CS 5 Green slides invaded by farm animals!

Hey! These are our slides!

Yeah, what are you doing here!?

The CS 5 gold and black aliens usually have a cow when someone tries to hog their slides!
Gauging your Workload

On your worksheet…
  On the front, bottom-right corner…

How many hours, outside of class or lab time, did you spend on this course this past week?
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td>Stop!</td>
<td></td>
</tr>
<tr>
<td>read rX</td>
<td>Place user input in register rX</td>
<td></td>
</tr>
<tr>
<td>write rX</td>
<td>Print contents of register rX</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td>Do nothing</td>
<td></td>
</tr>
</tbody>
</table>

### System instructions

### Setting register data

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setn rX N</td>
<td>Set register rX equal to the integer N (-128 to +127)</td>
</tr>
<tr>
<td>addn rX N</td>
<td>Add integer N (-128 to 127) to register rX</td>
</tr>
<tr>
<td>copy rX rY</td>
<td>Set rX = rY</td>
</tr>
<tr>
<td>mov</td>
<td></td>
</tr>
</tbody>
</table>

### Arithmetic

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add rX rY rZ</td>
<td>Set rX = rY + rZ</td>
</tr>
<tr>
<td>sub rX rY rZ</td>
<td>Set rX = rY - rZ</td>
</tr>
<tr>
<td>neg rX rY</td>
<td>Set rX = -rY</td>
</tr>
<tr>
<td>mul rX rY rZ</td>
<td>Set rX = rY * rZ</td>
</tr>
<tr>
<td>div rX rY rZ</td>
<td>Set rX = rY / rZ (integer division; no remainder)</td>
</tr>
<tr>
<td>mod rX rY rZ</td>
<td>Set rX = rY % rZ (returns the remainder of integer division)</td>
</tr>
</tbody>
</table>

### Jumps!

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jumpn N</td>
<td>Set program counter to address N</td>
</tr>
<tr>
<td>jumpr X</td>
<td>Set program counter to address in rX</td>
</tr>
<tr>
<td>jeqzn rX N</td>
<td>If rX == 0, then jump to line N</td>
</tr>
<tr>
<td>jnezr X N</td>
<td>If rX != 0, then jump to line N</td>
</tr>
<tr>
<td>jgtzn rX N</td>
<td>If rX &gt; 0, then jump to line N</td>
</tr>
<tr>
<td>jltzn rX N</td>
<td>If rX &lt; 0, then jump to line N</td>
</tr>
<tr>
<td>calln rX N</td>
<td>Copy the next address into rX and then jump to mem. addr. N</td>
</tr>
<tr>
<td>jump</td>
<td></td>
</tr>
<tr>
<td>jeqz</td>
<td></td>
</tr>
<tr>
<td>jnez</td>
<td></td>
</tr>
<tr>
<td>jgtz</td>
<td></td>
</tr>
<tr>
<td>jltz</td>
<td></td>
</tr>
<tr>
<td>call</td>
<td></td>
</tr>
</tbody>
</table>

### Interacting with memory (RAM)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadn rX N</td>
<td>Load register rX with the contents of memory address N</td>
</tr>
<tr>
<td>storen rX N</td>
<td>Store contents of register rX into memory address N</td>
</tr>
<tr>
<td>loadr rX rY</td>
<td>Load register rX with data from the address location held in reg. rY</td>
</tr>
<tr>
<td>storer rX rY</td>
<td>Store contents of register rX into memory address held in reg. rY</td>
</tr>
<tr>
<td>loadi, load</td>
<td></td>
</tr>
<tr>
<td>storei, store</td>
<td></td>
</tr>
</tbody>
</table>
### A function call in Python:

```python
def main():
    r1 = input()
    result = factorial(r1)
    print result

def factorial(r1):
    # do work
    return result
```

### Hmmmm’s `calln` operation:

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>read r1</td>
</tr>
<tr>
<td>1</td>
<td>calln r14 4</td>
</tr>
<tr>
<td>2</td>
<td>write r13</td>
</tr>
<tr>
<td>3</td>
<td>halt</td>
</tr>
<tr>
<td>4</td>
<td>do stuff and</td>
</tr>
<tr>
<td>5</td>
<td>answer in r13</td>
</tr>
<tr>
<td>6</td>
<td>jump r14</td>
</tr>
</tbody>
</table>

Who ya gonna call?
def main():
    r1 = input()
    result = factorial(r1)
    print(result)

def factorial(r1):
    # do work
    return result
A function call in Python:

```python
def main():
    r1 = input()
    result = factorial(r1)
    print(result)

def factorial(r1):
    # do work
    return result
```

Hmmm’s `call` operation:

- Functions always receive their input in register `r1` (and `r2`, `r3`, and so forth if there more inputs)
- Functions return their answers in register `r13`
- Functions `jump r14` to return to where they were called

United Nations Resolution 424242

0 read `r1`
1 `calln r14 4`
2 write `r13`
3 `halt`

4 `do stuff and`
5 `answer in r13`
6 `jump r14`
What does this do?

