

INSTITUT PAUL LAMBIN

BAC 2 INFORMATIQUE DE GESTION

CAHIER DES CHARGES

Outil de gestion des bugs et des demandes d'améliorations

Auteurs :

Christopher SACRÉ

Antoine MANIET

Alexandre MANIET

Timothé BOUVIN

Damien MEUR

Professeur :

B. LEHMANN

27 novembre 2016

Table des matières

1	Histoire de l'informatique	5
1.1	Théorie des logarithmes	5
1.1.1	Définition	5
1.1.2	Propriétés	5
1.2	XVII ^e siècle	5
1.3	XIX ^e siècle	5
1.4	XX ^e siècle	5
1.5	L'ENIAC	6
1.6	Les générations de processeurs/ordinateurs	6
2	Représentation des informations : représentation des nombres	8
2.1	Les différentes bases de représentation des nombres	8
2.2	Les conversions d'une base à une autre	8
2.2.1	Conversion d'un nombre réel en binaire	8
2.3	Addition d'un nombre binaire	9
2.4	Représentation des nombres entiers négatifs	9
2.5	Soustraction d'un nombre binaire	10
2.6	Multiplication d'un nombre binaire	10
2.7	Division d'un nombre binaire	10
2.8	Stockage des nombres réels : virgule fixe	10
2.9	Stockage des nombres réels : virgule flottante	10
2.9.1	Virgule flottante, simple précision (en binaire)	11
2.9.2	Virgule flottante, double précision (en binaire)	11
2.10	Opérations en virgule flottante	11
2.11	L'endianisme	12
3	Représentation des informations : représentation des caractères	12
3.1	L'ASCII	12
3.2	L'unicode	12
4	L'algèbre de Boole	13
4.1	Fonctions logiques et circuits combinatoires	13
4.1.1	Définitions	13
4.1.2	Fonction NON (NOT)	13
4.1.3	Fonction OU (OR)	13
4.1.4	Fonction ET (AND)	14
4.1.5	Propriétés algébrique du ET et du OU logique	14
4.1.6	Fonction OU EXCLUSIF (XOR)	14
4.1.7	Fonction NON OU (NOR)	15
4.1.8	Fonction NON ET (NAND)	15
4.1.9	Fonction NON OU EXCLUSIF (EXNOR)	16
4.1.10	Fonctions universelles : NAND et NOR	16
4.1.11	Réalisation de portes logiques	17
4.1.12	Exemples de portes logiques à transistors bipolaires :	17

4.2	Fonctions logiques de plusieurs variables	18
4.2.1	MINTERM	18
4.2.2	MAXTERM	19
4.2.3	Simplifications de fonctions logiques	19
4.3	Les tables de Karnaugh	20
4.4	Les circuits de bases	21
4.4.1	L'additionneur simple	21
4.4.2	L'additionneur complet	22
4.4.3	Le comparateur binaire 2X4 bits	22
4.4.4	Le multiplexeur	23
4.4.5	Le démultiplexeur	23
4.5	L'ALU	24
4.5.1	L'ALU - 1 bit	24
4.5.2	Etat de l'ALU	24
5	Logique séquentielle	25
5.1	L'horloge	25
5.1.1	Les délais de propagation	25
5.2	Le circuit séquentiel	26
5.2.1	Les automates finis	26
5.3	Les bascules	27
5.3.1	Le bascule <i>Reset-Set</i>	27
5.3.2	Le bascule <i>Reset-Set-Time</i>	27
5.3.3	La bascule D (<i>Delay</i>)	28
5.3.4	La bascule <i>JK</i>	28
5.4	Les registres	28
6	Le CPU (Central Processing Unit)	29
6.1	Les éléments de base	29
6.2	Les liaisons de données	30
6.3	Le décodage d'adresses	30
6.4	La mémoire externe	30
6.5	Les différents types d'architecture de processeur	30
6.5.1	L'architecture de Harvard	30
6.5.2	L'architecture de von Neumann	30
6.6	Registre de mémoire externe	31
6.7	Le périphérique	31
6.8	Mémoire et périphériques	32
6.9	Les registres	32
6.10	Le cycle machine CPU	33
6.11	Le système informatique	33

7	Le processeur	34
7.1	L'unité de contrôle	34
7.2	Le registre d'instruction	34
7.3	Le décodeur	34
7.4	Le séquenceur	34
7.5	Le registre d'état	35
7.6	PC : Program Counter	35
7.7	Taille des instructions x86	35
7.8	Les CISC	36
7.9	Les RISC	36
7.10	Architecture combinée	37
7.11	Le pipeline	37
7.11.1	Le pipeline RISC classique	37
7.12	Le processeur superscalaire	38
7.13	Le Symmetric Multiprocessing (SMP)	38
7.14	Le thread	39
7.14.1	Le multithreading	39
7.14.2	Le simultaneous multithreading	39
7.15	Le socket	40
8	Le BUS	41
8.1	Définitions et caractéristiques	41
8.2	La carte-mère	41
8.3	Largeur de bus	41
8.3.1	Largeur de bus :série	42
8.3.2	Largeur de bus :parallèle	42
8.4	Le débit binaire	42
8.5	Le protocole	42
8.6	BUS synchrone et BUS asynchrone	42
8.6.1	BUS synchrone	42
8.6.2	BUS asynchrone	42
8.7	La latence	43
8.8	La technologie	43
8.9	La topologie	43
8.10	BUS processeur	43
8.11	Front Side BUS	44
8.12	Connexions vers le monde extérieur	44
8.12.1	BUS PCI	44
8.12.2	BUS PCI-express	45
9	Communication avec les périphériques	46
9.1	Architecture générale d'un contrôleur de périphériques	46
9.2	Les différents types de périphériques	46
9.3	La communication par interrogation : Polling	46
9.4	La communication par interruption	47
9.4.1	La réponse à une interruption	47

9.4.2	Le circuit contrôleur d'interruption (8259)	47
9.5	La communication par Direct Memory Access (DMA)	48
9.5.1	Le transfert DMA	48
9.5.2	Les instructions DMA	48
9.6	Les exceptions	48
9.7	Le chipset	49
9.7.1	Le Northbridge	49
9.7.2	Le Southbridge	49
10	Les mémoires	50
10.1	Les types de mémoires	50
10.2	Les niveaux de mémoire	50
10.3	Mémoire morte	50
10.3.1	Mémoire morte : ROM	50
10.3.2	Mémoire morte : EPROM	51
10.3.3	Mémoire morte : EEPROM	51
10.3.4	Mémoire morte : Flash EEPROM	51
10.4	Mémoire vive	52
10.4.1	Ram statique	52
10.4.2	Ram dynamique	52
10.4.3	DRAM	52
10.5	Organisation de la mémoire	53
10.6	Horloge	54
10.6.1	Types de DRAM asynchrones	54
10.6.2	Types de RAM synchrones	54
11	Questions/Réponses examen	56
11.1	Histoire de l'informatique	56
11.2	Représentation des informations : représentation des nombres . .	57
11.3	Représentation des informations : représentation des caractères .	58
11.4	L'algèbre de Boole	58
11.5	Logique séquentiel	60
11.6	Le CPU	60
11.7	Le processeur	61
11.8	Le BUS	62
11.9	Communication avec les périphériques	63
11.10	Les mémoires	64

1 Histoire de l'informatique

1.1 Théorie des logarithmes

1.1.1 Définition

Logarithme : Le logarithme de base b d'un nombre réel strictement positif est la puissance à laquelle il faut élever la base b pour obtenir ce nombre.

1.1.2 Propriétés

- $\log_b(x.y) = \log_b(x) + \log_b(y)$
- $\log_b(\frac{x}{y}) = \log_b(x) - \log_b(y)$
- $\log_b(x^p) = p. \log_b(x)$

1.2 XVII^e siècle

Développements scientifiques importants, besoin de résoudre des calculs de plus en plus complexes.

Pascal en 1642 invente la Pascaline.

- Machine permettant d'additionner ou soustraire des nombres de 6 chiffres décimaux
- Roues dentées
- Multiplications par additions successives

Leibniz en 1673

- Modification de la machine de Pascal pour réaliser et automatiser les 4 opérations
- Additions successives pour multiplier
- Invention du système binaire sous sa forme moderne

1.3 XIX^e siècle

- **George Boole**
 - Formulation mathématique de propositions logiques
 - Algèbre Booléenne
- **Hermann Hollerith**
 - Automatisation de l'encodage avec des cartes perforées

1.4 XX^e siècle

Calculatrice mécanique de table, machines analogiques (Mécanique ET électricité pour représenter les fonctions logiques).

- **Edison :**
 - Invention de la lampe à incandescence

- **Shannon (1938)**
 - Associe nombres binaires, algèbre booléenne et circuits électriques
 - Théorie de l'information : association de nombres binaires pour relations logiques (**bit**)
- **Stibitz**
 - Premier additionneur logique à relais électromagnétiques : 0 = fermé, 1 = ouvert
- **Alan Turing (1936)**, création de la machine de Turing qui repose sur le modèle de la machine universelle (caractéristiques de l'ordinateur moderne) :
 - un « ruban » divisé en cases consécutives pouvant contenir un symbole parmi un alphabet fini.
 - une « tête de lecture/écriture » qui peut lire et écrire les symboles sur le ruban, et se déplacer vers la gauche ou vers la droite du ruban
 - un « registre d' état » qui mémorise l'état courant de la machine de Turing. Le nombre d'états possibles est toujours fini.
 - un « table d'actions » qui indique à la machine quel symbole écrire, comment déplacer la tête de lecture.

1.5 L'ENIAC

L'ENIAC (Electronic Numerical Integrator And Computer) est le premier calculateur universel électronique. Grâce à l'électronique, il introduisit la vitesse dans le monde des calculateurs. Il servait à calculer des trajectoires balistiques. Il était composé de plus de 17500 tubes à vide. Il fait partie de la première génération des ordinateurs à tubes à vide.¹

1.6 Les générations de processeurs/ordinateurs

1. **Les tubes à vide 1942-1958 :**
 - La grille permet de contrôler le flux d'électrons entre l'anode et la cathode
 - Relais électromécaniques et tubes électroniques
 - Coût très élevé des circuits logiques
 - Volume important, grande consommation d'électricité, fragilité (climatisation, protection)
 - Traitements privilégiés : calculs importants. (balistique militaire,...)
 - Programmation difficile faisant appel à des spécialistes de haut niveau.
 - Exemples : le colossus (1943 -premier calculateur binaire à tube), l'ENIAC(1946), voir section 1.5, le Von Neumann (1945 - programme stockée en mémoire), l'EDVAC (1949).

1. <http://loulau.chez.com/page16.html>

2. Les transistors bipolaires 1949-1964 :

- repose sur l'invention du transistor par Shockley (1948) grâce aux propriétés des matériaux semi-conducteurs. Voir section 4.1.11.
- Machines bien plus fiables
- Moins encombrantes
- Plus simples d'emploi
- Moins coûteuses
- Exemples : Le Harwell CADET (1955 -environ 400 transistors), IBM 1401

3. Les circuits intégrés (TTL) 1964 -1970 :

- apparition avec l'IBM 360 en 1964
- Utilisation de la microprogrammation
- Un format unique de représentation des données, l'octet
- Utilisation de mémoire magnétique (torres de ferrites)

4. Les circuits intégrés de haut niveau (miniaturisation des transistors)(1970-...)

- Réduction de taille des transistors
- Augmentation de la puissance de calcul
- Augmentation de la vitesse de calcul
- Loi de Morre : 2 fois plus de transistors tous les 18 mois.²

2. <http://hypermedia.univ-paris8.fr/Verroust/cours/CHAP7.HTM1>

2 Représentation des informations : représentation des nombres

2.1 Les différentes bases de représentation des nombres

- **Base 10** : 10 chiffres (0 ... 9) (décimal), nombres positifs ou négatifs, puissances entières de 10.
- **Base 2** : 2 chiffres (binaire), puissances entières de 2, représentation plus longue qu'en décimal mais facile à manipuler par un ordinateur.
- **Base 8** : 8 chiffres (0 ... 7) (octal), utilisation pour faciliter lecture du binaire (on fait des groupements de bits).
- **Base 16** : 16 chiffres (0 ... F) (hexadécimal), utilisation pour faciliter lecture du binaire (on fait des groupements de bits)

2.2 Les conversions d'une base à une autre

Pour les conversions :

- *binaire* \Longleftrightarrow *decimal*
- *octal* \Longleftrightarrow *decimal*
- *hexadecimal* \Longleftrightarrow *decimal*

\Rightarrow Relire cours de langage assemblage

2.2.1 Conversion d'un nombre réel en binaire

Pour convertir un nombre réel en binaire, on le décompose en deux parties (entière et fractionnaire). Pour la partie entière, on fait comme vu dans le cours de langage assemblage. Pour la partie fractionnaire, on procède comme suit :

1. Multiplier la partie fractionnaire par 2.
2. Isoler partie réelle et fractionnaire.
3. Recommencer jusqu'à ce que soit :
 - (a) La partie fractionnaire = 0
 - (b) Le nombre de bits maximal du nombre binaire est atteint.
4. La lecture des parties entières des multiplications de haut en bas donne le résultat de conversion binaire en puissances entières négatives de 2.

Exemple : Convertir 12,375 en binaire :

Partie entière : 12

$\frac{12}{2} = 6$	reste 0	\uparrow
$\frac{6}{2} = 3$	reste 0	
$\frac{3}{2} = 1$	reste 1	
$\frac{1}{2} = 0$	reste 1	
$12_d = 1100_b$		

Partie décimale : 0,375

$0,375 \cdot 2 = 0,75$	$= \mathbf{0} + 0,75$	\downarrow
$0,75 \cdot 2 = 1,5$	$= \mathbf{1} + 0,5$	
$0,5 \cdot 2 = 1,0$	$= \mathbf{1} + 0$	
$0,375_d = 0,011_b$		

Nombre réel : $12,375_d = 1100,011_b$

2.3 Addition d'un nombre binaire

On effectue la somme binaire, bit par bit, avec **report** éventuel. On travaille sur un nombre de bits fixés.

$$\begin{array}{rcccccccc}
 & & & \mathbf{1} & \mathbf{1} & \mathbf{1} & & & \\
 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & (20_d) \\
 + & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & (158_d) \\
 \hline
 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & (178_d)
 \end{array}$$

Note : Attention, si la somme donne lieu à un dépassement de capacité, celui-ci donnera un nombre négatif. Le résultat sera faux et le processeur ne détectera pas l'erreur.

