CS52 RECURSION

David Kauchak
CS 52 – Spring 2017

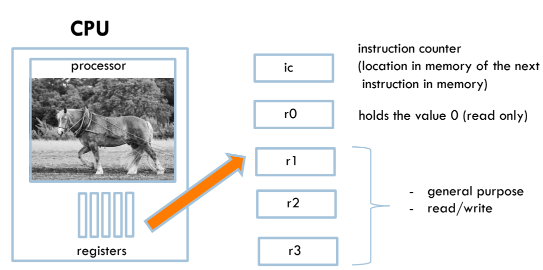
Admin

- Final grade
- Assignment grading
- Assignment 4
- Reading

Examples from this lecture

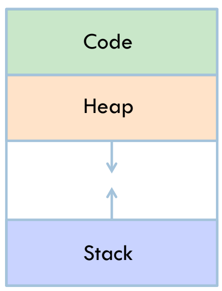
http://www.cs.pomona.edu/~dkauchak/classes/cs52/examples/cs52machine/

CS52 machine



Memory layout

0



Where dynamically allocated program data is stored

Where program/function execution information is stored, parameters, and local variables

Stack frame

Key unit for keeping track of a function call

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

CS52 function call conventions

r1: reserved for the stack pointer

r2: contains the return address

r3: contains the first parameter

additional parameters go on the stack (more on this)

the result (i.e. the return value) should go in r3

Sum revisited

```
fun sum x =
  if x <= 0 then
    0
  else
    x + sum (x-1);
```

Note to future Dave from past Dave: write the function up on the board ☺

```
sum
        psh r2              ; save the return address on the stack
        bgt r3 r0 recurse   ; check base case
        add r3 r0 0         ; if x <= 0, result is 0
        brs done

recurse
        psh r3              ; save n on the stack
        sub r3 r3 1         ; x = x-1

        lcw r2 sum          ; make recursive call
        cal r2 r2           ; sum (x-1), answer should be in r3

        pop r2              ; get n into r2
        add r3 r3 r2        ; r3 = n + sum (x-1)
done
        pop r2              ; get the return address
        jmp r2              ; go back to where we were called from
```

Function startup

base cases

recursive call

recursive case

answer calculation

Function cleanup and return

```
sum
        psh r2              ; save the return address on the stack
        bgt r3 r0 recurse   ; check base case
        add r3 r0 0         ; if n <= 0, result is 0
        brs done

recurse
        psh r3              ; save n on the stack
        sub r3 r3 1         ; n = n-1

        lcw r2 sum          ; make recursive call
        cal r2 r2           ; sum(n-1), answer should be in r3

        pop r2              ; get n into r2
        add r3 r3 r2        ; r3 = n + sum(n-1)
done
        pop r2              ; get the return address
        jmp r2              ; go back to where we were called from
```

Notice symmetry of psh and pop

Calling sum

```
loa r3 r0

lcw r2 sum
cal r2 r2

sto r3 r0
hlt
```

r2	
r3	

sp (r1)

Stack

Calling sum

```
loa r3 r0

lcw r2 sum
cal r2 r2

sto r3 r0
hlt
```

r2	
r3	2

sp (r1)

Stack

Calling sum

```
loa r3 r0

lcw r2 sum
cal r2 r2

sto r3 r0
hlt
```

r2	
r3	2

sp (r1)

Stack

Calling sum

```
loa r3 r0

lcw r2 sum
cal r2 r2

sto r3 r0
hlt
```

r2	loc:sum
r3	2

sp (r1)

Stack

Calling sum

```
loa r3 r0

lcw r2 sum
cal r2 r2

sto r3 r0
hlt
```

r2	loc:sum
r3	2

sp (r1)

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal0
r3	2

sp (r1)

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal0
r3	2

← sp (r1)

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal0
r3	2

← sp (r1)

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal0
r3	2

← sp (r1)

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal0
r3	2

← sp (r1)

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal0
r3	2

Why psh r3?

← sp (r1)

2
loc:cal0
Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal0
r3	2

- We're about to make a function call
- The result of that call will go into r3 so we'll lose what's in there if we don't save it!
- **x + sum (x-1)**

← sp (r1)

2
loc:cal0
Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal0
r3	2

← sp (r1)

