

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:

Assume we start with $r0 = 17$, $r1 = 3$

loop:

After this runs, we'll have:

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

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

Inf loop

cpsr

N	Z	C	V	

r0

<i>5 3 X -X X 5</i>

r1

<i>X Z X Z</i>

r2

0x5

mod:

loop0: subs r0, r1
addmi r0, r1
bpl loop0

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

notprime: mov r0, #0
end:

r15(pc)

~~isprime~~ *mod ... isprime*

@ The current factor

@ Increment the current factor (start at 2)

@ If the current factor is bigger

@ then we've tried everything, it's prime

@ Get our test value into r0, then

@ compute r0 % r1 using mod

@ if that value is 0

@ then we know the number isn't prime

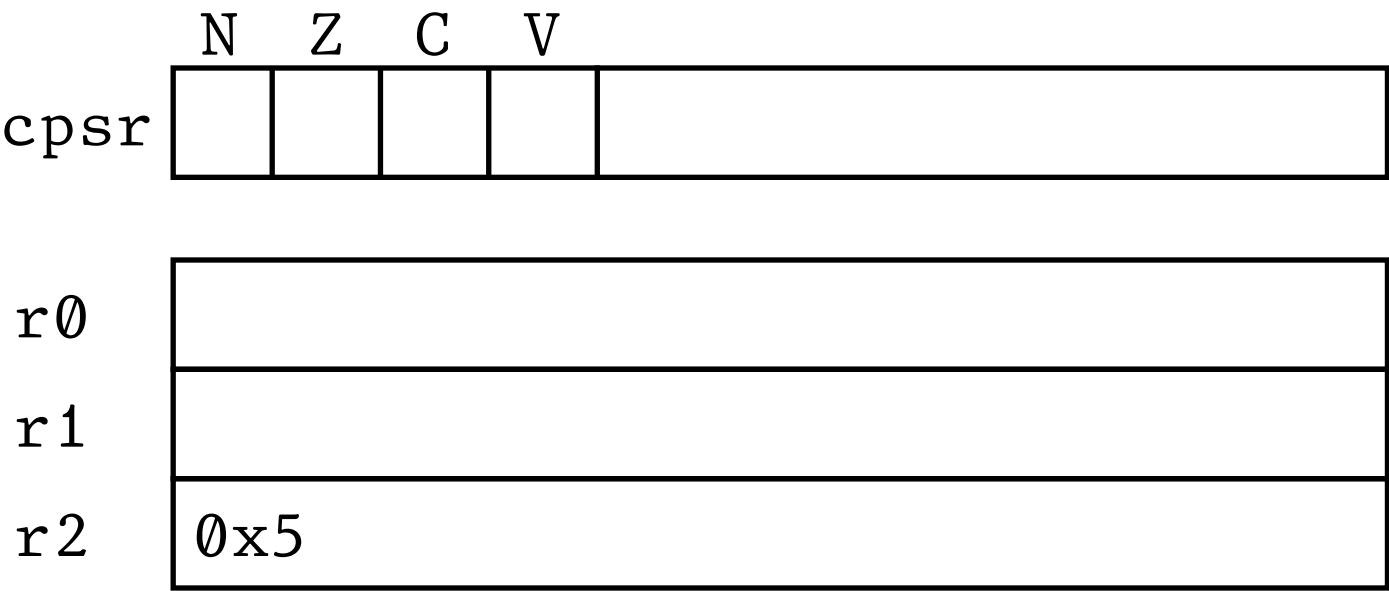
@ Otherwise, keep incrementing

@ Store #1 in r0 for "yes it's prime" ~ true

@ Store #0 in r0 for "no it's not" ~ false

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)
- C: prime
- D: Somewhere else



```
mod:
loop0:  subs r0, r1
        addmi r0, r1
        bpl loop0
```

