Organisation des ordinateurs
Examen de seconde session 2022
Énoncés et solutions

Énoncés

1. L'état d'un système physique possède 4 configurations équiprobables, notées $(0,0)$, $(0,1)$, $(1,0)$ et $(1,1)$. Pour des raisons fondamentales, quand on mesure un état, on est incapable de distinguer les deux configurations $(0,1)$ et $(1,0)$. En d'autres termes, pour ces deux configurations, le résultat de la mesure est identique. Les autres configurations sont quant à elles toujours mesurées précisément.
   (a) Quelle quantité d'information reçoit-on si l'on mesure un état égal à $(0,1)$ ou $(1,0)$?
   (b) Combien de mesures d'état est-on capable de mémoriser, en moyenne, à l'aide d'un composant de mémoire vive de 16 GB ?

2. (a) Donner un nombre représentable sur n bits par complément à deux mais pas par complément à un.
   (b) Calculer le produit $3 \times (-3)$ en représentant les nombres par complément à deux sur 6 bits.
   (c) Quelle est la relation entre un nombre entier non signé représenté sur n bits et le nombre réel représenté par la même suite de bits en virgule fixe, avec k bits après la virgule ? Dans le cas de nombres signés représentés par complément à deux, cette relation est-elle toujours valable ?
   (d) Quel est le plus grand nombre réel dont la représentation dans le standard IEEE754 en simple précision possède une mantisse dénormalisée ?

3. (a) Qu'est-ce que l'unité arithmétique et logique d'un processeur ? À quels autres composants du processeur est-elle reliée, et dans quel but ?
   (b) À quoi servent les drapeaux CF et ZF d'un processeur d'architecture x86-64 ? Donner un exemple concret d'instruction en assembleur qui lève ces deux drapeaux.
   (c) Expliquer en détail l'opération effectuée par l'instruction RET de l'architecture x86-64, en expliquant notamment quels registres du processeur elle modifie, et comment elle les modifie.

(d) Expliquer comment se déroulera l'exécution du fragment de code assembleur x86-64 suivant. Combien d'itérations de la boucle seront-elles effectuées ?

```
        MOV RAX, -42
        AND RAX, 0xffff
boucle: DEC RAX
        JNZ boucle
```

4. On souhaite programmer une fonction qui ne prend pas d'argument, et qui retourne à l'issue de son exécution l'adresse d'un tableau `t` de 2^{16} octets. Ce tableau doit être tel que pour chaque indice i tel que $0 \leq i < 2^{16}$, la case `t`$[i]$ du tableau contient 1 si i est un multiple de 16, et 0 sinon.

   (a) Écrire, en pseudocode ou en langage C (au choix), un algorithme permettant de résoudre ce problème.

   (b) Traduire cet algorithme en un fragment de code assembleur x86-64, en veillant à respecter la convention d'appel de fonctions des systèmes *Unix*.

Exemples de solutions

1. (a) La probabilité d'obtenir une mesure dont le résultat est $(0,1)$ ou $(1,0)$ vaut $1/2$. La quantité d'information correspondante est donc égale à 1 bit.

   (b) La probabilité d'obtenir un résultat égal à $(0,0)$ ou $(1,1)$ est (dans chaque cas) égale à $1/4$, ce qui correspond à 2 bits d'information. On a donc en résumé :
   une probabilité de $1/4$ de mesurer $(0,0)$, apportant 2 bits,
   une probabilité de $1/4$ de mesurer $(1,1)$, apportant 2 bits,
   une probabilité de $1/2$ de mesurer $(0,1)$ ou $(1,0)$, apportant 1 bit.
   La quantité d'information moyenne d'une mesure vaut donc

$$\frac{1}{4}2 + \frac{1}{4}2 + \frac{1}{2}1 = 1{,}5 \text{ bits.}$$

   Un composant de mémoire vive de 16 GB permet de mémoriser

$$8 \,.\, 16 \,.\, 2^{30} = 2^{37} \text{ bits,}$$

   ce qui correspond à

$$\frac{2^{37}}{1{,}5} \approx 91{,}6 \,.\, 10^9 \text{ mesures.}$$

2. (a) -2^{n-1}, qui est le plus petit nombre représentable sur n bits.

   (b)

$3:$		0	0	0	0	1	1
$-3:$	$\times$	1	1	1	1	0	1
		0	0	0	0	1	1
		0	0	1	1		
		0	1	1			
		1	1				
	$+$	1					
$-9:$		1	1	0	1	1	1

   (c) Appelons z le nombre entier et r le nombre réel mentionnés dans l'énoncé. On a $z = 2^k\, r$, et cette relation est aussi valable pour des nombres signés représentés par complément à deux.

   (d) Pour avoir une mantisse dénormalisée, l'exposant doit être égal à -127. La plus grande mantisse possible est obtenue en forçant à 1 tous les bits de sa représentation, ce qui la rend égale à $2 - 2^{-22}$. Le nombre demandé vaut donc

$$2^{-127}\,(2 - 2^{-22}) = 2^{-126} - 2^{-149}.$$

3. (a) L'unité arithmétique et logique (ALU) est le composant du processeur chargé d'effectuer les opérations de traitement de données, comme par exemple des additions ou des multiplications de nombres. L'ALU est reliée au bus interne du processeur, ce qui lui permet de recevoir les données à traiter et de transmettre les résultats de ses calculs. L'ALU est également connectée à l'unité de contrôle du processeur par le biais de lignes de contrôle, qui lui indiquent à chaque instant quelle opération elle doit effectuer.

   (b) Le drapeau CF indique un dépassement arithmétique dans le cas de nombres non signés. Le drapeau ZF indique que le résultat d'une opération est nul. Par exemple, l'opération

```
ADD AL, AL
```

   lève ces deux drapeaux lorsque la valeur initiale de `AL` est égale à 0x80.

   (c) L'instruction `RET` dépile une adresse de 64 bits et effectue un saut inconditionnel vers cette adresse. Cette instruction réalise donc les opérations suivantes :

      i. Lire 8 octets en mémoire depuis l'adresse donnée par la valeur de `RSP`, et les recopier dans le registre `RIP`.

      ii. Incrémenter `RSP` de 8 unités.

(d) Les deux premières instructions écrivent la valeur -42 dans le registre `RAX`, et forcent ensuite à 0 tous les bits de ce registre à l'exception des 16 bits de poids faible. À l'issue de ces instructions, le registre `RAX` contient donc la valeur $2^{16} - 42 = 65494$. Ce nombre correspond au nombre d'itérations qui seront effectuées, puisque chacune d'entre elles décrémente `RAX` jusqu'à ce que ce registre devienne nul.

4. (a)
```
char t[0x10000] = { };  // Éléments initialisés à 0.

char *f(void)
{
  unsigned i;

  for (i = 0; i < 0x10000; i += 16)
    t[i] = 1;

  return t;
}
```

(b)
```
          .intel_syntax noprefix
          .data
t:        .fill 0x10000, 1, 0
          .text
          .global f
          .type f, @function
f:        MOV   RDI, 0
boucle:   MOV   byte ptr[RDI + t], 1
          ADD   DI, 16
          JNZ   boucle
          MOV   RAX, offset flat:t
          RET
```