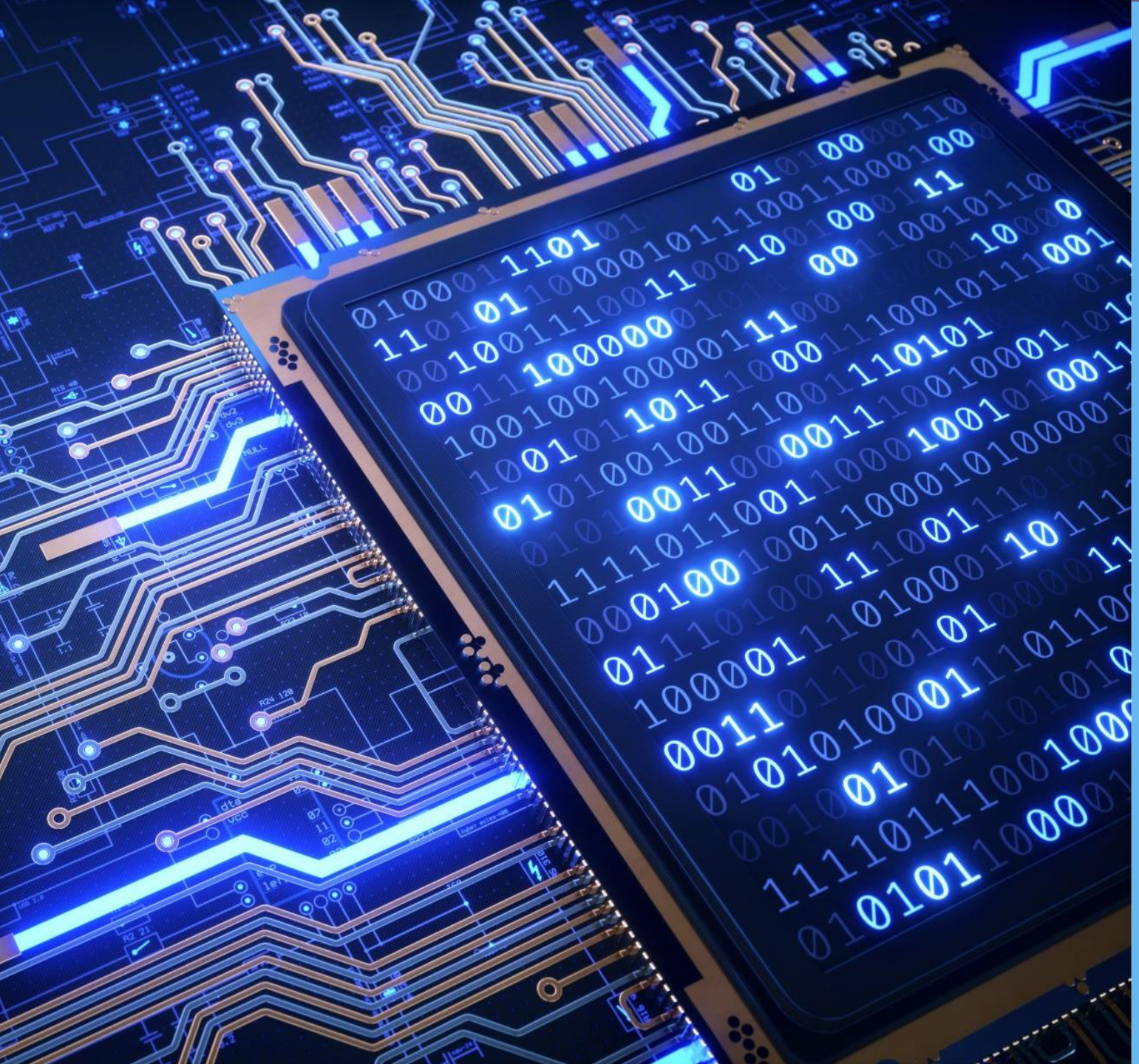


Computer Systems

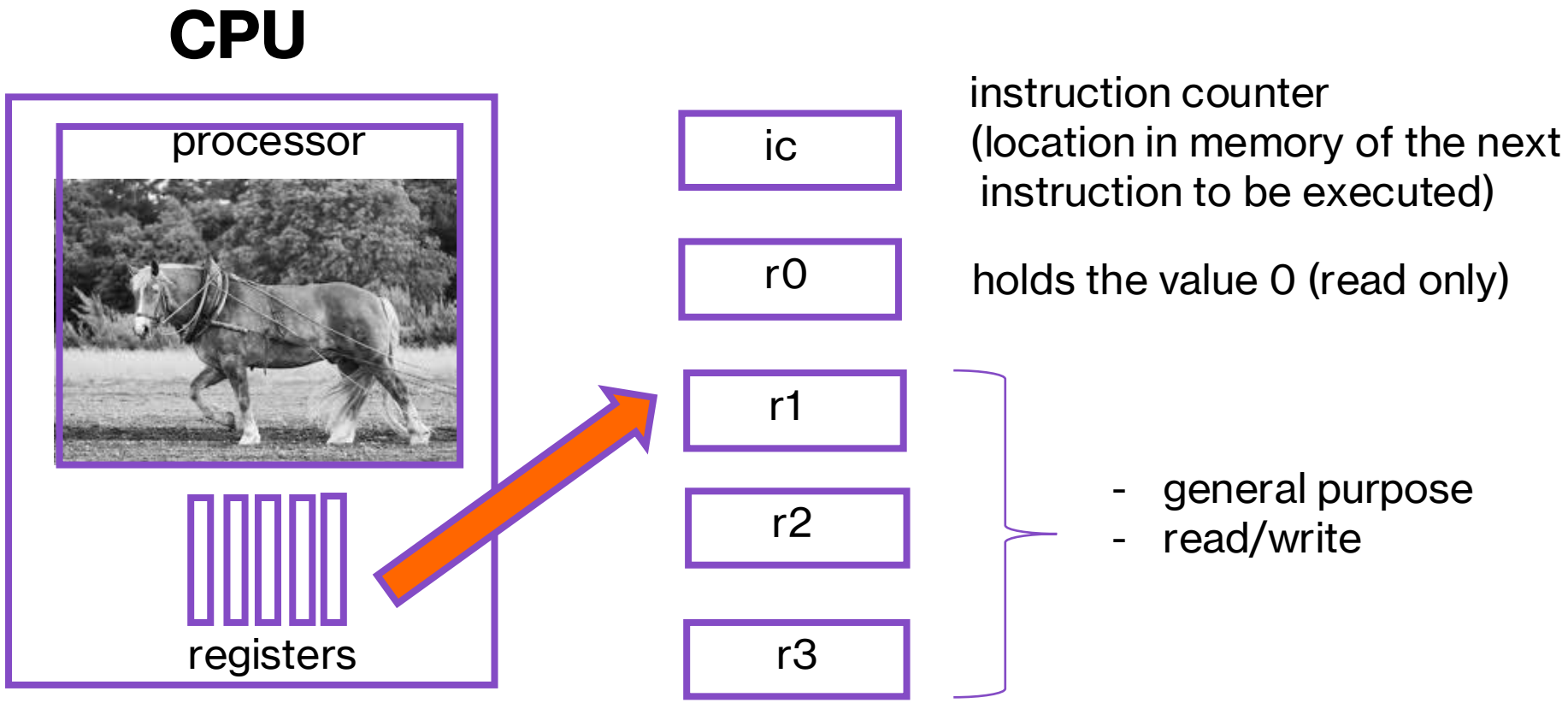
CS51 machine – calling functions

CS51 – Fall 2026

Examples from this lecture

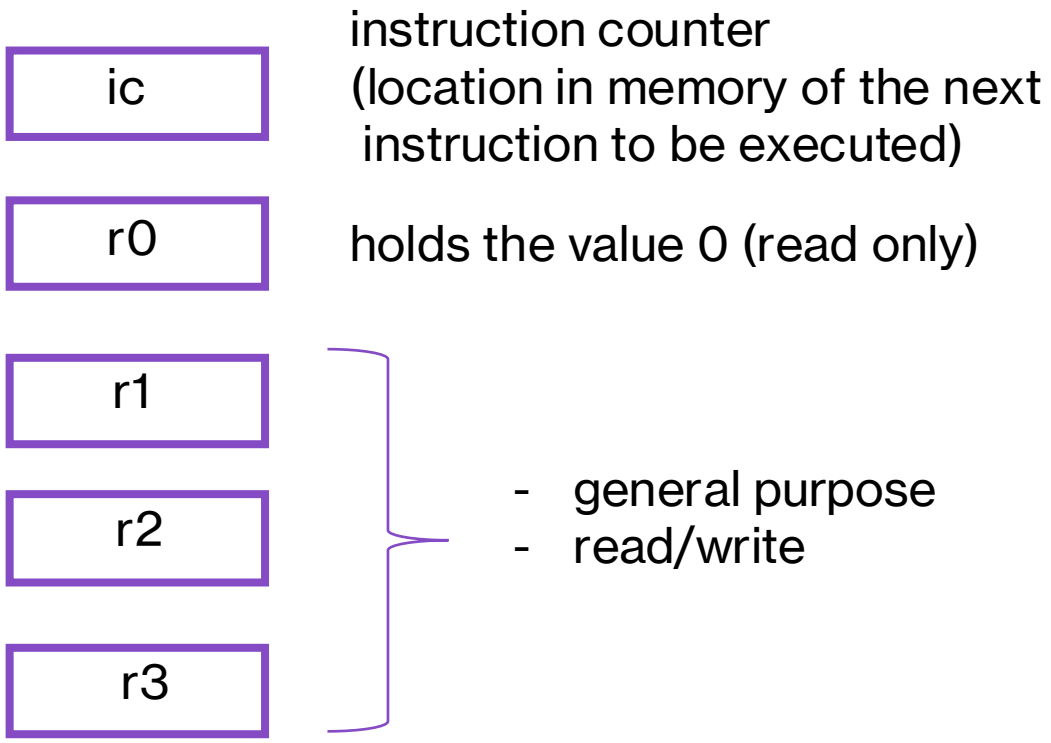
http://www.cs.pomona.edu/classes/cs51/cs51machine/

CS51 machine (processor)



Program execution in the CS51 Machine

- When executing a program, the CS51 machine loops over the follow:
  - Fetch the value from $\mathrm{mem}[\mathrm{ic}]$ for use as an instruction
  - Increment ic by 2
  - Decode the instruction and then execute it



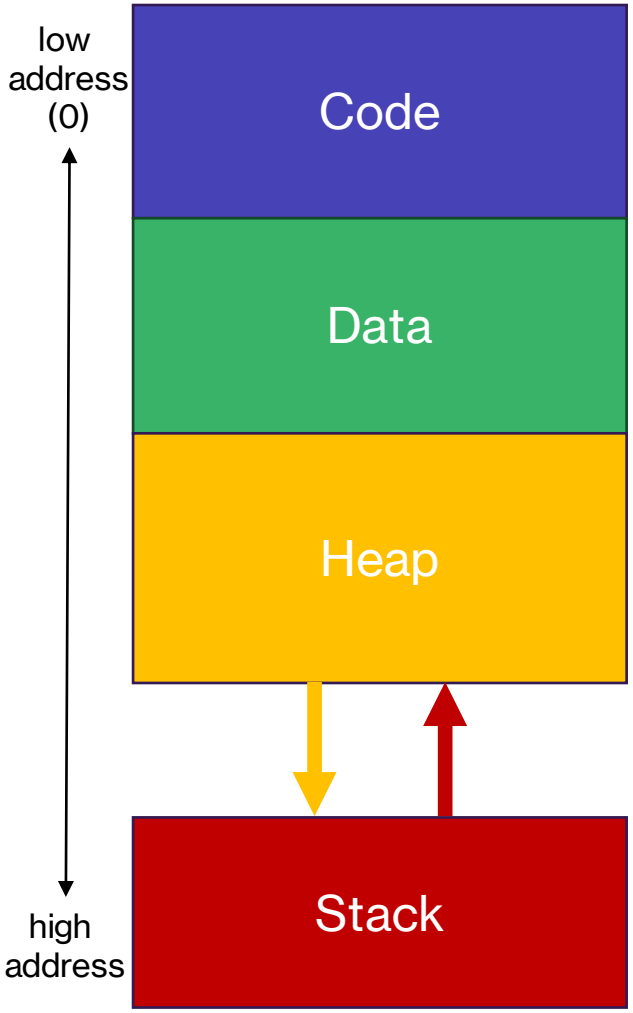
Basic structure of CS51 program

```
; great comments at the top!
;
        instruction1            ; comment
        instruction2            ; comment
        ...
label1
        instruction3            ; comment
        instruction4            ; comment
label2
        ...
        hlt
```



- whitespace before operations/instructions
- labels go here

The organization of memory



- Memory in modern computers is organized in four basic areas:
  - **Code**: Where the instructions of our program are stored.
  - **Data**: Where the global data are stored and live for the entire duration of our program. Think of global data as variables that are created outside a function. The size of this memory region is fixed.
  - **Heap**: Where dynamic user-defined objects, lists, dictionaries, etc are created. The size is automatically adjusted in Python.
    - We will see such data structures later in this course and you will also see how to work with objects (in Java) in CS62.
  - **Stack**: Devoted to managing subprogram calls and stores their local variables and return address. Depending on the language, subprograms are also called *functions*, *methods,* or *procedures*. In Python, subprograms come in the form of functions (and methods).



The stack abstract data type

- A stack is a data structure (a way of organizing and storing data) that maintains a collection of elements.
- A stack is like a stack of dishes or cafeteria trays.
  - Dishes may be placed on the top of the stack or removed from the top of the stack, not the middle.
- If a stack is not empty, we may **pop** the top element.
- To add an element to the stack, we **push** it to the top.
- The element popped is the one most recently pushed. This is known as last-in-first-out or **LIFO** principle.

Think

Two operations

- `psh R`: push the value of the register R to the top of the stack
- `pop R`: pop the value from the top of the stack and put it in the register R

```
add r3 r0 8
psh r3
add r3 r0 0
pop r3
sto r3 r0
```

What does this program do?

