université angers
FACULTÉ
DES SCIENCES
Unité de formation et de recherche

Département Informatique
L3 Mention Informatique
Jean-Michel Richer
Architecture des Ordinateurs
2022/2023

jean-michel.richer@univ-angers.fr

Travaux Dirigés

1 Représentation de l'information

1.1 Représentation binaire naturelle

En informatique les nombres sont codés en base :

— 2 (binaire) _b

— 8 (octal) _o

— 10 (décimal) _d

— 16 (héxadécimal) _h

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

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

où les x_i sont des coefficients prenant leurs valeurs dans l'intervalle $0, 1, \ldots, B-1$ et k_1 et k_2 étant des entiers relatifs.

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

- 101010_b, 10011_b
- 201_3, 1111_3
- 421_o, 732_o
- $A0\_h$, FF_h

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 - Utilisez la méthode par complémentation afin de coder en notation binaire naturelle non signée, les nombres suivants :

- 249
- 1011
- 16373

- 131069

Exercice 4 - Définir un procédé permettant de passer rapidement du binaire à l'hexadécimal. Prendre par exemple le nombre suivant : `1011_1001_1101_0001_b`

Exercice 5 - Calculer la somme des nombres naturels suivants en base 2 sur 8 bits. Que remarquez-vous ?

- $0000\_0010\_b + 0000\_0011\_b$
- $0000\_1010\_b + 0000\_1111\_b$

Exercice 6 - 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

$$\begin{array}{lll} 0111\_1111\_b & \text{représente} & 127\_d \\ 1111\_1111\_b & \text{représente} & -127\_d \end{array}$$

Exercice 7 - Calculer la somme des nombres signés suivants sur 8 bits. Que remarquez vous ?

- $0000\_0111\_b + 0000\_0101\_b$
- $0000\_0111\_b + 1000\_0101\_b$

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 8 - 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 9 - Calculer la somme des nombres en complément à deux suivants sur 8 bits. Que remarquez vous ?

- $0000\_0111\_\bar{b} + 0000\_0101\_\bar{b}$
- $0000\_0111\_\bar{b} + 1000\_0101\_\bar{b}$
- $0000\_0011\_\bar{b} + 1111\_1011\_\bar{b}$

- $0100\_0000\_\bar{b} + 0100\_0001\_\bar{b}$

Exercice 10 - Calculer le **produit** des nombres en complément à deux suivants sur 8 bits. Que remarquez vous ?

- 7×5
- 7×-5
- 48×-2
- 48×-3

Exercice 11 -

— 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

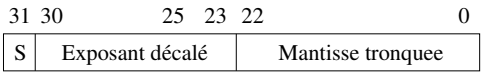

FIGURE 1 – Norme IEEE 754

Exercice 12 - Représentez en norme IEEE 754, les nombres suivants :

- $133,875_{10}$
- $-14,6875_{10}$
- $5,59375_{10}$
- $0,66_{10}$

Exercice 13 - Trouvez à quels nombres réels correspondent les représentations IEEE 754 :

- $42\_C8\_40\_00_{16}$
- $48\_92\_F5\_40_{16}$
- $C2\_92\_F0\_00_{16}$
- $C3\_B0\_30\_00_{16}$

2 Assembleur

Exercice 14 -

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.

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	1_024
11	2_048
12	4_096
13	8_192
14	16_384
15	32_768

N	2^N
16	65_536
20	1_048_576
31	2_147_483_648
32	4_294_967_296
63	$9,22 \times 10^{18}$
64	$18,44 \times 10^{18}$

TABLE 1 – Puissances de 2

Unité	Symbole	Puissance	Nombre
kilo octet	ko	2^{10}	1_024
méga octet	Mo	2^{20}	1_048_576
giga octet	Go	2^{30}	1_073_741_824
téra octet	To	2^{40}	$1,09951162810^{12}$
péta octet	Po	2^{50}	$1,12589990710^{15}$
exa octet	Eo	2^{60}	$1,15292150510^{18}$
zetta octet	Zo	2^{70}	$1,18059162110^{21}$

TABLE 2 – Unités utilisées en Informatique - Zak Etait Petit Trés Gros, Maousse Kosto

```
1 int a = 1, b = 2;
2
3 if (a <= b) {
4   i1;
5 } else {
6   i2;
7 }
```

2. traduire en assembleur x86 le code C suivant :

```
1 int sum = 0;
2 for (int i = 1; i < n; i++) {
3   sum = sum + i;
4 }
```

3. traduire en assembleur x86 le code C suivant :

```
1 int sum = 0, n = ...;
2 int i = ...;
3 while ((i > 10) && (i <= n)) {
4   sum = sum + i;
5   ++i;
6 }
```

4. traduire en assembleur x86 le code C suivant :

```
1 int prod = 1, n = ...;
2 int i = ...;
3 while ((i  % n) == 0) || (i == 3)) {
4   prod = prod * i;
5   ++i;
6 }
7 printf("%d", prod);
```

