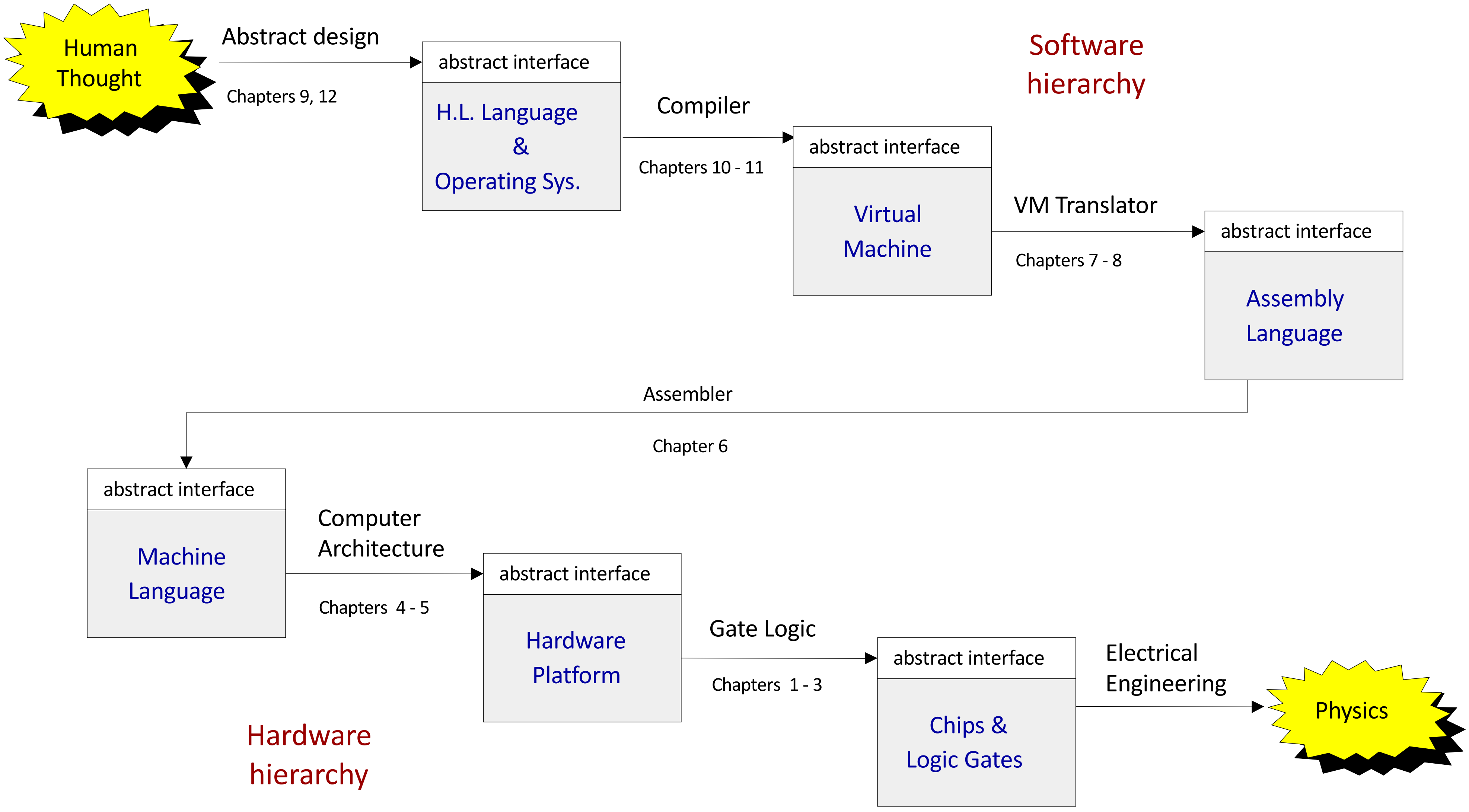
School of Computer Science

COMP SCI 2000 Computer Systems
Lecture 1

The Course

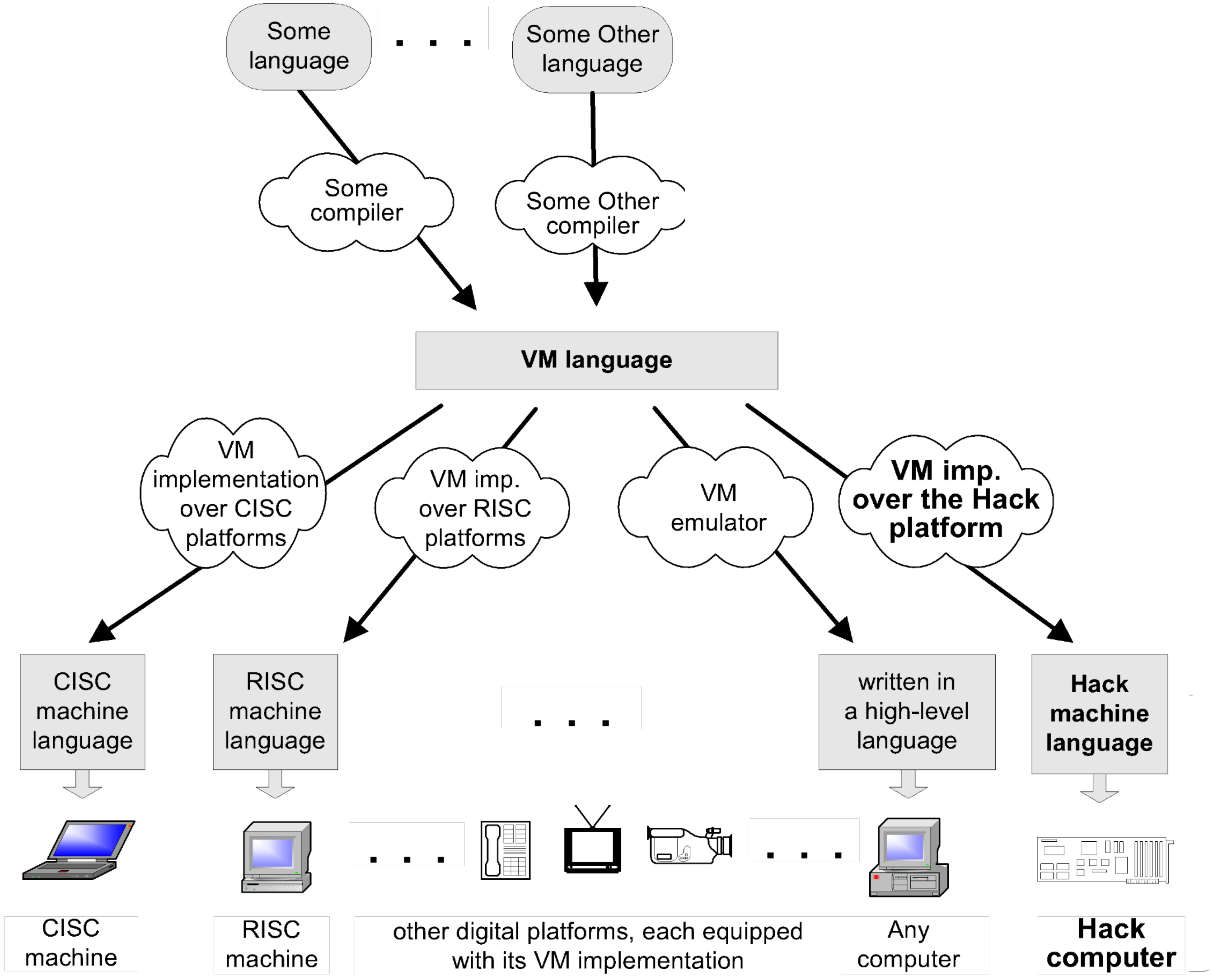
- This course is based on the Nand2Tetris course
  - "Elements of Computing Systems", Nisan & Schocken
  - You will build a complete computer system, layer by layer, starting with Nand gates and ending with a compiler.
- How it will work
  - We cover material in lectures supported by the text book, you get to practice in workshops and assignments with quizzes and exams to help you test your knowledge.
  - Our assignments are not the Nand2Tetris projects.
  - You will keep a logbook of your software development process that will be used by us to track how you're going. It will be interleaved with your **svn** commits and web submissions.
  - Do all of the work and do it yourself. We are teaching you **effective** techniques at the moment – the goal is your learning rather than just getting projects done!

