**1.7 Exercises**

* 1. Explain how a digital computer based on computational model of a black box works.
  2. Explain how bio-molecular computer based on computational model of another black box works.
  3. What are respectively a digital program and a bio-molecular program between a digital computer and bio-molecular computer?
  4. What are subsequently a memory subsystem and a tube between a digital computer and bio-molecular computer?
  5. What are respectively an input/output subsystem and a tube between a digital computer and bio-molecular computer?

1.6 What are bit 0 and bit 1 to a digital computer and bio-molecular computer?

* 1. Explain how a digital computer based on the von Neumann architecture works.
  2. Explain how bio-molecular computer based on the von Neumann architecture works.

**2.7 Exercises**

2.1 For a digital computer and bio-molecular computer, a bit is the smallest unit of data in which its vale either 1 or 0. Answer the following questions about how to encoding a bit:

a. How are to a bit its values 0 and 1 encoded in a digital computer?

b. How are to a bit its values 0 and 1 encoded in bio-molecular computer?

2.2 A bit pattern is a uniform representation of information. Answer the following questions about how to encoding a bit pattern:

a. How is a bit pattern encoded in a digital computer?

b. How is a bit pattern encoded in bio-molecular computer?

2.3 Write a program of a digital computer to convert a *hexadecimal* number to its corresponding *binary* number.

2.4 Write a program of a digital computer to convert a *binary* number to its corresponding *hexadecimal* number.

2.5 Write a program of a digital computer to convert an *octal* number to its corresponding *binary* number.

2.6 Write a program of a digital computer to convert a *binary* number to its corresponding *octal* number.

**3.12 Exercises**

3.1 The truth table of a logical operation **NOT** with one input and one output is shown in Table 3.12.1. Based on Table 3.12.1, write a bio-molecular program to implement the function of the logical operation **NOT**.

|  |  |
| --- | --- |
| The first input | The first output |
| 0 | 1 |
| 1 | 0 |

Table 3.12.1: The truth table of a logical operation **NOT** is shown.

3.2 The truth table of a logical operation **AND** with two inputs and one output is shown in Table 3.12.2. Based on Table 3.12.2, write a bio-molecular program to implement the function of the logical operation **AND**.

|  |  |  |
| --- | --- | --- |
| The first input | The second input | The first output |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Table 3.12.2: The truth table of a logical operation **AND** is shown.

3.3 The truth table of a logical operation **OR** with two inputs and one output is shown in Table 3.12.3. Based on Table 3.12.3, write a bio-molecular program to implement the function of the logical operation **OR**.

|  |  |  |
| --- | --- | --- |
| The first input | The second input | The first output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Table 3.12.3: The truth table of a logical operation **OR** is shown.

3.4 The truth table of a logical operation **BUFFER** with one input and one output is shown in Table 3.12.4. Based on Table 3.12.4, write a bio-molecular program to implement the function of the logical operation **BUFFER**.

|  |  |
| --- | --- |
| The first input | The first output |
| 0 | 0 |
| 1 | 1 |

Table 3.12.4: The truth table of a logical operation **BUFFER** is shown.

3.5 The truth table of a logical operation **NAND** with two inputs and one output is shown in Table 3.12.5. Based on Table 3.12.5, write a bio-molecular program to implement the function of the logical operation **NAND**.

|  |  |  |
| --- | --- | --- |
| The first input | The second input | The first output |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Table 3.12.5: The truth table of a logical operation **NAND** is shown.

3.6 The truth table of a logical operation **NOR** with two inputs and one output is shown in Table 3.12.6. Based on Table 3.12.6, write a bio-molecular program to implement the function of the logical operation **NOR**.

|  |  |  |
| --- | --- | --- |
| The first input | The second input | The first output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Table 3.12.6: The truth table of a logical operation **NOR** is shown.

3.7 The truth table of a logical operation **Exclusive-OR** with two inputs and one output is shown in Table 3.12.7. Based on Table 3.12.7, write a bio-molecular program to implement the function of the logical operation **Exclusive-OR**.

|  |  |  |
| --- | --- | --- |
| The first input | The second input | The first output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Table 3.12.7: The truth table of a logical operation **Exclusive-OR** is shown.

