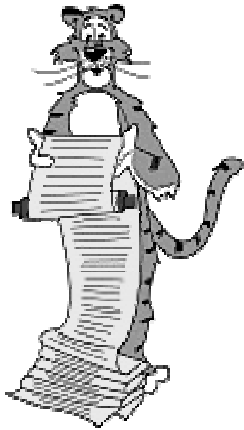


# Lecture A1: The TOY Machine



## What is TOY?

An imaginary machine similar to:

- Ancient computers.
- Today's microprocessors.

Why study?

- Simplified machine.
  - easier to understand
  - captures essence of modern microprocessors
- Machine language programming.
  - how do C programs relate to computer?
  - still (a few) situations today where it is really necessary
- Computer architecture.
  - how do machines work?

4

## Inside the Box

Switches.

- Input data and programs.

Lights.

- View data.

Registers.

- Fastest form of storage.
- Use as scratch space during computation.
- 8 registers.
  - each stores 16 bits

Program counter (PC).

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

5

## Inside the Box

Memory.

- Store data and programs.
- 256 "words".
  - each word stores 16 bits

ALU (arithmetic-logic unit).

- Execute instructions and manipulate data.

6

## 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.

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

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

7

## Shorthand Notation

Use hexadecimal (base 16) representation.

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

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			

$$\begin{aligned}
 6375 &= 1 * 16^3 + 8 * 16^2 + 11 * 16^1 + 7 * 16^0 \\
 &= 4096 + 2048 + 224 + 7
 \end{aligned}$$

9

## Machine Dump

Everything is encoded in binary (hex).

- Integers, machine instructions, text, reals, etc.

Registers							
R7	R6	R5	R4	R3	R2	R1	R0
0000	0788	E700	0010	0401	0002	0003	00A0

Contents of machine in hexadecimal "dump".

- Record of what program has done.
- Determines (with PC) what program will do.

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

10

## Program and Data

Program:

- Sequence of instructions.

Instruction:

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

Data:

- 16-bit word (interpreted other ways).

Program counter (PC):

- Stores memory address of "next instruction."

Instructions	
0:	halt
1:	add
2:	subtract
3:	multiply
4:	system call
5:	jump
6:	jump if greater
7:	jump and count
8:	jump and link
9:	load
A:	store
B:	load address
C:	xor
D:	and
E:	shift right
F:	shift left

11

## How to Use the TOY Machine

### To run a program:

- Load the program and data.  
(set switches, press LOAD for each word)
- Set switches to address of first instruction.
- Press GO.

12

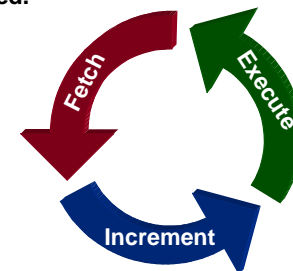
## Basic Cycle

### GO button:

- Loads PC from address switches.
- Initializes fetch-increment-execute cycle.
- Machine runs until halt instruction reached.

### Basic cycle:

- FETCH (get instruction from memory into ALU).
- INCREMENT program counter.
- EXECUTE (may require data from or to memory).



### Output:

- System call can write output to output device (tty).
- Read contents of memory word in lights.

13

## TOY Instructions

### Each instruction consists of 16 bits.

- Leftmost four bits (first hex digit) encode instruction type or **opcode**.
- Divide up remaining 12 bits to denote component of each instruction.
- Format 1 and Format 2 instructions.
  - different ways of dividing up the 12 bits

Instructions	
0:	halt
1:	add
2:	subtract
3:	multiply
4:	system call
5:	jump
6:	jump if greater
7:	jump and count
8:	jump and link
9:	load
A:	store
B:	load address
C:	xor
D:	and
E:	shift right
F:	shift left

14

## Format 1

### Register-register.

- Bits 12-15 encode opcode.
- Bits 8-11 encode destination register.
- Bits 4-7 encode source register A.
- Bits 0-3 encode source register B.

