Assume we start with r0 = 17, r1 = 3

```
loop:
    subs r0, r1
    addmi r0, r1
    bpl loop
```

After this runs, we'll have:

A: r0 = 3

B: r0 = 2

C: r0 = 5

D: r0 = 6

E: Something else

```
mod:

loop:
   subs r0, r1
   addmi r0, r1
   bpl loop
```

Assume we start with r0 = 17, r1 = 3

After this runs, we'll have:

A: `r0 = 3`

B: `r0 = 2`

C: `r0 = 5`

D: `r0 = 6`

E: Something else

Today's lecture:

I like mod, and don't want to copy/paste it everywhere I use it.

What will be in r0 at end?

A: 0
B: 1
C: 2
D: 8
E: Something else

	N	Z	C	V	
cpsr	1				

r0	~~5~~ ~~3~~ ~~1~~ ~~-2~~ 1
r1	~~1~~ 2
r2	0x5
r15(pc)	~~isprime~~ ~~mod~~ loop+8

```
mod:
loop:      subs r0, r1
           addmi r0, r1
           bpl loop

isprime:   mov r1, #1      @ The current factor
loop2:     add r1, #1      @ Increment the current factor (start at 2)
           cmp r1, r2      @ If the current factor is bigger
           bge prime       @   then we've tried everything, it's prime
           mov r0, r2      @ Get our test value into r0, then
           b mod           @   compute r0 % r1 using mod
           cmp r0, #0      @   if that value is 0
           beq notprime    @   then we know the number isn't prime
           b loop2         @ Otherwise, keep incrementing
prime:     mov r0, #1      @ Store #1 in r0 for "yes it's prime" ~ true
           b end
notprime:  mov r0, #0      @ Store #0 in r0 for "no it's not" ~ false
end:
```

Where do we want the pc to go after the evaluation of mod is complete?

