Logical Operations

- Logical operations perform operations on the **bits** themselves rather than the values the bits represent:

AND OR EOR NOT

Logical AND

Truth Table

0	0	0
0	1	0
1	0	0
1	1	1

Logical AND Examples

```
0110 1001
1010 0111 &
0010 0001
```

```
and.b     d6,d7
and.l     $200A,d3
and.w     #$1000,d4
```


Inclusive Logical (OR)

Truth Table

0	0	0
0	1	1
1	0	1
1	1	1

Inclusive Logical (OR) examples

```
0110 1001
1010 0111 |
1110 1111
```

```
or.b    d2,d7
or.l    $2010,d3
or.w    #$0456,d4
```


Exclusive Logical OR (EOR)

Truth Table

0	0	0
0	1	1
1	0	1
1	1	0

Exclusive Logical OR (EOR) Examples

```
0110 1001
1010 0111 ⊕
1100 1110
```

```
eor.w   d2,d7
eor.w   d3,$2018
```

Note: The **immediate** addressing mode is not permitted.

Logical Complement NOT

- This is the equivalent of an invert operation (also known as 1's complement).
- Note there is only a single operant
  - -> **unary** operation

```
0110 1001 -> 1001 0110
```

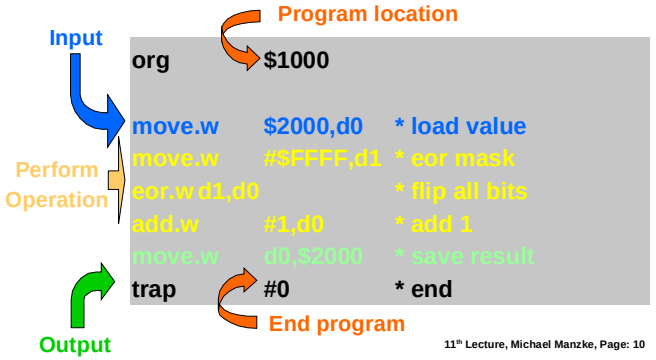
```
not.b   d5
not.w   $2010
```

Note: There is no **immediate** mode for the **not** operation.

EOR Application

- Write a program to determine the 2's complement binary representation of a word located at $2000.
- Method:
  - Invert the bits and add 1.

2's Complement Program



Set the 3rd bit of the d3 register

Use the or operation:

```
1001 0011   <- Previous d3 contents
0000 0100 | <- mask
1001 0111
```

```
or.b #$04,d3
or.b #4,d3
or.b #%00000100,d3
or.b #%100,d3
```

Equivalent Instructions

Clear the 2nd and the 7th bits of d3

Use the and operation:

```
1110 1110   <- Previous d3 contents
1011 1101 & <- mask
1010 1100
```

```
and.b #$bd,d3
and.b #%10111101,d3
```

Equivalent Instructions

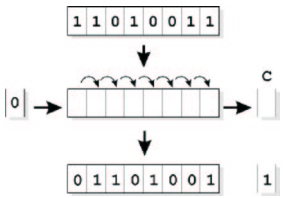
Setting & Clearing Bits

- Use **OR** for setting bits
- Use **AND** for clearing bits

Bit Manipulation Instructions

- Shift:
  - Two types:
    - **Logical** (0 shifted into free space)
      - lsr, lsl
    - **Arithmetic** (sign bit is handled correctly)
      - asr, asl
- Rotate:
  - ror, rol

Logical Shift Right (lsr)



The bits of the destination are shifted a number of places to the right.

Last bit to be shifted out of the destination is stored in **C** (and in **X**).

Examples (lsr)

```
0110 1001 -> shift right by 1 -> 0011 0100
                                 C = 1
```

```
move.b   #4, d0
lsr.b    d0, d2
lsr.w    d1
```

Valid instruction but is not recognized by the Robot assembler.

Use lsr.w #1,d1 instead

Logical Shift Left(lsl)

```
0110 1001 ->
   shift left by 1
        -> 1101 0010
            C = 0
```


Arithmetric Shift Opeartions

- asl:
  - Exactly the same as lsl
- asr:
  - Same as lsr exept the MSB is retained and copied into the empty bit positions.

```
1010 1001 -> asr by 3 -> 1111 0101
```

In this way the sign of the number is preserved.