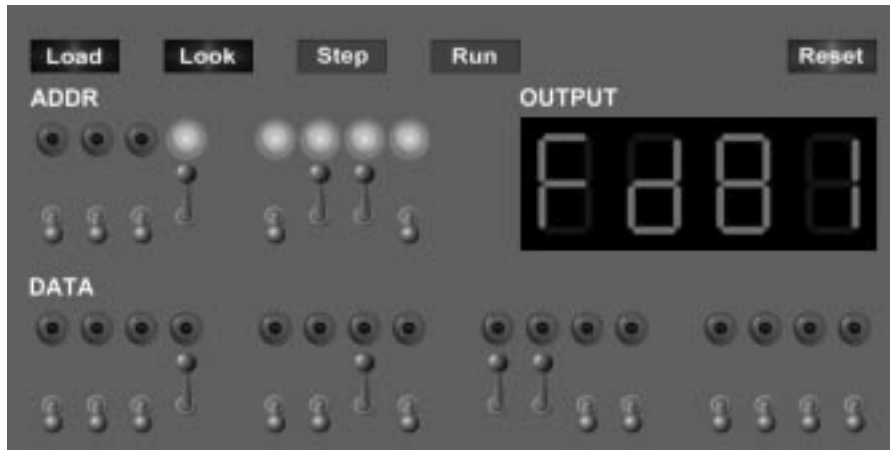


# Lecture A1: The X-TOY Machine



## What is X-TOY?

An imaginary machine similar to:

- Ancient computers.
- Today's microprocessors.

Why study?

- Machine language programming.
  - how do C programs relate to computer?
  - still (a few) situations today where it is really necessary
- Computer architecture.
  - how is a computer put together?
  - how does it work?
- Simplified machine.
  - captures essence of real computers

## Inside the Box

Switches.

- Input data and programs.

Lights.

- View data.

Registers.

- Fastest form of storage.
  - each stores 16 bits
- Use as scratch space during computation.
- 16 registers.
  - each stores 16 bits
- Register 0 is always 0.

Program counter (PC).

- An extra 8-bit register.
- Keeps track of next instruction to be executed.

Memory.

- Store data and programs.
- 256 "words."
  - each word stores 16 bits
- FF for stdin / stdout.

ALU (arithmetic-logic unit).

- Manipulate data.

## Data and Programs Encoded in Binary

Each bit consists of two states:

- Switch is ON or OFF.
- High voltage or low voltage.
- 1 or 0.
- True or false.

Dec	Bin	Dec	Bin
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

How to represent integers?

- Use binary encoding.
- Ex:  $6375_{10} = 0001100011100111_2$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1

$$\begin{aligned}
 6375_{10} &= +2^{12} + 2^{11} && +2^7 + 2^6 + 2^5 && +2^2 + 2^1 + 2^0 \\
 &= 4096 + 2048 && +128 + 64 + 32 && +4 + 2 + 1
 \end{aligned}$$

## Shorthand Notation

Use hexadecimal (base 16) representation.

- Binary code, four bits at a time.
- Ex:  $6375_{10} = 0001100011100111_2 = 18E7_{16}$

Dec	Bin	Hex	Dec	Bin	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1
1				8				E				7			

$$6375_{10} = 1 \times 16^3 + 8 \times 16^2 + 14 \times 16^1 + 7 \times 16^0$$

$$= 4096 + 2048 + 224 + 7$$

7

## Machine "Core" Dump

EVERYTHING is encoded in binary (hex).

- Integers.
- Machine instructions.
- Text.
- Reals.
- ...

Machine contents at a particular place and time.

- Record of what program has done.
- Completely determines what program will do.

Registers								PC
0	1	2	3	4	5	6	7	10
0000	0788	B700	0010	0401	0002	0003	00A0	10
8	9	A	B	C	D	E	F	
0000	0788	B700	0010	0401	0002	0003	00A0	

Main Memory								
00:	0000	0000	0000	0000	0000	0000	0000	0000
08:	0000	0000	0000	0000	0000	0000	0000	0000
10:	9222	9120	1121	A120	1121	A121	7211	0000
18:	0000	0001	0002	0003	0004	0005	0006	0007
20:	0008	0009	000A	000B	000C	000D	000E	000F
28:	0000	0000	0000	FE10	FACE	CAFE	ACED	CEDE
.								
.								
E8:	1234	5678	9ABC	DEF0	0000	0000	F00D	0000
F0:	0000	0000	EEEE	1111	EEEE	1111	0000	0000
F8:	B1B2	F1F5	0000	0000	0000	0000	0000	0000

8

## Program and Data

Program:

- Sequence of instructions.

