Sports: HMC CS Professor to coach 2012 U.S. Olympic chocolate-eating team
Weather: 63.79% chance of weather today

CS 5 Today

News in Brief

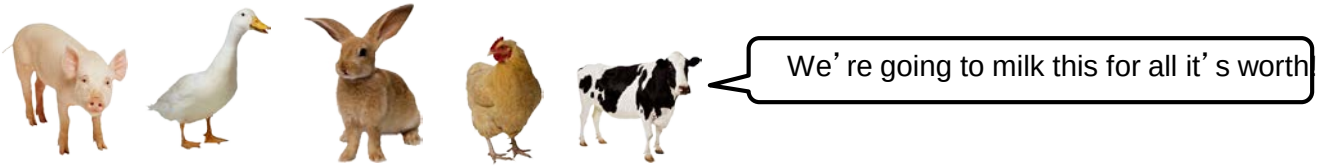
CS 5 alien abducted by aliens
(p. 42)

Page 42 will no longer be published
(p. 42)

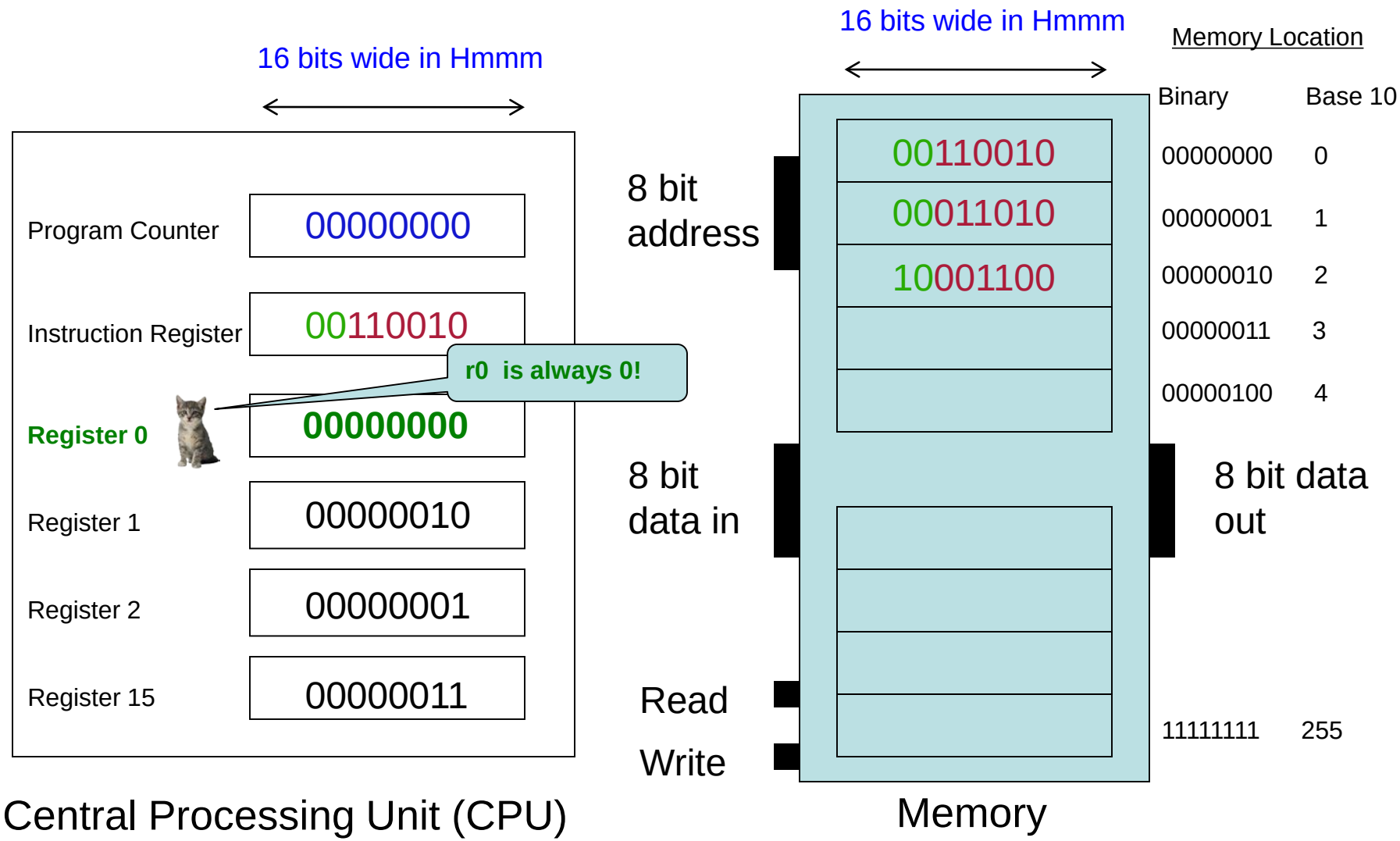
Farm animals displace penguins, invade CS 5 notes
(p. 42)

Psychics predict that there was no CS 5 lecture yesterday. Definitive proof of paranormal phenomena!

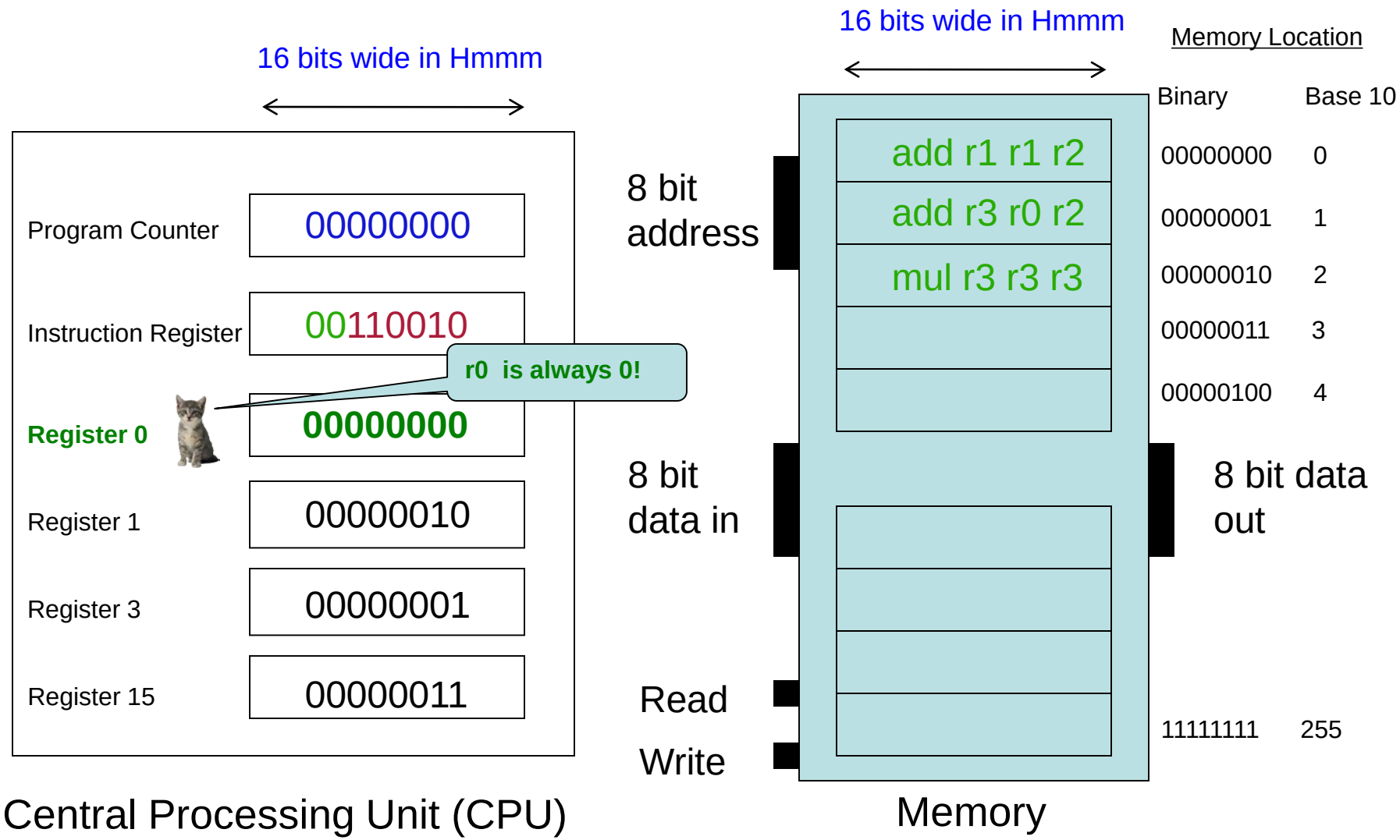
(Claremont AP): A group of psychics has made an extraordinary set of predictions that, one-by-one, are being corroborated by scientists. "It is indeed true that we didn't have CS 115 yesterday," said one CS 115 professor. The psychics have also predicted that an exam will occur sometime within the next 3-10 days.



Machine Language Versus…



…Assembly Language!



Hmmm Assembly Language

```
add r2 r2 r2
```

`reg2 = reg2 + reg2`

crazy, perhaps, but used ALL the time

```
sub r2 r1 r4
```

`reg2 = reg1 - reg4`

which is why it is written this way in python!

```
mul r7 r6 r2
```

`reg7 = reg6 * reg2`

```
div r1 r1 r1
```

`reg1 = reg1 / reg1`

INTEGER division—no remainders

```
setn r1 42
```

`reg1 = 42`

you can replace 42 with anything from -128 to 127

```
addn r1 -1
```

`reg1 = reg1 - 1`

a shortcut

```
read r10
write r1
```

read from keyboard and write to screen

Each of these instructions (and many more) gets implemented for a particular processor and particular machine…

jumps

Unconditional jump

jumpn 42

Replaces the PC (program counter) with **42**. "Jump to program line number **42**."

Conditional jumps

jeqzn r1 n — IF **r1 == 0** THEN jump to line number **n**

jgtzn r1 n — IF **r1 > 0** THEN jump to the location in **n**

jltzn r1 n — IF **r1 < 0** THEN jump to the location in **n**

jnezn r1 n — IF **r1 != 0** THEN jump to the location in **n**

Register jump

jumpr r1

Jump to the line **n** stored in **reg1**!

This IS making me jumpy!

Worksheet

Feeling Jumpy?

1 Write an assembly-language program that reads ***one*** integer as keyboard input. Then, the program should compute the ***factorial*** of that input in register r13 and write it out. You may assume without checking that the input will be a positive integer.

Registers - CPU

r0	0
r1	
r2	
r3	
r4	
r5	

Memory - RAM

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

2 Write an assembly-language program that reads ***two*** integers r1 and r2 as keyboard input. Then, the program should compute $r1^{r2}$ in register r13 and write it out. You may assume that r2 >= 0.

Registers - CPU

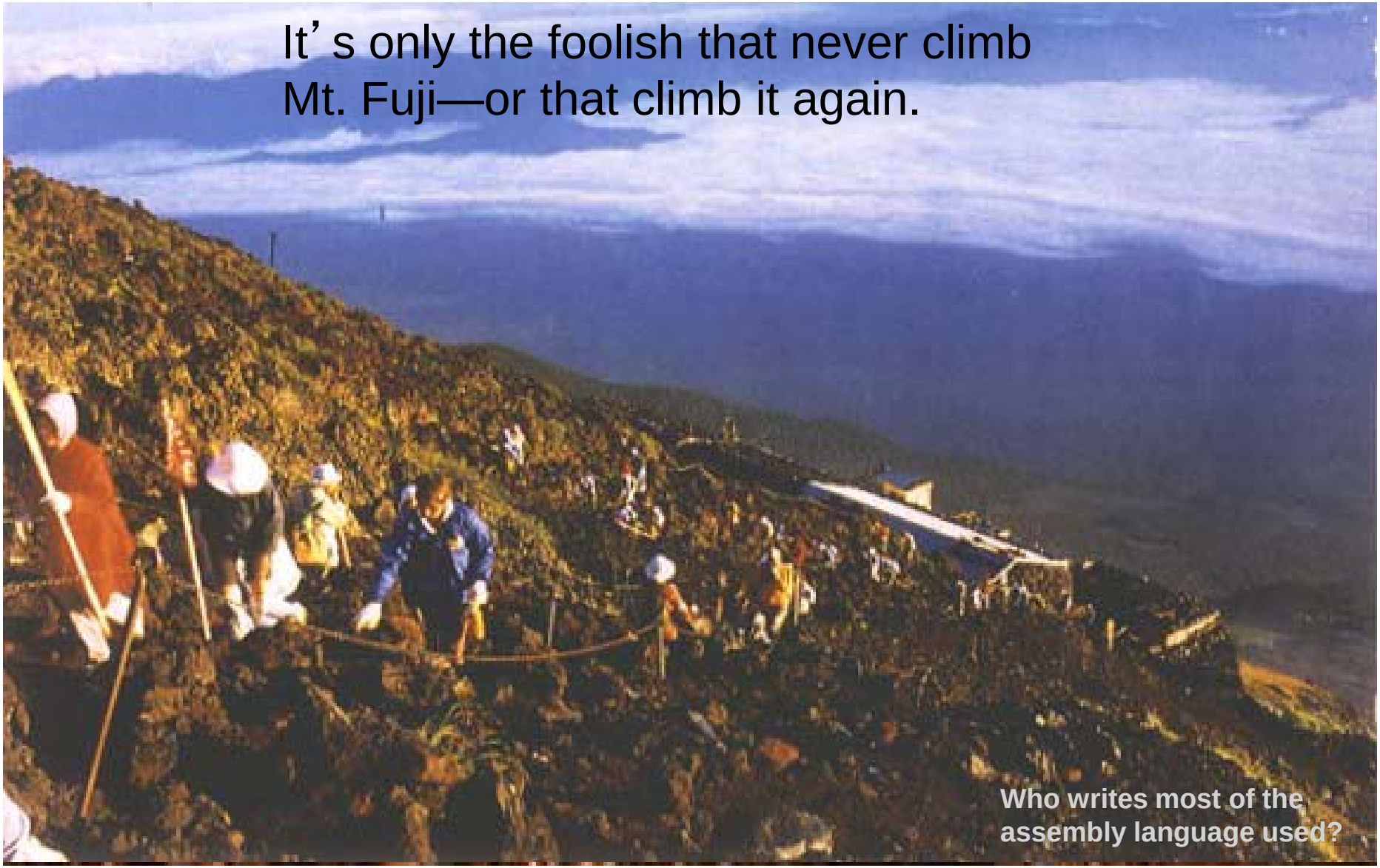
r0	0
r1	
r2	
r3	
r4	
r5	

Memory - RAM

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

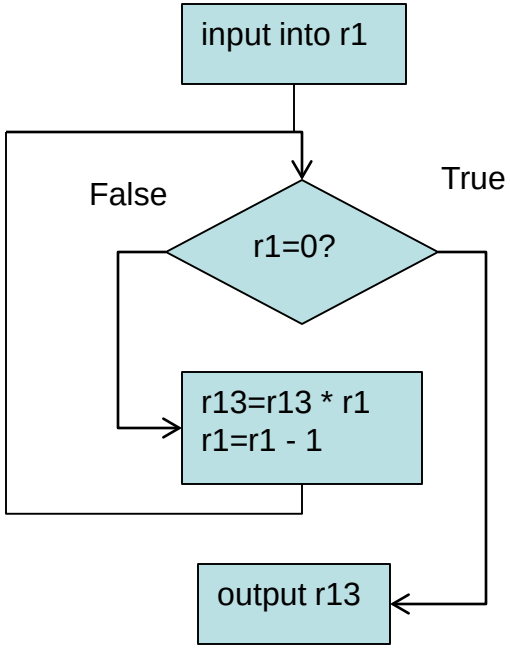
Instruction	Description	Aliases
System instructions		
halt	Stop!	
read rX	Place user input in register rX	
write rX	Print contents of register rX	
nop	Do nothing	
Setting register data		
setn rX N	Set register rX equal to the integer N (-128 to +127)	
addn rX N	Add integer N (-128 to 127) to register rX	
copy rX rY	Set rX = rY	**mov**
Arithmetic		
add rX rY rZ	Set rX = rY + rZ	
sub rX rY rZ	Set rX = rY - rZ	
neg rX rY	Set rX = -rY	
mul rX rY rZ	Set rX = rY * rZ	
div rX rY rZ	Set rX = rY / rZ (integer division; no remainder)	
mod rX rY rZ	Set rX = rY % rZ (returns the remainder of integer division)	
Jumps!		
jumpn N	Set program counter to address N	
jumpr rX	Set program counter to address in rX	**jump**
jeqzn rX N	If rX == 0, then jump to line N	**jeqz**
jnezn rX N	If rX != 0, then jump to line N	**jnez**
jgtzn rX N	If rX > 0, then jump to line N	**jgtz**
jltzn rX N	If rX < 0, then jump to line N	**jltz**
calln rX N	Copy the next address into rX and then jump to mem. addr. N	**call**
Interacting with memory (RAM)		
loadn rX N	Load register rX with the contents of memory address N	
storen rX N	Store contents of register rX into memory address N	
loadr rX rY	Load register rX with data from the address location held in reg. rY	**loadi, load**
storer rX rY	Store contents of register rX into memory address held in reg. rY	**storei, store**

Why Assembly Language?



