# Assembly Lab


- Authors:
  - Angela Zeng, zenga8@mcmaster.ca
  - Emma Wigglesworth, wigglee@mcmaster.ca
- Group ID on Avenue: 50
- Gitlab URL: https://gitlab.cas.mcmaster.ca/zenga8/l3-assembly


# F1: Global Variables and First Visits

In [ ]:
```
# add_sub.py

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

In [ ]:
```
# simple.py
         BR      program
x:   .BLOCK  2
program: LDWA    3, i
         ADDA    2,i
         STWA    x, d
         DECO    x, d
         .END
```

**The translator uses NOP1 instructions. Any ideas why?**

The NOP instruction is an instruction that essentially does nothing. NOP1 is used in case there is no instruction after a label (in this case, t1). This will ensure the program is assembled correctly since its not guaranteed an instruction will follow after the label.

**Look at the code of translator.py**
**It relies on two visitors and two generators. Explain the role of each element.**

In translator.py, we use two visitors and two generators. The visitor traverses the AST recursively to extract information. There are two visitors. The GlobalVariableExtraction class records the variable names (to allocate later in the generator). The TopLevelProgram class gives different

assembly instructions depending on what node it visits. Generators are responsible for printing the instructions in PEP/9 assembly. You can also have backend generators for other assembly languages. The EntryPoint class generates the assembly instructions produced by the TopLevelProgram visitor. The StaticMemoryAllocation class reserves the memory for the variables visited by the GlobalVariableExtraction.

**Explain the limitations of the current translation code in terms of software engineering.** One limitation of the translation code is that it only works for Pep/9 so its very limited in terms of translatability. To overcome this, you can make a general translator interface so a class implementing that interface can be used to generate platform specific assembly. Another limitation is that because the output to assembly is printed instead of being written to a file, it becomes more difficult to debug the translator because if we were to print something (for debugging purposes), it would get mixed in with the printed output to assembly.

# Self-reflection questions

As part of the self-reflection dimension of an experiential course, each member of the group is expected to answer to the following four questions:

- How much did you know about the subject before we started? (backward)
- What did/do you find frustrating about this assignment? (inward)
- If you were the instructor, what comments would you make about this piece? (outward)
- What would you change if you had a chance to do this assignment over again? (forward)