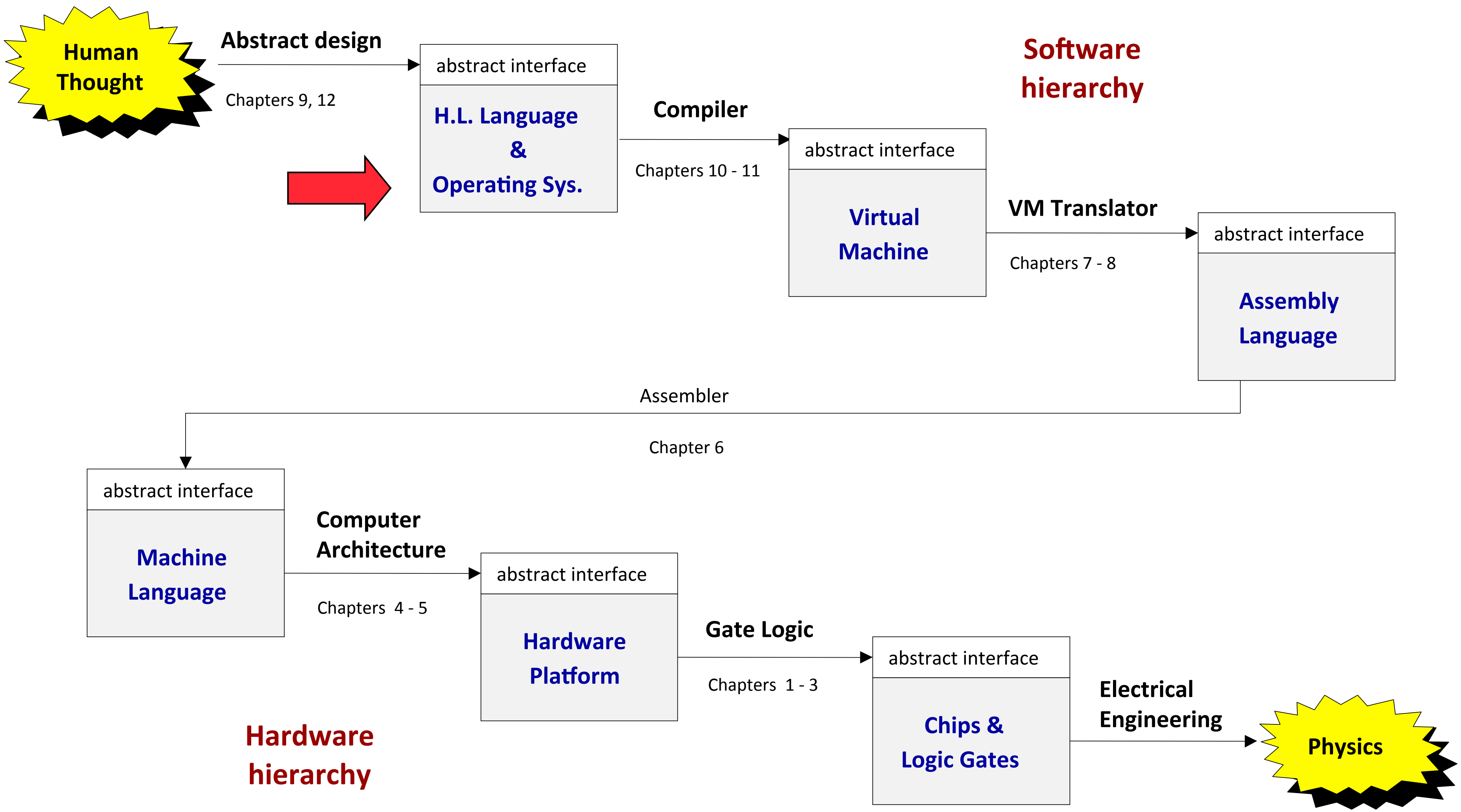
School of Computer Science

COMP SCI 2000 Computer Systems
Lecture 21

Where we are at:



Lecture Plan

- The Operating System
  - Basic libraries
  - Single user system
- More efficient integer arithmetic
  - Multiply
  - Square root
- Clocks
  - Signal propagation
  - Power wall
  - How to go faster

Jack revisited

```
/** Computes the average of a sequence of integers. */
class Main
{
  function void main()
  {
    var Array a;
    var int length;
    var int i, sum;

    let length = Keyboard.readInt("How many numbers? ");
    let a = Array.new(length); // Constructs the array
    let i = 0;

    while (i < length)
    {
      let a[i] = Keyboard.readInt("Enter the next number: ");
      let sum = sum + a[i];
      let i = i + 1;
    }

    do Output.printString("The average is: ");
    do Output.printInt(sum / length);
    do Output.println();
    return;
  }
}
```

Typical OS functions

Language extensions / standard library

- Mathematical operations
  (`abs`, `sqrt`, ...)
- Abstract data types
  (`String`, `Date`, ...)
- Output functions
  (`printChar`, `printString` ...)
- Input functions
  (`readChar`, `readLine` ...)
- Graphics functions
  (`drawPixel`, `drawCircle`, ...)
- And more ...

System-oriented services

- Memory management
  (objects, arrays, ...)
- I/O device drivers
- Mass storage
- File system
- Multi-tasking
- UI management (shell / windows)
- Security
- Communications
- And more ...

The Jack OS

- **`Math:`** Provides basic mathematical operations;
- **`String:`** Implements the **`String`** type and string-related operations;
- **`Array:`** Implements the **`Array`** type and array-related operations;
- **`Output:`** Handles text output to the screen;
- **`Screen:`** Handles graphic output to the screen;
- **`Keyboard:`** Handles user input from the keyboard;
- **`Memory:`** Handles memory operations;
- **`Sys:`** Provides some execution-related services.

A typical OS:

- Is modular and scalable
- Empowers programmers (language extensions)
- Empowers users (file system, GUI, …)
- Closes gaps between software and hardware
- Runs in “protected mode”
- Typically written in some high level language
- Typically grows gradually, assuming more and more functions
- Must be efficient.

Efficiency

We have to implement various operations on n-bit binary numbers (n = 16, 32, 64, …).

For example, consider *multiplication*

- Naïve algorithm: to multiply x * y: { for i = 1 … y do sum = sum + x }
  - Run-time is proportional to y
  - In a 64-bit system, y can be as large as 2^{64}.
  - Multiplications can take years to complete
- Algorithms that operate on n-bit inputs can be either:
  - Naïve: run-time is proportional to the *value* of the n-bit inputs
  - Good: run-time is proportional to n, *the input's size.*

Example I: multiplication

The "steps"

		1	0	1	1	=	1	1
			1	0	1	=		5
		1	0	1	1			
	0	0	0	0				
1	0	1	1					
1	1	0	1	1	1	=	5	5

The algorithm explained (first 4 of 16 iteration)

x:	0	0	0	1	0	1	1	
y:	0	0	0	0	1	0	1	j'th bit of y
	0	0	0	**1**	**0**	**1**	**1**	1
	0	0	**1**	**0**	**1**	**1**	0	0
	0	**1**	**0**	**1**	**1**	0	0	1
	1	**0**	**1**	**1**	0	0	0	0
$x \cdot y$:	0	1	1	0	1	1	1	sum

multiply(x, y):

// Where $x, y \geq 0$

$sum = 0$

$shiftedX = x$

for $j = 0 \ldots (n-1)$ do

  if (j-th bit of y) = 1 then

    $sum = sum + shiftedX$

  $shiftedX = shiftedX * 2$

- Run-time: proportional to n
- Can be implemented in SW or HW
- Division: similar idea.