The whole system

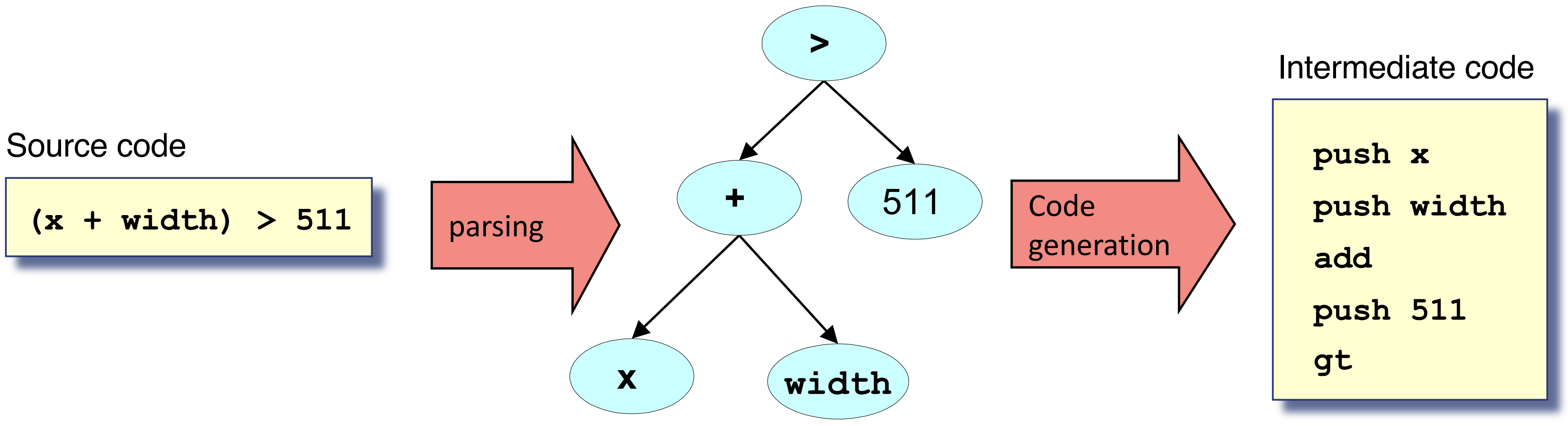


(Abstraction–implementation paradigm)

