

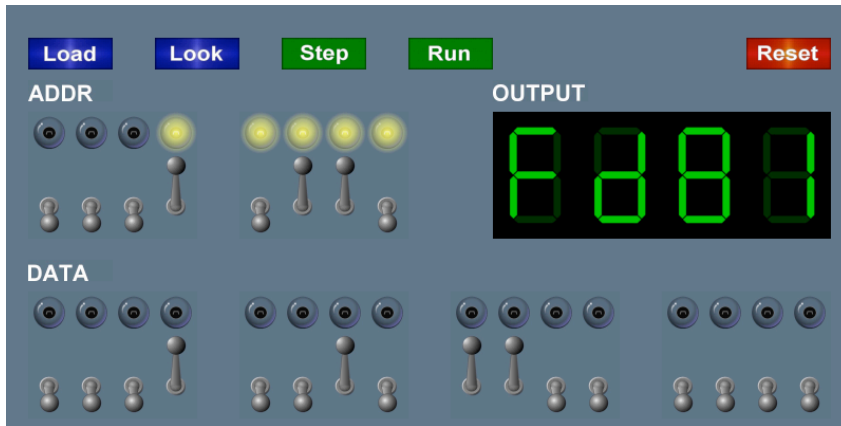
5: The TOY Machine

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.



Introduction to Computer Science · Robert Sedgewick and Kevin Wayne · <http://www.cs.Princeton.EDU/IntroCS>

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.

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			

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

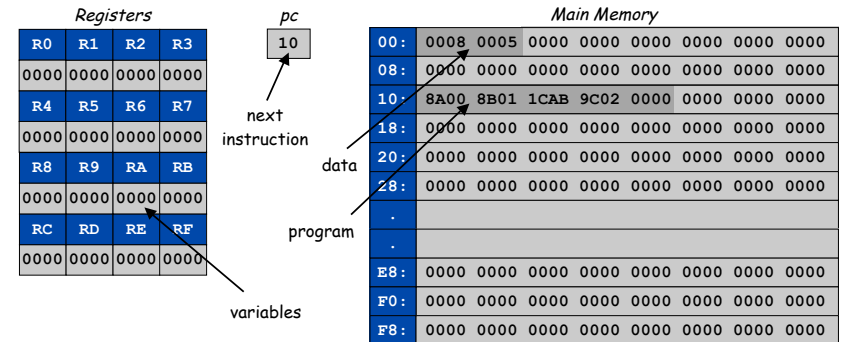
$$= 4096 + 2048 + 224 + 7$$

7

Machine "Core" Dump

Machine contents at a particular place and time.

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

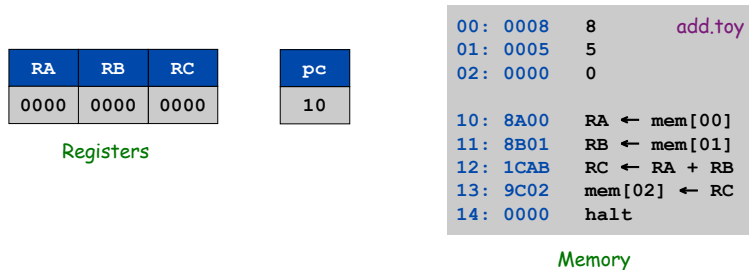


8

A Sample Program

A sample program.

- Adds $8 + 5 = D$.



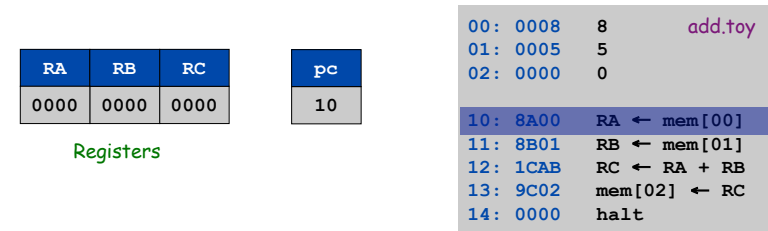
Since PC = 10, machine interprets 8A00 as an instruction.

