Structure du Programme :

Le bytecode est organisé en une liste de “commandes”, chacune commençant par un identifiant de commande, suivi de ses arguments. Chaque commande est traitée séquentiellement, et sera utilisée pour construire le programme final.

Commandes :

Une commande consiste en un identifiant de commande et un nombre variable d’arguments. L’identifiant de commande est un entier de 1 octet, et les arguments peuvent être de différents types. L’identifiant de commande détermine quelle commande est exécutée.

La structure d’une commande est la suivante dans le bytecode :

```
<command_id> <arg1_size> <arg1> <arg2_size> <arg2> ...
command_id: entier de 1 octet
arg1_size: entier de 1 octet (taille du premier argument)
arg1: taille variable (dépend de arg1_size), valeur du premier argument

EX:
Supposons que nous ayons une commande avec l'id 1, qui prend 2 arguments entiers et 1 argument chaîne. Le bytecode pour cette commande ressemblerait à ceci :

0102010002000008616161616161616161

cmd_id  arg1_size  arg1  arg2_size  arg2  arg3_size arg3
01      02         0100  02         0000  08        6161616161616
```

Arguments :

Il y a 5 types d’arguments :

1. Integer

Un entier signé de 16 bits.
Important : Stocké au format little-endian.

2. BigInt

Un entier signé de 64 bits.
Important : Stocké au format little-endian.

3. String

Une chaîne de caractères encodée en UTF-8.
Important : La chaîne n’est pas terminée par un caractère nul et est en big endian

4. Bytes

Une séquence d’octets.

5. Enums

Un entier signé de 16 bits représentant une valeur d’énumération.
Important : Stocké au format little-endian.

Voici les 2 énumérations utilisées et leurs valeurs :

Register:

```
0x01 -> RBX
0x02 -> R10
0x03 -> R11
0x04 -> R13
0x05 -> R14
0x06 -> R15
```

JumpCondition:

```
0x01 -> EQUAL
0x02 -> NOT_EQUAL
0x03 -> GREATER
0x04 -> GREATER_OR_EQUAL
0x05 -> LESS
0x06 -> LESS_OR_EQUAL
```