16 instruction types:

- 16-bit word (interpreted one way).
- Changes contents of registers, memory, and PC in specified, well-defined ways.

Data:

- 16-bit word (interpreted other way).

Program counter (PC):

- Stores memory address of "next instruction."

Instructions	
0:	halt
1:	add
2:	subtract
3:	and
4:	xor
5:	shift left
6:	shift right
7:	load address
8:	load
9:	store
A:	load indirect
B:	store indirect
C:	branch zero
D:	branch positive
E:	jump register
F:	jump and link

9

## Using the X-TOY Machine: Input, Output

To enter a program or data:

- Set 8 memory address switches.
- Set 16 data switches.
- Press LOAD.
  - data written into addressed word of memory

To view the results of a program:

- Set 8 memory address switches.
- Press LOOK.
  - contents of addressed word of memory appears in lights



10

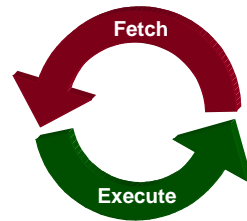
## Using the X-TOY Machine: Run

### To run the program:

- Set 8 memory address switches to address of first instruction.
- Press the RUN or STEP button.
  - loads PC from address switches
  - initiates fetch-execute cycle
  - machine repeats fetch-execute cycle until halt instruction

### Fetch-execute cycle.

- **FETCH.**
  - get instruction from memory
- **EXECUTE.**
  - update PC
  - move data to or from memory, registers
  - perform calculations



11

## X-TOY Instruction Set Architecture

### X-TOY instruction set architecture (ISA).

- Interface that specifies behavior of machine.
- 16 register, 256 words of main memory, 16-bit words.
- 16 instructions.

### Each instruction consists of 16 bits.

- Bits 12-15 encode one of 16 instruction types or opcodes.
- Bits 8-11 encode destination register d.
- Bits 0-7 encode:
  - Format 1: source registers s and t
  - Format 2: 8-bit memory address or constant

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	1	0	1	0	0	0	0	0	0	1	0	0
<b>Format 1</b>	opcode				dest d				source s				source t			
<b>Format 2</b>	opcode				dest d				addr							

12

## Load Address

### Load address. (opcode 7)

- Loads an 8-bit integer into a register.
- Format 2.
- 7A04 means:
  - load the value 0004 into register A
  - $R[A] \leftarrow 0004$

C code
a = 4;

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	0	0	0	1	0	0
7 <sub>16</sub>				A <sub>16</sub>				0 <sub>16</sub>				4 <sub>16</sub>			
opcode				dest d				addr							

13

## Load, Store

### Load. (opcode 8)

- Loads the contents of some memory location into a register.
- 8A04 means:
  - load the contents of memory location 04 into register A
  - $R[A] \leftarrow \text{mem}[04]$

### Store. (opcode 9)

- Opposite of load.
- Store the contents of a register into main memory.

load-store.toy	
04:	1111
05:	0000
10:	8A04 R[A] ← mem[04]
11:	9A05 mem[05] ← R[A]
12:	0000 halt

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

14

## Add, Subtract

### Add. (opcode 1)

- Add contents of two registers and store sum in a third.
- 1CAB means:
  - add contents of registers A and B
  - put result in register C
  - $R[C] \leftarrow R[A] + R[B]$

add.toy	
00:	0005 5
01:	0008 8
10:	8A00 $R[A] \leftarrow \text{mem}[00]$
11:	8B01 $R[B] \leftarrow \text{mem}[01]$
12:	1CAB $R[C] \leftarrow R[A] + R[B]$
13:	9C02 $\text{mem}[02] \leftarrow R[C]$
14:	0000 halt

### Subtract. (opcode 2)

- Analogous to add.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	1	0	1	0	1	1
1 <sub>16</sub>				C <sub>16</sub>				A <sub>16</sub>				B <sub>16</sub>			
opcode				dest d				source s				source t			

15

## Branch Zero, Branch Positive

### Branch if zero. (opcode C)

- Changes PC depending of value of some register.
- Used to implement loops, if-else.

