

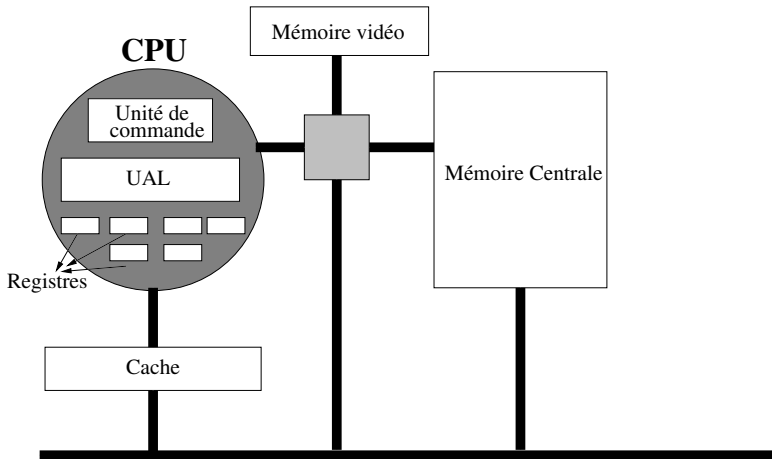
Les principes du calcul sur un ordinateur

De la représentation à la micro architecture

Sophie Robert

Université d'Orléans - LIFO

Schéma très simplifié d'une architecture



Sommaire

- 1 La représentation de l'information
- 2 Transformer un calcul en circuits
- 3 CPU Central Processing Unit
- 4 L'assembleur

La représentation des informations

Des bits

- Le cœur des ordinateurs : des composants électroniques tels que toute information puisse être transformée en un signal électrique.
- Les circuits utilisent des transistors avec deux états (le courant passe ou non) qui permettent de représenter deux informations : 0 ou 1 soit **le bit (binary digit)**

nb bits	représentation	nb info.
1	0, 1	2
2	00, 01, 10, 11	4
3	000, 001, 010, 011, 100, 101, 110, 111	8

D'où l'unité Kbits = 1024 bits = 2^{10} bits

Notations et unités

Des puissances de 2

Symbole	Signification
K (kilo)	$2^{10} = 1024$
M (méga)	$2^{20} = 1048576$
G (giga)	$2^{30} = 1073741824$
T (téra)	$2^{40} = 1099511627776$

Unité	Signification
quartet	4 bits
octet/byte	8 bits
Gb	Gigabit : unité le bit
GB	Gigabyte : unité l'octet

Représentation fonction de l'information

Les données numériques en fonction de leur type

- Les entiers naturels : par changement de base (décimal \rightarrow binaire)
- Les entiers signés : par changement de base + complément à 1 ou complément à 2
- Les flottants : par séparation exposant et mantisse

Les données non numériques en fonction de leur type

- Les alphabets : par des tables et des normes de traduction
- Le son ou l'image : par codage associé à de la compression

Conversion en base b

Définitions

- Soit un entier $b > 1$, tout nombre x peut s'écrire :

$$x = \sum_{i=0}^n a_i b^i \text{ avec } a_i \in \{0, 1, \dots, b-1\}$$

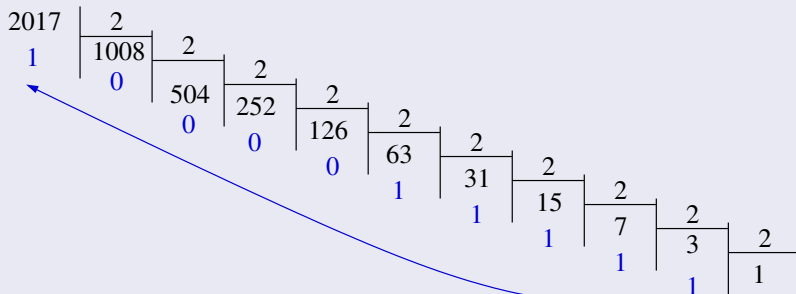
- On dit alors qu'en base b , x s'écrit
 - * $a_n a_{n-1} \dots a_1 a_0$ selon le codage **little endian**,
 - * $a_0 a_1 \dots a_{n-1} a_n$ selon le codage **big endian**.

Ordinateur : base 2

Conversion rapide

- Connaître les puissances de 2
- $2017 = 1024 + 512 + 256 + 128 + 64 + 32 + 1 = 11111100001$

Conversion par divisions successives



Deux autres bases sont également couramment utilisées

La base 8 (octal)

- Exemple

$$2017_{10} = 3 \times 8^3 + 7 \times 8^2 + 4 \times 8^1 + 1 \times 8^0 = 3741_{octal}$$

- A partir de la base 2

011	111	100	001
3	7	4	1

Deux autres bases sont également couramment utilisées

La base 16 (hexadécimal)

- Exemple

$$2017_{10} = 7 \times 16^2 + 14 \times 16^1 + 1 \times 16^0 = 7141_{hexa}$$

- * chiffre 141 ????

- * Alphabet: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

$$2017_{10} = 7E1_{hexa} = 0x7E1$$

- A partir de la base 2

0111	1110	0001
7	E	1

Représentation limitée, à précision finie

Nombre de bits limité

- Un ordinateur va travailler sur un nombre limité de bits pour représenter l'information
 - > On parle alors de **mots** qui sont classiquement de longueur 16, 32 ou 64 bits.
- pour les entiers il s'agit de n'en représenter qu'un sous ensemble avec le risque d'effectuer des opérations induisant un **dépassement de capacité**.
- pour les réels, on parle de **précision finie** avec des arrondis qui dépendent de la représentation.

