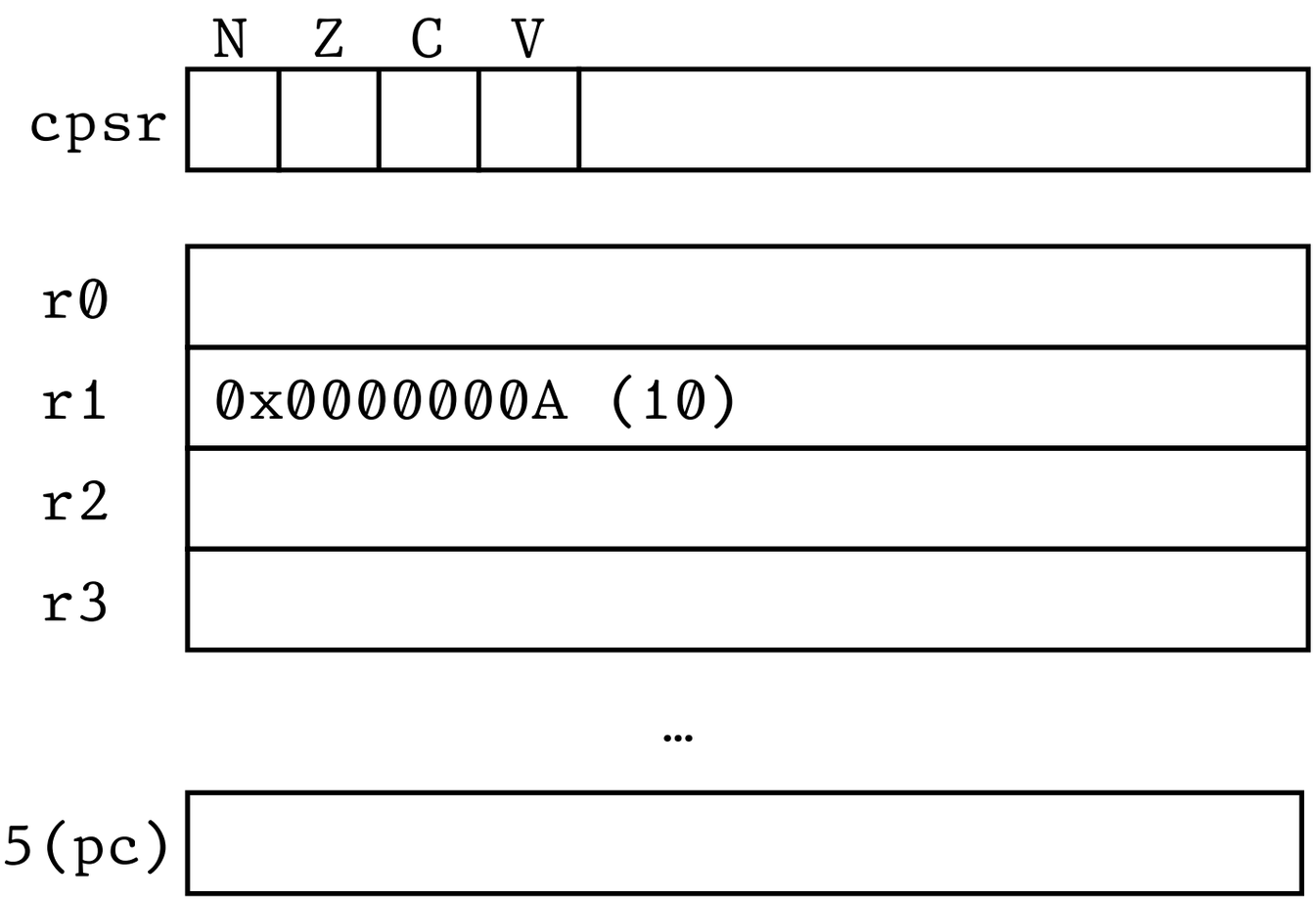
Registers



The **Program Counter** determines which instruction to run next. It's "just" another register.

```
0x100 mov r0, #0
0x104 add r0, r1, r0
0x108 subs r1, #1
0x10c
```

Each instruction is at an address

```
@go back and loop!
```

Compute the sum from 0 to the value stored in r1, put the answer in r0.