### Ex: 1234 means

- Add registers R3 and R4.
- Put result in register R2.
- $R2 \leftarrow R3 + R4$

Format 1 Instructions	
0:	halt
1:	add
2:	subtract
3:	multiply
4:	system call
C:	xor
D:	and

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0
$1_{16}$				$2_{16}$				$3_{16}$				$4_{16}$			
opcode				dest				regA				regB			

15

## Format 1: AND

Logic operations are BITWISE:

- $1234_{16} \& \text{FAD}2_{16} = 1210_{16}$

x	y	AND
0	0	0
0	1	0
1	0	0
1	1	1

0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0
$1_{16}$				$2_{16}$				$3_{16}$				$4_{16}$			

&

1	1	1	1	1	0	1	0	1	1	0	1	0	0	1	0
$F_{16}$				$A_{16}$				$D_{16}$				$2_{16}$			

=

0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0
$1_{16}$				$2_{16}$				$1_{16}$				$0_{16}$			

16

## Format 1: XOR

Logic operations are BITWISE:

- $1234_{16} \wedge \text{FAD}2_{16} = \text{E8E6}_{16}$

x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0
$1_{16}$				$2_{16}$				$3_{16}$				$4_{16}$			

^

1	1	1	1	1	0	1	0	1	1	0	1	0	0	1	0
$F_{16}$				$A_{16}$				$D_{16}$				$2_{16}$			

=

1	1	1	0	1	0	0	0	1	1	1	0	0	1	1	0
$E_{16}$				$8_{16}$				$E_{16}$				$6_{16}$			

17

## Other Logical Operations

Any logical operation can be implemented with AND and XOR.

- See Boolean circuit lecture.

Build OR from AND and XOR.

- $(x \& y) \wedge (x \wedge y)$

x	y	OR
0	0	0
0	1	1
1	0	1
1	1	1

Build NOT from XOR.

- $1 \wedge x = x'$

x	NOT
0	1
1	0

18

## Format 2

Register-memory / register-immediate.

- Bits 12-15 encode opcode.
- Bits 8-11 encode destination register.
- Bits 0-7 encode memory address or arithmetic constant.

Ex: B234 means

- Load the value  $34_{16}$  into register R2.
- $R2 \leftarrow 0034$

### Format 2 Instructions

5:	jump
6:	jump if greater
7:	jump and count
8:	jump and link
9:	load
A:	store
B:	load address
E:	shift left
F:	shift right

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	1	0	0	0	1	1	0	1	0	0
$B_{16}$				$2_{16}$				$34_{16}$							
opcode				dest				addr							

19

## Format 2

Ex: B234 means

- Load the value  $34_{16}$  into register R2.
- $R2 \leftarrow 0034$

Ex: 9234 means

- Load contents of memory location  $34_{16}$  into register R2.
- $R2 \leftarrow \text{mem}[34]$

Ex: A234 means

- Store the contents of register R2 into memory location  $34_{16}$ .
- $\text{mem}[34] \leftarrow R2$

Format 2 Instructions	
5:	jump
6:	jump if greater
7:	jump and count
8:	jump and link
9:	load
A:	store
B:	load address
E:	shift left
F:	shift right

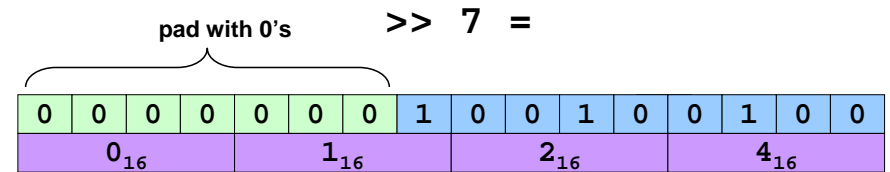
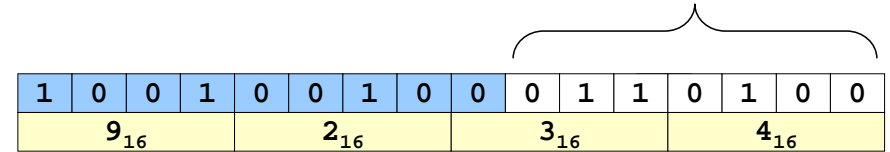
Don't confuse opcodes '9' and 'B'.