2.4 Représentation des nombres entiers négatifs

La première idée pour représenter les nombres négatifs était de réserver un bit pour le signe en début de nombre. Cependant, cela causait un problème lors des additions d'un entier positif avec un négatif car le résultat donnait le négatif de la somme des valeurs absolues des deux termes. Pour palier à ça, on utilise, **le complément à 2** qui consiste à :

1. Nombre binaire positif
2. **Inversion des bits** de ce nombre (complément à 1)
3. On ajoute 1 (on ignore le dépassement de capacité éventuel)

Pour **lire un nombre négatif** on fait l'opération inverse :

1. **Inversion des bits** de ce nombre (complément à 1)
2. On ajoute 1 (on ignore le dépassement de capacité éventuel)

Note : L'étendue des entiers signés sur n bits est $[-2^{n-1}; 2^{n-1} - 1]$ et celle des entiers non-signés $[0; 2^n - 1]$.

2.5 Soustraction d'un nombre binaire

Pour soustraire deux nombre binaires, on utilise le complément à deux du nombre à soustraire qu'on additionne ensuite au premier terme.

2.6 Multiplication d'un nombre binaire

La multiplication d'un nombre binaire est réalisée par une suite de sommes. On peut cependant passer par un simple décalage n bits vers la gauche lorsqu'on multiplie par une puissance n de 2.

2.7 Division d'un nombre binaire

La division d'un nombre binaire est réalisée par une suite de soustractions.

$$\begin{array}{r}
 1\ 0\ 0\ 1,\ 0\ 0\ 1 \\
 -\ 1\ 0\ 1 \\
 \hline
 0\ 1\ 0\ 0\ 0 \\
 -\ 1\ 0\ 1 \\
 \hline
 0\ 0\ 1\ 1\ 0 \\
 -\ 1\ 0\ 1 \\
 \hline
 0\ 0\ 1\ 1
 \end{array}
 \left| \begin{array}{r}
 1\ 0\ 1 \\
 0\ 1,\ 1\ 1\ 0
 \end{array} \right.$$

On continue la division jusqu'à ce que le reste soit nul ou que le quotient ait atteint la limite maximum autorisée par le nombre de bits alloué. Si on divise par une puissance n de 2, on peut passer par un simple décalage de n bits vers la droite (on a alors n nombres derrière la virgule, si les n^{er} nombres ne sont pas égales à 0).

2.8 Stockage des nombres réels : virgule fixe

Le processeur ne sait pas gérer les virgules dans ce type de stockage. Le programmeur doit donc gérer le nombre de bits qu'il alloue aux valeurs entières et aux valeur fractionnaire. Il doit également prévoir un bit de signe dans le cas où il veut représenter un nombre signé.

Exemple : 1 bit pour la valeur entière et 15 bits pour la valeur fractionnaire :

- 0,000 000 0000 à 1,999 969 4824

2.9 Stockage des nombres réels : virgule flottante (floating point)

Dans le cas du stockage des nombre réels à virgule flottante. On utilise un format normalisé, similaire à la notation scientifique. On stocke 3 grandeurs en mémoire :

- Le signe
- La mantisse : les chiffres significatifs du nombre

- L'exposant : la puissance de 10 par laquelle il faut multiplier la mantisse

Exemple :

$$-2777489,812 = -2777489,812 \cdot 10^6$$

$$\text{Stockage} = -; 2(,)777489812; 6$$

2.9.1 Virgule flottante, simple précision (en binaire)

Pour le format simple précision des stockages de nombres à virgule flottante, on alloue **32 bits** d'espace mémoire.

- **Signe** : 1bit
- **Exposant** : 8bits (sous forme d' un entier non signé)
 $E = [p(\text{la puissance de } 2) + 127]$
- **Mantisse** : 23 bits, toujours au format 1,xxxxxx : les chiffres significatifs
 On ne mémorise pas ce premier 1, on a donc 24 bits effectifs de précision.

Exemple :

$$\text{Nombre} = -1001,101_b = -1,001101 \cdot 2^3$$

$$E = [p(= 3) + 127] = 130_d = 128_d + 3_d = 10000000_b + 00001001_b = 10001001_b$$

$$M = 001101_b \text{ (on ajoute des "0" pour arriver à 23)} \rightarrow 0000001111000000000000_b$$

$$S = 1_b$$

$$\text{Nombre} = (-1)^S \cdot M \cdot 2^{E-127}$$

2.9.2 Virgule flottante, double précision (en binaire)

Pour le format double précision des stockages de nombres à virgule flottante, on alloue **64 bits** d'espace mémoire.

- **Signe** : 1bit
- **Exposant** : 11bits (sous forme d' un entier non signé)
 $E = [p(\text{la puissance de } 2) + 1023]$
- **Mantisse** : 52 bits, toujours au format 1,xxxxxx : les chiffres significatifs
 On ne mémorise pas ce premier 1, on a donc 53 bits effectifs de précision.

2.10 Opérations en virgule flottante

- Multiplication
 1. Addition des exposants
 2. Multiplication des mantisses
 3. Normalisation du résultat si nécessaire
- Division
 1. Soustraction des exposants
 2. Division des mantisses
 3. Normalisation du résultat si nécessaire

Exemple :

- $(1,10)_b \cdot 2^{-3} * (1,11)_b \cdot 2^7 = ?$
- $-3 + 7 = 4$

- $(1, 10)_b * (1, 11)_b = (10, 1010)_b$ car $6_d (= 110_b) * 7_d (= 111_b) = 42_d = 101010_b \rightarrow (1, 10)_b * (1, 11)_b = 10, 1010_b$
- Avant normalisation : $(10, 1010)_b * 2^4$
- Après normalisation : $(1, 01010)_b * 2^5$

2.11 L'endianisme

Comment stocker un nombre de 32 bits dans des cases mémoires de 8 bits (1 octet)? Il existe deux convention pour l'ordre de stockage des nombres.

- **Le big endian** : Octets de « poids fort » en premier (exemple : 0xA0B70708 sera stocké dans l'ordre suivant : A0 B7 07 08. (Motorla, Sparc,...))
- **Le little endian** : Octets de « poids faible » en premier (exemple : 0xA0B70708 sera stocké dans l'ordre suivant : 08 07 B7 A0. (Intel (comme en ASM avec le 8086), ARM,...)).

3 Représentation des informations : représentation des caractères

3.1 L'ASCII

L'ASCII ou *American Standard Code for Information Interchange* est une norme de codage de caractères en informatique ancienne et connue pour son influence incontournable sur les codages de caractères qui lui ont succédé.³

- Codage des caractères (pas de valeurs numériques!)
- 7 bits (1960)
- Chiffres : valeur binaire + 30h

3.2 L'unicode

L'unicode est une norme pour l'identification unique pour chaque caractère existant. Il vise à permettre le codage de texte écrit en donnant à tout caractère de n'importe quel système d'écriture un nom et un identifiant numérique, et ce de manière unifiée, quelle que soit la plate-forme informatique ou le logiciel.⁴

- Indépendant du logiciel, processeur ou langue
- Représentation sur 32 bits

3. https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange

4. <https://fr.wikipedia.org/wiki/Unicode>

4 L'algèbre de Boole

4.1 Fonctions logiques et circuits combinatoires

4.1.1 Définitions

Circuit combinatoire : circuit idéalisé au niveau temporel : les signaux de sortie d'un circuit logique ne dépendent que des signaux d'entrées à l'instant considéré.

Fonction logique : une fonction logique est définie par le tableau de correspondance entre les états d'entrée et les états de sortie. Une fonction booléenne renvoie un résultat qui ne dépend que des valeurs des variables d'entrée.

Etat d'entrée : combinaison des valeurs prises par les entrées.

Etat de sortie : combinaison des valeurs prises par les sorties.

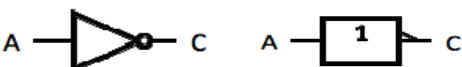
Toute fonction logique peut être réalisée à l'aide d'un petit nombre de fonctions logiques de base appelées opérateurs logiques ou portes logiques (logic gate).

4.1.2 Fonction NON (NOT)

- Fonction logique **d'une seule variable**
- Réalise une opération de **complémentation** ou **d'inversion**
- Table de vérité :

A	C
0	1
1	0

- Equation logique : $C = \overline{A}$

- Symboles : 

4.1.3 Fonction OU (OR)

- Fonction logique **de deux variables**
- Ou logique : réalise l'un ou l'autre ou les deux.
- Table de vérité :

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

- Equation logique : $C = A + B$

- Symboles :  C  C

4.1.4 Fonction ET (AND)

- Fonction logique de deux variables
- ET logique : les deux ensemble.
- Table de vérité :

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

- Equation logique : $C = A.B$

- Symboles :  C  C

4.1.5 Propriétés algébrique du ET et du OU logique

	<u>Forme ET</u>	<u>Forme OU</u>
· idempotence	(1) $a.a = a$	(1') $a+a = a$
· complémentation	(2) $a.\bar{a} = 0$	(2') $a+\bar{a} = 1$
· add. et mult. par constantes :	(3) $a.0 = 0$	(3') $a+1 = 1$
	(4) $a.1 = a$	(4') $a+0 = a$
· commutativité	(5) $a.b = b.a$	(5') $a+b = b+a$
· distributivité	(6) $a.(b+c) = a.b+a.c$	(6') $a+(b.c) = (a+b).(a+c)$
· associativité	(7) $a.(b.c) = (a.b).c$	(7') $a+(b+c) = (a+b)+c$
· absorption	(8) $a.(a+b) = a$	(8') $a+(a.b) = a$
	(9) $a.(b+c.\bar{a}) = a.b$	(9') $a+b.(c+\bar{a}) = a+b$
	(10) $a.(b+\bar{a}) = a.b$	(10') $a+(b.\bar{a}) = a+b$
· lois de de Morgan	(11) $\bar{a.b} = \bar{a} + \bar{b}$	(11') $\overline{a+b} = \bar{a} . \bar{b}$

FIGURE 1 – Propriétés algébriques du ET et du OU logique

4.1.6 Fonction OU EXCLUSIF (XOR)

- Fonction logique de deux variables
- OU exclusif logique : L'un ou l'autre mais PAS les deux ensemble.
- Table de vérité :
- Equation logique : $C = A \oplus B$

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

- Symboles :  C  C

$A \oplus A = 0$
$A \oplus 0 = A$
$A \oplus 1 = \bar{A}$
$A \oplus \bar{A} = 1$
$A \oplus B = B \oplus A$: commutativité
$A \oplus (B \oplus C) = (A \oplus B) \oplus C$: associativité
$A \oplus B = \bar{A}.B + A.\bar{B}$
$\bar{A} \oplus \bar{B} = A.B + \bar{A}.B$
$\bar{A} \oplus \bar{B} = \bar{A} \oplus B = A \oplus \bar{B}$
$A \oplus B = \bar{A} \oplus \bar{B}$

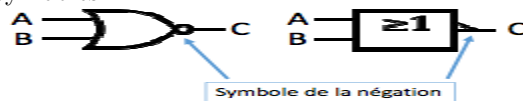
FIGURE 2 – Propriétés du OU exclusif

4.1.7 Fonction NON OU (NOR)

- Fonction logique de deux variables
- OU logique nié : ni l'un ni l'autre ni les deux.
- Table de vérité :

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

- Equation logique : $C = \overline{A + B}$
- Symboles :

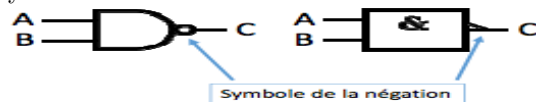


4.1.8 Fonction NON ET(NAND)

- Fonction logique de deux variables
- ET logique nié : pas les deux en même temps.
- Table de vérité :

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

- Equation logique : $C = \overline{A.B}$
- Symboles :

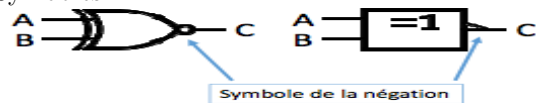


4.1.9 Fonction NON OU EXCLUSIF(EXNOR)

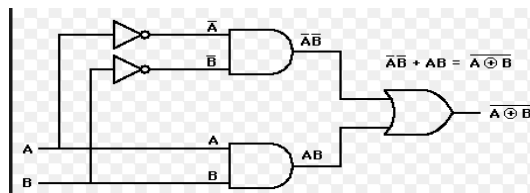
- Fonction logique **de deux variables**
- OU EXCLUSIF logique nié : aucun des deux ou les deux en même temps.
- Table de vérité :

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

- Equation logique : $C = \overline{A + B}$
- Symboles :



Représentation de la fonction EXNOR à partir des fonctions : AND, OR et NOT :



4.1.10 Fonctions universelles : NAND et NOR

Les fonctions NAND et NOR sont appelées **fonctions complètes** : toute fonction booléenne peut être représentée par une combinaison de ces portes.

