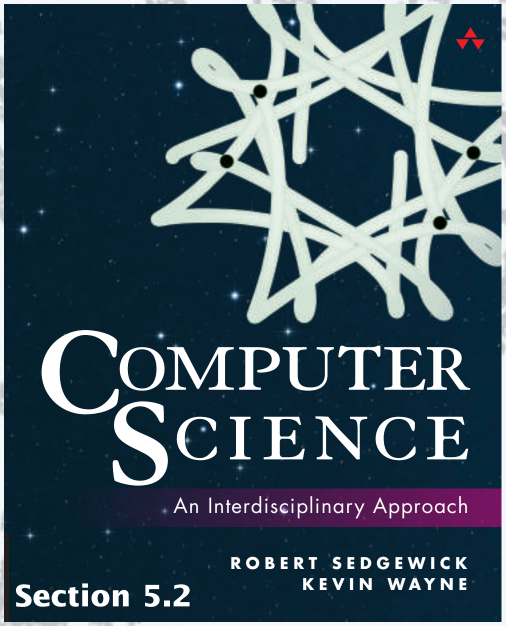
COMPUTER SCIENCE
SEDGEWICK / WAYNE

PART II: ALGORITHMS, MACHINES, and THEORY



http://introcs.cs.princeton.edu

17. A Computing Machine

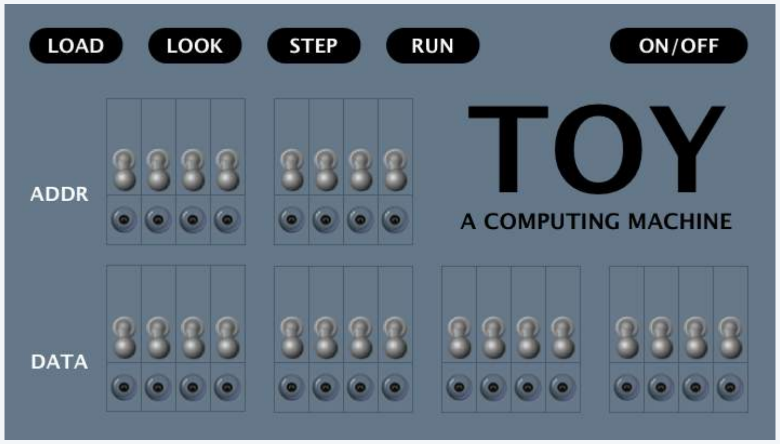
17. A Computing Machine

- **Overview**
- Data types
- Instructions
- Operating the machine
- Machine language programming

A TOY computing machine

TOY is an imaginary machine similar to:

- Ancient computers.
- Today's smartphone processors.
- Countless other devices designed and built over the past 50 years.



Smartphone processor, 2010s

PDP-8, 1970s

Reasons to study TOY

Prepare to learn about computer architecture

- How does your computer's processor work?
- What are its basic components?
- How do they interact?

Learn about machine-language programming.

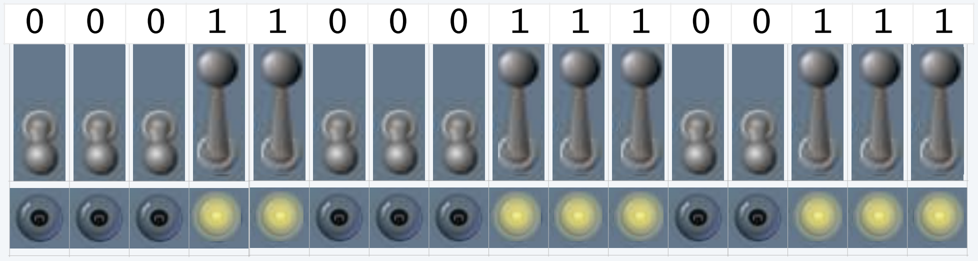
- How do Java programs relate to your computer?
- Key to understanding Java references.
- Intellectual challenge of a new programming regime.
- Still necessary in some modern applications.

multimedia, computer games, embedded devices, scientific computing,...

Learn fundamental abstractions that have informed processor design for decades.

Bits and words

Everything in TOY is encoded with a sequence of *bits* (value 0 or 1).
- Why? Easy to represent two states (on and off) in real world.
- Bits are organized in 16-bit sequences called *words.*



More convenient for humans: *hexadecimal notation* (base 16)
- 4 *hex digits* in each word.
- Convert to and from binary 4 bits at a time.

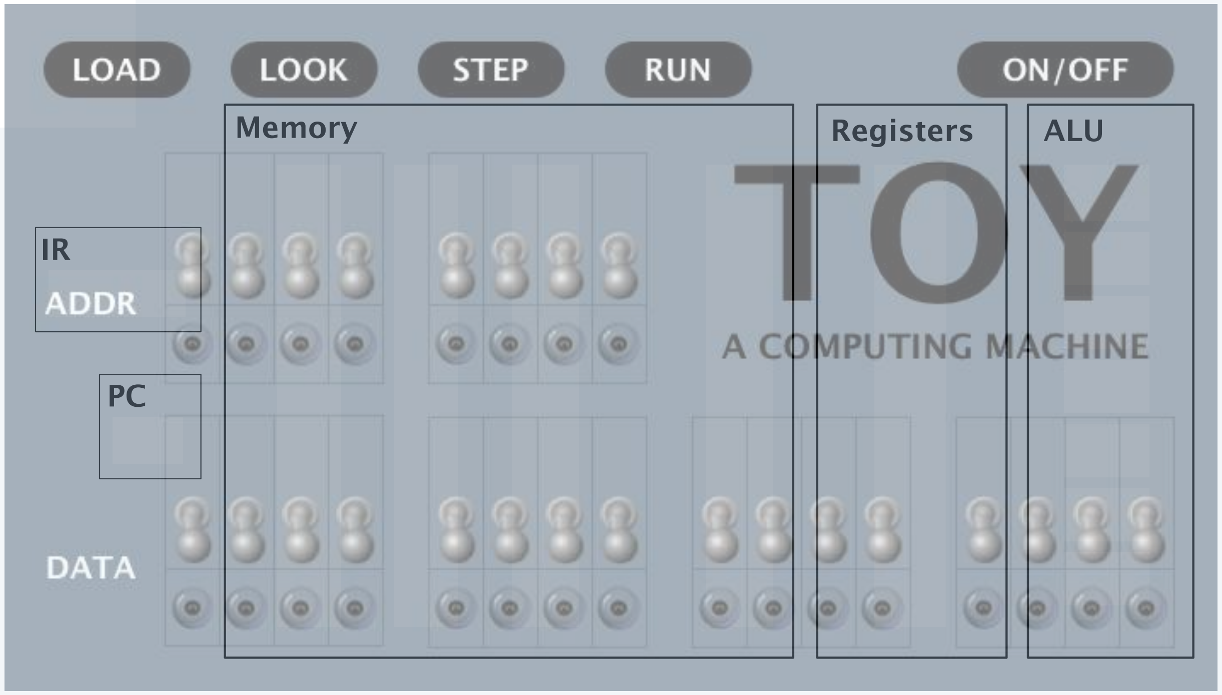
0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1
1				8				E				7			

binary	*hex*
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Inside the box

Components of TOY machine

- Memory
- Registers
- Arithmetic and logic unit (ALU)
- Program counter (PC)
- Instruction register (IR)



Memory

Holds data and instructions

- 256 words
- 16 bits in each word
- Connected to registers
- Words are *addressable*

Use *hexadecimal* for addresses

- Number words from 00 to FF
- *Think in hexadecimal*
- Use array notation
- Example: `M[2A]` = **C 0 2 4**

Memory

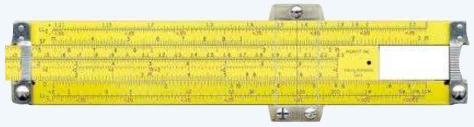
00	0 0 0 0	10	8 A 0 1	20	7 1 0 1		F0	F 0 F 0
01	F F F E	11	8 B 0 2	21	8 A F F		F1	0 5 0 5
02	0 0 0 D	12	1 C A B	22	7 6 8 0		F2	0 0 0 D
03	0 0 0 3	13	9 C 0 3	23	7 B 0 0		F3	1 0 0 0
04	0 0 0 1	14	0 0 0 1	24	C A 2 B		F4	0 1 0 1
05	0 0 0 0	15	0 0 1 0	25	8 C F F		F5	0 0 1 0
06	0 0 0 0	16	0 1 0 0	26	1 5 6 B		F6	0 0 0 1
07	0 0 0 0	17	1 0 0 0	27	B C 0 5	. . .	F7	0 0 1 0
08	0 0 0 0	18	0 1 0 0	28	2 A A 1		F8	0 1 0 0
09	0 0 0 0	19	0 0 1 0	29	2 B B 1		F9	1 0 0 0
0A	0 0 0 0	1A	0 0 0 1	2A	C 0 2 4		FA	0 1 0 0
0B	0 0 0 0	1B	0 0 1 0	2B	0 0 0 0		FB	0 0 1 0
0C	0 0 0 0	1C	0 1 0 0	2C	0 0 0 0		FC	0 0 0 1
0D	0 0 0 0	1D	1 0 0 0	2D	0 0 0 0		FD	0 0 1 0
0E	0 0 0 0	1E	0 1 0 0	2E	0 0 0 0		FE	0 1 0 0
0F	0 0 0 0	1F	0 0 1 0	2F	0 0 0 0		FF	0 1 0 0

Table of 256 words *completely specifies* contents of memory.

Arithmetic and logic unit (ALU)

ALU

- TOY's computational engine
- A *calculator*, not a computer
- Hardware that implements *all* data-type operations
- How? Stay tuned for computer architecture lectures



Registers

Registers

- 16 words, addressable in hex from 0 to F (use names R[0] through R[F])
- Scratch space for calculations and data movement.
- Connected to memory and ALU
- *By convention,* R[0] *is always* 0. ← often simplifies code (stay tuned)

In our code, we often also keep 0001 in R[1].

Q. Why not just connect memory directly to ALU?

A. Too many different memory names (addresses).

Q. Why not just connect memory locations to one another?

A. Too many different connections.

Table of 16 words *completely specifies* contents of registers.

