université angers
FACULTÉ DES SCIENCES
Unité de formation et de recherche

Département Informatique
L3 Mention Informatique
Jean-Michel Richer
Architecture des Ordinateurs
jean-michel.richer@univ-angers.fr
2015/2016

Travaux Dirigés

1 Représentation de l'information

En informatique, on doit pouvoir représenter des nombres entiers naturels ou relatifs ainsi que des nombres à virgule. On distingue au moins 4 types de représentations :

- la représentation **binaire naturelle** utilisée pour représenter les entiers naturels,
- la représentation **binaire signée**
- la représentation **binaire en complément à deux** utilisée pour représenter les entiers relatifs
- la représentation en **virgule flottante** utilisée pour représenter les réels

1.1 Représentation binaire naturelle

En informatique les nombres sont codés en base :

- 2 (binaire),
- 8 (octal),
- 10 (décimal),
- 16 (héxadécimal)

Un nombre N exprimé en base B a pour valeur décimale :

$$N = \sum_{i=k_1}^{i=k_2} x_i * B^i$$

où les x_i sont des coefficients prenant leurs valeurs dans l'intervalle $0, 1, \ldots, B-1$.

Exercice 1 - Trouvez l'équivalent décimal des nombres suivants :

- 101010_2, 10011_2
- 201_3, 1111_3
- 421_8, 732_8
- $A0_{16}$, FF_{16}

Exercice 2 - Convertir les nombres décimaux suivants

- 11 et 10 en base 2
- 26 et 210 en base 8
- 250 et 49 en base 16

Exercice 3 - Définir un procédé permettant de passer rapidement du binaire à l'hexadécimal. Prendre par exemple le nombre suivant : `1011.1001.1101.0001`

Exercice 4 - Réaliser la somme des nombres naturels suivants en base 2. Que remarquez-vous ?

- 0000.0010_2 + 0000.0011_2
- 0000.1010_2 + 0000.1111_2

Exercice 5 - Quels sont les plus grands entiers naturels que l'on peut représenter avec 8, 16 ou 32 bits ?

1.2 Représentation signée

En représentation **binaire signée** le bit de poids le plus fort (c'est à dire, le bit le plus à gauche) indique le signe du nombre. On utilise 0 pour indiquer que le nombre est positif et 1 pour un nombre négatif. Par exemple

$$0111.1111_2 \quad \text{représente} \quad 127_{10}$$
$$1111.1111_2 \quad \text{représente} \quad -127_{10}$$

Exercice 6 - Calculer la somme des nombres signés suivants. Que remarquez vous ?

- 0000.0111_2 + 0000.0101_2
- 0000.0111_2 + 1000.0101_2

1.3 Représentation en complément à deux

La représentation en complément à deux représente les nombres négatifs différemment (les nombres positifs ne changent pas). Pour convertir un nombre négatif, on procède comme suit :

- représenter le nombre sous forme positive en binaire,
- complémenter chaque bit (0 est transformé en 1 et 1 en 0),
- ajouter 1 au résultat précédent.

Exercice 7 - Donner la représentation en complément à deux des nombres décimaux suivants : $-1, -2, -127, -128, -129$. Combien de nombres peut-on représenter avec 8 bits en notation en complément à deux ?

Exercice 8 - Calculer la somme des nombres en complément à deux suivants. Que remarquez vous ?

- 0000.0111_2 + 0000.0101_2
- 0000.0111_2 + 1000.0101_2
- 0000.0011_2 + 1111.1011_2
- 0100.0000_2 + 0100.0001_2

Exercice 9 - Calculer le **produit** des nombres en complément à deux suivants. Que remarquez vous ?

- 7×5
- 7×-5
- 48×-2
- 48×-3

Exercice 10 -

- comment *multiplier* simplement un nombre binaire par 2, 4, 8 ou 2^n ?
- comment *diviser* simplement un nombre binaire par 2, 4, 8 ou 2^n ?

