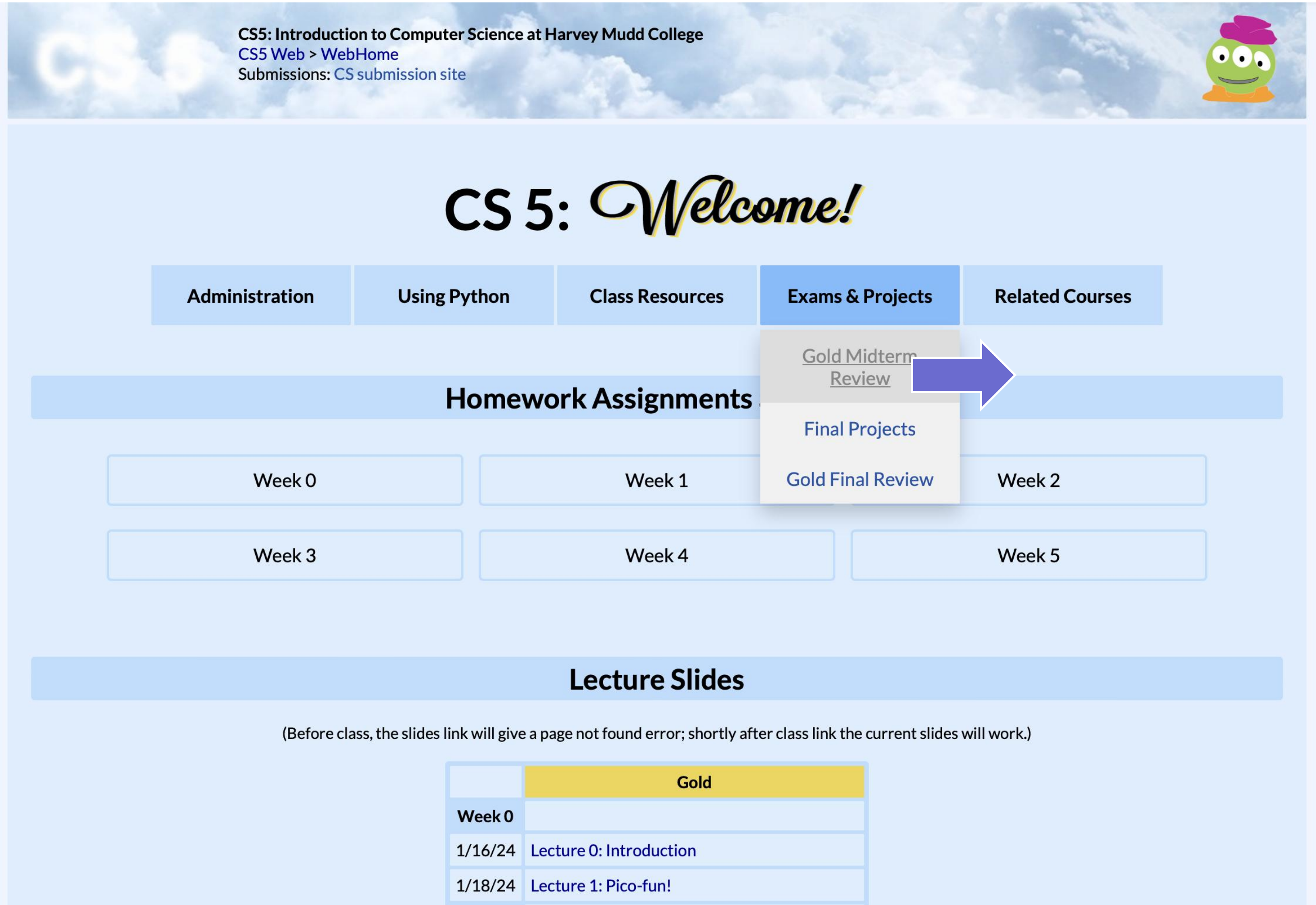
Thursday, Mar 13 will be the CS 5 in-class midterm



CS5: Introduction to Computer Science at Harvey Mudd College
CS5 Web > WebHome
Submissions: CS submission site

CS 5: Welcome!

