Using the sbasm.py Assembler

- Assembler is written in python.; provided along w/ Lab2
- automatically generates machine code for mv, mvi, add, sub, ld, st, mvnz, mvnc
- Supports .define symbols and labels. Can include data at the end of your code. using .word see example in Lab2.

Example Loop // code that uses a loop to make a delay.

```
MAIN:   mvi    r0, #8888
        mvi    r1, #1
        mv     r3, r7    // put loop address into r3
LOOP:   sub    r0, r1
        mvnz   r7, r3
        // delay loop is executed 8888 times
        // Note: mvnc would have executed 8889 times.
```

Example Subroutine

```
main( ) {
    int x,y;
    y=10;

  x=my_sub(y);
    :
  int my_sub(int x){
      return (x+x);
  }
```

- Every processor needs to have a method of calling a subroutine and then returning back to the caller. In our case: ay.

```
MAIN:   mvi   r0, #10        // r0 is y
        mv    r5, r7         // set return address
        mvi   r7, my_sub     // called sub routine.
          ⋮

        // input must be in r0
        // result is returned in r0

my_sub: mvi   r1, #1
        add   r0, r0
        add   r5, r1         // adjust the return addr.
        add   r5, r1         // adjust the return addr.
        mv    r7, r5         // return.
```

- In this case our subroutine is trivial and uses few registers. But in general you may need several registers in a subroutine.
- If these registers are being used to hold data in the caller, then we cannot change them in the subroutine.
- Solution: we need to save the registers at the start of the sub-routine, and then restore them before returning. We can use the concept of a stack data structure.