Examen d'Architecture et Système

19 janvier 2018

Durée : 3 heures.

1 Arithmétique des entiers

Dans ce qui suit, on traite des entiers non-signés ; on suppose que les `short` ont 16 bits, les `int` ont 32 bits et les `long` en ont 64. On notera les valeurs hexadécimales avec le préfixe `0x`.

Soient `x` et `y` deux `short`.

(a) Donnez la représentation binaire de `x` et `y` si `x=46` et `y=26`.

Solution: $x = 32 + 8 + 4 + 2 = (101110)_2$ et $y = 16 + 8 + 2 = (011010)_2$

(b) Que donne l'expression `x & y`, où `&` est le et binaire ?

Solution: Les seuls bits présents dans les deux sont 8 et 2, du coup le resultat en est $(001010)_2 = 10$.

(c) Soit `x` $= k$, où $k \in [0, 255]$. Du coup le premier octet de `x` est 0, le deuxième k. Donnez un expression de la forme `x` $* c$, avec une constante c, tel que le résultat est un `short` dont les deux octets sont k.

Solution: $c = 256 + 1 = 257 = (0101)_{16}$

(d) Soit `x=0x1234`. Que donne `x % 15`, où `%` est l'opérateur modulo ? En général, que fait une expression `x %` n, où $n = 2^k - 1$, pour un $k \geq 1$? Donnez une justifaction mathématique.

Solution: x peut être représenté sous une forme $x = \sum_{i=0}^{\infty} x_i \cdot (n+1)^i$. Puisque le modulo se distribue par rapport à $+$ et $\cdot$ on obtient

$$\begin{aligned} x \bmod n &= \sum_{i=0}^{\infty} x_i \cdot (n+1)^i \\ &= \left(\sum_{i=0}^{\infty} (x_i \bmod n) \cdot ((n+1) \bmod n)^i \right) \bmod n \\ &= \left(\sum_{i=0}^{\infty} x_i \right) \bmod n \end{aligned}$$

Dans le cas de `0x1234 % 15` on obtient donc $(1+2+3+4) \bmod 15 = 10$.

(e) Soit `z` un `int` dont la valeur est dans $[0, 15]$. Donnez une expression avec trois opérations pour calculer le nombre d'uns dans la représentation binaire de `z`, en vous appuyant sur les résultats précédents.

Solution: `z` possède 4 bits significatifs, disons $(abcd)_2$. Avec les énoncés précédents, on possède tous les éléments pour la solution :

- D'abord on duplique $abcd$ plusieurs fois (en multipliant).
- Ensuite, on séléctionne certain bits (avec "et" logique) tel que les bits séléctionnés sont équidistants et qu'on séléctionne exactement une copie de a, de b, de c et de d.
- Ensuite on utilise le modulo pour faire la somme $a+b+c+d$.

Il y a plusieurs solutions concrètes, en voici un exemple :

```
((x * 0x111) & 0x249) % 7
```

Résultat après multiplication : $(abcdabcdabcd)_2$
Après "et logique" : $(00c000b00a00d)_2$
Résultat final : $(a+b+c+d) \bmod 7 = a+b+c+d$

2 Détection et correction d'erreurs

Un problème d'ingénierie des ordinateurs sont les *fuites de mémoire* : un bit dans la mémoire peut changer sa valeur de façon imprévue dû à des fautes éléctriques. On s'intéresse aux méthods pour détecter et (automatiquement) corriger ces erreurs. On va supposer que les erreurs sont rares et que, dans un mot binaire, au plus deux bits sont erronés.

La *parité* d'un mot est 1 si dans sa représentation binaire le nombres d'uns est impair, sinon 0. Autrement dit, un mot avec son bit de parité possède un nombre pair d'uns. Si n'importe quel bit (en inclus celui de parité) change de valeur, cette dernière propriété ne tient plus ce qui permet de détecter la présence d'une erreur.

(a) En analogie avec la question précédente, trouvez une expression avec quatre opérations pour calculer la parité d'un octet avec des opérations sur les `long`.

Solution: La solution est analogue à celle de la Question 1 (a) : on compte le nombre d'uns, puis on teste si ce nombre est impair (avec modulo ou "et logique"). Dans l'exemple suivant on crée cinq copies et séléctionne les bits sur les positions dont l'indice est un multiple de 5 :

```
(((x * 0x0101010101) & 0x0842108421) % 31) & 1
```

Un bit de parité ne permet pas de détecter si deux bits sont erronés. Pire, en cas d'une seule erreur on détecte sa présence mais sans savoir quel bit est fautif. Notre objectif est (i) d'identifier le bit fautif dans le cas d'une seule erreur, et (ii) de détecter la présence de deux erreurs (sans identifier de quels bits il s'agit). Pour cela, on étudie les *codes de Hamming*.

Dans un mot de Hamming, on mélange les bits de données et de parités. La position 0 est inutilisée, les positions 1,2,4,8,... sont utilisées pour des bits de parité, les autres pour des bits de données. Un bit de parité sur position p donne la parité des bits sur les positions x où p figure dans la représentation binaire de x (du coup, le bit sur position 1 compte la parité des positions impaires, celui de 2 la parité des positions 2,3,6,7,... etc.) Dans 15 bits, on code donc 11 bits de données et 4 bits de parité.

(b) Complétez les bits de parité dans le mot suivant:

position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
bit	1	0	1	1	1	0	1		0	0	1		1		0

Solution: De gauche à droite, 1, 0 et 1.