3.8 The truth table of a logical operation **Exclusive-NOR** with two inputs and one output is shown in Table 3.12.8. Based on Table 3.12.8, write a bio-molecular program to implement the function of the logical operation **Exclusive-NOR**.

|  |  |  |
| --- | --- | --- |
| The first input | The second input | The first output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Table 3.12.8: The truth table of a logical operation **Exclusive-NOR** is shown.

3.9 The truth table of a logical operation **NULL** with two inputs and one output is shown in Table 3.12.9. Based on Table 3.12.9, write a bio-molecular program to implement the function of the logical operation **NULL**.

|  |  |  |
| --- | --- | --- |
| The first input | The second input | The first output |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Table 3.12.9: The truth table of a logical operation **NULL** is shown.

3.10 The truth table of a logical operation **IDENTITY** with two inputs and one output is shown in Table 3.12.10. Based on Table 3.12.10, write a bio-molecular program to implement the function of the logical operation **IDENTITY**.

|  |  |  |
| --- | --- | --- |
| The first input | The second input | The first output |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Table 3.12.10: The truth table of a logical operation **IDENTITY** is shown.

**4.7 Exercises**

4.1 Concisely explain what a decimal system and a binary system are.

4.2 Convert two decimal numbers 128 and 33 to their corresponding binary numbers with *eight* bits.

4.3 Convert two binary numbers 10000000 and 00100001 to their corresponding decimal numbers.

4.4 Write a digital program to convert a decimal number to a binary number.

4.5 Write a digital program to convert a binary number to a decimal number.

4.6 It is assumed that an *n*-bit binary number, *xn* … *x*1 is used to represent an unsigned integer of *n* bits, where the value of each bit *xk* is either 1 or 0 for 1 ≤ *k* ≤ *n*. The bits *xn* and *x*1 is applied to represent, respectively, the most significant bit and the least significant bit for an unsigned integer of *n* bits. An unsigned integer of *n* bits ranges between 0 and 2*n* − 1. Write a bio-molecular program to construct the range of the value for an unsigned integer of *thre*e bits.

4.7 It is assumed that an *n*-bit binary number, *xn* … *x*1 is used to represent a sign-and-magnitude integer of *n* bits, where the value of each bit *xk* is either 1 or 0 for 1 ≤ *k* ≤ *n*. The bit *xn* is used to represent the sign, and the bits *xn* − 1 and *x*1 is applied to represent, respectively, the most significant bit and the least significant bit for a sign-and-magnitude integer of *n* bits. A sign-and-magnitude integer of *n* bits ranges between −(2*n* − 1 − 1) and +(2*n* − 1 − 1). Write a bio-molecular program to construct the range of the value for a sign-and-magnitude integer of *three* bits.

4.8 It is supposed that an *n*-bit binary number, *xn* … *x*1 is used to represent a one’s complement integer of *n* bits, where the value of each bit *xk* is either 1 or 0 for 1 ≤ *k* ≤ *n*. A one’s complement integer of *n* bits ranges between −(2*n* − 1 − 1) and +(2*n* − 1 − 1). Write a bio-molecular program to construct the range of the value for a one’s complement integer of *three* bits.

4.9 It is assumed that an *n*-bit binary number, *xn* … *x*1 is used to represent a two’s complement integer of *n* bits, where the value of each bit *xk* is either 1 or 0 for 1 ≤ *k* ≤ *n*. A two’s complement integer of *n* bits ranges between −(2*n* − 1) and +(2*n* − 1 − 1). Write a bio-molecular program to construct the range of the value for a two’s complement integer of *thre*e bits.

4.10 It is supposed that a 32-bit binary number, *x*32 … *x*1 is used to represent a floating-point number of32 bits in form of single precision format based on Excess\_127, where the value of each bit *xk* is either 1 or 0 for 1 ≤ *k* ≤ 32. A floating-point number of32 bits in form of single precision format based on Excess\_127 ranges between −(2128 × 1.11111111111111111111111) and +(2128 × 1.11111111111111111111111). Write a bio-molecular program to construct the range of the value for a floating-point number of 32 bits in form of single precision format based on Excess\_127.