4.1.11 Réalisation de portes logiques avec des transistors bipolaires

Pour réaliser des portes logiques, on utilise des **transistors** qui agissent comment des interrupteurs dont l'état est commandé par une tension. Le transistor est composé de 3 bornes :

- La Base (B) : borne de commande du transistor
- Le collecteur (C) : borne de l'interrupteur
- L'émetteur (E) : borne de l'interrupteur



- Si la tension sur la base est supérieure à un certain seuil de tension, le transistor est dit « passant » (interrupteur fermé)
- Si la tension sur la base est inférieure à ce même seuil de tension, le transistor est dit « bloqué » (interrupteur ouvert)

4.1.12 Exemples de portes logiques à transistors bipolaires :

Fonction NAND	Fonction NOR

4.2 Fonctions logiques de plusieurs variables

4.2.1 MINTERM

1. Construction de la table de vérité
2. Réalisation de la fonction logique
 - Somme de fonctions ET des n variables, pour les états d'entrée pour lesquels la fonction logique vaut 1 puis on effectue le produit de ces sommes pour obtenir le **MINTERM**

Exemple : la fonction « majorité » à 3 bits

A	B	C	R	Minterms
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}.B.C$
1	0	0	0	
1	0	1	1	$A.\overline{B}.C$
1	1	0	1	$A.B.\overline{C}$
1	1	1	1	$A.B.C$
$\text{Minterm} = \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$				

TABLE 1 – Majorité : table de vérités et Minterm

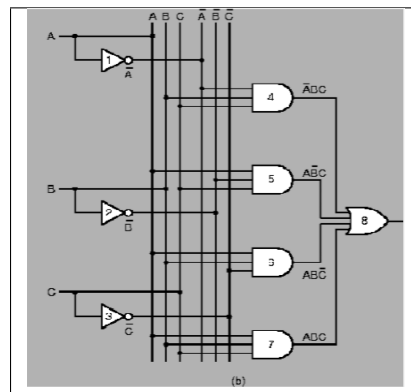


FIGURE 3 – Schéma des portes logiques du MINTERM de la fonction « majorité » à 3bits

4.2.2 MAXTERM

1. Construction de la table de vérité
2. Réalisation de la fonction logique
 - Produit de fonctions ET des n variables, pour les états d'entrée pour lesquels la fonction logique vaut 0 puis on effectue la somme de ces produits pour obtenir le **MAXTERM**

Exemple : la fonction « majorité » à 3 bits

A	B	C	R	Minterms	Maxterms
0	0	0	0		$A + B + C$
0	0	1	0		$A + B + C$
0	1	0	0		$A + B + C$
0	1	1	1	$\overline{A.B.C}$	
1	0	0	0		$\overline{A} + B + C$
1	0	1	1	$A.\overline{B}.C$	
1	1	0	1	$A.B.\overline{C}$	
1	1	1	1	$A.B.C$	
$\text{Minterm} = \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$					
$\text{Maxterm} = (\overline{A} + B + C) * (\overline{A} + \overline{B} + C) * (\overline{A} + B + \overline{C}) * (\overline{A} + B + C)$					

TABLE 2 – Majorité : table de vérités, Minterm et Maxterm

4.2.3 Simplifications de fonctions logiques

Toute fonction logique peut toujours être réalisée avec des portes ET, OU, NON

- Somme de produits
- Produits de sommes

Seulement, traduire des fonctions logiques telles qu'elles consomment beaucoup de transistors. On essaye donc de simplifier les fonctions logiques grâce à l'algèbre de Boole.

Exemple de simplification :

$$S = a * \overline{b} * c + a * b * \overline{c} + a * b * c$$

$$\begin{aligned}
 S &= a * (\overline{b} * c + b * \overline{c} + b * c) && (\text{par la propriété de distributivité (6)}) \\
 S &= a * (\overline{b} * c + b * (\overline{c} + c)) && (\text{par la propriété de distributivité (6)}) \\
 S &= a * ((\overline{b} * c + b)) && (\text{par la propriété de complémentation (2)}) \\
 S &= a * (c + b) && (\text{par la propriété d'absorption (10)}) \\
 S &= ac + ab
 \end{aligned}$$

Note : Ici la solution minimale de S est en réalité $S = a * (c + b)$ car on peut le représenter comme suit :

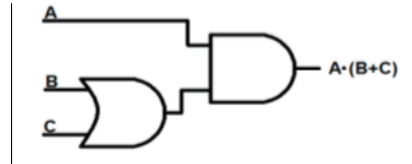


FIGURE 4 – Circuit combinatoire de S une fois simplifié

4.3 Les tables de Karnaugh

Une **table de Karnaugh** est une table qui sert à simplifier des équations logiques ou à trouver l'équation logique correspondant à une table de vérité.⁵ Elles sont très utiles lorsque la longueur d'une équation logique permet difficilement de simplifier celle-ci rapidement par la voie des propriétés algébriques.

- Simplification par groupement de variables.
- Construction d'une table dans laquelle on représente la fonction logique pour chaque état d'entrée.
- L'état d'entrée de chaque case ne diffère que d'un bit des cases voisines

Pour utiliser la table de Karnaugh, on procède comme suit :

- On représente la table, chaque case correspond à un état d'entrée, par exemple dans la case du haut à gauche. On rentrera dans l'équation l'état de sortie de S lorsque toutes les variables sont égales à 0. On fait la même chose avec chaque case.
- Ensuite, la méthode diffère selon que S soit une somme de produits (ou juste un seul produit) ou un produit de somme (ou juste une seule somme)
- Si S est une somme de produits (ou juste un seul produit), on regroupe des rectangles ou des carrés de **1** par puissance de 2, pour chacune de ces formes on garde le produit des variables qui ne varient pas. On additionne ensuite tous les produits de variables de ces différentes formes et on obtient notre **S'** , c'est à dire notre **S simplifié**.
- Si S est un produit de somme (ou juste une seule somme), grâce à la loi de Morgan, on sait qu'on peut prendre les **0**, de la même manière que dans le cas d'une somme de produit. On obtient dès lors différents produits de sommes qui représentent notre **S'** , c'est à dire notre **S simplifié**.⁶

5. https://fr.wikipedia.org/wiki/Table_de_Karnaugh

6. <http://robert.cireddu.free.fr/Ressources/AII/Cours%20sur%20les%20tableaux%20de%20karnaugh/index.htm>

Exemple (une somme de produits) :

$$S = \bar{a} * b * \bar{c} * \bar{d} + a * b * c * d + a * \bar{b} * c * d + a * b * \bar{c} * \bar{d}$$

		00	01	11	10
	ab				
00		0	1	1	0
01		0	0	0	0
11		0	0	1	1
10		0	0	0	0
cd					

FIGURE 5 – Table de Karnaugh de S, une somme de produits

- **1^{er} regroupement** : a change d'état et est éliminé, il reste $b * \bar{c} * \bar{d}$
- **2^{ème} regroupement** : b change d'état et est éliminé, il reste $a * c * d$

Dès lors $S' = b * \bar{c} * \bar{d} + a * c * d$, S' est la version simplifiée de S.

4.4 Les circuits de bases

4.4.1 L'additionneur simple

L'additionneur simple est un circuit qui a pour fonction de faire l'addition binaire (*attention, on parle bien d'addition binaire pas d'addition du OU logique*) de 2 variables d'entrée A et B.

Il existe deux bits de sorties pour ce circuit :

- **le bit de somme binaire (S)** : $S = A \oplus B$
- **le bit de report (Carry) (C)** : $C = A * B$

A	B	Somme (S)	Report (C)
0	1	1	0
1	0	1	0
1	1	0	1

TABLE 3 – Table de vérité de l'additionneur simple

4.4.2 L'additionneur complet

Un **additionneur complet** est un circuit qui nécessite une entrée supplémentaire à celui de l'additionneur simple : une retenue (C_{in}). L'intérêt de celle-ci est de permettre le chaînage des circuits.⁷ C'est à dire de tenir compte d'un report éventuel d'un étage précédent (d'un autre additionneur qu'on aurait branché à celui-ci).

Sortie vers le bit de report (C_{in})					Sortie vers le bit de somme				
C_{i-1}	A_i	B_i	00	01	11	10	S_i	A_i	B_i
0					1		0		
1			1	1	1	1	1		
$C_{out} = AB + AC_i + BC_i$ $C_{out} = AB + C_i \cdot (A \oplus B)$					$S = A \bar{B} \bar{C} + \bar{A} \bar{B} C + \bar{A} B \bar{C} + A B C$ $S = A \oplus B \oplus C_{in}$				

FIGURE 6 – Table de Karnaugh de l'additionneur complet

A	B	C_{in}	Somme (S)	C_{out}
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

TABLE 4 – Table de vérité de l'additionneur complet

4.4.3 Le comparateur binaire 2X4 bits

Un **comparateur binaire** (ici de 2X4 bits) est un circuit ayant 2 mots d'entrée A et B de 4 bits chacun, et une sortie indiquant si $A=B$ en termes de nombres binaires.⁸

7. <https://fr.wikipedia.org/wiki/Additionneur>

8. <http://tic01.tic.ec-lyon.fr/~muller/trotek/cours/logique/comparateur.html>.
fr

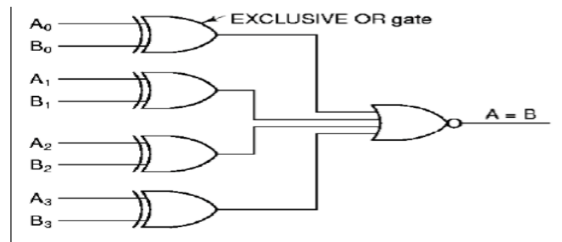


FIGURE 7 – Schéma du circuit logique du comparateur 2X4 bits

4.4.4 Le multiplexeur

Un **multiplexeur** est un circuit qui a pour rôle de faire circuler sur une seule voie les informations provenant de plusieurs sources.⁹

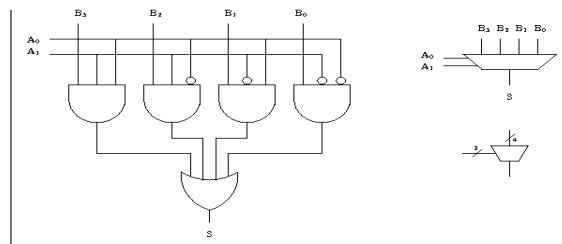


FIGURE 8 – Schéma du circuit logique du multiplexeur

A	B	Q
0	0	A
0	0	B
0	1	C
0	1	D

TABLE 5 – Table de vérité du multiplexeur

4.4.5 Le démultiplexeur

Un **démultiplexeur** est un circuit qui a pour rôle de redistribuer sur plusieurs voies l'information provenant d'une seule source : C'est l'opération inverse du multiplexage.¹⁰

9. <http://www.mongosukulu.com/index.php/en/contenu/genie-electrique4/electronique-numerique/527-multiplexeur-demultiplexeur>

10. <http://www.mongosukulu.com/index.php/en/contenu/genie-electrique4/electronique-numerique/527-multiplexeur-demultiplexeur>

Il permet d'envoyer le bit J vers la sortie correspondant au décodage des signaux binaires a et b.

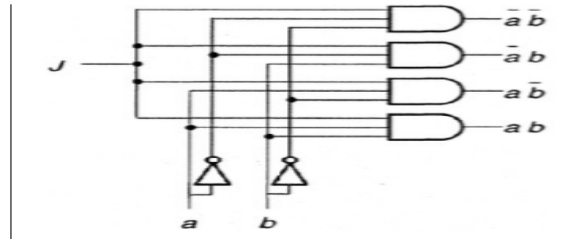


FIGURE 9 – Schéma du circuit logique du démultiplexeur

4.5 L'ALU

4.5.1 L'ALU - 1 bit

- Les opérations arithmétiques : addition de 2 bits
- Les opérations logiques : A et B, A ou B, Non B
- Sélection via les bits F_0 et F_1

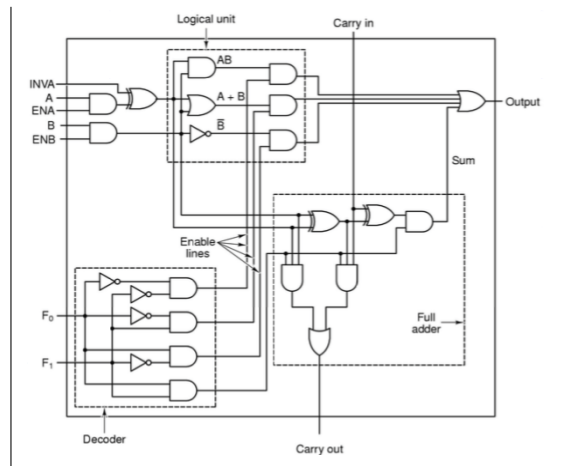


FIGURE 10 – Schéma du circuit logique de l'ALU 1 bit

4.5.2 Etat de l'ALU

L'état de l'ALU est une combinaison binaire qui permet d'indiquer à l'ALU quelle est l'opération à effectuer : instruction.

5 Logique séquentielle

5.1 L'horloge

Le CPU est doté d'une horloge car si en logique combinatoire théorique, la sortie ne dépend que des états d'entrées et pas du temps, en pratique les changements d'état dépendent des **délais de propagation** (temps nécessaire à une porte logique pour changer d'état). Ceci peut entraîner des risques d'erreurs (**glitch**) avec des états transitoires dus aux délais des portes logiques.

L'horloge est en réalité un signal permettant de valider l'état de sortie d'une fonction logique. L'horloge se trouve dans « un état invalide » pendant un temps au moins égal à la somme des délais de propagation de chacune des portes utilisées.

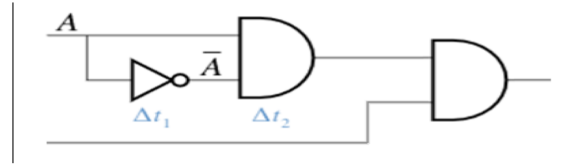


FIGURE 11 – Tant que le signal clock est à 0, la sortie data est forcée à zéro et déclarée invalide.

5.1.1 Les délais de propagation : exemple de l'additionneur

Suivant le nombre de portes logiques pour réaliser une fonction, le temps de délai total est plus ou moins long.

Exemple : variation du bit A_i : seulement 2 délais (**XOR**, **XOR**) sur la fonction S_i et 3 délais (**XOR**, **AND**, **OR**) sur la fonction C_{i+1} .