Registers

	N	Z	C	V	
cpsr					

r0	**0x00000003 (3)**
r1	**0x00000004 (4)**
r2	
r3	

…

r15(pc)	

```
0x100 mov r2, #1
0x104 mul r2, r2, r1
0x108 subs r1, #1
0x10c subne r15, #8
```

Each instruction is at an address

What is in r2 after the program runs?

A: 0x0000000c (12)
B: 0x00000051 (81)
C: 0x00000027 (27)
D: 0x00000003 (3)

Write a program that subtracts 7 from r1 until r1 is below zero, and stores the number of subtractions in r2.

Table 6.3 Condition mnemonics

cond	Mnemonic	Name	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\overline{Z}$
0010	CS/HS	Carry set / unsigned higher or same	C
0011	CC/LO	Carry clear / unsigned lower	$\overline{C}$
0100	MI	Minus / negative	N
0101	PL	Plus / positive or zero	$\overline{N}$
0110	VS	Overflow / overflow set	V
0111	VC	No overflow / overflow clear	$\overline{V}$
1000	HI	Unsigned higher	$\overline{Z}C$
1001	LS	Unsigned lower or same	Z OR $\overline{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\overline{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	Z OR $(N \oplus V)$
1110	AL (or none)	Always / unconditional	Ignored

```
0x100 mov r2, #1
0x104 mul r2, r2, r1
0x108 subs r1, #1
0x10c subne r15, #8
```

```
0x100 mov r2, #1
0x104 mul r2, r2, r1
0x108 add r2, #3        ←——— Add a statement
0x10c subs r0, #1
0x110 subne r15, #8     ←——— Now this is wrong!
```

Can we avoid explicit offsets? They seem annoying.

```
            mov r2, #1
begin_loop: mul r2, r2, r1
            add r2, #3
            subs r0, #1
            bne begin_loop
```

assembler →

```
0x100 mov r2, #1
0x104 mul r2, r2, r1
0x108 add r2, #3
0x10c subs r0, #1
0x110 bne 0x104
```

Label

`begin_loop:` →

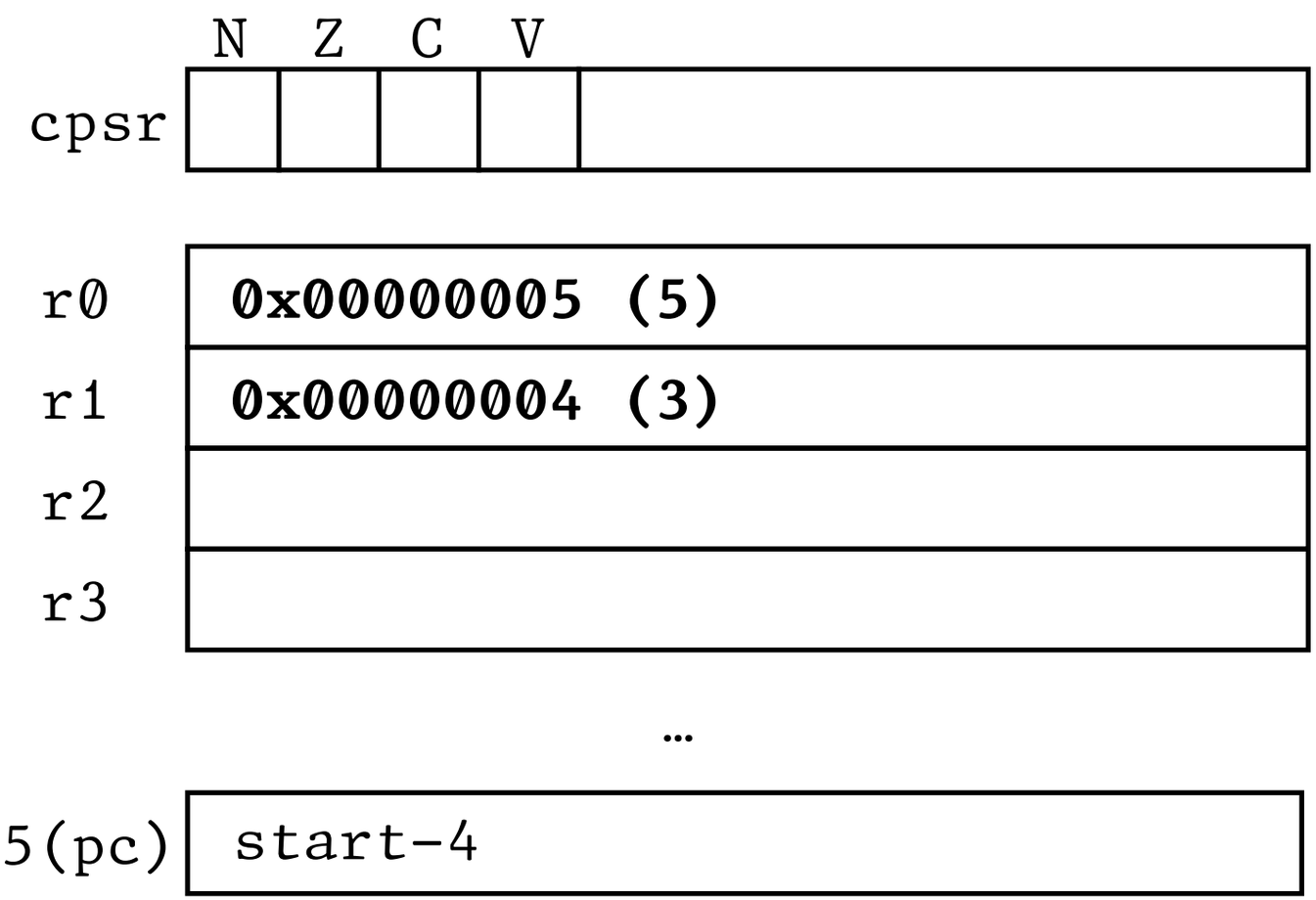
No output! Just used by other instructions.

