

# 7.1 Notes

2023年1月12日 22:56

## **Stages of the development cycle**

- Analysis
- Design
- Coding
- Testing

### **7.1.1 Analysis**

- The requirement is identified using abstraction and decomposition tool
- Requirements
  - A problem is clearly defined and set out so anyone working on the solution understands what is needed
  - Think in input and output
- Abstraction
  - The removal of unnecessary detail
- Decomposition
  - The breaking down of a complex problem into smaller parts and then subdivided into even smaller parts
  - Decomposed into: input, processes, output, storage

### **7.1.2 Design**

- Show the programmer what is to be done
  - All the task needed, how tasks perform and how they work together
- Three ways to document the design
  - Structure charts
  - Flowchart
  - Pseudocode

### **7.1.3 Coding and iterative testing**

- The programmers write the program
- The program is split into modules, which can then be developed by different programmers at the same time before being put together as a final program
- Iterative testing
  - The testing of each individual module over and over again and fixing the problems until it works correctly

### **7.1.4 Testing**

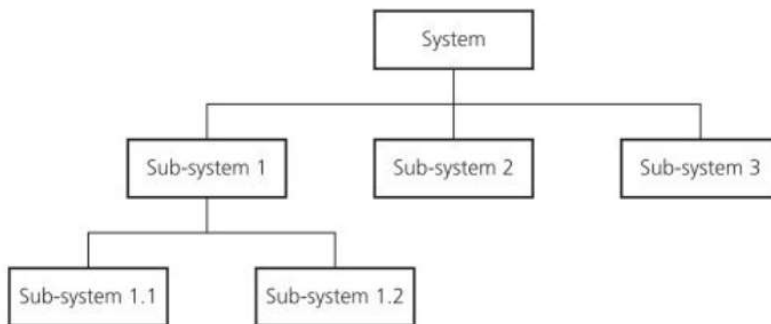
- Test data: Each of the modules are put together to create a final program
- The program is run many times with different sets of test data to see if it produces the correct outcomes and meets the program requirements
- This ensures that all the tasks completed work together as specified in the program design

# 7.2 Notes

2023年1月30日 19:11

## 7.2.1 Sub-systems

- Top-down design
  - The decomposition of a computer system into a set of sub-systems
  - Then breaking each sub-system down into a set of smaller sub-systems
  - Keeps breaking down until each sub-system just performs a single action
- Stepwise refinement
  - The process of breaking down into smaller sub-systems
- Benefits
  - Several programmers can work independently to develop and test different sub-systems for the same system at the same time
  - Reduces the development and testing time
- Sub-systems can be shown as a structure diagram



- Sub-routines can be shown in flow chart or pseudocode

## 7.2.2 Decomposing a problem

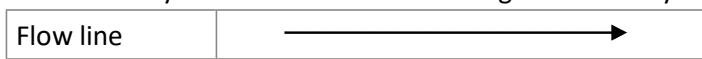
- Problems need to be decomposed into component parts
- Inputs
  - The data used by the system that needs to be entered while the system is active
- Processes
  - The tasks that need to be performed using the input data and any other previously stored data
- Outputs
  - Information that needs to be displayed or printed for the users of the system
- Storage
  - Data that needs to be stored in files on an appropriate medium for use in the future


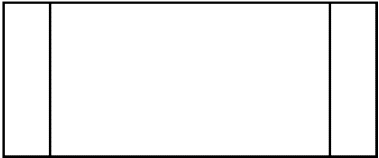
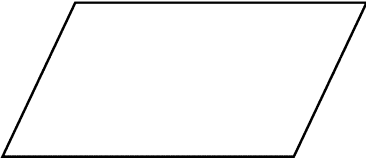
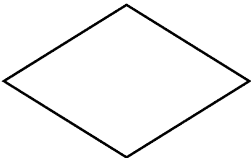
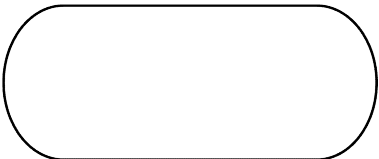
## 7.2.3a Structure diagrams

