Architecture des ordinateurs
IUP GEII - informatique & télécommunications
1ère année

Patrick Marcel

24 janvier 2001

Table des matières

Table des figures

Chapitre 1

Préambule

Contexte Ceci est un support de cours disponible aux formats html (http://www.blois.univ-tours.fr/ marcel/archi) et postscript. Il concerne le cours d'archiecture des ordinateurs proposé aux étudiants de première année de l'IUP GEII option informatique et télécommunications de Blois.

Objectif Qu'est ce qu'un ordinateur ? Selon un dictionnaire Hachette :

> ordinateur : n. m. INFORM *Machine* capable d'effectuer *automatiquement* des *opérations arithmétiques* et *logiques* (à des fins scientifiques, administratives, comptables, etc.) à partir de *programmes* définissant la *séquence* de ces opérations.

Ce cours propose de répondre à la question “qu'est ce qu'un ordinateur ?” en répondant à la question “comment fonctionne un ordinateur ?”.

Moyens Nous disposons de 40 heures de cours et 15 heures de TD durant lesquelles nous aborderons les points suivants :

- introduction : historique et aperçu (cours 2)
- pré-requis : codage et algèbre de Boole (cours 3)
- arithmétique et circuits de calcul (cours 4)
- mémoire (cours 5)
- interconnexions - bus (cours 6)
- jeux d'instructions (cours 7)
- CPU (cours 8)
- unité de commande (9)

La méthode de progression est la suivante : le cours introduit les principes généraux et quelques exemples fondamentaux. Les TDs font l'objet de séances particulière durant lesquelles les notions de cours sont rappelées, explicités, au besoin étendues. Ils consistent en une série d'exercice d'application et d'approfondissement. Les TDs, ainsi que leur correction, sont évalués.

Je souhaite également mentionner comme moyen toute contribution que le lecteur voudra m'apporter.

Remerciements et bibliographie Pour ce support de cours, je me suis inspiré

- du cours d'architecture de Mohamed Taghelit que le présent cours remplace,
- du livre “computer organization and architecture (fifth edition)” de William Stallings,
- de documents et supports de cours trouvés sur le web, notamment ceux de Michel Cubero-Castan, Guy Chesnot, M. Billaud, Patrick Trau et Reto Zimmermann.

Merci à toutes ces personnes pour m'avoir permis l'accès à leur documents.

Chapitre 2

Introduction

2.1 Introduction

Le but de ce premier cours est d'avoir une première idée de ce qui se passe à l'intérieur de la machine lorsqu'un utilisateur interragit avec elle. C'est un cours d'introduction qui comprend :

- introduction
- historique
- machine Von Neumann
- aperçu
- conclusion

A titre d'exemple, on s'interesse à la séquence d'évènements suivante : lecture d'un nombre au clavier, opération sur ce nombre, affichage du résultat à l'écran.

La machine perçue par l'utilisateur *Interactions* avec la machine au travers des *périphériques* (dispositifs externes à *l'unité centrale*) :

- clavier,
- souris,
- écran,
- imprimante,
- disquette, etc...

Interactions = entrées de commandes, lancements d'application, visualisation, etc...

La machine invisible à l'utilisateur La boite noire qu'est l'unité centrale contient en fait 3 unités fonctionnelles :

- l'automate,
- la partie calcul,
- la mémoire.

! *Toute action de l'utilisateur se traduit par l'exécution d'une séquence d'opérations (faisant intervenir ces unités).*

2.2 Historique

! *La motivation principale est la volonté d'automatiser une suite de tâches élémentaires.*

D'abord existent des machines / automates spécialisés : boite à musique, métiers à tisser... puis on associe automate et machine à calcul. Charles Babbage (19ème sciècle) fut le premier à décrire les principes d'un calculateur d'application

générale (machine pouvant répéter des séquences d'opérations et choisir une série d'opérations particulière en fonction de l'état du calcul). Le modèle Von Neumann (1946) pose les bases des machines universelles (cf. section 2.3).
On observe généralement 5 générations (étapes décisives) dans l'évolution (principalement technologique) de ces machines :

1945-1958

- ordinateurs dédiés, exemplaire uniques
- machines volumineuses et peu fiables
- technologie à lampes, relais, resistances
- 10^4 éléments logiques
- programmation par cartes perforées

! *la plupart des concepts architecturaux (notamment le bug !) des ordinateurs modernes datent de cette époque.*

Notons que le premier ordinateur numérique généraliste, l'ENIAC, commandé par l'armée américaine en 1943 et réalisé en 1946, a eu pour première tâche des calculs complexes pour l'étude de faisabilité de la bombe H.

1958-1964

- usage général, machine fiable
- technologie à transistors
- 10^5 éléments logiques
- apparition des langages de programmation évolués (COBOL, FORTRAN, LISP)

1964-1971 ou 75 ou 78

- technologie des circuits intégrés (S/MSI small/medium scale integration)
- 10^6 éléments logiques
- avènement du système d'exploitation complexe, des mini-ordinateurs.

! *La loi de Moore (co-fondateur d'INTEL) veut que le nombre de transistors intégrables sur une seule puce double chaque année.*

1971/5/8-1978/85

- technologie LSI (large SI)
- 10^7 éléments logiques
- avènement de réseaux de machines
- traitement distribué/réparti

! *1971 : invention par INTEL du microprocesseur = toutes les composantes de la CPU sont réunies sur une même puce.*

après Le but originel de cette cinqième génération était les machines langages dédiées à l'IA...

- technologie VL/WSI (very large, wafer)
- 10^8 éléments logiques (le PII contient 7,5 millions de transistors, mémoire non comprise)
- systèmes distribués interactif
- multimédia, traitement de données non numériques (textes, images, paroles)
- parallèlisme massif

2.3 Machine Von Neumann

John Von Neumann est à l'origine (1946) d'un modèle de machine universelle (non spécialisée) qui caractérise les machines possèdant les éléments suivants :

- une mémoire contenant programme (instructions) et données,
- une unité arithmétique et logique (UAL ou ALU),
- une unité permettant l'échange d'information avec les périphériques : l'unité d'entrée/sortie (E/S ou I/O),
- une unité de commande (UC).

Ces dispositifs permettent la mise en œuvres des fonctions de base d'un ordinateur : le stockage de données, le traitement des données, le mouvement des données et le contrôle. Le fonctionnement schématique en est le suivant : l'UC

1. extrait une instruction de la mémoire,
2. analyse l'instruction,
3. recherche dans la mémoire les données concernées par l'instruction,
4. déclenche l'opération adéquate sur l'ALU ou l'E/S,
5. range au besoin le résultat dans la mémoire.

! *La plupart des machines actuelles s'appuient sur le modèle Von Neumann.*

2.4 Aperçu

Cet aperçu est une introduction au comportement des différentes éléments composant l'unité centrale. Il est très schématique et sera développé dans les cours suivants. Il introduit des notions de base comme le cycle d'horloge, les *chronogrammes*.

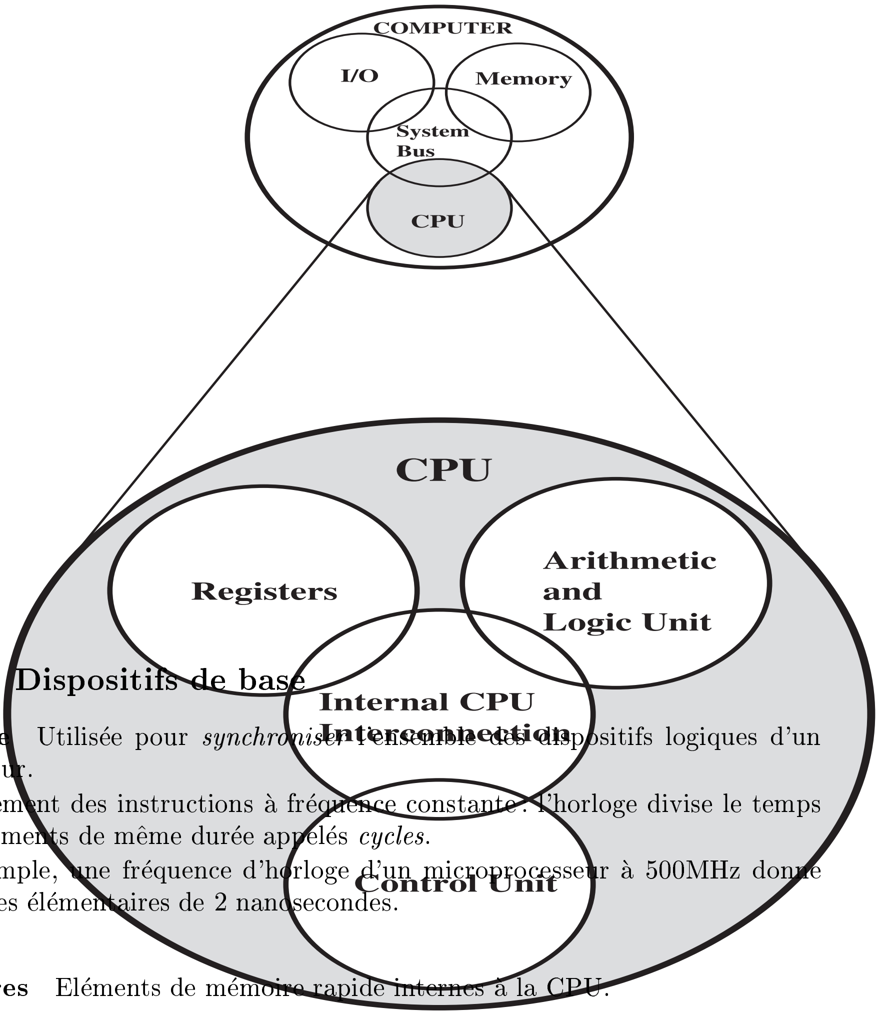


Figure 1.5 The Central Processing Unit (CPU)

2.4.1 Dispositifs de base

Horloge Utilisée pour *synchroniser* l'ensemble des dispositifs logiques d'un ordinateur.
Cadencement des instructions à fréquence constante : l'horloge divise le temps en battements de même durée appelés *cycles*.
Par exemple, une fréquence d'horloge d'un microprocesseur à 500MHz donne des cycles élémentaires de 2 nanosecondes.

Registres Eléments de mémoire rapide internes à la CPU.

! *Plus petit est l'espace de recherche, plus rapide est l'accès à l'information.*

Mémorisation commandée par un signal de chargement : 2 types de registres suivant que le chargement se fait sur *niveau ou sur front*.

Bus Ensemble de fils électriques sur lesquels transitent les informations entre les unités.
Largeur du bus = nombre de fils constituant le chemin = nombre d'*impulsions électriques* pouvant être envoyés en parallèle (en même temps).

2.4.2 Unités fonctionnelles

Mémoire Vecteur dont chaque composante est accessible par une *adresse*.
Les opérations permises sur la mémoires sont les opérations de *lecture* et *d'écriture*.
L'UC inscrit l'adresse d'une cellule dans un registre d'adresse (RA) et demande une opération de lecture ou d'écriture. Les échanges se font par l'intermédiaire d'un registre de mot (RM).
Le *mot* = l'unité d'information accessible en une seule opération de lecture (sa taille varie en fonction de la machine).

ALU Vue comme une fonction à 3 paramètres : 1 opération, 2 arguments. Elle renvoie un résultat.
Un registre lui est associé : l' accumulateur (ACC) pour par exemple mémoriser un résultat intermédiaire.

E/S Sert d'interface avec les périphériques.
Les opérations associées (lecture et/ou écriture) sont fonction du périphérique.

