

**ESIR SYS1 CC2 2013–2014**  
**11 juin 2014**

Nom et Prénom numéro d'étudiant :
--------------------------------------

Durée : 1h, Notation : sur 20 points

---

**1 Première partie : QCM (10 points)****Instructions :**

**Cochez** clairement la case de la réponse que vous pensez être juste. Il y a **une seule réponse** juste par question.

**Barème :**

+0.5 pour chaque réponse correcte

-0.5/m pour chaque réponse fausse (où m+1 est le nombre de réponses possibles)

**Question 1** Dans un ordinateur moderne, si deux processus modifient le contenu d'une même adresse (par exemple 0x00400000), s'agit-il en général :

- de deux 'cases' mémoire physiques distinctes (les modifications d'un processus ne sont pas visibles par l'autre processus).
- de la même 'case' mémoire physique (les modifications d'un processus sont visibles par l'autre processus).

**Question 2** Que fait la ligne de commande shell suivante ?

```
aaa >> bbb
```

- Cette ligne ajoute la sortie standard de la commande `aaa` à la fin du fichier `bbb`.
- Cette ligne redirige la sortie standard de la commande `aaa` vers l'entrée standard de la commande `bbb`.

**Question 3** Si le répertoire de travail est `/Users/ftaiani/`, quel répertoire représente `../ftaiani/Desktop/../../?`

- `/Users/ftaiani/Desktop/`
- `/Users/ftaiani/`
- `/Users/`

**Question 4** Par défaut, le passage de paramètres de types de base (`int`, `char`, `long`, ...) en C s'opère :

- par référence.
- par valeur.

**Question 5** Soit la ligne de commandes suivante :

```
a=Salut ; echo '$a'
```

Qu'imprime-t-elle sur la console ?

- `$a`
- `Salut`
- `'Salut'`

**Question 6** Soit le code suivant :

```
#include <stdio.h>
#include <stdlib.h>

void foo(int* input, int size) {
    int i ;
    int* array_int ;
    array_int = (int*)input ;
    array_int[size-1]++;
}

int main(int argc, char** argv) {
    int* table = malloc(2*sizeof(int));
    table[0]=1;
    table[1]=2;
    foo(table,2);
    printf("%i %i\n", table[0], table[1]);
}
```

Dans quelle zone mémoire la variable `array_int` est-elle allouée ?

- dans la zone des variables globales non initialisées (BSS)
- dans la pile
- dans le tas

**Question 7** Que fait un éditeur de lien ?

- Il implante le programme en mémoire et lance son exécution.
- Il forme un fichier exécutable à partir des différents modules objets en résolvant les références entre modules.
- Il permet aux développeurs de modifier de façon ergonomique les liens binaires d'un processus en exécution.

**Question 8** Comment désalloue-t-on explicitement de la mémoire sur le tas en Java ?

- avec la méthode `free()`
- avec l'opérateur `delete`
- On ne peut pas désallouer explicitement la mémoire du tas en Java.

**Question 9** Sous unix, comment rechercher la chaîne "java" dans les fichiers ayant l'extension ".txt" ?

- `grep java *.txt`
- `ls *.java txt`
- `find *.txt -name java`

**Question 10** Où sont allouées les variables automatiques ?

- sur la pile
- sur le tas
- sur le segment BSS

CORRECTION

**Question 11** Si le module M1 exporte la variable A, et utilise la variable B définie dans le module M2 :

- B se trouve dans la table des externes de M1.
- A se trouve dans la table des externes de M1.
- B se trouve dans la table des points d'entrée de M1.
- A se trouve dans la table des points d'entrée de M2.

**Question 12** Dans un script shell, avec quelle variable peut-on obtenir le nombre de paramètres passés sur la ligne de commande ?

- \$#
- \$\*
- \$%

**Question 13** Soit le code suivant

```
#include <stdio.h>

void foo(int* i) {
    *i = 10;
}

int main( int argc, char** argv) {
    int i = 5;
    foo(&i);
    printf("%i\n",i);
}
```

Quelle valeur ce code imprime-t-il sur la console ?

- 10
- le contenu de la mémoire à l'adresse 10
- 5

**Question 14** Dans un processeur moderne :

- le cache L2 est plus rapide que le cache L1.
- le cache L1 est plus rapide que le cache L2.

**Question 15** Quelle combinaison de touches permet-elle de mettre en pause un processus Unix s'exécutant dans un terminal ?

- contrôle C
- contrôle P
- contrôle Z

**Question 16** Que fait la ligne de commande suivante  
`xgalaga &`

- Elle lance le jeu `xgalaga` en mode protégé.
- Elle lance le jeu `xgalaga` en mode pause.
- Elle lance le jeu `xgalaga` en arrière plan.

