Introduction à l'Architecture des ordinateurs

Architecture des ordinateurs

Guillaume Blin

IGM-LabInfo UMR 8049,
Bureau 48066
Université de Marne La Vallée
gblin@univ-mlv.fr
http://igm.univ-mlv.fr/~gblin

Objectif de ce cours

- Comprendre les grands principes de fonctionnement d'un ordinateur
- Connaître les composants d'un ordinateur moderne et les technologies qu'ils utilisent
- Acquérir une connaissance à « bas niveau » de la programmation

Références

- Cours de Frédéric Goualard de l'Université de Nantes
  http ://supports.goualard.free.fr/
- Cours de Emmanuel Viennet de l'Université Paris 13
  http ://www-gtr.iutv.univ-paris13.fr/Cours/Mat/Architecture/

Plan

Plan

Motivations

- Pourquoi créer des ordinateurs ?
  - Accélération de calculs compliqués et/ou répétitifs
    - Mouvement des planètes
    - Tables de logarithmes
    - Horaires des marées
    - Calcul de trajectoires ballistiques
  - Traitement de gros volumes de données
    - Recensement

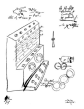
Une petite chronologie

de la "machine à calculer à engrenages" à "l'assembleur"

- 1623 : Schickard
- 1644 : Pascal
- 1801 : Jacquard
- 1823 - 1833 : Babbage
- 1840 : Lovelace
- 1854 : Boole

1623

Schickard
En 1623, Wilhelm Schickard (1592-1635) inventa pour Kepler ce qu'il appela une « horloge calculante » qui était destinée à calculer les éphémérides. Il utilisait des roues dentées et avait déjà abordé le problème du report de retenue.



Une petite chronologie

de la "machine à calculer à engrenages" à "l'assembleur"

- 1623 : Schickard
- 1644 : Pascal
- 1801 : Jacquard
- 1823 - 1833 : Babbage
- 1840 : Lovelace
- 1854 : Boole

1644

Pascal

En 1644, Blaise Pascal (1623–1662) inventa une machine ("La Pascaline") capable d'effectuer des additions et des soustractions (par complément à 10) afin d'aider son père, collecteur d'impôts a Rouen. Ce fut le premier modèle commercialisé.

1644

Soustraction par complément à 10

Dans la méthode des compléments à 10, pour faire une soustraction, chacun des chiffres du nombre à soustraire est remplacé par son complément à 9 et additionné au contenu du totalisateur, puis on ajoute 1 pour obtenir le résultat exact en ignorant le chiffre le plus à gauche.

$$\begin{array}{r} 563 \\ -27 \\ \hline 563 \\ +972 \\ +1 \\ \hline \not{1}536 \end{array}$$

1623-1644

Machines de Schickard et Pascal

- Pas de notion de « programme »
- Pas de mémorisation des résultats
- Usage déterminé
- Calcul requiert l'énergie humaine (tourner la manivelle)

Une petite chronologie

de la "machine à calculer à engrenages" à "l'assembleur"

- 1623 : Schickard
- 1644 : Pascal
- 1801 : Jacquard
- 1823 - 1833 : Babbage
- 1840 : Lovelace
- 1854 : Boole

1801

Le métier à tisser Jacquard

- Le métier Jacquard est un métier à tisser « programmable » utilisant la vapeur mis au point par le lyonnais Joseph Marie Jacquard en 1801
- La machine Jacquard combine les techniques des aiguilles de Basile Bouchon, les cartes perforées de Falcon et du cylindre de Vaucanson
- Cette utilisation de cartes perforées fait qu'il est parfois considéré comme l'ancêtre de l'ordinateur

Une petite chronologie

de la "machine à calculer à engrenages" à "l'assembleur"

- 1623 : Schickard
- 1644 : Pascal
- 1801 : Jacquard
- 1823 - 1833 : Babbage
- 1840 : Lovelace
- 1854 : Boole

1823-1833

La machine analytique

Charles Babbage entre 1823 et 1833 définit les principaux concepts sur lesquels reposent les machines informatiques, soit :

- un dispositif d'entrée et de sortie (un clavier et un moniteur) ;
- un organe de commande gérant le transfert des nombres et leur mise en ordre pour le traitement (une unité de commande) ;
- un magasin permettant de stocker les résultats intermédiaires ou finaux (mémoire vive, disque dur, supports amovibles) ;
- un moulin chargé d'exécuter les opérations sur les nombres (une unité de calcul) ;
- un dispositif d'impression (imprimante).

Une petite chronologie

de la "machine à calculer à engrenages" à "l'assembleur"

- 1623 : Schickard
- 1644 : Pascal
- 1801 : Jacquard
- 1823 - 1833 : Babbage
- 1840 : Lovelace
- 1854 : Boole

1840

L'algorithmique

- Ada Lovelace, mathématicienne britanique, définit le principe des itérations successives d'opérations dans l'exécution d'un programme
- En l'honneur du mathématicien Al Khowarizmi (780-850), elle nomme le processus logique d'exécution d'un programme : algorithme


Une petite chronologie

de la "machine à calculer à engrenages" à "l'assembleur"

- 1623 : Schickard
- 1644 : Pascal
- 1801 : Jacquard
- 1823 - 1833 : Babbage
- 1840 : Lovelace
- 1854 : Boole

1854

Algèbre de Boole

- Georges Boole développe une nouvelle forme de logique, à la fois symbolique et mathématique
- C'est une algèbre binaire n'acceptant que deux valeurs numériques : 0 et 1 ; et munie de deux lois de composition interne (le ET et le OU)
- A l'origine des ordinateurs à arithmétique binaire


George Boole

Une petite chronologie

de la "machine à calculer à engrenages" à "l'assembleur"

- 1890 : Hollerith
- 1938 - 1941 : Zuse
- 1944 : Eckert & Mauchly
- 1944 - 1952 : Eckert & Mauchly & von Neumann
- 1949 : Wilkes

1890

La machine à recenser

- Herman Hollerith *invente* la machine électrique/mécanique (une impulsion électrique déplace des roues dentées) à cartes perforées pour accélérer le recensement de 1890 aux États-Unis traité en trois ans seulement (au lieu des 9 de celui de 1880)
- Il fonde une compagnie qui sera rebaptisée en 1917 International Business Machine

1623-1890

Bilan

- Machines essentiellement mécaniques
- Machines spécialisées pour une tâche :
  - Quatre opérations entières
  - Métier à tisser
  - Comptage de valeurs
  - ...

