Assembly Language Review

Computer Science 111
Boston University

Vahid Azadeh-Ranjbar, Ph.D.

based in part on notes from the CS-for-All curriculum developed at Harvey Mudd College

Recall: Basic Hmmm Instructions

instruction	what it does	example
`read` rX	reads from keyboard into rX	`read r1`
`write` rX	writes value of rX to screen	`write r2`
`add` rX rY rZ	rX = rY + rZ	`add r1 r1 r3`
`sub` rX rY rZ	rX = rY – rZ	`sub r4 r3 r2`
`mul` rX rY rZ	rX = rY * rZ	`mul r3 r1 r2`
`div` rX rY rZ	rX = rY // rZ	`div r2 r5 r6`
`mod` rX rY rZ	rX = rY % rZ	`mod r2 r1 r3`
`setn` rX n	rX = n	`setn r1 7`
`addn` rX n	rX = rX + n	`addn r1 -1`
`copy` rX rY	rX = rY	`copy r2 r1`

Notation:
rX, rY, rZ is any register (`r1-r15`) n is any integer in [-128, 127]

Recall: Jumps in Hmmm

instruction	what it does	example
`jeqz` rX L	jumps to line L if rX == 0	`jeqz r1 12`
`jgtz` rX L	jumps to line L if rX > 0	`jgtz r2 4`
`jltz` rX L	jumps to line L if rX < 0	`jltz r3 15`
`jnez` rX L	jumps to line L if rX != 0	`jnez r1 7`
`jumpn` L	jumps to line L	`jumpn 6`
`jumpr` rX	jumps to line # stored in rX	`jumpr r2`

Notation:

- rX is any register name (`r1-r15`)
- L is the line number of an instruction

Some Assembly Required!

- Complete the program so that it:
  - reads two numbers from the user
  - outputs 1 if the numbers are equal
  - outputs 0 if they are not equal
- Don't add any `write` statements.
  - just use the one on line 9
- Equivalent Python for what we need to do:

```
if r1 == r2:
    r4 = 1
else:
    r4 = 0
```

0	`read r1`
1	`read r2`
2	
3	
4	
5	
6	
7	
8	
9	`write r4`
10	`halt`

Some Assembly Required!

- Complete the program so that it:
  - reads two numbers from the user
  - outputs 1 if the numbers are equal
  - outputs 0 if they are not equal
- Don't add any **`write`** statements.
  - just use the one on line 9
- Equivalent Python for what we need to do:

```
if r1 == r2:
    r4 = 1
else:
    r4 = 0
```

```
0  read r1
1  read r2
2  sub r3 r1 r2
3  jeqz r3 6
4  setn r4 0
5  jumpn 9
6  setn r4 1
7  nop
8  nop
9  write r4
10 halt
```

Recall: Factorial Using a Loop

Screen: 5 (input)

CPU ⟷ RAM

central processing unit

r1	r2
5	1
	5
4	

random access memory

```
0  read r1
1  setn r2 1
2  jeqz r1 8
3  mul r2 r2 r1
4  addn r1 -1
5  jumpn 2
6  nop
7  nop
8  write r2
9  halt
```

Recall: Factorial Using a Loop

Screen

5 (input)

120 (output)

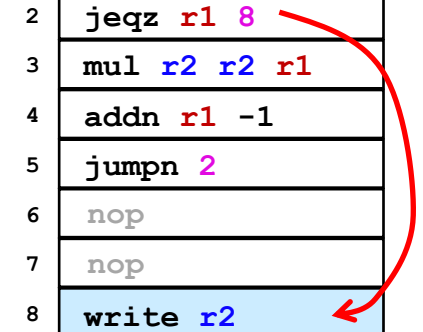
CPU ⟷ RAM

central processing unit

random access memory

r1	r2
5	1
	5
4	20
3	60
2	120
1	120
0	

0	`read r1`
1	`setn r2 1`
2	`jeqz r1 8`
3	`mul r2 r2 r1`
4	`addn r1 -1`
5	`jumpn 2`
6	`nop`
7	`nop`
8	`write r2`
9	`halt`



For the Input At Right, What Is the Output of this Program?

