STRUCTURE MACHINE

Module : structure machine

Volume : 1 cours + 1 TD

Objectifs du module :

1. Connaître les différents systèmes de numération et leurs conversions (2, 8, 16)
2. Connaître les différentes techniques de représentation des données dans la machine
3. Etudier l'algèbre de Bool pour l'analyse et la synthèse des circuits

---

Plan du cours :

Chap0 : Généralités

Chap1 : Les systèmes de numérations

Chap2 : Représentation des données

Chap3 : Algèbre de Bool

Chap4 : Les circuits combinatoires

Semchedine Moussa 2012

C'est quoi un ordinateur ?

C'est une machine électronique capable de traiter toute sorte d'information (texte, image….)

Pourquoi utiliser un ordinateur?

Pour accélérer le traitement et pour avoir des résultats de calcul corrects (précis).

Système informatique ?

C'est l'ensemble des programmes (software) et matériels (hardware) nécessaires pour satisfaire les besoins informatiques d'un utilisateur

Architecture de Von Neumann ?

L'architecture des ordinateurs suit la décomposition proposée par Von Neumann 1945

- UAL (unité arithmétique et logique)
- UC : (unité de contrôle)
- Mémoire centrale
- Périphériques d'entrée
- Périphériques de sortie

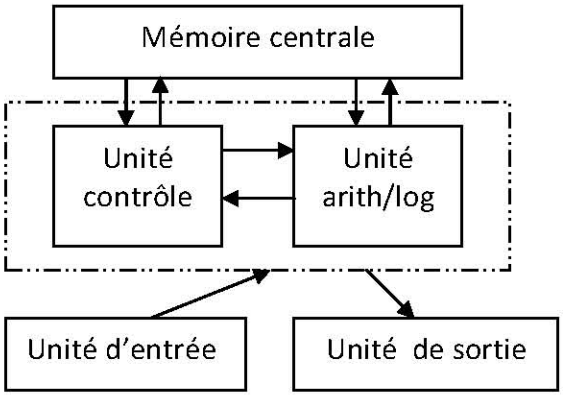

Figure1 : L'architecture de Von Neumann

Remarque : les différents composants sont reliés par des bus de communication

Comment les données sont représentées à l'intérieur de la machine ?

- Le bit (binary digit) est la plus petite unité de stockage d'information (0 ou 1)
- Le langage machine est la suite de 0 et des 1 (bits) compréhensible par l'ordinateur
- Les 1 et 0 sont des signes symboliques, réellement dans la machine il n'y a plus de 1 et 0

Pourquoi la machine travaille avec le binaire ?

Le matériel informatique fonctionne sur la base des impulsions **électriques**, le courant passe (valeur 1) ou il ne passe pas (0)

Unités de mesure en informatique :

1 octet = 8 bits

1 Ko (kilo octets) = 2^{10} octets=1 024

1 Mo (Mega octets) = 2^{20} octets=1 048 576

1 Go (Giga octets) = 2^{30} octets=1 073 741 824

1 To (Tera octets) = 2^{40} octets 1 Po (Peta octets) = 2^{50} octets

1 Eo (Exa octets) = 2^{60} octets 1 Zo (Zeta octets) = 2^{70} octets 1 Yo (Yotta octets) = 2^{80} octets

Définition:

Un système de numération désigne le mode de représentation des nombres à l'aide des symboles appelés chiffres. Le nombre de chiffres utilisés pour représenter les valeurs est appelé une base *b*.