**Question 17** Soit le code suivant

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void foo(char** c) {
    char a_string[] = "SYS1 c'est top!";
    *c = malloc(strlen(a_string)+1);
    strcpy(*c,a_string);
}

int main(int argc, char** argv) {
    char* c;
    printf("%s\n",c);
}
```

Dans quel région mémoire la chaîne contenue dans `a_string[]` est-elle stockée ?

- sur la pile  
 dans le segment de texte  
 dans le segment BSS  
 sur le tas

**Question 18** Le contenu de la zone mémoire des variables non initialisées se trouve-t-il dans un fichier objet (\*.o sous unix) ?

- non  
 oui

**Question 19** Combien de commandes unix la ligne suivante combine-t-elle ?

```
sed 's/[;".,: !?-\]/\n/g' pg84.txt | sort -f | uniq -c | sort -nrk1 > xyk
```

- 9  
 5  
 4  
 10

**Question 20** Dans l'exemple assembleur suivant

```
.DATA
X DB 10 DUP(?)
Y DB 7 DUP(?)
W EQU 15
N1 EQU W
Z EQU Y
```

La valeur associée à Z est-elle ?

- absolue  
 translatable

## 2 Seconde partie : Questions ouvertes & problèmes (10 pts)

**Question 21** On considère le programme C suivant.

```
#include <stdio.h>
#include <stdlib.h>

void foo(void) {
    printf("Still running\n");
    foo();
}

int main(int argc, char** argv) {
    foo();
}
```

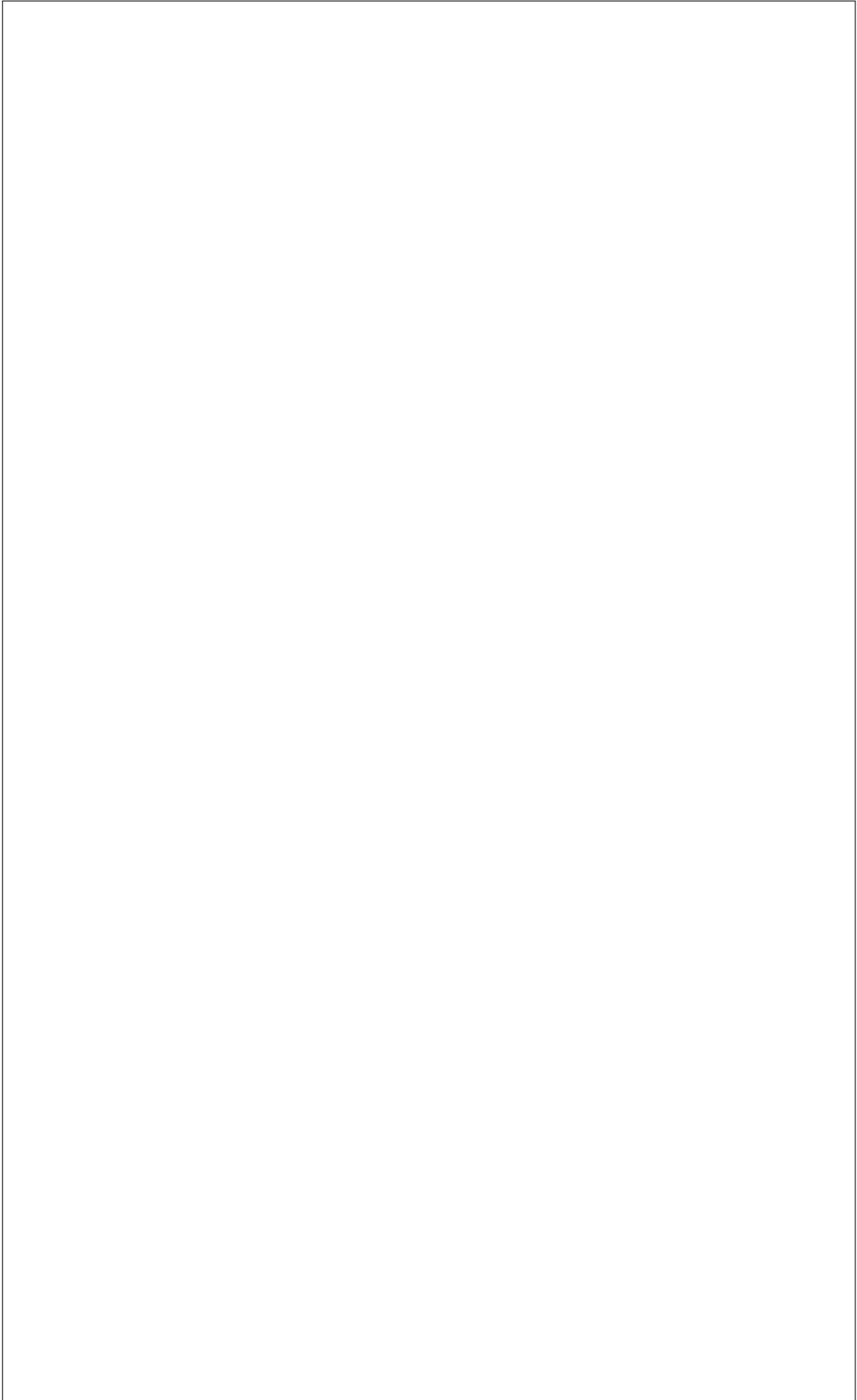
Après compilation (`gcc example_pile_v3.c`), ce programme produit l'erreur suivante à l'exécution :