Think

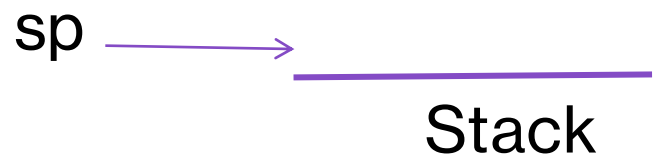
r3 8

Two operations

- `psh R`: push the value of the register R to the top of the stack

- `pop R`: pop the value from the top of the stack and put it in the register R

```
add r3 r0 8
psh r3
add r3 r0 0
pop r3
sto r3 r0
```



Think

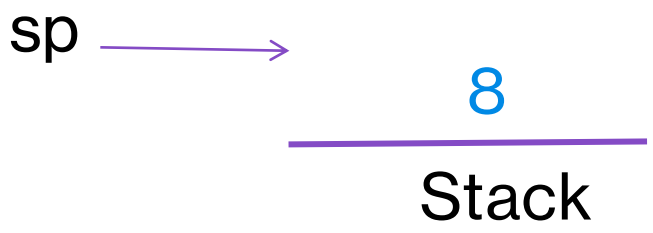
r3	8

Two operations

- `psh R`: push the value of the register R to the top of the stack

- `pop R`: pop the value from the top of the stack and put it in the register R

```
add r3 r0 8
psh r3
add r3 r0 0
pop r3
sto r3 r0
```



Think

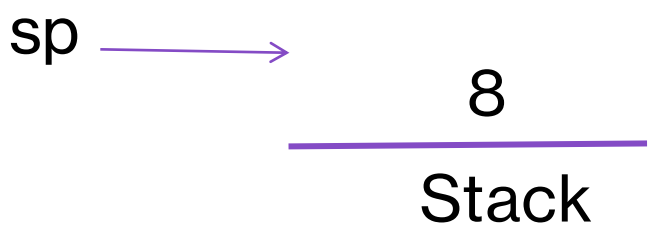
r3	0

Two operations

- `psh R`: push the value of the register R to the top of the stack

- `pop R`: pop the value from the top of the stack and put it in the register R

```
add r3 r0 8
psh r3
add r3 r0 0
pop r3
sto r3 r0
```



Think

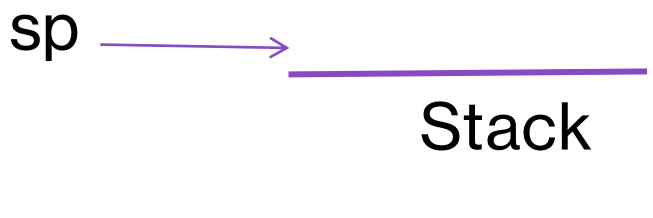
r3 8

Two operations

- `psh R`: push the value of the register R to the top of the stack

- `pop R`: pop the value from the top of the stack and put it in the register R

```
add r3 r0 8
psh r3
add r3 r0 0
pop r3
sto r3 r0
```



Think

r3	8

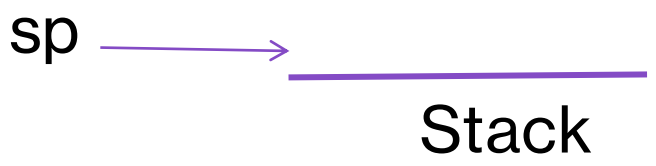
Two operations

- `psh R`: push the value of the register R to the top of the stack

- `pop R`: pop the value from the top of the stack and put it in the register R

```
add r3 r0 8
psh r3
add r3 r0 0
pop r3
sto r3 r0
```

prints 8



Stack frame

- When our program calls a function, the system reserves memory space for the function needs in a **stack frame**.
- The newly-created stack frame is pushed onto the top of the **call stack** and becomes the active stack frame.
- Stack frames keep track of the:
  - return address (where to go when we’re done executing)
  - parameters
  - local variables

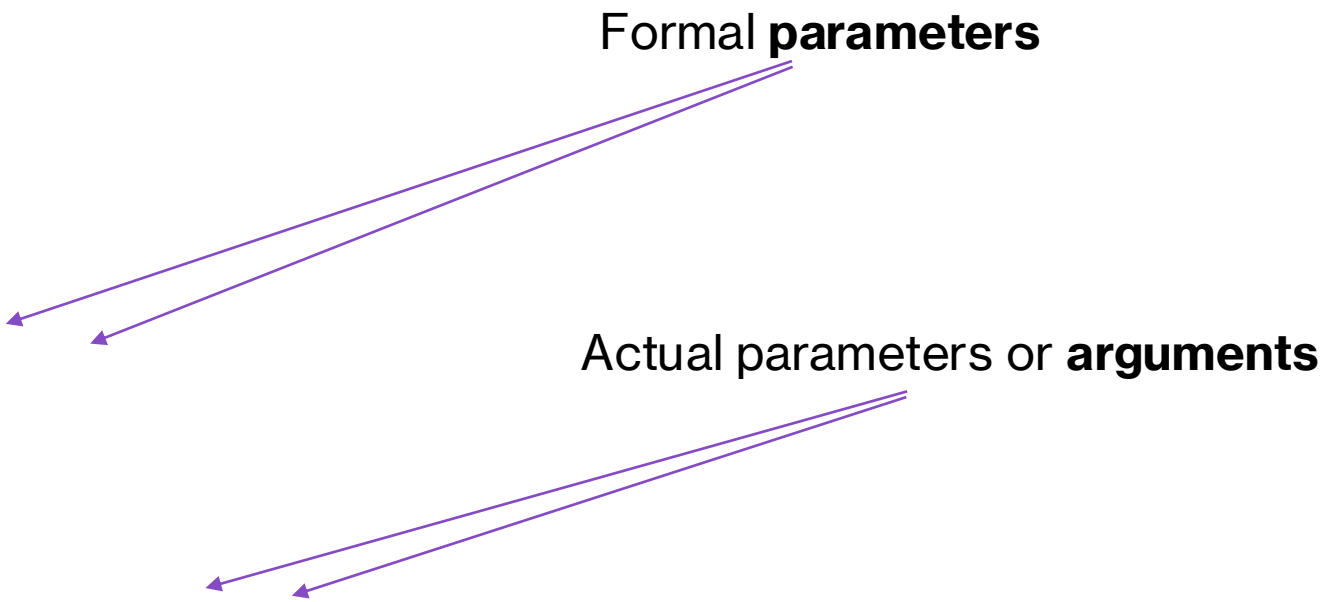
Look at the following Python program

```
def add(num1, num2):
    return num1 + num2

def double(num):
    return 2 * num

def add_then_double(a, b):
    added = add(a, b)
    return double(added)

>> add_then_double(-10, 4)
```



The call stack

```
def add(num1, num2):
    return num1 + num2

def double(num):
    return 2 * num

def add_then_double(a, b):
    added = add(a, b)
    return double(added)

>> add_then_double(-10, 4)
```

When calling a function, a stack frames gets created and pushed into the call stack. The stack frame keep track of:

- return address (where to go when we're done executing)
- parameters
- local variables



The call stack

```
def add(num1, num2):
    return num1 + num2

def double(num):
    return 2 * num

def add_then_double(a, b):
    added = add(a, b)
    return double(added)

>> add_then_double(-10, 4)
```



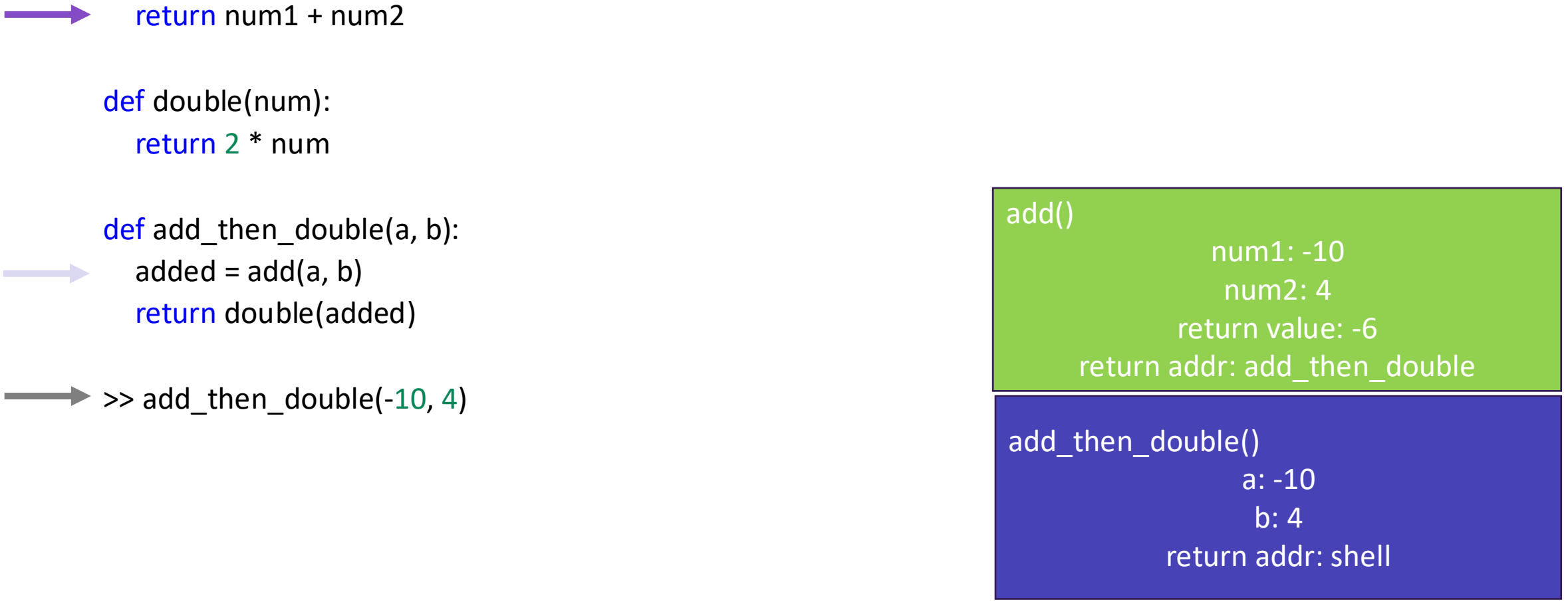
The call stack

```
def add(num1, num2):
    return num1 + num2

def double(num):
    return 2 * num

def add_then_double(a, b):
    added = add(a, b)
    return double(added)

>> add_then_double(-10, 4)
```



The call stack

```
def add(num1, num2):
    return num1 + num2

def double(num):
    return 2 * num

def add_then_double(a, b):
    added = add(a, b)
    return double(added)

>> add_then_double(-10, 4)
```





```
add_then_double()
        a: -10
        b: 4
    added: -6
return addr: shell
```

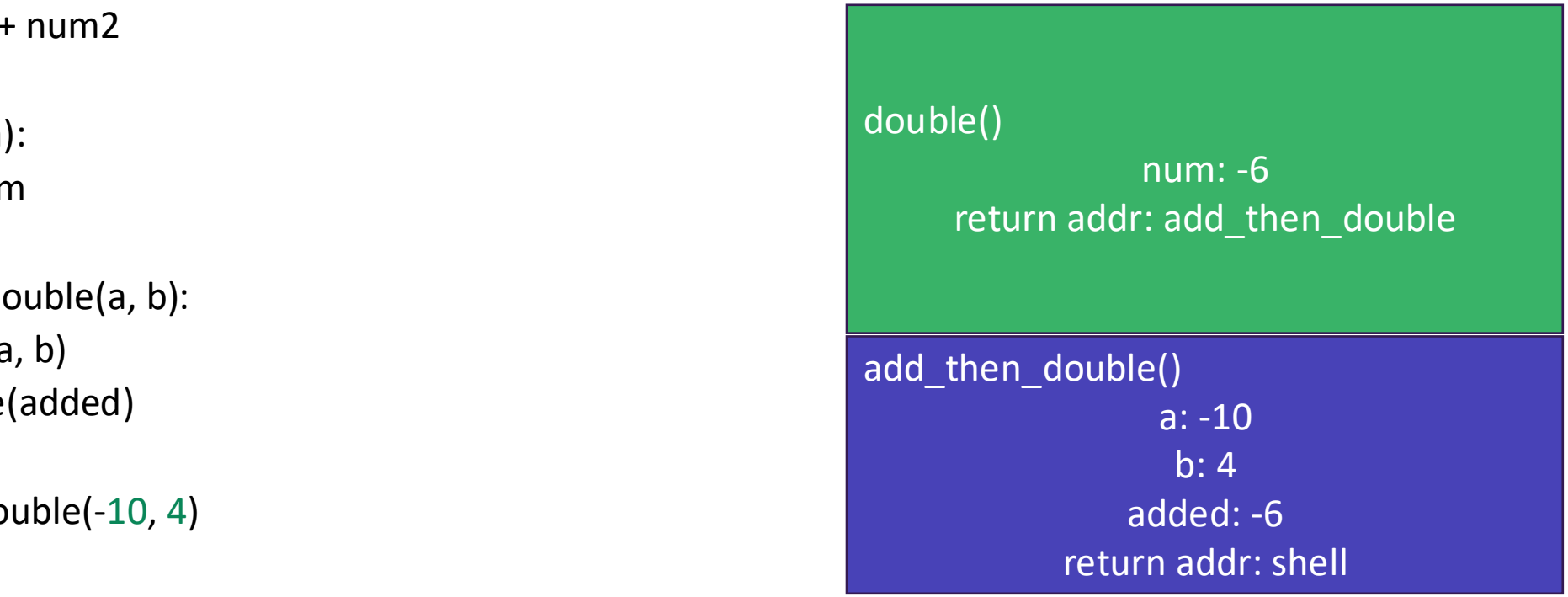
The call stack

```
def add(num1, num2):
    return num1 + num2

def double(num):
    return 2 * num

def add_then_double(a, b):
    added = add(a, b)
    return double(added)

>> add_then_double(-10, 4)
```



The call stack

```
def add(num1, num2):
    return num1 + num2

def double(num):
    return 2 * num

def add_then_double(a, b):
    added = add(a, b)
    return double(added)

>> add_then_double(-10, 4)
```



The call stack

```
def add(num1, num2):
    return num1 + num2

def double(num):
    return 2 * num

def add_then_double(a, b):
    added = add(a, b)
    return double(added)

>> add_then_double(-10, 4)
```

add_then_double()
a: -10
b: 4
added: -6
return value: -12
return addr: shell

The call stack

```
def add(num1, num2):
    return num1 + num2

def double(num):
    return 2 * num

def add_then_double(a, b):
    added = add(a, b)
    return double(added)

>> add_then_double(-10, 4)
```