1. Système de base *b*

La représentation d'un nombre dans un système de numération de base ***b***, où les chiffres utilisés $\in \{ c_0, c_1, c_2, \ldots, c_i \}$ et $(c_i < b)$ est comme suit $(c_n c_{n-1} \ldots c_1 c_0)_b$

Pour convertir un nombre de base **b** en décimal, il suffit d'utiliser l'écriture polynomiale suivante :

$$c_n x b^n + c_{n-1} x b^{n-1} + \ldots\ldots\ldots\ldots + c_1 x b^1 + c_0 x b^0$$

- le poids faible b^0
- le poids fort b^n

2. Système décimal (base 10)

C'est le système de numérotation le plus utilisé par l'Homme (choix naturel). La base b=10 et les chiffres $\in \{0,1,2,3,4,5,6,7,8,9\}$, ainsi l'écriture polynomiale d'un nombre décimal est sous la forme :

$$c_n x 10^n + c_{n-1} x 10^{n-1} + \ldots\ldots\ldots\ldots + c_1 x 10^1 + c_0 x 10^0$$

Exemple : 7319= $7x10^3 + 3x10^2 + 1x10^1 + 9x10^0$

3. Système binaire (base 2)

C'est le système de numération utilisé en informatique. La base b=2 et les chiffres $\in \{0,1\}$

Exemple : $(11001)_2$ est un nombre binaire

3.1 Conversion (décimal → binaire) : il faut suivre ces étapes :

- Diviser le nombre décimal par 2 et noter le quotient et le reste de la division
- Répéter le processus jusqu'à ce que le quotient soit 0
- Réécrire les restes de division (du bas vers le haut, de gauche vers la droite)

Exemple : $19=(10011)_2$ $64=(1000000)_2$

3.2 **Conversion (binaire → décimal)** : utiliser l'écriture polynomiale avec b=2

- Multiplier chaque chiffre binaire par la puissance de 2 correspondant à son poids
- Faire la somme des produits obtenus

Exemple : $(11111)_2 = 31$ $(11010)_2 = 26$

4. Système octal (base 8)

La base *b*=8 et les chiffres ∈ {0, 1, 2, 3, 4, 5, 6,7}

Exemple : $(371)_8$ est un nombre octal

4.1 Conversion (décimal → octal) : de la même façon que (décimal → binaire) sauf que la division est par 8

Exemple : *$20=(24)_8$* *$98=(142)_2$*

4.2 Conversion (octal → décimal) : utiliser l'écriture polynomiale avec b=8

Exemple : *$(27)_8=23$* *$(134)_8=92$*

5. Système hexadécimal (base 16)

C'est un système très utilisé en informatique, notamment avec la programmation de bas niveau tel que l'assembleur. La base *b*=16 et les chiffres ∈ {0, 1, 2, 3, 4, 5, 6,7, 8, 9, A, B, C, D, E, F}

Exemple : $(1A5)_{16}$ est un nombre hexadécimal

5.1 Conversion (décimal → hexa) : de la même façon que (décimal → binaire) sauf que la division est par 16

Exemple : *$29=(1D)_{16}$* *$20=(14)_{16}$*

5.2 Conversion (hexa → décimal) : utiliser l'écriture polynomiale avec b=16

Exemple : *$(1E)_{16}=30$* *$(11)_{16}=17$*

6. Conversion (p→q) (p, q € 2, 8 ,16)

- (octal → binaire) :
  - soit on utilise la base intermédiaire 10
  - Ou décomposer chaque chiffre octal en son équivalent en base 2 sur 3 bits en commençant par la droite

  Exemple : *$(57)_8 = (101111)_2$*
- (binaire → octal)
  - soit on utilise la base intermédiaire 10
  - Ou regrouper chaque 3 chiffres de la base binaire en leurs équivalents en base 10 en commençant par la droite. Si le dernier groupe ne contient pas trois chiffres, ajouter des zéros à gauche

  Exemple : *$(101111)_2 = (57)_8$* *$(11010)_2 = (32)_8$*

- (hexa → binaire)
  - soit on utilise la base intermédiaire 10
  - Ou décomposer chaque chiffre de la base 16 en son équivalent en base 2 sur 4 bits en commençant par la droite

  Exemple : $(A6)_{16} = (10100110)_2$ $(1E)_{16} = (00011110)_2$

- (binaire → hexa)
  - soit on utilise la base intermédiaire 10
  - Ou regrouper chaque 4 chiffres de la base 2 en leurs équivalents en base 16 en commençant par la droite. Si le dernier groupe ne contient pas 4 chiffres, ajouter des zéros à gauche