Une petite chronologie

de la "machine à calculer à engrenages" à "l'assembleur"

- 1890 : Hollerith
- 1938 - 1941 : Zuse
- 1944 : Eckert & Mauchly
- 1944 - 1952 : Eckert & Mauchly & von Neumann
- 1949 : Wilkes

1938

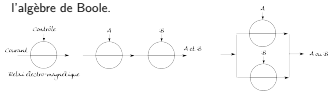
Le Z1

- Konrad Zuse, 1910–1995 invente une machine multi-usage
  - Mémoire pour conserver les données
  - Unité arithmétique
  - Unité de contrôle (quelle opération et sur quelles données ?)
  - Unités d'entrées/sorties des données
  - Calcul interne en binaire
  - Opérations décrites avec l'algèbre de Boole

1941

Le Z3

- Ce calculateur électromécanique était la première machine programmable pleinement automatique ce qui en ferait le premier ordinateur du monde.
- Le Z3 était composé de 2200 relais électromécaniques, avait une vitesse d'horloge de 5,33 Hz et une longueur de mots de 22 bits.
- Conçut sur la base du système binaire et reposant sur l'algèbre de Boole.



Une petite chronologie

de la "machine à calculer à engrenages" à "l'assembleur"

- 1890 : Hollerith
- 1938 - 1941 : Zuse
- 1944 : Eckert & Mauchly
- 1944 - 1952 : Eckert & Mauchly & von Neumann
- 1949 : Wilkes

1944

ENIAC

- John Eckert (1919–1995) et John Mauchly (1907–1980) invente la "Electronic Numerical Integrator & Calculator"
  - Premier ordinateur électronique
  - Utilisation de tubes à vide
  - Poids : 30 tonnes
  - Données en mémoire (base 10)

Une petite chronologie

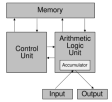
de la "machine à calculer à engrenages" à "l'assembleur"

- 1890 : Hollerith
- 1938 - 1941 : Zuse
- 1944 : Eckert & Mauchly
- 1944 - 1952 : Eckert & Mauchly & von Neumann
- 1949 : Wilkes

1944-1952

Architecture de von Neumann

- l'unité arithmétique et logique (UAL), qui effectue les opérations de base ;
- l'unité de contrôle, qui est chargée du séquençage des opérations ;
- la mémoire, qui contient à la fois les données et le programme
- les dispositifs d'entrée-sortie, qui permettent de communiquer avec le monde extérieur.



Une petite chronologie

de la "machine à calculer à engrenages" à "l'assembleur"

- 1890 : Hollerith
- 1938 - 1941 : Zuse
- 1944 : Eckert & Mauchly
- 1944 - 1952 : Eckert & Mauchly & von Neumann
- 1949 : Wilkes

1949

EDSAC

- Maurice Wilkes, met au point le premier ordinateur à architecture « von Neumann » : EDSAC
  - EDSAC = 3000 tubes à vide
  - Programmes entrés sur cartes perforées
  - Sorties sur imprimantes
  - Programmes codés en assembleur (sous forme binaire puis avec des mnémoniques)

Plan

Types d'informations

- Les informations traitées par un ordinateur peuvent être de différents types (texte, nombres, etc.)
- Pour autant, elles sont toujours représentées et manipulées par l'ordinateur sous forme binaire.
- Architecture utilisant l'absence ou la présence de courant (tubes ou transistors)
- Toute information sera traitée comme une suite de 0 et de 1.
- L'unité d'information est le chiffre binaire (0 ou 1), que l'on appelle *bit*

Codage et représentation binaire

- Le codage d'une information consiste à établir une correspondance entre la représentation externe de l'information, et sa représentation interne dans la machine, qui est une suite de bits.
- On utilise la représentation binaire car elle est simple, facile à réaliser techniquement et les opérations arithmétiques de base sont faciles à exprimer en base 2.

Les nombres binaires

- En électronique : deux états internes symbolisés par 0 (du courant) et 1 (pas de courant)
- Informations :

Nom	Valeur
Bit	0 ou 1
Octet	00000000 à 11111111

- Multiples définis depuis 1998 :

Nom	Notation	Valeur
1 kilobit	1 kbit	10^3 ou $2^{10} = 1\,024$ bits
1 megaoctet	1 MB/1 Mo	10^6 ou $2^{20} = 1\,048\,576$ octets
1 gigaoctet	1 GB/1 Go	10^9 ou $2^{30} = 1\,073\,741\,824$ octets

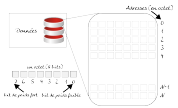
dépend de la base d'expression (2 ou 10)

Types de données fondamentaux

- bit : □ 1
- nibble : □□□□ 4
- byte/octet : □□□□□□□□ 8
- word/mot : □□□□□□□□□□□□□□□□ 16
- doubleword/mot double : □□□□□□□□ . . . □□□□□□□□ 32

La mémoire

- On veut stocker :
  - des entiers positifs (12, 534256, . . .)
  - des entiers négatifs (-56, -435345, . . .)
  - des caractères ('a', 'Z', '5', '+', . . .)
  - des chaînes de caractères ("bonjour", . . .)
  - des réels (12.34, -670.5552, . . .)
  - des instructions
- PB : une case mémoire contient uniquement des bits
- SOL : tout coder sous forme d'entiers positifs en binaire



Représentation « décimale codée binaire »

