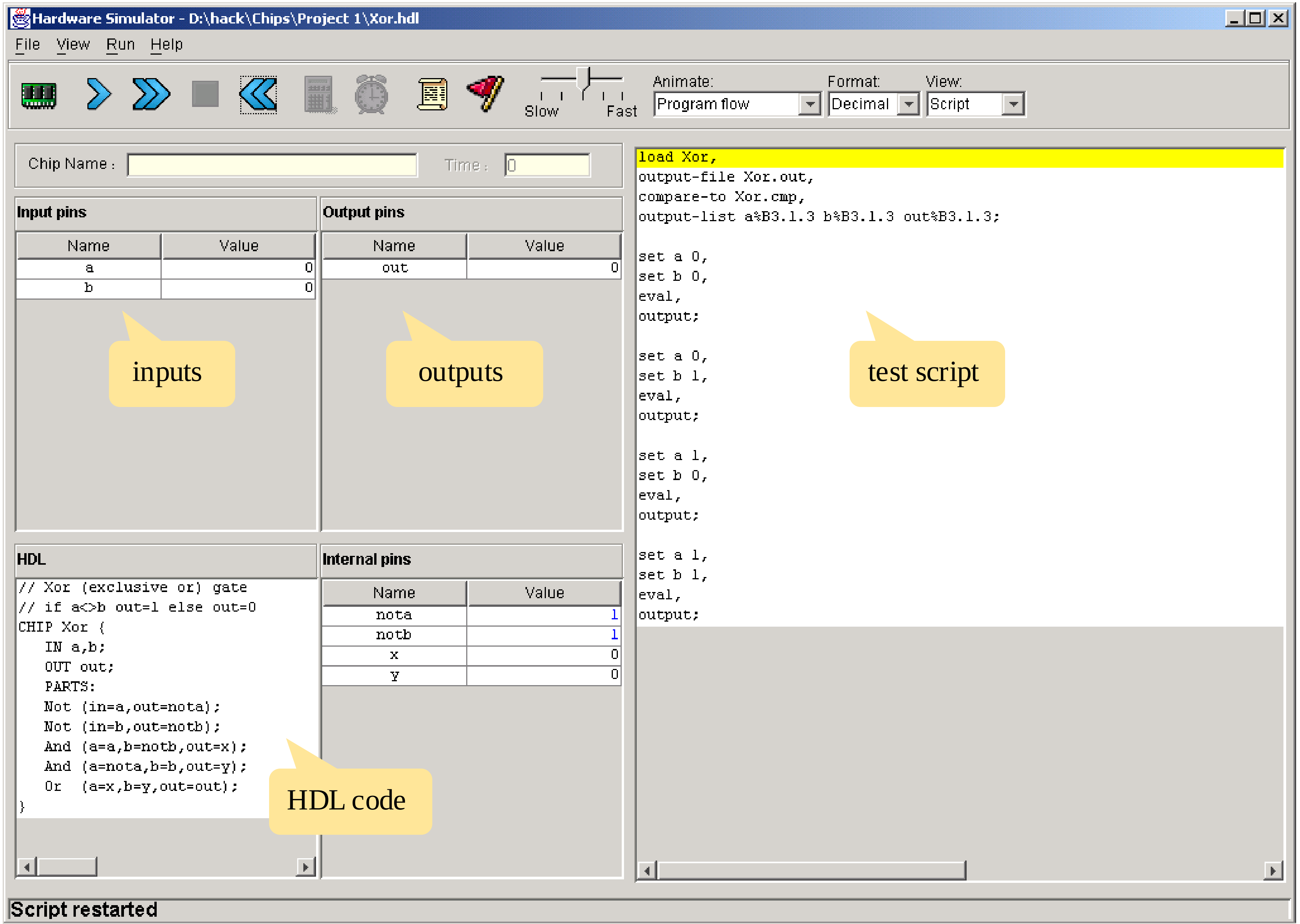
Internal pins

Name	Value
nota	1
notb	1
x	0
y	0

```
load Xor,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

set a 0,
set b 0,
eval,
output;

set a 0,
set b 1,
eval,
output;

set a 1,
set b 0,
eval,
output;

set a 1,
set b 1,
eval,
output;
```

Script restarted

Chip simulation



Hardware Simulator - D:\hack\Chips\Project 1\Xor.hdl

Animate: Program flow

Format: Decimal

View: Script

Chip Name :

Time : 0

Input pins

Name	Value
a	0
b	0

inputs

Output pins

Name	Value
out	0

outputs

HDL

```
// Xor (exclusive or) gate
// if a<>b out=1 else out=0
CHIP Xor {
   IN a,b;
   OUT out;
   PARTS:
   Not (in=a,out=nota);
   Not (in=b,out=notb);
   And (a=a,b=notb,out=x);
   And (a=nota,b=b,out=y);
   Or  (a=x,b=y,out=out);
}
```

