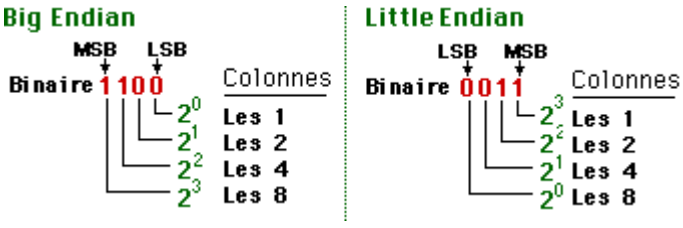
La mémoire d'un ordinateur est une longue suite de bits (interrupteurs qui peuvent être 0 ou 1). Ces bits sont regroupés en octets (groupe de 8 bits). Les octets sont regroupés en mots (16 bits), long mots (32 bits), quadruple mots (64 bits) et de plus grands regroupements encore.

Imaginez que j'écris la valeur 12 (décimal) en binaire. Cela s'écrit par un 1 dans la colonne des 8, un 1 dans la colonne des 4, un 0 en dans la colonne des 2 et un 0 dans la colonne des 1. Si on additionne, cela donne 12. La colonne des 8 représente le bit le plus significatif (MSB, Most Signifiant Bit), puisqu'elle renferme la plus grande valeur; et la colonne des 1 représente le bit le moins significatif (LSB, Least Signifiant Bit), puisqu'elle renferme la plus petite valeur.

Et bien on peut écrire en binaire en utilisant l'une ou l'autre de ces manières :



Vous noterez que la valeur demeure la même, peut importe la manière d'écrire le binaire. Si on fait la lecture des bits du moins significatif au plus significatif, la manière dont le chiffre binaire a été écrit n'importe pas. Elle importe uniquement pour les structures internes (d'une puce) -- tant que celle-ci demeure conséquente, le type «d'Endian» n'importe pas au monde extérieur. Pour ces raisons, lorsque quelqu'un discute du type «d'Endian», il parle de l'arrangement des octets et non pas de l'arrangement des bits.

La plupart des gens exprime la forme binaire comme nous exprimons la forme décimale, le chiffre (bit) le plus significatif à gauche.

Malheureusement, la manière (Endian) utilisée pour exprimer la valeur d'un octet à l'intérieur d'un groupe (mot - 16 bits, long mot - 32 bits, et ainsi de suite) a un effet certain sur le résultat.

L'arrangement des octets est le sujet de discussion des gens lorsqu'ils parlent du problème Endian.

Imaginez deux processeurs (32 bits) qui arrangent les octets différemment. Rappelez-vous, un octet est un regroupement de 8 bits, et peut représenter un caractère ASCII (technique de codage standard pour les lettres de l'alphabet et les symboles). Maintenant imaginez ce qui se passe lorsque ces deux processeurs traitent les même données --

Le processeur «Big Endian» emmagasinera les données dans un ordre (direction). Le problème c'est que l'autre processeur lisant ces données utilise les groupements dans un ordre différent. Ainsi un des processeurs sortirait la donnée comme ceci : U-N-I-X, tandis que l'autre la sortirait ainsi X-I-N-U; en admettant que ce sont des processeurs 32 bits (lesquels regroupent les caractères en paquets de 4). Gros problème.

Les anciens processeurs étaient des processeurs 16 bits -- lorsque ce problème est apparu pour la première fois. Dans ces conditions, le premier processeur aurait sorti 'UN' et 'IX', soit deux groupes de 16 bits, et l'autre processeur aurait sorti 'NU' et 'XI' -- ou 'NUXI'. Voilà pourquoi certains programmeurs UNIX appellent le problème Endian, «Le problème NUXI».

Ce problème ne s'applique pas uniquement aux chaînes de caractères, mais également aux valeurs binaires. Rappelez-vous que plusieurs valeurs sont formées par des combinaisons d'octets. Un octet renferme une valeur de 0-255 (ou 256 valeurs). Avec deux octets (16 bits au total), la valeur représentée est de 0-65535 (256 x 256 valeurs). Cependant un des octets est le moins significatif et représente une fois sa valeur, l'autre octet est le plus significatif et représente 256 fois sa valeur. Alors sur un ordinateur

dont la valeur du MSB est de 50 et celle du LSB de 10 -- ce qui représente 50 x 256 + 10 = 12,810 -- celles-ci seraient dans un autre ordre sur un autre ordinateur avec le LSB valant 50 et le MSB 10 -- ce qui représente 10 x 256 + 50 = 2,610. Toujours le même problème.