? *Quels sont les périphériques où seule est permise la lecture ? l'écriture ? les deux ?*

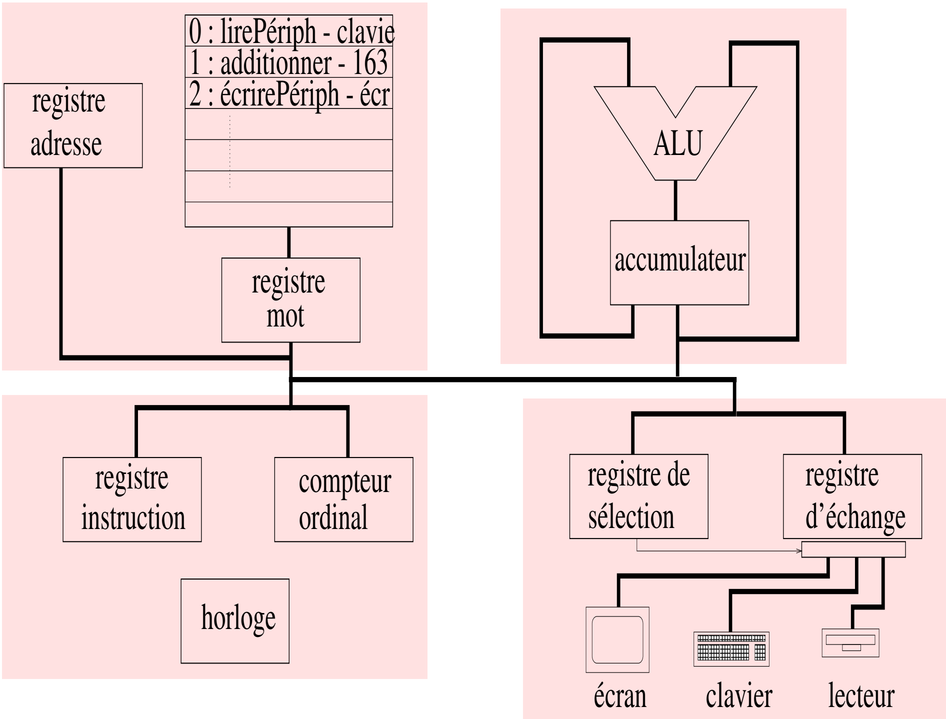
De manière simiulaire à la mémoire, on dispose d'un registre mémorisant l'adresse du périphérique (le registre de sélection du périphérique (RSP)) et d'un registre d'échange de données (RE).

Unité de commandes Son fonctionnement est celui décrit plus haut.
Compteur ordinal (PC) = registre contenant l'adresse mémoire de l'instruction à exécuter.
Registre d'instruction (RI) mémorise l'instruction (une instruction est composée de plusieurs parties, ou champs)

La machine complète Une mémoire, une ALU, une unité de commande, une unité d'E/S, un bus, et hop !

Jeux d'instructions Différents formats d'instruction suivant le nombre de parties réservées aux opérandes (ou adresses).
Par exemples :

| code opération | opérande | (format 1 adresse)
| code opération | opérande 1 | opérande 2 | (format 2 adresse)



Exemple d'instructions au format 1 adresse :

- lirePériph - nomPériph
- additionner - adresse

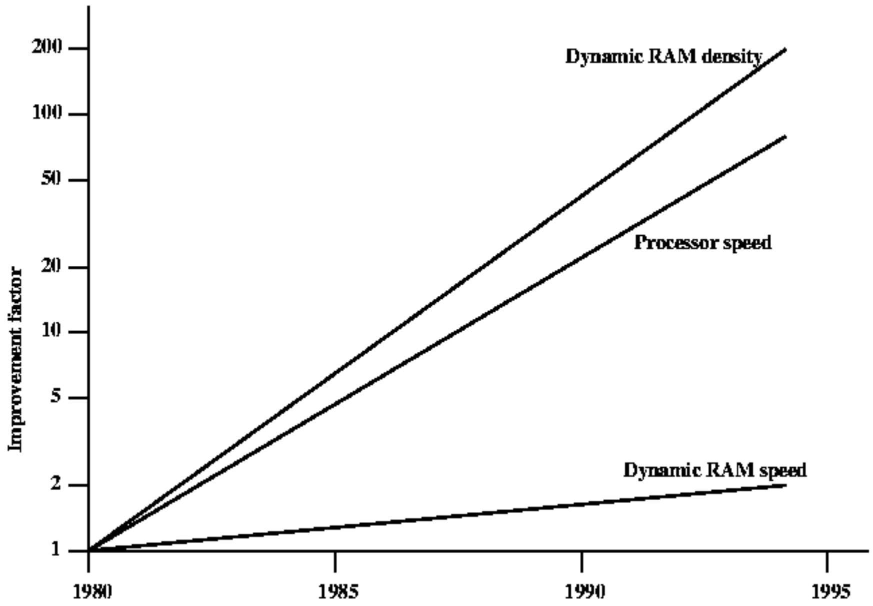
Exemple de fonctionnement avec une horloge à 4 phases (pour un format 1 adresse) : cas d'une acquisition au clavier, d'une addition de la valeur lue avec une donnée en mémoire, puis affichage du résultat à l'écran.

$H_0 : RA \leftarrow PC$	
$H_1 : RI \leftarrow RM, PC \leftarrow PC + 1$	l'instruction est lirePériph - nomPériph
$H_2 : RSP \leftarrow RI_p$	RI_p est la partie nomPériph de l'instruction
$H_3 : ACC \leftarrow RE$	
$H_0 : RA \leftarrow PC$	nouveau cycle, nouvelle instruction
$H_1 : RI \leftarrow RM, PC \leftarrow PC + 1$	l'instruction est additionner - adresse
$H_2 : RA \leftarrow RI_p$	RI_p est la partie adresse de l'instruction
$H_3 : ACC \leftarrow ACC + RM$	
$H_0 : RA \leftarrow PC$	nouveau cycle, nouvelle instruction
$H_1 : RI \leftarrow RM, PC \leftarrow PC + 1$	l'instruction est écrirePériph - nomPériph
$H_2 : RSP \leftarrow RI_p$	RI_p est la partie nomPériph de l'instruction
$H_3 : RE \leftarrow ACC$	

chargeur = programme qui lit un programme (sur un support externe) et l'écrit en mémoire. Le chargeur est chargé par un dispositif matériel/ROM (appelé microchargeur).

2.5 Conclusion

L'évolution technologique des ordinateurs est liée à l'évolution des besoins (en terme d'application). Ce qui explique la course aux performances (et indirectement le challenge qu'il y a à gérer la diversité dans l'évolution des performances de chaque composant).



! *Le principe de conception d'une machine est lié à la performance.*

Quelles sont les mesures de performance objectives :
- la fréquence d'horloge ? objectif pour une même architecture
- nombre d'instruction /s ? objectif pour un même jeu d'instructions (MIPS, MFLOPS)
- temps d'exécution des programmes (benchmarks) ? oui !

! *La performance dépend fortement de l'architecture !*

Chapitre 3

Pré-requis

3.1 Introduction

Le but de ce cours est de présenter les pré-requis à l'étude des unités fonctionnelles d'une machine. Ces pré-requis sont principalement la numération binaire et la logique booléenne. De manière plus précise, ce cours étudie :

- codage
- numération
- algèbre de Boole
- circuits combinatoires
- circuits séquentiels

3.2 Codage

3.2.1 Principe de codage

Soit I un ensemble d'informations. Soit $A = \{a_1, \dots, a_n\}$ un ensemble fini de symboles appelé *alphabet*. Les a_i sont appelés *caractères* de A. Un ensemble ordonné de caractères est appelé *mot*. La *base* du codage est le cardinal de l'ensemble A.
Coder I consiste à faire correspondre à chaque éléments de I un mot de A. Un codage est *redondant* si un élément est associé à plusieurs codes.
Un codage peut être à longueur fixe :

- numéro de téléphone d'un particulier
- numéro de sécurité sociale
- code postal

ou à longueur variable

- alphabet morse
- ADN

? *Pour un codage de longueur fixe n, si b est la base du codage, monter que l'on peut représenter b^n éléments, et que l'on a alors $b^n!$ codages possibles.*

3.2.2 Quelques codes

! *Ne pas confondre un nombre et sa représentation !*

Notation positionelle La représentation d'un nombre est un codage. La représentation d'un nombre dans une base b donnée:

$$a_n a_{n-1} \dots a_1 a_0 = \sum_{i=0}^{n} a_i \times b^i$$

Code binaire naturel Alphabet de 2 caractères : 0 et 1 (caractères binaires appelés bits)
Correspondance basée sur la représentation des nombres en base 2.

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

? *Pourquoi ce tableau ne contient pas le code de 8 ?*

Généralement, les bits les plus à gauche sont appelés poids forts.

! *Le codage binaire naturel n'est qu'un codage particulier.*

Code BCD/DCB Décimal Codé Binaire : chaque chiffre d'un nombre est sur codé 4 bits

0	0000
1	0001
2	0011
⋮	⋮
10	0001 0000
11	0001 0001

Ce code simplifie la conversion décimal binaire.

Code de Gray Distance de 1 entre deux mots de code consécutif

0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Ce code évite le changement simultané de 2 bits, et donc les états transitoires indésirables.

Détection d'erreurs La transformation d'un 0 en 1 ou d'un 1 en 0 est une erreur fréquente en informatique (problème de transmission, défaillance d'un circuit, etc...)
Un code redondant peut être utilisé pour *détecter* des erreurs.
Exemple : ajout d'un bit (dit bit de parité). Le bit supplémentaire maintient la parité de l'information :

- 0110 devient 01100,
- 1011 devient 10111

Il existe d'autres codes *correcteur* d'erreurs, où l'augmentation de la redondance permet de détecter et corriger une erreur. Dans le code de Hamming, 3 bits sont ajoutés à quatre bits de données pour contrôler la parité de trois groupes de trois bits de données différents.

? *Comment un tel codage permet t'il la détection d'une erreur ?*

Codage des données alphanumériques Face aux multiple possibilités de codage, des organismes de normalisation ont vu le jour (le plus célèbre étant l'ISO).
Code ASCII = code employé pour les caractères alphanumériques (a,b,1,?, ,,,).
Code sur 7 bits (+ 1 bit de parité).

- A est codé par 1000001,
- e est codé par 1100101,
- 7 est codé par 0110111,
- ! est codé par 0100001,
- etc...

Unicode = code récent sur 16 bits contenant pratiquement tous les alphabets existants (employé dans java).

3.3 Système de numération

L'homme a 10 doigt. Le système de numération humain est le système décimal.
L'ordinateur a 2 état significatifs (impulsion électriques). Le système de numération qu'il emploie est donc le système binaire.

! *L'homme compte sur ses doigts, l'ordinateur compte sur ses bits.*

3.3.1 Conversions binaire-décimale

Base b vers base 10 $a_n a_{n-1} \ldots a_1 a_0$ exprimé en base b (noté ${a_n a_{n-1} \ldots a_1 a_0}_b$) vers une représentation en base 10 :

$$a_n \times b^n + a_{n-1} \times b^{n-1} + \ldots + a_1 \times b + a_0$$

Cas de la partie fractionnaire :

$$a_1 \times b^{-1} + a_2 \times b^{-2} + \ldots + a_n \times b^{-n}$$

Base 10 vers base b

$$A_{10} = a_n \times b^n + a_{n-1} \times b^{n-1} + \ldots + a_1 \times b + a_0$$

$$= ((\ldots(a_n \times b + a_{n-1}) \times b + \ldots) \times b + a_1) \times b + a_0$$

a_0 est le reste de la division entière du nombre par la base b. Des divisions entières successives par la base donnent donc tous les a_i.
Cas de la partie fractionnaire : sur un principe similaire, des multiplications successives par la base donnent tous les a_i.

? *Quelles sont les représentations de 100 dans toutes les bases inférieures à 10 ?*

? *Conversion avec une base puissance de la base de départ, par exemple octale ($8 = 2^3$) ou hexadécimale ($16 = 2^4$) et binaire.*

3.3.2 Représentation des nombres négatifs

Signe et valeur absolue Sur n bits : signe = bit de poids fort (0 : positif, 1 : négatif), n-1 bits = valeur absolue.
Intervalle de valeurs représentés pour n bits : $[-2^{n-1} + 1, 2^{n-1} - 1]$

? *Quelles sont les 2 représentations possibles pour zéro ?*

Notations complémentés Complément à 1: inversion de chaque bit de la valeur absolue du nombre à représenter.

