18. von Neumann Machines

1. Perspective
2. A note of caution
3. Practical implications
4. Simulation

TOY vs. your laptop

Two different computing machines
- Both implement basic data types, conditionals, loops, and other low-level constructs.
- Both can have arrays, functions, libraries, and other high-level constructs.
- Both have infinite input and output streams.

Q. Is 256 words enough to do anything useful?
A. Yes! (Stay tuned.)

OK, we definitely want a faster version with more memory when we can afford it...

Is 4096 bits of memory enough to do anything useful?

Core memory from the Apollo Guidance Computer, 1966-1975
Is thousands of bits of memory enough to do anything useful?

LINC computer, MIT
12x2048 = 24,576 bits of memory
Used for many biomedical and other experiments

Wes Clark, 1963

Doug Clark and his father Wes, 2013

Is 4096 bits of main memory enough to do anything useful?

Contents of memory, registers, and PC at a particular time
• Provide a record of what a program has done.
• Completely determines what the machine will do.

Total number of bits in the state of the machine
• 255 x 16 (memory)
• 15 x 16 (registers)
• 8 (PC)

Total number of different states: $2^{253} \times 10^{1302}$ (!!!)

Total number of different states that could be observed if every electron in the universe had a supercomputer examining states for its entire lifetime: $< 10^{100}$.

Bottom line: We will never know what a machine with 4096 bits of main memory can do.

An early computer

ENIAC. Electronic Numerical Integrator and Calculator
• First widely-known general-purpose electronic computer.
• Conditional jumps, programmable, but no memory.
• Programming: Change switches and cable connections.
• Data: Enter numbers using punch cards.

ENIAC 1946

John von Neumann 1903–1957

A famous memo

First Draft of a report on the EDVAC, 1945
• Written by John von Neumann, Princeton mathematician
• EDVAC: second computer proposed by Eckert and Mauchly.
• Memo written on a train trip to Los Alamos.
• A brilliant summation of the stored-program concept.
• Influenced by theories of Alan Turing.
• Has influenced the design of every computer since.

Who invented the stored-program computer?
• Fascinating controversy.
• Eckert-Mauchly discussed the idea before von Neumann arrived on the scene.
• Goldstine circulated von Neumann’s first draft because of intense interest in the idea.
• Public disclosure prevented EDVAC design from being patented.
• von Neumann never took credit for the idea, but never gave credit to others, either.
Another early computer

EDSAC. Electronic Delay Storage Automatic Calculator

- Another stored-program computer (just after EDVAC).
- Data and instructions encoded in binary.
- Could load programs, not just data, into memory.
- Could change program without rewiring.

Implications

Stored-program (von Neumann) architecture is the basis of nearly all computers since the 1950s.

Practical implications

- Can load programs, not just data, into memory (download apps).
- Can write programs that produce programs as output (compilers).
- Can write programs that take programs as input (simulators).

Profound implications (see theory lectures)

- TOY can solve any problem that any other computer can solve (!)
- Some problems cannot be solved by any computer at all (!)
Arrays

To implement an array
• Keep items in an array contiguous starting at memory address a.
• Access a[i] at M[a+i].

To access an array element, use **indirection**
• Keep array address in a register.
• Add index
• Indirect load/store uses contents of a register.

Example: Indirect store

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 7A80</td>
<td>Load the address 80 into R[A]</td>
</tr>
<tr>
<td>13 7900</td>
<td>Set R[9] to 0 i is the index</td>
</tr>
<tr>
<td>16 1CA9</td>
<td>M[R[C]] = R[A] + R[9] compute address of a[i]</td>
</tr>
<tr>
<td>17 BD0C</td>
<td>M[R[C]] = R[D] a[i] = d</td>
</tr>
</tbody>
</table>

Array of length 11

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0001</td>
</tr>
<tr>
<td>3</td>
<td>0002</td>
</tr>
<tr>
<td>4</td>
<td>0003</td>
</tr>
<tr>
<td>5</td>
<td>0005</td>
</tr>
<tr>
<td>6</td>
<td>0008</td>
</tr>
<tr>
<td>7</td>
<td>0015</td>
</tr>
<tr>
<td>8</td>
<td>0022</td>
</tr>
<tr>
<td>9</td>
<td>0037</td>
</tr>
</tbody>
</table>

opcode | instruction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>load address</td>
</tr>
<tr>
<td>A</td>
<td>load indirect</td>
</tr>
<tr>
<td>B</td>
<td>store indirect</td>
</tr>
</tbody>
</table>

