Functions in Hmmm Assembly

Functions in Python vs. assembly

```
r1 = int(input())          0  read   r1
r13 = f(r1)                1  calln  r14, 4
print(r13)                 2  write  r13
                           3  halt
def f(r1):                 4  copy   r13, r1
    r13 = r1*(r1-1)        5  addn   r13, -1
    return r13             6  mul    r13,r1,r13
                           7  jumpr  r14
```

```
Write a NEW FUNCTION that returns 1 if the input
is > 0 and 2 if the input is <= 0
```

Why Functions?

Function is just a block of computation, no real magic. We can use "jumpn" to accomplish the same goal.

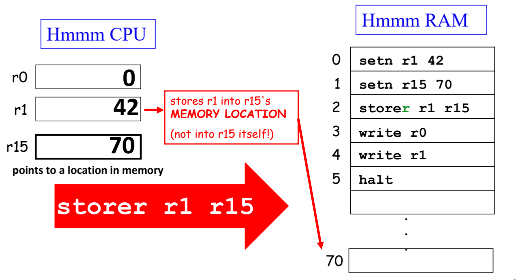
```
# computes n*(n-1) without function
0   read  r1
1   jumpn 4
2   write r13
3   halt
4   copy  r13, r1
5   addn  r13, -1
6   mul   r13,r1,r13
7   jumpn 2
```

This program does exactly the same as the function before without function ("calln"). We "hard-coded" the return address "jumpn 2."

But, what if another place in the program needs this part of the computation??? "jumpn 2" will lead to a wrong place! "jumpr r14" (thus function) will be needed!

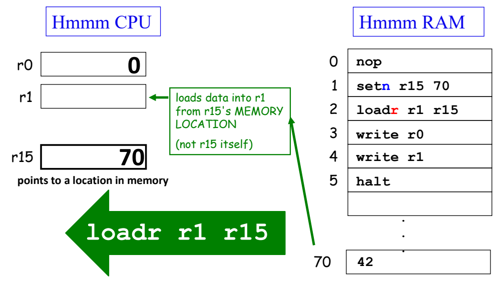
storer stores TO memory

storer rX rY # stores the content of rX into memory address held in rY



loadr loads FROM memory

loadr rX rY # load value into rX from memory address held in rY



A function example

```
0    read    r1          # Get the "x" for our function

1    setn   r15, 70      # Set the stack pointer, (i.e.,
                         # load address of stack into r15)
2    storer r1, r15      # Store r1, since f overwrites it

3    calln   r14, 7      # Call our function f(x)
                         # Set r14 to be 4, next PC
4    loadr   r1, r15     # Load r1 back in place

5     write  r13        # Print result
6     halt              # Stop the program

7    addn    r1, 1       # Compute f(x) = x + 1
8    copy    r13,r1      # Save result into r13
9    jumpr   r14         # Finished function, jump back
```

# Try 18_fun_example.hmmm

Are there any difference between instructions and values (numbers)?

From computers' point of view, the memory has separate dedicated area for data and instructions. So the computer knows which piece is data, which piece is instruction. But human beings can't tell data from instructions just from its form.

The program on the previous pages are compiled into machine form in red.

```
0 : 0110 0000 0000 0000     # 0  nop
1 : 0001 1111 0100 0110     # 1  setn r15, 70
2 : 0100 0001 1111 0000     # 2  loadr r1, r15
3 : 0000 0000 0000 0010     # 3  write r0
4 : 0000 0001 0000 0010     # 4  write r1
5 : 0000 0000 0000 0000     # 5  halt
...
70: 0000 0000 0010 1010     # 70: integer 42
```

Jumps in Hmmm

- *Unconditional jump*
  - – jumpn n # jump to line n (set PC to n)
- *Conditional jumps*
  - – jeqzn rx n # if reg x == 0, jump to line n
  - – jnezn rx n # if reg x != 0, jump to line n
  - – jltzn rx n # if reg x < 0, jump to line n
  - – jgtzn # if reg x > 0, jump to line n
- *Indirect jump*
  - – jumpr rx # jump to the value in reg x