La représentation des entiers positifs

Les entiers **positifs** sont représentés par leur codage binaire.

- Si n bits sont disponibles, les 2^n entiers de 0 à $2^n - 1$ sont représentés.
- Représentation de X en fonction de n le nombre de bits

n	$x = 20$	$x = 100$	$x = 1025$
4	ddc ^a	ddc	ddc
6	010100	ddc	ddc
8	00010100	01100100	ddc
12	000000010100	000001010000	010000000001
n min	5	7	11

^adépacement de capacité

La représentation des entiers

Comment gérer à la fois les entiers positifs et négatifs ?

- La représentation est sur n bits avec un bit réservé pour le signe.
- On peut représenter les entiers dont la valeur absolue est représentable sur $n - 1$ bits.
- Il existe plusieurs représentations mais dans tous les cas l'entier positif A sera représenté par $0a$ où a est la représentation binaire de A .

Les entiers signés sur n bits

Les règles

- La valeur absolue de l'entier X est représentée sur $n - 1$ bits,
- Le bit de signe est construit explicitement: 0 si l'entier est positif, 1 s'il est négatif.

X	$ X _2$	$n = 4$	$n = 6$	$n = 8$
3	11	0011	000011	00000011
-3	11	1011	100011	10000011
20	10100	ddc	010100	00010100
-20	10100	ddc	110100	10010100
50	110010	ddc	ddc	00110010
-50	110010	ddc	ddc	10110010
100	1100100	ddc	ddc	01100100
-100	1100100	ddc	ddc	11100100

La représentation des entiers en complément à un

Représentation sur n bits

- L'entier positif est codé par sa représentation binaire sur n bits si sa valeur absolue est représentable sur $n - 1$ bits.
 - 1 $|27|_2 = 11011$
 - 2 Sur 8 bits 00011011
 - 3 $27_{cà1} = 00011011$
- L'entier négatif est codé par le complémentaire de sa valeur absolue sur n bits si sa valeur absolue est représentable sur $n - 1$ bits.
 - 1 $|-27|_2 = 11011$
 - 2 Sur 8 bits 00011011
 - 3 $-27_{c1} = 11100100$

La représentation des entiers en complément à un

Caractéristiques de la représentation

- > Double représentation du 0,
- > n bits permettent de représenter les $2^n - 1$ entiers de $-(2^{n-1} - 1)$ à $2^{n-1} - 1$,
- > Le bit de poids fort renseigne toujours sur le signe de l'entier.

Exemples

X	$ X _2$	$n = 4$	$n = 6$	$n = 8$
3	11	0011	000011	00000011
-3	11	1100	111100	11111100
-20	10100	ddc	101011	11101011
-100	1100100	ddc	ddc	10011011

La représentation des entiers en complément à 2

Représentation sur n bits

- L'entier positif est codé par sa représentation binaire sur n bits si sa valeur absolue est représentable sur $n - 1$ bits.
 - 1 $|27|_2 = 11011$
 - 2 Sur 8 bits 00011011
 - 3 $27_{\text{cà1}} = 00011011$
- L'entier négatif est codé par le complément binaire sur n bits auquel on ajoute 1 si sa valeur absolue est représentable sur $n - 1$ bits.
 - 1 $|27|_2 = 11011$
 - 2 Sur 8 bits 00011011
 - 3 $-27_{\text{cà1}} = 11100100$
 - 4 $-27_{\text{cà2}} = 11100101$

La représentation des entiers en complément à 2 sur n bits

Caractéristiques de la représentation

- > La représentation $10...0$ ($-0_{\text{cà}1}$) code une information supplémentaire : -2^{n-1} ,
- > Une unique représentation de 0,
- > n bits permettent de représenter 2^n entiers de -2^{n-1} à $2^{n-1} - 1$,
- > Le bit de poids fort renseigne sur le signe de l'entier.

La représentation des entiers en complément à 2 sur n bits

Exemples

X	$ X _2$	$n = 4$	$n = 6$	$n = 8$
3	11	0011	000011	00000011
-3	11	1101	111101	11111101
-20	10100	ddc	101100	11101100
-100	1100100	ddc	ddc	10011100


Les opérations sur les entiers

l'addition $a + (\mp b)$

- Préservation de l'addition si quelque soit le signe des entiers, l'addition bit à bit des représentations permet de trouver le résultat.
- En fonction de la représentation
 - * Pour les entiers signés la représentation ne préserve pas l'addition
 - * Les deux compléments préservent l'addition avec une règle simple

L'addition pour les entiers signés

Algorithme

- ❶ traiter le bit de signe à part
- ❷ si les bits de signe sont égaux : ajouter bit à bit
 -  il ne faut pas que le résultat soit sur plus de $n-1$ bits
- ❸ si les bits de signe sont différents :
 - * Trouver l'entier le plus grand en valeur absolue et ordonner l'opération de soustraction
 - * Construire explicitement le bit de signe du résultat

La comparaison

- En parcourant de gauche à droite on cherche le premier bit x qui diffère
- L'entier le plus grand en valeur absolue est celui pour lequel $x = 1$