10

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.



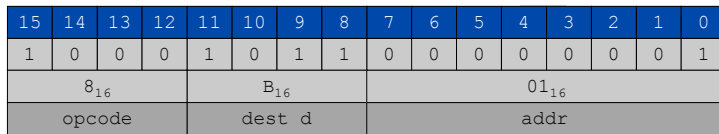
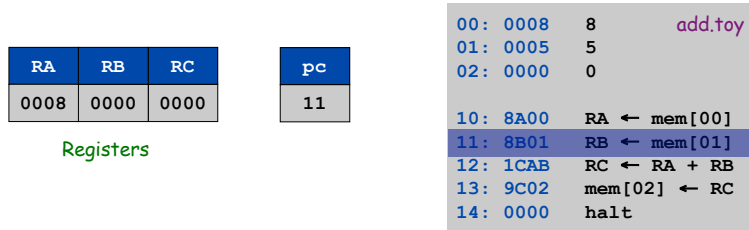
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
8_{16}				A_{16}				00_{16}							
opcode				dest d				addr							

11

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.

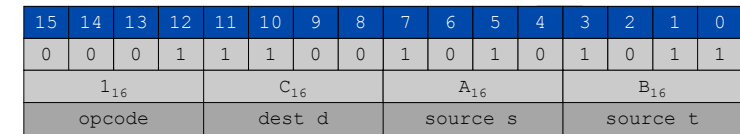
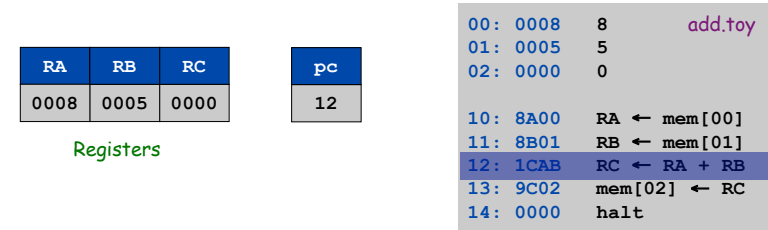


12

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.

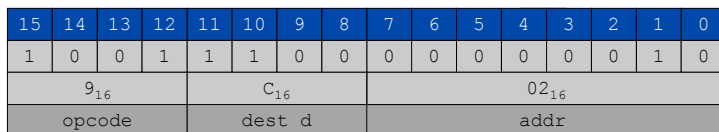
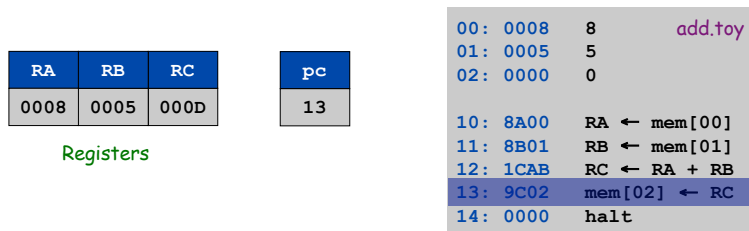


13

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.

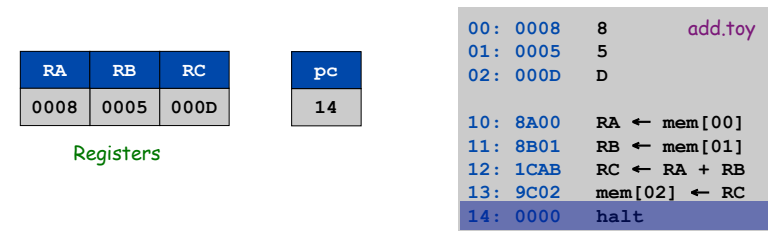


14

Halt

Halt. (opcode 0)

- Stop the machine.



15

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

16

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 <i>d</i>				source <i>s</i>				source <i>t</i>			
Format 2				opcode				dest <i>d</i>				addr							

17

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.



