5. The TOY Machine
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.
- Some situations today where it is really necessary.

Computer architecture
- How does it work?
- How is a computer put together?

**TOY machine**: Optimized for simplicity, not cost or speed.
Data and Programs Are Just Bits

Each bit consists of two states:
• Switch is on or off; wire has high voltage or low voltage.

Everything in a computer is stored as bits.
  – Data **and programs**
  – Text, pictures, sounds, movies, programs, ...
Compilers

Compilers translate from high-level languages (like Java) to native machine instructions (bits).

```
int x = 8;
int y = 5;
int sum = x + y;
```

Compilation

<table>
<thead>
<tr>
<th>Java</th>
<th>TOY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable names</td>
<td>✔</td>
</tr>
<tr>
<td>Function names</td>
<td>✔</td>
</tr>
<tr>
<td>Structure</td>
<td>✔</td>
</tr>
<tr>
<td>Types</td>
<td>✔</td>
</tr>
<tr>
<td>Error Checking</td>
<td>✔</td>
</tr>
<tr>
<td>*, /, %, useful libraries</td>
<td>✔</td>
</tr>
<tr>
<td>Bits</td>
<td>✗</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compilation Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>00: 0008 8</td>
</tr>
<tr>
<td>01: 0005 5</td>
</tr>
<tr>
<td>02: 0000 0</td>
</tr>
<tr>
<td>10: 8A00 RA ← mem[00]</td>
</tr>
<tr>
<td>11: 8B01 RB ← mem[01]</td>
</tr>
<tr>
<td>12: 1CAB RC ← RA + RB</td>
</tr>
<tr>
<td>13: 9C02 mem[02] ← RC</td>
</tr>
<tr>
<td>14: 0000 halt</td>
</tr>
</tbody>
</table>

Memory addresses:
- add.toy
- instructions (bits, written in hex)
- assembly (human-readable mnemonics)
Binary People

There are only 10 types of people in the world:
Those who understand binary and those who don't.

http://www.thinkgeek.com/tshirts-apparel/unisex/frustrations/5aa9
Inside the TOY Box

Switches: Input data and programs.

Lights: View data.

Memory:
- Stores data and programs.
- 256 16-bit "words."
- Special word for stdin / stdout.

Program counter (PC):
- An extra 8-bit register.
- Next instruction to be executed.

Registers:
- Fastest form of storage.
- Scratch space during computation.
- 16 16-bit registers.
- Register 0 is always 0.

Arithmetic-Logic Unit (ALU): Manipulate data stored in registers.

Standard input/output: Interact with outside world.
TOY Machine "Core" Dump

A core dump is the contents of machine at a particular place and time.

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

A sample program: Adds $0008 + 0005 = 000D$.

<table>
<thead>
<tr>
<th>RA</th>
<th>RB</th>
<th>RC</th>
<th>pc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>10</td>
</tr>
</tbody>
</table>

Registers

Program Counter

<table>
<thead>
<tr>
<th>00: 0008</th>
<th>01: 0005</th>
<th>02: 0000</th>
<th>03: 0000</th>
<th>04: 0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA ← mem[00]</td>
<td>RB ← mem[01]</td>
<td>RC ← RA + RB</td>
<td>mem[02] ← RC</td>
<td>halt</td>
</tr>
</tbody>
</table>

TOY memory (program and data)
A Sample Program

Program counter: The pc is initially 10, so the machine interprets 8A00 as an instruction.

<table>
<thead>
<tr>
<th>RA</th>
<th>RB</th>
<th>RC</th>
<th>pc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:</td>
<td>0008</td>
</tr>
<tr>
<td>01:</td>
<td>0005</td>
</tr>
<tr>
<td>02:</td>
<td>0000</td>
</tr>
<tr>
<td>10:</td>
<td>8A00 RA ← mem[00]</td>
</tr>
<tr>
<td>11:</td>
<td>8B01 RB ← mem[01]</td>
</tr>
<tr>
<td>12:</td>
<td>1CAB RC ← RA + RB</td>
</tr>
<tr>
<td>13:</td>
<td>9C02 mem[02] ← RC</td>
</tr>
<tr>
<td>14:</td>
<td>0000 halt</td>
</tr>
</tbody>
</table>

Index of next instruction to execute.
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."

