Module V:
Minds and Machines

Part 2: Inside the Machine

Representing Programs Digitally

- programs are examples of text: letters, digits, punctuation marks, spaces and other symbols (e.g. +, tab, new-line)
- like all other data, text is stored in bits (0/1, on/off) in computer memory
- how might you represent text using bits?

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Extended ASCII representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>01100001</td>
</tr>
<tr>
<td>b</td>
<td>01100010</td>
</tr>
<tr>
<td>c</td>
<td>01100011</td>
</tr>
<tr>
<td>d</td>
<td>01100100</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>00110101</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(</td>
<td>00101000</td>
</tr>
</tbody>
</table>
Representing Programs Digitally

**extended ASCII representation of text**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Extended ASCII representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>01100001</td>
</tr>
<tr>
<td>b</td>
<td>01100010</td>
</tr>
<tr>
<td>c</td>
<td>01100011</td>
</tr>
<tr>
<td>d</td>
<td>01100100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Text</th>
<th>Extended ASCII representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>bad</td>
<td>01100010 01100011 01100100</td>
</tr>
<tr>
<td></td>
<td>01100011 01100001 01100010</td>
</tr>
</tbody>
</table>

Representing Programs Digitally

- high-level programs are represented using extended ASCII
- extended ASCII is an example of a code which maps symbols over one alphabet to symbols over another
- codes are used in many non-computing contexts—can you think of examples?

Representing Programs Digitally

- high-level languages are English-language-like in appearance (e.g., Scratch, Python, JavaScript)
- programs written in high-level languages are translated into programs in low-level (simpler) machine language before they can be executed
- an interpreter or compiler is a program that does the translation

Representing Programs Digitally

- high-level languages are designed to support humans in specifying processes precisely
- low-level machine languages are designed to facilitate fast execution by computer hardware
Machine Language

Machine language is linked to machine architecture.

- Memory or RAM (random access memory): where program instructions and associated data are stored when running on the computer.
- I/O ports: communication with peripherals.
- Processor or CPU: where program instructions are executed; contains ALU (arithmetic and logic unit) and control unit.

Machine architecture abstracted.

Machine Language

Machine architecture: memory organization (32 bits).

- Memory is a giant array of words:
  - Each word stores 4 bytes.
  - Each byte stores 8 bits.
  - The location of each byte has an address.
- Memory stores both data and machine language instructions, each encoded using 32 bits.

Machine Language

Machine architecture: processor organization.

- Data is stored in registers.
- The processor (CPU) contains an arithmetic and logic unit.
**Machine Language**

*machine language instruction examples*

- **add, subtract, multiply** data stored in the processor (i.e., in registers)
- **load** data from memory to register
- **store** data to memory from register
- **branch** and **jump** instructions
- (instructions for communication with IO devices)

---

**Machine Language: Clicker Question**

*machine language instruction examples*

The diagram indicates values before instruction **add 3, 4, 2** is executed. What value is in register 2 after?

A. 7  
B. 9  
C. 14  
D. 16

---

**Machine Language**

*machine language instruction examples*

**add 3, 4, 2:**
add the contents of registers 3 and 4 and store the answer in register 2

This **does not** mean add the numbers 3, 4 and 2

---

**Machine Language: Clicker Question**

*machine language instruction examples*

The diagram indicates values before instruction **load 3, 7001** is executed. What value is in register 3 after?

A. 4  
B. 5  
C. 7  
D. 8

---
Machine Language

machine language instruction examples

- to explain jmp and branch, we need to introduce the program counter
- machine language instructions are stored sequentially in memory
- the program counter contains the address of the instruction which is currently being executed

Machine Language: Clicker Question

machine language instruction examples

The diagram indicates values before the first instruction is executed. What value is in the program counter after?

A. 1000  
B. 1001  
C. 1004  
D. 7000

Machine Language: Clicker Question

What value is in the program counter after the instruction referred to by the program counter is executed?

A. 1000  
B. 1001  
C. 1004  
D. 7000
Machine Language

high-level language to machine language

if \((x < y)\) 
{ \(z = y + x;\) } 
else 
{ \(z = y - x;\) }

- the translator assigns each variable a place in memory

```
7000

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
</tbody>
</table>
```

Machine Language: Summary

- machine language instructions strongly reflect machine architecture, referring to specific registers and memory locations
- machine language instructions are simple and can be stored in one word of memory (32 bits)
Fetch/Decode/Execute Cycle

1. **fetch** the instruction specified by program counter from memory
2. **decode** the instruction’s parts, particularly the type of operation and its arguments (register numbers, memory addresses, etc.)
3. **execute** the operation (which may send a value to registers, memory, or the program counter)
4. update the program counter to prepare for the next instruction

Fetch/Decode/Execute Cycle

**initial configuration**

- Memory:
  - Instruction memory:
    - 1000: load 3, 7000
    - 1004: load 4, 7001
    - 1008: add 3, 4, 3
    - 1012: store 3, 7002
    - 1016: jmp 4
  - Data memory:
    - 7000: 7 8 0

Fetch/Decode/Execute Cycle

**after first instruction**

- Memory:
  - Instruction memory:
    - 1000: load 3, 7000
    - 1004: load 4, 7001
    - 1008: add 3, 4, 3
    - 1012: store 3, 7002
    - 1016: jmp 4
  - Data memory:
    - 7000: 7 8 0
Fetch/Decode/Execute Cycle

after second instruction

<table>
<thead>
<tr>
<th>Memory</th>
<th>Processor (CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>1008</td>
</tr>
<tr>
<td>Data Memory</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>1012</td>
</tr>
<tr>
<td>Data Memory</td>
<td></td>
</tr>
</tbody>
</table>

Fetch/Decode/Execute Cycle

after third instruction

<table>
<thead>
<tr>
<th>Memory</th>
<th>Processor (CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>1016</td>
</tr>
<tr>
<td>Data Memory</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>1036</td>
</tr>
<tr>
<td>Data Memory</td>
<td></td>
</tr>
</tbody>
</table>

Fetch/Decode/Execute Cycle

after fourth instruction

<table>
<thead>
<tr>
<th>Memory</th>
<th>Processor (CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>1004</td>
</tr>
<tr>
<td>Data Memory</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>1008</td>
</tr>
<tr>
<td>Data Memory</td>
<td></td>
</tr>
</tbody>
</table>

Fetch/Decode/Execute Cycle

after fifth instruction

<table>
<thead>
<tr>
<th>Memory</th>
<th>Processor (CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>1004</td>
</tr>
<tr>
<td>Data Memory</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>1008</td>
</tr>
<tr>
<td>Data Memory</td>
<td></td>
</tr>
<tr>
<td>Fetch/Decode/Execute Cycle</td>
<td>Fetch/Decode/Execute Cycle</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>clocks</td>
<td>clocks: “remember your nanoseconds”</td>
</tr>
</tbody>
</table>

- A clock controls the timing of each step in the instruction cycle.
- When it is time for a signal to be sent, the wire containing that signal is *enabled*.
- At other times, the signal is *disabled*.
- Modern processors may have clock speeds of more than 1 gigahertz, which means 1 cycle per nanosecond, or $10^9$ instructions per second.

“The ‘nanoseconds’ that Grace Hopper handed out were lengths of wire, cut to not quite 12 inches in length, equal to the distance traveled by an electron along the wire in the space of a nanosecond - one billionth of a second. In teaching efficient programming methods, Admiral Hopper wanted to make sure her students would not waste nanoseconds.”

- From “Tribute to Grace Murray Hopper” by Merry Maisal
  
  http://www.sdsc.edu/Hopper/GHC_INFO/hopper.html