Apprenez à créer un sapin de Noël style 70's avec CESIL et un Raspberry Pi

CESIL - acronyme signifiant Computer Education in Schools Instructional Language - a été conçu dans les années 70 comme tentative de faire découvrir le monde de la programmation informatique à la jeune génération. Sans ordinateur dans les écoles, les élèves devaient écrire les programmes sur papier et les envoyer à leur centre d'informatique local. Les résultats leur étaient retournés par la poste une semaine plus tard !

CESIL est un langage d'assemblage très simplifié, avec une base applicative très limitée, il est facile de l'apprendre et d'écrire des programmes simples avec. Comme ce n'est pas le propre de CESIL d'être vraiment passionnant, j'en ai donc écrit un interpréteur en BASIC, et ajouté un sapin de Noël avec des lumières féériques programmables ! Le sapin a 4 rangées de 8 lampes. Imaginez-le comme une grille de 8 de large et de 4 de haut.

Un programme CESIL est essentiellement constitué de trois colonnes de texte. La première colonne (qui peut être vide) est l'étiquette. C'est un emplacement dans le programme vers lequel il est possible de "sauter" depuis un autre endroit. La colonne du milieu contient l'opérateur - il s'agit de l'instruction à exécuter, et la dernière colonne est l'opérande - c'est-à-dire la donnée que l'instruction va utiliser. Cette donnée peut être le nom d'une étiquette s'il s'agit d'une instruction de saut, elle peut être un nombre ou faire référence à un emplacement mémoire nommé, ou une variable.

Mes ajouts à la machine CESIL consistent en deux registres supplémentaires (trois au total) afin de conserver les positions des rangées et colonnes des lampes, une instruction de couleur pour définir la couleur d'une lampe ainsi qu'une fonctionnalité de sous-programme. Le programme peut atteindre 256 lignes et contenir jusqu'à 256 variables.

La meilleure façon d'expliquer son fonctionnement consiste à étudier un programme réel. Celui-ci lit un nombre depuis le clavier et affiche une table de multiplication :

```
# mtable:
# Générateur de table de multiplication
  line
  print "Générateur de table de
multiplication"
  line
  print "Quelle table"
  in
  store table
  load 1
  store index # Index fois ....

loop: load index
  out
  print " FOIS "
  load table
  out
  print " = "
  mul index # Table était dans
l'accumulateur
  out
  line
  load index # Ajoute 1 à l'index
  add 1
  store index
  sub 11 # Soustrait 11 pour compter de
1 à 10
  jineg loop # Si <0 alors sauter à
l'étiquette loop
  halt
```

Les lignes vides sont autorisées et les commentaires commencent par le symbole #. Quasiment tout devrait pouvoir se passer d'explication, mais avec seulement un accumulateur, absolument tout doit être transféré depuis ou vers la mémoire - l'instruction "store table" enregistre l'accumulateur dans une variable appelée "table".