Try \( r_1 = 1, r_1 = 3 \)

### main function

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>read r1</td>
<td># get number from user and put in r1</td>
</tr>
<tr>
<td>01</td>
<td>jeqzn r1 5</td>
<td># if r1==0, then jump to line 5</td>
</tr>
<tr>
<td>02</td>
<td>calln r14 6</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>write r13</td>
<td># convention to store answer in r13</td>
</tr>
<tr>
<td>04</td>
<td>jumpn 0</td>
<td># jump to line 0</td>
</tr>
<tr>
<td>05</td>
<td>halt</td>
<td># stop!</td>
</tr>
<tr>
<td>06</td>
<td>setn r13 1</td>
<td># r13 = 1</td>
</tr>
<tr>
<td>07</td>
<td>setn r2 2</td>
<td># r2 = 2</td>
</tr>
<tr>
<td>08</td>
<td>jeqzn r1 12</td>
<td># if r1==0, then jump to line 12</td>
</tr>
<tr>
<td>09</td>
<td>addn r1 -1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>mul r13 r13 r2</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>jumpn 8</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>jump r14</td>
<td># jump to address in r14</td>
</tr>
</tbody>
</table>

### helper function

Once you figure out what this helper function is doing, it will be more than a hare familiar!

Hmmm
def main():
    r1 = input()  # r1=3
    r13 = jessica(r1)  # jessica(3)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

r1=3
r1=3
print(r13)

In other words, hare-y up and figure out what main will return...
def main():
    r1 = input()  # r1=3
    r13 = jessica(r1)  # jessica(3)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1  # r1=4
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

You should be worried!
def main():
    r1 = input()  # r1=3
    r13 = jessica(r1)  # jessica(3)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):  # r1=3
    r1 = r1 + 1  # r1=4
    r13 = eliot(r1)  # eliot(4)
    r13 = r13 + r1
    return r13

def eliot(r1):  # r1=4
    r1 = r1 + 42  # r1=46
    r13 = r1 + 1
    return r13
def main():
    r1 = input() ←→ r1=3
    r13 = jessica(r1) ←→ jessica(3)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1 ←→ r1=4
    r13 = eliot(r1) ←→ eliot(4)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42 ←→ r1=46
    r13 = r1 + 1 ←→ r13=47
    return r13 ←→ return(47)
def main():
    r1 = input() ← r1=3
    r13 = jessica(r1) ← jessica(3)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1 ← r1=4
    r13 = eliot(r1) ← eliot(4)  r13=47
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42 ← r1=46
    r13 = r1 + 1 ← r13=47
    return r13 ← return(47)
Function Calls…

def main():
    r1 = input() ← r1=3
    r13 = jessica(r1) ← jessica(3)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1 ← r1=4
    r13 = eliot(r1) ← eliot(4)  r13=47
    r13 = r13 + r1 ← r13=??
    return r13

def eliot(r1):
    r1 = r1 + 42 ← r1=46
    r13 = r1 + 1 ← r13=47
    return r13 ← return(47)
def main():
    r1 = input() ← r1=3
    r13 = jessica(r1) ← jessica(3)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1 ← r1=4
    r13 = eliot(r1) ← eliot(4) ← r13=47
    r13 = r13 + r1 ← r13=51
    return r13 ← return(51)

def eliot(r1):
    r1 = r1 + 42 ← r1=46
    r13 = r1 + 1 ← r13=47
    return r13 ← return(47)
Function Calls…

def main():
    r1 = input()  # r1=3
    r13 = jessica(r1)  # jessica(3)  r13=51
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1  # r1=4
    r13 = eliot(r1)  # eliot(4)  r13=47
    r13 = r13 + r1  # r13=51
    return r13  # return(51)

def eliot(r1):
    r1 = r1 + 42  # r1=46
    r13 = r1 + 1  # r13=47
    return r13  # return(47)
def main():
    r1 = input()  # r1=3
    r13 = jessica(r1)  # jessica(3)  # r13=51
    r13 = r13 + r1  # r13=??
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1  # r1=4
    r13 = eliot(r1)  # eliot(4)  # r13=47
    r13 = r13 + r1  # r13=51
    return r13  # return(51)