Registers

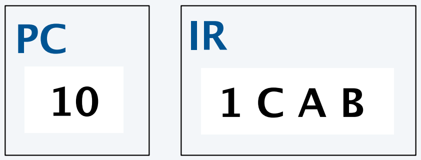
R[0]	0 0 0 0
R[1]	0 0 0 1
R[2]	F F F E
R[3]	1 C A B
R[4]	0 0 0 1
R[5]	0 0 0 0
R[6]	F A C E
R[7]	0 0 0 0
R[8]	F 0 0 1
R[9]	0 0 0 0
R[A]	0 0 0 5
R[B]	0 0 0 8
R[C]	0 0 0 D
R[D]	0 0 0 0
R[E]	0 0 0 0
R[F]	0 0 0 0

Program counter and instruction register

TOY operates by executing a sequence of instructions.

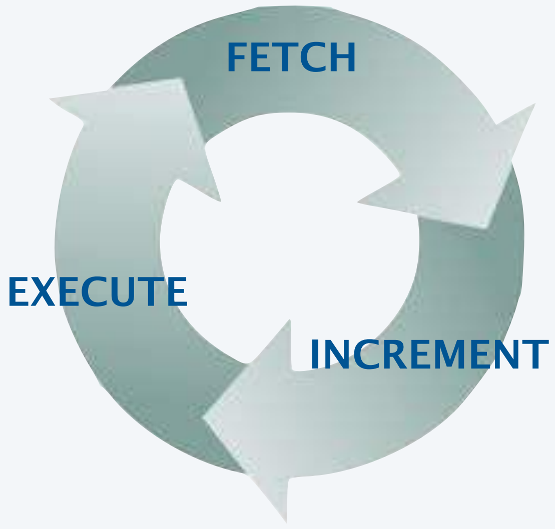
Critical abstractions in making this happen

- **Program Counter (PC)**. Memory address of next instruction.
- **Instruction Register (IR)**. Instruction being executed.



Fetch-increment-execute cycle

- Fetch: Get instruction from memory into IR.
- Increment: Update PC to point to *next* instruction.
- Execute: Move data to or from memory, change PC, or perform calculations, as specified by IR.



The state of the machine

Contents of memory, registers, and PC at a particular time

- Provide a record of what a program has done.
- Completely determines what the machine will do.

ALU and IR hold intermediate states of computation

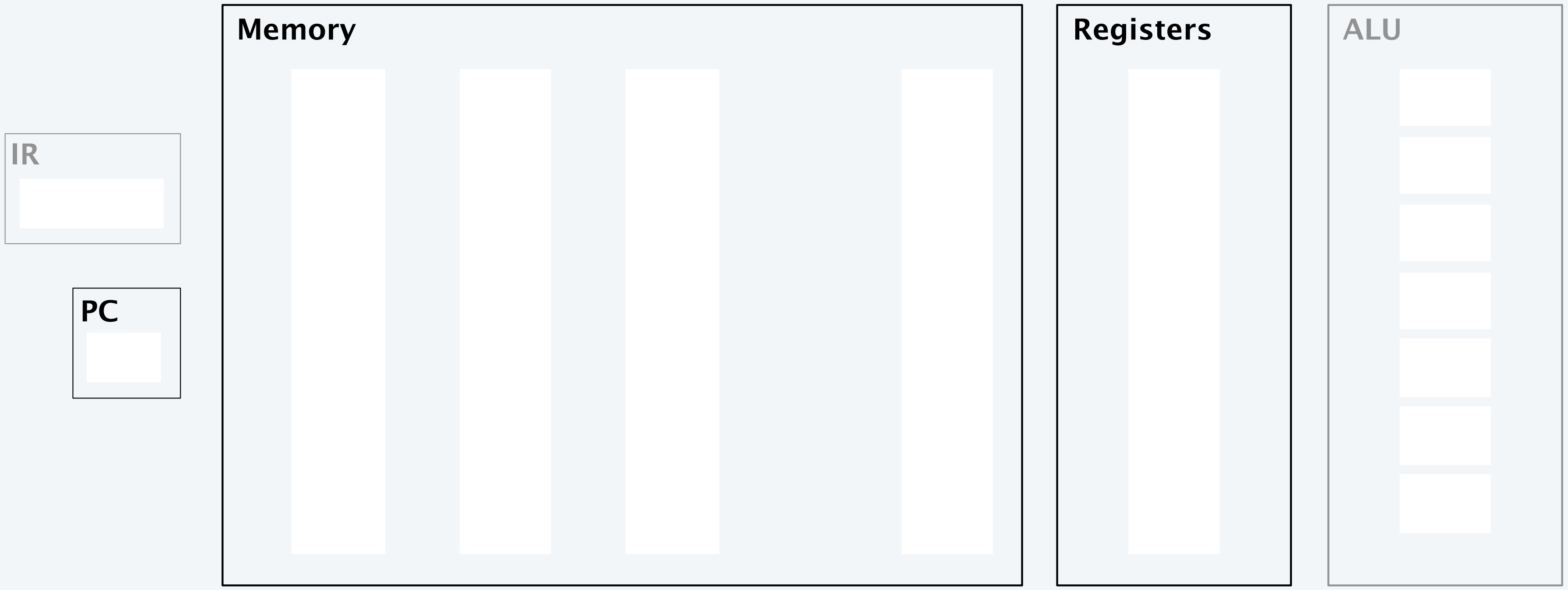


Image sources

http://pixabay.com/en/man-flashlight-helmet-detective-308611/

http://en.wikipedia.org/wiki/Marchant_calculator#/media/File:Marchant_-_Odhner_clone_1950.png

http://en.wikipedia.org/wiki/Marchant_calculator#/media/File:SCM_Marchant_calculator.jpg

http://commons.wikimedia.org/wiki/File:Calculator_casio.jpg

http://commons.wikimedia.org/wiki/File:Abacus_5.jpg

17. A Computing Machine

TOY data type

A data type is a set of values and a set of operations on those values.

TOY's data type is 16-bit *two's complement* integers.

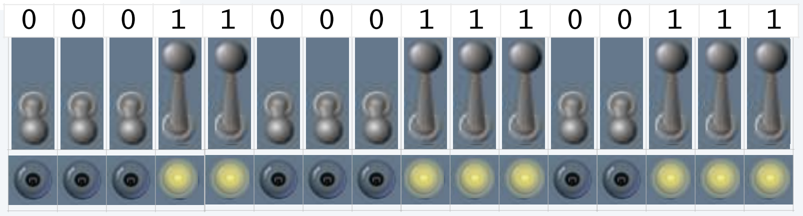
Two kinds of operations

- Arithmetic.
- Bitwise.

All other types of data must be implemented with *software*

- 32-bit and 64-bit integers.
- 32-bit and 64-bit floating point values.
- Characters and strings.
- ...

All values are represented in 16-bit words.



TOY data type (original design): Unsigned integers

Values. 0 to $2^{16} - 1$, encoded in binary (or, equivalently, hex).

Example. 6375_{10}.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binary	0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1
				2^{12}	$+2^{11}$				$+2^{7}$	$+2^{6}$	$+2^{5}$			$+2^{2}$	$+2^{1}$	$+2^{0}$

hex	1	8	E	7
	1×16^3	$+ 8 \times 16^2$	$+ 14 \times 16$	$+ 7$
	4096	+ 2048	+ 224	+ 7

Operations.
- Add.
- Subtract.
- Test if 0.

Example. 18E7 + 18E7 = 31CE

	0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1
+	0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1
=	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1	0

Warning. TOY ignores overflow.

TOY data type (better design): two's complement

Values. -2^{15} to $2^{15}-1$, encoded in *16-bit two's complement*.

includes negative integers!

Operations.

- Add.
- Subtract.
- Test if positive, negative, or 0.

16 bit two's complement

- 16-bit binary representation of x for positive x.
- 16-bit binary representation of $2^{16} - |x|$ for negative x.

Useful properties

- Leading bit (bit 15) signifies sign.
- 0000000000000000 represents zero.
- Add/subtract is *the same* as for unsigned.

decimal	*hex*	*binary*
+32,767	7FFF	0111111111111111
+32,766	7FFE	0111111111111110
+32,765	7FFD	0111111111111101
...		
+3	0003	0000000000000011
+2	0002	0000000000000010
+1	0001	0000000000000001
0	0000	0000000000000000
−1	FFFF	1111111111111111
−2	FFFE	1111111111111110
−3	FFFD	1111111111111101
...		
−32,766	8002	1000000000000010
−32,767	8001	1000000000000001
−32,768	8000	1000000000000000

slight annoyance: one extra negative value

Two's complement: conversion

To convert from decimal to two's complement

- If greater than +32,767 or less than –32,768 report error.
- Convert to 16-bit binary.
- If not negative, done.
- If negative, *flip all bits and add 1*.

Examples

$+13_{10}$	0000000000001101	000D
-13_{10}	1111111111110011	FFF3
$+256_{10}$	0000000100000000	0100
-256_{10}	1111111100000000	FF00

To convert from two's complement to decimal

- If sign bit is 1, *flip all bits and add 1* and output minus sign.
- Convert to decimal.

Examples

0001	0000000000000001	1_{10}
FFFF	1111111111111111	-1_{10}
FF0D	1111111100001101	-243_{10}
00F3	0000000011110011	$+243_{10}$

To add/subtract

- Use same rules as for unsigned binary.
- (Still) ignore overflow.

