# Task 1

Link to my repository under the “exam” branch is: https://github.com/00010023/csf.coursework/tree/exam

# Task 2

**Task A (Part 1):** In order to convert a decimal number to binary, we need to divide by the base 2 to get the digits from the remainders:

| Division by 2 | Quotient (number after division) | Remainder (in Digit form) | Bit in numbers order |
| --- | --- | --- | --- |
| (10023)/2 | 5011 | 1 | 0 |
| (5011)/2 | 2505 | 1 | 1 |
| (2505)/2 | 1252 | 1 | 2 |
| (1252)/2 | 626 | 0 | 3 |
| 626/2 | 313 | 0 | 4 |
| 313/2 | 156 | 1 | 5 |
| (156)/2 | 78 | 0 | 6 |
| (78)/2 | 39 | 0 | 7 |
| (39)/2 | 19 | 1 | 8 |
| (19)/2 | 9 | 1 | 9 |
| (9)/2 | 4 | 1 | 10 |
| (4)/2 | 2 | 0 | 11 |
| (2)/2 | 1 | 0 | 12 |
| (1)/2 | 0 | 1 | 13 |

**Task A (Part 2):** In order to convert a decimal number to binary, we need to divide by the base 16 to get the digits from the remainders:

|  |  |  |  |
| --- | --- | --- | --- |
| **Division by 16** | **Quotient (number after division)** | **Remainder (in Digit form)** | **Bit in numbers order** |
| (00010023)/16 | 626 | 7 | 0 |
| (626)/16 | 39 | 2 | 1 |
| (39)/16 | 2 | 7 | 2 |
| (2)/16 | 0 | 2 | 3 |

**Task B (Part 1):**

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

**Task B (Part 2):**

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

**Task C:**

To be clarified, hexadecimal is often used by programmers in order to address lots of thing to its memory. As an example, errors or colors. Even, if you mentioned that 16 is 2’s 4th power, hexadecimal is still related to binary system whereas we as developers use it in order to simplify some of binary things. Also, usage of hexadecimal can be proficient in memory addressing as I said before. For example in assembly programming language you will be addressing all datas by its hexadecimal.

# Task 3

**Father =>** OBID

**Mother =>** ZEBO

**Both =>** OBIDZEBO

|  |  |  |
| --- | --- | --- |
| Characters | Frequency | Encoding |
| O | 2 | 01 |
| B | 2 | 11 |
| I | 1 | 000 |
| D | 1 | 100 |
| Z | 1 | 010 |
| E | 1 | 110 |
|  | | |

**Encoding: 2\*2+2\*2+1\*3+1\*3+1\*3+1\*3=4+4+3+3+3+3=8+6+6=8+12=20 bits**

**\*True raw version of picture is available here: <https://sauce.genemator.me/> a file namely csf.heic**

# Task 4

* **Instruction:** co-operating *10023* with *45237* and getting **[ 1, 0, 0, 2, 3, 4, 5, 2, 3, 7 ]**
* **Sorting:** [ ~~1~~, ~~0~~, ~~0~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~2~~, ~~3~~, ~~7~~ ] and getting ☞ [ 0, 0, 1, 2, 2, 3, 3, 4, 5, 7 ]
* **Looking for number “2”:**
  1. X = “2”
  2. Midpoint is “2” in the 5th position (10/2 => 5) [ ~~0, 0, 1, 2,~~ 2, ~~3, 3, 4, 5, 7~~ ]
  3. x = midpoint (2 == 2), number is found
  4. Stop program, ez katka =)
* **But in case of another “2” (to avoid xalyava):**
  1. X = “2” => “1”
  2. Midpoint is “2” in the 5th position (10/2 => 5) [ 0, 0, 1, 2, 2, ~~3, 3, 4, 5, 7~~ ]
  3. X = midpoint (2 == 2), ignoring & searching for another “2”
  4. Midpoint is 1 in the 3rd position (5/2 = 2.5 => 3)
  5. X > midpoint (2 > 1), ignoring the range on the left
  6. Midpoint is 2 (3/2 = 1.5 => 2) => (2 == 2), number is found

# Task 5

**Explanation:** Paged Memory management differentiates with other with its unique allocation where page is a unit of logical and frame is a unit of physical memory of a program. Also, the size of pages and frames are the same.