  Exemple : $(10010011)_2 = (93)_{16}$ $(11100)_2 = (1C)_{16}$

- (octal → hexa) ou (hexa → octal)

  Soit on utilise la base intermédiaire 10 ou la base intermédiaire 2 (la plus rapide)

  Exemple : $(54)_8 = (2C)_{16}$ $(3A)_{16} = (72)_8$

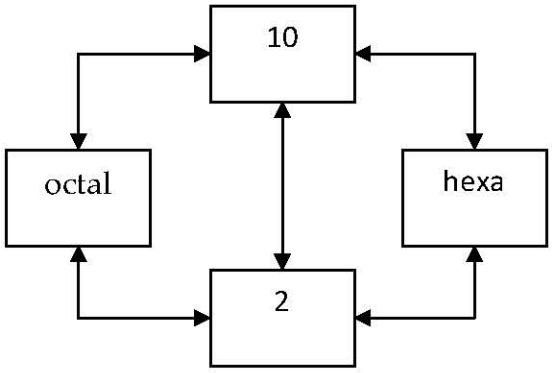

Figure1 : Schéma général de conversion

7. Conversion des nombres fractionnaires

Un nombre fractionnaire est un nombre composé de deux parties : la partie entière et partie fractionnaire : $(c_n c_{n-1} \ldots\ldots c_1 c_0, c_{-1} c_{-2} \ldots\ldots c_{-m})_b$

Exemple : 13.125 $(1101,101)_2$ $(75,14)_8$ $(F,7)_{16}$

7.1 Conversion décimal→ (2,8,16)

- Convertir la partie entière par divisions successives
- Convertir la partie décimale par des multiplications par ***b***

Exemple :

$8.125 = (100,001)_2$

$13,25 = (15,2)_8$

$28.75 = (1C,C)_{16}$

Remarque 1 :

La conversion d'un nombre fractionnaire décimal n'est pas toujours effectuée avec précision

Exemple : 4.8 = $(100,11001100.....)_2$

7.2 Conversion (2,8,16) → décimal

Pour convertir le nombre N en décimal, il faut traiter séparément la partie entière et la partie décimale

Partie entière = $c_n xb^n + c_{n-1} xb^{n-1} + ... + c_1 xb^1 + c_0 xb^0$

Partie fractionnaire = $c_{-1} xb^{-1} + c_{-2} xb^{-2} + ... + c_m xb^{-m}$

Exemple :

$(1001,01)_2$ = 9 + 0.25 = 9.25

$(76,4)_8$ = 62 + 0.5 = 62.5

$(2E,C)_8$ = 46 + 0.75 = 46.75

8. Opérations arithmétiques

Les opérations arithmétiques (+ - * /) s'effectuent en base ***b*** avec les mêmes méthodes qu'en base 10. Une retenue apparait lorsque le résultat dépasse la valeur ***b***

Exemple : calculer les opérations suivantes

$(100010)_2 + (100111)_2$ $(1010)_2 - (11)_2$ $(110)_2 x (11)_2$ $(10010)_2 / (11)_2$ $(10110)_2 / (110)_2$

8.1 Le Débordement (Overflow)

Dans un ordinateur la taille des registres est fixe, ce qui impose une limite aux nombres représentés. Lors d'une opération arithmétique sur des nombres s'il y a une retenue qui est générée par le bit du poids fort on dit qu'il y a un débordement ou dépassement de capacité

Exemple : Un ordinateur travaille sur des registres de 4 bits.

- Effectuer l'addition suivante : $(1110)_2 + (0110)_2$
- Vérifier les résultats en décimal ?
- Quelle est la solution à ce problème ?