L'addition pour le complément à 1

Nouvelle expression du complément à un sur n bits

$$a \in \mathbb{N} \quad (-a)_{\text{cà1}} = (2^n - 1 + (-a))_2$$

L'addition pour le complément à 1

Règle de l'addition : ajouter la retenue

- si deux entiers positifs : pas de retenue
- si deux entiers négatifs $-a$ et $-b$:
 $2^n - 1 + (-a) + 2^n - 1 + (-b)$ génère une retenue.
 $\textcolor{red}{2}^n - 1 + (-a) + 2^n - 1 + (-b) + \textcolor{red}{1} = 2^n - 1 + (-(a + b))$
- si $a + (-b)$ avec $|a| < |b|$,
 $a + 2^n - 1 + (-b)$ ne génère pas de retenue.
 $a + (2^n - 1) + (-b) = 2^n - 1 + (a - b)$
- si $a + (-b)$ avec $|a| > |b|$,
 $a + 2^n - 1 + (-b)$ génère une retenue.
 $a + \textcolor{red}{2}^n - 1 + (-b) + \textcolor{red}{1} = (a - b)$

L'addition pour le complément à 2

Nouvelle expression du complément à deux sur n bits

$$a \in \mathbb{N} \quad (-a)_{\text{cà2}} = (2^n + (-a))_2 = (-a)_{\text{cà1}} + 1$$

L'addition pour le complément à 2

Règle de l'addition : ajouter la retenue

- si deux entiers positifs : pas de retenue
- si deux entiers négatifs $-a$ et $-b$:
 $2^n + (-a) + 2^n + (-b)$ génère une retenue.
 $\textcolor{red}{2}^n + (-a) + 2^n + (-b) = 2^n + (-(a + b))$
- si $a + (-b)$ avec $|a| < |b|$,
 $a + 2^n + (-b)$ ne génère pas de retenue.
 $a + 2^n + (-b) = 2^n + (a - b)$
- si $a + (-b)$ avec $|a| > |b|$,
 $a + 2^n + (-b)$ génère une retenue. $a + \textcolor{red}{2}^n + (-b) = (a - b)$

L'opération d'addition

Synthèse

- Représentation binaire signée : gérer le bit de signe à part, effectuer la bonne addition et construire le bit de signe du résultat explicitement
- Complément
 - à 1 : ajouter la retenue à l'addition bit à bit
 - à 2 : oublier la retenue de l'addition bit à bit
 - dépassement de capacité : si le bit de signe n'est pas correcte !

Extension du codage binaire pour les nombres décimaux

Définition

$$x(\in \mathbf{Q}) = \sum_{i=-l}^m a_i b^i$$

Exemple

$$x = 15,625 = 1111,101$$

$$0,625 \times 2 = 1,25 - > 1$$

$$0,25 \times 2 = 0,5 - > 0$$

$$0,5 \times 2 = 1 - > 1$$

Représentation normalisée

Nouvelle représentation dite à virgule flottante

Pour toute base b

$$X_b = m \times b^e$$

où m est la **mantisse** et e l'**exposant**.

Exemple : représentation (m, e) pour $x_{10} = 3.14$

- ❶ $(3.14, 0) \rightarrow x = 3.14 \times 10^0$
- ❷ $(314, -2) \rightarrow x = 314 \times 10^{-2}$
- ❸ $(0.314, 1) \rightarrow x = 0.314 \times 10^1$

Représentation normalisée

- Lorsque $1 \leq m \leq b - 1$ la représentation est **normalisée**.

Représentation normalisée

Caractéristiques de la représentation

- Cette représentation normalisée est ainsi unique et permet de s'abstraire de la représentation de la virgule
- Elle est également plus compacte pour une meilleure précision
- Les chiffres significatifs sont représentés par la mantisse et plus le nombre de positions pour la représenter est grand plus la précision sera importante.
- Le nombre de positions pour représenter l'exposant indique l'étendu des nombres représentables.

Représentation normalisée (3/6)

Représentation de la mantisse et de l'exposant

- A partir de la représentation $(-1)^s m \times b^e$ on obtient le codage

S	E	$x(.)M$
---	---	---------

avec

- s codé par $S = 0$ pour un réel positif et $S = 1$ sinon
- m codé par (x, M) de la même manière qu'un entier positif après normalisation avec $x \in \{1, \dots, b-1\}$ codé sur une position
 - * **En binaire, x vaut toujours 1 et n'a pas besoin d'être représenté.**
- e est codé par E selon un **codage par excédent**.
 - * Soit b_s le biais associé au codage par excédent alors E est la représentation binaire de $e + b_s$

Un standard : IEEE 754

32 ou 64 bits

- Deux précisions sont normalisées :

précision	signe	exposant	mantisse	biais
simple	1 bit	8 bits	23 bits	127
double	1 bit	11 bits	52 bits	1023

Exemples

10,5

$$10 = (1010)_2$$

$$0,5 = 0,1$$

$$10,5 = 1010,1$$

$$10,5 = 1,0101 \times 2^3$$

$$M = 010100000000000000000000$$

$$e = 3, E = 127 + 3 = 10000010$$

$$10,5 = \text{01000001001010000000000000000000}$$

$$0x41280000$$

Exemples

-56,3

$$56 = 111000$$

$$0,3 \times 2 = 0,6 \quad 0,4 \times 2 = 0,8$$

$$0,6 \times 2 = 1,2 \quad 0,8 \times 2 = 1,6$$

$$0,2 \times 2 = 0,4 \quad 0,6 \times 2 = 1,2$$

$$56,4 = 111000,0100110011001.. \quad 56,4 = 1,110000100110011001.. \times 2^5$$

$$M = 11000010011001100110011$$

$$e = 5, E = 127 + 5 = 10000100$$

$$-56,3 = \textcolor{red}{1}\textcolor{blue}{0000100}\textcolor{blue}{1}\textcolor{green}{0000100}\textcolor{green}{11001100110011}$$