- Codage Binary Coded Decimal, gourmand (12 bits seulement en binaire pour coder 3406)
- Opérations arithmétiques pas faciles et pas efficaces

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
xxxx	illégal

Décimal : 3 4 0 6
BCD : 0011 0100 0000 0110

14 bits au minimum

Représentation positionnelle

- Choix d'une base b : (ex. : 10, 2, . . .)
- Choix de b symboles
- Par exemple, Base 10 : $12 = 1 * 10^1 + 2 * 10^0$
- Base 2 (0, 1) : $1100_2 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0 = 12_{10}$
- Base 3 (★, ♠, ♣) :
  ♣★♠★♣ $= 2 * 3^4 + 0 * 3^3 + 1 * 3^2 + 0 * 3^1 + 2 * 3^0 = 173$

Représentation positionnelle

- Expression d'un nombre a en base b :
  $a_b = (a^n a^{n-1} \dots a^2 a^1 a^0 . a^{-1} a^{-2} \dots a^{-m})_b$
  $= a^n b^n + \dots + a^2 b^2 + a^1 b + a^0 + a^{-1} b^{-1} + \dots a^{-m} b^{-m}$
- Exemples :
  $98_{10} = 9 * 10^1 + 8 * 10^0$
  $101_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 5_{10}$
  $136_8 = 1 * 8^2 + 3 * 8^1 + 6 * 8^0 = 94_{10}$
  $3A_{16} = 3 * 16^1 + 10 * 16^0 = 58_{10}$
  $110.01_2 = 1 * 2^2 + 1 * 2^1 + 0 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} = 6.25_{10}$

Passer de la base 10 à une autre base

- Pour les *nombres entiers*, on procède par divisions successives.
- On divise le nombre par la base, puis le quotient obtenu par la base, et ainsi de suite jusqu'a obtention d'un quotient nul.
- La suite des restes obtenus correspond aux chiffres dans la base visée, $a_0 a_1 \dots a_n$.
- Exemple : $(23)_{10}$ vers la base 10, 2 et 3

$23 = 2 * 10 + 3$	$23 = 11 * 2 + 1$	$23 = 7 * 3 + 2$
$2 = 0 * 10 + 2$	$11 = 5 * 2 + 1$	$7 = 2 * 3 + 1$
	$5 = 2 * 2 + 1$	$2 = 0 * 3 + 2$
	$2 = 1 * 2 + 0$	
	$1 = 0 * 2 + 1$	

Donc $(23)_{10} = (23)_{10} = (10111)_2 = (212)_3$.

Codage des entiers positifs (non-signés)

- Stockage d'un entier positif en mémoire :
  1. Représentation du nombre en binaire
  2. Découpage de la représentation binaire en octets
  3. Stockage de chaque octet consécutivement
- Exemple :
  (1) $(1345)_2 = 10101000001$
  (2)$(1345)_2 = 00000101\ 01000001$
  (3) En mémoire :

	Little endian	Big endian
	□□□□□□□□	□□□□□□□□
	□□□□□□□□	□□□□□□□□
	0 0 0 0 0 1 0 1	0 1 0 0 0 0 0 1
	0 1 0 0 0 0 0 1	0 0 0 0 0 1 0 1
	...	...
0	□□□□□□□□	□□□□□□□□

Big endian vs. little endian

- Architecture ix86 : adressage par octet little-endian
- Stockage d'infos sur plus d'un octet :
  - msb (Most Significant Byte – bit de poids fort) à l'adresse la plus petite = big-endian
  - lsb (Least Significant Byte – bit de poids faible) à l'adresse la plus petite = little-endian
- Exemple $x = 3345467 = (00330c3b)_{16}$

	Big endian			Little endian	
10000005	...		10000005	...	
10000004	3b	$\leftarrow x$	10000004	00	
10000003	0c		10000003	33	
10000002	33		10000002	0c	
10000001	00		10000001	3b	$\leftarrow x$
10000000	...		10000000	...	

Les entiers positifs (non-signés)

Entiers représentables sur 1 octet :

Base 2	Base 10	Base 16
11111111	255	FF
11111110	254	FE
00010001	17	11
00010000	16	10
00001111	15	0F
00001110	14	0E
00001101	13	0D
00001100	12	0C
00001011	11	0B
00001010	10	0A
00001001	9	09
00001000	8	08
00000111	7	07
00000110	6	06
00000101	5	05
00000100	4	04
00000011	3	03
00000010	2	02
00000001	1	01
00000000	0	00

Passage de base 2 à base 16 et inversement :

$$(\underbrace{0010}_{2}\ \underbrace{0010}_{2}\ \underbrace{1001}_{9})_2 = (29)_{16}$$

Les entiers signés

- Entiers non-signés : ensemble d'entiers positifs
- Entiers signés : ensemble d'entiers positifs et négatifs
- Comment représenter des entiers négatifs ?
- Convention de recodage des chaînes de bits
  - Magnitude signée
  - Complément à 1
  - Complément à 2
  - Biaisée

Représentation par magnitude signée

- Dans un entier de k bits, le bit de poids fort code le signe :
  - 0 = positif
  - 1 = négatif
- Exemples (sur 8 bits) :
  - $+25_{10} = 00011001_2$
  - $-25_{10} = 10011001_2$
- Inconvénient = deux représentations pour 0 :
  - $+0_{10} = 00000000_2$
  - $-0_{10} = 10000000_2$
- Sur 8 bits : $-127 \dots +127$

chaîne de bits	non signé	signé
1111	15	-7
1110	14	-6
1101	13	-5
1100	12	-4
1011	11	-3
1010	10	-2
1001	9	-1
1000	8	-0
0111	7	7
0110	6	6
0101	5	5
0100	4	4
0011	3	3
0010	2	2
0001	1	1
0000	0	0