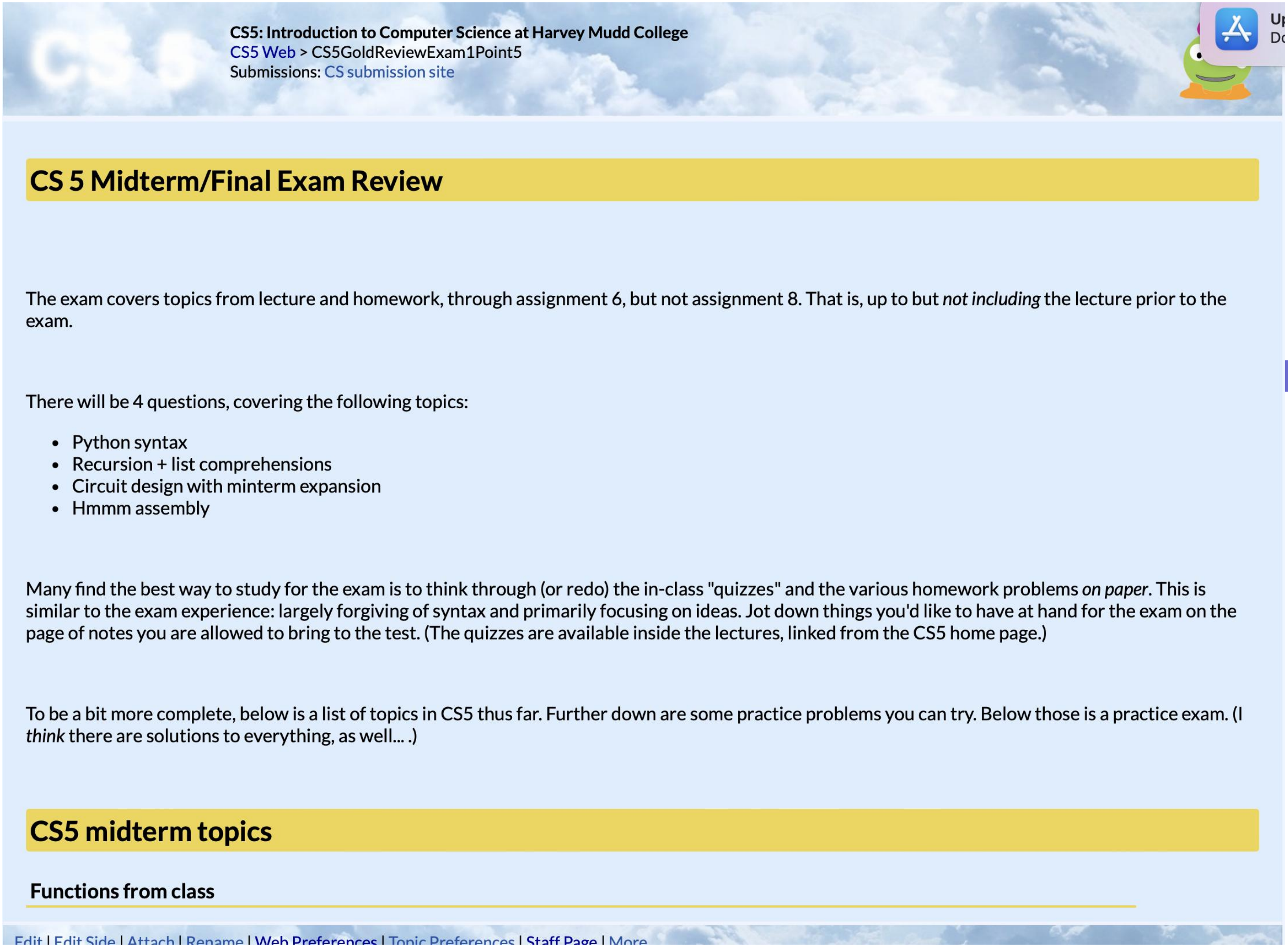
Homework Assignments

Lecture Slides

(Before class, the slides link will give a page not found error; shortly after class link the current slides will work.)

	Gold
Week 0	
1/16/24	Lecture 0: Introduction
1/18/24	Lecture 1: Pico-fun!

Thursday, Mar 13 will be the CS 5 in-class midterm



CS5: Introduction to Computer Science at Harvey Mudd College
CS5 Web > CS5GoldReviewExam1Point5
Submissions: CS submission site

CS 5 Midterm/Final Exam Review

The exam covers topics from lecture and homework, through assignment 6, but not assignment 8. That is, up to but *not including* the lecture prior to the exam.

There will be 4 questions, covering the following topics:

- Python syntax
- Recursion + list comprehensions
- Circuit design with minterm expansion
- Hmmm assembly

Many find the best way to study for the exam is to think through (or redo) the in-class "quizzes" and the various homework problems *on paper*. This is similar to the exam experience: largely forgiving of syntax and primarily focusing on ideas. Jot down things you'd like to have at hand for the exam on the page of notes you are allowed to bring to the test. (The quizzes are available inside the lectures, linked from the CS5 home page.)

To be a bit more complete, below is a list of topics in CS5 thus far. Further down are some practice problems you can try. Below those is a practice exam. (I *think* there are solutions to everything, as well... .)

CS5 midterm topics

Functions from class

CS5
Up
Do
Edit | Edit Side | Attach | Rename | Web Preferences | Topic Preferences | Staff Page | More

Thursday, Mar 13 will be the CS 5 "in-class" midterm

***Un*-warnings:**

only 5 minutes?

concerns? accommodations? flexibility:

five pr., 75 min., *written* worth 1 hw assignment

score worries? *Extra* extra-credit in hw9 and beyond

Suggestions:

go over hwk problems and our in-class exercises ...