A modern compilation model



Compilation



Source code

```
(x + width) > 511
```

parsing

Code generation

Intermediate code

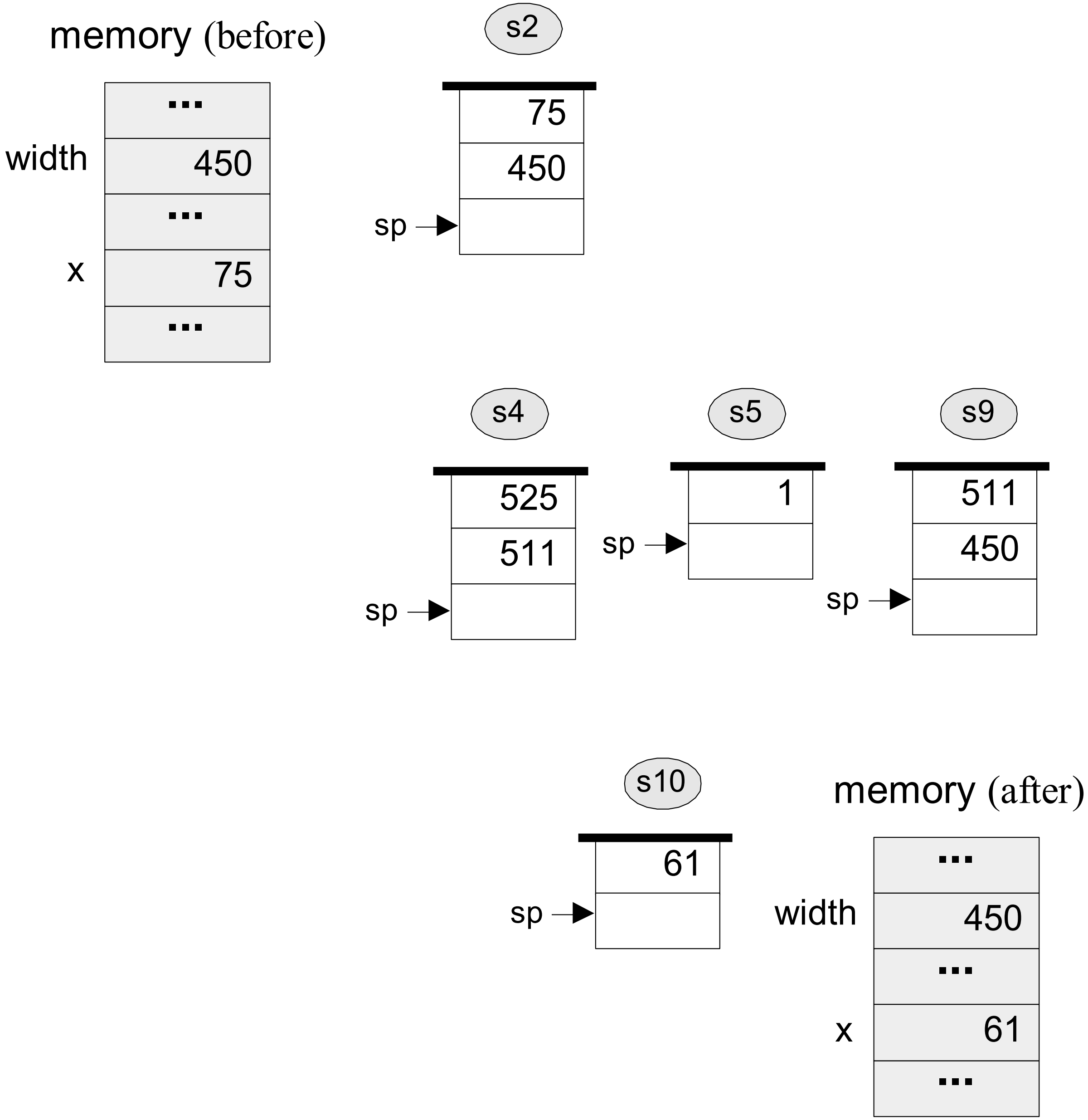
```
push x
push width
add
push 511
gt
```

