What We’ve Learned About TOY

Data representation. Binary and hex.

TOY.
- Box with switches and lights.
- 16-bit memory locations, 16-bit registers, 8-bit pc.
- 4,328 bits = \((2^8 \times 16) + (15 \times 16) + (8)\) = 541 bytes!
- von Neumann architecture.

TOY instruction set architecture. 16 instruction types.
TOY machine language programs. Variables, arithmetic, loops.

What We Do Today

Data representation. Negative numbers.

Input and output. Standard input, standard output.

Manipulate addresses. References (pointers) and arrays.

TOY simulator in Java and implications.

Data Representation
Digital World

Data is a sequence of bits. (interpreted in different ways)
• Integers, real numbers, characters, strings, ...
• Documents, pictures, sounds, movies, Java programs, ...

Ex. 01110101
• As binary integer: \( 1 + 4 + 16 + 32 + 64 = 117 \) (base ten).
• As character: 117\textsuperscript{th} Unicode character = ‘u’.
• As music: 117/256 position of speaker.
• As grayscale value: 45.7% black.

Representing Negative Integers

TOY words are 16 bits each.
• We could use 16 bits to represent 0 to \( 2^{16} - 1 \).
• We want negative integers too.
• Reserving half the possible bit-patterns for negative seems fair.

Highly desirable property. If \( x \) is an integer, then the representation of \( -x \), when added to \( x \), is zero.

Adding and Subtracting Binary Numbers

Decimal and binary addition.

```
  1
+ 013
-----
  111
```

Subtraction. Add a negative integer.

```
e.g., 6 - 4 = 6 + (-4)
```

Q. How to represent negative integers?

Two’s Complement Integers

To compute \(-x\) from \( x\):
• Start with \( x\).
• Flip bits.
• Add one.

```
+4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
-5 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
```

```
-4 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
```
## Two’s Complement Integers

<table>
<thead>
<tr>
<th>dec</th>
<th>hex</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>+3</td>
<td>0003</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1</td>
</tr>
<tr>
<td>+2</td>
<td>0002</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>+1</td>
<td>0001</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1</td>
</tr>
<tr>
<td>+0</td>
<td>0000</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1</td>
</tr>
<tr>
<td>-1</td>
<td>FFF1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1</td>
</tr>
<tr>
<td>-2</td>
<td>FFFE</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>-3</td>
<td>FFFD</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1</td>
</tr>
<tr>
<td>-4</td>
<td>FFFC</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0</td>
</tr>
</tbody>
</table>

## Representing Other Primitive Data Types in TOY

### Bigger integers
- Use two 16-bit words per int.

### Real numbers
- Use “floating point” (like scientific notation).
- Use four 16-bit words per double.

### Characters
- Use ASCII code (8 bits / character).
- Pack two characters per 16-bit word.

### Note
- Real microprocessors add hardware support for int and double.

## Properties of Two’s Complement Integers

**Properties.**
- Leading bit (bit 15) signifies sign.
- 0000000000000000 represents zero.
- Negative integer \(-x\) represented by \(2^{16} - x\).
- Addition is easy.
- Checking for arithmetic overflow is easy.

**Not-so-nice property.** Can represent one more negative integer than positive integer.

\[-32768 = 2^{15} \]

\[32767 = 2^{15} - 1\]

**Remark.** Java `int` data type is 32-bit two’s complement integer.

## Standard Input and Output
Standard Output

Standard output.
- Writing to memory location FF sends one word to TOY stdout.
- Ex. 9AFF writes the integer in register A to stdout.

```
00: 0000 0
01: 0001 1
10: 8A00 RA ← mem[00]    a = 0
11: 8B01 RB ← mem[01]    b = 1
12: 9AFF write RA to stdout while(a > 0) {
    print a
    RA = RA + RB    a = a + b
    RB = RB - RB    b = a - b
} 15: DA12 if (RA > 0) goto 12
16: 0000 halt
```
fibonacci.toy

Standard Input

Standard input.
- Loading from memory address FF loads one word from TOY stdin.
- Ex. 8AFF reads an integer from stdin and store it in register A.

Ex: read in a sequence of integers and print their sum.
- In Java, stop reading when EOF.
- In TOY, stop reading when user enters 0000.

```
while (!StdIn.isEmpty()) {
    a = Stdin.readInt();
    sum = sum + a;
} StdOut.println(sum);
```

```
00: 0000 0
10: 8C00 RC ← mem[00]
11: 8AFF read RA from stdin
12: CA15 if (RA == 0) pc ← 15
13: 1CCA RC ← RC + RA
14: C011 pc ← 11
15: 9CFF write RC
16: 0000 halt
```

Standard Input and Output: Implications

Standard input and output enable you to:
- Get information out of machine.
- Put information from real world into machine.
- Process more information than fits in memory.
- Interact with the computer while it is running.

Pointers
Load Address (a.k.a. Load Constant)

Load address. [opcode 7]
- Loads an 8-bit integer into a register.
- 7A30 means load the value 30 into register A.

Applications.
- Load a small constant into a register.
- Load a 8-bit memory address into a register.

Arrays in TOY

TOY main memory is a giant array.
- Can access memory cell 30 using load and store.
- 8C30 means load mem[30] into register C.
- Goal: access memory cell i where i is a variable.

Load indirect. [opcode A]
- AC06 means load mem[R6] into register C.

Store indirect. [opcode B]
- BC06 means store contents of register C into mem[R6].

TOY Implementation of Reverse

TOY implementation of reverse.
- Read in a sequence of integers and store in memory 30, 31, 32, ...
- Stop reading if 0000.
- Print sequence in reverse order.