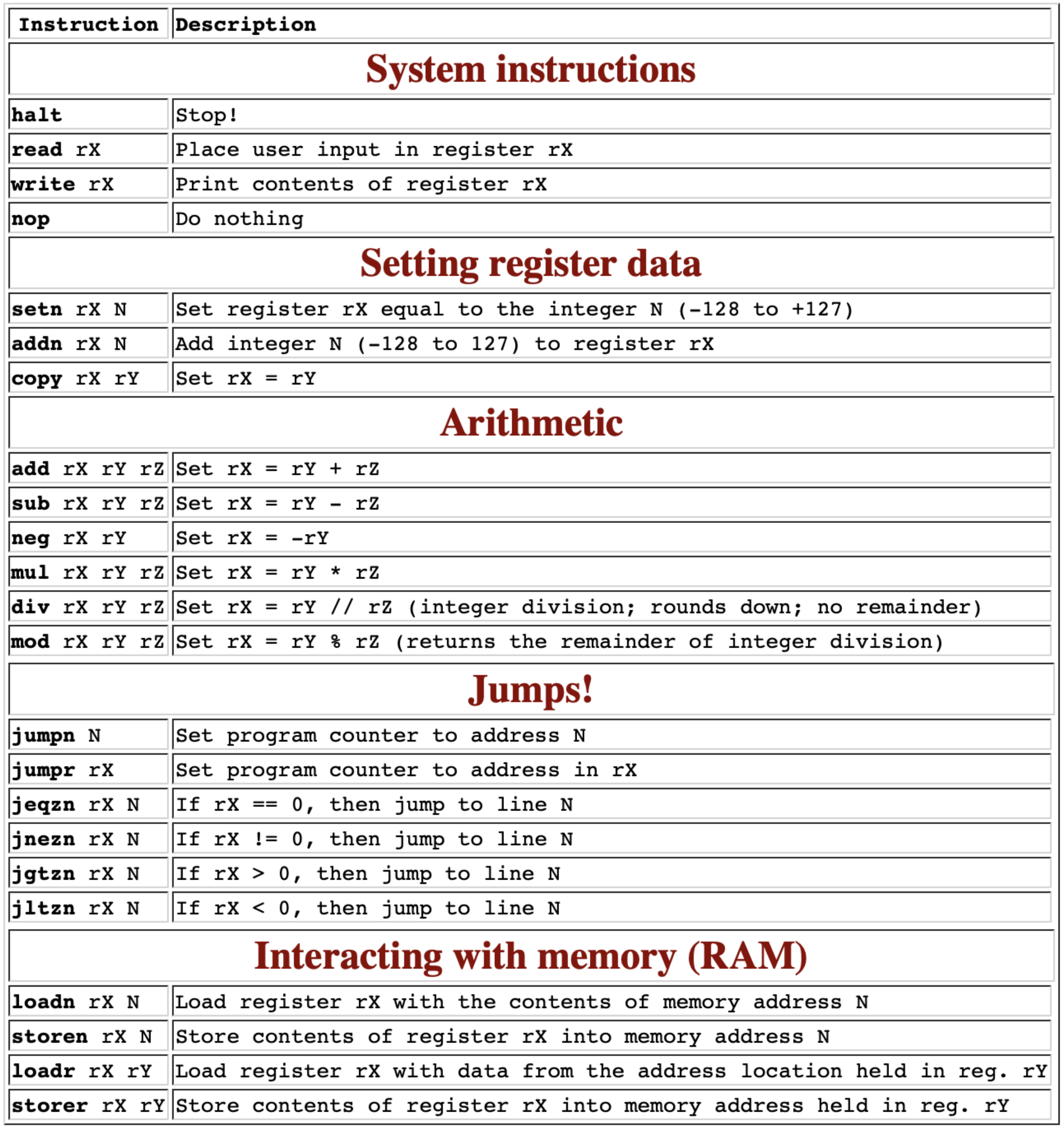
create a page of notes, 2-sided is OK

consider small *variations* of old problems...
... and how the solutions would change…

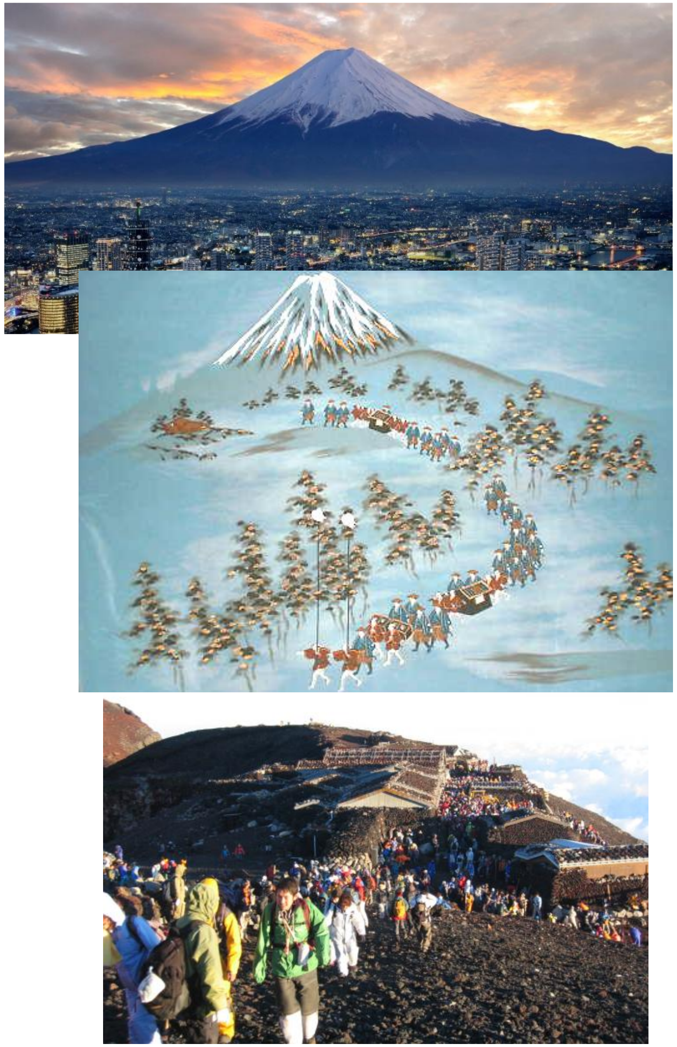
that's our approach...

Assembly Language

Hey, three instructions are missing here...



Instruction	Description
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
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; rounds down; 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
jeqzn rX N	If rX == 0, then jump to line N
jnezn rX N	If rX != 0, then jump to line N
jgtzn rX N	If rX > 0, then jump to line N
jltzn rX N	If rX < 0, then jump to line N
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
storer rX rY	Store contents of register rX into memory address held in reg. rY



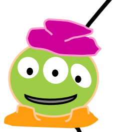
It's only the foolish who never climb Mt. Fuji -- *or who climb it again*.

「富士山に一度も登らぬバカ、二度登るバカ」

functions vs. **instructions**

Functions: Python

Instructions: Hmmm



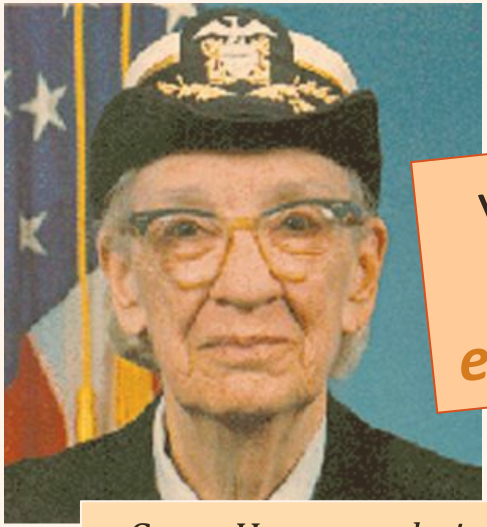
Python!

How does Python ***function***?