Représentation par complément à 1

- Dans un entier de k bits, le bit de poids fort code le signe :
  - 0 = positif
  - 1 = négatif
- Un nombre négatif s'obtient en complémentant bit à bit sa valeur absolue avec 1 (cf. complément à 9 de la pascaline)
- Exemple pour -25_{10} :
  - $25_{10} = 00011001_2$
  - d'où $-25_{10} = 11100110_2$
- Inconvénient = deux représentations pour 0 :
  - $+0_{10} = 00000000_2$
  - $-0_{10} = 11111111_2$
- Sur 8 bits : $-127 \dots +127$

chaîne de bits	non signé	signé
1111	15	-0
1110	14	-1
1101	13	-2
1100	12	-3
1011	11	-4
1010	10	-5
1001	9	-6
1000	8	-7
0111	7	7
0110	6	6
0101	5	5
0100	4	4
0011	3	3
0010	2	2
0001	1	1
0000	0	0

Représentation par complément à 2

- Dans un entier de k bits, le bit de poids fort code le signe :
  - 0 = positif
  - 1 = négatif
- Un nombre négatif s'obtient en ajoutant 1 au complément à 1 de sa valeur absolue (et inversement).
- Exemple pour -25_{10} :
  - $25_{10} = 00011001_2$
  - complément à 1 de $25_{10} = 11100110_2$
  - ajout de 1 : $-25_{10} = 11100111_2$
- une seule représentation pour 0 : $+0_{10} = -0_{10} = 00000000_2$
- Sur 8 bits : $-128 \ldots +127$

chaîne de bits	non signé	signé
1111	15	-1
1110	14	-2
1101	13	-3
1100	12	-4
1011	11	-5
1010	10	-6
1001	9	-7
1000	8	-8
0111	7	7
0110	6	6
0101	5	5
0100	4	4
0011	3	3
0010	2	2
0001	1	1
0000	0	0

Représentation biaisée

- Représentation des nombres négatifs par ajout d'un biais les rendant positifs.
- Le biais est ajouté aussi aux nombres positifs
- Exemple de codage sur 8 bits avec un biais de $2^{8-1} - 1 = 127$:
  - $-12_{10} = -12 + 127 = 115 = 01110011_2$
  - $30_{10} = 30 + 127 = 157 = 10011101_2$
- une seule représentation pour 0 : $+0_{10} = -0_{10} = 01111111_2$
- Sur 8 bits : $-127 \ldots +128$

chaîne de bits	non signé	signé
1111	15	8
1110	14	7
1101	13	6
1100	12	5
1011	11	4
1010	10	3
1001	9	2
1000	8	1
0111	7	0
0110	6	-1
0101	5	-2
0100	4	-3
0011	3	-4
0010	2	-5
0001	1	-6
0000	0	-7

Les nombres négatifs : résumé

binaire	décimal	signe + magnitude	complément à 1	complément à 2	représentation biaisée
0000	0	0	0	0	-7
0001	1	1	1	1	-6
0010	2	2	2	2	-5
0011	3	3	3	3	-4
0100	4	4	4	4	-3
0101	5	5	5	5	-2
0110	6	6	6	6	-1
0111	7	7	7	7	0
1000	8	-0	-7	-8	1
1001	9	-1	-6	-7	2
1010	10	-2	-5	-6	3
1011	11	-3	-4	-5	4
1100	12	-4	-3	-4	5
1101	13	-5	-2	-3	6
1110	14	-6	-1	-2	7
1111	15	-7	-0	-1	8 (biais = 7)

- Quelle est la valeur de la chaîne de bits : 1010 ?

Représentation des caractères

- Plusieurs formats pour représenter des caractères (imprimables et de contrôle) sous forme binaire :
  - EBCDIC (*Extended Binary-Coded Decimal Interchange Code*)
    - Représentation sur 8 bits (256 caractères possibles)
    - Utilisé autrefois sur les mainframes IBM
  - ASCII (*American Standard Code for Information Interchange*)
    - Représentation sur 7 bits (pas d'accents)
    - ASCII étendu : sur 8 bits mais pas de normalisation
  - Unicode : encodage sur 16 bits (65536 possibilités) pour représenter tous les caractères de toutes les langues

Exemple : table ASCII restreinte

USASCII code chart

b_4 b_3 b_2 b_1	b_7 b_6 b_5 = 000	001	010	011	100	101	110	111
0 0 0 0	0 NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1 SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2 STX	DC2	"	2	B	R	b	r
0 0 1 1	3 ETX	DC3	#	3	C	S	c	s
0 1 0 0	4 EOT	DC4	$	4	D	T	d	t
0 1 0 1	5 ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6 ACK	SYN	&	6	F	V	f	v
0 1 1 1	7 BEL	ETB	'	7	G	W	g	w
1 0 0 0	8 BS	CAN	(	8	H	X	h	x
1 0 0 1	9 HT	EM	)	9	I	Y	i	y
1 0 1 0	10 LF	SUB	*	:	J	Z	j	z
1 0 1 1	11 VT	ESC	+	;	K	[	k	{
1 1 0 0	12 FF	FS	,	<	L	\	l	|
1 1 0 1	13 CR	GS	-	=	M	]	m	}
1 1 1 0	14 SO	RS	.	>	N	^	n	~
1 1 1 1	15 SI	US	/	?	O	_	o	DEL

$A = 65_{10}$

Opérations arithmétiques

- Opérations $+, -, \times, \div$ sur :
  - Nombres non-signés
  - Nombres signés *en complément à 2*
- Le calcul se fait indépendamment de l'interprétation des chaînes de bits.
- Pas l'interprétation du résultat ...

