Organisation des ordinateurs
Examen d'août 2025
Énoncés et solutions

Énoncés

1. (a) On effectue une expérience qui consiste à lancer trois fois de suite une pièce de monnaie non truquée, et à compter le nombre de fois parmi ces trois lancers qu'elle retombe sur son côté pile. Quelle quantité d'information obtient-on dans le cas où cette expérience produit un résultat égal à 2 ?

   (b) Quelle quantité d'information, exprimée en térabits, un disque dur vendu comme ayant une capacité de 12 téraoctets permet-il de mémoriser ?

2. (a) Donner la représentation hexadécimale sur 16 bits du nombre -2025 selon les encodages
   — par valeur signée,
   — par complément à un, et
   — par complément à deux
   des entiers.

   (b) Calculer le plus précisément possible la somme $-1 + \frac{7}{16}$ en complément à deux en virgule fixe avec 3 bits avant et 3 bits après la virgule.

   (c) Calculer le produit $-4 \,.\, (-4)$ à partir de la représentation en complément à deux sur n bits des nombres, où n est le plus petit entier qui permet d'effectuer l'opération sans dépassement arithmétique.

   (d) Donner la représentation en simple précision du nombre $1{,}5 \,.\, 2^{-130}$ selon le standard IEEE754. Cette représentation est-elle exacte ou approximée ? (Justifier votre réponse.)

3. (a) En quoi la mémoire vive et la mémoire morte diffèrent-elles ? Donner un exemple d'utilisation de chacun de ces types de mémoire dans un ordinateur moderne.
   (b) Décrire trois modes d'adressage de votre choix définis par l'architecture x86-64. Illustrer chacun d'entre eux à l'aide d'un exemple concret d'ins-

truction qui l'utilise, en précisant la valeur des registres concernés avant et après l'exécution de l'instruction.