Hmmm

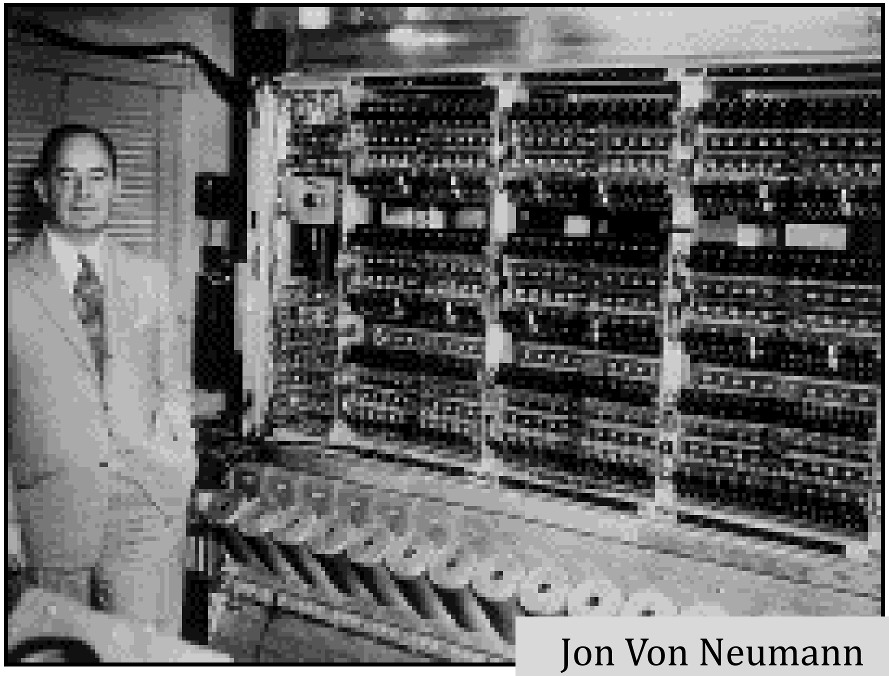
CS 5 Today

EPIC ~~RAP~~ CS BATTLES OF HISTORY



winning side - ***eventually***

Grace Hopper, *admiral + author of the first high-level (human-level) language, COBOL*



Jon Von Neumann

Functions

Instructions

Moral equivalents...

```
0: 0000 0001 0000 0001             00 read r1
1: 0000 0010 0000 0001             01 read r2
2: 0111 0011 0001 0010             02 sub r3 r1 r2
3: 0110 0000 0000 0000             03 nop
4: 1110 0011 0000 0111             04 jgtzn r3 7
5: 0000 0001 0000 0010             05 write r1
6: 1011 0000 0000 1000             06 jumpn 8
7: 0000 0010 0000 0010             07 write r2
8: 0000 0000 0000 0000             08 halt
```

```python
x = int(input("Num: "))
y = int(input("Num: "))
diff = x - r2y
if diff < 0:
    print(r1)
else:
    print(r2)
```

Moral equivalents…

```
0: 0000 0001 0000 0001                00 read r1
1: 0000 0010 0000 0001
2: 0111
3: 0110
4: 1110
5: 0000
6: 1011
7: 0000
8: 0000
```

Goal: To **<u>automatically</u>** translate Python functions into Hmmm instructions…

How? ***Conventions!***

Get into a rut -- and stay there! - V. Rokhlin

```
print(r2)
```

“Mindless” translation…

Make it so simple it can be *automated*…

- Put variables into registers in order they occur (starting with `r1`, then `r2`, etc…)
- Use jumps to simulate if blocks.

These are “conventions”…

What about functions?

```
def min(x, y):
    diff = x - y
    if diff < 0:
        return x
    else:
        return y
```

```
10   sub r3 r1 r2
11   jgtzn r3 14
12   copy r13 r1
13   …?? return ??…
14   copy r13 r2
15   …?? return ??…
```

```
a = int(input("Num: "))
b = int(input("Num: "))
r = min(a, b)
print(r)
```

```
00   read r1
01   read r2
02   … ??? …
03   halt
```

- Put variables into registers (starting with `r1`, then `r2`, etc…)
- Use jumps to simulate if blocks.
- Return the result in `r13`

What about functions?

```
def min(x, y):
    diff = x - y
    if diff < 0:
        return x
    else:
        return y
```

```
10   sub r3 r1 r2
11   jgtzn r3 14
12   copy r13 r1
13   jumpn 03
14   copy r13 r2
15   jumpn 03
```

```
a = int(input("Num: "))
b = int(input("Num: "))
r = min(a, b)
print(r)
```

```
00   read r1
01   read r2
02   jumpn 10
03   write r13
04   halt
```

- Put variables into registers (starting with `r1`, then `r2`, etc...)
- Use jumps to simulate if blocks.
- Return the result in `r13`

What about functions?

```
def min(x, y):
    diff = x - y
    if diff < 0:
        return x
    else:
        return y
```

```
10   sub r3 r1 r2
11   jgtzn r3 14
12   copy r13 r1
13   jumpr r14
14   copy r13 r2
15   jumpr r14
```

jumpr r14

jumps to the line ***held in r14***

```
a = int(input("Num: "))
b = int(input("Num: "))
r = min(a, b)
print(r)
```

```
00   read r1
01   read r2
02   setn r14 04
03   jumpn 10
04   write r13
05   halt
```

- Put variables into registers (starting with `r1`, then `r2`, etc…)
- Use jumps to simulate if blocks.
- Return the result in `r13`
- Use `r14` to hold *where* to return to…

What about functions?

```
def min(x, y):
    diff = x - y
    if diff < 0:
        return x
    else:
        return y
```

```
10   sub r3 r1 r2
11   jgtzn r3 14
12   copy r13 r1
13   jumpr r14
14   copy r13 r2
15   jumpr r14
```

jumpr r14

jumps to the line ***held in r14***

```
a = int(input("Num: "))
b = int(input("Num: "))
r = min(a, b)
print(r)
```

```
00   read r1
01   read r2
02   calln r14 10
03   write r13
04   halt
```

calln r14 10

*jumps to line 10, and puts the **next line #** **into r14***

- Put variables into registers (starting with `r1`, then `r2`, etc…)
- Use jumps to simulate if blocks.
- Return the result in `r13`
- Use `r14` to hold *where* to return to (+ use `calln` instruction)