4.11 It is assumed that a 64-bit binary number, *x*64 … *x*1 is applied to represent a floating-point number of64 bits in form of double precision format based on Excess\_1023, where the value of each bit *xk* is either 1 or 0 for 1 ≤ *k* ≤ 64. A floating-point number of64 bits in form of double precision format based on Excess\_1023 ranges between −(21024 × 1.1111111111111111111111111111111111111111111111111111) and +(21024 × 1.1111111111111111111111111111111111111111111111111111). Write a bio-molecular program to construct the range of the value for a floating-point number of 64 bits in form of double precision format based on Excess\_1023.

**5.6 Exercises**

5.1 The binary operator ∨ defines logical operation **OR**. The truth table to a logical operation, *x* ∨ 0, is shown in Table 5.6.1, where *x* is a Boolean variable that is the first input and 0 is the second input. Based on Table 5.6.1, write a bio-molecular program to implement the function of the logical operation, *x* ∨ 0.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input | *x* ∨ 0 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |

Table 5.6.1: The truth table to a logical operation, *x* ∨ 0, is shown.

5.2 The binary operator ∧ defines logical operation **AND**. The truth table to a logical operation, *x* ∧ 1, is shown in Table 5.6.2, where *x* is a Boolean variable that is the first input and 1 is the second input. Based on Table 5.6.2, write a bio-molecular program to implement the function of the logical operation, *x* ∧ 1.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input | *x* ∧ 1 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

Table 5.6.2: The truth table to a logical operation, *x* ∧ 1, is shown.

5.3 The binary operator ∨ defines logical operation **OR**, and the unary operator ′ defines logical operation **NOT**. The truth table to a logical operation, *x* ∨ *x*′, is shown in Table 5.6.3, where *x* is a Boolean variable that is the first input and *x*′ is its negation that is the second input. Based on Table 5.6.3, write a bio-molecular program to implement the function of the logical operation, *x* ∨ *x*′.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*x*′) | *x* ∨ *x*′ |
| 0 | 1 | 1 |
| 1 | 0 | 1 |

Table 5.6.3: The truth table to a logical operation, *x* ∨ *x*′, is shown.

5.4 The binary operator ∧ defines logical operation **AND**, and the unary operator ′ defines logical operation **NOT**. The truth table to a logical operation, *x* ∧ *x*′, is shown in Table 5.6.4, where *x* is a Boolean variable that is the first input and *x*′ is its negation that is the second input. Based on Table 5.6.4, write a bio-molecular program to implement the function of the logical operation, *x* ∧ *x*′.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*x*′) | *x* ∧ *x*′ |
| 0 | 1 | 0 |
| 1 | 0 | 0 |

Table 5.6.4: The truth table to a logical operation, *x* ∧ *x*′, is shown.

5.5 The binary operator ∨ defines logical operation **OR**. The truth table to a logical operation, *x* ∨ *x*, is shown in Table 5.6.5, where *x* is a Boolean variable that is the first input and the second input. Based on Table 5.6.5, write a bio-molecular program to implement the function of the logical operation, *x* ∨ *x*.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*x*) | *x* ∨ *x* |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

Table 5.6.5: The truth table to a logical operation, *x* ∨ *x*, is shown.

5.6 The binary operator ∧ defines logical operation **AND**. The truth table to a logical operation, *x* ∧ *x*, is shown in Table 5.6.6, where *x* is a Boolean variable that is the first input and the second input. Based on Table 5.6.6, write a bio-molecular program to implement the function of the logical operation, *x* ∧ *x*.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*x*) | *x* ∧ *x* |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

Table 5.6.6: The truth table to a logical operation, *x* ∧ *x*, is shown.

5.7 The binary operator ∨ defines logical operation **OR**. The truth table to a logical operation, *x* ∨ 1, is shown in Table 5.6.7, where *x* is a Boolean variable that is the first input and 1 is the second input. Based on Table 5.6.7, write a bio-molecular program to implement the function of the logical operation, *x* ∨ 1.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input | *x* ∨ 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

Table 5.6.7: The truth table to a logical operation, *x* ∨ 1, is shown.