Introduction

La représentation interne de l'information consiste à lui donner un codage en binaire

Information externe→ codage→ information interne (binaire)

1. Typologie de l'information

Les ordinateurs traitent avec deux types d'informations :

1. Les instructions : représentent les opérations effectuées par l'ordinateur
2. Les données : ce sont les opérandes sur lesquels portent les opérations

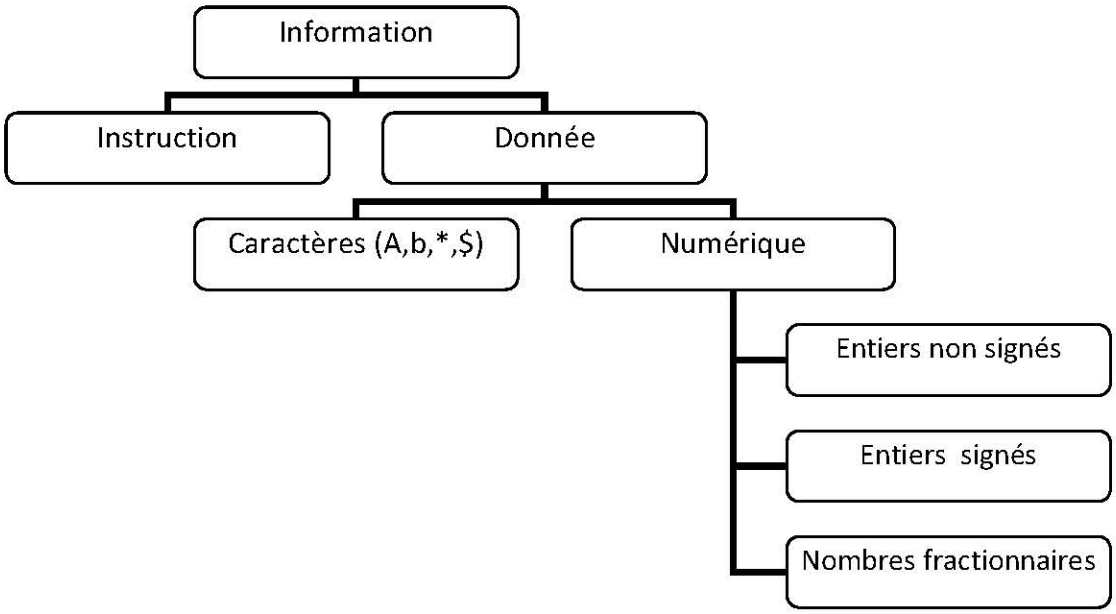

Figure3 : Typologie de l'information

2. Représentation des entiers

2.1 Entiers non signés

Le nombre est représenté en binaire sur les **n** bits. L'intervalle des nombres représentables sur n bits : **[0, 2^n-1]**

Exemple : représenter la valeur 35 sur 8bits, réponse $(00100011)_2$

2.2 Entiers signés

Il existe 3 méthodes pour représenter les entiers signés :

- Signe et valeur absolue (SVA)
- Complément à 1 (C1)
- Complément à 2 (C2)

2.2.1 Signe et valeur absolue

la valeur absolue du nombre est codée sur **n – 1** bits et le signe sur le bit du poids **fort**, par convention 0 représente un nombre positif et 1 un nombre négatif. Ainsi L'intervalle des nombres représentables sur n bits en SVA est : $[-(2^{n-1}-1), +(2^{n-1}-1)]$

- *Passage SVA-----décimal*

il faut convertir la valeur absolue en décimal et précéder cette dernière par le signe (+) si le bit du poids fort=0 ou (-) sinon

Exemple : donnez la valeur en décimal des nombres suivants représentés en SVA sur 8 bits ?

$(00001101)_2 = +13$ $(10000101)_2 = -5$

- *Avantages et inconvénients de SVA*

SVA est assez simple, mais présente l'inconvénient de la double représentation du zéro -0 et +0, et aussi la difficulté des opérations arithmétiques qui sont compliquées, à cause du bit de signe qui doit être traité à part

Exemple : calculer +5 + (-2) sur 4 bits en SVA ?

2.2.2 Complément à 1

Dans cette représentation les nombres positifs gardent le format binaire. Les nombres négatifs sont complémentés c.-à-d. les 0 sont transformés en 1, et les 1 en 0. Ainsi L'intervalle des nombres représentables sur n bits en C1 est : $[-(2^{n-1}-1), +(2^{n-1}-1)]$

Exemple : donnez la représentation des nombres sur 4 bits en complément à 1 ?