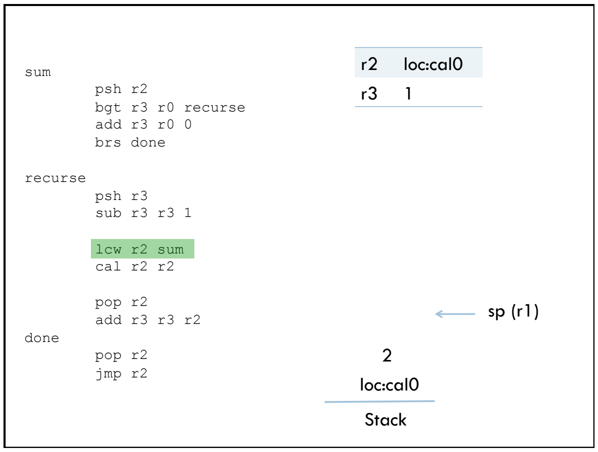
2
loc:cal0
Stack

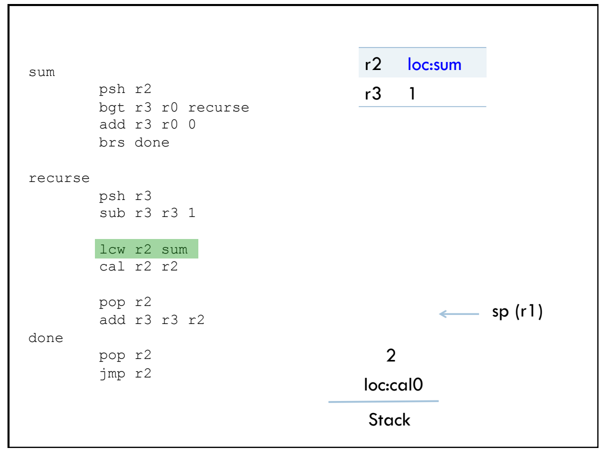
```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

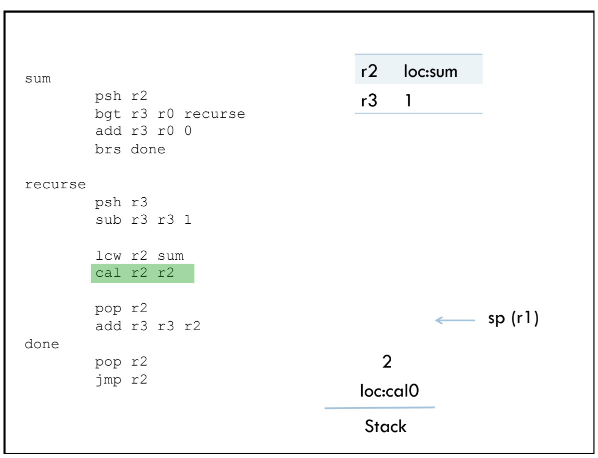
r2	loc:cal0
r3	1

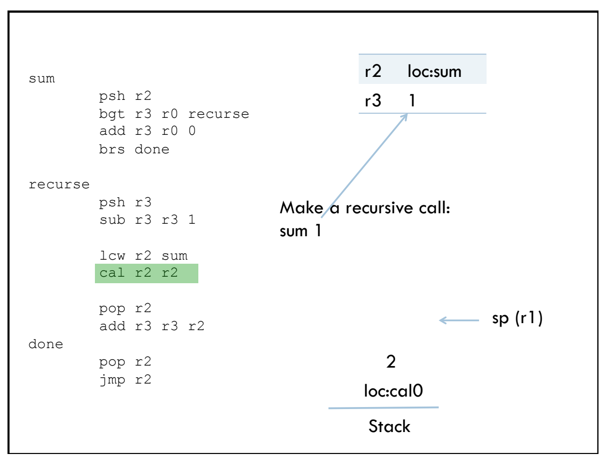
← sp (r1)

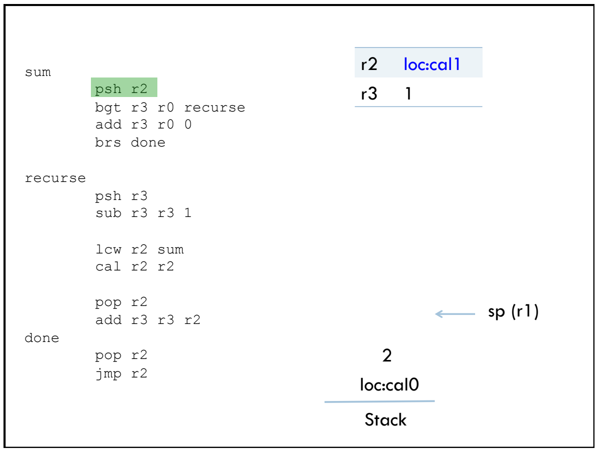
2
loc:cal0
Stack

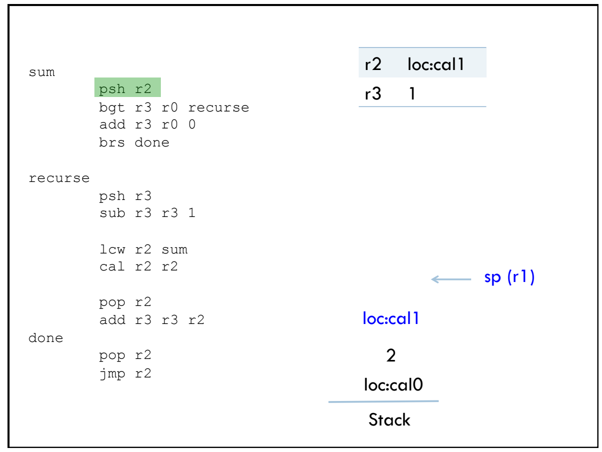

sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
r2 loc:cal0
r3 1
sp (r1)
2
loc:cal0
Stack

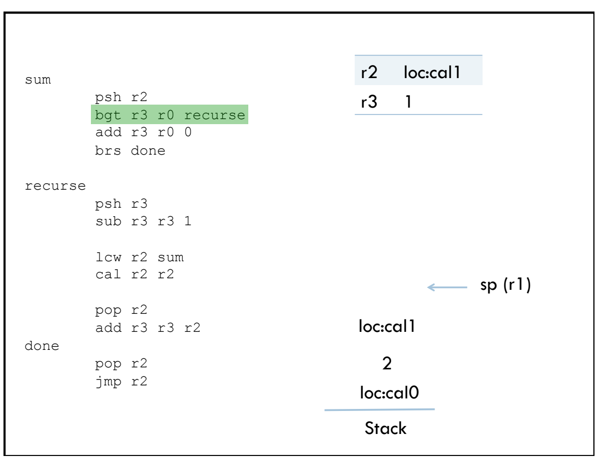

sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
r2 loc:sum
r3 1
sp (r1)
2
loc:cal0
Stack

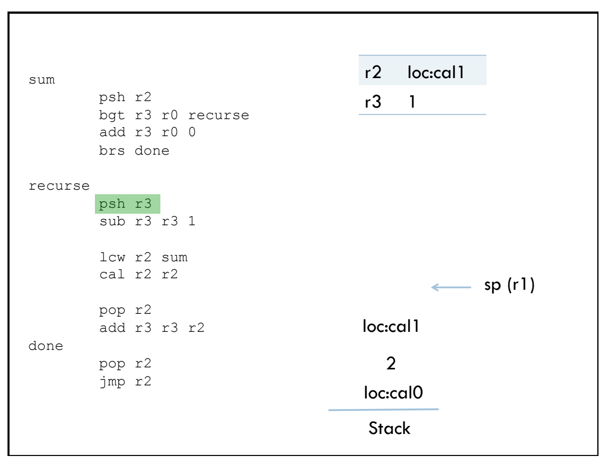

sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
r2 loc:sum
r3 1
sp (r1)
2
loc:cal0
Stack


sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
r2 loc:sum
r3 1
Make a recursive call:
sum 1
sp (r1)
2
loc:cal0
Stack


sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
r2 loc:call
r3 1
sp (r1)
2
loc:cal0
Stack


sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
r2 loc:cal1
r3 1
sp (r1)
loc:cal1
2
loc:cal0
Stack


sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
r2 loc:cal1
r3 1
sp (r1)
loc:cal1
2
loc:cal0
Stack


sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
r2 loc:cal1
r3 1
sp (r1)
loc:cal1
2
loc:cal0
Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	1

← sp (r1)

1

loc:cal1

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	1

← sp (r1)

1

loc:cal1

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	0

← sp (r1)

1

loc:cal1

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	0

← sp (r1)

1

loc:cal1

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2 loc:sum
r3 0

sp (r1)

1
loc:cal1
2
loc:cal0
Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2 loc:sum
r3 0

sp (r1)

1
loc:cal1
2
loc:cal0
Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2 loc:sum
r3 0

Make a recursive call:
sum 0

sp (r1)

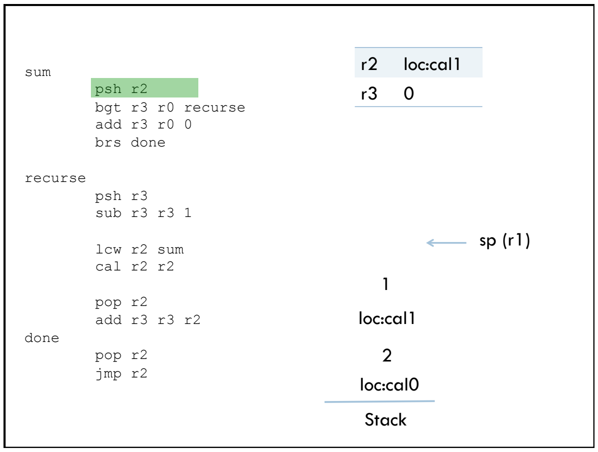
1
loc:cal1
2
loc:cal0
Stack

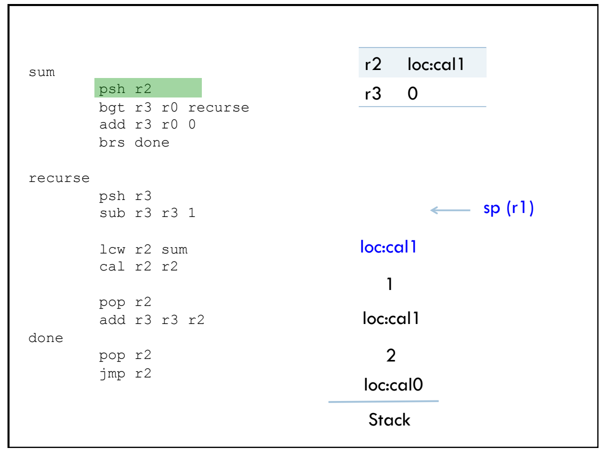
```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

