

Architecture des ordinateurs

Fiche de TP 4

ESIPE - IR/IG 1 2018-2019

Crible d'Ératosthène

Table des matières

1	Algorithme	2
2	Implantation	2
3	Annexe	4

Cette fiche est à faire en une séance (soit 2 h, sans compter le temps de travail personnel), et en binôme. Il faudra

1. réaliser un rapport soigné à rendre **au format pdf** contenant les réponses aux questions de cette fiche, une introduction et une conclusion ;
2. écrire les — différents — fichiers sources des programmes demandés. Veiller à nommer correctement les fichiers sources. Ceux-ci doivent **impérativement** être des fichiers compilables par Nasm ;
3. réaliser une archive **au format zip** contenant les fichiers des programmes et le rapport. Le nom de l'archive doit être sous la forme AO-TP4_NOM1_NOM2.zip où NOM_1 et NOM_2 sont respectivement les noms de famille des deux binômes dans l'ordre alphabétique ;
4. déposer l'archive sur la plate-forme de rendu.

Tous les fichiers complémentaires nécessaires à ce TP se trouvent sur le site

<http://igm.univ-mlv.fr/~giraudo/Enseignements/2018-2019/AO/AO.html>

Le but de cette séance est de réaliser un programme assembleur implantant le crible d'Ératosthène. Ce TP demande l'application de toutes les techniques et éléments vus au cours des séances précédentes.

1 Algorithme

L'algorithme du *crible d'Ératosthène* permet de calculer tous les nombres premiers inférieurs à un nombre n fixé. Il procède ainsi :

1. on commence par générer une liste d'entiers $L := [2, 3, 4, \dots, n]$. Chaque entier de cette liste possède un état : il peut être *marqué* ou bien *non marqué*. Initialement, tous les entiers de L sont non marqués ;
2. ensuite, tant qu'il existe des nombres non marqués dans la liste L :
 - (a) on marque le plus petit nombre k de L non encore marqué ;
 - (b) pour tout nombre $\ell > k$ de la liste L :
 - si k est un diviseur de ℓ , alors on supprime l'élément ℓ de la liste ;
3. la liste L — qui contient maintenant exclusivement des nombres marqués — est la liste des nombres premiers inférieurs à n .

Par exemple, si on applique cet algorithme pour $n := 30$, on obtient successivement les valeurs de L suivantes (les nombres marqués sont soulignés) :

```
[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30],
[2,3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29 ],
[2,3, 5, 7, 11, 13, 17, 19, 23, 25, 29 ],
[2,3, 5, 7, 11, 13, 17, 19, 23, 29 ],
[2,3, 5, 7, 11, 13, 17, 19, 23, 29 ],
[2,3, 5, 7, 11, 13, 17, 19, 23, 29 ],
[2,3, 5, 7, 11, 13, 17, 19, 23, 29 ],
[2,3, 5, 7, 11, 13, 17, 19, 23, 29 ],
[2,3, 5, 7, 11, 13, 17, 19, 23, 29 ],
[2,3, 5, 7, 11, 13, 17, 19, 23, 29 ],
[2,3, 5, 7, 11, 13, 17, 19, 23, 29 ].
```

2 Implantation

On peut se servir de la bibliothèque `asm_io.inc` pour gérer les entrées/sorties. Le programme doit définir une constante `N` de valeur 128. Ceci est réalisé par l'instruction :

```
%define N 128
```

Le programme doit également réserver `N-1` octets à partir d'une adresse `tab` et un octet à l'adresse `pgm`. Ceci est réalisé par les déclarations

```
tab : resb N-1
pgm : resb 1
```

à disposer dans la section `.bss`.