- *Passage C1-----décimal*

Les nombres positifs sont évalués en conversion binaire →décimal. Les nombres négatifs sont évalués en inversant les **n-1** bits puis introduire le signe moins

Exemple : donnez la valeur en décimal des nombres suivant représentés en C1 sur 8 bits ?

$(00001000)_2 = +8$ $(11110011)_2 = -12$

- *L'addition en complément à 1*

Elle se base sur le principe suivant :

- Si aucune retenue n'est générée par le bit de signe, le résultat est correct, il est représenté en C 1
- sinon, elle sera additionnée au résultat de l'opération, celui-ci est représenté en C1

Exemple : calculer +5 + 2 -5-2 +5 + (-2) -5 +2 sur 4 bits en C1 ?

Remarque :

L'inconvénient de la représentation en C1 est la **double** représentation du zéro +0 et - 0

2.2.3 Complément à 2

En complément à 2 les nombres positifs gardent le format binaire. Les nombres négatifs par contre, sont obtenus en calculant d'abord le complément à 1, puis ajouter un 1. C2 = C1+1.

L'intervalle des nombres représentables sur n bits en C1 est : **$[-2^{n-1}, +(2^{n-1}-1)]$**

- ***Passage C2-----décimal***

Les nombres positifs sont évalués en conversion binaire →décimal. Les nombres négatifs sont évalués en faisant la somme des poids (on affecte une valeur négative au bit du poids fort)

Exemple : donnez la valeur en décimal des nombres suivant représentés en C2 sur 8 bits ?

$(00001101)_2 = +13$ $(10001000)_2 = -120$

- ***L'addition en complément à 2***

Elle se base sur le principe suivant :

- S'il y a une retenue qui est générée par le bit de signe, elle est ignorée
- Le résultat est en complément à 2

Exemple : calculer +5 + 2 -5-2 +5 + (-2) -5 +2 sur 4 bits en C2 ?

- ***Avantages de complément à 2***

- la représentation la plus utilisée pour les nombres négatifs
- Une seule représentation du zéro
- Un nombre négatif supplémentaire

Remarques :

- En C1 ou C2, les opérations arithmétiques sont avantageuses. En effet, la soustraction d'un nombre se réduit à l'addition de son complément. Ainsi il n'y a pas de traitement particulier pour le bit de signe
- Il y a un débordement en complément à 2 si :
    1. la somme de deux nombres négatifs donne un nombre positif ou la somme de deux nombres positifs donne un nombre négatif
    2. le résultat de l'opération dépasse l'ntervalle des valeurs représentables
- il n'y a pas de débordement lorsque les deux opérandes sont de signes différents

Représentation des entiers sur 4 bits

SVA	déc
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

C1	déc
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1111	- 0
1110	-1
1101	-2
1100	-3
1011	-4
1010	-5
1001	-6
1000	-7

C2	déc
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

3. Représentation des Nombres fractionnaires

3.1 Représentation en virgule Fixe

Cette représentation pose un problème au niveau de la machine. La première solution était de ne pas représenter matériellement la virgule et de traiter le nombre fractionnaire comme un nombre entier. La virgule est virtuelle, elle est gérée par le programmeur, c'est lui qui définit sa position, chose qui n'est pas facile, d'où son inconvénient. Un autre inconvénient est la limitation de représenter des nombres

Exemple : On considère un nombre représenté sur 6 bits

1 bit : pour le signe

3 bits pour la partie entière

2 bits pour la partie fractionnaire Valeur Min= $(111011)_2$ = **-7.75** Valeur MAX $(011111)_2$ = **+7.75**

3.2 Représentation en virgule flottante

Au début, la représentation en virgule flottante n'était par normalisée où chaque ordinateur utilisait son propre format. Plusieurs normes ont été définies, parmi elles la norme *IEEE 754 (Institute of Electrical and Electronics Engineers)*

Cette norme définit trois formats de représentations des nombres flottants :

