CS41B RECURSION

David Kauchak
CS 52 – Fall 2015

Admin

Assignment 4 out: due Monday (2/29 at 11:59pm)

Survey in assignment 4

Assignment 2 scores

Midterm back
- Average: 23.5 (81%)
- Q1: 26.9 (93%)
- Median: 23.75 (82%)
- Q3: 20.25 (70%)

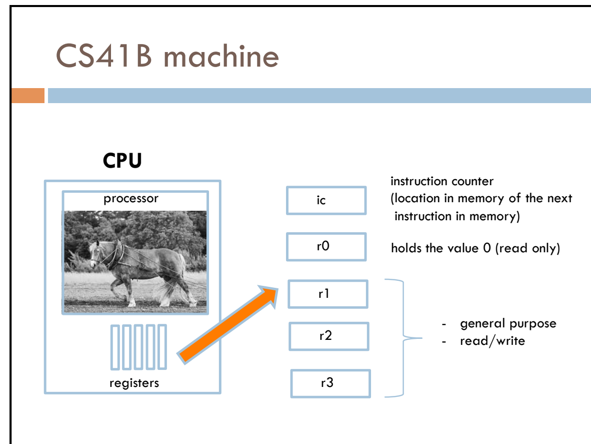
Academic Honesty

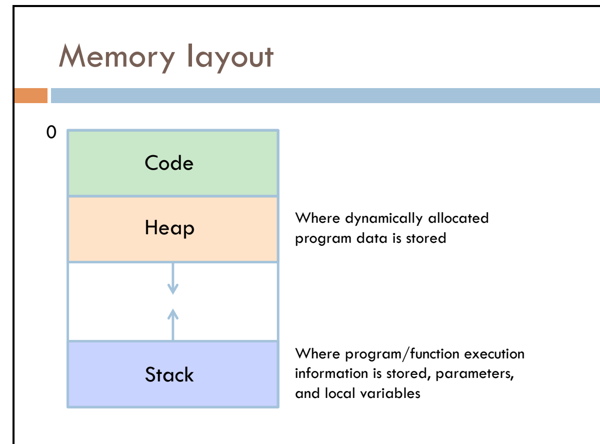
A few rules to follow for this course to keep you out of trouble:

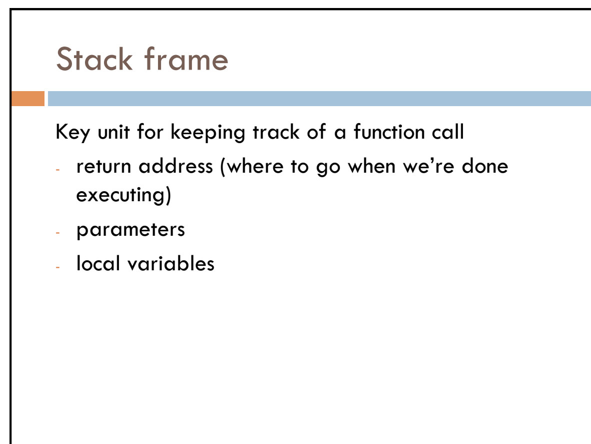
- If you talk with someone in the class about a problem, you should not take notes. If you understand the material you talked about, you should be able to recreate it on your own.
- Similarly, if you talk with someone, you must wait 5 minutes before resuming work on the problem. Stretch. Use the restroom. Go for a quick walk. This will ensure that you really understand the material.
- You may not sit next to (or where you can see the screen of) anyone you are talking with about the assignment.
- The only time you may look at someone else's screen is if they are asking you for help with a basic programming problem (e.g. syntax error). You should not look at someone else's code to help yourself!

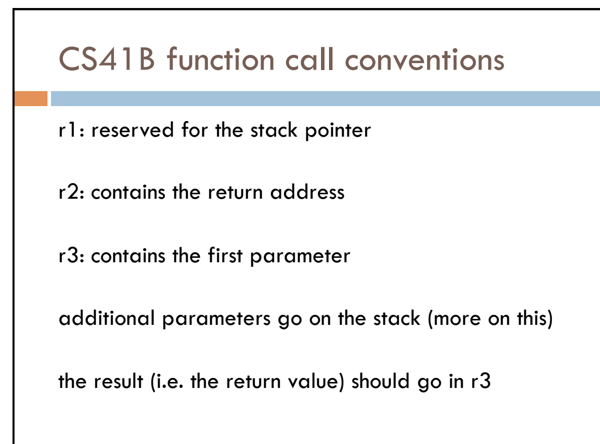
Examples from this lecture

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


CS41B machine
CPU
processor
registers
ic
r0
r1
r2
r3
instruction counter
(location in memory of the next instruction in memory)
holds the value 0 (read only)
- general purpose
- read/write


Memory layout
0
Code
Heap
Stack
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


CS41B 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

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

Our first function call

```
     loa r3 r0          ; get variable

     lcw r2 increment   ; call increment
     cal r2 r2

     sto r0 r3          ; write result,
     hlt                ;    and halt

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

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	RR[S]	dest = mem[src0 + arg]

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	RR[S]	dest = mem[src0 + arg]

- 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
```

loa	RR[S]	dest = mem[src0 + arg]

- 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

Multiple arguments

```
max
     psh r2
     loa r2 r1 4

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

What does this code do?

Multiple arguments

```
max
     psh r2
     loa r2 r1 4

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

max, as a function!

Calling max

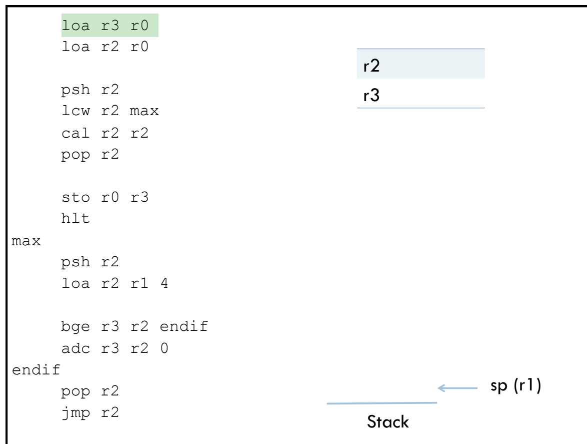
```
loa r3 r0
loa r2 r0

psh r2
lcw r2 max
cal r2 r2
pop r2

sto r0 r3
hlt
```

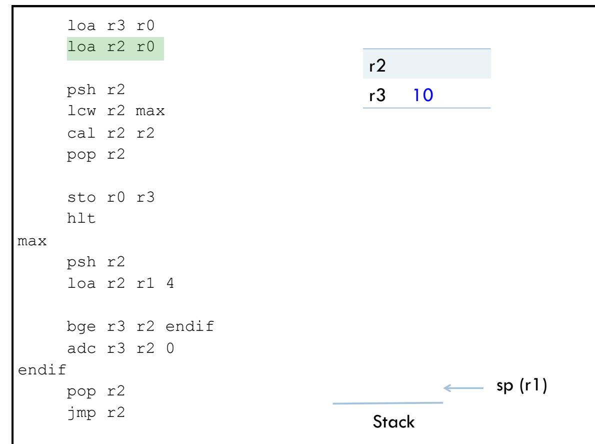
Anything different?