Branch Instruction

`bne <label>`

Means "put the label's address in pc."
Assembler keeps track of label addresses for us.

Registers



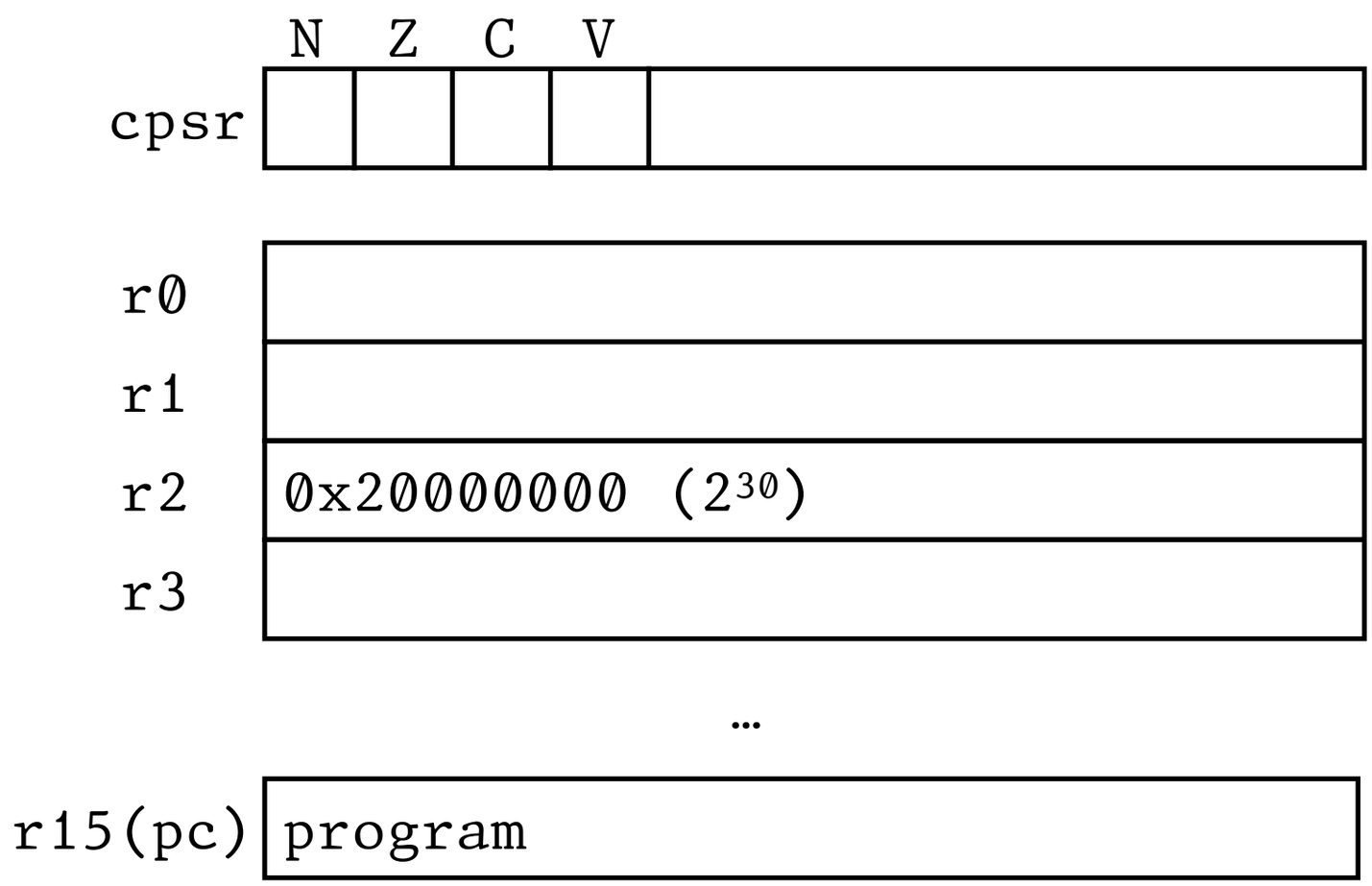

```
start-4   mov r2, #1
start:    mul r2, r2, r1
start+4   subs r1, #1
start+8   bne start
```