5.8 The binary operator ∧ defines logical operation **AND**. The truth table to a logical operation, *x* ∧ 0, is shown in Table 5.6.8, where *x* is a Boolean variable that is the first input and 0 is the second input. Based on Table 5.6.8, write a bio-molecular program to implement the function of the logical operation, *x* ∧ 0.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input | *x* ∧ 0 |
| 0 | 0 | 0 |
| 1 | 0 | 0 |

Table 5.6.8: The truth table to a logical operation, *x* ∧ 0, is shown.

5.9 The unary operator ′ defines logical operation **NOT**. The truth table to a logical operation, (*x*′)′, is shown in Table 5.6.9, where *x* is a Boolean variable that is the first input. Based on Table 5.6.9, write a bio-molecular program to implement the function of the logical operation, (*x*′)′.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | *x*′ | (*x*′)′ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

Table 5.6.9: The truth table to a logical operation, (*x*′)′, is shown.

5.10 The binary operator ∧ defines logical operation **AND**, and the unary operator ′ defines logical operation **NOT**. The truth table to a logical operation, *x* ∧ *y*′, is shown in Table 5.6.10, where *x* and *y* are two Boolean variables that are respectively the first input and the second input. Based on Table 5.6.10, write a bio-molecular program to implement the function of the logical operation, *x* ∧ *y*′.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *x* ∧ *y*′ |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Table 5.6.10: The truth table to a logical operation, *x* ∧ *y*′, is shown.

**6.10 Exercises**

6.1 The binary operator ∨ defines logical operation **OR**, and the binary operator ∧ defines logical operation **AND**. The truth table to a logical operation, *x* ∨ (*x* ∧ *y*), is shown in Table 6.10.1, where *x* and *y* are Boolean variables that are respectively the first input and the second input. Based on Table 6.10.1, write a bio-molecular program to implement the function of the logical operation, *x* ∨ (*x* ∧ *y*).

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *x* ∨ (*x* ∧ *y*) |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Table 6.10.1: The truth table to a logical operation, *x* ∨ (*x* ∧ *y*), is shown.

6.2 The binary operator ∨ defines logical operation **OR**, and the binary operator ∧ defines logical operation **AND**. The truth table to a logical operation, *x* ∧ (*x* ∨ *y*), is shown in Table 6.10.2, where *x* and *y* are Boolean variables that are subsequently the first input and the second input. Based on Table 6.10.2, write a bio-molecular program to implement the function of the logical operation, *x* ∧ (*x* ∨ *y*).

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *x* ∧ (*x* ∨ *y*) |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Table 6.10.2: The truth table to a logical operation, *x* ∧ (*x* ∨ *y*), is shown.

6.3 The binary operator ∨ defines logical operation **OR**, and the binary operator ∧ defines logical operation **AND**. The truth table to a logical operation, *y* ∨ (*y* ∧ *x*), is shown in Table 6.10.3, where *x* and *y* are Boolean variables that are subsequently the first input and the second input. Based on Table 6.10.3, write a bio-molecular program to implement the function of the logical operation, *y* ∨ (*y* ∧ *x*).

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *y* ∨ (*y* ∧ *x*) |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Table 6.10.3: The truth table to a logical operation, *y* ∨ (*y* ∧ *x*), is shown.

6.4 The binary operator ∨ defines logical operation **OR**, and the binary operator ∧ defines logical operation **AND**. The truth table to a logical operation, *y* ∧ (*y* ∨ *x*), is shown in Table 6.10.4, where *x* and *y* are Boolean variables that are respectively the first input and the second input. Based on Table 6.10.4, write a bio-molecular program to implement the function of the logical operation, *y* ∧ (*y* ∨ *x*).

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *y* ∧ (*y* ∨ *x*) |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Table 6.10.4: The truth table to a logical operation, *y* ∧ (*y* ∨ *x*), is shown.

6.5 The unary operator ′ defines logical operation **NOT**, and the binary operator ∧ defines logical operation **AND**. The truth table to a logical operation, *x*′ ∧ *y*, is shown in Table 6.10.5, where *x* and *y* are Boolean variables that are subsequently the first input and the second input. Based on Table 6.10.5, write a bio-molecular program to implement the function of the logical operation, *x*′ ∧ *y*.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *x*′ ∧ *y* |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Table 6.10.5: The truth table to a logical operation, *x*′ ∧ *y*, is shown.

