# Property based testing with hypothesis

Aim: develop a proof of concept for CPU testing using property testing frameworks.

## Hypothesis

We use the hypothesis library to generate the randomised programs because it has useful built in strategies, helpful errors messages and is great for discovering unknown corner-cases.

## Tasks

1. Build spike, toolchain + pk and run a hello world program in C
   1. This is in the ‘setup\_initial’ file
2. Understand how to use basic concepts in hypothesis (from intro tutorial)
3. Test harness to run a given test on two emulators and compare outputs (following the flow from the notes)
4. Create a randomised linear program
5. Create a randomised program with branching

## Creating a randomised linear program

1. Define an assembly template

This is the assembly file for the hello.c program:

Graphical user interface, text, application

Description automatically generated

Lines 1-6: always needed

Lines 7-10: list any read only data (rodata)

Lines 11-15: have a main

Lines 16-28: instructions that we’ll randomise later

Line 30: will be removed, just an identifier

## Choosing linear instructions

* Could have any of the following:
  + Load and store
  + Arithmetic
  + Bitwise instructions

We can just stick to the load and store instructions (just chose one) because this suffices as a ‘proof of concept’. The implementation of more concepts doesn’t seem to add much more to the overall aim.

## Class hierarchy:

* Parent object: a block
* Specific instructions that inherit from ‘block’, including ‘instruction’ which is a single instruction, ‘linear’ which is a list of instructions, and blocks for higher level concepts like loops.
  + Rationale: a program can be reduced to a list of instructions. We want hypothesis to go through all combinations of instructions as defined in ISA. However, the jump instruction can create programs that don’t halt. So we replace ‘jump’ blocks (or anything that can cause an issue) with a variety of high level blocks that incorporate it ie.g. procedures and if statements

## Screenshots: program works to create randomised program with branching

#### For loop

Text

Description automatically generated

#### If inside a for loop

Text

Description automatically generated

#### If with instruction subblocks

Text

Description automatically generated

## Improvements with strategies

### Merging strategies implementation

The first implementation of the program had two main aims for strategies:

1. To choose a sub-block
2. Populate the member variables within a block e.g. register values and offsets

We could merge this into a single function that creates a list of blocks with pointers to the next block, and then we can have a separate function that walks the tree.

It also calls strategies recursively because each time we create a sub-block, strategies is called again. The issues with this is:

1. Lots of memory is used
2. When detecting a bug, strategies tells us which parent node it comes from, even if the bug itself comes from a leaf node
   1. One solution is to inspect the log that spike gives us. This means we are not taking advantage of the shrinking in strategies

However, if we have a single function where hypothesis operates from (i.e. only one decorator in the whole program), we can give it specific shrinking instructions, using [What you can generate and how — Hypothesis 6.23.1 documentation](https://hypothesis.readthedocs.io/en/latest/data.html#interactive-draw).

So an alternative program would:

1. Have a single function that fills a block and decides the type of any sub blocks
2. Have a get\_asm() method, this is called recursively
3. All classes have member data that includes a pointer to the next node
4. Have a block class called Main

The flow would be almost identical to before, but we’re just calling the function in the constructor rather than using @given each time.

Pseudocode for this:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Here, could use st.deferred() or st.data() – see documentation for details ([What you can generate and how — Hypothesis 6.23.1 documentation](https://hypothesis.readthedocs.io/en/latest/data.html#interactive-draw))

Another small optimisation for the shrinking: use st.one\_of rather than st.sampled\_from, with the liner instructions as the first parameter. This is because when shrinking, st.one\_of looks at the earlier parameters first. It would be useful to look at the leaf nodes first when finding an issue.

### Reducing repeated code

This still has lots of case statements/if instructions that could be merged.

Another idea is to use the parent block more:

Text

Description automatically generated

Shape

Description automatically generated with medium confidence

## Useful links

<https://shakti.org.in/docs/risc-v-asm-manual.pdf>

[What you can generate and how — Hypothesis 6.21.1 documentation](https://hypothesis.readthedocs.io/en/latest/data.html)