(c) Dans les mots suivants, existe-t-il une erreur ? Si c'est le cas, comment identifier la position du bit fautif ?

position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
bit	0	1	1	1	0	0	0	1	1	1	0	1	1	0	1
bit	1	1	0	1	1	1	1	1	0	1	0	0	0	0	1

Solution: Dans le premier mots, tous les bits de parité sont corrects.
Dans le deuxième mot, les bits aux positions 8 et 2 sont fautifs et ceux auz positions 4 et 1 sont corrects. En supposant qu'il s'agit d'une seule erreur, on obtient la position fautive ainsi :

- Puisuqe le bit 8 est fautif, l'erreur doit être parmi les positions de 8 à 15.
- Puisque le bit 4 est correct, l'erreur ne peut être parmi les positions 12 â 15, il en reste de 8 à 11.
- En appliquant le même raisonnement sur les bits 2 et 1, on limite le champs de recherche à la position 10.

(d) Le schéma proposé ne permet pas encore de détecter la présence de deux erreurs. Pourquoi ? Proposez une amélioration qui le permettra.

Solution: Le schéma proposé ne permet pas de distinguer le cas d'une seule erreur de deux erreurs. Par exemple, dans le second mot de (c), il se pourrait que les deux bits 8 et 2 sont fautifs (au lieu du seul bit de 10).

Pour corriger le problème, il convient de rajouter un bit de parité pour le mot entier sur la position numéro 0 (qui elle n'est pas pris en compte par les autres bits de parité). Dans ce cas :

- Si deux erreurs interviennent, le bit de parité 0 reste "correct". Par contre au moins une des erreurs concernera les bits 1 à 15, et du coup au moins un des autres bit de parité indiquera une erreur.
- Si une seule erreur intervient, le bit de parité 0 sera incorrect et les autres bits de parité en indiqueront la position (entre 0 et 15).
- Évidemment, sans aucune erreur, tous les bits de parité seront corrects.

Attention, il ne suffit pas d'introduire un bit de parité qui compterait uniquement les autres bits de parité, ou uniquement les données !

3 Comportement bizarre d'un programme

On considère le programme C suivant.

```
int c=6, d=7;
void foo() {
   int a[1];
   a[c] += d;
}
void main() {
   int x;
   x = 0;
   foo();
   x = 1;
   printf("x=%d",x);
}
```

Sur certains ordinateurs, ce programme affiche `x=1`, sur d'autres `x=0`.

(a) Comment expliquer cette différence ?

Solution: L'instruction `a[c] += d` dépasse les limites de `a`. Puisque `a` est local, il est alloé sur la pile, et l'instruction modifie une autre adresse sur la pile. Sachant que l'adresse de retour se trouve ...

(b) Si le programme répond 0, que nous enseignent les valeurs de `c` et `d` sur l'emplacement de certains données dans la mémoire ?

(c) Si le programme répond `x=1` sur votre ordinateur, expliquez (en bref) quels outils vous utiliseriez pour obtenir les "bonnes" valeurs pour `c,d` qui provoquent la réponse `x=0`.

4 Sémaphores

On simulera le comportement d'un petit train automatique qui bascule entre deux stations A et B avec une capacité de 20 passagers maximum. Il y a des passsagers qui veulent voyager de A à B ou de B à A. A chaque arrêt, les passagers disposent de 30 secondes pour entrer ou sortir par une seule porte. Après avoir fait cinq allers-retours, le train rentre dans sa remise sans passagers.

Dans notre simulation, le train et chaque passager est un thread. Il existe deux fonctions `passagerAB` et `passagerBA` qui vont être instantiés un nombre illimité de fois et à n'importe quel moment. Un ne regarde que `passagerAB`, l'autre étant analogue. Un thread passager se termine après avoir fait son voyage.

Le code (incomplet) du train et du premier type de passager est comme suit :

```
void train() {
  for (int i=0; i<5; i++) {
    println("je suis à A");
    sleep(30);
    println("je démarre vers B");
    println("je suis à B");
    sleep(30);
    println("je démarre vers A");
  }
}

void passagerAB() {
  println("j'attends l'arrivée du train à A");
  println("j'entre dans le train");
  println("j'attends l'arrivée du train à B");
  println("je sors du train")
}
```

(a) Complétez le code en rajoutant des sémaphores et leurs opérations pour assurer le bon transport des passagers. (Dans un premier temps on va supposer que la boucle `for` ne termine pas.) Pour les sémpahores, on utilisera du pseudo-code (`init`(s, n), `wait`(s), `post`(s)). Attention, un passager doit être sûr de bien entrer à A et de sortir à B. Les sémaphores seront le seul moyen de communication entre le train et les passagers.

(b) Dans un deuxième temps, étendez le code pour être sûr que le train peut rentrer dans sa remise *sans passager*.

5 Codage de caractères

Un mail consiste des *entêtes* qui donnent le sujet, la date, l'auteur, le destinataire etc et le message propre. Le protocole pour l'échange des mails sur Internet exige que les entêtes ne contiennent que des caractères ASCII (codes de 0 à 127). Pour écrire des sujets avec d'autres caractères on trouvera donc des lignes telle que :

```
Subject: =?iso-8859-15?Q?Conf=E9rence?=
```

Il s'agit d'un *mot encodé MIME* où un caractère non-ASCII est codé en hexadécimal et puis interpreté selon un standard donné (dans ce cas, ISO-8859-15 ce qui donnerait é). Une autre méthode s'appelle Base64, indiqué par un `B` au lieu du `Q`. Elle n'utilise que les caractères ASCII; par exemple tous les majuscules (à partir de position 65), tous les minuscules (à partir de position 97), les chiffres 0..9 (à partir de position 48), et deux autres, p.ex. `+` (43) et `/` (47).

(a) Expliquez comment une telle idée peut être utilisée pour coder n'importe quelle chaîne de caractères.

(b) Donnez l'encodage selon votre idée du mot *ému*.