Software

Python

How does Python **function** ?

Hmmm

4 Hmmm problems
+ 1 loop problem
due **next Monday**

Read Sections
4.5-4.5.3

RAM

registers

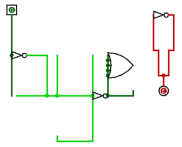
1-bit memory: flip-flops

arithmetic

bitwise functions

logic gates

transistors / switches



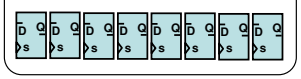
Hardware

CS 5 this week

Some memory is more equal than others...

Registers
on the Central Processing Unit

8 flip-flops are an 8-bit **register**



100 Registers of 64 bits each

~ 10,000 bits

Main Memory
(replaceable RAM)

10 GB memory

~ 100 billion bits

Disk Drive
magnetic storage

4 TB drive

~ 42 trillion bits (or more)

memory from *logic gates*

"Leaky Bucket" capacitors

remagnetizing surfaces

"640K ought to be enough for anybody"

- Bill Gates (*contested*)

Von Neumann Architecture

Processing

CPU

Central Processing Unit ***registers***

Von Neumann bottleneck

Program

RAM

Random-Access Memory locations

r1 General-purpose register, r1

r2 General-purpose register, r2

0	0000 0001 0000 0001
1	1000 0010 0001 0001
2	0110 0010 0010 0001
3	0000 0010 0000 0010
4	0000 0000 0000 0000
5	(all bits)
6	

Programs are stored in memory in *machine language*

Hmmm: Harvey Mudd Miniature Machine

CPU

Central Processing Unit ***registers***

Von Neumann bottleneck

RAM

Random-Access Memory locations

r1 General-purpose register r1

r2

16 registers

0	`read r1`
1	`mul r2 r1`
2	`a`
3	`wr`
4	`halt`

256 memory locations

vs. today?

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`
`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	

Hmmm
the complete reference

At www.cs.hmc.edu/~cs5grad/cs5/hmmm/documentation/documentation.html

Today

Thurs.

Assembly Language

ought to be called ***register*** language

```
read  r1          reads from keyboard into reg1
write r2          outputs reg2 onto the screen

setn  r1  42      reg1 = 42          you can replace 42 with anything from -128 to 127
addn  r1  -1      reg1 = reg1 - 1    a shortcut

add   r3  r1  r2  reg3 = reg1 + reg2
sub   r3  r1  r2  reg3 = reg1 - reg2
mul   r2  r1  r1  reg2 = reg1 * reg1
div   r1  r1  r2  reg1 = reg1 / reg2   ints only!
mod   r1  r1  r2  reg1 = reg1 % reg2   ints only!
```

This is because they're written R to L in Python!

Loops and **if**s

We ***couldn't*** implement Python using Hmmm so far... It's too linear!

"Straight-line code"

```
0  setn  r1  42
1  write r1
2  addn  r1  1
3  jumpn 1
4  halt
```

CPU
central processing unit

r1 — General-purpose register r1

r2 — not used in this program... — General-purpose register r2

Screen

RAM
random access memory

```
0  setn  r1  42
1  write r1
2  addn  r1  1
3  jumpn 1
4  halt
```

i

jumpn!

*What would happen **IF**...*

- we replace line 3's 1 with a 0?
- we replace line 3's 1 with a 2?
- we replace line 3's 1 with a 3?
- we replace line 3's 1 with a 4?

Jumps in Hmmm

Conditional jumps

`jeqzn r1 42`	IF **r1 == 0** THEN jump to line number **42**
`jgtzn r1 42`	IF **r1 > 0** THEN jump to line number **42**
`jltzn r1 42`	IF **r1 < 0** THEN jump to line number **42**
`jnezn r1 42`	IF **r1 != 0** THEN jump to line number **42**

Unconditional jump

`jumpn 42` Jump to program line # **42**

Indirect jump

`jumpr r1` Jump to the line# ***stored*** in `r1`

`jgtzn`

What Python function is this?

CPU
central processing unit

`r1` — General-purpose register r1

`r2` — General-purpose register r2

Screen: **-6** (input)

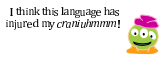
RAM
random access memory

0	`read r1`
1	`jgtzn r1 7`
2	`setn r2 -1`
3	`mul r1 r1 r2`
4	`nop`
5	`nop`
6	`nop`
7	`write r1`
8	`halt`

Space for future expansion!