Factorial

Put input in r1
Set r13 to 1
If r1 is 0 go to output; otherwise
Set r13 = r13 * r1
Decrease r1 = r1 – 1
Go to If
Output r13




```
0    read    r1              # Get n (assume n >= 0)
1    setn    r13  1          # initialize r13
2    jeqzn   r1 6            # done if r1 is 0
3    mul     r13 r13 r1      # change r13 = r13 * r1
4    addn    r1 -1           # change r1 = r1 - 1
5    jumpn   2               # repeat
6    write   r13
7    halt
```

The Compiler

A program that translates from human-usable language into assembly language and machine langauge

```
x = 6
y = 7
z = x*y
print z
```

the code

Compiler

```
setn r1 6
setn r2 7
mul r3 r1 r2
write r3
```

assembly or byte-code

```
0000 0001 0000 0001
1000 0010 0001 0001
0110 0010 0010 0001
0000 0010 0000 0010
0000 0000 0000 0000
```

executable machine code

Interpreter

```
0000 0001 …
0010 0001 …
0110 0010 …
1000 0001 …
```

machine code

interpreting byte code

Examples

```
x = 6
y = 7
z = x*y
print z
```

The code

Core 2 Duo

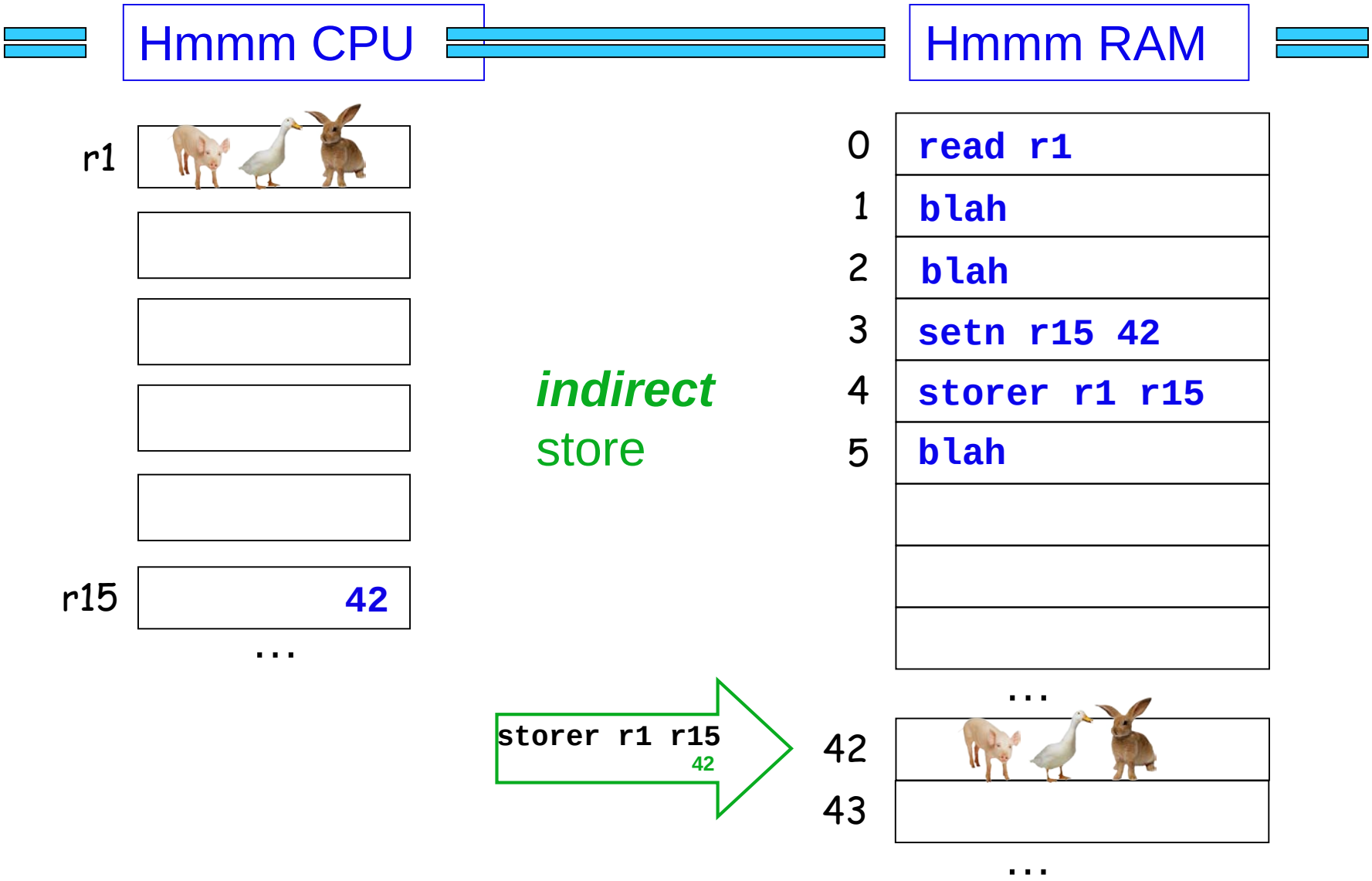
```
.globl main
    .type   main   function
main
.LFB2
    pushq    rbp
.LCFI
    movq     rsp   rbp
.LCFI1
    subq     16   rsp
.LCFI2
    movl     6   12  rbp
    movl     7   8  rbp
    movl     12  rbp    eax
    imull    8  rbp    eax
    movl     eax   4  rbp
    movl     4  rbp    esi
    movl     .LC0,  edi
    movl     0,  eax
    call    printf
    leave
    ret
```

Power PC

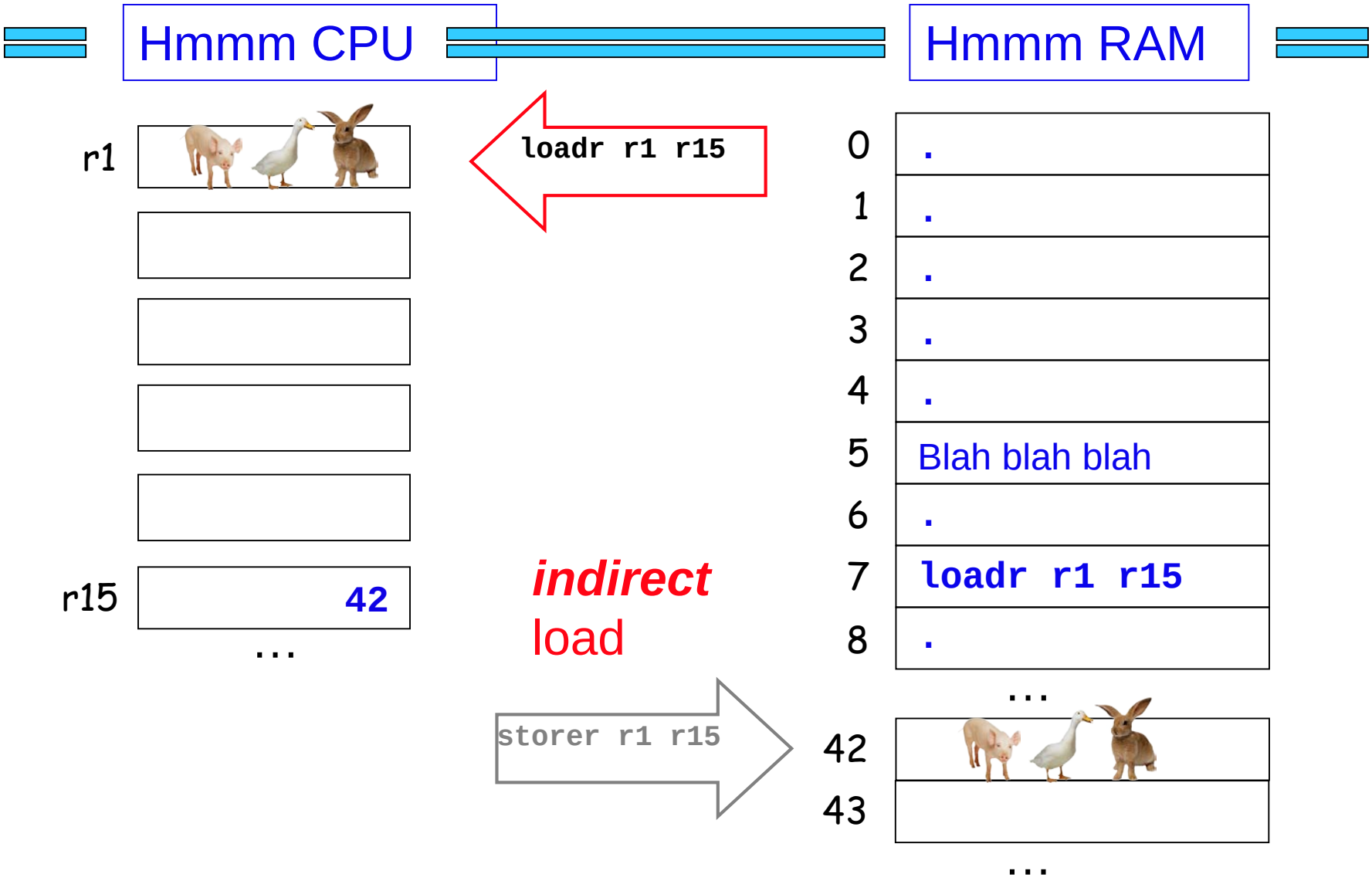
```
LC0:
    .ascii "z is %d\12\0"
    .text
    .align 2
    .globl _main
_main:
    mflr r0
    stmw r30,-8(r1)
    stw r0,8(r1)
    stwu r1,-96(r1)
    mr r30,r1
    bcl 20,31,"L00000000001$pb"
"L00000000001$pb":
    mflr r31
    li r0,6
    stw r0,64(r30)
    li r0,7
    stw r0,60(r30)
    lwz r2,64(r30)
    lwz r0,60(r30)
    mullw r0,r2,r0
    stw r0,56(r30)
    addis r2,r31,ha16(LC0-"L00000000001$pb")
    la r3,lo16(LC0-"L00000000001$pb")(r2)
    lwz r4,56(r30)
    bl L_printf$LDBLStub$stub
    lwz r1,0(r1)
    lwz r0,8(r1)
    mtlr r0
    lmw r30,-8(r1)
    blr
```

Each processor has its own *endearing* idiosyncrasies...

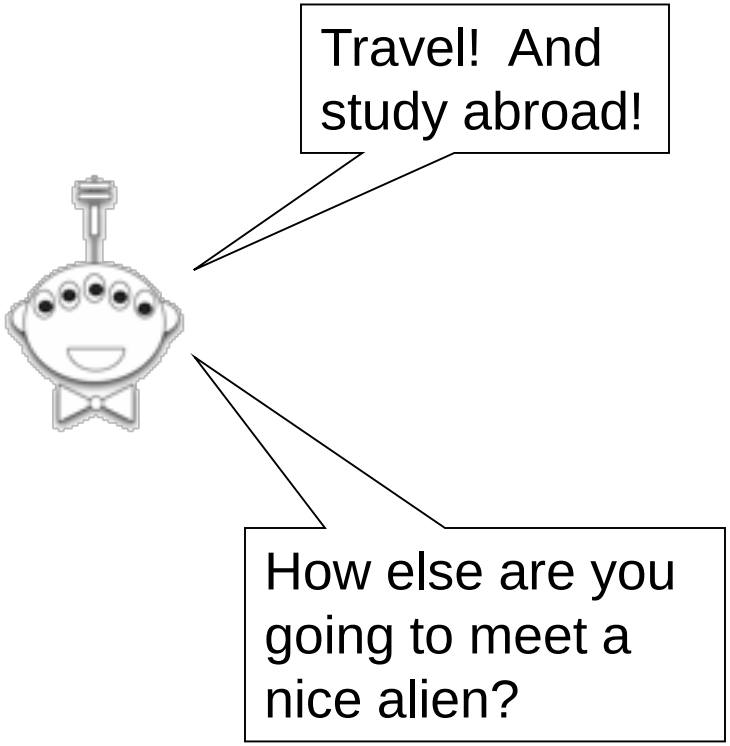
`storer` Goes TO Memory



`loadr` Comes FROM Memory



The Alien's Life Advice



calln = setn + jumpn!

A function call in python:

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

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

Hmmm's **call** operation:

puts NEXT line # into r14, then jumps to line 4

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

```
4    do stuff and
5    answer in r13
6    jumpr r14
```

Factorial: *Function Call!*

Hmmm CPU

r0 | 0

Input value: x

r1 |

Final result - return value - *in progress*

r13 |

location / line to return TO

r14 | 2

Hmmm RAM

0	`read r1`	} input
1	`calln r14 4`	function call
2	`write r13`	} output
3	`halt`	
4	`setn r13 1`	the function!
5	`jeqzn r1 9`	loop
6	`mul r13 r13 r1`	
7	`addn r1 -1`	
8	`jumpn 5`	
9	`jumpr r14`	return

Which Factorial Is It?

0	`read r1`	} input
1	`setn r13 1`	← initialize result to 1
2	`jeqzn r1 6`	loop
3	`mul r13 r13 r1`	
4	`addn r1 -1`	
5	`jumpn 2`	
6	`write r13`	} output
7	`halt`	

```
def fac1():
    r1 = input()
    r13 = 1
    while r1 != 0:
        r13 = r13 * r1
        r1 += -1
    print r13
    return

def fac2(r1):
    if r1 == 0:
        return 1
    else:
        return r1 * fac2(r1-1)
```

Which Factorial Is It?

