Organisation des ordinateurs
Examen de juin 2023
Énoncés et solutions

Énoncés

1. Un générateur de signaux fonctionne de la façon suivante. Chaque fois qu'un nouveau signal doit être émis, le générateur tire au hasard deux nombres entiers appartenant à l'intervalle $[1, 3]$, et donne ensuite au signal une valeur égale à la somme de ces deux nombres. Chacun des deux tirages est indépendant de l'autre, et présente une probabilité identique de produire chaque nombre dans l'intervalle spécifié.
   (a) Quelle est la quantité d'information contenue dans un signal émis par le générateur lorsque la valeur de ce signal est égale à 3 ?
   (b) On utilise un disque dur d'une capacité commerciale de 2 TB pour enregistrer les signaux produits par le générateur. Si 10^9 signaux sont émis chaque seconde, et sont mémorisés de la façon la plus efficace possible, combien de temps en moyenne faudra-t-il pour que ce disque dur soit entièrement rempli ?

2. (a) Donner un nombre strictement négatif dont les représentations sur 4 bits par complément à deux et par valeur signée sont identiques.
   (b) Calculer le produit -3×-7 en représentant les nombres par complément à deux. Vous êtes libre de choisir la taille des représentations.
   (c) Comment représenteriez-vous le nombre $\frac{1}{3}$ en virgule fixe, avec 2 chiffres avant et 4 chiffres après la virgule ?
   (d) Donner un exemple d'addition de deux nombres de 4 bits représentés par complément à un, aboutissant au résultat -7. On demande de détailler toutes les étapes du calcul.
   (e) Construire la représentation en simple précision du nombre -2^{-126} selon le standard IEEE754, et en donner une écriture hexadécimale.

3. (a) En quoi les architectures *Von Neumann* et *Harvard* diffèrent-elles ?
   (b) Que sont les registres d'un processeur ? À quoi servent-ils ?
   (c) Donner un exemple d'instruction x86-64 employant l'adressage direct, et expliquer l'opération effectuée par cette instruction.

(d) Expliquer comment se déroulera l'exécution du fragment de code assembleur x86-64 suivant, en faisant l'hypothèse que la pile est initialement correctement configurée. Quelles seront les valeurs finales de `RAX` et `RBX` ?

```
MOV  RAX, 0
PUSH RAX
OR   byte ptr [RSP + 1], 1
POP  RBX
```

4. On souhaite programmer une fonction `indice_negatif` acceptant en arguments un tableau d'entiers `t` et le nombre d'éléments `n` de celui-ci, et capable de retourner le plus petit indice $i \in [0, \mathtt{n} - 1]$ tel que $\mathtt{t}[i] < 0$, ou -1 si cet indice n'existe pas.

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

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

Exemples de solutions

1. (a) Émettre un signal nécessite de tirer au hasard deux nombres entiers n_1 et n_2 dans l'intervalle $[1, 3]$, indépendamment et avec une probabilité uniforme. Il n'y a que deux possibilités d'obtenir un total égal à 3 : $n_1 = 1$ et $n_2 = 2$, ou $n_1 = 2$ et $n_2 = 1$. Chacune de ces deux possibilités présente une probabilité égale à $1/9$ de se produire ; la probabilité d'obtenir un signal égal à 3 vaut donc $2/9$. La quantité d'information contenue dans un tel signal est donc

$$\log_2 \frac{9}{2} \approx 2{,}17 \text{ bits.}$$

   (b) Un disque dur vendu comme contenant 2 TB permet de mémoriser 16.10^{12} bits d'information, soit

$$\frac{16.10^{12}}{2{,}17} \approx 7{,}374.10^{12} \text{ signaux.}$$

   Au rythme de 10^9 signaux par seconde, il faudra donc en moyenne environ 7374 secondes, soit un peu moins de 2 heures et 3 minutes, pour remplir le disque dur.

