Organisation des ordinateurs
Examen de juin 2024
Énoncés et solutions

Énoncés

1. Le code génétique d'un être humain est représenté par une séquence de nucléotides composant les brins d'ADN contenus dans ses chromosomes. Il y a quatre nucléotides appelés Adénine (A), Guanine (G), Thymine (T) et Cytosine (C). Le génome complet est composé d'environ 6.10^9 nucléotides, avec en moyenne 20,5% de C, 20,5% de G, 29,5% de A et 29,5% de T.
   (a) Calculer la quantité d'information contenue dans un génome dont la proportion de nucléotides correspond aux valeurs moyennes.
   (b) Une machine capable de séquencer l'ADN dispose d'un disque dur dont la capacité commerciale est égale à 8 TB. Combien de génomes satisfaisant les hypothèses du point (a) ce disque dur permet-il de mémoriser ?

2. (a) Quels sont les plus petits entiers représentables sur 8 bits de façon non-signée, par valeur signée, par complément à un et par complément à deux ?
   (b) Calculer la somme $(-\frac{11}{4}) + \frac{9}{8}$ en complément à deux en virgule fixe avec 3 bits avant et 3 bits après la virgule.
   (c) Calculer le produit $9 \times (-3)$ en représentant les nombres par complément à deux. Vous pouvez choisir, en le justifiant, le nombre de bits utilisés pour réaliser cette opération.
   (d) Encoder le nombre 2^{-150} dans le format IEEE 754 simple précision et double précision, le plus précisément possible. Dans chaque cas, indiquer si les représentations obtenues sont exactes ou approximées.

3. (a) Qu'est-ce que le code machine ? Quelles sont les étapes permettant de traduire un code source en langage C vers du code machine ?
   (b) Que sont les drapeaux et en quoi sont-ils utiles lors de l'exécution d'un programme ? Donner un exemple d'instruction x86-64 qui lève simultanément les drapeaux `CF` et `OF`.
   (c) Expliquer étape par étape comment se déroulera l'exécution du fragment de code assembleur x86-64 suivant, en faisant l'hypothèse que la pile est initialement correctement configurée. Quelle sera la valeur finale de `RAX` ?

```
MOV  RAX, -0x11
XOR  RAX, -1
SUB  RSP, RAX
MOV  qword ptr[RSP], RAX
MOV  AH, byte ptr[RSP]
MOV  word ptr[RSP + 2], AX
POP  RAX
```

4. On souhaite programmer une fonction `f` acceptant en arguments un tableau d'entiers signés `t` et le nombre d'éléments `n` de ce tableau. Cette fonction doit parcourir le tableau élément par élément et permuter les valeurs des cases i et $i+1$ si la valeur de la case i est plus grande que celle de la case $i+1$, pour $i = 0, 1, \ldots, \mathtt{n}-2$. Pour ce faire, on définira une fonction `permutation` qui prend en entrée deux pointeurs vers des entiers, et permute ces entiers.

   (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*. En particulier, ce programme doit contenir une implémentation de la fonction `permutation` qui suit cette convention.

Exemples de solutions

1. (a) Un nucléotide C ou G présente une probabilité d'occurrence de 0,205, ce qui correspond à une quantité d'information égale à

$$\log_2 \frac{1}{0{,}205} \approx 2{,}286 \text{ bits.}$$

Pour un nucléotide A ou T, on obtient

$$\log_2 \frac{1}{0{,}295} \approx 1{,}761 \text{ bit.}$$

Un génome composé de 6.10^9 nucléotides dans les proportions moyennes contient donc

$$6.10^9(2\,.\,0{,}205\,.\,2{,}286 + 2\,.\,0{,}295\,.\,1{,}761) \approx 11{,}859\,.\,10^9 \text{ bits}$$

d'information, c'est-à-dire environ 11,045 Gbits.

(b) Un disque dur d'une capacité commerciale de 8 TB permet de mémoriser $64 \,.\, 2^{40} \approx 70{,}369 \,.\, 10^{12}$ bits d'information, ce qui permet de représenter 5933 génomes complets.

2. (a) — De façon non-signée : 0.
— Par valeur signée : $-(2^7 - 1) = -127$.
— Par complément à un : $-2^7 + 1 = -127$.
— Par complément à deux : $-2^7 = -128$.

(b) Puisqu'il y a 3 bits après la virgule, cette opération revient à calculer la somme $-22 + 9$ avec des entiers représentés par complément à deux sur 6 bits. On a :

$$\begin{array}{cccccccc} & & \boxed{1} & & & & & \\ & 1 & 0 & 1 & 0 & 1 & 0 \\ + & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline & 1 & 1 & 0 & 0 & 1 & 1 \end{array}$$

Le résultat représente l'entier signé -13, qui correspond à $-13/8 = -1{,}625$ en virgule fixe avec 3 bits avant et 3 bits après la virgule.

(c) La plus petite représentation en complément à deux de 9 (égale à 01001) possède 5 bits et celle de -3 (égale à 101) possède 3 bits. On peut donc effectuer l'opération de multiplication sur $5 + 3 = 8$ bits. On obtient :

$$\begin{array}{cccccccccc} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ \times & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ \hline & \boxed{1} & \boxed{1} & \boxed{1} & \boxed{1} & & & & \\ & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ & 0 & 0 & 1 & 0 & 0 & 1 & & \\ & 0 & 1 & 0 & 0 & 1 & & & \\ & 1 & 0 & 0 & 1 & & & & \\ & 0 & 0 & 1 & & & & & \\ & 0 & 1 & & & & & & \\ + & 1 & & & & & & & \\ \hline & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}$$

Le résultat représente bien le nombre -27.