HDL code

Internal pins

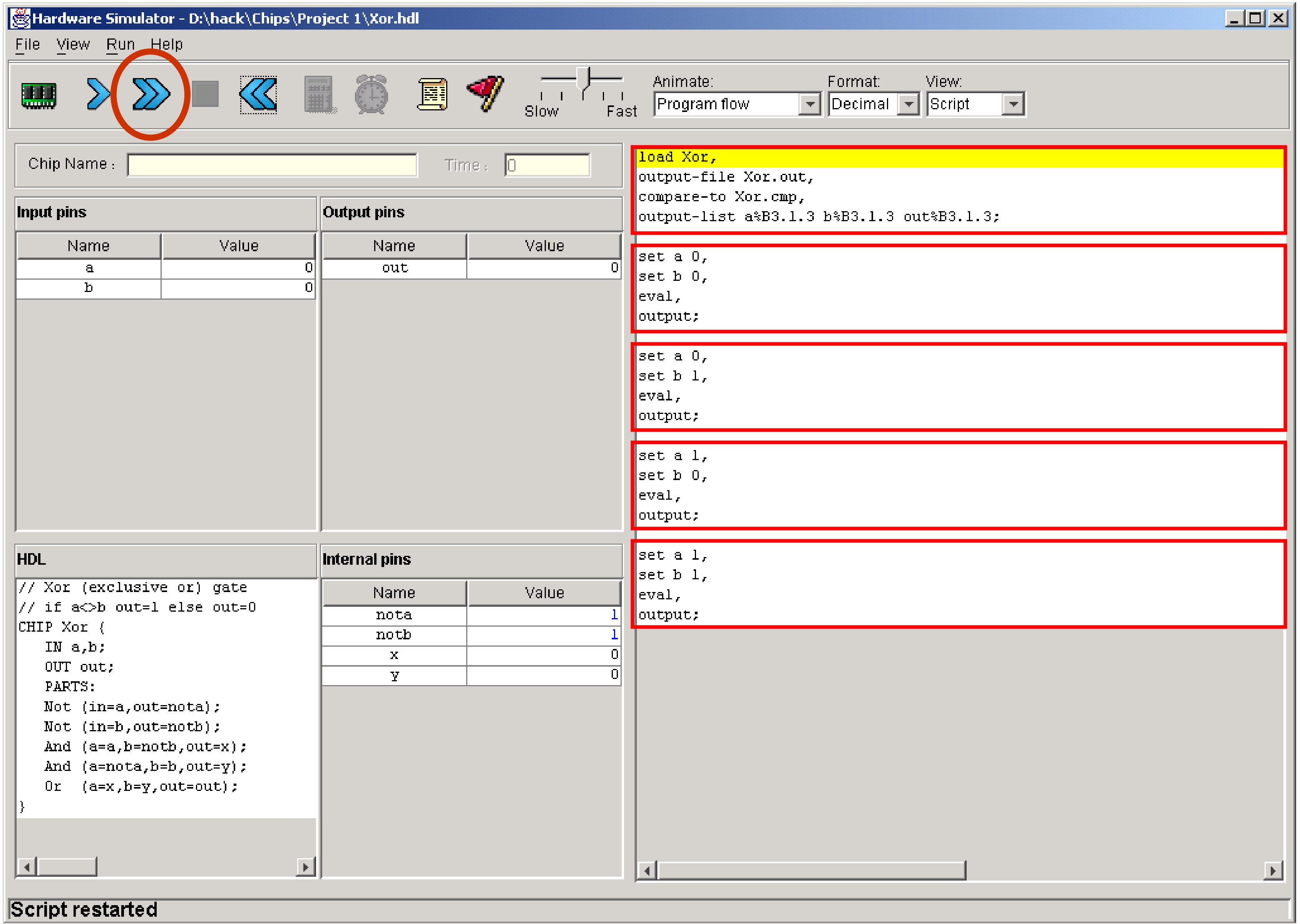
Name	Value
nota	1
notb	1
x	0
y	0

```
load Xor,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

set a 0,
set b 0,
eval,
output;

set a 0,
set b 1,
eval,
output;

set a 1,
set b 0,
eval,
output;

set a 1,
set b 1,
eval,
output;
```

test script

Script restarted

Chip simulation



Animate: Program flow
Format: Decimal
View: Script

Chip Name :
Time : 0

Input pins

Name	Value
a	0
b	0

Output pins

Name	Value
out	0

HDL

```
// Xor (exclusive or) gate
// if a<>b out=1 else out=0
CHIP Xor {
   IN a,b;
   OUT out;
   PARTS:
   Not (in=a,out=nota);
   Not (in=b,out=notb);
   And (a=a,b=notb,out=x);
   And (a=nota,b=b,out=y);
   Or  (a=x,b=y,out=out);
}
```