### Multiply.

- Load in integers a and b, and store  $c = a \times b$ .
- Brute-force algorithm:
  - initialize  $c = 0$
  - add b to c, a times
- Problems.

multiply.c
<code>int a =7, b = 8;</code>
<code>int c;</code>
<code>for (c = 0; a != 0; a--)</code>
<code>    c += b;</code>

### Branch if positive. (opcode D)

- Analogous.

16

## Branch Zero, Branch Positive

### Branch if zero. (opcode C)

- Changes PC depending of value of some register.
- Used to implement loops, if-else.

### Multiply.

- Load in integers a and b, and store  $c = a \times b$ .
- Brute-force algorithm:
  - initialize  $c = 0$
  - add b to c, a times

multiply.toy	
0A:	0007 7
0B:	0008 8
10:	8A0A $R[A] \leftarrow \text{mem}[0A]$
11:	8B0B $R[B] \leftarrow \text{mem}[0B]$
12:	7C00 $R[C] \leftarrow 00$ ← result
13:	7101 $R[1] \leftarrow 01$ ← always 1
14:	CA18 $\text{if } (R[A] == 0) \text{ pc} \leftarrow 18$
15:	1CCB $R[C] += R[B]$
16:	2AA1 $R[A]--$
17:	C014 $\text{pc} \leftarrow 14$
18:	9C0C $\text{mem}[0C] \leftarrow R[C]$
19:	0000 halt

Annotations: A red circle around line 17 is labeled "loop". A pink box around line 14 is labeled "opcode C branch if zero".

17

## Step-By-Step Trace of multiply.toy

		R[1]	R[A]	R[B]	R[C]
10:	8A0A		0003		
11:	8B0B			0007	
12:	7C00				0000
13:	7101	0001			
14:	CA18				
15:	1CCB				0007
16:	2AA1		0002		
17:	C014				
14:	CA18				
15:	1CCB				000E
16:	2AA1		0001		
17:	C014				
14:	CA18				
15:	1CCB				0015
16:	2AA1		0000		
17:	C014				
14:	CA18				
18:	9C0C				
19:	0000				

18

## A Little History

### ENIAC. (Eckert and Mauchly, 1946)

- First general purpose electronic computer.
- 30 x 50 x 8.5 ft.
- 17,468 vacuum tubes.
- 300 multiplication per second.
- Conditional jumps, programmable.
  - code: set switches
  - data: punch cards
  - used to compute artillery firing tables

19

## Basic Characteristics of X-TOY Machine

### X-TOY is a general-purpose computer.

- Sufficient power to perform any computation.
- Limited only by amount of memory (and time).

### Stored-program computer. (von Neumann memo, 1944)

- Data and instructions encoded in binary.
- Data and instructions stored in SAME memory.
- Can change program (control) without rewiring.
  - immediate applications
  - profound implications
- EDSAC (Wilkes 1949).
  - first stored-program computer
- Same ideas in Turing's work.

All modern computers are general-purpose computers and have same (von Neumann) architecture.

21

## TOY Cheat Sheet

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Format 1	opcode				dest d				source s				source t			
Format 2	opcode				dest d				addr							

#	Operation	Fmt	Pseudocode
0:	halt	1	exit(0)
1:	add	1	$R[d] \leftarrow R[s] + R[t]$
2:	subtract	1	$R[d] \leftarrow R[s] - R[t]$
3:	and	1	$R[d] \leftarrow R[s] \& R[t]$
4:	xor	1	$R[d] \leftarrow R[s] \wedge R[t]$
5:	shift left	1	$R[d] \leftarrow R[s] \ll R[t]$
6:	shift right	1	$R[d] \leftarrow R[s] \gg R[t]$
7:	load addr	2	$R[d] \leftarrow \text{addr}$
8:	load	2	$R[d] \leftarrow \text{mem}[\text{addr}]$
9:	store	2	$\text{mem}[\text{addr}] \leftarrow R[d]$
A:	load indirect	1	$R[d] \leftarrow \text{mem}[R[t]]$
B:	store indirect	1	$\text{mem}[R[t]] \leftarrow R[d]$
C:	branch zero	2	if $(R[d] == 0)$ $pc \leftarrow \text{addr}$
D:	branch positive	2	if $(R[d] > 0)$ $pc \leftarrow \text{addr}$
E:	jump register	2	$pc \leftarrow R[d]$
F:	jump and link	2	$R[d] \leftarrow pc; pc \leftarrow \text{addr}$

23