Calling max

```
loa r3 r0
loa r2 r0

psh r2
lcw r2 max
cal r2 r2
pop r2

sto r0 r3
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 r0 r3
    hlt
max
    psh r2
    loa r2 r1 4

    bge r3 r2 endif
    adc 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 r0 r3
    hlt
max
    psh r2
    loa r2 r1 4

    bge r3 r2 endif
    adc 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 r0 r3
    hlt
max
    psh r2
    loa r2 r1 4

    bge r3 r2 endif
    adc 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 r0 r3
    hlt
max
    psh r2
    loa r2 r1 4

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

r2	max
r3	10

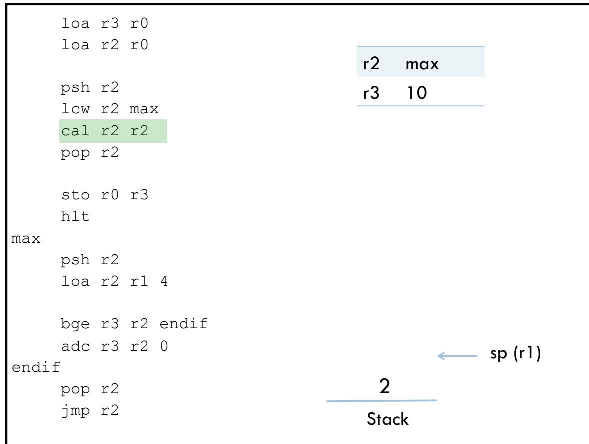
Notice that we overwrote the value in r2

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

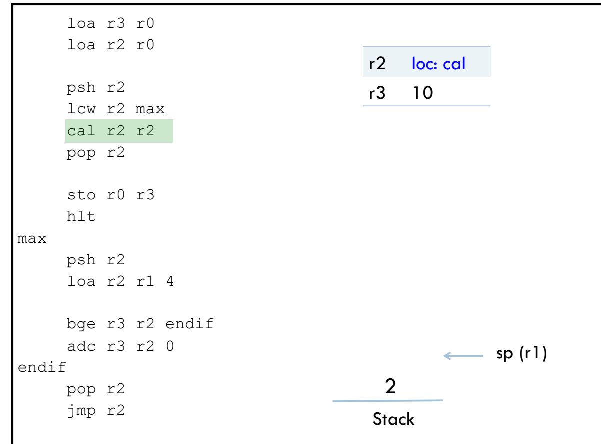
sp (r1)

2

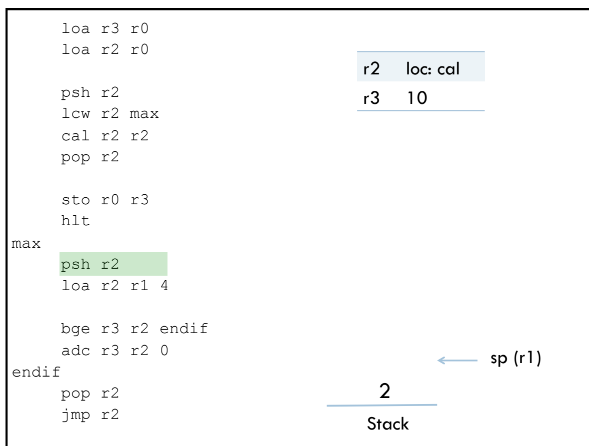
Stack



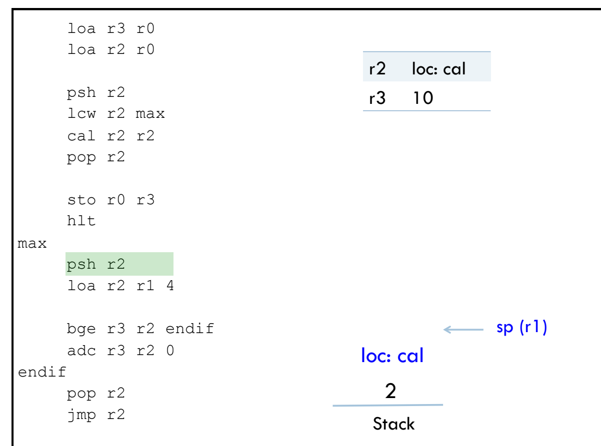
loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r0 r3
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
adc r3 r2 0
endif
pop r2
jmp r2
r2 max
r3 10
sp (r1)
2
Stack



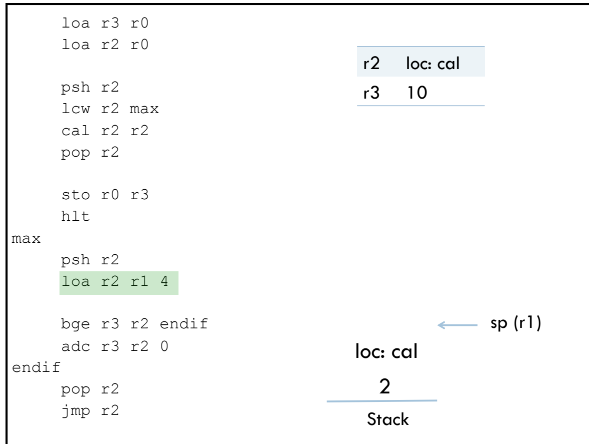
loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r0 r3
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
adc r3 r2 0
endif
pop r2
jmp r2
r2 loc: cal
r3 10
sp (r1)
2
Stack

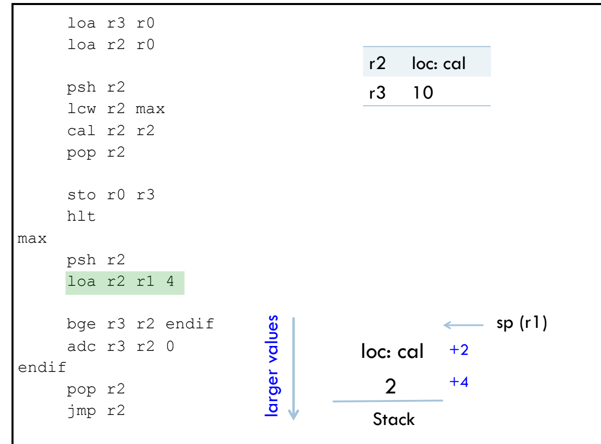


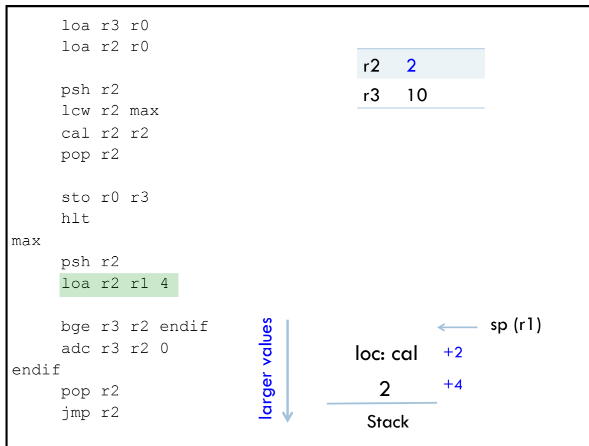
loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r0 r3
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
adc r3 r2 0
endif
pop r2
jmp r2
r2 loc: cal
r3 10
sp (r1)
2
Stack

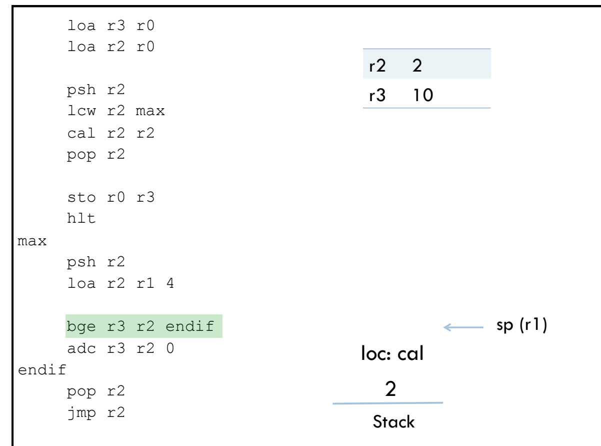


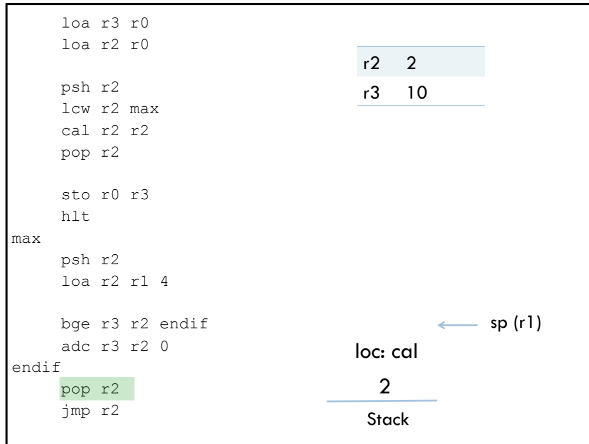
loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r0 r3
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
adc r3 r2 0
endif
pop r2
jmp r2
r2 loc: cal
r3 10
sp (r1)
loc: cal
2
Stack

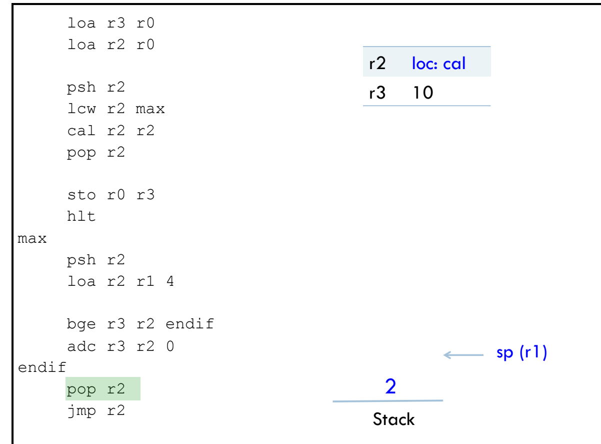

loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r0 r3
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
adc r3 r2 0
endif
pop r2
jmp r2
r2 loc: cal
r3 10
sp (r1)
loc: cal
2
Stack

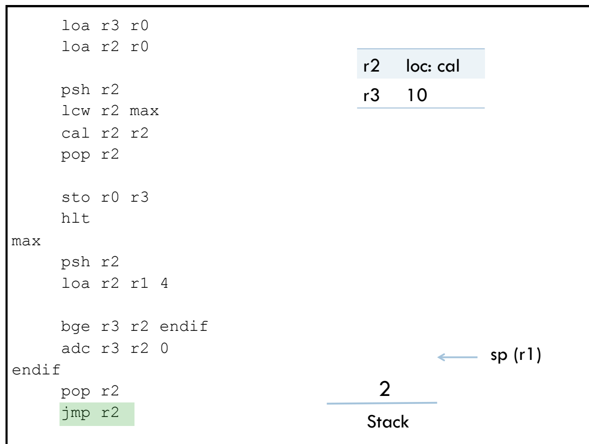

loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r0 r3
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
adc r3 r2 0
endif
pop r2
jmp r2
r2 loc: cal
r3 10
larger values
sp (r1)
loc: cal
+2
2
+4
Stack

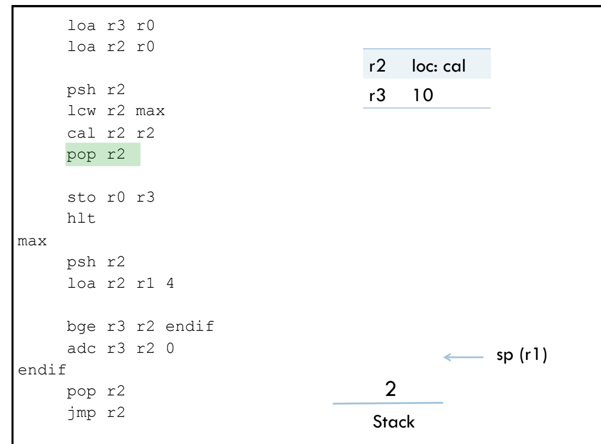

loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r0 r3
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
adc r3 r2 0
endif
pop r2
jmp r2
r2 2
r3 10
larger values
sp (r1)
loc: cal
+2
2
+4
Stack

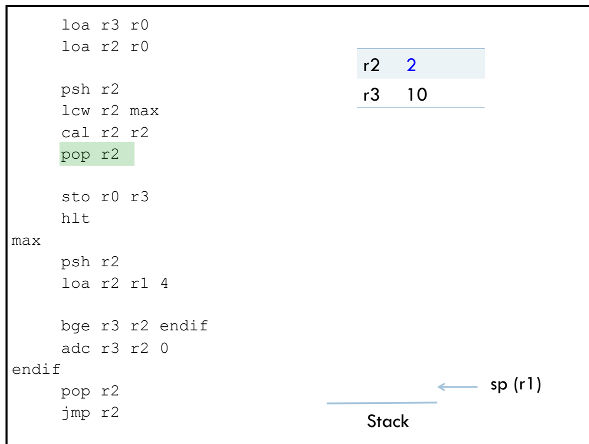

loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r0 r3
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
adc r3 r2 0
endif
pop r2
jmp r2
r2 2
r3 10
sp (r1)
loc: cal
2
Stack


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

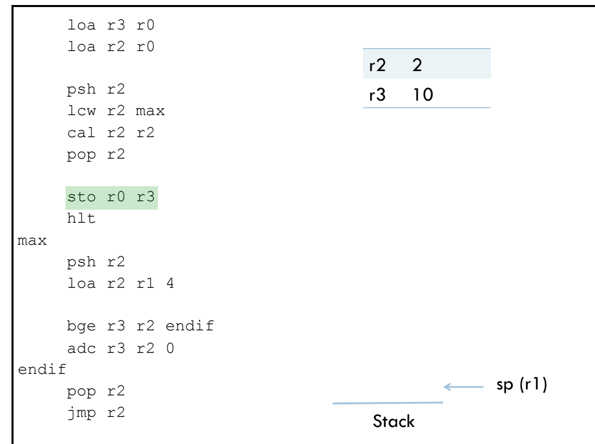

loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r0 r3
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
adc r3 r2 0
endif
pop r2
jmp r2
r2 loc: cal
r3 10
sp (r1)
2
Stack


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

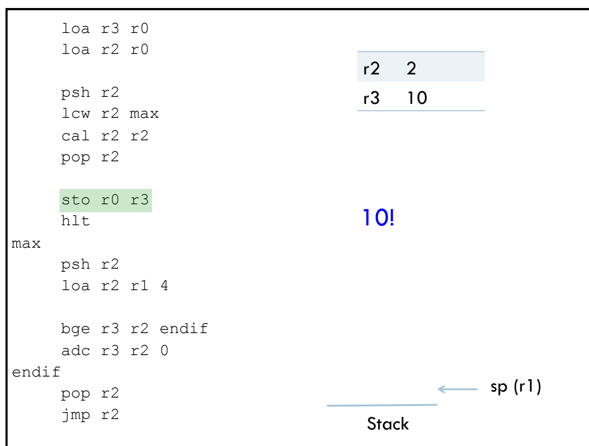

loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r0 r3
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
adc r3 r2 0
endif
pop r2
jmp r2
r2 loc: cal
r3 10
sp (r1)
2
Stack



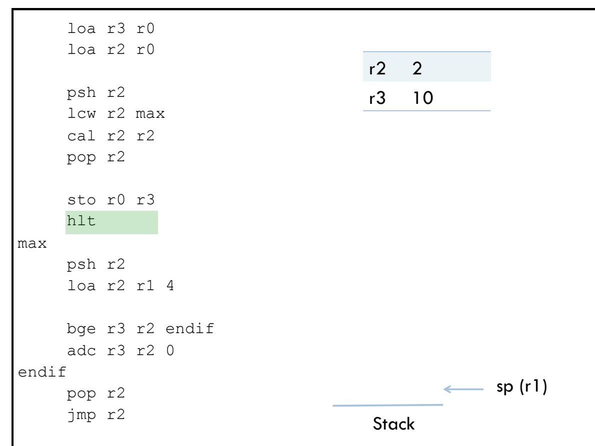
loa r3 r0
loa r2 r0
psh r2
lcw r2 max
cal r2 r2
pop r2
sto r0 r3
hlt
max
psh r2
loa r2 r1 4
bge r3 r2 endif
adc r3 r2 0
endif
pop r2
jmp r2
r2 2
r3 10
sp (r1)
Stack



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

Real structure of CS41B program

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

       instruction1         ; comment
       instruction2         ; comment
       ...
       hlt

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

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

Reserve 50 words for the stack

Recursion

```
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
       adc r3 r0 0      ; return 0
       brs endif
else
       sbc 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

       loa r2 r1 2      ; load a into r2 off of the stack
       add r3 r3 r2     ; r3 = a + mult(a, b-1)
       pop r0           ; remove first argument (a) from stack
endif
       pop r2           ; get the return address
       jmp r2           ; return
```

Function startup

Base case

Recursive call

answer calculation

Recursive case

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
        adc r3 r0 0     ; return 0
        brs endif
else
        sbc 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

        loa r2 r1 2     ; load a into r2 off of the stack
        add r3 r3 r2    ; r3 = a + mult(a, b-1)
        pop r0          ; remove first argument (a) from stack
endif
        pop r2          ; get the return address
        jmp r2          ; return
```