La suite de N octets `tab` peut être vue comme un tableau. Celui-ci est utilisé pour mémoriser les nombres effacés pendant l'exécution de l'algorithme. Pour cela, l'octet à l'adresse

$$\text{tab} + i - 2 \quad (1)$$

vaut 1 si le nombre i n'est pas effacé et 0 s'il est effacé. Il s'agit pour débiter de bien comprendre l'adresse spécifiée par la formule (1). En effet, comme la liste des entiers à considérer commence par 2 (car aucun nombre strictement inférieur à 2 n'est premier), l'octet à l'adresse $\text{tab} + 2 - 2 = \text{tab}$ est dédié à coder la présence/absence de 2. En poursuivant le raisonnement, c'est bien l'octet à l'adresse $\text{tab} + 3 - 2 = \text{tab} + 1$ qui est dédié à coder la présence/absence de 3. Et ainsi de suite, ce qui explique la formule générale (1).

Par ailleurs, en plus de tout cela, la variable à l'adresse `pgm` contient la valeur du **plus grand nombre marqué** (c'est-à-dire donc le dernier nombre marqué).

- Important.**
1. Toutes les fonctions implantées par la suite doivent respecter la convention d'appel du C.
 2. Chaque fonction implantée doit être documentée en mentionnant son rôle, ses arguments attendus et son renvoi.
 3. On utilisera le fichier `Squelette.asm` qui contient un squelette du code à fournir. Il suffit de le remplir ses parties manquantes conformément aux questions suivantes.

Exercice 1. 1. *Écrire une fonction effacer qui prend en argument un nombre i et positionne la case correspondante du tableau `tab` à 0, signifiant que le nombre i est effacé.*

Note : il se peut que la réponse à cette question figure déjà dans le fichier `Squelette.asm`. Il est important de bien la comprendre avant d'aller plus loin.

2. *Écrire une fonction ajouter qui prend en argument un nombre i et positionne la case correspondante du tableau `tab` à 1, signifiant que le nombre i n'est pas effacé.*
3. *Écrire une fonction lire qui prend en argument un nombre i et renvoie dans `eax` la valeur de la case correspondante dans le tableau `tab`.*

Dorénavant, on utilisera **uniquement** les trois fonctions précédentes pour accéder au tableau `tab`, que ce soit en lecture ou en écriture.

4. *Écrire une fonction initialiser qui positionne tous les octets du tableau `tab` à 1. Cette fonction doit aussi positionner la variable `pgm` à la valeur 0 pour spécifier qu'aucun nombre du tableau n'est marqué.*
5. *Écrire une fonction effacer_multiples qui prend un argument p et qui positionne à 0 les cases du tableau `tab` correspondant aux entiers qui sont multiples de p .*
6. *Écrire une fonction premier_suivant qui prend un argument i et renvoie dans `eax` la valeur du premier nombre supérieur à i tel que l'octet correspondant dans le tableau `tab` vaut 1. Si une telle valeur n'existe pas, la fonction renvoie $N + 1$.*

7. Écrire une fonction *afficher* qui affiche dans l'ordre croissant les nombres qui sont codés comme présents par le tableau *tab*.
8. Rassembler et utiliser les fonctions écrites en un programme complet qui implante l'algorithme d'Ératosthène. Le programme, lorsqu'il est exécuté, doit afficher les nombres premiers inférieurs à N .
9. Améliorer le programme afin qu'il sorte de la boucle principale (instruction 2.) lorsque le plus grand nombre marqué dépasse \sqrt{N} .
Note : Il n'est pas nécessaire de savoir calculer la racine carrée de N .
10. (Facultatif) Améliorer le programme pour qu'il utilise de manière plus efficace la mémoire. On utilise le fait qu'un octet permet de conserver l'état — effacé ou non effacé — de huit nombres et non plus d'un seul. Si le programme est déjà bien écrit, il ne suffit de modifier que les trois premières fonctions. Le nouveau programme doit s'appeler *F1.asm*.
11. (Facultatif) Améliorer le programme pour faire en sorte qu'il fonctionne pour des valeurs de N non fixées. L'utilisateur du programme saisit sur l'entrée standard une valeur pour N et le programme doit afficher tous les nombres premiers inférieurs à N . Le nouveau programme doit s'appeler *F2.asm*.

3 Annexe

La table 1, qui recense les instructions les plus courantes en assembleur, pourra être utile.

