CS52 MACHINE: RECURSION

David Kauchak
CS 52 – Fall 2022

1

Admin

Grading

2

Getting involved in the department

Join the CS slack: https://tinyurl.com/PomonaCSSlack

Get on the colloquium mailing list:

- Go to: https://listserv.pomona.edu/scripts/wa.exe?A0=CSCOLLOQ
- Click the small button in the upper right and select "Subscribe or Unsubscribe"

Questions… reach out to the liaisons: liaisons@cs.pomona.edu

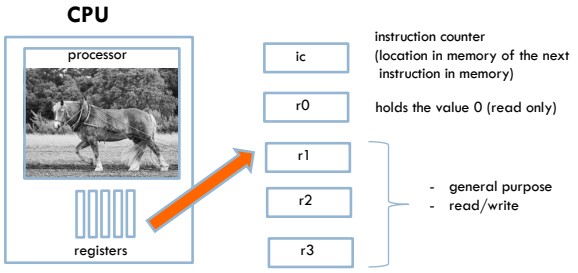
3

Examples from this lecture

https://cs.pomona.edu/classes/cs54/examples/cs52machine/

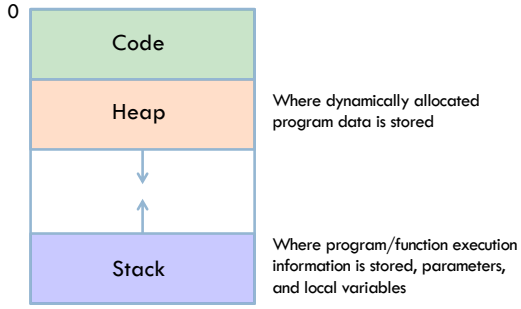
4

CS52 machine



5

Memory layout



6

Stack frame

Key unit for keeping track of a function call

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

7

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

8

Real structure of CS52 program

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

       instruction1        ; comment
       instruction2        ; comment
       ...
       hlt

;
; 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

9

Revisit increment example

10

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 ☺

11

```
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 x 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 x into r2
       add r3 r3 r2        ; r3 = x + 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 case

recursive call

answer calculation

Function cleanup and return

12

```
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
                                  Notice symmetry of psh and pop
recurse
        psh r3          ; save x 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 x into r2
        add r3 r3 r2    ; r3 = x + sum (x-1)
done
        pop r2          ; get the return address
        jmp r2          ; go back to where we were called from
```

13

Calling sum