"-12" is returned

Function calls in assembly

For high-level languages, the call stack is managed for you

In assembly, **we will manage the stack!**



CS51 machine call stack conventions

r1 is reserved for the stack pointer, pointing to the top of the stack address

r2 contains the return address (a memory address in the code portion of where we should come back to when the function is done)

r3 contains the first parameter for the function in the active call stack frame

additional parameters go on the stack (more on this in a bit)

the result after the function completes its work should go in **r3**

Structure of a *single* parameter function

```
function_name
        psh r2                  ; save return address on stack
        ...                     ; do work using r3 as argument
                                ; put result in r3
        pop r2                  ; restore return address from stack
        jmp r2                  ; return to caller
```

`jmp` "jumps" to the line of code at r2. Really: sets ic = r2

- r2 has the return address
- argument is in r3
- r1 is off-limits since it's used for the stack pointer
- return value goes in r3

Our first function call

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2


14   sto r3 r0          ; write result,
16   hlt                ;    and halt

increment
18    psh r2             ; save the return address on the stack
22    add r3 r3 1        ; add 1 to the input parameter
24    pop r2             ; get the return address from stack
28    jmp r2             ; go back to where we were called from
```

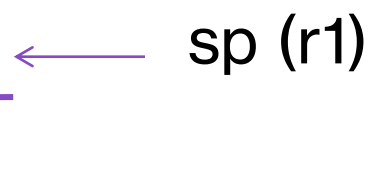
Our first function call

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2


14   sto r3 r0          ; write result,
16   hlt                ;    and halt


increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	6
r1	130
r2	0
r3	0



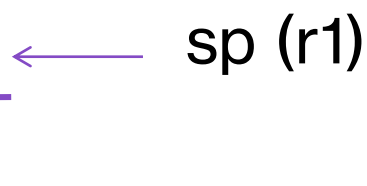
Our first function call

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2


14   sto r3 r0          ; write result,
16   hlt                ;    and halt


increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	6
r1	130
r2	0
r3	10



Our first function call

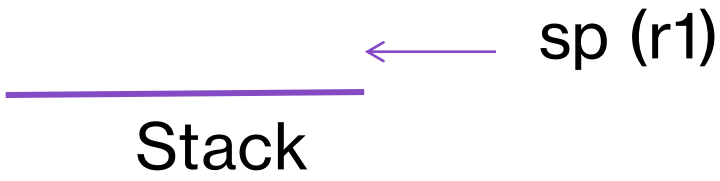
`lcw`: put the memory address of the label into the register r2

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2


14   sto r3 r0          ; write result,
16   hlt                ;    and halt


increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	8
r1	130
r2	0
r3	10



Our first function call

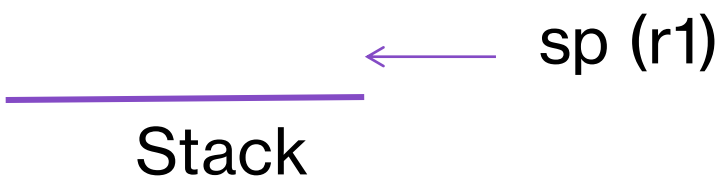
`lcw`: put the memory address of the label into the register r2

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2

14   sto r3 r0          ; write result,
16   hlt                ;    and halt

increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	8
r1	130
r2	18
r3	10



Our first function call

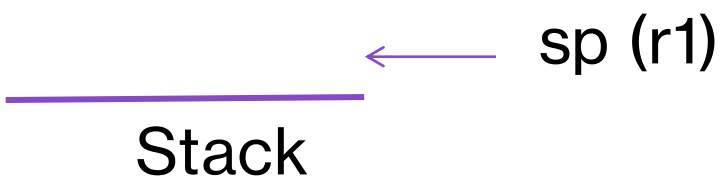
`cal r2 r2`: call a function
`rY`: set ic to address in rY
`rX`: save the current ic to rX, i.e. next instruction that would have been executed

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2

14   sto r3 r0          ; write result,
16   hlt                ;    and halt

increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	12
r1	130
r2	18
r3	10



Our first function call

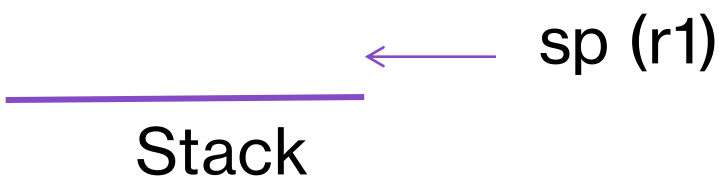
`cal r2 r2`: call a function
`rY`: set ic to address in rY
`rX`: save the current ic to rX, i.e. next instruction that would have been executed

```
6   loa r3 r0          ; get input from user for input parameter
8   lcw r2 increment   ; call increment
12  cal r2 r2


14  sto r3 r0          ; write result,
16  hlt                ;    and halt


increment
18   psh r2            ; save the return address on the stack
22   add r3 r3 1       ; add 1 to the input parameter
24   pop r2            ; get the return address from stack
28   jmp r2            ; go back to where we were called from
```

ic	18
r1	130
r2	14
r3	10



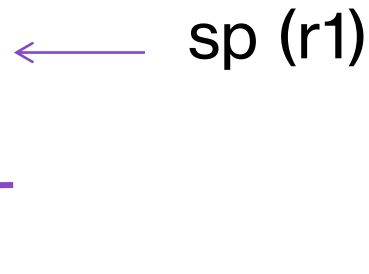
Our first function call

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2


14   sto r3 r0          ; write result,
16   hlt                ;    and halt

increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	18
r1	128
r2	14
r3	10



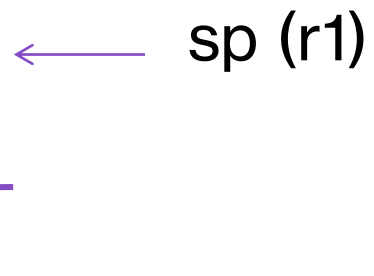
Our first function call

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2


14   sto r3 r0          ; write result,
16   hlt                ;    and halt

increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	22
r1	128
r2	14
r3	11



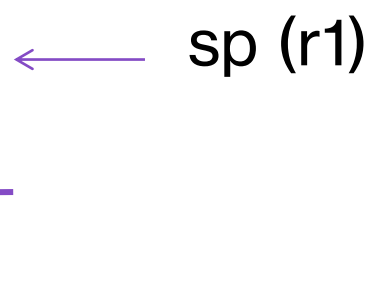
Our first function call

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2


14   sto r3 r0          ; write result,
16   hlt                ;    and halt


increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	24
r1	128
r2	14
r3	11



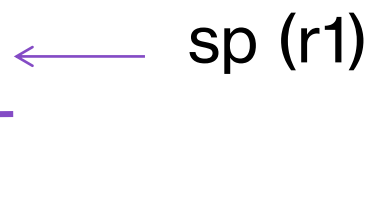
Our first function call

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2


14   sto r3 r0          ; write result,
16   hlt                ;    and halt

increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	24
r1	130
r2	14
r3	11



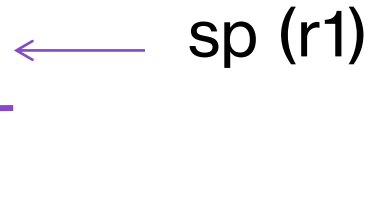
Our first function call

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2


14   sto r3 r0          ; write result,
16   hlt                ;    and halt

increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	28
r1	130
r2	14
r3	11



Our first function call

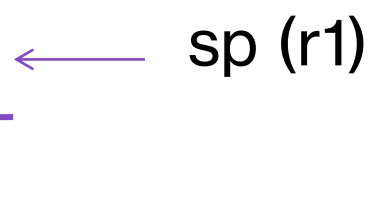
```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2

14   sto r3 r0          ; write result,
16   hlt                ;    and halt

increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	14
r1	130
r2	14
r3	11

11 ☺



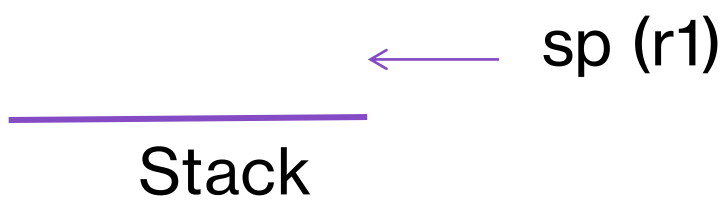
Our first function call

```
6    loa r3 r0          ; get input from user for input parameter
8    lcw r2 increment   ; call increment
12   cal r2 r2


14   sto r3 r0          ; write result,
16   hlt                ;    and halt

increment
18    psh r2            ; save the return address on the stack
22    add r3 r3 1       ; add 1 to the input parameter
24    pop r2            ; get the return address from stack
28    jmp r2            ; go back to where we were called from
```

ic	16
r1	130
r2	14
r3	11

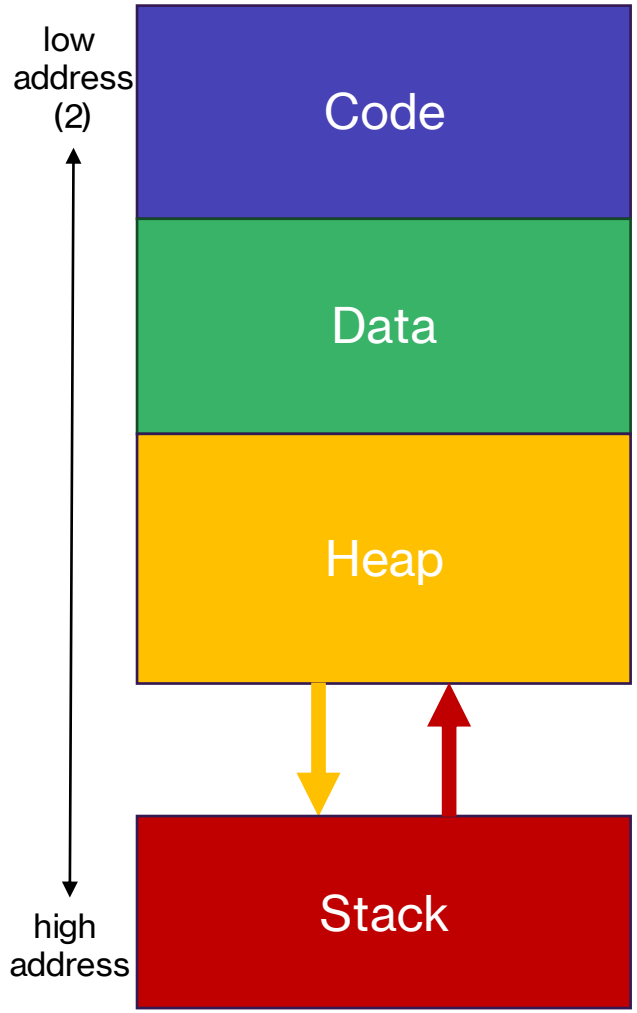


Real structure of CS51 program

```
; great comments at the top!
;
        lcw r1 stack

        instruction1            ; comment
        instruction2            ; comment
        ...
        hlt


;
; stack area: 50 words
;
        dat 100
stack
```



To the simulator!



look at increment.a51 code

Functions with multiple parameters

When we call a function:

ic	instruction counter
r0	holds the value 0 (read only)
r1	holds the address of the top of the stack
r2	return address
r3	first input to function

Functions with multiple arguments

```
function_name
        psh r2              ; save return address on stack
        loa r2 r1 4         ; load the second parameter into r2
        ...                 ; do work using r3 and r2 as arguments
                            ; put result in r3
        pop r2              ; restore return address from stack
        jmp r2              ; return to caller
```

loa R_a R_b: R_a = mem[R_b]

What does this operation do? What is the 4?

Functions with multiple arguments

```
function_name
        psh r2                  ; save return address on stack
        loa r2 r1 4             ; load the second parameter into r2
        ...                     ; do work using r3 and r2 as arguments
                                ; put result in r3
        pop r2                  ; restore return address from stack
        jmp r2                  ; return to caller
```

$$\texttt{loa } R_a\ R_b\texttt{:} \quad R_a = \texttt{mem}[R_b]$$

$$\texttt{loa } R_a\ R_b\ \texttt{S:} \quad R_a = \texttt{mem}[R_b + \texttt{S}]$$

What does this operation do? What is the 4?

Functions with multiple arguments

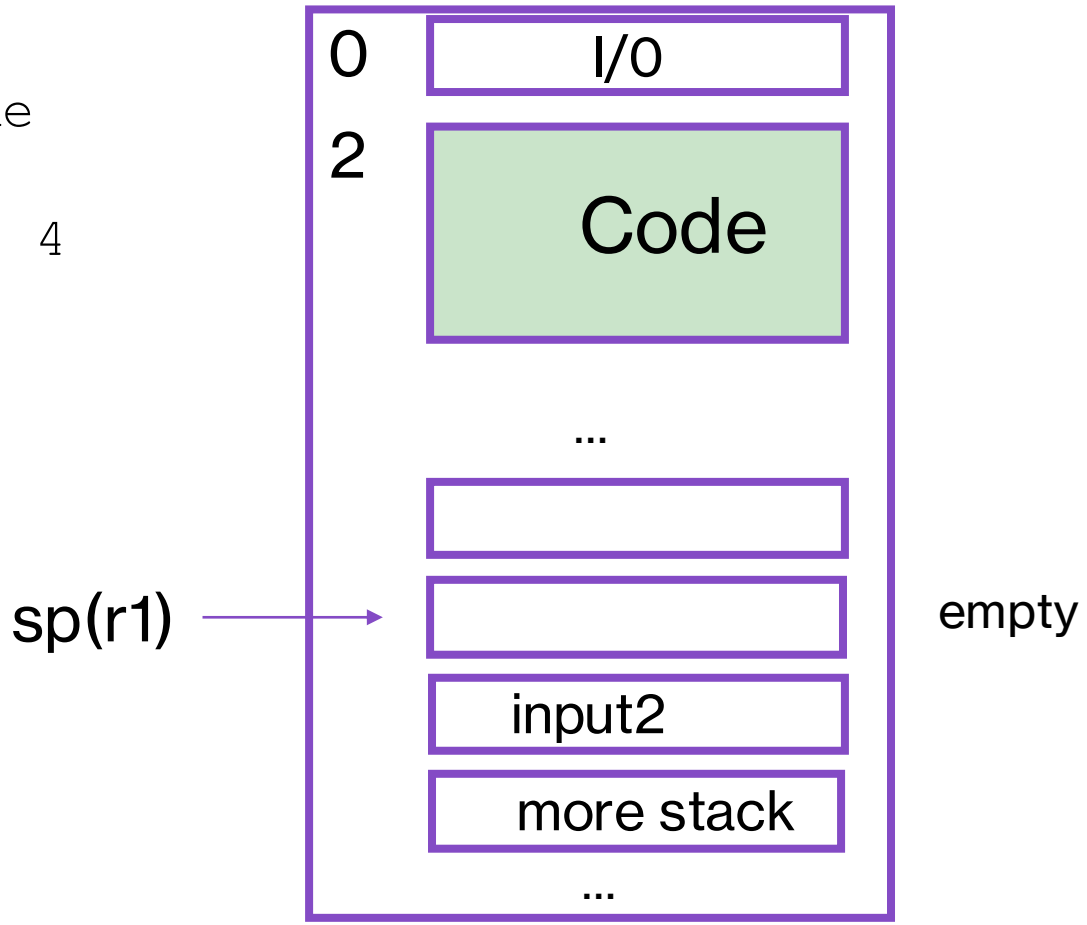
```
function_name
        psh r2                  ; save return address on stack
        loa r2 r1 4             ; load the second parameter into r2
        ...                     ; do work using r3 and r2 as arguments
                                ; put result in r3
        pop r2                  ; restore return address from stack
        jmp r2                  ; return to caller
```

$$\texttt{loa } R_a\ R_b\ S\texttt{:}\quad R_a = \texttt{mem}[R_b + S]$$

- r1 is the stack pointer and points at the top (next) slot
- stacks grow towards smaller memory values