(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, 0x1908
MOV  RBX, 0x2025
PUSH RAX
PUSH RBX
POP  RAX
ADD  RAX, qword ptr[RSP]
POP  RBX
SUB  RSP, 8
XOR  RBX, RAX
MOV  qword ptr[RSP], RBX
POP  RAX
```

4. On souhaite programmer une fonction `nb_changements_signe` acceptant en arguments un tableau d'entiers signés `t` représentés sur 16 bits, et le nombre d'éléments `n` de ce tableau donné sous la forme d'un entier non signé de 32 bits. Cette fonction doit retourner le nombre de changements de signe entre les paires d'éléments consécutifs du tableau. On considère qu'il y a un changement de signe chaque fois que deux éléments consécutifs $\texttt{t}[i]$ et $\texttt{t}[i+1]$ sont tels qu'un des deux est positif ou nul et l'autre strictement négatif. Par exemple, si `t` contient $[-2, 1, -1, 3]$, alors la fonction doit retourner 3. Si `t` contient $[1, 0, 1]$ ou si `t` est vide, alors la fonction doit retourner 0.

   (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) Les trois lancers conduisent à 8 séquences possibles qui possèdent chacune la même probabilité de se produire, c'est-à-dire $\frac{1}{8}$. Trois de ces séquences (*[Pile ; Pile ; Face]*, *[Pile ; Face ; Pile]* et *[Face ; Pile ; Pile]*) contiennent deux occurrences du côté pile. La probabilité d'obtenir un résultat égal à 2 vaut donc $\frac{3}{8}$. La quantité d'information demandée est donc égale à

$$\log_2 \frac{8}{3} \approx 1{,}415 \text{ bit.}$$

   (b) Un téraoctet commercial contient 8.10^{12} bits d'information, et un térabit standard 2^{40} bits. La capacité réelle d'un disque dur vendu comme possédant 12 téraoctets est donc de

$$\frac{12.8.10^{12}}{2^{40}} \approx 87{,}311 \text{ Tb.}$$

2. (a) — *Valeur signée :* La représentation binaire non signée de 2025 sur 15 bits est 000 0111 1110 1001. En la préfixant du bit de signe 1, on obtient 1000 0111 1110 1001, qui se traduit par 87E9 en hexadécimal.

   — *Complément à un :* Le complément de la représentation non signée de 2025 sur 15 bits vaut 111 1000 0001 0110. En le préfixant du bit de signe 1, on obtient 1111 1000 0001 0110, c'est-à-dire F816 en hexadécimal.

   — *Complément à deux :* Le nombre étant négatif, cette représentation est décalée d'une position par rapport à celle par complément à un, et vaut donc F817 en hexadécimal.

   (b)

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

   Le résultat représente $-\frac{5}{8}$. Notons que le deuxième opérande a perdu un chiffre significatif, ce qui a conduit à l'arrondir de $\frac{7}{16}$ à $\frac{3}{8}$.

(c)

	1	1	1	1	0	0
×	1	1	1	1	0	0
	1	1	0	0		
+	1	0	0			
	0	1	0	0	0	0

Le résultat représente bien +16.

(d) Ce nombre est positif, donc le bit de signe de la représentation vaut 0. Avec une mantisse normalisée, on aurait un exposant égal à -130, ce qui n'est pas possible en simple précision. On est donc amené à utiliser une mantisse dénormalisée, ce qui fixe l'exposant à -127. Cet exposant est représenté par les 8 bits 00000000. La mantisse vaut alors

$$1{,}5\,\frac{2^{-130}}{2^{-127}} = \frac{3}{16},$$

ce qui se représente 00011000000000000000000. En combinant les trois parties de la réprésentation, on obtient donc finalement

$$00000000000011000000000000000000,$$

qui représente exactement 2^{-130}.

3. (a) La mémoire vive permet des modifications rapides et illimitées de son contenu, et ne préserve pas ce dernier quand l'ordinateur est éteint. Dans un ordinateur personnel, elle est par exemple utilisée pour mémoriser les programmes en cours d'exécution ainsi que les données manipulées par ces programmes. La mémoire morte, en revanche, ne permet pas de modifier son contenu pendant son fonctionnement normal, ou bien le permet de façon limitée. Ce contenu est préservé lorsque l'ordinateur est mis hors-tension. Ce type de mémoire est notamment utilisé pour retenir le programme chargé d'effectuer les premières opérations de l'ordinateur lorsque celui-ci est allumé ou redémarré.

(b) — Le mode d'adressage immédiat permet d'exprimer une opérande constante. *Exemple :* `MOV AL, 0x10` (où ce mode d'adressage est employé pour le second opérande) remplace le contenu du registre AL par 0x10.

— Le mode d'adressage registre indique une valeur devant être lue ou écrite dans un registre. *Exemple :* `MOV AL, 0x10` (où ce mode

d'adressage est employé pour le premier opérande) remplace le contenu du registre AL par 0x10.

- Le mode d'adressage direct indique une valeur devant être lue ou écrite en mémoire à une adresse donnée.
  Exemple : `MOV AL, byte ptr[100]` (où ce mode d'adressage est employé pour le second opérande) remplace le contenu du registre AL par l'octet lu depuis l'adresse 100 de la mémoire.

4. — Les deux premières instructions placent respectivement les constantes 0x1908 et 0x2025, représentées sur 64 bits, dans les registres RAX et RBX.

   — Les deux instructions suivantes empilent successivement ces deux valeurs (0x1908 puis 0x2025).

   — L'instruction `POP RAX` dépile 0x2025 et place cette valeur dans RAX. À ce stade, RAX et RBX contiennent tous deux 0x2025, et la pile contient 0x1908.

   — L'instruction `ADD` suivante ajoute à RAX l'entier de 64 bits situé en sommet de pile, c'est-à-dire 0x1908. RAX prend donc la valeur 0x392D.

   — L'instruction `POP RBX` dépile 0x1908 et place cette valeur dans RBX.

   — L'instruction `SUB RSP, 8` replace dans le pointeur de pile RSP la valeur qu'avait ce registre avant la précédente instruction POP. Cela a donc pour effet de replacer 0x1908 en sommet de pile.

   — L'instruction `XOR RBX, RAX` calcule le "ou exclusif" de 0x1908 et de 0x392D, c'est-à-dire 0x2025, et place cette valeur dans RBX.

   — L'instruction `MOV` suivante remplace l'entier de 64 bits situé en sommet de pile par 0x2025.

   — La dernière instruction dépile 0x2025 et place cette valeur dans RAX. La valeur finale de RAX est donc égale à 0x2025.

5. (a)
```
unsigned nb_changements(short t[], unsigned n)
{
  unsigned i, nb;

  nb = 0;
  i = 0;
  while (++i < n)
    if ((t[i - 1] >= 0 && t[i] < 0) ||
        (t[i - 1] < 0 && t[i] >= 0))
      nb++;

  return nb;
}
```

(b)
```
        .intel_syntax noprefix
        .text
        .global nb_changements
        .type nb_changements, @function
nb_changements:
        PUSH RBP
        MOV  RBP, RSP
        MOV  EAX, 0
        MOV  RCX, 0
boucle: MOV  RDX, RCX
        INC  ECX
        CMP  ECX, ESI
        JAE  fin
        CMP  word ptr[RDI + 2 * RDX], 0
        JL   suite
        CMP  word ptr[RDI + 2 * RCX], 0
        JGE  suite
incr:   INC  EAX
        JMP  boucle
suite:  CMP  word ptr[RDI + 2 * RDX], 0
        JGE  boucle
        CMP  word ptr[RDI + 2 * RCX], 0
        JL   boucle
        JMP  incr
fin:    POP  RBP
        RET
```