```
loa r3 r0

lcw r2 sum
cal r2 r2

sto r3 r0
hlt
```

r2
r3

sp (r1) ←

Stack

14

Calling sum

```
loa r3 r0

lcw r2 sum
cal r2 r2

sto r3 r0
hlt
```

r2
r3 2

sp (r1) ←

Stack

15

Calling sum

```
loa r3 r0

lcw r2 sum
cal r2 r2

sto r3 r0
hlt
```

r2
r3 2

sp (r1) ←

Stack

16

Calling sum

```
    loa r3 r0

    lcw r2 sum
    cal r2 r2

    sto r3 r0
    hlt
```

r2	loc:sum
r3	2

sp (r1) ←

Stack

17

Calling sum

```
    loa r3 r0

    lcw r2 sum
    cal r2 r2

    sto r3 r0
    hlt
```

r2	loc:sum
r3	2

sp (r1) ←

Stack

18

```
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

19

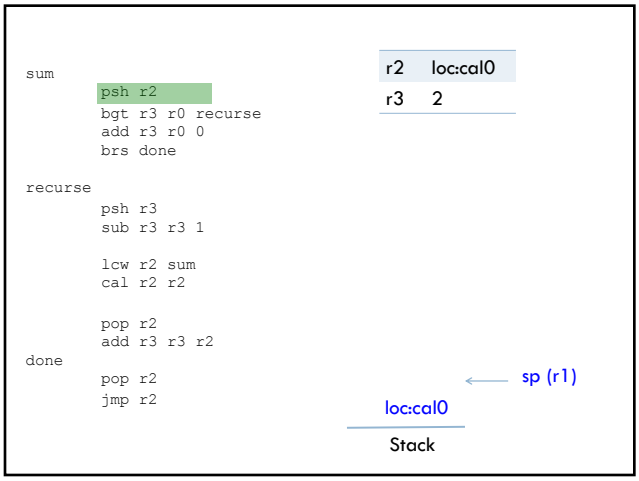
```
sum
        psh r2
        bgt r3 r0 recurse
        add r3 r0 0
        brs done

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

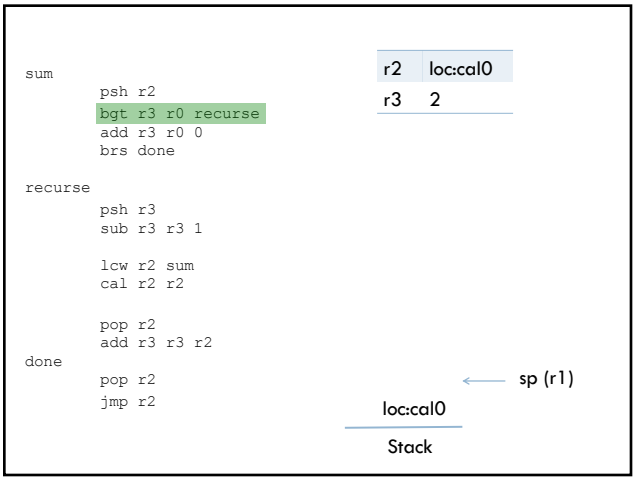
sp (r1) ←

Stack

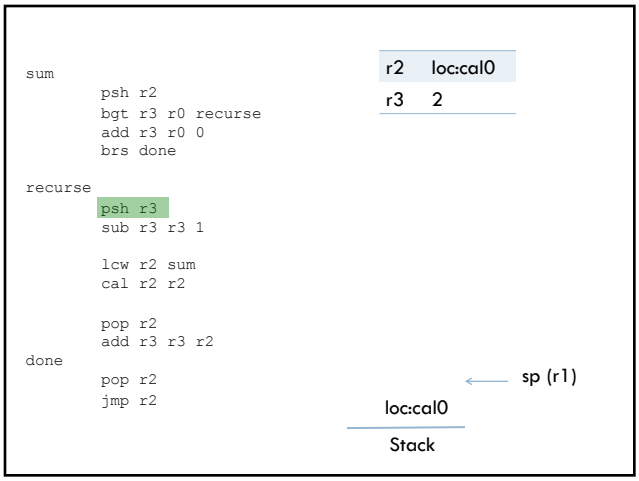
20


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

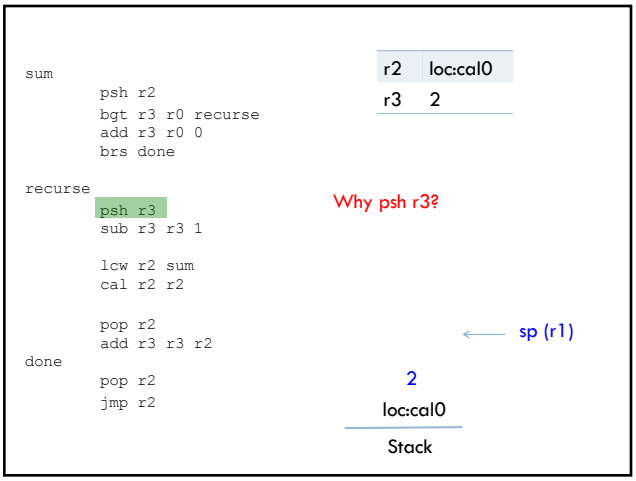
21


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

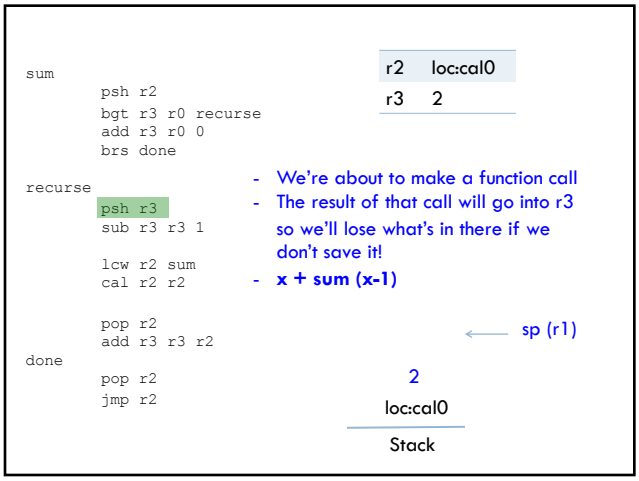
22


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

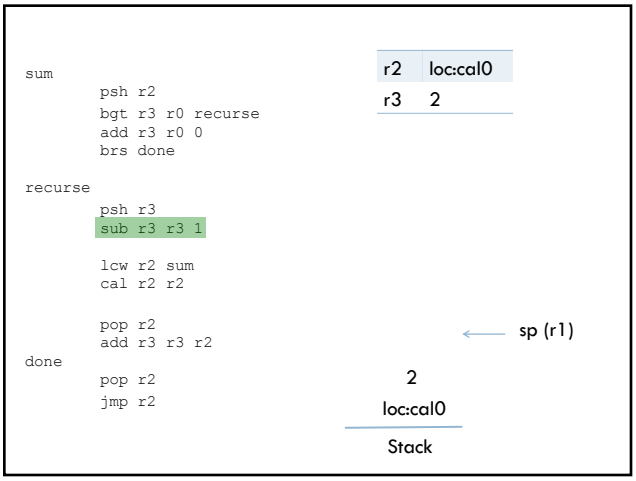
23


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

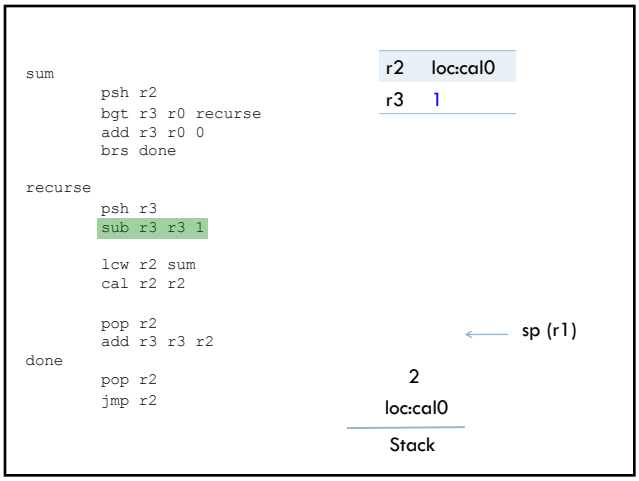
24


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

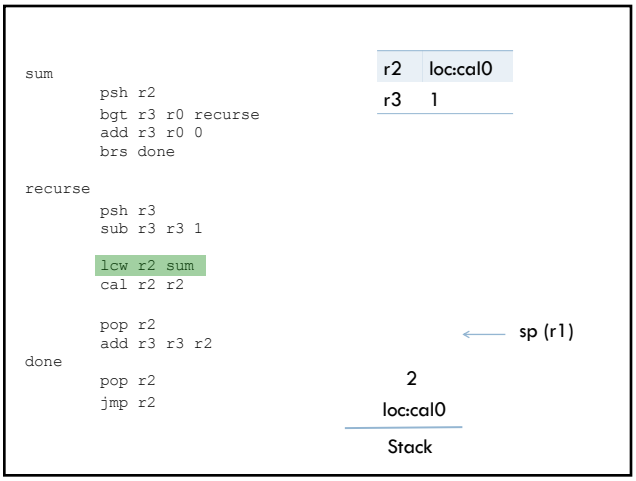
25


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

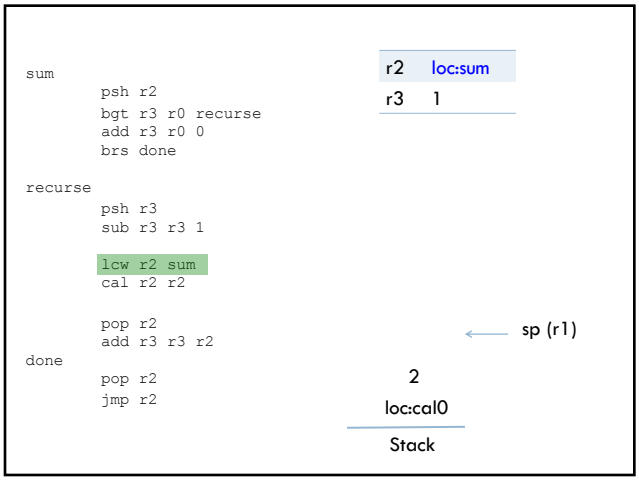
26


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

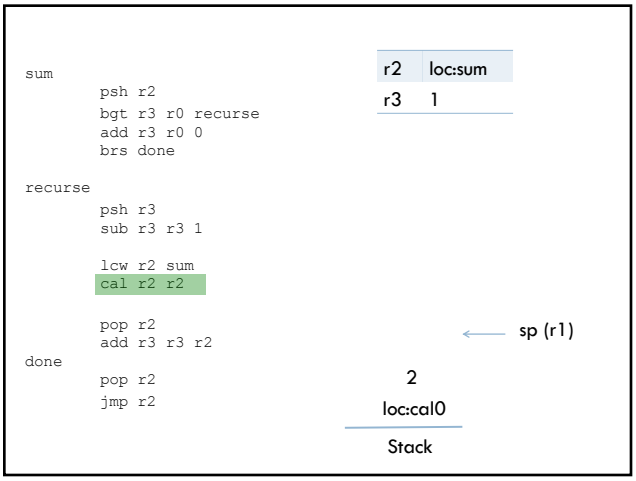
27


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

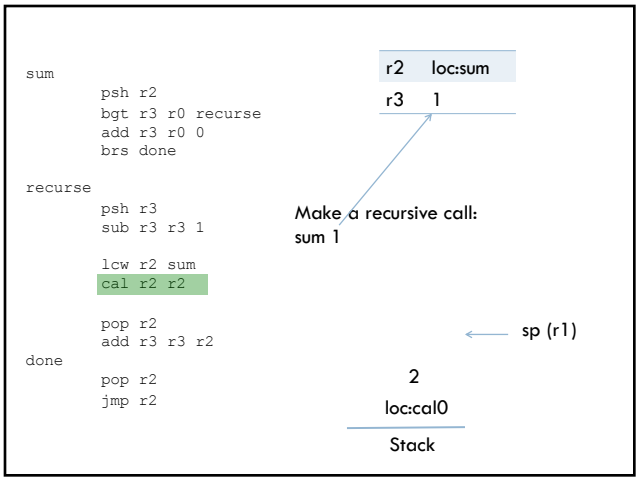
28


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

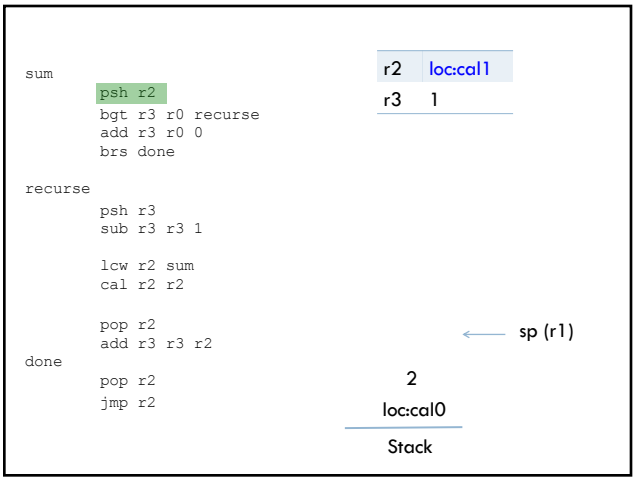
29


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

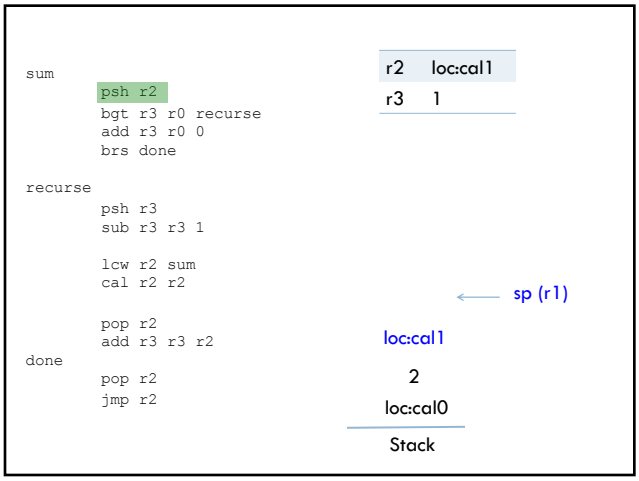
30


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