Function startup

if(b <= 0)
return 0

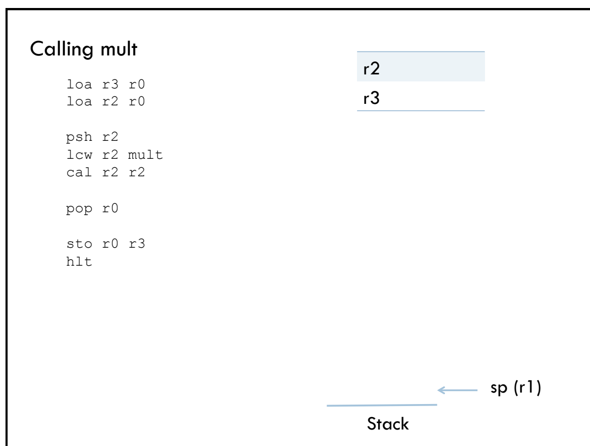
mystery(a, b-1)

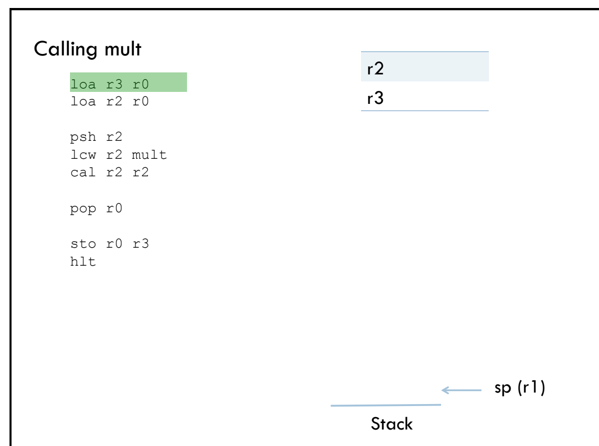
a + mystery(a, b-1)

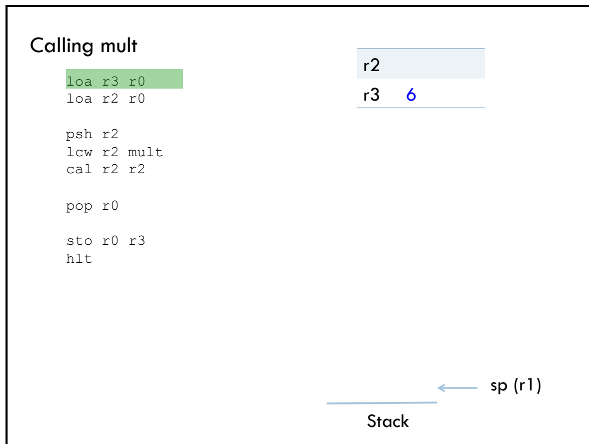
Function cleanup and return

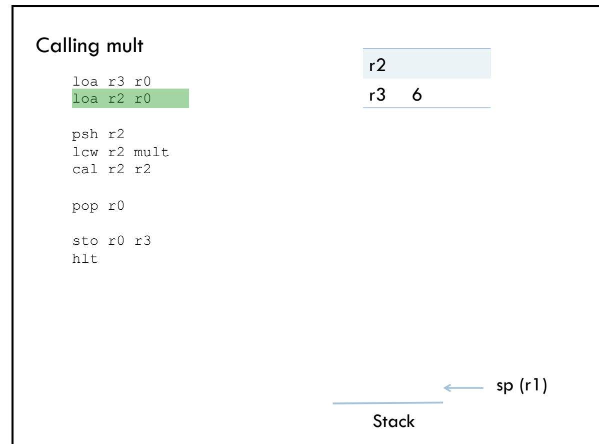
```
mult
        psh r2          ; save the return address
        loa r2 r1 4     ; get at the 2nd argument, b
                        ; a = r3, b = r2

        bgt r2 r0 else  ; r2 > 0, i.e. recursive case
        adc r3 r0 0     ; return 0
        brs endif
else
        sbc 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

        loa r2 r1 2     ; load a into r2 off of the stack
        add r3 r3 r2    ; r3 = a + mult(a, b-1)
        pop r0          ; remove first argument (a) from stack
endif
        pop r2          ; get the return address
        jmp r2          ; return
```

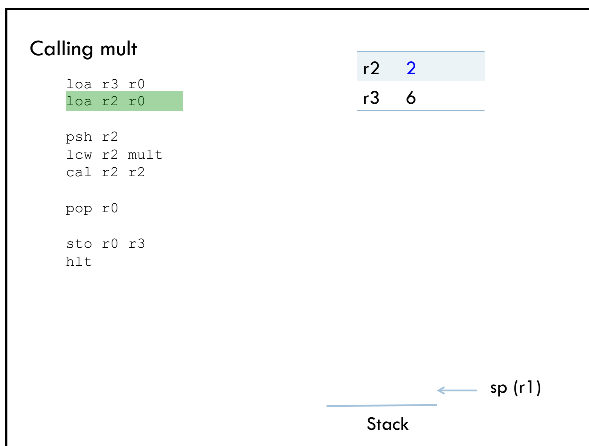
Notice symmetry of psh and pop

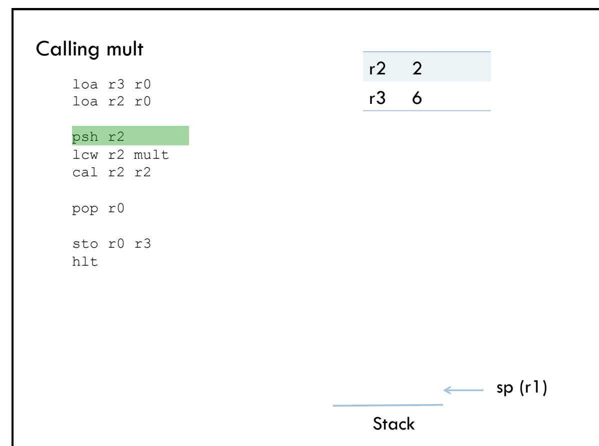


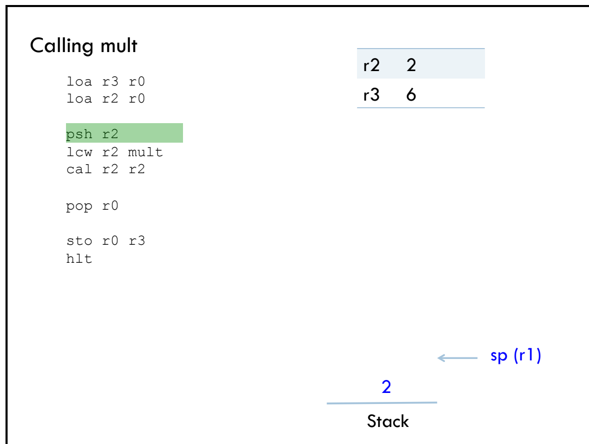


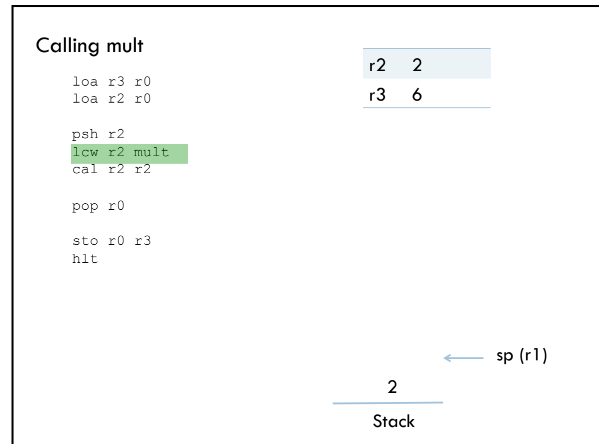

Calling mult
loa r3 r0
loa r2 r0
psh r2
lcw r2 mult
cal r2 r2
pop r0
sto r0 r3
hlt
r2
r3 6
sp (r1)
Stack

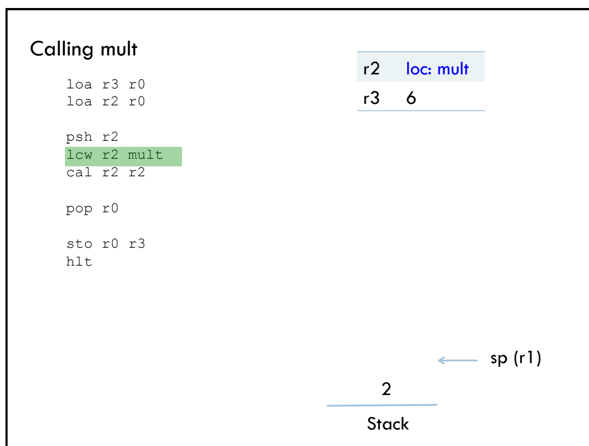

Calling mult
loa r3 r0
loa r2 r0
psh r2
lcw r2 mult
cal r2 r2
pop r0
sto r0 r3
hlt
r2
r3 6
sp (r1)
Stack

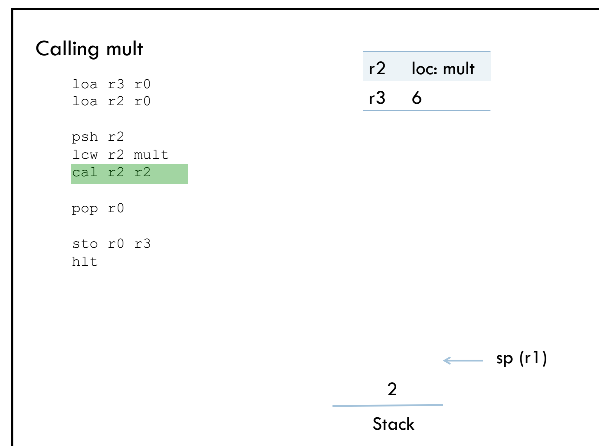

Calling mult
loa r3 r0
loa r2 r0
psh r2
lcw r2 mult
cal r2 r2
pop r0
sto r0 r3
hlt
r2 2
r3 6
sp (r1)
Stack

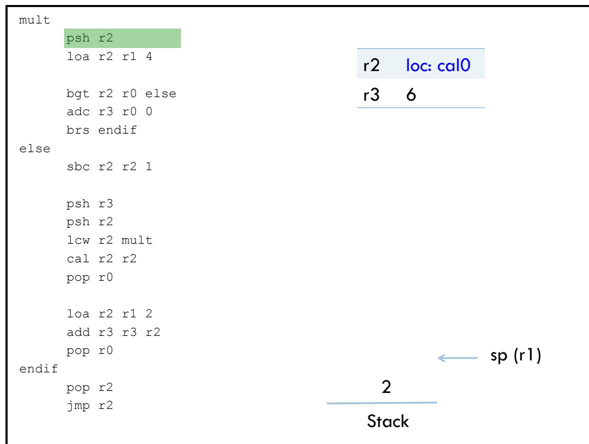

Calling mult
loa r3 r0
loa r2 r0
psh r2
lcw r2 mult
cal r2 r2
pop r0
sto r0 r3
hlt
r2 2
r3 6
sp (r1)
Stack


Calling mult
r2 2
r3 6
loa r3 r0
loa r2 r0
psh r2
lcw r2 mult
cal r2 r2
pop r0
sto r0 r3
hlt
sp (r1)
2
Stack

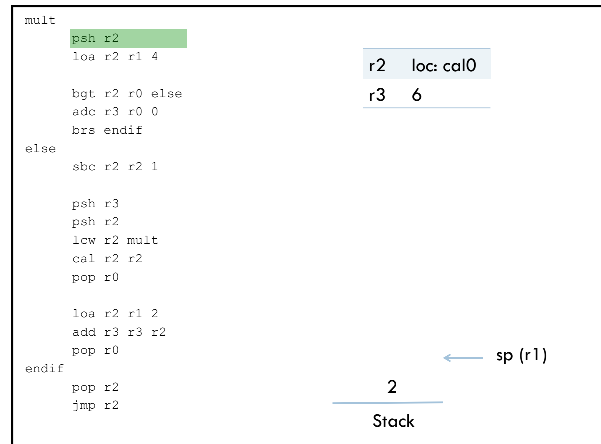

Calling mult
r2 2
r3 6
loa r3 r0
loa r2 r0
psh r2
lcw r2 mult
cal r2 r2
pop r0
sto r0 r3
hlt
sp (r1)
2
Stack