Each instruction is at an address. Useful to think of them as relative to labels. The gray/unbolded stuff is for us, not code.

What is in r2 after the program runs?

A: `0x00000019 (25)`
B: `0x0000000F (15)`
C: `0x0000007D (125)`
D: `0x0000001B (27)`
E: Something else

Registers




```
program: lsl r2, #1
```

Instruction Set

1. **lsl**: Perform a "logical shift left." That is, move all the bits left by the specified number of positions. Replace empty spots on the right with 0s.
2. **rsl**: Same idea, to the right.

What is in r2 after the program runs?

A: `0x10000000`
B: `0x40000000`
C: `0x40000001`
D: `0x20000001`
E: Something else

Registers

	N	Z	C	V	
cpsr					

r0	
r1	
r2	0x80000000
r3	
…	
r15(pc)	program

```
program: lsls r2, #1
```

Instruction Set

1. **lsl**: Perform a "logical shift left." That is, move all the bits left by the specified number of positions. Replace empty spots on the right with 0s.
2. **rsl**: Same idea, to the right.

What is in NZCV after the program runs?

A: N=0, Z=1, C=1, V=0
B: N=0, Z=1, C=0, V=1
C: N=0, Z=0, C=1, V=0
D: N=0, Z=1, C=0, V=0
E: Something else

Registers

	N	Z	C	V	
cpsr					

r0	
r1	
r2	0x80000000
r3	

...

r15(pc)	program

```
program:  asl r2, #3
```

Instruction Set

1. **asl**: Perform an "arithmetic shift right."

What is in r2 after the program runs?

A: 0xC0000000
B: 0x10000000
C: 0xF0000000
D: 0x40000000
E: Something else

Registers

	N	Z	C	V	
cpsr					

r0	
r1	
r2	0x10000000 (2^{29})
r3	

…

r15(pc)	start-4

Table 6.3 Condition mnemonics

cond	Mnemonic	Name	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\overline{Z}$
0010	CS/HS	Carry set / unsigned higher or same	C
0011	CC/LO	Carry clear / unsigned lower	$\overline{C}$
0100	MI	Minus / negative	N
0101	PL	Plus / positive or zero	$\overline{N}$
0110	VS	Overflow / overflow set	V
0111	VC	No overflow / overflow clear	$\overline{V}$
1000	HI	Unsigned higher	$\overline{Z}C$
1001	LS	Unsigned lower or same	Z OR $\overline{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\overline{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	Z OR $(N \oplus V)$
1110	AL (or none)	Always / unconditional	Ignored

```
start-4     mov r1, #0
start:      lsls r2, #1
start+4     bcs end
start+8     add r1, #1
start+12    b start
end:
```

"Count the number of leading 0s in r2"

What is in r1 after the program runs?

A: 0x00000002 (2)
B: 0x00000003 (3)
C: 0x0000000A (10)
D: 0x0000001D (29)