Lab 3: Self-modifying code

Objectives

- to become more familiar with programming for the HYMN architecture.
- to learn about self-modifying programs.

A **self-modifying program** is a program that modifies itself. Computer scientists regard such programs as very bad form. Still, though, such programs can provide for some interesting behaviors.

As an example of a use for self-modifying programs, consider accessing arrays on HYMN. For example, suppose we want a program that reads an integer i and then displays the ith number of an array. Intuitively, such a program is impossible to write, since the address we want to read from depends on what the user enters, but the only way to read within memory is using an instruction (such as LOAD) where the memory address is directly encoded in the instruction.

A program might, however, create the LOAD instruction upon reading i, and then it could execute this created LOAD instruction. The following code accomplishes this.

```
       READ
       ADD loada     # We compute the next instruction...
       STORE loada   # and then store it so that...
loada: LOAD a        # we can now execute the computed instruction.
       WRITE
       HALT

a:     2  # here's the array (I've placed prime numbers into it)
       3
       5
       7
       11
       13
       17
```

Such a program is self-modifying, since the program actually changes the instructions that it executes (in this case, the instruction at `loada`) — as it executes itself.

For the following problems, you should use assembly language within the HYMN simulator, *SimHYMN*. To start *SimHYMN*, go to the K-menu and select “150cpusim” from the CS Programs submenu. To write assembly language in *SimHYMN*, select Show Editor under the Assembler menu.

1. Write a program (in `CS210/Lab3/primes.hymn`) that prints the first 10 prime numbers. Your program should work by simply displaying ten numbers occurring in a pre-defined array in memory.

2. A **virus**, which copies itself from one location of memory to another, is a particularly interesting form of self-modifying program. Because of HYMN's very limited memory capacity, we can write only a partial virus.

Write a HYMN program (in `CS210/Lab3/virus.hymn`) that copies its own instructions in addresses 0 to 9 into addresses 20 to 29. As it copies, it should to add 20 to all of the memory addresses appearing in the program, since the child program appears 20 bytes later in memory than the original. Do not worry about minor copying errors (which may occur when you copy instructions that change as the program executes).

In addition to the introduction and conclusion that should be in all your lab reports, the body of your lab report need contain only the code for your two programs, commented appropriately. Also, run “`handincs 210 3`” to submit your code electronically.