<table>
<thead>
<tr>
<th>RA</th>
<th>RB</th>
<th>RC</th>
<th>pc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pc</th>
<th>RA</th>
<th>RB</th>
<th>RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0008</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0005</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>0000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8A00</td>
<td>RA ← mem[00]</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>8B01</td>
<td>RB ← mem[01]</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1CAB</td>
<td>RC ← RA + RB</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>9C02</td>
<td>mem[02] ← RC</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0000</td>
<td>halt</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>addr</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0016</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0016</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>opcode</th>
<th>dest d</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0016</td>
<td>816</td>
<td>A16</td>
<td>0016</td>
</tr>
</tbody>
</table>

Penn Engineering
Load

Load [opcode 8]

- Loads the contents of some memory location into a register.
- 8B00 means "load the contents of memory cell 00 into register B."

<table>
<thead>
<tr>
<th>RA</th>
<th>RB</th>
<th>RC</th>
<th>pc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0008</td>
<td>0000</td>
<td>0000</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>02</td>
</tr>
<tr>
<td>09</td>
</tr>
<tr>
<td>0A</td>
</tr>
<tr>
<td>0B</td>
</tr>
<tr>
<td>0C</td>
</tr>
<tr>
<td>0D</td>
</tr>
<tr>
<td>0E</td>
</tr>
<tr>
<td>0F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opcode</th>
<th>dest d</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0008</td>
<td>8</td>
</tr>
<tr>
<td>01</td>
<td>0005</td>
<td>5</td>
</tr>
<tr>
<td>02</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pc</th>
<th>RA</th>
<th>RB</th>
<th>RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>0008</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>RA</th>
<th>RB</th>
<th>RC</th>
<th>pc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0008</td>
<td>0005</td>
<td>0000</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pc</th>
<th>opcode</th>
<th>dest d</th>
<th>source s</th>
<th>source t</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0008</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0005</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>0000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8A00</td>
<td>RA ← mem[00]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>8B01</td>
<td>RB ← mem[01]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1CAB</td>
<td>RC ← RA + RB</td>
<td>mem[02] ← RC</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>9C02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0000</td>
<td>halt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Store

Store [opcode 9]

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

<table>
<thead>
<tr>
<th>RA</th>
<th>RB</th>
<th>RC</th>
<th>pc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0008</td>
<td>0005</td>
<td>000D</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pc</th>
<th>00: 0008</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>01: 0005</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>02: 0000</td>
<td>0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>opcode</th>
<th>dest d</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>916</td>
<td>C16</td>
<td>0216</td>
</tr>
</tbody>
</table>

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

Penn Engineering
Halt

Halt [opcode 0]

- Stop the machine.

<table>
<thead>
<tr>
<th>RA</th>
<th>RB</th>
<th>RC</th>
<th>pc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0008</td>
<td>0005</td>
<td>000D</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:</td>
<td>0008</td>
<td>8</td>
</tr>
<tr>
<td>01:</td>
<td>0005</td>
<td>5</td>
</tr>
<tr>
<td>02:</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>10:</td>
<td>8A00</td>
<td>RA ← mem[00]</td>
</tr>
<tr>
<td>11:</td>
<td>8B01</td>
<td>RB ← mem[01]</td>
</tr>
<tr>
<td>12:</td>
<td>1CAB</td>
<td>RC ← RA + RB</td>
</tr>
<tr>
<td>13:</td>
<td>9C02</td>
<td>mem[02] ← RC</td>
</tr>
<tr>
<td>14:</td>
<td>0000</td>
<td>halt</td>
</tr>
</tbody>
</table>
Program and Data

Program: Sequence of 16-bit integers, interpreted one way.