Arrays example: Read an array from standard input

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 7101</td>
<td>R[1] = 1</td>
</tr>
<tr>
<td>11 8BF8</td>
<td>R[B] = stdin</td>
</tr>
<tr>
<td>12 7A80</td>
<td>R[A] = 80</td>
</tr>
<tr>
<td>13 7900</td>
<td>R[9] = 0</td>
</tr>
<tr>
<td>14 22B9</td>
<td>while (i &lt; N)</td>
</tr>
<tr>
<td>15 C21B</td>
<td>if (R[2] == 0) PC = 1B</td>
</tr>
<tr>
<td>16 1CA9</td>
<td>R[C] = R[A] + R[9]</td>
</tr>
<tr>
<td>17 8DF8</td>
<td>R[D] = stdin</td>
</tr>
<tr>
<td>18 BD0C</td>
<td>M[R[C]] = R[D]</td>
</tr>
<tr>
<td>1A C014</td>
<td>PC = 14</td>
</tr>
<tr>
<td>1B</td>
<td>[array processing code]</td>
</tr>
</tbody>
</table>

Arrays example: Read an array from standard input

Register trace

<table>
<thead>
<tr>
<th>index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
</tr>
<tr>
<td>A</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>21</td>
<td>5</td>
</tr>
<tr>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>23</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
</tr>
<tr>
<td>25</td>
<td>9</td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0000</td>
</tr>
<tr>
<td>41</td>
<td>0000</td>
</tr>
<tr>
<td>42</td>
<td>0000</td>
</tr>
<tr>
<td>43</td>
<td>0000</td>
</tr>
<tr>
<td>44</td>
<td>0008</td>
</tr>
<tr>
<td>45</td>
<td>0000</td>
</tr>
</tbody>
</table>

PC

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>7101</td>
</tr>
<tr>
<td>11</td>
<td>8BF8</td>
</tr>
<tr>
<td>12</td>
<td>7A80</td>
</tr>
<tr>
<td>13</td>
<td>7900</td>
</tr>
<tr>
<td>14</td>
<td>22B9</td>
</tr>
<tr>
<td>15</td>
<td>C21B</td>
</tr>
<tr>
<td>16</td>
<td>1CA9</td>
</tr>
<tr>
<td>17</td>
<td>8DF8</td>
</tr>
<tr>
<td>18</td>
<td>BD0C</td>
</tr>
<tr>
<td>19</td>
<td>1991</td>
</tr>
<tr>
<td>1A</td>
<td>C014</td>
</tr>
<tr>
<td>1B</td>
<td>1</td>
</tr>
</tbody>
</table>

An instructive scenario

Alice, a scientist, develops a procedure for her experiments.

• Uses a scientific instrument connected to a paper tape punch.
• Takes the paper tape to a **computer** to process her data.
• Uses array code just described to load her data.
• Writes array-processing code that analyzes her data.
• Punches out the results on paper tape to save them.
An instructive scenario (continued)

Alice, a scientist, develops a procedure for her experiments.
- Uses a scientific instrument connected to a paper tape punch.
- Takes the paper tape to a computer to process her data.
- Uses array code from last lecture to load her data.
- Writes array-processing code that analyzes her data.

Eve, a fellow scientist, runs some experiments, too.

What happens with Eve's tape

Not what Alice expects!
- Memory 80–FE fills with $\text{8888}$.
- $\text{8888}$ appears on output.
- Address overflow from FF to 00.
- Memory 00–0F is overwritten.

And then things get worse...

What happens with Eve's tape when things get worse

Eve's tape
What happens when things get worse: Eve Owns Alice's computer

1. 8 8 8 8
2. 8 8 8 8
3. 8 8 8 8
4. 10
5. 8 8 8 8
6. 12
7. 8 8 1 1
8. 9 8 8 9
9. C 0 1 2
10. 13
11. 8 C F F
12. 16
13. 1 5 6 B
14. 1 7
15. B C 0 5
16. 1 8
17. B B 1
18. 1 9
19. Z A A 1
20. 1 A
21. C 0 1 4
22. 1 8
23. ...

Remember me? [maniacal laugh]

She could have loaded any program at all...

Buffer overflow in the real world

C/C++/Objective C string/array overflow
- Program does not check for long string.
- Hacker puts code at end of long string.
- Hacker Owns your computer.

1988
Morris Worm
infected research
computers
throughout US

2010-present
iPhone/iPad
Buffer overflow
is "top 5 vulnerability"

Note: Java tries to help us write secure code
- Array bounds checking.
- Type safety.