def eliot(r1):
    r1 = r1 + 42  # r1=46
    r13 = r1 + 1  # r13=47
    return r13  # return(47)
def main():
    r1 = input()  # r1=3
    r13 = jessica(r1)  # jessica(3)  r13=51
    r13 = r13 + r1  # r13=54
    print(r13)  # 54
    return

def jessica(r1):
    r1 = r1 + 1  # r1=4
    r13 = eliot(r1)  # eliot(4)  r13=47
    r13 = r13 + r1  # r13=51
    return r13  # return(51)

def eliot(r1):
    r1 = r1 + 42  # r1=46
    r13 = r1 + 1  # r13=47
    return r13  # return(47)
Function Calls…

def main():
    r1 = input()  # r1=3
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

r1

r13

return

United Nations Resolution 424242

- Functions always receive their input in register \texttt{r1}
- Functions return their answers in register \texttt{r13}
- Functions jump to return to where they were called

Hmmm code up here!
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

Hmmm code up here!

United Nations Resolution 424242

- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

United Nations Resolution 424242
- Functions always receive their input in register \texttt{r1}
- Functions return their answers in register \texttt{r13}
- Functions jump \texttt{r14} to return to where they were called
The stack in RAM!

```
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
```

United Nations Resolution 424242

- Functions always receive their input in register `r1`
- Functions return their answers in register `r13`
- Functions jump `r14` to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r13 + r1
    return r13

The stack in RAM!

Hmmm code up here!

United Nations Resolution 424242
- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

United Nations Resolution 424242
• Functions always receive their input in register r1
• Functions return their answers in register r13
• Functions jump r14 to return to where they were called
Function Calls…

def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

United Nations Resolution 424242
- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump to return to where they were called
Function Calls…

def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

United Nations Resolution 424242

- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump r14 to return to where they were called
Function Calls...

```python
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
```

The stack in RAM!

United Nations Resolution 424242
- Functions always receive their input in register \texttt{r1}
- Functions return their answers in register \texttt{r13}
- Functions jump to \texttt{r14} to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r13 + r1
    return r13

United Nations Resolution 424242
• Functions always receive their input in register \texttt{r1}
• Functions return their answers in register \texttt{r13}
• Functions jump to return to where they were called
```python
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
```

**United Nations Resolution 424242**
- Functions always receive their input in register `r1`
- Functions return their answers in register `r13`
- Functions jump to `r14` to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

Hmmm code up here!

The stack in RAM!

r1

3 4

r13

return address

3

4

United Nations Resolution 424242

- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump r14 to return to where they were called
Function Calls...

```python
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
```

The stack in RAM!

- Functions always receive their input in register \texttt{r1}
- Functions return their answers in register \texttt{r13}
- Functions jump \texttt{r14} to return to where they were called

United Nations Resolution 424242
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

United Nations Resolution 424242
• Functions always receive their input in register r1
• Functions return their answers in register r13
• Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

United Nations Resolution 424242
- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump r14 to return to where they were called

United Nations Resolution 424242
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

- Functions always receive their input in register \texttt{r1}
- Functions return their answers in register \texttt{r13}
- Functions jump \texttt{r14} to return to where they were called

United Nations Resolution 424242
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

United Nations Resolution 424242
• Functions always receive their input in register r1
• Functions return their answers in register r13
• Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

United Nations Resolution 424242
- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

United Nations Resolution 424242
• Functions always receive their input in register r1
• Functions return their answers in register r13
• Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

United Nations Resolution 424242

- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

United Nations Resolution 424242

- Functions always receive their input in register \texttt{r1}
- Functions return their answers in register \texttt{r13}
- Functions jump to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

United Nations Resolution 424242

- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

Hmmm code up here!

The stack in RAM!

- Functions always receive their input in register $r1$
- Functions return their answers in register $r13$
- Functions jump to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

United Nations Resolution 424242
• Functions always receive their input in register r1
• Functions return their answers in register r13
• Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

United Nations Resolution 424242
• Functions always receive their input in register \texttt{r1}
• Functions return their answers in register \texttt{r13}
• Functions jump to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
Function Calls…

```python
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
```

The stack in RAM!

- Functions always receive their input in register `r1`
- Functions return their answers in register `r13`
- Functions jump to return to where they were called

United Nations Resolution 424242
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r13 + 1
    return r13

The stack in RAM!