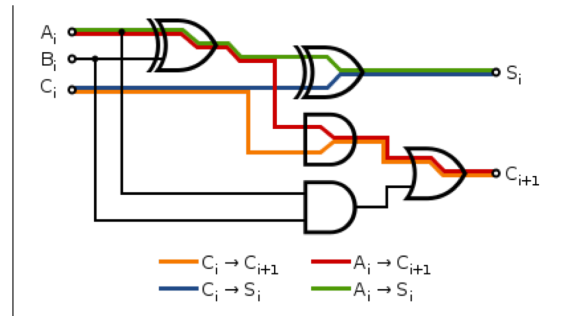


FIGURE 12 – Schéma du délais de propagation sur le circuit de l'additionneur

5.2 Le circuit séquentiel

Le circuit séquentiel est un circuit logique possédant des entrées et des sorties et présentant un comportement où les sorties ne dépendent pas seulement des entrées, mais également des séquences des entrées passées.

¹¹

- Un circuit séquentiel possède un nombre d'états possibles donné (2^n états possibles pour n bits).
- L'état des sorties du circuit séquentiel au coup d'horloge suivant (instant $t+1$) dépend de l'état de sortie actuel (instant t) et de l'état actuel des entrées combinatoires
- L'état suivant dépend de l'état actuel et des entrées combinatoires.
- **Rétroaction** : l'état actuel de sortie est utilisé comme état d'entrée permettant de calculer l'état suivant :
 - Sortie S
 - Entrée combinatoire actuelle $E(t)$
 - Etat $Q(t)$: état actuel
 - Temps : t et $t+1$
- **Sortie en $t+1$** : $S(t+1) = f(Q(t), E(t))$
- **Etat en $t+1$** : $Q(t+1) = g(Q(t), E(t))$

5.2.1 Les automates finis

Les automates de Moore sont des automates pour lesquels les sorties dépendent uniquement de l'état présent. $S(t+1) = f(Q(t+1))$

Les automates de Mealy sont des automates pour lesquels les sorties sont fonctions de l'état présent ainsi que des entrées. $S(t+1) = f(E(t), Q(t))$

où $Q(t+1) = g(E(t), Q(t))$.

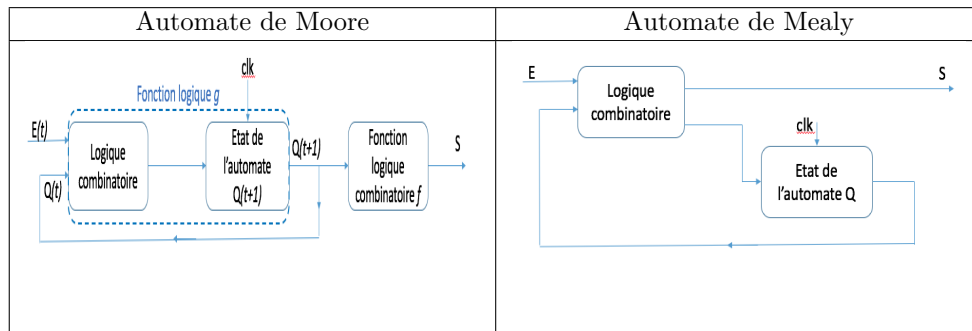


FIGURE 13 – Automates de Moore et de Mealy

11. <http://www.groupe.polymtl.ca/circuits-logiques/help/Chapitre06.pdf>

5.3 Les bascules

Les **bascules** (*bistables, flip-flop, latch* en anglais) sont des automates à 2 états stables. Ils peuvent servir de mémoire à 1 bit.

5.3.1 Le bascule *Reset-Set*

La bascule **Reset-Set** ou *mémoire RS* permet le stockage d'information (1 bit). L'opération de stockage s'appelle « **Set** ». Tandis que l'effacement s'appelle « **Reset** ». A la mise sous tension la bascule est déjà positionnée c'est pour cela que l'état $[0, 0]$ est fonction de l'état précédent. Si on peut décrire le comportement de la bascule pour les états $[0,1]$ et $[1, 0]$, on est par contre incapable de décrire son fonctionnement pour l'état $[1,1]$, on peut dire qu'il est instable.¹²

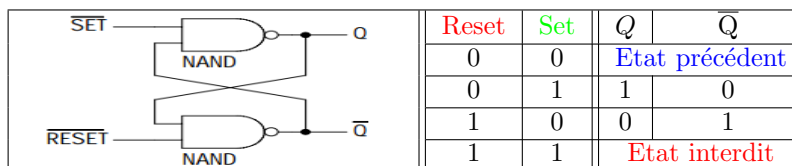


FIGURE 14 – La bascule Reset-Set

5.3.2 Le bascule *Reset-Set-Time*

La mémoire RS peut être synchronisée par une horloge. Cette bascule se comporte comme une bascule RS quand $T = 1$. Lorsque $T = 0$ la bascule reste bloquée.

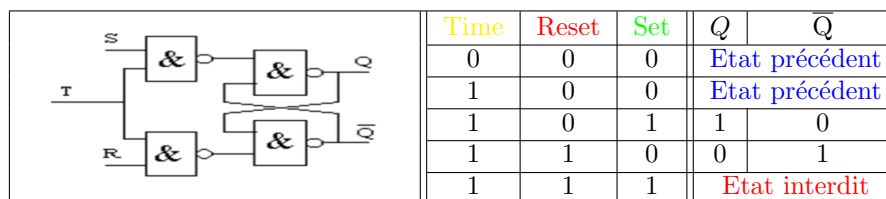


FIGURE 15 – La bascule Reset-Set-Time

¹². http://philippe.berger2.free.fr/automatique/tp/1_TP01_lbe/notions_de_bascule.htm

5.3.3 La bascule D (*Delay*)

La **bascule D** recopie sur sa sortie Q le signal appliqué à son entrée, avec un retard d'une période d'horloge.

Si $clk(E) = 1 \rightarrow Q(t+1) = D$

Si $clk(E) = 0 \rightarrow Q(t+1) = Q$ La sortie ne change pas.

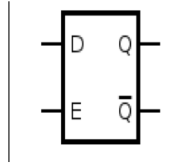


FIGURE 16 – Schéma simplifiée de la bascule Delay

5.3.4 La bascule JK

La bascule JK a le même mode de fonctionnement que la la bascule RS. Cependant l'état [1,1] n'est plus interdit. Il renvoie le complément de l'état précédent ($\overline{Q(t)}$).

5.4 Les registres

Le registre est une mémoire très rapide, utilisée dans un circuit intégré (processeur ou autre). C'est ce qu'on pourrait appeler une *case mémoire*. Il est formé de n bascules pour former une mémoire de n bits.

- On peut associer les bascules pour mémoriser un octet (8 bits) ou plus (16 bits, 32 bits voir 64 bits) et ainsi former des registres avec une mémoire correspondante.
- Le signal horloge est dans ce cas utilisé comme signal de lecture ou d'écriture pour le registre.

Exemple : Les registres x86 du processeur intel 8086 utilisés dans le cours de langage assemblage (AX, BX, CX, ...).

Les registres sont utilisés notamment pour stocker les données d'entrée, d'instructions, d'état et de stockage des valeurs de sortie.

6 Le CPU (Central Processing Unit)

6.1 Les éléments de base

- **L' ALU** : unité de calcul proprement dite.
- **Les registres** : se sont les *cases mémoires*. Elles servent pour le stockage :
 - des données en entrée de l'ALU
 - des données en sortie de l'ALU
 - de l'instruction à exécuter

Cependant les registres de bases ne suffisent pas, le CPU a besoin de *cases mémoires* supplémentaires. Il devra dès lors communiquer avec des périphériques de stockage.

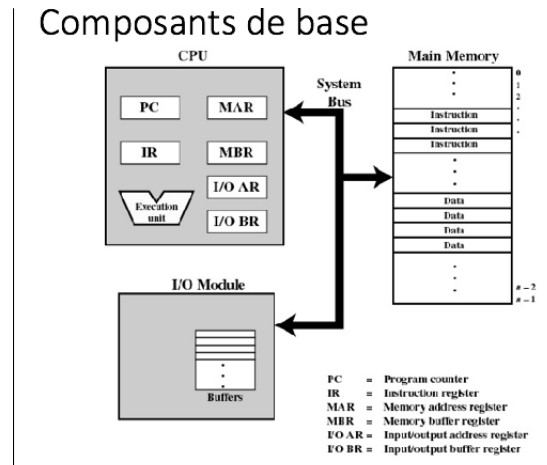


FIGURE 17 – Structure interne de la mémoire

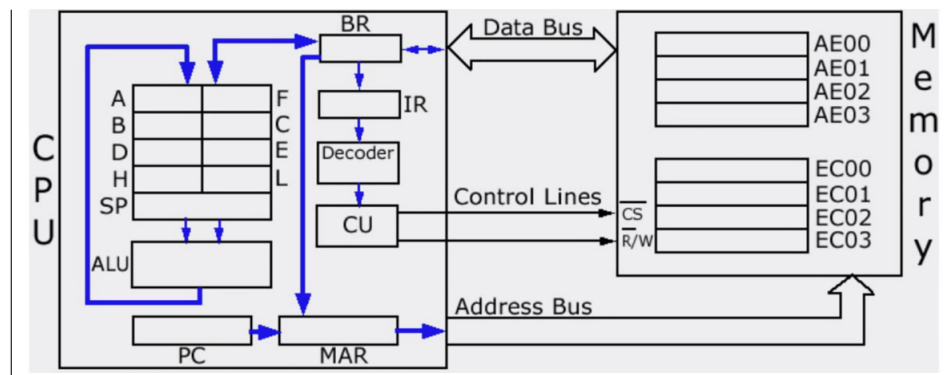


FIGURE 18 – Structure interne du processeur

6.2 Les liaisons de données

Le **bus** permet de faire la liaison entre 2 ou plusieurs composants.

Le **bus de donnée** est l'ensemble des connections qui permettent d'échanger des données entre l'ALU et un registre.

Le **bus de contrôle** est l'ensemble des connections des signaux de contrôle : sélection d'un registre, horloge, signal d'écriture, de lecture.

Le **bus d'adresse** permet l'adressage mémoire entre tous les composants. Chaque registre possède sa propre adresse binaire.

6.3 Le décodage d'adresses

Les adresses sont des nombres binaires codés sur n bits (16 bits sur l'intel 8086). Pour sélectionner le registre auquel on s'adresse, il faudra décoder cette adresse au moyen d'un **décodeur binaire** (voir section 7.3). Les autres registres étant mis en « haute impédance » pour éviter toutes collisions de données.

6.4 La mémoire externe

Comme dit plus haut, les registres du processeur ne suffisent pas pour stocker l'information dont aurait besoin un ordinateur. On va dès lors devoir exporter des instructions à exécuter et des données dans une **mémoire extérieure**. Ceci offre une capacité de stockage plus importante mais diminue la vitesse de stockage de l'information. L'« export » de données se réalise grâce aux :

- **Bus d'adresse** : sélection d'une case mémoire dans la mémoire externe.
Le décodage d'adresse est fait dans le circuit mémoire même.
- **Bus de données** : données lues ou écrites dans la mémoire.
- **Bus de contrôle** : signaux de lecture, écriture

6.5 Les différents types d'architecture de processeur

6.5.1 L'architecture de Harvard

- **Mémoire** pour les instructions
- **Mémoire** pour les données
- **Avantage** : les instructions peuvent être codées sur un nombre de bits adapté
- **Inconvénient** : 2 mémoires = 2 bus différents (adresses, contrôle, data)

6.5.2 L'architecture de von Neumann

- **Une seule mémoire** pour les instructions et les données
- **Avantage** : une seule mémoire
- **Inconvénient** : risque de « conflit » pour l'accès au bus

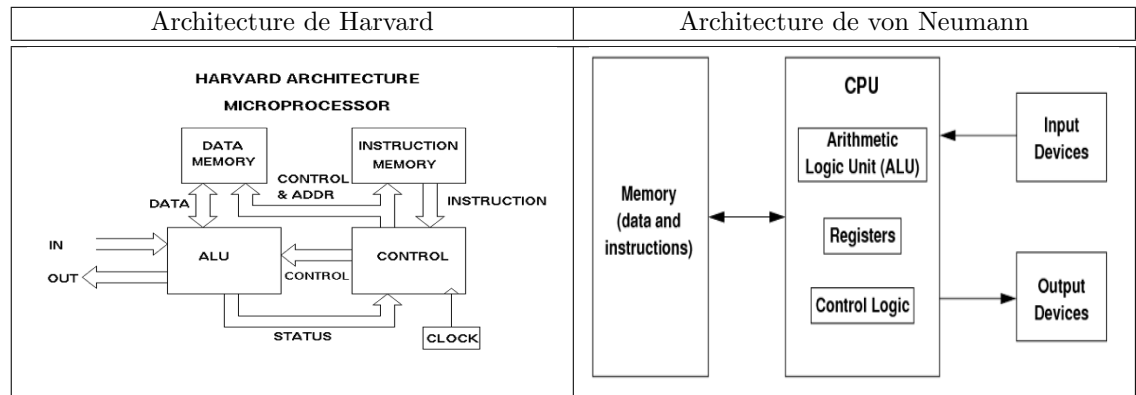


FIGURE 19 – Les architectures de processeur

6.6 Registre de mémoire externe

Il existe deux types de registre de mémoire externe

Le **Memory Address Register** :

- Mémorise sur le bus d'adresses l'adresse mémoire à laquelle on veut accéder

Le **registre tampon(buffer)** :

- mémorise sur le bus de données la donnée à écrire en mémoire (le temps que la mémoire ait le temps de réaliser l'écriture).
- mémorise depuis le bus de données la valeur envoyée par la mémoire lors d'une écriture (en attendant qu'elle soit traitée par le CPU)

6.7 Le périphérique

Le périphérique est le composant permettant une communication avec le « monde extérieur ».