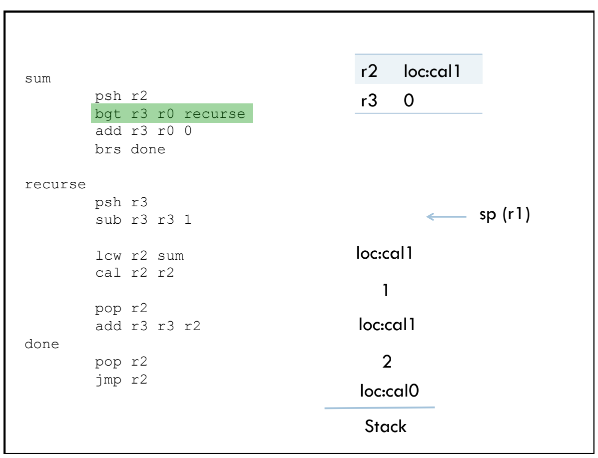
r2 loc:cal1
r3 0

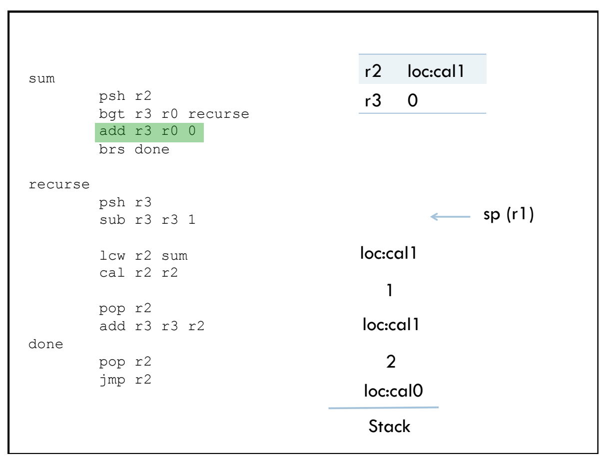
sp (r1)

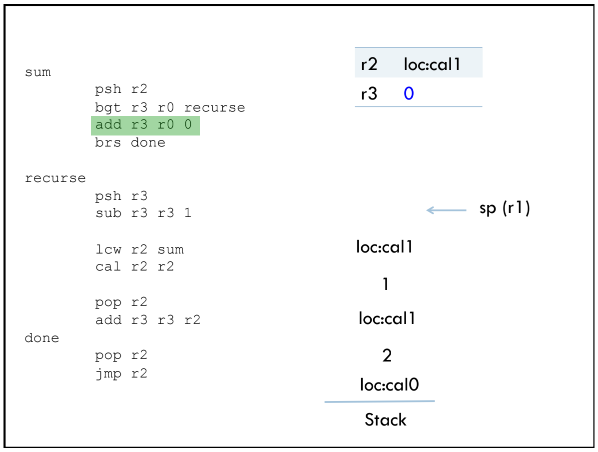
1
loc:cal1
2
loc:cal0
Stack

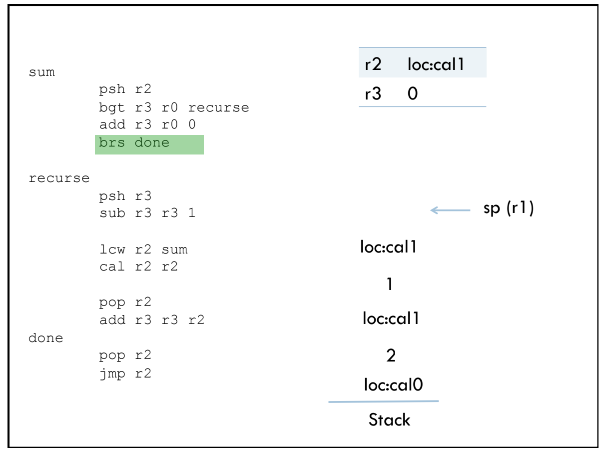

r2 loc:cal1
r3 0
sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
sp (r1)
1
loc:cal1
2
loc:cal0
Stack

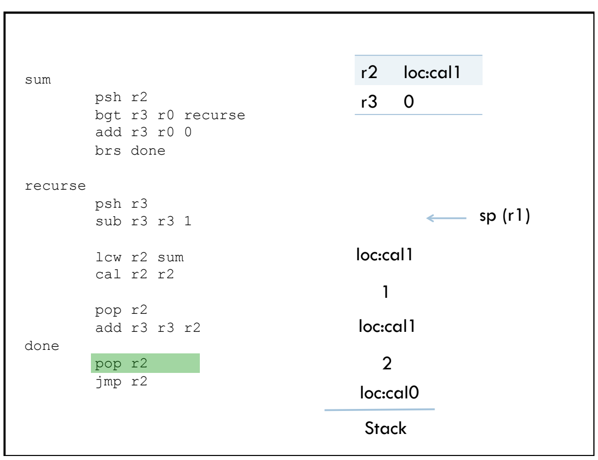

r2 loc:cal1
r3 0
sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
sp (r1)
loc:cal1
1
loc:cal1
2
loc:cal0
Stack

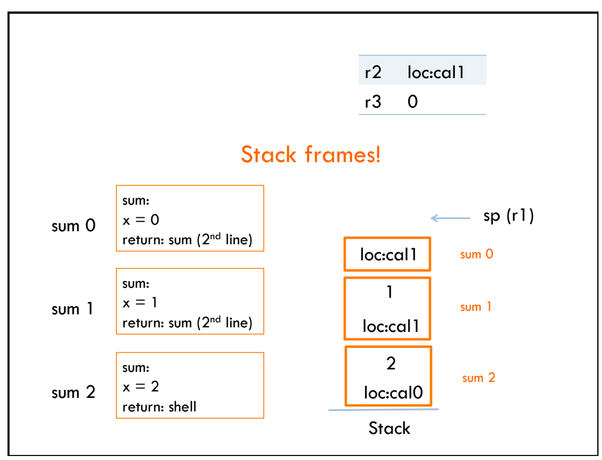

r2 loc:cal1
r3 0
sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
sp (r1)
loc:cal1
1
loc:cal1
2
loc:cal0
Stack


r2 loc:cal1
r3 0
sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
sp (r1)
loc:cal1
1
loc:cal1
2
loc:cal0
Stack


sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
r2 loc:cal1
r3 0
sp (r1)
loc:cal1
1
loc:cal1
2
loc:cal0
Stack


sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
r2 loc:cal1
r3 0
sp (r1)
loc:cal1
1
loc:cal1
2
loc:cal0
Stack


sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
lcw r2 sum
cal r2 r2
pop r2
add r3 r3 r2
done
pop r2
jmp r2
r2 loc:cal1
r3 0
sp (r1)
loc:cal1
1
loc:cal1
2
loc:cal0
Stack


r2 loc:cal1
r3 0
Stack frames!
sum 0
sum:
x = 0
return: sum (2nd line)
sum 1
sum:
x = 1
return: sum (2nd line)
sum 2
sum:
x = 2
return: shell
sp (r1)
loc:cal1
sum 0
1
loc:cal1
sum 1
2
loc:cal0
sum 2
Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	0

← sp (r1)

loc:cal1

1

loc:cal1

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	0

← sp (r1)

1

loc:cal1

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	0

← sp (r1)

1

loc:cal1

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	0

← sp (r1)

1

loc:cal1

2

loc:cal0

Returning with answer in r3

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	0

Why are we doing this?