Functions with multiple arguments

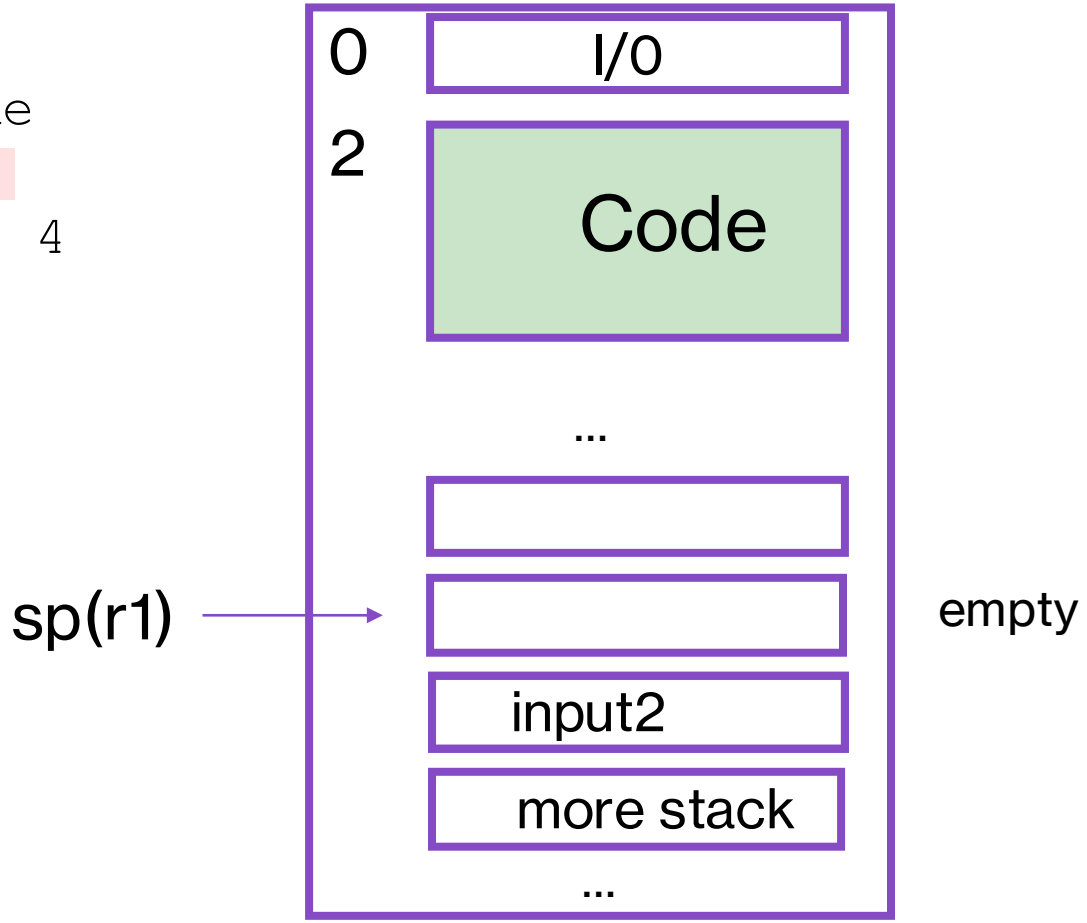
```
function_name
    psh r2
    loa r2 r1 4
```



When we call a function, input1 is stored in r3, additional arguments on top of stack

Functions with multiple arguments

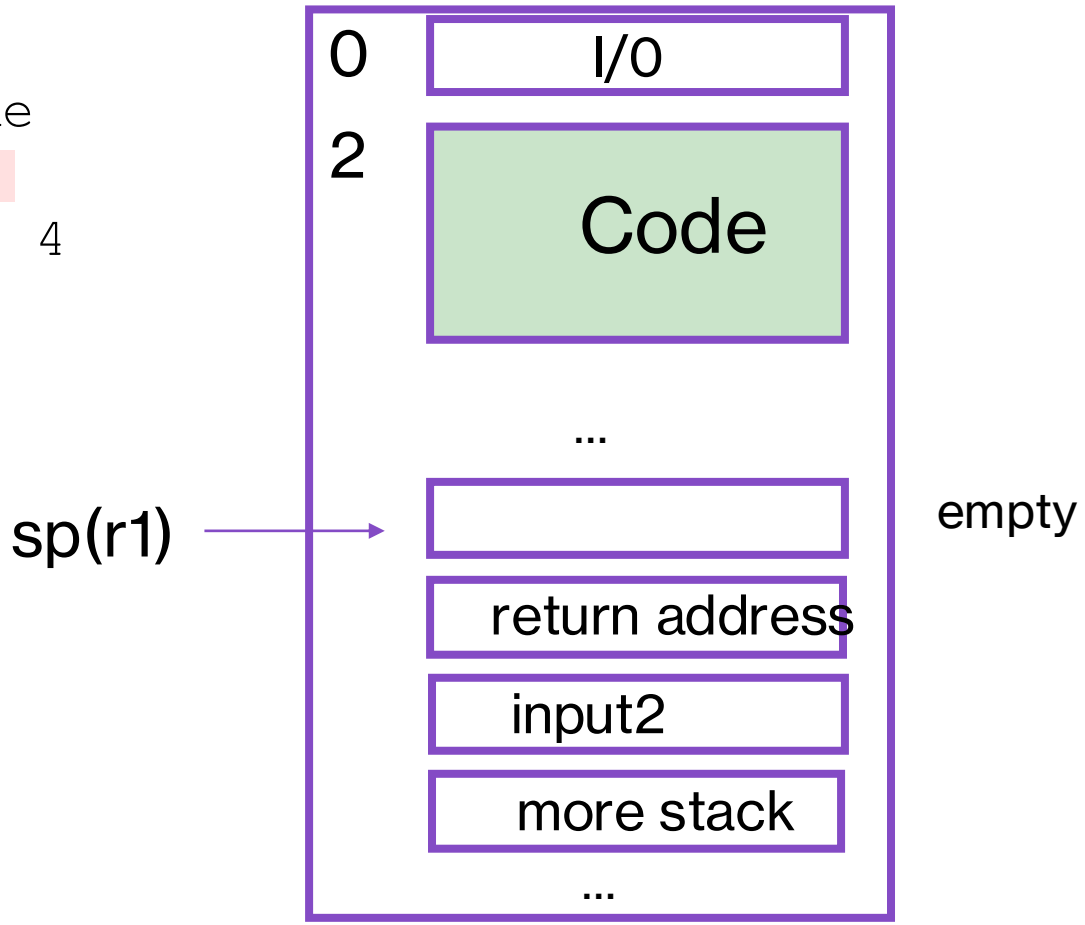
```
function_name
   psh r2
   loa r2 r1 4
```



First thing we do in a function is save the return address

Functions with multiple arguments

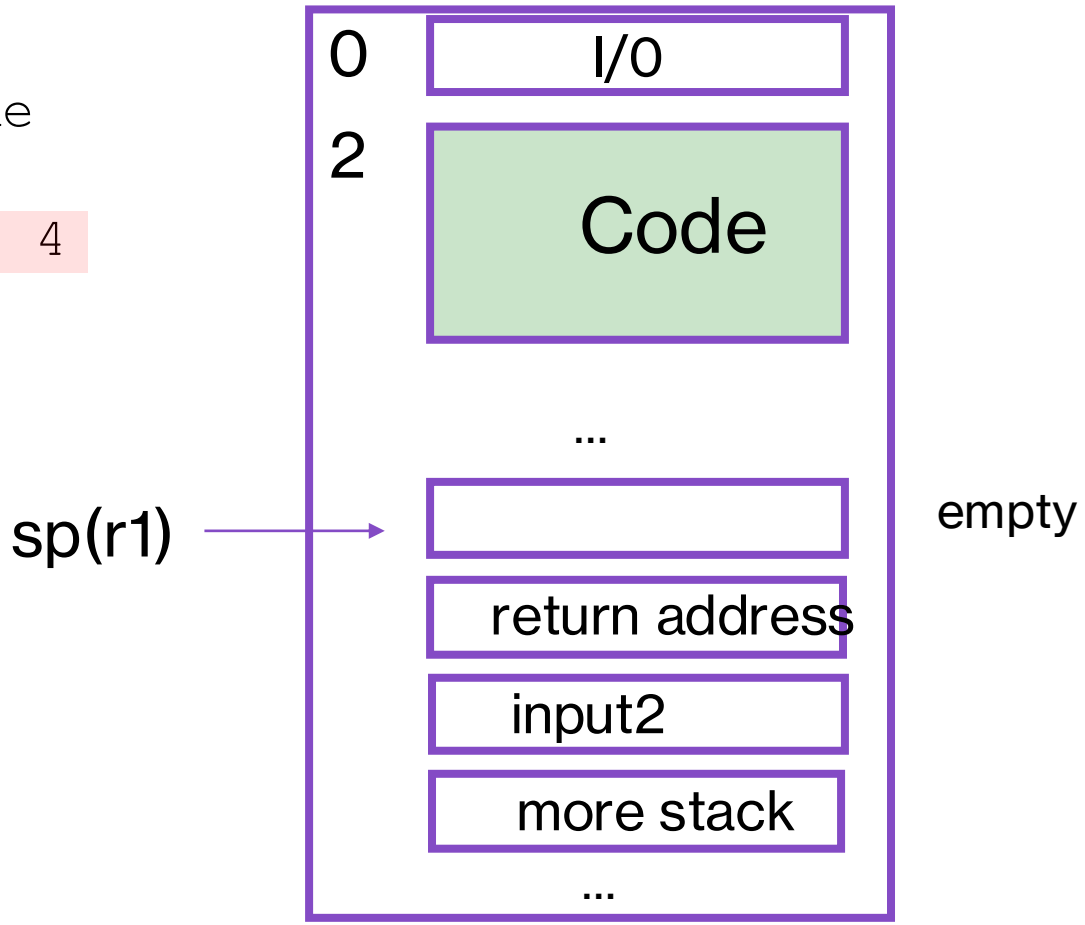
```
function_name
   psh r2
   loa r2 r1 4
```



First thing we do in a function is save the return address

Functions with multiple arguments

```
function_name
    psh r2
    loa r2 r1 4
```

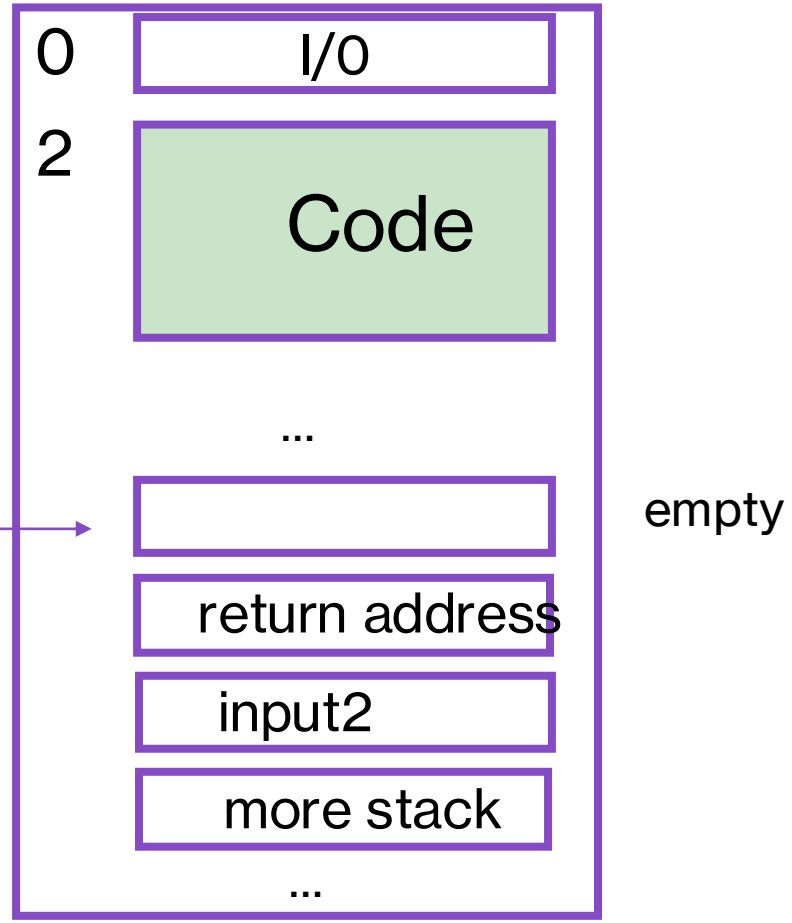


We've freed up r2. *How do we get input2 into r2?*

Functions with multiple arguments

```
function_name
   psh r2
   loa r2 r1 4
```

```
r2 = mem[r1] + 4
```