Calling mult
r2 loc: mult
r3 6
loa r3 r0
loa r2 r0
psh r2
lcw r2 mult
cal r2 r2
pop r0
sto r0 r3
hlt
sp (r1)
2
Stack

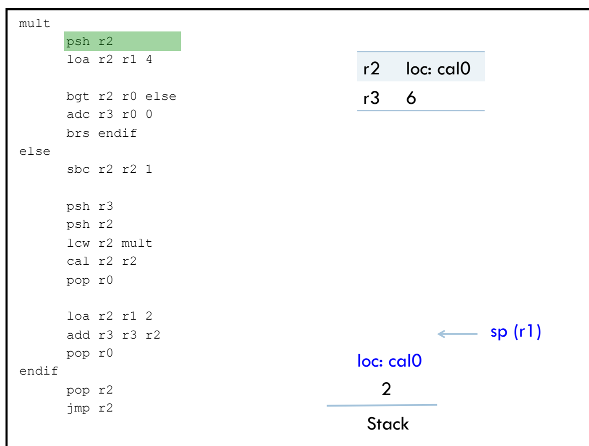

Calling mult
r2 loc: mult
r3 6
loa r3 r0
loa r2 r0
psh r2
lcw r2 mult
cal r2 r2
pop r0
sto r0 r3
hlt
sp (r1)
2
Stack



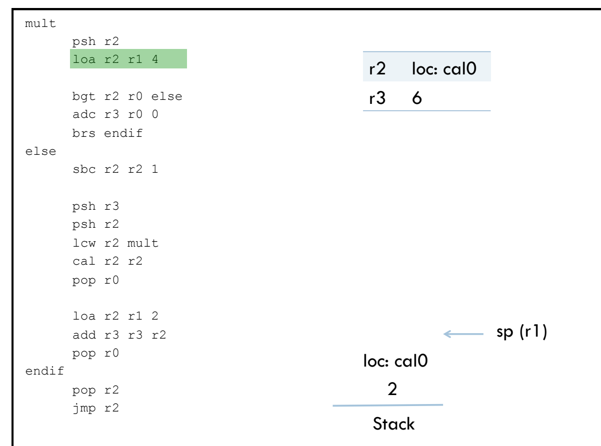
mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal0
r3 6
sp (r1)
2
Stack



mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
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
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
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
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
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
	adc r3 r0 0
	brs endif
else
	sbc r2 r2 1

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

	loa r2 r1 2
	add r3 r3 r2
	pop r0
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
	adc r3 r0 0
	brs endif
else
	sbc r2 r2 1

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

	loa r2 r1 2
	add r3 r3 r2
	pop r0
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
	adc r3 r0 0
	brs endif
else
	sbc r2 r2 1

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

	loa r2 r1 2
	add r3 r3 r2
	pop r0
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
	adc r3 r0 0
	brs endif
else
	sbc r2 r2 1

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

	loa r2 r1 2
	add r3 r3 r2
	pop r0
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
      adc r3 r0 0
      brs endif
else
      sbc r2 r2 1

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

      loa r2 r1 2
      add r3 r3 r2
      pop r0
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
      adc r3 r0 0
      brs endif
else
      sbc r2 r2 1

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

      loa r2 r1 2
      add r3 r3 r2
      pop r0
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
      adc r3 r0 0
      brs endif
else
      sbc r2 r2 1

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

      loa r2 r1 2
      add r3 r3 r2
      pop r0
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
      adc r3 r0 0
      brs endif
else
      sbc r2 r2 1

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

      loa r2 r1 2
      add r3 r3 r2
      pop r0
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!

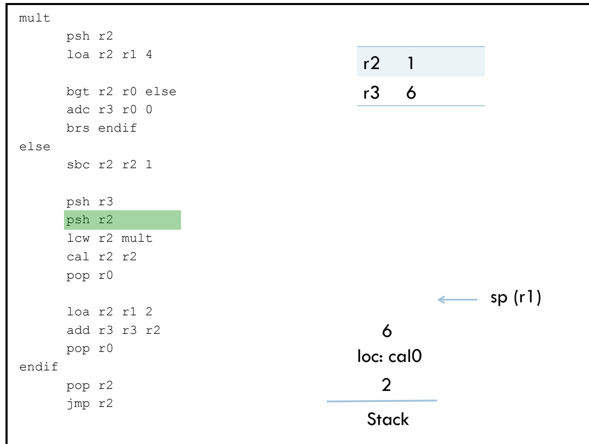
sp (r1)

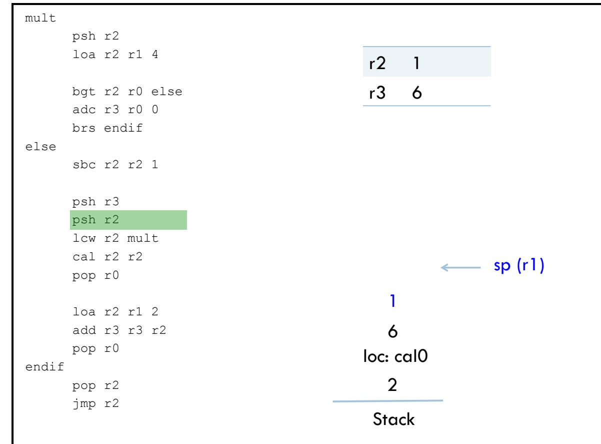
6

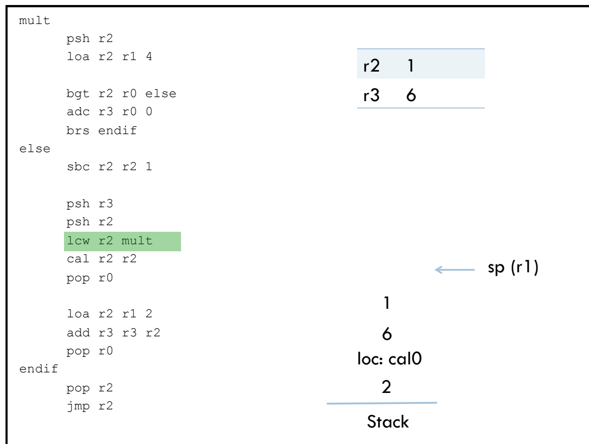
loc: cal0

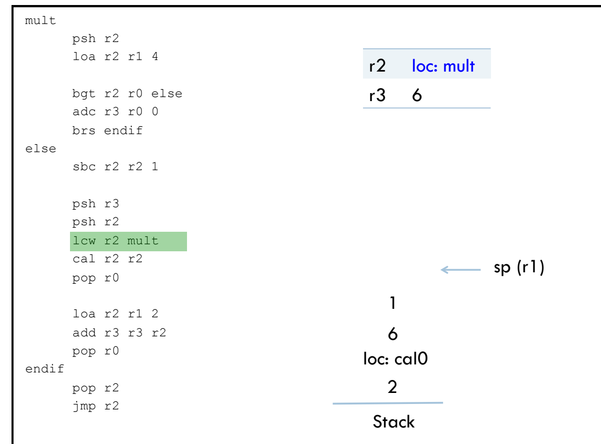
2

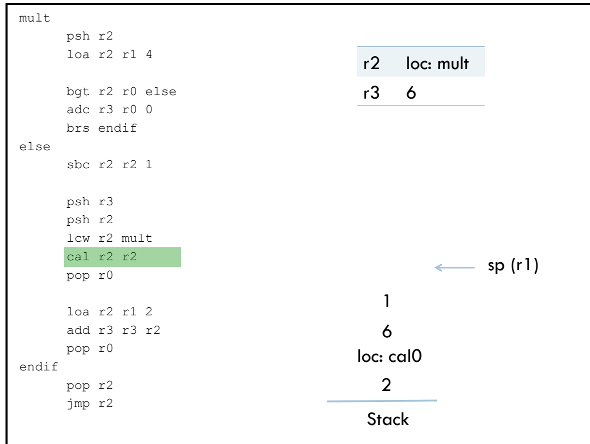
Stack

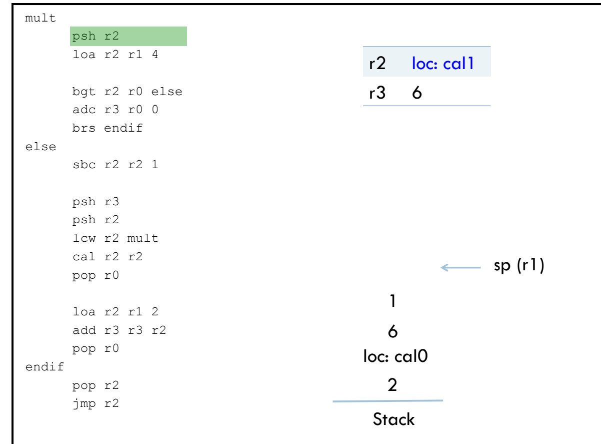

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 1
r3 6
sp (r1)
6
loc: cal0
2
Stack

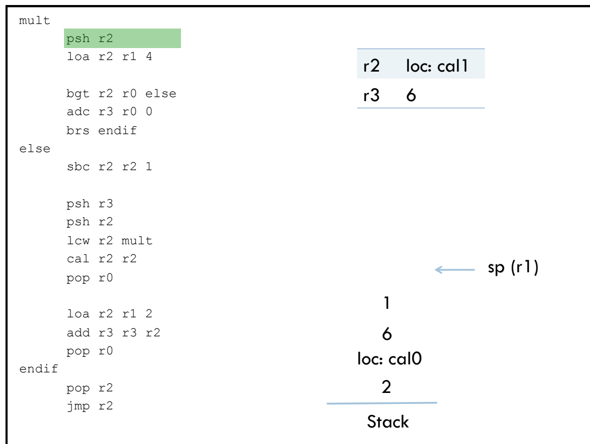

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 1
r3 6
sp (r1)
1
6
loc: cal0
2
Stack

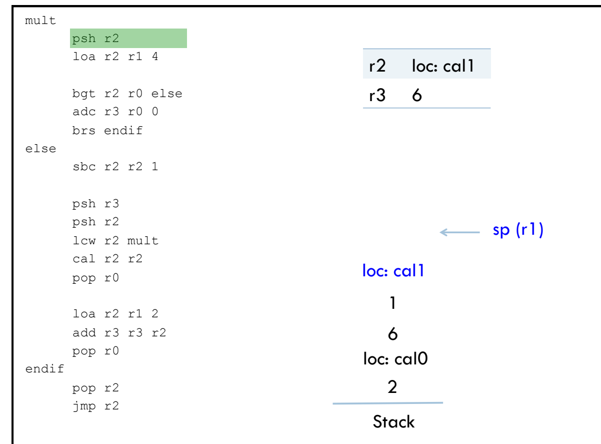

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 1
r3 6
sp (r1)
1
6
loc: cal0
2
Stack

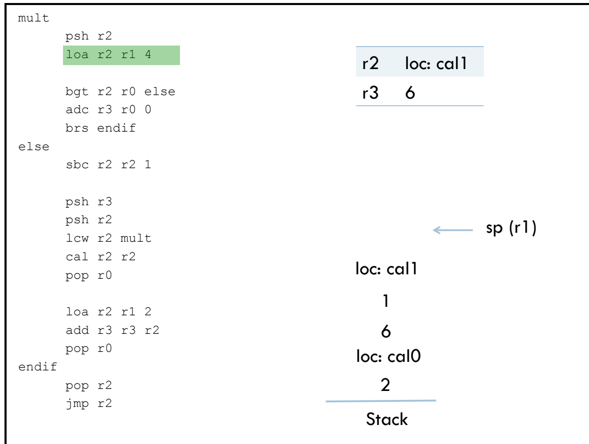

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: mult
r3 6
sp (r1)
1
6
loc: cal0
2
Stack