18

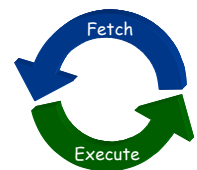
Using the TOY Machine: Run

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.



19

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 $c = a \times b$.
- 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.

20

21

Multiply

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

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

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
```

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

22

23

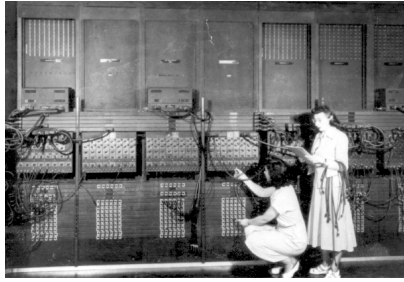
A Little History

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>

24

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)

25

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.

26

5: Supplemental Notes

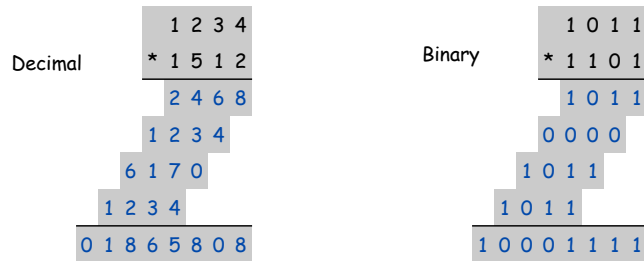
An Efficient Multiplication Algorithm

Inefficient multiply.

- Brute force multiplication algorithm loops a times.
- In worst case, 65,535 additions!

"Grade-school" multiplication.

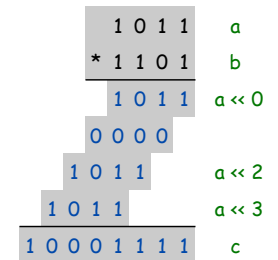
- Always 16 additions to multiply 16-bit integers.



Binary Multiplication

Grade school binary multiplication algorithm to compute $c = a \times b$.

- Initialize $c = 0$.
- Loop over i bits of b . $\leftarrow b_i = i^{\text{th}}$ bit of b
 - if $b_i = 0$, do nothing
 - if $b_i = 1$, shift a left i bits and add to c



Implement with built-in TOY shift instructions.