Screen

8 (input)

A. -1

B. 0

C. 1

D. 2

E. 3

Use a table!

r1	r2	r3	r4	r5

0	`read r1`
1	`jeqz r1 11`
2	`jltz r1 11`
3	`setn r2 0`
4	`setn r3 1`
5	`setn r4 2`
6	`sub r5 r1 r3`
7	`jeqz r5 12`
8	`div r1 r1 r4`
9	`addn r2 1`
10	`jumpn 6`
11	`setn r2 -1`
12	`write r2`
13	`halt`

- Which lines make up the loop?
- When would the output be -1?

For the Input At Right, What Is the Output of this Program?

Screen

8 (input)

A. -1
B. 0
C. 1
D. 2
E. 3

- Which lines make up the loop?
- When would the output be -1?

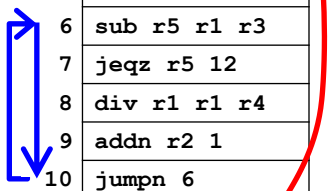
```
0  read r1
1  jeqz r1 11
2  jltz r1 11
3  setn r2 0
4  setn r3 1
5  setn r4 2
6  sub r5 r1 r3
7  jeqz r5 12
8  div r1 r1 r4
9  addn r2 1
10 jumpn 6
11 setn r2 -1
12 write r2
13 halt
```

For the Input At Right, What Is the Output of this Program?

Screen

0 (input)
-1 (output)

r1	r2	r3	r4	r5
8	0	1	2	7
4	1			3
2	2			1
1	3			0



```
0  read r1
1  jeqz r1 11
2  jltz r1 11
3  setn r2 0
4  setn r3 1
5  setn r4 2
6  sub r5 r1 r3
7  jeqz r5 12
8  div r1 r1 r4
9  addn r2 1
10 jumpn 6
11 setn r2 -1
12 write r2
13 halt
```

- Which lines make up the loop?
  6-10
- When would the output be -1?
  when input <= 0

Recall: Functions in Assembly

```
0  read r1
1  call r14 4
2  write r13
3  halt
4  copy r13 r1
5  addn r1 -1
6  mul r13 r13 r1
7  jumpr r14
```

input

function call

output

the function

"*return*"

For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)

A. 2
B. 5
C. 6
D. 8
E. none of these

- Which lines make up the function?
- How many times is it called?
- What does the function do?
- What does the program do?

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```

For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)

A. 2
B. 5
C. 6
D. 8
E. none of these

- Which lines make up the function?
- How many times is it called?
- What does the function do?
- What does the program do?

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```

For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```

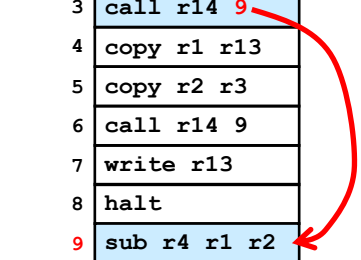
For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```



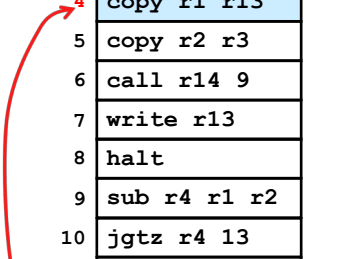
For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4
			6	2	

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```



For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4
			6	2	
2	5				

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```

prepare inputs for the 2nd call!

