**Theori 2 : Gweithrediadau Rhesymegol**

|  |  |
| --- | --- |
|  | Lluniadu gwirlenni ar gyfer mynegiadau Boole gan gynnwys  gweithrediadau rhesymegol AC, NEU, NID a NEUA.  Cymhwyso gweithrediadau rhesymegol at gyfuniadau o  amodau mewn rhaglennu, gan gynnwys clirio cofrestri a masgio.    Symleiddio mynegiadau Boole gan ddefnyddio hunaniaethau a rheolau Boole. |

Mae cyfrifiaduron modern yn defnyddio gwerthoedd deuaidd. Mae’r gwerthoedd yma yn cynrychioli 1 neu 0 (*true or false*). Trwy ddefnyddio **gweithrediadau rhesymegol** (*logic gates*) gallwn gymryd mewnbwn ac yna creu allbwn am bob gwerth mewnbwn posib.

**Y prif gweithrediadau rhesymegol yw :**

* AC (*AND*)
* NEU (*OR*)
* NID (*NOT*)
* NEUA (NEU Anghynhwysol) (*XOR – Exclusive OR*)
* NIEU (NOR)
* NIAC (NAND)

Gellir defnyddio **gwirlenni** (*truth tables*) ar gyfer mynegiadau **Boole**. A a B yw’r mewnbwn ac R yw’r allbwn.

### AC (AND Gate)

Dyma’r tabl a diagram i ddangos AC (*AND*). Weithiau defnyddir • neu ‘.’

|  |  |  |
| --- | --- | --- |
| AC (AND) • | | |
| A | B | R |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Tebyg iawn i lluosi

### 

### NEU (OR Gate)

Dyma’r tabl am NEU (*OR*). Defnyddir + i gynrychioli hwn.

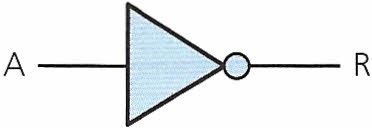
|  |  |  |
| --- | --- | --- |
| NEU (OR) + | | |
| A | B | R |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Tebyg iawn i adio

**NID (NOT Gate)**

Dyma’r tabl ar gyfer NID. Weithiau defnyddir (llinell uwchben y mewnbwn).

|  |  |
| --- | --- |
| NID (NOT) | |
| A | R |
| 0 | 1 |
| 1 | 0 |



**Cofiwch:**

Wrth ysgrifennu allan gweithrediadau Boole, defnyddir symbolau i gynrychioli

AC (•), NEU(+) and NID (­ ¯ ).

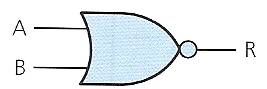
### Cylchedau (*Circuits)*

Gallwch cyfuno y gweithrediadau rhesymegau AC (AND) a NEU (OR) gyda’r gweithred NID (NOT) er mwyn creu’r gweithrediadau NIAC (NAND) a NEUA (XOR)

**NIAC(NAND)**

|  |  |  |
| --- | --- | --- |
| NIAC (NAND) | | |
| A | B | R |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**NIEU (NOR)**

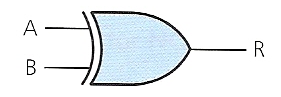
****

|  |  |  |
| --- | --- | --- |
| NIEU (NOR) | | |
| A | B | R |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Mae’r rhesymeg gweithredol NEU (OR) yn meddwl ‘un neu’r ddau’. Wrth I ni siarad ar lafar defnyddir NEU (OR)T i feddwl un neu’r llall ond nid y ddau. Yn rhesymegol, gelwir hwn yn NEU Anghynhwysol (Exclusive Or). Defnyddir NEUA (XOR) I ddangos hwn.

**XOR**

|  |  |  |
| --- | --- | --- |
| XOR | | |
| A | B | R |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



**Esiampl**

Er enghraifft, R = • B yn cyfartal I ganlyniad NID A AC B. Gallwch gweithio allan yr holl canlyniadau posib trwy greu gwirlenni (truth table).

|  |  |  |  |
| --- | --- | --- | --- |
|  | R = • B | |  |
| A | B |  | R |
| 0 | 0 | 1 | 0  B |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |

Er mwyn cyfrifo’r uchod byddwch yn gwneud y NID A cyntaf ac yna A AC B I gyrraedd R.

Gallwch ddefnyddio mwy na 2 mewnbwn mewn gwirlenni (truth tables). Mae’r nifer o ganlyniadau yn dwbli wrth I ni ychwanegu mewnbwn. Ar gyfer 3 mewnbwn mae yna 8 canlyniad posib.

**Esiampl**

For example, the expression R = (A • B) +

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | R = (A • B) + | |  |  |
| A | B | C | (A • B) |  | R |
| 0 | 0 | 0 | 0 | 1 | 1  C |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |

Wrth I chi gwneud gwaith boole mae rhaid dilyn rheolau. Rydych yn dilyn y strwythur NID, AC yna NEU, felly nid yw’r ymadrodd yn y cromfachau yn gwneud unrhyw gwahaniaeth I’r canlyniad.