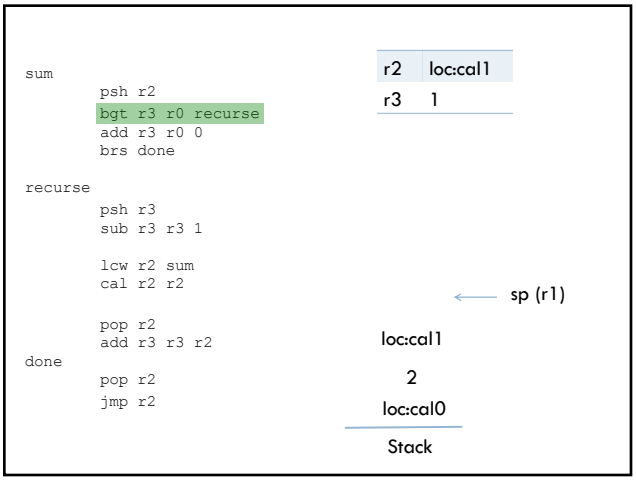
31


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

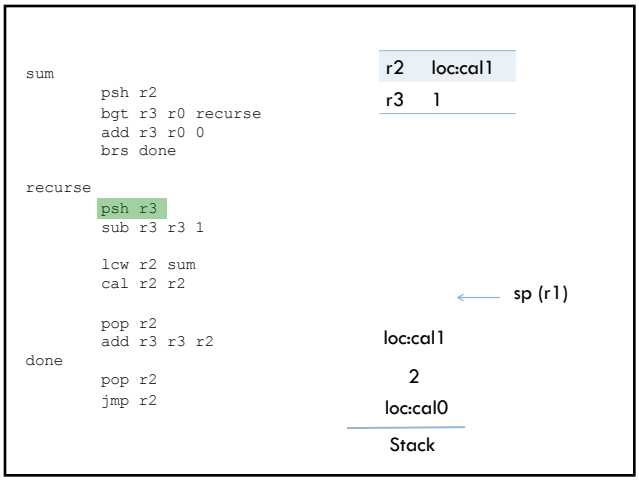
32


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

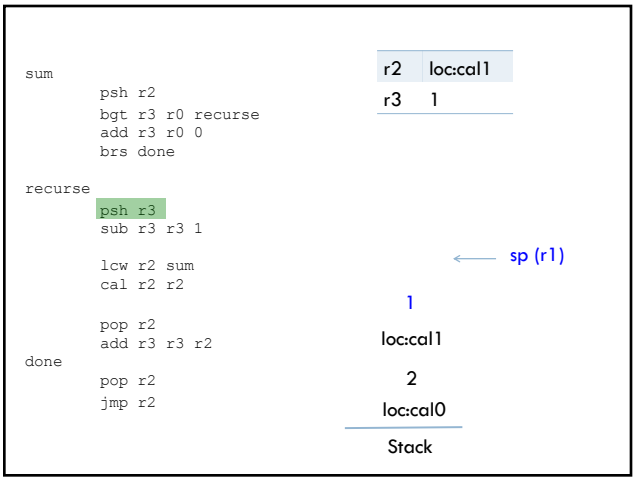
33


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

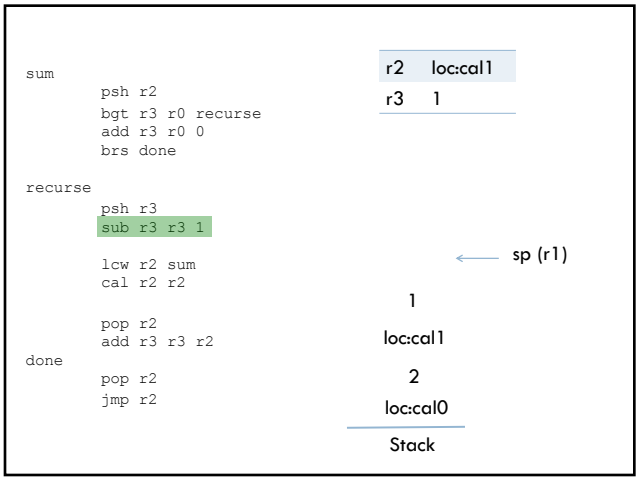
34


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

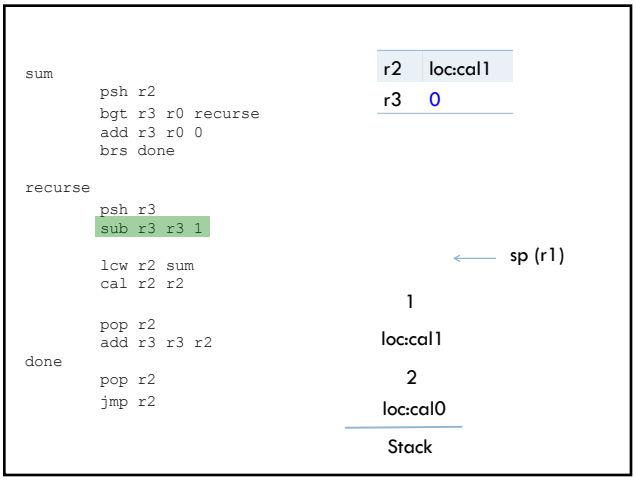
35


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

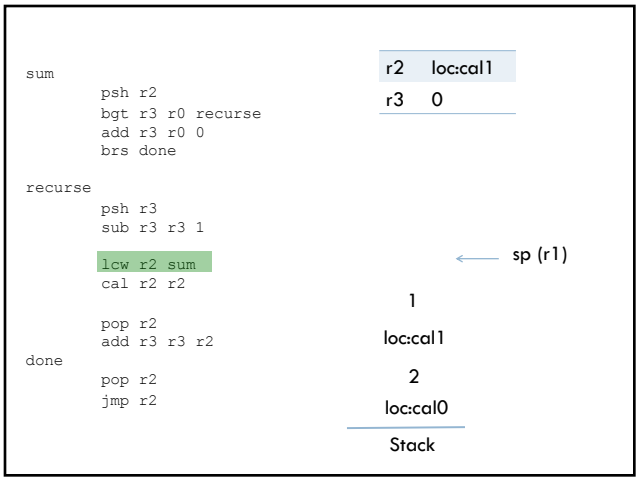
36


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

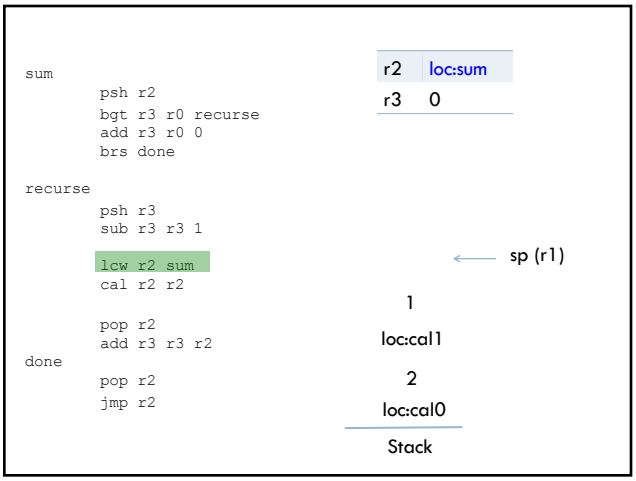
37


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

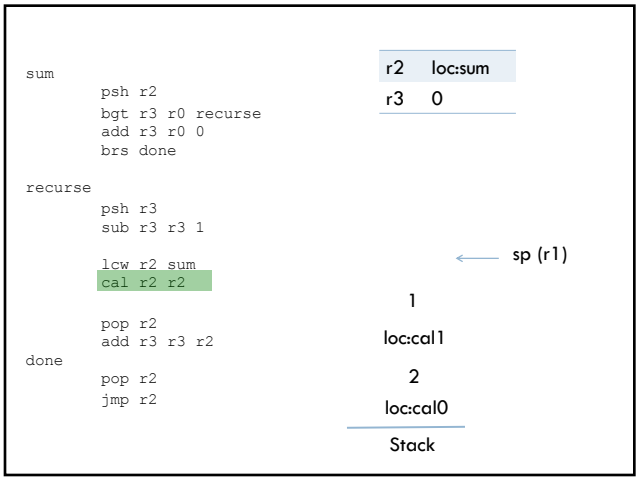
38


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

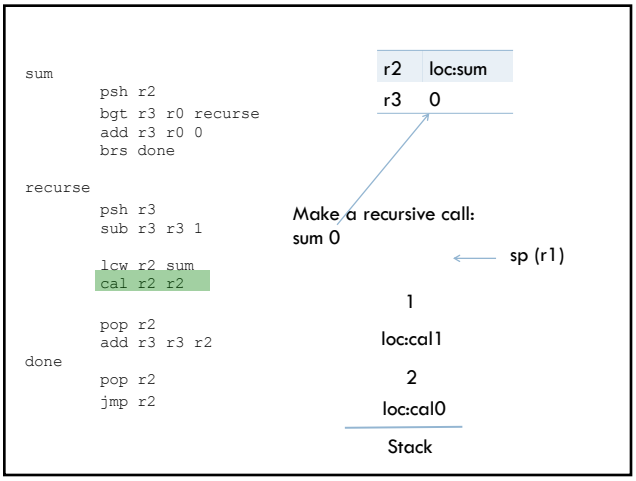
39


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

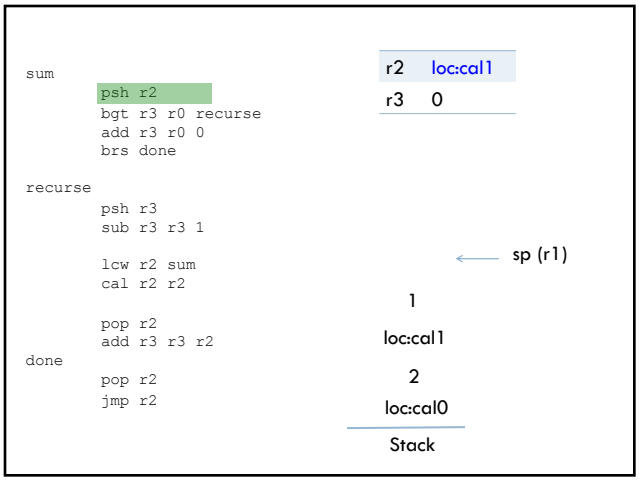
40


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

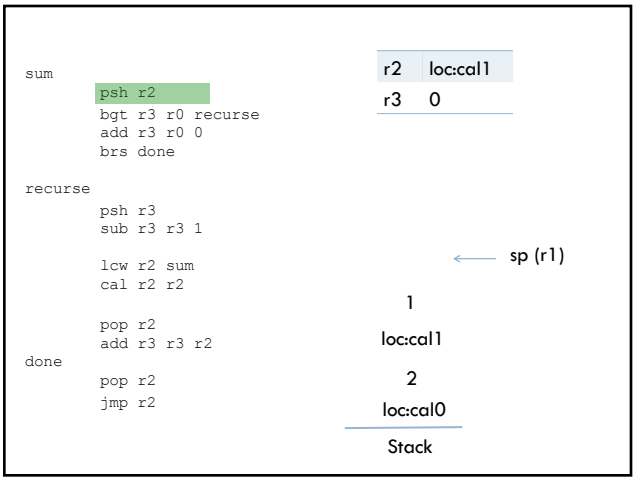
41


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

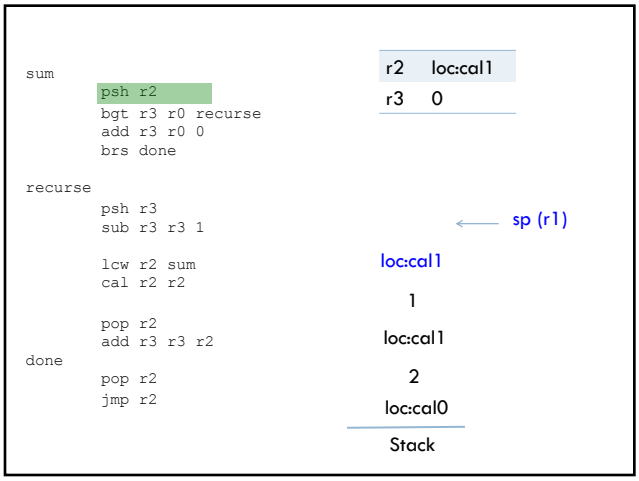
42


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

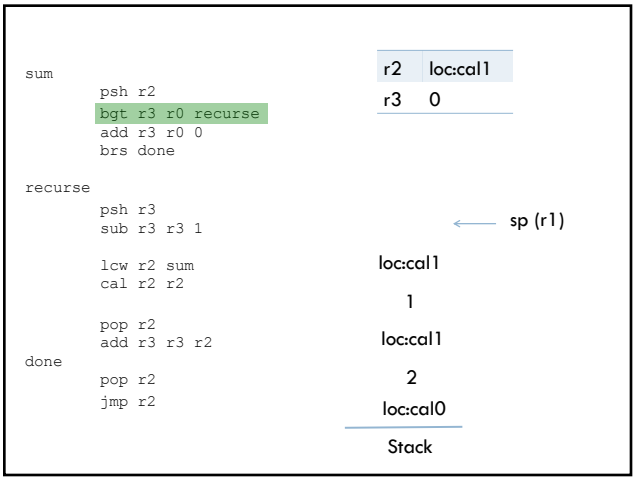
43


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

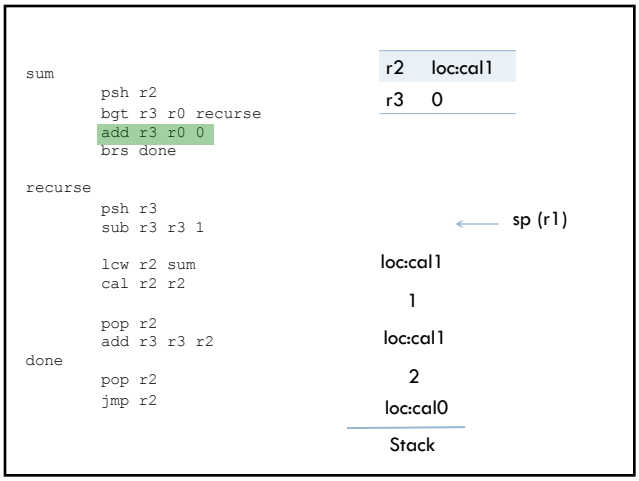
44


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

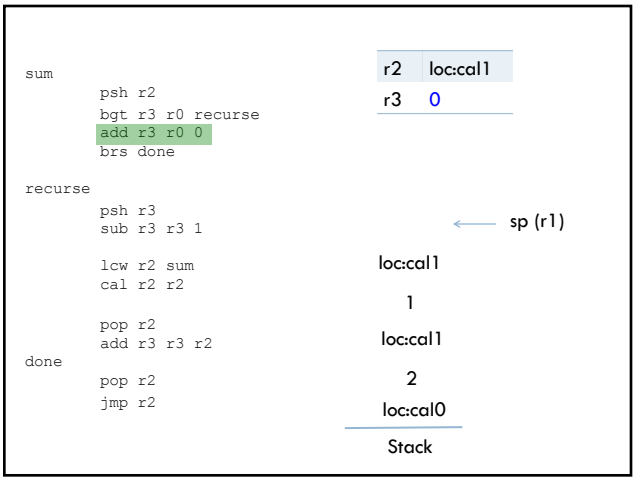
45


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

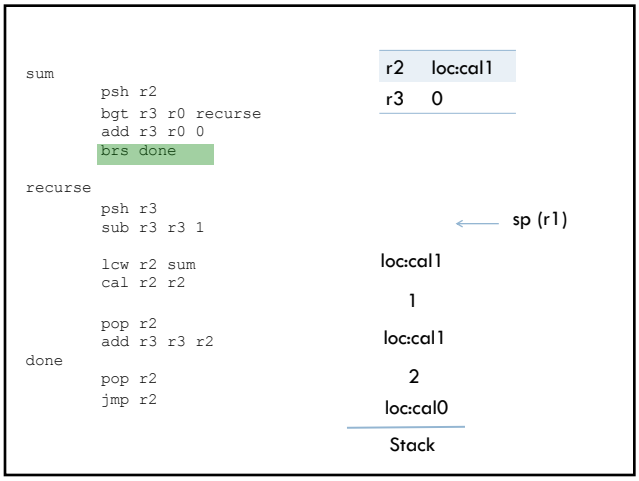
46


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

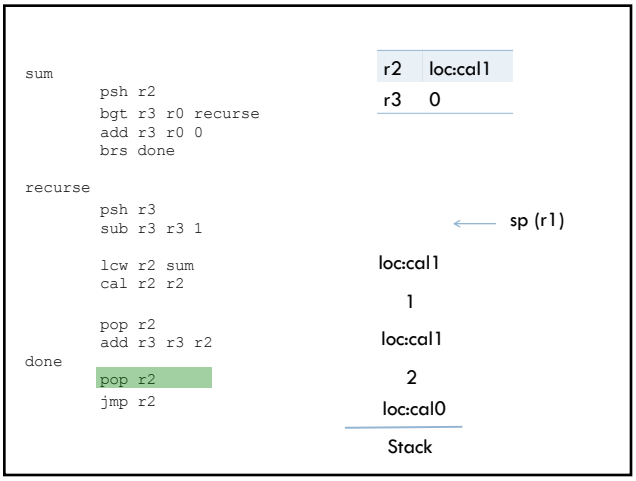
47


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

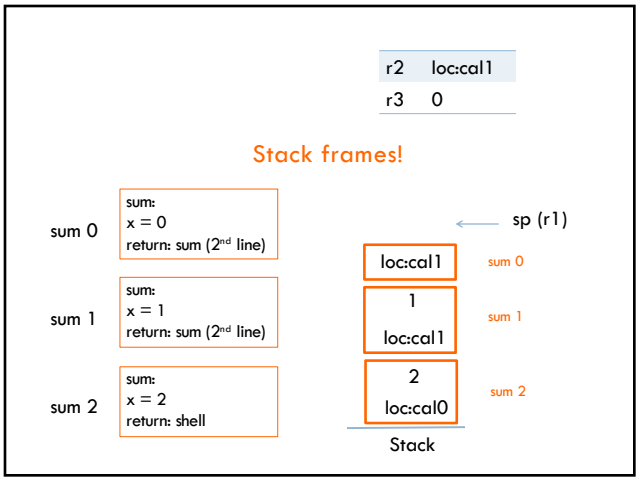
48


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

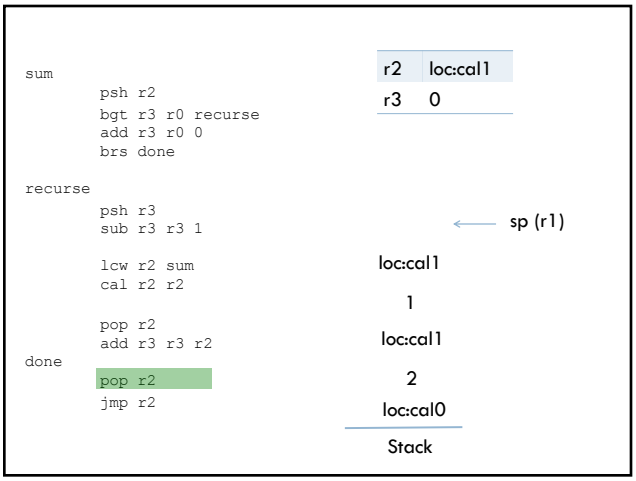
49


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