-1	110
-2	101
-3	100

Intervalle de valeurs représentés pour n bits : $[-2^{n-1}+1, 2^{n-1}-1]$

? *Quelles sont les 2 représentations possibles pour zéro ?*

Complément à 2 = complément à 1 + 1

-1	111
-2	110
-3	101

Intervalle de valeurs représentés pour n bits : $[-2^{n-1}, 2^{n-1}-1]$

? *Combien de représentation possible pour zéro ?*

Notation excédentaire Ajout au nombre de la valeur d'un excès (souvent translation de 2^{n-1}, ainsi le bit de poids fort fait office de bit de signe).
Intervalle de valeurs représentés pour n bits avec un excès de 2^{n-1} : $[-2^{n-1}, 2^{n-1}-1]$
Intérêt : simplifie toutes les opérations ou les comparaisons (qui se font uniquement sur des nombres positifs).

3.3.3 Représentation des nombres réels

Plusieurs représentations possibles :
- virgule fixe : revient à manipuler des entiers (caisses enregistreuses)
- couple (numérateur,dénominateur) : représentation juste uniquement pour les nombres rationnels
- virgule flottante

Virgule flottante Un nombre réel $\pm m \times b^e$ est représenté par un signe, une mantisse m, un exposant e, et une base b.

! *La représentation en virgule flottante est la représentation des réels la plus utilisée.*

Il existe une infinité de représentation du même nombre. Représentation *normalisée* : pour une base b, la mantisse est prise dans l'intervalle $[1, b[$ (zéro admet une représentation particulière).
La *précision* et l'intervalle de valeurs représentées dépendent du nombre de bits utilisés pour coder la mantisee et l'exposant.

Norme IEEE754

- nombres codés sur 32 bits (simple précision), ou 64 bits (double précision)
- la mantisse appartient à l'intervalle $[1,0, 10,0[$ (en binaire)
- le seul chiffre à gauche de la virgule étant toujours 1, n'est pas représenté
- l'exposant est codé avec un excès de 127 (simple précision) ou 1023 (double précision)

Sur 32 bits : $(-1)^S \times 1, M \times 2^{E-127}$

S 1 bits	exposant E 8 bits	mantisse M 23 bits

Exemple : -5 est codé par 1100 0000 1010 0000 0000 0000 0000 0000

? *Vérifier que ce codage est le bon !*

! *La norme IEEE 754 est la norme la plus utilisée pour représenter les réels.*

Précision La représentation des nombre réels sur une machine se base sur un nombre fini de valeurs. C'est donc une représentation approchée.
Précision de la représentation = différence entre les mantisses de deux nombres réels consécutifs.

! *La précision ne dépend que de la mantisse.*

IEEE 754, simple précision (32 bits) : la précision est de 2^{-23}.
Les *valeurs particulières* représentés avec IEEE 754 :

E	M	valeur représentée
max	0	$\pm\infty$
max	$\neq 0$	NaN
0	0	0
0	$\neq$	nombres dénormalisés

La valeur (propageable) NaN signifie Not A Number. Elle est pratique pour représenter le résultat de certains calcul (e.g., $\sqrt{-1}$).

3.4 Algèbre de Boole

Constitue une partie des travaux de Georges Boole (1815-1864) relatif à l'élaboration d'une base mathématique au raisonnement logique.
Une algèbre de Boole est la donnée de :

- un ensemble E,
- deux éléments particuliers de E : 0 et 1,
- deux opérations binaires sur E : + et . (le . sera parfois omis),
- une opération unaire sur E.

qui vérifient les *axiomes* suivants : soient $a, b \in E$

commutativité	a+b=b+a	ab=ba
associativité	(a+b)+c=a+(b+c)	(ab)c=a(bc)
distributivité	a(b+c)=ab+ac	a+(bc)=(a+b)(a+c)
éléments neutres	a+0=a	a1=a
complémentation	a+$\overline{a}$=1	a$\overline{a}$=0

Les *théorèmes* suivants peuvent être déduits :

idempotence	a+a=a	aa=a
absorption	a+ab=a	a(a+b)=a
De Morgan (dualité)	$\overline{a+b}=\overline{a}.\overline{b}$	$\overline{ab}=\overline{a}+\overline{b}$
éléments absorbants	a+1=1	a0=0

? *Démontrer que aa=a*

On se restreind ici à l'algèbre de Boole minimale, où $E = \{0, 1\}$. Cette algèbre peut être interprétée comme suit :

- 1 est interprété "vrai" et 0 "faux",
- + est interprété "ou" (disjonction logique)
- . est interprété "et" (conjonction logique)
- $-$ est interprété "non" (négation logique)

? *L'algèbre des ensembles munis des opérations d'union, d'intersection et de complémentation est elle une algèbre de boole ?*

Tables de vérité et portes logiques On a les *tables de vérités* suivantes :

a	$\overline{a}$
0	1
1	0

a	b	a + b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	ab
0	0	0
0	1	0
1	0	0
1	1	1

? *Comment peut être interprétée l'opération logique dont la table est :*

a	b	$a \oplus b$
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*0*

Ces opérations sont mises en œuvres par des circuits logiques de base appelés *portes logiques.*

! *La plupart des circuits intégrés des ordinateurs actuels sont conçus à partir de portes NON-ET (NAND) et NON-OU (NOR).*

? *Montrer que les opérations ET, OU et NON sont réalisables à partir de portes NAND.*

Expressions et fonctions booléennes *Expression booléenne* = terme construit à partir de variables booléennes et d'opérateur booléens.
Fonction booléenne = fonction binaire à variables binaires (de $\{0,1\}^n$ dans $\{0,1\}$).
Une fonction booléenne peut être décrite par :

- sa table de vérité (recensant toutes les combinaisons de valeur des variables)
- une expression booléenne (décrivant la sortie 1 de la fonction, par convention, $\overline{a}$ désigne la valeur 0 pour a)

Exemple : fonction booléenne majorité s'exprime par

a	b	c	majorité
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

ou : majorité(a,b,c) = $\overline{a}$bc + a$\overline{b}$c + ab$\overline{c}$ + abc

? *Montrer qu'il y a 2^{2^n} fonctions booléennes à n arguments ?*

! *Les portes NAND et NOR sont dites complètes car on peut réaliser n'importe quelle fonction boléenne avec l'une ou l'autre.*

Simplificaton d'expressions booléennes Deux fonctions booléennes sont équivalentes ssi les valeurs de leurs sorties sont les mêmes pour toutes les configurations identiques de leurs variables d'entrées.
Il est intéressant de minimiser le coût (en nombre de portes logiques) de réalisation d'une fonction en trouvant l'expression booléenne de cette fonction la moins couteuse.

? *Montrer que la fonction majorité(a,b,c) se simplifie en bc + ac + ab*

? *Un afficheur 7 segments est composé de 7 diodes notés a, b, ..., g disposées "en 8". Des nombres sont fournis en binaire sur 4 bits, $x_1x_2x_3x_4$. Donner l'expression des 7 fonctions $a(x_1x_2x_3x_4)$, $b(x_1x_2x_3x_4)$, etc... qui permettent d'afficher les nombres fournis.*

3.5 Les circuits logiques combinatoires

Circuits combinatoires = circuits dont la fonction de sortie s'exprime par une expression logique des seules variables d'entrées.
Voici quelques circuits combinatoires intervenant dans la réalisation des composants logiques d'un ordinateur.

Décodeur Circuit permettant d'envoyer un signal à une sortie choisie.
Il dispose de

- n lignes d'entrées
- 2^n lignes de sortie

La table de vérité d'un décodeur "2 vers 4" ($n = 2$) est la suivante :

e_1	e_0	s_0	s_1	s_2	s_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Et voici la réalisation d'un tel décodeur :

Multiplexeur Circuit permettant de sélectionner une entrée parmi plusieurs. Il dispose de

- 2^n entrées
- 1 sortie
- n lignes de sélection

Voici la réalisation d'un multiplexeur "4 voies" ($n = 2$) :

? *Sachant sa réalisation, quelle est la table de vérité de ce multiplexeur ?*

Comparateur Circuit effectuant la comparaison de deux nombres binaires.

? *Réaliser un comparateur de mots de 4 bits.*

3.6 Les circuits séquentiels

Aussi appelés circuits de mémorisation car leur sorties dépendent :

- de l'état des variables d'entrée
- de l'état antérieur de certaines variables de sortie

Un circuit séquentiel possède des entrées E, des sorties S et un état interne Q. Il est défini par deux fonctions $S = f(E, Q)$ et $Q' = g(E, Q)$ indiquant respectivement la nouvelle sortie et le nouvel état.

Notion d'état stable Bistable = 2 portes NON montées en oppposition.

Ce circuit a 2 états *stables* différents :

- $Q_1 = 0$ et $Q_2 = 1$
- $Q_1 = 1$ et $Q_2 = 0$

! *Le circuit composé d'une porte NON rebouclée sur elle-même est un circuit* astable.

Bascule RS Ajout de commandes reset (R) et set (S) au bistable.

Table de vérité (Q_i représente la sortie Q à l'instant i, x un état quelconque) :

R	S	Q_i	Q_{i+1}
0	0	x	x
0	1	x	1
1	0	x	0
1	1	x	interdit

? *Que se passe t'il si R et S valent 1 simultanément ? S'ils repassent ensuite simultanément à 0 ?*

Bascule RST L'horloge peut être vue comme la synchronisation de l'odre d'apparition des variables logiques.

L'ajout d'une entrée d'horloge permet de valider les commandes R et S :
- si T = 1, fonctionnement comme une bascule RS
- si T = 0, conservation de l'état courant

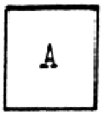
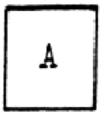
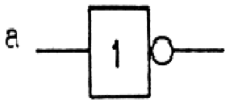
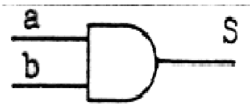
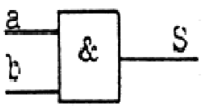
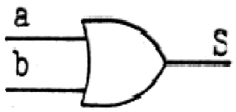
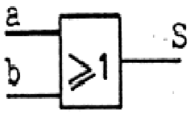
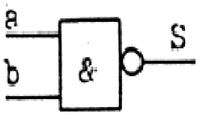
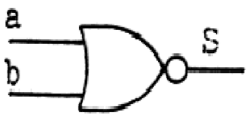
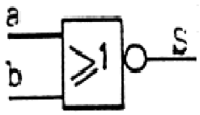
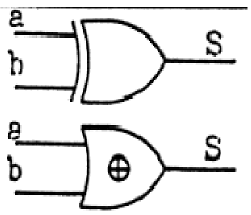
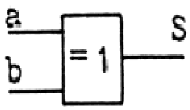
Bascule D Aussi appelée latch. Intérêt :
- pas d'état instable
- mémorisation de l'entrée

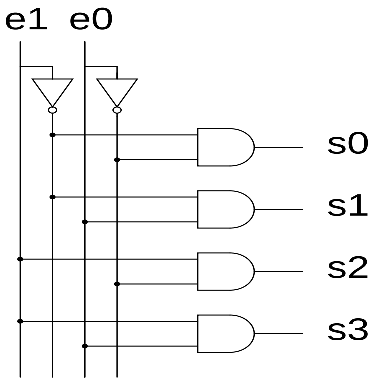
! *La bascule D est utilisée pour la conception des mémoires.*

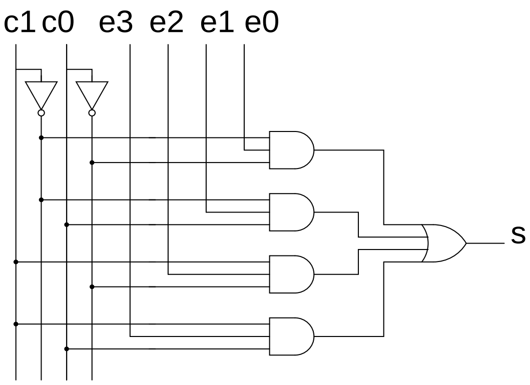
Bascule JK Bascule RS avec sorties rebouclées sur les entrées : pas d'état interdit.

