5. The TOY Machine

What is TOY?

An imaginary machine similar to:
- Ancient computers.
- Today's microprocessors.
- And practically everything in between!

Why Study TOY?

Machine language programming.
- How do Java programs relate to computer?
- Key to understanding Java references.
- Still 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 performance.
Inside the 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.
• Keeps track of 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, standard output. Interact with outside world.

Data and Programs Are Encoded in Binary

Each bit consists of two states:
• 1 or 0; true or false.
• Switch is on or off; wire has high voltage or low voltage.

Everything stored in a computer is a sequence of bits.
• Data and programs.
• Text, documents, pictures, sounds, movies, executables, ...

6

How to represent integers?
• Use binary encoding.
• Ex: \(6375_{10} = 0001100011100111_2\)

\[
<table>
<thead>
<tr>
<th>\text{Dec}</th>
<th>\text{Bin}</th>
<th>\text{Dec}</th>
<th>\text{Bin}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>
\]

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

\[
6375_{10} = 6375_{10} = +2^{12} + 2^{11} + 2^7 + 2^6 + 2^3 + 2^2 + 2^1 + 2^0
= 4096 + 2048 + 128 + 64 + 32 + 4 + 2 + 1
\]

M = 77_{10} = 01001101 = 4D_{16}
O = 79_{10} = 01001111 = 4F_{16}
M = 77_{10} = 01001101 = 4D_{16}

Hexadecimal Encoding

How to represent integers?
• Use hexadecimal encoding.
• Binary code, four bits at a time.
• Ex: \(6375_{10} = 0001100011100111_2\)

\[
<table>
<thead>
<tr>
<th>\text{Dec}</th>
<th>\text{Bin}</th>
<th>\text{Hex}</th>
<th>\text{Dec}</th>
<th>\text{Bin}</th>
<th>\text{Hex}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
<td>10</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
<td>11</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
<td>12</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
<td>13</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
<td>14</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
<td>15</td>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>
\]

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

\[
6375_{10} = 6375_{10} = 1 \times 16^3 + 8 \times 16^2 + 14 \times 16^1 + 7 \times 16^0
= 4096 + 2048 + 224 + 7
\]
Machine "Core" Dump

Machine contents at a particular place and time.
• Record of what program has done.
• Completely determines what machine will do.

![Machine Core Diagram]

A Sample Program

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

TOY code to compute 0008₁₀ + 0005₁₀

![A Sample Program Diagram]

Program counter. The pc is initially 10, so the 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.

<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
- RA: 0000
- RB: 0000
- RC: 0000

#### Program counter
- pc: 10

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

### 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.

<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>

#### Registers
- RA: 0008
- RB: 0005
- RC: 0000

#### Program counter
- pc: 12

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

#### Registers
- RA: 0008
- RB: 0005
- RC: 000D

#### Program counter
- pc: 13

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

### Memory Addresses

<table>
<thead>
<tr>
<th>addr</th>
<th>RA</th>
<th>RB</th>
<th>RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>01</td>
<td>0005</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>02</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>03</td>
<td>0005</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>04</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

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

Registers | Program counter

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

TOY code to compute $0008_{16} + 0005_{16}$

---

**Same Program, Different Data**

Program. Sequence of instructions.

Instruction addresses. 10, 11, 12, 13, and 14 (executed when pc points to it).

Data addresses. 00, 01, and 02 (used and changed by instructions).

<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

00: 1CAB 7339
01: 1CAB 7339
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

The TOY computes $1CAB_{16} = 1 \times 16^3$

+ $12 \times 16^2$

+ $10 \times 16^1$

+ $11 \times 16^0$

= 4096 + 3072 + 160 + 11 = 7339_{10}

add toy

---

**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>

Registers | Program counter

00: 1CAB 7339
01: 1CAB 7339
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

add toy

---

**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.

<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>11</td>
</tr>
</tbody>
</table>

Registers | Program counter

00: 1CAB 7339
01: 1CAB 7339
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

add toy
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.

```
Add.
00: 1CAB 7339
01: 1CAB 7339
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
```

Halt  [opcode 0]
• Stop the machine.

```
Halt.
00: 1CAB 7339
01: 1CAB 7339
02: 3956 14678
10: 8A00 RA ← mem[00]
11: 8B01 RB ← mem[01]
12: 1CAB RC ← RA + RB
13: 9C02 mem[02] ← RC
14: 0000 halt
```

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.

```
Store.
00: 1CAB 7339
01: 1CAB 7339
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
```

Program and Data

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

**Data.** Sequence of 16-bit integers, interpreted another 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.
**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></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>
<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>Format 1</td>
<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>
<tr>
<td>Format 2</td>
<td>opcode</td>
<td>dest d</td>
<td>source s</td>
<td>source t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 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.