Exemples : l'interface utilisateur, l'interface réseau, les capteurs, mémoire de masse, son, vidép,...

La **communication avec le périphérique** est semblable à la gestion de mémoire (utilisation d'un bus d'adresse et de décodeurs pour sélectionner le périphérique ,utilisation d'un bus de données pour les échanges de données avec le CPU et des signaux de contrôles).

L'**architecture générale d'un contrôleur de périphérique** est composé de deux registres :

Le **registre de données** :

- *Registre d'entrée*, en écriture seule : le CPU envoie des données au périphérique.
- *Registre de sortie*, en lecture seule : le CPU vient lire une donnée dans le périphérique.

Les registres de contrôles :

- En écriture seule : *registre de commande* : le CPU configure le périphérique pour qu'il exécute une opération.
- En lecture seule : *registre d'état* : le CPU interroge le périphérique sur son état actuel.

6.8 Mémoire et périphériques

Pour utiliser à la fois la mémoire et les périphériques, il existe deux types de configurations pour les dissocier :

- **Même espace d'adresse pour la mémoire et les périphériques :**
 - Adresses non utilisables pour le matériel (Exemple : PC : adresses réservées pour le matériel)
- **Espace d'adresses séparés pour mémoire et périphériques :**
 - Instructions spécifiques suivant qu'on écrit dans la mémoire ou dans un périphérique
 - Besoin d'un bit supplémentaire pour séparer les espaces d'adresses (Exemple : IE Z80 : bus d'adresse 16 bits avec 65535 adresses mémoires données sur 8 bits et un 17^e bit qui indique une adresse périphérique, 8 bits d'adresse)

6.9 Les registres

- **PC (Program Counter) :**
 - Contient l'adresse de la prochaine instruction à aller chercher en mémoire.
 - Incrémenté à chaque instruction.
 - Changement de valeur : appel d'une routine, saut conditionnel, interruption,...
- **L'IR : Instruction register :**
 - Stocke l'instruction en vue de son exécution par le CPU.
- **MAR : Memory Address Register :**
 - Contient l'adresse mémoire à laquelle on veut accéder. Relié au bus d'adresses'
- **MBR : Memory Buffer Register :**
 - Contient la donnée en provenance ou destination de la mémoire. Relié au bus de données.
- **L'accumulateur A :**
 - Registre principal de travail de l'ALU
- **I/O AR : Input/Output Address Register :**
 - Contient l'adresse du périphérique auquel on veut accéder. Relié au bus d'adresses des I/O.
- **I/O : Input/Output Buffer Register :**
 - Contient la donnée en provenance ou destination du périphérique. Relié au bus de données.

- **Registre à usage général :**
 - Dans le CPU, utilisables directement par l'unité de calcul.
- **Registres à utilisation particulière :**
 - Dans le CPU : pointeur de pile, registre d'état (FLAGS)
- **Registres à utilisation particulière :**
 - Adresses, données, registres de contrôles (configuration et état des périphériques)

6.10 Le cycle machine CPU

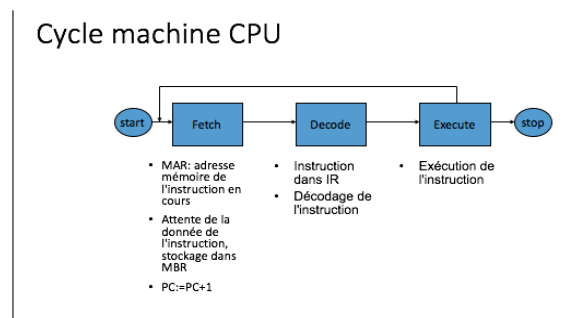


FIGURE 20 – Cycle machine du CPU

6.11 Le système informatique

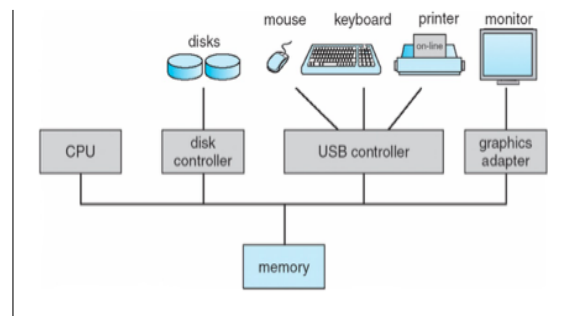


FIGURE 21 – Schéma général d'un système informatique

7 Le processeur

7.1 L'unité de contrôle

L'unité de contrôle a plusieurs fonctions :

- Gérer le déroulement des processus en cours d'exécution
- Aller chercher l'instruction suivante (**Fetch**)
- Eventuellement les opéramdes nécessaires
- Décodage de l'instruction (**decode**)
- Exécution de l'instruction (**execute**)
- Générer tous les signaux de contrôles nécessaires à chaque étape (R/W ; select, adresses,...)

7.2 Le registre d'instruction

Le **registre d'instruction** contient l'instruction à exécuter. L'instruction peut-être en plusieurs parties :

- Code Opération(**opcode**) : opération à effectuer par le CPU
- Code opérande : paramètres de l'opération : données, adresse mémoire, registre

7.3 Le décodeur

Le **décodeur** transforme les bits de l'instruction en bits et signaux de contrôle

- Opération à effectuer → ALU
- Opérations nécessaires pour aller chercher les opérandes éventuelles
 - Adresses mémoires (registres, mémoires, périphériques)
 - Signaux de sélection (registres, mémoires, périphériques)
 - Signaux de lecture/écriture

7.4 Le séquenceur

Le **séquenceur** génère une séquence de **microcommandes** sur base de l'instruction en cours et l'**état (logique) actuel** du processeur.

La **microcommande** est une commande élémentaire à l'intérieur du processeur permettant d'exécuter l'instruction de l'utilisateur.

Le rôle du séquenceur est de :

- Piloter les différentes parties du processeur impliquées par l'instruction en cours
- Synchroniser le processeur avec la mémoire
- Générer les signaux nécessaires vers l'extérieur du processeur

7.5 Le registre d'état

Le **registre d'état** est le registre contenant des informations sur l'**état actuel du processeur** (modifié par le **séquenceur** pendant l'exécution de l'instruction et modifié par l'ALU en tant que résultat de l'instruction en cours. C'est le registre contenant les bits de drapeaux (**flags** ou **PSW** (C, Z, OF, S,...)).

7.6 PC : Program Counter

Le **PC : Program Counter** ou *compteur ordinal*, *compteur d'instruction* est un registre spécial où le séquenceur écrit l'adresse de l'instruction suivante à aller chercher en mémoire (Adresse de l'instruction actuelle + taille de l'instruction actuelle + 1). L'adresse de nouvelle instruction peut être un saut à une autre partie du processus, un saut à un autre processus,.... Il est initialisé avec l'adresse de la première instruction à exécuter. Dans les processeurs x86, il s'appelle **IP** (**Instruction Pointer**).

7.7 Taille des instructions x86

Note : possibilité de trouver la même information en français dans le cours de Langage Assemblage.

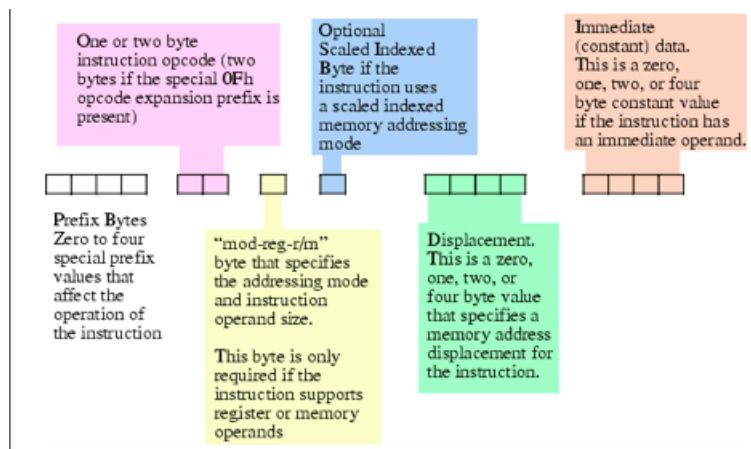


FIGURE 22 – La taille de l'instruction sur un processeur x86

7.8 Les CISC

Les **CISC** (*Complex Instruction Set Computing*, en français *microprocesseur à jeu d'instruction étendu*), désigne un microprocesseur possédant un jeu d'instructions comprenant de très nombreuses instructions mixées à des modes d'adressages complexes.¹³

Ces instructions peuvent être des :

- **Instruction simples :**
 - 1 ou 2 octets
 - Quelques coups d'horloge
 - Fonctions logiques ou arithmétiques simples
 - Exemple : NOP
- **Instructions complexes :**
 - Plusieurs octets d'instructions et opérandes
 - Beaucoup de coups d'horloge pour exécuter l'instruction
 - Fonctions logiques ou arithmétiques complexes
 - Exemple : FSQRT (racine carrée)

Exemples d'architecture CISC : processeur x86 (8086,286,486), VAX, AMD AM386,...

Les caractéristiques du CISC sont :

- « facilité » de programmation d'opérations complexes
- Temps d'exécution très variable suivant l'instruction
 - Nombre d'opérandes
 - nombre de micro-instructions du séquenceur
- Compilateur plus simple

7.9 Les RISC

Les **RISC** (*Reduced Instruction Set Computing* en français *microprocesseur à jeu d'instructions réduit*) est un type d'architecture matérielle de microprocesseurs qui se caractérise par un jeu d'instructions réduit, facile à décoder et comportant uniquement des instructions simples.¹⁴

Les caractéristiques du RISC sont :

- Uniquement des instructions simples
- Nombre d'instructions limité
- Réalisées en 1 ou 2 coups d'horloge, même temps pour chaque instruction
- Instructions codées sur le même nombre d'octets (un seul accès mémoire par instruction)
- Coeur du processeur plus simple (fréquence d'horloge plus élevée, consommation moindre)
- Les instructions complexes sont réalisées par combinaison d'opérations simples
 - Lisibilité du code plus complexe

13. https://fr.wikipedia.org/wiki/Complex_instruction_set_computer

14. https://fr.wikipedia.org/wiki/Reduced_instruction_set_computer

- Taille du code plus importante
- Compilateur plus complexe (optimisation du code)
- Opérations complexes plus lente (Exemple : pas de multiplication câblée → boucles avec somme.
- Pas d'opérations en direct sur la mémoire
 - Architecture load-store : toutes les valeurs pour une opérations doivent être chargée en mémoire (registre) avant d'être utilisée.
 - Nombre de registres généraux plus élevés

Exemples d'architecture RISC : powerpc 601,603,G3,G4,G5 (IBM/Motorola/Appple), ARM (Raspberry PI, Samsung,...), SPARC,....

7.10 Architecture combinée

- Combinaison d'instructions CISC sur un coeur RISC
- Traduction automatique dans le processeur des instructions CISC en séquence d'instructions RISC
- Les instructions de type RISC sont exécutées directement par le processeur
- Le processeur possède certaines unités de calcul complexe fréquemment utilisées (Exemples : multiplication, opérations vectorielles, FPU,...) en hardware en plus par rapport à un « vrai » RISC.
- Plus de contrainte load-store

7.11 Le pipeline

Un **pipeline** (ou *chaîne de traitement*) est l'élément d'un processeur dans lequel l'exécution des instructions est découpée en plusieurs étapes.¹⁵

L'exécution d'une instruction est divisée en étape (Fetch → decode → execute), le principe du pipeline est de séparer les étapes : pendant qu'on exécute une instruction, on peut déjà décoder la suivante et aller chercher la troisième. Le nombre d'étapes dépend du processeur (PIC :2 étapes, RISC :6 étapes, ...).

7.11.1 Le pipeline RISC classique

Les 5 étapes du pipeline RISC classique sont :

1. Instruction fetch
2. Instruction decode and register fetch
3. Execute
4. Memory access
5. Register write back

Attention, risque de conflits de ressources !

¹⁵. [https://fr.wikipedia.org/wiki/Pipeline_\(architecture_des_processeurs\)](https://fr.wikipedia.org/wiki/Pipeline_(architecture_des_processeurs))

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

FIGURE 23 – Exemple de découpe de pipeline de RISC

7.12 Le processeur superscalaire

Un **processeur superscalaire** est un processeur capable d'exécuter plusieurs instructions simultanément (un même cycle d'horloge) parmi une suite d'instructions. Pour cela, il comporte plusieurs unités de calcul, et est capable de détecter l'absence de dépendances entre instructions.¹⁶

Les caractéristiques du processeur superscalaire sont :

- Plusieurs files de pipelines en parallèle
- Plusieurs instructions d'un même processus exécutées en parallèle par le CPU
- Opérations sur plusieurs sections du CPU en parallèle : FPU, multiplicateur,...
- Inventé par l'entreprise Cray (en 1965)

7.13 Le Symmetric Multiprocessing (SMP)

Un **multiprocesseur symétrique**, ou symmetric shared memory multiprocessor (SMP), est une architecture parallèle qui consiste à multiplier les processeurs identiques au sein d'un ordinateur, de manière à augmenter la puissance de calcul, tout en conservant une unique mémoire.¹⁷

Il y a dès lors plusieurs processeurs sur une même carte-mère qui se partagent les ressources (horloge, mémoire, périphériques,...). Exemple : intel Xeon.