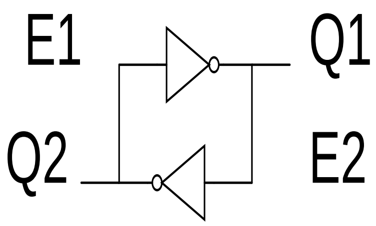
J	K	Q_i	Q_{i+1}
0	0	x	x
0	1	x	0
1	0	x	1
1	1	x	$\overline{x}$

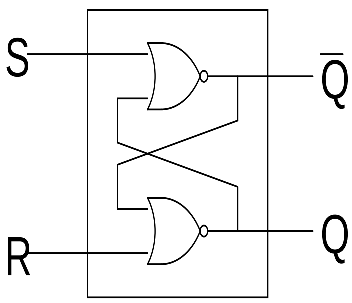
SYMBOLES DES OPERATEURS LOGIQUES

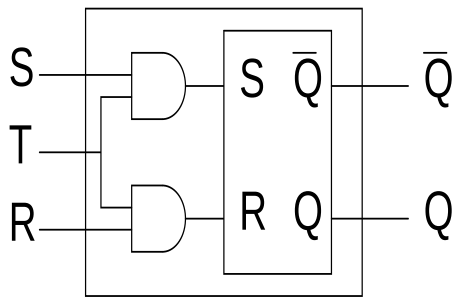
Symbole (Norme MILSTD 086B)	**Equations**	**Symbole** (notation française)
	Ensemble quelconque (la fonction est notée ou symbolisée à l'intérieur)	
	Inverseur	
	Ampli inverseur	
	Opérateur ET (AND) $S = a.b$	
	Opérateur OU (OR) $S = a + b$	
	Opérateur NON ET (NAND) $S = \overline{a.b} = \overline{a} + \overline{b}$	
	Opérateur NON OU (NOR) $S = \overline{a + b} = \overline{a}.\overline{b}$	
	Opérateur OU Exclusif (XOR) $S = a\overline{b} + \overline{a}b$ $S = a \oplus b$	

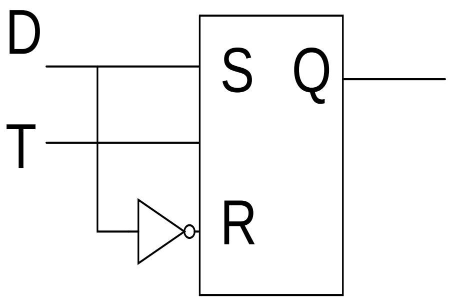

e1 e0
s0
s1
s2
s3

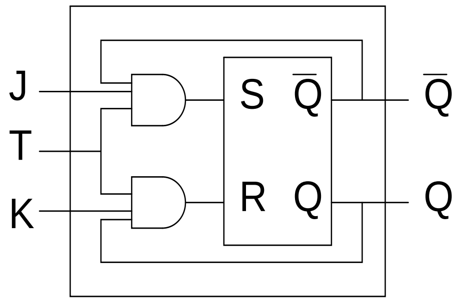

c1 c0 e3 e2 e1 e0
s


E1
Q1
Q2
E2


S
$\overline{Q}$
R
Q


S
T
R
S
$\overline{Q}$
R
Q
$\overline{Q}$
Q


D
T
S
Q
R


J
T
K
S
$\overline{Q}$
R
Q
$\overline{Q}$
Q

Chapitre 4

Circuits de calcul

4.1 Introduction

! *INTEL a dépensé 300 millions de dollars pour le remplacement de cicruits PENTIUM bogués...*

Ce cours suit le cours sur les prérequis en continuant les manipulation de nombres codés selon différentes représentations binaires (simple, complémentés, IEEE 754, ...) Il est constitué d'une première partie portant sur les opérations arithmétiques (en binaire pure, et en flottant) et d'une deuxième partie sur la mise en oeuvre de ces opérations.

! *Dans ce cours, le symbole "+" désigne l'addition et non le ou logique. Le ou logique sera désigné par le symbole "$\vee$" et le et logique par le symbole "$\wedge$".*

4.2 Arithmétique

4.2.1 Arithmétique binaire

Les opérations en binaire s'effectuent comme en base 10 :

- pour l'addition et la soustraction, on opère chiffre par chiffre, des poids faibes aux poids forts, en propageant la retenue,
- pour la multiplication et la division, on procède par série d'additions ou de soustractions.

Addition et soustraction Ces opérations peuvent être décrites par les tables de vérité suivantes :

a	b	somme	retenue
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

somme(a,b) = a $\oplus$ b
retenue(a,b) = ab

a	b	différence	retenue
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

différence(a,b) = a $\oplus$ b
retenue(a,b) = $\overline{a}$b

Exemple : en binaire naturel
27 + 22 = 49 : 11011 + 10110 = 110001
27 - 22 = 5 : 11011 - 10110 = 000101

Débordement (overflow) Il intervient lorsque le résultat de l'opération n'est pas représentable dans le système utilisé (i.e., avec le nombre de chiffres utilisé). Dans le cas d'une addition de deux nombres en codage binaire naturel, le débordement correspond à une retenue sortante à 1 (cas de 27 + 22 sur 5 bits).

Addition avec le complément à deux Considérons d'abord le cas d'une addition *sans débordement* (on ne considère que les n bits résultats d'une opération sur deux nombres sur n bits). Détaillons les 4 cas (selon les signes des opérandes) :

5	00101	-5	11011	-5	11011	5	00101
9	01001	-9	10111	9	01001	-9	10111
14	01110	-14	10010	4	00100	-4	11100

Comment est-ce possible ? Cas de l'addition de X et Y deux entiers négatifs. Soit $Z = X + Y$. En complément à deux, $-\mid X \mid = 2^n - \mid X \mid$. Donc

$$
\begin{aligned}
Z &= (2^n - \mid X \mid) + (2^n - \mid Y \mid) \\
&= 2^{n+1} - (\mid X \mid + \mid Y \mid) \\
&= 2^{n+1} - \mid Z \mid
\end{aligned}
$$

ce qui sur n bits donne bien 2^n - $\mid Z \mid$.
Considérons le cas du débordement : il y a débordement si

- les opérandes X et Y sont de même signe, et
- $\mid X \mid + \mid Y \mid \geq 2^{n+1}$

	13	01101	-13	10011
	9	01001	-9	10111
sur n bits	-10	10110	10	01010
sur n+1 bits	22	010110	-22	101010

Le débordement peut être détecté :

- en comparant le signe des opérandes au signe du résultat : s'ils sont différents, il y a débordement, ou
- en comparant la retenue entrante dans le bit de poids fort avec la retenue sortante : s'ils sont différents, il y a débordement.

? *Comment fonctionne l'addition de nombres codés en complément à 1 ?*

Multiplication et division La multiplication est très simple : elle ne consiste qu'en des additions successives du multiplicande avec lui même décalé. Le résultat d'une multiplication avec des opérandes de n bits est codé sur $2n$ bits.

! *Ces opérations fonctionnent aussi avec des nombres codés en notations complémentées.*

Exemple :

11	01011	-5	1...11011	11	01011
13	01101	-3	1...11101	-3	1...11101
143	010001111	15	000001111	-33	111011111

La division consiste en des soustraction successives. Par exemple : $15_{10}/3_{10} = 1111/11 = 101$

? *Calculer* $1101,11 \times 110,01$ *et* $1110,10/1101$

4.2.2 Arithmétique flottante

Multiplication C'est l'opération la plus simple en virgule flottante. Pour deux nombres $x_1 = (-1)^{s_1} \times m_1 \times 2^{e_1}$ et $x_2 = (-1)^{s_1} \times m_2 \times 2^{e_2}$, on a $x_1 \times x_2 = (-1)^{s_1} \times (-1)^{s_2} \times m_1 \times m_2 \times 2^{e_1+e_2}$.
Le principe général est donc :

1. calcul du signe $(s_1 \oplus s_2)$
2. multiplication entière des mantisses : $m_1 \times m_2$ (cf plus haut)
3. mise de la mantisee à p bits (cf plus bas)
4. calcul de l'exposant : e_1+e_2 (enlever une fois la valeur de l'excès au besoin !)

Arrondi La multiplication des deux mantisses sur p bits donne un résultat sur $2p$ bits, duquel on doit conserver une mantisse de p bits. Il faut donc arrondir et éventuellement ajuster l'exposant pour normaliser le résultat.
Pour arrondir, on s'appuie sur :

- *le bit de garde* : $p + 1^{ieme}$ bit
- *le bit d'arrondi* : $p + 2^{ieme}$ bit
- *le bit persistant* : un ou logique des bits éliminés

Les p bits gardés sont fonction du bit de poids fort du résultat :

- si le bit de poids fort est 0, le résultat obtenu est de la forme 01,... On doit conserver le bit de garde dans le résultat.
- si le bit de poids fort est 1, le résultat est de la forme 10,... On ajuste l'exposant, le bit de garde devient le bit d'arrondi, et l'ancien bit d'arrondi entre dans le calcul du bit persistant.

? *Multiplication de 5 par 10 et 5 par 15 avec mantisses de 4 bits.*

Il y a plusieurs manières d'arrondir...

Les arrondis IEEE Soient r le bit d'arrondi, s le bit persistant, et p_0 bit de poid faible du résultat. Le standard IEEE propose 4 approches d'arrondi :

mode d'arrondi	résultat positif ou nul	résultat négatif
au plus près	résultat + (r $\wedge p_0$) $\vee$ (r $\wedge$ s)	résultat + (r $\wedge p_0$) $\vee$ (r $\wedge$ s)
vers - ∞	inchangé	résultat + (r $\vee$ s)
vers + ∞	résultat + (r $\vee$ s)	inchangé
0	inchangé	inchangé

L'arrondi au plus près, qui est l'approche par défaut, consiste à arrondir à la plus proche valeur représentable.

? *En utilisant une mantisse sur 6 bits, calculer la mantisse du résultat de l'opération :* $1,01011 \times 1,00101$ *en utilisant un arrondi au plus près.*

Débordement Il intervient lorsque l'exposant du résultat est supérieur à la valeur maximum ou inférieur à la valeur minimum.

Addition L'algorithme d'addition de deux nombres flottants est un peu plus compliqué, car il faut non seulement tenir compte des signes, mais aussi aligner les virgules.

? *Appliquer l'algorithme (distribué en cours) : effectuer sur 4 bits* $-1,001 \times 2^{-2} + -1,111 \times 2^{0}$*, puis* $1,25 - 1,5$.

Division Pour deux nombres $x_1 = s_1 \times 2^{e_1}$ et $x_2 = s_2 \times 2^{e_2}$, on a $x_1/x_2 = s_1/s_2 \times 2^{e_1-e_2}$.
Pour diviser deux mantisses sur p bits, on calcul le résultat sur p bits, plus deux bits supplémentaires : le bit de garde et le bit d'arrondi. Le reste donne la valeur du bit persistant.

Considérons 8/3, c'est à dire : $1,000 \times 2^3 / 1,100 \times 2^1$

$$
\begin{array}{ccccccccc|cccccc}
1, & 0 & 0 & 0 & & & & & & 1, & 1 & 0 & 0 & & \\
\cline{10-15}
1, & 0 & 0 & 0 & 0 & & & & & 0, & 1 & 0 & 1 & 0 & 1 \\
- & 1 & 1 & 0 & 0 & & & & & & & & & g & a \\
\cline{2-5}
 & & 1 & 0 & 0 & 0 & 0 & & & & & & & & \\
 & & - & 1 & 1 & 0 & 0 & & & & & & & & \\
\cline{4-7}
 & & & & 1 & 0 & 0 & 0 & 0 & & & & & & \\
 & & & & - & 1 & 1 & 0 & 0 & & & & & & \\
\cline{6-9}
 & & & & & & 1 & 0 & 0 & & & & & & \\
 & & & & & & p & p & p & & & & & &
\end{array}
$$

où g désigne le bit de garde, a le bit d'arrondi et p le bit persistant. Pour l'instant, le résultat est $0,101 \times 2^2$. La normalisation donne $1,010 \times 2^1$, c'est à dire 2,5, et l'arrondi $1,011 \times 2^1$ soit 2,75 (le résultat correct est 2,6666...).

4.3 Circuits arithmétiques

Cette section présente quelques circuits typiques de mise en oeuvre de opérations arithmétiques décrites ci-dessus.