Want to look ahead two “words” from where the stack pointer is

Think: What does this function do?

```
mystery
    psh r2                  ; save the return address
    loa r2 r1 4             ; get the second argument
    add r3 r3 r2
    pop r2                  ; get the return address
    jmp r2                  ; return (answer is in r3)
```

- conventions:
  - r2 has the return address
  - input1 is in r3
  - input2 is on top of stack
  - return value goes in r3

add (add.a51)

```
2    lcw r1 stack            ; setup the stack
6    loa r3 r0               ; get input1
8    loa r2 r0               ; get input2

10   psh r2                  ; save input2 on the stack
14   lcw r2 add              ; setup call to add_then_double
18   cal r2 r2               ; call add
20   pop r0                  ; remove input2 from stack

24   sto r3 r0               ; print out the result
26   hlt

add
28   psh r2                  ; save the return address
32   loa r2 r1 4             ; get the second argument
34   add r3 r3 r2
36   pop r2                  ; get the return address
40   jmp r2                  ; return (answer is in r3)

42     dat 100
stack
```

notice the symmetry of psh/pop

add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	2
r1	142
r2	0
r3	0

add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

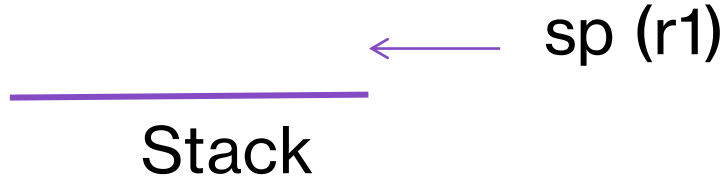
10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	6
r1	142
r2	0
r3	10



add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

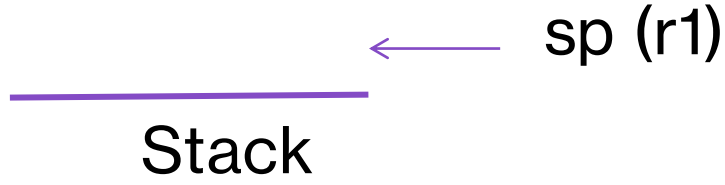
10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	8
r1	142
r2	4
r3	10



add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

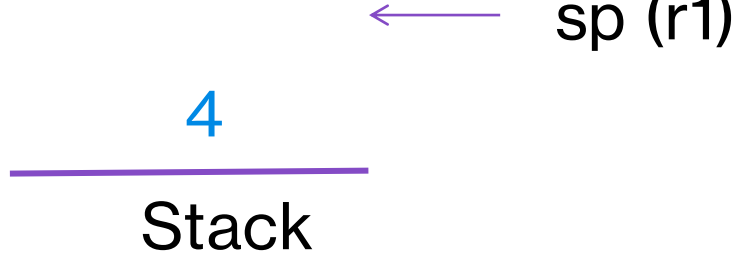
10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	10
r1	140
r2	4
r3	10



add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	14
r1	140
r2	28
r3	10



add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

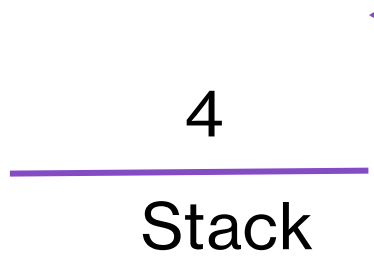
10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	18
r1	140
r2	28
r3	10



add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	28
r1	140
r2	20
r3	10



add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

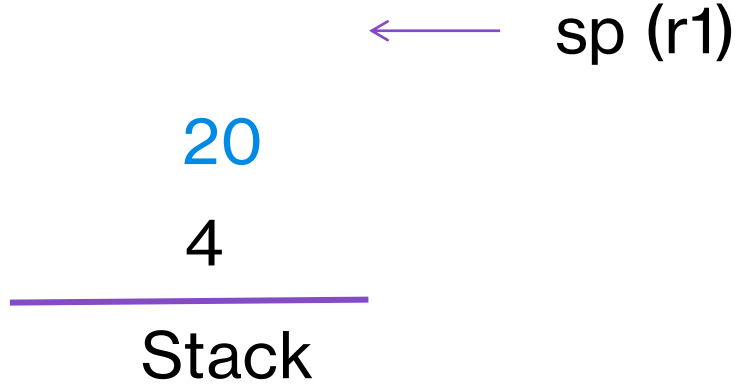
10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	28
r1	138
r2	20
r3	10



add (add.a51)

```
2    lcw r1 stack            ; setup the stack
6    loa r3 r0               ; get input1
8    loa r2 r0               ; get input2

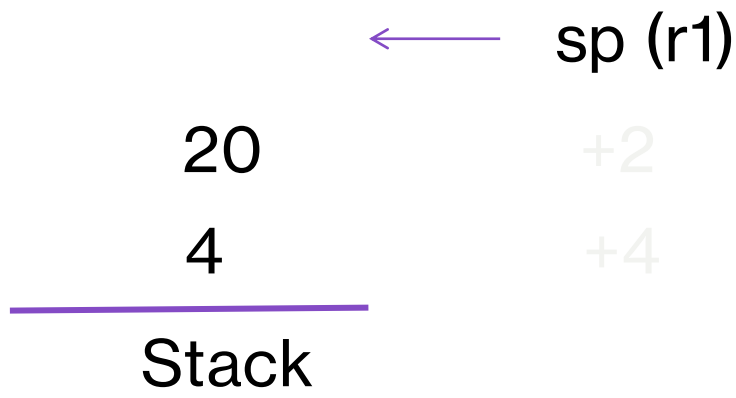
10   psh r2                  ; save input2 on the stack
14   lcw r2 add              ; setup call to add
18   cal r2 r2               ; call add
20   pop r0                  ; remove input2 from stack

24   sto r3 r0               ; print out the result
26   hlt

add
28   psh r2                  ; save the return address
32   loa r2 r1 4             ; get the second argument
34   add r3 r3 r2
36   pop r2                  ; get the return address
40   jmp r2                  ; return (answer is in r3)

42     dat 100
stack
```

ic	32
r1	138
r2	20
r3	10



add (add.a51)

```
2    lcw r1 stack            ; setup the stack
6    loa r3 r0               ; get input1
8    loa r2 r0               ; get input2

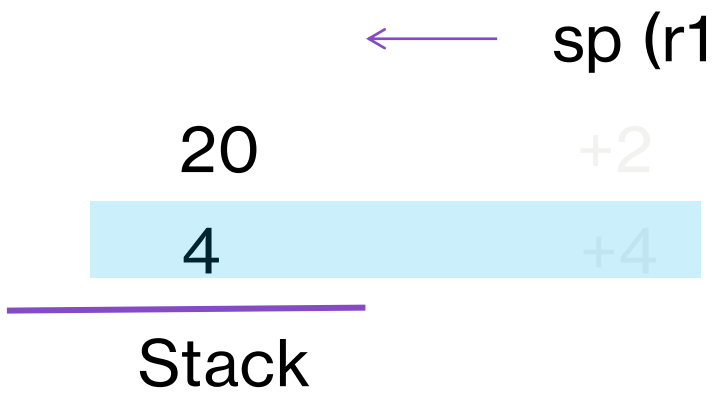
10   psh r2                  ; save input2 on the stack
14   lcw r2 add              ; setup call to add
18   cal r2 r2               ; call add
20   pop r0                  ; remove input2 from stack

24   sto r3 r0               ; print out the result
26   hlt

add
28   psh r2                  ; save the return address
32   loa r2 r1 4             ; get the second argument
34   add r3 r3 r2
36   pop r2                  ; get the return address
40   jmp r2                  ; return (answer is in r3)

42     dat 100
stack
```

ic	32
r1	138
r2	4
r3	10



add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

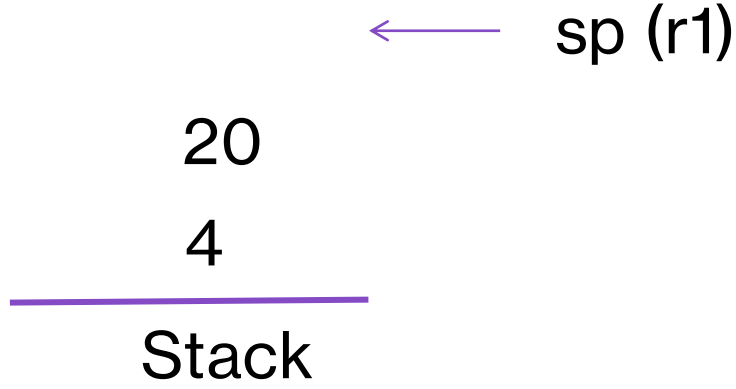
10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	34
r1	138
r2	4
r3	14



add (add.a51)

```
2    lcw r1 stack            ; setup the stack
6    loa r3 r0               ; get input1
8    loa r2 r0               ; get input2

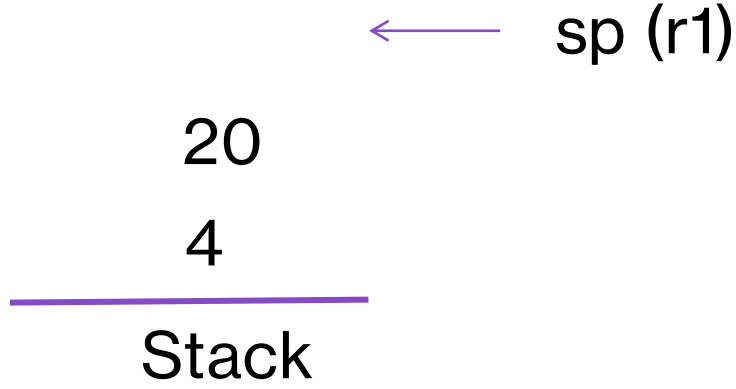
10   psh r2                  ; save input2 on the stack
14   lcw r2 add              ; setup call to add
18   cal r2 r2               ; call add
20   pop r0                  ; remove input2 from stack

24   sto r3 r0               ; print out the result
26   hlt

add
28   psh r2                  ; save the return address
32   loa r2 r1 4             ; get the second argument
34   add r3 r3 r2
36   pop r2                  ; get the return address
40   jmp r2                  ; return (answer is in r3)

42     dat 100
stack
```

ic	36
r1	138
r2	4
r3	14



add (add.a51)

```
2    lcw r1 stack            ; setup the stack
6    loa r3 r0               ; get input1
8    loa r2 r0               ; get input2

10   psh r2                  ; save input2 on the stack
14   lcw r2 add              ; setup call to add
18   cal r2 r2               ; call add
20   pop r0                  ; remove input2 from stack

24   sto r3 r0               ; print out the result
26   hlt

add
28   psh r2                  ; save the return address
32   loa r2 r1 4             ; get the second argument
34   add r3 r3 r2
36   pop r2                  ; get the return address
40   jmp r2                  ; return (answer is in r3)

42     dat 100
stack
```

ic	36
r1	140
r2	20
r3	14



add (add.a51)

```
2    lcw r1 stack            ; setup the stack
6    loa r3 r0               ; get input1
8    loa r2 r0               ; get input2

10   psh r2                  ; save input2 on the stack
14   lcw r2 add              ; setup call to add
18   cal r2 r2               ; call add
20   pop r0                  ; remove input2 from stack

24   sto r3 r0               ; print out the result
26   hlt

add
28   psh r2                  ; save the return address
32   loa r2 r1 4             ; get the second argument
34   add r3 r3 r2
36   pop r2                  ; get the return address
40   jmp r2                  ; return (answer is in r3)

42     dat 100
stack
```

ic	40
r1	140
r2	20
r3	14



add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	20
r1	140
r2	20
r3	14



add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

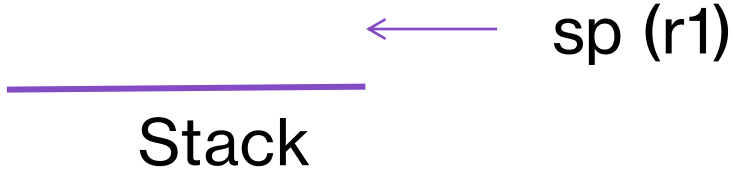
10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	20
r1	142
r2	20
r3	14



add (add.a51)

```
2    lcw r1 stack            ; setup the stack
6    loa r3 r0               ; get input1
8    loa r2 r0               ; get input2

10   psh r2                  ; save input2 on the stack
14   lcw r2 add              ; setup call to add
18   cal r2 r2               ; call add
20   pop r0                  ; remove input2 from stack

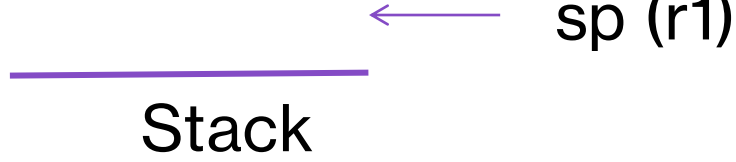
24   sto r3 r0               ; print out the result
26   hlt

add
28   psh r2                  ; save the return address
32   loa r2 r1 4             ; get the second argument
34   add r3 r3 r2
36   pop r2                  ; get the return address
40   jmp r2                  ; return (answer is in r3)

42     dat 100
stack
```