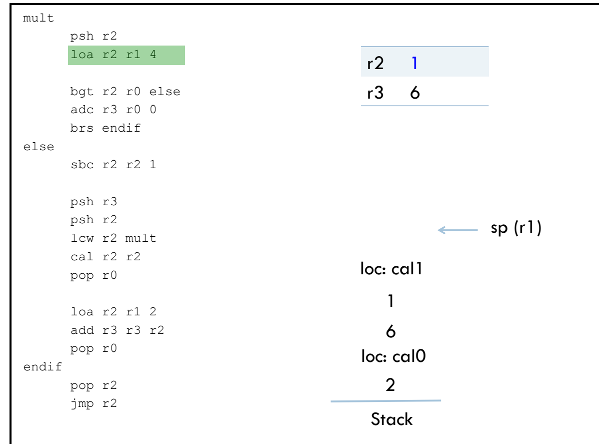

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: mult
r3 6
sp (r1)
1
6
loc: cal0
2
Stack

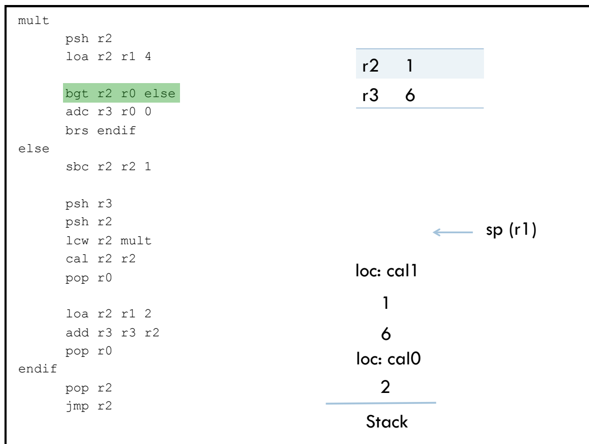

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
1
6
loc: cal0
2
Stack

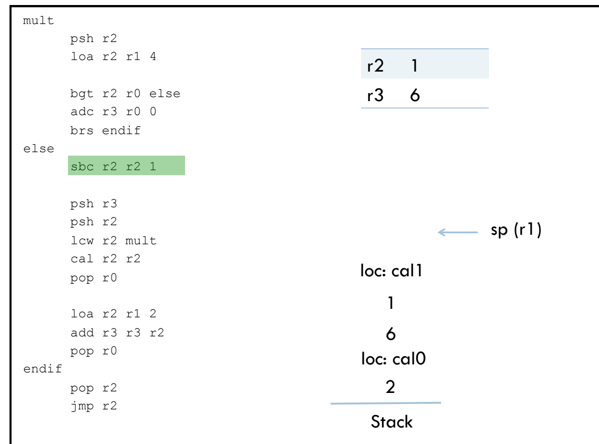

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
1
6
loc: cal0
2
Stack

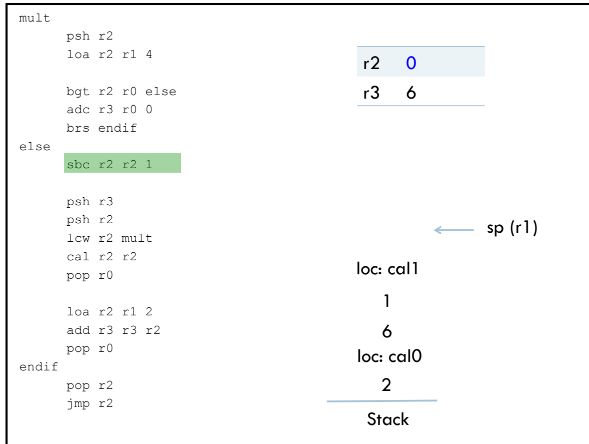

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
loc: cal1
1
6
loc: cal0
2
Stack

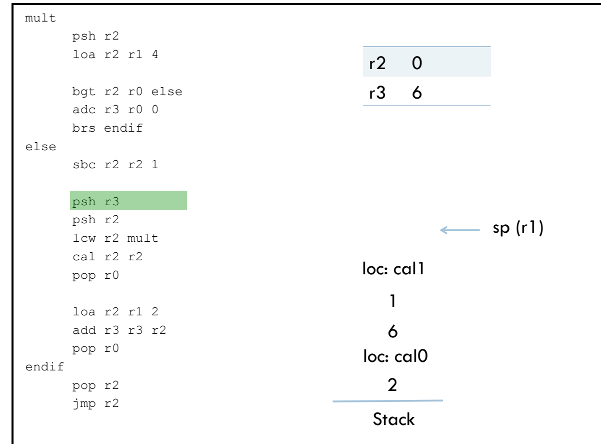

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
loc: cal1
1
6
loc: cal0
2
Stack

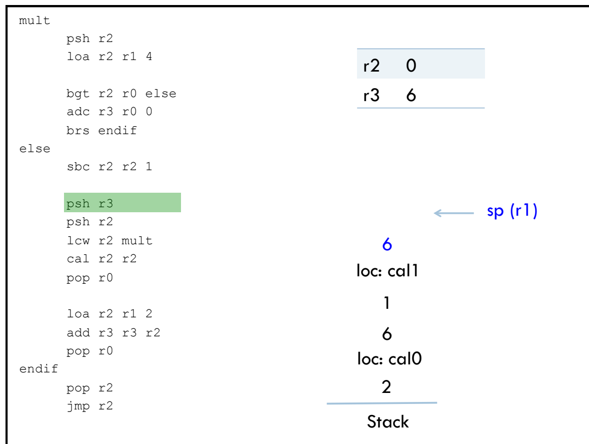

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 1
r3 6
sp (r1)
loc: cal1
1
6
loc: cal0
2
Stack

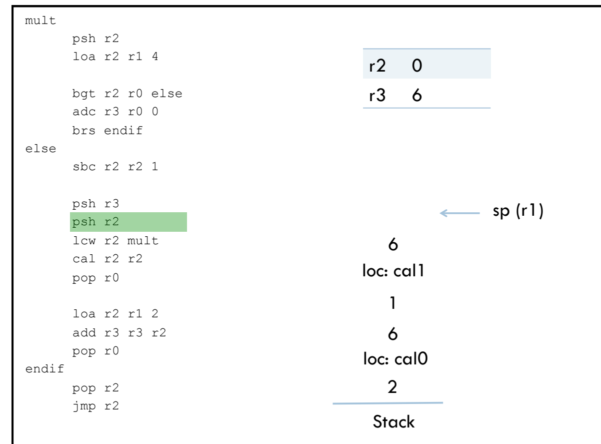

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 1
r3 6
sp (r1)
loc: cal1
1
6
loc: cal0
2
Stack

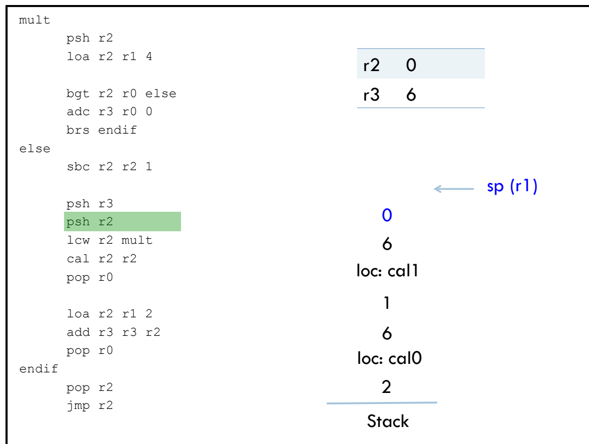

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 1
r3 6
sp (r1)
loc: cal1
1
6
loc: cal0
2
Stack

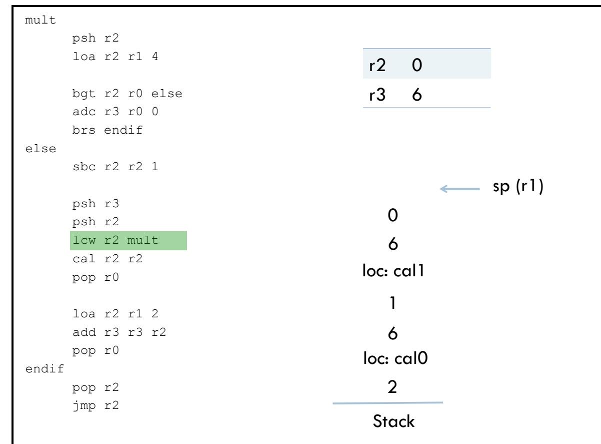

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 0
r3 6
sp (r1)
loc: cal1
1
6
loc: cal0
2
Stack

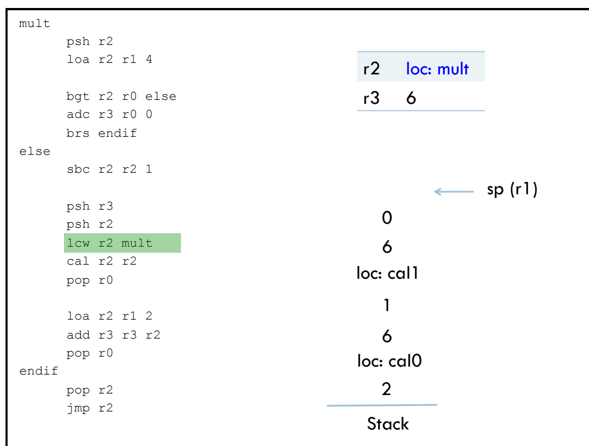

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 0
r3 6
sp (r1)
loc: cal1
1
6
loc: cal0
2
Stack

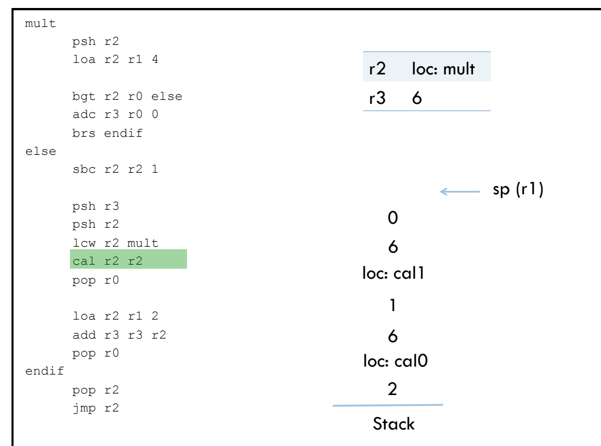

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 0
r3 6
sp (r1)
6
loc: cal1
1
6
loc: cal0
2
Stack

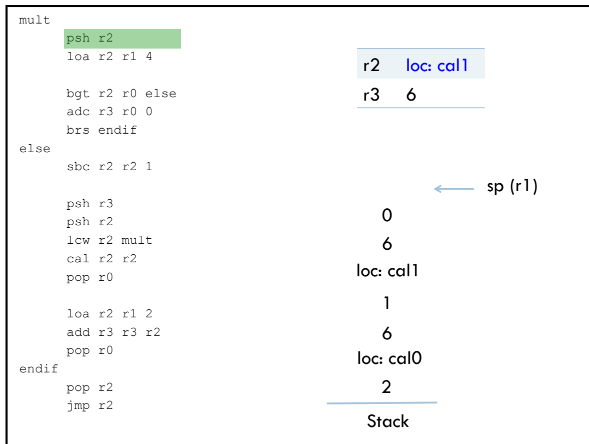

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 0
r3 6
sp (r1)
6
loc: cal1
1
6
loc: cal0
2
Stack

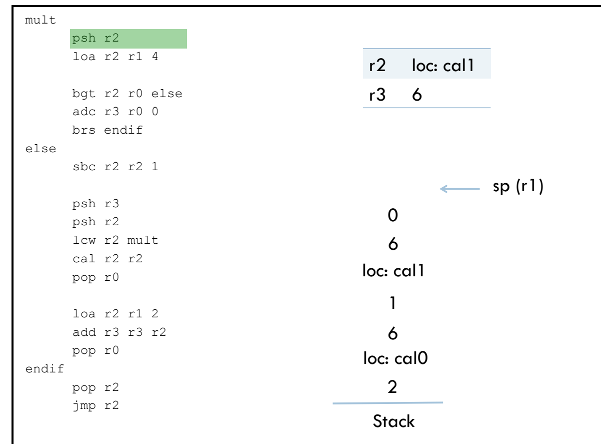

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 0
r3 6
sp (r1)
0
6
loc: cal1
1
6
loc: cal0
2
Stack