```
int c = 0;
for (int i = 15; i >= 0; i--)
    if ((b >> i) & 1) == 1
        c = c + (a << i);
```

$\leftarrow b_i = i^{\text{th}}$ bit of b

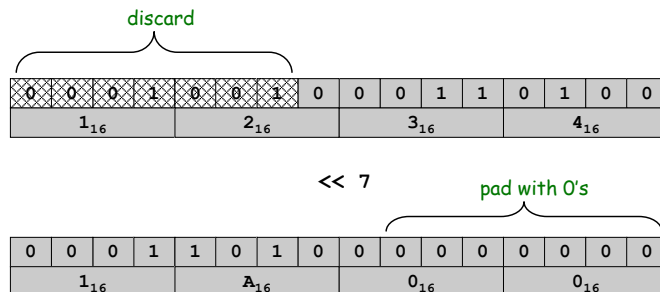
28

29

Shift Left

Shift left. (opcode 5)

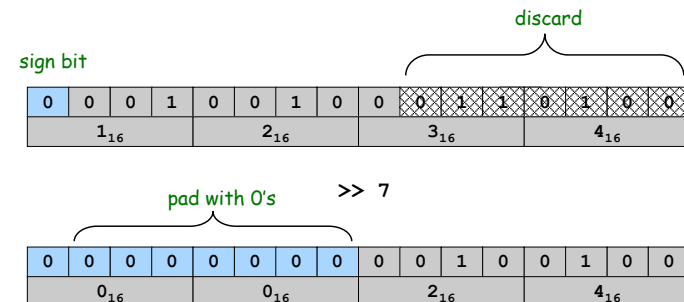
- Move bits to the left, padding with zeros as needed.
- $1234_{16} \ll 7 = 1A00_{16}$



Shift Right

Shift right. (opcode 6)

- Move bits to the right, padding with sign bit as needed.
- $1234_{16} \gg 7 = 0024_{16}$



30

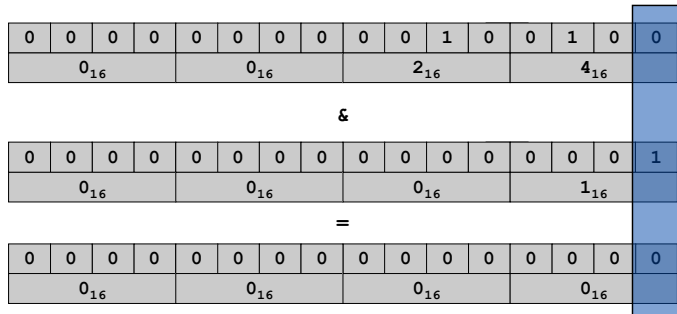
31

Bitwise AND

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

Logical AND. (opcode B)

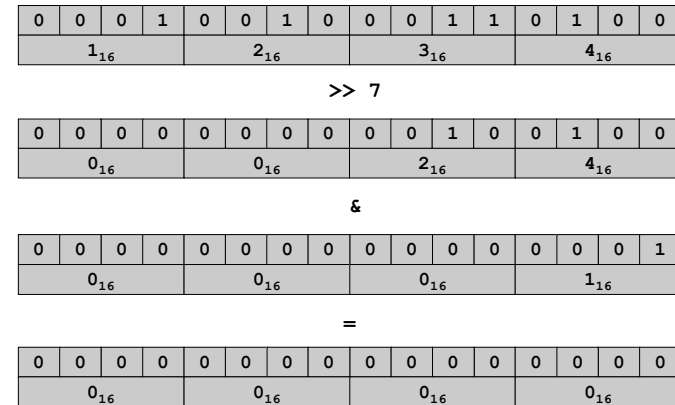
- Logic operations are BITWISE.
- $0024_{16} \& 0001_{16} = 0000_{16}$



Shifting and Masking

Shift and mask: get the 7th bit of 1234.

- Compute $1234_{16} \gg 7_{16} = 0024_{16}$.
- Compute $0024_{16} \& 1_{16} = 0_{16}$.



32

33

Binary Multiplication

```

0A: 0007   3
0B: 0008   9 ← inputs
0C: 0000   0 ← output
0D: 0000   0
0E: 0001   1 ← constants
0F: 0010  16

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: 820F   R2 ← mem[0F]   i = 16 ← 16 bit words

do {
  i--
  a << i
  b >> i
  bi = ith bit of b
  if bi is 1
    add a << i to sum
} while (i > 0);

15: 2221   R2 ← R2 - R1
16: 53A2   R3 ← RA << R2
17: 64B2   R4 ← RB >> R2
18: 3441   R4 ← R4 & R1
19: C41B   if (R4 == 0) goto 1B
1A: 1CC3   RC ← RC + R3
1B: D215   if (R2 > 0) goto 15

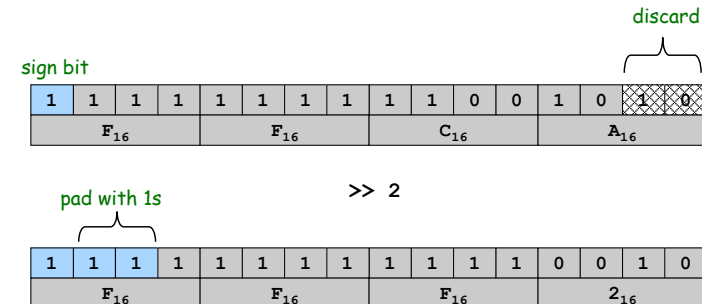
1C: 9C0C   mem[0C] ← RC
    
```

multiply-fast.toy

Shift Right (Sign Extension)

Shift right. (opcode 6)

- Move bits to the right, padding with sign bit as needed.
- $FFCA_{16} \gg 2_{16} = FFF2_{16}$
- $-53_{10} \gg 2_{10} = -13_{10}$



34

36