14 ☺

ic	24
r1	142
r2	20
r3	14



add (add.a51)

```
2    lcw r1 stack              ; setup the stack
6    loa r3 r0                 ; get input1
8    loa r2 r0                 ; get input2

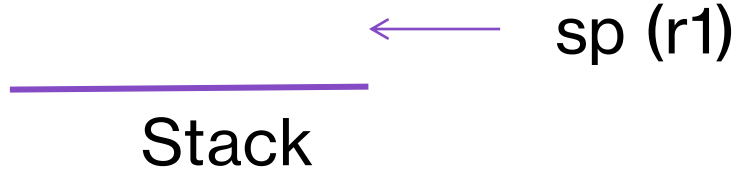
10   psh r2                    ; save input2 on the stack
14   lcw r2 add                ; setup call to add
18   cal r2 r2                 ; call add
20   pop r0                    ; remove input2 from stack

24   sto r3 r0                 ; print out the result
26   hlt

add
28   psh r2                    ; save the return address
32   loa r2 r1 4               ; get the second argument
34   add r3 r3 r2
36   pop r2                    ; get the return address
40   jmp r2                    ; return (answer is in r3)

42     dat 100
stack
```

ic	26
r1	142
r2	20
r3	14

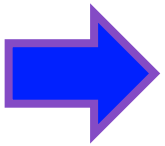


Call stack in action

```
def add(num1, num2):
        return num1 + num2

def double(num):
        return 2 * num

def add_then_double(a, b):
        added = add(a, b)
        return double(added)
```



```
 lcw r1 stack    ; setup the stack
    loa r3 r0        ; get input1
    loa r2 r0        ; get input2

    psh r2           ; save input2 on the stack
    lcw r2 add_then_double ; setup call to add_then_double
    cal r2 r2        ; call add_then_double
    pop r0           ; remove input2 from stack

    sto r3 r0        ; print out the result
    hlt

add
    psh r2           ; save the return address
    loa r2 r1 4      ; get the second argument
    add r3 r3 r2     ; add r3 = arg1 + arg2
    pop r2           ; get the return address
    jmp r2           ; return

double
    psh r2           ; save the return address
    add r3 r3 r3     ; double the input2
    pop r2           ; get the return address
    jmp r2           ; return

add_then_double
    psh r2           ; save the return address
    loa r2 r1 4      ; get the second argument

    psh r2           ; setup call for add
    lcw r2 add       ; r3 has input one, push input2 on stack
    cal r2 r2        ; call add
    pop r0           ; remove input2 from stack

    lcw r2 double    ; result of add is in r3
    cal r2 r2        ; call double

    pop r2           ; get the return address
    jmp r2           ; return

    dat 100          ; setup the stack
stack
```

add_then_double.a51

```
2   lcw r1 stack     ; setup the stack
6   loa r3 r0        ; get input1
8   loa r2 r0        ; get input2

10  psh r2           ; save input2 on the stack
14  lcw r2 add_then_double ; setup call to add_then_double
18  cal r2 r2        ; call add_then_double
20  pop r0           ; remove input2 from stack

24  sto r3 r0        ; print out the result
26  hlt
```

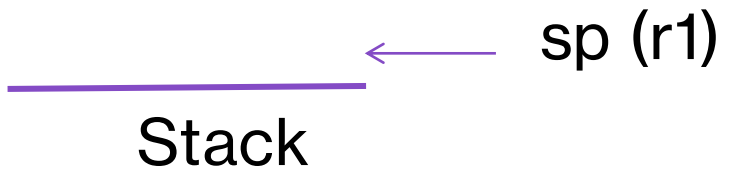
add_then_double.a51

ic	2
r1	186
r2	0
r3	0

```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt
```



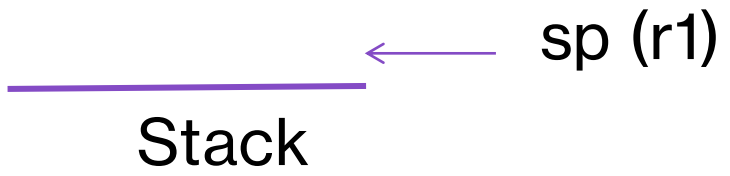
add_then_double.a51

ic	8
r1	186
r2	4
r3	-10

```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt
```



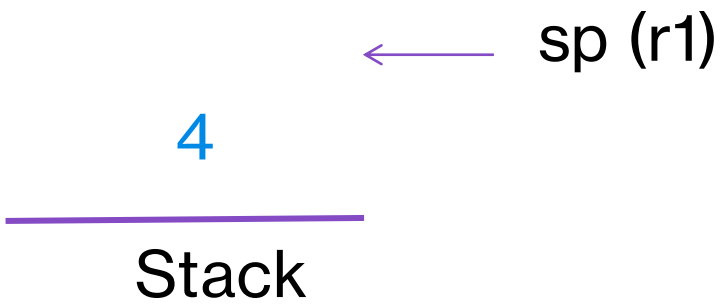
add_then_double.a51

ic	10
r1	184
r2	4
r3	-10

```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt
```



add_then_double.a51

ic	14
r1	184
r2	4
r3	-10

```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

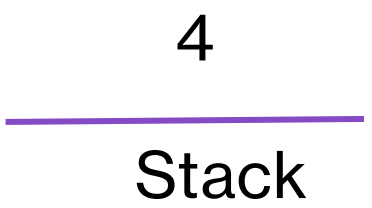
10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt
```

..

```
add_then_double
54 psh r2
```





add_then_double.a51

ic	14
r1	184
r2	54
r3	-10

```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt
```

..

```
add_then_double
54 psh r2
```



add_then_double.a51

ic	14
r1	184
r2	54
r3	-10

```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt

..

add_then_double
54 psh r2
```



add_then_double.a51

ic	54
r1	184
r2	20
r3	-10

```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt

..

add_then_double
54  psh r2
```



add_then_double.a51

ic	54
r1	182
r2	20
r3	-10

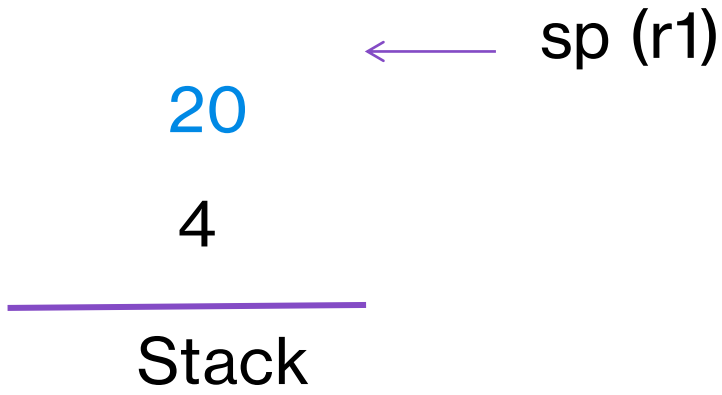
```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt

..

add_then_double
54  psh r2
```



add_then_double.a51

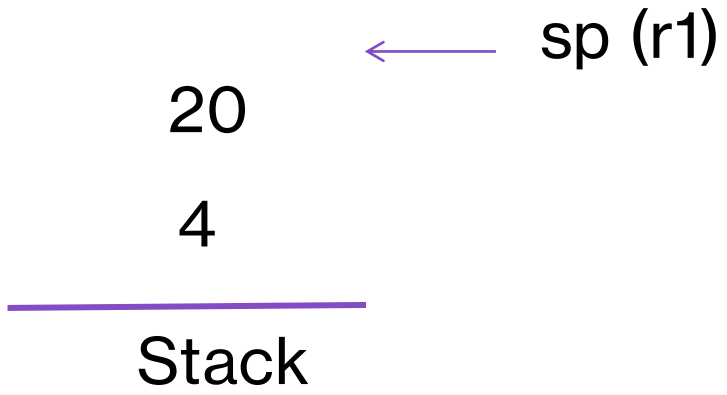
ic	54
r1	182
r2	20
r3	-10

```
add_then_double
54 psh r2           ; save the return address
60 loa r2 r1 4      ; get the second argument

62 psh r2           ; setup call for add
66 lcw r2 add       ; r3 has input one, push input2 on stack
70 cal r2 r2        ; call add
72 pop r0           ; remove input2 from stack

76 lcw r2 double    ; result of add is in r3
80 cal r2 r2        ; call double

82 pop r2           ; get the return address
86 jmp r2           ; return
```



add_then_double.a51

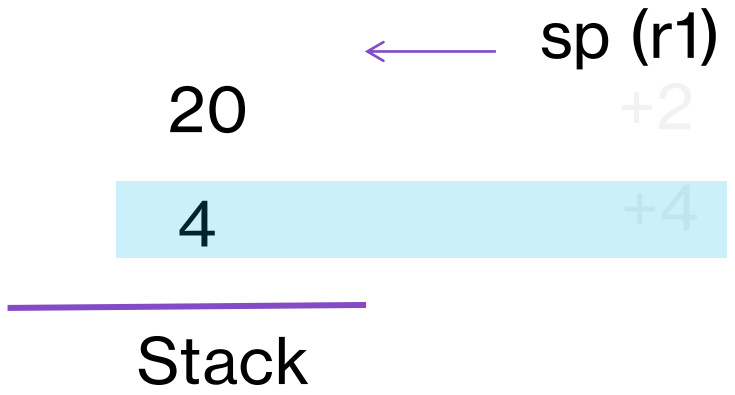
ic	60
r1	182
r2	4
r3	-10

```
add_then_double
54 psh r2           ; save the return address
60 loa r2 r1 4      ; get the second argument

62 psh r2           ; setup call for add
66 lcw r2 add       ; r3 has input one, push input2 on stack
70 cal r2 r2        ; call add
72 pop r0           ; remove input2 from stack

76 lcw r2 double    ; result of add is in r3
80 cal r2 r2        ; call double

82 pop r2           ; get the return address
86 jmp r2           ; return
```



add_then_double.a51

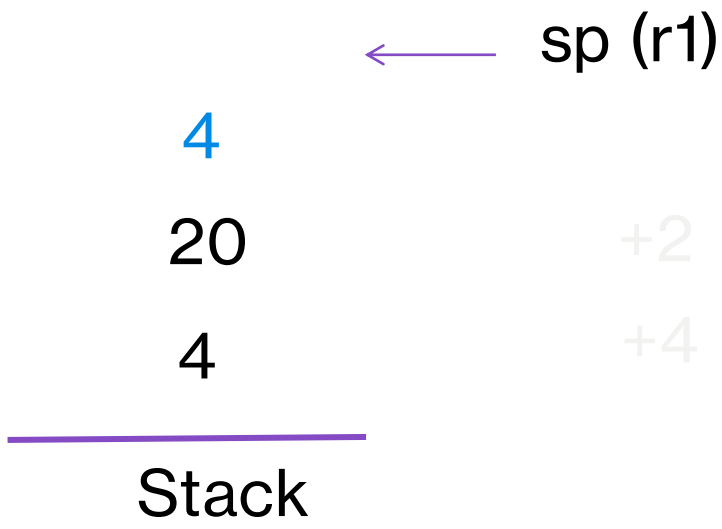
ic	62
r1	180
r2	4
r3	-10

```
add_then_double
54 psh r2           ; save the return address
60 loa r2 r1 4      ; get the second argument

62 psh r2           ; setup call for add
66 lcw r2 add       ; r3 has input one, push input2 on stack
70 cal r2 r2        ; call add
72 pop r0           ; remove input2 from stack

76 lcw r2 double    ; result of add is in r3
80 cal r2 r2        ; call double

82 pop r2           ; get the return address
86 jmp r2           ; return
```



add_then_double.a51

```
add
28     psh r2              ; save the return address
…

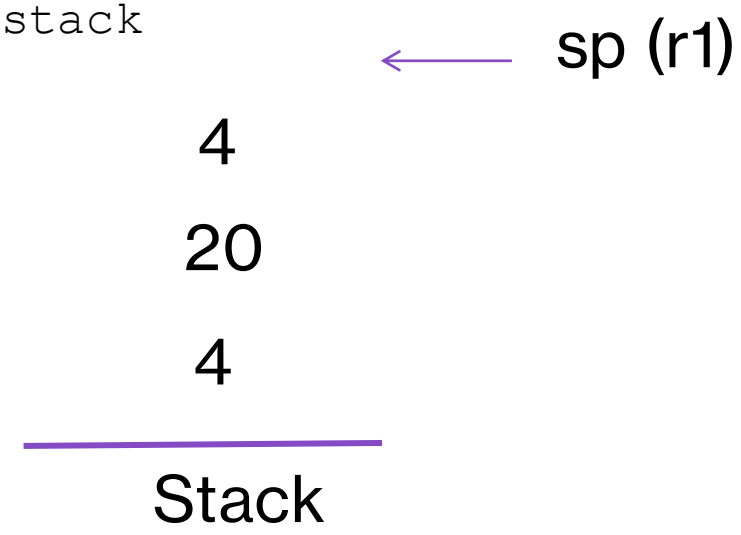
add_then_double
54  psh r2                 ; save the return address
60  loa r2 r1 4            ; get the second argument

62  psh r2                 ; setup call for add
66  lcw r2 add             ; r3 has input one, push input2 on stack
70  cal r2 r2              ; call add
72  pop r0                 ; remove input2 from stack

76  lcw r2 double          ; result of add is in r3
80  cal r2 r2              ; call double

82  pop r2                 ; get the return address
86  jmp r2                 ; return
```

ic	66
r1	180
r2	28
r3	-10



add_then_double.a51

ic	70
r1	180
r2	72
r3	-10

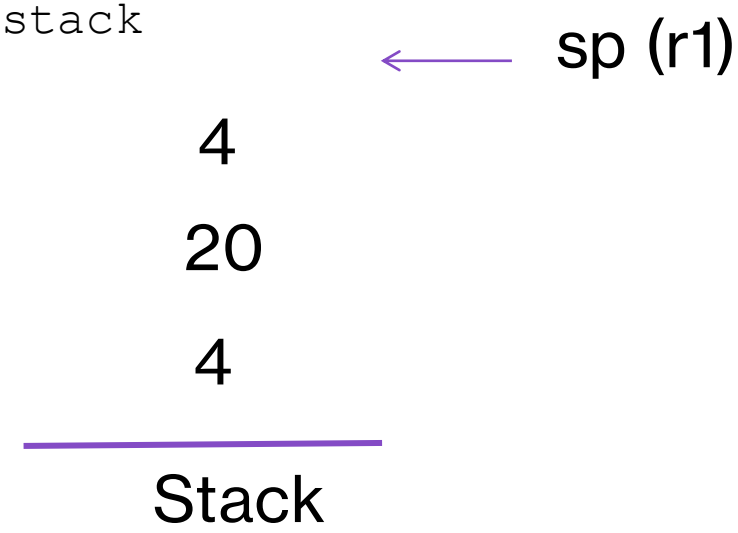
```
add
28     psh r2              ; save the return address
…

add_then_double
54  psh r2                 ; save the return address
60  loa r2 r1 4            ; get the second argument

62  psh r2                 ; setup call for add
66  lcw r2 add             ; r3 has input one, push input2 on stack
70  cal r2 r2              ; call add
72  pop r0                 ; remove input2 from stack

76  lcw r2 double          ; result of add is in r3
80  cal r2 r2              ; call double

82  pop r2                 ; get the return address
86  jmp r2                 ; return
```



