Chapter 5

Computer Architecture

These slides support chapter 5 of the book

*The Elements of Computing Systems*

By Noam Nisan and Shimon Schocken

MIT Press
Computer Architecture: lecture plan

- Von Neumann Architecture
  - Fetch-Execute Cycle
  - The Hack CPU
  - The Hack Computer
  - Project 5 Overview
Universality

Same **hardware** can run many different **software** programs

**Theory**

Alan Turing: Universal Turing Machine

**Practice**

John Von Neumann: Stored Program Computer
Computer architecture
Computer architecture
Computer architecture
Computer architecture:

- Memory
  - Program
  - Data

- CPU
  - Registers
  - ALU

- Control bus
- Data bus
Computer architecture
Computer architecture
Computer architecture

![Diagram of computer architecture with Memory and CPU sections connected by address and data buses.](image)
Computer architecture

- Memory
- CPU
  - Registers
  - ALU
- Address bus
- Data bus
Computer architecture
Computer architecture
Computer Architecture: lecture plan

- Von Neumann Architecture
- Fetch-Execute Cycle
  - The Hack CPU
  - The Hack Computer
  - Project 5 Overview
Basic CPU loop

Repeat:

- *Fetch* an instruction from the program memory
- *Execute* the instruction.
Fetching

• Put the location of the next instruction in the Memory address input
• Get the instruction code by reading the contents at that Memory location

- Default: PC++
- Jump: PC = address
Executing

• The instruction code specifies “what to do”
  - Which arithmetic or logical instruction to execute
  - Which memory address to access (for read / write)
  - If / where to jump
  - …

• Executing the instruction involves:
  - accessing registers
    and / or:
  - accessing the data memory.
Computer architecture

Memory

program

data

CPU

Registers

ALU

control bus

address bus

data bus
Fetch – Execute

- Program
- Data
- Instruction
- Address
- ALU
- Control bus
- Data address
- Data
- Instruction address
- Address bus
- Address bus

(data flows not shown, to minimize clutter)
If the Memory is one address space:
This scheme will not work:
• one Memory input
• one Memory output
Fetch – Execute clash

If the Memory is one address space:

This scheme will not work:

- one Memory input
- one Memory output
Solution: multiplex

- Memory output
  - data, when *executing*
  - instruction, when *fetching*

- Memory address input

- Memory
  - program
  - data

- mux
  - instruction address
  - data address
  - control bus
  - address bus
  - address bus

- Instruction, when fetching
- Data, when executing
Solution: multiplex, using an instruction register

Instruction register

Data, when executing

Fetch / execute bit

Load on fetch

Control bus

Address bus

Multiplex, using an instruction register
Solution: multiplex, using an instruction register

- **Memory**
  - Program
  - Data
- **ALU**
- **Instruction Register**
  - Load on fetch
  - Fetch / execute bit
  - Instruction
- **Mux**
  - Instruction address
  - Data address
- **Control Bus**
- **Address Bus**

When executing, data is fetched and loaded into the instruction register.
Solution: multiplex, using an instruction register

The diagram illustrates a multiplexing scheme to efficiently handle both instruction and data paths. The key components are:

- **Memory**: Stores the program and data.
- **Instruction Register**: Holds the currently executing instruction.
- **ALU**: Performs arithmetic and logic operations.
- **Multiplexer (mux)**: Selects between instruction and data inputs.
- **Control Bus**: Sends control signals to the ALU and memory.
- **Address Bus**: Provides addresses for memory and ALU operations.

The diagram shows how instructions and data are fetched, loaded into the instruction register, and then passed to the ALU for execution. The multiplexer ensures that the right path is selected based on the control signals. The diagram includes notes on the flow of data through the bus to minimize clutter.
Simpler solution: separate memory units

Variant of von Neumann Architecture (used by the Hack computer):

Two physically separate memory units:

- Instruction memory
- Data memory

Each can be addressed and manipulated separately, and simultaneously

• Advantage:
  - Complication avoided

• Disadvantage:
  - Two memory chips instead of one
  - The size of the two chips is fixed.

