Un compilateur pour un micro-assembleur

La clarté et la précision des commentaires de vos codes seront prises en compte dans l'évaluation.
En cas de doute concernant l'énoncé,
vous poursuivrez votre réponse en expliquant vos hypothèses.

Internet est autorisé

GIA, ING1

1. Syntaxe d'un programme écrit en micro_assembleur

Un programme micro-assembleur est composé de deux parties : une partie dédiée à la déclaration des variables et une autre à la définition des instructions. Les instructions sont numérotées de manière séquentielle partant de 0 (voir Figure 1).

Var :

variable1: type1;

variable2: type2;

variable3: type3;

...

Instructions :

0 : Instruction 0

1 : Instruction 1 ;

2 : Instruction 2 ;

3 : Instruction 3 ;

...

Figure 1 : Syntaxe d'un programme micro-assembleur

2. Partie déclaration

La partie déclaration décrit les variables du programme. Une variable a un nom et un type qui est soit :

- 1 octet, défini par le mot clé $byte$,
- une suite de n octets, définie par le mot clé $Array$ sous la forme : $Array[n]$.

Exemple

Var

x : byte,

y : Array[20]

z : byte

On suppose que les variables des programmes micro-assembleur sont stockées dans un bloc, appelé **segment de données**. Chaque élément du segment de données contient un octet. Par la suite, vous manipulerez des octets représentant des entiers compris dans l'intervalle [−128,127].

3. Le Segment de données

- On suppose que le segment de données (voir Figure 2) contient 700 octets. Les octets sont numérotés de manière séquentielle, le premier octet a le numéro 0, le second 1, …

		…….		

Octet 0 octet 699

Figure 2 : Segment de données

- Les variables sont stockées de manière contiguë dans le segment de données, à partir de l'octet 0, dans l'ordre dans lequel elles apparaissent dans le bloc Var.

- La première variable de la partie déclaration d'un programme micro-assembleur a l'adresse 0. L'adresse d'une variable, autre que la première, est égale à l'adresse de la variable qui la précède dans le segment de données, augmentée du nombre d'octets occupés par cette variable précédente.

Variable	Adresse
x	0
y	1
z	21

4. Le Segment de pile

On suppose que le segment de pile contient 500 octets. Les octets sont numérotés de manière séquentielle, le premier octet a le numéro 0, le second 1, …

5. Flags et compteur ordinal

Avant de présenter la partie instructions d'un programme micro-assembleur, nous introduisons :

- Les $flags\ \ d'états$ qui permettent de gérer l'exécution des programmes. Un flag d'état fournit des informations sur la dernière opération arithmétique ou logique effectuée. Dans ce travail, nous nous intéressons à un ensemble très restreint de flags (ZF, SF et OF). Nous allons les simuler à l'aide de variables :
  - ZF, égal à 1 si la dernière opération arithmétique ou logique effectuée a donné lieu un résultat nul sinon ZF est égal à 0.
  - SF égal à 1 si la dernière opération arithmétique ou logique effectuée a donné lieu un résultat supérieur ou égal à 0 sinon SF est égal à 0.
  - OF est égal à 1 si la dernière opération arithmétique ou logique effectuée a débordé sinon 0. On parle de débordement si le résultat n'est pas dans l'intervalle [-127,+128].

  On suppose que les flags sont initialement nuls.

- Le compteur ordinal qui contient le numéro de la prochaine instruction à exécuter. Le compteur ordinal est simulé par une variable CO dont la valeur initiale est égale à 0.

- Les registres AX, BX, CX et DX considérés comme des variables globales à accès rapide.

Les variables simulant les flags et le compteur ordinal ne sont pas stockés dans le segment de données, exclusivement réservé pour les variables définies dans la partie Var. Les instructions sont numérotées de manière séquentielle.