```
...
Still running
Still running
Still running
Still running
zsh: segmentation fault a.out
```

Expliquez précisément pourquoi ce message d'erreur est produit. [1.5 points]

A  B  C

CORRECTION



CORRECTION

**Question 22** On active maintenant le second niveau d'optimisation du compilateur gcc (gcc -O2 example\_pile\_v3.c). Avec ce niveau d'optimisation, le programme de la question 21 ne produit plus l'erreur `segmentation fault`, et imprime indéfiniment la même chaîne "Still running". Pour comprendre pourquoi la version optimisée se comporte différemment, l'on décompile chacun des deux exécutable (en utilisant `objdump -S -Mintel -d a.out`). La version non-optimisée donne le code assembleur suivant pour la fonction `foo()` :

```
0000000004004e4 <foo>:
 4004e4: 55                push   rbp
 4004e5: 48 89 e5          mov    rbp,rsq
 4004e8: bf fc 05 40 00    mov    edi,0x4005fc
 4004ed: e8 ee fe ff ff    call  4003e0 <puts@plt>
 4004f2: e8 ed ff ff ff    call  4004e4 <foo>
 4004f7: c9                leave
 4004f8: c3                ret
```

La version optimisée (avec l'option -O2) fournit le code suivant :

```
0000000004004f0 <foo>:
 4004f0: 48 83 ec 08      sub    rsp,0x8
 4004f4: 0f 1f 40 00      nop   DWORD PTR [rax+0x0]
 4004f8: bf 0c 06 40 00    mov    edi,0x40060c
 4004fd: e8 de fe ff ff    call  4003e0 <puts@plt>
 400502: eb f4            jmp   4004f8 <foo+0x8>
 400504: 66 66 66 2e 0f 1f 84 nop   WORD PTR cs:[rax+rax*1+0x0]
 40050b: 00 00 00 00 00
```

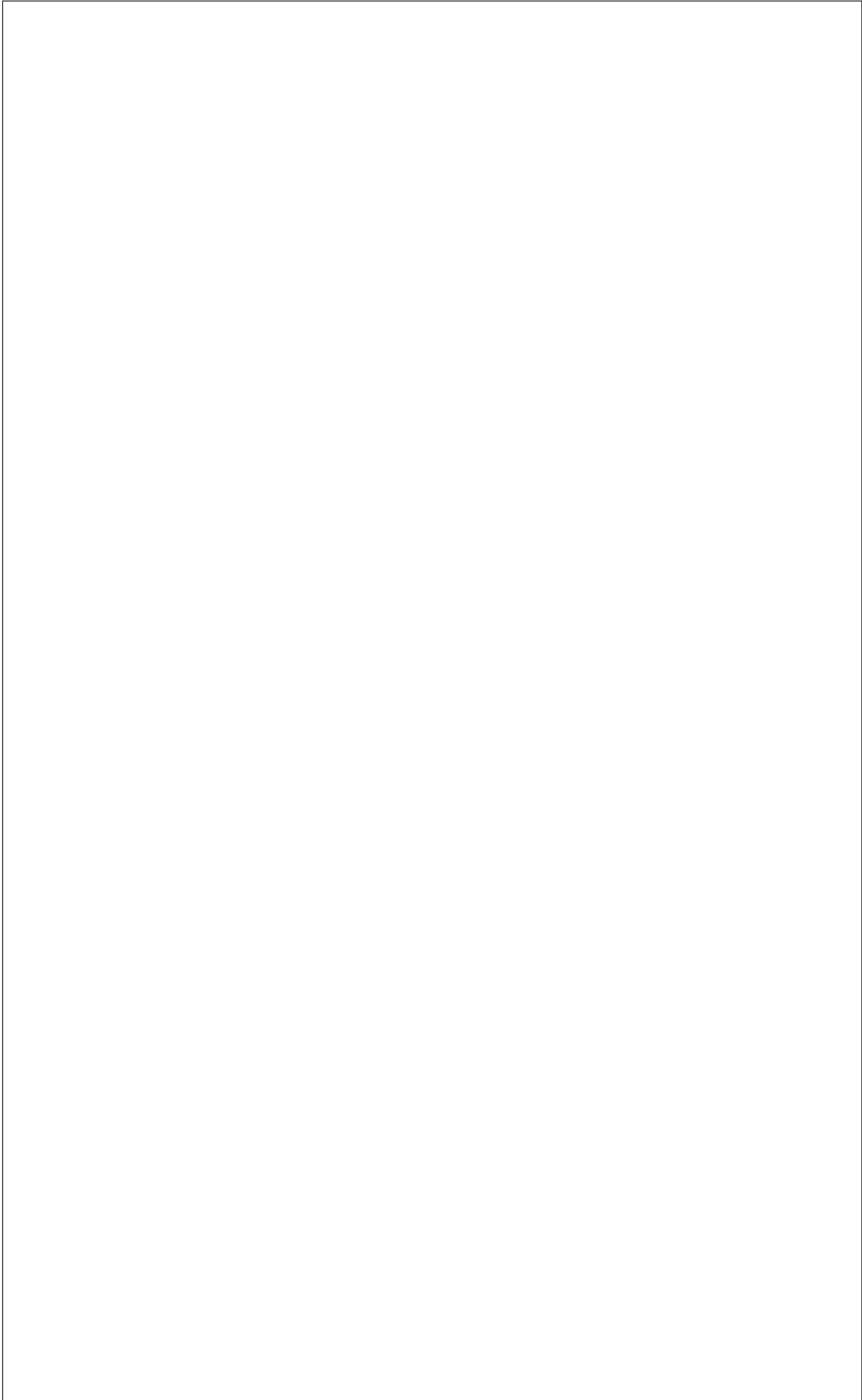
Dans ce code, les instructions `nop` n'ont pas d'effet ("no operation") et servent à aligner le code de façon à optimiser sa vitesse.

Expliquez quel a été l'effet de l'optimisation, et pourquoi l'erreur `segmentation fault` a disparu.

[1.5 points]

A  B  C  F

CORRECTION





## CORRECTION

Soit le module M1 écrit en langage d'assemblage :