Sometimes called “Harvard Architecture”
Computer Architecture: lecture plan

- Von Neumann Architecture
- Fetch-Execute Cycle
- The Hack CPU
  - The Hack Computer
  - Project 5 Overview
Hack computer

Computer System

input

instruction memory

instruction

address of next instruction

ALU

D register

A register

PC

data memory

data out

data in

output
Hack CPU: A 16-bit processor, designed to:

- Execute the current instruction: dataOut = instruction(dataIn)
- Figure out which instruction to execute next.
Hack CPU Interface

data in

\{\begin{align*}
\text{inM} & \rightarrow \text{outM} \\
\text{instruction} & \rightarrow \text{writeM} \\
\text{reset} & \rightarrow \text{pc}
\end{align*}\}

data out

\{\begin{align*}
\text{outM} & \rightarrow \text{data out} \\
\text{writeM} & \rightarrow \text{address of next instruction} \\
\text{pc} & \rightarrow \text{address of next instruction}
\end{align*}\}
Hack CPU Implementation
Hack CPU Implementation

(each "c" symbol represents a control bit)
Hack CPU Implementation

Disclaimer:
• In what follows we don’t explain all the details of the CPU implementation, intentionally
• We give enough information to let you figure out the implementation on your own, when you will actually build the CPU in project 5.
CPU operation
CPU operation: instruction handling
CPU operation: instruction handling

A-instruction

instruction

@5 0000000000000101
CPU operation: handling A-instructions

CPU handling of an A-instruction:

- Decodes the instruction into:
  - op-code
  - 15-bit value
- Stores the value in the A-register
- Outputs the value (not shown in this diagram).
CPU operation: instruction handling

D = D + 1; JMP 1110011111010111

C-instruction
CPU operation: handling C-instructions

CPU handling of a C-instruction:

- Decodes the instruction bits into:
  - Op-code
  - ALU control bits
  - Destination load bits
  - Jump bits

- Routes these bits to their chip-part destinations
- The chip-parts (most notably, the ALU) execute the instruction.
CPU operation: handling c-instructions

ALU data inputs:
- Input 1: from the D-register
- Input 2: from either:
  - A-register, or
  - data memory

ALU control inputs:
- control bits
  (from the instruction)
CPU operation: handling c-instructions

ALU data output:
- Result of ALU calculation
- Fed simultaneously to: D-register, A-register, data memory
- Which destination *actually* commits to the ALU output is determined by the instruction’s destination bits.
CPU operation: handling c-instructions

ALU control outputs:
- is the output negative?
- is the output zero?
CPU operation: control
CPU operation: control

- The computer is loaded with some program;
- Pushing **reset** causes the program to start running.
CPU operation: control
CPU operation: control

PC operation (abstraction)
Emits the address of the next instruction:

- **restart:** $PC = 0$
- **no jump:** $PC++$
- **goto:** $PC = A$
- **conditional goto:** if $(condition)$ $PC = A$ else $PC++$
CPU operation: control

PC operation (implementation)
if (reset==1) PC=0
else
  // in the course of handling the current instruction:
  load = f (jump bits, ALU control outputs)
  if (load == 1) PC=A  // jump
  else PC++  // next instruction
Hack CPU Implementation

That’s It!
Computer Architecture: lecture plan

- Von Neumann Architecture
- Fetch-Execute Cycle
- The Hack CPU
- The Hack Computer
- Project 5 Overview
**Hack Computer**

**Abstraction:**
A computer capable of running programs written in the Hack machine language

**Implementation:**
Built from the Hack chip-set.
Hack CPU
Hack CPU

CPU abstraction:
Executes a Hack instruction and figures out which instruction to execute next

CPU Implementation:
Discussed before.
Hack Computer
Memory
Memory: abstraction

- Address 0 to 16383: data memory
- Address 16384 to 24575: screen memory map
- Address 24576: keyboard memory map
Memory: implementation

- Address 0 to 16383: data memory
- Address 16384 to 24575: screen memory map
- Address 24576: keyboard memory map
The Hack RAM is realized by the RAM16K chip implemented in project 3.
Screen

![Diagram of memory and screen addressing]

- **Memory**
  - **RAM** (16K)
  - **Screen** (8K memory map)
  - **Keyboard**

- **Addressing**
  - Input: 16 bits
  - Output: 16 bits

- **Screen Output**
  - Label: "Hello, world"
Screen memory map

To set pixel \((row, col)\) on/off:

1. \(word = \text{RAM}[16384 + 32 \times row + col/16]\)
2. Set the \((col \% 16)th\) bit of \(word\) to 0 or 1
3. \(\text{RAM}[i] = word\)
• The Hack screen is realized by a built-in chip named Screen

• Screen: a regular RAM + display output side-effect.
Memory Keyboard

Keyboard

Memory

0

RAM
(16K)

16,383

16,384

Screen
(8K memory map)

24,575

24,576

in 16

address 15

out 16
Keyboard memory map

The Keyboard chip emits the scan-code of the currently pressed key, or $\emptyset$ if no key is pressed.