← sp (r1)

1

loc:cal1

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	0

- Need to calculate x + sum (x-1)
- Saved x on the stack

← sp (r1)

1

loc:cal1

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	1
r3	0

← sp (r1)

loc:cal1

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	1
r3	0

← sp (r1)

loc:cal1

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	1
r3	1

← sp (r1)
loc:cal1
2
loc:cal0
Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	1
r3	1

← sp (r1)
loc:cal1
2
loc:cal0
Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	1

← sp (r1)
2
loc:cal0
Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	1

← sp (r1)
2
loc:cal0
Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	1

sp (r1)

2

loc:cal0

Stack

Returning with answer in r3

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal1
r3	1

sp (r1)

2

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	2
r3	1

sp (r1)

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	2
r3	1

sp (r1)

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	2
r3	3

sp (r1)

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	2
r3	3

sp (r1)

loc:cal0

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal0
r3	3

sp (r1)

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal0
r3	3

sp (r1)

Stack

```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

recurse
        psh r3
        sub r3 r3 1

        lcw r2 sum
        cal r2 r2

        pop r2
        add r3 r3 r2
done
        pop r2
        jmp r2
```

r2	loc:cal0
r3	3

sp (r1)

Returning with answer in r3

Stack

Calling sum

```
    loa r3 r0

    lcw r2 sum
    cal r2 r2

    sto r3 r0
    hlt
```

r2	loc:cal0
r3	3

sp (r1)

Stack

Calling sum

```
    loa r3 r0

    lcw r2 sum
    cal r2 r2

    sto r3 r0
    hlt
```

r2	loc:cal0
r3	3

Print the answer: 3!

sp (r1)

Stack

Calling sum

```
    loa r3 r0

    lcw r2 sum
    cal r2 r2

    sto r3 r0
    hlt
```

r2	loc:cal0
r3	3

sp (r1)

Stack

Calling sum

```
loa r3 r0

lcw r2 sum
cal r2 r2

sto r3 r0
hlt
```

r2	loc:cal0
r3	3

Notice that when we're all done, the stack is empty

sp (r1)

Stack

Real structure of CS52 program

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

      instruction1       ; comment
      instruction2       ; comment
      ...

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

Save address of highest end (highest address) of the stack in r1

Reserve 50 words for the stack

Structure of a single parameter function

```
fname
      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
```

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

Functions with multiple arguments

```
fname
      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
```

conventions:
- first argument is in r3
- r1 is off-limits since it's used for the stack pointer
- return value goes in r3

Functions with multiple arguments

```
fname
      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]

loa R_a R_b S: R_a = mem[R_b + S]

What does this operation do? What is the 4?

Functions with multiple arguments

```
fname
      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]

loa R_a R_b S: R_a = 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

```
fname
      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
```

- r1 is the stack pointer and points at the top (next) slot
- stacks grow towards smaller memory values
- r1+2 is then the top value of the stack
- r1+4 is the 2nd value of the stack

Another recursive example

```
int mystery(int a, int b){
   if( b <= 0 ){
     return 0
   else
     return a + mystery(a, b-1)
   }
}
```

What does this function do?

Recursion

```
int mystery(int a, int b){
   if( b <= 0 ){
     return 0
   else
     return a + mystery(a, b-1)
   }
}
```

Multiplication… a*b (assuming b is positive)

Note to future Dave from past Dave: write the function up on the board ☺

```
mult
       psh r2           ; save the return address
       loa r2 r1 4      ; get at the 2nd argument, b
                        ; a = r3, b = r2

       bgt r2 r0 else   ; r2 > 0, i.e. recursive case
       add r3 r0 0      ; return 0
       brs endif
else
       sub r2 r2 1      ; r2 = b-1

       psh r3           ; save first argument, a, on stack
                        ; (it's going to get overwritten by the return!)
       psh r2           ; add r2 as 2nd argument, r3 shouldn't have changed
       lcw r2 mult      ; call mult recursively
       cal r2 r2
       pop r0           ; pop 2nd argument off stack

       pop r2           ; pop 'a' into r2 off of the stack
       add r3 r3 r2     ; r3 = a + mult(a, b-1)
endif
       pop r2           ; get the return address
       jmp r2           ; return
```

Function startup

Base case

Recursive case

Recursive call

answer calculation

Function cleanup and return

```
mult
       psh r2           ; save the return address
       loa r2 r1 4      ; get at the 2nd argument, b
                        ; a = r3, b = r2

       bgt r2 r0 else   ; r2 > 0, i.e. recursive case
       add r3 r0 0      ; return 0
       brs endif
else
       sub r2 r2 1      ; r2 = b-1

       psh r3           ; save first argument, a, on stack
                        ; (it's going to get overwritten by the return!)
       psh r2           ; add r2 as 2nd argument, r3 shouldn't have changed
       lcw r2 mult      ; call mult recursively
       cal r2 r2
       pop r0           ; pop 2nd argument off stack

       pop r2           ; pop 'a' into r2 off of the stack
       add r3 r3 r2     ; r3 = a + mult(a, b-1)
endif
       pop r2           ; get the return address
       jmp r2           ; return
```

Function startup

if(b <= 0)
return 0

mystery(a, b-1)

a + mystery(a, b-1)

Function cleanup and return

```
mult
       psh r2           ; save the return address
       loa r2 r1 4      ; get at the 2nd argument, b
                        ; a = r3, b = r2

       bgt r2 r0 else   ; r2 > 0, i.e. recursive case
       add r3 r0 0      ; return 0
       brs endif
else
       sub r2 r2 1      ; r2 = a-1

       psh r3           ; save first argument, a, on stack
                        ; (it's going to get overwritten by the return!)
       psh r2           ; add r2 as 2nd argument, r3 shouldn't have changed
       lcw r2 mult      ; call mult recursively
       cal r2 r2
       pop r0           ; pop 2nd argument off stack

       pop r2           ; load a into r2 off of the stack
       add r3 r3 r2     ; r3 = a + mult(a, b-1)
endif
       pop r2           ; get the return address
       jmp r2           ; return
```

Notice symmetry of psh and pop

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2
r3

sp (r1)

Stack

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2
r3

sp (r1)

Stack

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	
r3	6

sp (r1)

Stack

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	
r3	6

sp (r1)

Stack

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	2
r3	6

sp (r1)

Stack

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	2
r3	6

sp (r1)

Stack

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	2
r3	6

sp (r1)

2

Stack

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	2
r3	6

sp (r1)

2

Stack

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	loc: mult
r3	6

sp (r1)

2

Stack

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	loc: mult
r3	6

sp (r1)

2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal0
r3	6

sp (r1)