$$0xC2613333$$

Et puis

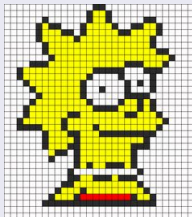
Les caractères

46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[123	{
60	<	92	\	124	

Et puis

Les images

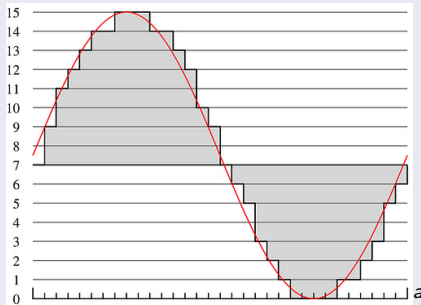
- Un pixel : une couleur en RGB
- une couleur : une représentation binaire par composante



Et puis

Du son

- Echantillonnage et Quantification



^a<http://technicien-du-son.com>

Sommaire

- 1 La représentation de l'information
- 2 Transformer un calcul en circuits**
- 3 CPU Central Processing Unit
- 4 L'assembleur

Faire une addition

Bit à bit

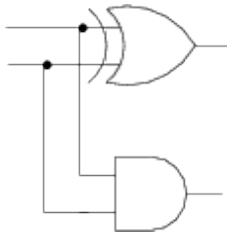
	1	1	1	1	1	1	1	1	0	1
	1	1	1	1	1	0	1	0	1	0
				1	1	1	1	1	1	0
	1	0	0	0	1	1	0	1	0	0

Le demi additionneur

- La table de vérité

<i>a</i>	<i>b</i>	<i>c</i>	<i>ret</i>
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- $f(a, b) = a \oplus b = c$
- ret la retenue générée
 $g(a, b) = a.b = ret$



L'additionneur avec retenue

- La table de vérité

a	b	ret	c	ret^+
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

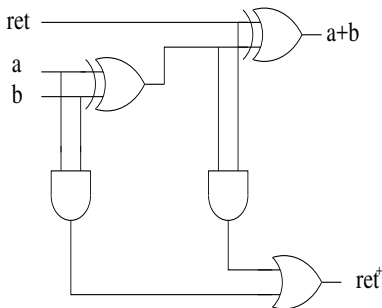
- $f_1(a, b, ret) = c = \bar{a}.\bar{b}.ret + \bar{a}.b.\overline{ret} + a.\bar{b}.\overline{ret} + a.b.ret$
- $f_2(a, b, ret) = ret^+ = \bar{a}.b.ret + a.\bar{b}.ret + a.b.\overline{ret} + a.b.ret$

L'additionneur avec retenue

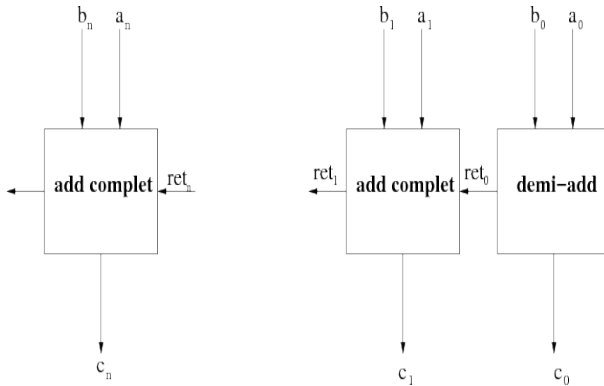
- La simplification de f_1 et f_2 donne les fonctions booléennes suivantes :

$$f_1(a, b, ret) = a \oplus b \oplus ret$$

$$f_2(a, b, ret) = (a \oplus b).ret + a.b$$



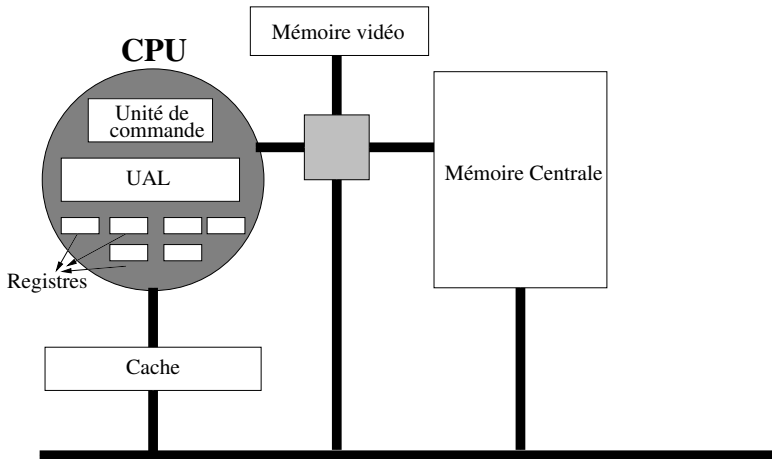
L'additionneur parallèle n bits



Sommaire

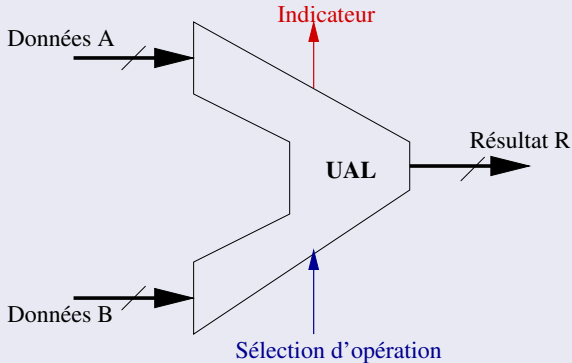
- 1 La représentation de l'information
- 2 Transformer un calcul en circuits
- 3 CPU Central Processing Unit**
- 4 L'assembleur