Does it work?

```
a = int(input("Num: "))
b = int(input("Num: "))
c = int(input("Num: "))
r1 = min(a, b)
r2 = min(r1, c)
print(r2)


def min(x, y):
    diff = x - y
    if diff < 0:
        return x
    else:
        return y
```

	Mem Location	Instruction
	00	read r1
	01	read r2
	02	read r3
	03	calln r14 10
main	04	copy r1 r13
	05	copy r2 r3
	06	calln r14 10
	07	write r13
	08	halt
	09	nop
	10	sub r3 r1 r2
	11	jgtzn r3 14
min	12	copy r13 r1
	13	jumpr r14
	14	copy r13 r2
	15	jumpr r14

- Put variables into registers (starting with `r1`, then `r2`, etc…)
- Use jumps to simulate if blocks.
- Return the result in `r13`
- Use `r14` to hold *where* to return to (+ use `calln` instruction)

Quiz

Name(s) ____________________________

What var(s)?

r1

r2

r3

r13

r14

	Mem Location	Instruction
main	00	read r1
	01	read r2
	02	read r3
	03	calln r14 10
	04	copy r1 r13
	05	copy r2 r3
	06	calln r14 10
	07	write r13
	08	halt
	09	nop
min	10	sub r3 r1 r2
	11	jgtzn r3 14
	12	copy r13 r1
	13	jumpr r14
	14	copy r13 r2
	15	jumpr r14

Try with the input 5, 42, 54

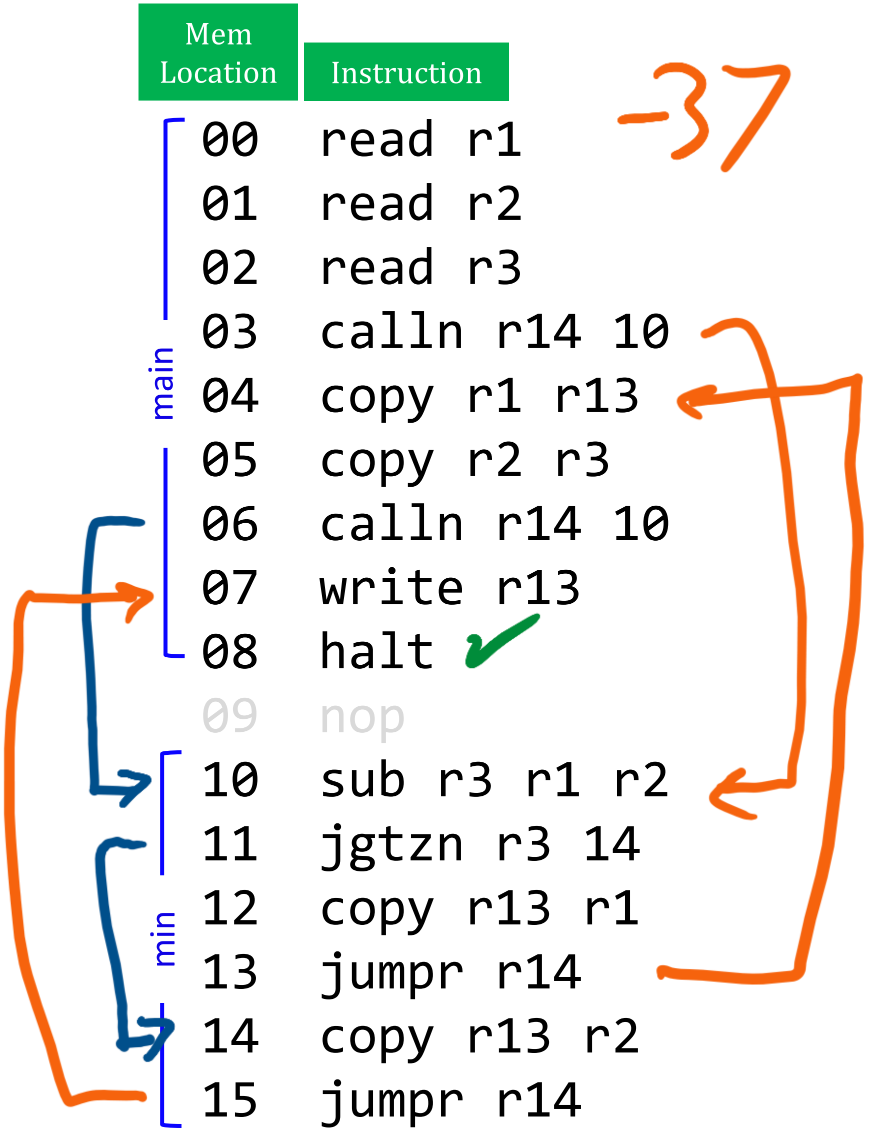
- What variable(s) from the original Python do r1, r2 and r3 hold?
- Does it work? If not, what went wrong and how could you fix it...?

Quiz

Name(s) ____________________

What var(s)?

r1	5 ~~8~~
r2	-37 ~~42~~
r3	42 ~~-37~~ ~~54~~
r13	-37 ~~5~~
r14	7 ~~4~~



-37

main

min

Mem Location	Instruction
00	read r1
01	read r2
02	read r3
03	calln r14 10
04	copy r1 r13
05	copy r2 r3
06	calln r14 10
07	write r13
08	halt ✓
09	nop
10	sub r3 r1 r2
11	jgtzn r3 14
12	copy r13 r1
13	jumpr r14
14	copy r13 r2
15	jumpr r14

Try with the input 5, 42, 54

- What variable(s) from the original Python do r1, r2 and r3 hold?
- Does it work? If not, what went wrong and how could you fix it…?

Fixing it...