United Nations Resolution 424242
- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r13 + r1
    return r13

United Nations Resolution 424242
- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump to r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

Hmmm code up here!

United Nations Resolution 424242
• Functions always receive their input in register r1
• Functions return their answers in register r13
• Functions jump r14 to return to where they were called
Function Calls...

```python
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
```

The stack in RAM!

- Functions always receive their input in register `r1`.
- Functions return their answers in register `r13`.
- Functions jump to return to where they were called.

United Nations Resolution 424242
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

United Nations Resolution 424242
• Functions always receive their input in register r1
• Functions return their answers in register r13
• Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

United Nations Resolution 424242

- Functions always receive their input in register r1
- Functions return their answers in register r13
- Functions jump r14 to return to where they were called
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

The stack in RAM!

Hmmm code up here!

- Functions always receive their input in register \( r1 \)
- Functions return their answers in register \( r13 \)
- Functions jump \( r14 \) to return to where they were called
store goes TO memory

Hmmm CPU

r1

...  

r15

42

Hmmm RAM

0  
1  
2  
3  
4  
5  

read r1

.  

storer r1 r15

.  

...  

indirect store

storer r1 r15  

aliens 42

42  
43  

...
**load** comes FROM memory

---

**Hmmm CPU**

```
  r1  
  ...  
  r15  
      42
```

**Hmmm RAM**

```
  0  
  1  
  2  
  3  
  4  
  5  
  6  
  7  
      loadr r1 r15
  8  
      stor r1 r15
  ...
  ...
```

*indirect load*
Amended UN Resolution

424242

- Functions always receive their input in register \texttt{r1}
- Functions return their answers in register \texttt{r13}
- Functions jump to return to where they were called
- Register \texttt{r15} stores the stack pointer
def main():
    r1 = input()
    r13 = jessica(r1)
    r13 = r13 + r1
    print(r13)
    return

def jessica(r1):
    r1 = r1 + 1
    r13 = eliot(r1)
    r13 = r13 + r1
    return r13

def eliot(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
def fac(N):
    if N == 0:
        return 1
    else:
        return N * fac(N-1)

fac(5)
5 * fac(4)
4 * fac(3)
3 * fac(2)
2 * fac(1)
1

“The Stack”
Remembers all of the individual calls to fac
Factorial via Recursion...

**Python**

```python
N = input()
answer = fac(N)
print(answer)
```

```python
def fac(N):
    '''recursive factorial!'''
    if N == 0:
        return 1
    else:
        REC = fac(N-1)
        return N*REC
```

This is same as `return N*f(N-1)` but done in 2 steps...

First let's try $N=1$ and then $N=3$
Python

```python
N = input()
answer = fac(N)
print(answer)

def fac(N):
    """recursive factorial!""
    if N == 0:
        return 1
    else:
        REC = fac(N-1)
        return N*REC
```

Hmmm

```
00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 stor r1 r15
09 addn r15 1
10 stor r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jump r14
```

- **r13** is the answer
- **r14** is the “return address”
- **r15** is the “stack pointer”
### Python

```python
N = input()
answer = fac(N)
print(answer)
```

### Hmmm

```hmmm
00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
```

```hmmm
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 store r1 r15
09 addn r15 1
10 store r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jump r14
```

**match each piece of Python code with the Hmmm assembly code that implements it**

- **N = input()**
  - Prepare for function call! All precious belongings get on the stack!
  - Store set up input
  - Load base case
  - Store recursive call

- **answer = fac(N)**
  - Function call over! All precious belongings back into their registers!
  - Load set up input
  - Store

- **print(answer)**
  - Set `r13` is the `answer`
  - Store

- **def fac(N):**
  - """recursive factorial!""
  - if `N == 0`:
    - Return `1`
  - else:
    - `REC = fac(N-1)`
    - Return `N*REC`

- **r13** is the `answer`
- **r14** is the "return address"
- **r15** is the "stack pointer"
Implementing functions

non-destructively!

(0) Use r15 as the stack pointer.

(1) Before the function call, Store all valuable data to the stack

(2) Get r1, (r2), (r3), … ready as function “inputs”.

(3) Make the function call. The result, if any, will be in r13.

