ESIR SYS1 CC2 2018–2019
3 avril 2019

Nom et Prénom
| |
numéro d'étudiant : 99999

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 Combien de bits utilisent les processeurs x86 typiquement utilisés de nos jours dans les ordinateurs personnels :

- ☐ 8 bits
- ☐ 16 bits
- ☐ 32 bits
- ☐ 64 bits

Question 2 Comment désalloue-t-on explicitement de la mémoire sur le tas en Java ?

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

Question 3 À quoi la seconde ligne du code suivant est-elle équivalente ?

```
char* s = "Hi!";
*(s+1) = 1 ;
```

- ☐ s[1] = 1
- ☐ s = 0
- ☐ s = s + 1

Question 4 On considère l'extrait de code en C suivant :

```
int16_t i   = 10;
char* ptr_i = (char*)&i;
ptr_i[0]    = 2 ;
printf("%i\n", (int)i );
```

La ligne `printf("%i\n", (int)i );` permet d'imprimer un entier. Quelle sortie produit cet extrait de programme lorsqu'il s'exécute sur un processeur Intel de la famille x86 ?

- ☐ 2
- ☐ 10
- ☐ 256
- ☐ 522

Question 5 Soit le programme assembleur suivant :

```
MOV BX,100
MOV WORD PTR [BX],10 ;; WORD PTR indique que BX pointe vers un mot mémoire (2 octets)
MOV AL,[BX+1]
```

Si l'on suppose la mémoire initialisée à zéro au lancement de ce programme, que contient AL à la fin du programme ?

- ☐ 0
- ☐ 1
- ☐ 10
- ☐ 11
- ☐ 100
- ☐ 101

Question 6 Si deux programmes `A` et `B` sont compilés avec la même librairie dynamique `L`, le code de la librairie `L` se retrouve–t–il dupliqué dans chacun des fichiers exécutables de `A` et de `B` ?

- ☐ oui
- ☐ non

Question 7 Comment peut–on compiler le fichier `test.c` avec `gcc` sans réaliser l'édition de lien (*linking*) ?

- ☐ `gcc -o test.c`
- ☐ `gcc test.c`
- ☐ `gcc -c test.c`

Question 8 Quelle fonction C permet–elle de libérer de la mémoire allouée sur le tas (*heap*) ?

- ☐ delete
- ☐ free
- ☐ une telle fonction n'existe pas.

Question 9 Si `BX` contient l'adresse d'une variable stockée sur deux octets, comment affecte–t–on l'octet de poids faible de cette variable au registre `AH` ?

- ☐ `MOV AH,[BL]`
- ☐ `MOV AH,BL`
- ☐ `MOV AH,BX`
- ☐ `MOV AH,[BX]`

Question 10 En assembleur, quel est le registre par rapport auquel se font les accès aux paramètres passés par la pile et aux variables locales à une fonction ?

- ☐ `BX`
- ☐ `SX`
- ☐ `BP`
- ☐ `SP`

Question 11 On considère l'extrait de code en C suivant :

```
char x = 'H';
x--;
printf("%c\n", x );
```

La ligne `printf("%c\n", x );` permet d'imprimer un caractère. Quelle sortie produit cet extrait de programme ?

- ☐ Aucune sortie, le programme ne compile pas.
- ☐ H
- ☐ G

Question 12 Sur combien de bits fonctionne le registre `RAX` présent sur certaines architectures x86 ?

- ☐ 32
- ☐ 64
- ☐ 128

Question 13 Dans un programme utilisant l'ABI Linux x86-64 (System V), les paramètres en entrée d'une procédure sont principalement passés :

- ☐ sur la pile
- ☐ par des registres

Question 14 On considère le programme TASM suivant :

```
.MODEL SMALL      ; memory model
.STACK            ; memory space for program instructions in the stack
.DATA             ; data segment
        MSG DB 'MYSTERE',10,13,'$'
.CODE
start:
        MOV AX, @DATA             ; initialising
        MOV DS, AX                ; stack segment (needed on 16bit)

        MOV BX, offset MSG
        MOV SI, 1

boucl:  MOV byte ptr BX[SI], 'X'  ; ensures BX[SI] treated as a byte
        INC SI
        CMP SI,4
        JL boucl
        MOV byte ptr BX[SI+1],'$' ; ensures BX[SI+1] treated as a byte

        MOV DX, offset MSG        ; DX contains start of string to be printed
        MOV AH, 09H               ; 09H is the number of the DOS print routine
        INT 21H                   ; calling DOS print routine

        MOV AH, 4CH               ; 4CH is the number of the DOS exit routine
        INT 21H                   ; calling DOS exit routine

END start
```

L'annotation `byte ptr` indique que l'opérande qui suit doit être traitée comme pointant vers un octet. Que va imprimer ce programme ?