Abstraction Syntax Analysis Parse Tree Semantic Synthesis

Inside a virtual machine

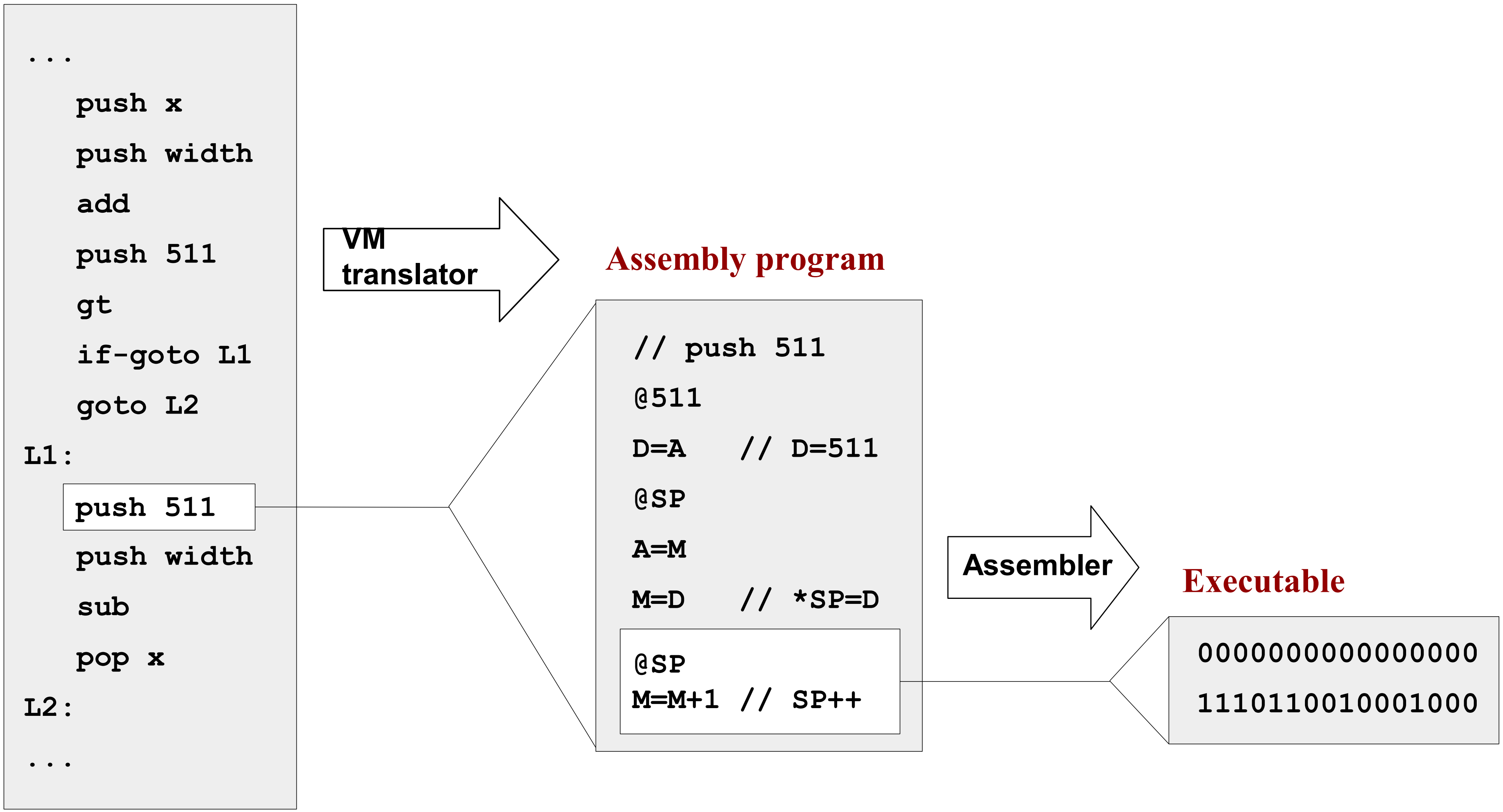
```
if ((x+width)>511)
{
    let x=511-width;
}
```

```
// 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 low-level programming path

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:
...
```