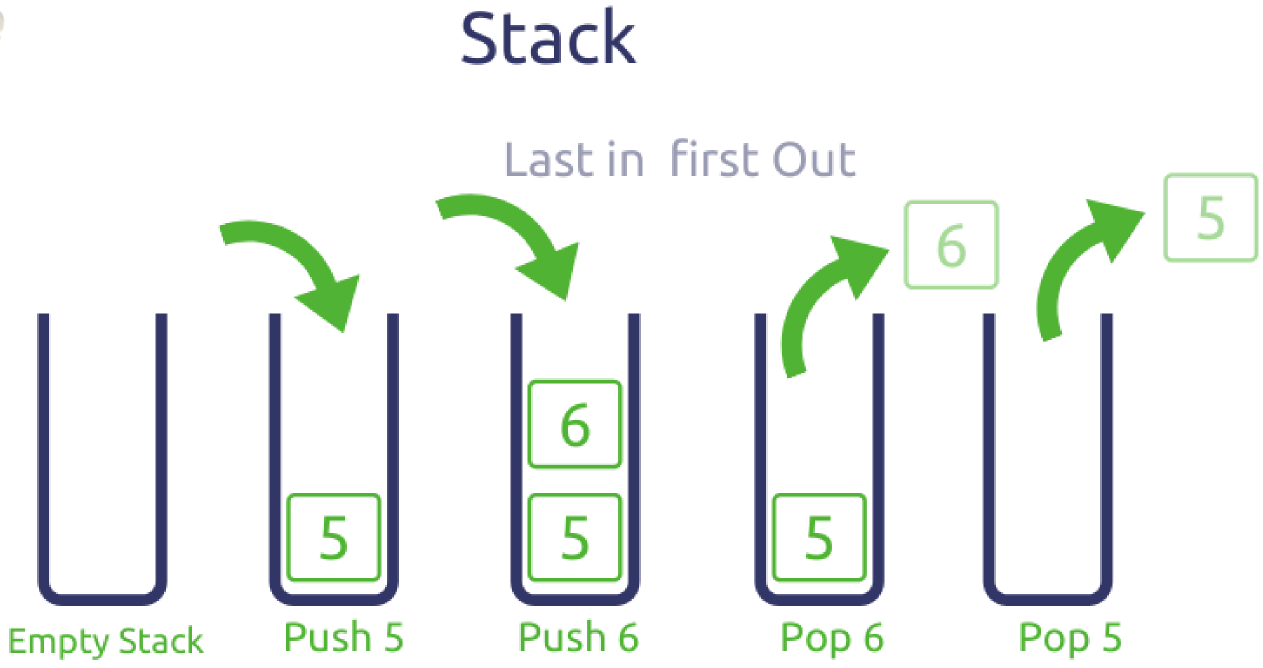
```
a = int(input("Num: "))
b = int(input("Num: "))
c = int(input("Num: "))
r1 = min(a, b)
r2 = min(r1, c)
print(r2)


def min(x, y):
    diff = x - y
    if diff < 0:
        return x
    else:
        return y
```

	Mem Location	Instruction
main	00	read r1
	01	read r2
	02	read r3
	03	calln r14 10
	04	copy r1 r13
	05	copy r2 r3
	06	calln r14 10
	07	write r13
	08	halt
	09	nop
min	10	***SAVE r3***
	11	sub r3 r1 r2
	12	jgtzn r3 15
	13	copy r13 r1
	14	jumpn 16
	15	copy r13 r2
	16	***RESTORE r3***
	17	jumpr r14

- Save/restore registers that we'll "clobber".

Big idea for save/restore: Stack!



Fixing it...

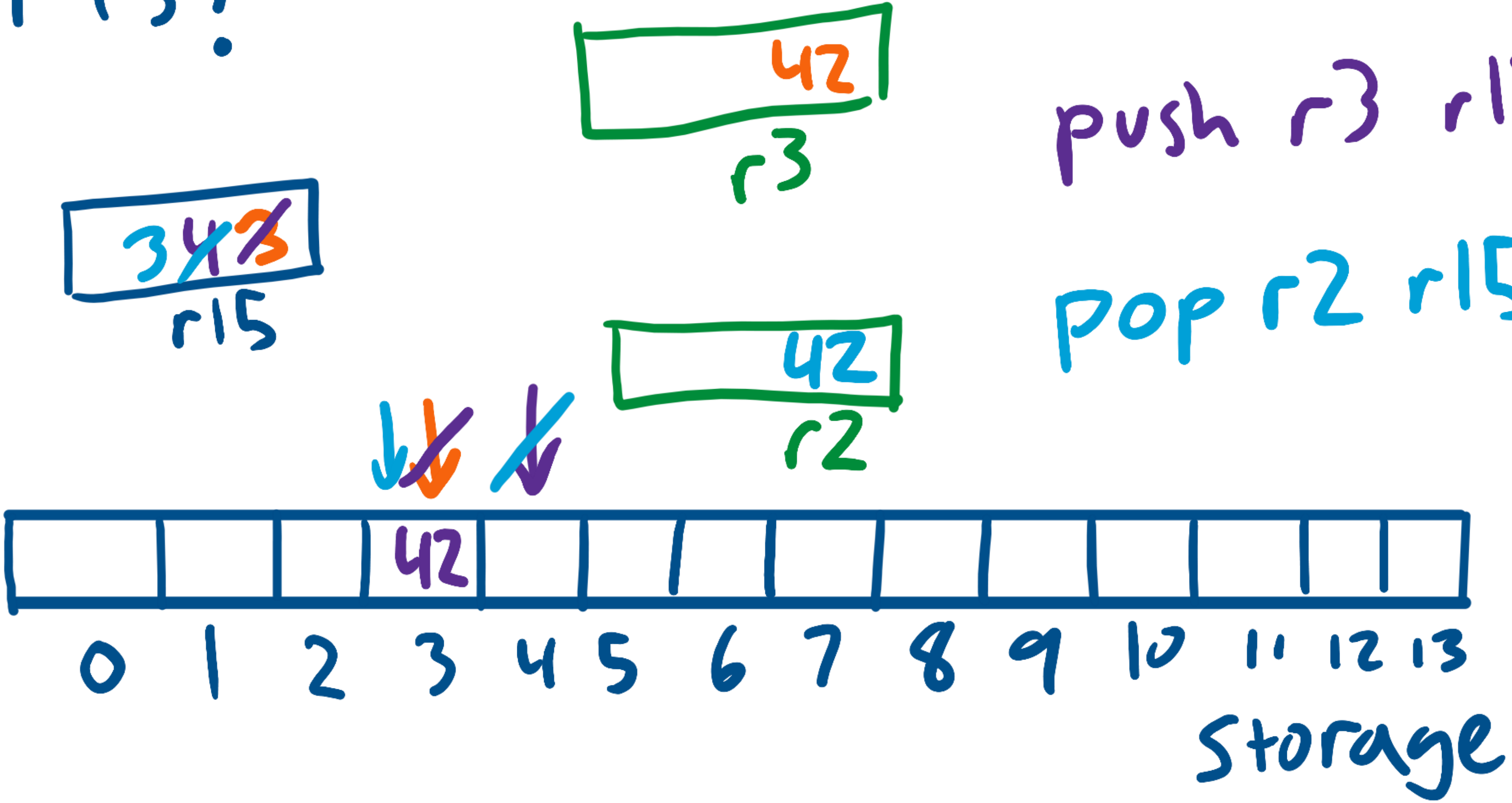
```
a = int(input("Num: "))
b = int(input("Num: "))
c = int(input("Num: "))
r1 = min(a, b)
r2 = min(r1, c)
print(r2)


def min(x, y):
    diff = x - y
    if diff < 0:
        return x
    else:
        return y
```

