Architecture des ordinateurs

ESIPE - IR1 2015-2016

Fiche de TP 4

Crible d'Ératosthène

Table des matières

Le but de cette séance est de réaliser un programme assembleur implantant le crible d'Ératosthène.

Cette fiche est à faire en une séance (soit 2 h), et en binôme. Il faudra :

1. réaliser un — bref — rapport à rendre **au format pdf** contenant les réponses aux questions de cette fiche ;
2. écrire les — différents — fichiers sources des programmes demandés. Veillez à nommer correctement vos fichiers sources. Les sources des programmes doivent **impérativement** être des fichiers compilables par Nasm ;
3. réaliser une archive **au format zip** contenant les sources des programmes et le rapport. Le nom de l'archive doit être sous la forme `IR1_Archi_TP1_NOM1_NOM2.zip` ;
4. déposer l'archive sur la plate-forme de rendu.

Tous les fichiers complémentaires nécessaires à ce TP se trouvent sur la plate-forme.

1 Algorithme

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

1. on commence par générer une liste d'entiers non soulignés $L := [2, 3, 4, \ldots, n]$;

2. ensuite, tant qu'il existe des nombres non soulignés dans la liste L :
   (a) on souligne le plus petit nombre k de L non encore souligné ;
   (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 soulignés — 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 :

[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 :

```
1 %define N 128
```

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

```
1 tab resb N-1
2 pgs resb 1
```

Cette suite de `N` octets peut être vue comme un tableau. Celui-ci est utilisé pour mémoriser les nombres effacés pendant l'exécution de l'algorithme. L'octet à l'adresse `tab + i - 2` vaut `1` si le nombre `i` n'est pas effacé et `0` s'il est effacé. La variable `pgs` contient la valeur du **p**lus **g**rand nombre **s**ouligné (c'est à dire le dernier nombre souligné).

Important 1. Toutes les fonctions implantées par la suite doivent respecter la convention d'appel du C.

Important 2. Chaque fonction implantée doit être documentée en mentionnant son rôle, ses arguments attendus et son renvoi.

Astuce 1. Le fichier `Squelette.asm` contient un squelette du code a fournir. Il est possible de l'utiliser en remplissant les parties manquantes.

Question 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.

Question 2. Écrire une fonction `marquer` 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.

Question 3. Écrire une fonction `lire` qui prend en argument un nombre `i` et retourne dans `eax` la valeur de la case correspondante dans le tableau `tab`.

Important 3. Dorénavant, on utilisera *uniquement* les trois fonctions précédentes pour accéder au tableau `tab`.

Question 4. Écrire une fonction `initialiser` qui positionne tous les octets du tableau `tab` à `1`. Cette fonction doit aussi positionner la variable `pgs` à la valeur 0 pour spécifier qu'aucun nombre du tableau n'est souligné.

Question 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`.

Question 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`.

Question 7. Écrire une fonction `afficher` qui affiche dans l'ordre croissant les nombres dont la valeur dans le tableau `tab` est `1`.

Question 8. Écrire un programme `Q8.asm` qui implante l'algorithme d'Ératosthène et utilisant les fonctions précédentes. Le programme doit afficher les nombres premiers inférieurs à `N`.

Question 9. Modifier le programme afin qu'il sorte de la boucle principale (instruction 2.) lorsque le plus grand nombre souligné dépasse $\sqrt{N}$. Le nouveau programme doit s'appeler `Q9.asm`.

Astuce 2. Il n'est pas nécessaire de savoir calculer la racine carrée de N.

Bonus 1. 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 `B1.asm`.

Bonus 2. 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 `B2.asm`.

3 Annexe

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

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

TABLE 1 – Les principales instructions en assembleur.