Modern Computer Organization: Abstractions at Several Levels

- levels of computer architecture
- the Java Virtual Machine (level 5 - mainly)
- memory hierarchy and caches (level 1)
  - issues, and details
- the digital logic level (level 0)
- revision from lectures D1–C3

### Levels of Computer Architecture

- **Level 5: Problem-oriented language (e.g. Java, C?)**
  - compilation
- **Level 4: Assembly language**
  - translation
- **Level 3: Operating system machine**
  - partial interpretation
- **Level 2: Instruction set architecture**
  - interpretation (microprogram) or direct execution
  - e.g. registers, (virtual) memory access
- **Level 1: Microarchitecture**
  - hardware
  - e.g. datapaths, buses, ALUs, ‘hidden’ registers (CI, MAR),
- **Level 0: Digital logic**
  - e.g. circuits (wires and gates)

### High-Level Language Level: the Java Virtual Machine (JVM)

Ref: [Null&Lobur, sect 3.5]  
Fig 8.11, Fig 8.13
- JVM is a stack machine interpreter for Java bytecode (.class files)
- individual bytecode instructions are represented by numbers stored in a single byte
- the Java code sequence `i = i + 1;` (where int `i`;) might translate into the bytecode sequence:

<table>
<thead>
<tr>
<th>mnemonic</th>
<th>bytecode</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>iload_0</td>
<td>1A</td>
<td>push local variable 0 (i) onto stack</td>
</tr>
<tr>
<td>iconst_1</td>
<td>04</td>
<td>push constant 0 onto stack</td>
</tr>
<tr>
<td>iadd</td>
<td>60</td>
<td>pop the top two entries then push their sum on the stack</td>
</tr>
<tr>
<td>istore_0</td>
<td>3A</td>
<td>pop the stack and store value in local variable 0</td>
</tr>
</tbody>
</table>

- a JVM interprets these, and can produce the same effects on many different physical computers
- this stage corresponds to Level 2, but on a virtual (software) level
- OS level is abstracted via I/O-related methods (accessed via the JVM native methods, which accesses the C system calls)
- Q: why is it useful to design Java this way? why use a stack machine?

### The Microarchitectural Level Example: the Memory Hierarchy

Ref: [Null&Lobur, sect 6.3-6.4], [O’H&Bryant, sect 6.2-6.4]
- in a computer system, there is a memory hierarchy, because:
  - many different mediums for the storage of data
  - generally, there is a trade-off between speed and capacity
  - fast memories tend to be small; large memories tend to be slow

<table>
<thead>
<tr>
<th>medium</th>
<th>access time</th>
<th>typical size</th>
</tr>
</thead>
<tbody>
<tr>
<td>registers</td>
<td>~1 ns</td>
<td>&lt; 1 KB</td>
</tr>
<tr>
<td>cache mem.</td>
<td>~ 20 ns</td>
<td>&lt; 2 MB</td>
</tr>
<tr>
<td>main mem.</td>
<td>~ 100 ns</td>
<td>&lt; 2 GB</td>
</tr>
<tr>
<td>disk</td>
<td>~ 10^7 ns</td>
<td>&gt; 10 GB</td>
</tr>
</tbody>
</table>

- idea of cache memory: store recently accessed data from main memory in a small, faster memory (closer to / inside the CPU)
- Q: why might this be useful?
Cache Memory: Issues

- [O’H&Bryant, fig 6.22]
- issue: cache must store the memory address (as a ‘tag’) of the data, as well as the data itself!
- issue: (considering the cache is to be organized as an array) which cache entry (‘index’) holds the data for address X (if any)?
- idea (direct-mapped cache): for a cache of size \( C' = 2^c \) entries, all addresses with the same \( a_1 \) are mapped to the same entry (does all of X need be in the tag?)

\[
X = \begin{bmatrix}
31 & c & c-1 & 0
\end{bmatrix}
\]

The Digital Logic Level: Gates

- Ref: [Null&Lobur, sect 2.3–2.4], [Tanenbaum, ch 3]
- digital logic is lowest level of computer organization!
- digital circuit elements (including 1-bit memory cells) are made from gates (in turn, from 1 or more transistors or switches, [Tanenbaum, fig 3.1])
- [Null&Lobur, figs 3.1-2]: AND and OR gates manipulate voltage levels (0- low, 1 - high) according to truth tables (Boolean logic)

The Digital Logic Level: Half Adders

- from gates, higher-level components such as a half-adder are constructed ([Null&Lobur, figs 3.10,3.11]: Sum = X XOR Y, Carry = X AND Y)
- this can be extended to a full adder ([Null&Lobur, fig 3.13]) (truth table, circuit)
  - how could this be used to implement an integer add circuit?
  - bits can be stored in components with feedback circuits (with clocked inputs)
  - e.g. SR latch, [Null&Lobur, fig 3.21]
- discussion point: why is binary the representation used in computers?

Revision from Lectures D1–D3, C1–C3

- winners from Minute Paper 1 are:
  - big and little endian (D2)
  - floating point issues
  - Q: 32-bit IEEE floating point has 8 bits for the exponent and 23 bits for the mantissa. If this were changed to 10 bits for the exponent and 21 bits for the mantissa:
    - (a) it would make no real difference
    - (b) you would have greater range but less precision
    - (c) you would have greater precision but less range
  - conversions
  - instruction sets: RISC vs CISC
  - pointers in C
  - dynamically allocated arrays