1.4 Représentation en virgule flottante

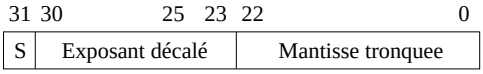

FIGURE 1 – Norme IEEE 754

Exercice 11 - Représentez $133,875_{10}$ en norme IEEE 754. Pour ce faire procédez par étapes :
- transformez $133,875$ en base 2.
- normalisez le résultat en supprimant le premier chiffre 1
- ajoutez 127 à l'exposant
- codez le résultat final sur les 32 bits de la norme IEEE

Exercice 12 - Trouvez à quel nombre réel correspond la représentation IEEE 754 : $42\,C8\,40\,00_{16}$

N	2^N
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

N	2^N
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768

N	2^N
16	65536
20	1.048.576
31	2.147.483.648
32	4.294.967.296

TABLE 1 – Puissances de 2

Unité	Symbole	Nombre d'octets
kilo octet	ko	2^{10}
méga octet	Mo	2^{20}
giga octet	Go	2^{30}
téra octet	To	2^{40}

TABLE 2 – Unités utilisées en Informatique

Exercice 13 - Dans la suite on supposera que :

- les entiers sont modélisés sur 4 octets
- les caractères sont modélisés sur 1 octet
- la procédure appelante libère les paramètres de la pile
- les résultats des fonctions sont retournés dans le registre EAX

1. traduire en assembleur x86 le code C suivant où `i1` et `i2` représentent des séries d'instructions quelconques (on supposera que les variables `a` et `b` sont des variables entières.

```
int a, b;

if (a <= b) {
  i1;
} else {
  i2;
}
```

2. traduire en assembleur x86 le code C suivant (on supposera que les variables `i`, `n` et `sum` sont des variables entières.

```
sum=0;
for (i = 0; i < n; i++) {
  sum = sum + i;
}
```

3. traduire en assembleur x86 le code C suivant :

```
sum=0;
while ((i > 10) && (i <= n)) {
  sum = sum + i;
  ++i;
}
```

4. traduire en assembleur x86 le code C suivant :

```
sum=1;
while ((i  % n) ==0) || (i == 3)) {
  sum = sum * i;
  ++i;
}
```

Exercice 14 - Traduire la procédure suivante en assembleur x86.

```
void init( char t[], int n) {
  int i;
  for (i = 0; i < n; i++) {
    t[i] = '0';
  }
}
```

- donner une traduction assembleur de la procédure
- donner des versions optimisées en utilisant LOOP, puis STOSB

Exercice 15 - Traduire la procédure suivante en assembleur x86.

```
void init( int t[], int n) {
  int i;
  for (i = 0; i < n; i++) {
    t[i] = 0;
  }
}
```

- donner une traduction mot à mot de la procédure
- donner des versions optimisées en utilisant `LOOP`, puis `STOSD`

Exercice 16 - Traduire la fonction suivante en assembleur x86 :

```
int f( int X, int Y, int Z )
{
  if (Z ==0) {
    return ((X+Y)*(X-Y))/((X/Y)*3+(X+Y)*(X-Y));
  } else {
    return (X-Z+Y)*(X+Z+Y)*(1-Z*Z);
  }
}
```

Exercice 17 - Comment multiplier efficacement sur un 8086, la valeur contenue dans le registre AX par 10 sans utiliser l'instruction `mul` ? On rappelle que `mul r16` utilise 118 à 133 cycles sur un 8086.

Exercice 18 - Écrire le code assembleur x86 de la fonction de Fibonacci :

```
int fib( int n ) {
  if (n <= 1) return n;
  else return fib(n-1) + fib(n-2);
}
```

Exercice 19 - Écrire le code assembleur x86 de la fonction de Fibonacci optimisée :

```
int fib_iter( int a, int b, int count ) {
  if (count == 0) return b;
  else return fib_iter(a+b,a,count-1);
}
```

L'appel de la procédure se fait par : `fib_iter(1,0,n)` Implanter et tester ces 2 procédures et notez les résultats comparatifs entre les 2 versions pour $n = 40$ à 60.

