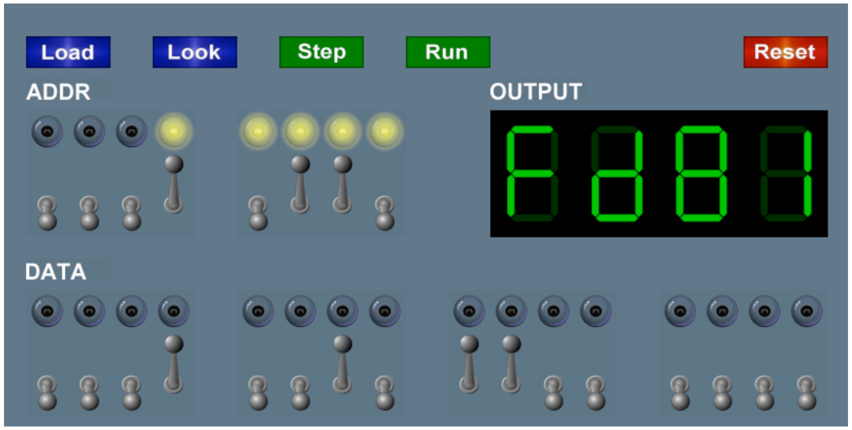
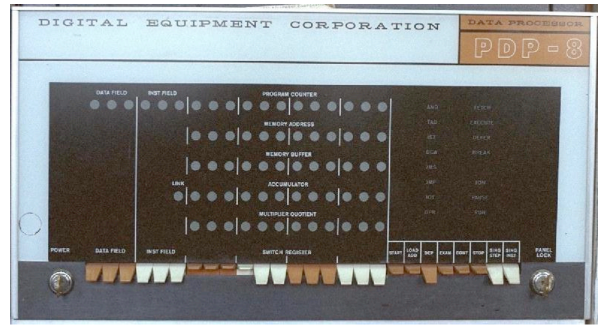
Lecture 8: The TOY Machine



What is TOY?

An imaginary machine similar to:

- Ancient computers.



What is TOY?

An imaginary machine similar to:

- Ancient computers.
- Today's microprocessors.

Pentium Celeron

What is TOY?

An imaginary machine similar to:

- Ancient computers.
- Today's microprocessors.

Why study TOY?

- Machine language programming.
  - how do Java programs relate to computer?
  - key to understanding Java references (and C pointers)
- Computer architecture.
  - how is a computer put together?
  - how does it work?
- Optimized for understandability, not cost or performance.

Inside the Box

Switches. Input data and programs.

Lights. View data.

Memory.

- Stores data and programs.
- 256 "words." (16 bits each)
- FF for stdin / stdout.

Program counter (PC).

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

Registers.

- Fastest form of storage.
- Scratch space during computation.
- 16 registers. (16 bits each)
- Register 0 is always 0.

Arithmetic-logic unit (ALU). Manipulate data stored in registers.

Standard input, standard output. Interact with outside world.

Data and Programs Are 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

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

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

$6375_{10} = +2^{12} +2^{11} +2^7 +2^6 +2^5 +2^2 +2^1 +2^0$

$= 4096 +2048 +128 +64 +32 +4 +2 +1$

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			

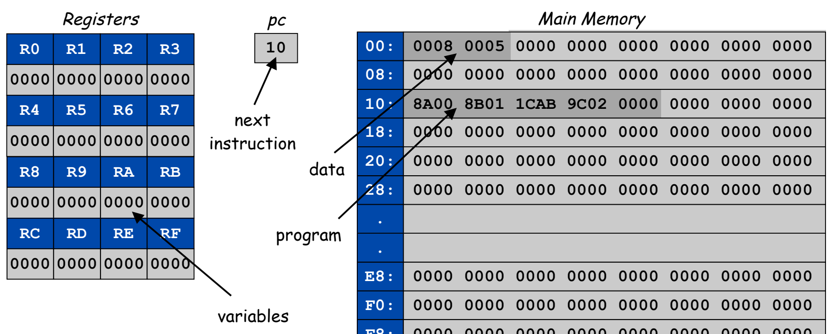
$6375_{10} = 1 \times 16^3 + 8 \times 16^2 + 14 \times 16^1 + 7 \times 16^0$

$= 4096 + 2048 + 224 + 7$

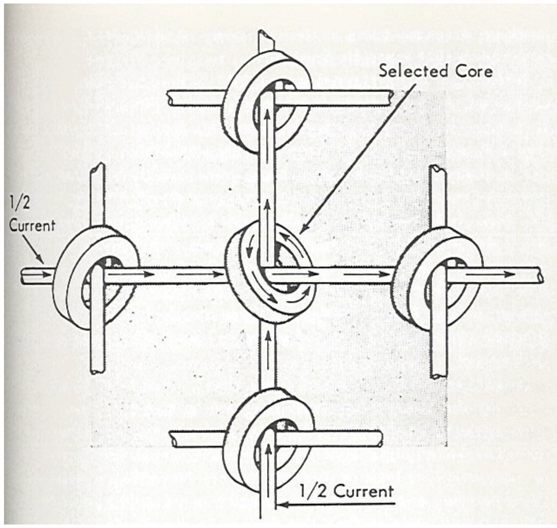
Machine "Core" Dump

Machine contents at a particular place and time.