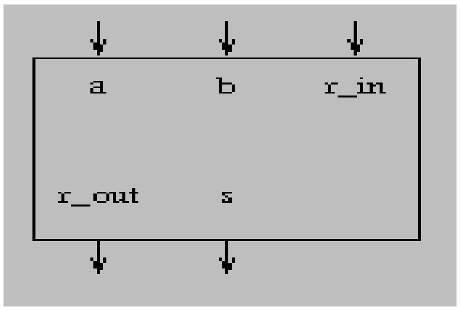
4.3.1 Addition

La conception d'un additionneur n bits (appelé full adder ou additionneur complet) demande de prendre en compte systématiquement la retenue (carry) entrant dans un étage d'addition et de générer un bit de résultat et un bit de retenue. Ainsi la table de vérité de l'addition devient :

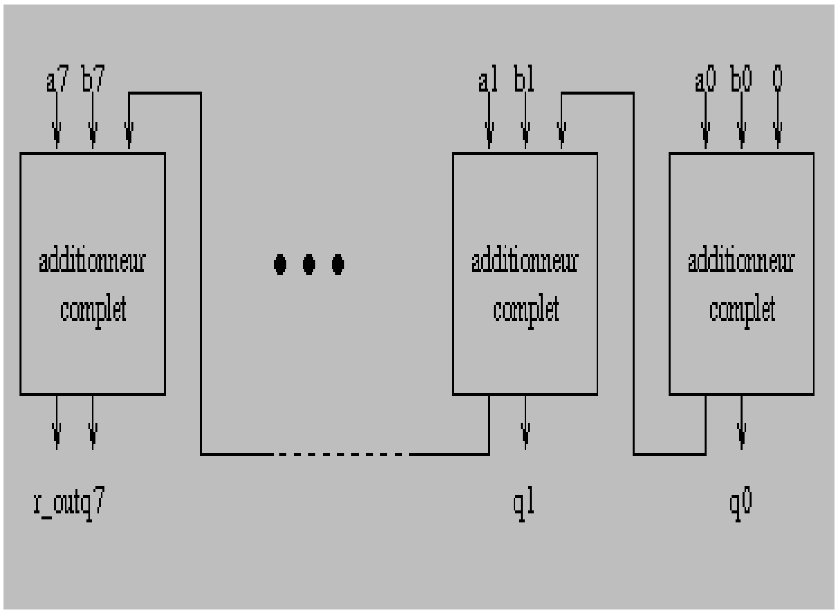
r_{in}	a	b	somme	r_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

? *Le circuit implantant la table de vérité de l'addition sans retenue entrante est appelé demi-additionneur. Construire un additionneur complet à partir de demi-additionneurs.*

Suivent quelques exemples d'additionneurs. L'amélioration d'un additionneur (en terme de temps de calcul) porte sur la propagation de la retenue.



Additionneur à propagation simple de retenue (CPA : carry propagate adder) Un additionneur n bits est réalisé en utilisant n additionneurs complets, chaque retenue sortant de l'étage d'addition i servant de retenue entrante dans l'étage $i+1$.
La retenue entrante au premier étage est positionnée à 0. La retenue sortante du dernier étage pourra servir d'indicateur de débordement.



? *Faire la synthèse logique d'un additionneur complet sur 1 bit le plus rapide possible en utilisant des portes NOT, AND, OR, NAND, NOR. Déterminer le temps maximum pour effectuer une addition en supposant que le temps de* *traversée d'une porte NOT, NAND ou NOR est de τ et le temps de traversée d'une porte AND ou OR est de 2τ.*

! *L'inconvénient majeur de ce circuit est le temps nécessaire à la propagation de la retenue, qui est proportionnel au nombre d'étage. La complexité en temps d'un additionneur n bits est donc en $O(n)$.*

Additionneur à retenue anticipée (CLA : carry look-ahead adder) Le principe de cet additionneur est, pour chaque étage d'addition, d'anticiper (to look ahead) la retenue avant de calculer la somme.
Le fait d'obtenir une retenue sortante à un étage d'addition est due à

- une *génération* de retenue à cet étage, ou
- une *propagation* de retenue par cet étage

Ce qui se retrouve dans l'expression de la retenue pour un étage i :

$$r_{i+1} = (a_i \wedge b_i) \vee (a_i \wedge r_i) \vee (b_i \wedge r_i) = (a_i \wedge b_i) \vee (a_i \vee b_i) \wedge r_i$$

Ainsi

- $g_i = (a_i \wedge b_i)$ est appelé *générateur* de retenue et
- $p_i = (a_i \vee b_i)$ est appelé *propagateur* de retenue.

Donc

$$r_{i+1} = g_i \vee (p_i \wedge r_i)$$

ce qui, si on développe les r_i, donne :

$$r_{i+1} = g_i \vee (p_i \wedge (g_{i-1} \vee (p_{i-1} \wedge (\ldots g_0 \vee (p_0 \wedge r_0) \ldots)$$

En développant encore :

$$\begin{aligned} r_{i+1} = g_i \quad & \vee \quad (p_i \wedge g_{i-1}) \\ & \vee \quad (p_i \wedge p_{i-1} \wedge g_{i-2}) \\ & \vee \quad \ldots \\ & \vee \quad (p_i \wedge p_{i-1} \wedge \ldots \wedge p_1 \wedge g_0) \\ & \vee \quad (p_i \wedge p_{i-1} \wedge \ldots \wedge p_1 \wedge p_0 \wedge r_0) \end{aligned}$$

Ceci permet de calculer chaque retenue en fonction de r_0 et des a_i, b_i des étages précédents.

Mise en oeuvre de l'additionneur à anticipation de retenue Un moyen d'implanter l'additionneur obtenu est de réaliser un circuit calculant la propagation et la génération de retenue pour un étage en fonction de la propagation et la génération des étages précédents.
Il y a propagation à l'étage j si dans les étages i à j il y a toujours eu propagation :

$$P_{j,i} = p_j \wedge p_{j-1} \dots p_{i+1} \wedge p_i$$

Il y a génération à l'étage j si c'est l'étage j lui même qui génère, ou si les étages d'avant ont tous généré et propagé :

$$G_{j,i} = g_j \vee p_j \wedge g_{j-1} \vee p_j \wedge p_{j-1} \wedge g_{j-2} \vee \dots \vee p_j \wedge p_{j-1} \dots p_{i+1} \wedge g_i$$

Donc la retenue à l'étage j peut s'exprimer en fonction de la retenue à l'étage $i, i < j$:

$$r_{j+1} = G_{j,i} \vee P_{j,i} \wedge r_i$$

Utilisons un circuit pouvant générer les signaux $P_{1,0}, G_{1,0}$ et les retenues r_1, r_2 en fonction des couples (p_0, g_0), (p_1, g_1) et de la retenue entrante r_0.
Pour un tel circuit :

$$G_{1,0} = g_1 \vee p_1 \wedge g_0$$

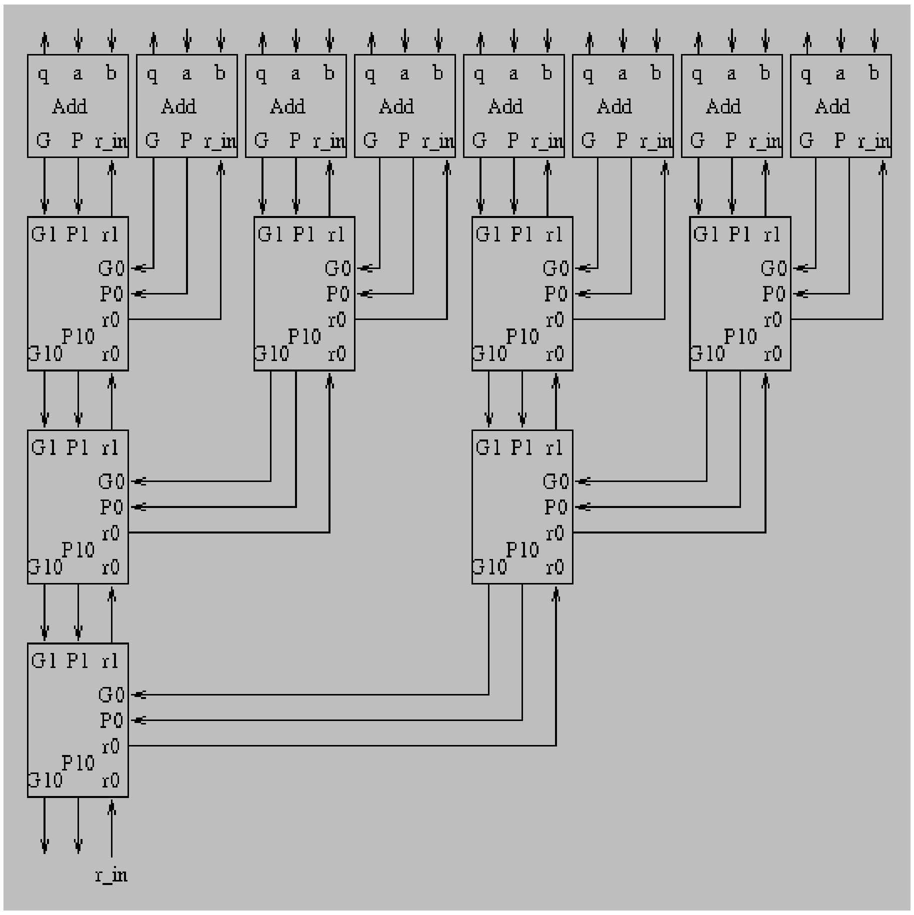
$$P_{1,0} = p_1 \wedge p_0$$

$$r_1 = g_0 \vee p_0 \wedge r_0$$

Supposons qu'un additionneur un bit nous calcule aussi la propagation et la génération. Il est possible d'organiser un générateur de retenue en arbre, qui permet d'anticiper chaque retenue avant d'additionner.

! *Un additionneur à anticipation de retenue sur n bits fonctionne en log(n) étapes pour le calcul des n retenues + 1 étape pour l'addition finale.*

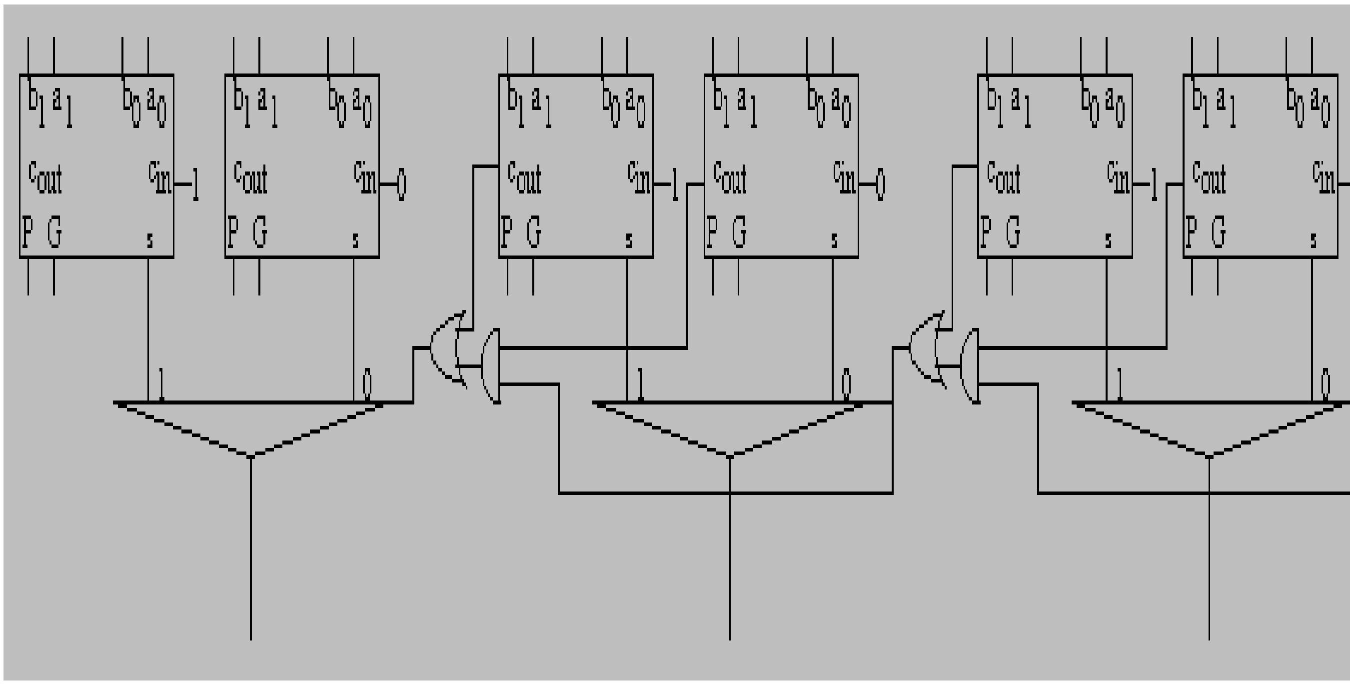
? *Faire fonctionner cet additionneur sur 00101011 + 10110111*



Additionneur à saut de retenue (carry-skip adder) Considérons un étage d'addition : s'il y a propagation de retenue, alors la retenue sortante est la retenue entrante, sinon la retenue sortante est la retenue générée.
L'intérêt de cette constatation est que pour un additionneur n bits, le signal de propagation est plus simple (rapide) à calculer que la propagation elle même.