Exercice 20 - Ecrire un programme assembleur nasm pour processeur x86 et interfacé avec le langage C qui récupère les arguments en ligne de commande du programme et les affiche en utilisant la fonction `printf` du langage C.

Exercice 21 - Ecrire un programme en langage C qui étant donné un tableau de `MAX` entiers, réalise l'initialisation du tableau par des valeurs aléatoires comprises entre 0 et 20, puis affiche à l'écran la somme des valeurs par appel à une fonction assembleur :

```
int sum(int nbr,int tab[])
```

La fonction `sum` prend en paramètres le nombre de valeurs du tableau et l'adresse du tableau.

Exercice 22 - On désire écrire en assembleur la fonction suivante sans utiliser d'instruction de branchement :

```
int mini(int p, int q) {
  return (p < q) ? p : q;
}
```

pour cela, on va utiliser une variable temporaire r telle que :

$$r = (p - q) >> 31 = \begin{cases} 0, \text{si} & p > q \\ 1, \text{si} & p < q \end{cases}$$

on soustrait q à p et on garde le bit le plus significatif que l'on propage (bit 31 = bit de signe, utiliser l'instruction sar r32,imm8). A partir de ce moment, il suffit de calculer : $(p \wedge r) \vee (q \wedge \neg r))$.

Exercice 23 - Réaliser le dépliage de boucle par 4 (loop unrolling) sur le code C suivant :

```
void f(int *v, int *w, int k, int n) {
  for (int i=0; i<n; ++i) {
    v[i] += w[i] * k;
  }
}
```

Traduire en assembleur, puis donner une version en utilisant les instructions SSE suivantes dont on aura pris soin de consulter la documentation afin de comprendre leur comportement :

- movd
- pshufd
- pmulld
- paddd
- movdqu
- movdqa

Exercice 24 - Traduire en assembleur x86 version `64 bits` le code C suivant qui correspond au produit de deux matrices carrées de dimension `N` (constante) :

```
void prod(int a[N][N], int b[N][N], int c[N][N]) {
  int i,j,k;
  for (i=0;i<N;i++) {
    for (j=0;j<N;j++) {
      int sum=0;
      for (k=0;k<N;k++) sum+=a[i][k]*b[k][j];
      c[i][j]=sum;
    }
  }
}
```

Exercice 25 - Donner le code assembleur du calcul suivant :

$$\frac{\sqrt{X+Y}+Y^2+tan(\frac{1}{X-Y})}{(X+Y)\times sin(X-Y)}$$

Exercice 26 - Ecrire une version SSE du calcul de la suite de Fibonacci.

Exercice 27 - Coder la fonction suivante en assembleur classique puis donner une version qui utilise les registres SSE.

```
int chr_replace(char *src, char *dst, int n, char c, char d) {
  int changes=0;
  for (int i=0; i < n; ++i) {
    if (src[i] == c) {
      dst[i] = d;
      ++changes;
    } else {
      dst[i] = src[i];
    }
  }
  return changes;
}
```

Exercice 28 - Démontrez à l'aide de tables de vérité, les équivalences suivantes :

(a) $\overline{XYZ} = \bar{X} + \bar{Y} + \bar{Z}$
(b) $X + YZ = (X + Y)(X + Z)$
(c) $X + \bar{X}Y = X + Y$

Exercice 29 - Démontrez algébriquement les égalités suivantes :