16. https://fr.wikipedia.org/wiki/Processeur_superscalaire

17. https://fr.wikipedia.org/wiki/Symmetric_multiprocessing

7.14 Le thread

Un **thread** (ou *fil* ou *tâche*), est similaire à un **processus** (programme en-cours d'exécution) car tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur. Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en parallèle. Toutefois, là où chaque processus possède ses propres **ressources** (Code, fichier, données, périphériques), les threads d'un même processus se partagent sa mémoire virtuelle. Par contre, tous les threads possèdent leur propre **pile, registres et Program Counter**¹⁸

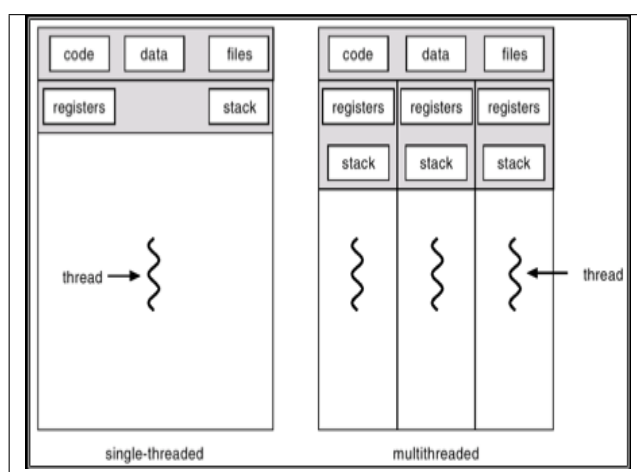


FIGURE 24 – Illustration des ressources externe et des ressources interne d'un processeur single-threaded et d'un processeur multithreaded

7.14.1 Le multithreading

Un **processeur multithread** est un processeur qui est capable d'exécuter efficacement plusieurs threads simultanément.¹⁹ Attention le **processeur multicore** quant à lui possède différents coeur avec pour chacun ses propres ressources, il peut donc gérer facilement plusieurs processus mais si chacun de ses coeurs ne sont pas eux-même multithread, il gèrera mal le multithreading.

7.14.2 Le simultaneous multithreading

Le **SMT** est le partage d'un coeur de processeur superscalaire (les pipelines, les unités de calcul et les caches) entre plusieurs threads. Les pro-

18. [https://fr.wikipedia.org/wiki/Thread_\(informatique\)](https://fr.wikipedia.org/wiki/Thread_(informatique))

19. <https://fr.wikipedia.org/wiki/Multithreading>

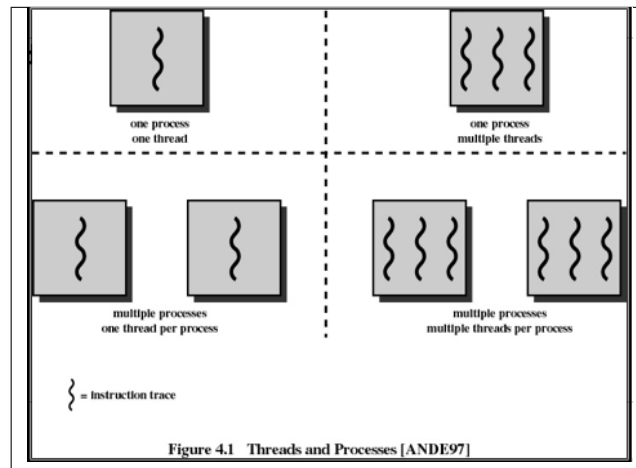


FIGURE 25 – Illustration de la gestion de un ou plusieurs processus sur un ou plusieurs threads

cesseurs non SMT passent alternativement d'un thread à l'autre pour l'exécution des instructions, alors que des processeurs SMT peuvent allouer des unités de calcul à des threads différents simultanément. Le but est d'améliorer l'utilisation des ressources.²⁰

- Les instructions **de plus d'un thread** peuvent être exécutés à n'importe quel étage d'un pipeline d'un processeur superscalaire.
- Fetch des instructions de plusieurs threads en un cycle
- Nombre de registres plus élevé
- 2-8 threads par coeur de processeur
- Appelé Hyper-threading chez Intel

7.15 Le socket

Le **socket** est le support pour le processeur sur la carte-mère. Il dépend du type de boîtier dans lequel est placé le processeur et est doté d'un système de blocage mécanique.

20. https://fr.wikipedia.org/wiki/Simultaneous_multithreading

8 Le BUS

8.1 Définitions et caractéristiques

Le bus est un ensemble de liaisons (électriques) entre plusieurs composants d'un système informatique.

Il permet de faire circuler des :

- Données
- Adresses
- Signaux de contrôle
 - Etat
 - Commande

Remarque : On utilise le mot « liaison » ou « port » plutôt que le mot « bus » quand il s'agit d'une connexion point à point entre 2 composants.

Les caractéristiques qui décrivent un bus sont :

1. Largeur	Nombre de bits transmis simultanément sur le bus
2. Débit binaire	Nombre de bit/seconde (parfois exprimé en byte ou octet)
3. Protocole	Définit comment les signaux sont gérés sur le bus
4. Synchrone/asynchrone	Synchrone : avec signal d'horloge ; Asynchrone : pas de signal d'horloge
5. Latence	Temps d'attente que met une donnée à être transférée sur le bus
6. Technologie	Codage des bits ; Niveaux de tension ; support physique (Cuivre, fibre)
7. Topologie	Daisy chain, parallèle, étoile

TABLE 6 – Caractéristiques d'un BUS

8.2 La carte-mère

La carte-mère (ou *motherboard*) est un circuit imprimé qui comporte tous les circuits électroniques nécessaires au fonctionnement du processeur.

- Socket processeur
- Mémoire
- Alimentations
- Périphériques
- Connecteurs
- Bus

8.3 Largeur de bus

La largeur de bus est le nombre de bits qu'un BUS peut transférer simultanément.

8.3.1 Largeur de bus :série

- **largeur de bus** : 1 bit
- Les bits de données sont transmis **les uns derrière les autres**
- Besoin de signaux de contrôle supplémentaire.

8.3.2 Largeur de bus :parallèle

- **largeur de bus** : n bits
- Les n bits de données sont transmis **en même temps** sur les n connexions
- Largeur typique : 8/16/32/64 bits
- Besoin de signaux de contrôle supplémentaire.

8.4 Le débit binaire

Le **débit binaire** est une mesure de la quantité de données numériques transmises par unité de temps. Il est le plus souvent exprimé en bits par secondes.²¹

On peut calculer débit par seconde de la manière suivante :

$$\begin{aligned} \text{Debit binaire} &= \frac{\text{bit}}{\text{seconde}} * \text{largeur du bus} \\ \text{Baud} &= \frac{\text{symboles}}{\text{seconde}} \end{aligned}$$

8.5 Le protocole

Le **protocole** est la manière dont les transferts sont gérés sur le bus.

- **MSB** (*Most Significant Byte* - octet de poids le plus fort) ou **LSB** (*Lowest Significant Byte* - octet de poids le plus faible) (Voir la section 2.11 à la page 12 pour plus d'information) en premier ?
- Méthode de synchronisation entre signaux de contrôles, données et adresses.

8.6 BUS synchrone et BUS asynchrone

8.6.1 BUS synchrone

- Signal d'horloge
- Temps de réponse fixe
- Problème de synchronisation d'horloge (du à la longueur du fil).

8.6.2 BUS asynchrone

- Pas de signal d'horloge
- Signaux supplémentaire de synchronisation
 - Request
 - Acknowledge

21. https://fr.wikipedia.org/wiki/debit_binaire

8.7 La latence

La latence est le temps d'attente que met une donnée à être transférée sur le bus.

Elle dépend :

- du Protocole du bus
- de la Fréquence d'horloge

8.8 La technologie

Il existe plusieurs technologies différentes pour les bus, le support physique (généralement le cuivre), la représentation des bits (soit traditionnel 1= niveau haut et 0 = niveau bas ou la technologie **NRZ** : 1= pas de changement de niveau et 0= changement de niveau). Les niveaux de tensions peuvent varier d'une technologie à une autre pour représenter l'état du bit (PCI : 0v/5v, hypertransport : 0v/1,2v,...)

8.9 La topologie

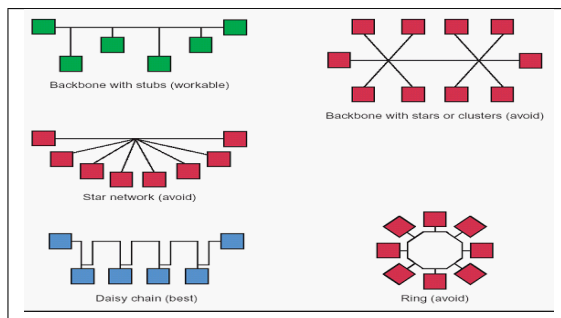


FIGURE 26 – Illustrations de plusieurs forme de topologie de bus

8.10 BUS processeur

- **Le BUS ISA (1981)**
 - Directement lié aux lignes d'adresses et data du CPA
 - 8 puis 16 bits de données
 - Utilisé pour communiquer avec les périphériques « internes » (clavier,...) ou externes via les connecteurs d'extension
 - Sélection de l'adresse d'interruption sur chaque carte
 - lent : 16 bits alors que CPU de 32 bits
 - Débit binaire de 8 MB/s
- **BUS EISA**
 - 32 bits de données

- VESA Local Bus (486)
 - Bande passante ISA trop limitée pour les applications graphiques
 - Extension de ISA
 - Application pour video
 - Contrôleurs de stockage
- NuBus (Apple)
 - 32 bits
 - Plug and Play
 - 40 MB/s

8.11 Front Side BUS

Le **front side bus (FSB)** , (aussi appelé *bus système ou bus interne-* internal bus en anglais), est traditionnellement le bus informatique qui relie le processeur qui gère les échanges avec les périphériques proches du processeur (CPU), notamment avec la mémoire vive.²²

- A l'origine, nom utilisé par intel (Pentium Pro)
- Bus de communication du CPU vers le monde extérieur
 - Périphériques
 - Mémoire Ram
- largeur de bus : nombre de bits du bus
- Généralement un multiple de la fréquence CPU (CPU :32000 MHz / FSB : 8000 Mhz)
- Vitesse de transfert
 - **Bande passante (Bandwidth) ou débit binaire :**
 - Fréquence horloge * largeur du bus * nombre de coups horloge
 - Exemple : PII (100 MHz, 64 bits) $\rightarrow 100 \text{ Mhz} * 64 * 1 = 6400 \text{ Mbits/s} = 800 \text{ Moctets/s}$
 - Core 2 Duo (333 Mhz, 64 bits, X4) $\rightarrow * 85248 \text{ Mbits/s} = 10656 \text{ Mo/s}$
- Spécifique à chaque CPU

8.12 Connexions vers le monde extérieur

Les bus permettent aussi d'interfacer le processeur vers le « monde extérieur » grâce à des cartes d'extension pour des périphériques particuliers.

8.12.1 BUS PCI

Le **bus PCI (Peripheral Component Interconnect)** est un un bus intermédiaire situé entre le bus processeur (NorthBridge) et le bus d'entrées-sorties (SouthBridge).²³

- Interconnexion processeur/périphériques

22. https://fr.wikipedia.org/wiki/Front_side_bus

23. <http://www.commentcamarche.net/contents/758-bus-pci>

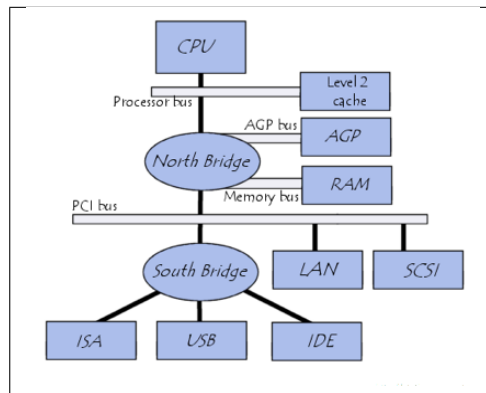


FIGURE 27 – Illustration de la connexion entre le processeur et le monde extérieur via les BUS

- Intégrés à la carte-mère
- Externes via connecteur d'extension
- Bus parallèle, largeur 32/64 bits
- Indépendant du processeur
- Buffer de données entre CPU et Périphériques
- Partage du bus entre les différents périphériques
- Plug and Play

8.12.2 BUS PCI-express

Le **bus PCI Express (Peripheral Component Interconnect Express)** est un bus d'interconnexion permettant l'ajout de cartes d'extension dans l'ordinateur. Le bus PCI Express a été mis au point en juillet 2002. Contrairement au bus PCI, qui fonctionne en interface **parallèle**, le bus PCI Express fonctionne en **interface série**, ce qui lui permet d'obtenir une bande passante beaucoup plus élevée que ce dernier.²⁴

- Bus série
- Liaison point à point bidirectionnelle (**full duplex** - la synchronisation de l'émetteur et du récepteur et le contrôle de flux sont assurés de façon logicielle (contrôle de flux programmé)²⁵).
- 2Gb/s (250 MB/s) dans chaque sens
- Connexion interne ou via connecteur
- Encapsulation des données en paquets

24. <http://www.commentcamarche.net/contents/757-bus-pci-express-pci-e>

25. [https://fr.wikipedia.org/wiki/Duplex_\(canal_de_communication\)](https://fr.wikipedia.org/wiki/Duplex_(canal_de_communication))

9 Communication avec les périphériques

9.1 Architecture générale d'un contrôleur de périphériques

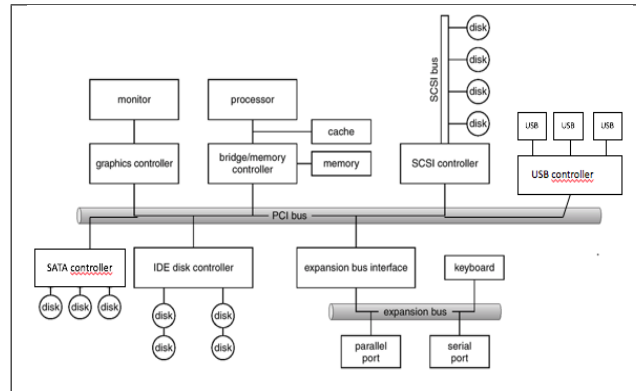


FIGURE 28 – Architecture général de la communication via les BUS entre les périphériques

9.2 Les différents types de périphériques

Un **périphérique informatique** est un dispositif connecté à un système informatique qui ajoute à ce dernier des fonctionnalités.²⁶

Il existe différents types de périphériques :

- Les périphériques de stockage :HDD, USB,...
- Les périphériques de transmission : interface réseau,...
- Les périphériques interface homme-machine (**IHM**) : clavier, souris,...
- Les périphériques spécialisées : imprimantes, scanner, ...

