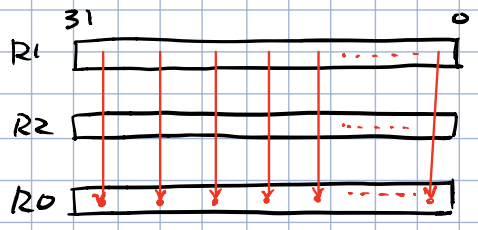
Logic Instructions (AND, ORR, EOR)

these are bit-wise operations.




```
AND   R0, R1, R2

mov   R1, #0x55555555
mov   R2, #0xAAAAAAAA
AND   R3, R1, R2    // R3 ← 00000000
ORR   R3, R1, R2    // R3 ← FFFFFFFF
AND   R3, R1, #1    // R3 ← 0000 0001  (used to isolate bit 0 of R1)
EOR   R2, R1, R2    // R2 ← FFFF FFFF
EOR   R1, R2, R1    // R1 ← AAAA AAAA.
EOR   R2, R2, R1    // R2 ← 5555 5555
```

} swap R1 ↔ R2

Shifting / rotate instructions.

LSR — logical shift right.

ASR — Arithmatic shift right.

```
LSR   R2, R3, #1
```

R2 ← contents of R3 shifted right by one position. A 0 is shifted into the MSB (equivalent to div by 2 for an unsigned number)

```
mov   R3, #24       R3 ← 0x00000018
LSR   R2, R3, #2    R2 ← 0x00000006   (24 ÷ 4 = 6)
ASR   R2, R3, #1    R2 ← 0x0000000C   (24 ÷ 2 = 12)
```

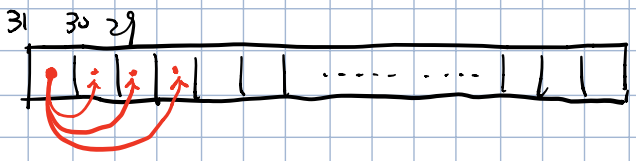
18 (hex)

00011000 LSR #2

00000110

ASR & LSR are the same for positive values. But ASR works for signed values.

ASR



- when we shift to the right, the msb (sign bit) is replicated. Thus, this is equivalent to division of signed number.

Note: the amount to shift can alternatively specified in a register

LSR R1, R2, R3 R1 ← R2 shifted right by the amount specified in (the 5 least significant bit of) R3

There's also a logical shift left. LSL instruction, which is equivalent to the multiplication by "powers of 2".

LSL R1, R2, #1 R1 ← R1 × 2

LSL R1, #3 R1 ← R1 × 8

There's a rotate right instruction. ROR that shifts right in a circular fashion.

Example.

ROR R1, R2, #8 R1 ← R2 ↻ 8 times.

ROR R1, #31 R1 ← R2 rotated left one position. (there is no ROL instruction)

.... 0001 000 0

.... 0010 0000 0

- Example Program (Lab 4 prep)

- Find the longest string of 1's in a word of data.

做3 4次, 所以input string of 1s = 4

```
MOV   R1, #TEST_NUM    R1 ← address of Test_NUM
LDR   R1, [R1]         R1 ← TEST_NUM
```