6.6 The unary operator ′ defines logical operation **NOT**, the binary operator ∨ defines logical operation **OR**, and the binary operator ∧ defines logical operation **AND**. The truth table to a logical operation, (*x* ∧ *y*) ∨ (*x*′ ∧ *y*′), is shown in Table 6.10.6, where *x* and *y* are Boolean variables that are respectively the first input and the second input. Based on Table 6.10.6, write a bio-molecular program to implement the function of the logical operation, (*x* ∧ *y*) ∨ (*x*′ ∧ *y*′).

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*y*) | (*x* ∧ *y*) ∨ (*x*′ ∧ *y*′) |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Table 6.10.6: The truth table to a logical operation, (*x* ∧ *y*) ∨ (*x*′ ∧ *y*′), is shown.

6.7 The unary operator ′ defines logical operation **NOT**. The truth table to a logical operation, *y*′, is shown in Table 6.10.7, where *x* and *y* are Boolean variables that are subsequently the first input and the second input. Based on Table 6.10.7, write a bio-molecular program to implement the function of the logical operation, *y*′.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *y*′ |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Table 6.10.7: The truth table to a logical operation, *y*′, is shown.

6.8 The unary operator ′ defines logical operation **NOT**. The truth table to a logical operation, *x*′, is shown in Table 6.10.8, where *x* and *y* are Boolean variables that are respectively the first input and the second input. Based on Table 6.10.8, write a bio-molecular program to implement the function of the logical operation, *x*′.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *x*′ |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Table 6.10.8: The truth table to a logical operation, *x*′, is shown.

6.9 The unary operator ′ defines logical operation **NOT**, and the binary operator ∨ defines logical operation **OR**. The truth table to a logical operation, *x* ∨ *y*′, is shown in Table 6.10.9, where *x* and *y* are Boolean variables that are subsequently the first input and the second input. Based on Table 6.10.9, write a bio-molecular program to implement the function of the logical operation, *x* ∨ *y*′.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *x* ∨ *y*′ |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Table 6.10.9: The truth table to a logical operation, *x* ∨ *y*′, is shown.

6.10 The unary operator ′ defines logical operation **NOT**, and the binary operator ∨ defines logical operation **OR**. The truth table to a logical operation, *x*′ ∨ *y*, is shown in Table 6.10.10, where *x* and *y* are Boolean variables that are respectively the first input and the second input. Based on Table 6.10.10, write a bio-molecular program to implement the function of the logical operation, *x*′ ∨ *y*.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *x*′ ∨ *y* |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Table 6.10.10: The truth table to a logical operation, *x*′ ∨ *y*, is shown.

**7.9 Exercises**

7.1 The binary operator ∨ defines logical operation **OR**. The truth table to a logical operation, *x* ∨ 1, is shown in Table 7.9.1, where *x* is a Boolean variable that is the first input and 1 is the second input. Based on Table 7.9.1, write a bio-molecular program to show *x* ∨ 1 = 1.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input | *x* ∨ 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

Table 7.9.1: The truth table to a logical operation, *x* ∨ 1, is shown.

7.2 The binary operator ∧ defines logical operation **AND**. The truth table to a logical operation, *x* ∧ 0, is shown in Table 7.9.2, where *x* is a Boolean variable that is the first input and 0 is the second input. Based on Table 7.9.2, write a bio-molecular program to demonstrate *x* ∧ 0 = 0.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input | *x* ∧ 0 |
| 0 | 0 | 0 |
| 1 | 0 | 0 |

Table 7.9.2: The truth table to a logical operation, *x* ∧ 0, is shown.

7.3 The binary operator ∨ defines logical operation **OR**. The truth table to logical operations, *x* ∨ 0 and *x*, is shown in Table 7.9.3, where for *x* ∨ 0 *x* is a Boolean variable that is the first input and 0 is the second input. Based on Table 7.9.3, write a bio-molecular program to prove *x* ∨ 0 = *x*.

|  |  |  |  |
| --- | --- | --- | --- |
| The first input (*x*) | The second input | *x* ∨ 0 | *x* |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |

Table 7.9.3: The truth table to logical operations, *x* ∨ 0 and *x*, is shown.