Data: Sequence of 16-bit integers, interpreted other way.

Program Counter (pc): Holds memory address of the next "instruction" and determines which integers get interpreted as instructions.

16 instruction types: Changes contents of registers, memory, and pc in specified, well-defined ways.

Instructions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>halt</td>
</tr>
<tr>
<td>1</td>
<td>add</td>
</tr>
<tr>
<td>2</td>
<td>subtract</td>
</tr>
<tr>
<td>3</td>
<td>and</td>
</tr>
<tr>
<td>4</td>
<td>xor</td>
</tr>
<tr>
<td>5</td>
<td>shift left</td>
</tr>
<tr>
<td>6</td>
<td>shift right</td>
</tr>
<tr>
<td>7</td>
<td>load address</td>
</tr>
<tr>
<td>8</td>
<td>load</td>
</tr>
<tr>
<td>9</td>
<td>store</td>
</tr>
<tr>
<td>A</td>
<td>load indirect</td>
</tr>
<tr>
<td>B</td>
<td>store indirect</td>
</tr>
<tr>
<td>C</td>
<td>branch zero</td>
</tr>
<tr>
<td>D</td>
<td>branch positive</td>
</tr>
<tr>
<td>E</td>
<td>jump register</td>
</tr>
<tr>
<td>F</td>
<td>jump and link</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Format 1
- opcode
- dest d
- source s
- source t

Format 2
- opcode
- dest d
- addr

← add, subtract, ...
← load, store, ...
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.

Penn Engineering
Interfacing with the TOY Machine

To execute the program:

- Set 8 memory address switches to address of first instruction.
- Press **Look** to set pc to first instruction.
- Press **Run** 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.
Flow Control

Flow Control

• To harness the power of TOY, need loops and conditionals.
• Manipulate \texttt{pc} to control program flow.

\begin{verbatim}
if (boolean expression) {
  statement 1;
  statement 2;
}
\end{verbatim}

\begin{verbatim}
while (boolean expression) {
  statement 1;
  statement 2;
}
\end{verbatim}

Branch if zero [opcode C]

• Changes \texttt{pc} depending on whether value of some register is zero.
• Used to implement: \texttt{for, while, if-else}.

Branch if positive [opcode D]

• Changes \texttt{pc} depending on whether value of some register is positive.
• Used to implement: \texttt{for, while, if-else}.
An Example: Multiplication

Multiply: Given integers \( a \) and \( b \), compute \( c = a \times b \).

TOY multiplication: No direct support in TOY hardware.

Brute-force multiplication algorithm:

- Initialize \( c \) to 0.
- Add \( b \) to \( c \), \( a \) times.

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

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

Issues ignored: Slow, overflow, negative numbers.
Multiply

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A:</td>
<td>0003</td>
<td>RA</td>
<td>$a = 3$</td>
</tr>
<tr>
<td>0B:</td>
<td>0009</td>
<td>RB</td>
<td>$b = 9$</td>
</tr>
<tr>
<td>0C:</td>
<td>0000</td>
<td>RC</td>
<td>$c = 0$</td>
</tr>
<tr>
<td>0D:</td>
<td>0000</td>
<td>R1</td>
<td>always 1</td>
</tr>
<tr>
<td>0E:</td>
<td>0001</td>
<td>pc</td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>8A0A</td>
<td>RA</td>
<td>$a = 3$</td>
</tr>
<tr>
<td>11:</td>
<td>8B0B</td>
<td>RB</td>
<td>$b = 9$</td>
</tr>
<tr>
<td>12:</td>
<td>8C0D</td>
<td>RC</td>
<td>$c = 0$</td>
</tr>
<tr>
<td>13:</td>
<td>810E</td>
<td>R1</td>
<td>always 1</td>
</tr>
<tr>
<td>14:</td>
<td>CA18</td>
<td>if (RA == 0) pc ← 18 while (a != 0) {</td>
<td></td>
</tr>
<tr>
<td>15:</td>
<td>1CCB</td>
<td>RC ← RC + RB</td>
<td>$c = c + b$;</td>
</tr>
<tr>
<td>16:</td>
<td>2AA1</td>
<td>RA ← RA - R1</td>
<td>$a = a - 1$;</td>
</tr>
<tr>
<td>17:</td>
<td>C014</td>
<td>pc ← 14</td>
<td>}</td>
</tr>
<tr>
<td>18:</td>
<td>9C0C</td>
<td>mem[0C] ← RC</td>
<td></td>
</tr>
<tr>
<td>19:</td>
<td>0000</td>
<td>halt</td>
<td></td>
</tr>
</tbody>
</table>

