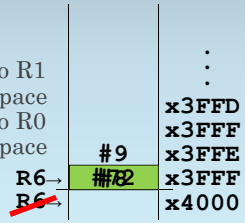
To Multiply: Pop Twice, Multiply, Push Product

```
STACKMULT
LDR R1,R6,#0     ; pop 9 into R1
ADD R6,R6,#1     ; remove space
LDR R0,R6,#0     ; pop 8 into R0
ADD R6,R6,#1     ; remove space
JSR MULT         ; R0 is 72
ADD R6,R6,#-1    ; push R0
STR R0,R6,#0
```



Use the same instructions as before!

That's it!

Subroutine Can Mean More than Just Adding RET

```
STACKMULT
LDR R1,R6,#0     ; pop 9 into R1
ADD R6,R6,#1     ; remove space
LDR R0,R6,#0     ; pop 8 into R0
ADD R6,R6,#1     ; remove space
JSR MULT         ; R0 is 72
ADD R6,R6,#-1    ; push R0
STR R0,R6,#0

RET
```

But what if we want a subroutine?

Good enough?

NO!

A Subroutine that Uses JSR or TRAP Must Protect R7

```
STACKMULT
; R7 has the return address here.
LDR R1,R6,#0     ; pop 9 into R1
ADD R6,R6,#1     ; remove space
LDR R0,R6,#0     ; pop 8 into R0
ADD R6,R6,#1     ; remove space
JSR MULT         ; R0 is 72
ADD R6,R6,#-1    ; push R0
STR R0,R6,#0

RET
```

Where does R7 point after JSR?

Here.

So RET creates a loop…

Add a Space with a Label, then Save and Restore R7

```
STACKMULT
ST R7,SM_R7      ; save R7
LDR R1,R6,#0     ; pop 9 into R1
ADD R6,R6,#1     ; remove space
LDR R0,R6,#0     ; pop 8 into R0
ADD R6,R6,#1     ; remove space
JSR MULT         ; R0 is 72
ADD R6,R6,#-1    ; push R0
STR R0,R6,#0
LD R7,SM_R7      ; restore R7
RET
SM_R7 .BLKW #1   ; space for R7
```

Now the subroutine is complete.

