**Debugging**

You've written your program and it doesn't work.  
**Now what?**

What do you do when you’re lost in a city?  
- Drive around randomly and hope you find it?  
- Return to a known point and look at a map?

In debugging, the equivalent to looking at a map is **tracing** your program.
- Examine the sequence of instructions being executed.  
- Keep track of results being produced.  
- Compare result from each instruction to the **expected** result.

---

**Debugging Operations**

Any debugging environment should provide means to:
1. Display values in memory and registers.  
2. Deposit values in memory and registers.  
3. Execute instruction sequence in a program.  
4. Stop execution when desired.

Different programming levels offer different tools.
- High-level languages (C, Java, ...) usually have source-code debugging tools.  
- For debugging at the machine instruction level:
  - Simulator
    - any universal computing device can emulate another UCD
  - operating system "monitor" tools
  - in-circuit emulators (ICE)  
    - plug-in hardware replacements that give instruction-level control

---

**Types of Errors**

**Syntax Errors**
- You made a typing error that resulted in an illegal operation.  
- Not usually an issue with machine language, because almost any bit pattern corresponds to some legal instruction.  
- In high-level languages, these are often caught during the translation from language to machine code.

**Logic Errors**
- Your program is legal, but wrong, so the results don’t match the problem statement.  
- Trace the program to see what’s really happening and determine how to get the proper behavior.

**Data Errors**
- Input data is different than what you expected.  
- Test the program with a wide variety of inputs.
Tell the simulator to stop executing when it reaches 0001101101111111.

Breakpoints
- Tell the simulator to stop executing when it reaches a specific instruction.
- Check overall results at specific points in the program.
- Let you quickly execute sequences to get a high-level overview of the execution behavior.
- Quickly execute sequences that you believe are correct.

Watchpoints (not available in PennSim)
- Tell the simulator to stop when a register or memory location changes or when it equals a specific value.
- Useful when you don’t know where or when a value is changed.

Example 1: Multiply
This program is supposed to multiply the two unsigned integers in R4 and R5.

Example 2: Summing an Array of Numbers
This program is supposed to sum the numbers stored in 10 locations beginning with x3100, leaving the result in R1.

Example 3: Looking for a 5
This program is supposed to set R0=1 if there's a 5 in one of ten memory locations, starting at x3100.

Debugging the Multiply Program

Debugging the Summing Program

Debugging the Summing Program
Running the the data below yields R1 = x0024, but the sum should be x8135. What happened?

Example 3: Looking for a 5
This program is supposed to set R0=1 if there's a 5 in one of ten memory locations, starting at x3100.

Else, it should set R0 to 0.

Loading contents of M[x3100], not address.
Change opcode of x3003 from 0010 (LD) to 1110 (LEA).
Debugging the Fives Program

Running the program with a 5 in location x3108 results in $R0 = 0$, not $R0 = 1$. What happened?

Address Contents

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>x3100</td>
<td>9</td>
</tr>
<tr>
<td>x3101</td>
<td>7</td>
</tr>
<tr>
<td>x3102</td>
<td>32</td>
</tr>
<tr>
<td>x3103</td>
<td>-6</td>
</tr>
<tr>
<td>x3104</td>
<td>0</td>
</tr>
<tr>
<td>x3105</td>
<td>19</td>
</tr>
<tr>
<td>x3106</td>
<td>6</td>
</tr>
<tr>
<td>x3107</td>
<td>13</td>
</tr>
<tr>
<td>x3108</td>
<td>5</td>
</tr>
<tr>
<td>x3109</td>
<td>61</td>
</tr>
</tbody>
</table>

Perhaps we didn’t... Put a breakpoint at x3007 to see how many times we branch back.

Branch uses condition code set by loading $R2$ with $M[R4]$, not by decrementing $R3$. Swap x300B and x300C, or remove x300C and branch back to x3007.

---

Example 4: Finding First 1 in a Word

This program is supposed to return (in $R1$) the bit position of the first 1 in a word. The address of the word is in location x3009 (just past the end of the program). If there are no ones, $R1$ should be set to –1.

$R1 = 15$

$R2 = data$

$R2[15] = 1?$

Decrement $R1$

Shift $R2$ left one bit

HALT

Branch uses condition code set by loading $R2$ with $M[R4]$, not by decrementing $R3$. Swap x300B and x300C, or remove x300C and branch back to x3007.

---

Debugging: Lessons Learned

Trace program to see what’s going on.

- Breakpoints, single-stepping

When tracing, make sure to notice what’s really happening, not what you think should happen.

- In summing program, it would be easy to not notice that address x3107 was loaded instead of x3100.

Test your program using a variety of input data.

- In Examples 3 and 4, the program works for many data sets.

- Be sure to test extreme cases (all ones, no ones, ...).