0	`read r1`	input
1	`setn r13 1`	initialize result to 1
2	`jeqzn r1 6`	loop
3	`mul r13 r13 r1`	loop
4	`addn r1 -1`	loop
5	`jumpn 2`	loop
6	`write r13`	output
7	`halt`	output

```
def fac1():
    r1 = input()
    r13 = 1
    while r1 != 0:
        r13 = r13 * r1
        r1 += -1
    print r13
    return

def fac3(r1,r13):
    if r1 == 0:
        return r13
    else:
        return fac3(r1-1, r13*r1)
```

Trace this: `fac3(input,1)`

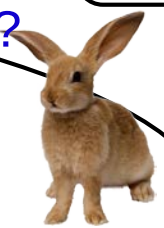
Function Calls…

```
def main():
    r1 = input()  ←——  r1=3
    r13 = emma(r1) ← emma(3)
    r13 = r13 + r1
    print r13
    return

def emma(r1):  ←——  r1=3
    r1 = r1 + 1 ←——  r1=4
    r13 = sarah(r1) ← sarah(4)
    r13 = r13 + r1 ←  r13=??
    return r13

def sarah(r1): ←——  r1=4
    r1 = r1 + 42 ←——  r1=46
    r13 = r1 + 1 ←——  r13=47
    return r13   ←——  return(47)
```

Chew on this…



You should be worried!
r13=47

Function Calls…

```
def main():
    r1 = input()  ←——  r1=3
    r13 = emma(r1) ← emma(3)  r13=51
    r13 = r13 + r1 ← r13=??
    print r13
    return

def emma(r1):  ←——  r1=3
    r1 = r1 + 1 ←——  r1=4
    r13 = sarah(r1)← sarah(4) r13=47
    r13 = r13 + r1 ← r13=51
    return r13  ←——return(51)

def sarah(r1): ←——  r1=4
    r1 = r1 + 42 ←——  r1=46
    r13 = r1 + 1 ←——  r13=47
    return r13  ←——return(47)
```

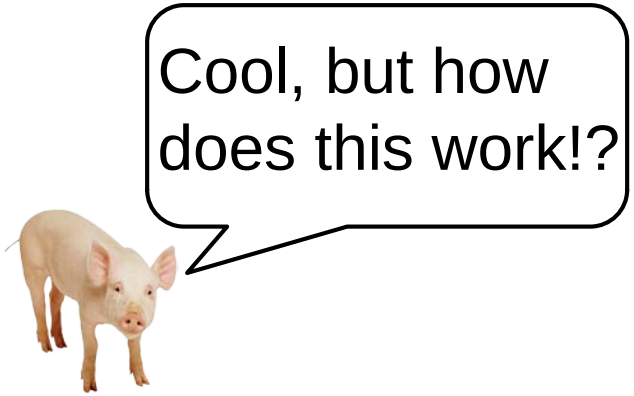
Function Calls…

```
def main():
    r1 = input()  ←——  r1=3
    r13 = emma(r1) ← emma(3)  r13=51
    r13 = r13 + r1 ← r13=54
    print r13
    return
```

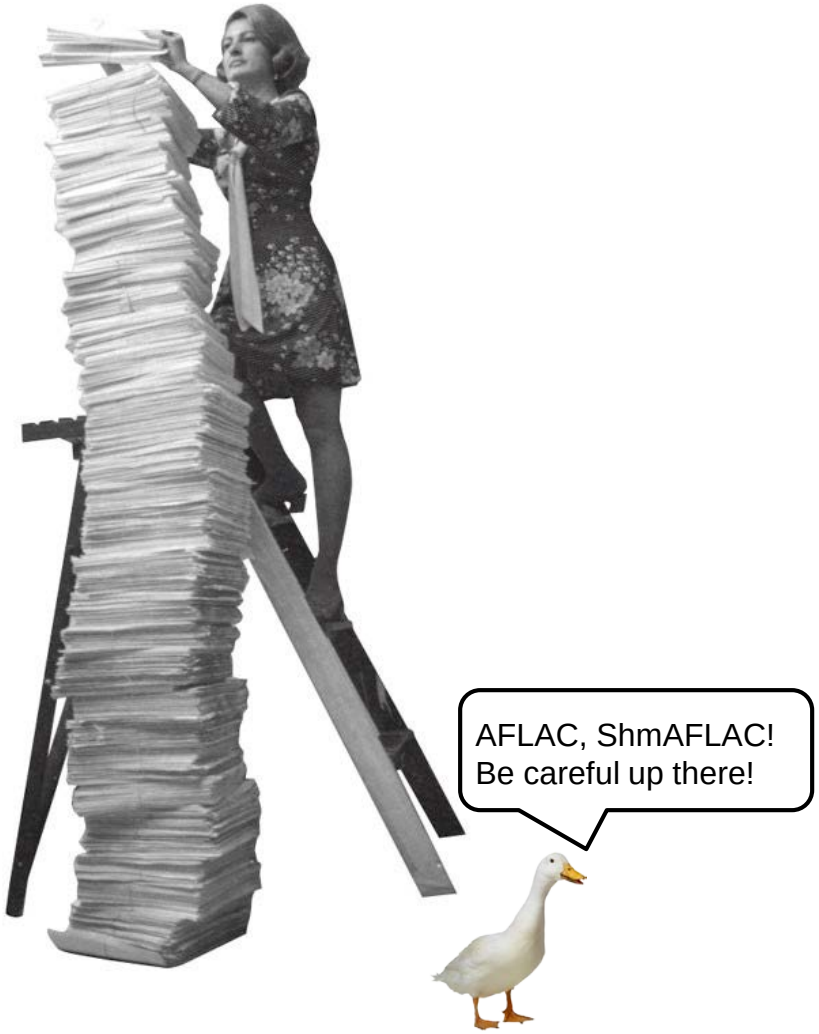
54

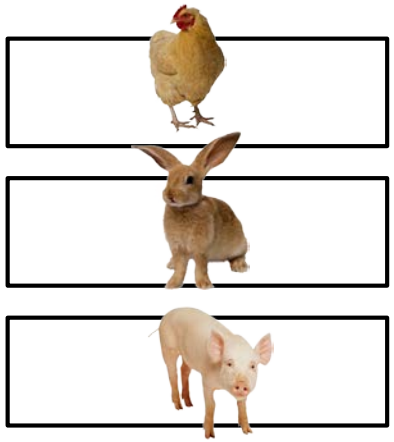
```
def emma(r1):  ←——  r1=3
    r1 = r1 + 1 ←——  r1=4
    r13 = sarah(r1)← sarah(4) r13=47
    r13 = r13 + r1 ← r13=51
    return r13  ←——return(51)
```

```
def sarah(r1): ←——  r1=4
    r1 = r1 + 42 ←——  r1=46
    r13 = r1 + 1 ←——  r13=47
    return r13  ←——return(47)
```



The Stack!





Insert ("push")

Remove ("pop")

Watch carefully…

What if I don’t give a hoot?!

Function Calls…

Hmmm code up here!

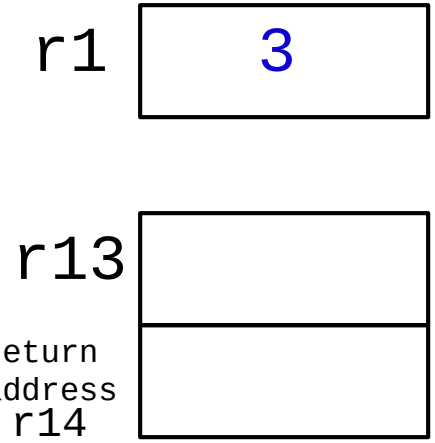
The stack in RAM!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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

r1=3



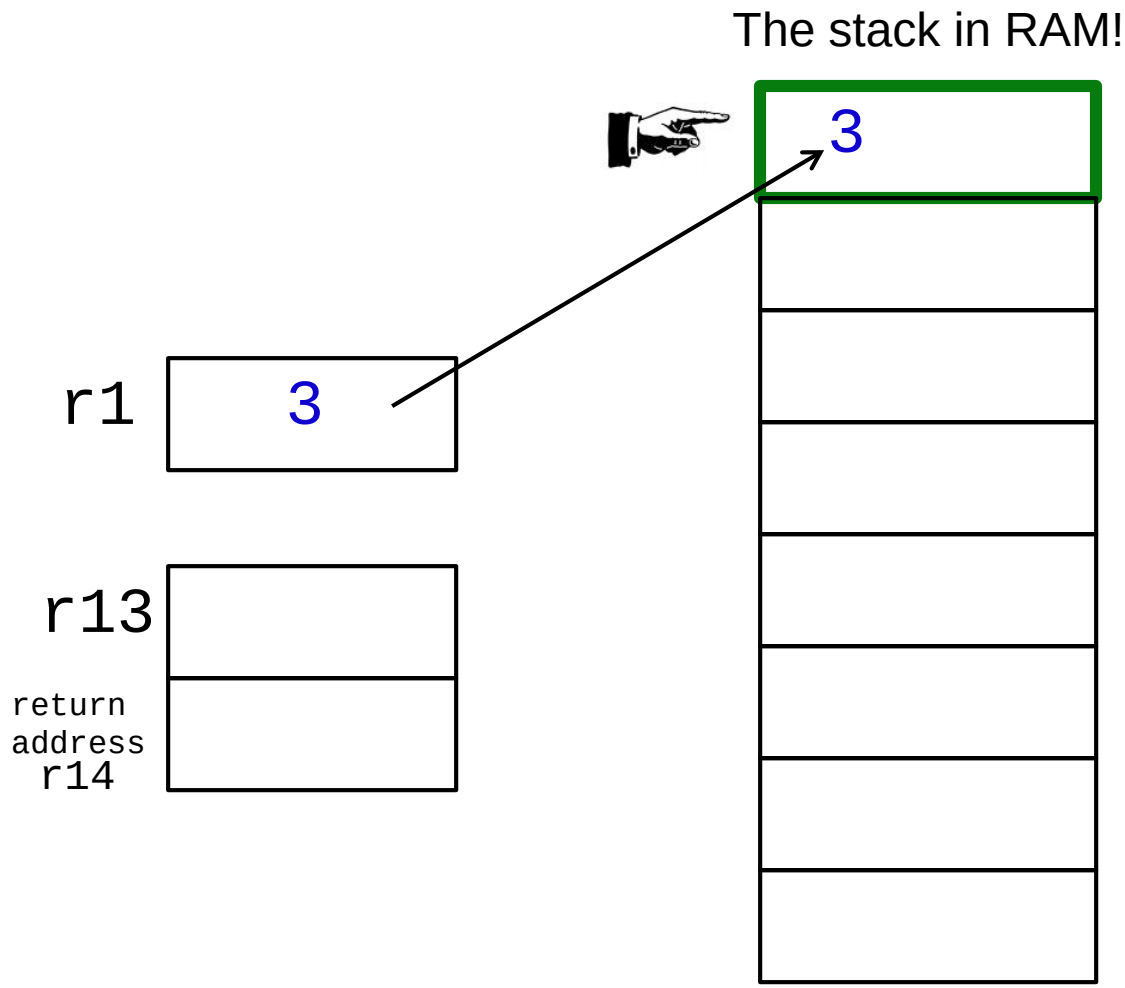
Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



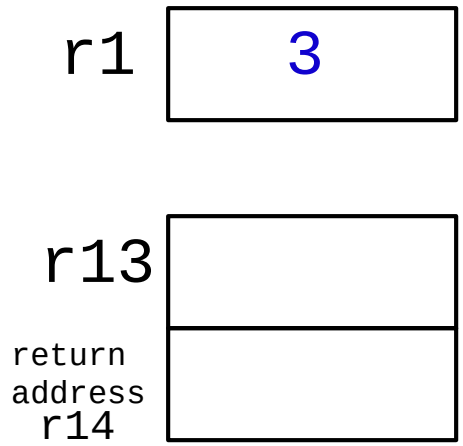
Function Calls…

Hmmm code up here!

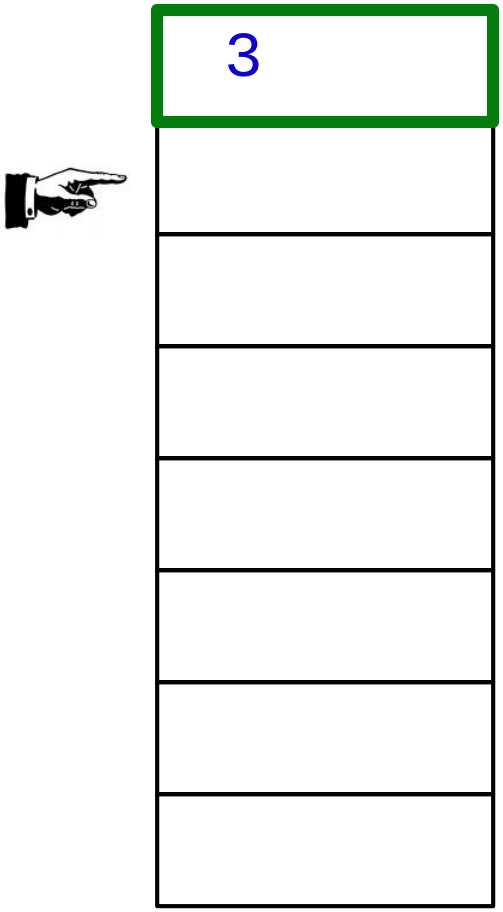
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



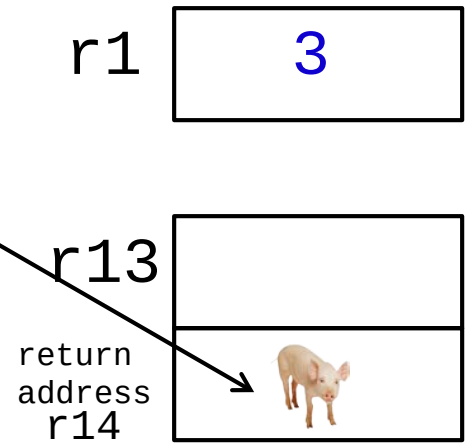
Function Calls…

Hmmm code up here!

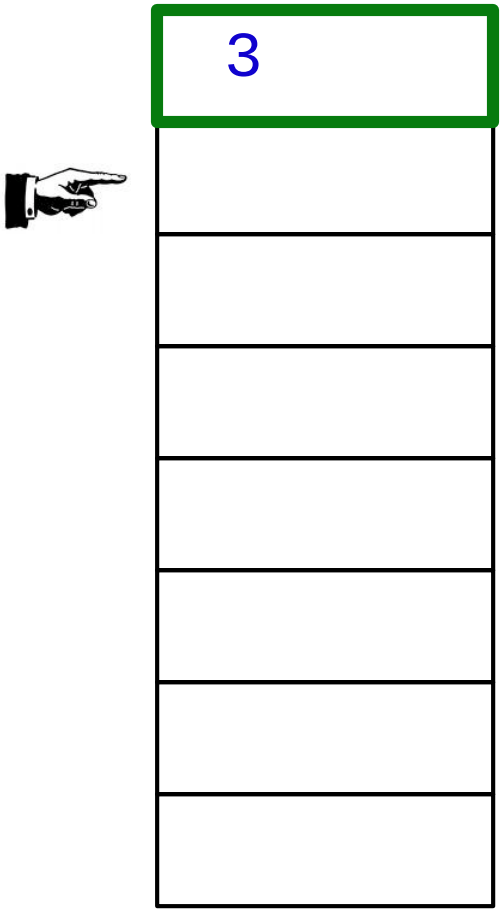
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



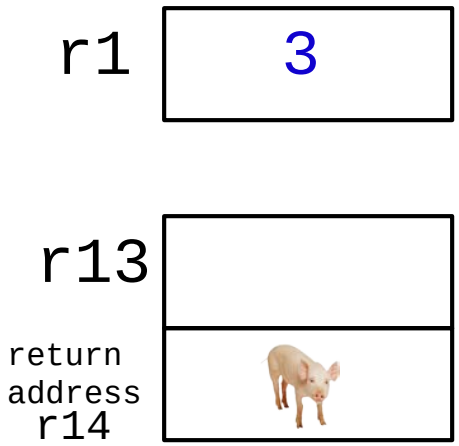
Function Calls…

Hmmm code up here!