Example

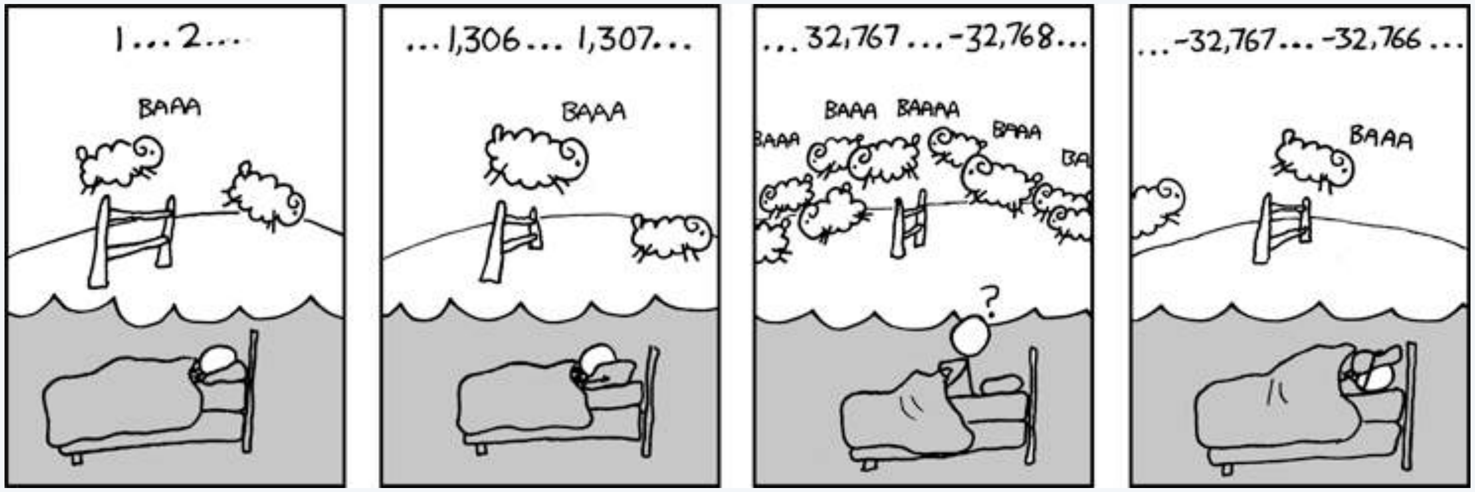
-256_{10}	1111111100000000	FF00
$+13_{10}$	+0000000000001011	+000D
$= -243_{10}$	=1111111100001101	=FF0D

Overflow in two's complement

$32{,}767_{10} = 2^{15} - 1$	0111111111111111	7FFF	
$+1$	+ 0000000000000001	+ 0001	
	= 1000000000000000	= 8000	$= -2^{15} = -32{,}768_{10}$

largest (positive) number

smallest (negative) number



http://xkcd.com/571/

TOY data type: Bitwise operations

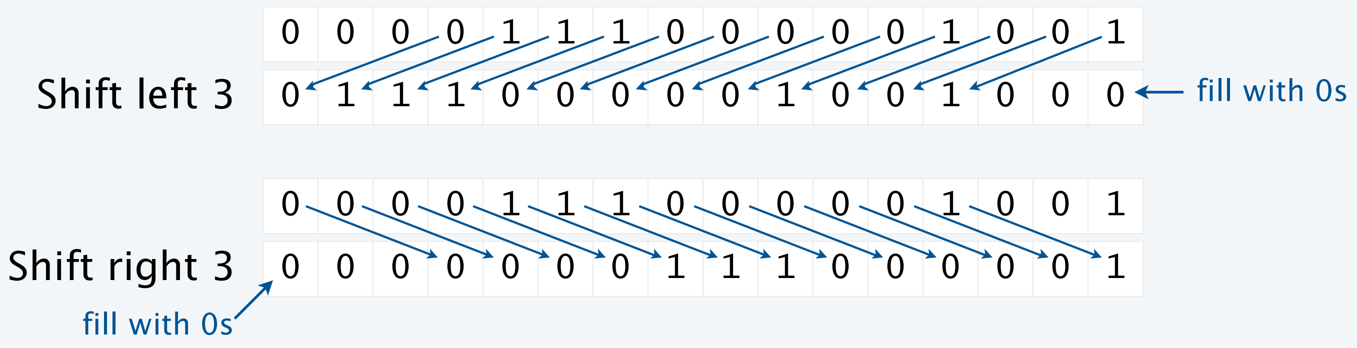
Operations
- Bitwise AND.
- Bitwise XOR.
- Shift left.
- Shift right.

	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0
AND	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	1
=	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0

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

	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0
XOR	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	1
=	0	1	0	0	0	1	1	0	0	1	0	0	1	1	0	1

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



Special note: Shift left/right operations also implement multiply/divide by powers of 2 for integers.

shift right fills with 1s if leading bit is 1

Image sources

http://pixabay.com/en/network-media-binary-computer-65923/

https://xkcd.com/571/

17. A Computing Machine

- Overview
- Data types
- **Instructions**
- Operating the machine
- Machine language programming

TOY instructions

ANY 16-bit (4 hex digit) value defines a TOY instruction.

First hex digit specifies which instruction.

Each instruction changes machine state in a well-defined way.

category	*opcodes*	*implements*	*changes*
operations	**1 2 3 4 5 6**	data-type operations	registers
data movement	**7 8 9 A B**	data moves between registers and memory	registers, memory
flow of control	**0 C D E F**	conditionals, loops, and functions	PC

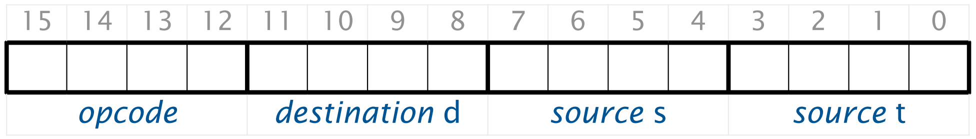
opcode	*instruction*
0	halt
1	add
2	subtract
3	bitwise and
4	bitwise xor
5	shift left
6	shift right
7	load address
8	load
9	store
A	load indirect
B	store indirect
C	branch if zero
D	branch if positive
E	jump register
F	jump and link

Encoding instructions

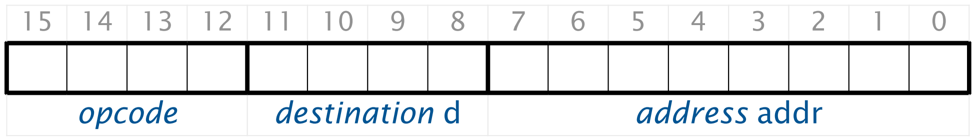
ANY 16-bit (4 hex digit) value defines a TOY instruction.

Two different instruction formats

- Type RR: Opcode and 3 registers.



- Type A: Opcode, 1 register, and 1 memory address.



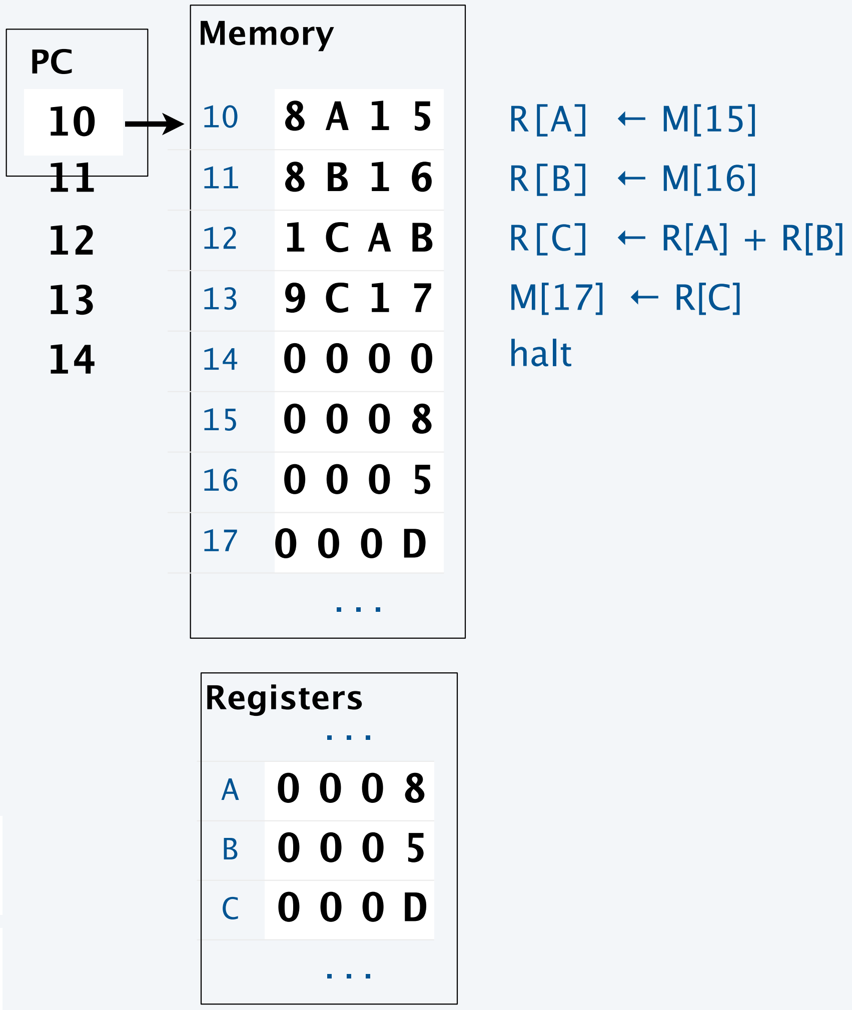
Examples

1CAB	*Add* `R[A]` *to* `R[B]` *and put result in* `R[C]`.
8A15	*Load into* `R[A]` *data from* `M[15]`.

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

A TOY program

Add two integers
- Load operands from memory into registers.
- Add the registers.
- Put result in memory.



Memory		
10	8 A 1 5	R[A] ← M[15]
11	8 B 1 6	R[B] ← M[16]
12	1 C A B	R[C] ← R[A] + R[B]
13	9 C 1 7	M[17] ← R[C]
14	0 0 0 0	halt
15	0 0 0 8	
16	0 0 0 5	
17	0 0 0 D	
	. . .	

Registers	
	. . .
A	0 0 0 8
B	0 0 0 5
C	0 0 0 D
	. . .

