Bit-wise operations

Bit-wise operators

- Instructions that manipulate individual bits given to them in their operands
- Generally best understood by considering values in their binary form
- Many of these will feel familiar from the first half of this unit

AND – Logical bit-wise AND

- Syntax:

```
AND R3, R1, R2
```

- Performs a bit-wise logical AND on the two inputs given in the second and third operands
- Stores the result in the destination, (first operand)
- Eg,

If R1 = #0b0110

R2 = #0b1100

R1 “AND” R2”: R3 = #0b0100

ORR – Logical bit-wise OR

- Syntax:

```
ORR R3, R1, R2
```

- Performs a bit-wise logical OR on the two inputs given in the second and third operands
- Stores the result in the destination, (first operand)
- Eg,

If R1 = #0b0110

R2 = #0b1100

R1 “ORR” R2”: R3 = #0b1110

EOR – Logical bit-wise Exclusive OR

- Syntax:

```
EOR R3, R1, R2
```

- Performs a bit-wise logical XOR on the two inputs given in the second and third operands
- Stores the result in the destination, (first operand)
- Eg,

If R1 = #0b0110

R2 = #0b1100

R1 “EOR” R2”: R3 = #0b1010

Observations on AND,ORR,EOR

- Have clear analogies to the logical gates we have seen previously
- Generally useful for specific bit manipulations to control hardware, as well as bit masking (e.g, to extract part but not all of a 32 bit string).
- Also for performing comparisons between, or merging bit strings.

LSL – Logical Shift Left

- Syntax:

```
LSL R1, R1,#3
```

- Shifts each bit of the input value to the ***left***, by the number of places specified in the third operand, losing the leftmost bits, and adding zeros on the right.
- Stores the result in the destination, (first operand)

Eg,

If R1 = #0b01101110, and shift = #3

R1 becomes #0b01110000

LS – Logical Shift Left

- Syntax:

```
LSR R1, R1,#3
```

- Shifts each bit of the input value to the ***right***, by the number of places specified in the third operand, losing the rightmost bits, and adding zeros on the left.
- Stores the result in the destination, (first operand)

Eg,

If R1 = #0b01101110, and shift = #3

R1 becomes #0b00001101

LSL and LSR Observations

- Assuming Little Endian notation:
  - A single LSL doubles the value represented by the bit string
  - A single LSR halves the value represented by the bit string
- Both instructions invoke a shift register to perform task in hardware
  - As with all bit-wise operators, they can be performed in one CPU cycle.
- Form the backbone for implementing multiplication and division operations, which typically require multiple CPU cycles to calculate (we'll come back to this later)