	Mem Location	Instruction
main	00	setn r15 42
	01	read r1
	02	read r2
	03	read r3
	04	calln r14 10
	05	copy r1 r13
	06	copy r2 r3
	07	calln r14 10
	08	write r13
	09	halt
min	10	push r3 r15
	11	sub r3 r1 r2
	12	jgtzn r3 15
	13	copy r13 r1
	14	jumpn 16
	15	copy r13 r2
	16	pop r3 r15
	17	jumpr r14

- Use r15 to refer to stack memory to pushr/popr...

r15?



push r3 r15

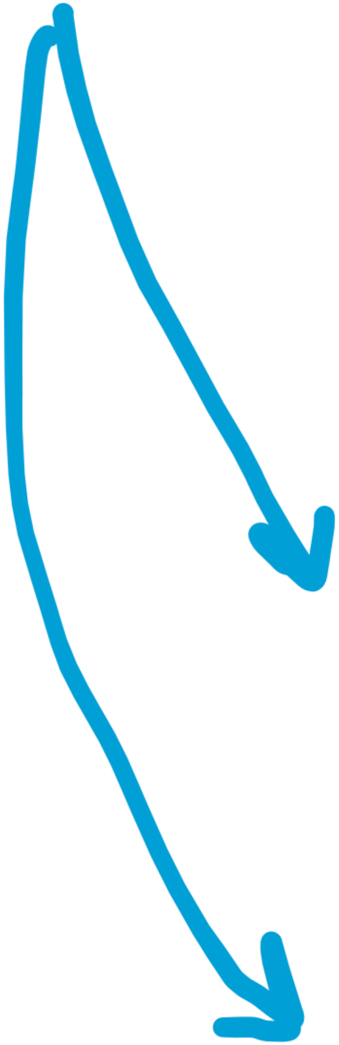
pop r2 r15

*r15 holds the offset, not the values.

Behind the scenes...

```
a = int(input("Num: "))
b = int(input("Num: "))
c = int(input("Num: "))
r1 = min(a, b)
r2 = min(r1, c)
print(r2)


def min(x, y):
    diff = x - y
    if diff < 0:
        return x
    else:
        return y
```



	Mem Location	Instruction
main	00	setn r15 42
	01	read r1
	02	read r2
	03	read r3
	04	calln r14 10
	05	copy r1 r13
	06	copy r2 r3
	07	calln r14 10
	08	write r13
	09	halt
min	10	storer r3 r15
	11	addn r15 1
	12	sub r3 r1 r2
	13	jgtzn r3 16
	14	copy r13 r1
	15	jumpn 17
	16	copy r13 r2
	17	addn r15 -1
	18	loadr r3 r15
	19	jumpr r14

- Use r15 to refer to scratch memory to save/restore...

Fixing it...

```python
a = int(input("Num: "))
b = int(input("Num: "))
c = int(input("Num: "))
r1 = min(a, b)
r2 = min(r1, c)
print(r2)


def min(x, y):
    diff = x - y
    if diff < 0:
        return x
    else:
        return y
```

	Mem Location	Instruction
main	00	setn r15 42
	01	read r1
	02	read r2
	03	read r3
	04	calln r14 10
	05	copy r1 r13
	06	copy r2 r3
	07	calln r14 10
	08	write r13
	09	halt
min	10	push r3 r15
	11	sub r3 r1 r2
	12	jgtzn r3 15
	13	copy r13 r1
	14	jumpn 16
	15	copy r13 r2
	16	pop r3 r15
	17	jumpr r14