Addition binaire entière

- L'addition se fait classiquement avec les règles :
  $0 + 0 = 0$
  $0 + 1 = 1$
  $1 + 0 = 1$
  $1 + 1 = 0$ avec retenue de 1
- Exemples :

```
   11                  111111
  00011010  26       00010011  19
+ 00001100  12     + 01111110  62
  00100110  38       01010001  81
```

- Résultat sur 9 bits :
  - Non signé : dépassement de capacité
  - Signé : pas de signification

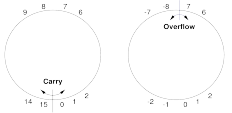
Soustraction binaire entière

- La soustraction suit les règles suivantes :
  $0 - 0 = 0$
  $0 - 1 = 1$ et on prend 1 à gauche
  $1 - 0 = 1$
  $1 - 1 = 0$
- Exemples :

```
  00100101  37       00110011  51
- 00010001  17     - 00010110  22
    1                  111
  00010100  20       00011101  29
```

- On peut aussi faire une addition avec le complément à 2 du deuxième opérande. $37 - 17 = 37 + (-17)$

Carry vs. Overflow



Non signé		Signé		Non signé		Signé	
1110	14	1110	-2	0111	7	0111	7
+ 1011	11	+ 1011	-5	+ 0001	1	+ 0001	1
11001	25>15	11001	-7	1000	8	1000	-8

Détection :
1 bit de plus en sortie

Détection :
Opérandes de même signe
Résultat de signe différent

Multiplication binaire entière

- La multiplication suit les règles suivantes :

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

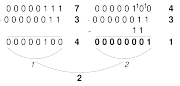
- Exemple :

$$\begin{array}{rr} 00101001 & 41 \\ \times\ 00000110 & 6 \\ \hline 00000000 & \\ 00101001 & \\ 00101001 & \\ \hline 00111110 & 246 \end{array}$$

- On peut aussi faire des additions itérées

Division binaire entière

- Division obtenue par itération de soustractions jusqu'à ce que le résultat de la soustraction soit inférieur au diviseur :
  - Quotient = nombre de soustractions
  - Reste = résultat de la dernière soustraction
- Exemple : division de 7 par 3



- Résultat : quotient = 2 et reste = 1
- On peut aussi faire comme une division classique en décimal

Calcul sur les réels (1)

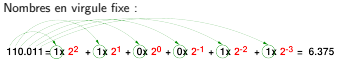
- Infinité de nombres entiers
  - ⇒ *Mais* représentation correcte dans un intervalle
- Infinité de nombres réels
  - Impossibilité de représentation correcte même d'un petit intervalle :

$$\forall a, b \in \mathbb{R}\ \exists c \in \mathbb{R} \quad \text{t.q.} \quad a \leq c \leq b$$

⇒ Représentation d'un sous-ensemble de $\mathbb{Q}$

Calcul sur les réels (2)

- Nombres en virgule fixe :



- Nombres en virgule flottante (notation scientifique) :

$$101010.10 = 1.0101010 \times 2^5$$
$$0.0010001 = 1.00011 \times 2^{-3}$$

⇒ Usage de l'arithmétique en virgule flottante majoritaire

Changement de base pour les nombres réels

- Passage d'un nombre réel de base 10 vers base 2 en virgule fixe :
  - Partie entière : comme pour les entiers
  - Partie décimale : multiplications itérées par 2
- Exemple : conversion de 14.375_{10} en base 2 ?
  - $14_{10} = 1110_2$ (divisions itérées par 2)
  - $0.375_{10} = ???_2$

$$\begin{array}{lll} 0.375 & \times 2 & = \mathbf{0}.75 \\ 0.75 & \times 2 & = \mathbf{1}.5 \\ 0.5 & \times 2 & = \mathbf{1}.0 \end{array}$$

- Résultat : $14.375_{10} = 1110.\mathbf{011}_2$

Représentation des flottants

- Nombre flottant x en binaire :
  - un bit de signe s
  - un exposant E
  - une mantisse m

$$\mathbf{x} = (-\mathbf{1})^{\mathbf{s}} \times \mathbf{m} \cdot \mathbf{2}^{\mathbf{E}}$$

- Représentations équivalentes :
  - (a) $\mathbf{0.0000011}1010 \cdot 2^0$
  - (b) $\mathbf{0.000000}111010 \cdot 2^2$
  - (c) $\mathbf{1.111010} \cdot 2^{-5}$
- Taille de mantisse fixée ⇒ forme (c) plus précise

Représentation IEEE 754

- Représentation *normalisée* (forme (c)) ;
- Toujours un 1 avant la virgule ⇒ pas codé (*hidden bit*)

$$\mathbf{m} = \mathbf{1.00101} \quad \longrightarrow \quad \mathbf{f} = \mathbf{00101}$$

- exposants négatifs et positifs : codage par biais :

$$\mathbf{E} \quad \longrightarrow \quad \mathbf{e} = \mathbf{E} + \text{biais}$$

	s	e	f
single (1,8,23)	1	10101010101	101010101010...10100101
double (1,11,52)			
ix87 reg. (1,15,64)			

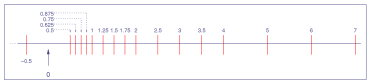
Signe *Exposant biaisé* *Partie fractionnaire*

Représentation IEEE 754 - Exemple du format tiny

- Exemple : format `tiny` sur 5 bits $(1, 2, 2)$ de biais 1 :
- Nombres positifs représentables :

