

5. The TOY Machine II



Laboratory Instrument Computer (LINIC)

Introduction to Computer Science: An Interdisciplinary Approach · Robert Sedgwick and Kevin Wayne · Copyright ©2002-2011 · 19/3/2012 B-44:53

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)

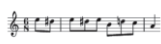
- Integers, real numbers, characters, strings, ...
- Documents, pictures, sounds, movies, Java programs, ...

Ex. 01110101

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



```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```



Adding and Subtracting Binary Numbers

Decimal and binary addition.

$$\begin{array}{r}
 \begin{array}{r}
 1 \\
 013 \\
 + 092 \\
 \hline
 105
 \end{array}
 \qquad
 \begin{array}{r}
 \text{carries} \\
 \begin{array}{r}
 11 \\
 00001101 \\
 + 01011100 \\
 \hline
 01101001
 \end{array}
 \end{array}
 \end{array}$$

Subtraction. Add a negative integer.

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

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.

$$\begin{array}{r}
 x \quad \quad \quad 00110100 \\
 + (-x) \quad + ? ? ? ? ? ? \\
 \hline
 0 \quad \quad \quad 00000000
 \end{array}$$

$$\begin{array}{r}
 x \quad \quad \quad 00110100 \\
 + (-x) \quad + 11001011 \\
 \hline
 0 \quad \quad \quad 00000000
 \end{array}$$

flip bits and add 1

Two's Complement Integers

To compute $-x$ from x :

- Start with x . 
- Flip bits. 
- Add one. 

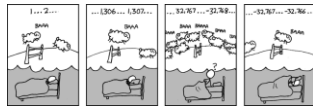
Two's Complement Integers

dec	hex	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+32767	7FFF	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
+4	0004	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
+3	0003	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
+2	0002	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
+1	0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
+0	0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	FFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
-2	FFFE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
-3	FFFD	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
-4	FFFC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
-32768	8000	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Properties of Two's Complement Integers

- Properties.
- Leading bit (bit 15) signifies sign.
 - Addition and subtraction are easy.
 - 0000000000000000 represents zero.
 - Checking for arithmetic overflow is easy.
 - Negative integer $-x$ represented by $2^{16} - x$.
 - Not symmetric: can represent $-32,768$ but not $32,768$.

Java. Java's `int` data type is a 32-bit two's complement integer.
 Ex. $2147483647 + 1$ equals -2147483648 .



<http://xos4.com/571>

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 (like 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`.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4																
5																
6																
7																



M = 0 1001 1012 = 4 D16
 O = 0 1001 1112 = 4 F16
 M = 0 1001 1012 = 4 D16

Note. Real microprocessors add hardware support for `int` and `double`.

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
                        do {
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
    
```

fibon acci . toy

0000
0001
0001
0002
0003
0005
0008
000B
0015
0022
0037
0059
0090
00E9
0179
0262
0308
0630
0A18
1055
1A6D
2AC2
452F
6FF1

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);
    
```

```

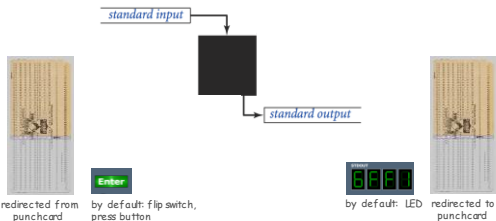
00: 0000 0
10: 8C00 RC ← mem[00]
11: 8AFF read RA from stdin
12: CA15 if (RA == 0) pc ← 15
13: 1CCA RC ← RC + RA
14: C011 pc ← 11
15: 9CFF write RC
16: 0000 halt
    
```

00AE
0046
0003
0000
00F7

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.

```
a = 0x30;
```

Java code

register stores "pointer" to a memory cell

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	1	0	0	0	1	1	0	0	0	0
7 ₁₆				A ₁₆				3 ₁₆				0 ₁₆			
opcode				dest. d				addr							

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] a variable index

- ac06 means load mem[R6] into register c.

Store indirect. [opcode B] a variable index

- bc06 means store contents of register c into mem[R6].

```

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

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

30	0000
31	0001
32	0002
33	0003
34	0004
35	0005
36	0006
37	000B
...	...