Schéma très simplifié d'une architecture



L'Unité Arithmétique et Logique (UAL)

Un symbole



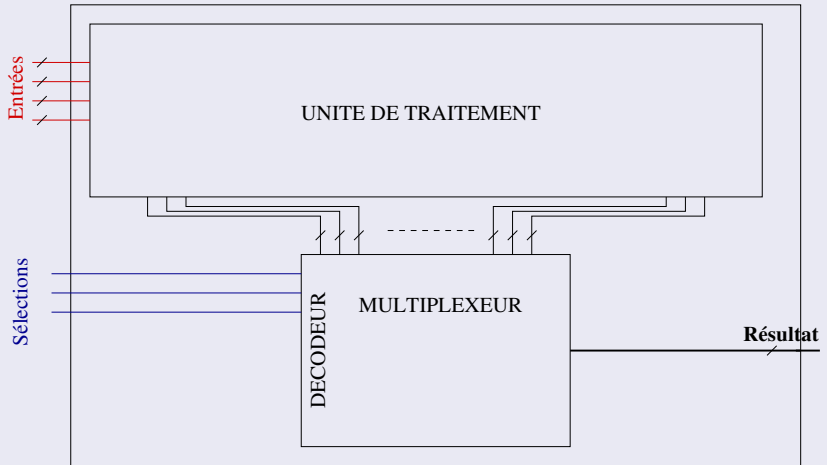
L'UAL

Le lieu des calculs

- Elle doit permettre de réaliser des opérations
 - * logiques (AND, OR, ...),
 - * arithmétiques (ADD, MUL, ...),
 - * diverses (décalage, ...).
- Elle est constituée de portes logiques qui permettent des opérations plus ou moins complexes.
- Elle nécessite
 - * Un **décodeur** pour interpréter les commandes en entrée,
 - * Un **multiplexeur** pour sélectionner la ou les sorties demandées.

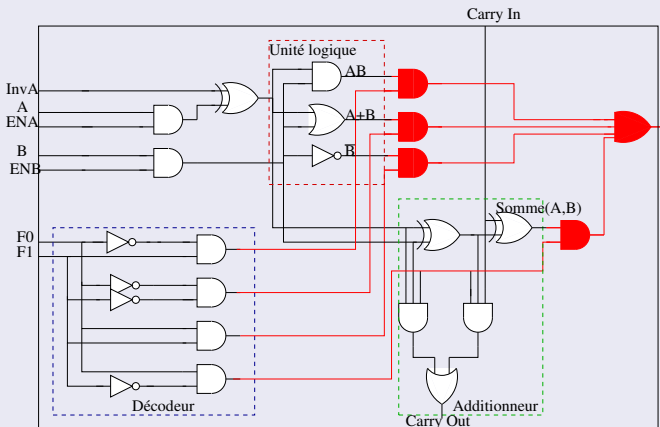
L'UAL

Le lieu de calculs



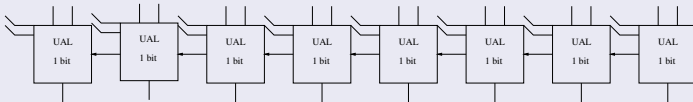
Un exemple UAL sur 1 bit

MIC-1 (Structured Computer Organization Andrew S. Tanenbaum)



Un exemple UAL sur 8 bits

UAL 1 bit enchaînées



- Un ordre de sélection commun à tous les UAL 1 bit
- L'entrée Carry In est la sortie Carry Out précédente

Les calculs sur l'UAL ?

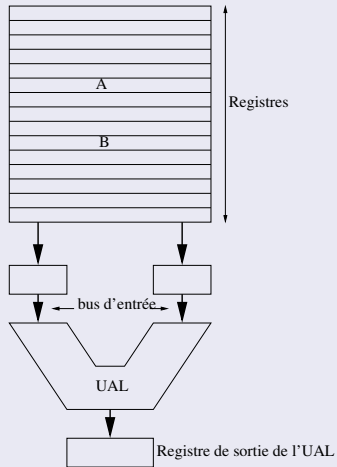
Le chemin de données

C'est l'ensemble des composants et des moyens empruntés par l'information au cours de son traitement.

- Un ensemble de registres
- Des bus
- L'unité arithmétique et logique
- **Et l'enchaînement**

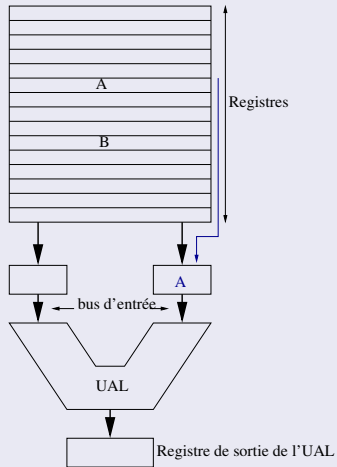
Les calculs sur l'UAL ?