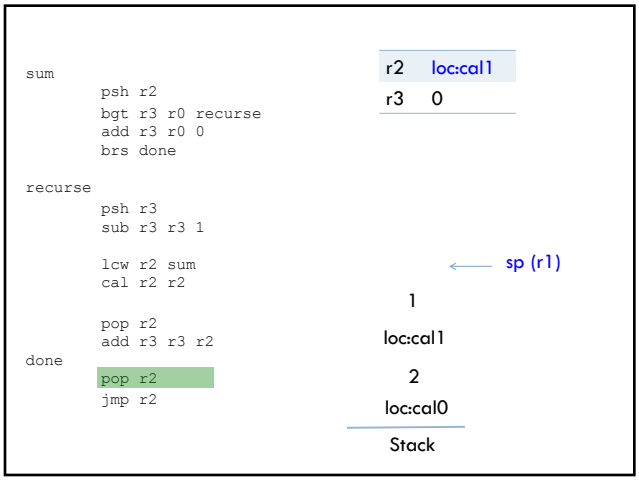
50


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

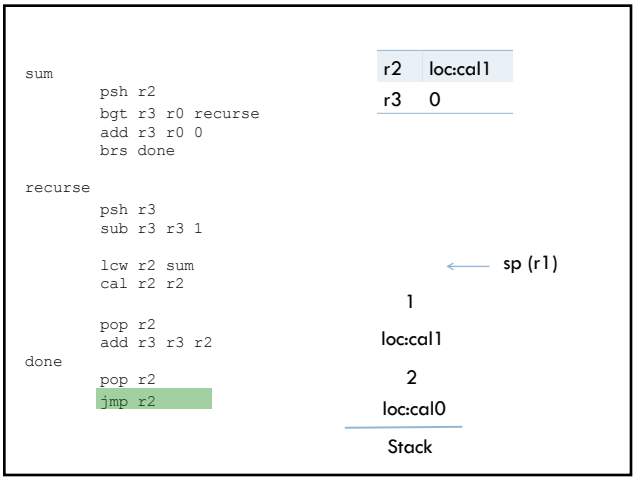
51


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

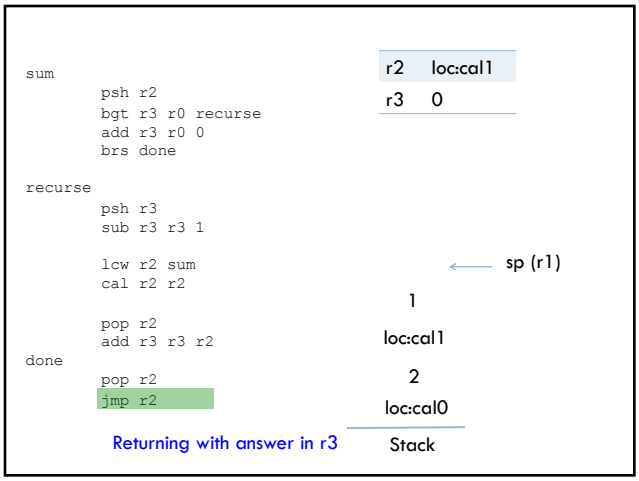
52


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

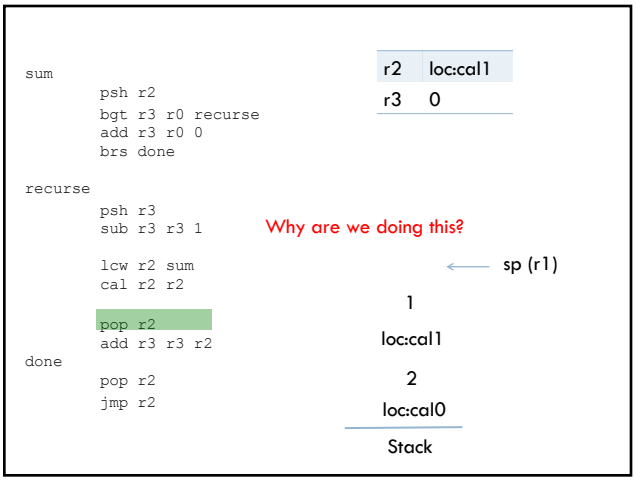
53


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

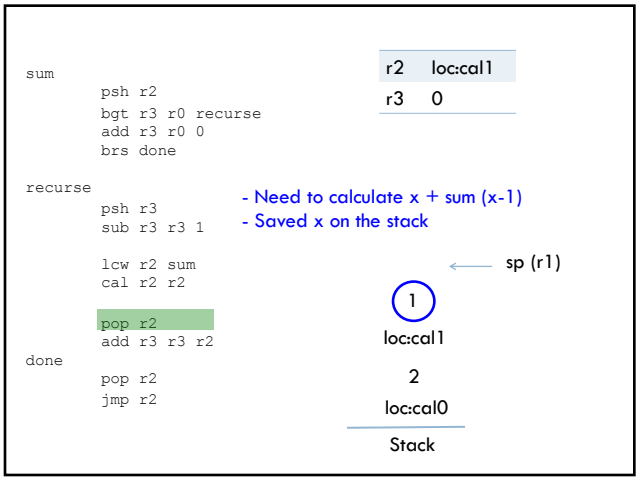
54


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
Returning with answer in r3
r2 loc:cal1
r3 0
sp (r1)
1
loc:cal1
2
loc:cal0
Stack

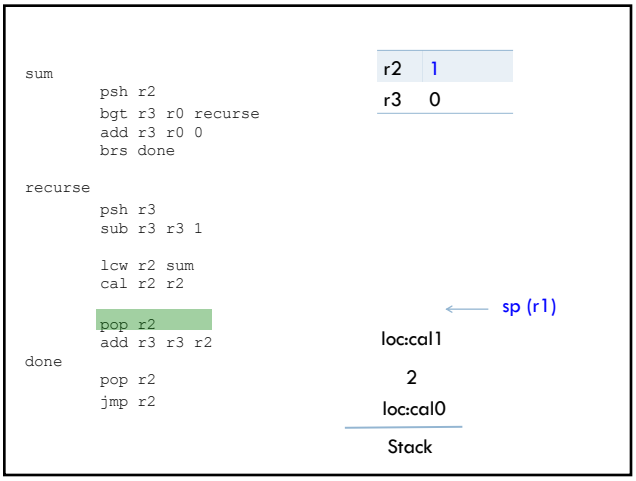
55


sum
psh r2
bgt r3 r0 recurse
add r3 r0 0
brs done
recurse
psh r3
sub r3 r3 1
Why are we doing this?
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

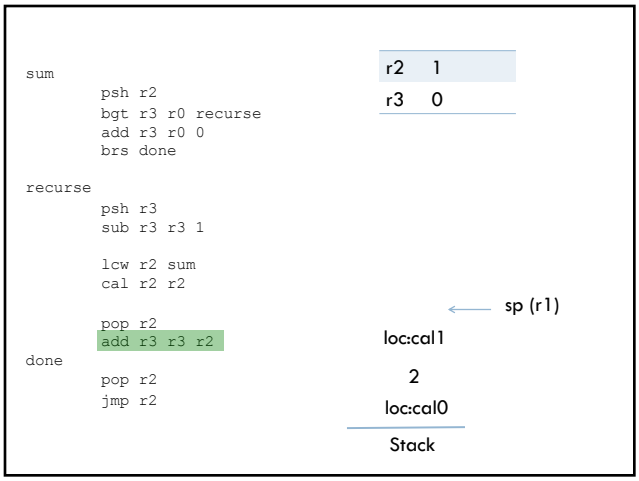
56


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

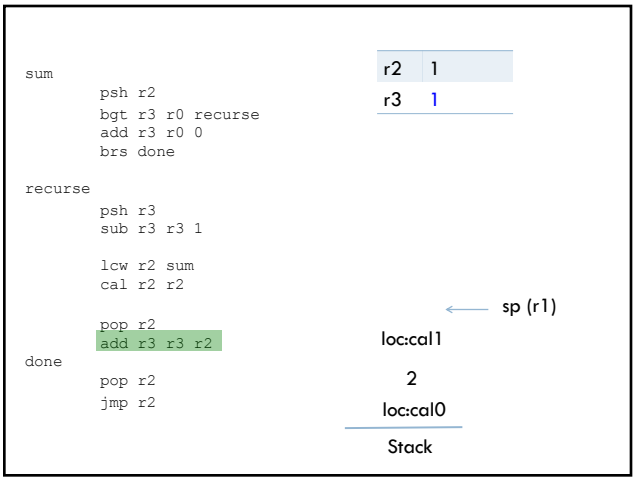
57


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

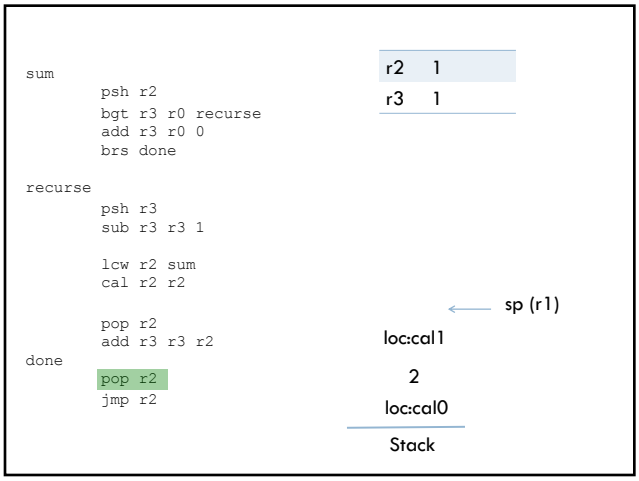
58


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

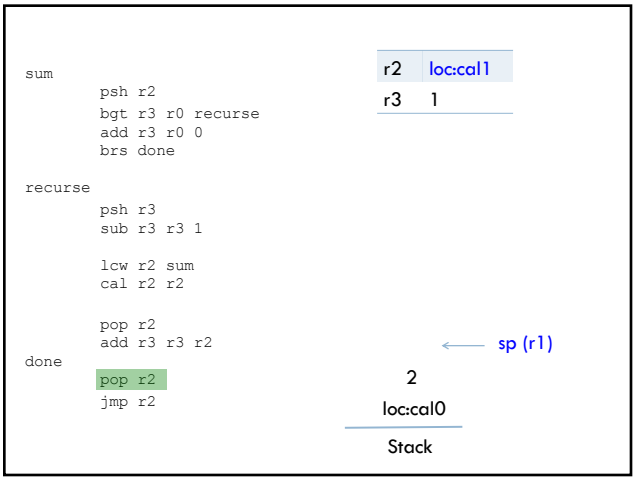
59


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

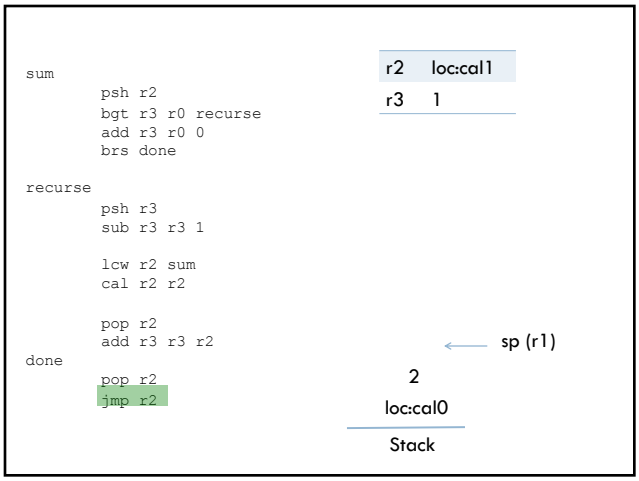
60


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

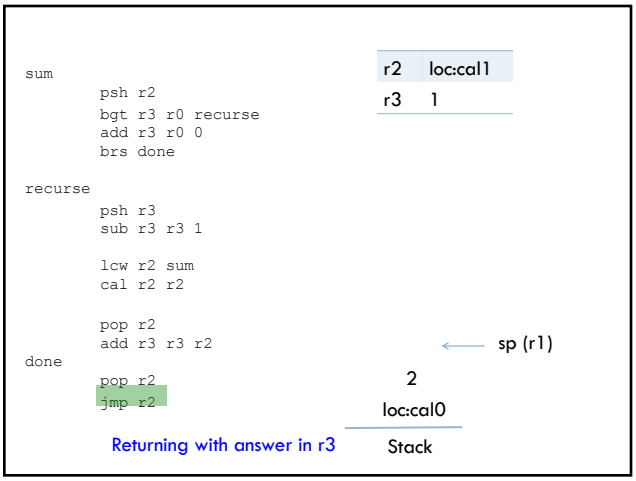
61


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

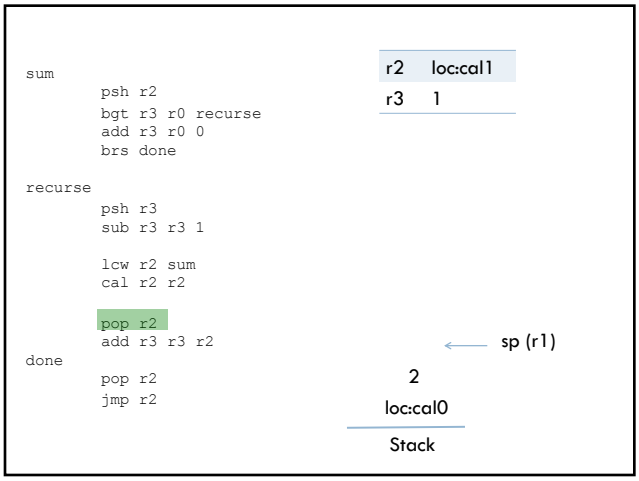
62


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

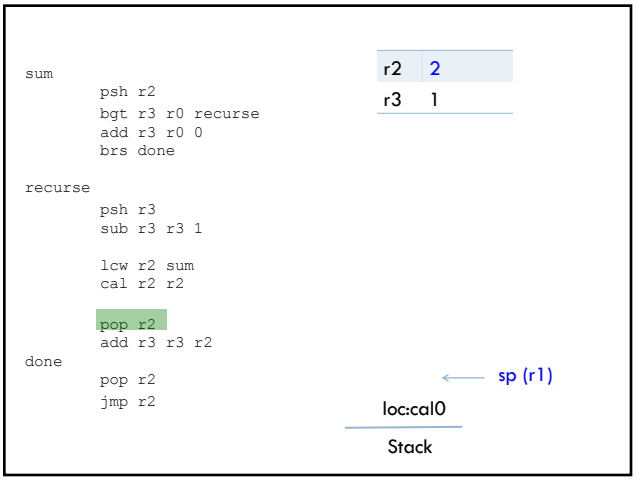
63


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

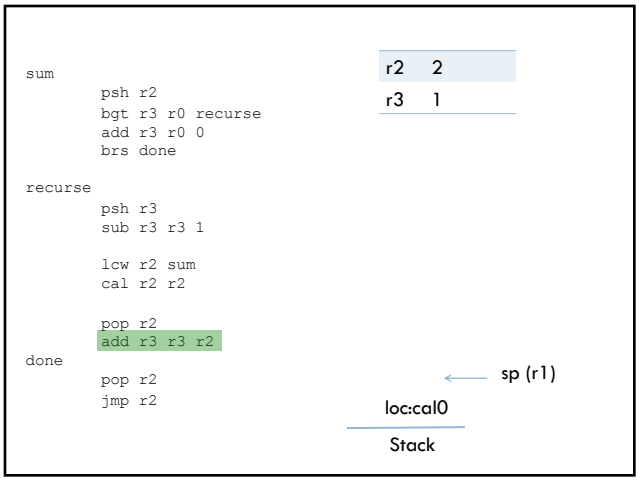
64


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

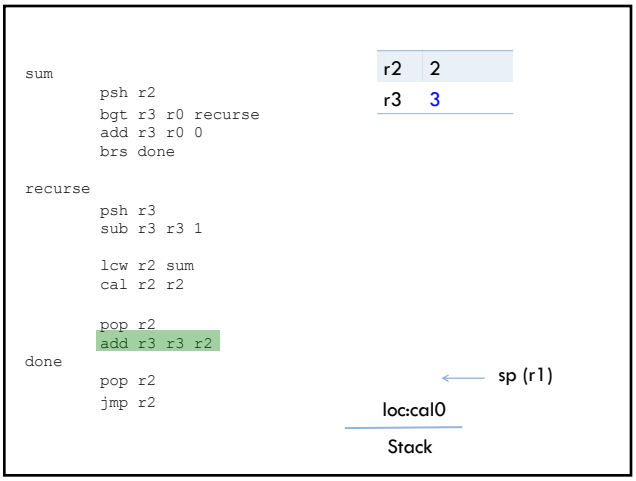
65


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

66


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

67


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

68