9.3 La communication par interrogation : Polling

Le **polling** est une méthode de communication avec un périphérique qui vise à questionner de façon continue le périphérique pour vérifier que des données peuvent être lues ou écrites

Pour ce faire :

- Le microprocesseur détermine l'état du périphérique
 - Prêt pour une commande
 - Occupé
 - Erreur
- Le microprocesseur reste en cycle d'attente active (*busy waiting*) pour attendre les opérations d'entrée/sortie du périphérique

26. https://fr.wikipedia.org/wiki/Peripherique_informatique

Bien que le polling soit une façon simple d'accéder à un périphérique , le débit des données dans un périphérique est parfois beaucoup plus lent que la vitesse de fonctionnement d'un processeur , et le polling peut donc être très inefficace . On lui préfère en général la méthode par interruptions.²⁷

Exemple : L'imprimante

9.4 La communication par interruption

Le communication par interruption est une méthode de communication avec un périphérique où le périphérique interrompt le microprocesseur (il fait une interruption) lorsque des données peuvent être lues ou écrites. Une fois que le processeur a effectué l'instruction demandée, il reprend son travail initial.

Une interruption est un événement qui provoque l'arrêt du programme en cours et provoque le branchement du microprocesseur à un sous-programme particulier dit de « traitement de l'interruption ».²⁸

Les interruptions sont :

- **masquables** : le microprocesseur peut ignorer, retarder ou hiérarchiser certaines interruptions.
- attribuées à un certain numéro d'interruption durant le démarrage par le gestionnaire d'interruption.

9.4.1 La réponse à une interruption

1. Fin de l'exécution de l'instruction en cours d'exécution
2. Sauvegarde de l'état actuel du CPU
 - Sauver l'état des registres sur la pile (avec un **push**) et du Program Counter.
3. Saut à la routine du gestionnaire de l'interruption
4. Détermination de la cause de l'interruption
5. Exécution de la routine appropriée
6. Retour à l'état d'exécution présent avant l'interruption
 - Restauration de l'état des registres et du PC du processus interrompu

9.4.2 Le circuit contrôleur d'interruption (8259)

Le circuit contrôleur d'interruption est un composant programmable qui rassemble les signaux d'interruptions de divers éléments périphériques.²⁹

27. http://www.technologuepro.com/microprocesseur/chap5_microprocesseur.htm

28. http://www.technologuepro.com/microprocesseur/chap5_microprocesseur.htm

29. <http://www.courstechinfo.be/Techno/Interrupt.html>

9.5 La communication par Direct Memory Access (DMA)

La communication par DMA est une méthode de communication entre un périphérique et la mémoire sans passer par les registres du CPU. Elle nécessite un circuit spécialisé (un **contrôleur d'accès direct mémoire**) placé entre la mémoire, le CPU et les périphériques.

Exemples d'utilisation : Grands transferts de données (HDD, CD, DVD,...).

9.5.1 Le transfert DMA

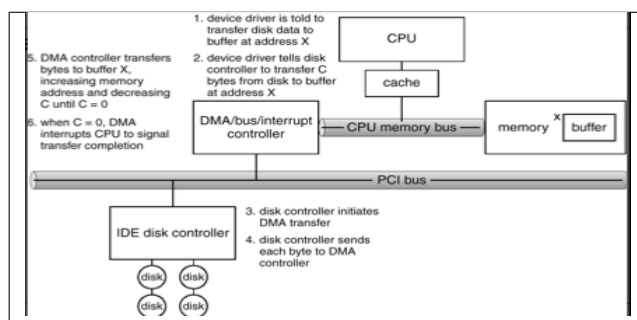


FIGURE 29 – Illustration d'un transfert DMA entre un contrôleur de disques et la mémoire

9.5.2 Les instructions DMA

- Instruction de DMA pour transfert d'un bloc de données :
 - Pointeur source
 - Pointeur destination
 - Nombre d'octets à transférer
- Le contrôleur de DMA place directement les adresses sur le bus sans intervention du CPU
- Vol de cycles mémoires au CPU
 - CPU travaille sur la mémoire cachée
 - Ralentissement possible du CPU
 - DMA prioritaire sur CPU
- Interruption en fin de transfert

9.6 Les exceptions

Une **exception** est un appel système causant une **interruption software** due à des erreurs (erreurs de division par zéro, accès non permis à une zone mémoire,... qui fait appel à une routine particulière du système d'exploitation (un genre de try... catch...))

Attention les exceptions (« interruption software ») ne doivent pas être confondues avec les interruptions (« interruption hardware ») !

9.7 Le chipset

Un **chipset** est un jeu de composants électroniques inclus dans un circuit intégré préprogrammé permettant de gérer les flux de données numériques entre le ou les processeur(s), la mémoire et les périphériques.³⁰

9.7.1 Le Northbridge

Le **southbridge** est l'ensemble des contrôleurs de périphériques regroupés dans un seul circuit.

9.7.2 Le Southbridge

Le **southbridge** est l'ensemble des circuits de gestion mémoire regroupés dans un seul circuit.

30. <https://fr.wikipedia.org/wiki/Chipset>

10 Les mémoires

10.1 Les types de mémoires

La **mémoire externe** est une mémoire qui n'est pas directement accessible par le processeur.

On compte plusieurs type de mémoire :

- Mémoire accessibles directement par le CPU
- Mémoire **morte**
 - Non modifiable par le CPU directement
 - Non volatile (contenu reste mémorisé si l'alimentation électrique disparaît)
- Mémoire **vive**
 - Modifiable par le CPU
 - Généralement volatile

10.2 Les niveaux de mémoire

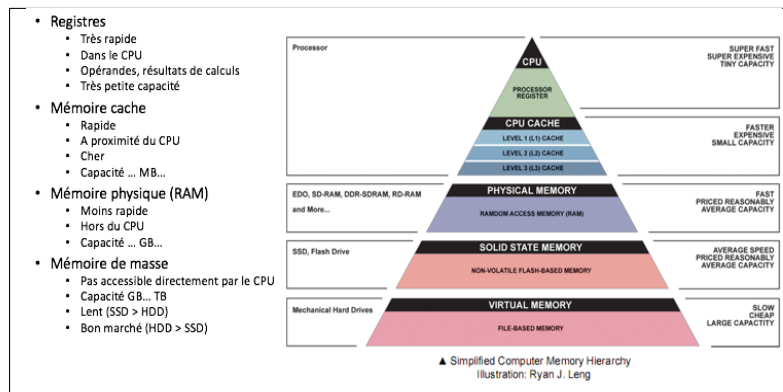


FIGURE 30 – Illustration des niveaux de mémoires

10.3 Mémoire morte

Une **mémoire morte** est une mémoire non volatile, c'est-à-dire une mémoire qui ne s'efface pas lorsque l'appareil qui la contient n'est plus alimenté en électricité.

10.3.1 Mémoire morte : ROM

- Mémoire morte : Read Only Memory, non volatile
- Elle est accessible en lecture seule; non modifiable directement par le CPU :

- **ROM** : Read Only Memory
- **PROM** : Programmable ROM ou **OTPROM** = One Time PROM : programmable une seule fois par l'utilisateur.
- Programmation en grillant des fusibles dans le circuit :
 - Par défaut : tous les bits sont à 1
 - Griller un fusible fait passer l'état à 0
 - Durée de vie théoriquement infinie

10.3.2 Mémoire morte : EPROM

- EPROM : Erasable Programmable ROM (1971)
 - Mémoire effaçable (via l'exposition aux rayons UV)
 - Capacité maximale : 32 Mibit = 4Mio
- Chaque bit est constitué d'un transistor à grille isolée
 - Charges électriques présentes → transistor conducteur → état 1
 - Pas de charge électrique → transistor ne conduit pas → état 0
- Des charges électriques sont « piégées » sur la grille isolée
 - Tension plus élevée de programmation (12 volts)
 - Mémorisation 0 ou 1
 - Durée de vie 20... 30 ans (diminue à chaque reprogrammation..100 reprogrammations).

10.3.3 Mémoire morte : EEPROM

- EEPROM = Electrically Erasable Programmable ROM (1978)
 - Programmable ou effaçable électriquement
 - Avec des tensions plus élevées (12..25v) qu'en utilisation normale (5v)
 - Nombre de reprogrammations limité (100 000)
 - Durée de vie 20... 30 ans (diminue à chaque reprogrammation..100 reprogrammations).

10.3.4 Mémoire morte : Flash EEPROM

- Principe quasi identique à l'EEPROM
 - Transistor à grille isolée
 - Programmation/effacement avec des tensions plus élevées
- Effacement/programmation **par blocs** et pas par octet (NAND flash)
- 100 000 à 1 million de reprogrammations

Types de mémoires Flash

Flash NOR

- Cellules en parallèle, accès aléatoire possible, capacité max : 4 Gb
- Vitesse de lecture élevée (≥250 Mbits/s)
- Lente à l'écriture (0,5 Mbits/s)
- Effacement : 900 ms

- Application : BIOS, programmation de microcontrôleurs

Flash NAND

- Cellules en série, accès par bloc, capacité max ...TB
- Vitesse de lecture ~ 100 Mbits/s
- Rapide en écriture
- Effacement : 2 ms
- Applications : stockage de masse (clé USB, SSD)

10.4 Mémoire vive

- Mémoire de travail du CPU
- Code, données, variables
- Généralement mémoire volatile
- RAM : Random Access Memory : Mémoire à accès aléatoire
 - Par opposition à mémoire à **accès séquentiel** (bande) ou **par bloc** (disque dur)
- Chaque octet/mot de la mémoire peut être accédé directement par le CPU

10.4.1 Ram statique

SRAM : RAM Statique

- Bascule RS à transistors
- Cout en transistors très élevés (4-6 transistors par bit)
- Très rapide
- Stockage de la valeur pendant quelques heures (MOSFET)
- Utilisation pour cache ou registres
- Quelques nanosecondes de temps d'accès

10.4.2 Ram dynamique

DRAM : RAM dynamique

- Chaque bit est constitué d'un transistor et d'un condensateur
 - Condensateur chargé = 1
 - Condensateur déchargé = 0
- Le transistor donne accès au condensateur quand il est activé
- On « lit » la valeur stockée sur le condensateur

10.4.3 DRAM

- Problème : décharge naturelle du condensateur
- Besoin de venir réécrire la valeur régulièrement sur le condensateur
 - Rafraîchissement de la mémoire
 - Intervalle de temps : quelques ms
 - lecture et réécriture de la donnée

- Un seul transistor par bit
 - Grande densité possible
 - Faible coût
 - Consommation moindre
- Temps d'accès plus lent que SRAM
 - Temps durant lequel la mémoire est inaccessible plus long pendant le rafraichissement

10.5 Organisation de la mémoire

Si on organisait la mémoire selon une structure linéaire :

- **Faible capacité** : Ok avec un décodeur d'adresse
- **Grosse capacité** : pas OK avec un simple décodeur d'adresse car structure trop complexe à réaliser

Du coup, on organise la mémoire en tableau de bits (« pages »)

- Matrice de lignes et colonnes
- Chaque intersection correspond à 1 bit (SRAM ou DRAM)

Lignes et colonnes Dans le cas d'une organisation de la mémoire en lignes et colonnes, on utilise 2 décodeurs $n/2$ bits vers $2^{\frac{n}{2}}$ bits (**plus simple et rapide que la mémoire linéaire**). Par exemple : adresse sur 14 bits : → 16384 adresse → tableau de 128*128 bits avec 2 décodeurs 7 bits vers 128 bits (au lieu d'un seul décodeur 14 bits vers 16384 lignes dans le cas de la mémoire linéaire). L'adresse mémoire est donc découpée en :

- Numéro de ligne
- Numéro de colonne

Modules mémoires On regroupe la mémoire en modules. On utilise N composants mémoire « 1 bit » pour obtenir une mémoire de mots de taille N

Barettes de mémoires La taille des mots est plus importante. On regroupe la mémoire en bancs (de 8 bits - 1 octet).

- Banc 0 : octets 0;4;8;...
- Banc 1 : octet 1;5;9;...

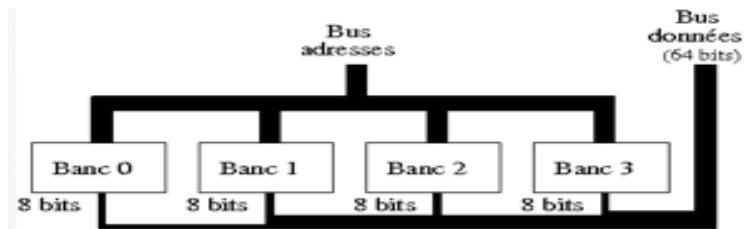


FIGURE 31 – Illustration de barettes mémoire

10.6 Horloge

La mémoire a un « temps de réaction » avant de fournir à un CPU le contenu d'une case mémoire .

- Décodage d'adresse
- Temps de réaction des portes logiques
- Délai de stabilisation des signaux sur les BUS

10.6.1 Types de DRAM asynchrones

Dans le cas des RAM asynchrones : PAS d'horloge

- Le CPU attend « le temps qu'il faut » pour avoir la donnée

Les différentes DRAM asynchrones :

- RAM
 - Mémoire directement intégrée à la carte mère
- FPRAM (1987)
 - Fast Page RAM (386 ; 486)
 - Plusieurs accès à des données successives : pas besoin de changer l'adresse de colonne → plus rapide
- EDO RAM (1990)
 - Extend Data Out RAM
 - Démarrage d'un nouveau cycle de lecture pendant que la donnée précédente est lue.