Le chemin de données



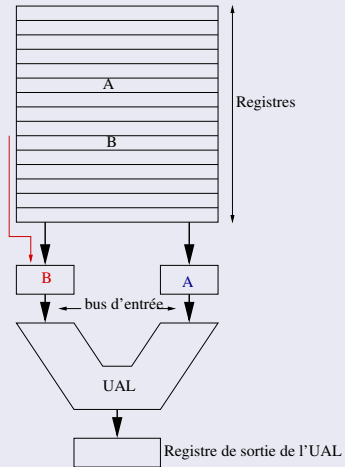
Les calculs sur l'UAL ?

Le chemin de données



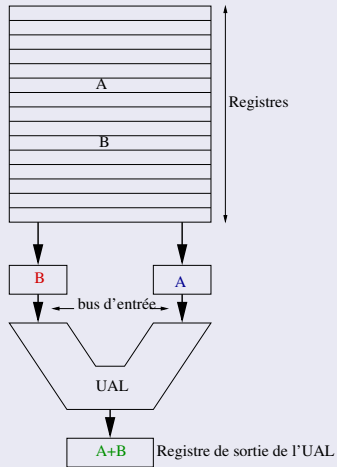
Les calculs sur l'UAL ?

Le chemin de données



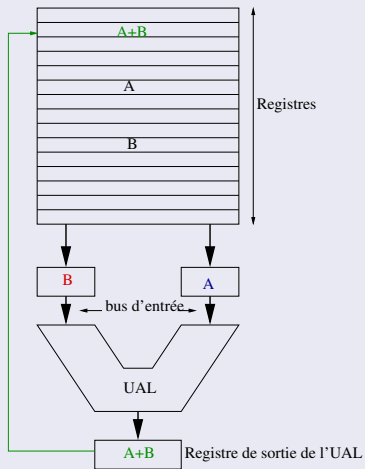
Les calculs sur l'UAL ?

Le chemin de données



Les calculs sur l'UAL ?

Le chemin de données



Un jeu d'instructions par CPU

Fiche technique d'un processeur

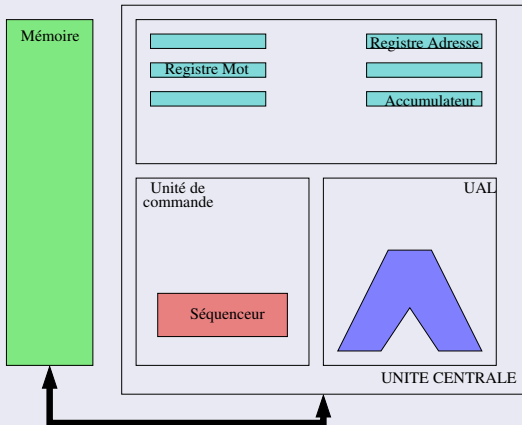
- Un jeu d'instructions : toutes les opérations élémentaires que ce processeur est capable d'exécuter.
- A chaque instruction un cycle d'exécution en micro instructions sur le chemin des données.

Il existe deux familles de processeurs

- RISC (Reduced Instruction Set Computer) : un nombre limité d'instructions associé à un grand nombre de registres pour optimiser le fonctionnement par compilation.
- CISC (Complex Instruction Set Computer) : un grand nombre d'instructions pour simplifier le compilateur.

Une micro architecture par CPU

Les éléments



Le principe de l'unité de commande

Coordination

- C'est un ensemble de dispositifs qui coordonne le fonctionnement du CPU
 - * Un compteur ordinal : numéro de la prochaine instruction à exécuter
 - * Un registre instruction : l'instruction à décoder et exécuter
 - * Un décodeur d'instruction : détermine l'opération à exécuter
 - * Un **séquenceur** : génère les signaux de commandes pour l'unité de traitement (UAL)

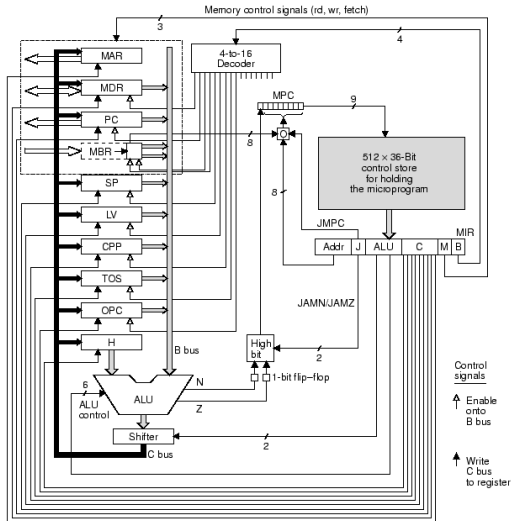
Le jeu d'instructions MIC-1

Code	Mnémonique	Signification
10	BIPUSH octet	
59	DUP	
A7	GOTO offset	
60	IADD	
7E	IAND	
99	IFEQ offset	
9B	IFLT offset	
9F	IF_ICMPEQ offset	
84	IINC numvar const	
15	ILOAD numvar	
B6	INVOKEVIRTUAL dép	
80	IOR	
AC	IRETURN	
36	ISTORE numvar	
64	ISUB	
13	LDC W index	
57	POP	
5F	SWAP	
C4	WIDE	