- ☐ XXXTERE
- ☐ XXXXERE
- ☐ XXX
- ☐ MXXXE

Question 15 On considère le code TASM suivant pour implémenter une fonction `foo` qui ajoute 33 au paramètre qui lui est passé en entrée :

```
foo:  ADD AX,33
      RET
```

Quel type de passage de paramètre(s) cette fonction `foo` utilise–t–elle ?

- ☐ le passage de paramètre(s) par la pile
- ☐ le passage de paramètre(s) par registre

Question 16 Si l'on modifie l'implantation d'une bibliothèque dynamique (ou partagée, *shared librairies*), faut–il nécessairement recompiler tous les exécutables qui utilisent cette bibliothèque pour leur permettre d'utiliser la nouvelle version de la bibliothèque ?

- ☐ non
- ☐ oui

Question 17 Quel est l'ordre de grandeur de la taille d'un cache de niveau 1 (L1) sur un processeur moderne ?

- ☐ 10 octets
- ☐ 1 megaoctet
- ☐ 1 gigaoctet
- ☐ 64 kilooctets

Question 18 Quel type de bibliothèque permet–il d'économiser de la place sur le disque dur et en mémoire vive ?

- ☐ les bibliothèques statiques (*static libraries*)
- ☐ les bibliothèques dynamiques (ou partagée, *shared librairies*)

Question 19 L'outil `make` recompile–t–il systématiquement l'intégralité d'un projet en cas de modification de l'une des parties du projet ?

- ☐ non, pas systématiquement
- ☐ oui, systématiquement

Question 20 On considère la règle suivante extraite d'un fichier Makefile :

```
myProg: myProg.o libmyLib.a
  echo "Je fais l'édition de lien"
  gcc -L. myProg.o -lmyLib -o myProg
```

Quelle est la cible (*target*) de cette règle ?

- ☐ `myProg.o`
- ☐ `gcc -L. myProg.o -lmyLib -o myProg`
- ☐ `myProg`

2 Deuxième partie : Questions ouvertes et problèmes (10 points)

Question 21 On considère le programme assembleur suivant :

```
.MODEL SMALL      ; memory model
.STACK            ; memory space for program instructions in the stack
.DATA             ; data segment
        MESSAGE DB 'HELLOWORLD','$'
.CODE
foo:    PUSH SI
        MOV SI,0

boucl:  CMP byte ptr BX[SI],'$' ; 'byte ptr' ensures BX[SI] is treated as a byte
        JE  fin
        ADD byte ptr BX[SI],AL  ; 'byte ptr' ensures BX[SI] is treated as a byte
        INC SI
        JMP boucl

fin:    POP SI
        RET

start:  MOV AX, @DATA           ; initialising
        MOV DS, AX              ; stack segment (needed on 16bit)

        MOV BX, offset MESSAGE
        MOV AL, -1
        CALL foo

        MOV DX, offset MESSAGE  ; DX contains start of string to be printed
        MOV AH, 09H             ; 09H is the number of the DOS print routine
        INT 21H                 ; calling DOS print routine

        MOV AH, 4CH             ; 4CH is the number of the DOS exit routine
        INT 21H                 ; calling DOS exit routine
END start
```

L'annotation `byte ptr` indique que l'opérande qui suit doit être traitée comme pointant vers un octet.

1. Que va imprimer ce programme ? Proposez un code C équivalent pour la fonction `foo`. **[1 point]**
2. La fonction `foo` utilise un passage de paramètres par registre. Quels sont les deux registres utilisés ? Quel sont leurs rôles ? **[1 point]**
3. Transformez le programme ci-dessus pour que `foo` utilise un passage de paramètres par la pile. Vous prendrez soin de commencer par représenter votre schéma de pile. Votre solution devra détailler le nouveau code de `foo`, et la manière dont `foo` doit maintenant être invoqué. Les directives d'organisation de la mémoire, l'initialisation du segment de données et les appels au BIOS n'ont en revanche pas besoin d'être répétés. **[4 points]**

☐A+ ☐A ☐B ☐C ☐D ☐E ☐F

Question 22 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. **[2 points]** ☐A+ ☐A ☐B ☐F

Question 23 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 22 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 exécutables (en utilisant `objdump -S -Mintel -d a.out`). La version non-optimisée donne le code assembleur suivant pour la fonction `foo()` :

```
00000000004004e4 <foo>:
  4004e4: 55                    push   rbp
  4004e5: 48 89 e5              mov    rbp,rsp
  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 :

```
00000000004004f0 <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.

[2 points] ☐A+ ☐A ☐B ☐F