Organisation des ordinateurs
Examen de juin 2024
Énoncés et solutions

Énoncés

1. Une machine analyse des échantillons de roche pour détecter la présence d'un type spécifique de minéral, qui a une probabilité de 15% d'être présent. Toutes les 5 millisecondes, la machine envoie un signal à un ordinateur pour indiquer la présence ou l'absence de ce minéral.

   (a) Calculer la quantité d'information moyenne contenue dans un signal émis par la machine.

   (b) Quelle quantité moyenne de mémoire vive l'ordinateur doit-il allouer pour retenir les données transmises par la machine pendant 10 heures de mesures ininterrompues ?

2. (a) Quel sont les nombres entiers possédant la représentation hexadécimale 0xC49, selon les représentations
   — signée
   — par complément à un
   — par complément à deux
   sur 12 bits ?

   (b) Calculer la somme $(-\frac{5}{2}) + \frac{21}{8}$ en complément à deux en virgule fixe avec 3 bits avant et 3 bits après la virgule.

   (c) Calculer la somme $30 + (-18)$ en complément à un sur 6 bits.

   (d) En plus des représentations en simple et en double précision vues au cours, la norme IEEE754 définit également une représentation en demi-précision. Celle-ci est basée sur les mêmes principes, mais représente les nombres réels sur 16 bits, décomposés en 1 bit de signe, 5 bits pour l'exposant et 10 bits pour la mantisse.

   Quels sont le plus petit nombre positif différent de 0 et le plus grand nombre représentables de façon exacte en demi-précision ? On demande de donner ces nombres, ainsi que leur représentation dans ce format.

3. (a) Qu'est-ce que la mémoire morte, et à quoi sert-elle ? Donner un exemple de technologie de mémoire morte et en expliquer les spécificités.

(b) En quoi consiste l'adressage indirect indexé ? Donner un exemple d'instruction assembleur utilisant ce type d'adressage, et expliquer l'opération qu'elle effectue.

(c) Expliquer étape par étape comment se déroulera l'exécution du fragment de code assembleur x86-64 suivant. Quelle sera la valeur finale de `RAX` ?

```
MOV RAX, 0x101
MOV dword ptr[0x100], EAX
ADD word ptr[RAX - 1], AX
ADD AL, byte ptr[RAX]
XOR byte ptr[RAX - 1], AH
OR byte ptr[RAX], AH
MOV AX, word ptr[RAX - 1]
ADD byte ptr[RAX], 0x11
MOV AH, byte ptr[RAX]
```

4. On souhaite programmer une fonction `f` acceptant en arguments un pointeur vers un tableau d'octets `t` et la taille `n` de ce tableau. On suppose que `t` contient une séquence de caractères Unicode représentés correctement selon le procédé d'encodage UTF-8. La fonction `f` doit retourner le nombre de caractères contenus dans `t`.

   Par exemple, si `t` contient les 6 octets 0x31, 0x30, 0x20, 0xE2, 0x82 et 0xAC, représentant la séquence de caractères "10 €", alors la fonction `f` doit retourner 4.

   (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) Un signal indiquant la présence du minéral a une probabilité de 0,15 de se produire et apporte donc

$$\log_2 \frac{1}{0{,}15} \approx 2{,}734 \text{ bits}$$

d'information. Un signal d'absence présente quant à lui une probabilité d'occurrence de 0,85, et contient donc

$$\log_2 \frac{1}{0{,}85} \approx 0{,}234 \text{ bits}$$

d'information. La quantité d'information moyenne d'un signal vaut donc

$$0{,}15 \,.\, 2{,}734 + 0{,}85 \,.\, 0{,}234 \approx 0{,}601 \text{ bit.}$$

(b) Le nombre de signaux émis pendant 10 heures est égal à

$$200 \,.\, 3600 \,.\, 10 = 7{,}2 \,.\, 10^6.$$

On a donc au total

$$0{,}601 \,.\, 7{,}2 \,.\, 10^6 \approx 4390850 \text{ bits}$$

de données, ce qui nécessite une mémoire de 536 Koctets.

2. (a) Ce nombre possède la représentation binaire 1100 0100 1001. Il vaut donc

$-(2^{10} + 2^6 + 2^3 + 2^0) = -1097$ pour la représentation signée,

$-(2^9 + 2^8 + 2^7 + 2^5 + 2^4 + 2^2 + 2^1) = -950$ par complément à un,

$-2^{11} + 2^{10} + 2^6 + 2^3 + 2^0 = -951$ par complément à deux.

(b) Puisqu'il y a 3 bits après la virgule, cette opération revient à calculer la somme de $(-5/2) \,.\, 2^3 = -20$ et de $(21/8) \,.\, 2^3 = 21$ avec des entiers représentés par complément à deux sur 6 bits. On a :

	$\boxed{1}$	$\boxed{1}$	$\boxed{1}$			
	1	0	1	1	0	0
+	0	1	0	1	0	1
	0	0	0	0	0	1

Le résultat représente l'entier signé 1, qui correspond à $1/2^3 = 0{,}125$ en virgule fixe avec 3 bits avant et 3 bits après la virgule.

(c)

$\boxed{1}$	$\boxed{1}$	$\boxed{1}$	$\boxed{1}$			
	0	1	1	1	1	0
+	1	0	1	1	0	1
				$\boxed{1}$	$\boxed{1}$	
	0	0	1	0	1	1
+						1
	0	0	1	1	0	0

Le résultat représente bien l'entier 12.

