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 = (255 \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.

Data Representation
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: 117th 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.

Two's Complement Integers

To compute $-x$ from $x$:
- Start with $x$.
- Flip bits.
- Add one.

Decimal and binary addition.

Subtraction. Add a negative integer.

Q. How to represent negative integers?
### Two's Complement Integers

<table>
<thead>
<tr>
<th>dec</th>
<th>hex</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>+32767</td>
<td>7FFF</td>
<td>0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>+4</td>
<td>0004</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</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</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 0</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 0</td>
</tr>
<tr>
<td>-1</td>
<td>FFFF</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 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</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 0</td>
</tr>
<tr>
<td>-32768</td>
<td>8000</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

### Properties of Two's Complement Integers

- 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.

\[-32,768 = -2^{15}\]

\[32,768 = 2^{15}\]

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

### 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.
Standard Input and Output

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.

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

Standard Output

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

```plaintext
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 print a
13: 1AAB RA ← RA + RB a = a + b
14: 2BAB RB ← RA - RB b = a - b
15: DA12 if (RA > 0) goto 12 while (a > 0)
16: 0000 halt
```

Standard Input and Output: Implications

- 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 an 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 \(\text{mem}[30]\) into register \(C\).
  - Goal: access memory cell \(i\) where \(i\) is a variable.

**Load indirect.** [opcode A]

- \(AC06\) means load \(\text{mem}[R6]\) into register \(C\).

**Store indirect.** [opcode B]

- \(BC06\) means store contents of register \(C\) into \(\text{mem}[R6]\).

```java
for (int i = 0; i < N; i++)
    a[i] = StdIn.readInt();
for (int i = 0; i < N; i++)
    StdOut.println(a[N-1-i]);
```

**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.

```
10: 7101  R1 ← 0001
11: 7A30  RA ← 0030
12: 7B00  RB ← 0000
while(true) {
    13: 8CFF  read RC
    14: CC19  if (RC == 0) goto 19
    15: 16AB  R6 ← RA + RB
    16: BC06  \(\text{mem}[R6]\) ← RC
    17: 1BB1  RB ← RB + R1
    18: CO13  goto 13
}
```

read in the data
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.

```plaintext
19: CB20 if (RB == 0) goto 20
1A: 16AB R6 ← RA + RB
1B: 2661 R6 ← R6 - R1
1C: AC06 RC ← mem[R6]
1D: 9CFF RC ← mem[R6]
1E: 2BB1 RB ← RB - R1
20: 0000 halt
```

Q. What happens if we make array start at 00 instead of 30?
A. Self modifying program; can overflow buffer and run arbitrary code!

```plaintext
10: 7101 R1 ← 0001
11: 7A00 RA ← 0000
12: 7800 RB ← 0000
13: 8CFF read RC
14: CC19 if (RC == 0) goto 19
15: 16AB R6 ← RA + RB
16: BC06 mem[R6] ← RC
17: 1BB1 RB ← RB + R1
18: C013 goto 13
```

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.

**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.
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.

00: 7001  R1 ← 0001
01: 7210  R2 ← 0010
02: 73FF  R3 ← 00FF
          i = 10
03: AA02  RA ← mem[R2]  do {
04: 9AFF  write RA
             a = mem[i]
             print a
          05: 1221  R2 ← R2 + R1
          06: 2432  R4 ← R3 - R2
          07: D403  if (R4 > 0) goto 03
                   } while (i < 255)
08: 0000  halt
dump.toy

Q. How do you get it back?

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

00: 7001  R1 ← 0001
01: 7210  R2 ← 0010
02: 73FF  R3 ← 00FF
          i = 10
03: 8AFF  read RA
          a = mem[i]
          print a
          04: BA02  mem[R2] ← RA
                   mem[i] = a
          05: 1221  R2 ← R2 + R1
          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!
Fetch. Extract destination register of 1CAB by shifting and masking.

```
TOY Simulator: Fetch

int inst = ... 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0
016 016 016 F16
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
016 016 0 C16
15
(inst >> 8) & 15
```

TOY Simulator: Execute

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

switch (op) {
   case  1: R[d] = R[s] + R[t]; pc = addr; break;
   case 14: pc = R[d]; pc; pc = addr; break;
   case 15: R[d] = pc; pc = addr; break;
}
```

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.

Consequences of simulation.

- Test out new machine or microprocessor using simulator.
  - cheaper and faster than building actual machine
- Easy to add new functionality to simulator.
  - trace, single-step, breakpoint debugging
- simulator more useful than TOY itself

- Reuse software from old machines.

Ancient programs still running on modern computers.

- Ticketron.
- Lode Runner on Apple IIe.