Load into R[A] data from M[15]
Load into R[B] data from M[16]
Add R[A] and R[B] and put result into R[C]
Store R[C] into M[17]
Halt

Q. How can you tell whether a word is an instruction?

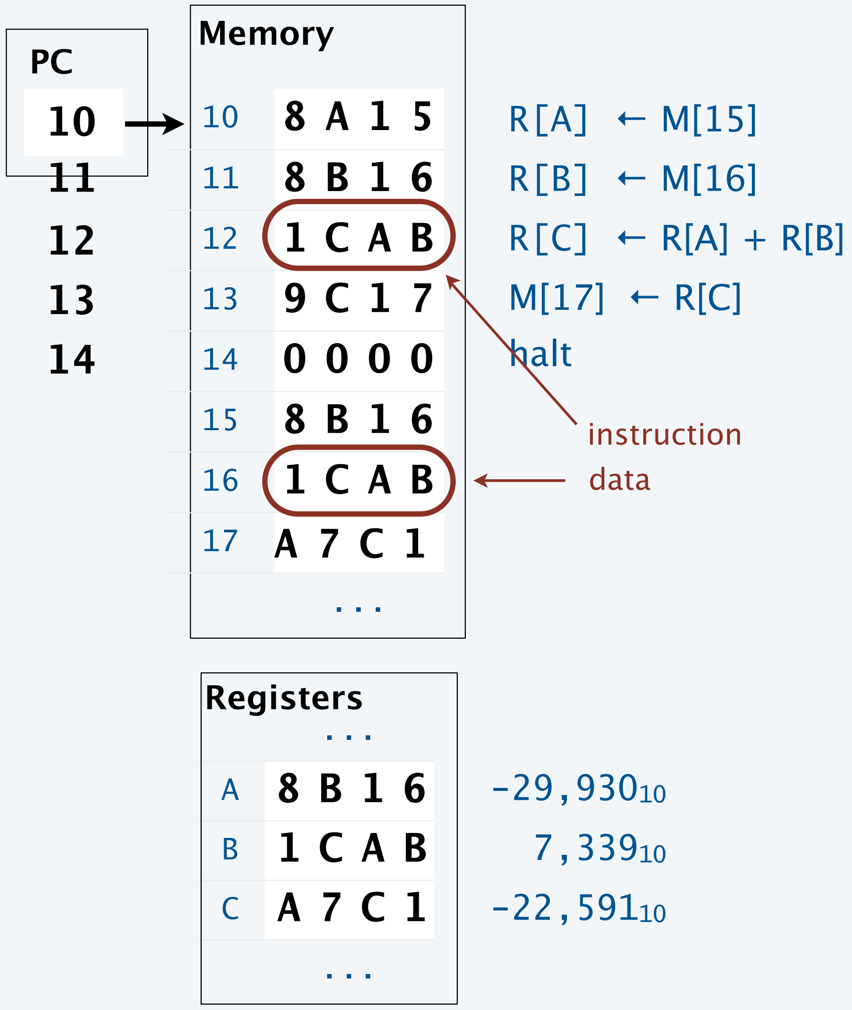
A. If the PC has its address, it *is* an instruction!

Same program with different data

Add two integers

- Load operands from memory into registers.
- Add the registers.
- Put result in memory.

Load into R[A] data from M[15]
Load into R[B] data from M[16]
Add R[A] and R[B] and put result into R[C]
Store R[C] into M[17]
Halt



Q. How can you tell whether a word is data ?

A. If it is added to another word, it *is* data !

COMPUTER SCIENCE
SEDGEWICK/WAYNE

CS.17.C.MachineI.Instructions

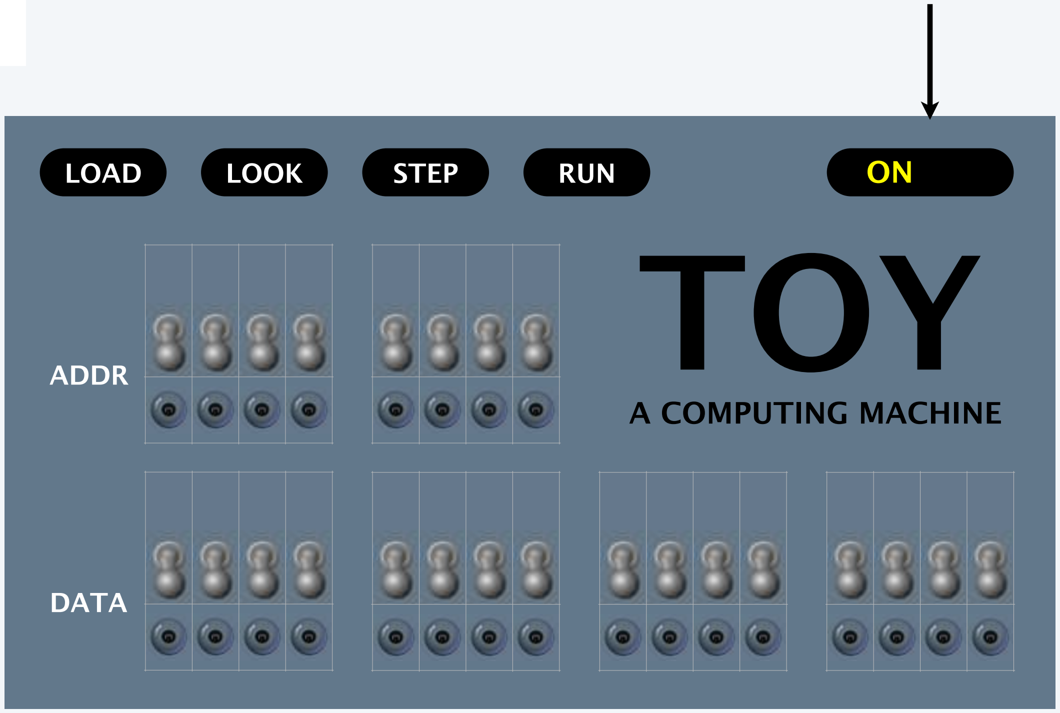
17. A Computing Machine

Outside the box

User interface

- Switches.
- Lights.
- Control Buttons.

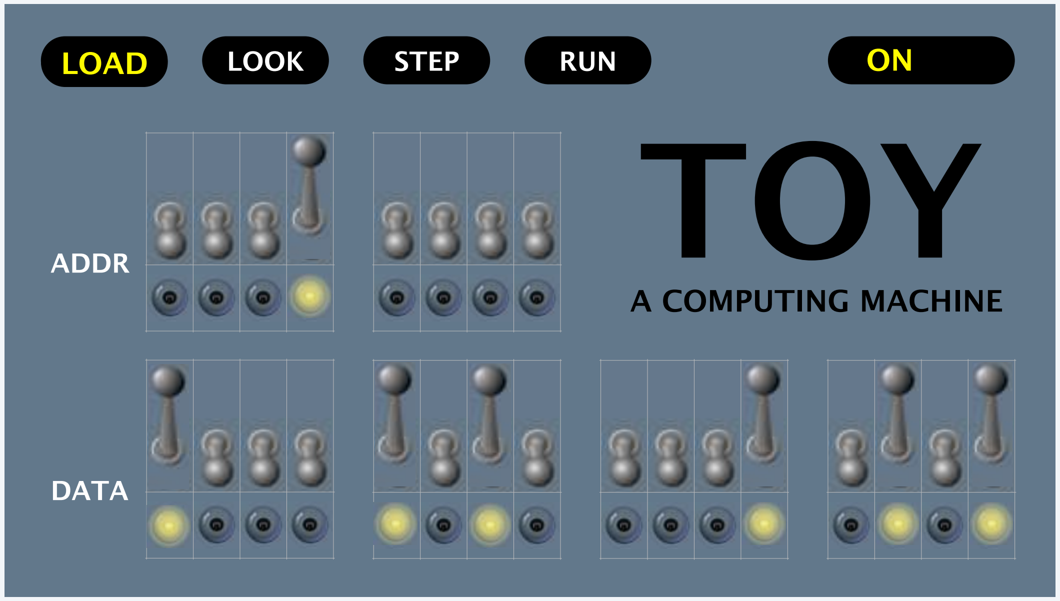
First step: Turn on the machine!

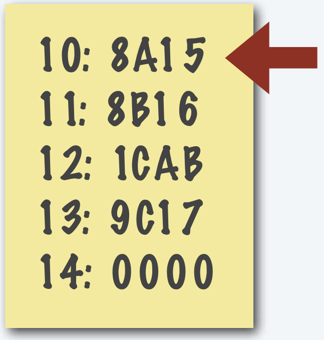


Loading a program into memory

To load an instruction

- Set 8 memory address switches.
- Set 16 data switches to instruction encoding.
- Press LOAD to load instruction from switches into addressed memory word.

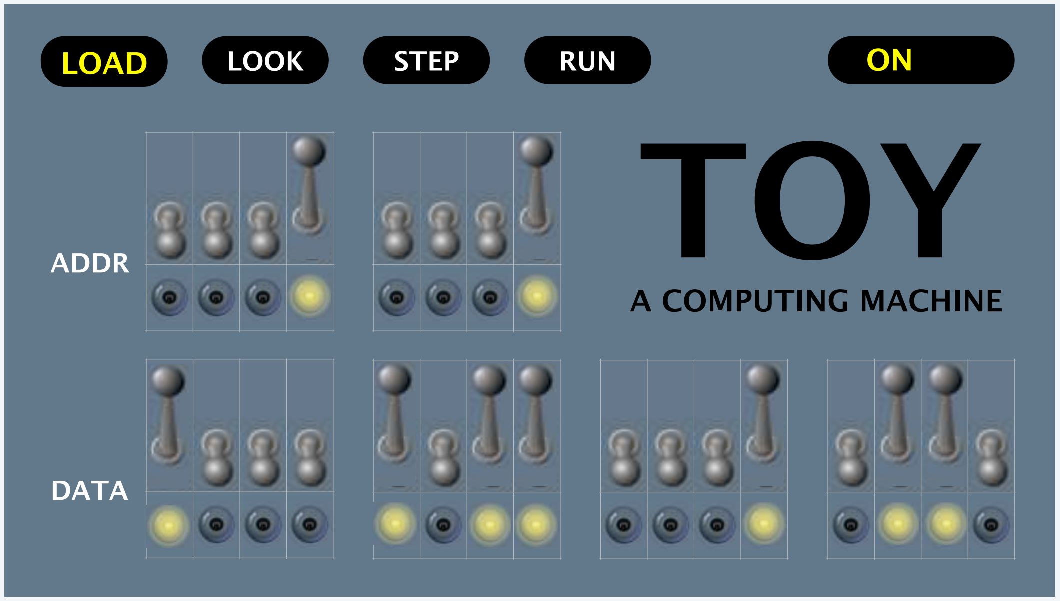


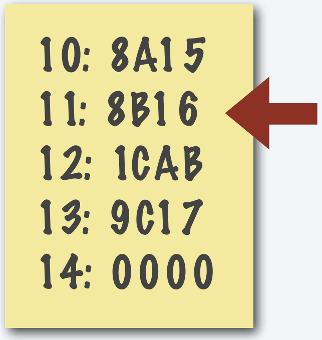


Loading instructions into memory

To load an instruction

- Set 8 memory address switches.
- Set 16 data switches to instruction encoding.
- Press LOAD to load instruction from switches into addressed memory word.