```
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

69

```
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

70

```
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

71

```
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

72

Calling sum

```
loa r3 r0

lcw r2 sum
cal r2 r2

sto r3 r0
hlt
```

r2	loc:cal0
r3	3

← sp (r1)

Stack

73

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

74

Calling sum

```
loa r3 r0

lcw r2 sum
cal r2 r2

sto r3 r0
hlt
```

r2	loc:cal0
r3	3

← sp (r1)

Stack

75

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

76

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

77

Look at sum example

78

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

79

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
- additional arguments are put on the stack!
- return value goes in r3

80

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?

81

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

82

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

83

Calling functions with multiple arguments

```
; put the first argument into r3
; put the second argument into r2
psh r2          ; push the second argument onto the stack
lcw r2 func     ; setup a call to the two-parameter function "func"
cal r2 r2       ; call the function
pop r0          ; cleanup the stack!
                ; if you wanted the value, you could also do pop r2
                ; which puts the value in r2
```

- Push the second argument onto the stack
- Make sure that after the function returns, you pop off the second argument
  - If you don't need the value, you can pop it to r0
  - If you want the value, you can pop it to r2 (r3 will have the return value of the function)

84

multiply

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

85

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

86

More examples

I didn't have time to cover the next two examples in class, but left it in the notes as examples of functions that take two parameters

The first example is multiply (which is recursive) and the second example is max (which is not recursive).

87

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?

88

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 ☺

89

```
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
    lcw r2 mult     ; call mult recursively
    cal r2 r2
    pop r0          ; pop 2nd argument off stack and throw away

    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

90

```
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
    lcw r2 mult     ; call mult recursively
    cal r2 r2
    pop r0          ; pop 2nd argument off stack and throw away

    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

91

```
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
    lcw r2 mult     ; call mult recursively
    cal r2 r2
    pop r0          ; pop 2nd argument off stack and throw away

    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

92

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

93

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

94

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

95

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

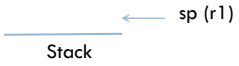
sp (r1)
Stack

96

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

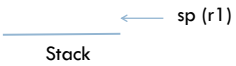


97

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

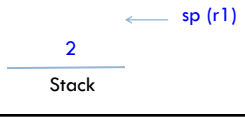


98

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

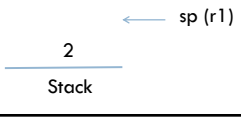


99

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



100

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

101

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

102

```
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

103

```
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

104

```
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

105

```
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

106

```
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

107

```
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

108

```
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

109

```
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

110

```
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

111

```
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

112

```
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?

6
loc: cal0
2
Stack

113

```
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!

6
loc: cal0
2
Stack

114

```
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

6
loc: cal0
2
Stack

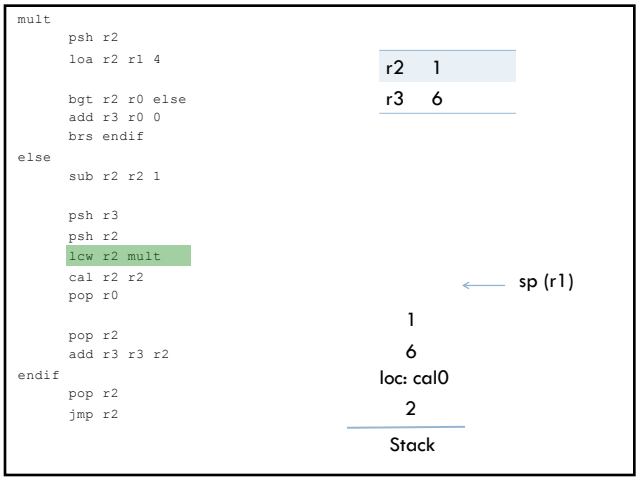
115