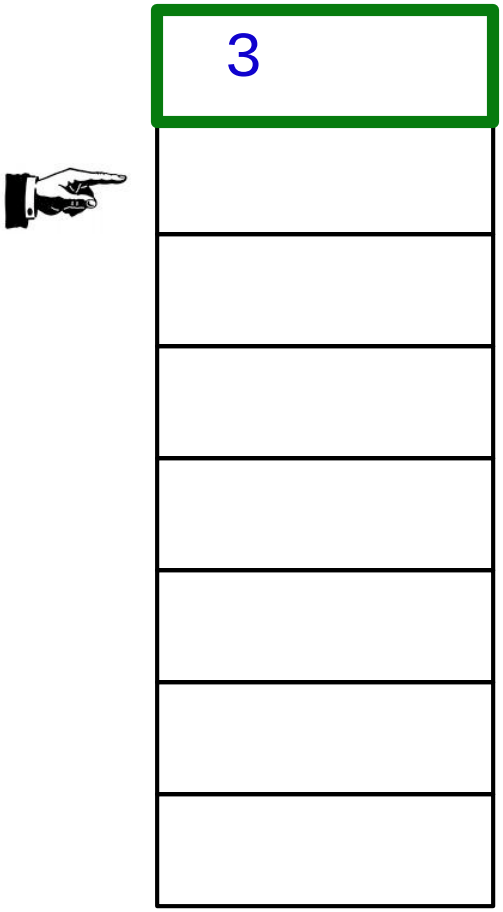
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



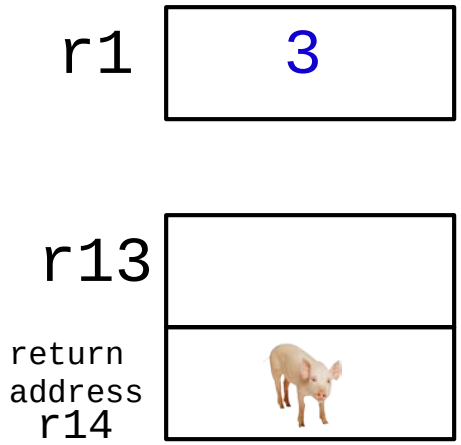
Function Calls…

Hmmm code up here!

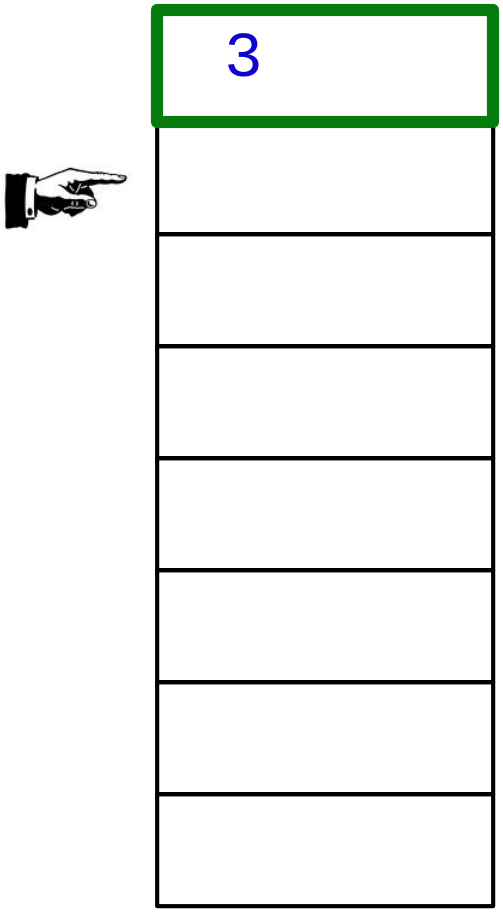
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



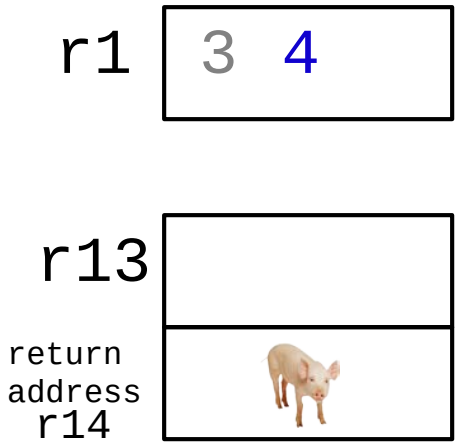
Function Calls…

Hmmm code up here!

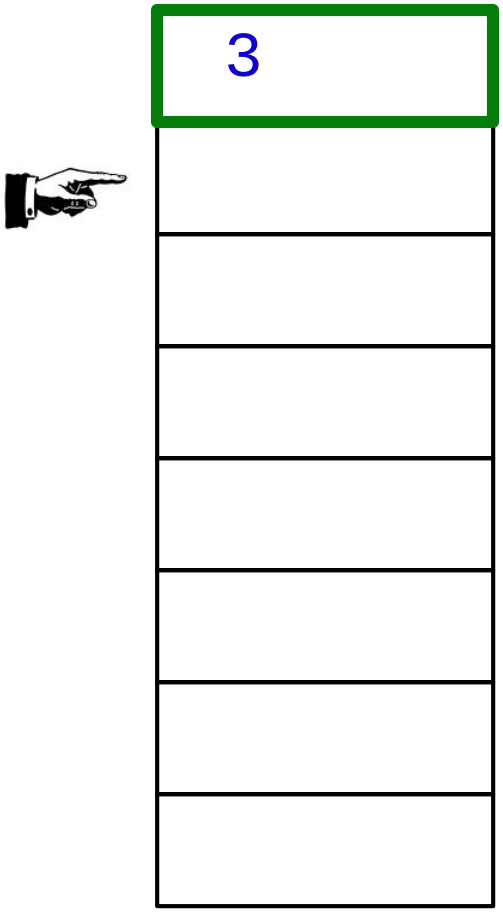
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



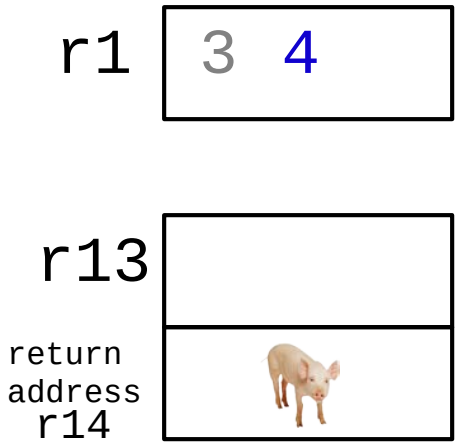
Function Calls…

Hmmm code up here!

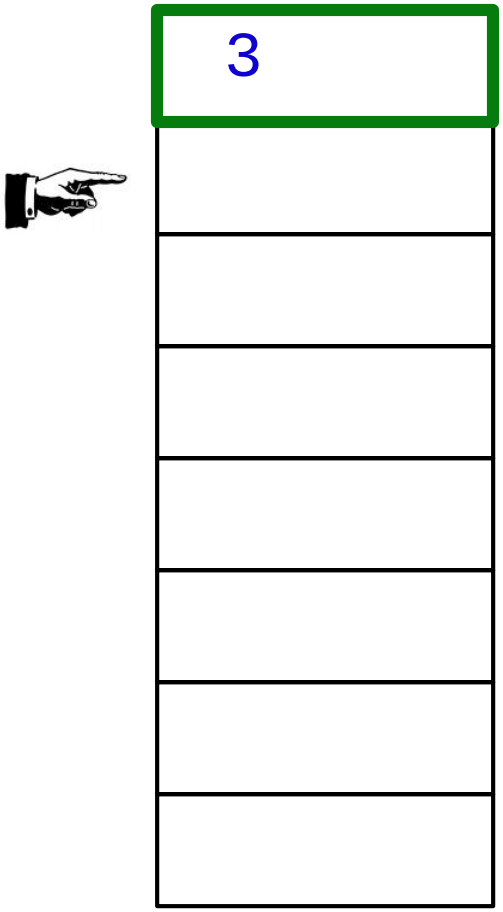
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



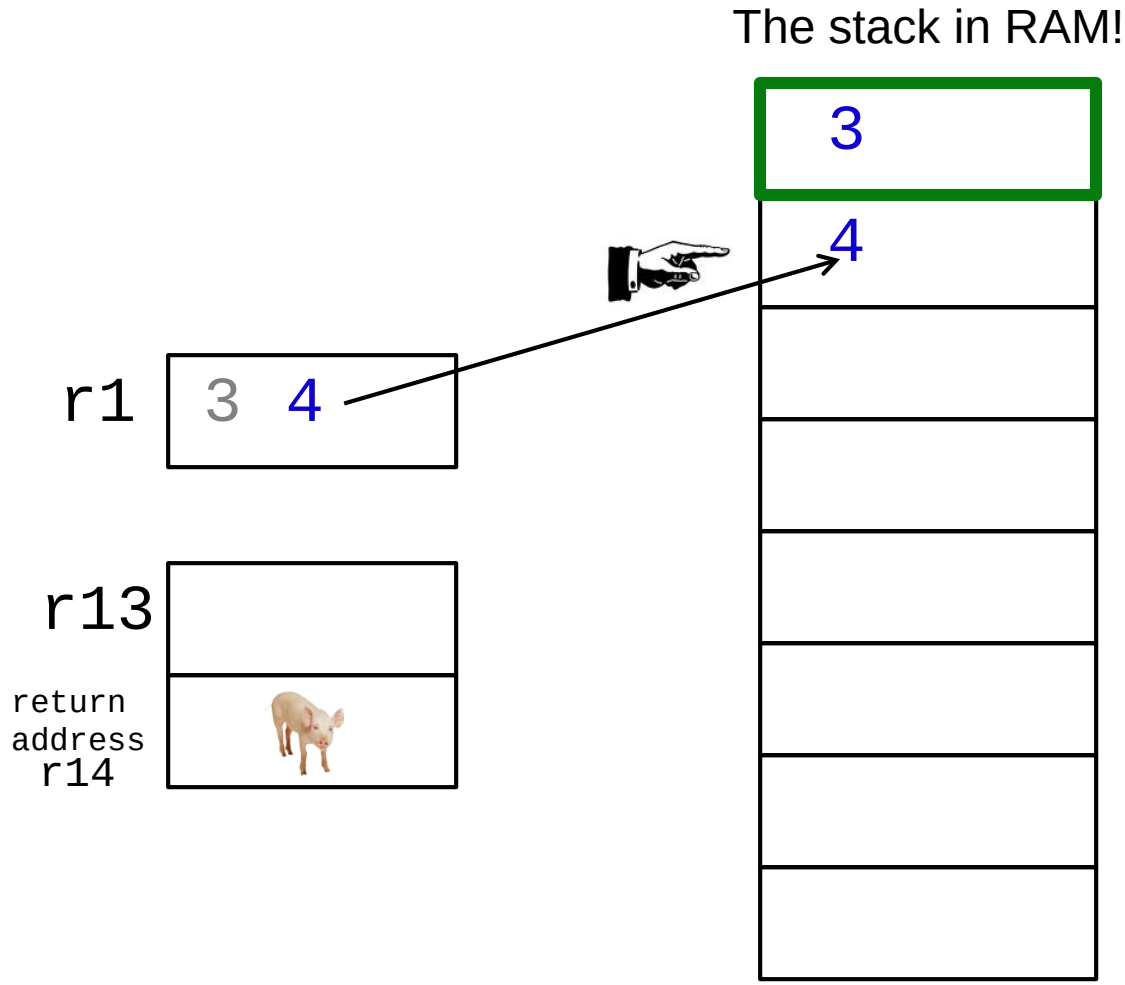
Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



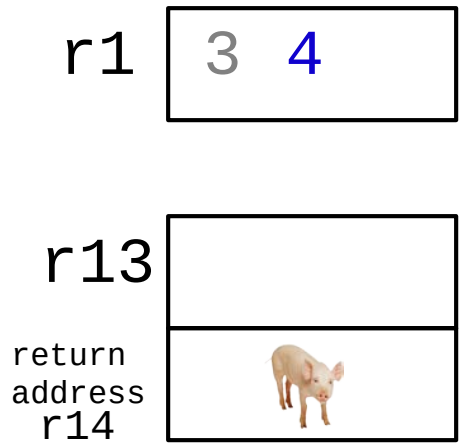
Function Calls…

Hmmm code up here!

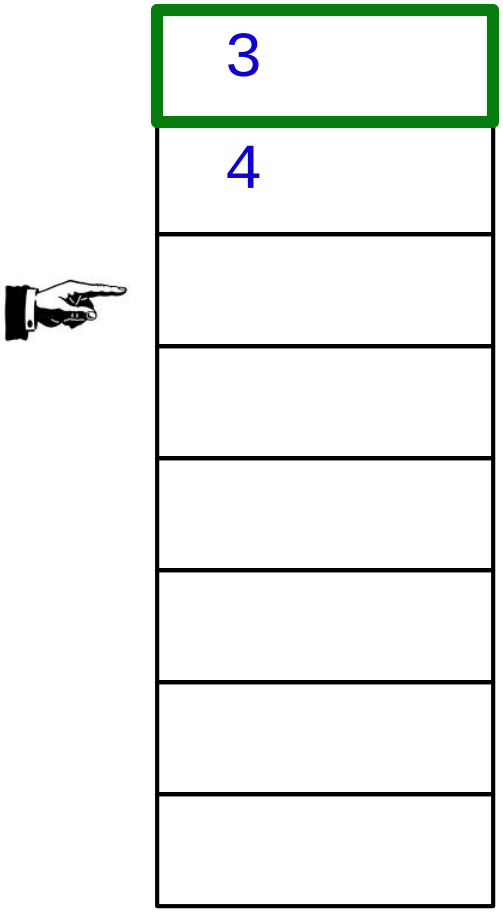
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!

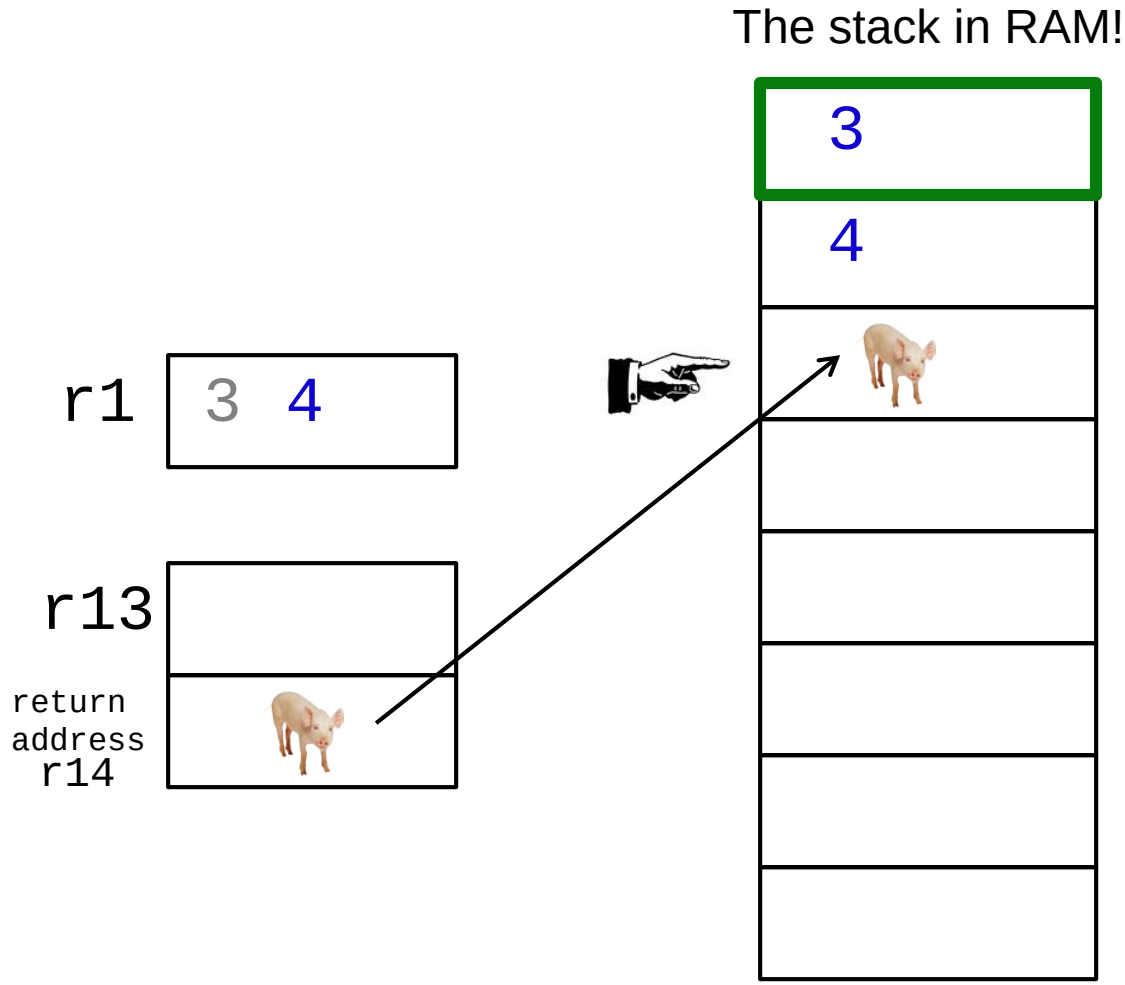


Function Calls…

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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