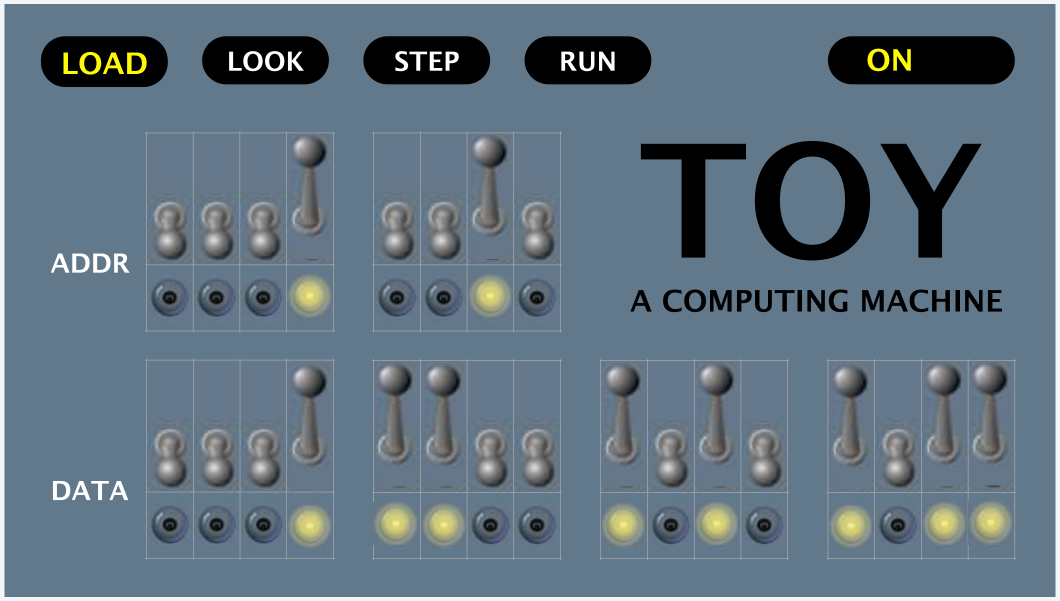


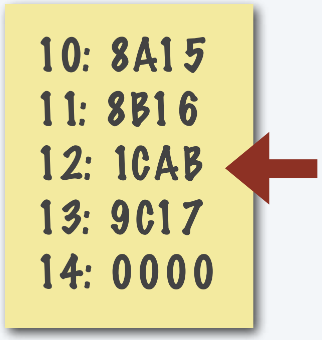

10: 8A15
11: 8B16
12: 1CAB
13: 9C17
14: 0000

Loading instructions into memory

To load an instruction

- Set 8 memory address switches.
- Set 16 data switches to instruction encoding.
- Press LOAD to load instruction from switches into addressed memory word.

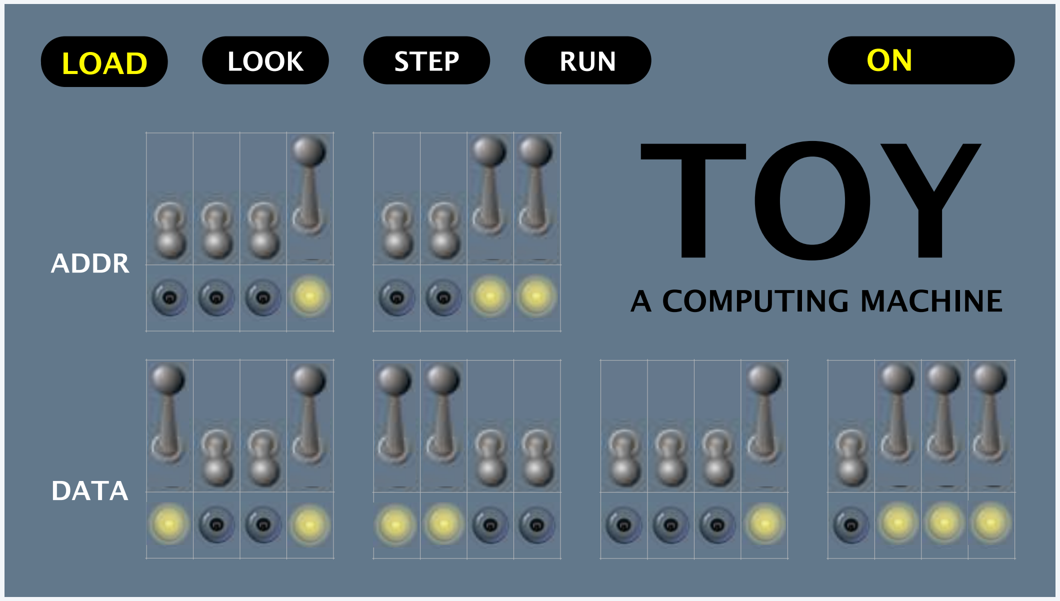


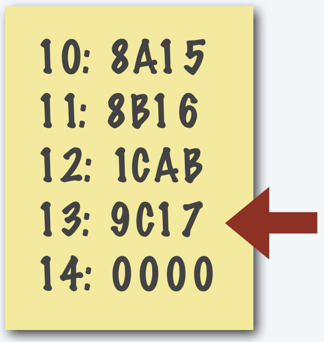

10: 8A15
11: 8B16
12: 1CAB
13: 9C17
14: 0000

Loading instructions into memory

To load an instruction

- Set 8 memory address switches.
- Set 16 data switches to instruction encoding.
- Press LOAD to load instruction from switches into addressed memory word.

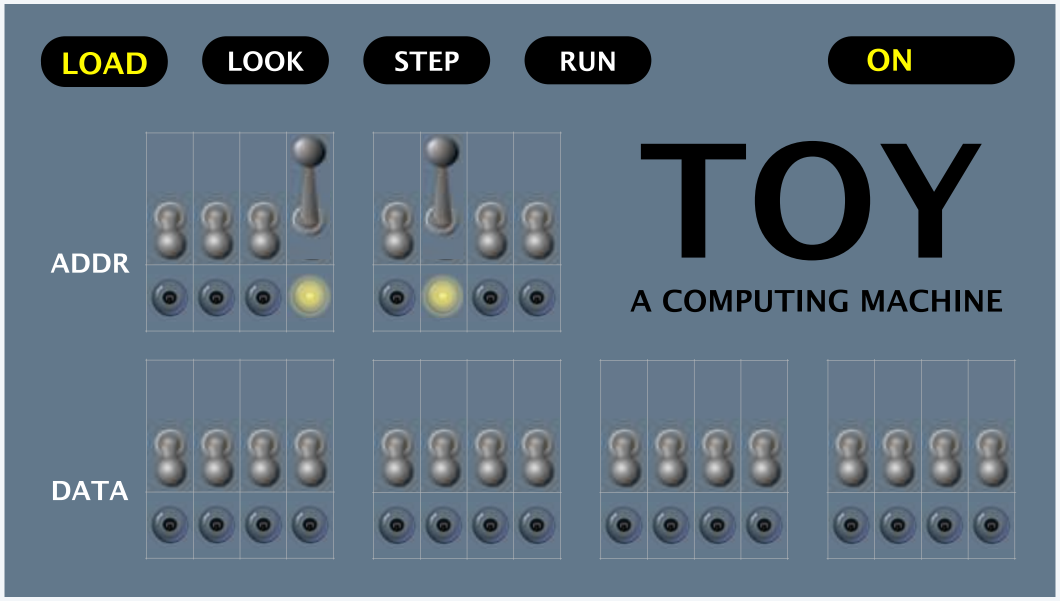


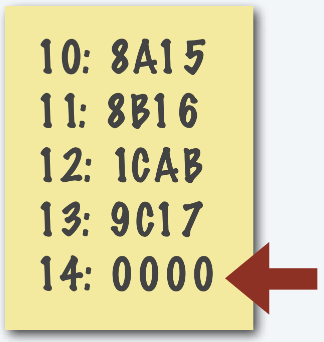


Loading instructions into memory

To load an instruction

- Set 8 memory address switches.
- Set 16 data switches to instruction encoding.
- Press LOAD to load instruction from switches into addressed memory word.

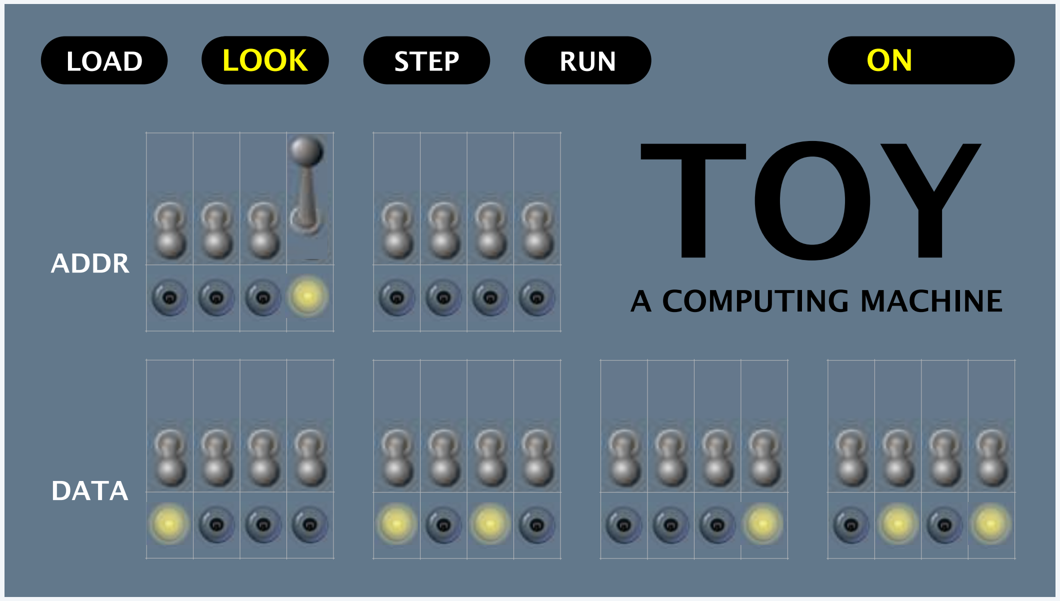


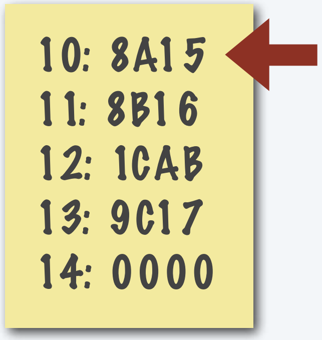


Looking at what's in the memory

To double check that you loaded the data correctly

- Set 8 memory address switches.
- Press LOOK to examine the addressed memory word.

