Summing up and looking forward

John Lapinskas, University of Bristol

Knowing what we don't know

Here are some things we haven't done:

- Use an HDL like Verilog to specify far more complex circuitry with less scope for input errors.

Knowing what we don't know

Here are some things we haven't done:

- Use an HDL like Verilog to specify far more complex circuitry with less scope for input errors.
- Learn a more advanced assembly language like MIPS and using it to build something in a low-power environment.

Knowing what we don't know

Here are some things we haven't done:

- Use an HDL like Verilog to specify far more complex circuitry with less scope for input errors.
- Learn a more advanced assembly language like MIPS and using it to build something in a low-power environment.
- Hardware capabilities: Advanced operations, interrupts, pipelining, caching.

Knowing what we don't know

Here are some things we haven't done:

- Use an HDL like Verilog to specify far more complex circuitry with less scope for input errors.
- Learn a more advanced assembly language like MIPS and using it to build something in a low-power environment.
- Hardware capabilities: Advanced operations, interrupts, pipelining, caching.
- An operating system with support for multiple processes.

Knowing what we don't know

Here are some things we haven't done:

- Use an HDL like Verilog to specify far more complex circuitry with less scope for input errors.
- Learn a more advanced assembly language like MIPS and using it to build something in a low-power environment.
- Hardware capabilities: Advanced operations, interrupts, pipelining, caching.
- An operating system with support for multiple processes.
- Better compilers: Type checking, program verification, code optimisation.

Knowing what we don't know

Here are some things we haven't done:

- Use an HDL like Verilog to specify far more complex circuitry with less scope for input errors.
- Learn a more advanced assembly language like MIPS and using it to build something in a low-power environment.
- Hardware capabilities: Advanced operations, interrupts, pipelining, caching.
- An operating system with support for multiple processes.
- Better compilers: Type checking, program verification, code optimisation.
- More language features: True object-oriented programming, concurrency.

Knowing what we don't know

Here are some things we haven't done:

- Use an HDL like Verilog to specify far more complex circuitry with less scope for input errors.
- Learn a more advanced assembly language like MIPS and using it to build something in a low-power environment.
- Hardware capabilities: Advanced operations, interrupts, pipelining, caching.
- An operating system with support for multiple processes.
- Better compilers: Type checking, program verification, code optimisation.
- More language features: True object-oriented programming, concurrency.
- Internet connectivity: Twitch plays Tetris?

Knowing what we don't know

Here are some things we haven't done:

- Use an HDL like Verilog to specify far more complex circuitry with less scope for input errors.
- Learn a more advanced assembly language like MIPS and using it to build something in a low-power environment.
- Hardware capabilities: Advanced operations, interrupts, pipelining, caching.
- An operating system with support for multiple processes.
- Better compilers: Type checking, program verification, code optimisation.
- More language features: True object-oriented programming, concurrency.
- Internet connectivity: Twitch plays Tetris?

Any of these could be fine subjects for individual projects over summer!

The programming languages research group is a good place to go for compiler-type projects, while the (systems) cybersecurity and HPC research groups are good places to go for hardware- or OS-type projects.

What we've done

But let's not lose sight of what we *have* done:

- Compiled programs in a C-like language (Jack) into a stack machine:
  - Using a grammar and recursion to turn complex expressions into long strings of postfix-form instructions.
  - Implementing user-defined types as pointer-based arrays.
  - Allocating and freeing memory on the heap with a non-leaky algorithm.
  - Managing variables in different scopes using symbol tables.
  - Rendering complex program flow statements like `while` loops into simple gotos.

What we've done

But let's not lose sight of what we *have* done:

- Compiled programs in a C-like language (Jack) into a stack machine:
- Compiled programs from a stack machine into assembly language:
  - Implementing the stack itself in raw assembly.
  - Implementing function calls with the stack.
  - Mapping virtual memory segments back to physical memory in both stack and heap.
  - Combining multiple files together into one to support libraries.