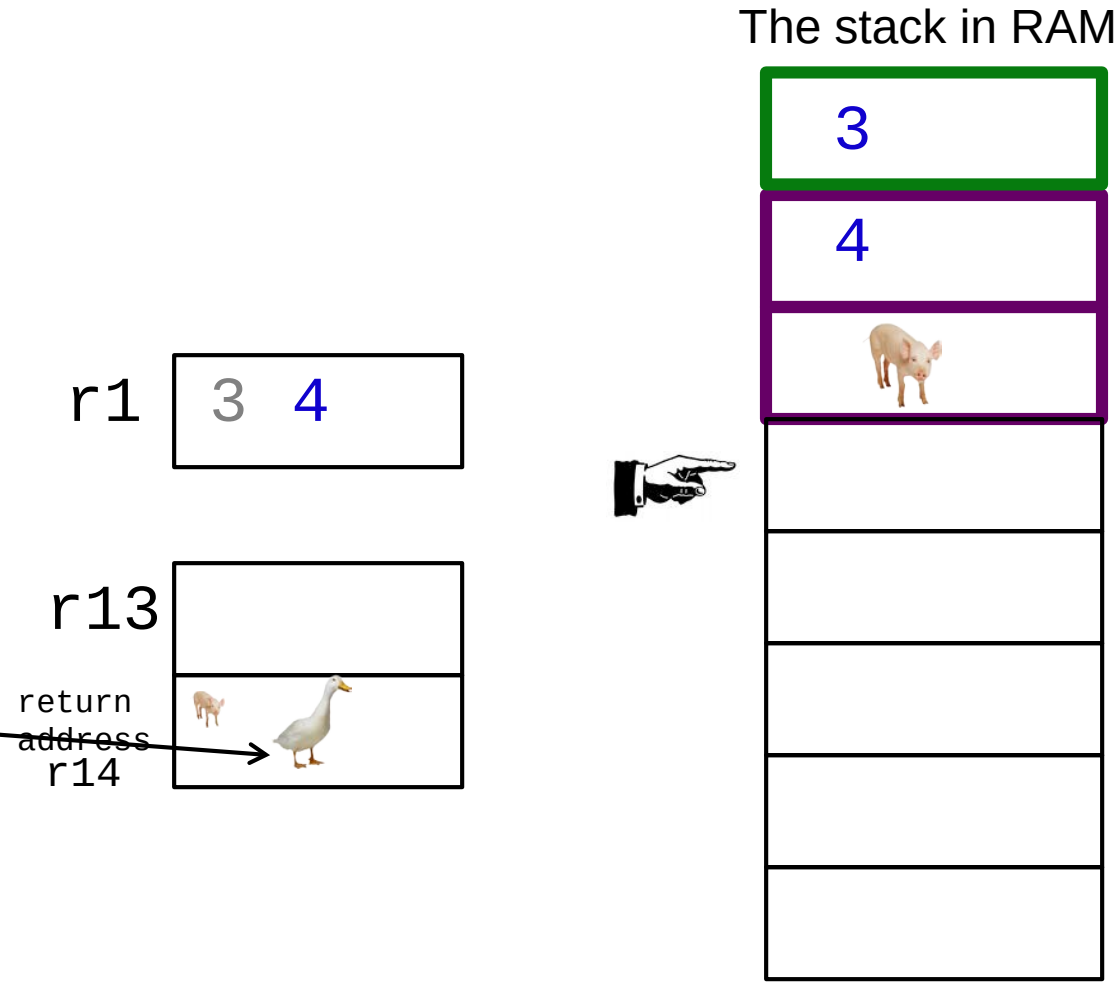


Function Calls…

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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

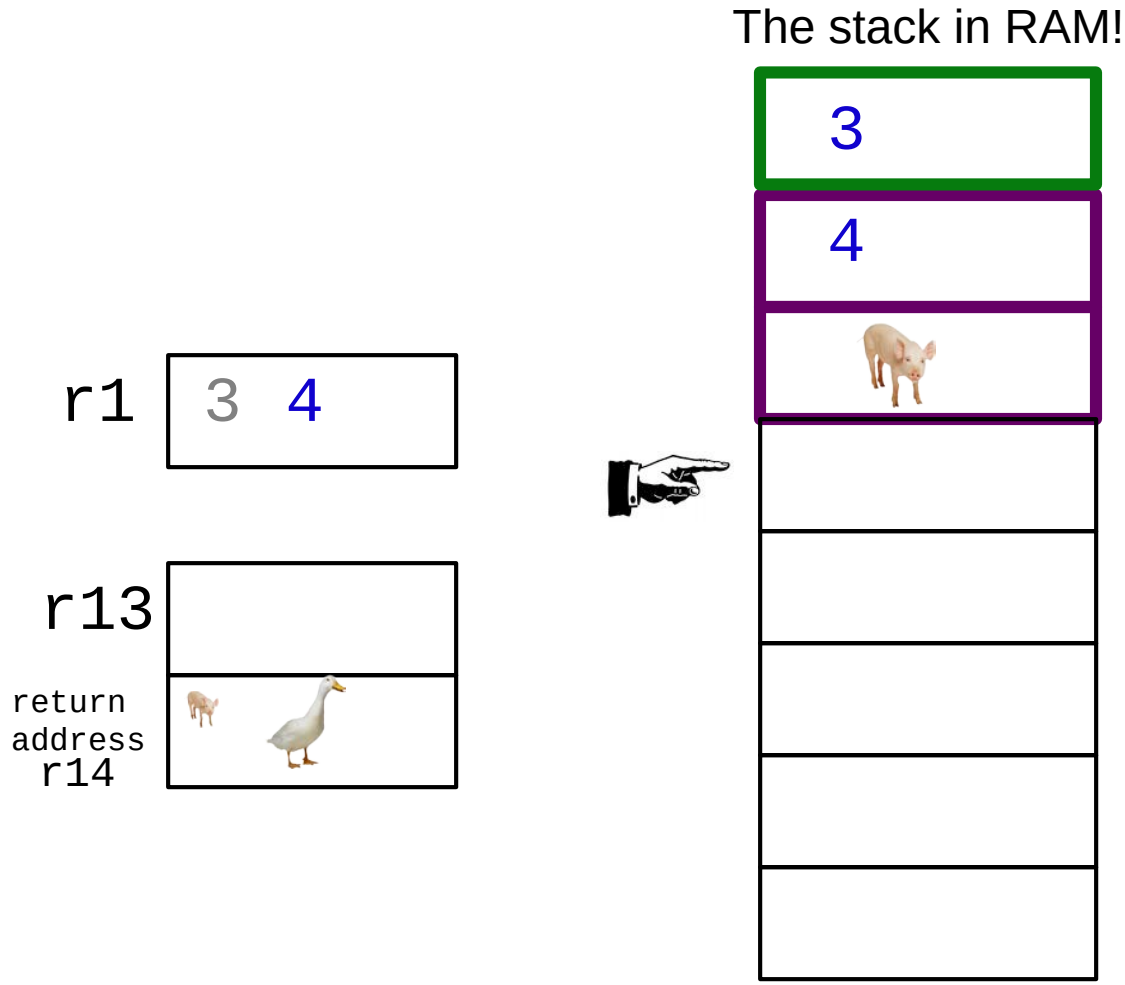


Function Calls…

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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

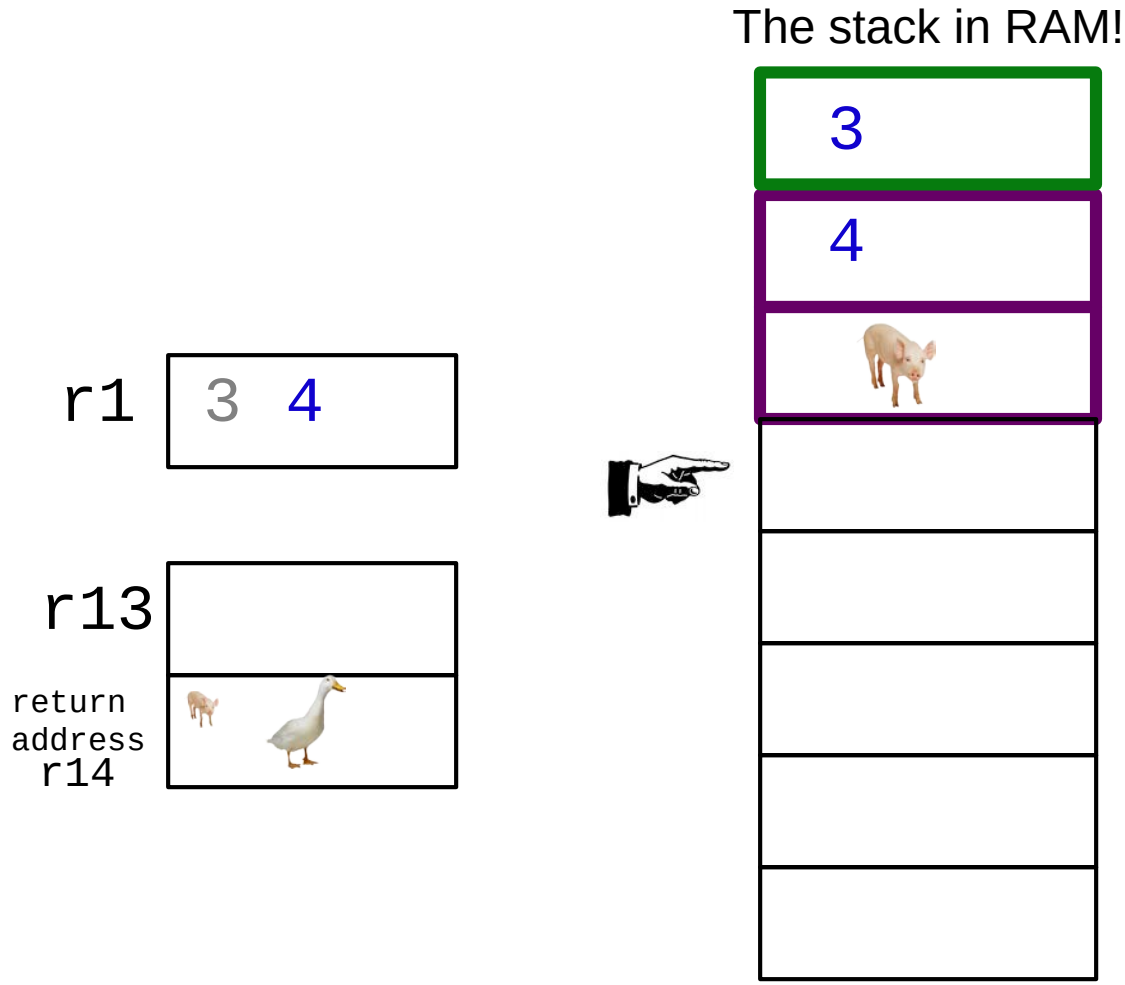


Function Calls…

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



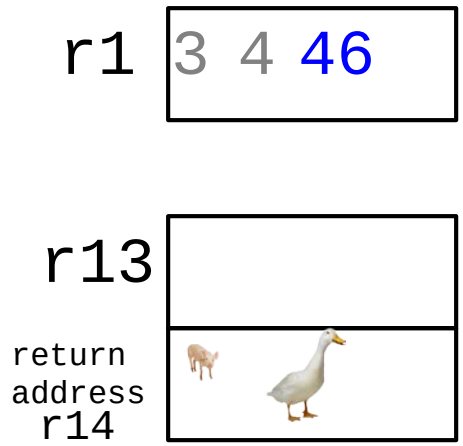
Function Calls…

Hmmm code up here!

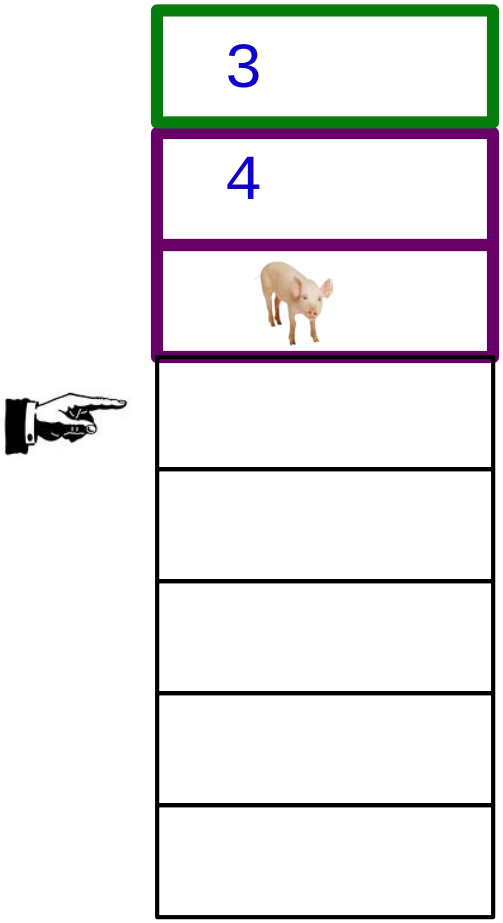
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



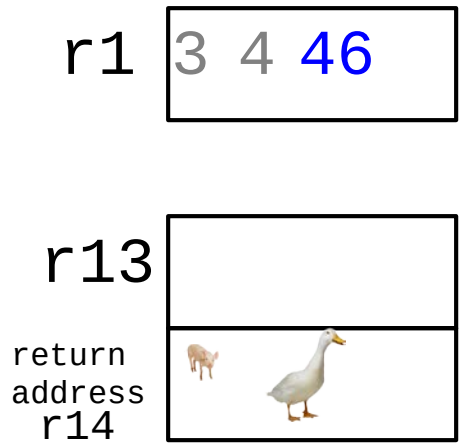
Function Calls…

Hmmm code up here!