For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4
			6	2	
2	5				7

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```

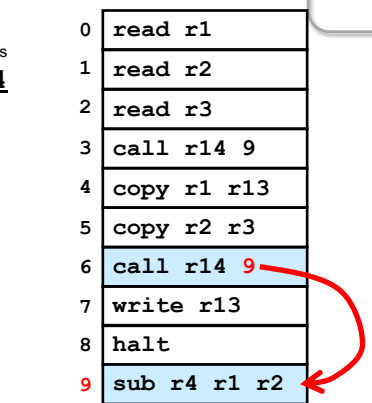
For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4
			6	2	
2	5				7

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```



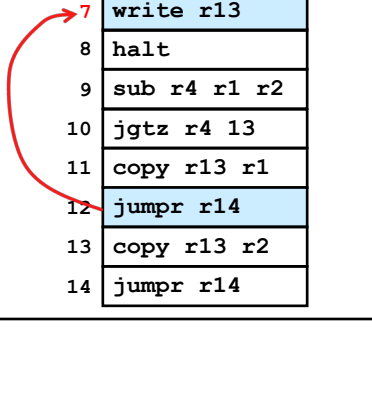
For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4
			6	2	
2	5				7
			-3	2	

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```



For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)
2 (output)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4
			6	2	
2	5				7
			-3	2	

- Which lines make up the function?
- How many times is it called?

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```

For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)
2 (output)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4
			6	2	
2	5				7
			-3	2	

- Which lines make up the function?
  lines 9-14
- How many times is it called?
  2

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpn ???
13 copy r13 r2
14 jumpn ???
```

unconditional jumps would *not* work!

For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)
2 (output)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4
			6	2	
2	5				7
			-3	2	

- *What does the function do?*

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```

For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)
2 (output)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4
			6	2	
2	5				7
			-3	2	

- *What does the function do?*
  finds the min of r1 and r2

  it is equivalent to:

```
if r1 > r2:
    return r2
else:
    return r1
```

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```


For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)
2 (output)

r1	r2	r3	r4	return value r13	return address r14
8	2	5			4
			6	2	
2	5				7
			-3	2	

- What does the function do?
  finds the min of r1 and r2
- *What does the program do?*


```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```

For the Inputs At Right, What Is the Output of this Program?

Screen

8 (1st input)
2 (2nd input)
5 (3rd input)
2 (output)

r1	r2	r3	r4	r13 (return value)	r14 (return address)
8	2	5			4
			6	2	
2	5				7
			-3	2	

```
0  read r1
1  read r2
2  read r3
3  call r14 9
4  copy r1 r13
5  copy r2 r3
6  call r14 9
7  write r13
8  halt
9  sub r4 r1 r2
10 jgtz r4 13
11 copy r13 r1
12 jumpr r14
13 copy r13 r2
14 jumpr r14
```

prepare inputs for the 2nd call!

- What does the function do?
  finds the min of r1 and r2
- *What does the program do?*
  finds the min of the user's 3 inputs
  - first call computes min(input 1, input 2)
  - second call computes min(first call's result, input 3)

Recall Our Earlier Example

```
def foo(x):
    y = x*(x-1)
    return y

x = int(input())
y = foo(x)
print(y)
```

```
0  read r1
1  call r14 4
2  write r13
3  halt
4  copy r13 r1
5  addn r1 -1
6  mul r13 r13 r1
7  jumpr r14
```

the function

Recall: The Need for the Stack

```
def foo(x):
    y = x*(x-1)
    return y

x = int(input())
y = foo(x)
y = y + x
print(y)
```

```
0   read r1
1   call r14 4
2   write r13
3   halt
4   copy r13 r1
5   addn r1 -1
```

What if we add this line?

We want to add the user's input **x** to the function's return value **y** before printing **y**.