7.4 The binary operator ∧ defines logical operation **AND**. The truth table to logical operations, *x* ∧ 1 and *x*, is shown in Table 7.9.4, where for *x* ∧ 1 *x* is a Boolean variable that is the first input and 1 is the second input. Based on Table 7.9.4, write a bio-molecular program to demonstrate *x* ∧ 1 = *x*.

|  |  |  |  |
| --- | --- | --- | --- |
| The first input (*x*) | The second input | *x* ∧ 1 | *x* |
| 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Table 7.9.4: The truth table to logical operations, *x* ∧ 1 and *x*, is shown.

7.5 The binary operator ∨ defines logical operation **OR**, and the unary operator ′ defines logical operation **NOT**. The truth table to a logical operation, *x* ∨ *x*′, is shown in Table 7.9.5, where *x* is a Boolean variable that is the first input and its negation is the second input. Based on Table 7.9.5, write a bio-molecular program to show *x* ∨ *x*′ = 1.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input(*x*′) | *x* ∨ *x*′ |
| 0 | 1 | 1 |
| 1 | 0 | 1 |

Table 7.9.5: The truth table to a logical operation, *x* ∨ *x*′, is shown.

7.6 The binary operator ∧ defines logical operation **AND**, and the unary operator ′ defines logical operation **NOT**. The truth table to a logical operation, *x* ∧ *x*′, is shown in Table 7.9.6, where *x* is a Boolean variable that is the first input and its negation is the second input. Based on Table 7.9.6, write a bio-molecular program to prove *x* ∧ *x*′ = 0.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input(*x*′) | *x* ∧ *x*′ |
| 0 | 1 | 0 |
| 1 | 0 | 0 |

Table 7.9.6: The truth table to a logical operation, *x* ∧ *x*′, is shown.

7.7 The binary operator ∧ defines logical operation **AND**. The truth table to a logical operation, *x* ∧ *x*, is shown in Table 7.9.7, where *x* is a Boolean variable that is the first input and the second input. Based on Table 7.9.7, write a bio-molecular program to show *x* ∧ *x* = *x*.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input(*x*) | *x* ∧ *x* |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

Table 7.9.7: The truth table to a logical operation, *x* ∧ *x*, is shown.

7.8 The binary operator ∨ defines logical operation **OR**. The truth table to a logical operation, *x* ∨ *x*, is shown in Table 7.9.8, where *x* is a Boolean variable that is the first input and the second input. Based on Table 7.9.8, write a bio-molecular program to demonstrate *x* ∨ *x* = *x*.

|  |  |  |
| --- | --- | --- |
| The first input (*x*) | The second input(*x*) | *x* ∨ *x* |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

Table 7.9.8: The truth table to a logical operation, *x* ∨ *x*, is shown.

7.9 The binary operator ∨ defines logical operation **OR**. The truth table to logical operations, *x* ∨ *y* and *y* ∨ *x*, is shown in Table 7.9.9, where *x* and *y* are Boolean variables that are respectively the first input and the second input. Based on Table 7.9.9, write a bio-molecular program to prove *x* ∨ *y* = *y* ∨ *x* that satisfies the commutative law.

|  |  |  |  |
| --- | --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *x* ∨ *y* | *y* ∨ *x* |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |

Table 7.9.9: The truth table to logical operations, *x* ∨ *y* and *y* ∨ *x*, is shown.

7.10 The binary operator ∧ defines logical operation **AND**. The truth table to logical operations, *x* ∧ *y* and *y* ∧ *x*, is shown in Table 7.9.10, where *x* and *y* are Boolean variables that are respectively the first input and the second input. Based on Table 7.9.10, write a bio-molecular program to show *x* ∧ *y* = *y* ∧ *x* that satisfies the commutative law.

|  |  |  |  |
| --- | --- | --- | --- |
| The first input (*x*) | The second input (*y*) | *x* ∧ *y* | *y* ∧ *x* |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Table 7.9.10: The truth table to logical operations, *x* ∧ *y* and *y* ∧ *x*, is shown.