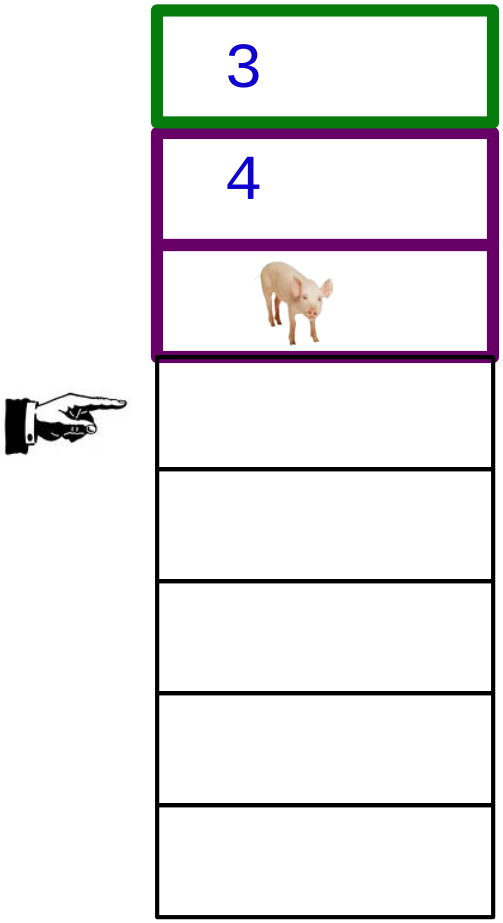
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



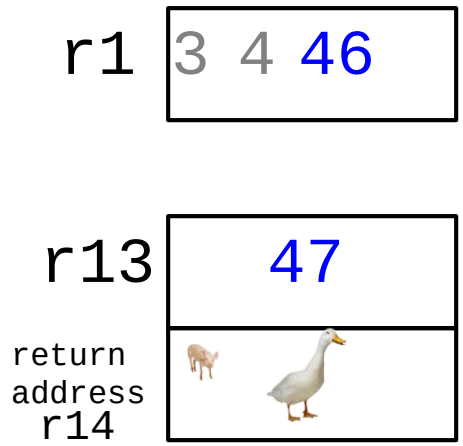
Function Calls…

Hmmm code up here!

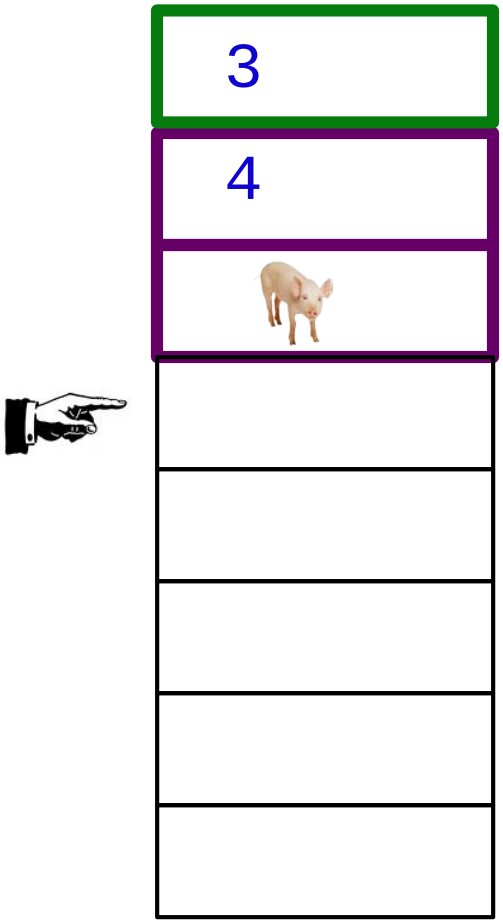
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



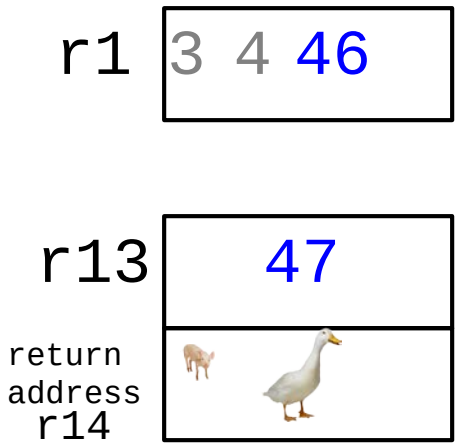
Function Calls…

Hmmm code up here!

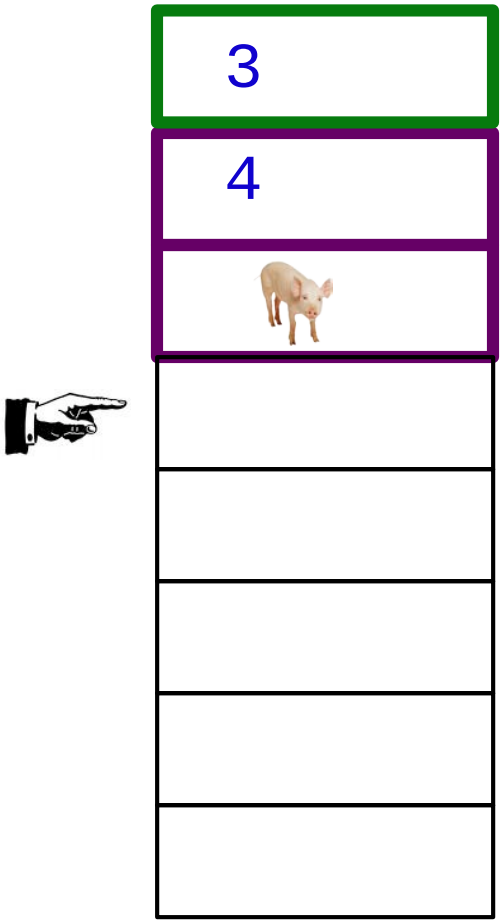
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



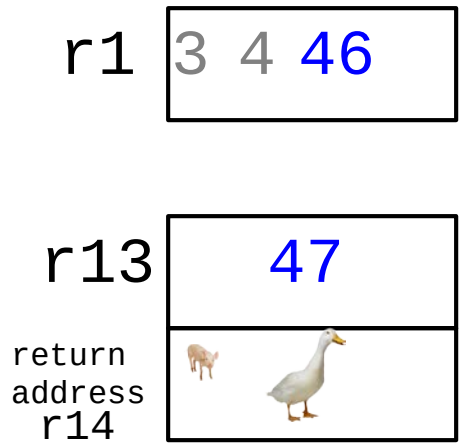
Function Calls…

Hmmm code up here!

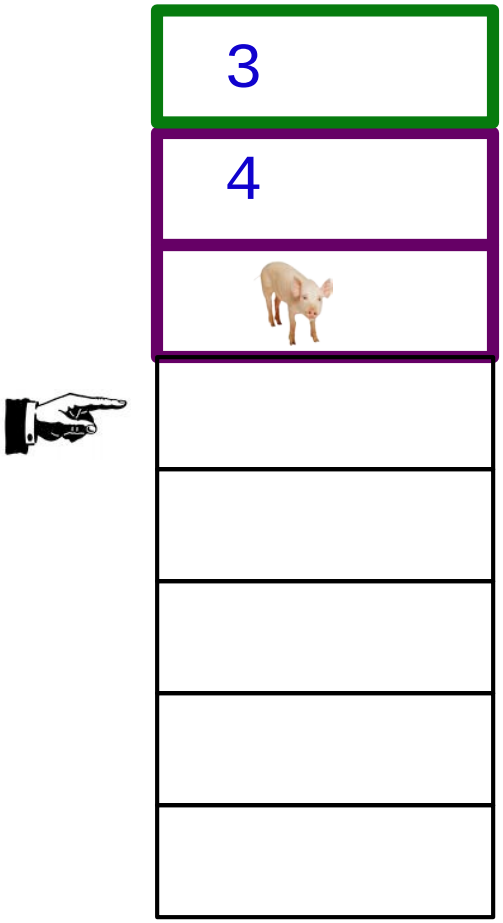
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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

r1 | 3 4 46

r13 | 47

return address r14

The stack in RAM!

3
4

Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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

r1 | 3 4 46

r13 | 47

return address r14

The stack in RAM!

3
4

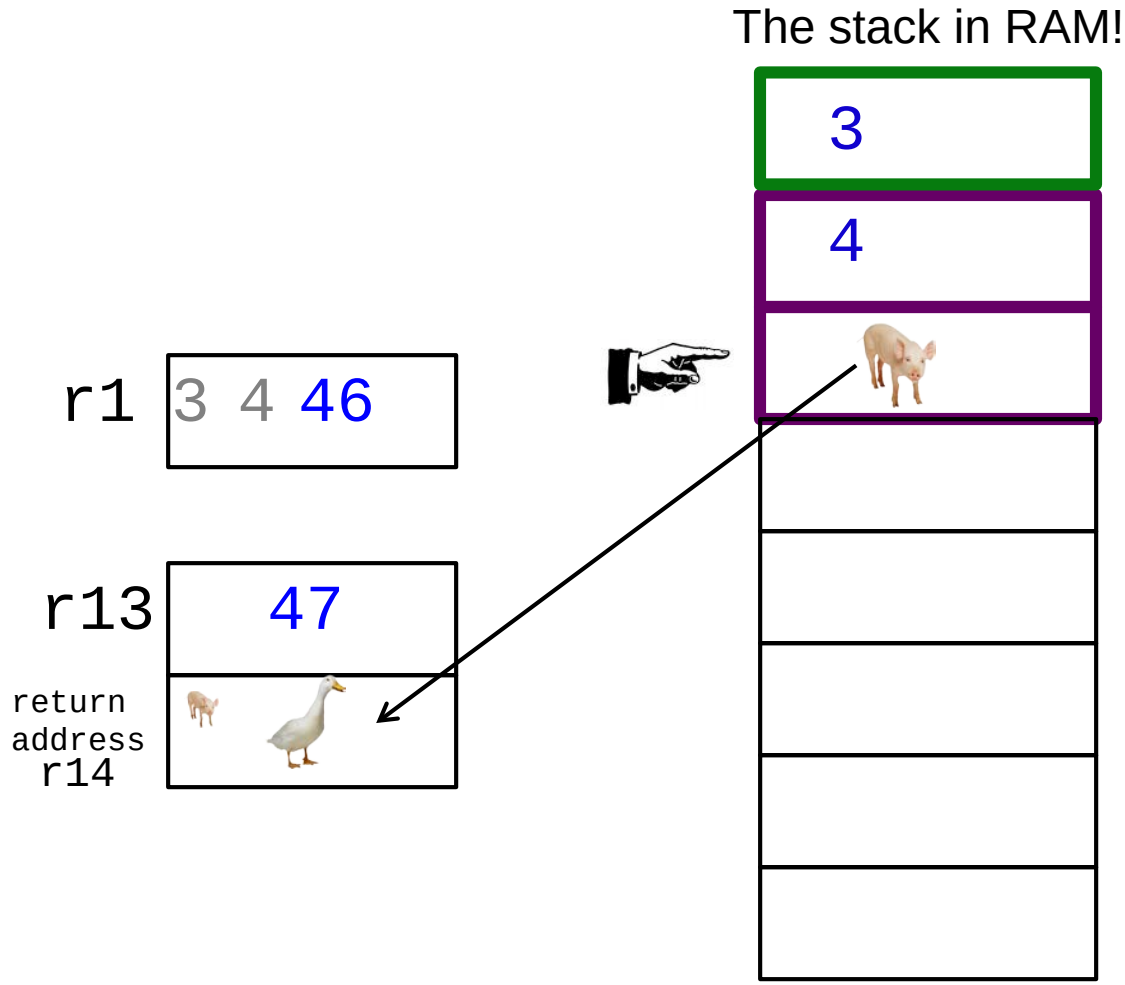
Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



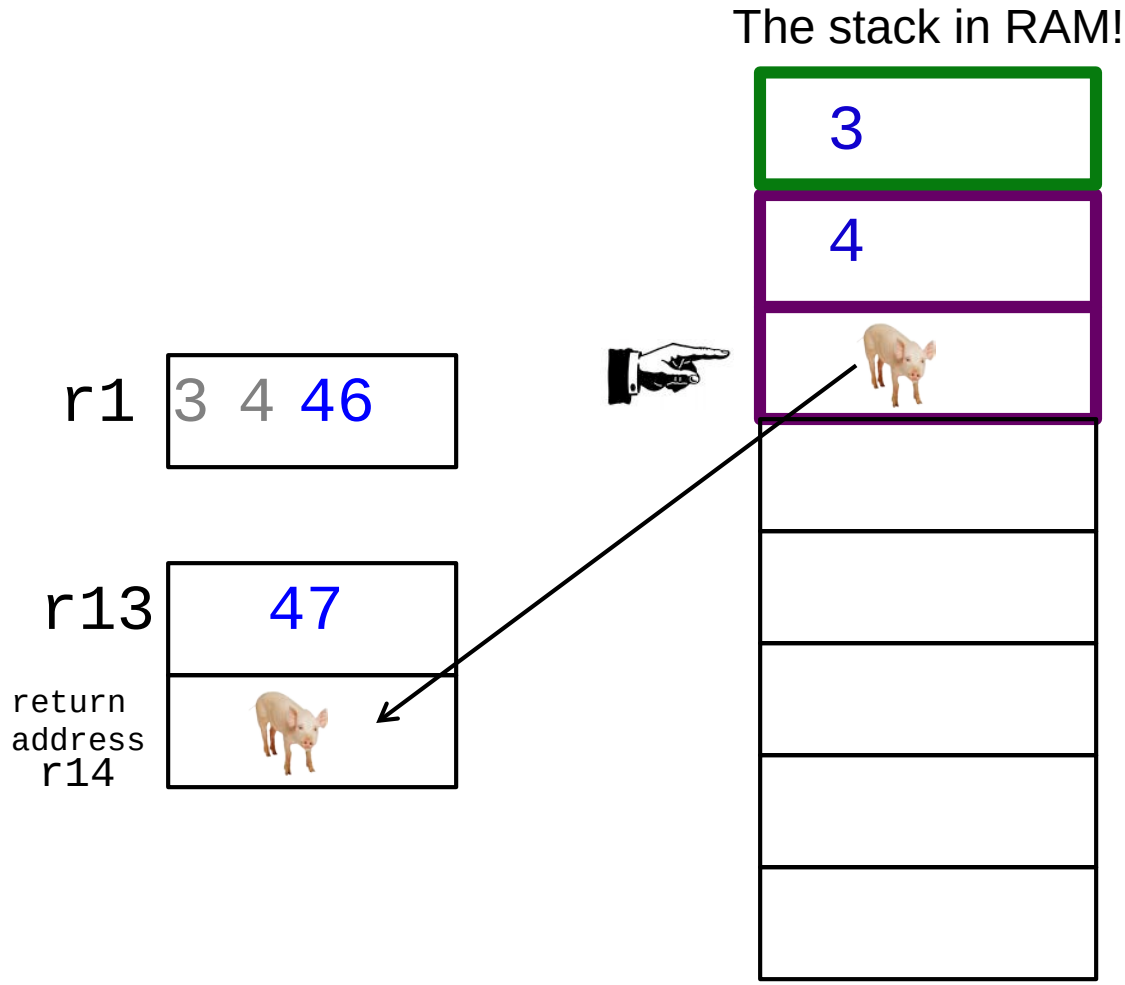
Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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

r1 | 3 4 46

r13 | 47

return address r14

The stack in RAM!

