5. The TOY Machine II

Laboratory Instrument Computer (LINC)

What We’ve Learned About TOY

TOY machine:
- 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 programming:
- TOY instruction set architecture: 16 instruction types.
- 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

Digital World

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

Ex. 01110110:
- As binary integer: 1 \times 4 + 16 + 32 + 64 = 117_{10}
- As character: 117th Unicode character = \text{u' }
- As music: 117/256 position of speaker.
- As grayscale value: 45.7% black.

Adding and Subtracting Binary Numbers

Decimal and binary addition,

\[
\begin{array}{c|c}
\text{CARRY} & \text{COFFER} \\
\hline
1 & 1 \\
013 & 0 0 0 0 1 1 0 1 \\
+ 092 & + 0 1 0 1 1 1 0 0 \\
105 & 0 1 1 0 1 0 0 1 \\
\end{array}
\]

Subtraction. Add a negative integer.

\[
\begin{array}{c|c|c}
\text{NEGATIVE} & \text{ADDITION} & \text{RESULT} \\
\hline
1 & 1 & 1 \\
0 0 0 0 0 1 1 0 1 & 0 1 0 1 1 1 0 0 & 1 0 1 0 0 0 1 1 \\
\end{array}
\]

Q. How to represent negative integers?
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.

```
x = 0000000000000000
(+x) = 0000000000000000
0
```

```
x = 0000000000000000
(+x) = 0000000000000000
x = 0000000000000000
```

```
Flip bits and add 1
```

Representing Other Primitive Data Types in TOY

Bigger integers. Use two 16-bit TOY words per 32-bit Java int.

Real numbers.
- Use IEEE floating point (e.g., scientific notation).
- Use four 16-bit TOY words per 64-bit Java double.

Characters. Use one 16-bit TOY word per 16-bit Java Unicode char.

Standard Input and Output

Properties of Two's Complement Integers

- Leading bit (bit 15) signifies sign.
- Addition and subtraction are easy.
- Checking for arithmetic overflow is easy.
- Negative integer $-x$ represented by $2^{16} - x$.
- Not symmetric: can represent $-32768$ but not $32768$.

Java. Java’s int data type is a 32-bit two’s complement integer. Ex. $2147483647 + 1$ equals $-2147483648$.
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: E000 RA ← mem[00] a = 0
11: B001 RB ← mem[01] b = 1
do {
12: 9AFF write RA to stdout
13: 1AAB RA ← RA + RB
14: 2BAB RB ← RA
15: DA12 if (RA > 0) goto 12
} while (a > 0)
16: 0000 halt
```

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.

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

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.

Arrays in TOY

TOY main memory is a giant array.

- Can access memory cell 30 using load and store.
- Goal: access memory cell i where i is a variable.

Load indirect. [opcode: A]

- 30A4 means load mem[i+4] into register c.

Store indirect. [opcode: B]

- 30A4 means store contents of register c into mem[i].

```java
for (int i = 0; i < N; i++)
a[i] = Stdin.readInt();
```

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

- Loads an 8-bit integer into a register.
- 30A3 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.

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

Pointers

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

`aligned std`
TOY Implementation of Reverse

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

```plaintext
10: 7101 R1 0001 constant 1
11: 7A30 RA 0000 n
12: 7B00 RB 0000

while(true) {
    13: BCFF read RC
    if (RC == 0) goto 19
    if (c == 0) break;
    15: 16AB R6 RA + RB address of a[n]
    16: BC06 mem[R6] RC a[n] = c;
    17: 1BB1 RB RB + R1 n++;
    18: C013 goto 13
}
```

What Can Happen When We Lose Control (in C or C++)?

Buffer overflow.
- Array buffer[] has size 100.
- User might enter 200 characters.
- Might lose control of machine behavior.

Consequences.
- Viruses and worms.
- Java enforces security.
  - Type safety.
  - Array bounds checking.
  - Not foolproof.

Buffer Overflow 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.

Unsafe Code at any Speed

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

Q. Work all day to develop operating system in mem[10] to mem[FF]. How to save it?

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

```toy
00: 7001     R1 ← 0001
01: 7210     R2 ← 0010
02: 73FF     R3 ← 00FF
03: AA02     RA ← mem[R2]
04: 9AFF     write RA
            print a
05: 1221     R2 ← R2 + R1
06: 2432     R4 ← R3 - R2
07: D403     if (R4 > 0) goto 03
08: 0000     halt
```

Booting

Q. How do you get it back?

A. Write short program boot.toy and run it to read contents of mem[10] to mem[FF] from tape.

```toy
00: 7001     R1 ← 0001
01: 7210     R2 ← 0010
02: 73FF     R3 ← 00FF
03: BA02     RA ← mem[R2]
04: BB0F     read RA
            print a
05: 1221     R2 ← R2 + R1
06: 2432     R4 ← R3 - R2
07: D403     if (R4 > 0) goto 03
08: 0000     halt
```

Extra Slides

Two's Complement Arithmetic

Addition is carried out as if all integers were positive.

- It usually works.
- But overflow can occur.

```
-3 + 4 =
```

Java and TOY

Correspondence between Java constructs and TOY mechanisms.

<table>
<thead>
<tr>
<th>Java</th>
<th>TOY</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignment</td>
<td>load, store</td>
</tr>
<tr>
<td>arithmetic expressions</td>
<td>add, subtract</td>
</tr>
<tr>
<td>logical expressions</td>
<td>xor, and, shift</td>
</tr>
<tr>
<td>loops (for, while)</td>
<td>jump absolute, branch</td>
</tr>
<tr>
<td>branches (if-else, switch)</td>
<td>branch if zero, positive</td>
</tr>
<tr>
<td>array, linked lists</td>
<td>indirect addressing</td>
</tr>
<tr>
<td>function call</td>
<td>jump and link, jump indirect</td>
</tr>
<tr>
<td>recursion</td>
<td>implement stack with arrays</td>
</tr>
<tr>
<td>whitespace</td>
<td>no-op 1000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>