Recall: The Need for the Stack

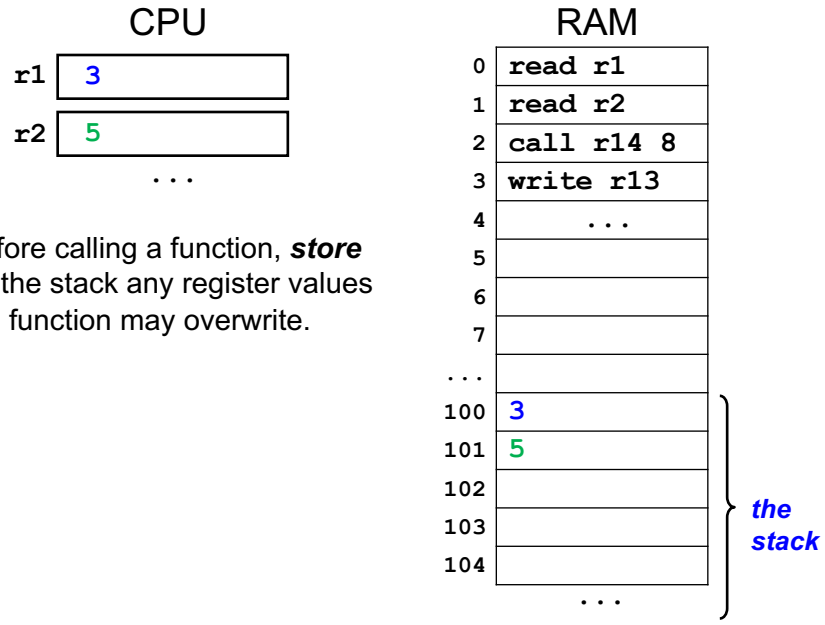
```
def foo(x):
    y = x*(x-1)
    return y

x = int(input())
y = foo(x)
y = y + x
print(y)
```

```
0   read r1
1   call r14 4
1.5 add r13 r13 r1
2   write r13
3   halt
4   copy r13 r1
5   addn r1 -1
6   mul r13 r13 r1
7   jumpr r14
```

doesn't work!