6. Opérandes des Instructions

Les opérandes des instructions décrites ci-dessous sont soit des constantes entières signées, soit des variables. Une variable est spécifiée par son nom. En ce qui concerne les variables de type tableau, l'opérande est sous la forme nomArray[indice], où *indice* est une expression arithmétique. Si l'adresse nomArray augmentée par indice est supérieure à 699, alors **l'exécution du programme est arrêtée.**

Exemples :

+83, -12, x, z, y[0],y[i+3],y[2*i] sont des exemples d'opérandes.

7. Instructions

Le tableau suivant donne la liste des instructions du langage micro-assembleur.

Instruction	Description	Opérandes
mov op1,op2	op1← op2	op1 ne peut pas être une constante
add op1,op2	op1← op1+op2	op1 ne peut pas être une constante
sub op1,op2	op1← op1-op2	op1 ne peut pas être une constante
mult op1,op2	op1← op1*op2	op1 ne peut pas être une constante
div op1,op2	op1← op1/op2	op1 ne peut pas être une constante
and op1, op2	op1← op1 and op2	op1 ne peut pas être une constante
or op1,op2	op1← op1 or op2	op1 ne peut pas être une constante
not op	op1← not op1	op1 ne peut pas être une constante
jmp i	se brancher à l'instruction dont le numéro i.	
jz i	se brancher à l'instruction dont le numéro est i si ZF=1	
js i	se brancher à l'instruction dont le numéro est i si SF=1	
jo i	se brancher à l'instruction dont le numéro est i si OF=1	
input(op)	lire à partir de la console un entier signé et le sauvegarder dans op	
print(op)	afficher op	
halt	arrêter le programme	
# chaines de caractères	Un commentaire	
push op	Empile op dans le segment de pile	
pop op	dépile un élément de le segment de pile et l'affecte à op	
isFull	vérifie si le segment de pile est vide	
call programme	Lancer l'exécution de programme. Quand celui-ci est fini, le dernier programme interrompu est repris.	

8. Application

Le code suivant donne l'exemple d'un programme micro-assembleur. Ce programme lit 10 entiers et affiche leur somme.

```
#déclaration des variables
Var
i:byte
tab :Array[10]
size :byte
sum :byte
#partie instruction
Instruction
0: mov size,10
1: mov i,0
2: mov sum,0
# remplissage du tableau
3: sub size,i
4:jz 9
5: input(tab[i])
6:  add i,1
7: add sum, tab[i]
8:jmp 3
9:print(sum)
```

Questions

1. Donner une grammaire pour le langage micro-assembleur.
2. Donner une implémentation d'un compilateur pour un programme micro-assembleur. Le compilateur est écrit en C ou en Python (selon votre groupe). Le code exécutable généré par le compilateur est écrit en C.

3. Donnez un pdf de toutes les fonctions implémentées.
4. Prévoir deux modes d'exécution : exécution normale et une exécution pas à pas. Pour ce dernier mode d'exécution, le segment de données, le segment de pile, les registres, les flags et le compteur ordinal sont affichés. Prévoyez un affichage paginé.
5. Enrichir votre implémentation pour détecter la première erreur d'un programme micro-assembleur et faire une proposition de correction.
6. Ecrire un programme qui inverse un tableau. Par exemple,
   Z :Array[4]
   Supposons que z[0]=1, z[1]=2, z[2]=3 et z[3]=4. Après inversion, on aura

   que z[0]=4, z[1]=3, z[2]=2 et z[3]=1.

Rendu

1. Un pdf avec la grammaire et des détails de votre conception et implémentation.
2. Des captures d'images des codes des fonctions implémentées (uniquement celles-ci). Prenez un jeu d'essai complet. Prévoyez un programme avec une cascade d'appels de programmes (au moins trois). Les programmes partagent des données.
3. Est-t-il possible d'écrire avec votre langage un programme récursif. Si oui, donner un code récursif et faites une exécution pas à pas.
4. Votre code final.

Surtout ne demandez pas à une IA de coder à votre place. Nous disposons d'outils pour détecter des codes fournis par les IAs. En cas de triche, vous aurez un zéro.