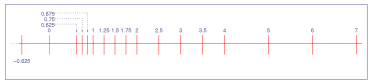
$$\begin{array}{ll}
\mathbf{0\,00\,00} \rightsquigarrow \mathbf{1.00} \times \mathbf{2^{-1}} = \mathbf{0.5} & \mathbf{0\,10\,00} \rightsquigarrow \mathbf{1.00} \times \mathbf{2^{1}} = \mathbf{2} \\
\mathbf{0\,00\,01} \rightsquigarrow \mathbf{1.01} \times \mathbf{2^{-1}} = \mathbf{0.625} & \mathbf{0\,10\,01} \rightsquigarrow \mathbf{1.01} \times \mathbf{2^{1}} = \mathbf{2.5} \\
\mathbf{0\,00\,10} \rightsquigarrow \mathbf{1.10} \times \mathbf{2^{-1}} = \mathbf{0.75} & \mathbf{0\,10\,10} \rightsquigarrow \mathbf{1.10} \times \mathbf{2^{1}} = \mathbf{3} \\
\mathbf{0\,00\,11} \rightsquigarrow \mathbf{1.11} \times \mathbf{2^{-1}} = \mathbf{0.875} & \mathbf{0\,10\,11} \rightsquigarrow \mathbf{1.11} \times \mathbf{2^{1}} = \mathbf{3.5} \\
\mathbf{0\,01\,00} \rightsquigarrow \mathbf{1.00} \times \mathbf{2^{0}} = \mathbf{1} & \mathbf{0\,11\,00} \rightsquigarrow \mathbf{1.00} \times \mathbf{2^{2}} = \mathbf{4} \\
\mathbf{0\,01\,01} \rightsquigarrow \mathbf{1.01} \times \mathbf{2^{0}} = \mathbf{1.25} & \mathbf{0\,11\,01} \rightsquigarrow \mathbf{1.01} \times \mathbf{2^{2}} = \mathbf{5} \\
\mathbf{0\,01\,10} \rightsquigarrow \mathbf{1.10} \times \mathbf{2^{0}} = \mathbf{1.5} & \mathbf{0\,11\,10} \rightsquigarrow \mathbf{1.10} \times \mathbf{2^{2}} = \mathbf{6} \\
\mathbf{0\,01\,11} \rightsquigarrow \mathbf{1.11} \times \mathbf{2^{0}} = \mathbf{1.75} & \mathbf{0\,11\,11} \rightsquigarrow \mathbf{1.11} \times \mathbf{2^{2}} = \mathbf{7}
\end{array}$$

Représentation IEEE 754 - Exemple du format tiny



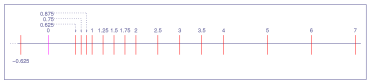
- Pas de codage pour 0

Représentation IEEE 754 - Exemple du format tiny



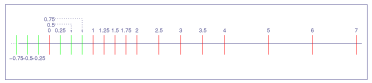
- Pas de codage pour 0
  - Réserver 0 00 00 et 1 00 00 pour ± 0 (perte de ± 0.5)

Représentation IEEE 754 - Exemple du format tiny



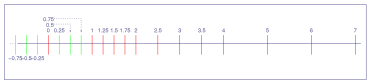
- Pas de codage pour 0
  - Réserver 0 00 00 et 1 00 00 pour ± 0 (perte de ± 0.5)
- Grand trou autour de 0

Représentation IEEE 754 - Exemple du format tiny



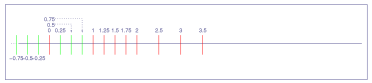
- Pas de codage pour 0
  - Réserver 0 00 00 et 1 00 00 pour ± 0 (perte de ± 0.5)
- Grand trou autour de 0
  - Réserver $e = 0$ pour les nombres *dénormalisés*
    - $\Rightarrow$ Plus de *hidden bit* à 1

Représentation IEEE 754 - Exemple du format tiny



- Pas de codage pour 0
  - Réserver 0 00 00 et 1 00 00 pour ± 0 (perte de ± 0.5)
- Grand trou autour de 0
  - Réserver $e = 0$ pour les nombres *dénormalisés*
    - $\Rightarrow$ Plus de *hidden bit* à 1
- Notions d'infinis mathématiques et de résultat indéfini :

Représentation IEEE 754 - Exemple du format tiny



- Pas de codage pour 0
  - Réserver 0 00 00 et 1 00 00 pour ± 0 (perte de ± 0.5)
- Grand trou autour de 0
  - Réserver $e = 0$ pour les nombres *dénormalisés*
    - $\Rightarrow$ Plus de *hidden bit* à 1
- Notions d'infinis mathématiques et de résultat indéfini :
  - Réserver $e = 3$ (11)

Représentation IEEE 754 - Exemple du format tiny

- Interprétation des bits :

$$\begin{cases}
\mathbf{e = 3}, & \mathbf{f \neq 0} & : \mathbf{v} = \text{NaN} \\
\mathbf{e = 3}, & \mathbf{f = 0} & : \mathbf{v} = (-1)^{\mathbf{s}} \times \infty \\
\mathbf{0 < e < 3} & & : \mathbf{v} = (-1)^{\mathbf{s}} \times (\mathbf{1.f}) \cdot 2^{\mathbf{e}-1} \\
\mathbf{e = 0}, & \mathbf{f \neq 0} & : \mathbf{v} = (-1)^{\mathbf{s}} \times (\mathbf{0.f}) \cdot 2^{\mathbf{0}} \\
\mathbf{e = 0}, & \mathbf{f = 0} & : \mathbf{v} = (-1)^{\mathbf{s}} \times \mathbf{0}
\end{cases}$$

00000	0.5	00000	0
00001	0.625	00001	0.25
00010	0.75	00010	0.5
00011	0.875	00011	0.75
00100	1	00100	1
00101	1.25	00101	1.25
00110	1.5	00110	1.5
00111	1.75	00111	1.75
01000	2	01000	2
01001	2.5	01001	2.5
01010	3	01010	3
01011	3.5	01011	3.5
01100	4	01100	$+\infty$
01101	5	01101	NaN
01110	6	01110	NaN
01111	7	01111	NaN

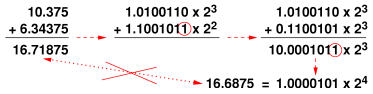
Représentation IEEE 754 - simple précision

Sign	Exponent (e)	Fraction (f)	Value
0	00..00	00..00	$+0$
0	00..00	00..01 11..11	Positive Denormalized Real $0.f * 2^{(-b+1)}$
0	00..01 11..10	XX..XX	Positive Normalized Real $1.f * 2^{(e-b)}$
0	11..11	00..00	$+$Infinity
0	11..11	00..01 11..11	NaN
1	00..00	00..00	-0
1	00..00	00..01 11..11	Negative Denormalized Real $-0.f * 2^{(-b+1)}$
1	00..01 11..10	XX..XX	Negative Normalized Real $-1.f * 2^{(e-b)}$
1	11..11	00..00	$-$Infinity
1	11..11	00..01 11..11	NaN