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

use saved

save PC

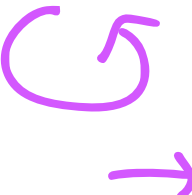


@ The current factor
@ Increment the current factor (start at 2)
@ If the current factor is bigger
@ then we've tried everything, it's prime
@ Get our test value into r0, then
@ compute r0 % r1 using mod
@ if that value is 0
@ then we know the number isn't prime
@ Otherwise, keep incrementing
@ Store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

```

mod:
loop0: subs r0, r1
      addmi r0, r1
      bpl loop0
      mov pc, lr

```



```

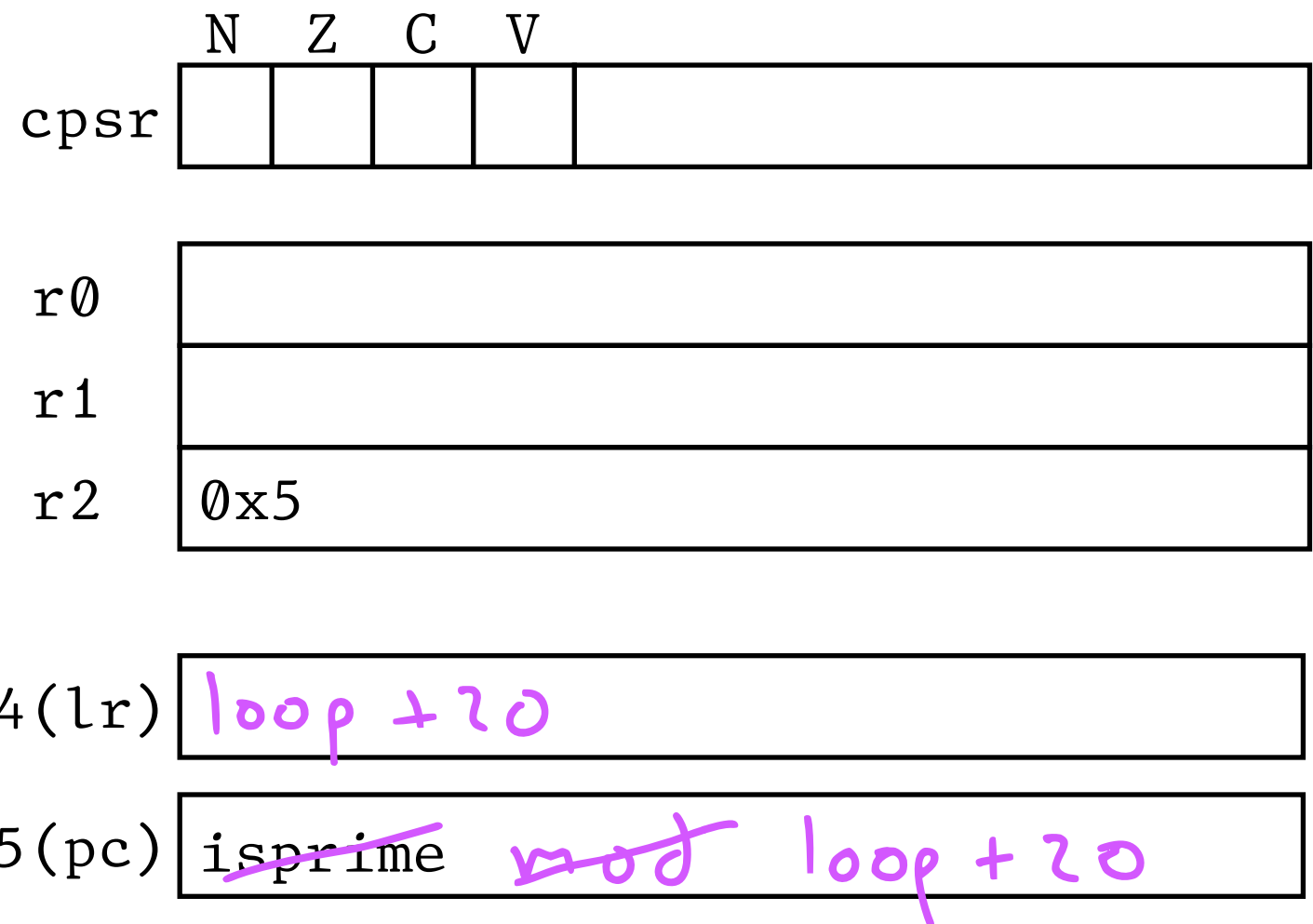
isprime: mov r1, #1
loop:    add r1, #1
        cmp r1, r2
        bge prime
        mov r0, r2
        bl mod
        → cmp r0, #0
        beq notprime
        b loop
prime:   mov r0, #1
        b end
notprime: mov r0, #0
end:

```

```

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

```



`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

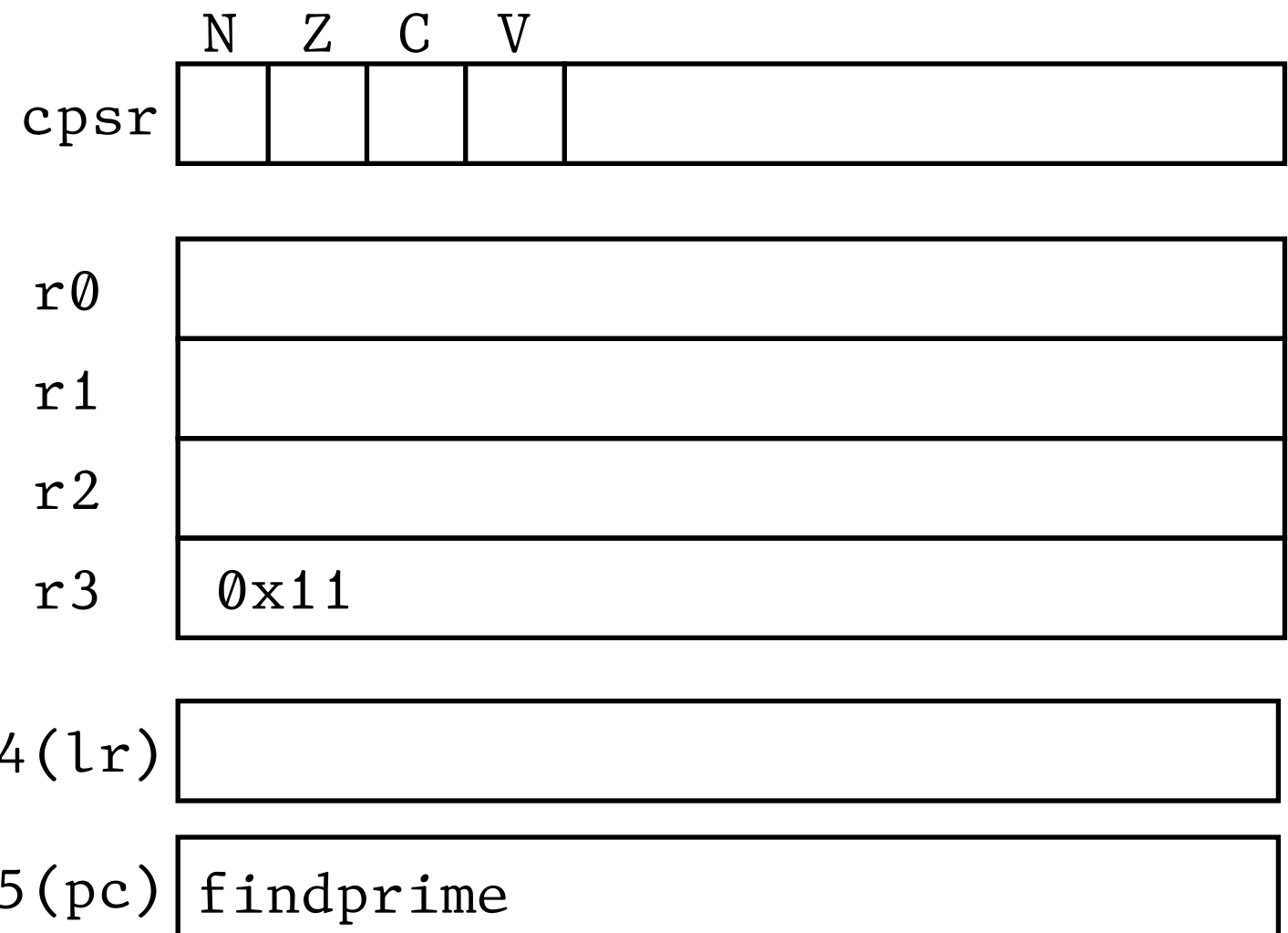
```

```

foundit:  mov r0, r3

```

@ (compute the first prime larger than the value in r3, store in r0)




```

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					
r1					
r2	11				
r3	0x11				
r14(lr)	loop2+8 loop+20				
r15(pc)	findprime isprime mod				

...
loop+20

```