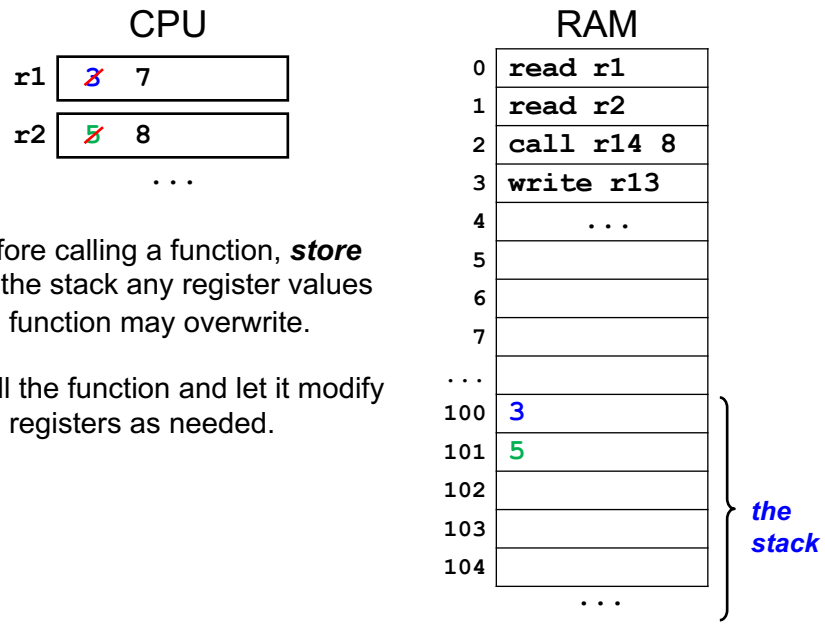
the user's input gets changed

Recall: Storing Things on the Stack



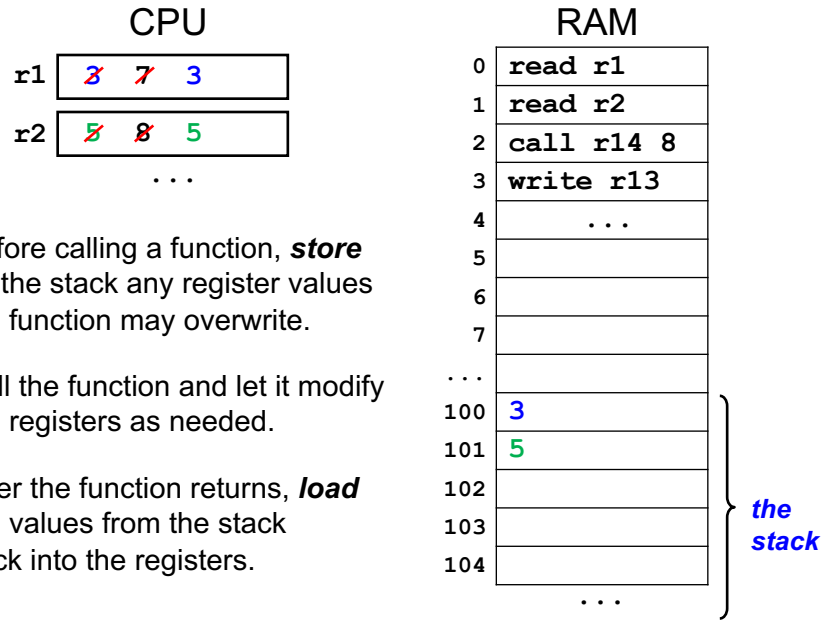
- Before calling a function, ***store*** on the stack any register values the function may overwrite.

Recall: Storing Things on the Stack



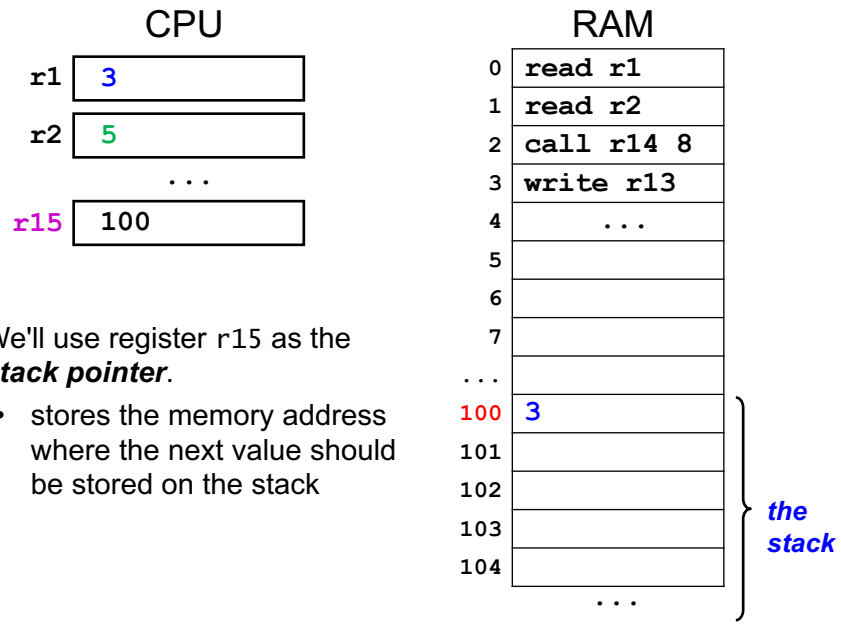
- Before calling a function, ***store*** on the stack any register values the function may overwrite.
- Call the function and let it modify the registers as needed.

Recall: Storing Things on the Stack



- Before calling a function, ***store*** on the stack any register values the function may overwrite.
- Call the function and let it modify the registers as needed.
- After the function returns, ***load*** the values from the stack back into the registers.

The Stack Pointer



- We'll use register `r15` as the ***stack pointer***.
  - stores the memory address where the next value should be stored on the stack

Using the Stack: `storer` and `loadr`

```
0  read r1
1  setn r15 100
2  storer r1 r15
3  call r14 8
4  loadr r1 r15
5  add r13 r13 r1
6  write r13
7  halt
8  copy r13 r1
9  addn r1 -1
10 mul r13 r13 r1
11 jumpr r14
```

```
# initialize the stack pointer
# store r1's value on the stack
  (in the memory address found
   in r15 – not in r15 itself)
# load back r1's value from the
  stack (from the memory address
  found in r15 – not from r15 itself)
```

Tracing Our Fixed Program

Screen: **5** (input)