Example II: square root

The square root function has two convenient properties:

- It's inverse function is computed easily
- Monotonically increasing

Functions that have these two properties can be computed by binary search:

sqrt(x):

// Compute the integer part of $y = \sqrt{x}$. Strategy:

// Find an integer y such that $y^2 \le x < (y+1)^2$ (for $0 \le x < 2^n$)

// By performing a binary search in the range $0 \ldots\ 2^{n/2} - 1$.

$y = 0$

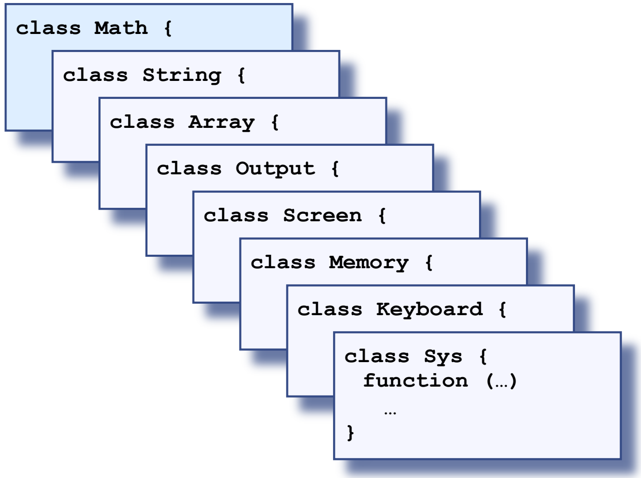
for $j = n/2 - 1 \ldots 0$ do

    if $(y + 2^j)^2 \le x$ then $y = y + 2^j$

return y

Number of loop iterations is bounded by n/2, thus the run-time is O(n).

Math operations (in the Jack OS)




```
class Math {

   function void init()

   function int abs(int x)

✓  function int multiply(int x, int y)

✓  function int divide(int x, int y)

   function int min(int x, int y)

   function int max(int x, int y)

✓  function int sqrt(int x)

}
```

The remaining functions are simple to implement.

Perspective

- What we presented can be described as a:
  - mini OS
  - Standard library
- Many classical OS functions are missing
- No separation between user mode and OS mode
- Some algorithms (e.g. multiplication and division) are standard
- Other algorithms (e.g. line- and circle-drawing) can be accelerated with special hardware
- And, by the way, we've just finished building the computer.

Clocks

- The Hack computer runs off a digital clock
  - One instruction is executed in each clock cycle
  - The entire machine uses the same clock
- What might limit how fast a digital computer can run ?
  - The Power Wall
  - Signal propagation

The Power Wall

- The power consumption of a processor
  - Capacitive Load x Voltage^2 x Frequency
- Challenges
  - Capacitive load reduction requires new manufacturing processes
  - Voltage reductions may be nearing their limits
  - Removing the generated heat can be expensive
  - We want lower power consumption

Signal Propagation

- How long must a clock cycle be ?
  - The output from every gate needs time to propagate to the next gate
  - Every gate needs time to respond to its inputs
  - The longest path through the machine must be able to complete
- Clock skew
  - The clock signal needs time to propagate to everywhere it is used
  - Careful routing of the clock signal is required

What Can Be Done ?

- Shorten the longest path
  - More efficient adders
    - Our 16-bit adder requires a signal to traverse 32 gates
    - Carry look ahead could reduce this to 5 gates
    - A 16-bit multiplier could be implemented with a 20 gate delay
  - Split the processor into smaller parts linked by registers
    - pipelining
- Use separate clocks for different components
  - Processor
  - Memory hierarchy
  - I/O devices

Some Final notes

- CS is a science
- What is science?
- Reductionism
- Life science: From Aristo (millions of rules) to Darwin (3 rules) to Watson and Crick (1 rule)
- Computer science: We *knew* in advance that we could build a computer from almost nothing. In this course we actually did it.
- Key lessons:
  - Elegance
  - Clarity
  - Simplicity
  - Playfulness.

a	b	Out
0	0	1
0	1	1
1	0	1
1	1	0