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

```

leaf
function

```

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

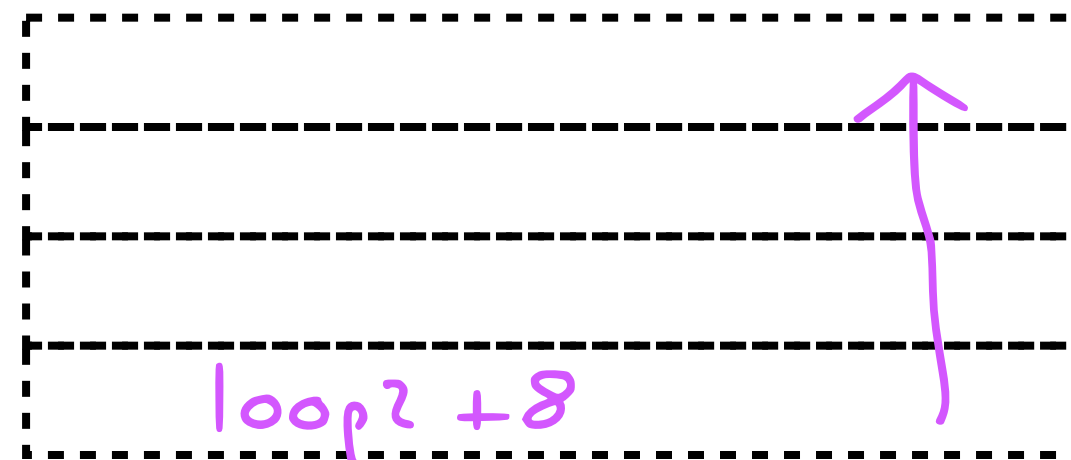
```

@ (compute the first prime larger than the value in r3, store in r0)

	N	Z	C	V
cpsr				
r0				
r1				
r2	11			
r3	0x11			
r13(sp)	0x108 0x104 0x108			
r14(lr)	loop2+8 loop+20			
r15(pc)	findprime isprime			

Memory

0x0fc
0x100
0x104
0x108




```

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