- Used to show top-down design in a diagrammatic form
- Hierarchical
  - Showing how a solution can be divided
  - Each level gives a more detailed breakdown

## 7.2.3b Flowchart

- Shows diagrammatically the steps required to complete a task and the order that they are to be performed
- These steps and order together are called an algorithm
- Effective way to communicate how an algorithm in a system or sub-system works



|                |   |
|----------------|---|
| Process        |   |
| Subroutine     |  |
| Input / Output |  |
| Decision       |  |
| Terminator     |  |

### 7.2.3c Pseudocode

- Rules
  - Use a non-proportional font - Consolas is suggested
  - All keywords are written in capital letters (e.g. OUTPUT)
  - All names given to data items and subroutines start with a capital letter (e.g. AdultPrice)
  - Where conditional and loop statements are used, repeated or selected statements are indented by two spaces.
- Code

|                           |  |
|---------------------------|--|
| If ... then ... else      | <pre>IF ?   THEN     ...   ELSE     ... ENDIF</pre>              |
| Case                      | <pre>CASE OF ?   ? : ...   ? : ...   OTHERWISE ... ENDCASE</pre> |
| For ... next              | <pre>FOR ? ← ? TO ?   ... NEXT</pre>                             |
| Repeat ... until          | <pre>REPEAT   ... UNTIL ?</pre>                                  |
| While ... do ... endwhile | <pre>WHILE ? DO   ... ENDWHILE</pre>                             |

## 7.4 Standards methods of solution

2024年1月18日 17:40

### 7.4.1 Totalling

```
Total ← 0
OUTPUT "Enter Class Size"
INPUT ClassSize
FOR Counter ← 0 TO ClassSize
    Total ← Total + StudentMark[Counter]
NEXT Counter
OUTPUT Total
```

### 7.4.2 Counting

```
PassCount ← 0
OUTPUT "Enter Class Size"
INPUT ClassSize
FOR Counter ← 0 TO ClassSize
    INPUT StudentMark
    IF StudentMark > 50
        THEN
            PassCount ← PassCount + 1
NEXT Counter
OUTPUT PassCount
```

### 7.4.3 Maximum, minimum and average

```
MaximumMark ← StudentMark[0]
MinimumMark ← StudentMark[0]
OUTPUT "Enter Class Size"
INPUT ClassSize
FOR Counter 1 ← TO ClassSize
    IF StudentMark[Counter] > MaximumMark
        THEN
            MaximumMark ← StudentMark[Counter]
    ENDIF
    IF StudentMark[Counter] < MinimumMark
        THEN
            MinimumMark ← StudentMark[Counter]
    ENDIF
NEXT Counter
OUTPUT MaximumMark
OUTPUT MinimumMark
```

### 7.4.4 Linear search

```
OUTPUT "Please enter the name to find"
INPUT Name
OUTPUT "Enter ClassSize"
INPUT ClassSize
Found ← FALSE
Counter ← 1
REPEAT
    IF Name = StudentName[Counter]
        THEN
            Found ← TRUE
        ELSE
            Counter ← Counter + 1
```

```

    ENDF
UNTIL Found OR Counter > ClassSize
IF Found
    THEN
        OUTPUT Name, "found at position", Counter, "in the list"
    ELSE
        OUTPUT Name, "not found"
    ENDF
ENDIF

```

#### **7.4.5 Bubble sort**

```

swapped ← "yes"
WHILE swapped = "yes" DO
    swapped ← "no"
    FOR index ← 0 TO 4
        IF numbers[index] > numbers[index +1]
            THEN
                temp ← numbers[index]
                numbers[index] ← numbers[index +1]
                numbers[index +1] ← temp
                swapped ← "yes"
            ENDF
        NEXT index
    ENDWHILE
    OUTPUT "The list is now in ascending order"
    OUTPUT numbers