loop

multiply.toy
## Step-By-Step Trace

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

*multiply.toy*
A Little History

Electronic Numerical Integrator and Calculator (ENIAC):
• First widely known general purpose electronic computer.
• 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)
Standard Output

- Writing to memory location \texttt{FF} sends one word to TOY stdout.
- Ex. \texttt{9AFF} writes the integer in register \texttt{A} to stdout.

\begin{verbatim}
00: 0000 0
01: 0001 1
10: 8A00 RA \leftarrow \text{mem}[00] \quad a = 0
11: 8B01 RB \leftarrow \text{mem}[01] \quad b = 1
12: 9AFF \text{write RA to stdout} \quad \text{do \{ print a}
13: 1AAB RA \leftarrow RA + RB \quad a = a + b
14: 2BAB RB \leftarrow RA - RB \quad b = a - b
15: DA12 if (RA > 0) goto 12 \quad } \text{while (a > 0)}
16: 0000 \text{halt}
\end{verbatim}

fibonacci.toy
while (!StdIn.isEmpty()) {
    a = StdIn.readInt();
    sum = sum + a;
}
StdOut.println(sum);

Ex: read in a sequence of integers and print their sum.

• In Java, stop reading when EOF.
• In TOY, stop reading when user enters 0000.
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.

```
Decimal
1 2 3 4
* 1 5 1 2
---
  2 4 6 8
 1 2 3 4
1 2 3 4
 6 1 7 0
 1 2 3 4
  0 1 8 6 5 8 0 8

Binary
1 0 1 1
* 1 1 0 1
---
1 0 1 1
0 0 0 0
1 0 1 1
1 0 1 1
1 0 0 0 1 1
1 1 1 1
```
Binary Multiplication

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

- Initialize \( c = 0 \).
- Loop over \( i \) bits 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.

\[
\begin{array}{c|c|c|c|c}
& 1 & 0 & 1 & 1 \\
\times & 1 & 1 & 0 & 1 \\
\hline
& 1 & 0 & 1 & 1 \\
& 0 & 0 & 0 & 0 \\
& 1 & 0 & 1 & 1 \\
& 1 & 0 & 1 & 1 \\
\hline
& 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\text{int } c = 0; \\
\text{for (int } i = 15; i >= 0; i--) \\
\quad \text{if } (((\text{b} >> i) & 1) == 1) \\
\quad \quad c = c + (\text{a} << i); \\
\]
Shift Left

**Shift Left [opcode 5]**

- Move bits to the left, padding with zeros as needed.

  \[ 1234_{16} \ll 7_{16} = 1A00_{16} \]

\[
\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
\hline
1_{16} & 2_{16} & 3_{16} & 4_{16} \\
\end{array}
\]

**discard**

\[
\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
1_{16} & A_{16} & 0_{16} & 0_{16} \\
\end{array}
\]

**pad with 0’s**
Shift Right

Shift Right [opcode 6]

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

Logical AND [opcode B]

