3XB3 L3 - Assembly Lab

Luigi Quattrociocchi (`quattrl@mcmaster.ca`)
Dennis Fong (`fongd1@mcmaster.ca`)

Avenue Group: 42

December 8, 2022

F$_1$: Global Variables and First Visits

Manual translation

The following is a translation of `simple.py` into Pep/9.

```
      BR      main
x:    .BLOCK 2
main: LDWA 3, i
      ADDA 2, i
      STWA x, d
      DECO x, d
      .END
```

The following is a translation of `add_sub.py` into Pep/9. Note that there are multiple instances of `STWA` immediately followed by `LDWA`. These pairs are redundant and can be removed.

```
      BR      main
_UNIV:    .WORD 42
variable: .WORD 3
value:    .BLOCK 2
result:   .BLOCK 2
main: DECI value, d
      LDWA value, d
      ADDA _UNIV, d
      STWA result, d
      LDWA result, d
      SUBA variable, d
      STWA result, d
      LDWA result, d
      SUBA 1, i
      STWA result, d
      DECO result, d
      .END
```

Global vs Local variables

In python, global variables are any variables that are declared at the top level of the program (not within any function). These variables are typically referred to as ”static” variables, and they are stored within the data or bss sections of the binary. Local variables in python exist at the function scope level. These variables are stored on the call stack, and they exist until the function containing them returns and the scope is destroyed.

NOP1 instructions

The translator emits `NOP1` instructions after the entry point label for seemingly no reason. However, this instruction is necessary to avoid syntax errors due to how Pep/9 works. Every label must have an instruction on the same line, otherwise the assembler gives the error `ERROR: Must have mnemonic or dot command after symbol definition.` . Since we can't guarantee that an instruction will follow the label (the souce file could be empty), we emit a dummy instruction regardless.

Explanation of Visitors and Generators

Visitors and Generators are two concepts used in compilers after the AST is parsed. Visitors (from the Visitor design pattern) are commonly used in traversals of recursive structures (such as trees) in which the behaviour of the visitation is polymorphic and independent of the structure itself. Most compilers (notably clang) use the visitor pattern to traverse their AST. Each node in the tree is a visitor which is responsible for walking the tree and visiting its children. After the visitors are used to traverse the syntax tree and accumulate the instructions that should be emitted, the generators are used to emit said instructions. Generators are useful because if implemented correctly, cross platform support can easily be achieved. In our case, the bytecode created by our visitors are already basically in Pep/9 format, so the generators emit them verbatim.

Limitations of Current Translation

Currently the translator is very simple, leading to a couple common software engineering pitfalls. Firstly, the code is not very scalable. Everything is dumped to stdout rather than a specific file descriptor, the visitors violate the single responsibility principle by directly interpolating the string instructions to be emitted, and and there is no room for adding optimization passes without violating the open-closed principle. Secondly (and probably less importantly), the code is emitted in a very naive way. The `NOP1` instruction after the entry label, constant still stored in memory (and are not actually constant), and the translation is very literal and error prone (for instance, the length of labels is not constrained).