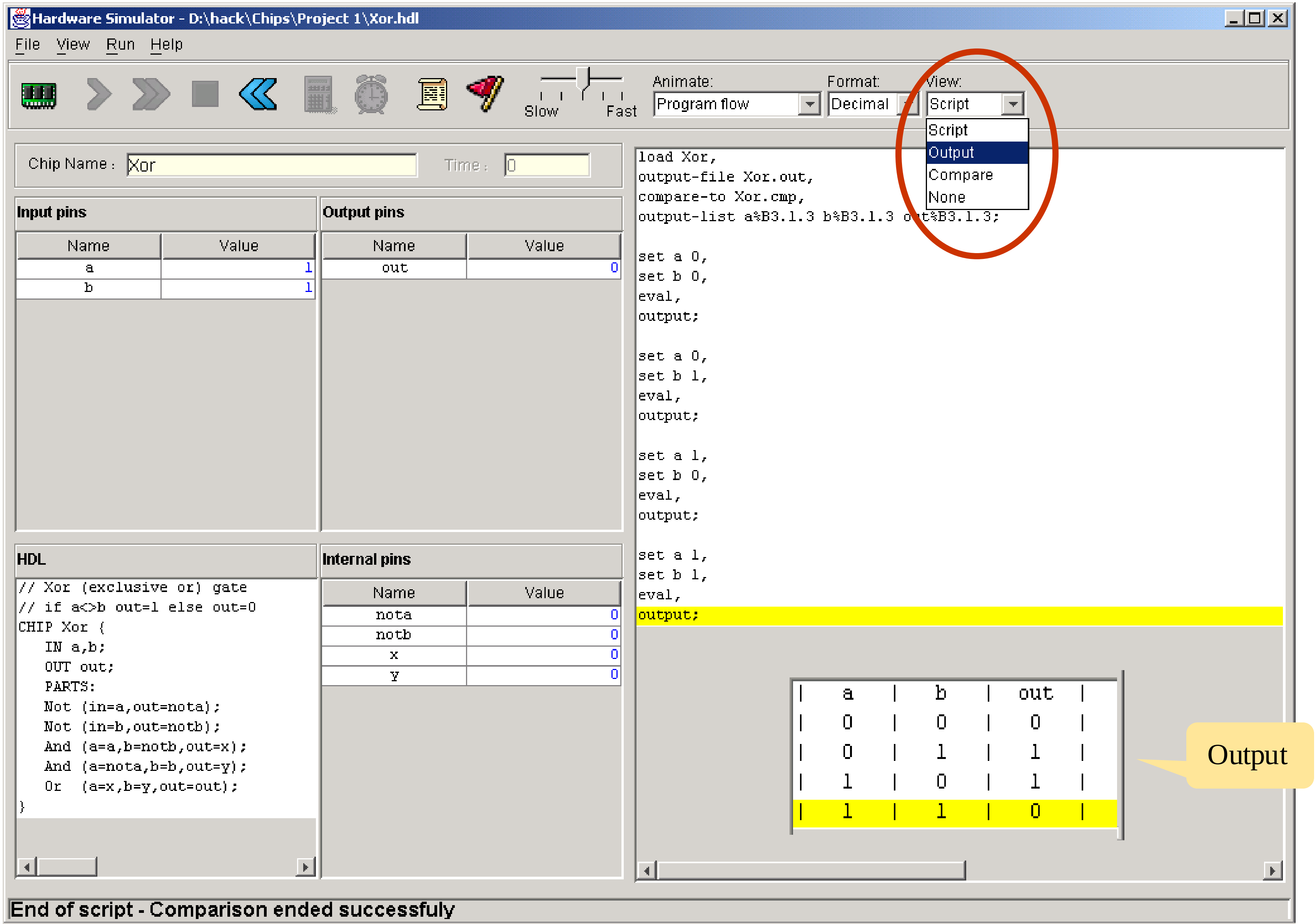
Internal pins

Name	Value
nota	1
notb	1
x	0
y	0

```
load Xor,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

set a 0,
set b 0,
eval,
output;

set a 0,
set b 1,
eval,
output;

set a 1,
set b 0,
eval,
output;

set a 1,
set b 1,
eval,
output;
```

Script restarted

Chip simulation



Animate: Program flow

Format: Decimal

View: Script

Chip Name : Xor

Time : 0

Input pins

Name	Value
a	1
b	1

Output pins

Name	Value
out	0

HDL

```
// Xor (exclusive or) gate
// if a<>b out=1 else out=0
CHIP Xor {
   IN a,b;
   OUT out;
   PARTS:
   Not (in=a,out=nota);
   Not (in=b,out=notb);
   And (a=a,b=notb,out=x);
   And (a=nota,b=b,out=y);
   Or  (a=x,b=y,out=out);
}
```

Internal pins

Name	Value
nota	0
notb	0
x	0
y	0

```
load Xor,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

set a 0,
set b 0,
eval,
output;

set a 0,
set b 1,
eval,
output;

set a 1,
set b 0,
eval,
output;

set a 1,
set b 1,
eval,
output;
```

```
|  a  |  b  | out |
|  0  |  0  |  0  |
|  0  |  1  |  1  |
|  1  |  0  |  1  |
|  1  |  1  |  0  |
```

End of script - Comparison ended successfuly

Output

Hardware platform

Given: `Nand`

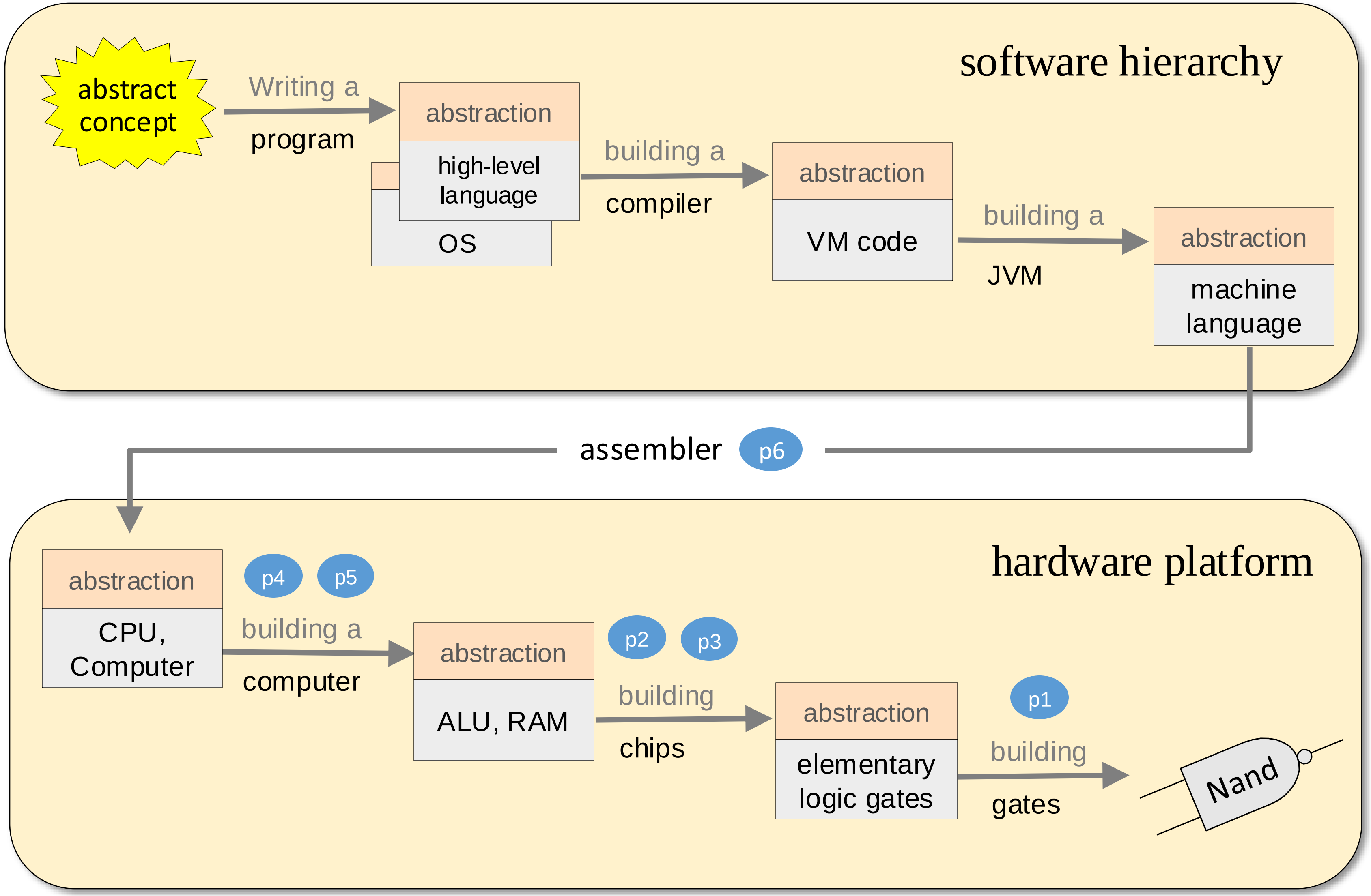
Build:

```
Not

And

Or

Xor

Mux

...

Adder

ALU

RAM

CPU

Computer
```