Exercice 15 - Traduire la procédure suivante en assembleur x86.

```
1 void init(char t[], int n) {
2   for (int i = 0; i < n; i++) {
3     t[i] = '0';
4   }
5 }
```

- donner une traduction assembleur de la procédure
- donner des versions optimisées en utilisant `loop`, puis `stosb`

Exercice 16 - Traduire la procédure suivante en assembleur x86.

```
1 void init( int t[], int n) {
2   for (int i = 0; i < n; i++) {
3     t[i] = 0;
4   }
5 }
```

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

Exercice 17 - Traduire la fonction suivante en assembleur x86 :

```
1 int f(int X, int Y, int Z ) {
2   if (Z ==0) {
3     return ((X+Y)*(X-Y))/((X/Y)*3+(X+Y)*(X-Y));
4   } else {
5     return (X-Z+Y)*(X+Z+Y)*(1-Z*Z);
6   }
7 }
```

Exercice 18 - Comment multiplier efficacement sur un 8086 (cf. Table 3), 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. On pensera au fait que $10 = 5 \times 2$ ou encore $10 = 8 + 2$.

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

Instruction	Cycles
add r16, r16	3
div r16	144-162
mov r16, r16	2
mov r16, imm	4
mul r16	118-133
pop r16	8
push r16	11
shl r16, CL	8 + 4/bit

TABLE 3 – Intel 8086 : nombre de cycles pour exécuter une instruction

```
1 int fib(int n) {
2   if (n <= 1)
3     return n;
4   else
5     return fib(n-1) + fib(n-2);
6 }
```

Exercice 20 - Écrire le code assembleur x86 de la fonction de Fibonacci optimisée :

```
1 int fib_iter(int a, int b, int count ) {
2   if (count == 0)
3     return b;
4   else
5     return fib_iter(a+b, a, count-1);
6 }
```

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

Exercice 21 - 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 22 - 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 23 - On désire écrire en assembleur la fonction suivante sans utiliser d'instruction de branchement. Traduire dans un premier temps cette fonction en plaçant p dans **ecx** et q dans **edx**.

```
1  int mini(int p, int q) {
2    return (p < q) ? p : q;
3  }
```

Afin de supprimer le ou les branchements, on va utiliser une variable temporaire r telle que :

$$r = (p - q) >> 31 = \begin{cases} \texttt{0x00000000}, \text{ si} & p > q \\ \texttt{0xFFFFFFFF}, \text{ si} & p < q \end{cases}$$

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` Shift Arithmetic Right). A partir de ce moment, il suffit de calculer : $(p \wedge r) \vee (q \wedge \neg r))$.

Exercice 24 - Implanter la fonction POPCNT (Population Count) qui compte le nombre de bits à 1 dans un registre. Par exemple `popcnt ebx, eax` met dans `ebx` le nombre de bits à 1 dans `eax`. En ce qui concerne l'implantation, on utilisera une table de conversion et on travaillera sur chaque octet du registre `eax`.

Exercice 25 - Réaliser le dépliage de boucle par 4 (loop unrolling) sur le code C suivant :

```
1  void f(int *v, int *w, int k, int n) {
2    for (int i = 0; i < n; ++i) {
3      v[i] += w[i] * k;
4    }
5  }
```

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

```
1  void prod(int a[N][N], int b[N][N], int c[N][N]) {
2    int i,j,k;
3    for (i = 0; i < N; i++) {
4      for (j = 0; j < N; j++) {
5        int sum=0;
6        for (k = 0; k < N; k++) sum += a[i][k] * b[k][j];
7        c[i][j] = sum;
8      }
9    }
10 }
```

Exercice 27 - 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 28 - Ecrire une version SSE du calcul de la suite de Fibonacci.

Exercice 29 - Coder la fonction suivante en assembleur classique puis donner une version qui utilise les registres SSE.

```
1  int chr_replace(char *src, char *dst, int n, char c, char d) {
2    int changes=0;
3    for (int i = 0; i < n; ++i) {
4      if (src[i] == c) {
5        dst[i] = d;
6        ++changes;
7      } else {
8        dst[i] = src[i];
9      }
10   }
11   return changes;
12 }
```

Exercice 30 - 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 31 - Écrire une fonction en C qui permet de déterminer les nombres autodescriptifs tels que 1210, 2020, 21200, 3211000. Pour rappel, un nombre autodescriptif est un entier naturel dont le premier chiffre indique le nombre de 0 qu'il contient, le deuxième chiffre le nombre de 1, etc.

3 Logique, algèbre de Boole, Tableaux de Karnaugh

Exercice 32 - 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 33 - 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 34 - 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 35 - 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 36 - 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 37 - 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) $XZ + XY\bar{Z} + W\bar{X}\bar{Y}$