(d) — En simple précision, ce nombre est trop petit pour pouvoir être représenté à l'aide d'une mantisse normalisée. En effet, cela nécessiterait une mantisse au moins égale à 1 et un exposant supérieur ou égal à -126, or on a $2^{-150} < 2^{-126}$.

Avec une mantisse dénormalisée, l'exposant est égal à -127, ce qui rend la mantisse égale à

$$\frac{2^{-150}}{2^{-127}} = 2^{-23}.$$

En simple précision, une mantisse dénormalisée est encodée avec un poids faible égal à 2^{-22}. Il n'est donc pas possible de représenter exactement le nombre 2^{-23}, qui se retrouve arrondi à zéro. La représentation demandée correspond donc à la représentation de $+0$, composée de 32 bits tous égaux à 0. Il s'agit d'une représentation approximée de 2^{-150}.

— En double précision, on a une mantisse normalisée égale à 1 et un exposant égal à -150. La représentation demandée est donc composée
  — d'un bit de signe égal à $\boxed{0}$;
  — d'un exposant représenté par l'encodage non signé de $-150 + 1023 = 873$ sur 11 bits, c'est-à-dire $\boxed{01101101001}$;
  — d'une mantisse représentée par 52 bits égaux à 0.

  En résumé, la représentation demandée est $\boxed{001101101001000\cdots0}$. Cette représentation de 2^{-150} est exacte.

3. (a) Le code machine est le code directement exécutable par le processeur, encodé sous la forme de valeurs numériques placées dans la mémoire de programme. Pour traduire un code source C en code machine, il faut d'abord le compiler afin d'obtenir du code assembleur, et convertir celui-ci en code objet grâce au programme d'assemblage. Ensuite, l'édition de liens combine l'ensemble du code objet constituant le programme en un fichier exécutable contenant le code machine.

   (b) Les drapeaux sont des éléments de la banque de registres du processeur retenant des valeurs binaires. Celles-ci fournissent des informations sur le résultat d'opérations effectuées par l'ALU, en particulier sur la production ou non d'un report arithmétique ou d'un dépassement. Par exemple, si le registre `AL` contient initialement la valeur 0x80, alors l'instruction

   ```
   ADD AL, AL
   ```

   lève les drapeaux `CF` et `OF`, indiquant un dépassement pour des valeurs respectivement non signées et signées.

   (c) — L'instruction `MOV RAX, -0x11` écrit la valeur -17 dans `RAX`. Tous les bits de ce registre deviennent donc égaux à 1 sauf celui à la position 4 qui vaut 0.

— L'instruction `XOR RAX, -1` complémente tous les bits de `RAX`, qui deviennent donc égaux à 0 sauf celui à la position 4 qui vaut 1. En d'autres termes, `RAX` vaut maintenant 16.

— L'instruction `SUB RSP, RAX` alloue 16 octets sur la pile.

— L'instruction `MOV qword ptr[RSP], RAX` écrit la valeur 16 (représentée sur 64 bits) dans les 8 octets situés au sommet de la pile.

— L'instruction `MOV AH, byte ptr[RSP]` recopie la valeur 16 dans les bits situés aux positions 8 à 15 du registre `RAX`. Celui-ci prend donc la valeur 0x1010.

— L'instruction `MOV word ptr[RSP + 2], AX` recopie la valeur 0x1010 dans les 16 bits situés au positions 16 à 31 de la valeur de 64 bits placée en sommet de pile. Celle-ci devient donc égale à 0x10100010.

— La dernière instruction `POP RAX` dépile cette valeur 0x10100010 et la place dans `RAX`.

4. (a)
```
void permutation(int *p1, int *p2)
{
  int aux;

  aux = *p1;
  *p1 = *p2;
  *p2 = aux;
}

void f(int t[], unsigned long n)
{
  unsigned long i;

  for (i = 0; i + 1 < n; i++)
    if (t[i] > t[i + 1])
      permutation(t + i, t + i + 1);
}
```

(b)
```
        .intel_syntax noprefix
        .text
        .global permutation
        .type permutation, @function
```

```
permutation:
        MOV  EAX, dword ptr[RDI]
        MOV  ECX, dword ptr[RSI]
        MOV  dword ptr[RDI], ECX
        MOV  dword ptr[RSI], EAX
        RET
        .global f
        .type f, @function
f:      PUSH RBP
        MOV  RBP, RSP
        MOV  RAX, 0    # i = 0
boucle: MOV  RCX, RAX
        INC  RCX       # i + 1
        CMP  RCX, RSI
        JAE  fin       # i + 1 >= n ?
        MOV  EDX, dword ptr[RDI + 4 * RAX]
        CMP  EDX, dword ptr[RDI + 4 * RCX]
        JLE  suite     # t[i] <= t[i + 1] ?
        PUSH RDI
        PUSH RSI
        PUSH RAX       # sauvegardes
        SUB  RSP, 8    # (pour RSP multiple de 16)
        ADD  RDI, RAX
        ADD  RDI, RAX
        ADD  RDI, RAX
        ADD  RDI, RAX  # t + i
        MOV  RSI, RDI
        ADD  RSI, 4    # t + i + 1
        CALL permutation
        ADD  RSP, 8
        POP  RAX
        POP  RSI
        POP  RDI       # récupération sauvegardes
suite:  INC  RAX       # i++
        JMP  boucle
fin:    POP  RBP
        RET
```

Remarque : Étant donné que la fonction `permutation` ne manipule pas la pile et n'effectue pas d'appel de fonction, il est inutile qu'elle crée un *stack frame*.