UNIX

2004
JPEG of death
Windows browsers
buffer overflow
on an image

Xbox/Zelda/Pokemon
Buffer overflow
enables use of unlicensed
games

Unsafe C code

#include <stdio.h>
int main(void)
{
    char buffer[100];
scanf("%s", buffer);
printf("%sn", buffer);
return 0;
}

Unsafe code

Unsafe code

Unsafe code

Unsafe code
Programs that process programs on TOY

von Neumann architecture
- No difference between data and instructions.
- Same word can be data one moment, an instruction the next.

Early programmers immediately realized the advantages
- Can save programs on physical media (dump).
- Can load programs at another time (boot).
- Can develop higher-level languages (assembly language).

Dumping

Q. How to save a program for another day?
- Day’s work represents patches and other code entered via switches.
- Must power off (vacuum tubes can’t take the heat).

A. Write a short program to dump contents of memory to tape.
- Key in program via switches in memory locations 00–08.
- Run it to save data/instructions in memory 10–FE.

Why not FF? It’s StdIn/StdOut.

DUMP code

```hex
00 7 1 0 1 R[1] = 1
01 7 2 1 0 R[2] = 10
02 7 3 F F R[3] = 00FF
03 A A 0 2 R[A] = M[R[A]]
04 9 A F F write R[A] to stdout
05 1 2 2 1 R[2] = R[2] + 1
06 2 4 3 2 R[4] = 00FF - R[2]
07 D 4 0 3 if (R[4] > 0) PC = 03
08 0 0 0 0 halt
```

Assembly language

Assembly language
- Program in a higher-level language.
- Write a machine-language program to translate.
- Used widely from early days through the 1990s.
- Still used today.

TOY machine code

```text
00 7 0 0 1
01 7 2 1 0
02 7 3 F F
03 B A 0 2
04 B A 0 2
05 1 2 2 1
06 2 4 3 2
07 D 4 0 3
08 0 0 0 0
```

TOY assembly code

```text
LA R1, 01
LA R2, 10
LA R3, FF
SI RA, R2
A R2, R2, R1
S R4, R3, R2
BP R4, LOOP
```

Advantages
- Mnemonics, not numbers, for opcodes.
- Symbols, not numbers, for addresses.
- Relocatable.
Tip of the iceberg

Practical implications of von Neumann architecture
- Installers that download applications.
- Compilers that translate Java into machine language.
- Simulators that make one machine behave like another (stay tuned).
- Cross-compilers that translate code for one machine on another.
- Dumping and booting.
- Viruses.
- Virus detection.
- Virtual machines.
- Thousands of high-level languages.
- [an extremely long list]
Toy simulator in Java

A Java program that simulates the TOY machine.

- Take program from a file named in the command line.
- Take TOY stdin/stdout from Java StdIn/StdOut.

```java
public class TOYlecture {
    public static void main(String[] args) {
        int pc = 0x10; // program counter
        int[] R = new int[32]; // registers
        int[] M = new int[16]; // main memory
        int ln = new In(args[0]);
        for (int i = 0x10; i < 0xFFAA; i += 0x10) // box 16
            M[i] = Integer.parseInt(In.readString(ln), 16);
        while (true) {
            int ir = M[pc++]; // fetch and increment
            // decode (next slide)
            // execute (second slide following)
        }
    }
}
```

TOY simulator: decoding instructions

Bitwhacking is the same in Java as in TOY

- Extract fields for both instruction formats.
- Use shift and mask technique.

**Example:** Extract destination d from 1CAB
```
I | C | A | B
---|---|---|---
1  | 0  | 1  | 1
ir >> 8 | 0  | 0  | 0  | 1  | 1  | 1  | 1  | 1
```

```
0xF

0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
```

```
(interval) & 0xF
```

```
ir >> 8 & 0xF
```

```
(data) & (mask)
```

```
ir >> 8 & 0xF
```

```
ir
```

Toy simulator in Java

Use Java switch statement to implement the simple state changes for each instruction.