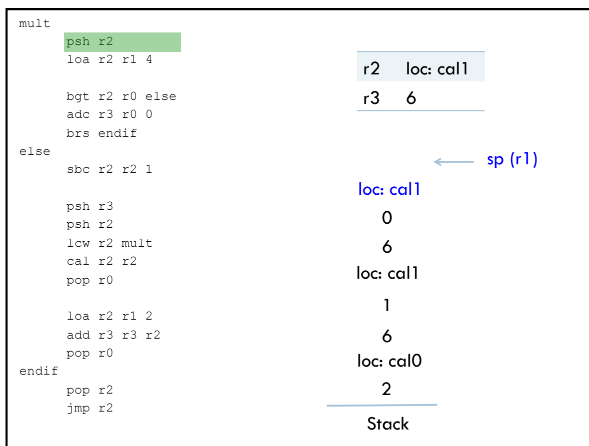

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 0
r3 6
sp (r1)
0
6
loc: cal1
1
6
loc: cal0
2
Stack

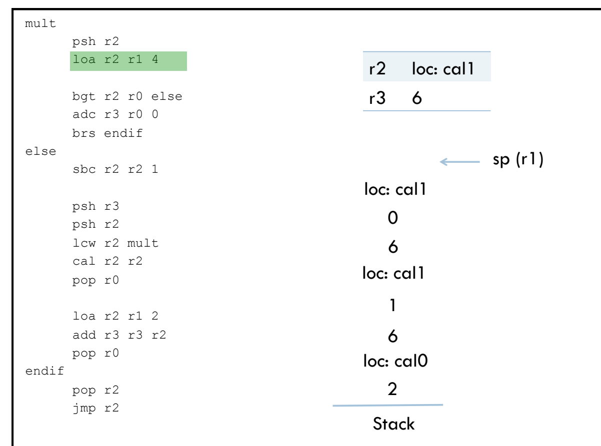

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: mult
r3 6
sp (r1)
0
6
loc: cal1
1
6
loc: cal0
2
Stack

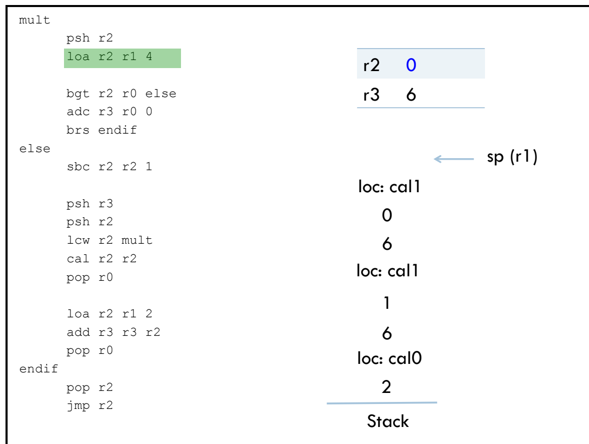

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: mult
r3 6
sp (r1)
0
6
loc: cal1
1
6
loc: cal0
2
Stack

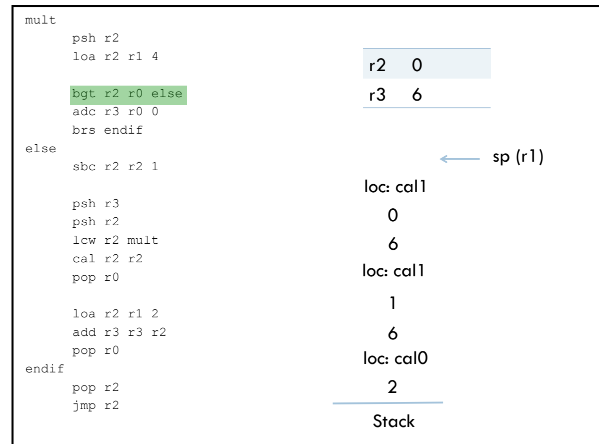

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
0
6
loc: cal1
1
6
loc: cal0
2
Stack

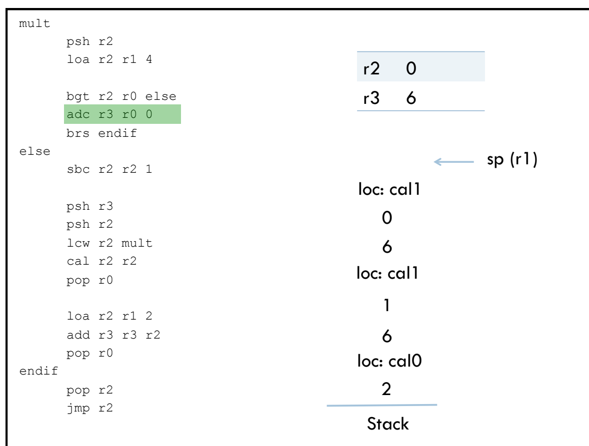

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
0
6
loc: cal1
1
6
loc: cal0
2
Stack

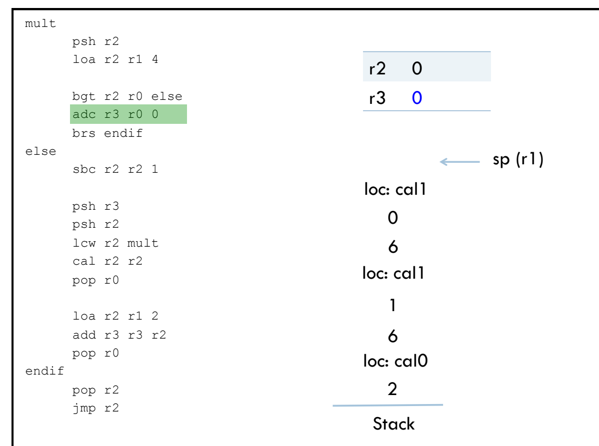

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2
Stack

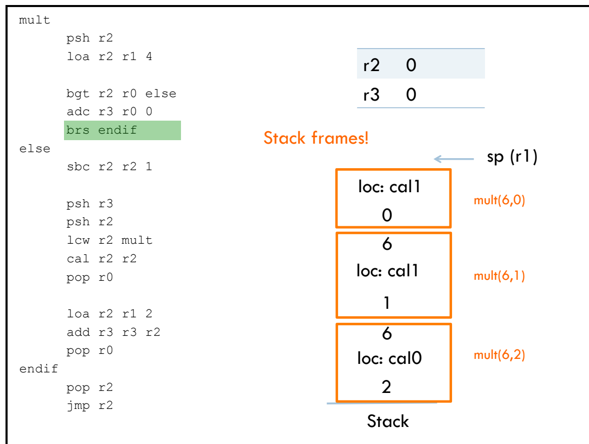

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2
Stack


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

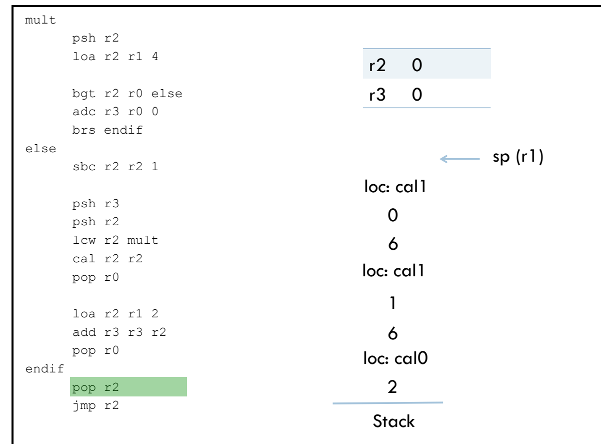

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 0
r3 6
sp (r1)
loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2
Stack


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

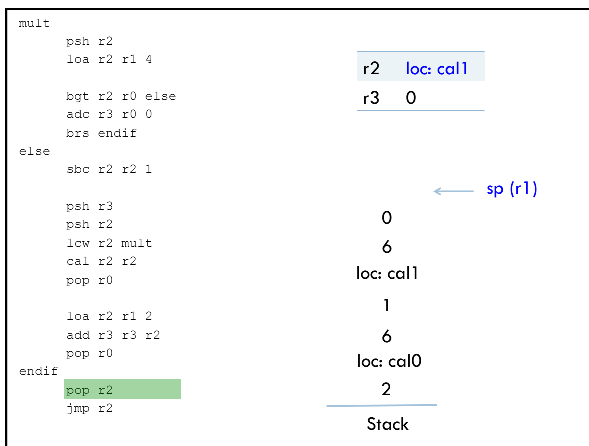

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 0
r3 0
sp (r1)
loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2
Stack



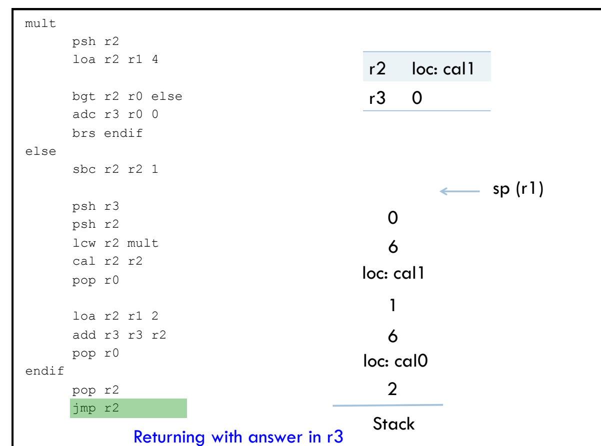
mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 0
r3 0
Stack frames!
sp (r1)
loc: cal1
0
mult(6,0)
6
loc: cal1
1
mult(6,1)
6
loc: cal0
2
mult(6,2)
Stack



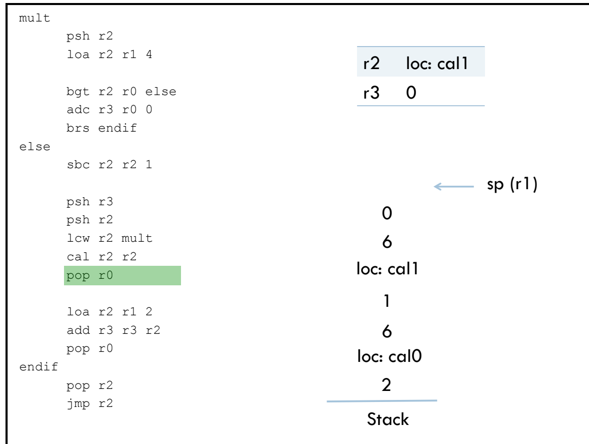
mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 0
r3 0
sp (r1)
loc: cal1
0
6
loc: cal1
1
6
loc: cal0
2
Stack

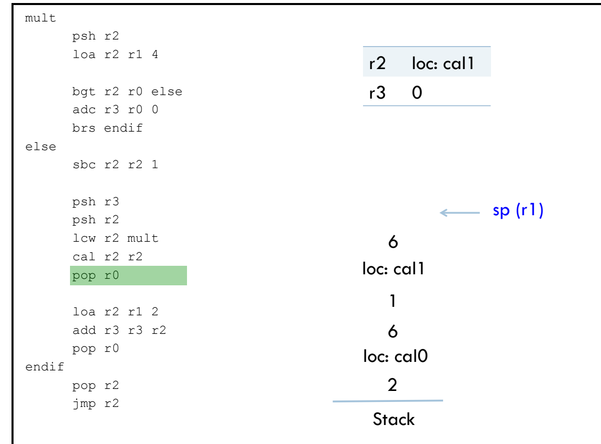


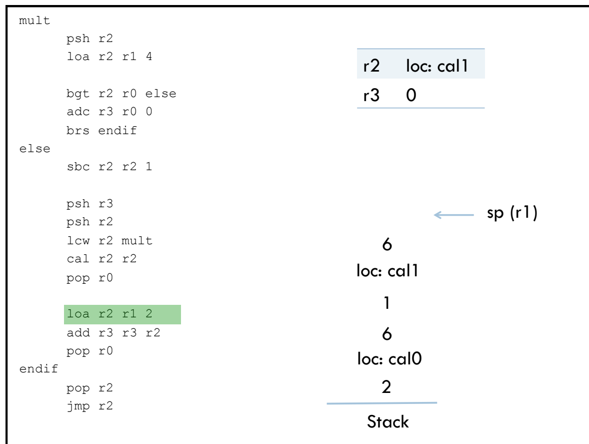
mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 0
sp (r1)
0
6
loc: cal1
1
6
loc: cal0
2
Stack

