ESIR SYS1 CC1 2013–2014
25 avril 2014

Nom et Prénom
Etudiant3
numéro d'étudiant : 99999903

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 À quoi la second ligne du code suivant est-elle équivalente ?

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

- [x] s[1] = 1
- [ ] s = 0
- [ ] s = s + 1

Question 2 Dans un processeur utilisant la technologie de *pipelining*, l'unité de chargement d'instructioin (Instruction Fetch Unit) et l'unité de décodage d'instruction (Instruction Decode Unit) d'un même coeur peuvent--elles être actives simultanément ?

- [ ] non
- [x] oui

Question 3 Comment réserver 32 octets (`0x20` en hexadécimal) sur la pile en x86 64 bits ?

- [x] `sub rsp,0x20`
- [ ] `add rbp,0x20`
- [ ] `sub rbp,0x20`
- [ ] `add rsp,0x20`

Question 4 Quel est l'encodage binaire de -3 sur 8 bits sur une architecture x86

- [x] `11111101`
- [ ] `11111111`
- [ ] `10000011`

Question 5 Que représente `f0` en notation hexadécimal ?

- [x] 240
- [ ] 15
- [ ] 160

Question 6 On considère le nombre hexadécimal `0xAABBCCDDEEFF`. Sur combien d'octets est-il stocké ?

- ☐ 16
- ■ 8
- ☐ 12
- ■ 6

Question 7 Les dernières architectures x86 Intel pour ordinateur personnel sont des architectures :

- ☐ 8 bit
- ☐ 16 bit
- ■ 64 bits
- ☐ 32 bit

Question 8 Un programme exécutable s'exécutant au dessus d'un OS (Linux, Windows)

- ■ s'exécute parfois directement sur le processeur physique
- ☐ ne s'exécute jamais directement sur le processeur physique

Question 9 Le bytecode java est directement exécutable par un processeur.

- ■ faux
- ☐ vrai

Question 10 Un code assembleur est directement exécutable par un processeur.

- ☐ vrai
- ■ faux

Question 11 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 12 Soit le programme assembleur suivant :

```
MOV BX,100
MOV [BX],256
INC BX
MOV AX,[BX]
```

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

- ☐ 256
- ☐ 0
- ■ 1

Question 13 Combien d'octets utilise la chaîne de caractères suivante en C ? `"\n\n\n"`

- ☐ 6
- ☐ 3
- ☐ 7
- ■ 4

Question 14 L'appel suivant est-il possible en assembleur x86 ?

```
CALL foo(10,20)
```

- [x] non
- [ ] oui

Question 15 La mémoire physique d'un ordinateur personnel est typiquement intégrée au processeur principal (CPU)

- [ ] vrai
- [x] faux

Question 16 Dans le langage C, combien d'octets occupe une valeur de type `int` ?

- [ ] 4 octets
- [ ] 2 octets
- [ ] 1 octet
- [x] cela dépend

Question 17 Sur combien de bits fonctionne le registre `ECX` ?

- [ ] 16
- [x] 32
- [ ] 64

Question 18 Soit le code assembleur suivant

```
MOV AX,4
MOV BX,5
PUSH AX
PUSH BX
MOV BX,3
MOV AX,4
POP BX
POP AX
```

À la suite de ce code :

- [ ] AX et BX contiennent 2 et 3 respectivement.
- [ ] AX et BX contiennent 5 et 4 respectivement.
- [x] AX et BX contiennent 4 et 5 respectivement.
- [ ] AX et BX contiennent 3 et 2 respectivement.

Question 19 Le code suivant compile-t-il en C ?

```
#include <stdio.h>

int main( int argc, char** argv) {
  int i = "Hi" - "Bye";
  printf("%i\n",i);
}
```

- [x] oui
- [ ] non

Question 20 On considère le nombre hexadécimal `0x10203040` stocké en mémoire sur une architecture Big Endian. Quel est la séquence d'octets qui encode ce nombre ?

- ☐ `0x10,0x20,0x30,0x40`
- ■ `0x40,0x30,0x20,0x10`

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

Question 21 Expliquez quel est le rôle de pile d'appel (*Call Stack*) en C. **[2 points]** ☐O ☐P ■V

Question 22 Expliquez le code assembleur suivant, fourni par l'outil **objdump** au début de la procédure **foo(..)** :

```
void foo(void) {
4004e4: 55                  push   rbp
4004e5: 48 89 e5            mov    rbp,rsp
4004e8: 48 83 ec 10         sub    rsp,0x10
```

[**2 points**] ☐O ☐P ■V

Question 23 Implémentez en assembleur le programme suivant.

```
int x = 5;
int y = 10;
int z ;

if (x==y) {
  z = x ;
} else {
  z = y - 1 ;
}
```

(La syntaxe de la segmentation du code n'est pas importante. Ce sont les instructions assembleurs qui nous intéressent. On supposera aussi que x, y, et z sont statiquement alloués, et peuvent être référencés par des symboles, par exemple :

```
.DATA
X DW  5
Y DW  10
Z DW  1 DUP(?)
```

) **[3 points]**

■**A** □**B** □**C** □**E**

Question 24 On considère le programme C suivant :

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

void strange_function(char* s) {
  int i;
  for(i=0; i < strlen(s); i++) {
    s[i] = 'z' - s[i] + 'a';
  } // EnFor
} // EndProcedure strange_function

int main( int argc, char** argv) {

  char x[] = "alice";
  printf("%s\n", x );

  strange_function(x);
  printf("%s\n", x );

  strange_function(x);
  printf("%s\n", x );

} // EndMain
```

On observe l'exécution suivante

```
$ gcc surprise.c
$ ./a.out
alice
zorxv
alice
```

Expliquer pourquoi le programme produit cette sortie.[**3 points**] ■**A** □**B** □**C** □**E**

Correction