**Flow Control**

Flow control.
- To harness the power of TOY, need loops and conditionals.
- Manipulate \( pc \) to control program flow.

Branch if zero. \([\text{opcode } C]\)
- Changes \( pc \) depending on whether value of some register is **zero**.
- Used to implement: **for**, **while**, **if-else**.

Branch if positive. \([\text{opcode } D]\)
- Changes \( pc \) depending on whether value of some register is **positive**.
- Used to implement: **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.

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.

### Step-By-Step Trace

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>RA</th>
<th>RB</th>
<th>RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>0003</td>
<td></td>
<td>0009</td>
<td>0000</td>
</tr>
<tr>
<td>10</td>
<td>8A0A</td>
<td>RA</td>
<td>mem[OA]</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>8B0B</td>
<td>RB</td>
<td>mem[OB]</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>8C0D</td>
<td>RC</td>
<td>mem[OD]</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>810E</td>
<td>R1</td>
<td>mem[OE]</td>
<td></td>
</tr>
</tbody>
</table>
| 14 | CA18  | if (RA == 0) pc ← 18  
| 15 | 1CCB  | RC  | RC + RB|     
| 16 | 2AA1  | RA  | RA - R1|     
| 17 | C014  | pc  | 14    |     
| 18 | 9C0C  | mem[OC] ← RC|     
| 19 | 0000  | halt |     |
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.

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

Shift Left

Shift left. (opcode 5)
- Move bits to the left, padding with zeros as needed.
- \( 1234_{16} \ll 7 = 1A00_{16} \)

```
0 0 0 1 1 0 1 1 0 1 1 0 1 1 1 1 c
```

Shift Right

Shift right. (opcode 6)
- Move bits to the right, padding with sign bit as needed.
- \( 1234_{16} \gg 7 = 0024_{16} \)

```
0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 1 c
```

Bitwise AND

Logical AND. (opcode B)
- Logic operations are BITWISE.
- \( 0024_{16} \& 0001_{16} = 0000_{16} \)

```
0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 x \& y = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 x \& = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```
Shifting and Masking

**Shift and mask:** get the 7th bit of 1234.
- Compute $1234_{16} >> 7 = 0024_{16}$.
- Compute $0024_{16} & 1_{16} = 0_{16}$.

![Binary Multiplication](image)

**Binary Multiplication**

| OA: 0003 | input |
| OB: 0009 | 9     |
| OC: 0000 | output |
| OD: 0000 | 0     |
| OE: 0001 | 1     |
| OF: 0010 | 16    |

<table>
<thead>
<tr>
<th>i = 16</th>
<th>16 bit words</th>
</tr>
</thead>
<tbody>
<tr>
<td>i --</td>
<td>a &lt;&lt; i</td>
</tr>
<tr>
<td>b &gt;&gt; i</td>
<td></td>
</tr>
<tr>
<td>b_i</td>
<td>i^th bit of b</td>
</tr>
<tr>
<td></td>
<td>add a &lt;&lt; i to sum</td>
</tr>
<tr>
<td></td>
<td>while (i &gt; 0);</td>
</tr>
</tbody>
</table>

TOY Reference Card

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

Useful TOY "Idioms"

**Jump absolute.**
- Always jump to a fixed memory address.
  - branch if zero with register 0 and destination
  - register 0 is always 0

**Register copy (or move).**
- No instruction that transfers contents of one register into another.
- Pseudo-instruction that simulates copy:
  - add with register 0 as one of two source registers

**No-op.**
- Instruction that does nothing.
- Plays the role of whitespace in C programs.
  - numerous other possibilities!
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.

ENIAC, Ester Gerston (left), Gloria Gordon (right)

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

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

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 program encoded in binary.
• Data and program 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.

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.