(4) After the function call, Load valuable data back from the stack

To store the return address:

store the return address r14 and the inputs: r1, (r2), (r3)

There may or may not be some lines of code necessary to do this.

line # of the function

For each item stored
Worksheet

Write down what happens in the registers and memory (the stack) as this program runs…

Program (“low part of RAM”)

```
00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jmpsr r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jmpsr r14
```

**CPU Registers**

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>0</td>
</tr>
<tr>
<td>r1</td>
<td>2</td>
</tr>
<tr>
<td>r13</td>
<td></td>
</tr>
<tr>
<td>r14</td>
<td></td>
</tr>
<tr>
<td>r15</td>
<td></td>
</tr>
</tbody>
</table>

**Memory (“high part of RAM”)**

<table>
<thead>
<tr>
<th>Memory Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
</tr>
<tr>
<td>43</td>
</tr>
<tr>
<td>44</td>
</tr>
<tr>
<td>45</td>
</tr>
<tr>
<td>46</td>
</tr>
<tr>
<td>47</td>
</tr>
<tr>
<td>48</td>
</tr>
<tr>
<td>49</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>51</td>
</tr>
<tr>
<td>52</td>
</tr>
</tbody>
</table>

**How low *could* we start the stack? How deep does the stack get? What are the possible values of r14?**
Write down what happens in the registers and memory (the stack) as this program runs…

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt

05 jnezn r1 8
06 setn r13 1
07 jumper r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jumper r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Worksheet

Write down what happens in the registers and memory (the stack) as this program runs...

Program ("low part of RAM")

```
00  read  r1
01  setn  r15  42
02  calln  r14  5
03  write  r13
04  halt

05  jnezn  r1  8
06  setn  r13  1
07  jumper  r14
08  storer  r1  r15
09  addn  r15  1
10  storer  r14  r15
11  addn  r15  1
12  addn  r1  -1
13  calln  r14  5
14  addn  r15  -1
15  loadr  r14  r15
16  addn  r15  -1
17  loadr  r1  r15
18  mul  r13  r13  r1
19  jumper  r14
```

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Worksheet

Write down what happens in the registers and memory (the stack) as this program runs...

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt

05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jump r14

The input is 2.

CPU Registers
with labels

always-0 register

input: N

result, return value

return address (line #)

stack pointer

Memory ("high part of RAM")
"the stack"

42
43
44
45
46
47
48
49
50
51
52

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt

05 jnezn r1 8
06 setn r13 1
07 jump r14
08 stor r1 r15
09 addn r15 1
10 stor r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jump r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program ("low part of RAM")

```assembly
00  read  r1
01  setn  r15  42
02  calln  r14  5
03  write  r13
04  halt
05  jnezn  r1  8
06  setn  r13  1
07  jmprr  r14
08  storerr  r1  r15
09  addnr  r15  1
10  storerr  r14  r15
11  addnr  r15  1
12  addnr  r1  -1
13  calln  r14  5
14  addnr  r15  -1
15  loadrr  r14  r15
16  addnr  r15  -1
17  loadrr  r1  r15
18  mulrr  r13  r13  r1
19  jmprr  r14
```

**CPU Registers with labels**

- **r0**: always-0 register
- **r1**: input: N
- **r13**: result, return value
- **r14**: return address (line #)
- **r15**: stack pointer

**Memory ("high part of RAM")**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How low *could* we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program ("low part of RAM")

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jumper r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jumper r14

The input is 2.

CPU Registers with labels

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>0</td>
</tr>
<tr>
<td>r1</td>
<td>2</td>
</tr>
<tr>
<td>r13</td>
<td></td>
</tr>
<tr>
<td>r14</td>
<td>03</td>
</tr>
<tr>
<td>r15</td>
<td>43</td>
</tr>
</tbody>
</table>

always-0 register
input: N
result, return value
return address (line #)
stack pointer

Memory ("high part of RAM")
"the stack"

42 2
43 03
44
45
46
47
48
49
50
51
52

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnez r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 1
17 load r1 r15
18 mul r13 r13 r1
19 jump r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt

05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jump r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program ("low part of RAM")

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jumper r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jumper r14

The input is 2.