Supposons-nous écrivons un nombre entier de quatre bytes long 67305985.en hexadécimal cela ferai 0x04030201, le byte le plus significative serai 04 (msb) et le byte le moins significative serai 01 (lsb).

Supposons qu'ils sont notés dans une case mémoire noté x. ces valeur seront toutes notés consécutivement dans les adresses noté x à x+3. on peut se demander quel byte de données entre dans quel endroit de la mémoire ?cela dépend du processeur.

Exemple

processeur Sun et Motorola (big-endian).

x	04
x+1	03
x+2	02
x+3	01

Little-Endian le systeme enregistre le Lsb en premier. Exemple Intel x86 family, Vaxes, Alphas (little endian).

x	01
x+1	02
x+2	03
x+3	04

Ce terme est rarement utilisé et concerne un ordonnancement anormal. Les arrangements normaux sont Big Endian (4-3-2-1) ou Little Endian (1-2-

3-4). Imaginez un processeur 32 bits qui se comporte comme deux processeurs 16 bits -- il pourrait emmagasiner les données selon un ordre 3-4-1-2 ou 2-1-4-3. C'est ce qu'on appelle le Middle-Endian pour un processeur 32 bits. Certains mini-ordinateurs utilisent ce format pour représenter les décimaux compactés -- mais c'est rarement utilisé.

Faisons le même teste que precedement :

x	03
x+1	04
x+2	01
x+3	02

EXEMPLE :

Fonction permettant l’exportation de donnée de little endian à big endian :

Extait de code personnel:

```
float floatSwap( float f ){
```

```
  union {       //            sa sert a faire un cast du float
    float f;
    unsigned char b[4];
  } dat1, dat2;

  dat1.f = f;
  dat2.b[0] = dat1.b[3];
  dat2.b[1] = dat1.b[2];
  dat2.b[2] = dat1.b[1];
  dat2.b[3] = dat1.b[0];
  return dat2.f;
}
//On lis le fichier binaire avec cette commande :
fread (&p_object->mapcoord[i].u, sizeof (float), 1, l_file);

p_object->mapcoord[i].u=floatSwap(p_object->mapcoord[i].u);
    //Récupère un float qu'il range dans la variable p_object->mapcoord[i].u qui a besoin //
    d'une coordonner floatant et donc sur une sun le floatant est lu a l'envers ce qui donne //
    une valeur erroné dans la variable p_object->mapcoord[i].u

static unsigned int getint(fp)  //permet le decalage de byte (ici int 4 octets)
    FILE *fp;
{
  int c, c1, c2, c3;

  // get 4 bytes
  c = getc(fp);
  c1 = getc(fp);
  c2 = getc(fp);
  c3 = getc(fp);

  return ((unsigned int) c) +
    (((unsigned int) c1) << 8) +
    (((unsigned int) c2) << 16) +
    (((unsigned int) c3) << 24);
}

static unsigned int getshort(fp) //meme chose sauf ici short =2octets
    FILE *fp;
{
  int c, c1;
```

```
  //get 2 bytes
  c = getc(fp);
  c1 = getc(fp);

  return ((unsigned int) c) + (((unsigned int) c1) << 8);
}
```

Il n'y a pas de solution miracle au problème de l'arrangement des octets (Endian). Chacun doit s'entendre sur le format d'emmagasinage des données. Un des processeurs aura à traduire (changer l'ordre) des données provenant de l'autre processeur.

Certains nouveaux processeurs (tel que le PowerPC) peuvent être Bi-Endian (ils peuvent utiliser un format ou l'autre) -- mais habituellement le système d'exploitation qui les utilise est dépendant d'un certain type «d'Endian». Alors ils sont fixés sur une méthode et rarement vont-ils utiliser l'autre.

Quelques faits : les processeurs Intel (x86 et Pentium) sont Little Endian et les processeurs Motorola (la série 680x0) sont Big Endian. Le MacOS est Big Endian et Windows est Little Endian.

Source : http://www.gamedev.net/reference/articles/article2091.asp