L3 informatique - architecture des ordinateurs et des systèmes

Année universitaire : 2019-2020

Durée : 2h

Consignes : interactions interdites avec les autres étudiants

Examen terminal de l'UE « architecture des ordinateurs et des systèmes » Deuxième session

En raison de la crise sanitaire COVID-19, l'examen est à réaliser à distance, en temps limité.

- Vous devez déposer un document unique sur l'ENT à la fin du temps imparti. Ce document devra comporter les réponses aux deux parties de cet examen.
- Si vous avez des difficultés numériques pendant l'épreuve, vous devez écrire un email à alexandre.guitton@uca.fr en décrivant le problème rencontré (si vous avez accès à vos emails), ou contacter la scolarité par téléphone au 04 73 40 70 05. Une absence de contact sera considérée comme une absence de problème.
- Si vous n'arrivez pas à soumettre votre document sur l'ENT, vous pouvez aussi l'envoyer par email à alexandre.guitton@uca.fr (vous recevrez une confirmation quelques heures après l'envoi).

Le sujet est individualisé afin de limiter les interactions possibles entre étudiants. Vous devez répondre au sujet qui vous a été affecté (il porte votre nom). Vérifiez bien que le nom du fichier d'examen est à votre nom : une erreur de sujet entraînera une note de 0.

L'examen est en deux parties :

- Une partie sur l'architecture des ordinateurs (d'une durée d'environ 1h)
- Une partie sur l'architecture des systèmes (d'une durée d'environ 1h)

Bon courage !

Partie « Architecture des ordinateurs » - durée : environ 1h

Exercice 1 : Traduction d'un programme C en assembleur (10 points)

Question 1 (10 points) : La fonction ci-dessous détermine si une chaîne full contient une sous-chaîne non consécutive nommée part. Elle renvoie 1 si c'est le cas, et 0 sinon.

- Par exemple, la fonction retourne 1 si full vaut « absolument » et si part vaut « bon » (en effet, la chaîne « absolument » contient « bon » en ignorant certains caractères).
- Par exemple, la fonction retourne 0 si full vaut « banano » et si part vaut « bon » (tous les caractères de « bon » sont présents, mais pas dans le bon ordre).

```
int nonConsecutive(char * full, char * part) {
   int i = 0, j = 0;
   while ((part[j]!='\0') && (full[i]!='\0')) {
      if (full[i]==part[j]) {
         j++;
      }
      i++;
   }
   if (part[j]=='\0') {
      return 1;
   }
   else {
      return 0;
   }
}
```

Traduisez cette fonction en un programme assembleur. Les deux chaînes full et part peuvent être définies comme des variables initialisées dans la section « .data ».

Exercice 2 : Fautes dans un programme assembleur (10 points)

Question 2 (5 points) : Le programme ci-dessous calcule dans EAX (sans l'afficher) la longueur de la plus longue sous-chaîne (consécutive) extraite d'une chaîne principale, dont les caractères sont tous en minuscule.

- Par exemple, la longueur de la plus longue sous-chaîne de caractères minuscules de la chaîne « Je suis Dominique » est 8 (la sous-chaîne en question étant « ominique »).

```
section .data
   chaine db 'Je suis Dominique'
   length equ $-chaine
section .text
global _start
_start:
   mov eax, 0
   mov ebx, 0
   mov edi, chaine
   boucle:
      mov cl, [edi]
      cmp ecx, 0
      je fin
      cmp cl, 'a'
      cmp cl, 'z'
         inc ebx
         cmp ebx, eax
         jbe suite
            jmp suite
      non_min:
      mov ebx, 0
      suite:
      jmp boucle
fin:
   mov eax, 1
   xor ebx, ebx
   int 80h
```

Ce programme contient cinq erreurs (une erreur par ligne ; une ligne oubliée compte pour une erreur). Corrigez-les.

Question 3 (5 points) : Le programme ci-dessous détermine si un entier n est un carré parfait ou non, et stocke le résultat (1 si c'est un carré parfait, 0 sinon) dans EAX, sans l'afficher, juste avant de quitter.

- Par exemple, 144 est un carré parfait (car 144=12*12).
- Pour rappel, l'instruction MUL x sur 32 bits multiplie EAX par x, et stocke le résultat dans le registre étendu EDX:EAX (les 32 bits de poids fort du résultat sont dans EDX, les 32 bits de poids faible du résultat sont dans EAX).

```
section .data
   n equ 144
section .text
global _start
_start:
   mov ebx, 2
   boucle:
      mov eax, ebx
      cmp eax, n
      jne fin_ok
      ja fin_non_ok
      inc ebx
      add eax, 5
      mul eax
      jmp boucle
   fin_ok:
      mov eax, 1
   fin_non_ok:
      mov eax, 0
fin:
   mov eax, 1
   xor ebx, ebx
   int 80h
```

Ce programme contient cinq erreurs (une erreur par ligne ; une ligne oubliée compte pour une erreur). Corrigez-les.