**execute**

- `if (op == 0) break; // Halt`

```java
switch (op) {
    case 1: R[d] = R[i] + R[c]; break;
    case 2: R[d] = R[i] - R[c]; break;
    case 3: R[d] = R[i] & R[c]; break;
    case 4: R[d] = R[i] ^ R[c]; break;
    case 5: R[d] = R[i] << R[c]; break;
    case 6: R[d] = R[i] >> R[c]; break;
    case 7: R[d] = addr; break;
    case 8: R[c] = R[addr]; break;
    case 9: M[addr] = R[d]; break;
    case 10: R[d] = M[R[i]]; break;
    case 11: M[R[i]] = R[d]; break;
    case 12: if (R[d] == 0) pc = addr; break;
    case 13: if (R[d] > 0) pc = addr; break;
    case 14: pc = R[d]; break;
    case 15: R[d] = pc; pc = addr; break;
}
```
Toy simulator in Java

```java
public class TOYlecture {
    public static void main(String[] args) {
        int pc = 0; // program counter
        int[] R = new int[16]; // registers
        int[] A = new int[10]; // main memory
        int[] in = new int[6]; // input
        int[] out = new int[6]; // output

        int op = 0; // fetch and increment

        for (int i = 0; in[i]; i++) {
            % more read-array.toy
            7300
            7400
            7600
            7800
            % more eyes-tape.txt
            0100
            8868
            8868
            % java TOYlecture read-array.toy < eyes-tape.txt
            8868
            8868
            8868
        }
    }
}
```

Comments.

- Runs any TOY program!
- Easy to change design.
- Can develop TOY code on another machine.
- Could implement in TOY (0).

Toy development environment

- Another Java program that simulates the TOY machine
- Includes graphical simulator.
- Includes single stepping, full display of state of machine, and many other features.
- Includes many simple programs.
- Written by a graduate of this course.
- Available on the booksite.
- YOU can develop TOY software.

Same approach used for all new systems nowadays

- Build simulator and development environment.
- Develop and test software.
- Build and sell hardware.

Backward compatibility

Q. Time to build a new computer. What to do about old software?

Approach 1: Rewrite it all

- Costly and time-consuming.
- Error-prone.
- Boring.

Approach 2: Simulate the old computer on the new one.

- Not very difficult.
- Still likely more efficient.
- Succeeds for all old software.

Result. Old software remains available.

Disturbing thought: Does anyone know how it works?

Another note of caution

An urban legend about backward compatibility.

- Space shuttle solid rocket booster needed to be transported by rail.
- US railroads were built by English expats, so the standard rail gauge is 4 feet 8.5 inches.
- English rail gauge was designed to match ruts on old country roads.
- Ruts on old country roads were first made by Roman war chariots.
- Wheel spacing on Roman war chariots was determined by the width of a horse’s back end.

End result. Key space shuttle dimension determined by the width of a war horse’s back end.

Worthwhile takeaway. Backwards compatibility is Not Necessarily Always a Good Thing.
**Backward compatibility is pervasive in today’s world**

- Documents need backward compatibility with .doc format.
- Broadcast TV needs backward compatibility with analog M&W.
- Web pages need compatibility with new and old browsers.
- Business software is written in a dead language and run with many layers of emulation.
- IPhone software is written in an unsafe language.

Much of our infrastructure was built in the 1970s on machines not so different from TOY.

Time to design and build something suited for today’s world? Go for it! That means YOU!

**Virtual machines**

- Building a new rocket? Simulate it to test it.
  - Issue 1: Simulation may not reflect reality.
  - Issue 2: Simulation may be too expensive.

- Building a new computer? Simulate it to test it.
  - Advantage 1: Simulation is reality (it defines the new machine).
  - Advantage 2: Can develop software without having machine.
  - Advantage 3: Can simulate machines that may never be built.

Examples in today’s world:

- Virtual memory.
- Java virtual machine.
- Amazon cloud.

Virtual machines of many, many types (old and new) are available for use on the web.

Internet commerce is moving to such machines.

Forming a startup? Use a virtual machine.

It is likely to perform better for you than whatever real machine you might be able to afford.

**Layers of abstraction**

- Computer systems are built by accumulating layers of abstraction.

- Is TOY real?
  - TOY program
  - TOY simulator
  - Java
  - Java virtual machine
  - Machine language
  - Processor

- Is your computer real?

**Approaching a new problem?**

- Build an (abstract) language for expressing solutions.
- Design an (abstract) machine to run programs written in the language.
- Food for thought: Why build the machine? Just simulate it instead!

**Turing and von Neumann**

**Theorem (Turing, 1936).** It is possible to invent a single machine which can be used to do any computable task.

**Proof sketch.** (See theory lectures.)

- Any task can be described as a Turing machine.
- A “universal” TM (UTM) can simulate any TM.
- Key concept: Program as data.

**First Draft of a report on the EDVAC.** (von Neumann, 1945).

- A computer design with an ALU, memory, and I/O.
- Physical realization of program as data concept.

**Bottom line:** Program as data concept has always stood at the foundation of computer science.
18. von Neumann Machines