```

```

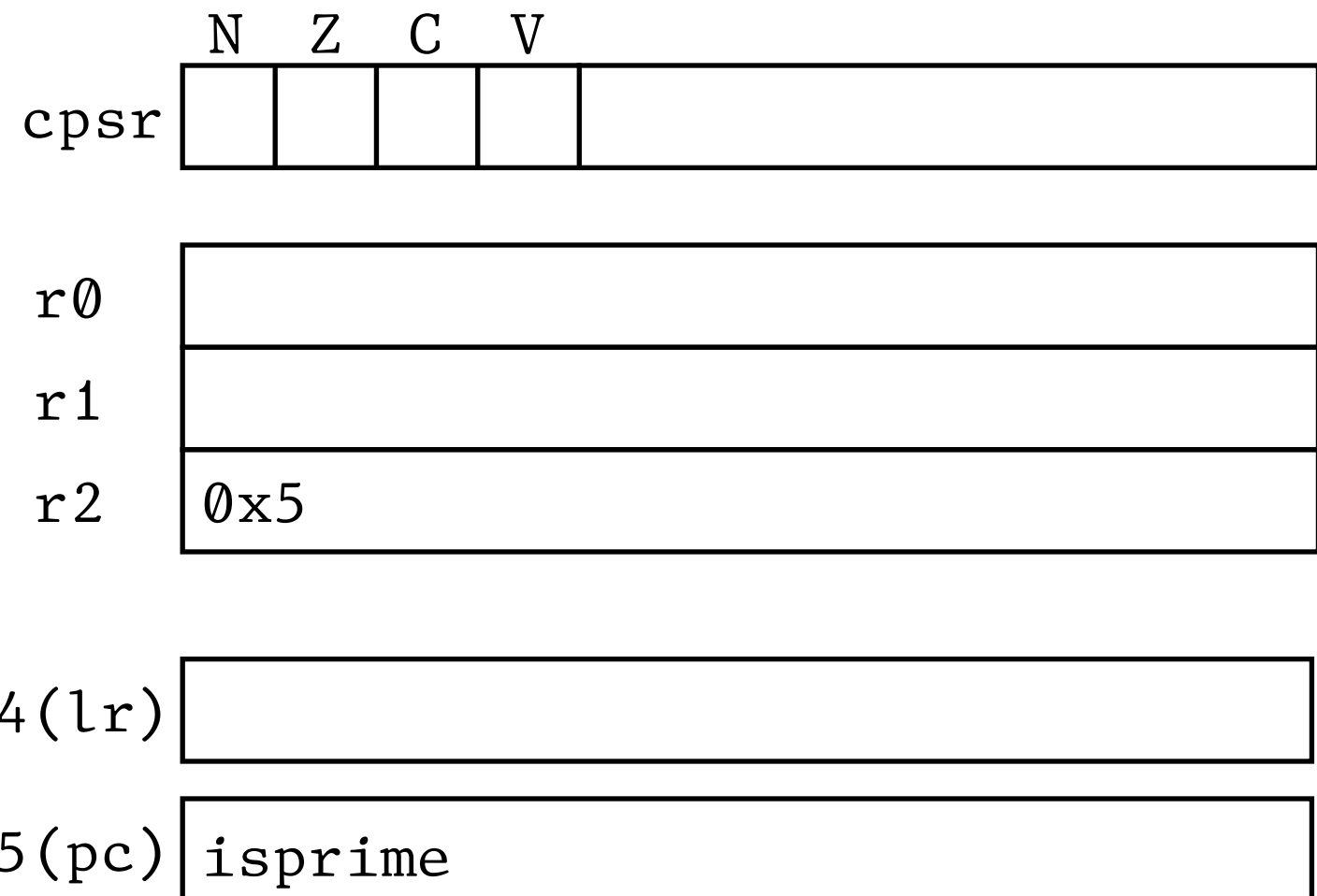
findprime: @ compute the first prime larger than the value in r3, store in r0
loop:      mov r2, r3
          bl isprime
          cmp r0, #1
          beq foundit
          addne r3, #1
          bne loop