A: `loop (add r1, #1)`

B: `loop + 20 (cmp r1 0, #0)`

C: `prime`

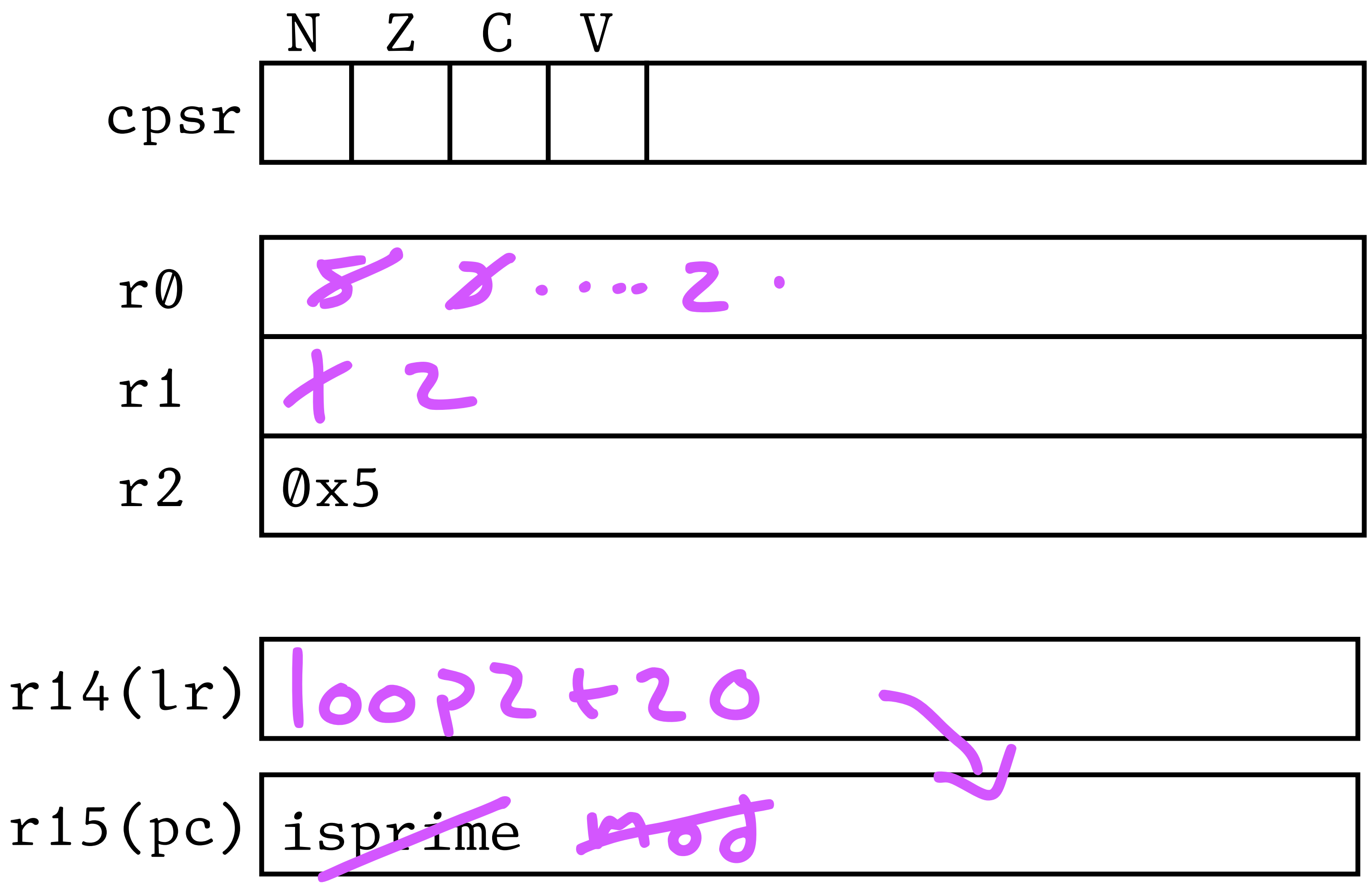
D: Somewhere else

	N	Z	C	V	
cpsr					

r0	
r1	
r2	0x5

r15(pc)	isprime

```
mod:
loop:         subs r0, r1
              addmi r0, r1
              bpl loop
              b ret
isprime:      mov r1, #1      @ The current factor
loop:   A     add r1, #1      @ Increment the current factor (start at 2)
              cmp r1, r0      @ If the current factor is bigger
              bge prime       @   then we've tried everything, it's prime
save ------>  mov r0, r2      @ Get our test value into r0, then
              b mod           @   compute r0 % r1 using mod
  pc    B     cmp r0, #0      @   if that value is 0
ret:          beq notprime    @   then we know the number isn't prime
              b loop          @ Otherwise, keep incrementing
prime:        mov r0, #1      @ Store #1 in r0 for "yes it's prime" ~ true
              b end
notprime:     mov r0, #0      @ Store #0 in r0 for "no it's not" ~ false
end:
```


cpsr: N Z C V

r0: ~~8~~ ~~3~~ .... 2 .

r1: ~~1~~ 2

r2: 0x5

r14(lr): loop2+20

r15(pc): ~~isprime~~ ~~mod~~

```
mod:
loop:       subs r0, r1
            addmi r0, r1
            bpl loop
            mov pc, lr

isprime:    mov r1, #1      @ The current factor
loop2:      add r1, #1      @ Increment the current factor (start at 2)
            cmp r1, r2      @ If the current factor is bigger
            bge prime       @   than we've tried everything, it's prime
            mov r0, r2      @ Get our test value into r0, then
            bl mod          @   compute r0 % r1 using mod
            cmp r0, #0      @   if that value is 0
            beq notprime    @   then we know the number isn't prime
            b loop2         @ Otherwise, keep incrementing
prime:      mov r0, #1      @ Store #1 in r0 for "yes it's prime" ~ true
            b end
notprime:   mov r0, #0      @ Store #0 in r0 for "no it's not" ~ false
end:
```

`bl <label>`

Store the next instruction address in `r14 (lr)`
Then, branch to `<label>`

Very much related to "call a function"

Next challenge: Implement `findprime`, which searches for the first prime, starting at the value in r3. Use `isprime` as a helper to do it.

Question!

(How) will `isprime` need to change?