Instruction	Opérande 1	Opérande 2	Effet
<code>mov</code>	<i>dst</i>	<i>src</i>	recopie <i>src</i> dans <i>dst</i>
<code>xchg</code>	<i>ds1</i>	<i>ds2</i>	échange <i>ds1</i> et <i>ds2</i>
<code>push</code>	<i>src</i>		place <i>src</i> au sommet de la pile
<code>pop</code>	<i>dst</i>		supprime le sommet de la pile et le place dans <i>dst</i>
<code>add</code>	<i>dst</i>	<i>src</i>	ajoute <i>src</i> à <i>dst</i>
<code>adc</code>	<i>dst</i>	<i>src</i>	ajoute <i>src</i> à <i>dst</i> avec retenue
<code>sub</code>	<i>dst</i>	<i>src</i>	soustrait <i>src</i> à <i>dst</i>
<code>sbb</code>	<i>dst</i>	<i>src</i>	soustrait <i>src</i> à <i>dst</i> avec retenue
<code>mul</code>	<i>src</i>		multiplie <i>eax</i> par <i>src</i> (résultat dans <i>edx eax</i>)
<code>imul</code>	<i>src</i>		multiplie <i>eax</i> par <i>src</i> (complément à deux)
<code>div</code>	<i>src</i>		divise <i>edx eax</i> par <i>src</i> (<i>eax</i> =quotient, <i>edx</i> =reste)
<code>idiv</code>	<i>src</i>		divise <i>edx eax</i> par <i>src</i> (complément à deux)
<code>inc</code>	<i>dst</i>		place <i>dst</i> + 1 dans <i>dst</i>
<code>dec</code>	<i>dst</i>		place <i>dst</i> – 1 dans <i>dst</i>
<code>neg</code>	<i>dst</i>		place – <i>dst</i> dans <i>dst</i>
<code>not</code>	<i>dst</i>		place (not <i>dst</i>) dans <i>dst</i>
<code>and</code>	<i>dst</i>	<i>src</i>	place (<i>src</i> AND <i>dst</i>) dans <i>dst</i>
<code>or</code>	<i>dst</i>	<i>src</i>	place (<i>src</i> OR <i>dst</i>) dans <i>dst</i>
<code>xor</code>	<i>dst</i>	<i>src</i>	place (<i>src</i> XOR <i>dst</i>) dans <i>dst</i>
<code>sal</code>	<i>dst</i>	<i>nb</i>	décalage arithmétique à gauche de <i>nb</i> bits de <i>dst</i>
<code>sar</code>	<i>dst</i>	<i>nb</i>	décalage arithmétique à droite de <i>nb</i> bits de <i>dst</i>
<code>shl</code>	<i>dst</i>	<i>nb</i>	décalage logique à gauche de <i>nb</i> bits de <i>dst</i>
<code>shr</code>	<i>dst</i>	<i>nb</i>	décalage logique à droite de <i>nb</i> bits de <i>dst</i>
<code>rol</code>	<i>dst</i>	<i>nb</i>	rotation à gauche de <i>nb</i> bits de <i>dst</i>
<code>ror</code>	<i>dst</i>	<i>nb</i>	rotation à droite de <i>nb</i> bits de <i>dst</i>
<code>rcl</code>	<i>dst</i>	<i>nb</i>	rotation à gauche de <i>nb</i> bits de <i>dst</i> avec retenue
<code>rcr</code>	<i>dst</i>	<i>nb</i>	rotation à droite de <i>nb</i> bits de <i>dst</i> avec retenue
<code>cmp</code>	<i>sr1</i>	<i>sr2</i>	compare <i>sr1</i> et <i>sr2</i>
<code>jmp</code>	<i>adr</i>		saute vers l'adresse <i>adr</i>
<code>jxx</code>	<i>adr</i>		saute conditionné par <i>xx</i> vers l'adresse <i>adr</i>
<code>loop</code>	<i>adr</i>		répétition de la boucle <i>nb</i> de fois (<i>nb</i> dans <i>ecx</i>)
<code>loopx</code>	<i>adr</i>		répétition de la boucle conditionnée par <i>x</i>

TABLE 1 – Les principales instructions en assembleur.