? *Comment réaliser un tel additionneur ?*

Additionneur à sélection de retenue (carry select adder) Un tel additionneur est réalisé avec deux additionneurs fonctionnant en parallèle, l'un avec une retenue entrante à 0, l'autre avec une retenue entrante à 1. Dés réception de la retenue entrante effective, une simple sélection permet d'obtenir le résultat.



! *L'addition est donc réalisable très rapidement.*

4.3.2 Multiplication

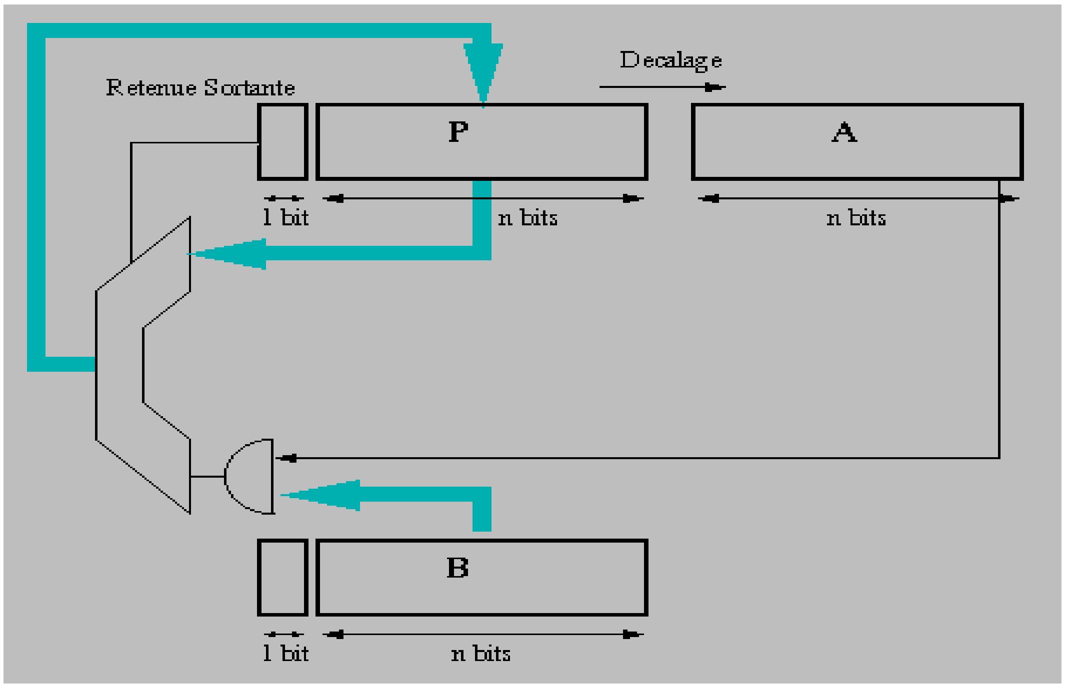
Le principe de la multiplication est la génération de produits partiels et leur addition. Ainsi, l'amélioration d'un multiplieur passe par ces deux aspects : notamment, elle consiste en la réduction du nombre de produits partiels (donc d'additions), ou l'accélération de l'addition des produits partiels.

Multiplication séquentielle Supposons que notre ALU contienne des registres de n bits. Un multiplieur séquentiel opère sur 2 nombres non signés, $A = a_{n-1} \ldots a_1 a_0$ et $B = b_{n-1} \ldots b_1 b_0$. Ces deux nombres sont contenus dans deux registres A et B, et on utilise un registre supplémentaire P. L'algorithme de calcul de $A \times B$ est le suivant :

1. $P \leftarrow 0$
2. faire n fois
   (a) si le chiffre de poid faible de A est 1 alors
      - $P \leftarrow P + B$,
      - sinon $P \leftarrow P + 0$

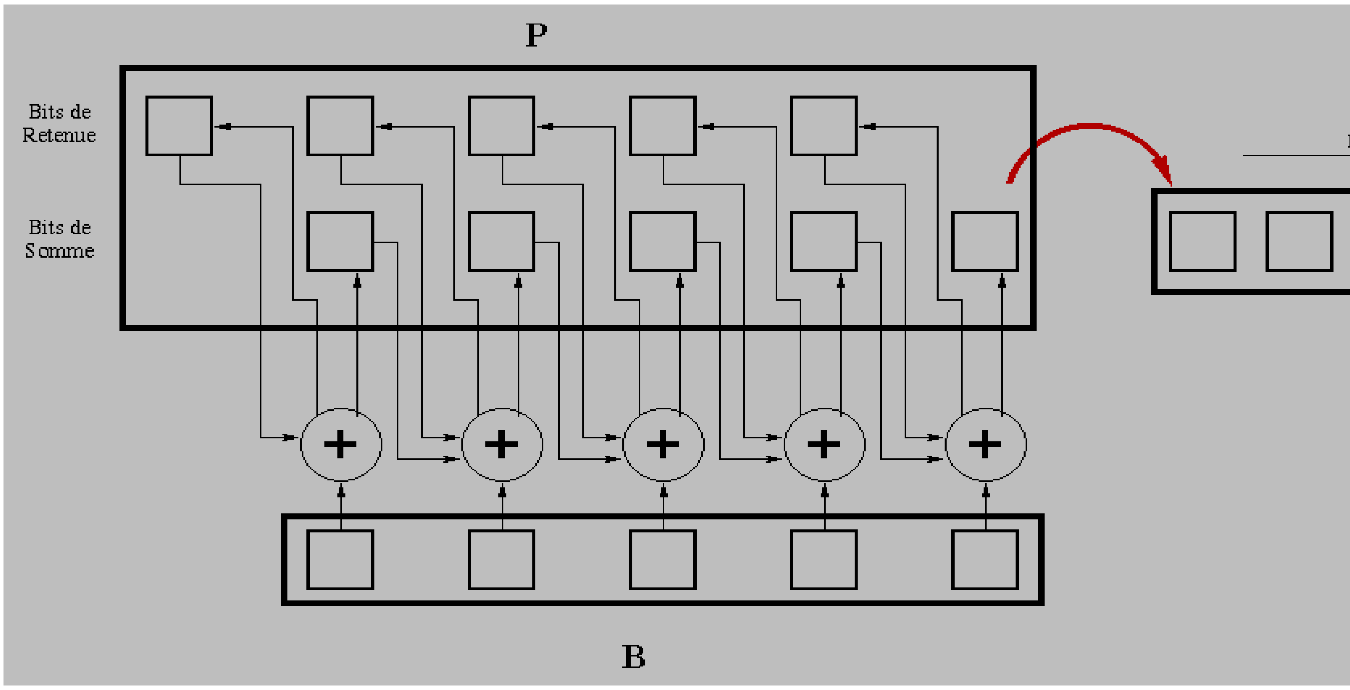
(b) décalage des registres P et A vers la droite : le bit de poids fort de P reçoit la retenue sortante, le bit de poids faible de P est transféré dans le bit de poids fort de A. L'ancien bit de poids faible de A est perdu.

Le résultat se trouve dans P (les bits de poids fort) et dans A (les bits de poids fables).



! *n additions sont nécessaires*

Multiplication à conservation de retenue L'idée de ce multiplieur est d'associer un mécanisme de propagation de retenue au mécanisme de décalage du registre P. Ainsi la retenue n'est plus propagée sur chaque étage d'une addition, mais est conservée pour l'addition suivante.



Recodage de Booth Le but de ce recodage est de réduire le nombre de produits partiels. Il est basé sur la constatation suivante : $2^{i+j} - 2^i = 2^{i+j-1} + 2^{i+j-2} + \ldots + 2^i$ Ainsi lors d'une multiplication, lorsqu'une séquence de j 1 apparait dans le multiplicande (du bit $i+j-1$ au bit i), on peut *remplacer* les j additions par :

- une addition pour le rang $i+j$ et
- une soustraction pour le rang i.

Par exemple le nombre 000111110011100 se recode en 0010000$\overline{1}$0100$\overline{1}$00, où $\overline{1}$ indique une soustraction et 1 indique une addition.

? *Multiplier 3 par 15 en employant le recodage de Booth.*

! *Recodage non lié à une implantation particulière.*

Multiplieur à base plus élevée (radix-4 modified Booth recoding) Une multiplication à base plus élevée (sous entendu que 2) considére plusieurs bits simultanément (au lieu de 1 habituellement). Ainsi, un multiplieur à base 2^k considère k bits simultanément.

! *Le nombre de produits partiels est divisé par k.*

Considérons un multiplieur à base 4. On considère les bits deux par deux, en s'appuyant sur le bit précédent la paire courante. De plus, on emploi un recodage de Booth. Pour un nombre A, on a

$$A = \sum_{i=0}^{n/2} (a_{2i-1} + a_{2i} - 2a_{2i+1}) 2^{2i}$$

avec $a_{-1} = 0$ (par exemple, $3 = 2 \times 2^1 - 1 \times 2^0$)

! *Une multiplication par la base = un décalage vers la gauche.*

L'opération à faire sur le résultat intermédiaire est donnée par la table suivante (en fonction des bits du multiplicande A) :

a_{2i+1}	a_{2i}	a_{2i-1}	résultat intermédiaire
0	0	0	0
0	0	1	+B
0	1	0	+B
0	1	1	+2B
1	0	0	-2B
1	0	1	-B
1	1	0	-B
1	1	1	0

Exemple : la multiplication de 7 ($A = 00111$) par 3 ($B = 00011$) se fait comme suit :

$$\begin{array}{cccccccccccc}
 & & & & & 0 & 0 & 1 & 1 & 1 & \\
 & & & & \times & 0 & 0 & 0 & 1 & 1 & \\
\hline
 & & & & - & 0 & 0 & 0 & 1 & 1 & \text{(-B)} \\
 & + & 0 & 0 & 0 & 1 & 1 & & & & \text{(+2B)} \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 &
\end{array}$$

? *Donner la table permettant de déterminer le multiplicande intermédiaire à ajouter pour une multiplication en base 8.*

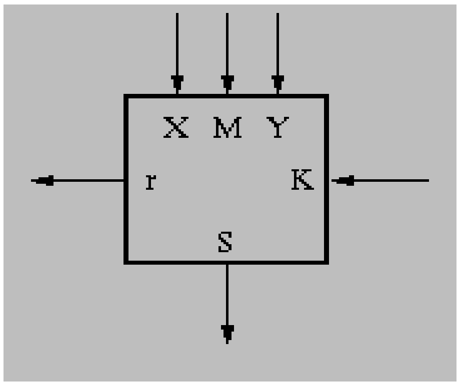
Multiplication en réseau (parallel multiplier) L'idée est d'utiliser un circuit pour chaque élément $x_i y_j$ d'un produit partiel :

$$\begin{array}{cccccccc}
 & & & & x_3 & x_2 & x_1 & x_0 \\
 & & & & y_3 & y_2 & y_1 & y_0 \\
\cline{5-8}
 & & & & x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 \\
 & & & x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 & \\
 & & x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 & & \\
 & x_3y_3 & x_2y_3 & x_1y_3 & x_0y_3 & & & \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}$$

Chaque élément $x_i y_j$:

- multiplie x_i par y_j $(x_i \wedge y_j)$
- additionne la somme partielle provenant de l'élément $x_{i+1}y_{j-1}$
- tient compte de la retenue provenant de l'élément $x_{i-1}y_j$
- envoie la retenue à l'élément $x_{i+1}y_j$
- envoie la somme à l'élément $x_i y_{j+1}$

Le circuit utilisé est le suivant, où $(r, S) = X \times Y + K + M$ (S est la somme, et r est la retenue sortante).