2

Stack

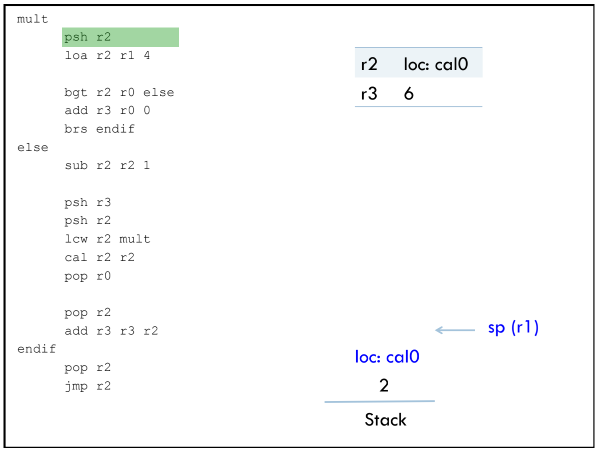
```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

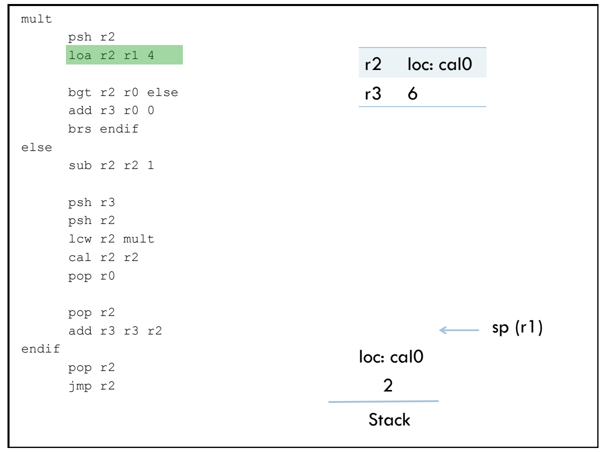
r2	loc: cal0
r3	6

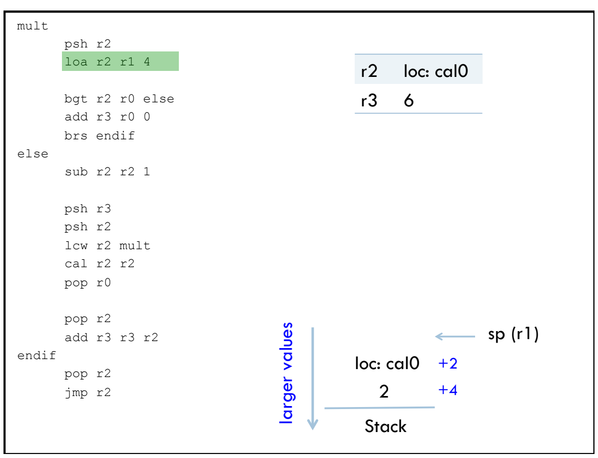
sp (r1)

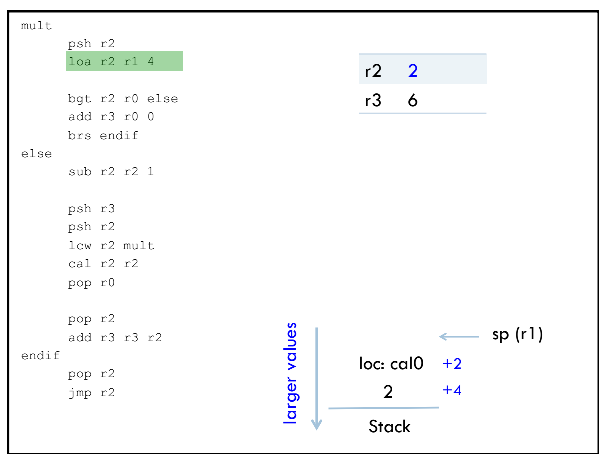
2

Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 loc: cal0
r3 6
sp (r1)
loc: cal0
2
Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 loc: cal0
r3 6
sp (r1)
loc: cal0
2
Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 loc: cal0
r3 6
larger values
sp (r1)
loc: cal0 +2
2 +4
Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 2
r3 6
larger values
sp (r1)
loc: cal0 +2
2 +4
Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	2
r3	6

sp (r1)

loc: cal0

2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	2
r3	6

sp (r1)

loc: cal0

2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	1
r3	6

sp (r1)

loc: cal0

2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	1
r3	6

sp (r1)

loc: cal0

2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	1
r3	6

Why psh r3?

← sp (r1)

6

loc: cal0

2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	1
r3	6

- We're about to make a function call
- The result of that call will go into r3 so we'll lose what's in there if we don't save it!

← sp (r1)

6

loc: cal0

2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	1
r3	6

← sp (r1)

6

loc: cal0

2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	1
r3	6

← sp (r1)

1

6

loc: cal0

2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 1
r3 6

sp (r1)

1
6
loc: cal0
2
Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 loc: mult
r3 6

sp (r1)

1
6
loc: cal0
2
Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 loc: mult
r3 6

sp (r1)

1
6
loc: cal0
2
Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 loc: cal1
r3 6

sp (r1)

1
6
loc: cal0
2
Stack

```
mult
        psh r2
        loa r2 r1 4

        bgt r2 r0 else
        add r3 r0 0
        brs endif
else
        sub r2 r2 1

        psh r3
        psh r2
        lcw r2 mult
        cal r2 r2
        pop r0

        pop r2
        add r3 r3 r2
endif
        pop r2
        jmp r2
```

r2 loc: cal1
r3 6

← sp (r1)

1
6
loc: cal0
2

Stack

```
mult
        psh r2
        loa r2 r1 4

        bgt r2 r0 else
        add r3 r0 0
        brs endif
else
        sub r2 r2 1

        psh r3
        psh r2
        lcw r2 mult
        cal r2 r2
        pop r0

        pop r2
        add r3 r3 r2
endif
        pop r2
        jmp r2
```

r2 loc: cal1
r3 6

← sp (r1)

loc: cal1
1
6
loc: cal0
2

Stack

```
mult
        psh r2
        loa r2 r1 4

        bgt r2 r0 else
        add r3 r0 0
        brs endif
else
        sub r2 r2 1

        psh r3
        psh r2
        lcw r2 mult
        cal r2 r2
        pop r0

        pop r2
        add r3 r3 r2
endif
        pop r2
        jmp r2
```

r2 loc: cal1
r3 6

← sp (r1)

loc: cal1
1
6
loc: cal0
2

Stack

```
mult
        psh r2
        loa r2 r1 4

        bgt r2 r0 else
        add r3 r0 0
        brs endif
else
        sub r2 r2 1

        psh r3
        psh r2
        lcw r2 mult
        cal r2 r2
        pop r0

        pop r2
        add r3 r3 r2
endif
        pop r2
        jmp r2
```

r2 1
r3 6

← sp (r1)

loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 1
r3 6

← sp (r1)

loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 1
r3 6

← sp (r1)

loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 0
r3 6

← sp (r1)

loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 0
r3 6

← sp (r1)

loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	0
r3	6

← sp (r1)

6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	0
r3	6

← sp (r1)

6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	0
r3	6

← sp (r1)

0
6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	0
r3	6

← sp (r1)

0
6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: mult
r3	6

sp (r1)

0
6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: mult
r3	6

sp (r1)

0
6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal1
r3	6

sp (r1)

0
6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal1
r3	6

sp (r1)

0
6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal1
r3	6

sp (r1)

loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal1
r3	6

sp (r1)

loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	0
r3	6

sp (r1)

loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2

Stack

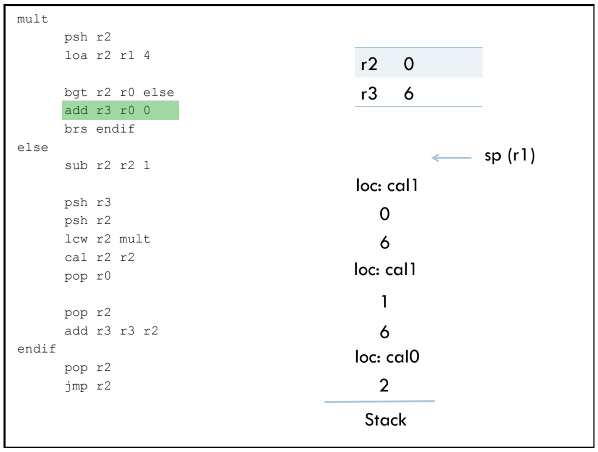
```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