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



Why do they call it "core"?



Reference: http://www.columbia.edu/acis/history/core.html

A Sample Program

A sample program.

- Adds 8 + 5 = D.

RA	RB	RC
0000	0000	0000

Registers

pc
10

```
00: 0008    8             add.toy
01: 0005    5
02: 0000    0

10: 8A00    RA ← mem[00]
11: 8B01    RB ← mem[01]
12: 1CAB    RC ← RA + RB
13: 9C02    mem[02] ← RC
14: 0000    halt
```

Memory

Since PC = 10, machine interprets 8A00 as an instruction.

Load

Load. (opcode 8)

- Loads the contents of some memory location into a register.
- 8A00 means load the contents of memory cell 00 into register A.

RA	RB	RC
0000	0000	0000

Registers

pc
10

```
00: 0008    8             add.toy
01: 0005    5
02: 0000    0

10: 8A00    RA ← mem[00]
11: 8B01    RB ← mem[01]
12: 1CAB    RC ← RA + RB
13: 9C02    mem[02] ← RC
14: 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	0	0	0
8_{16}				A_{16}				00_{16}							
opcode				dest d				addr							

Load

Load. (opcode 8)

- Loads the contents of some memory location into a register.
- 8B01 means load the contents of memory cell 01 into register B.

RA	RB	RC
0008	0000	0000

Registers

pc
11

```
00: 0008    8             add.toy
01: 0005    5
02: 0000    0

10: 8A00    RA ← mem[00]
11: 8B01    RB ← mem[01]
12: 1CAB    RC ← RA + RB
13: 9C02    mem[02] ← RC
14: 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	1	0	0	0	0	0	0	0	1
8_{16}				B_{16}				01_{16}							
opcode				dest d				addr							

Add

Add. (opcode 1)

- Add contents of two registers and store sum in a third.
- 1CAB means add the contents of registers A and B and put the result into register C.

RA	RB	RC
0008	0005	0000

Registers

pc
12

add.toy

```
00: 0008   8
01: 0005   5
02: 0000   0

10: 8A00   RA ← mem[00]
11: 8B01   RB ← mem[01]
12: 1CAB   RC ← RA + RB
13: 9C02   mem[02] ← RC
14: 0000   halt
```

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_{16}				C_{16}				A_{16}				B_{16}			
opcode				dest d				source s				source t			

Store

Store. (opcode 9)

- Stores the contents of some register into a memory cell.
- 9C02 means store the contents of register A into memory cell 02.

RA	RB	RC
0008	0005	000D

Registers

pc
13

add.toy

```
00: 0008   8
01: 0005   5
02: 0000   0

10: 8A00   RA ← mem[00]
11: 8B01   RB ← mem[01]
12: 1CAB   RC ← RA + RB
13: 9C02   mem[02] ← RC
14: 0000   halt
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0
9_{16}				C_{16}				02_{16}							
opcode				dest d				addr							

Halt

Halt. (opcode 0)

- Stop the machine.

RA	RB	RC
0008	0005	000D

Registers

pc
14

add.toy