A: It will work as-is

B: It will need to use **`mov pc, lr`** at the end just like `mod` did

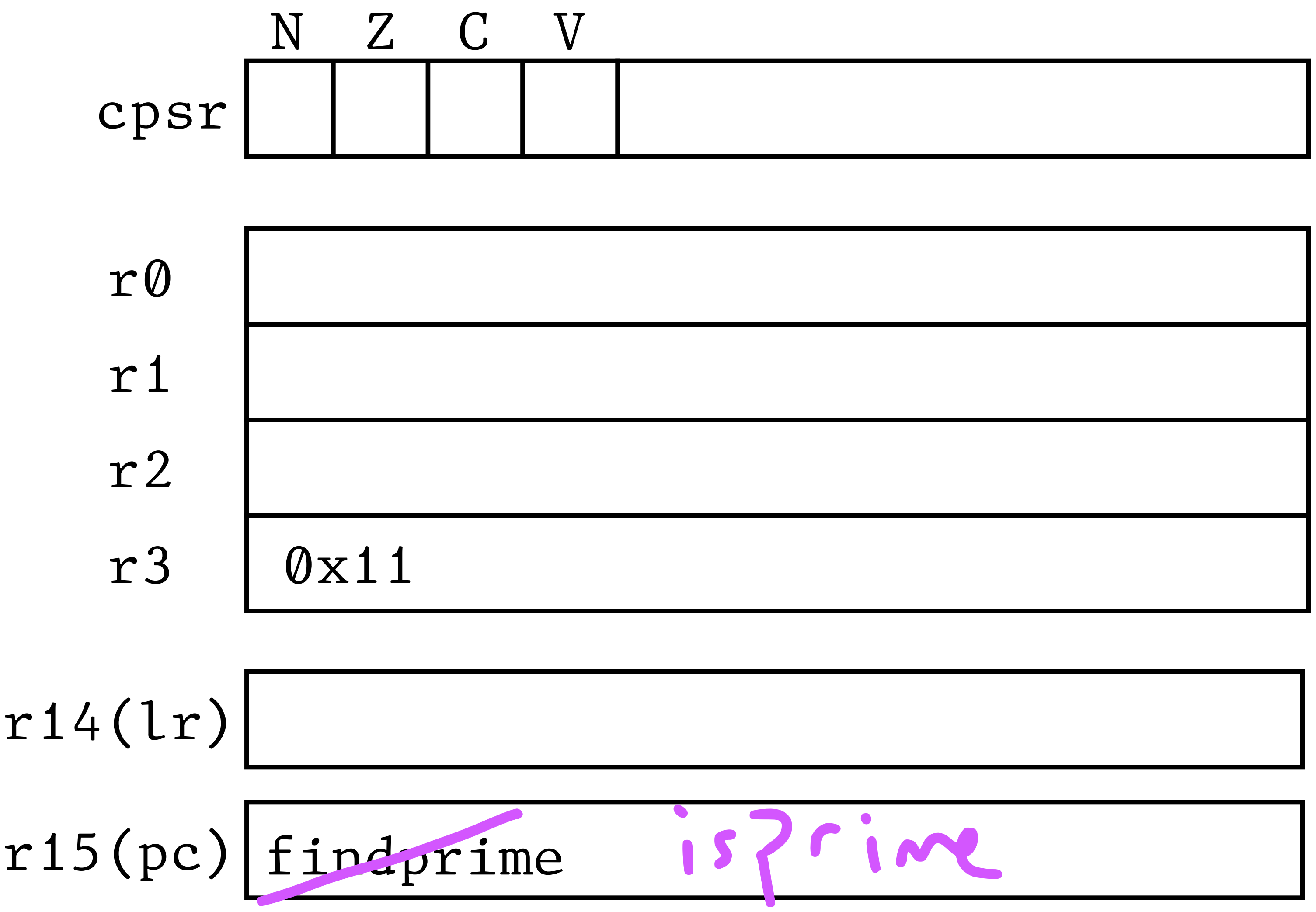
C: It will need to change in some other way (possibly including B)

```
mod:       subs r0, r1
           addmi r0, r1
           bpl mod
           mov pc, lr

isprime:   mov r1, #1
loop:      add r1, #1
           cmp r1, r2
           bge prime
           mov r0, r2
           bl mod
           cmp r0, #0
           beq notprime
           b loop
prime:     mov r2, #1
           b end
notprime:  mov r2, #0
end:

findprime:
loop2:     mov r2, r3
           bl isprime
           cmp r0, #1
           beq foundit
           addne r3, #1
           bne loop2

foundit:   mov r0, r3
@ (compute the first prime larger than the value in r3, store in r0)
```