CPU Registers
with labels

always-0 register

input: N

result, return value

return address (line #)

stack pointer

Memory ("high part of RAM")
"the stack"

42
43
44
45
46
47
48
49
50
51
52

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

**Program (“low part of RAM”)**

```
00  read  r1
01  setn  r15  42
02  calln  r14  5
03  write  r13
04  halt

05  jnezn  r1  8
06  setn  r13  1
07  jumpR  r14
08  storer  r1  r15
09  addn  r15  1
10  storer  r14  r15
11  addn  r15  1
12  addn  r1  -1
13  calln  r14  5
14  addn  r15  -1
15  loadr  r14  r15
16  addn  r15  -1
17  loadr  r1  r15
18  mul  r13  r13  r1
19  jumpR  r14
```

**CPU Registers with labels**

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>0</td>
</tr>
<tr>
<td>r1</td>
<td>1</td>
</tr>
<tr>
<td>r13</td>
<td></td>
</tr>
<tr>
<td>r14</td>
<td>14</td>
</tr>
<tr>
<td>r15</td>
<td>44</td>
</tr>
</tbody>
</table>

**Memory (“high part of RAM”)**

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>43</td>
<td>03</td>
</tr>
</tbody>
</table>
```

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program ("low part of RAM")

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jumper r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jumper r14

The input is 2.

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program ("low part of RAM")

- The input is 2.
- CPU Registers with labels
- Memory ("high part of RAM")

```
00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jumper r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jumper r14
```

Always-0 register
- Input: N
- Result, return value
- Return address (line #)
- Stack pointer

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jump r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Worksheet

Write down what happens in the registers and memory (the stack) as this program runs…

Program (“low part of RAM”)

<table>
<thead>
<tr>
<th>Program</th>
<th>CPU Registers with labels</th>
<th>Memory (“high part of RAM”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td></td>
<td>“the stack”</td>
</tr>
<tr>
<td>00 read r1</td>
<td>r0</td>
<td>42</td>
</tr>
<tr>
<td>01 setn r15 42</td>
<td>r1</td>
<td>43  03</td>
</tr>
<tr>
<td>02 calln r14 5</td>
<td></td>
<td>44  1</td>
</tr>
<tr>
<td>03 write r13</td>
<td></td>
<td>45  14</td>
</tr>
<tr>
<td>04 halt</td>
<td></td>
<td>46</td>
</tr>
<tr>
<td>05 jnezn r1 8</td>
<td></td>
<td>47</td>
</tr>
<tr>
<td>06 setn r13 1</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>07 jump r14</td>
<td></td>
<td>49</td>
</tr>
<tr>
<td>08 storer r1 r15</td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>09 addn r15 1</td>
<td></td>
<td>51</td>
</tr>
<tr>
<td>10 storer r14 r15</td>
<td></td>
<td>52</td>
</tr>
<tr>
<td>11 addn r15 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 addn r1 -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 calln r14 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 addn r15 -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 load r14 r15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 addn r15 -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 load r1 r15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 mul r13 r13 r1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19 jump r14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How low _could_ we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program (“low part of RAM”)

```
00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jumper r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jumper r14
```

CPU Registers

- **r0**: always-0 register
- **r1**: input: N
- **r13**: result, return value
- **r14**: return address (line #)
- **r15**: stack pointer

Memory (“high part of RAM”) — “the stack”

- **42**: 2
- **43**: 03
- **44**: 1
- **45**: 14
- **46**: 46
- **50**: 50
- **51**: 51
- **52**: 52

How low *could* we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program ("low part of RAM")

- The input is 2.
- CPU Registers with labels
  - r0: always-0 register
  - r1: input: N
  - r13: result, return value
  - r14: return address (line #)
  - r15: stack pointer

```
00  read  r1
01  setn  r15  42
02  calln  r14  5
03  write  r13
04  halt
05  jnezn  r1  8
06  setn  r13  1
07  jumper  r14
08  storer  r1  r15
09  addn  r15  1
10  storer  r14  r15
11  addn  r15  1
12  addn  r1  -1
13  calln  r14  5
14  addn  r15  -1
15  loadr  r14  r15
16  addn  r15  -1
17  loadr  r1  r15
18  mul  r13  r13  r1
19  jumper  r14
```

```
| 42 | 2 |
| 43 | 03 |
| 44 | 1 |
| 45 | 14 |
| 46 |   |
| 47 |   |
| 48 |   |
| 49 |   |
| 50 |   |
| 51 |   |
| 52 |   |
```

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Worksheet

Write down what happens in the registers and memory (the stack) as this program runs...

Program ("low part of RAM")

```
00  read  r1
01  setn  r15  42
02  calln  r14  5
03  write  r13
04  halt
05  jnezn  r1  8
06  setn  r13  1
07  jumper  r14
08  storerr  r1  r15
09  addn  r15  1
10  storerr  r14  r15
11  addn  r15  1
12  addn  r1  -1
13  calln  r14  5
14  addn  r15  -1
15  loadr  r14  r15
16  addn  r15  -1
17  loadr  r1  r15
18  mul  r13  r13  r1
19  jumper  r14
```