$2^{-149} \leq PDR \leq 2^{-126}(1-2^{-23})$ et $2^{-126} \leq PNR \leq 2^{127}(2-2^{-23})$

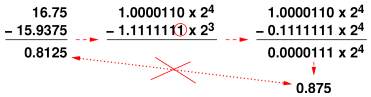
Addition en nombres flottants

- Addition possible si et seulement si les opérandes ont même exposant
- Exposants différents ⇒ décalage du nombre de *plus petit* exposant
- Exemple :



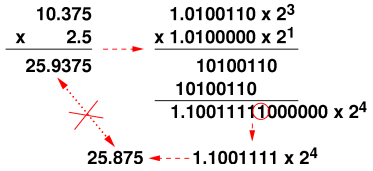
Soustraction en nombres flottants

- Soustraction possible si et seulement si les opérandes ont même exposant
- Exposants différents ⇒ décalage du nombre de *plus petit* exposant
- Exemple :



Multiplication en nombres flottants

- Multiplication des mantisses et ajout des exposants
- Exemple :



Plan

Une première définition

- Un *ordinateur* est une machine de traitement de l'information
  - pouvant acquérir de l'information,
  - la stocker,
  - la transformer.
- L'*information* correspond à tout ensemble de données et est stockée en binaire.

En quelques mots

- Les deux principaux constituants d'un ordinateur sont la mémoire principale et le processeur.
- La mémoire principale permet de stocker de l'information (programmes et données), tandis que le processeur exécute pas à pas les instructions composant les programmes.
- Un *programme* est une suite d'instructions élémentaires exécutées dans l'ordre par le processeur.
- Ces instructions correspondent à des actions très simples, comme additionner deux nombres, lire ou écrire une case mémoire, etc.

Notion de programme

- Chaque instruction d'un programme est codifiée en mémoire sur quelques octets.
- Le processeur est capable d'exécuter des programmes en langage machine, c'est à dire composés d'instructions très élémentaires suivant un codage précis.
- Chaque type de processeur est capable d'exécuter un certain ensemble d'instructions, son jeu d'instructions.
- Pour écrire un programme en langage machine, il faut donc connaître les détails du fonctionnement du processeur qui va être utilisé.

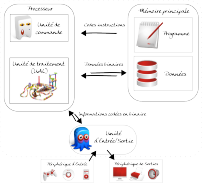
Le processeur

- Le processeur est un circuit éléctronique complexe qui exécute chaque instruction très rapidement, en quelques cycles d'horloges.
- Toute l'activité de l'ordinateur est cadencée par une horloge unique, de façon à ce que tous les circuits électroniques travaillent en- sembles.
- La fréquence de cette horloge s'exprime en MHz (millions de battements par seconde).
- Pour chaque instruction, le processeur effectue schématiquement les opérations suivantes :
  - lire en mémoire (MP) l'instruction à exécuter ;
  - effectuer le traitement correspondant ;
  - passer à l'instruction suivante.

Le processeur

Le processeur est divisé en deux parties :

- l'unité de commande est responsable de la lecture en mémoire et du décodage des instructions ;
- l'unité de traitement, aussi appelée Unité Arithmétique et Logique (U.A.L.), exécute les instructions qui manipulent les données.

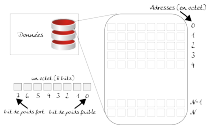


Structure de la mémoire principale

- La mémoire est divisée en emplacements de taille fixe (par exemple 8 bits) utilisés pour stocker instructions et données.
- En principe, la taille d'un emplacement mémoire pourrait être quelconque ; en fait, la plupart des ordinateurs en service aujourd'hui utilisent des emplacements mémoire d'un octet (byte en anglais, soit 8 bits, unité pratique pour coder un caractère par exemple).

Structure de la mémoire principale

- Dans une mémoire de taille N, on a N emplacements mémoires, numérotés de 0 à $N-1$.
- Chaque emplacement est repéré par son numéro, appelé adresse.
- L'adresse est le plus souvent écrite en hexadécimal.
- La capacité (taille) de la mémoire est le nombre d'emplacements, exprimé en général en kilo-octets ou en méga-octets, voire davantage.



Opérations sur la mémoire

- Seul le processeur peut modifier l'état de la mémoire[1]
- Chaque emplacement mémoire conserve les informations que le processeur y écrit jusqu'à coupure de l'alimentation électrique, où tout le contenu est perdu (contrairement aux mémoires externes comme les disquettes et disques durs).
- Les seules opérations possibles sur la mémoire sont :
  - écriture d'un emplacement : le processeur donne une valeur et une adresse, et la mémoire range la valeur à l'emplacement indiqué par l'adresse ;
  - lecture d'un emplacement : le processeur demande à la mémoire la valeur contenue à l'emplacement dont il indique l'adresse. Le contenu de l'emplacement lu reste inchangé.

[1] Sur certains ordinateurs, les controleurs d'entrées/sorties peuvent accéder directement à la mémoire (accès DMA), mais cela ne change pas le principe de fonctionnement.

Unité de transfert

- Notons que les opérations de lecture et d'écriture portent en général sur plusieurs octets contigüs en mémoire : un mot mémoire.
- La taille d'un mot mémoire dépend du type de processeur ; elle est de
  - 1 octet (8 bits) dans les processeurs 8 bits (par exemple Motorola 6502) ;
  - 2 octets dans les processeurs 16 bits (par exemple Intel 8086) ;
  - 4 octets dans les processeurs 32 bits (par ex. Intel 80486 ou Motorola 68030).