```

©Xingzhi Lu 2024

# 7.5 Validation and verification

2024年1月18日 9:54

## 7.5.1 Validation

- Validation
  - The automated checking by a program that data is reasonable before it is accepted into a computer system
  - Do not check if data is correct
  - If data is rejected then a message should be output explaining why the data was rejected along with another opportunity to enter the data
- Types of validations
  - Range check
  - Length check
  - Type check
  - Presence check
  - Format check
  - Check digit
- Range check
  - Checks that the value of a number is between an upper value and a lower value
  - e.g. check if percentage marks are all between 0 and 100 inclusive
- Length check
  - Checks either if the data contains an exact number of characters or if the data entered has a reasonable number of characters
  - e.g. a password should have exactly 8 characters, family name entered should be 2-30 characters long
- Type check
  - Check that the data entered is of a given data type
  - e.g. the number of siblings should be an integer, not a real
- Presence check
  - Checks to ensure that some data have been entered and the value have not been left blank
  - e.g. the name of the person must be entered
- Format check
  - Checks that characters entered conform to a pre-defined pattern
  - e.g. date must be in the form dd/mm/yyyy
- Check digit
  - The final digit included in a code that is calculated from all the other digits in the code to check if the code entered is correct
  - It is used for codes such as ISBN or VIN
  - Used to identify errors caused by mis-typing or mis-scanning a barcode
  - Errors that can be identified
    - An incorrect digit entered
    - Transposition errors where two numbers have changed order
    - Omitted or extra digits
    - Phonetic errors

## 7.5.2 Verification

- Verification
  - Checking data has been accurately copied from one source to another e.g. input into a computer or transferred from one part of a computer system to another
- Verification methods
  - Double entry
    - The data is entered twice, sometimes by different operators
    - The computer compares both entries and if they are different then it outputs an error message requiring the data is entered again.

- Screen / visual check
  - A manual check completed by the user who is entering the data
  - When the data entry is complete the data is displayed on the screen and the user is asked to confirm that it is correct before continuing
  - The user either checks the data on the screen against a paper document that is being used as an input form or check from their own knowledge

© Xingzhi Lu 2024

## 7.6 Test data

2024年4月3日 23:27

### **7.6.1 How to suggest and apply suitable test data**

- Purpose of using test data
  - Check that the program works as expected
  - Check for logic/runtime errors
  - Check that the program rejects any invalid data that is input
  - Check that the program only accepts reasonable data
- Types of test data
  - Normal data
    - Data that the algorithm would normally use
  - Abnormal data (or erroneous data)
    - Data that should be rejected
  - Extreme data
    - Data at the outer extremes of the range
  - Boundary data
    - Normal data on the valid boundary and abnormal data on the valid boundary
    - Boundary data is in a pair e.g. 0 and -1, 100 and 101

© Xingzhi Lu 2024



# 8.1 Programming concepts

2024年4月3日 23:28

## 8.1.6 Procedures and functions

|                             |  |
|-----------------------------|--|
| <b>Procedure pseudocode</b> | <pre>PROCEDURE ProcedureName (Parameter : TYPE)     //Code ENDPROCEDURE  CALL ProcedureName</pre>      |
| <b>Function pseudocode</b>  | <pre>FUNCTION FunctionName (Parameter : TYPE) RETURNS TYPE     //Code     RETURN ... ENDFUNCTION</pre> |

- Procedure
  - A set of programming statements grouped together under a single name that can be called to perform a task at any point in a program
  - A procedure needs to be called (CALL in Pseudocode)
  - Parameters may be passed to the procedure to be used
- Function
  - A set of programming statements grouped together under a single name that can be called to perform a task at any point in a program
  - In contrast to a procedure, a function will return a value back to the main program
  - A function is used as part of an expression and is called by its identifier
  - Parameters may be passed to the function to be used
- Parameters
  - Are the variables that store the values of the arguments passed to a procedure or function
  - Some but not all procedures and functions will have parameters.
- Why procedures / functions
  - It enables many programming statements to be grouped together using a single identifier
  - Procedures / functions can be reused in the same program or other programs / create modular programs
  - Less duplicated code is required - shorter program
  - Different programmers can work on different procedures / functions in the same project at the same time which speeds up development
- Why parameters
  - To pass values from the main program to a procedure / function
  - So that they can be used in the procedure / function
  - Parameters mean that the same procedure can be used with different data