20

## Format 2: Shift

Shift:

- Shift bits left or right.
- Pad with zeros.
- Ex (shift right):  $9234_{16} \gg 7_{16} = 0124_{16}$  discard



21

## Sample TOY Program 0

TOY code for C expression:  $t = b*b - 4*a*c$ .

- Suppose memory locations D0-D3 used to store data:
  - D0 stores a
  - D1 stores b
  - D2 stores c
  - D3 stores t
- Suppose memory locations 10-19 used to store instructions:
  - 10: 91D1 3111 B204 93D0 94D2 3223 3224 2112
  - 18: A1D3 0000

Set PC to 10; Press GO.

- TOY computes the value.

22

## Sample TOY Program 0

Step-by-step trace:

quadratic.toy	
10: 91D1	R1 <- b
11: 3111	R1 <- b*b
12: B204	R2 <- 4
13: 93D0	R3 <- a
14: 94D2	R4 <- c
15: 3223	R2 <- 4*a
16: 3224	R2 <- (4*a)*c
17: 2112	R2 <- (b*b) - (4*a*c)
18: A1D3	t <- (b*b - 4*a*c)
19: 0000	halt

C compiler produces code like this.

Stay tuned for what happens if:

- Result of subtraction is negative.
- Numbers are too big.
- We need the square root (!)

23

## Sample TOY Program 1: More Arithmetic

TOY code to compute  $1 + 2 + 3 + 4 + 5 + 6 = 21_{10} = 15_{16}$ .

- Suppose memory locations 10-1E used to store instructions:
  - 10: B001 B200 B101 1221 1110 1221 1110 1221
  - 18: 1110 1221 1110 1221 1110 1221 0000 0000

Set PC to 10; Press GO.

- TOY computes the value.

24

## Sample TOY Program 1: More Arithmetic

Step-by-step trace of sum1.toy.

		<u>R1</u>	<u>R2</u>
10: B001	R0 <- 0001		
11: B200	R2 <- 0000		0000
12: B101	R1 <- 0001	0001	
13: 1221	R2 <- R2 + R1		0001
14: 1110	R1 <- R1 + R0	0002	
15: 1221	R2 <- R2 + R1		0003
16: 1110	R1 <- R1 + R0	0003	
17: 1221	R2 <- R2 + R1		0006
18: 1110	R1 <- R1 + R0	0004	
19: 1221	R2 <- R2 + R1		000A
1A: 1110	R1 <- R1 + R0	0005	
1B: 1221	R2 <- R2 + R1		000F
1C: 1110	R1 <- R1 + R0	0006	
1D: 1221	R2 <- R2 + R1		0015
1E: 0000	halt		

25


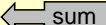

## Sample TOY Program 1: Loop

TOY code to compute  $N + (N - 1) + \dots + 2 + 1$  for any value of N loaded into R1 initially.

- Suppose memory locations 10-17 used to store instructions:
  - 10: B106 B200 B001 1221 2110 6113 0000 0000

Set PC to 10; Press GO.

- TOY computes the value.

sum2.toy			
10: B106		R1 <- 0006	
11: B200		R2 <- 0000	
12: B001		R0 <- 0001	
13: 1221		R2 <- R2 + R1	
14: 2110		R1 <- R1 - R0	
15: 6113		jump to 13 if (R1 > 0)	
16: 0000		halt	

26

## Sample TOY Program 1: Loop

Step-by-step trace of sum2.toy.