Scan-code of 'k' = 75
The Hack character set

<table>
<thead>
<tr>
<th>key</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(space)</td>
<td>32</td>
</tr>
<tr>
<td>!</td>
<td>33</td>
</tr>
<tr>
<td>“</td>
<td>34</td>
</tr>
<tr>
<td>#</td>
<td>35</td>
</tr>
<tr>
<td>$</td>
<td>36</td>
</tr>
<tr>
<td>%</td>
<td>37</td>
</tr>
<tr>
<td>&amp;</td>
<td>38</td>
</tr>
<tr>
<td>‘</td>
<td>39</td>
</tr>
<tr>
<td>(</td>
<td>40</td>
</tr>
<tr>
<td>)</td>
<td>41</td>
</tr>
<tr>
<td>*</td>
<td>42</td>
</tr>
<tr>
<td>+</td>
<td>43</td>
</tr>
<tr>
<td>,</td>
<td>44</td>
</tr>
<tr>
<td>-</td>
<td>45</td>
</tr>
<tr>
<td>.</td>
<td>46</td>
</tr>
<tr>
<td>/</td>
<td>47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>1</td>
<td>49</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>57</td>
</tr>
<tr>
<td>:</td>
<td>58</td>
</tr>
<tr>
<td>;</td>
<td>59</td>
</tr>
<tr>
<td>&lt;</td>
<td>60</td>
</tr>
<tr>
<td>=</td>
<td>61</td>
</tr>
<tr>
<td>&gt;</td>
<td>62</td>
</tr>
<tr>
<td>?</td>
<td>63</td>
</tr>
<tr>
<td>@</td>
<td>64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>65</td>
</tr>
<tr>
<td>B</td>
<td>66</td>
</tr>
<tr>
<td>C</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Z</td>
<td>90</td>
</tr>
<tr>
<td>[</td>
<td>91</td>
</tr>
<tr>
<td>]</td>
<td>93</td>
</tr>
<tr>
<td>^</td>
<td>94</td>
</tr>
<tr>
<td>_</td>
<td>95</td>
</tr>
<tr>
<td>`</td>
<td>96</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>97</td>
</tr>
<tr>
<td>b</td>
<td>98</td>
</tr>
<tr>
<td>c</td>
<td>99</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>z</td>
<td>122</td>
</tr>
<tr>
<td>{</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>125</td>
</tr>
<tr>
<td>~</td>
<td>126</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>newline</td>
<td>128</td>
</tr>
<tr>
<td>backspace</td>
<td>129</td>
</tr>
<tr>
<td>left arrow</td>
<td>130</td>
</tr>
<tr>
<td>up arrow</td>
<td>131</td>
</tr>
<tr>
<td>right arrow</td>
<td>132</td>
</tr>
<tr>
<td>down arrow</td>
<td>133</td>
</tr>
<tr>
<td>home</td>
<td>134</td>
</tr>
<tr>
<td>end</td>
<td>135</td>
</tr>
<tr>
<td>Page up</td>
<td>136</td>
</tr>
<tr>
<td>Page down</td>
<td>137</td>
</tr>
<tr>
<td>insert</td>
<td>138</td>
</tr>
<tr>
<td>delete</td>
<td>139</td>
</tr>
<tr>
<td>esc</td>
<td>140</td>
</tr>
<tr>
<td>f1</td>
<td>141</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>f12</td>
<td>152</td>
</tr>
</tbody>
</table>
Keyboard

- Realized by a built-in chip named Keyboard
- Keyboard: A read-only 16-bit register + a keyboard input side-effect.
Memory implementation

Implementation outline:

- Uses the three chip-parts RAM16K, Screen, and Keyboard (as just described)
- Routes the address input to the correct address input of the relevant chip-part.
Hack Computer

The diagram illustrates the components of a Hack Computer:

- **Instruction Memory**: Stores the program's instructions.
- **CPU**: Executes the instructions and processes data.
- **Data Memory**: Stores data that is processed by the CPU.

Connections:
- **Address**: Sent from the instruction memory to the CPU.
- **Instruction**: Sent from the instruction memory to the CPU.
- **Data In**: Sent from the CPU to the data memory.
- **Data Out**: Sent from the data memory to the CPU.

Reset button is shown in the diagram for initialization purposes.

The diagram also shows the output of the computer, displaying "Hello, world."
Instruction memory

To run a program on the Hack computer:
- Load the program into the Instruction Memory
- Press “reset”
- The program starts running.

Load a program into the Instruction Memory? How?
Loading a program into the Instruction Memory:

- Hardware implementation: plug-and-play ROM chips  
  (each comes pre-loaded with a program’s code)
- Hardware simulation: programs are stored in text files;  
  The simulator’s software features a load-program service.
The Hack Instruction Memory is realized by a built-in chip named ROM32K.