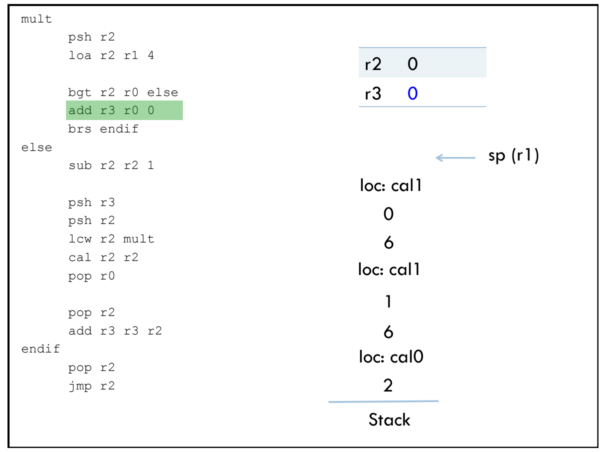
r2	0
r3	6

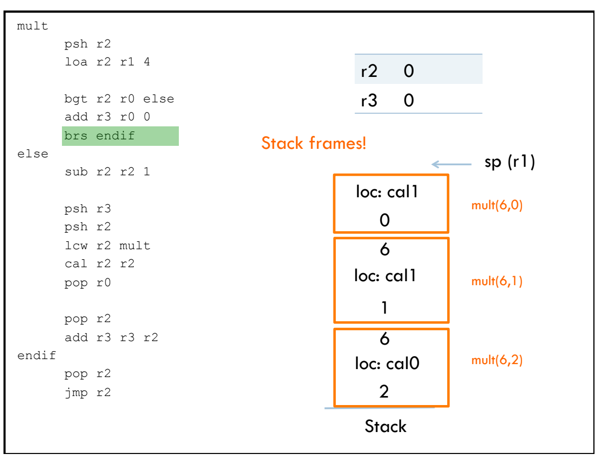
sp (r1)

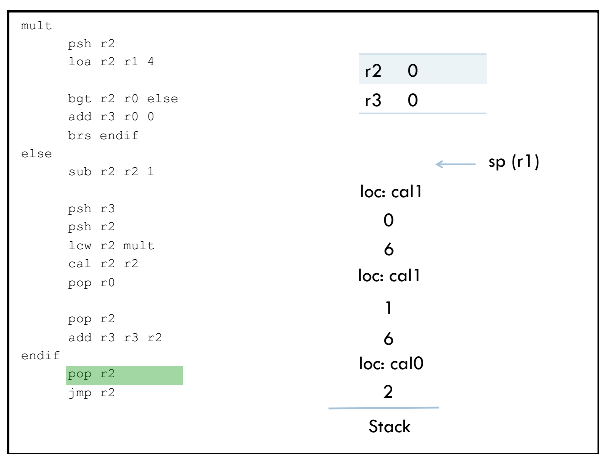
loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2

Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 0
r3 6
sp (r1)
loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2
Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 0
r3 0
sp (r1)
loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2
Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 0
r3 0
Stack frames!
sp (r1)
loc: cal1
0
mult(6,0)
6
loc: cal1
1
mult(6,1)
6
loc: cal0
2
mult(6,2)
Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 0
r3 0
sp (r1)
loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2
Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal1
r3	0

← sp (r1)

0
6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal1
r3	0

← sp (r1)

0
6
loc: cal1
1
6
loc: cal0
2

Stack

Returning with answer in r3

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal1
r3	0

← sp (r1)

0
6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal1
r3	0

← sp (r1)

6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal1
r3	0

← sp (r1)

6
loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	6
r3	0

← sp (r1)

loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	6
r3	0

← sp (r1)

loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	6
r3	6

← sp (r1)

loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	6
r3	6

← sp (r1)

loc: cal1
1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal1
r3	6

← sp (r1)

1
6
loc: cal0
2

Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2	loc: cal1
r3	6

← sp (r1)

1
6
loc: cal0
2

Stack

Returning with answer in r3

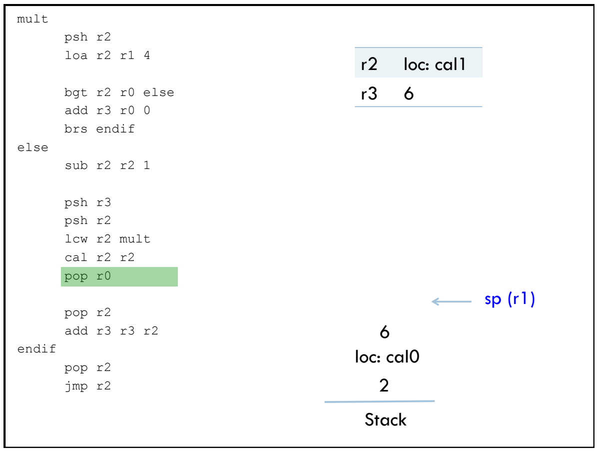
```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

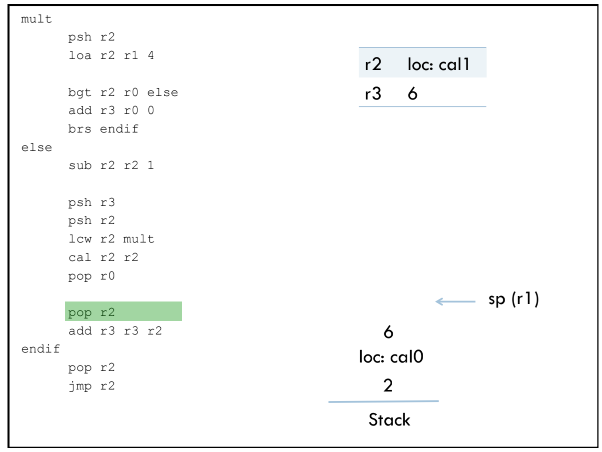
r2	loc: cal1
r3	6

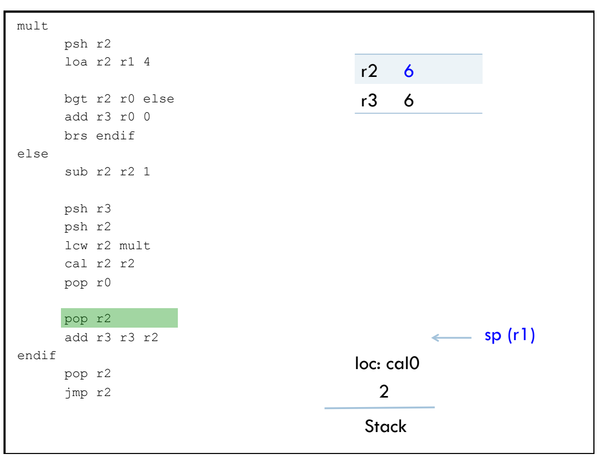
← sp (r1)

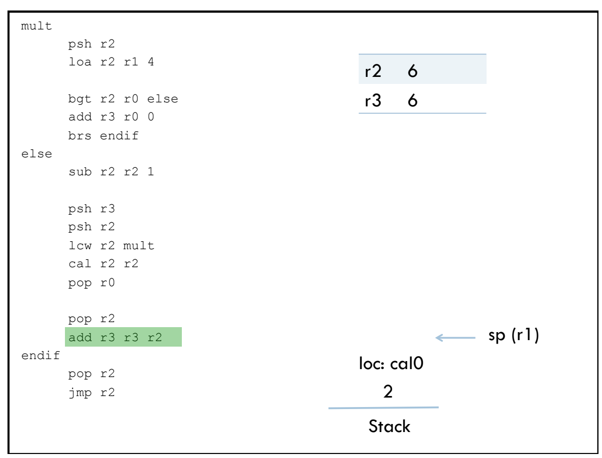
1
6
loc: cal0
2

Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
6
loc: cal0
2
Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
6
loc: cal0
2
Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 6
r3 6
sp (r1)
loc: cal0
2
Stack


mult
psh r2
loa r2 r1 4
bgt r2 r0 else
add r3 r0 0
brs endif
else
sub r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
pop r2
add r3 r3 r2
endif
pop r2
jmp r2
r2 6
r3 6
sp (r1)
loc: cal0
2
Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 6
r3 12

sp (r1)
loc: cal0
2
Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 6
r3 12

sp (r1)
loc: cal0
2
Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 loc: cal0
r3 12

sp (r1)
2
Stack