		<u>R1</u>	<u>R2</u>
10: B106	R1 <- 0006	0006	
11: B200	R2 <- 0000		0000
12: B001	R0 <- 0001		
13: 1221	R2 <- R2 + R1		0006
14: 2110	R1 <- R1 - R0	0005	
15: 6113	jump if (R1 > 0)		
13: 1221	R2 <- R2 + R1		000B
14: 2110	R1 <- R1 - R0	0004	
15: 6113	jump if (R1 > 0)		
13: 1221	R2 <- R2 + R1		000F
14: 2110	R1 <- R1 - R0	0003	
15: 6113	jump if (R1 > 0)		
13: 1221	R2 <- R2 + R1		0012
14: 2110	R1 <- R1 - R0	0002	
15: 6113	jump if (R1 > 0)		
13: 1221	R2 <- R2 + R1		0014
14: 2110	R1 <- R1 - R0	0001	
15: 6113	jump if (R1 > 0)		
13: 1221	R2 <- R2 + R1		0015
14: 2110	R1 <- R1 - R0	0000	
15: 6113	jump if (R1 > 0)		
16: 0000	halt		

27

## Program 2: Horner's Method

Goal: evaluate  $2x^3 + 3x^2 + 9x + 7$  at  $x = 10$ .

- Assume "data" stored in locations 30 - 34
  - x    a    b    c    d
  - 30: 000A 0002 0003 0009 0007 0000 0000 0000

First try:

- Compute  $x^3$ , multiply by a; compute  $x^2$ , multiply by b, ... (cumbersome, inefficient)

Efficient algorithm (Horner's method):

- Rewrite  $ax^3 + bx^2 + cx + d$  as  $((ax + b)x + c)x + d$ .
- Does polynomial evaluation for arbitrary x.
- Many applications (e.g., convert from decimal to hex).
- One raison d'être for early machines.

28

## Program 2: Horner's Method

Step-by-step trace.

- Converts from decimal to hex.
- $2397_{10} = 95D_{16}$ .

horner.toy		
10: 9430	R4 <- M[30]	000A x
11: 9531	R5 <- M[31]	0002 a
12: 3554	R5 <- R5 * R4	0014 a*x
13: 9632	R6 <- M[32]	0003 b
14: 1556	R5 <- R5 + R6	0017 a*x+b
15: 3554	R5 <- R5 * R4	00DC (a*x+b)*x
16: 9633	R6 <- M[33]	0009 c
17: 1556	R5 <- R5 + R6	00E5 (a*x+b)*x + c
18: 3554	R5 <- R5 * R4	0956 ((a*x+b)*x+c)*x
19: 9634	R6 <- M[34]	0007 d
1A: 1556	R5 <- R5 + R6	095D ((a*x+b)*x+c*x)+d
1B: 4502	write R5 to tty	
1C: 0000	halt	

29

## Program 3: Bit Manipulation

Example 3.

- Suppose memory location D0 is used to store LFBSR value:
  - D0: 0684
- Suppose memory locations 10-1B used to store instructions:
  - 10: 92D0 93D0 B001 E203 D220 E30A D330 C323
  - 18: 92D0 F201 1223 A2D0 0000 0000 0000 0000

Set PC to 10. Press GO. What happens?

- TOY simulates 1 step of LFBSR.

30

## Program 3: Bit Manipulation

Step-by-step trace.

lfbsr.toy			
10: 92D0	R2 <- mem[D0]	0000011010000100	← load shift register fill
11: 93D0	R3 <- mem[D0]	0000011010000100	
12: B001	R0 <- 1		
13: E203	R2 <- R2 >> 3	0000011010000	
14: D220	R2 <- R2 & 1	0	← bit 3
15: E30A	R3 <- R3 >> 10	000001	
16: D330	R3 <- R3 & 1	1	← bit 10
17: C323	R3 <- T2 ^ R3	1	
18: 92D0	R2 <- mem[D0]	0000011010000100	
19: F201	R2 <- R2 << 1	0000110100001000	
1A: 1223	R2 <- R2 + R3	0000110100001001	← updated value
1B: A2D0	mem[D0] <- R2		
1C: 0000	halt		

31

## 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

32

## Basic Characteristics of TOY Machine

### TOY is a general purpose computer.

- Sufficient power to perform and 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

34