Organisation des ordinateurs
Examen de septembre 2021
Énoncés et solutions

Note : En raison des mesures sanitaires qui étaient en vigueur en 2021, cet examen était plus court qu'habituellement.

Énoncés

1. Les premiers ordinateurs utilisaient des *cartes perforées* pour encoder les programmes. Une carte perforée est un morceau de carton rectangulaire structuré en 12 lignes et 80 colonnes. Chaque position (c'est-à-dire, chaque combinaison d'une ligne et d'une colonne) contient soit une perforation, soit une absence de perforation.
   (a) Sous l'hypothèse que les probabilités d'avoir ou non une perforation à une position donnée d'une carte sont égales et indépendantes de celles des autres positions, calculer la quantité totale d'information contenue dans un programme encodé à l'aide de 1000 cartes perforées.
   (b) Considérons à présent qu'à chaque position d'une carte, la probabilité d'y trouver une perforation est égale à $1/20$ et est indépendante de celle des autres positions. Si une carte présente 60 perforations, quelle quantité d'information totale contient-elle ?

2. (a) Pour un nombre entier strictement positif n donné, quelle est la taille de la plus petite représentation de -2^n par valeur signée, par complément à un et par complément à deux ? (Justifier votre réponse.)
   (b) Calculer le produit de -4 et de -5 représentés par complément à deux. (Vous êtes libres de choisir le nombre de bits des représentations.)
   (c) Représenter le plus précisément possible le nombre $1/3$ en virgule fixe avec 4 chiffres avant et 4 chiffres après la virgule.
   (d) Donner un exemple d'opération arithmétique impliquant deux nombres x et y représentés par le procédé IEEE754, et produisant un résultat z égal à *NotANumber (NaN)*. (Donner explicitement une représentation IEEE754 en simple précision de x, y et z. Il n'est pas demandé de décrire les détails du calcul effectué.)

3. À quoi le bus interne d'un processeur sert-il ? À quels composants du processeur est-il relié ?

4. On souhaite programmer une procédure prenant comme arguments l'adresse `t` d'un tableau d'octets, et le nombre `n` ≥ 0 d'éléments de ce tableau représenté de façon non signée sur 32 bits.

   Le travail effectué par cette procédure consiste à mettre à 1 le bit de poids faible et à 0 le bit de poids fort de tous les octets du tableau, les autres bits restant inchangés. Par exemple, si le tableau contient initialement [0x7e, 0xff, 0x84, 0xa9, 0x21], alors il doit contenir [0x7f, 0x7f, 0x05, 0x29, 0x21] à l'issue de l'exécution de la procédure.

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

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

Exemples de solutions

1. (a) Chaque perforation apporte 1 bit d'information. Un programme de 1000 cartes comprend $1000 \times 12 \times 80 = 960000$ perforations, ce qui correspond à 960000 bits, ou encore 937,5 Kbits.

   (b) Une carte possède au total $12 \times 80 = 960$ positions. La probabilité qu'une carte présente exactement 60 perforations est donc égale à

$$p = C_{960}^{60} \left(\frac{1}{20}\right)^{60} \left(\frac{19}{20}\right)^{900},$$

   où

$$C_{960}^{60} = \frac{960 \,.\, 959 \,.\, \cdots \,.\, 901}{60 \,.\, 59 \,.\, \cdots \,.\, 1} \approx 1{,}579 \,.\, 10^{96}$$

   représente le nombre de combinaisons possibles de 60 perforations parmi 960 possibilités.

   La quantité d'information contenue dans une telle carte vaut donc

$$\begin{aligned} \log_2 \frac{1}{p} &\approx -\log_2 1{,}579 \,.\, 10^{96} + 60 \log_2 20 + 900(\log_2 20 - \log_2 19) \\ &\approx 6{,}352 \text{ bits.} \end{aligned}$$

2. (a) Pour les représentations par valeur signée et par complément à un, l'intervalle des valeurs représentables sur k bits est $[-2^{k-1} + 1, 2^{k-1} - 1]$. Pour représenter le nombre -2^n, on doit donc choisir $k \geq n + 2$, ce qui signifie que la plus petite représentation de ce nombre possède $n + 2$ bits.

Pour le complément à deux, l'intervalle des valeurs représentable à l'aide de k bits est $[-2^{k-1}, 2^{k-1}-1]$, donc la plus petite représentation de -2^n possède $n+1$ bits.

(b)

-4 :		1	1	1	1	0	0
-5 :	$\times$	1	1	1	0	1	1
		1	1	1	1	0	0
		1	1	1	0	0	
		1	0	0			
		0	0				
	$+$	0					
20 :		0	1	0	1	0	0

(c) Cela revient à représenter de façon non signée sur 8 bits le nombre entier le plus proche de

$$2^4 \frac{1}{3} \approx 5{,}333,$$

c'est-à-dire 5. La représentation demandée est donc $\boxed{00000101}$.

(d) On peut par exemple choisir $x = y = 0$, et calculer $z = x/y$.
Dans ce cas, les représentations de x et de y sont composées de 32 bits égaux à 0. La représentation de z s'obtient en choisissant un bit de signe quelconque, un exposant le plus grand possible, et une mantisse comprenant au moins un bit non nul. Un exemple de telle représentation est $\boxed{0\,11111111\,10000000000000000000000}$.

3. Le bus interne du processeur est un canal de communication permettant aux composants de celui-ci de s'échanger des données. Il est relié à l'ensemble de ces composants, notamment la banque de registres, l'ALU, l'unité de contrôle et le gestionnaire de communications.

4. (a) Il suffit de parcourir le tableau et pour chacun de ses éléments, forcer à 1 son bit de poids faible à l'aide d'un ou logique et forcer à 0 son bit de poids fort à l'aide d'un et logique. On obtient le code C suivant.

```
void modifier_tableau(unsigned char t[], unsigned n)
{
  while (n)
  {
    *t |= 0x01;
    *t &= 0x7f;
    n--;
    t++;
  }
}
```

(b) Selon la convention d'appel de fonctions, le paramètre `t` correspond au registre `RDI`, et `n` à `ESI` (qui représente les 32 bits de poids faible de `RDI`). Une traduction directe du programme obtenu au point (a) fournit le code suivant.

```
            .intel_syntax noprefix
            .text
            .global modifier_tableau
            .type modifier_tableau, @function

modifier_tableau:
            PUSH  RBP
            MOV   RBP, RSP
boucle:     CMP   ESI, 0
            JE    fin
            OR    byte ptr[RDI], 0x01
            AND   byte ptr[RDI], 0x7f
            DEC   ESI
            INC   RDI
            JMP   boucle
fin:        POP   RBP
            RET
```