Using similar techniques, we build the entire chip-set, leading up to a general-purpose computer system

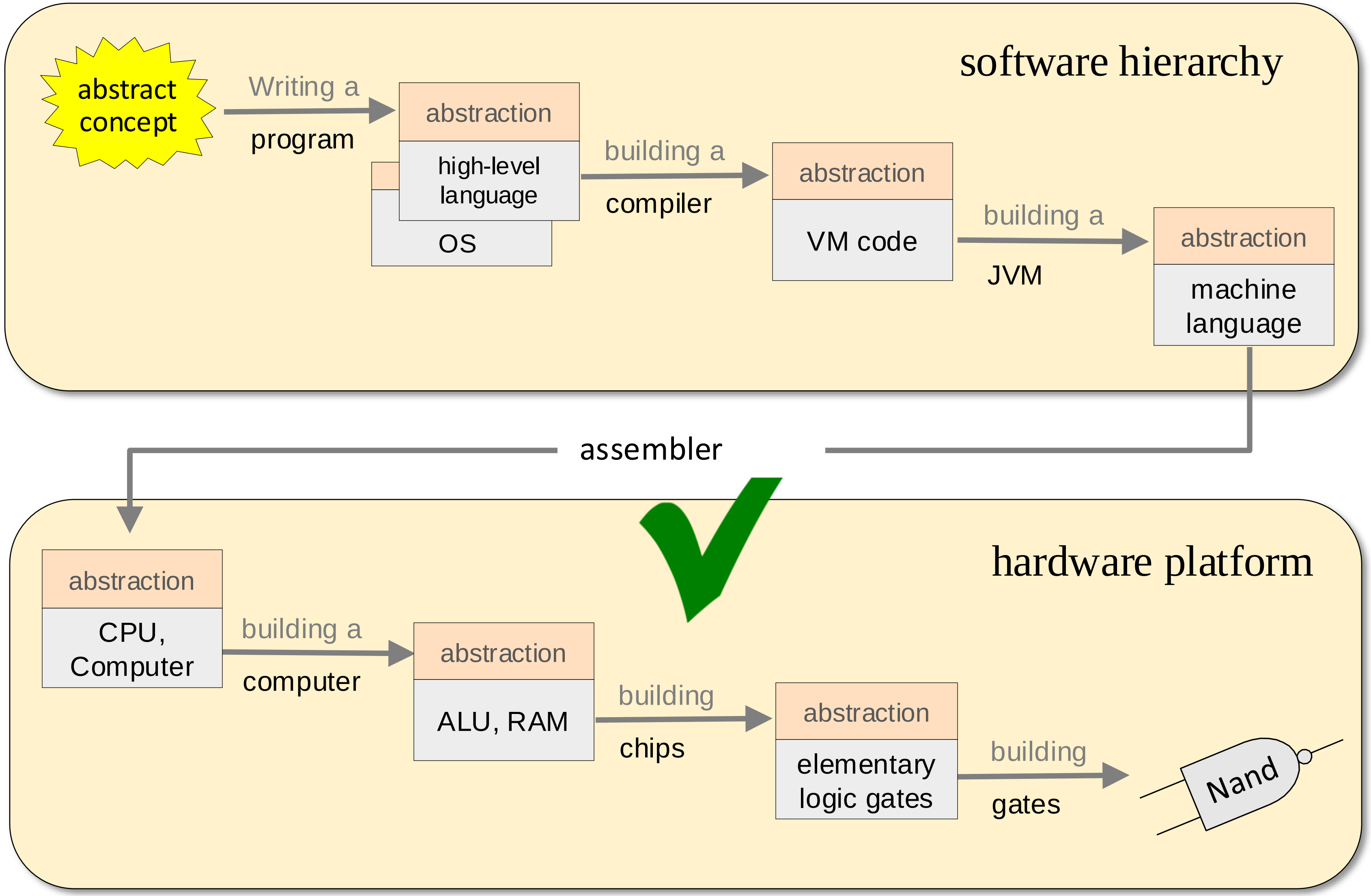
(Projects 1, 2, 3, 4, 5, 6)