## 8.1.7 Library routines

- Definition
    - A standard subroutine that is available for immediate use.
- ```
DIV(<identifier1>, <identifier2>)
```
- Returns the quotient of `identifier1` divided by `identifier2` with the fractional part discarded.
- ```
MOD(<identifier1>, <identifier2>)
```
- Returns the remainder of `identifier1` divided by `identifier2`
- The identifiers are of data type integer.

Examples – MOD and DIV

```
DIV(10, 3) returns 3
MOD(10, 3) returns 1
```

ROUND(<identifier>, <places>)

Returns the value of the identifier rounded to `places` number of decimal places.

The identifier should be of data type real, `places` should be data type integer.

RANDOM()

- Returns a random number between 0 and 1 inclusive.

Example – ROUND and RANDOM

```
Value ← ROUND (RANDOM() * 6, 0) // returns a whole number between 0 and 6
```

### **8.1.8 Creating a maintainable program**

- Why does a program need to be maintainable
  - People need to be able to know what the program does when it is altered by someone else / altered after a long period of time
  - A program might not have any documentation attached to it and therefore it is only the actual code that can be used to work out what the program does.
  - Using meaningful identifiers, including comments and splitting up the program using procedures and functions can make a program easier to understand, follow and maintain
- Methods to do so
  - Meaningful identifiers
    - Identifiers for variables, constants, arrays and procedures and functions should all be clear and meaningful
    - They will help the programmer / future programmers to recognise the purpose of a variable as well as tracking it through a program
  - Comments
    - Ensures that a programmer can easily find specific sections as well as knowing the purpose of that section of the code.
  - Procedures and functions
    - Code can be split into smaller sections using procedures and functions (subroutines or modules)
    - Modular programs are easier to update, understand, debug and maintain compared to one large program
  - White space

# String handling

2024年4月3日 23:14

## String operations (8.1.4 (e))

`LENGTH(<identifier>)`

Returns the integer value representing the length of string. The identifier should be of data type string.

`LCASE(<identifier>)`

Returns the string/character with all characters in lower case. The identifier should be of data type string or char.

`UCASE(<identifier>)`

Returns the string/character with all characters in upper case. The identifier should be of data type string or char.

`SUBSTRING(<identifier>, <start>, <length>)`

Returns a string of length `length` starting at position `start`. The identifier should be of data type string, `length` and `start` should be positive and data type integer.

Generally, a start position of 1 is the first character in the string.

### Example – string operations

`LENGTH("Happy Days")` will return 10

`LCASE('W')` will return 'w'

`UCASE("Happy")` will return "HAPPY"

`SUBSTRING("Happy Days", 1, 5)` will return "Happy"

© Xingzhi Lu

## 8.3 File handling

2024年4月3日 23:15

### 8.3.1 Purpose of storing data in a file

- When running a program, any data that is entered into the program will be lost when the program is completed
- Therefore, if data is required again by a computer program it can be stored in a file.
- Benefits
  - Data can be stored permanently
  - Data can be accessed by the same program at a later date
  - Data can be accessed by another program
  - Data can be sent to another computer
  - Data can be used as a back up of the data

### 8.3.2 Using files

```
DECLARE ExampleString : STRING
//Declares the ExampleString variable

OPENFILE "FileA.txt" FOR READ
READFILE "FileA.txt", ExampleString
//Reads the contents of FileA.txt and assigns it to ExampleString
CLOSEFILE "FileA.txt"

OPENFILE "FileB.txt" FOR WRITE
WRITEFILE "FileB.txt", ExampleString
//writes the value of ExampleString to FileB.txt
CLOSEFILE "FileB.txt"
```

#### Handling files (8.3.2)

It is good practice to explicitly open a file, stating the mode of operation, before reading from or writing to it. This is written as follows:

```
OPENFILE <File identifier> FOR <File mode>
```

The file identifier will be the name of the file with data type string. The following file modes are used:

`READ` for data to be read from the file

`WRITE` for data to be written to the file. A new file will be created and any existing data in the file will be lost.