2. (a) Le nombre recherché étant strictement négatif, ses représentations commencent nécessairement par un bit de signe égal à 1. Par complémenent à deux, le nombre représenté par $1b_2b_1b_0$ vaut

$$-8 + \sum_{i=0}^{2} b_i 2^i.$$

Par valeur signée, il vaut

$$-\sum_{i=0}^{2} b_i 2^i.$$

En égalant ces deux expressions, on obtient

$$\begin{aligned} -8 + \sum_{i=0}^{2} b_i 2^i &= -\sum_{i=0}^{2} b_i 2^i \\ \sum_{i=0}^{2} b_i 2^i &= 4. \end{aligned}$$

Le nombre en question est donc -4.

(b) La plus courte représentation de -3 est 101, et celle de -7 est 1001. On choisit donc d'effectuer l'opération sur $3 + 4 = 7$ bits. On obtient le calcul suivant (en négligeant les reports aux positions supérieures à 6).

-3 :		1	1	1	1	1	0	1
-7 :	$\times$	1	1	1	1	0	0	1
			$\boxed{1}$	$\boxed{1}$				
		1	1	1	1	1	0	1
		1	1	0	1			
		1	0	1				
		0	1					
	$+$	1						
21 :		0	0	1	0	1	0	1

(c) Puisqu'il y a quatre chiffres après la virgule, la représentation recherchée correspond à la représentation entière du nombre donné multiplié par 16. Le nombre entier le plus proche de 16/3 est 5, donc le résultat est égal à la représentation de 5 sur 6 bits, c'est-à-dire 000101.

(d) Le plus simple consiste à calculer la somme de $+0$ et de -7 :

$$\begin{array}{lcccccc} +0: & & 0 & 0 & 0 & 0 \\ -7: & + & 1 & 0 & 0 & 0 \\ \hline -7: & & 1 & 0 & 0 & 0 \end{array}$$

Étant donné qu'aucun report n'a été produit à la position 4, aucune correction du résultat ne doit être effectuée.

(e) Ce nombre est négatif, donc le bit de signe de sa représentation vaut $\boxed{1}$.
Il est possible de représenter ce nombre avec une mantisse normalisée, en fixant l'exposant à -126 et la mantisse à 1. L'exposant est alors représenté par $\boxed{00000001}$, et la mantisse par $\boxed{00000000000000000000000}$ (où les 23 bits sont tous nuls). La représentation demandée vaut donc

$$\boxed{10000000100000000000000000000000},$$

c'est-à-dire $\boxed{80800000}$ en hexadécimal.

3. (a) L'architecture *Von Neumann* place la mémoire de programme et la mémoire de données dans un espace d'adressage unique, alors que l'architecture *Harvard* sépare ces deux mémoires.

(b) Les registres du processeur sont des éléments de mémoire vive situés à l'intérieur du processeur. Ils servent à retenir les résultats intermédiaires manipulés par les programmes. Des registres particuliers retiennent également les données nécessaires au fonctionnement du processeur, telles que l'opcode de l'instruction en cours d'exécution ou l'adresse de la prochaine instruction à effectuer.

(c) L'instruction

```
MOV byte ptr [1000], 0
```

écrit un octet nul à l'adresse 1000 de la mémoire.

(d) Les deux premières instructions attribuent la valeur 0 à `RAX` et empilent cette valeur, représentée sur 8 octets. La troisième instruction force à 1 le bit de poids faible de l'octet situé immédiatement après celui qui est en sommet de pile. À ce stade, la pile contient donc à son sommet les 8 octets 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00 et 0x00 (énumérés dans l'ordre des adresses croissantes). L'architecture x86-64 étant petit-boutiste, la dernière instruction dépile donc la valeur 0x100, c'est-à-dire

256, et la place dans RBX. Les valeurs finales de RAX et RBX sont donc respectivement égales à 0 et 256.

4. (a)
```
int indice_negatif(int t[], unsigned n)
{
  unsigned i;

  for (i = 0; i < n; i++)
    if (t[i] < 0)
      return i;

  return -1;
}
```

(b)
```
        .intel_syntax noprefix
        .text
        .global indice_negatif
        .type indice_negatif, @function
indice_negatif:
        PUSH RBP
        MOV RBP, RSP
        MOV RAX, 0
boucle: CMP EAX, ESI
        JAE suite
        CMP dword ptr [RDI + 4 * RAX], 0
        JL  fin
        INC EAX
        JMP boucle
suite:  MOV EAX, -1
fin:    POP RBP
        RET
```