```
00: 0008   8
01: 0005   5
02: 000D   D

10: 8A00   RA ← mem[00]
11: 8B01   RB ← mem[01]
12: 1CAB   RC ← RA + RB
13: 9C02   mem[02] ← RC
14: 0000   halt
```


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

TOY Instruction Set Architecture

TOY instruction set architecture (ISA).
- Interface that specifies behavior of machine.
- 16 register, 256 words of main memory, 16-bit words.
- 16 instructions.

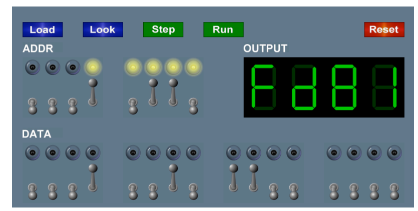
Each instruction consists of 16 bits.
- Bits 12-15 encode one of 16 instruction types or opcodes.
- Bits 8-11 encode destination register `d`.
- Bits 0-7 encode:
  - Format 1: source registers `s` and `t`
  - Format 2: 8-bit memory address or constant

	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
Format 1	opcode				dest d				source s				source t			
Format 2	opcode				dest d				addr							

Interfacing with the TOY Machine

To enter a program or data:
- Set 8 memory address switches.
- Set 16 data switches.
- Press LOAD.
  - data written into addressed word of memory

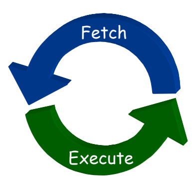
To view the results of a program:
- Set 8 memory address switches.
- Press LOOK: contents of addressed word appears in lights.



Using the TOY Machine: Run

To run the program:
- Set 8 memory address switches to address of first instruction.
- Press LOOK to set PC to first instruction.
- Press RUN button to repeat fetch-execute cycle until halt opcode.

Fetch-execute cycle.
- FETCH: get instruction from memory.
- EXECUTE: update PC, move data to or from memory and registers, perform calculations.



Programming in TOY

Hello, World. Add two numbers.

To harness the power of TOY, need loops and conditionals.
- Manipulate PC to control program flow.

Branch if zero. (opcode C)
- Changes PC depending of value of some register.
- Used to implement: `for`, `while`, `if-else`.

Branch if positive. (opcode D)
- Analogous.

An Example: Multiplication

Multiply.

- No direct support in TOY hardware.
- Load in integers a and b, and store c = a × b.
- Brute-force algorithm:
  - initialize c = 0
  - add b to c, a times

```
int a = 3;
int b = 9;
int c = 0;

while (a != 0) {
   c = c + b;
   a = a - 1;
}
```

Java

Issues ignored: slow, overflow, negative numbers.

Multiply

```
0A: 0003    3  ⇐ inputs
0B: 0009    9
0C: 0000    0  ⇐ output

0D: 0000    0  ⇐ constants
0E: 0001    1

10: 8A0A    RA ← mem[0A]              a
11: 8B0B    RB ← mem[0B]              b
12: 8C0D    RC ← mem[0D]              c = 0

13: 810E    R1 ← mem[0E]              always 1

14: CA18    if (RA == 0) pc ← 18      while (a != 0) {
15: 1CCB    RC ← RC + RB                 c = c + b
16: 2AA1    RA ← RA - R1                 a = a - 1
17: C014    pc ← 14                   }

18: 9C0C    mem[0C] ← RC
19: 0000    halt
```

loop

multiply.toy

Step-By-Step Trace

		R1	RA	RB	RC
10: 8A0A	RA ← mem[0A]		0003		
11: 8B0B	RB ← mem[0B]			0009	
12: 8C0D	RC ← mem[0D]				0000
13: 810E	R1 ← mem[0E]	0001			
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				0009
16: 2AA1	RA ← RA - R1		0002		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				0012
16: 2AA1	RA ← RA - R1		0001		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				001B
16: 2AA1	RA ← RA - R1		0000		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
18: 9C0C	mem[0C] ← RC				
19: 0000	halt				

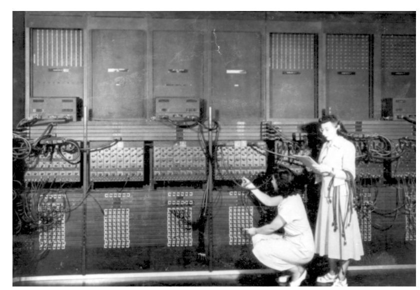
multiply.toy

A Little History

Electronic Numerical Integrator and Calculator (ENIAC).

- First widely known general purpose electronic computer.
- 30 tons, 30 x 50 x 8.5 ft, 17,468 vacuum tubes, 300 multiply/sec.
- Conditional jumps, programmable.
- Programming: change switches and cable connections.
- Data: enter numbers using punch cards.

John Mauchly (left) and J. Presper Eckert (right)
http://cs.swau.edu/~durkin/articles/history_computing.html



ENIAC, Ester Gerston (left), Gloria Gordon (right)
US Army photo: http://ftp.arl.mil/ftp/historic-computers

Basic Characteristics of TOY Machine

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 without rewiring.

Outgrowth of Alan Turing's work. (stay tuned)

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

John von Neumann

Maurice Wilkes (left)
EDSAC (right)

Harvard vs. Princeton

Harvard architecture.
- Separate program and data memories.
- Can't load game from disk (data) and execute (program).
- Used in some microcontrollers.

Von Neumann architecture.
- Program and data stored in same memory.
- Used in almost all computers.

Q. What's the difference between Harvard and Princeton?
A. At Princeton, data and programs are the same.