TOY memory

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.

```

10: 7101 R1 ← 0001      constant 1
11: 7A30 RA ← 0030      a[]
12: 7B00 RB ← 0000      n

13: 8CFF read RC          while(true) {
14: CC19 if (RC == 0) goto 19   c = StdIn.readInt();
15: 16AB R6 ← RA + RB         if (c == 0) break;
16: BC06 mem[R6] ← RC         memory address of
17: 1BB1 RB ← RB + R1         a[n]
18: C013 goto 13             }

                                a[n] = c;
                                n++;
                                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, ... until reading 0000.
- Print sequence in reverse order.

```

19: CB20 if (RB == 0) goto 20   while (n > 0) {
1A: 16AB R6 ← RA + RB         address of a[n]
1B: 2661 R6 ← R6 - R1         address of a[n-1]
1C: AC06 RC ← mem[R6]         c = a[n-1];
1D: 9CFF write RC             StdOut.println(c);
1E: 2BB1 RB ← RB - R1         n--;
1F: C019 goto 19              }
20: 0000 halt
    
```

print in reverse order

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!

```

10: 7101 R1 ← 0001      constant 1
11: 7A00 RA ← 0000      a[]
12: 7B00 RB ← 0000      n

13: 8CFF read RC          while(true) {
14: CC19 if (RC == 0) goto 19   c = StdIn.readInt();
15: 16AB R6 ← RA + RB         if (c == 0) break;
16: BC06 mem[R6] ← RC         address of a[n]
17: 1BB1 RB ← RB + R1         a[n] = c;
18: C013 goto 13             n++;
    
```

```

% more crazy8.txt
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
8888 8810
98FF C011
    
```

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.

```

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

unsafe C program

Consequences. Viruses and worms.

Java enforces security.

- Type safety.
- Array bounds checking.
- Not foolproof.



shine 50W bulb at DRAM [Apple-Govindavaghda '03]

Buffer Overflow Attacks

Stuxnet worm. [July 2010]

- Step 1. Natanz centrifuge fuel-refining plant employee plugs in USB flash drive.
- Step 2. Data becomes code by exploiting Window buffer overflow; machine is Owned.
- Step 3. Uranium enrichment in Iran stalled.



More buffer overflow attacks: Morris worm, Code Red, SQL Slammer, iPhone unlocking, Xbox softmod, JPEG of death, ...

Lesson.

- Not easy to write error-free software.
- Embrace Java security features.
- Keep your OS patched.

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.

but many applications install independent copies of 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 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.

```

00: 7001 R1 ← 0001           i = 10
01: 7210 R2 ← 0010
02: 73FF R3 ← 00FF           do {
03: AA02 RA ← mem[R2]       a = mem[i]
04: 98FF 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

15

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.

```

00: 7001 R1 ← 0001           i = 10
01: 7210 R2 ← 0010
02: 73FF R3 ← 00FF           do {
03: 8AFF 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

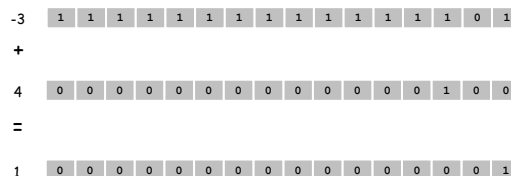
16

Extra Slides

40

Two's Complement Arithmetic

Addition is carried out as if all integers were positive.
 . It usually works.

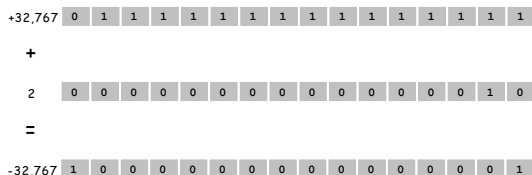


41

Two's Complement Arithmetic

Addition is carried out as if all integers were positive.
 . It usually works.
 . But overflow can occur.

carry into sign (leftmost) bit with no carry out
 or carry out out of sign bit with no carry in



42

Java and TOY

Correspondence between Java constructs and TOY mechanisms.

Java	TOY
assignment	load, store
arithmetic expressions	add, subtract
logical expressions	xor, and, shifts
loops (for, while)	jump absolute, branch
branches (if-else, switch)	branch if zero, positive
arrays, linked lists	indirect addressing
function call	jump and link, jump indirect
recursion	implement stack with arrays
whitespace	no-op 1000
...	...

43