**Case A: If the frame size is 1024, what is the physical address associated with the logical address <2, 85>? Provide explanation.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Page** | **0** | **1** | **2** | **3** | **4** |
| **Frame** | **4** | **3** | **5** | **6** | **7** |

**By using this formula: Number of Frame \* Page Size + Offset where 2 is indicator of frame and 86 is offset and also 1024 is page size. So, answer is:**

**5 \* 1024 + 85 = 5205**

**Case B: If the frame size is 1024, what is the physical address associated with the logical address <0, 1026>? Provide explanation.**

**By using this formula: Number of Frame \* Page Size + Offset where 0 is indicator of frame and 1026 is offset and also 1024 is page size. However, offset is bigger than frame where 1024 < 1026 and this question with its requirements itself is invalid.**

# Task 6

So in this task I’ll be comparing two process model methods which are waterfall and DevOps. DevOps actually aims to improves and focuses on time to market by lowering failure of the project by making consistent updates. However, Waterfall method is known for its being a traditional method. On waterfall methodology, you can’t proceed to the next step while above one won’t be completed, kind of a connected chain that depends on each other, where every details matters for 100%. To be mentioned, there is no way to go back to change or modify while on DevOps still it has chance for that.

# Task 7

In this task, I’ll be comparing tree and star topology. On star topology, all hosts are connected into a central machine namely hub by using point-to-point connection type. However, on the other hand, tree topology is better known as Hierarchical Topology and this is the most used type of topologies. In addition, this topology extends star topology by including properties of bus topology. In comparison, star topology is mostly suitable for startups and small companies where all hosts need to create a connectivity between each structure. Tree topology can be widely used in large companies where all connections are shared via buildings & structures.

# Task 8

a = 5

b = 10

*def* exampleReturn(first\_number, second\_number): *# <- can take argument*

result = first\_number + second\_number *# <- has ability to initiate inline variable*

*return* result *# <- return a specific value, so it can be passed as argument on another function*

*pass # <- ending function here*

*def* exampleVoid(first\_number, second\_number): *# <- can take argument*

result = first\_number + second\_number *# <- has ability to initiate inline variable*

print(result) *# <- does everything on inline mode and doesn't return something*

*pass # <- ending function here that does something without returning some value*

*if* \_\_name\_\_ == '\_\_main\_\_':

exampleVoid(exampleReturn(a, b), b)

*pass*

# Task 9

a = 5

b = 10

*def* calculator(first\_number, second\_number, operator):

*if* operator == "+" *or* operator == "add":

print(f"{first\_number} + {second\_number} is", first\_number + second\_number)

*elif* operator == "-" *or* operator == "subtract":

print(f"{first\_number} - {second\_number} is", first\_number - second\_number)

*elif* operator == "\*" *or* operator == "multiply":

print(f"{first\_number} \* {second\_number} is", first\_number \* second\_number)

*elif* operator == "/" *or* operator == "divide":

print(f"{first\_number} / {second\_number} is", first\_number / second\_number)

*else*:

print("Invalid operator!")

*pass*

*class* Calculator:

*def \_\_init\_\_*(*self*, first\_number, second\_number):

*self*.first\_number = first\_number

*self*.second\_number = second\_number

*def* addition(*self*):

print(f"The addition of {*self*.first\_number} and {*self*.second\_number} is:",

*self*.first\_number + *self*.second\_number)

*def* subtraction(*self*):

print(f"The addition of {*self*.first\_number} and {*self*.second\_number} is:",

*self*.first\_number - *self*.second\_number)

*def* multiplication(*self*):

print(f"The addition of {*self*.first\_number} and {*self*.second\_number} is:",

*self*.first\_number \* *self*.second\_number)

*def* division(*self*):

print(f"The addition of {*self*.first\_number} and {*self*.second\_number} is:",

*self*.first\_number / *self*.second\_number)

*def* is\_greater\_or\_not(*self*):

*if self*.first\_number > *self*.second\_number:

print(f"The number {*self*.first\_number} is greater than {*self*.second\_number}")

*elif self*.first\_number < *self*.second\_number:

print(f"The number {*self*.first\_number} is less than {*self*.second\_number}")

*elif self*.first\_number == *self*.second\_number:

print(f"The number {*self*.first\_number} is equal to {*self*.second\_number}")

*if* \_\_name\_\_ == '\_\_main\_\_':

*try*:

*# Class usage*

process = Calculator(a, b)

process.addition()

process.subtraction()

process.multiplication()

process.is\_greater\_or\_not()

*# Function usage*

calculator(a, b, "+")

*except* Exception *as* error:

print(f"Error occurred: {error}”)