	N	Z	C	V	
cpsr					

r0	
r1	
r2	
r3	0x11
r14(lr)	
r15(pc)	~~findprime~~ isprime

```
mod:       subs r0, r1
           addmi r0, r1
           bpl mod
           mov pc, lr

isprime:   mov r1, #1
loop:      add r1, #1
           cmp r1, r2
           bge prime
           mov r0, r2
           bl mod
           cmp r0, #0
           beq notprime
           b loop
prime:     mov r2, #1
           b end
notprime:  mov r2, #0
end:       mov pc, lr

findprime:
loop2:     mov r2, r3
           bl isprime
           cmp r0, #1
           beq foundit
           addne r3, #1
           bne loop2

foundit:   mov r0, r3
@ (compute the first prime larger than the value in r3, store in r0)
```

	N	Z	C	V	
cpsr					

r0	 1
r1	
r2	11
r3	0x11
r14(lr)	~~loop2+1C~~ loop+20
r15(pc)	~~findprime~~ ~~isprime~~ ~~mod~~ ~~loop+20~~ ~~mod~~ ~~loop+20~~ mod

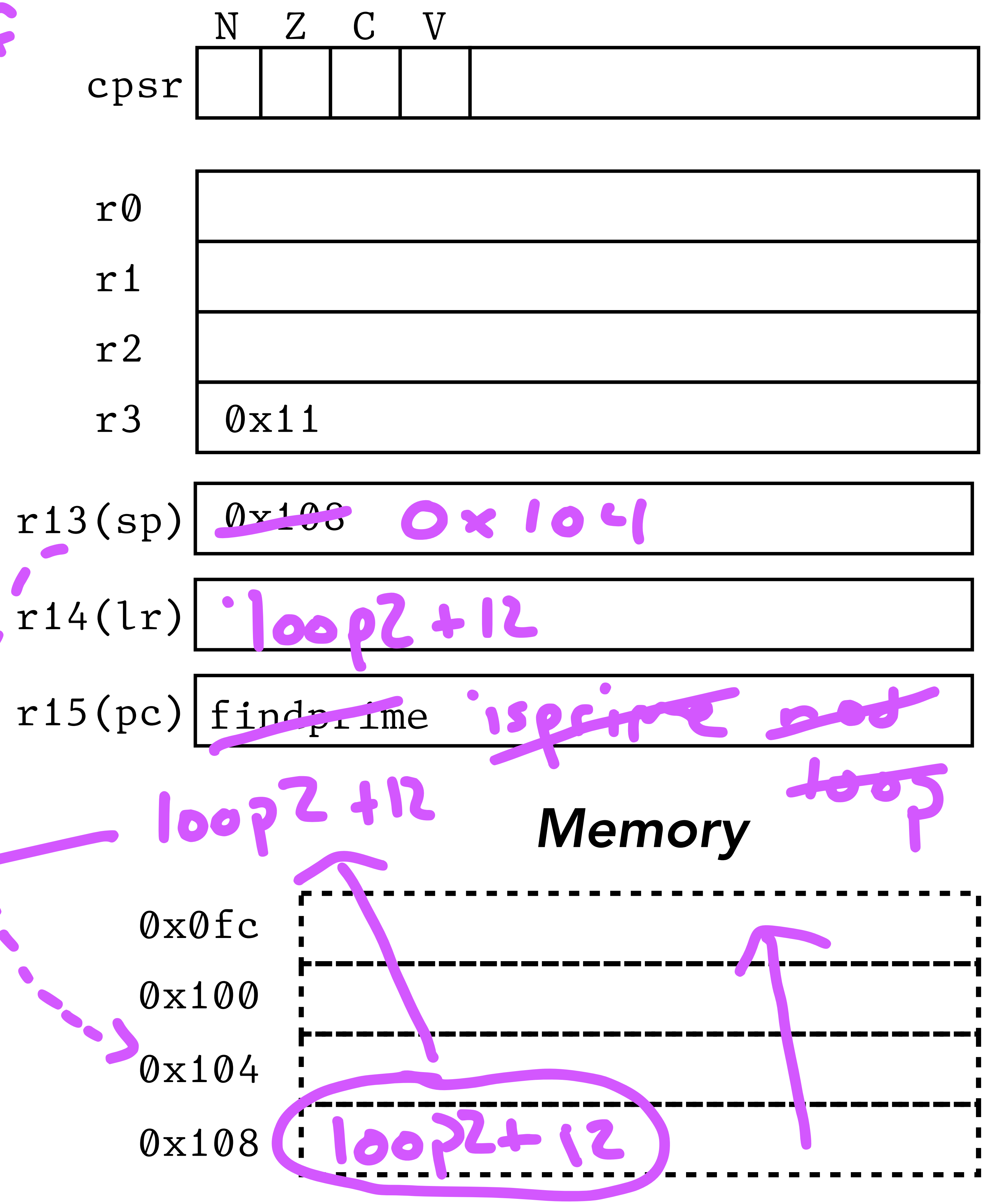
```
mod:        subs r0, r1
            addmi r0, r1
            bpl mod
            mov pc, lr

isprime:    push {lr}
            mov r1, #1
loop:       add r1, #1
            cmp r1, r2
            bge prime
            mov r0, r2
            bl mod
            cmp r0, #0
            beq notprime
            b loop
prime:      mov r2, #1
            b end
notprime:   mov r2, #0
end:        pop {pc}

findprime:
loop2:      mov r2, r3
            bl isprime
            cmp r0, #1
            beq foundit
            addne r3, #1
            bne loop2

foundit:    mov r0, r3
@ (compute the first prime larger than the value in r3, store in r0)
```