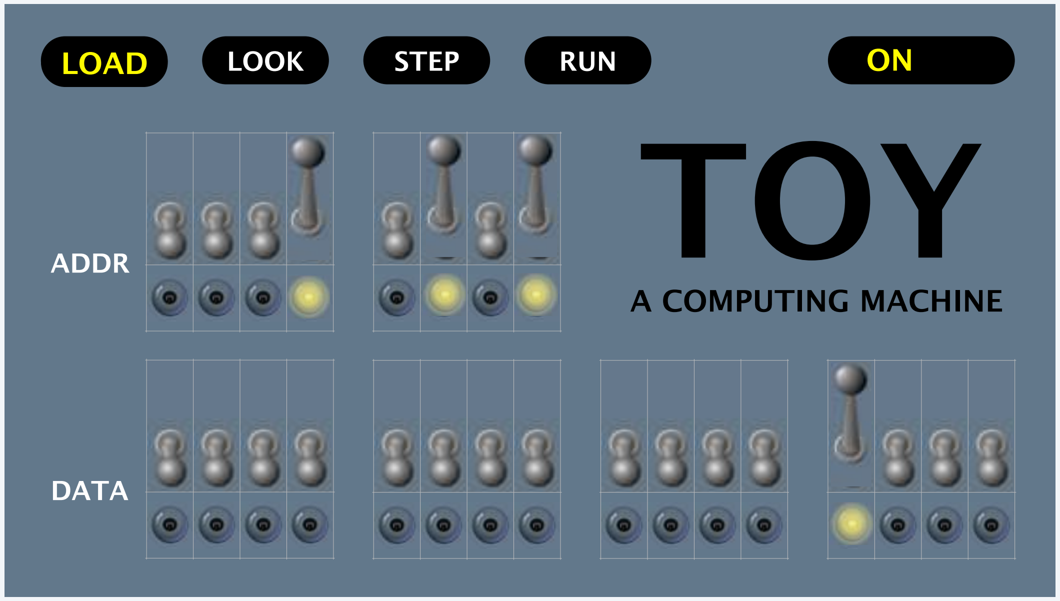


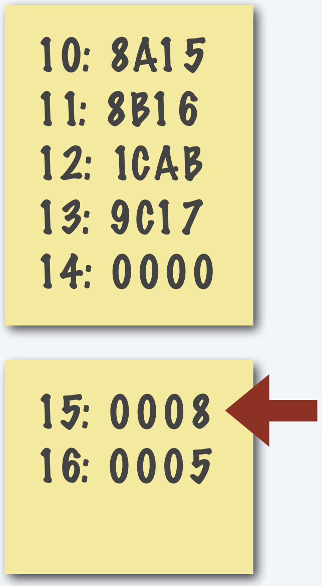


Loading data into memory

To load data, use the *same* procedure as for instructions

- Set 8 memory address switches.
- Set 16 data switches to *data* encoding.
- Press LOAD to load *data* from switches into addressed memory word.

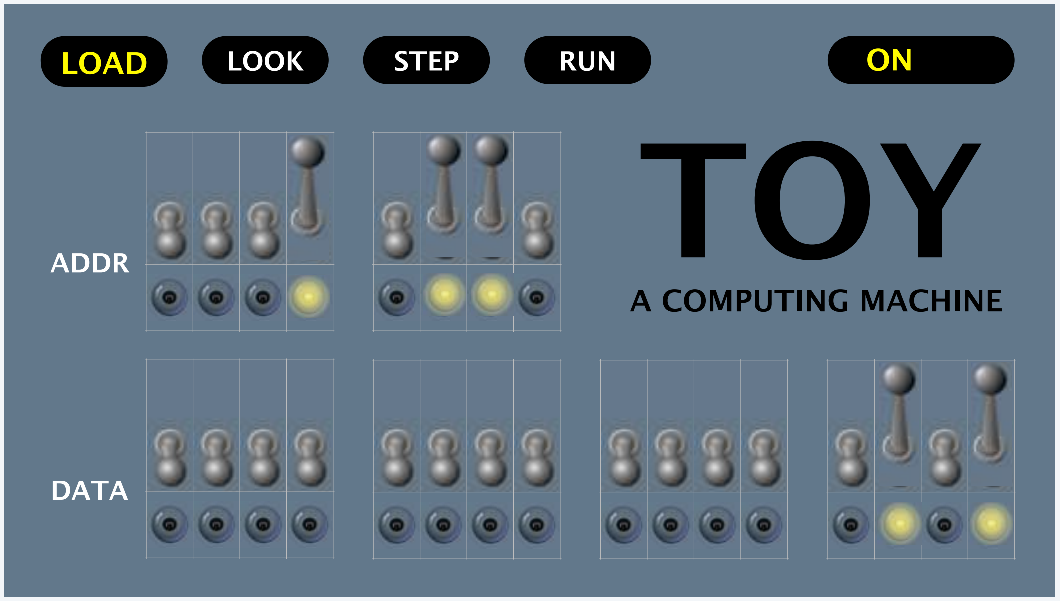


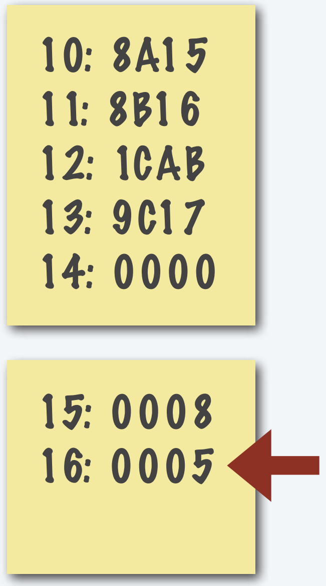


Loading data into memory

To load data, use the *same* procedure as for instructions

- Set 8 memory address switches.
- Set 16 data switches to *data* encoding.
- Press LOAD to load *data* from switches into addressed memory word.



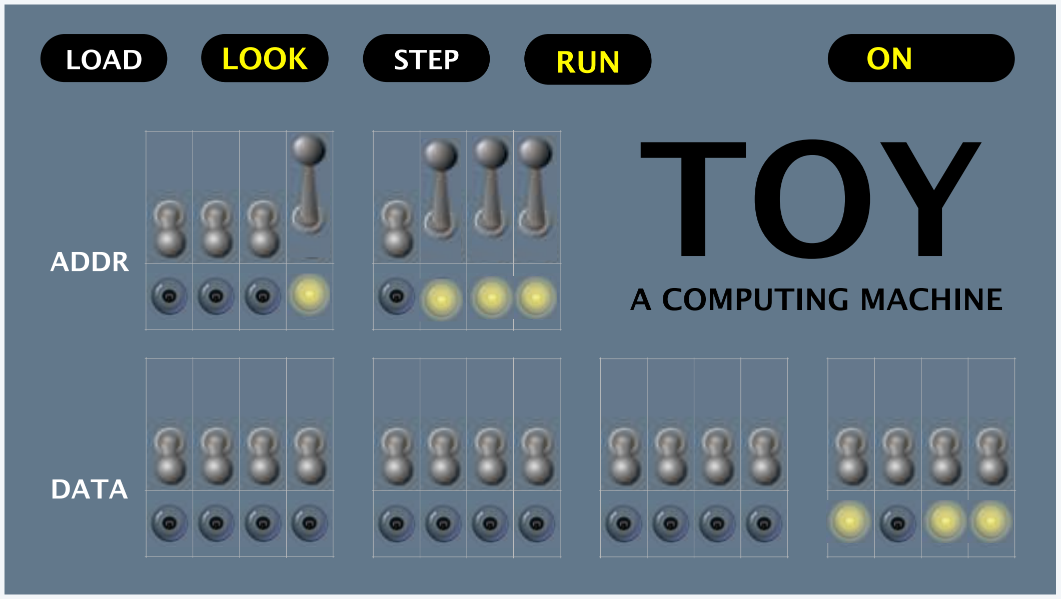


Running a program

To run a program, set the address switches to the address of first instruction and press RUN.

[data lights may flash, but all (and RUN light) go off when HALT instruction is reached]

To see the output, set the address switches to the address of expected result and press LOOK.