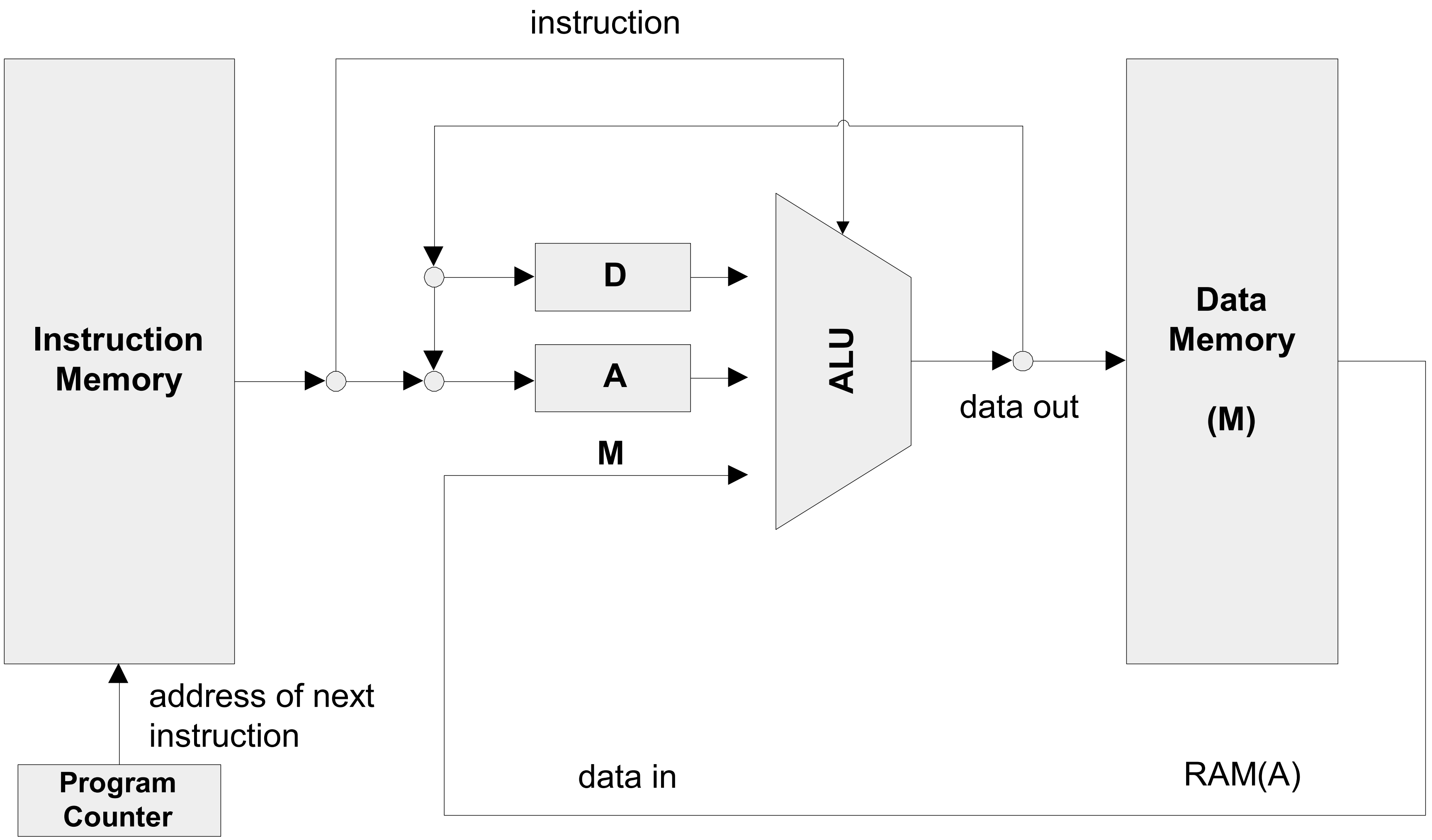
Assembly program

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

Executable

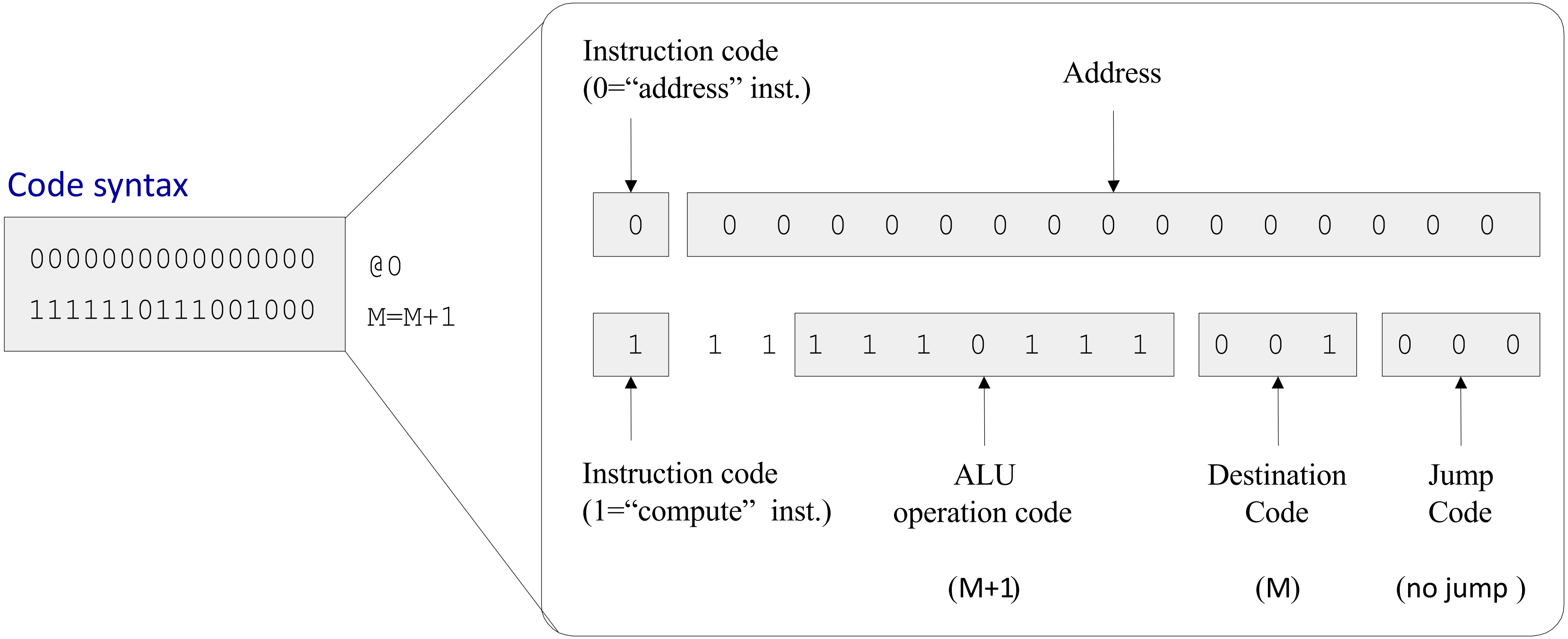
```
0000000000000000
1110110010001000
```

What do the instructions do?



The code directs elements of the processor in order to achieve results

Code semantics, as interpreted by the Hack hardware platform



Code syntax

```
0000000000000000    @0
1111110111001000    M=M+1
```

Instruction code
(0="address" inst.)

Address

0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

1	1 1	1 1 1 0 1 1 1	0 0 1	0 0 0

Instruction code
(1="compute" inst.)

ALU
operation code

(M+1)

Destination
Code

(M)

Jump
Code

(no jump)

Logic design

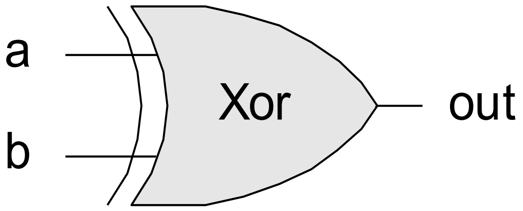
- Three types of logic we will be using:
  - Combinational logic – used for the ALU
  - Sequential logic – used for RAM
  - Gate logic – putting it all together to get a computer

What is gate logic?

- Our hardware is an inter-connected set of chips.
- Chips are built of simpler chips, down to the simplest structure of all – the elementary logic gate.
- Logic gates are hardware implementations of Boolean functions. This allows us to represent logical statements in computer form.
- Every chip and gate has:
  - An interface: Telling us what it does
  - An implementation: Telling us how it does it.