Felly, y trefn cyfrifo uchod yw I wneud y NID yn gyntaf, yna’r AC ac yna’r NEU I gyrraedd y canlyniad.

Fel gyda phob algebra, cewch nifer o rheolau i gyfrifo’r ymadroddion. Gwnewch bob ymdrech I ddysgu a chofio nhw. Bydd angen nhw yn yr arholiad.

### Commutative Laws of Boolean Algebra

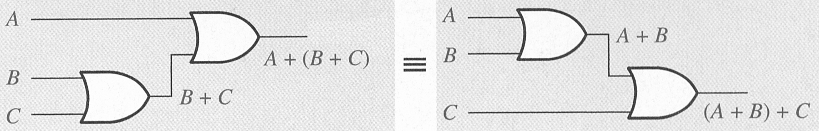


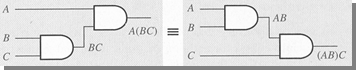
1. A + B = B + A



1. A • B = B • A

### Associative Laws of Algebra



* 1. A + (B + C) = (A + B) + C = A+B+C
  2. A • (B • C) = (A • B) • C = ABC

### Distributive Laws of Boolean Algebra

1. A • (B + C) = (A • B) + (A • C)
2. A(B + C) = AB + AC

### General Rules of Boolean Algebra \* LEARN / MEMORISE

|  |  |  |
| --- | --- | --- |
| AND | OR | MISC |
| 1. A • 0 = 0 2. A • 1 = A 3. A • A = A 4. A • = 0 | 1. A + 0 = A 2. A + 1 = 1 3. A + A = A 4. A + = 1 | 1. = A 2. A + AB = A 3. A + B = A + B 4. (A + B)(A + C) = A + BC |

**Remember these rules are pattern recognition. In other words the letters mean nothing i.e**. D • 0 = 0 is the same as A • 0 = 0 etc.

**Prove / simplify the following Boolean Expressions**

1. A = A + A . B
2. A . B + A . (B + C) + B . (B + C)
3. (A + B) . (A + )
4. A . (C + B . D) + . ) . C

**THERE ARE LOADS OF VIDEOS ON YOUTUBE THAT EXPLAIN BOOLEAN ALGEBRA SIMPLIFICATION REALLY WELL. I RECOMMEND THAT YOU WATCH SOME OF THESE VIDEOS!!!!**

**Use of Logical Operations**

### Systemau Rheoli

**Logical functions** can be used in a program for a **control system**.

**Example**: Suppose a control system consists of **8 switches**. These may be either **OFF** (0) or **ON** (1).

It is possible to turn switches on by using the **OR** instruction.

Eg If switches 1, 6 and 8 are already ON and we want to turn on switches 3 and 4...

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Switch | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Current position (1,6 and 8 are ON) | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| Use the instruction OR  0011000 | | | | | | | | |
| Instruction is OR | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| and the resulting output is | | | | | | | | |
| Switches 3 and 4 are switched on | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

The switches can be switched OFF by using the **XOR** instruction...

|  |  |
| --- | --- |
| Following on from the end of the above example, what would be the effect of the following instruction:  XOR 10011000? | Switches 1 and 4 would be switched OFF and switch 5 would be switched ON. |

### Masking

Another use for logical functions is in masking out bits of a number. For example, a **character** may be input from the keyboard and stored as its ASCII code.

The digit **5** for example would be stored as the ASCII pattern **00110101**

To convert it into a binary **number** we need to '**mask out**' the first 4 bits of the number. To do this we use the **AND** function...as shown in this table...

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The character '5' is input | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| Use the mask 00001111... | | | | | | | | |
| AND 00001111 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Resulting in... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| ..and this is the binary equivalent of the number 5. | | | | | | | | |

Masking is an important concept. The bits in the mask are chosen to manipulate the bits in the operand, allowing them through or blocking them.

**AND** can be used to return bits by using a 1, or exclude bits by using a 0. This is useful for checking conditions stored in a binary value.

**OR** can be used to reset particular bits in the binary value; using a 1 will always set the bit to 1, and using a 0 will return the matching bit in the original value.

**XOR** can be used to check if corresponding bits in two binary values are the same.

**Example**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Operand | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| Mask | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| AND | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Operand | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| Mask | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| AND | 1 | 1 | 1 | 0 | 1 | 1 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Operand | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| Mask | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| XOR | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

### Encryption

Take any number **A**, **XOR** it with another number **B** resulting in **C**.

Now take **C**, **XOR** it again with **B**, and the result will be **A**.

This fact is used in encryption, where A is the original data, B is the encryption key and C is the coded data.

Demonstration using the encryption key 10101010:

|  |  |
| --- | --- |
| A (the original data) | 10001101 |
| XOR B (the encryption key) | 10101010 |
| = C (coded data) | 00100111 |
| Data can now be transmitted to another computer before being decrypted | |
| XOR B(encryption key) | 10101010 |
| = A (original data) | 10001101 |