10: 8A15
11: 8B16
12: 1CAB
13: 9C17
14: 0000

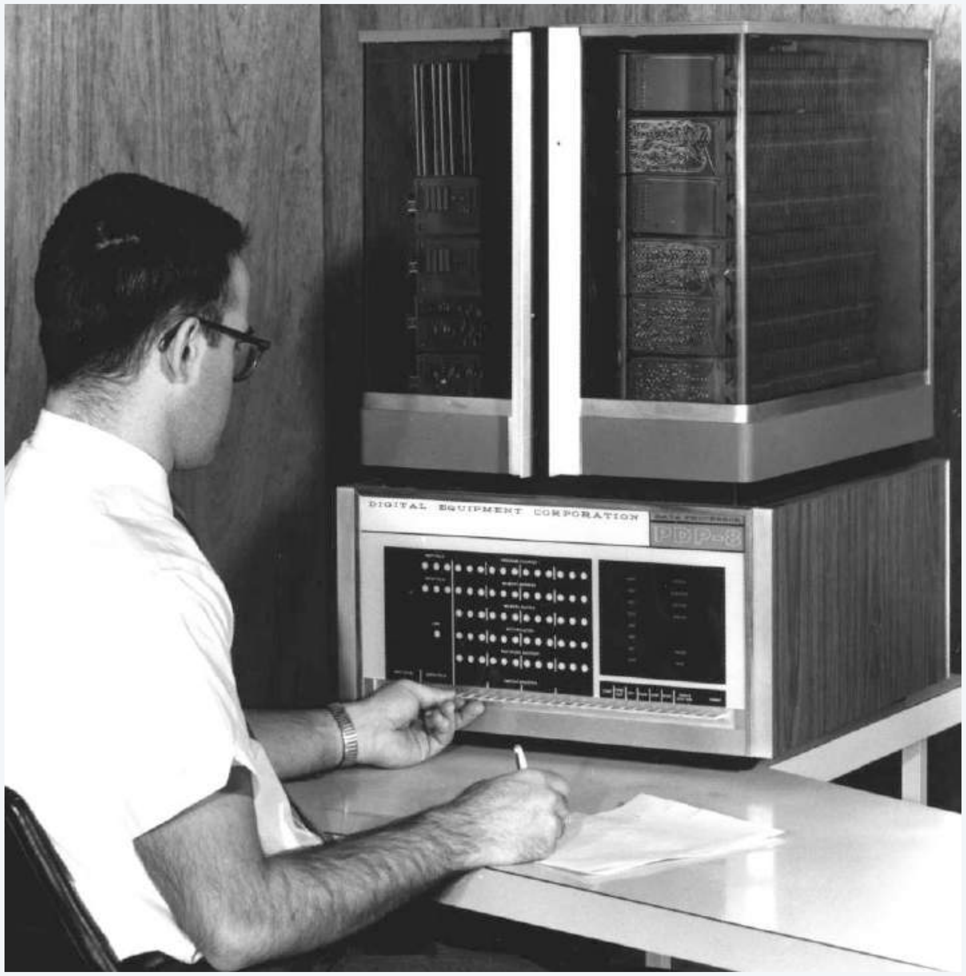
15: 0008
16: 0005
17: 000D

To run the program again, enter different data and press RUN again.

Switches and lights

Q. Did people really program this way?

A. Yes!



COMPUTER SCIENCE
S E D G E W I C K / W A Y N E

CS.11.D.MachineI.Operating

17. A Computing Machine

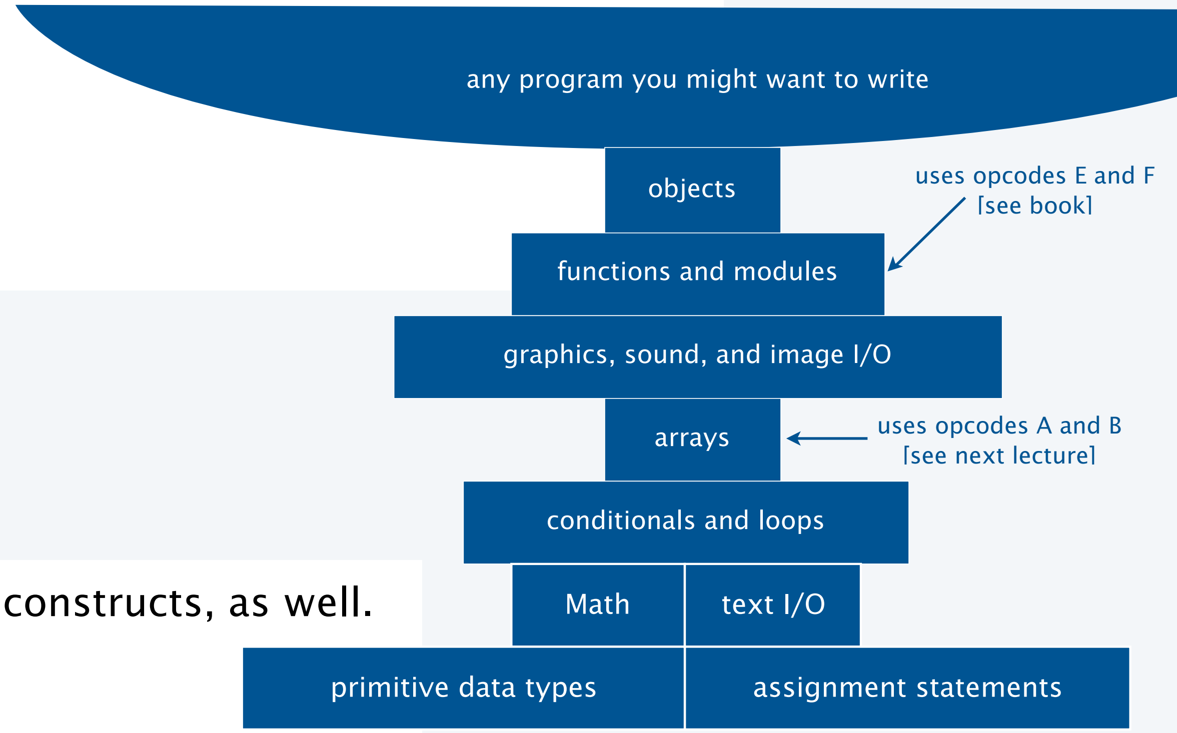
Machine language programming

TOY instructions support the same basic programming constructs as Java.

- Primitive data types.
- Assignment statements.
- Conditionals and loops.
- Arrays (next lecture).
- Standard input and output (next).

and can support advanced programming constructs, as well.

- Functions and libraries (see text).
- Linked structures (see text).


any program you might want to write
objects
uses opcodes E and F [see book]
functions and modules
graphics, sound, and image I/O
arrays
uses opcodes A and B [see next lecture]
conditionals and loops
Math
text I/O
primitive data types
assignment statements

Conditionals and loops

To control the flow of instruction execution

- Test a register's value.
- Change the PC, depending on the value.

opcode	*instruction*
C	branch if zero
D	branch if positive

Example: Absolute value of R[A]

10	**DA12**	If R[A] > 0 set PC to 12 (skip 11)
11	**2A0A**	Subtract R[A] from 0 (R[0]) and put result into R[A]
12	**...**	

Example: Typical `while` loop (assumes R[1] is 0001)

10	**CA15**	If R[A] is 0 set PC to 15
11	**...**	
12	**...**	
13	**2AA1**	Decrement R[A] by 1
14	**C010**	Set PC to 10
15	**...**	

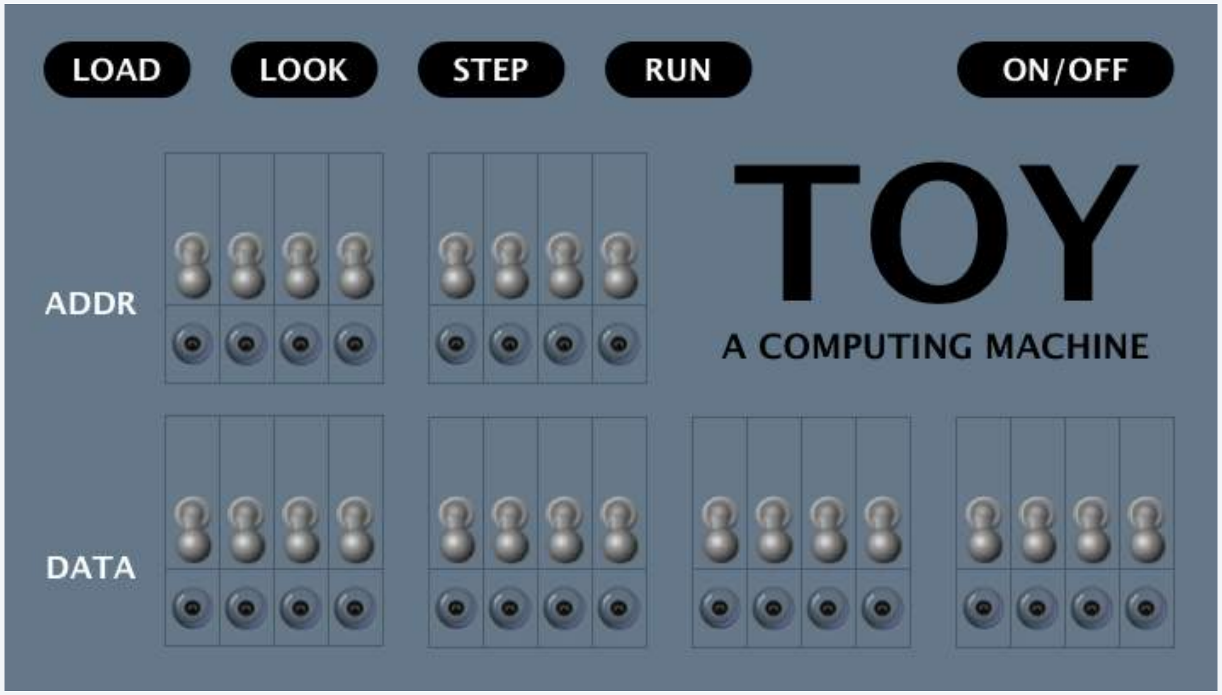
```
while (a != 0) {
  ...
  ...
   a--;
}
```

To infinity and beyond!

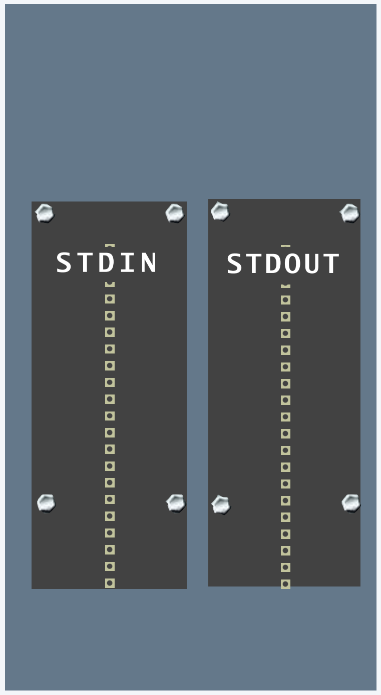
Standard input and output

An immediate problem

- We're not going to be able to address real-world problems with just switches and lights for I/O!
- One solution: Paper tape.



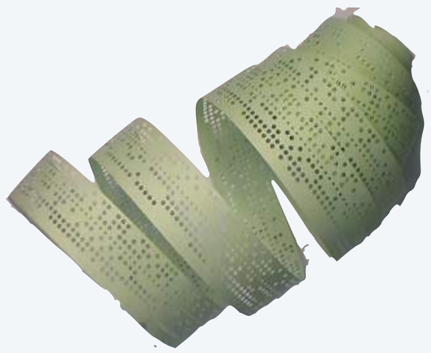
Need to bolt new I/O devices to the side of the machine.