ROM32K: a read-only, 16-bit, 32K RAM chip + program loading side-effect.
Hack Computer implementation
Hack Computer implementation

That’s it!
“We ascribe beauty to that which is simple; which has no superfluous parts; which exactly answers its end; which stands related to all things; which is the mean of many extremes.”

-- Ralph Waldo Emerson
Computer Architecture: lecture plan

- Von Neumann Architecture
- Fetch-Execute Cycle
- The Hack CPU
- The Hack Computer
- Project 5 Overview
Hardware organization: a hierarchy of chip parts

- computer
- memory
- RAM units
- registers
- elementary logic gates
- CPU
- PC
- ALU
- adder
Hardware projects

- Memory
- CPU
- RAM chips
- Registers
- Elementary logic gates
- Adder
- ALU
- PC

Diagram shows the components of a computer system, with relations between memory, CPU, RAM chips, registers, elementary logic gates, adder, and ALU.
Project 5: building the Hack Computer

- Computer
  - Memory
    - RAM chips
  - Registers
  - Elementary logic gates
  - CPU
    - PC
    - ALU
  - Adder
Abstraction

Hack Computer
Implementation

![Diagram of computer architecture with components such as ROM32K, CPU, Memory, and connections labeled as inM, writeM, outM, addressM, pc, and reset. There is also an output that says "Hello, world." ]
CPU Abstraction

Hack CPU

instruction

inM

reset

outM

writeM

addressM

PC
Implementation tips:

- Chip-parts: Mux16, ARegister, DRegister, PC, ALU, ...

- Control: use HDL subscripting to parse and route the instruction bits to the control bits of the relevant chip-parts.
CPU Implementation

/**
 * The Central Processing unit (CPU).
 * Consists of an ALU and a set of registers,
 * designed to fetch and execute instructions
 * written in the Hack machine language.
 */

CHIP CPU {

    IN
    inM[16], // value of M = RAM[A]
    instruction[16], // Instruction for execution
    reset; // Signals whether to re-start the current program
            // (reset == 1) or continue executing the current
            // program (reset == 0).

    OUT
    outM[16] // value to write into M = RAM[A]
    writeM, // Write into M?
    addressM[15], // RAM address (of M)
    pc[15]; // ROM address (of next instruction)

    PARTS:
    // Put you code here:
}

CPU.hdl
Hack Computer implementation
Memory implementation
Memory implementation

Implementation tips:

• Use the three chip-parts: RAM16K, Screen, and Keyboard

• Route the address input to the correct address input of the relevant chip-part.
Hack Computer implementation

Implementation tip:
Use the built-in ROM32K chip.
Hack Computer implementation

- ROM32K
- CPU
- Memory
- reset

- inM
- writeM
- outM
- addressM
- pc

- Hello, world
Hack Computer implementation

CHIP Computer {
    IN reset;

    PARTS:
    // Put your code here.
}
Project 5: Computer Architecture

Background
In previous projects we've built the computer's basic processing and storage devices (ALU and RAM, respectively). In this project we will put everything together, yielding the complete Hack Hardware Platform. The result will be a general-purpose computer that can run any program that you fancy.

Objective
Complete the construction of the Hack CPU and computer platform, leading up to the top-most Computer chip.

Chips

<table>
<thead>
<tr>
<th>Chip (HDL)</th>
<th>Description</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory.hdl</td>
<td>Entire RAM address space</td>
<td>Test this chip using Memory.tst and Memory.cmp</td>
</tr>
<tr>
<td>CPU.hdl</td>
<td>The Hack CPU</td>
<td>Recommended test files: CPU.tst and CPU.cmp.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alternative test files (less thorough but do not require using the built-in DRegister): CPU-external.tst and CPU-external.cmp.</td>
</tr>
<tr>
<td>Computer.hdl</td>
<td>The platform's top-most chip</td>
<td>Test by running some Hack programs on the constructed chip. See more instructions below.</td>
</tr>
</tbody>
</table>

All the necessary project 5 files are available in: nand2tetris/projects/05
More resources

• HDL Survival Guide
• Hardware Simulator Tutorial
• nand2tetris Q&A forum

All available in: www.nand2tetris.org
Best practice advice

• Try to implement the chips in the given order
• Strive to use as few chip-parts as possible

• You will have to use chips that you’ve implemented in previous projects

• The best practice is to use their built-in versions.
Computer Architecture: lecture plan

✓ Von Neumann Architecture

✓ Fetch-Execute Cycle

✓ The Hack CPU

✓ The Hack Computer

✓ Project 5 Overview
Chapter 5

Computer Architecture

These slides support chapter 5 of the book

*The Elements of Computing Systems*

By Noam Nisan and Shimon Schocken

MIT Press