Nand to Tetris: Part I



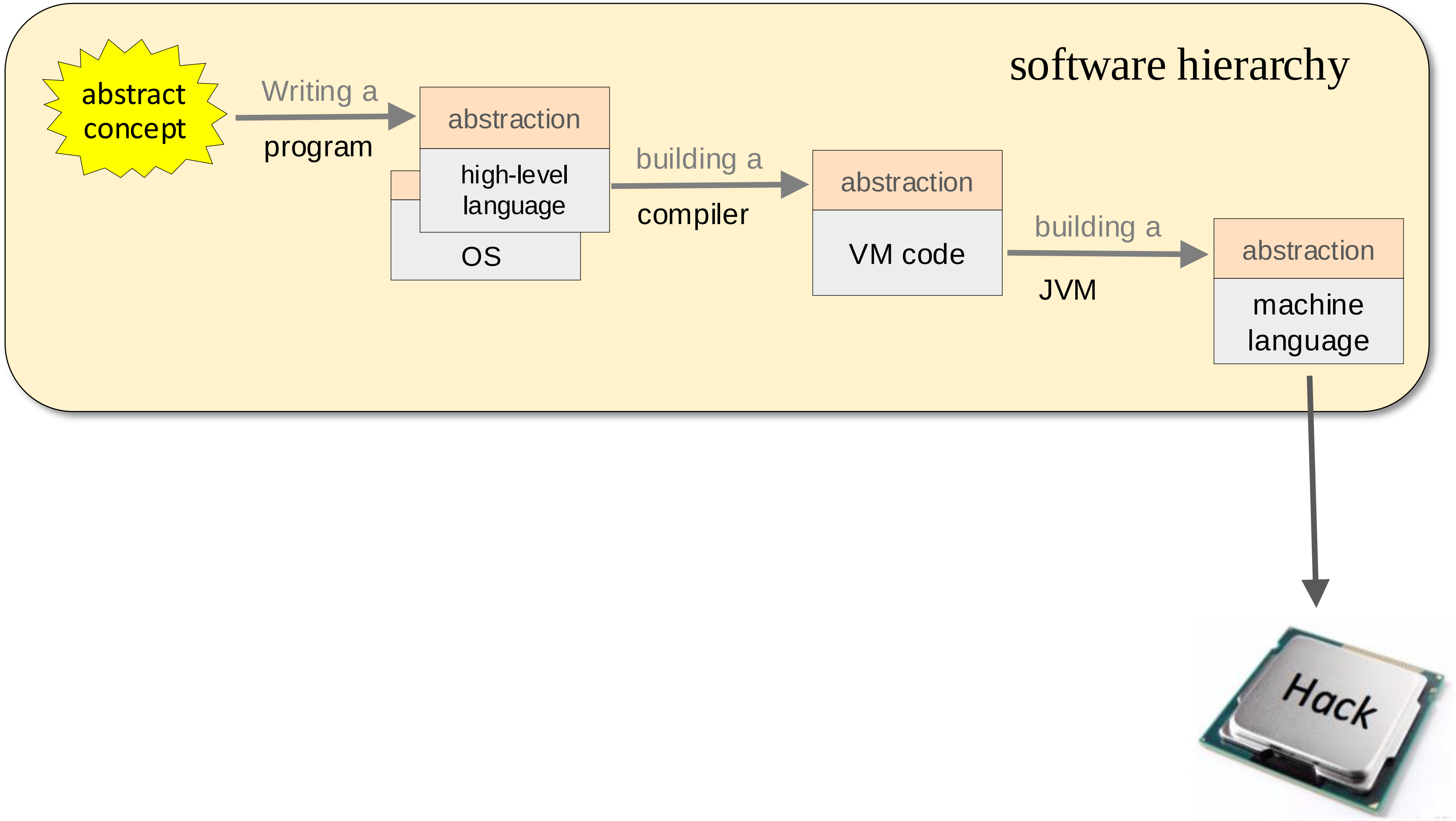
p = project,
lecture,
book chapter

Nand to Tetris: Part I



p = project,
lecture,
book chapter

Nand to Tetris: Part II



software hierarchy

abstract concept

Writing a program

abstraction

high-level language

OS

building a compiler

abstraction