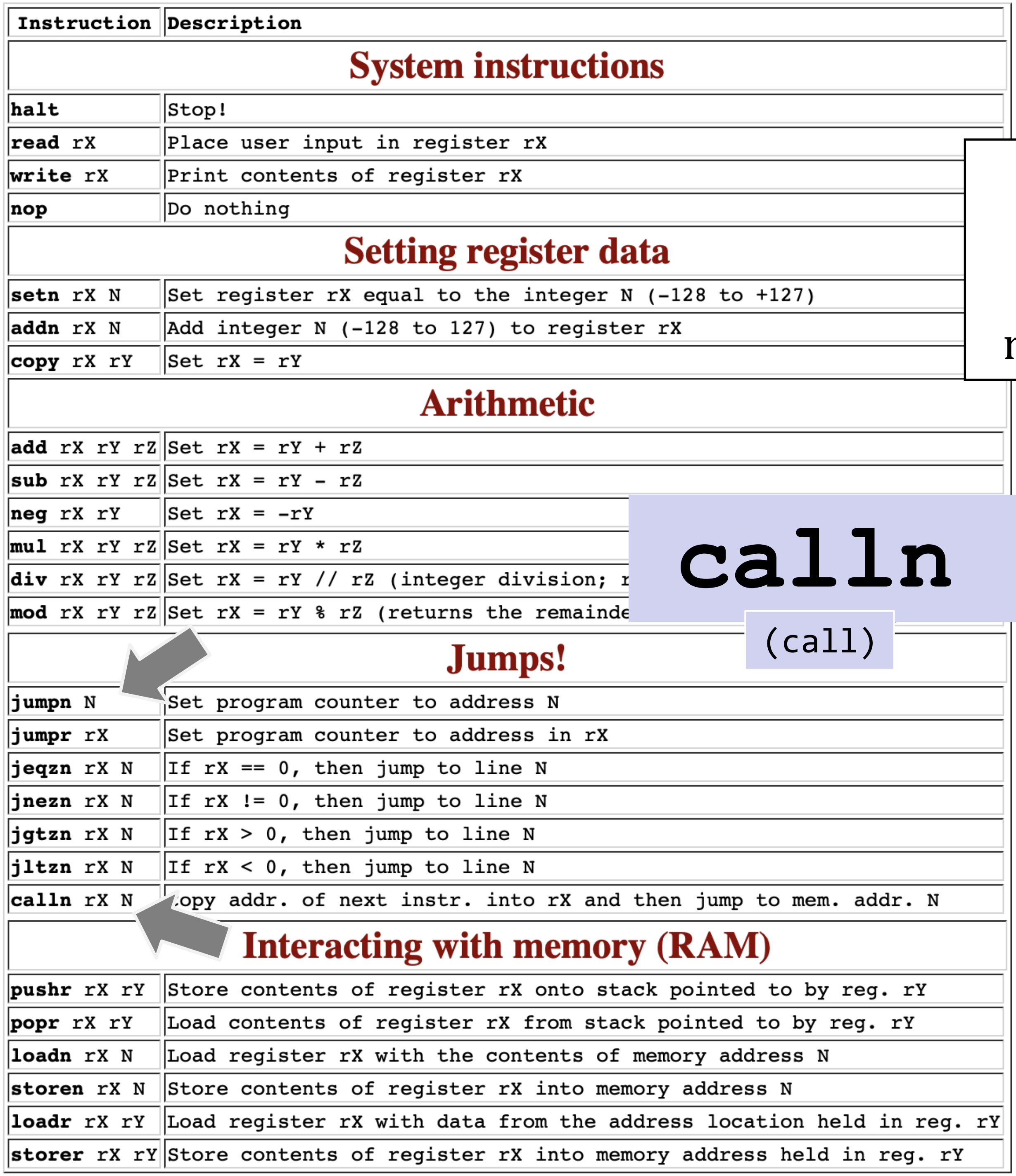
- Use r15 to refer to stack memory to pushr/popr...

These are “conventions”…

“Mindless” translation…

Make it so simple it can be *automated*…

- Put variables into registers in order they occur (starting with `r1`, then `r2`, etc…)
- Use jumps to simulate if blocks.
- Return the result in `r13`
- Use `r14` to hold where to return to (+ use `calln` instruction)
- Use `r15` to refer to stack memory
- Use `pushr`/`popr` to save/restore any register we'll clobber (i.e., whose value we don't want to lose)



Instruction	Description
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
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; r
mod rX rY rZ	Set rX = rY % rZ (returns the remainde
Jumps!	
jumpn N	Set program counter to address N
jumpr rX	Set program counter to address in rX
jeqzn rX N	If rX == 0, then jump to line N
jnezn rX N	If rX != 0, then jump to line N
jgtzn rX N	If rX > 0, then jump to line N
jltzn rX N	If rX < 0, then jump to line N
calln rX N	Copy addr. of next instr. into rX and then jump to mem. addr. N
Interacting with memory (RAM)	
pushr rX rY	Store contents of register rX onto stack pointed to by reg. rY
popr rX rY	Load contents of register rX from stack pointed to by reg. rY
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
storer rX rY	Store contents of register rX into memory address held in reg. rY

Hmmm

four instructions that make functions possible

calln

(call)

jumpr

(return)

(to the stack)

pushr

popr

(from the stack)

We must go deeper...

```
a = int(input("Num: "))
r = fact(a)
print(r)


def fact(n):
    if n == 0:
        return 1
    else:
        r = fact(n-1)
        r = r * n
        return r
```

We must go deeper...

```python
a = int(input("Num: "))
r = fact(a)
print(r)


def fact(n):
    if n == 0:
        return 1
    else:
        r = fact(n-1)
        r = r * n
        return r
```

```
00   nop
01   read r1
02   calln r14 10
03   write r13
04   halt
…
10   jnezn r1 13
11   setn r13 1
12   jumpr r14
13   nop
14   nop
15   addn r1 -1
16   calln r14 10
17   nop
18   nop
19   mul r13 r1 r13
20   jumpr r14
```

We must go deeper...

```python
a = int(input("Num: "))
r = fact(a)
print(r)


def fact(n):
    if n == 0:
        return 1
    else:
        r = fact(n-1)
        r = r * n
        return r
```

```
00   setn r15 42
01   read r1
02   calln r14 10
03   write r13
04   halt
…
10   jnezn r1 13
11   setn r13 1
12   jumpr r14
13   pushr r14 r15
14   pushr r1 r15
15   addn r1 -1
16   calln r14 10
17   popr r1 r15
18   popr r14 r15
19   mul r13 r1 r13
20   jumpr r14
```