10.6.2 Types de RAM synchrones

Dans le cas de RAM synchrones : nombre de cours d'horloge fixe entre la demande et la réception de la donnée

- SDRAM : Synchronous Dynamic RAM (1997)
- Fréquence d'horloge : 66 MHz ; 100 MHz ; 133 MHz ; 3,3V
- Temp d'attente : 2-3 cycles après la demande
- 4/8/16 bits combinés pour des mots jusque 64 bits
- 1 demande et 1 transfert par cycle : besoin d'un contrôleur pour gérer l'accès au bus de données.

Les différentes SDRAM et comment calculer leur vitesse de transferts :

DDR SDRAM - 2000

- lecture ou écriture de 2 mots par cycle pour 1 seule commande
- Horloge : 100MHz/266MHz
- Clock → 100 MHz
- DDR → 200 M transferts /s (
- 200*8 bits = 1600 Mo/s
- → **PC1600**

DDR2 SDRAM - 2003

- Fonctionnement équivalent à vitesse d'horloge de la RM doublée par rapport à celle du BUS
- Horloge : 200MHz/533MHz

Exemple : DDR2-200

- BUS Clock \rightarrow 100 MHz
- RAM Clock \rightarrow = BUSClock *2 = 200 MHz
- DDR \rightarrow 400 M transfert/s (**Rappel** : lecture ou écriture de 2 mots par cycle)
- $400 * 8 \text{ bits} = 3200 \text{ Mo/s}$
- \rightarrow **PC3200**

DDR3 SDRAM -2007

- 2 horloges décalées \rightarrow équivalent vitesse d'horloge RAM quadruplée
- Horloge :200MHz/533MHz

Exemple : DDR3-800

- BUS Clock \rightarrow 100 MHz
- RAM Clock \rightarrow = BUSClock *4 = 400 MHz
- DDR \rightarrow 800 M transfert/s (**Rappel** : lecture ou écriture de 2 mots par cycle)
- $800 * 8 \text{ bits} = 6400 \text{ Mo/s}$
- \rightarrow **PC6400**

11 Questions/Réponses examen

11.1 Histoire de l'informatique

Expliquez ce qu'est Eniac

Voir la section 1.5 à la page 6.

Remplacez ces machines (hollerith ; colossus ; stibitz ; pascaline dans l'ordre chronologique, de ?crivez-les en quelques mots

Voir la section 1.2 à la page 5, la section 1.3 à la page 5, la section 1.4 à la page 5 et la section 1.4 à la page 5.

Donnez les caractéristiques des différentes générations d'ordinateurs

Voir la section 1.6 à la page 6.

Quelle a été la contribution de H. Hollerith/Edison,... ? l'évolution de l'informatique

Voir la section 1.3 à la page 5 et la section 1.4 à la page 5.

La 1/2/3/4^e génération d'ordinateurs a été marquée par un composant électronique. De ?crivez-le. Expliquez ses avantages par rapport à la génération précédente

Voir la section 1.6 à la page 6.

Remplacez ces personnes (Boole, Pascal, Hollerith , ...) dans l'ordre chronologique, décrivez-leur contribution à l'évolution de l'informatique en quelques mots

Voir la section 1.2 à la page 5, la section 1.3 à la page 5 et la section 1.4 à la page 5.

Décrivez la machine universelle selon Turing ; comparez avec les PC actuels

Voir la section 1.4 à la page 5.

Expliquez la loi de Moore

Voir la section 1.6 à la page 6.

Décrivez la pascaline

Voir la section 1.2 à la page 5.

11.2 Représentation des informations : représentation des nombres

Expliquer le principe de multiplication d'entiers binaires ; effectuer un calcul donné

Voir la section 2.6 à la page 10.

Comment convertir un nombre décimal entier négatif en sa représentation binaire en complément à 2 ; méthode + exemple

Voir la section 2.4 à la page 9.

Transformer le nombre binaire/écimal/octal/hexadécimal en binaire/décimal/octal/hexadécimal

Voir la section 2.2 à la page 8.

Expliquez le codage d'un nombre en virgule flottante

Voir la section 2.9 à la page 10.

Codez le nombre décimal/binaire xxx en format virgule flottante

Voir la section 2.9 à la page 10.

Décodez le nombre en virgule flottante vers décimal

Voir la section 2.9 à la page 10.

Expliquez le problème de la représentation des nombres binaires négatifs. Quelles solutions existe-t-il ? Décrivez-les rapidement

Voir la section 2.4 à la page 9.

Effectuez la soustraction de 2 nombres binaires entiers ; expliquez les étapes

Voir la section 2.5 à la page 10.

Effectuer une division binaire

Voir la section 2.7 à la page 10.

Principe de la représentation des nombres en décimal/binaire/hexadécimal

Voir la section 2.1 à la page 8.

expliquer le problème du dépassement de capacité

Voir la section 2.3 à la page 9.

Qu'est-ce que l'endianisme ?

Voir la section 2.11 à la page 12.

Expliquez le principe du codage en virgule flottante simple précision (32bits)

Voir la section 2.9 à la page 10.

11.3 Représentation des informations : représentation des caractères

Qu'est-ce que le code ascii ?

Voir la section 3.1 à la page 12.

Expliquez le principe du codage unicode ; pourquoi a-t-il été développé ?

Voir la section 3.2 à la page 12.

11.4 L'algèbre de Boole

Donner les tables de vérité des fonctions logiques de base

Voir la section 4.1 à la page 13.

Représenter (donner le schéma) de la fonction logique $y = \dots$

Voir la section 4.1 à la page 13.

Etablir l'équation d'une fonction logique à partir de sa table de vérité, sous forme de somme de minterms

Voir la section 4.2.1 à la page 18.

Etablir l'équation d'une fonction logique à partir de sa table de vérité, sous forme de produit de maxterm

Voir la section 4.2.2 à la page 19.

Représenter (donner le schéma) de la fonction logique y à partir de sa table de vérité, en l'exprimant sous forme d'une somme de minterms

Voir la section 4.2.1 à la page 18.

**Qu'est-ce qu'une fonction logique ; à l'aide de quoi sont-elles réalisées ?
Donnez un exemple**

Voir la section 4.2.3 à la page 19.

Réalisez une porte XNOR à partir des portes logiques de base AND ;OR ;NOT

Voir la section 4.1.9 à la page 16.

Décrivez et comparez les fonctions OU et OU exclusif

Voir la section 4.1.3 à la page 13 et la section 4.1.6 à la page 14.

Qu'est-ce qu'un circuit combinatoire (par opposition à un circuit séquentiel)

Voir la section 4.1.1 à la page 13.

Etablir la table de vérité de la fonction logique y à partir de son équation/schéma/karnaugh

Voir la section 4.3 à la page 20.

Etablir l'équation d'une fonction logique à partir de sa table de Karnaugh/schéma

Voir la section 4.3 à la page 20.

Simplifier la fonction logique y par la méthode algébrique et par Karnaugh

Voir la section 4.3 à la page 20.

Qu'est-ce qu'une table de Karnaugh d'une fonction logique. A quoi sert-elle ? Donnez un exemple

Voir la section 4.3 à la page 20.

Expliquez ce qu'est une fonction logique universelle ou complète. Illustrez par un exemple

Voir la section 4.1.10 à la page 16.

Représentez un additionneur binaire avec des portes logiques. Donnez sa table de vérité

Voir la section 4.4.1 à la page 21.

Décrivez le multiplexeur/comparateur/démultiplexeur

Voir la section 4.4.3 à la page 22, la section 4.4.4 à la page 23 et la section 4.4.5 à la page 23.

Sur base du schéma de l'ALU et d'un état d'entrée donné, déterminez la fonction réalisée par l'ALU

Voir la section 4.5.1 à la page 24.

Expliquez le schéma suivant ; à quoi sert-il ? Multiplexeur ; démultiplexeur

Voir la section 4.4.4 à la page 23 et la section 4.4.5 à la page 23.

Expliquez le fonctionnement d'une ALU sur base du schéma

Voir la section 4.5.1 à la page 24.

11.5 Logique séquentiel

Décrivez l'automate de Mealy et de Moore

Voir la section 5.2.1 à la page 26.

Expliquez ce qu'est un glitch ; montrez-le sur un exemple ; solution ?

Voir la section 5.1 à la page 25.

Qu'est-ce qu'un circuit séquentiel ? A quoi sert l'horloge ?

Voir la section 5.2 à la page 26 et la section 5.1 à la page 25.

Qu'est-ce qu'un bistable ?

Voir la section 5.3 à la page 27.

11.6 Le CPU

Représenter le schéma de base d'un CPU. Expliquez rapidement le rôle des différents registres

Voir la section 6.1 à la page 29 et la section 6.9 à la page 32.

Décrivez l'architecture CPU de Von Neumann et ses caractéristiques essentielles

Voir la section 6.5.2 à la page 30.

Décrivez l'architecture CPU de Harvard et ses caractéristiques essentielles

Voir la section 6.5.1 à la page 30.

Comparez les architectures CPU de Von Neuman et Harvard

Voir la section 6.5.2 à la page 30.

Citez et expliquez le rôle des registres de base d'un CPU

Voir la section 6.9 à la page 32.

Donnez et expliquez les différentes étapes d'un cycle machine CPU

Voir la section 6.10 à la page 33.

Expliquez les différentes étapes d'exécution d'une instruction

Voir la section 6.10 à la page 33.

Décrivez les étapes du décodage d'une adresse mémoire

Voir la section 6.3 à la page 30.

11.7 Le processeur

Expliquez le rôle du décodeur dans le CPU

Voir la section 7.3 à la page 34.

Expliquez le rôle du séquenceur dans le CPU/ dans un ordinateur

Voir la section 7.4 à la page 34.

Expliquez ce qu'est un processeur superscalaire

Voir la section 7.12 à la page 38.

Expliquez ce qu'est un pipeline (en informatique)

Voir la section 7.11 à la page 37.

Qu'est-ce qu'un registre ; à quoi sert-il ? Donnez un exemple

Voir la section 7.2 à la page 34, la section 7.5 à la page 35 et la section 6.9 à la page 32.

Décrire et comparer les processeurs RISC et CISC ; qu'est-ce qu'une architecture combinée ?

Voir la section 7.8 à la page 36, la section 7.9 à la page 36 et la section 7.10 à la page 37.

Expliquez le multithreading, quels sont ses avantages ?

Voir la section 7.14.1 à la page 39.

Expliquez les étapes d'un pipeline RISC classique (5 étapes) ; avantages et inconvénients ?

Voir la section 7.11.1 à la page 37.

Qu'est-ce qu'un socket

Voir la section 7.15 à la page 40.

Décrire et comparer la notion de multithread et de multicore

Voir la section 7.14.1 à la page 39.

Expliquez le multiprocessing symétrique (SMP)

Voir la section 7.14.2 à la page 39.

Quels sont les rôles de l'unité de contrôle dans un CPU ?

Voir la section 7.1 à la page 34.

Qu'est-ce qu'un compteur ordinal ?

Voir la section 7.6 à la page 35.

Quel genre d'information trouve-t-on dans le registre d'état d'un processeur ?

Voir la section 7.5 à la page 35.

11.8 Le BUS

Qu'est-ce qu'un bus ; décrivez les caractéristiques d'un bus ?

Voir la section 8.1 à la page 41.

Décrivez les différents types de bus dans un ordinateur

Voir la section 8.10 à la page 43, la section 8.11 à la page 44 et la section 8.12 à la page 44.

Qu'est-ce que la vitesse de transfert d'un processeur, comment la calcule-t-on ?

Voir la section 8.4 à la page 42.

Décrivez et comparez le bus parallèle et le bus série

Voir la section 8.3 à la page 41.

Décrivez le front side bus

Voir la section 8.11 à la page 44.

Qu'est-ce que la largeur d'un bus ?

Voir la section 8.3 à la page 41.

Quelles sont les caractéristiques principales du bus ISA/PCI/PCIexpress ?

Voir la section 8.10 à la page 43, la section 8.12.1 à la page 44 et la section 8.12.2 à la page 45.

11.9 Communication avec les périphériques

Citez 4 types de périphériques

Voir la section 9.2 à la page 46.

Décrivez la communication CPU/périphérique par polling/interruption/Accès direct en mémoire

Voir la section 9.3 à la page 46, la section 9.4 à la page 47 et la section 9.5 à la page 48.

Quel est le rôle d'un circuit contrôleur d'interruption

Voir la section 9.4.2 à la page 47.

Qu'est-ce qu'une exception ? Expliquez la différence avec une interruption

Voir la section 9.6 à la page 48.

Quelles sont les étapes effectuées pour répondre à une interruption ?

Voir la section 9.4.1 à la page 47.

Qu'est-ce qu'une interruption ?

Voir la section 9.4 à la page 47.

Décrivez le rôle du Northbridge/Southbridge

Voir la section 9.7.1 à la page 49 et la section 9.7.2 à la page 49.

Qu'est-ce qu'un chipset ?

Voir la section 9.7 à la page 49.

Expliquez l'architecture générale d'un contrôleur de périphérique

Voir la section 9.1 à la page 46.

11.10 Les mémoires

Expliquez ce qu'est la mémoire externe ?

Voir la section 10.1 à la page 50.

Expliquez la notion de mémoire morte, mémoire vive

Voir la section 10.3 à la page 50 et la section 10.4 à la page 52.

Décrivez rapidement les différents niveau de mémoire

Voir la section 10.2 à la page 50.

Que signifient les abréviations ROM ; RAM ; EPROM

Voir la section 10.3 à la page 50 et la section 10.4 à la page 52.

Comment mémorise-t-on un bit dans une EPROM ? Comment l'efface-t-on ?

Voir la section 10.3 à la page 50.

Comparez RAM statique et RAM dynamique

Voir la section 10.4.1 à la page 52 et la section 10.4.2 à la page 52.

Comment est organisé un module de mémoire RAM

Voir la section 10.5 à la page 53.

Qu'est-ce qu'une mémoire RAM synchrones/asynchrones ?

Voir la section 10.6 à la page 54.

Quelle technique a-t-on utilisé pour augmenter les débits des mémoires RAM

Voir la section 10.6 à la page 54.