A file should be opened in only one mode at a time.

- Data is read from the file (after the file has been opened in `READ` mode) using the `READFILE` command as follows:

```
READFILE <File Identifier>, <Variable>
```

When the command is executed, the data item is read and assigned to the variable.

Data is written into the file after the file has been opened using the `WRITEFILE` command as follows:

```
WRITEFILE <File identifier>, <Variable>
```

When the command is executed, the data is written into the file. Files should be closed when they are no longer needed using the `CLOSEFILE` command as follows:

```
CLOSEFILE <File identifier>
```

# 9.1 Databases

2023年6月19日 18:49

## 9.1.1 Databases intro

- Database
  - Simply a collection of data that holds details about people, things or events
  - The data is structured carefully so that information can be extracted from the database as needed.
- Table
  - A table contains data about one type of person, thing or event
  - Given a meaningful name
- Record
  - A record in a table contains data about a single person, thing or event
  - Rows
- Field
  - A field contains a specific piece of data about each single item and will have a field name
  - Columns
- Validation
  - Type
    - Data entered is of a given data type
  - Format
    - Check that the characters entered confirm to a pre-defined pattern
  - Range
    - The value of a number is between an upper value and a lower value
  - Presence
    - Ensure that some data has been entered and the value has not been left blank
  - Length
    - Either:
      - The data contains an exact number of characters
      - Data entered is a reasonable number of characters
  - Check digit
    - The final digit included in a code calculated from all the other digits in the code
    - Used to identify errors in data entry caused by mis-typing or mis-scanning a barcode

## 9.1.2 Data types in databases

- Data types
  - Text/alphanumeric
  - Character
  - Boolean
  - Integer
  - Real
  - Date/time

## 9.1.3 Primary keys

- Primary Key
  - Contains no repeating values
  - A field used to uniquely identify each record in a database table
  - An existing field could be used as a primary key or an additional field needs to be added
- Using existing field / adding a field
  - Existing field
    - Some data will already contain a field that has unique values
    - The field can be used for the primary key
    - e.g. ISBN for books
  - Adding a field

- Other data has fields which all contain repeating data
- Another field has to be added to create a unique identifier
- This could simply be an incrementing number e.g. unique number for each student

#### 9.1.4 Structured Query Language (SQL)

```

i SELECT Title, Genre1, Blu-ray
FROM 2018MOV
WHERE Genre1 = "Fantasy";

```

```

★ SELECT Title, Genre1, Blu-ray (choose the fields to display)
FROM 2018MOV (identify the table the contains the data)
WHERE Genre1 = "Fantasy"; (specify the records to display by identifying criteria)

```

⚠ SELECT \* includes every field

⚠ Operators: = equal to, <> not equal to, > greater than, < less than, >= greater than or equal to, <= less than or equal to

#### • ORDER BY

```

SELECT HospitalNumber, FirstName, FamilyName
FROM PATIENT
WHERE Consultant = 'Mr Smith'
ORDER BY FamilyName;

```

- ORDER BY FamilyName DESC; would order the list in descending order (Z-A, 100-1) by FamilyName
- ORDER BY FamilyName, FirstName; would sort the output by FamilyName and then FirstName

#### • SUM

- SELECT SUM (BedNumber)
- FROM PATIENT
- WHERE Consultant = "Mr Smith";
- Only integer / real fields

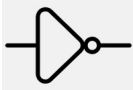
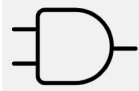
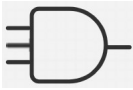
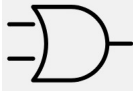
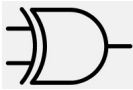
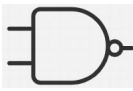
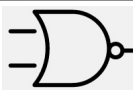
#### • COUNT