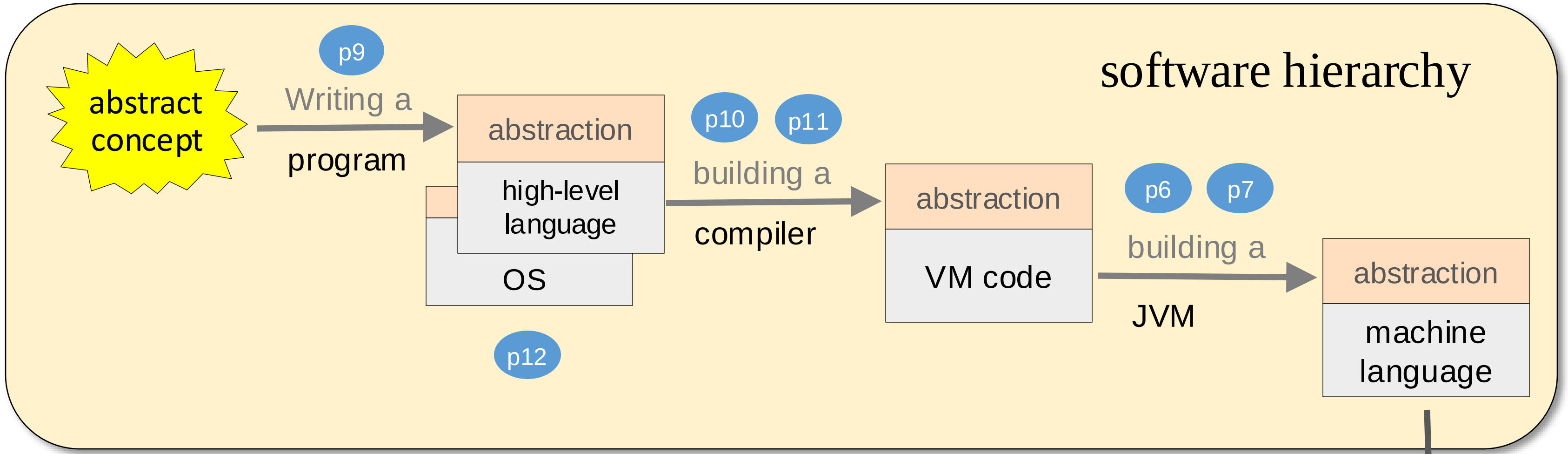
VM code

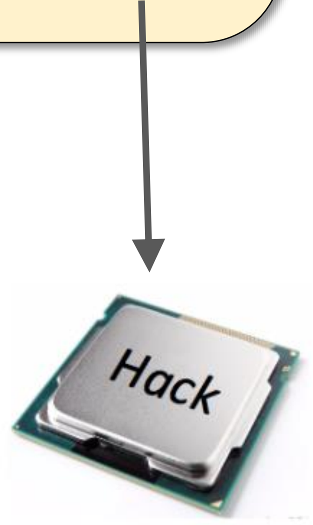
building a JVM

abstraction

machine language

Nand to Tetris: Part II



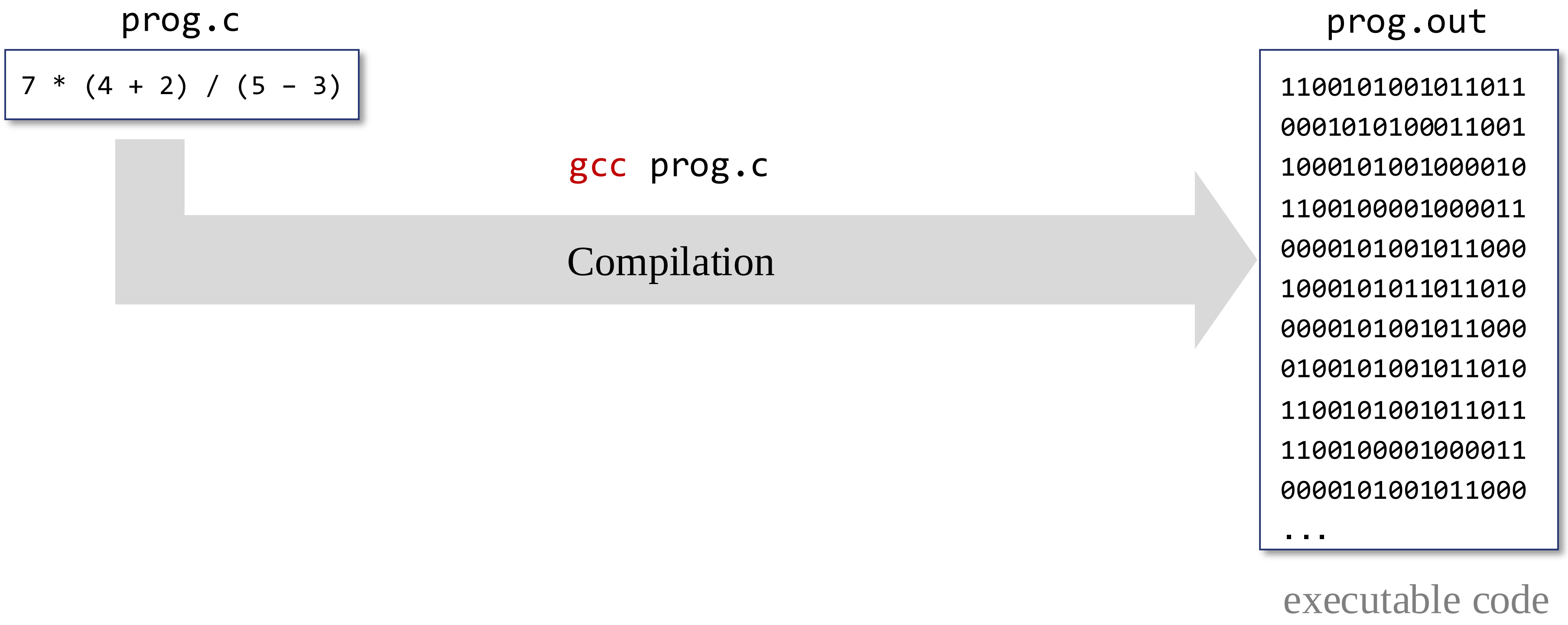


Part II:

Building a software infrastructure for compiling programs written in high-level languages (Java, Python, Jack, ...):