Définition

- Le processeur est parfois appelé CPU (de l'anglais *Central Processing Unit*) ou encore MPU (*Micro-Processing Unit*) pour les microprocesseurs.
- Un microprocesseur n'est rien d'autre qu'un processeur dont tous les constituants sont réunis sur la même puce électronique (pastille de silicium), afin de réduire les coûts de fabrication et d'augmenter la vitesse de traitement.
- Les microordinateurs sont tous équipés de microprocesseurs.
- L'architecture de base des processeurs équipant les gros ordinateurs est la même que celle des microprocesseurs.

Les registres et l'accumulateur

- Le processeur utilise toujours des registres.
- Registres = des petites mémoires internes très rapides d'accès utilisées pour stocker temporairement une donnée, une instruction ou une adresse.
- Chaque registre stocke 8, 16 ou 32 bits.
- Le nombre exact de registres dépend du type de processeur et varie typiquement entre une dizaine et une centaine.
- Parmi les registres, le plus important est le registre accumulateur, qui est utilisé pour stocker les résultats des opérations arithmétiques et logiques.
- L'accumulateur intervient dans une proportion importante des instructions.

Les registres et l'accumulateur

- Par exemple, examinons ce qu'il se passe lorsque le processeur exécute une instruction comme "Ajouter 5 au contenu de la case memoire d'adresse 180" :
  - Le processeur lit et décode l'instruction ;
  - le processeur demande à la mémoire la contenu de l'emplacement 180 ;
  - la valeur lue est rangée dans l'accumulateur ;
  - l'unité de traitement (UAL) ajoute 5 au contenu de l'accumulateur ;
  - le contenu de l'accumulateur est écrit en mémoire à l'adresse 180.
- C'est l'unité de commande qui déclenche chacune de ces actions dans l'ordre.
- L'addition proprement dite est effectuée par l'UAL.

Architectures d'un processeur à accumulateur

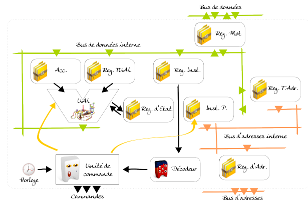


Schéma simplifié d'un processeur

Architectures d'un processeur à accumulateur

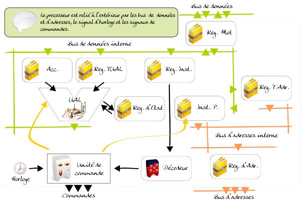


Schéma simplifié d'un processeur

Architectures d'un processeur à accumulateur

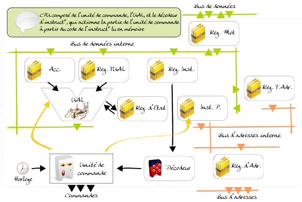


Schéma simplifié d'un processeur

Architectures d'un processeur à accumulateur

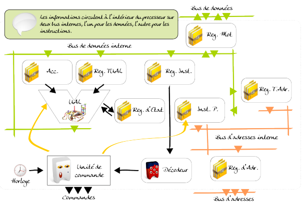


Schéma simplifié d'un processeur

Architectures d'un processeur à accumulateur

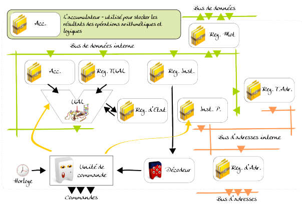


Schéma simplifié d'un processeur

Architectures d'un processeur à accumulateur

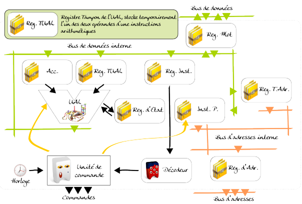


Schéma simplifié d'un processeur

Architectures d'un processeur à accumulateur

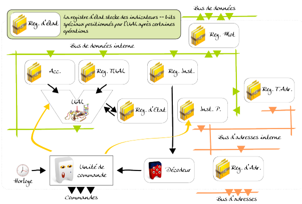


Schéma simplifié d'un processeur

Architectures d'un processeur à accumulateur

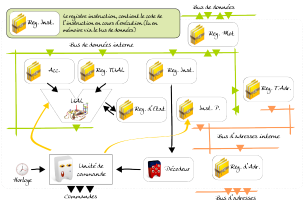


Schéma simplifié d'un processeur

Architectures d'un processeur à accumulateur

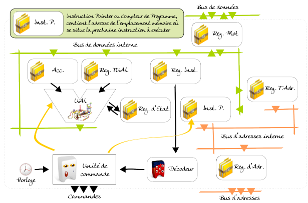


Schéma simplifié d'un processeur

Architectures d'un processeur à accumulateur

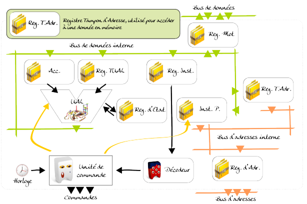


Schéma simplifié d'un processeur

Architectures d'un processeur à accumulateur

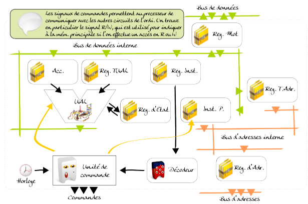


Schéma simplifié d'un processeur

Définition

- Les informations échangées entre la mémoire et le processeur circulent sur des bus.
- Un bus est simplement un ensemble de n fils conducteurs, utilisés pour transporter n signaux binaires.
- Le bus d'adresse est un bus unidirectionnel : seul le processeur envoie des adresses.
- Il est composé de k fils ; on utilise donc des adresses de k bits. La mémoire peut posséder au maximum 2^k emplacements (adresses 0 à $2^k - 1$).
- Le bus de données est un bus bidirectionnel. Lors d'une lecture, c'est la mémoire qui envoie un mot sur le bus (le contenu de l'emplacement demandé) ; lors d'une écriture, c'est le processeur qui envoie la donnée.

Introduction à l'Architecture des ordinateurs

Architecture des ordinateurs

Guillaume Blin

IGM-LabInfo UMR 8049,
Bureau 4B066
Université de Marne La Vallée
gblin@univ-mlv.fr
http://igm.univ-mlv.fr/~gblin