• Logic operations are BITWISE.
  – \(0\ 0\ 2\ 4\ 1\ 6\ \&\ 0\ 0\ 0\ 1\ 1\ 6\ =\ 0\ 0\ 0\ 0\ 1\ 6\)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>&amp;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0_{16} & 0_{16} & 2_{16} & 4_{16} \\
\end{array}
\]

\[
\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0_{16} & 0_{16} & 0_{16} & 1_{16} \\
\end{array}
\]

\[
\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0_{16} & 0_{16} & 0_{16} & 0_{16} \\
\end{array}
\]

\[
\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0_{16} & 0_{16} & 0_{16} & 0_{16} \\
\end{array}
\]

\[
\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0_{16} & 0_{16} & 0_{16} & 0_{16} \\
\end{array}
\]
Shifting and Masking

Shift and mask: get the 7th bit of 1234

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

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
1_{16} & 2_{16} & 3_{16} & 4_{16} \\
0_{16} & 0_{16} & 2_{16} & 4_{16} \\
0_{16} & 0_{16} & 0_{16} & 1_{16} \\
0_{16} & 0_{16} & 0_{16} & 0_{16} \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

$\gg 7$

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

$\&$

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

$= 0_{16}$
## Binary Multiplication

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A:</td>
<td>0003</td>
<td>inputs</td>
</tr>
<tr>
<td>0B:</td>
<td>0009</td>
<td>output</td>
</tr>
<tr>
<td>0C:</td>
<td>0000</td>
<td>constants</td>
</tr>
<tr>
<td>0D:</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0E:</td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>0F:</td>
<td>0010</td>
<td></td>
</tr>
</tbody>
</table>

10: 8A0A  \[RA \leftarrow \text{mem}[0A]\] a
11: 8B0B  \[RB \leftarrow \text{mem}[0B]\] b
12: 8C0D  \[RC \leftarrow \text{mem}[0D]\] c = 0
13: 810E  \[R1 \leftarrow \text{mem}[0E]\] always 1
14: 820F  \[R2 \leftarrow \text{mem}[0F]\] i = 16

\[
\text{do } \{ \\
15: 2221  \[R2 \leftarrow R2 - R1\] i-- \\
16: 53A2  \[R3 \leftarrow RA \lll R2\] a \lll i \\
17: 64B2  \[R4 \leftarrow RB \ggg R2\] b \ggg i \\
18: 3441  \[R4 \leftarrow R4 \& R1\] b_i = i^{th} bit of b \\
19: C41B  \text{if (R4 == 0) goto 1B if b_i is 1} \\
1A: 1CC3  \[RC \leftarrow RC + R3\] add a \lll i to sum \\
1B: D215  \text{if (R2 > 0) goto 15}\} \text{while (i > 0)};
\]

1C: 9C0C  \[\text{mem}[0C] \leftarrow RC\]

**multiply-fast.toy**
Bitwise XOR

Bitwise XOR [opcode 4]

- Logic operations are BITWISE.

\[ \begin{array}{cccc}
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 \\
\end{array} \begin{array}{cccc}
1 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 \\
3 & 1 & 1 & 0 \\
4 & 1 & 0 & 0 \\
\end{array} \begin{array}{cccc}
\begin{array}{cccc}
1 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 \\
3 & 1 & 1 & 0 \\
4 & 1 & 0 & 0 \\
\end{array} \end{array} \begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 \\
\end{array} \begin{array}{cccc}
F_{16} & A_{16} & D_{16} & 2_{16} \\
E_{16} & 8_{16} & E_{16} & 6_{16} \\
\end{array} \begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 \\
\end{array} \begin{array}{cccc}
1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 \\
\end{array} \begin{array}{cccc}
E_{16} & 8_{16} & E_{16} & 6_{16} \\
\end{array} \end{array} \]

\[1234_{16} \oplus \text{FAD2}_{16} = \text{E8E6}_{16}\]
Shift Right (Sign Extension)

Shift Right [opcode 6]

- Move bits to the right, padding with sign bit as needed.
  - \( \text{FFCA}_{16} \gg 2_{16} = \text{FFF2}_{16} \)
  - \(-53_{10} \gg 2_{10} = -13_{10}\)

```
\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\text{F}_{16} & \text{F}_{16} & \text{C}_{16} & \text{A}_{16} \\
\end{array}
```

Discard

```
\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\text{F}_{16} & \text{F}_{16} & \text{F}_{16} & 2_{16} \\
\end{array}
```
Load Address (a.k.a. Load Constant)

Load Address [opcode 7]

• Loads an 8-bit integer into a register.
• 7A30 means load the value 30 into register A.