(a) $\bar{Y}Z + Y\bar{Z} + YZ + \bar{Y}\bar{Z} = 1$
(b) $AB + A\bar{B} + \bar{A}\bar{B} = A + \bar{B}$
(c) $\bar{A} + AB + A\bar{C} + A\bar{B}\bar{C} = \bar{A} + B + \bar{C}$
(d) $A\bar{B} + \bar{A}\bar{C}\bar{D} + \bar{A}\bar{B}D + \bar{A}\bar{B}C\bar{D} = \bar{A}\bar{C}\bar{D} + \bar{B}$
(e) $XY + \bar{X}Z + YZ = XY + \bar{X}Z$
(f) $X + \bar{X}Y = X + Y$

Exercice 30 - Simplifiez les expressions suivantes :

(a) $ABC + AB\bar{C} + \bar{A}B$
(b) $(\overline{A + B})(\bar{A} + \bar{B})$
(c) $(A + \bar{B} + A\bar{B})(AB + \bar{A}C + BC)$
(d) $X + Y(Z + \overline{X + Z})$
(e) $\bar{W}X(\bar{Z} + \bar{Y}Z) + X(W + \bar{W}YZ)$

Exercice 31 - Soient deux fonctions booléennes E et F de trois variables dont les tables de vérité sont données. Exprimez E et F en fonction de X, Y, Z et simplifiez ces expressions.

X	Y	Z	$E(X,Y,Z)$	$F(X,Y,Z)$
0	0	0	1	0
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	0	1
1	1	1	0	1

Exercice 32 - Simplifiez les fonctions suivantes à l'aide d'un tableau de Karnaugh

(a) $F(X, Y, Z) = (1, 3, 6, 7)$
(b) $G(X, Y, Z) = (0, 3, 4, 5, 7)$
(c) $H(A, B, C, D) = (1, 5, 9, 12, 13, 15)$

Exercice 33 - Implantez les circuits suivants avec des portes NAND. Peut-on les simplifier et pourquoi ?

(a) $W\bar{X} + WXZ + \bar{W}\bar{Y}\bar{Z} + \bar{W}X\bar{Y} + WX\bar{Z}$
(b) $XZ + XY\bar{Z} + W\bar{X}\bar{Y}$

Exercice 34 - Imaginer un moyen d'échanger deux variables A et B sans utiliser une troisième variable ou un registre intermédiaire (pensez à l'instruction XOR).

Exercice 35 - Soit une machine dotée d'un bus d'adresses de 10 bits et d'un bus de données de 8 bits. Cette machine dispose également d'une mémoire cache à accès direct de 8 entrées.

1. quelle taille mémoire le processeur peut-il adresser ?
2. quelle est la taille du cache ?
3. on accède successivement aux adresses suivantes :

Adresse	Donnée
0000000000	D_1
0000010100	D_2
0000110110	D_3
0001110000	D_4
0000001100	D_5
0000100100	D_6

Adresse	Donnée
0000111000	D_7
0100110010	D_8
0100111000	D_9
1100110111	D_{10}
0010110011	D_{11}
1100110010	D_{12}

Déterminer l'état du cache.

Exercice 36 - On considère le problème de remplacement d'une entrée de la mémoire cache pour un cache *fully associative*. On utlise l'algorithme LRU (*Least Recently Used*) qui consiste à remplacer la ligne utilisée la moins récemment. Sa mise en oeuvre utilise une file qui contient en tête le numéro de l'entrée du cache correspondant à l'élément le plus récemment utilisé. La ligne qui devra être remplacée est la ligne en queue de file. Si on accède à une donnée présente dans le cache, le numéro de l'entrée correspondante est déplacée en tête de file.

Soit le cache suivant (on fait abstraction des valeurs des adresses) :

Entrée	Donnée
0	a
1	b
2	c
3	d

Simuler le fonctionnement de cet algorithme pour les accès successifs à des valeurs dans les deux cas suivants :

- *Cas 1 :* a,b,c,d,e,b,c,a,f,a,b,c
- *Cas 2 :* a,b,c,d,e,a,b,c,d,f,a,b,c,d

Que remarquez-vous ?