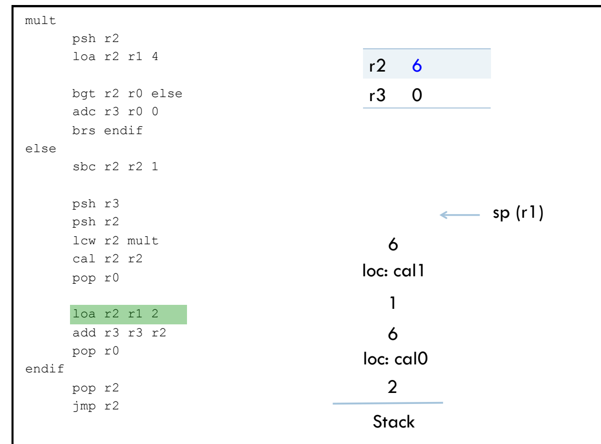


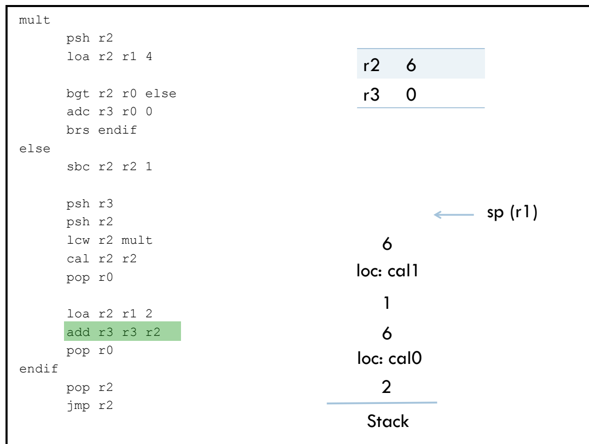
mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 0
sp (r1)
0
6
loc: cal1
1
6
loc: cal0
2
Stack
Returning with answer in r3

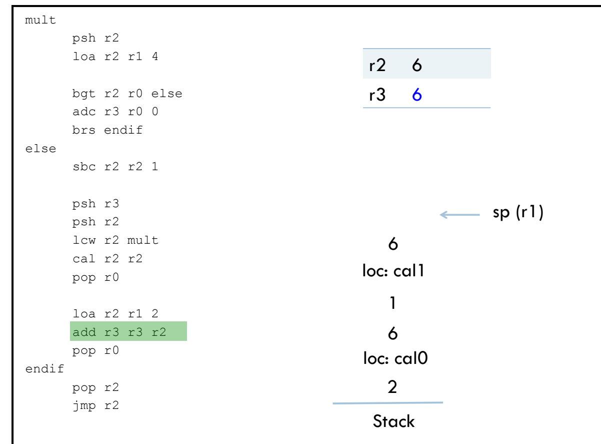

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 0
sp (r1)
0
6
loc: cal1
1
6
loc: cal0
2
Stack

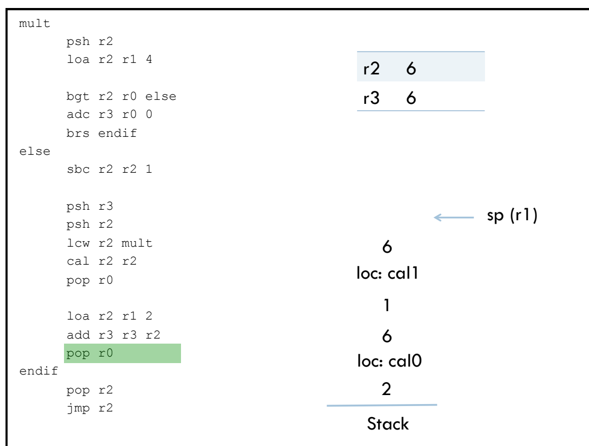

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 0
sp (r1)
6
loc: cal1
1
6
loc: cal0
2
Stack

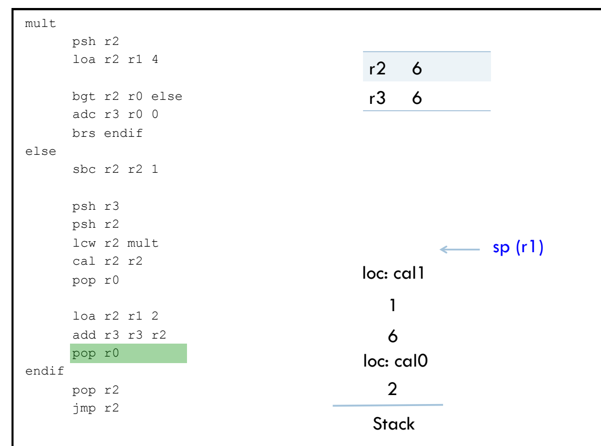

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 0
sp (r1)
6
loc: cal1
1
6
loc: cal0
2
Stack

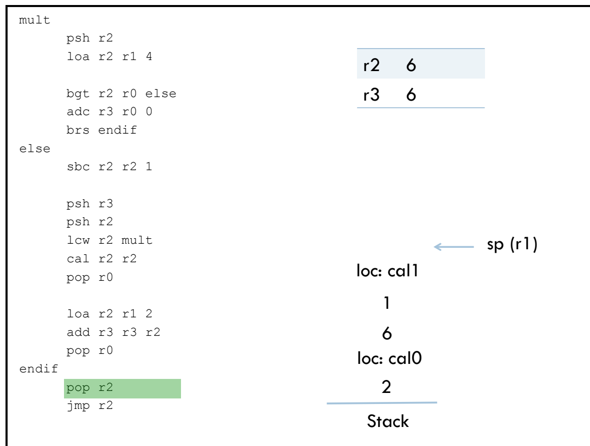

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 0
sp (r1)
6
loc: cal1
1
6
loc: cal0
2
Stack

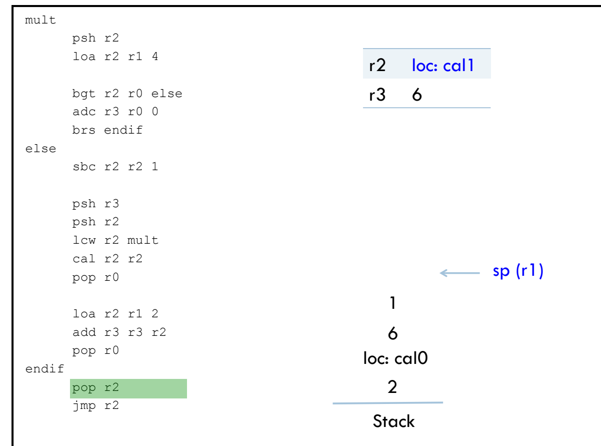

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 0
sp (r1)
6
loc: cal1
1
6
loc: cal0
2
Stack

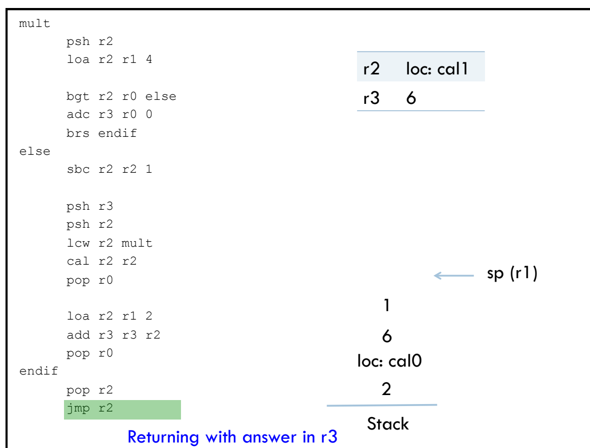

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 6
sp (r1)
6
loc: cal1
1
6
loc: cal0
2
Stack

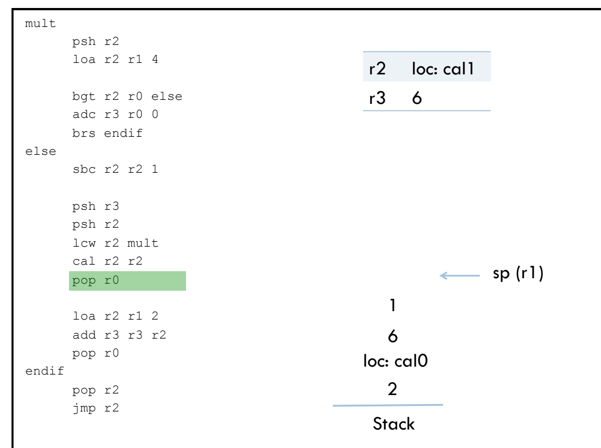

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 6
sp (r1)
6
loc: cal1
1
6
loc: cal0
2
Stack

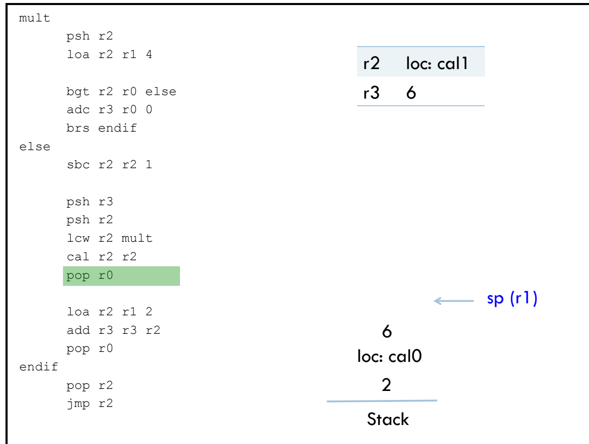

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 6
sp (r1)
loc: cal1
1
6
loc: cal0
2
Stack

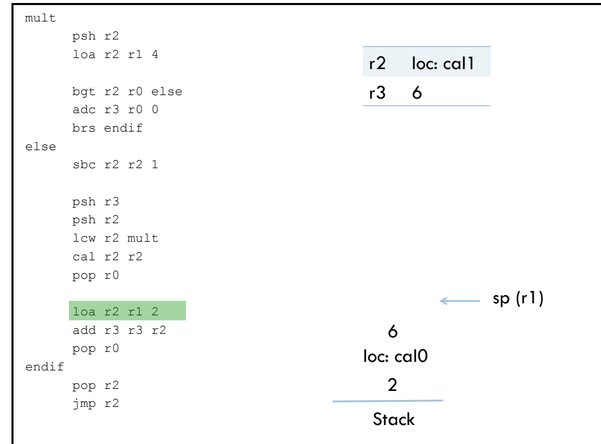

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 6
sp (r1)
loc: cal1
1
6
loc: cal0
2
Stack

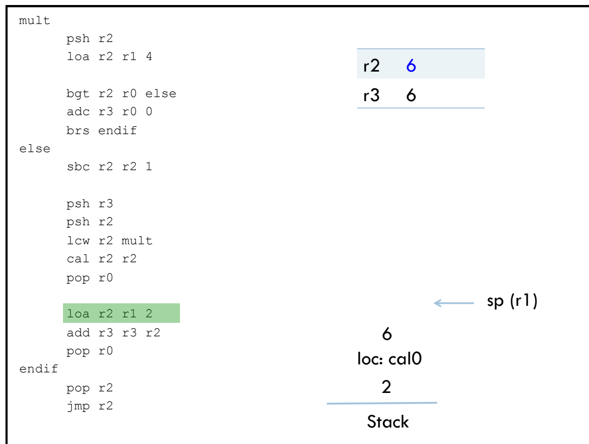

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
1
6
loc: cal0
2
Stack

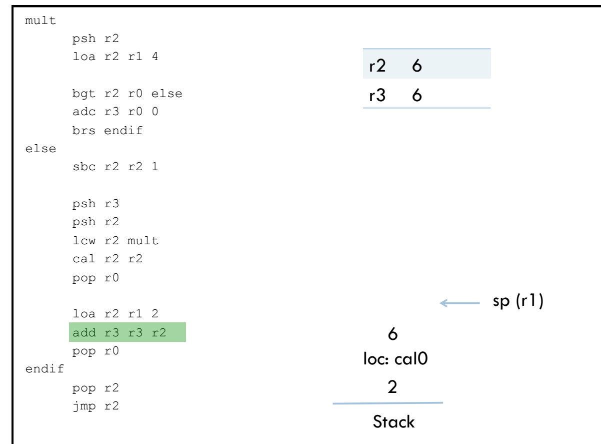

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
1
6
loc: cal0
2
Stack
Returning with answer in r3