```
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
```

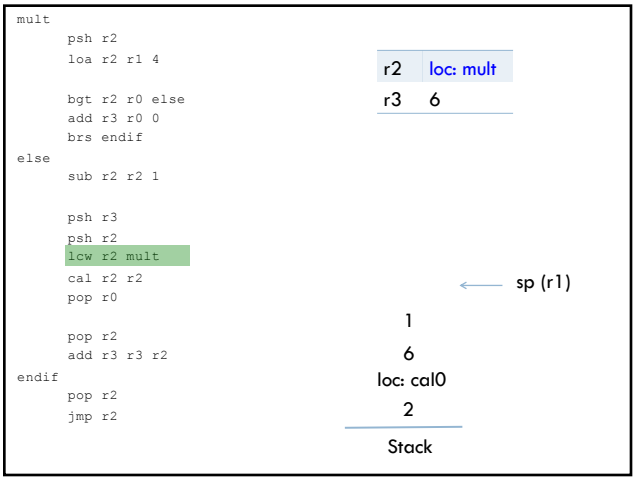
r2 1
r3 6

1
6
loc: cal0
2
Stack

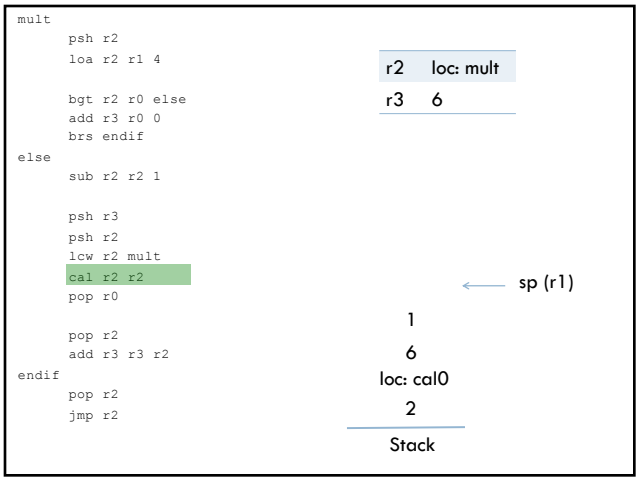
116



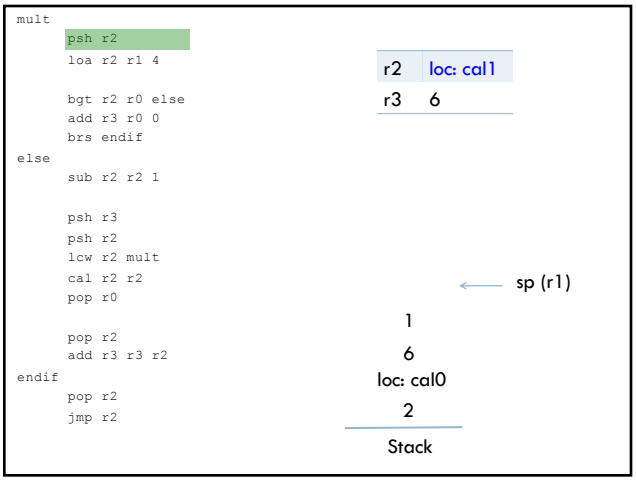
117



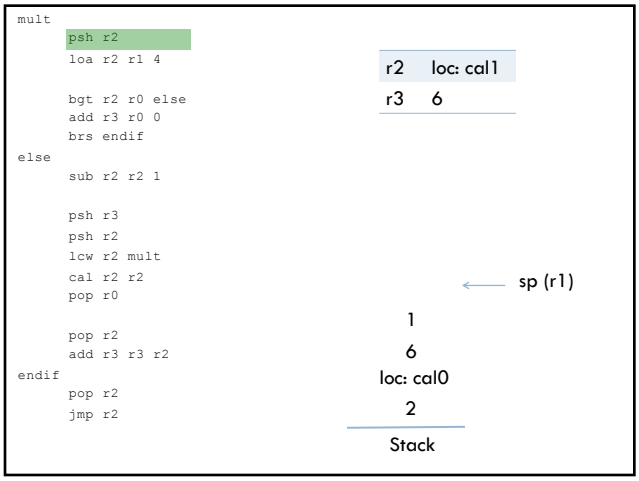
118



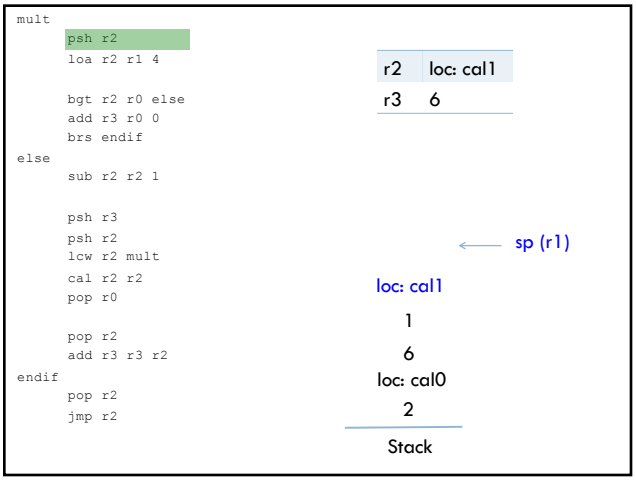
119



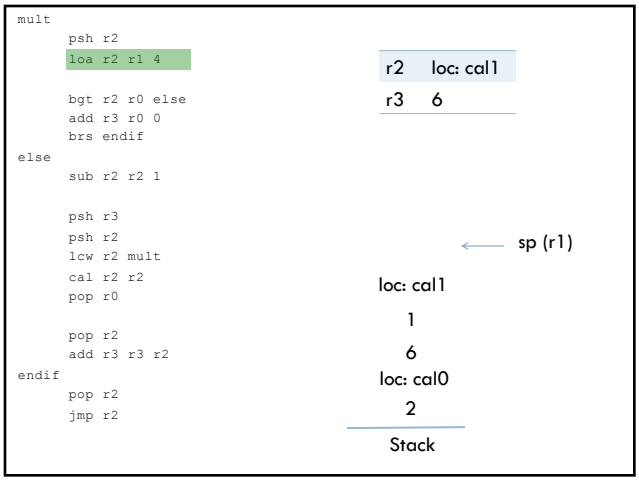
120



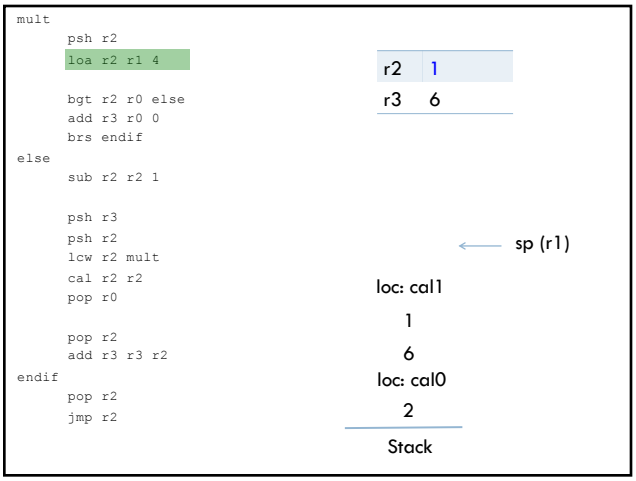
121



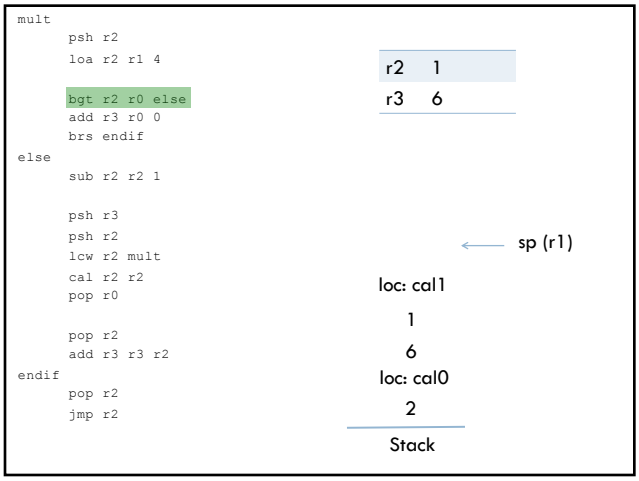
122



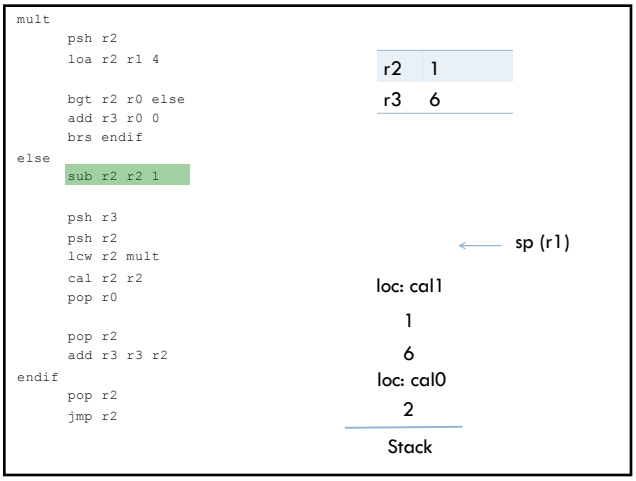
123



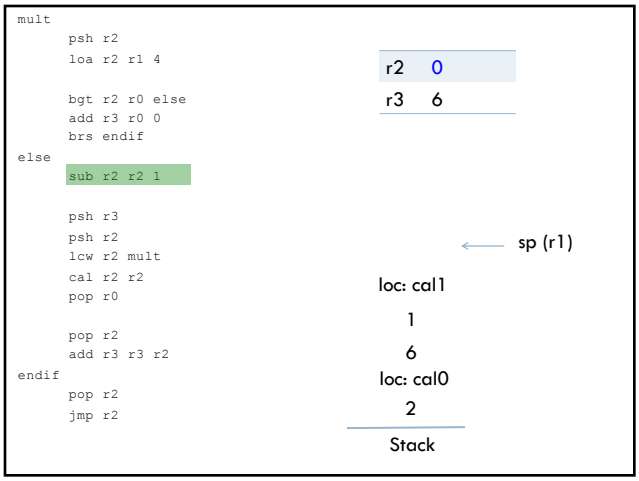
124



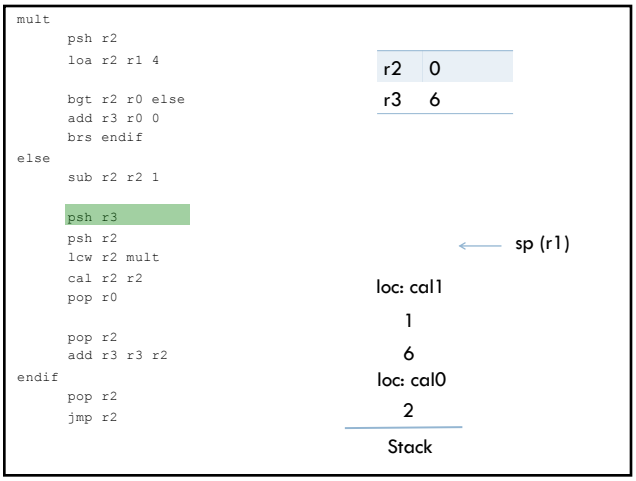
125



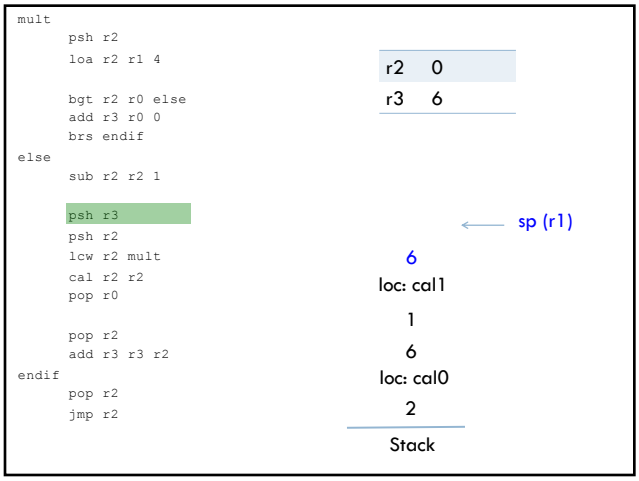
126



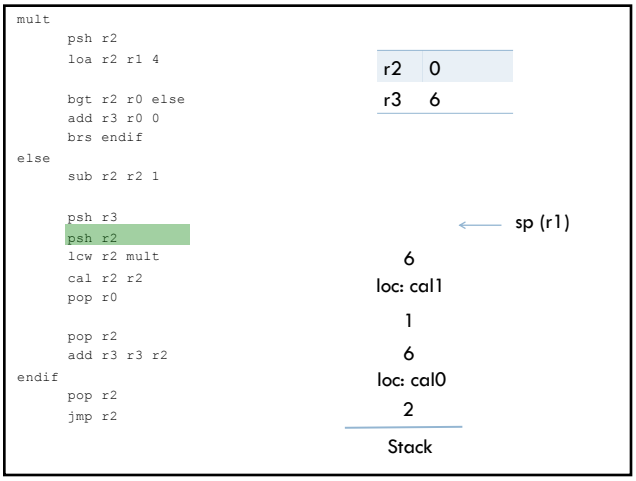
127



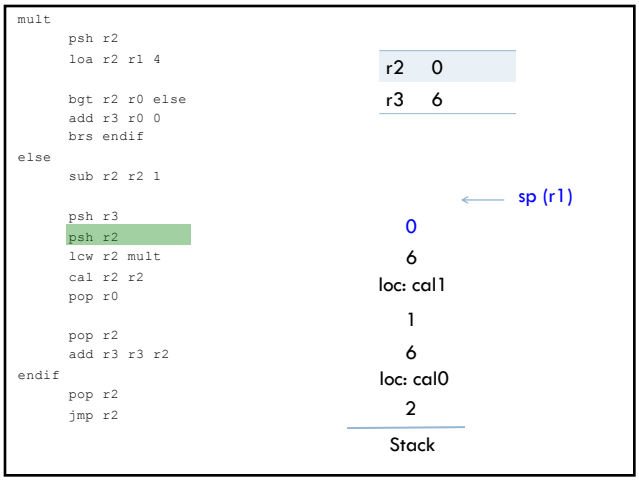
128



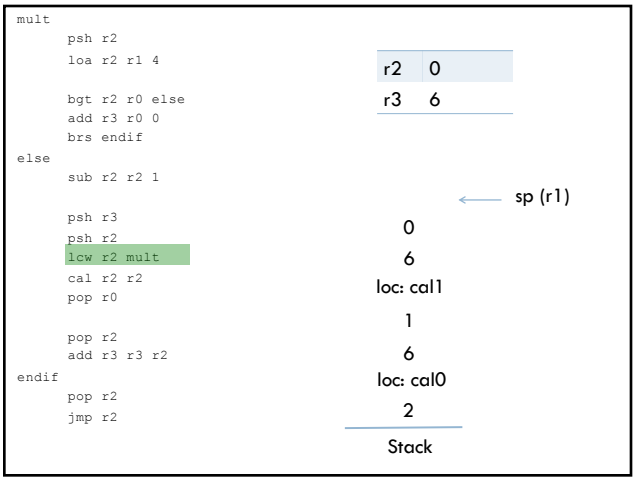
129



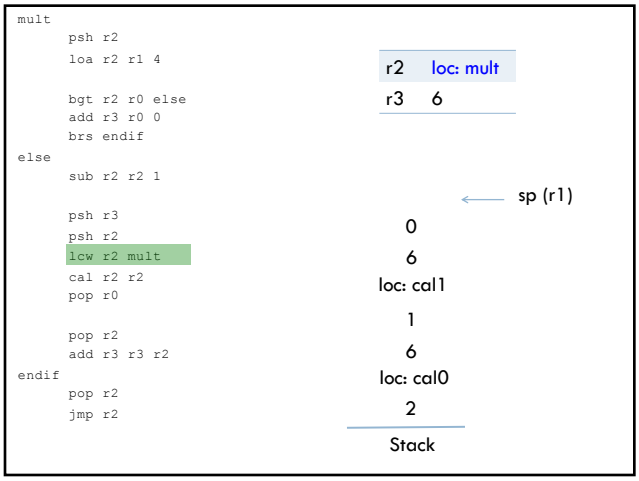
130



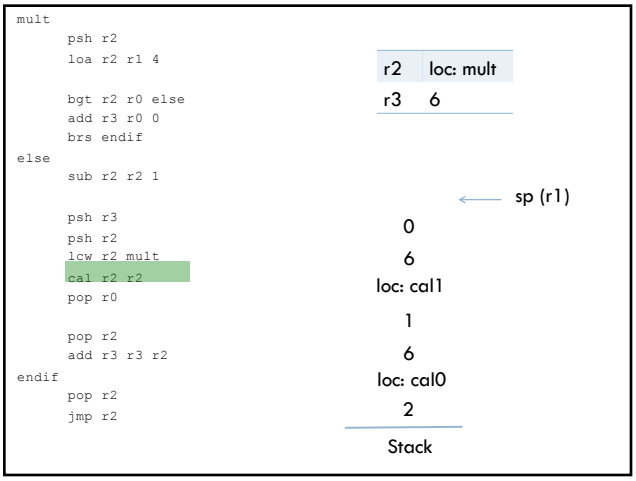
131



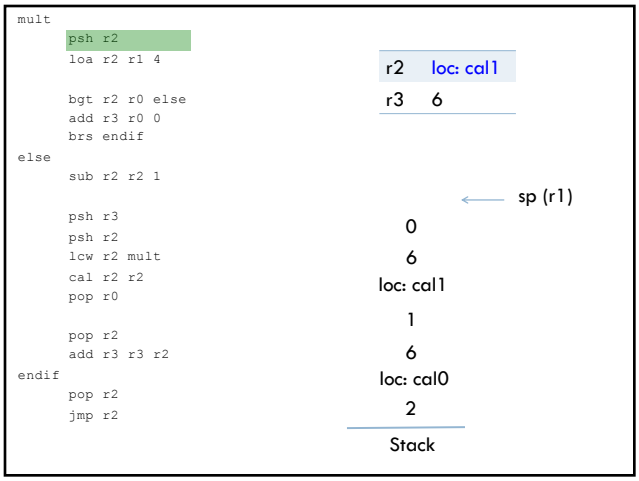
132



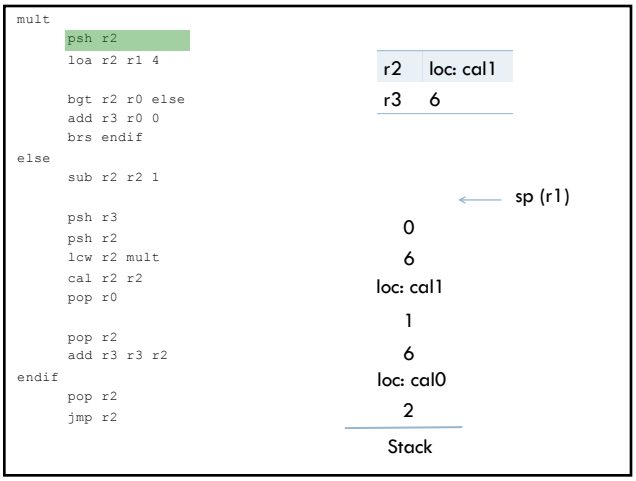
133



134



135



136

```
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

137

```
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

138

```
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

139

```
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
```

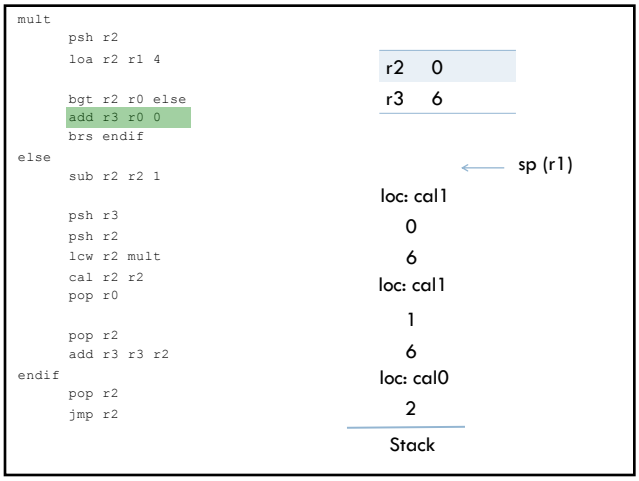
r2 0
r3 6

← sp (r1)