(d) Dans la représentation en demi-précision, l'exposant appartient à l'intervalle $[-2^4 + 1, 2^4] = [-15, 16]$. Le plus petit nombre strictement positif représentable est donc obtenu avec un exposant égal à -15. Avec cet exposant, la mantisse est dénormalisée, et sa plus petite valeur différente de zéro vaut 2^{-9}. Le nombre est donc égal à $2^{-15} \,.\, 2^{-9} = 2^{-24}$.

Le plus grand nombre représentable de façon exacte possède un exposant égal à 15 et une mantisse (normalisée) égale à $2 - 2^{-10}$. Ce nombre vaut donc $2^{15} \,.\, (2 - 2^{-10}) = 2^{16} - 2^5 = 65504$.

3. (a) La mémoire morte est un composant de l'ordinateur qui sert à retenir des données qui ne sont pas amenées à être modifiées lors de son mode de fonctionnement nominal, et qui doivent être préservées quand l'ordinateur est mis hors tension. Comme exemple de technologie de mémoire morte, on peut citer la mémoire *OTP (One-Time Programmable)*, qui retient des bits d'information à l'aide de composants internes analogues à des fusibles, dont la valeur peut être fixée une fois pour toutes lors d'une opération de programmation de la mémoire.

(b) L'adressage indirect indexé consiste à spécifier l'endroit où se trouve un opérande en mémoire en écrivant une expression composée d'un registre de 64 bits, auquel on ajoute un déplacement sous la forme d'un autre registre pouvant être multiplié par un facteur 1, 2, 4 ou 8, et une base constante. La taille de l'opérande est également exprimée sous la forme d'un qualificateur `byte` (8 bits), `word` (16 bits), `dword` (32 bits) ou `qword` (64 bits). Par exemple, l'instruction

```
MOV AL, byte ptr [RBX + 2 * RCX + 4]
```

recopie dans `AL` l'octet dont l'adresse s'obtient en ajoutant à la valeur de `RBX` un déplacement égal à deux fois `RCX`, et une base égale à 4.

(c) Ce problème nécessite de faire l'hypothèse que toutes les cellules de la mémoire adressées par ce code correspondent à de la mémoire vive.

— `MOV RAX, 0x101` attribue à `RAX` la valeur 0x101, étendue sur 64 bits.

— `MOV dword ptr [0x100], EAX` écrit successivement en mémoire, à partir de l'adresse 0x100, des octets égaux à 0x01, 0x01, 0x00 et 0x00.

— `ADD word ptr[RAX - 1], AX` ajoute 0x101 représenté sur 16 bits à l'entier situé à l'adresse 0x100 de la mémoire. On a donc à présent à partir de cette adresse les octets 0x02, 0x02, 0x00 et 0x00.

— `ADD AL, byte ptr[RAX]` ajoute à `AL`, qui est à présent égal à 0x01, l'octet situé à l'adresse 0x101 de la mémoire, qui vaut 0x02. Le registre `AL`, qui correspond aux 8 bits de poids faible de `RAX`, prend donc la valeur 0x03.

— `XOR byte ptr[RAX - 1], AH` complémente le bit de poids faible de l'octet situé à l'adresse 0x102 de la mémoire, qui devient égal à 0x01.

— `OR byte ptr[RAX], AH` force à 1 le bit de poids faible de l'octet situé à l'adresse 0x103 de la mémoire, qui devient égal à 0x01.

— `MOV AX, word ptr[RAX - 1]` recopie dans `AX` les deux octets situés à l'adresse 0x102 de la mémoire. Le registre `AX`, qui correspond aux 16 bits de poids faible de `RAX`, prend donc la valeur 0x101.

— `ADD byte ptr[RAX], 0x11` ajoute 0x11 à l'octet situé à l'adresse 0x101 de la mémoire, qui devient donc égal à 0x13.

— `MOV AH, byte ptr[RAX]` recopie cet octet dans `AH`, qui correspond aux 8 bits situés entre les positions 8 et 15 du registre `RAX`.

En définitive, `RAX` contient donc 0x1301.

4. (a) On peut résoudre ce problème en parcourant les caractères contenus dans le tableau, et en détectant ceux qui commencent l'encodage un symbole Unicode représenté sur plusieurs octets. Pour chaque caractère lu, on incrémente un compteur et on passe au caractère suivant en se déplaçant du nombre approprié d'octets.

```
unsigned nb_caracteres(char t[], unsigned n)
{
  unsigned nb, i;

  for (i = 0, nb = 0; i < n; i++, nb++)
    if ((t[i] & 0b11100000) == 0b11000000)
      i += 1;
    else if ((t[i] & 0b11110000) == 0b11100000)
      i += 2;
    else if ((t[i] & 0b11111000) == 0b11110000)
      i += 3;

  return nb;
}
```

(b)
```
        .intel_syntax noprefix
        .text
        .global nb_caracteres
        .type nb_caracteres, @function
nb_caracteres:
        MOV  RCX, 0
        MOV  RAX, 0
boucle: CMP  RCX, RSI
        JAE  fin
        MOV  DL, byte ptr[RDI + RCX]
        INC  RAX
        INC  RCX
        MOV  DH, DL
        AND  DH, 0b11100000
        CMP  DH, 0b11000000
        JNE  suite1
        INC  RCX
        JMP  boucle
suite1: MOV  DH, DL
        AND  DH, 0b11110000
        CMP  DH, 0b11100000
        JNE  suite2
        ADD  RCX, 2
        JMP  boucle
suite2: MOV  DH, DL
        AND  DH, 0b11111000
        CMP  DH, 0b11110000
        JNE  boucle
        ADD  RCX, 3
        JMP  boucle
fin:    RET
```

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