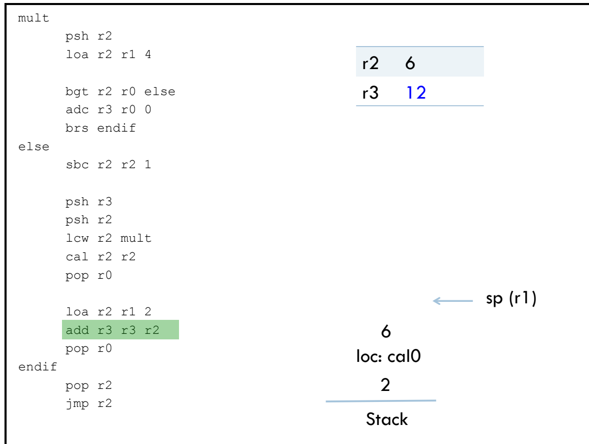

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
1
6
loc: cal0
2
Stack

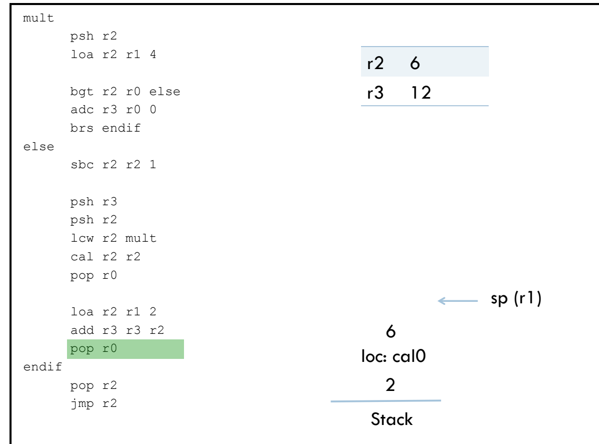

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
6
loc: cal0
2
Stack

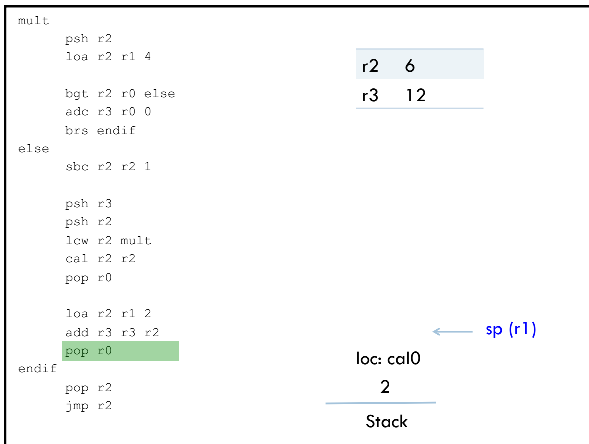

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal1
r3 6
sp (r1)
6
loc: cal0
2
Stack

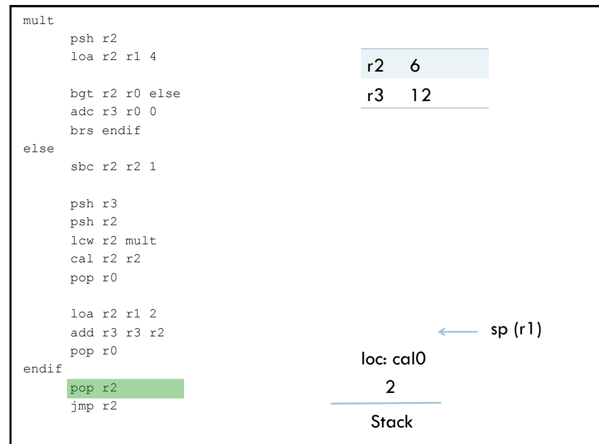

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 6
sp (r1)
6
loc: cal0
2
Stack

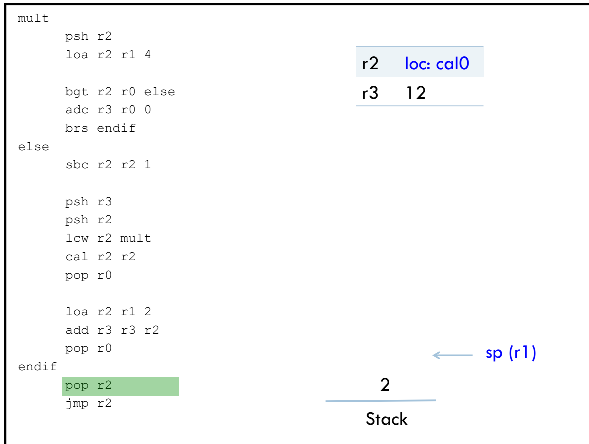

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 6
sp (r1)
6
loc: cal0
2
Stack

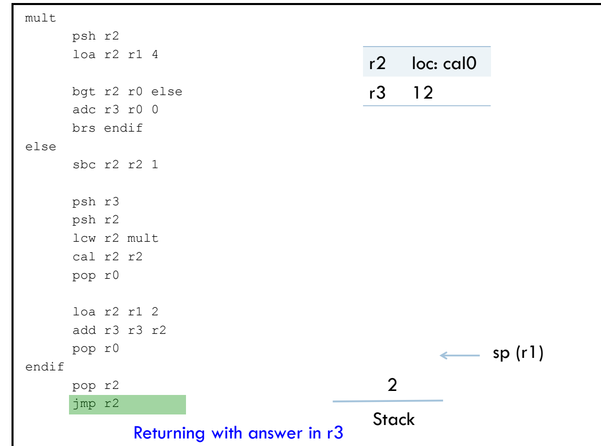

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 12
sp (r1)
6
loc: cal0
2
Stack

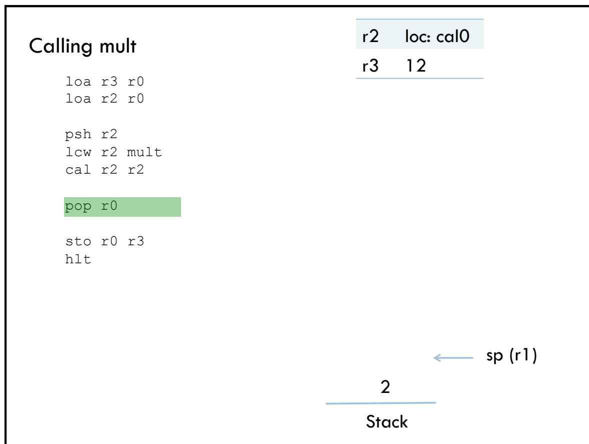

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 12
sp (r1)
6
loc: cal0
2
Stack

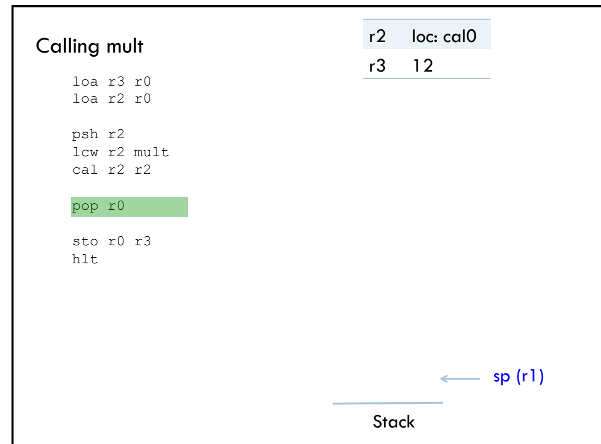

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 12
sp (r1)
loc: cal0
2
Stack

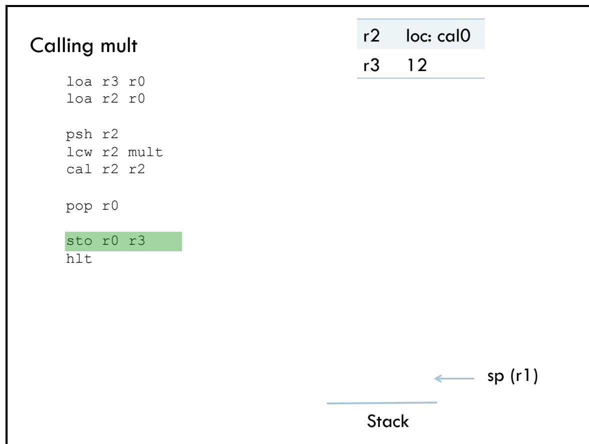

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 6
r3 12
sp (r1)
loc: cal0
2
Stack

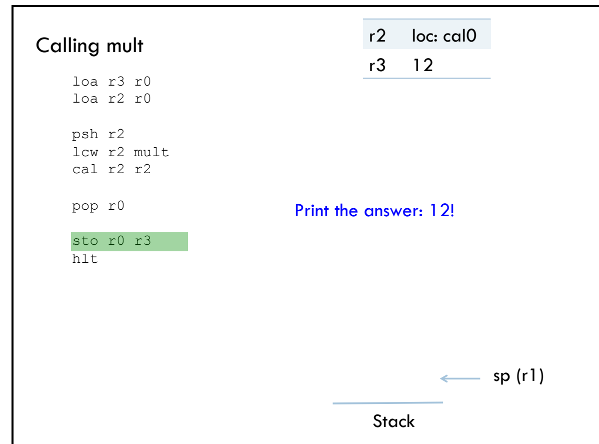

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal0
r3 12
sp (r1)
2
Stack

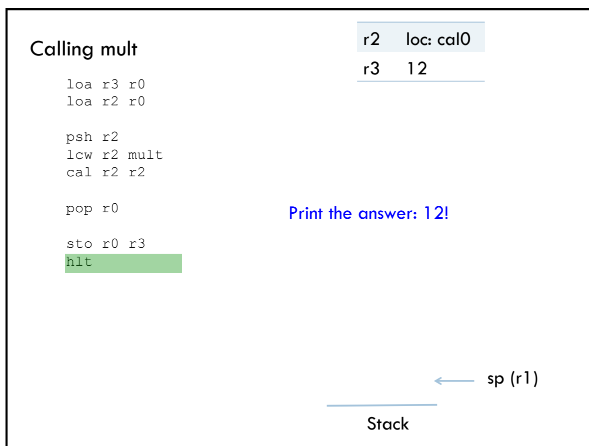

mult
psh r2
loa r2 r1 4
bgt r2 r0 else
adc r3 r0 0
brs endif
else
sbc r2 r2 1
psh r3
psh r2
lcw r2 mult
cal r2 r2
pop r0
loa r2 r1 2
add r3 r3 r2
pop r0
endif
pop r2
jmp r2
r2 loc: cal0
r3 12
sp (r1)
2
Stack
Returning with answer in r3

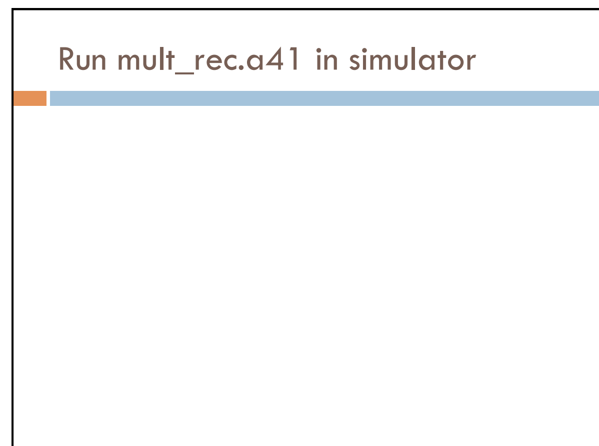

Calling mult
loa r3 r0
loa r2 r0
psh r2
lcw r2 mult
cal r2 r2
pop r0
sto r0 r3
hlt
r2 loc: cal0
r3 12
sp (r1)
2
Stack


Calling mult
loa r3 r0
loa r2 r0
psh r2
lcw r2 mult
cal r2 r2
pop r0
sto r0 r3
hlt
r2 loc: cal0
r3 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 r0 r3
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
Print the answer: 12!
sto r0 r3
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
Print the answer: 12!
sto r0 r3
hlt
sp (r1)
Stack


Run mult_rec.a41 in simulator

CS41B 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/