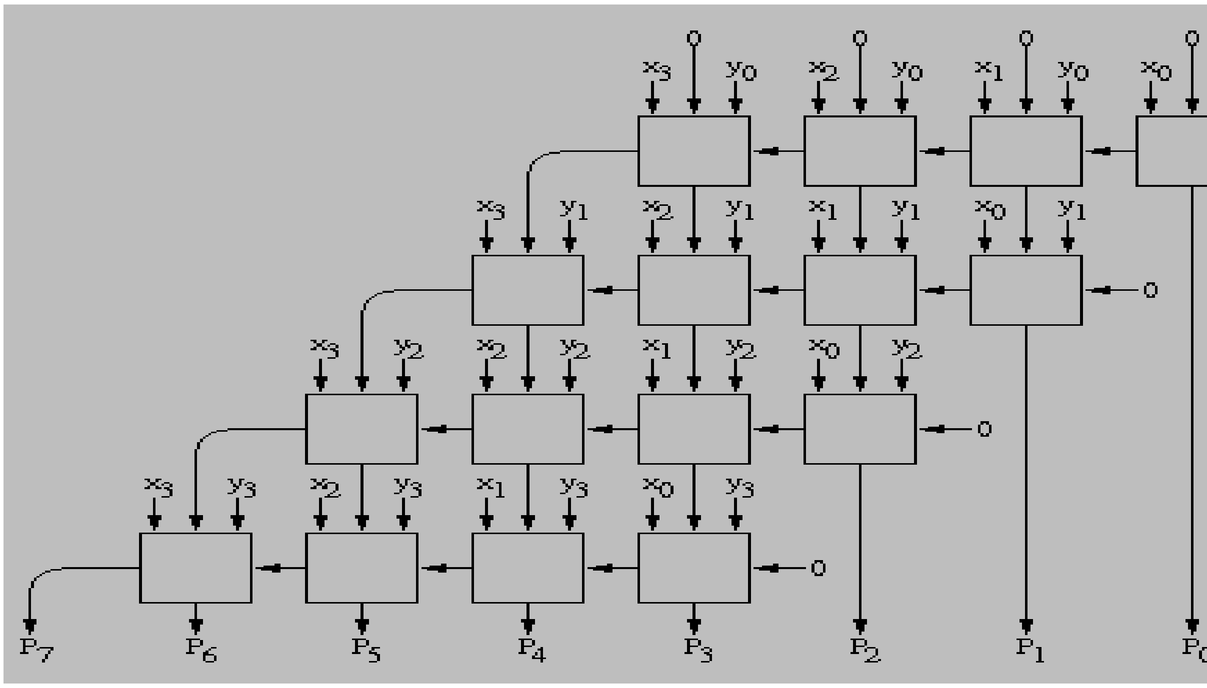
Ce qui permet d'organiser un multiplicateur réseau comme suit.

4.3.3 Division

L'amélioration des diviseurs porte sur la réduction du nombre d'opérations (addition, soustraction) intermédiaires nécessité par l'opération de division.
Dans tous les diviseurs présentés ci-dessous, les opérandes sont 2 nombres non signés de n bits, contenus dans deux registres : le registre A contient le dividende, et le registre B contient le diviseur. On utilise un registre supplémentaire P. En fin de division, le registre A contient le quotient et le registre P contient le reste.

Division avec restauration L'algorithme de division avec restauration est le suivant (dans ce qui suit, $p < 0$ s'interprète comme “P contient un nombre négatif”) :

1. $P \leftarrow 0$
2. faire n fois
   (a) décalage des registres P, A d'un bit vers la gauche : le bit de poids fort de A est injecté dans le bit de poids faible de P



(b) $P \leftarrow P - B$

(c) si $P < 0$ alors

le bit de poids faible de A est mis à 0,

$P \leftarrow P + B$ (étape de restauration)

sinon

le bit de poids faible de A est mis à 1

Division sans restauration L'étape de restauration peut être évitée en changeant la phase itérative par :

1. si $P < 0$

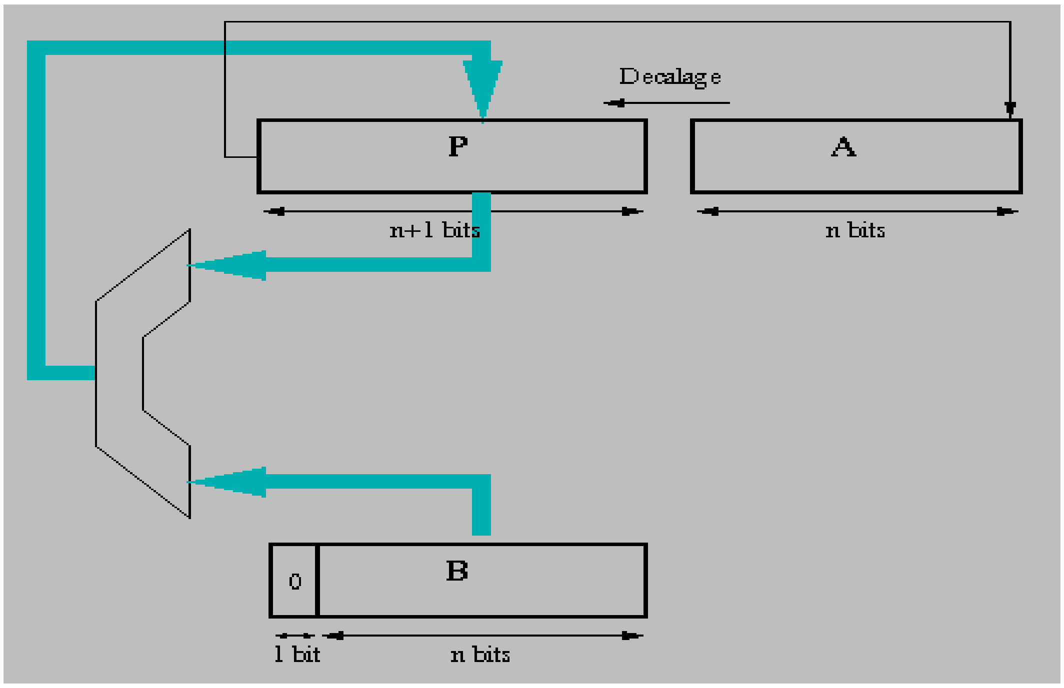
   décalage des registres P, A d'un bit vers la gauche : le bit de poids fort de A est injecté dans le bit de poids faible de P

   $P \leftarrow P + B$

   sinon

   décalage des registres P, A d'un bit vers la gauche : le bit de poids fort de A est injecté dans le bit de poids faible de P

   $P \leftarrow P - B$



2. si $P < 0$

   le bit de poids faible de A est mis à 0

   sinon

   le bit de poids faible de A est mis à 1

Une restauration finale est toutefois nécessaire si $P < 0$ en fin de division.

Division SRT A chaque étape de la division sans restauration, une addition ou une soustraction est réalisée. Un algorithme a été proposé indépendament par Sweeney, Robertson, Tocher pour éviter une ou plusieurs de ces additions dans certains cas.
Dans ce qui suit, les q_i sont les bits du quotient, injectés dans les bits de poids faible de A.

1. si il y a k 0 dans les bits de poids fort de B, on décale de k-1 positions vers la gauche les registres P,A et B.
2. pour i de 0 à n-1
   (a) si les trois bits de poids fort de P sont égaux

      décaler P,A d'une position à gauche : le bit de poids fort de A est injecté dans le bit de poids faible de P

      $q_i \leftarrow 0$

sinon

si $P < 0$

- décaler P,A d'une position vers la gauche : le bit de poids fort de A est injecté dans le bit de poids faible de P
- $q_i \leftarrow \overline{1}$
- $P \leftarrow P + B$

sinon

- décaler P,A d'une position vers la gauche : le bit de poids fort de A est injecté dans le bit de poids faible de P
- $q_i \leftarrow 1$
- $P \leftarrow P - B$

3. si $P < 0$ (le reste final est négatif)

   $P \leftarrow P + B$

   $A \leftarrow A - 1$
4. décaler le reste de k-1 positions vers la droite

? *Tester cet algorithme sur 8/3*

Chapitre 5

Mémoire

5.1 Introduction

Ce cours introduit la stratégie mémoire mise en oeuvre dans un ordinateur, et les caractéristiques associées à cette stratégie en terme d'organisation et de performance.

! *A ce jour, il n'existe pas de technologie optimale pour satisfaire le besoin en mémoire d'un ordinateur.*

On commence par aborder les caractéristiques d'une mémoire, puis on détaille les différents mode d'accès. Le cours se focalise ensuite sur la hiérarchie mémoire interne, c'est à dire la mémoire principale et la mémoire cache. L'exemple du pentium II est rapidement présenté.

5.2 Les caractéristiques

Dans ce chapitre, on appellera *mémoire* tout dispositif (électronique) capable de *conserver* et *restituer* une information.
On parlera de *mot mémoire* (ou plus simplement de *mot*) pour désigner l'ensemble de bits pouvant être lus ou écrits simultanément.

Les différents types physiques de mémoires Les différents supports utilisés sont principalement

- semi-conducteur (e.g., registre)
- magnétique (e.g., disquette)
- optique (e.g., cd-rom)

Durée de mémorisation Elle peut être fonction du temps (de quasi-permanente, e.g., disque, ROM à temporaire, ex. mémoire dynamique), ou fonction de la présence d'alimentation électrique (volatilité : e.g., RAM).

Emplacement Il correspond à la localisation de la mémoire dans la machine :

- dans le processeur (registre)
- interne (mémoire principale)
- externe (aussi appelée mémoire secondaire).

Capacité Elle Représente le nombre d'informations stockable. Exprimée en octet (byte) ou en mot (word) de 8, 16 ou 32 bits. On utilise des puissance de deux, avec les unités suivantes :

- kilo 1K = 2^{10} = 1024
- méga 1M = 2^{20} = 1 048 576
- giga 1G = 2^{30} = 1 073 741 824
- téra 1T = 2^{40} = 1 099 511 627 776
- péta 1P = 2^{50} = 1 125 899 906 842 620

Performance On considère principalement les informations suivantes :

- le *temps d'accès* : le temps nécessaire à une opération de lecture/écriture i.e., le temps qui sépare l'instant où l'opération est demandée (exemple : l'adresse est présentée à la mémoire pour une opération de lecture) du l'instant où l'opération est terminée (l'information est disponible),
- le *débit* : la quantité d'informations lues/écrites par unités de temps (exemple : Mo/s).

Mode d'accès Il s'agit de la manière de retrouver une information, d'accéder à un mot mémoire (cf section 5.3).

Hiérarchie L'idéal est de posséder une mémoire illimitée et très rapide. Or le temps d'accès augmente avec la capacité. L'idée adoptée pour l'organisation de la mémoire est donc de considérer que seules les données les plus utilisées nécessitent un temps d'accès très petit.
Ainsi la mémoire est organisée en une hiérarchie :

- du plus au moins rapide, c'est à dire
- de la capacité la plus faible à la capacité la plus grande, c'est à dire
- du composant le plus couteux au composant le moins couteux !

5.3 Mode d'accès

Le mode d'accès à une mémoire dépend surtout de l'utilsation qu'on veut en faire.

Accès aléatoire Il s'agit du mode d'accès le plus employé. Il est utilisé par

- les mémoires qui composent la mémoire principale,
- quelques mémoires caches.

A chaque mot mémoire est associée une adresse unique (pas d'ambigüité dans la désignation du mot recherché). Le fonctionnement est celui présentée déjà au chapitre introduction. N'importe quelle adresse peut être traitée (d'où le nom).
La taille d'une adresse dépend de la capacité de la mémoire. Par exemple, pour une mémoire de 4 Gbits il faut au moins une adresse de 32 bits : 1 gigabits = 2^{30} = 1 073 741 824 bits, 4Go = $4 \times 2^{30} = 2^2 \times 2^{30}$.
Les opérations associées à ce mode d'accès :

- lecture(adr),
- écriture(adr,donnée)

Le temps d'accès est constant (il est indépendant des accès précédents).