- Compiler
- VM
- OS

One-tier compilation example: C



prog.c

```
7 * (4 + 2) / (5 - 3)
```

```
gcc prog.c
```

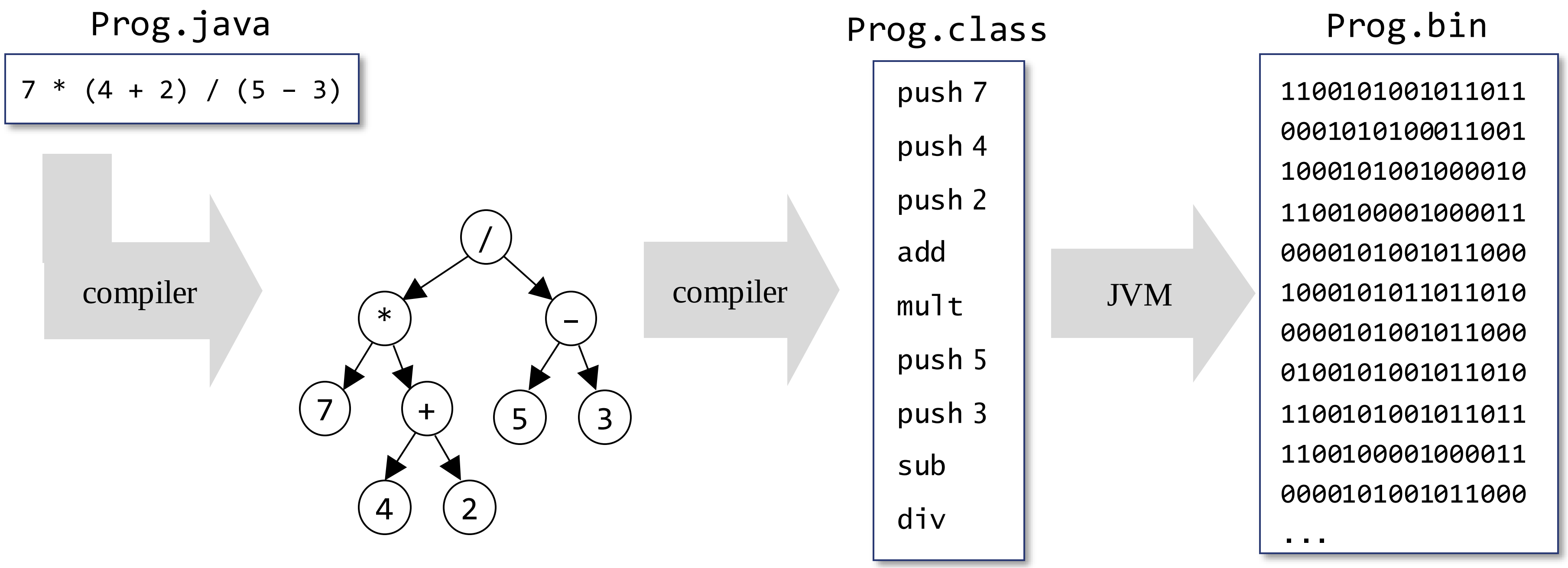
Compilation

prog.out

```
1100101001011011
0001010100011001
1000101001000010
1100100001000011
0000101001011000
1000101011011010
0000101001011000
0100101001011010
1100101001011011
1100100001000011
0000101001011000
...
```

executable code

Two-tier compilation example: Java



Prog.java

```
7 * (4 + 2) / (5 – 3)
```

source code

parse tree

Prog.class

```
push 7
push 4
push 2
add
mult
push 5
push 3
sub
div
```

(operates on a virtual, stack-based machine)

Prog.bin

```
1100101001011011
0001010100011001
1000101001000010
1100100001000011
0000101001011000
1000101011011010
0000101001011000
0100101001011010
1100101001011011
1100100001000011
0000101001011000
...
```

executable code

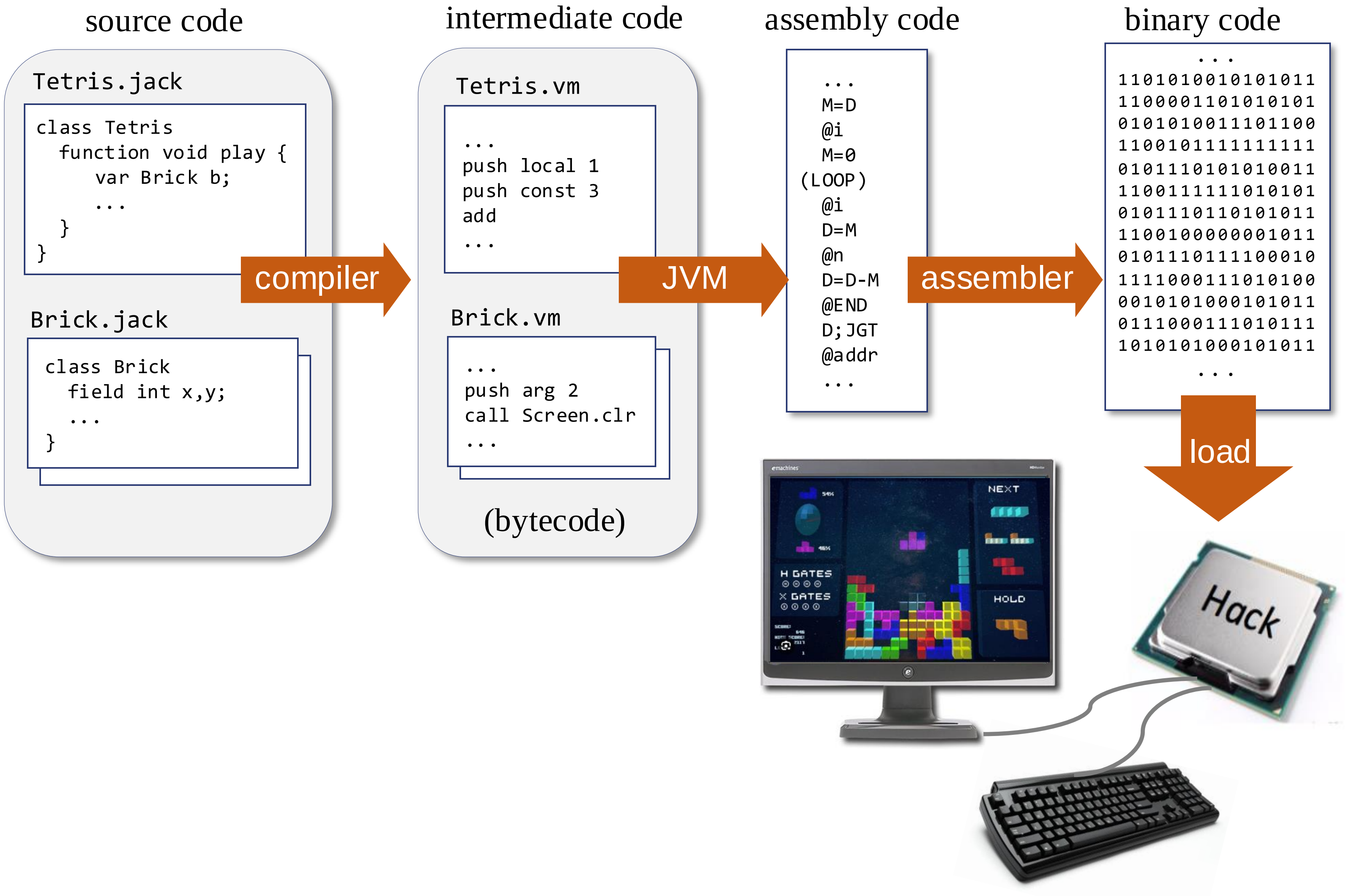
`javac Prog.java`

(translates Java code into bytecode)