```
mult
      psh r2
      loa r2 r1 4

      bgt r2 r0 else
      add r3 r0 0
      brs endif
else
      sub r2 r2 1

      psh r3
      psh r2
      lcw r2 mult
      cal r2 r2
      pop r0

      pop r2
      add r3 r3 r2
endif
      pop r2
      jmp r2
```

r2 loc: cal0
r3 12

sp (r1)
2
Stack

Returning with answer in r3

Calling mult

r2	loc: cal0
r3	12

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

sp (r1)

2

Stack

Calling mult

r2	loc: cal0
r3	12

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

sp (r1)

Stack

Calling mult

r2	loc: cal0
r3	12

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

sp (r1)

Stack

Calling mult

r2	loc: cal0
r3	12

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

Print the answer: 12!

sp (r1)

Stack

Calling mult

r2	loc: cal0
r3	12

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

Print the answer: 12!

sp (r1)

Stack

multiply the easy way

look at mult_easy.a52 code

we can import libraries (really just functions in other files) using the “inc” command

the included files must be in the same directory as the .a52 file running

A final aside

10101_2

10_2

11_2

1_2

1010_2

$101010010010010010000111111101001_2$

Which of these binary numbers is even?

A quick aside

$10101_2 = 21$

$10_2 = 2$

$11_2 = 3$

$1_2 = 1$

$1010_2 = 10$

$101010010010010010000111111101001_2 =$ 5,675,487,209

Is there an easier way to tell than just calculating the value?

A quick aside

The last digit represents the 2^0 = 1s digit

All other digits represent even values since they are powers of 2

Therefore:

- If the rightmost digit is 1 = odd number
- If the rightmost digit is 0 = even number

Bitwise logical operators

instruction name arguments

```
add ⎫
sub ⎪
and ⎬ RRR or RRS
orr ⎪
xor ⎭
```

Perform **and, or** and **xor** *per bit of the number*

Bitwise logical operators

add r2 r0 3
add r3 r0 6
and r3 r3 r2

Bitwise logical operators

add r2 r0 3
add r3 r0 6
and r3 r3 r2

What are r2 and r3 in binary after these instructions?

Bitwise logical operators

add r2 r0 3
add r3 r0 6
and r3 r3 r2

r2: 11
r3: 110

Bitwise logical operators

add r2 r0 3
add r3 r0 6
and r3 r3 r2

Perform **and, or** and **xor** *per bit of the number*

r2: 11
r3: 110

Bitwise logical operators

add r2 r0 3
add r3 r0 6
and r3 r3 r2

Perform **and, or** and **xor** *per bit of the number*

r2: 11
r3: 110

Bitwise logical operators

add r2 r0 3
add r3 r0 6
and r3 r3 r2

Perform **and, or** and **xor** *per bit of the number*

r2: 11
r3: 110

0

Bitwise logical operators

add r2 r0 3
add r3 r0 6
and r3 r3 r2

Perform **and, or** and **xor** *per bit of the number*

```
r2:    11
r3:   110
-----------
        0
```

Bitwise logical operators

add r2 r0 3
add r3 r0 6
and r3 r3 r2

Perform **and, or** and **xor** *per bit of the number*

```
r2:    11
r3:   110
-----------
       10
```

Bitwise logical operators

add r2 r0 3
add r3 r0 6
and r3 r3 r2

Perform **and, or** and **xor** *per bit of the number*

```
r2:    11
r3:   110
-----------
       10
```

Bitwise logical operators

add r2 r0 3
add r3 r0 6
and r3 r3 r2

Perform **and, or** and **xor** *per bit of the number*

```
r2:   011
r3:   110
-----------
       10
```

Bitwise logical operators

add r2 r0 3
add r3 r0 6
and r3 r3 r2

Perform **and, or** and **xor** *per bit of the number*

r2: 011
r3: 110

010

CS52 programming advice

1. Match your psh and pops
2. Follow the register conventions
3. Develop code incrementally
4. Debugging: write out stack, registers, etc. on paper and compare against system execution

Examples from this lecture

http://www.cs.pomona.edu/~dkauchak/classes/cs52/examples/cs41b/

Another example

I didn't have time to cover the next example in class, but left it in the notes as another example of a function that takes two parameters

Another example

```
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

What does this code do?

Another example

```
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

max, as a function!

Calling max

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 max
cal r2 r2
pop r0

sto r3 r0
hlt
```

Anything different?