3
4

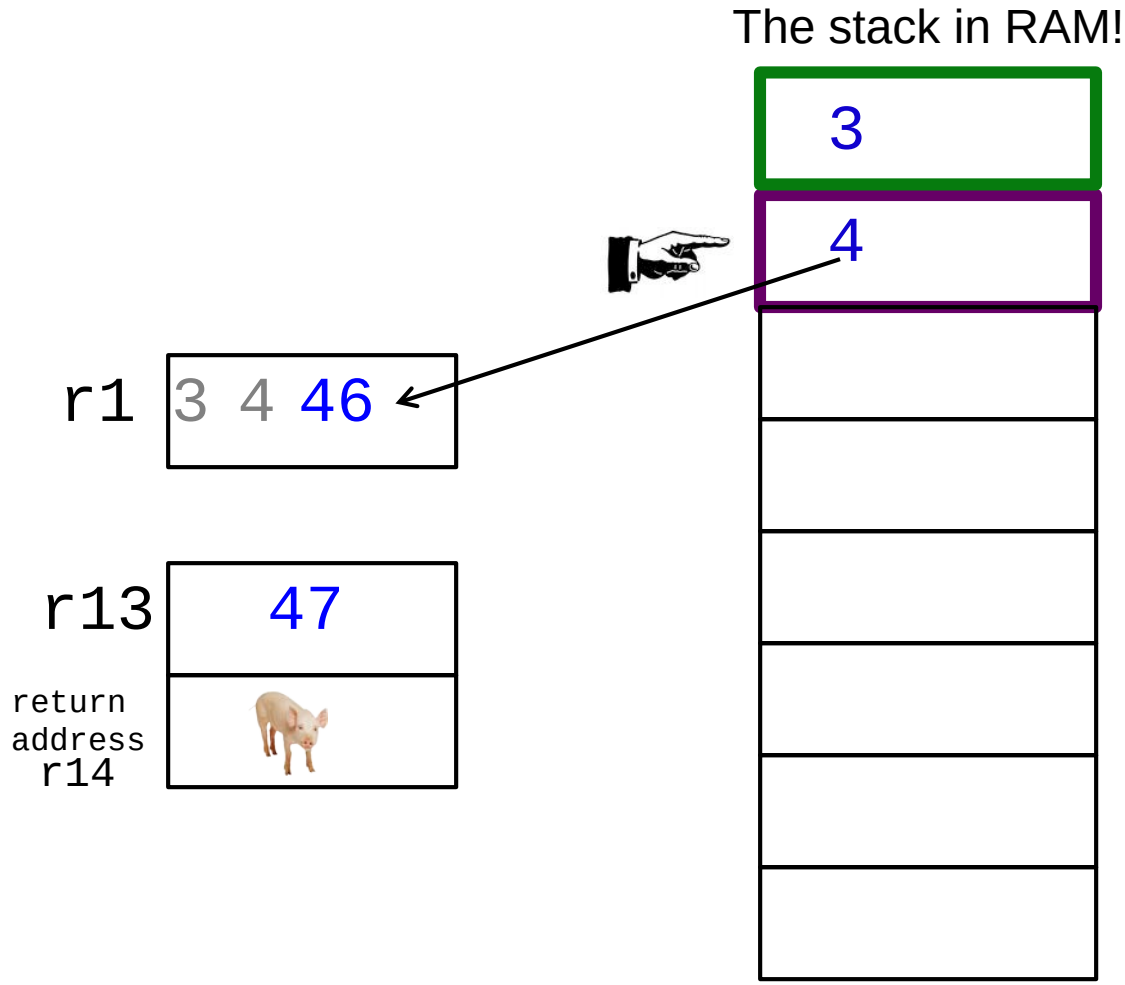
Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



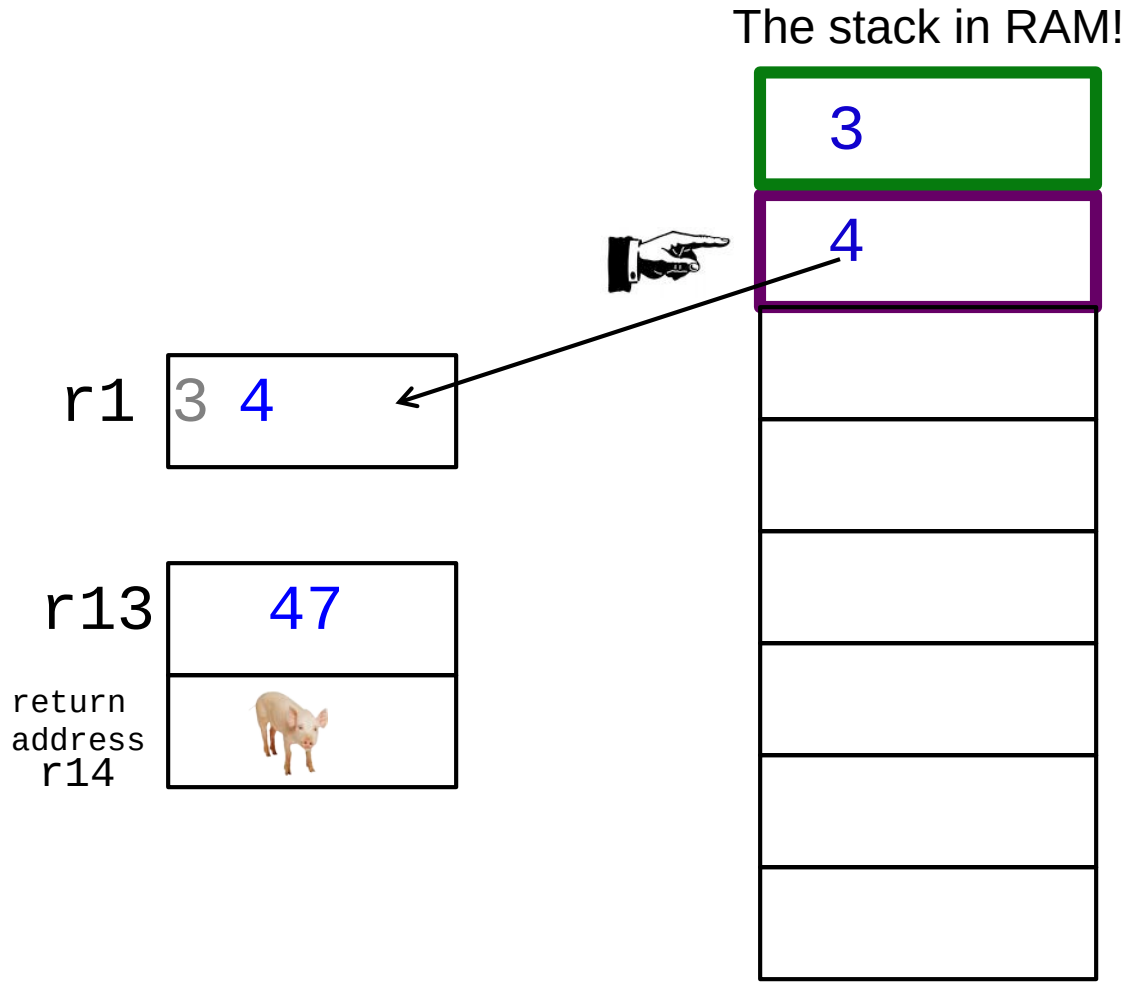
Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



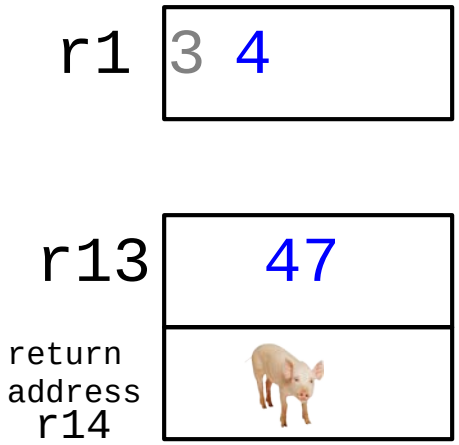
Function Calls…

Hmmm code up here!

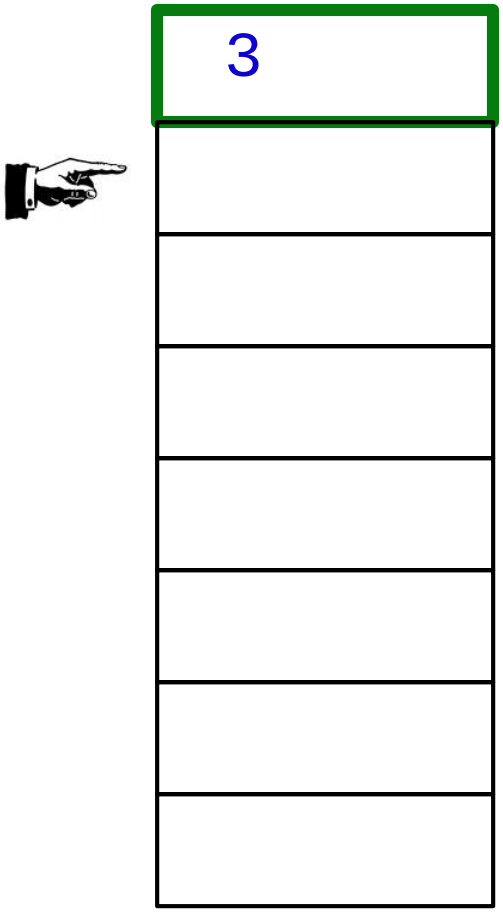
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



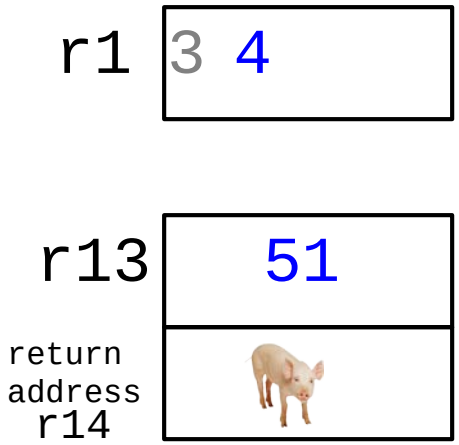
Function Calls…

Hmmm code up here!

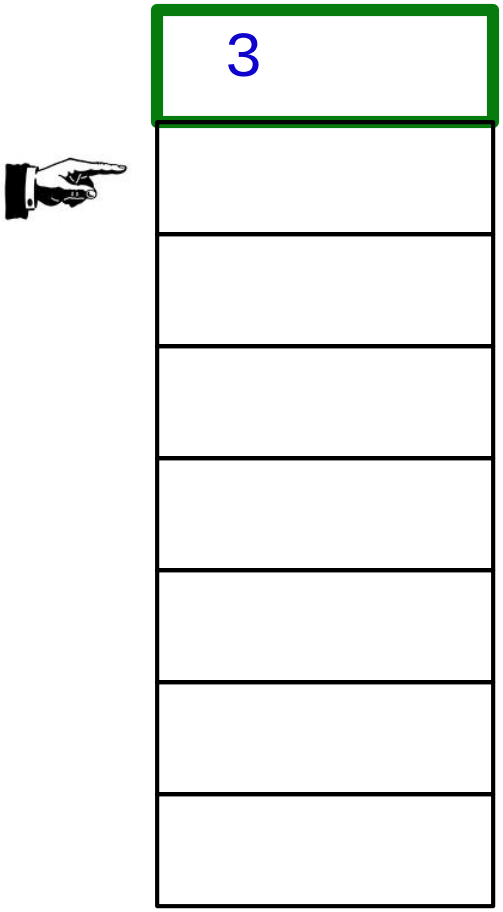
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



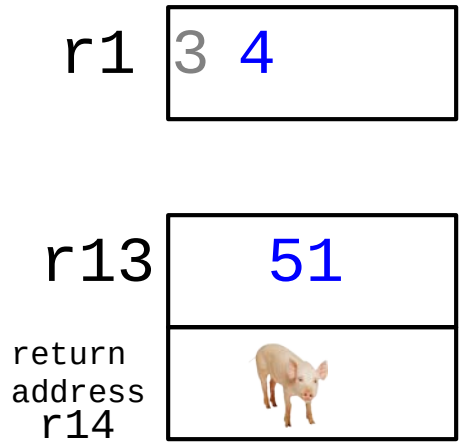
Function Calls…

Hmmm code up here!

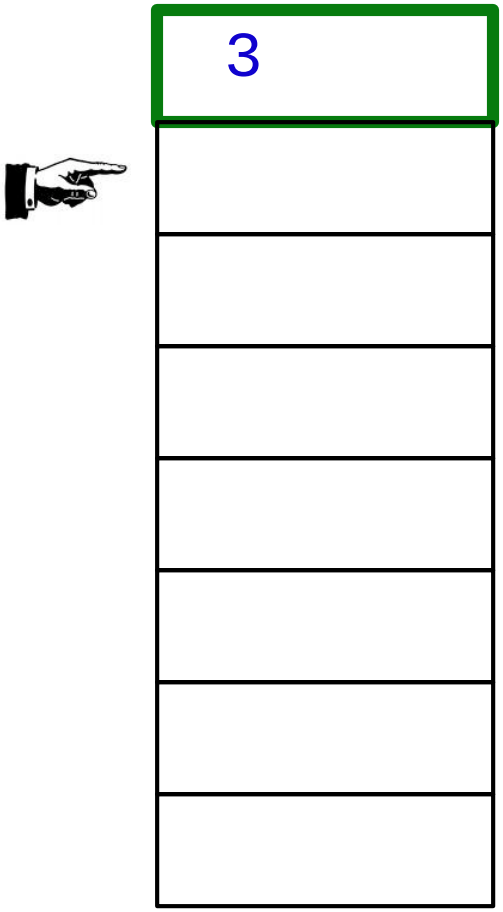
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



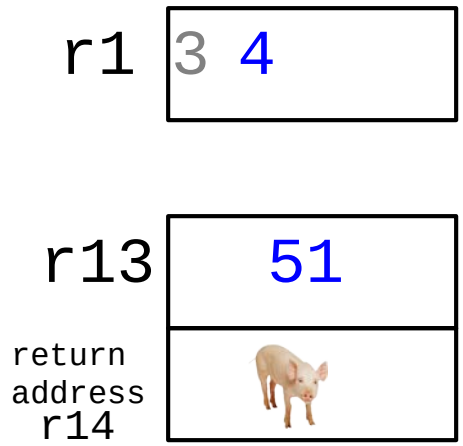
Function Calls…

Hmmm code up here!

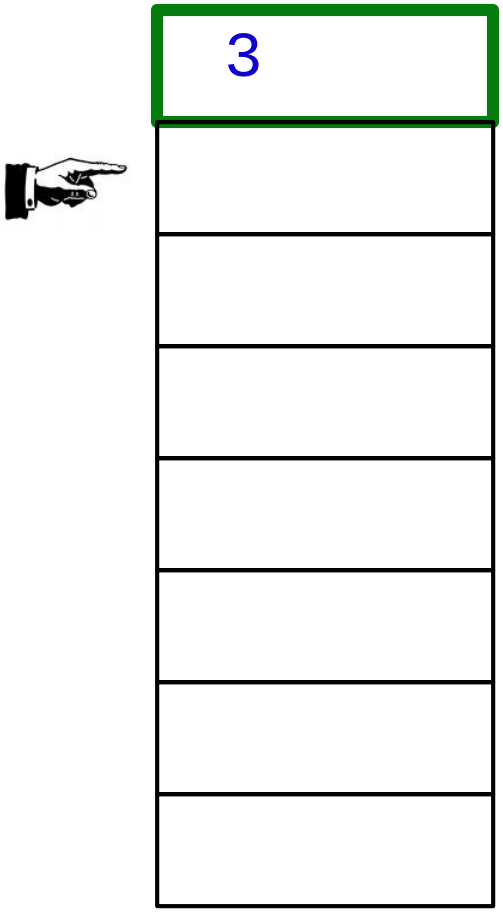
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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

r1 | 3 4

r13 | 51

return address r14

The stack in RAM!

3

Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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

r1 | 3 4

r13 | 51

return address r14

The stack in RAM!