Applications.
• Load a small constant into a register.
• Load a 8-bit memory address into a register.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

7_{16} \quad A_{16} \quad 3_{16} \quad 0_{16}

opcode \quad dest \ d \quad addr

register stores "pointer" to a memory cell

Java code

```java
a = 0x30;
```
Arrays in TOY

TOY main memory is a giant array

• Can access memory cell 30 using load and store.
• 8C30 means load mem[30] into register C.
• Goal: access memory cell i where i is a variable.

Load Indirect [opcode A]

• AC06 means load mem[R6] into register C.

Store Indirect [opcode B]

• BC06 means store contents of register C into mem[R6].

```java
for (int i = 0; i < N; i++)
a[i] = StdIn.readInt();

for (int i = 0; i < N; i++)
StdOut.println(a[N-i-1]);
```
TOY Implementation of Reverse

TOY implementation of reverse:

• Read in a sequence of integers and store in memory 30, 31, 32, ... until reading 0000.
• Print sequence in reverse order.

```java
while(true) {
    c = StdIn.readInt();
    if (c == 0) break;
    a[n] = c;
    n++;
}
```

read in the data
TOY Implementation of Reverse

TOY implementation of reverse:

• Read in a sequence of integers and store in memory 30, 31, 32, ... until reading 0000.

• Print sequence in reverse order.

```
while(true) {
    c = StdIn.readInt();
    if (c == 0) break;
    R6 ← RA + RB
    a[n] = c;
    n++;}
```

print in reverse order
Unsafe Code at any Speed

Q. What happens if we make array start at 00 instead of 30?
A. Self modifying program; can overflow buffer and run arbitrary code!

Q. What happens if we make array start at 00 instead of 30?
A. Self modifying program; can overflow buffer and run arbitrary code!
What If We Lose Control (in C or C++)?

Buffer overflow:
- Array buffer[] has size 100.
- User might enter 200 characters.
- Might lose control of machine behavior.

Consequences: Viruses and worms.

Java enforces security:
- Type safety.
- Array bounds checking.
- Not foolproof.

#include <stdio.h>
int main(void) {
    char buffer[100];
    scanf("%s", buffer);
    printf("%s
", buffer);
    return 0;
}

unsafe C program

shine 50W bulb at DRAM
[Appel-Govindavajhala '03]
Buffer Overflow Attacks

Stuxnet Worm [July 2010]
• Step 1. Natanz centrifuge fuel-refining plant employee plugs in USB flash drive.
• Step 2. Data becomes code by exploiting Windows buffer overflow; machine is owned.
• Step 3. Uranium enrichment in Iran stalled.

More buffer overflow attacks: Duqu, Flame, Morris worm, Code Red, SQL Slammer, iPhone unlocking, Xbox softmod, JPEG of death, ...

Lesson:
• Not easy to write error-free software.
• Embrace Java security features.
• Keep your OS patched.
Dumping

Q. Work all day to develop operating system in \texttt{mem[10]} to \texttt{mem[FF]}. How to save it?

A. Write \texttt{dump.toy} and run it to dump contents of memory onto tape.

\begin{center}
\begin{tabular}{c|c|c}
00: 7001 & R1 & 0001 \\
01: 7210 & R2 & 0010 \\
02: 73FF & R3 & 00FF \\
\hline
03: AA02 & RA & \texttt{mem[R2]} \\
04: 9AFF & write & RA \\
05: 1221 & R2 & R2 + R1 \\
06: 2432 & R4 & R3 - R2 \\
07: D403 & if (R4 > 0) & goto 03 \\
08: 0000 & halt & \\
\end{tabular}
\end{center}
## Booting

Q. How do you get it back?

A. Run `boot.toys` and to read `mem[10] to mem[FF]` from tape.
A Little History

Electronic Numerical Integrator and Calculator (ENIAC):

- First widely known general purpose electronic computer.
- Conditional jumps, programmable.
- Programming: change switches and cable connections.
- Data: enter numbers using punch cards.