Le jeu d'instructions MIC-1

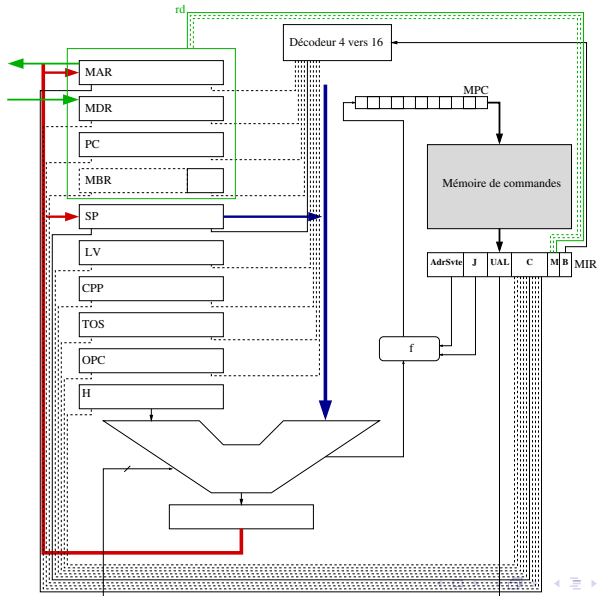
Code	Mnémonique	Signification
10	BIPUSH octet	empile l'octet sur la pile
59	DUP	duplique le mot du sommet de la pile
A7	GOTO offset	branchement inconditionnel
60	IADD	dépile deux fois la pile et empile la somme
7E	IAND	idem avec le ET logique
99	IFEQ offset	dépile et branchement si la valeur est à 0
9B	IFLT offset	dépile et branchement si la valeur est négative
9F	IF_ICMPEQ offset	dépile deux fois et branchement si égalité
84	IINC numvar const	addition
15	ILOAD numvar	empile la variable locale
B6	INVOKEVIRTUAL dép	invoque une procédure
80	IOR	Ou logique sur les deux éléments de la pile
AC	IRETURN	Retour de méthode
36	ISTORE numvar	dépile la pile dans une variable locale
64	ISUB	soustraction sur deux éléments de la pile
13	LDC W index	empile une constante sur le sommet
57	POP	dépile
5F	SWAP	permutation de deux éléments de la pile
C4	WIDE	instruction suivante

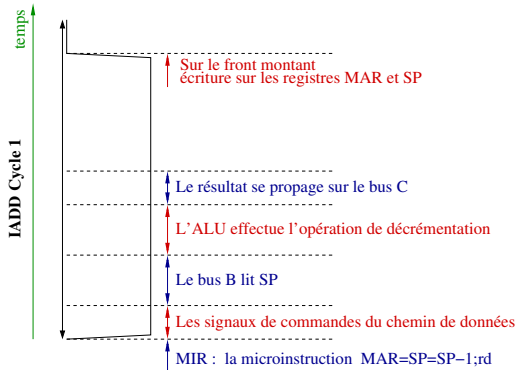
La micro architecture MIC-1

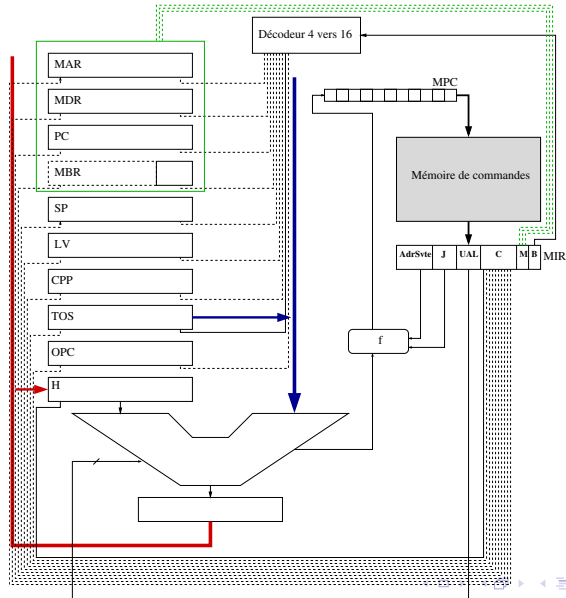


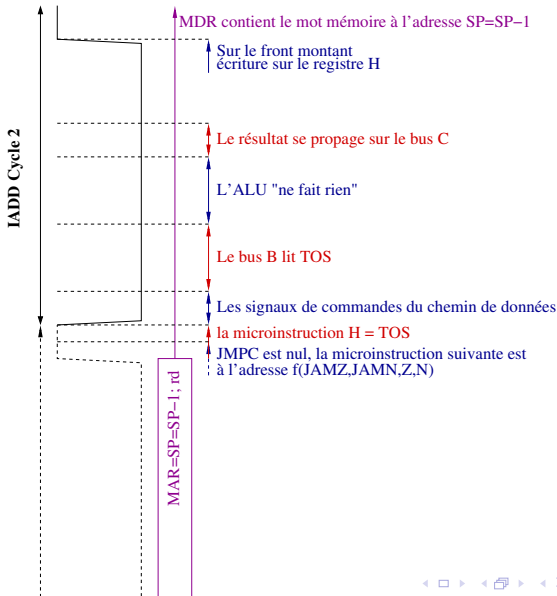
L'instruction IADD en détail

- L'instruction IADD n'a aucun argument, elle utilise l'adressage par pile
 - * Elle dépile deux fois
 - * Elle effectue la somme
 - * Elle empile le résultat
- L'instruction IADD s'effectue en 3 cycles et est décomposée en 3 microinstructions
 - * $MAR = SP = SP - 1; rd$
 - * $H = TOS$
 - * $MDR = TOS = MDR + H; wr; goto Main1$