```
; Module M1
.DATA
PUBLIC A
EXTRN C:WORD
EXTRN E:WORD

Y EQU 5
B EQU A
A DW 10
F DW Y

.CODE
DEB:
MOV AX,Y
MOV SI,C
MOV AX,E[SI]
MOV BX,A
MOV [BX],AX
END DEB
```

Dans vos réponses, toutes les valeurs et adresses seront données en base 10.

**Question 23** Donnez la table des identificateurs construite lors de l'assemblage de M1.  
(On utilisera quatre colonnes : **identificateur**, **valeur**, **type**, **translation**.) [1.5 points]

A  B  C  F

--

CORRECTION

**Question 24** Donnez le tableaux des externes de M1, ainsi que le résultat de l'assemblage de M1. Pour simplifier, on présentera le résultat de l'assemblage sous forme d'un tableau à trois colonnes pour le segment de données (**adresse, valeur, vecteur de translation**), et six colonnes pour le code (**adresse, code symbolique, registre, mode d'adressage, opérande, vecteur de translation**). On supposera que chaque instruction occupe 4 octets. On laissera le code symbolique des instructions et le nom symbolique des registres ; le mode d'adressage sera noté **im** (immédiat), **dir** (direct) **ind** (indirect), **ix** (indexé) etc ...

[1.5 points]

A  B  C  F

--

CORRECTION

**Question 25** Soit la fonction C suivante :

```
void foo(int i, int*j, char c) {  
    int tabl[3] = {1,2,3};  
    // does something  
}
```

On suppose dans cette question : (1) que tous les paramètres sont passés par la pile (contrairement à l'ABI x86-64) ; (2) que le compilateur n'opère aucun alignement ; et que (3) le compilateur n'opère aucune optimisation. Représentez le schéma de pile de la fonction `foo()` lorsqu'elle est appelée.

[1.5 points]

A  B  C  F

CORRECTION

**Question 26** Expliquez en détail ce que fait la combinaison de commandes unix suivante. Vous prendrez soin de bien décomposer cette ligne en chacun de ses éléments.

```
ls `pwd` | grep sig
```

[1.5 points]

A  B  C  F

CORRECTION

**Question 27** Le bug Heartbleed de la librairie OpenSSL permet à un attaquant d'obtenir une partie des données contenues dans le tas d'un serveur OpenSSL vulnérable. Expliquez en quelques mots pourquoi cela est problématique.

[1 points]

A  B  F

## CORRECTION