leaf call

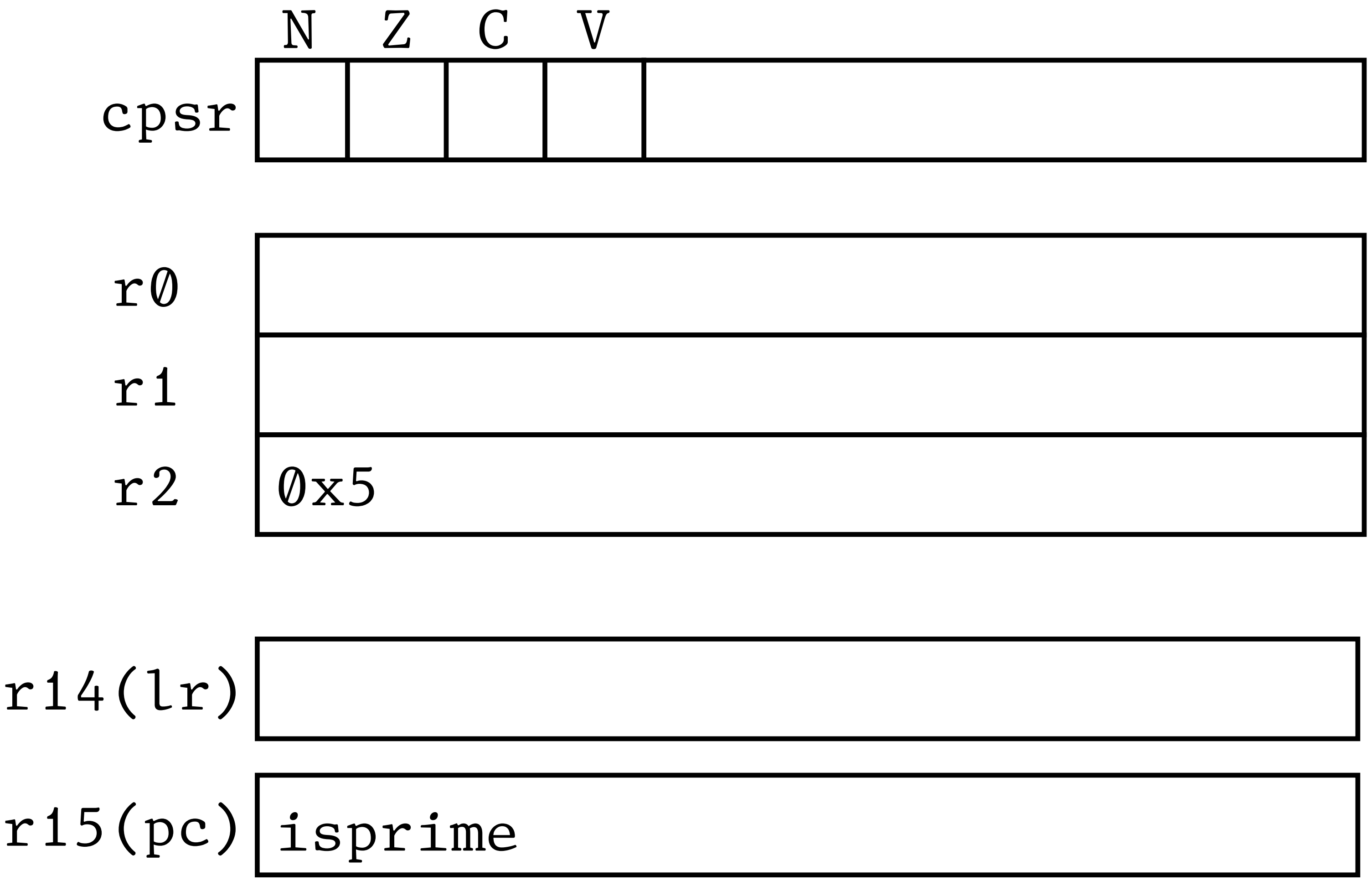


```
mod:       subs r0, r1
           addmi r0, r1
           bpl mod
           mov pc, lr

isprime:   mov r1, #1
loop:      add r1, #1
           cmp r1, r2
           bge prime
           mov r0, r2
           bl mod
           cmp r1, #0
           beq notprime
           b loop
prime:     mov r2, #1
           b end
notprime:  mov r2, #0
end:       mov pc, lr

findprime: @ compute the first prime larger than the value in r3, store in r0
loop:      mov r2, r3
           bl isprime
           cmp r0, #1
           beq foundit
           addne r3, #1
           bne loop

foundit:   mov r0, r3
```