```
00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jumper r14
08 store r1 r15
09 addn r15 1
10 store r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jumper r14
```

The input is 2.

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program ("low part of RAM")

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jump r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt

05 jnezn r1 8
06 setn r13 1
07 jumpr r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jumpr r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program ("low part of RAM")

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jump r14

The input is 2.

CPU Registers with labels

<table>
<thead>
<tr>
<th>register</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>0</td>
</tr>
<tr>
<td>r1</td>
<td>0</td>
</tr>
<tr>
<td>r13</td>
<td>1</td>
</tr>
<tr>
<td>r14</td>
<td>14</td>
</tr>
<tr>
<td>r15</td>
<td>45</td>
</tr>
</tbody>
</table>

Memory ("high part of RAM")

<table>
<thead>
<tr>
<th>memory location</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>43</td>
<td>03</td>
</tr>
<tr>
<td>44</td>
<td>1</td>
</tr>
<tr>
<td>45</td>
<td>14</td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
</tbody>
</table>

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jump r14

The input is 2.

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program (“low part of RAM”)

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>CPU Registers</th>
<th>Memory (“high part of RAM”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>read r1</td>
<td>r0: 0</td>
<td>42: 2</td>
</tr>
<tr>
<td>01</td>
<td>setn r15 42</td>
<td></td>
<td>43: 03</td>
</tr>
<tr>
<td>02</td>
<td>calln r14 5</td>
<td></td>
<td>44: 1</td>
</tr>
<tr>
<td>03</td>
<td>write r13</td>
<td>r1: 0</td>
<td>45: 14</td>
</tr>
<tr>
<td>04</td>
<td>halt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>jnezn r1 8</td>
<td>r13: 1</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>setn r13 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>jumper r14</td>
<td>r14: 14</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>storer r1 r15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>addn r15 1</td>
<td>r15: 44</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>storer r14 r15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>addn r15 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>addn r1 -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>calln r14 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>addn r15 -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>loadr r14 r15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>addn r15 -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>loadr r1 r15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>mul r13 r13 r1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>jumper r14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program ("low part of RAM")

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 store r1 r15
09 addn r15 1
10 store r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jump r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Worksheet

Write down what happens in the registers and memory (the stack) as this program runs...

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jump r14

01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jump r14

01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jump r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt

05 jnezn r1 8
06 setn r13 1
07 jumper r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jumper r14

The input is 2.

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program ("low part of RAM")

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jumper r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jumper r14

CPU Registers with labels

always-0 register

r0

input: N

r1

result, return value

r13

return address (line #)

r14

stack pointer

r15

The input is 2.

000 read r1

010 setn r15 42

020 calln r14 5

030 write r13

040 halt

050 jnezn r1 8

060 setn r13 1

070 jumper r14

080 storer r1 r15

090 addn r15 1

100 storer r14 r15

110 addn r15 1

120 addn r1 -1

130 calln r14 5

140 addn r15 -1

150 loadr r14 r15

160 addn r15 -1

170 loadr r1 r15

180 mul r13 r13 r1

190 jumper r14

Memory ("high part of RAM")

"the stack"

42

43

44

45

46

47

48

49

50

51

52

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program ("low part of RAM")

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jump r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program (“low part of RAM”)

```
00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 stor r1 r15
09 addn r15 1
10 stor r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jump r14
```

CPU Registers with labels

- **r0**: always-0 register
- **r1**: input: N
- **r13**: result, return value
- **r14**: return address (line #)
- **r15**: stack pointer

Memory (“high part of RAM”)

```
42 2
43 03
44 1
45 14
46
47
48
49
50
51
52
```

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program (“low part of RAM”)

```
00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 stor r1 r15
09 addn r15 1
10 stor r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jump r14
```

**CPU Registers with labels**

- r0: always-0 register
- r1: input: N
- r13: result, return value
- r14: return address (line #)
- r15: stack pointer

**Memory (“high part of RAM”) “the stack”**

- 42
- 43
- 44
- 45
- 46
- 47
- 48
- 49
- 50
- 51
- 52

The input is 2.

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program ("low part of RAM")

```
00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jump r14
```

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program (“low part of RAM”)  