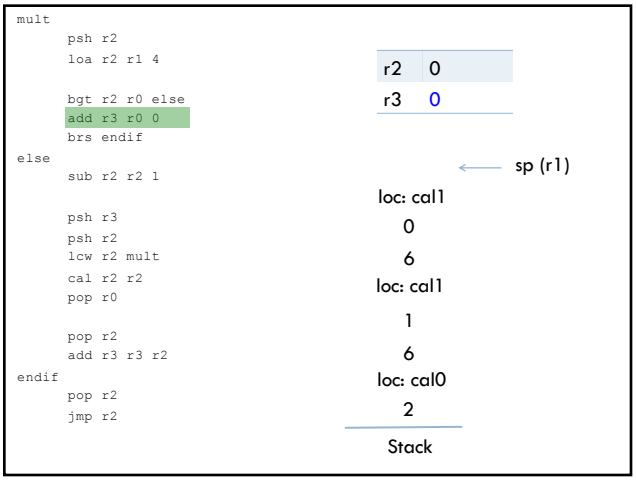
loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2

Stack

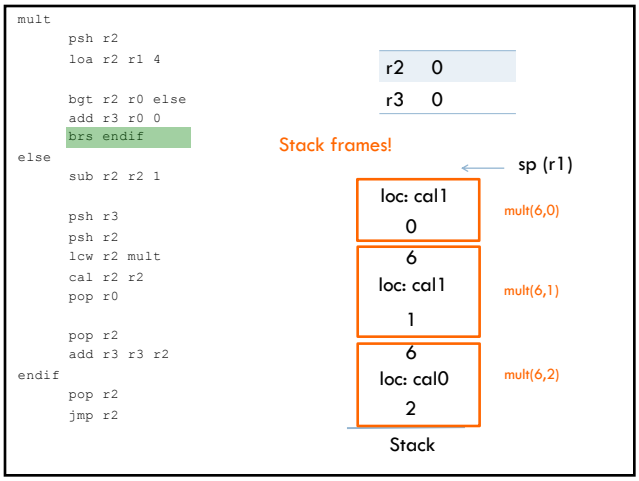
140



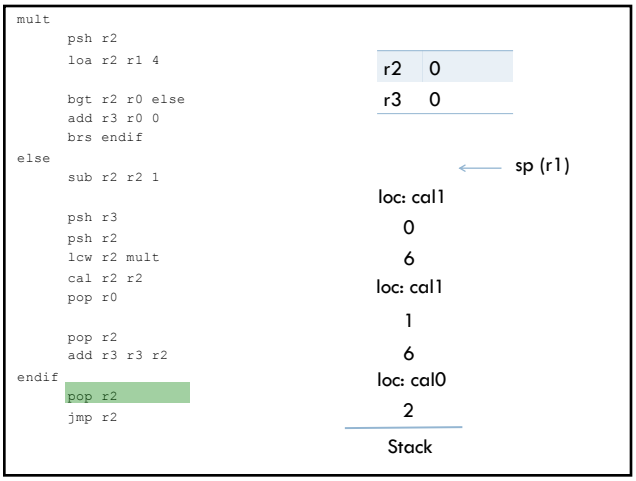
141



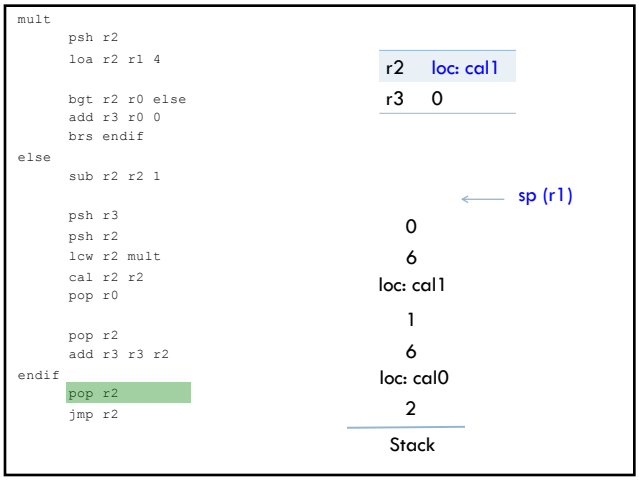
142



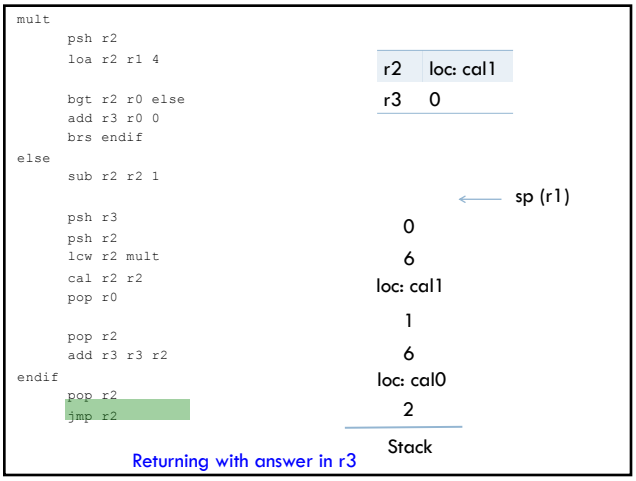
143



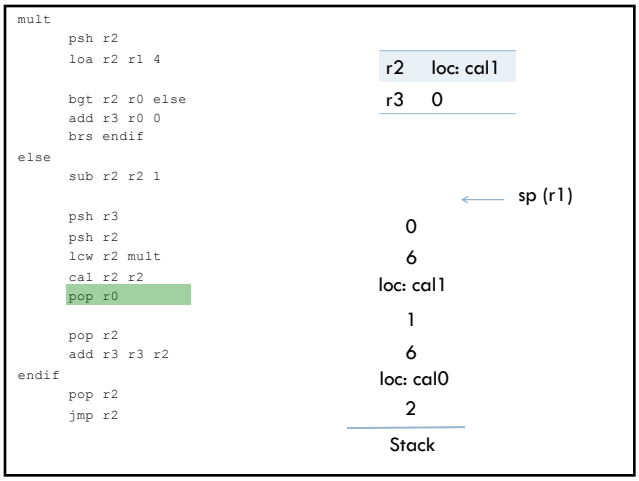
144



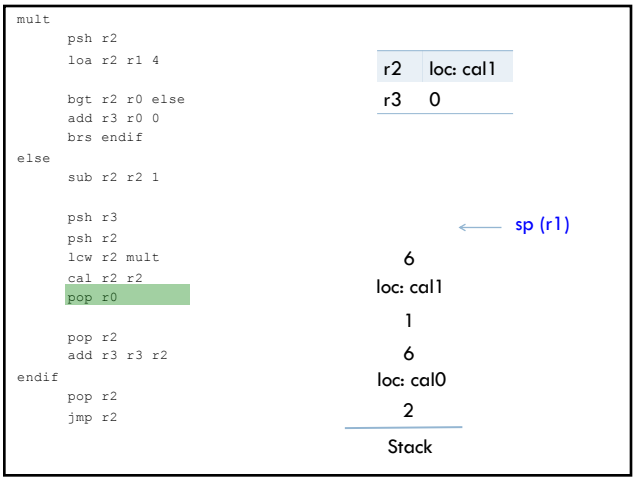
145



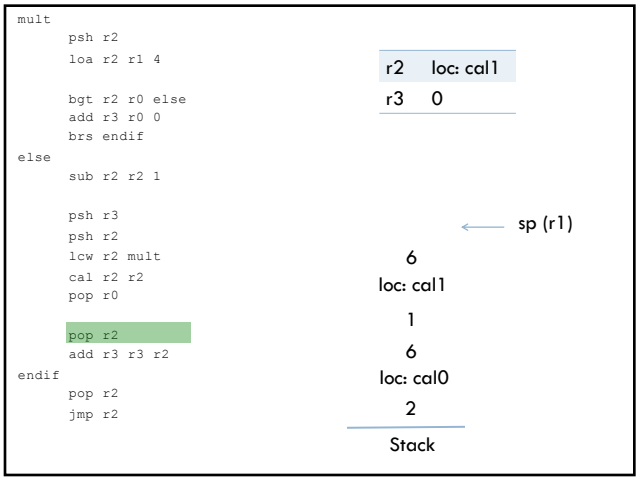
146



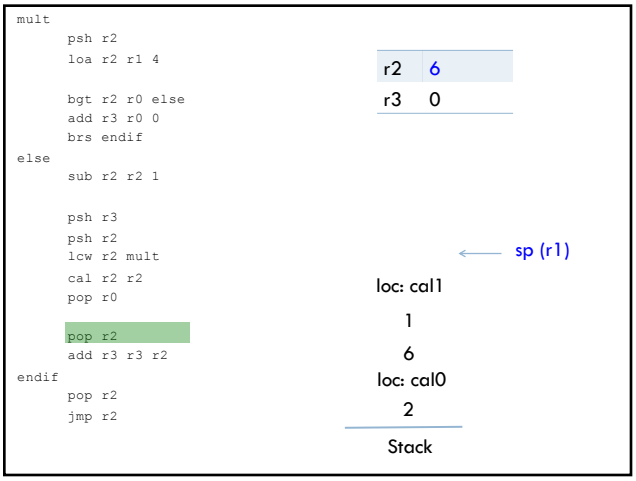
147



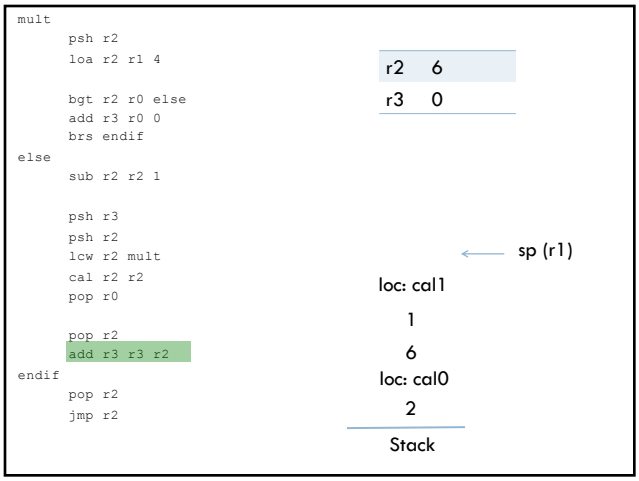
148



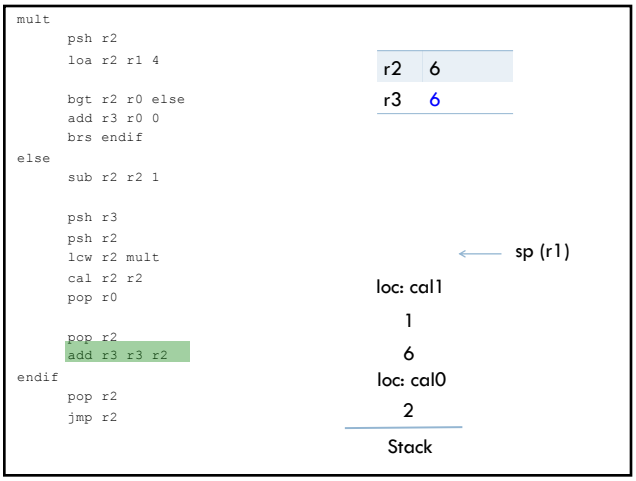
149



150



151



152

```
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

153

```
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

154

```
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

155

```
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
```

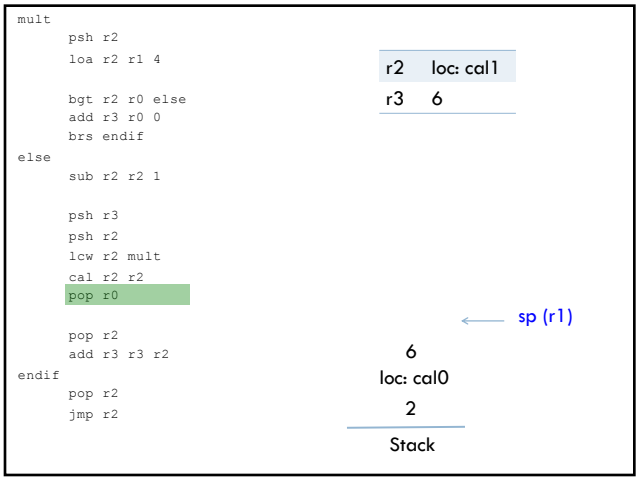
r2 loc: cal1
r3 6

← sp (r1)

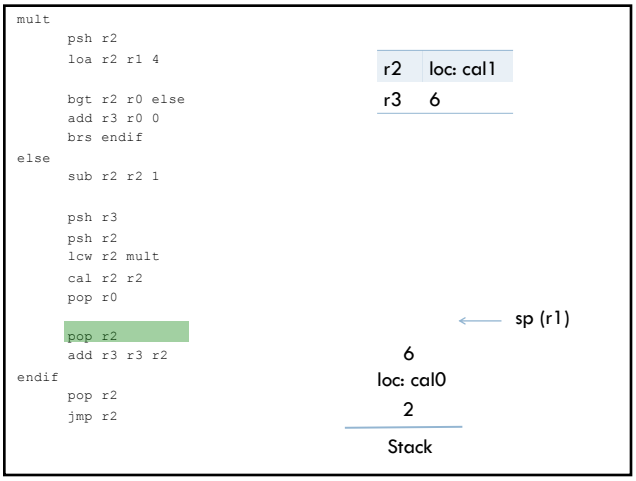
1
6
loc: cal0
2

Stack

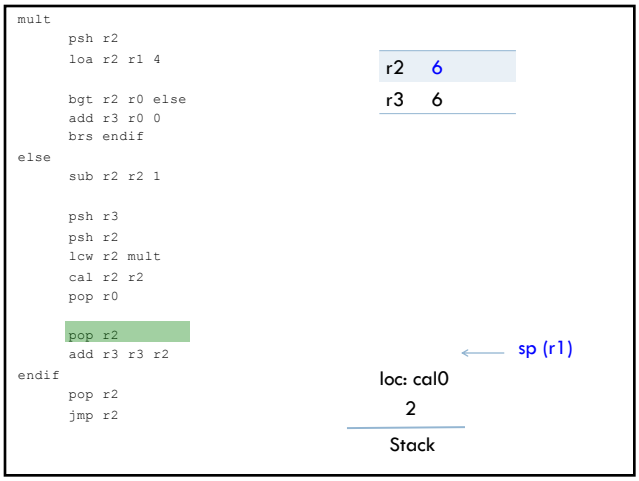
156



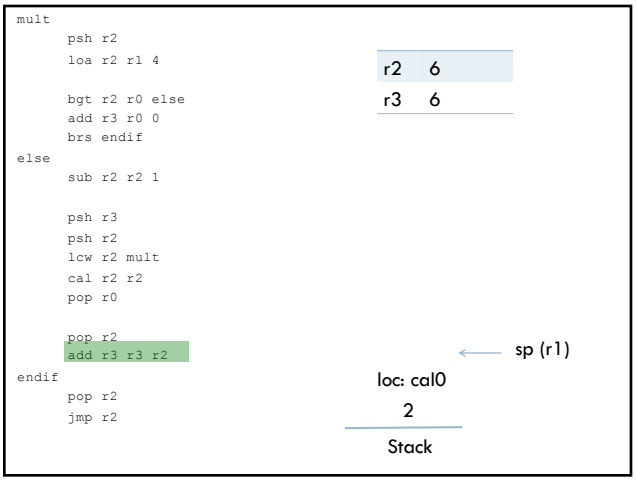
157



158



159



160

```
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

161

```
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

162

```
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

163

```
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

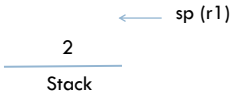
Returning with answer in r3

164

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	loc: cal0
r3	12

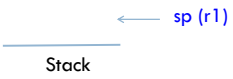


165

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	loc: cal0
r3	12

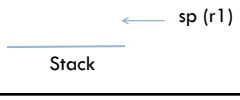


166

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	loc: cal0
r3	12



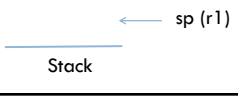
167

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	loc: cal0
r3	12

Print the answer: 12!



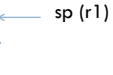
168

Calling mult