<table>
<thead>
<tr>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>7101</td>
<td>7A30</td>
<td>7B00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1 &lt;- 0001</td>
<td>RA &lt;- 0030</td>
<td>RB &lt;- 0000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

16: 8CFF read RC
17: CC19 if (RC == 0) goto 19
18: 16AB R6 <- RA + RB
19: 1BB1 RB <- RB + R1
20: C013 goto 13

read in the data

print in reverse order
Unsafe Code at any Speed

Q. What happens if we make array start at 00 instead of 30?

A. With enough data, becomes a self-modifying program
   - can overflow buffer
   - and run arbitrary code!

Buffer Overrun Example: JPEG of Death

Microsoft Windows JPEG bug. [September, 2004]
   - Step 1. User views malicious JPEG in IE or Outlook.
   - Step 2. Machine is Owned.
   - Data becomes code by exploiting buffer overrun in GDI+ library.

Fix. Update old library with patched one.

But many applications install independent copies of the GDI library.

Moral.
   - Not easy to write error-free software.
   - Embrace Java security features.
   - Don’t try to maintain several copies of the same file.
   - Keep your OS patched.
**Dumping**

**Q.** Work all day to develop operating system. How to save it?

**A.** Write short program `dump.toy` and run it to dump contents of memory onto tape.

```toy
00: 70001 R1 ← 0001
01: 7210 R2 ← 0010 i = 10
02: 73FF R3 ← 00FF
03: AA02 RA ← mem[R2] a = mem[i]
04: 9AFF write RA print a
05: 1221 R2 ← R2 + R1 i++
06: 2432 R4 ← R3 - R2
07: D403 if (R4 > 0) goto 03 } while (i < 255)
08: 0000 halt
```

dump.toy

**Booting**

**Q.** How do you get it back?

**A.** Write short program `boot.toy` and run it to read contents of memory from tape.

```toy
00: 70001 R1 ← 0001 i = 10
01: 7210 R2 ← 0010
02: 73FF R3 ← 00FF
03: BAFF read RA read a
04: BA02 mem[R2] ← RA mem[i] = a
05: 1221 R2 ← R2 + R1 i++
06: 2432 R4 ← R3 - R2
07: D403 if (R4 > 0) goto 03 } while (i < 255)
08: 0000 halt
```

boot.toy

**TOY Simulator**

**Goal.** Write a program to "simulate" the behavior of the TOY machine.

- TOY simulator in Java.
- TOY simulator in TOY!
TOY Simulator: Decode

Ex. Extract destination register of 1CAB by shifting and masking.

```
int inst = mem[pc++];  // fetch and increment
int op = (inst >> 12) & 15;  // opcode (bits 12-15)
int d = (inst >> 8) & 15;  // dest d (bits 08-11)
int t = (inst >> 4) & 15;  // source t (bits 04-07)
int addr = (inst >> 0) & 255;  // addr (bits 00-07)
```

TOY Simulator: Execute

```
if (op == 0) break;  // halt

switch (op)
{
    case 1: R[d] = R[s] + R[t];    break;
    case 2: R[d] = R[s]  - R[t];    break;
    case 3: R[d] = R[s]  & R[t];    break;
    case 4: R[d] = R[s]  ^ R[t];    break;
    case 5: R[d] = R[s]  << R[t];   break;
    case 6: R[d] = R[s]  >> R[t];   break;
    case 7: R[d] = addr;           break;
    case 8: R[d] = mem[addr];      break;
    case 9: mem[addr] = R[d];      break;
    case 10: R[d] = mem[R[t]];     break;
    case 11: mem[R[t]] = 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];   pc = addr; break;
    case 15: R[d] = pc;   pc = addr; break;
}
```

TOY Simulator: Omitted Details

Omitted details.

- Register 0 is always 0.
  - reset R[0] = 0 after each fetch-execute step

- Standard input and output.
  - if addr is FF and opcode is load (indirect) then read in data
  - if addr is FF and opcode is store (indirect) then write out data

- TOY registers are 16-bit integers; program counter is 8-bit.
  - Java int is 32-bit; Java short is 16-bit
  - use casts and bit-whacking

Complete implementation. See TOY.java on booksite.

Profound ideas stemming from simulation.

- Backwards compatibility
- Virtual machines
- Layers of abstraction
Backwards Compatibility

Building a new computer? Need a plan for old software.

Two possible approaches
- Rewrite software (costly, error-prone, boring, and time-consuming).
- Simulate old computer on new computer.

Ancient programs still running on modern computers.
- Payroll
- Power plants
- Air traffic control
- Ticketron.
- Games.

Effects of Backwards Compatibility: example 1

Q. Why is Space Shuttle SRB long and narrow?

A. Fits on standard US rail gauge.

Effects of Backwards Compatibility: Example 2

Napoleon’s march on Russia.

- Progress slower than expected.
- Eastern European ruts didn’t match Roman gauge.
- Stuck in the field during Russian winter instead of Moscow.
- Lost war.

Lessons.
- Maintaining backwards compatibility can lead to inelegance and inefficiency.
- Maintaining backwards compatibility is Not Always A Good Thing.
- May need fresh ideas to conquer civilized world.
Virtual machines

Building a new airplane? Simulate it to test it.
- Issue 1: Simulation may not reflect reality.
- Issue 2: May not be able to afford simulation.

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 you wouldn’t build.

Example 1: An operating system can implement a Virtual Memory that is much larger than a real memory by simulating programs and going to disk or the web to reference “memory”

Example 2: An operating system can implement multiple Virtual Machines on the same real machine by keeping track of multiple PCs and rotating control to the different machines

Layers of Abstraction

Is TOY real?

Is Java real?

Approaching a new problem?
- build an (abstract) language for expressing solutions
- design an (abstract) machine to execute the language
- food for thought: Why build the machine?

Examples: MATLAB, BLAST, AMP ...