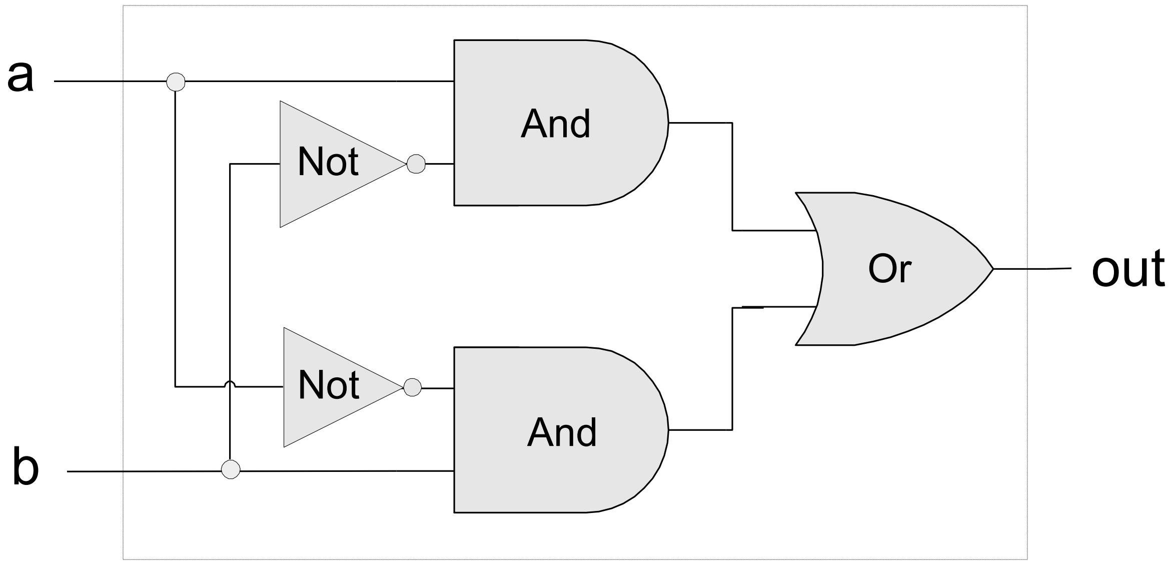
Example

Interface



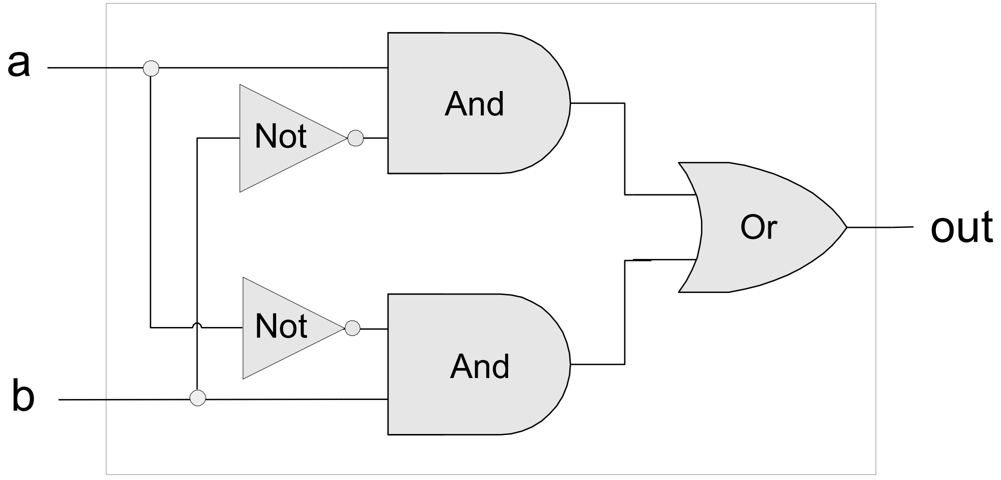
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Implementation



Building gates

- We won’t be building real gates, we’ll build them in simulation using a 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);
}
```

Summary

- We’re going to show you how software and hardware work together to build a computer system.
- Over the course, you will build parts of that system and get practice in combinational, sequential and gate logic, as well as learning how high level languages make things happen in real systems.
- You have a workshop this week on the tools you’ll need for the course.
- After this lecture read the first chapter of the text book.