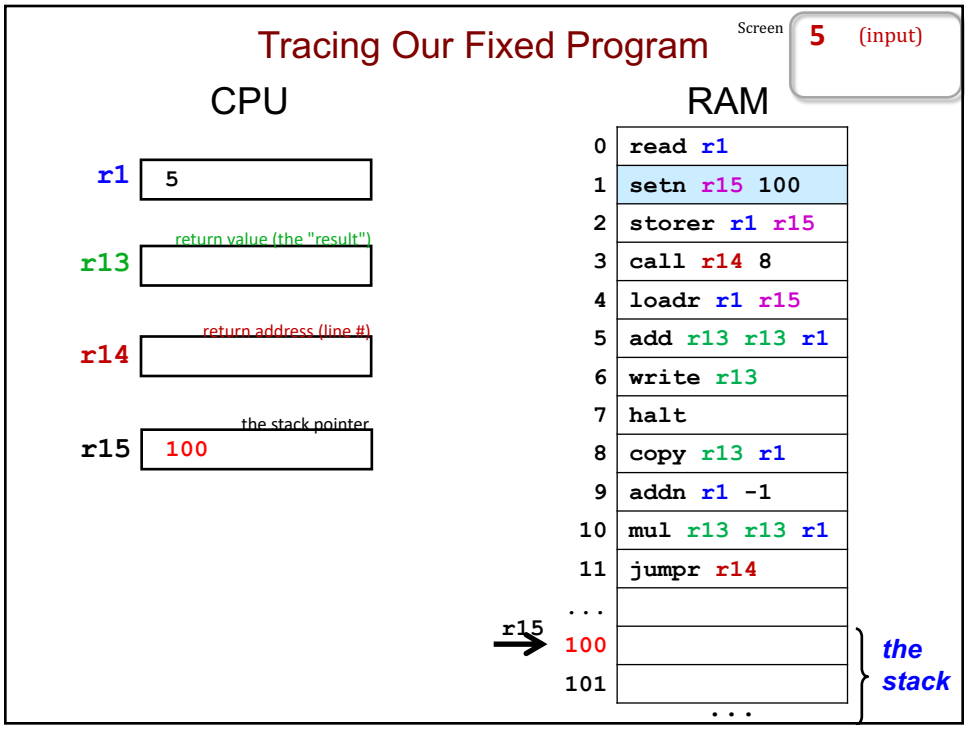
CPU

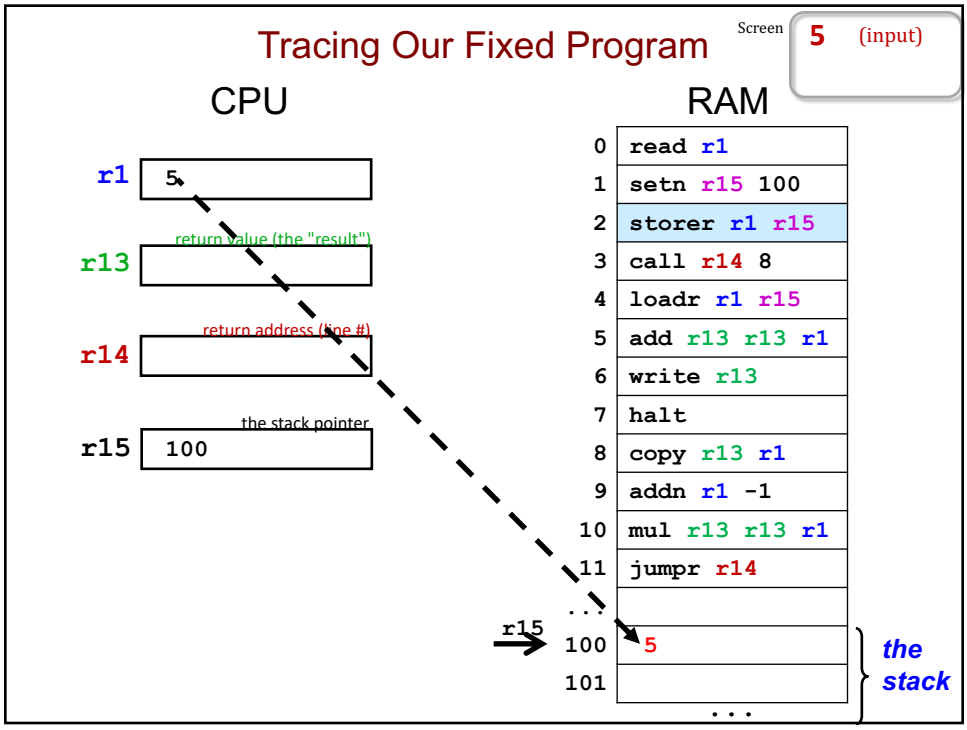
- r1
- r13 — return value (the "result")
- r14 — return address (line #)
- r15 — the stack pointer