- **Simple** précision sur 32 bits (1 bit de signe 8bits exposant décalé 23 bits mantisse)
- **Double** précision sur 64 bits (1 bit de signe 11bits exposant décalé 52 bits mantisse)
- **Précision étendue** sur 80 bits (1 bit de signe 15 bits exposant décalé 64 bits mantisse)
    - **Représentation en IEEE 754 simple précision**

Le nombre est exprimé comme suit :

- 1 bit de signe
- 8 bits pour l'exposant décalé = exposant réel +127. L'exposant est décalé de 2^{8-1} - 1. Ce décalage est utile car l'exposant peut être positif ou négatif
- 23 bits pour la mantisse

Pour écrire un nombre sous la forme IEEE754 il faut :

1. Convertir le nombre en binaire
2. Ecrire le nombre sous la forme normalisée **$N= (+/-)(1,m)_2.2^{ER}$**
3. Calculer l'exposant décalé= Exposant réel+ 127

Remarque:

Le 1 précédant la virgule n'est pas codé en machine (appelé bit caché)

Exemple : représenter en IEEE 754 le nombre N= -3.625

1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Pour convertir un nombre écrit sous la forme IEEE754 :

1. calculer l'exposant réel =Eb-127
2. Calculer $valeur = signe \times (1, mantisse)_2 \times 2^{Er}$, avec $signe = \pm 1$

Exemple : Quelle est la valeur en décimal du nombre suivant représenté en IEEE 754

1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Réponse : - **4.75**

Remarque:

L'avantage de la virgule flottante par rapport à la virgule fixe est que l'intervalle des valeurs possibles est plus étendu (elle est la plus utilisée par les ordinateurs)

Dans la norme IEEE754 il y a des valeurs spéciales

Le zéro : MANTISSE =0 EXPOSANT = 0

1/0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Infini : MANTISSE = 0 EXPOSANT = 1 PARTOUT

Une valeur infinie est utilisée pour indiquer par exemple un débordement ou une division par zéro.

Il y a deux valeurs infinies possibles, positive et négative.

1/0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

NaN (Not a Number) MANTISSE ≠ 0 EXPOSANT = 1 PARTOUT

NAN est le résultat d'une opération arithmétique erronée. Par exemple le calcul de la racine carrée des nombres négatifs

Opérations arithmétiques en IEEE754

A) **Addition et soustraction en virgule flottante**

Soit A, B deux nombres écrits en IEEE754. L'addition ou la soustraction en virgule flottante est comme suit :

* Alignement des mantisses si les exposants de A et B sont différents (élevé au plus grand exposant)
* Addition ou soustraction des mantisses alignées
* Normaliser de la somme S si elle n'est pas normalisée

Exemple : A= 1 10000011 10110000000000000000

B= 0 10000010 10100000000000000000

calculez A+B et A-B réponse -20.5 -33.5 vérifier les résultats en décimal ?

B) **Multiplication et division en virgule flottante**

On peut résumer la multiplication par la multiplication des mantisses normalisées et par l'addition des exposants

Exemple : calculez AxB

A= 1 10000010 00100000000000000000

B= 0 10000001 10000000000000000000

réponse AxB= -13.5

4. Représentation des Caractères

L'ensemble des caractères alphanumériques regroupe les lettres 'a'……'z' ,'A…….'Z', les chiffres 0…..9,et les caractères spéciaux +,-,*, ? # . Le codage le plus utilisé est ASCII *(American Standard Code for Information Interchange).* En ASCII chaque caractère est représenté sur 8 bits (256 caractères possibles)

Remarque :

Actuellement il existe le codage ***Unicode*** *où* chaque caractère est codé sur 16 bits, Il regroupe autres alphabets existants (arabe, hébreu,….)

Exemple : voir la table ASCII

Le code $(41)_{16} = (01000001)_2$ *correspond au caractère A*

Le code $(61)_{16} = (01100001)$ *correspond au caractère a*

Le code $(3F)_{16} = (00111111)_2$ *correspond au caractère ?*