- SELECT COUNT (FirstName)
- FROM PATIENT
- WHERE WardNumber = 6;
- Can be used on any field

# 10.1-2 Logic gate symbols and functions

2023年5月15日 18:42

## Types of logic gates

| Name  | Symbol and Screenshot from the simulator   | Truth Table  | Logic Notation |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
|-------|--|--|----------------|--------|--------|---|---|---|---|---|---------------------|---|---|---|------------------------|---|---|---|---|---|-------------------------|
| NOT   |   | <table border="1"> <thead> <tr> <th>Input</th> <th>Output</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>X</td> </tr> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>   | Input          | Output | A      | X | 0 | 1 | 1 | 0 | $X = \text{NOT } A$ |   |   |   |                        |   |   |   |   |   |                         |
| Input | Output   |  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| A     | X  |  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 0     | 1  |  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 1     | 0  |  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| AND   | <br> | <table border="1"> <thead> <tr> <th>Input</th> <th>Input</th> <th>Output</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>B</td> <td>X</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table> | Input          | Input  | Output | A | B | X | 0 | 0 | 0                   | 0 | 1 | 0 | 1                      | 0 | 0 | 1 | 1 | 1 | $X = A \text{ AND } B$  |
| Input | Input  | Output   |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| A     | B  | X  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 0     | 0  | 0  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 0     | 1  | 0  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 1     | 0  | 0  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 1     | 1  | 1  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| OR    |    | <table border="1"> <thead> <tr> <th>Input</th> <th>Input</th> <th>Output</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>B</td> <td>X</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table> | Input          | Input  | Output | A | B | X | 0 | 0 | 0                   | 0 | 1 | 1 | 1                      | 0 | 1 | 1 | 1 | 1 | $X = A \text{ OR } B$   |
| Input | Input  | Output   |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| A     | B  | X  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 0     | 0  | 0  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 0     | 1  | 1  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 1     | 0  | 1  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 1     | 1  | 1  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| XOR   |   | <table border="1"> <thead> <tr> <th>Input</th> <th>Input</th> <th>Output</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>B</td> <td>X</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table> | Input          | Input  | Output | A | B | X | 0 | 0 | 0                   | 0 | 1 | 1 | 1                      | 0 | 1 | 1 | 1 | 0 | $X = A \text{ XOR } B$  |
| Input | Input  | Output   |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| A     | B  | X  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 0     | 0  | 0  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 0     | 1  | 1  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 1     | 0  | 1  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 1     | 1  | 0  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| NAND  |   | <table border="1"> <thead> <tr> <th>Input</th> <th>Input</th> <th>Output</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>B</td> <td>X</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table> | Input          | Input  | Output | A | B | X | 0 | 0 | 1                   | 0 | 1 | 1 | 1                      | 0 | 1 | 1 | 1 | 0 | $X = A \text{ NAND } B$ |
| Input | Input  | Output   |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| A     | B  | X  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 0     | 0  | 1  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 0     | 1  | 1  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 1     | 0  | 1  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 1     | 1  | 0  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| NOR   |   | <table border="1"> <thead> <tr> <th>Input</th> <th>Input</th> <th>Output</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>B</td> <td>X</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> </tbody> </table>   | Input          | Input  | Output | A | B | X | 0 | 0 | 1                   | 0 | 1 | 0 | $A = A \text{ NOR } B$ |   |   |   |   |   |                         |
| Input | Input  | Output   |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| A     | B  | X  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 0     | 0  | 1  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |
| 0     | 1  | 0  |                |        |        |   |   |   |   |   |                     |   |   |   |                        |   |   |   |   |   |                         |

|  |  |   |   |   |  |
|--|--|---|---|---|--|
|  |  | 1 | 0 | 0 |  |
|  |  | 1 | 1 | 0 |  |

©Xingzhi Lu 2024