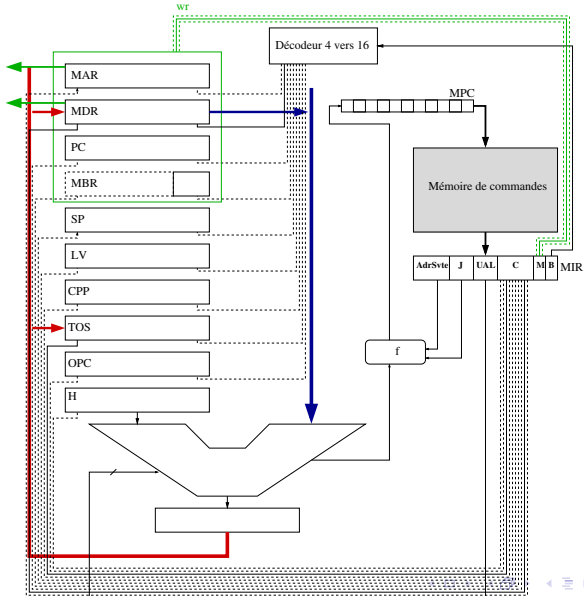


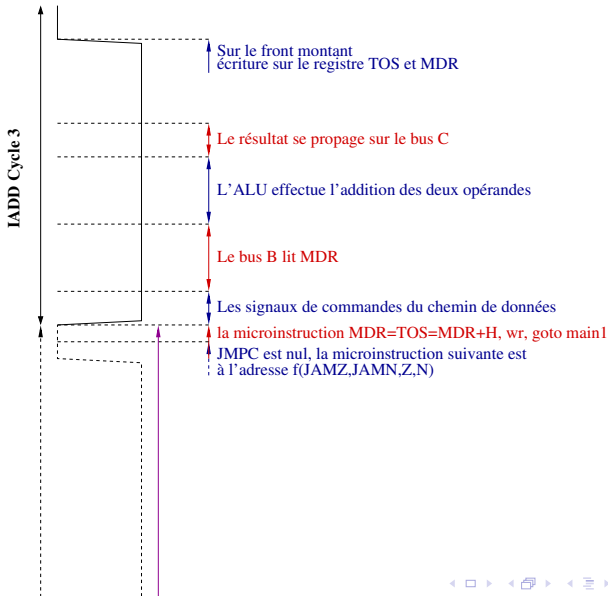


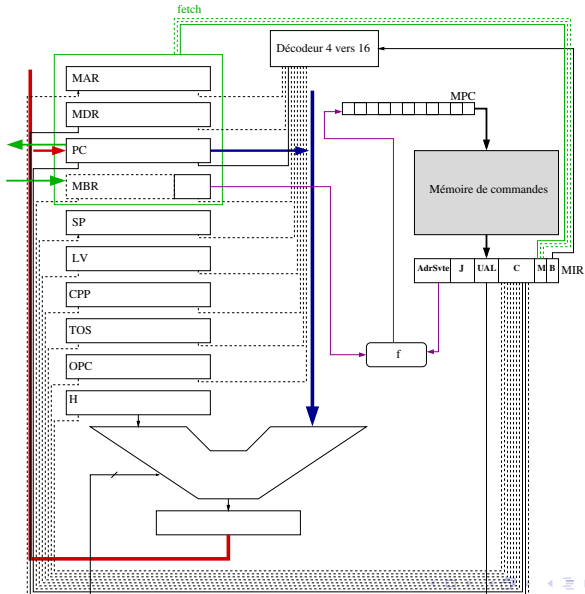


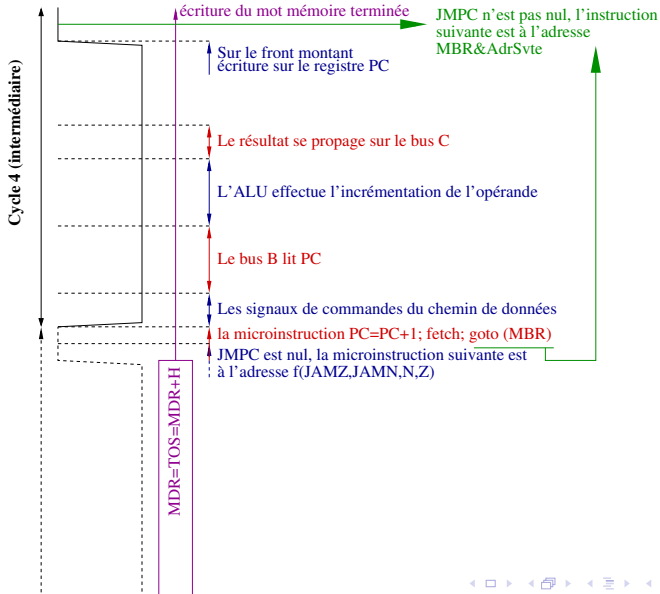


La représentation de l'information
Transformer un calcul en circuits
CPU Central Processing Unit
L'assembleur









Sommaire

- 1 La représentation de l'information
- 2 Transformer un calcul en circuits
- 3 CPU Central Processing Unit
- 4 L'assembleur**

Du langage machine à l'assembleur

- Une instruction machine ou une macroinstruction
 - * c'est un mot binaire **interprétable** par le contrôleur
 - * on lui associe un codage symbolique (mnémonique) pour faciliter sa représentation
- A la place d'écrire directement en langage machine, on utilise l'assembleur ou le langage d'assemblage qui lui est associé (unique pour chaque type de machine)
 - * Ce langage est **traduit** en langage machine
 - * On retrouve le codage symbolique ci-dessus
 - * Plus des directives ou pseudo-instructions
- En général on utilise un assembleur qui est un programme qui traduit du code source en langage machine