add_then_double.a51

ic	28
r1	180
r2	72
r3	-10

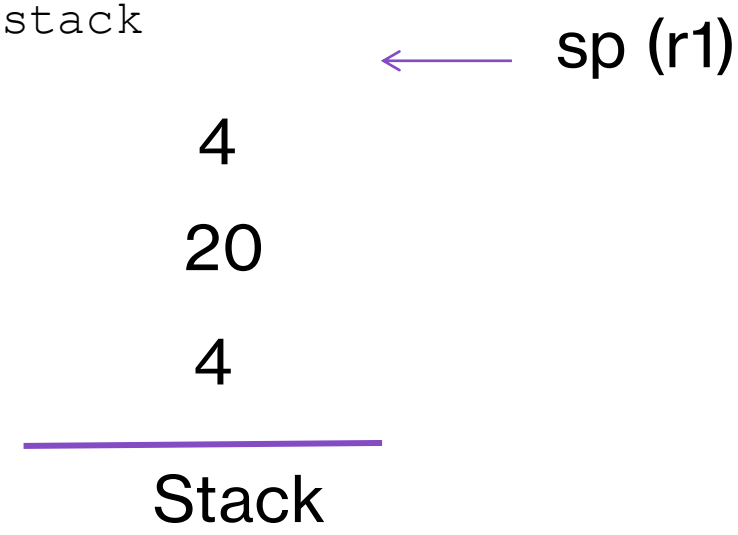
```
add
28     psh r2          ; save the return address
…

add_then_double
54 psh r2              ; save the return address
60 loa r2 r1 4         ; get the second argument

62 psh r2              ; setup call for add
66 lcw r2 add          ; r3 has input one, push input2 on stack
70 cal r2 r2           ; call add
72 pop r0              ; remove input2 from stack

76 lcw r2 double       ; result of add is in r3
80 cal r2 r2           ; call double

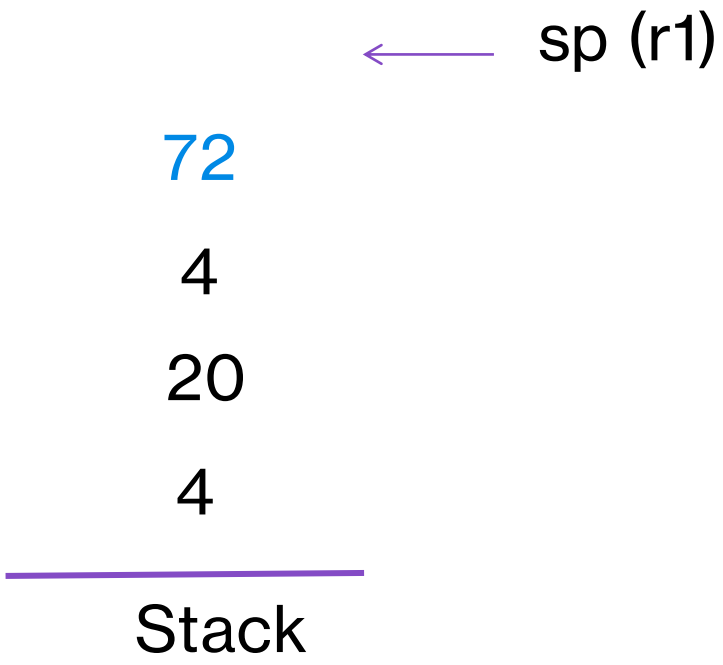
82 pop r2              ; get the return address
86 jmp r2              ; return
```



add_then_double.a51

```
add
28      psh r2          ; save the return address
32      loa r2 r1 4     ; get the second argument
34      add r3 r3 r2    ; add r3 = arg1 + arg2
36      pop r2          ; get the return address
40      jmp r2          ; return
```

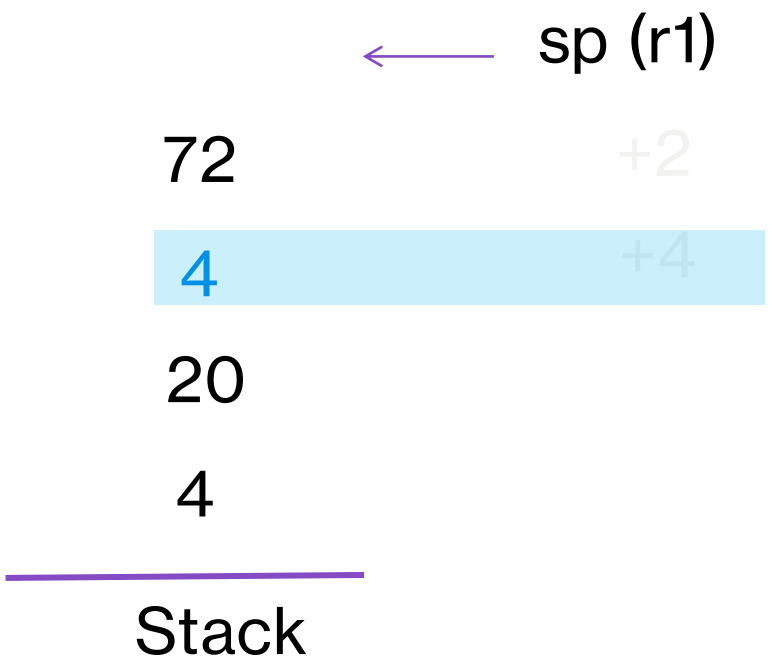
ic	28
r1	178
r2	72
r3	-10



add_then_double.a51

```
add
28      psh r2              ; save the return address
32      loa r2 r1 4         ; get the second argument
34      add r3 r3 r2        ; add r3 = arg1 + arg2
36      pop r2              ; get the return address
40      jmp r2              ; return
```

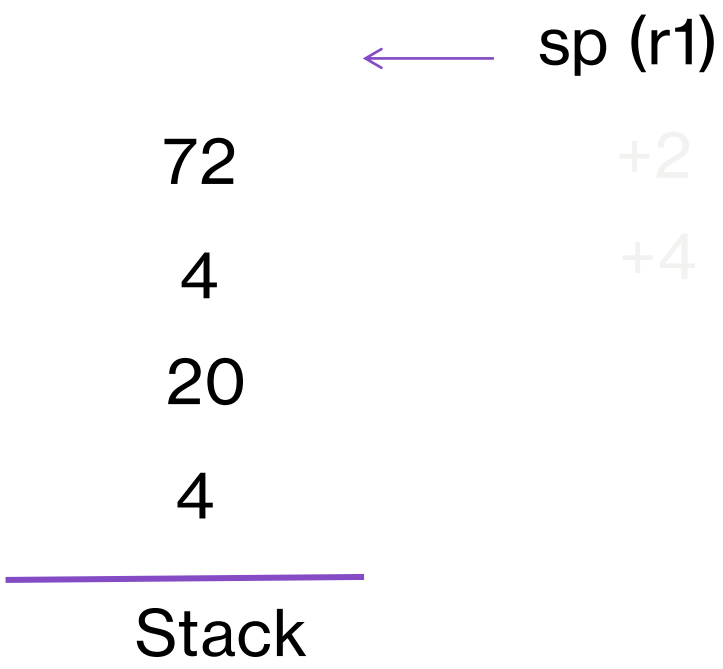
ic	32
r1	178
r2	4
r3	-10



add_then_double.a51

```
add
28      psh r2              ; save the return address
32      loa r2 r1 4         ; get the second argument
34      add r3 r3 r2        ; add r3 = arg1 + arg2
36      pop r2              ; get the return address
40      jmp r2              ; return
```

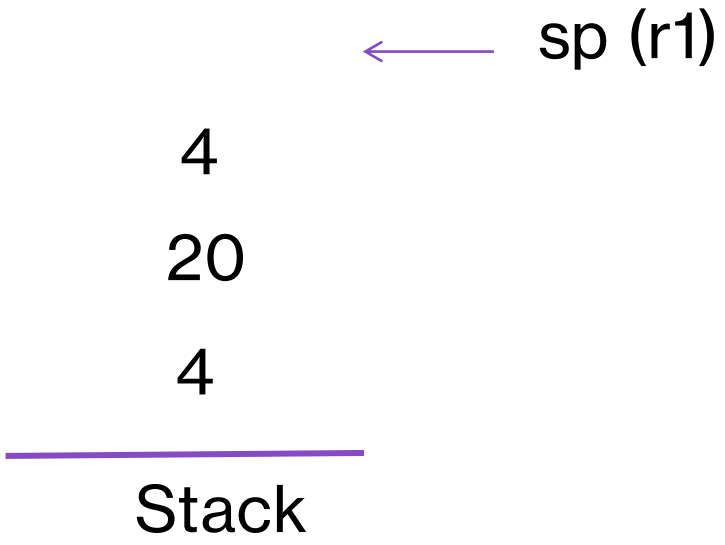
ic	34
r1	178
r2	4
r3	-6



add_then_double.a51

```
add
28      psh r2              ; save the return address
32      loa r2 r1 4         ; get the second argument
34      add r3 r3 r2        ; add r3 = arg1 + arg2
36      pop r2              ; get the return address
40      jmp r2              ; return
```

ic	34
r1	180
r2	72
r3	-6



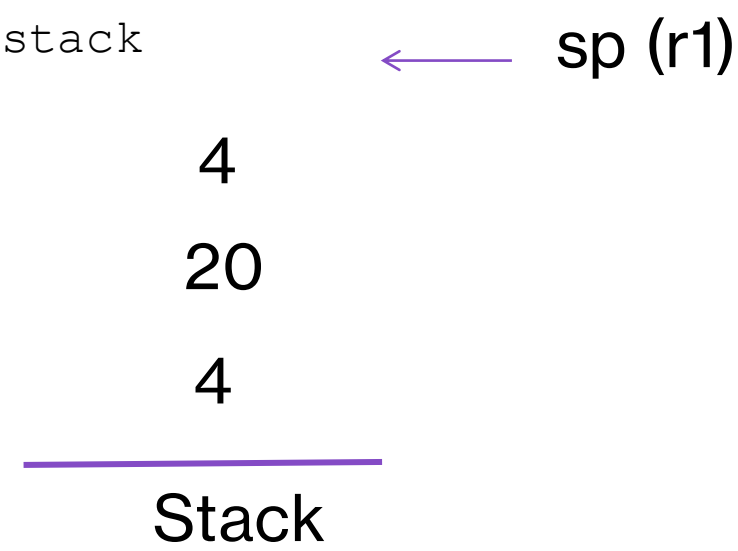
add_then_double.a51

ic	40
r1	180
r2	72
r3	-6

```
add
28      psh r2              ; save the return address
32      loa r2 r1 4         ; get the second argument
34      add r3 r3 r2        ; add r3 = arg1 + arg2
36      pop r2              ; get the return address
40      jmp r2              ; return
…
add_then_double
…
66  lcw r2 add          ; r3 has input one, push input2 on stack
70  cal r2 r2           ; call add
72  pop r0              ; remove input2 from stack

76  lcw r2 double       ; result of add is in r3
80  cal r2 r2           ; call double

82  pop r2              ; get the return address
86  jmp r2              ; return
```



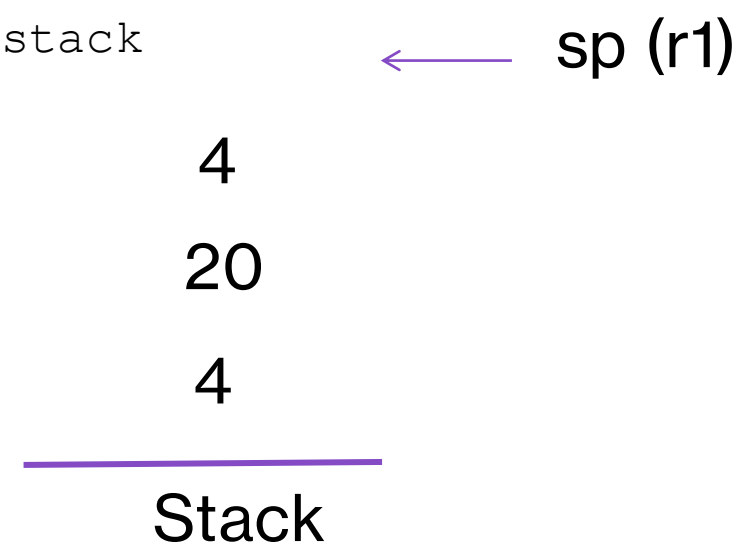
add_then_double.a51

ic	72
r1	180
r2	72
r3	-6

```
add
28      psh r2              ; save the return address
32      loa r2 r1 4         ; get the second argument
34      add r3 r3 r2        ; add r3 = arg1 + arg2
36      pop r2              ; get the return address
40      jmp r2              ; return
…
add_then_double
…
66  lcw r2 add          ; r3 has input one, push input2 on stack
70  cal r2 r2           ; call add
72  pop r0              ; remove input2 from stack

76  lcw r2 double       ; result of add is in r3
80  cal r2 r2           ; call double

82  pop r2              ; get the return address
86  jmp r2              ; return
```



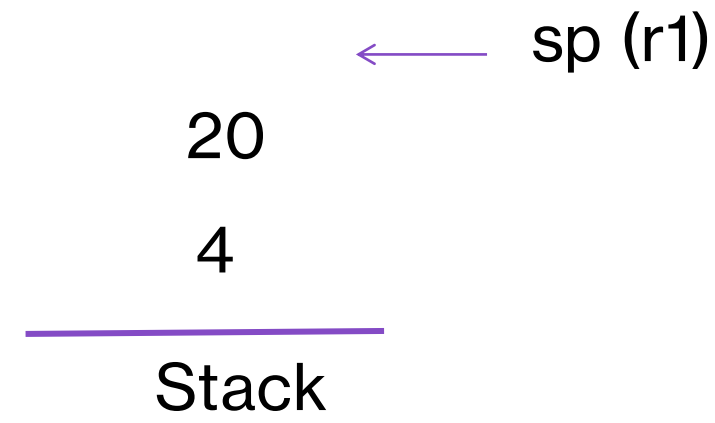
add_then_double.a51

ic	76
r1	182
r2	42
r3	-6

```
double
42      psh r2              ; save the return address
46      add r3 r3 r3        ; double the input2
48      pop r2              ; get the return address
52      jmp r2              ; return
…
add_then_double
…
66  lcw r2 add          ; r3 has input one, push input2 on stack
70  cal r2 r2           ; call add
72  pop r0              ; remove input2 from stack

76  lcw r2 double       ; result of add is in r3
80  cal r2 r2           ; call double

82  pop r2              ; get the return address
86  jmp r2              ; return
```