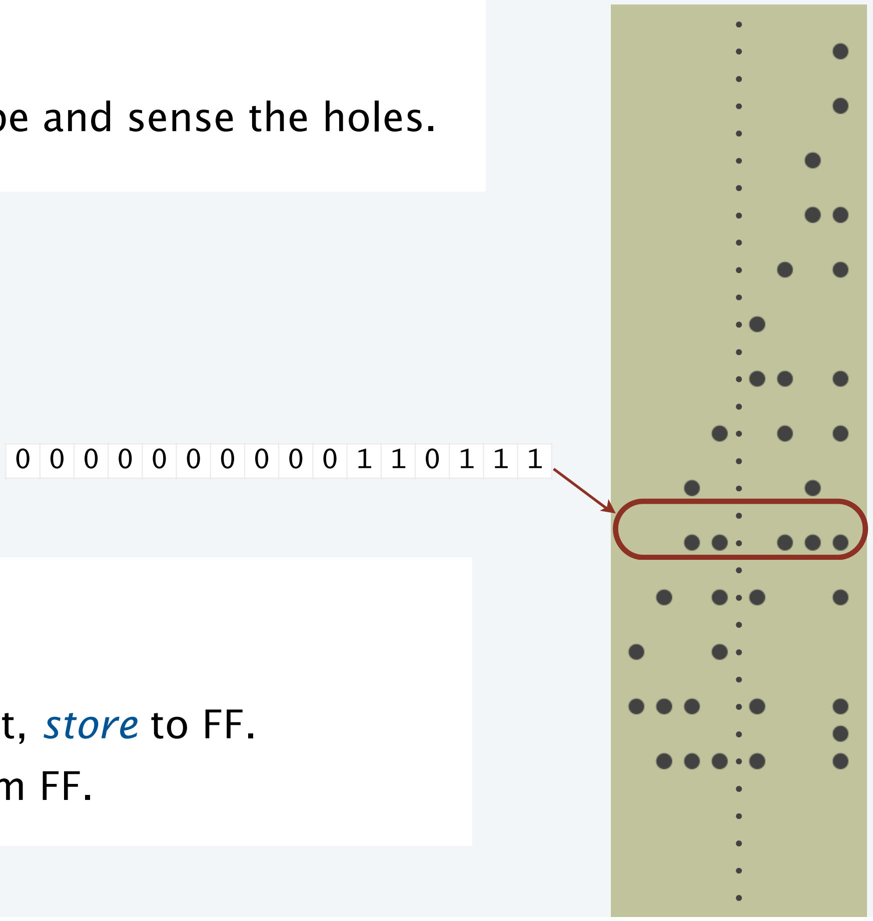


Standard input and output

Punched paper tape

- Encode 16-bit words in two 8-bit rows.
- To *write* a word, *punch a hole* for each 1.
- To *read* a word, shine a light behind the tape and sense the holes.





TOY mechanism

- Connect hardware to memory location FF.
- To *write* the contents of a register to stdout, *store* to FF.
- To *read* from stdin into a register, *load* from FF.

Flow of control and standard output example: Fibonacci numbers

Register trace

A	1	1	2	3	5	8	13	21	34	55	89
B	1	2	3	5	8	13	21	34	55	89	144
C	2	3	3	5	8	13	21	34	55	89	144
9	A	9	8	7	6	5	4	3	2	1	0

PC → 40

40	7 1 0 1	R[1] = 1
41	7 A 0 1	R[A] = 1
42	7 B 0 1	R[B] = 1
43	8 9 4 C	R[9] = M[4C]
44	C 9 4 B	if (R[9] == 0) PC = 4B
45	9 A F F	write R[A] to stdout
46	1 C A B	R[C] = R[A] + R[B]
47	1 A B 0	R[A] = R[B]
48	1 B C 0	R[B] = R[C]
49	2 9 9 1	R[9] = R[9] - 1
4A	C 0 4 4	PC = 44
4B	0 0 0 0	halt
4C	0 0 0 A	
	...	

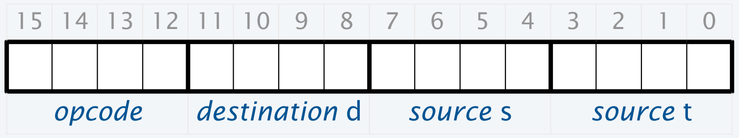
```
a = 1;
b = 1;
i = N;
while (i != 0) {
  StdOut.print(a);
  c = a + b;
  a = b;
  b = c;
  i = i - 1;
}
```

STDOUT

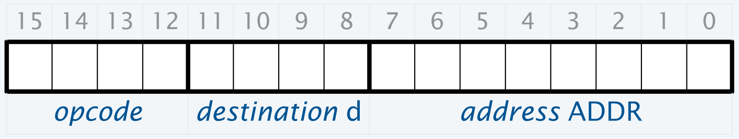
TOY reference card

opcode	*operation*	*format*	*pseudo-code*
0	halt	—	`halt`
1	add	RR	`R[d] = R[s] +  R[t]`
2	subtract	RR	`R[d] = R[s] -  R[t]`
3	bitwise and	RR	`R[d] = R[s] &  R[t]`
4	bitwise xor	RR	`R[d] = R[s] ^  R[t]`
5	shift left	RR	`R[d] = R[s] << R[t]`
6	shift right	RR	`R[d] = R[s] >> R[t]`
7	load addr	A	`R[d] = addr`
8	load	A	`R[d] = M[addr]`
9	store	A	`M[addr] = R[d]`
A	load indirect	RR	`R[d] = M[R[t]]`
B	store indirect	RR	`M[R[t]] = R[d]`
C	branch zero	A	`if (R[d] == 0) PC = addr`
D	branch positive	A	`if (R[d] > 0)  PC = addr`
E	jump register	RR	`PC = R[d]`
F	jump and link	A	`R[d] = PC + 1; PC = addr`

Format RR



Format A



ZERO	R[0] is always 0.
STANDARD INPUT	Load from FF.
STANDARD OUTPUT	Store to FF.

Pop quiz 1 on TOY

Q. What is the interpretation of

1A75 as a TOY instruction?

1A75 as a two's complement integer value?

0FFF as a TOY instruction?

0FFF as a two's complement integer value?

8888 as a TOY instruction?

8888 as a two's complement integer value? (Answer in base 16).

Pop quiz 2 on TOY

Q. How does one flip all the bits in a TOY register ?

Pop quiz 3 on TOY

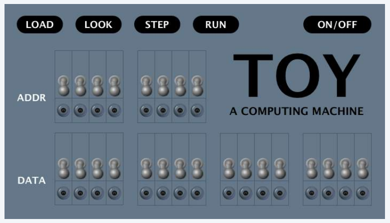
Q. What does the following TOY program leave in R[2] ?

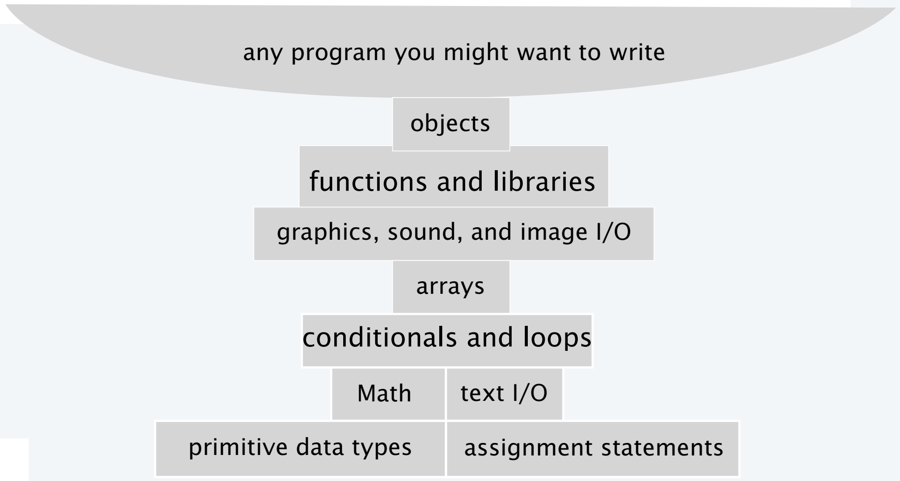
10	7 C 0 A	R[C] = 10_{10}
11	7 1 0 1	R[1] = 1
12	7 2 0 1	R[2] = 1
13	1 2 2 2	R[2] = R[2] + R[2]
14	2 C C 1	R[C] = R[C] - 1
15	D C 1 3	if (R[C] > 0) PC = 13
16	0 0 0 0	HALT

TOY vs. your laptop

Two different computing machines

- Both implement basic data types, conditionals, loops, and other low-level constructs.
- Both can have arrays, functions, and other high-level constructs.
- Both have infinite input and output streams.





Q. Is 256 words enough to do anything useful?

A. Yes! (See book, and stay tuned for next lecture.)

A. Yes! It is a Turing Machine, with a read/write I/O device (see theory lectures).

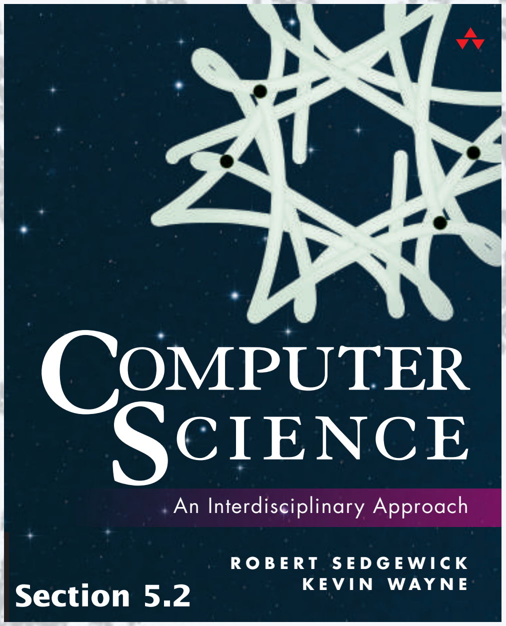
OK, we definitely want a faster version with more memory when we can afford it...

COMPUTER SCIENCE
SEDGEWICK / WAYNE

CS.17.E.MachineI.Programming

COMPUTER SCIENCE
SEDGEWICK / WAYNE

PART II: ALGORITHMS, MACHINES, and THEORY



http://introcs.cs.princeton.edu

17. A Computing Machine