What we've done

But let's not lose sight of what we *have* done:

- Compiled programs in a C-like language (Jack) into a stack machine:
- Compiled programs from a stack machine into assembly language:
- Compiled programs from assembly language into native machine code:
  - Mapping labels to ROM addresses ready to be jumped to.
  - Greedily allocating variables to designated areas of memory.
  - Using memory-mapped I/O to write to a screen and read from a keyboard.
  - Understanding the instruction set itself and how it was designed.

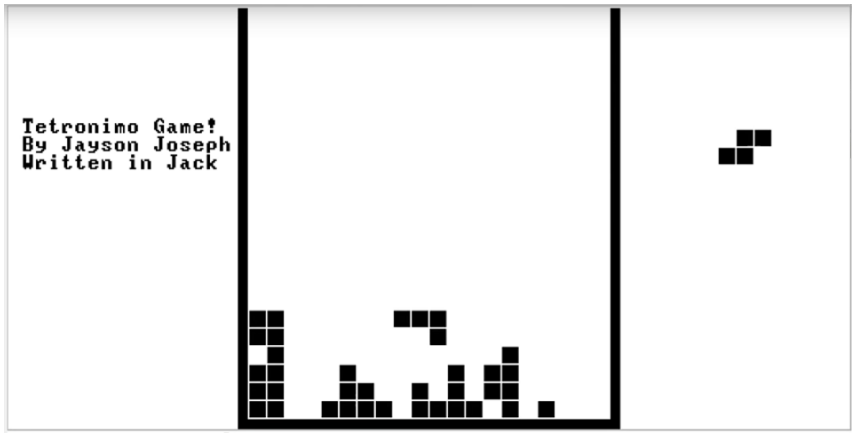
What we've done

But let's not lose sight of what we *have* done:

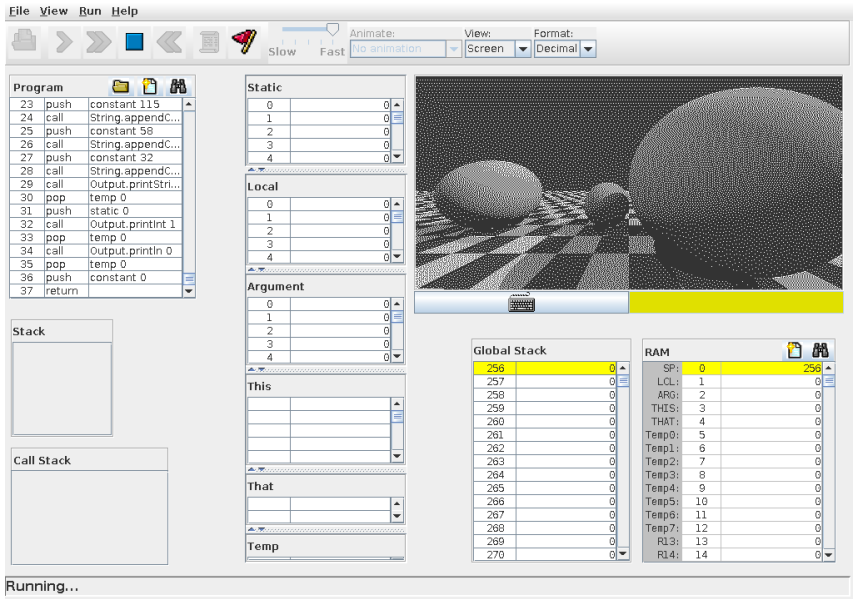
- Compiled programs in a C-like language (Jack) into a stack machine:
- Compiled programs from a stack machine into assembly language:
- Compiled programs from assembly language into native machine code:
- Built the computer running that machine code from scratch:
  - Building R-S latches into D flip-flops into registers and memory.
  - Routing instructions between components with multiplexers and demultiplexers.
  - Implementing complex arithmetic operations with simple gates.
- Coming all the way back down to the humble NAND gate with the help of truth tables and boolean algebra.

Hack and Nand2Tetris

Remember these from the intro talk?



Source: Jayson Joseph (here)



Source: Alex Quach (here)

You could make these now. *Truly* from scratch.

Congratulations on making it through, and good luck in the exam!