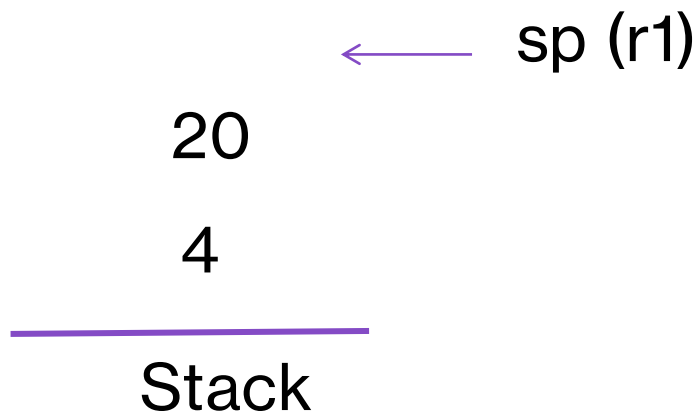
add_then_double.a51

ic	80
r1	182
r2	42
r3	-6

```
double
42      psh r2              ; save the return address
46      add r3 r3 r3        ; double the input2
48      pop r2              ; get the return address
52      jmp r2              ; return
…
add_then_double
…
66  lcw r2 add          ; r3 has input one, push input2 on stack
70  cal r2 r2           ; call add
72  pop r0              ; remove input2 from stack

76  lcw r2 double       ; result of add is in r3
80  cal r2 r2           ; call double

82  pop r2              ; get the return address
86  jmp r2              ; return
```



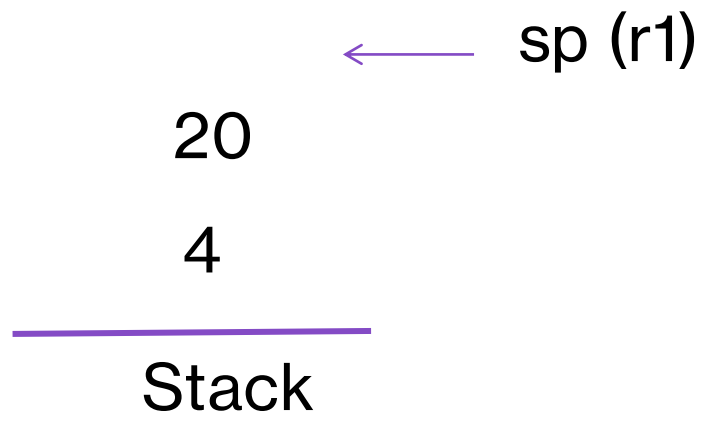
add_then_double.a51

ic	42
r1	182
r2	82
r3	-6

```
double
42      psh r2          ; save the return address
46      add r3 r3 r3    ; double the input2
48      pop r2          ; get the return address
52      jmp r2          ; return
…
add_then_double
…
66  lcw r2 add      ; r3 has input one, push input2 on stack
70  cal r2 r2       ; call add
72  pop r0          ; remove input2 from stack

76  lcw r2 double   ; result of add is in r3
80  cal r2 r2       ; call double

82  pop r2          ; get the return address
86  jmp r2          ; return
```



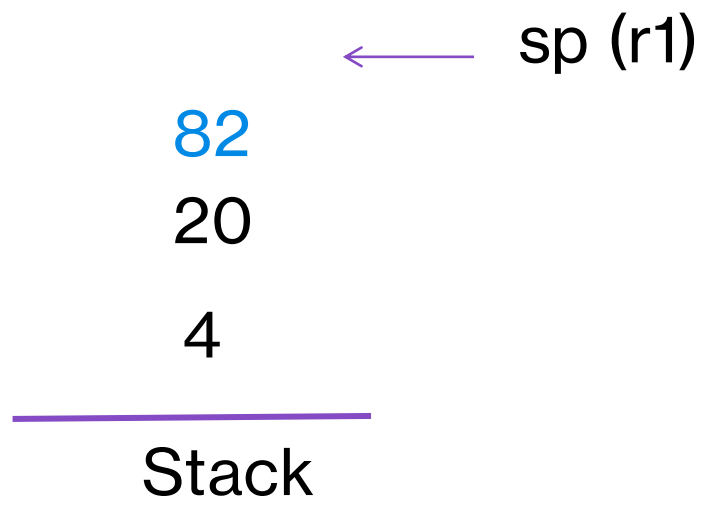
add_then_double.a51

ic	42
r1	180
r2	82
r3	-6

```
double
42      psh r2          ; save the return address
46      add r3 r3 r3    ; double the input2
48      pop r2          ; get the return address
52      jmp r2          ; return
…
add_then_double
…
66  lcw r2 add      ; r3 has input one, push input2 on stack
70  cal r2 r2       ; call add
72  pop r0          ; remove input2 from stack

76  lcw r2 double   ; result of add is in r3
80  cal r2 r2       ; call double

82  pop r2          ; get the return address
86  jmp r2          ; return
```



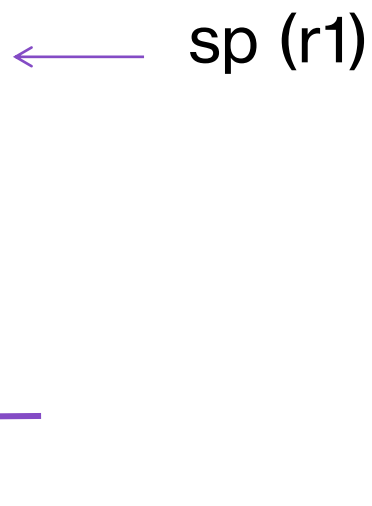
add_then_double.a51

ic	46
r1	180
r2	82
r3	-12

```
double
42      psh r2              ; save the return address
46      add r3 r3 r3        ; double the input2
48      pop r2              ; get the return address
52      jmp r2              ; return
…
add_then_double
…
66  lcw r2 add          ; r3 has input one, push input2 on stack
70  cal r2 r2           ; call add
72  pop r0              ; remove input2 from stack

76  lcw r2 double       ; result of add is in r3
80  cal r2 r2           ; call double

82  pop r2              ; get the return address
86  jmp r2              ; return
```



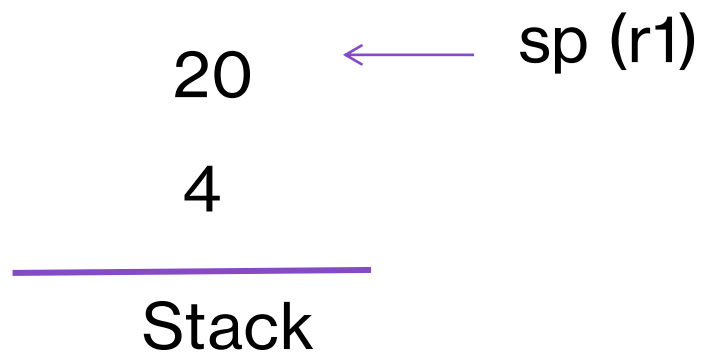
add_then_double.a51

ic	48
r1	182
r2	82
r3	-12

```
double
42      psh r2              ; save the return address
46      add r3 r3 r3        ; double the input2
48      pop r2              ; get the return address
52      jmp r2              ; return
…
add_then_double
…
66  lcw r2 add          ; r3 has input one, push input2 on stack
70  cal r2 r2           ; call add
72  pop r0              ; remove input2 from stack

76  lcw r2 double       ; result of add is in r3
80  cal r2 r2           ; call double

82  pop r2              ; get the return address
86  jmp r2              ; return
```



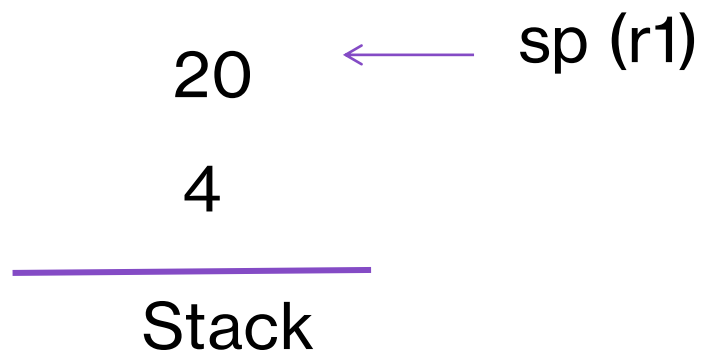
add_then_double.a51

ic	52
r1	182
r2	82
r3	-12

```
double
42      psh r2              ; save the return address
46      add r3 r3 r3        ; double the input2
48      pop r2              ; get the return address
52      jmp r2              ; return
…
add_then_double
…
66  lcw r2 add          ; r3 has input one, push input2 on stack
70  cal r2 r2           ; call add
72  pop r0              ; remove input2 from stack

76  lcw r2 double       ; result of add is in r3
80  cal r2 r2           ; call double

82  pop r2              ; get the return address
86  jmp r2              ; return
```



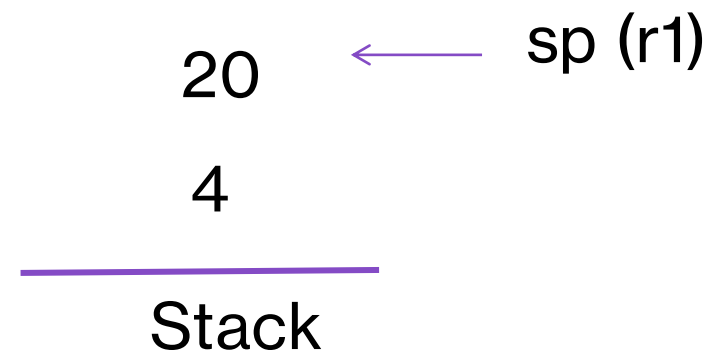
add_then_double.a51

ic	82
r1	182
r2	82
r3	-12

```
double
42      psh r2              ; save the return address
46      add r3 r3 r3        ; double the input2
48      pop r2              ; get the return address
52      jmp r2              ; return
…
add_then_double
…
66  lcw r2 add          ; r3 has input one, push input2 on stack
70  cal r2 r2           ; call add
72  pop r0              ; remove input2 from stack

76  lcw r2 double       ; result of add is in r3
80  cal r2 r2           ; call double

82  pop r2              ; get the return address
86  jmp r2              ; return
```



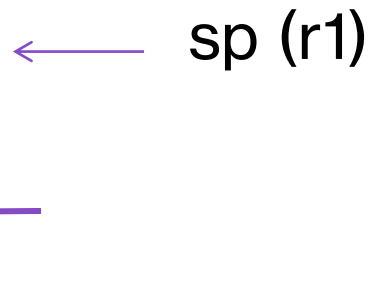
add_then_double.a51

ic	82
r1	184
r2	20
r3	-12

```
double
42      psh r2              ; save the return address
46      add r3 r3 r3        ; double the input2
48      pop r2              ; get the return address
52      jmp r2              ; return
…
add_then_double
…
66  lcw r2 add          ; r3 has input one, push input2 on stack
70  cal r2 r2           ; call add
72  pop r0              ; remove input2 from stack

76  lcw r2 double       ; result of add is in r3
80  cal r2 r2           ; call double

82  pop r2              ; get the return address
86  jmp r2              ; return
```



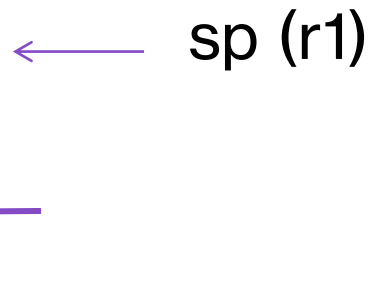
add_then_double.a51

ic	86
r1	184
r2	20
r3	-12

```
double
42      psh r2              ; save the return address
46      add r3 r3 r3        ; double the input2
48      pop r2              ; get the return address
52      jmp r2              ; return
…
add_then_double
…
66  lcw r2 add          ; r3 has input one, push input2 on stack
70  cal r2 r2           ; call add
72  pop r0              ; remove input2 from stack

76  lcw r2 double       ; result of add is in r3
80  cal r2 r2           ; call double

82  pop r2              ; get the return address
86  jmp r2              ; return
```



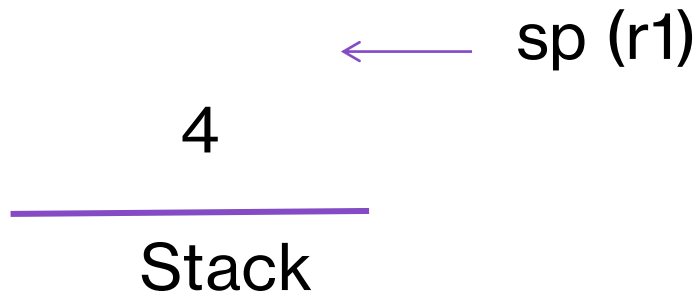
add_then_double.a51

```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt
```

ic	20
r1	184
r2	20
r3	-12



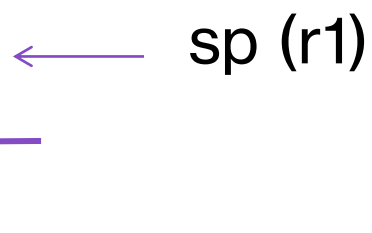
add_then_double.a51

```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt
```

ic	20
r1	186
r2	20
r3	-12



add_then_double.a51

```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt
```

ic	24
r1	186
r2	20
r3	-12

-12 ☺





add_then_double.a51

```
2    lcw r1 stack     ; setup the stack
6    loa r3 r0        ; get input1
8    loa r2 r0        ; get input2

10   psh r2           ; save input2 on the stack
14   lcw r2 add_then_double ; setup call to add_then_double
18   cal r2 r2        ; call add_then_double
20   pop r0           ; remove input2 from stack

24   sto r3 r0        ; print out the result
26   hlt
```

ic	26
r1	186
r2	20
r3	-12



sp (r1)

Stack

To the simulator!



look at add_then_double.a51 code

Legend:

`R` register	`B` signed byte offset	`F` string
`S` signed byte	`W` full word	`[ ]` optional
`U` unsigned byte	`A` 6-bit constant	

Arithmetic, bitwise logical, and shift instructions

`add`
`sub`
`and`
`orr`
`xor`
	`RRR` or `RRS`	`mov`
		`neg`
		`not`
			`RR`

Memory and data-moving instructions

`sto` `loa`	`RR[S]`	`psh` `pop`	`R`	`lcw RW`

Control instructions

`beq`
`bne`
`blt`
`bge`
`bgt`
`ble`
	`RRB`	`brs B`
		`cal RR`
		`jmp R`
			`nop`
			`hlt [A]`

Directives

`dat W`	`inc F`