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 mult
cal r2 r2

pop r0

sto r3 r0
hlt
```

r2	loc: cal0
r3	12

Print the answer: 12!



169

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

170

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?

186

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!

187

Calling max

```
    loa r3 r0
    loa r2 r0

    psh r2
    lcw r2 max
    cal r2 r2
    pop r0

    sto r3 r0
    hlt
```

Anything different?

188

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

189

```
    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
```

190

```
    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
```

191

```
    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

192

```
    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

193

```
    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

194

```
    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

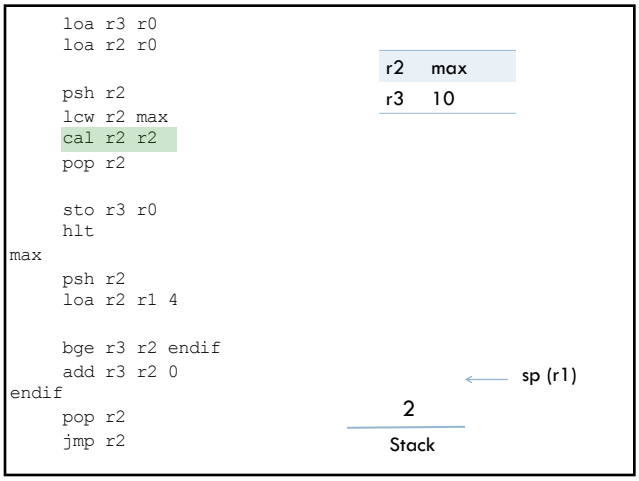
If we hadn't saved it on the stack, it would have been lost

← sp (r1)

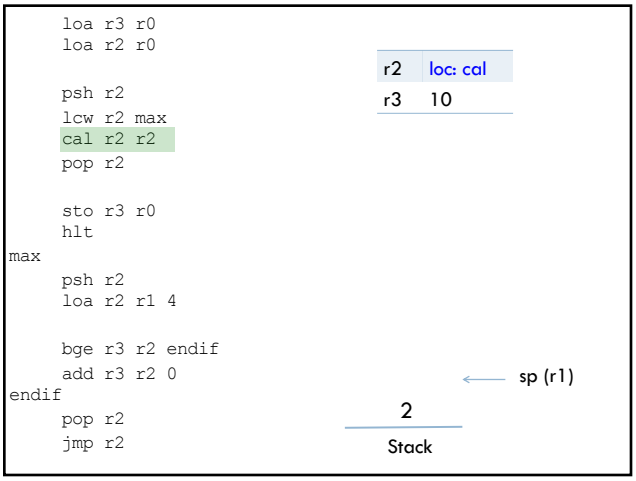
2

Stack

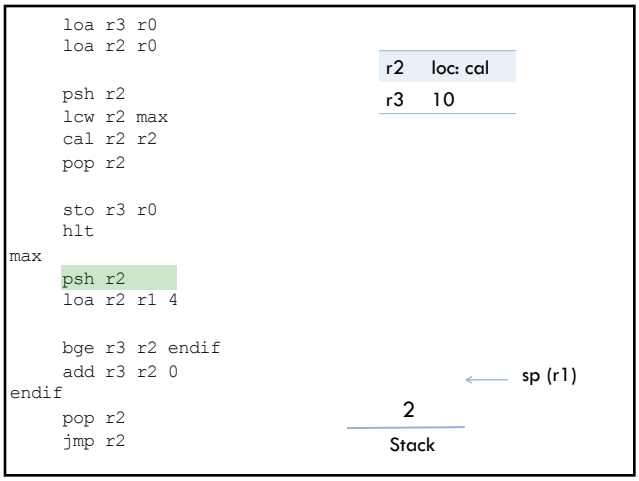
195



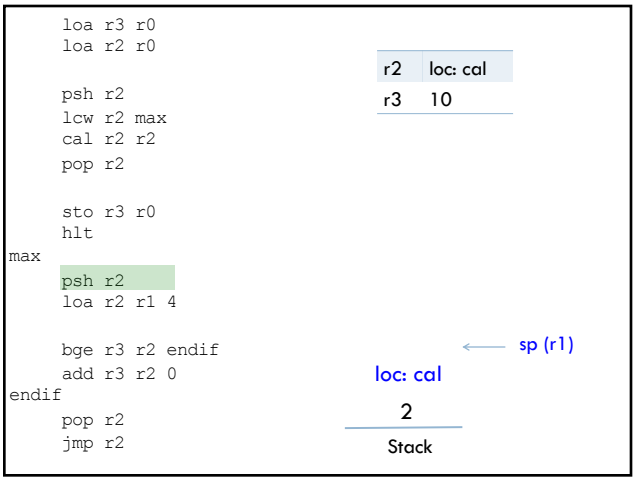
196



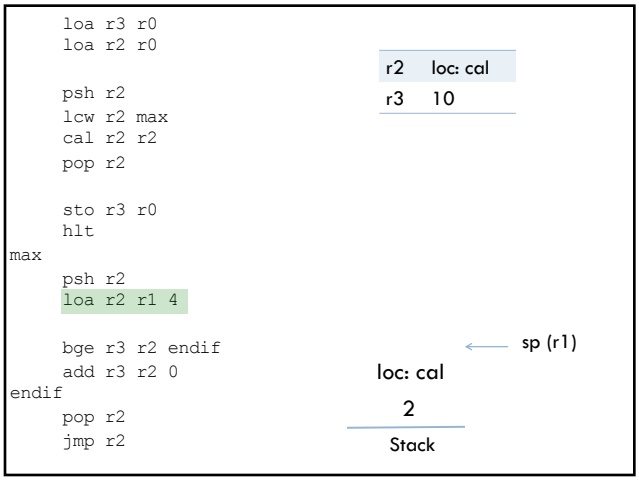
197



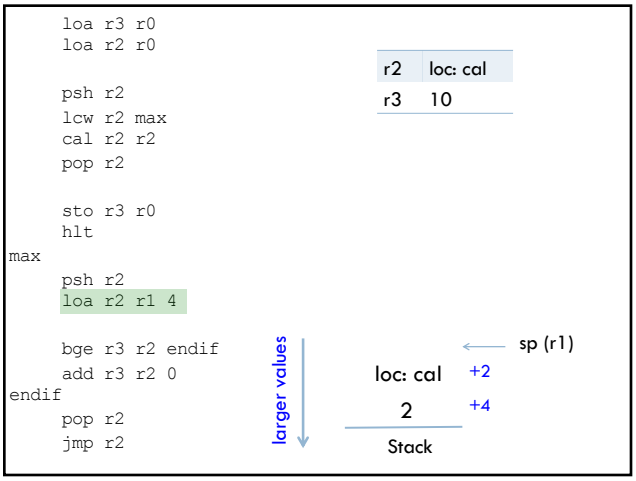
198



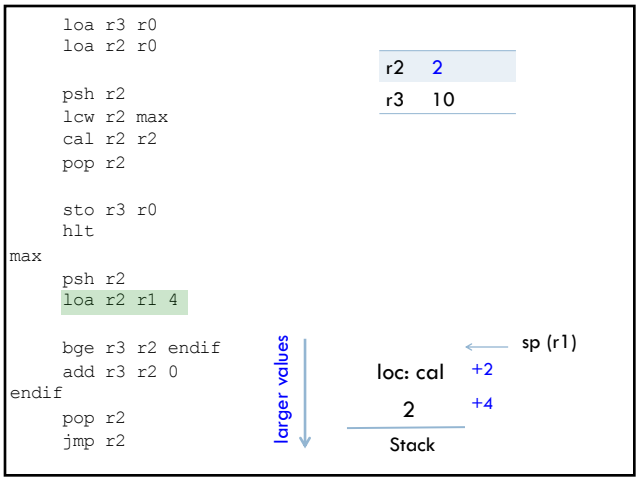
199



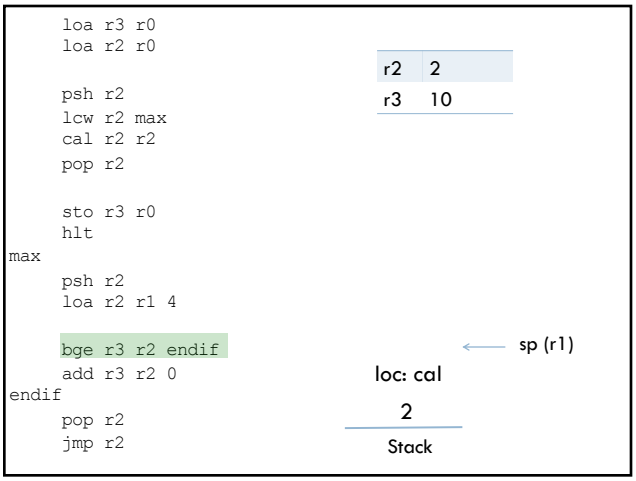
200



201



202



203

```
     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

204

```
     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

205

```
     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

206

```
     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

207

```
     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

208

```
     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

209

```
     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

210

```
     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

211