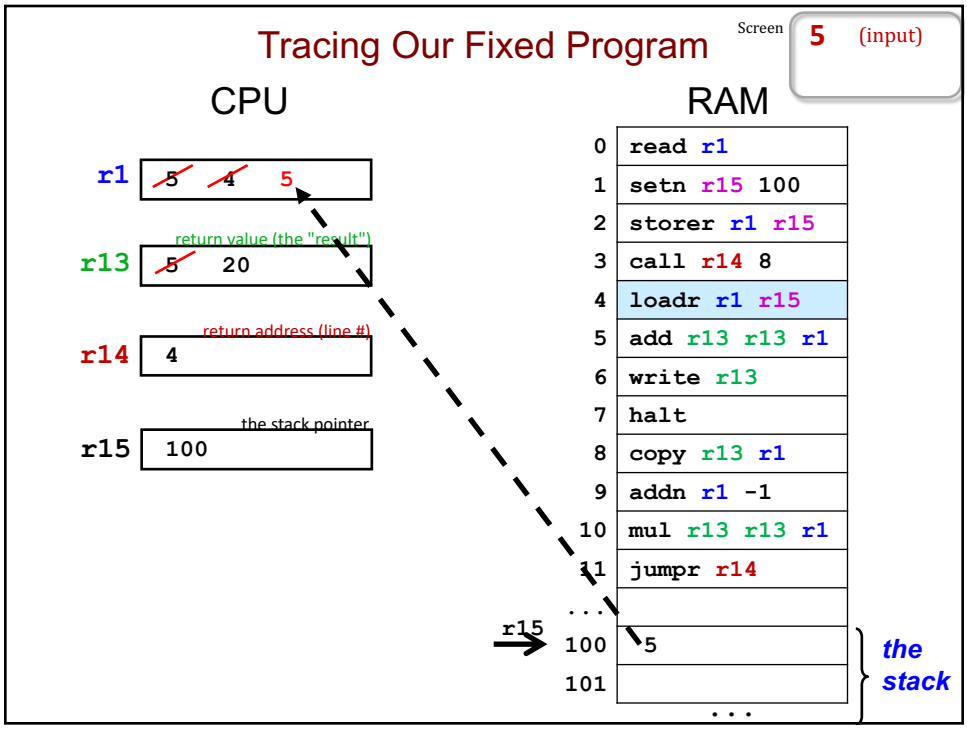
RAM

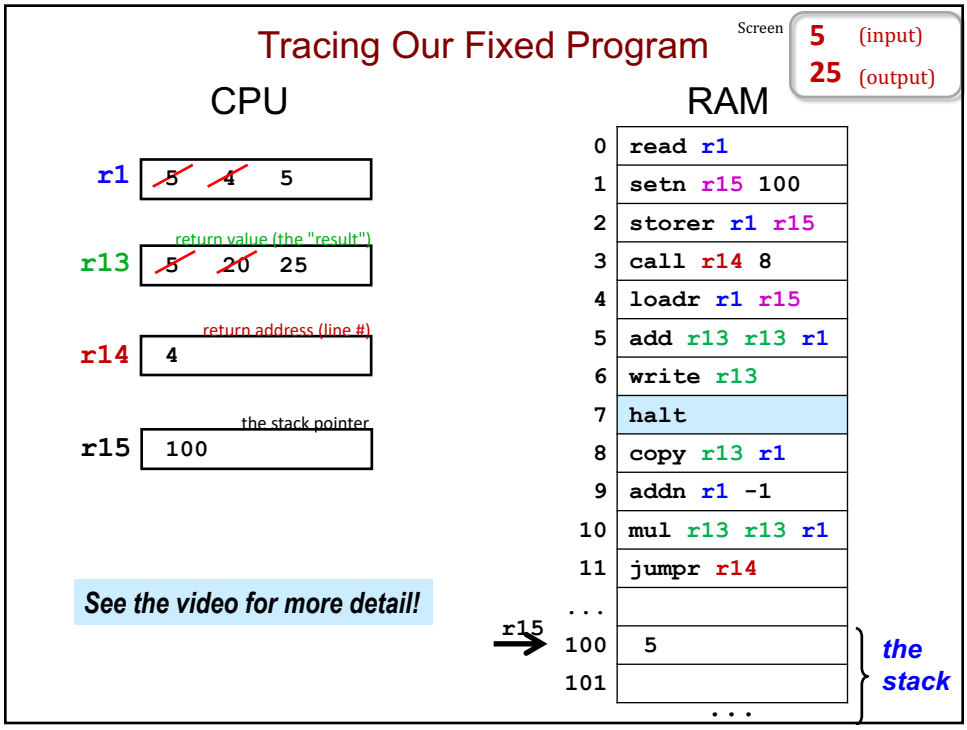
0	read r1
1	setn r15 100
2	storer r1 r15
3	call r14 8
4	loadr r1 r15
5	add r13 r13 r1
6	write r13
7	halt
8	copy r13 r1
9	addn r1 -1
10	mul r13 r13 r1
11	jumpr r14
...	
100	
101	
...	

the stack


Tracing Our Fixed Program
Screen
5 (input)
CPU
RAM
r1
5
return value (the "result")
r13
return address (line #)
r14
the stack pointer
r15
100
0 read r1
1 setn r15 100
2 storer r1 r15
3 call r14 8
4 loadr r1 r15
5 add r13 r13 r1
6 write r13
7 halt
8 copy r13 r1
9 addn r1 -1
10 mul r13 r13 r1
11 jumpr r14
...
r15
100
101
...
the stack


Tracing Our Fixed Program
Screen
5 (input)
CPU
RAM
r1
5
return value (the "result")
r13
return address (line #)
r14
the stack pointer
r15
100
0 read r1
1 setn r15 100
2 storer r1 r15
3 call r14 8
4 loadr r1 r15
5 add r13 r13 r1
6 write r13
7 halt
8 copy r13 r1
9 addn r1 -1
10 mul r13 r13 r1
11 jumpr r14
...
r15
100
5
101
...
the stack


Tracing Our Fixed Program
Screen
5 (input)
CPU
RAM
r1
5 4 5
return value (the "result")
r13
5 20
return address (line #)
r14
4
the stack pointer
r15
100
0 read r1
1 setn r15 100
2 storer r1 r15
3 call r14 8
4 loadr r1 r15
5 add r13 r13 r1
6 write r13
7 halt
8 copy r13 r1
9 addn r1 -1
10 mul r13 r13 r1
11 jumpr r14
...
r15
100 5
101
...
the stack


Tracing Our Fixed Program
Screen
5 (input)
25 (output)
CPU
RAM
r1
5 4 5
return value (the "result")
r13
5 20 25
return address (line #)
r14
4
the stack pointer
r15
100
0 read r1
1 setn r15 100
2 storer r1 r15
3 call r14 8
4 loadr r1 r15
5 add r13 r13 r1
6 write r13
7 halt
8 copy r13 r1
9 addn r1 -1
10 mul r13 r13 r1
11 jumpr r14
...
r15
100 5
101
...
the stack
See the video for more detail!