N Z C V

cpsr

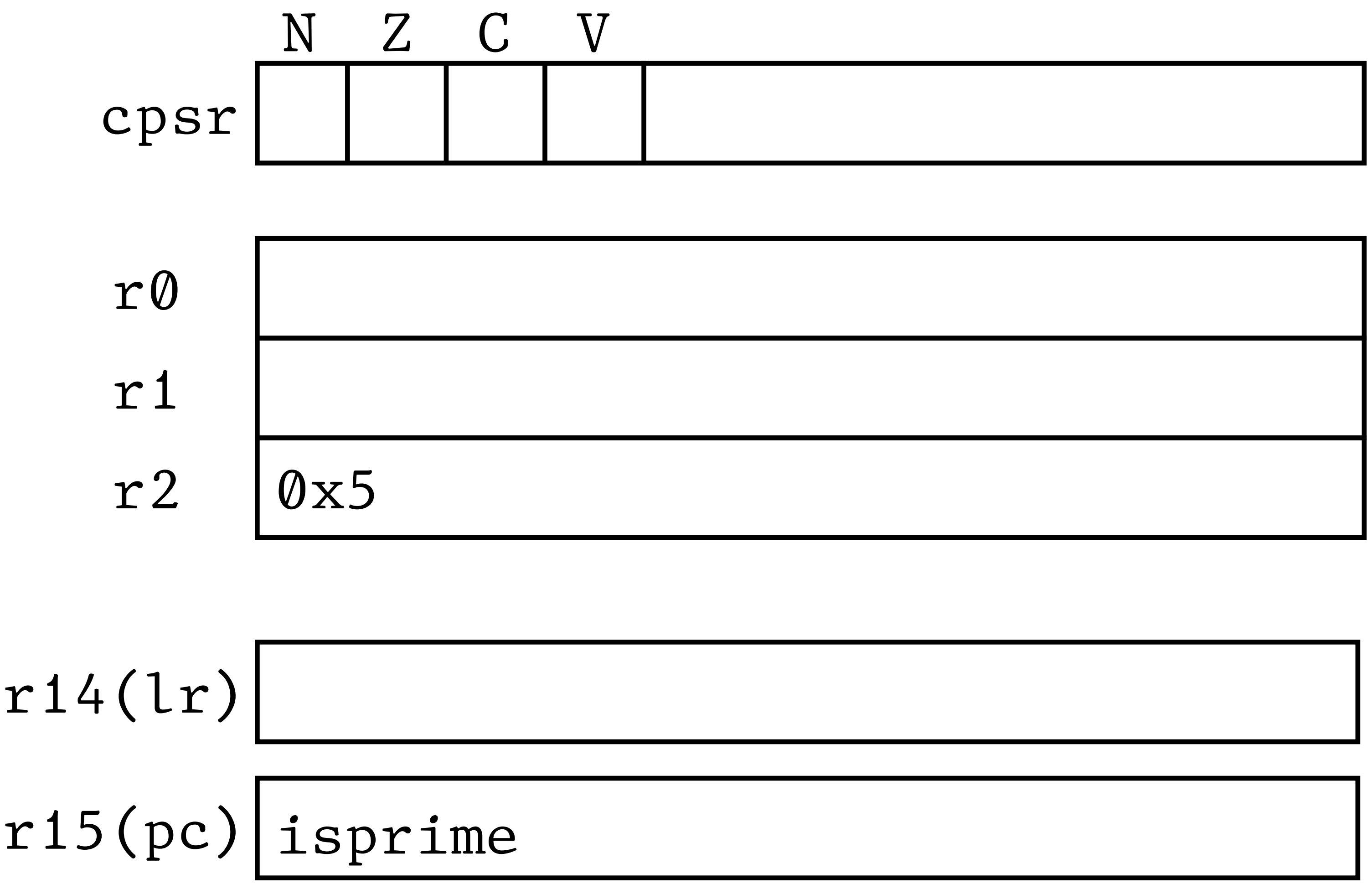
r0

r1

r2 0x5

r14(lr)

r15(pc) isprime



N Z C V

cpsr

r0

r1

r2 0x5

r14(lr)

r15(pc) isprime

```
mod:
loop:     subs r0, r1
          addne r0, r1
          bpl loop
          mov pc, lr

isprime:  mov r1, #1     @ The current factor
loop:     add r1, #1     @ Increment the current factor (start at 2)
          cmp r1, r2     @ If the current factor is bigger
          bge prime      @   than we've tried everything, it's prime
          mov r0, r2     @ Get our test value into r0, then
          bl mod         @   compute r0 % r1 using mod
          cmp r1, #0     @   if that value is 0
          beq notprime   @   then we know the number isn't prime
          b loop         @ Otherwise, keep incrementing
prime:    mov r0, #1     @ Store #1 in r0 for "yes it's prime" ~ true
          b end
notprime: mov r0, #0     @ Store #0 in r0 for "no it's not" ~ false
end:
```

What will be in r0 at end?

A: 0
B: 1
C: 2
D: 9
E: Something else

	N	Z	C	V	
cpsr					

r0	0x9
r1	
r2	

r15(pc)	isprime

```
mod:
loop:     subs r0, r1
          addne r0, r1
          bpl loop

isprime:  mov r1, #1       @ The current factor
loop:     add r1, #1       @ Increment the current factor (start at 2)
          cmp r1, r0       @ If the current factor is bigger
          bge prime        @   than we've tried everything, it's prime
          b mod            @ Compute r0 % r1 using mod
          cmp r1, #0       @   if that value is 0
          beq notprime     @   then we know the number isn't prime
          b loop           @ Otherwise, keep incrementing
prime:    mov r0, #1       @ Store #1 in r0 for "yes it's prime" ~ true
          b end
notprime: mov r0, #0       @ Store #0 in r0 for "no it's not" ~ false
end:
```