30 tons
30 x 50 x 8.5 ft
17,468 vacuum tubes
300 multiply/sec
15,000 watts

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

ENIAC, Ester Gerston (left), Gloria Gordon (right)
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, 1944]
• Data and program encoded in binary.
• Data and program stored in same memory.
• Can change program without rewiring.

Outgrowth of Alan Turing's work

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

#### Format 1

<table>
<thead>
<tr>
<th>#</th>
<th>Operation</th>
<th>Fmt</th>
<th>Pseudocode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>halt</td>
<td>1</td>
<td><code>exit(0)</code></td>
</tr>
<tr>
<td>1</td>
<td>add</td>
<td>1</td>
<td>[R[d] \leftarrow R[s] + R[t]]</td>
</tr>
<tr>
<td>2</td>
<td>subtract</td>
<td>1</td>
<td>[R[d] \leftarrow R[s] - R[t]]</td>
</tr>
<tr>
<td>3</td>
<td>and</td>
<td>1</td>
<td>[R[d] \leftarrow R[s] &amp; R[t]]</td>
</tr>
<tr>
<td>4</td>
<td>xor</td>
<td>1</td>
<td>[R[d] \leftarrow R[s] ^ R[t]]</td>
</tr>
<tr>
<td>5</td>
<td>shift left</td>
<td>1</td>
<td>[R[d] \leftarrow R[s] &lt;&lt; R[t]]</td>
</tr>
<tr>
<td>6</td>
<td>shift right</td>
<td>1</td>
<td>[R[d] \leftarrow R[s] &gt;&gt; R[t]]</td>
</tr>
<tr>
<td>7</td>
<td>load addr</td>
<td>2</td>
<td>[R[d] \leftarrow addr]</td>
</tr>
<tr>
<td>8</td>
<td>load</td>
<td>2</td>
<td>[R[d] \leftarrow mem[addr]]</td>
</tr>
<tr>
<td>9</td>
<td>store</td>
<td>2</td>
<td>[mem[addr] \leftarrow R[d]]</td>
</tr>
<tr>
<td>A</td>
<td>load indirect</td>
<td>1</td>
<td>[R[d] \leftarrow mem[R[t]]]</td>
</tr>
<tr>
<td>B</td>
<td>store indirect</td>
<td>1</td>
<td>[mem[R[t]] \leftarrow R[d]]</td>
</tr>
<tr>
<td>C</td>
<td>branch zero</td>
<td>2</td>
<td>[if (R[d] == 0) pc \leftarrow addr]</td>
</tr>
<tr>
<td>D</td>
<td>branch positive</td>
<td>2</td>
<td>[if (R[d] &gt; 0) pc \leftarrow addr]</td>
</tr>
<tr>
<td>E</td>
<td>jump register</td>
<td>2</td>
<td>[pc \leftarrow R[d]]</td>
</tr>
<tr>
<td>F</td>
<td>jump and link</td>
<td>2</td>
<td>[R[d] \leftarrow pc; pc \leftarrow addr]</td>
</tr>
</tbody>
</table>

#### Format 2

<table>
<thead>
<tr>
<th>#</th>
<th>Operation</th>
<th>Fmt</th>
<th>Pseudocode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>opcode</td>
<td>2</td>
<td>[R[d] \leftarrow \text{mem}]</td>
</tr>
</tbody>
</table>

Register 0 always reads 0.
Loads from mem[FF] from stdin.
Stores to mem[FF] to stdout.

16-bit registers.
16-bit memory.
8-bit program counter.