```
00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jumper r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jumper r14
```

**CPU Registers with labels**

- **r0**: always-0 register
- **r1**: input: N
- **r13**: result, return value
- **r14**: return address (line #)
- **r15**: stack pointer

**Memory (“high part of RAM”) “the stack”**

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>43</td>
<td>03</td>
</tr>
<tr>
<td>44</td>
<td>1</td>
</tr>
<tr>
<td>45</td>
<td>14</td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>42</td>
</tr>
</tbody>
</table>

How low *could* we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs...

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt

05 jnezn r1 8
06 setn r13 1
07 jmp r14
08 stor r1 r15
09 addn r15 1
10 stor r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jmp r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Write down what happens in the registers and memory (the stack) as this program runs…

Program (“low part of RAM”)

00 read r1
01 setn r15 42
02 calln r14 5
03 write r13
04 halt
05 jnezn r1 8
06 setn r13 1
07 jump r14
08 storer r1 r15
09 addn r15 1
10 storer r14 r15
11 addn r15 1
12 addn r1 -1
13 calln r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jump r14

How low could we start the stack? How deep does the stack get? What are the possible values of r14?
Towers of Hanoi

This puzzle can get Hanoi’ing!

\[ \text{hanoi}(\text{Disks}, \text{From}, \text{To}) \]
\[ \text{hanoi}(3, 1, 3) \]
This puzzle can get Hanoi'ing!

Towers of Hanoi

```
hanoi(Disks, From, To)
hanoi(3, 1, 3)
1 to 3
```
Towers of Hanoi

This puzzle can get Hanoi'ing!

\[
\text{hanoi}(\text{Disks}, \text{From}, \text{To})
\]
\[
\text{hanoi}(3, 1, 3)
\]
1 to 3
1 to 2

Peg 1
Peg 2
Peg 3
Towers of Hanoi

This puzzle can get Hanoi'ing!

hanoi(Disks, From, To)

hanoi(3, 1, 3)
1 to 3
1 to 2
3 to 2
Towers of Hanoi

This puzzle can get Hanoi'ing!

\[
\text{hanoi}(\text{Disks}, \text{From}, \text{To})
\]

\[
\text{hanoi}(3, 1, 3)
\]

1 to 3
1 to 2
3 to 2
1 to 3
This puzzle can get Hanoi’ing!

```
hanoi(Disks, From, To)
hanoi(3, 1, 3)
  1 to 3
  1 to 2
  3 to 2
  1 to 3
  2 to 1
```
Towers of Hanoi

This puzzle can get Hanoi'ing!

```prolog
hanoi(Disks, From, To)
hanoi(3, 1, 3)
1 to 3
1 to 2
3 to 2
1 to 3
2 to 1
2 to 3
```

Peg 1

Peg 2

Peg 3
Towers of Hanoi

This puzzle can get Hanoi’ing!

\[
hanoi(\text{Disks, From, To})
\]

\[
hanoi(3, 1, 3)
\]

1 to 3
1 to 2
3 to 2
1 to 3
2 to 1
2 to 3
1 to 3

\[7 = 2^3 - 1\] moves
Towers of Hanoi

```python
hanoi (Disks, From, To)
    if Disks == 1:
        print(From, "","", To)
        return
    else:
        # COMPUTE "Other" peg
        hanoi(Disks-1, From, Other)
        hanoi(1, From, To)
        hanoi(Disks-1, Other, To)
        return
```

```python
>>> hanoi(1, 1, 3)
1 , 3
```

```python
>>> hanoi(2, 1, 2)
```
Towers of Hanoi

```python
hanoi (Disks, From, To)
    if Disks == 1:
        print(From, "", To)
        return
    else:
        # COMPUTE "Other" peg
        hanoi(Disks-1, From, Other)
        hanoi(1, From, To)
        hanoi(Disks-1, Other, To)
        return
```

```python
>>> hanoi(1, 1, 3)
1 , 3
```

```python
>>> hanoi(2, 1, 2)
1 , 3
```

```python
>>> hanoi(1, 1, 3)
1 , 3
```

```python
>>> hanoi(1, 1, 2)
1 , 2
```

```python
>>> hanoi(1, 1, 3)
1 , 3
```

```python
>>> hanoi(1, 1, 2)
1 , 2
```

```python
>>> hanoi(1, 3, 2)
3 , 2
```

```python
>>> hanoi(1, 3, 2)
3 , 2
```

```python
>>> hanoi(1, 3, 2)
3 , 2
```

Peg 1

Peg 2

Peg 3
```python
hanoi (Disks, From, To)
    if Disks == 1:
        print(From, ",", To)
        return
    else:
        # COMPUTE "Other" peg
        hanoi(Disks-1, From, Other)
        hanoi(1, From, To)
        hanoi(Disks-1, Other, To)
        return
```
See you in lab!