```

```

foundit:  mov r0, r3

```



```

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

```

```

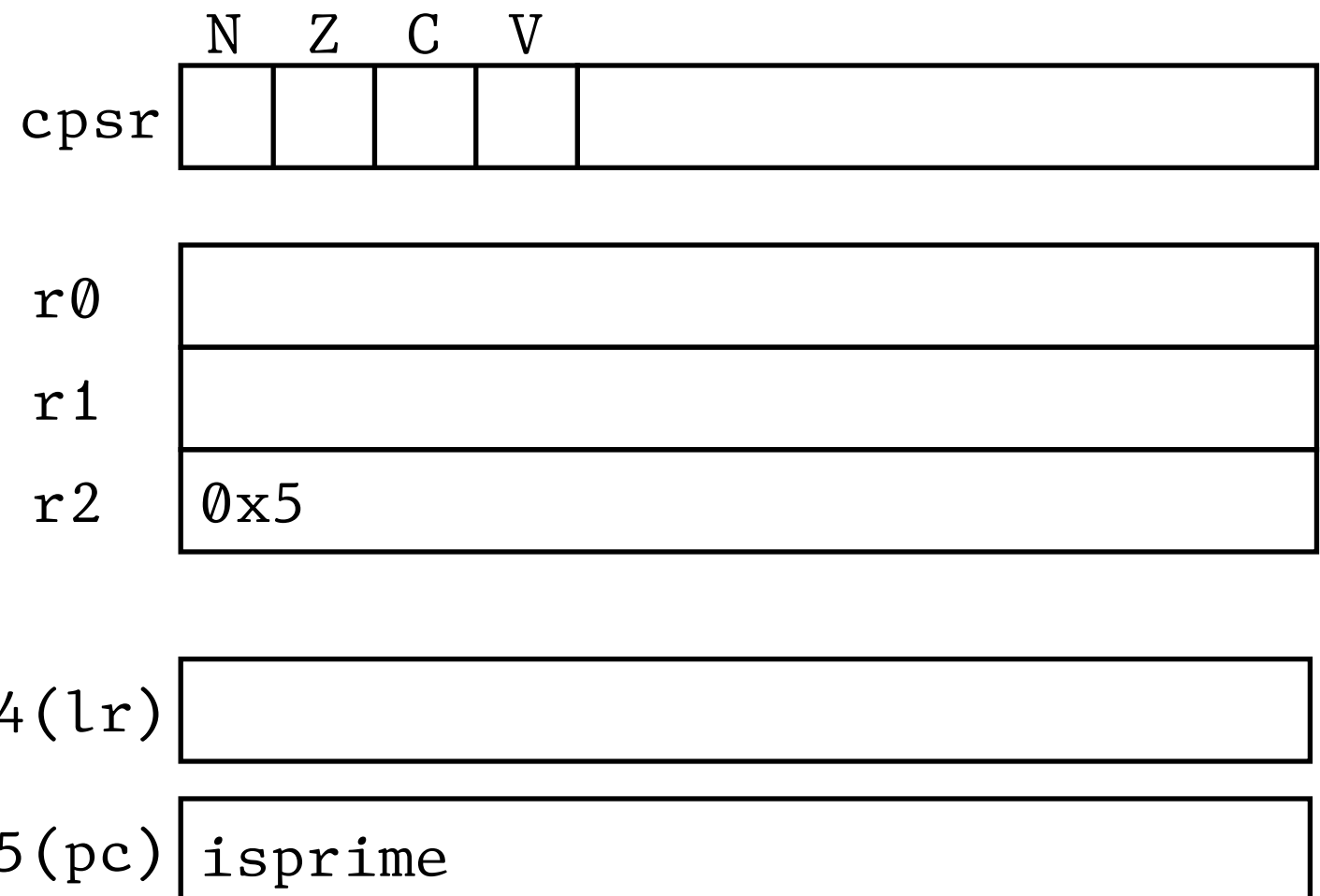
isprime: mov r1, #1
loop:    add r1, #1
      cmp r1, r2
      bge prime
      mov r0, r2
      bl mod
      cmp r1, #0
      beq notprime
      b loop
prime:   mov r0, #1
      b end
notprime: mov r0, #0
end:

```

```

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

```



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

mod:

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

r15(pc) isprime

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

```
notprime: mov r0, #0
end:
```

@ The current factor

@ Increment the current factor (start at 2)

@ If the current factor is bigger

@ than we've tried everything, it's prime

@ Compute r0 % r1 using mod

@ if that value is 0

@ then we know the number isn't prime

@ Otherwise, keep incrementing

@ Store #1 in r0 for "yes it's prime" ~ true

@ Store #0 in r0 for "no it's not" ~ false