Calling max

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 max
cal r2 r2
pop r0

sto r3 r0
hlt
```

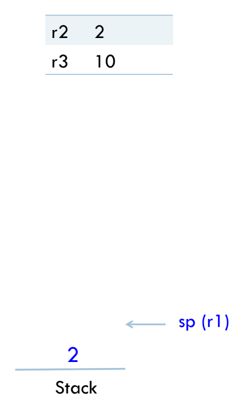
For the second argument, psh it on the stack

```
    loa r3 r0
    loa r2 r0

    psh r2
    lcw r2 max
    cal r2 r2
    pop r2

    sto r3 r0
    hlt
max
    psh r2
    loa r2 r1 4

    bge r3 r2 endif
    add r3 r2 0
endif
    pop r2
    jmp r2
```


```
    loa r3 r0
    loa r2 r0

    psh r2
    lcw r2 max
    cal r2 r2
    pop r2

    sto r3 r0
    hlt
max
    psh r2
    loa r2 r1 4

    bge r3 r2 endif
    add r3 r2 0
endif
    pop r2
    jmp r2
```


```
    loa r3 r0
    loa r2 r0

    psh r2
    lcw r2 max
    cal r2 r2
    pop r2

    sto r3 r0
    hlt
max
    psh r2
    loa r2 r1 4

    bge r3 r2 endif
    add r3 r2 0
endif
    pop r2
    jmp r2
```


```
    loa r3 r0
    loa r2 r0

    psh r2
    lcw r2 max
    cal r2 r2
    pop r2

    sto r3 r0
    hlt
max
    psh r2
    loa r2 r1 4

    bge r3 r2 endif
    add r3 r2 0
endif
    pop r2
    jmp r2
```



```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

r2	2
r3	10

← sp (r1)

2

Stack

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

r2	max
r3	10

Notice that we overwrote the value in r2

If we hadn't saved it on the stack, it would have been lost

← sp (r1)

2

Stack

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

r2	max
r3	10

← sp (r1)

2

Stack

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

r2	loc: cal
r3	10

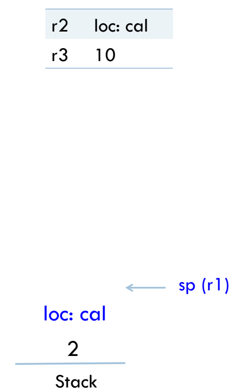
← sp (r1)

2

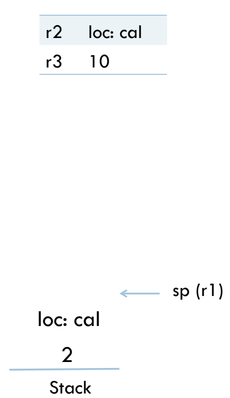
Stack

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

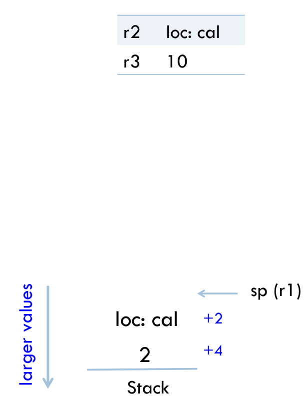

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

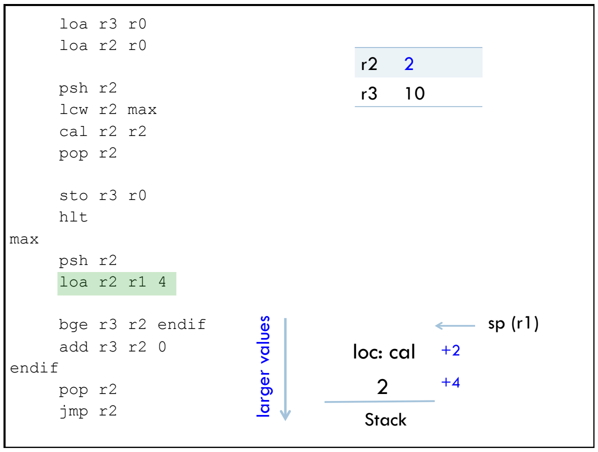


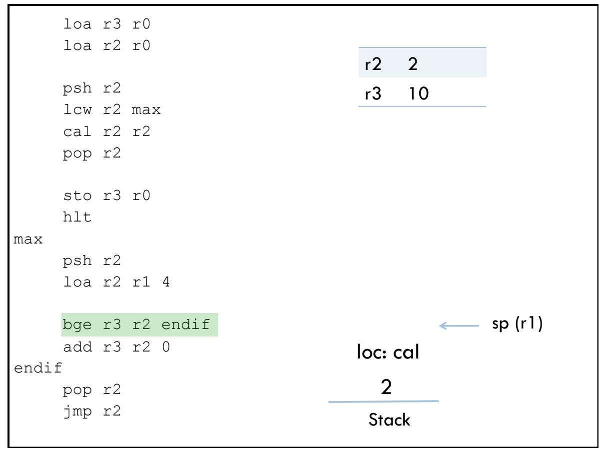

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

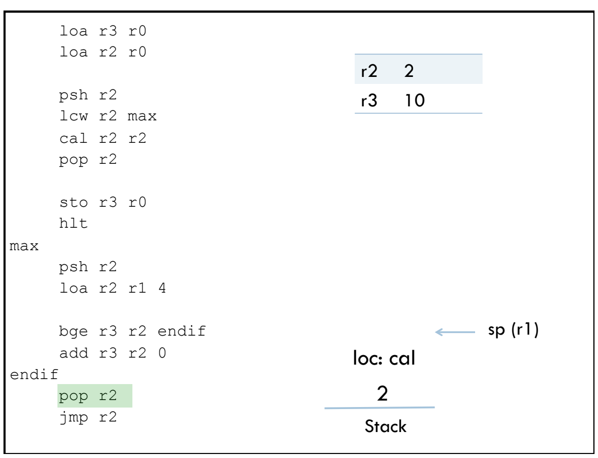


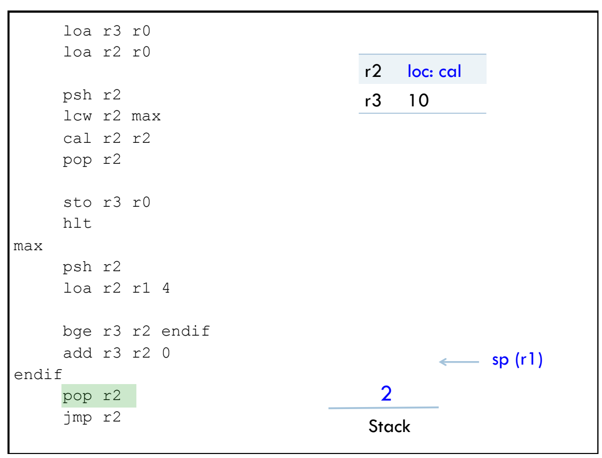

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```




loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r3 r0
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
add r3 r2 0
endif
pop r2
jmp r2
r2 2
r3 10
larger values
sp (r1)
loc: cal
+2
2
+4
Stack


loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r3 r0
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
add r3 r2 0
endif
pop r2
jmp r2
r2 2
r3 10
sp (r1)
loc: cal
2
Stack


loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r3 r0
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
add r3 r2 0
endif
pop r2
jmp r2
r2 2
r3 10
sp (r1)
loc: cal
2
Stack


loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r3 r0
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
add r3 r2 0
endif
pop r2
jmp r2
r2 loc: cal
r3 10
sp (r1)
2
Stack

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

r2	loc: cal
r3	10

sp (r1)

2

Stack

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

r2	loc: cal
r3	10

sp (r1)

2

Stack

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

r2	2
r3	10

sp (r1)

Stack

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

r2	2
r3	10

sp (r1)

Stack

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

r2	2
r3	10

10!

sp (r1)

Stack

```
     loa r3 r0
     loa r2 r0

     psh r2
     lcw r2 max
     cal r2 r2
     pop r2

     sto r3 r0
     hlt
max
     psh r2
     loa r2 r1 4

     bge r3 r2 endif
     add r3 r2 0
endif
     pop r2
     jmp r2
```

r2	2
r3	10

sp (r1)

Stack