3

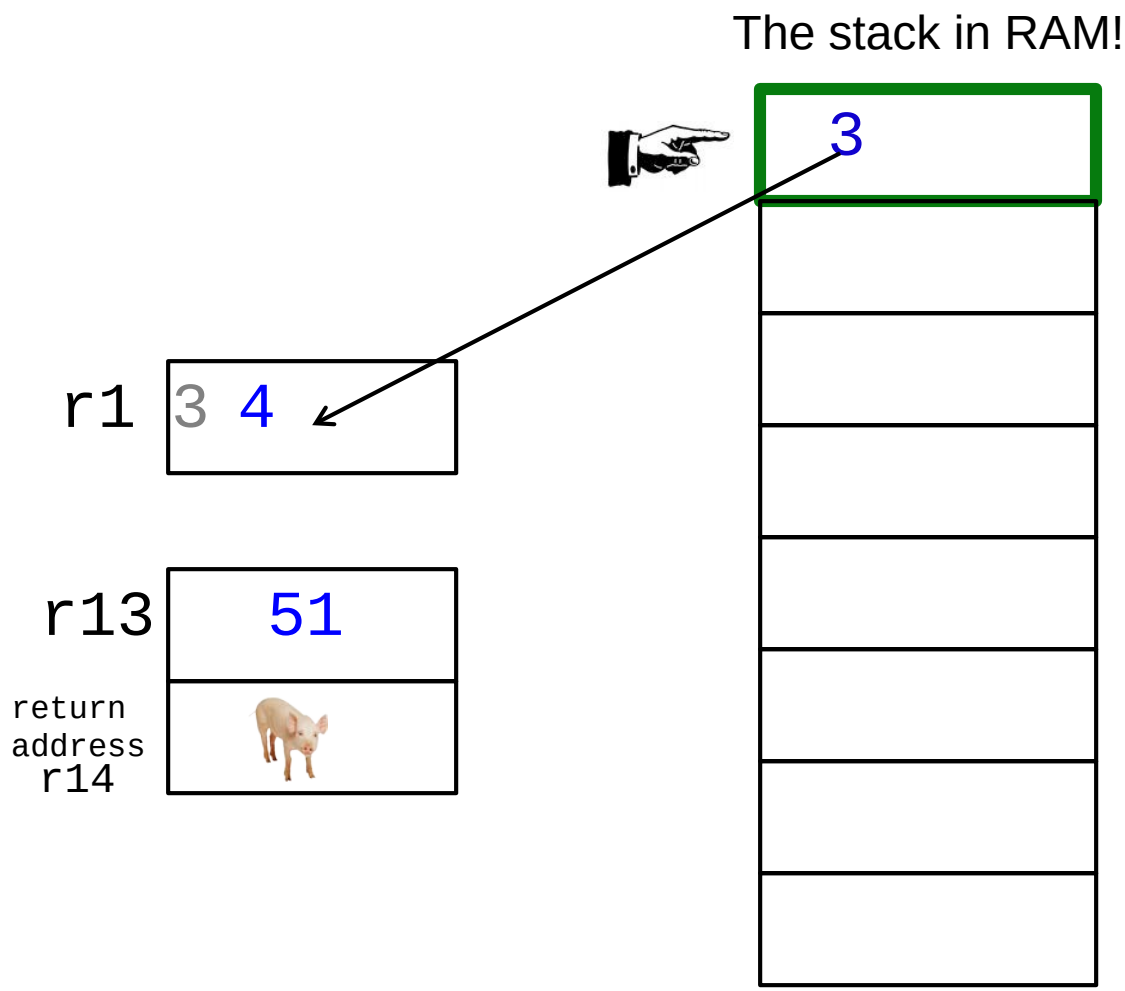
Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



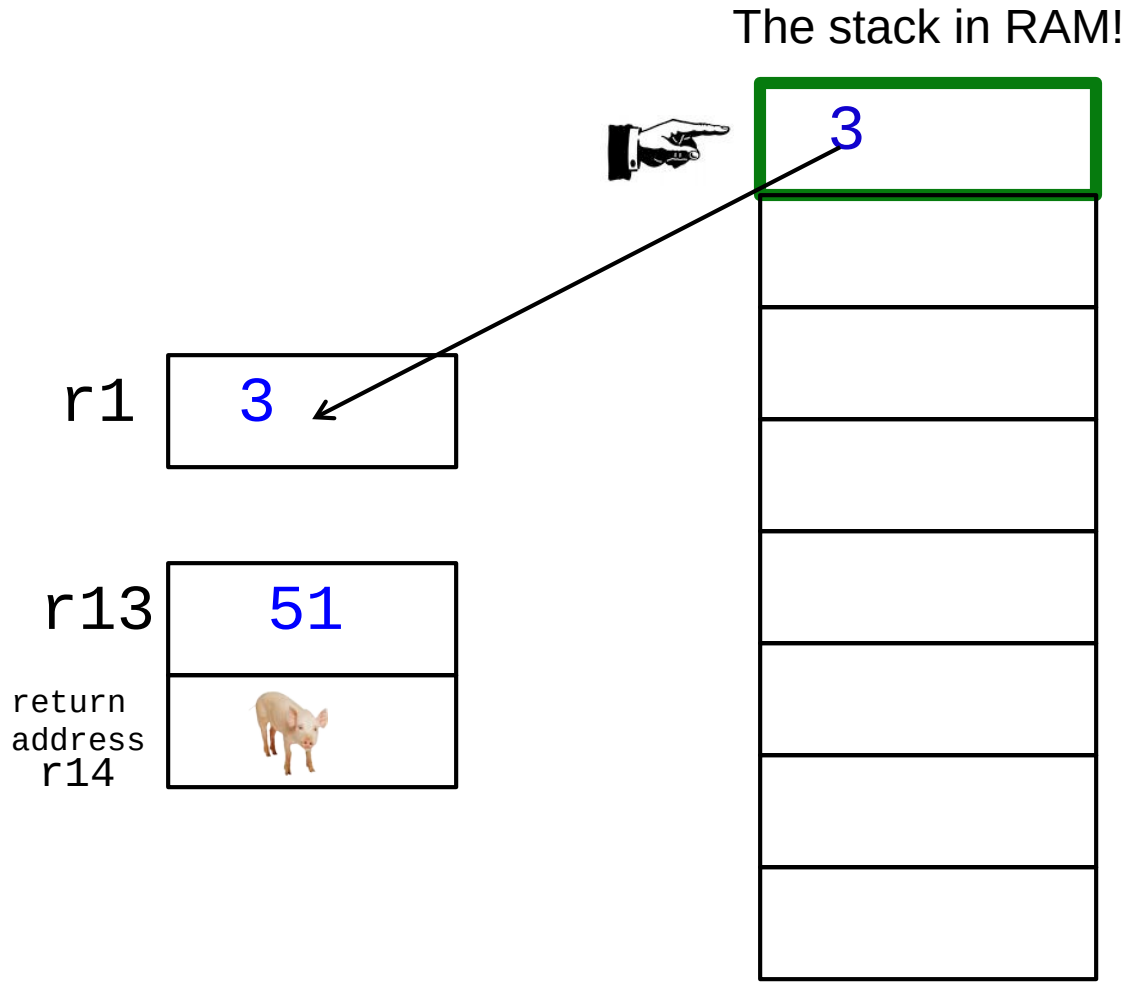
Function Calls…

Hmmm code up here!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



Function Calls…

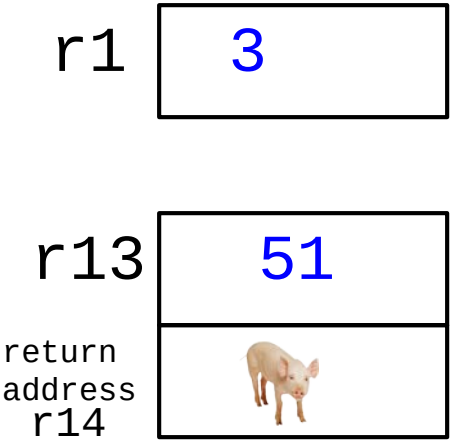
Hmmm code up here!

The stack in RAM!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



Function Calls…

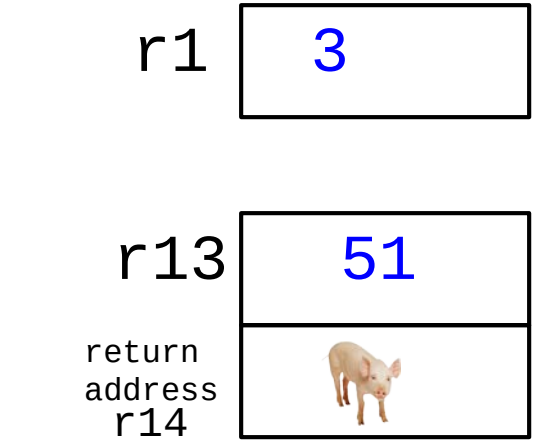
Hmmm code up here!

The stack in RAM!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



Function Calls…

Hmmm code up here!

The stack in RAM!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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

r1	3
r13	54
return address r14	

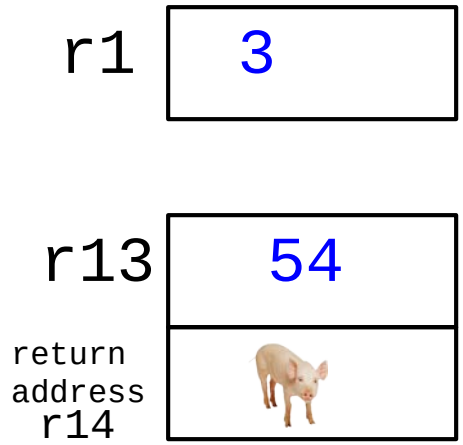
Function Calls…

Hmmm code up here!

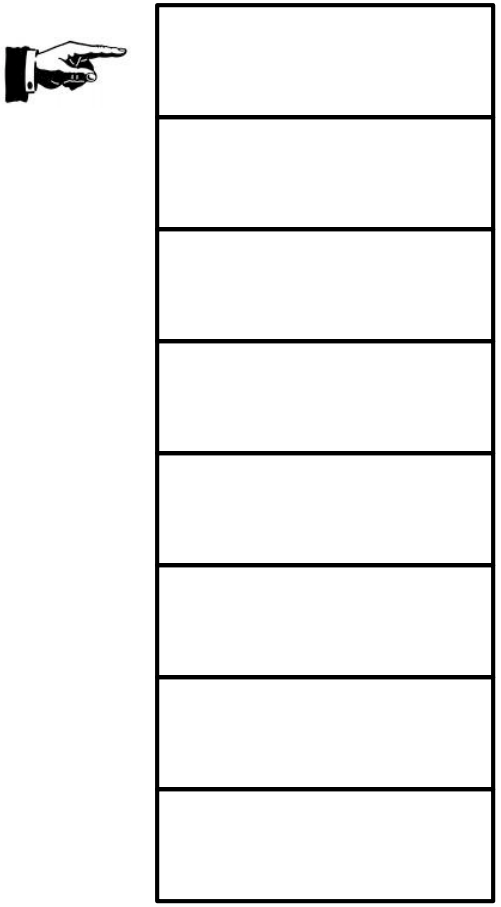
```
def main():
    r1=input()
    r13 = emma(r1)
    r13 = r13+ r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



The stack in RAM!



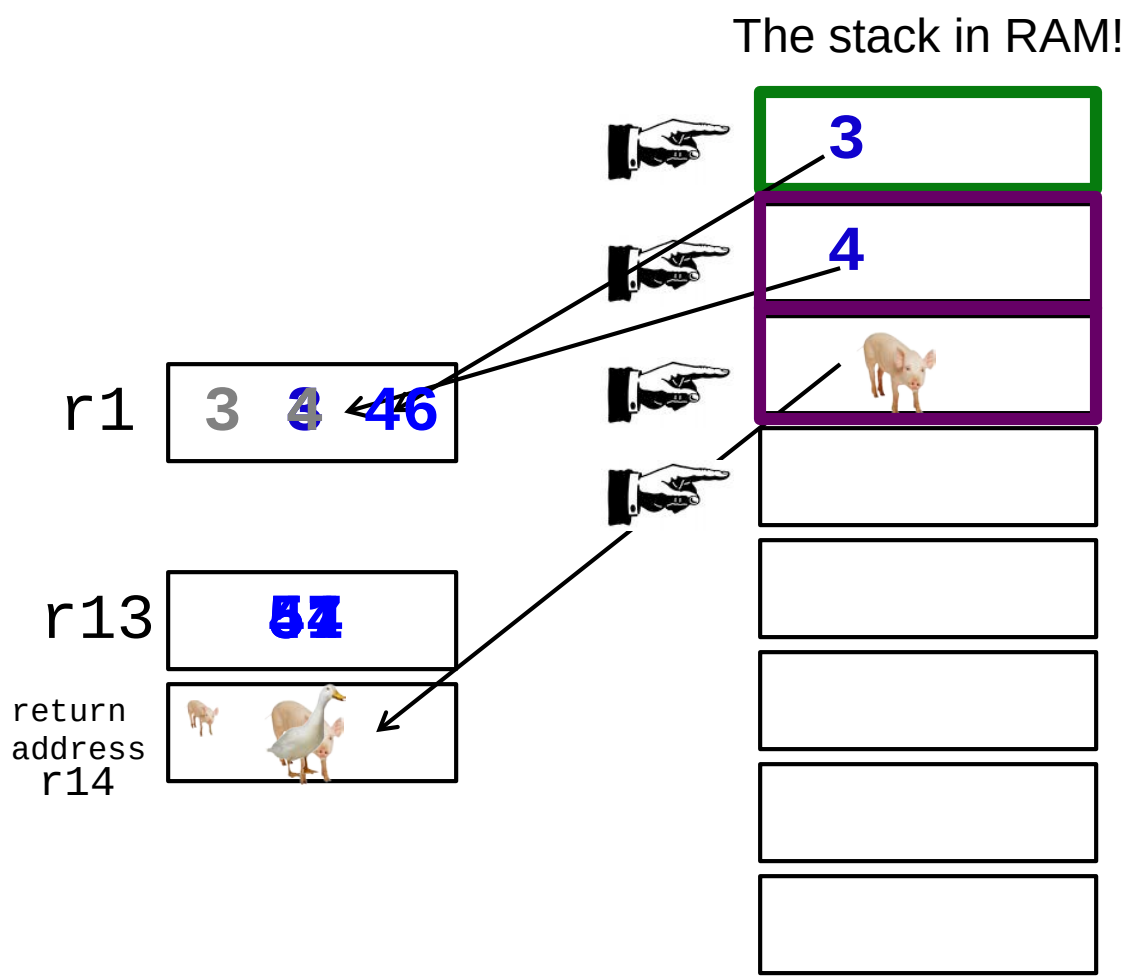
Summary? Function Calls…

Hmmm code up here!

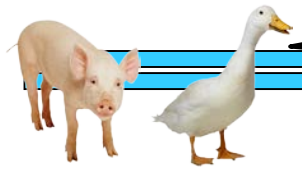
```
def main():
    r1 = input()
    r13 = emma(r1)
    r13 = r13 + r1
    print r13
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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



Now Without Pigs and Geese!



We object to this!!!

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

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

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

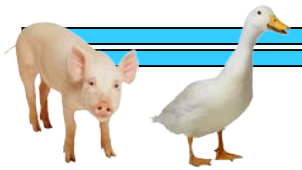
r1=3

r1	3	
r13		return value
r14		return address
r15	25	stack pointer

RAM!

0	01001001
1	11001011
…	
25	01001011
26	
27	
28	
29	
…	
255	

Now Without Pigs and Geese!



It was better with pigs and geese!

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

```
def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13
```

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

```
00 setn r15 26     # set stack pointer to 26
01 read r1         # start of main
02 storer r1 r15   # store r1 on the stack
03 addn r15 1      # increment stack pointer
04 calln r14 10    # call emma
05 addn r15 -1     # decrement stack pointer
06 loadr r1 r15    # load r1 from the stack
07 add r13 r13 r1  # r13 = r13 + r1
08 write r13
09 halt
10 addn r1 1       # start of emma!
11 storer r1 r15   # store r1 on the stack
12 addn r15 1      # increment stack pointer
13 storer r14 r15  # save return addr on stack
14 addn r15 1      # increment stack pointer
15 calln r14 22    # call sarah
16 addn r15 -1     # decrement stack pointer
17 loadr r14 r15   # load ret addr from stack
18 addn r15 -1     # decrement stack pointer
19 loadr r1 r15    # load r1 from the stack
20 add r13 r13 r1  # r13 = r13 + r1
21 jumpr r14       # return!
22 addn r1 42      # start of sarah!
23 setn r2 1       # put 1 in a register
24 add r13 r1 r2
25 jumpr r14       # return
```

save

restore

save

restore