# **CSE3015 / CSE3215 DIGITAL LOGIC DESIGN TERM PROJECT**

**1st Iteration**



Ömer Faruk Atlıoğlu 150120018

Ömer Faruk Geysoğlu 150120060

Furkan Acar 150121534

Ufuk Acar 150121071

**1. Instruction Set Architecture (ISA) Design**

Table Implementation

metin, ekran görüntüsü, sayı, numara, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu

Our team successfully created a table that significantly aided the development process. As per our instructor's guidelines, we ensured that the opcodes (operation codes) were distinct. This was a critical aspect of our ISA design, as unique opcodes are essential for the clear identification and execution of instructions by a processor.

We declared these opcodes by default, setting a standard format for them. This preemptive step was fundamental in streamlining the coding process later on, as it allowed us to reference a consistent, predefined set of opcodes.

The table we created served as a blueprint for implementing our code. It provided a clear, organized view of our ISA, allowing us to efficiently map out and develop the necessary functionalities.

**2. Coding and Implementation**

*Adherence to Object-Oriented Programming (OOP)*

In our coding approach, we strictly adhered to the principles of object-oriented programming (OOP). This methodology was pivotal in organizing our code, making it more modular, reusable, and easier to understand.

*Class Implementation for Instructions*

For each instruction, we implemented a separate class. These classes shared a common field, opcode, to signify the operation code associated with each instruction. To promote code reusability and maintainability, we extended all these classes from a base class where the opcode was declared by default, in line with our ISA table.

*The Operation Class*

A crucial part of our implementation was the Operation class. This class served as the hub for handling various actions related to our instructions. It includes several functions, such as:

Converting binary to decimal.

Converting binary to hexadecimal.

Combining array lists.

Concatenating array lists into strings.

convertBinaryToHex Function

An example of these functions is convertBinaryToHex, which takes an ArrayList<Character> representing a binary number and converts it into hexadecimal. This conversion is achieved through intermediate conversion to decimal, utilizing Java's library functions. The result is returned as an ArrayList<Character> representing the hexadecimal number.

*Model Fields and Data Handling*

Each model in our system has fields corresponding to the ISA table, and their data types are ArrayList<Character>. At the project's culmination, we concatenate all these fields and convert them to the required format (binary, decimal, or hexadecimal), as needed.

*checkInstructionType Class*

We developed the checkInstructionType class to determine the type of instruction being processed and the corresponding operations to be performed. This class plays a crucial role in interpreting and executing the instructions correctly.

**File Handling**

For input and output operations, we utilize input.txt and output.txt files, respectively. The input.txt file serves as the source of instructions for our program, and the processed results are written to output.txt. Users can easily modify the input.txt file to change the input, ensuring flexibility and ease of use.

**Conclusion**

In summary, our project's development involved meticulous planning and execution, adhering to OOP principles and efficient coding practices. The ISA design, facilitated by the opcode table, and the modular implementation of classes, especially the Operation class, were key to our project's success. The ability to handle and convert data efficiently and the flexibility offered by external text files for input and output further enhanced the functionality and user-friendliness of our software. **Input and Output Examples:  
metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

****