With an input of **-6**, what does this code write out?

Try it!



1 Follow this Hmmm program.
First run: use **r1 = 42** and **r2 = 5**.
Next run: use **r1 = 5** and **r2 = 42**.

Registers - CPU

	Run 1	Run 2
r1	42	5
r2	5	42
r3		

Output 1 Output 2

Memory - RAM

0	`read r1`
1	`read r2`
2	`sub r3 r1 r2`
3	`nop`
4	`jgtzn r3 7`
5	`write r1`
6	`jumpn 8`
7	`write r2`
8	`halt`

(1) What **common function** does this compute?
Hint: try the inputs in both orders...

(2) ***Extra!*** How could you change only line 3 so that, if inputs **r1** and **r2** are ***equal***, the program will ask for new inputs?

2 Write an assembly-language program that reads a positive integer into **r1**. The program should compute the ***factorial*** of the input in **r2**. Once it's computed, it should write out that factorial. Two lines are provided:

Registers - CPU

r1 (input): 5
r2 (result – so far): 1
r3: not needed; OK to use

Memory - RAM

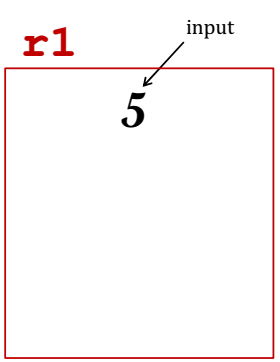
0	`read r1`
1	`setn r2 1`
2	
3	
4	
5	
6	
7	
8	`write r2`
9	`halt`

Hint: On line 2, could you write a test that checks if the factorial is finished? If it's not, compute one piece and then jump back!

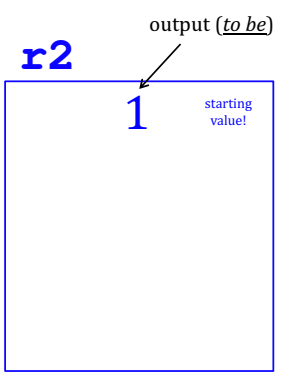
Extra! How few lines can you use here? (Fill the rest with `nop`s...)

factorial: the *plan* ...

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

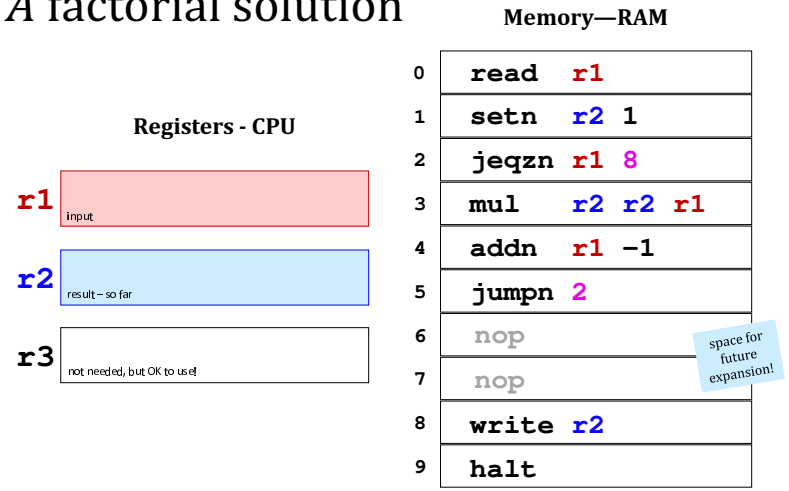


Let r1 be the input and the "counter"



Let r2 ***become*** the output

A factorial solution



Name(s) _____

Quiz

screen

100 (input)

(output)

Fill in what happens at each step

CPU

Central Processing Unit

r1

General-purpose register r1

r2

General-purpose register r2

r3

General-purpose register r3

r4

General-purpose register r4

Hmmm...!?

Extra! Change only line 4's instruction to create an output of **0** or **6** or **349** instead?

RAM

Random-Access Memory

Python

100

```
0  read  r1
1  setn  r2 7
2  mod   r4 r1 r2
3  div   r3 r1 r4
4  sub   r3 r3 r2
5  addn  r3 -1
6  write r3
7  halt
```

r1 = 100