`java Prog`

(translates bytecode into binary code, and executes it)

Compilation: Nand to Tetris



source code

Tetris.jack

```
class Tetris
  function void play {
     var Brick b;
     ...
  }
}
```

Brick.jack

```
class Brick
  field int x,y;
  ...
}
```

intermediate code

Tetris.vm

```
...
push local 1
push const 3
add
...
```

Brick.vm

```
...
push arg 2
call Screen.clr
...
```

(bytecode)

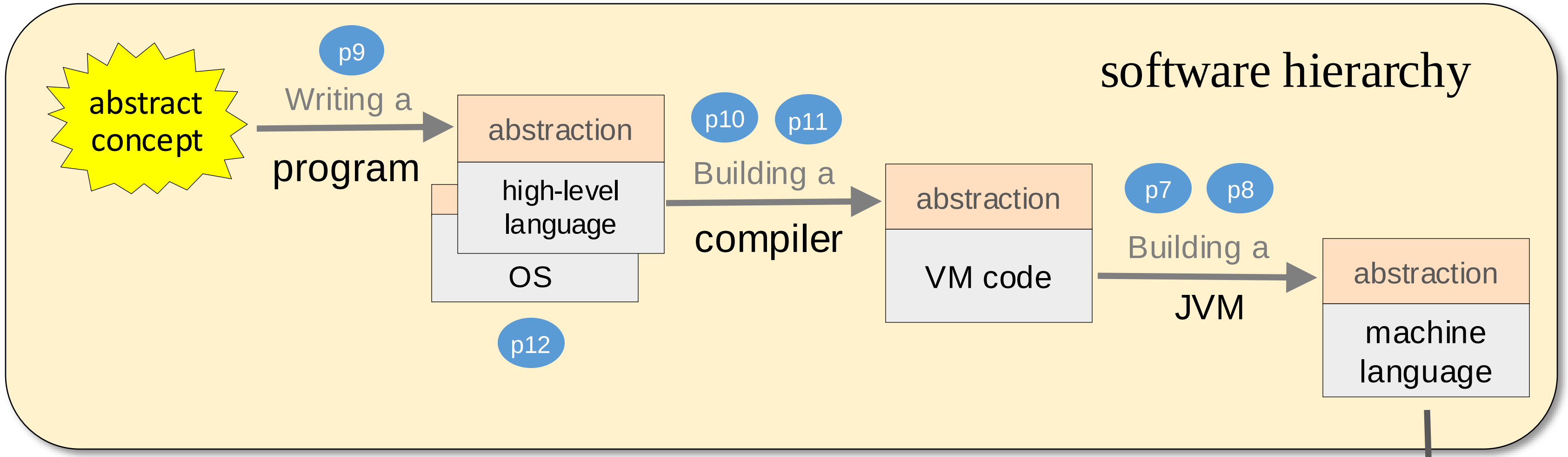
assembly code

```
   ...
   M=D
   @i
   M=0
 (LOOP)
   @i
   D=M
   @n
   D=D-M
   @END
   D;JGT
   @addr
   ...
```

binary code

```
       ...
1101010010101011
1100001101010101
0101010011101100
1100101111111111
0101110101010011
1100111111010101
0101110110101011
1100100000001011
0101110111100010
1111000111010100
0010101000101011
0111000111010111
1010101000101011
       ...
```

Nand to Tetris Roadmap: Part II



software hierarchy

abstract concept

p9

Writing a program

abstraction

high-level language

OS

p12

p10 p11

Building a compiler

abstraction

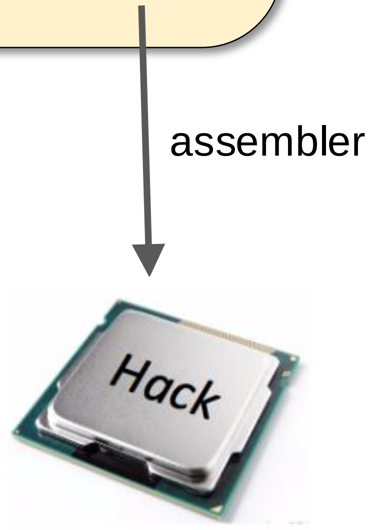
VM code

p7 p8

Building a JVM

abstraction

machine language



assembler

Hack

Part II Projects

7, 8: building a JVM

9: writing a some Jack programs

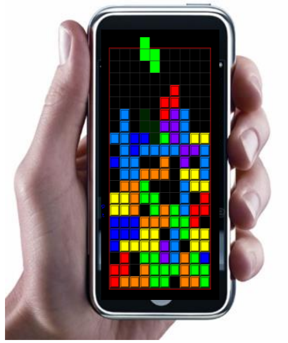
10, 11: building the compiler

12: building an operating system.

Take home lessons

a	b	Nand
0	0	1
0	1	1
1	0	1
1	1	0

Nand to Tetris



Hardware: Logic gates, Boolean arithmetic, multiplexors, flip-flops, registers, RAM units, counters, Hardware Description Language, chip simulation and testing.

Architecture: ALU/CPU design and implementation, addressing modes, memory-mapped I/O, machine code, assembly language programming,

Programming Languages: Object-based design and programming, abstract data types, scoping rules, syntax and semantics, references.

Compilation: Lexical analysis, top-down parsing, symbol tables, pushdown automata, virtual machine, code generation, implementation of arrays and objects.

Data structures and algorithms: Stacks, trees, hash tables, lists, recursion, arithmetic algorithms, geometric algorithms, time / space complexity

Engineering: Abstraction / implementation, modular design, API design and documentation, unit testing, quality assurance, programming at the large.