Accès par le contenu Ce mode d'accès caractérise les mémoires appelées mémoires associatives.
Il est employé principalement par les mémoire caches.

Le principe est similaire à la mémoire à accès aléatoire sans notion d'adresse : un mot est retrouvé par une partie de son contenu.
En général, une mémoire associative est divisée en 2 parties :

1. une partie contenant un descripteur (clé) et permettant une comparaison en parallèle de ce descripteur avec un autre descripteur, et
2. une deuxième partie fournissant le mot associé au descripteur.

Les opérations associées à ce mode d'accès :

- écriture(clé,donnée)
- lecture(clé)
- existe(clé)
- retirer(clé)

Le temps d'accès est constant.

Accès séquentiel Employé pour l'archivage d'importants volumes de données (par exemple sur bandes magnétiques).
Les diverses informations sont écrites les une derrière les autres : pour accéder à une donnée, il faut avoir lu les précédentes.
Les opérations associées à ce mode d'accès :

- début : se positionner sur la première donnée
- lecture : lire une donnée
- écriture(données) : écrire donnée
- fin : se positionner après la dernière donnée

Le temps d'accès est variable.

Accès direct Employé pour les disques (durs ou souple).
Chaque bloc de données a une adresse unique. Une donnée est accédée en accédant le bloc qui la contient, puis en se déplaçant dans le bloc jusqu'à sa position.
Les opérations associées à ce mode d'accès :

- lecture(bloc,déplacement)
- écriture(bloc,déplacement,donnée)

Le temps d'accès est variable.

5.4 Mémoire principale

! *L'accès à la mémoire principale est le chemin le plus important dans l'ordinateur.*

Types Les mémoires composant la mémoire principale sont des mémoires à base de semi-conducteurs, employant un mode d'accès aléatoire. Elles sont de deux types : volatiles ou non.
Le terme RAM correspond aux mémoires volatiles. Elles stockent des données temporaires. Actuellement on en trouve principalement 2 types :
- RAM dynamique (DRAM) : des condensateurs sont utilisés comme unités de mémorisation. Elles nécessitent un rafraichissement périodique. Elles sont simples, denses, peu couteuses.
- RAM statique : des bascules sont utilisées comme unités de mémorisation. Elles sont plus rapides, et ne nécessitent pas de rafaichissement.

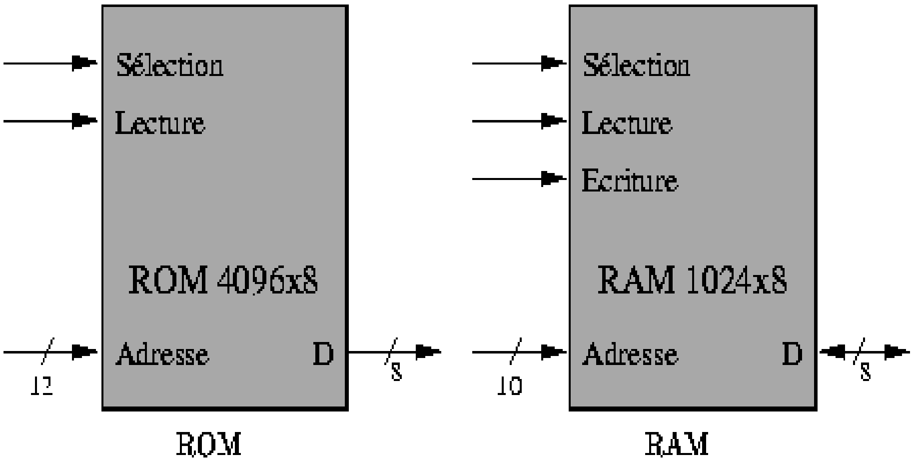
! *Le circuit DRAM demeure (depuis plus de 20 ans) la brique de base de la mémoire principale.*

Les ROM (Read-Only, Read-Mostly Memory) sont utilisées pour stocker des informations permanentes (programmes systèmes, microprogrammation, cf plus tard). On en trouve de plusieurs types, selon comment sont faites/possibles la lecture/écriture
- ROM : écriture unique lors de la fabrication
- PROM : écriture unique après fabrication
- EPROM : admet un nombre d'écriture limité (effaçage par ultra-violet).

Organisation L'élément de base d'une mémoire semi-conducteur est appelé cellule. Une cellule possède 3 connexions :
- une entrée de sélection indiquant si la cellule est concernée par l'opération courante
- une entrée de contrôle indiquant l'opération courante est une lecture (Output Enable) ou une écriture (Write Enable)
- une ligne bidirectionnelle pour les données.

A partir de cet élément de base, réalisons un circuit mémoire RAM de M mots de B bits chacun. On peut organiser cette mémoire en une matrice de M lignes et B colonnes. Ainsi, outre les entrées de sélection, lecture et écriture (toutes sur 1 bit), on doit disposer de $log_2(M)$ lignes d'adresse (par exemple, $10 = log_2(1024)$) et de B lignes de données.

Supposons maintenant qu'à partir de circuits RAM de 1024×8 bits et de circuits ROM de 4096×8 bits, on cherche à réaliser un espace adressable de 2^{16} mots de 32 bits, et disposer de 4096 mots mémoires en RAM et 4096 mots mémoire en ROM. Pour obtenir un mot, il faut en parallèle 4 circuits de 8 bits.
On définit une carte d'adresse mémoire, c'est à dire l'occupation effective de l'espace adressable :
- les 4096 mots mémoires RAM correspondront aux adresses les plus basses (de 0 à 4096),



- les 4096 mots de mémoires ROM correspondront aux adresses les plus élevées (de 61440 à 65535),
- les autres adresses resteront inoccupées.

étage	bits d'adresse
RAM 0	0000 00xx xxxx xxxx
RAM 1	0000 01xx xxxx xxxx
RAM 2	0000 10xx xxxx xxxx
RAM 3	0000 11xx xxxx xxxx
inoccupé	
ROM 4	1111 xxxx xxxx xxxx

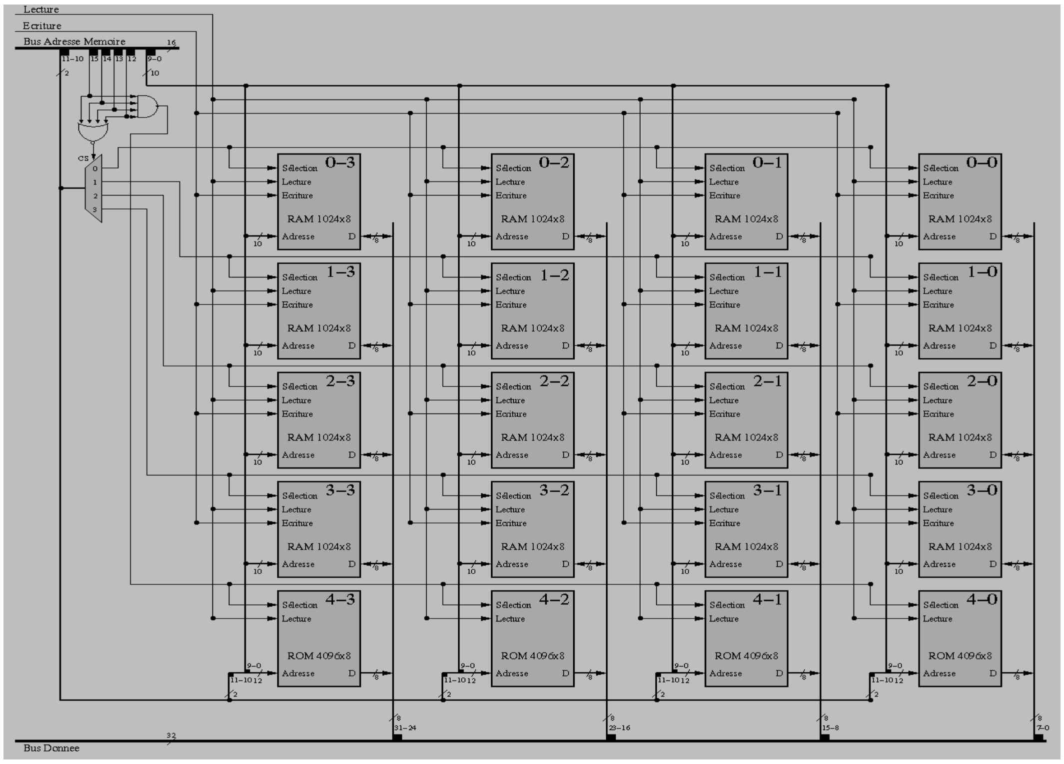
Cette mémoire possède les particularités suivantes :

- elle est adressable au mot : deux valeurs consécutives d'adresse désignent deux mots consécutifs,
- l'organisation physique mots par mot reflète l'organisation logique.

Il faut choisir une convention pour l'adresse de l'octet de poids faible dans le mot :

- adresse la plus faible du mot : convention petit bout (little endian, employé par la famille i386),
- adresse la plus élevée du mot : convention gros bout (big endian, employé par la famille 68000)

Mémoire dynamique DRAM Du fait de leur grande capacité et de leur structure interne, les circuits de mémoire dynamique multiplexent le bus adresse : l'adresse de n bits d'un mot est envoyé en 2 fois $n/2$ bis. Deux signaux indiquent si la demi adresse courante est en fait les bits de poids faible ou de poits fort de

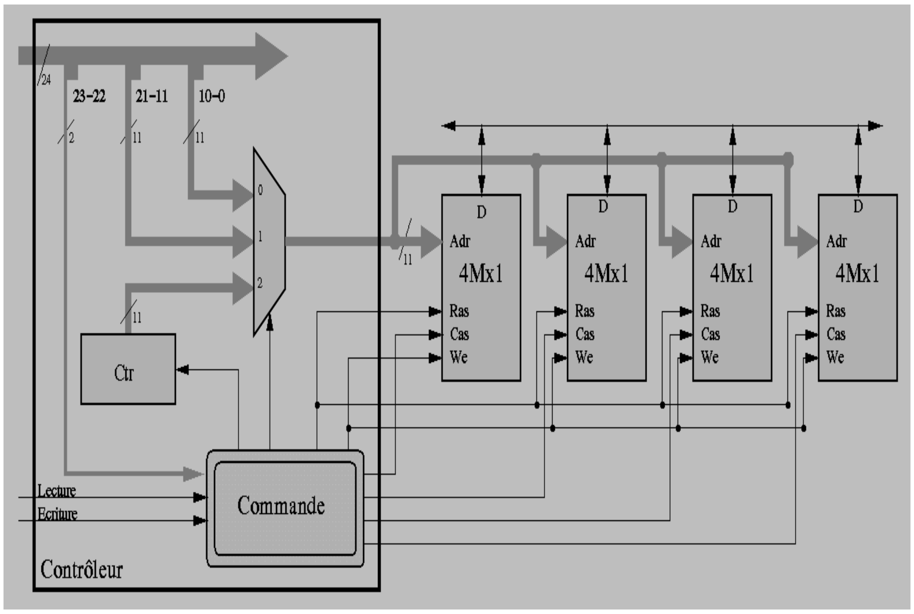


l'adresse. Ces signaux sont nommés RAS (Row Adress Select) et CAS (Column Adress Select).
Ainsi pour réaliser une mémoire de 16 M mots de 1 bit à partir de circuits de 4 M mots de 1 bit, on assemble 4 circuits différenciés par le signal CAS.
Ce type de mémoire nécessite un compteur mémorisant le numéro de ligne devant subir un cycle de rafraichissement (refresh counter).
Un contrôleur de mémoire dynamique est un circuit intégré qui regroupe :

- la partie compteur
- la partie multiplexage d'adresses
- la partie génération des signaux CAS et RAS
- d'autres fonctionalités (contrôle d'accès au bus de données, cf plus tard...)

5.5 Mémoire cache

! *La performance des microprocesseurs augmente de environ 55% par an depuis 1987. La performance des mémoire augmente de environ 7% par an.*



Le cache mémoire est un niveau de mémorisation intermédiaire très rapide (plusieurs dizaines de fois que la mémoire principale) et de petite capacité destiné à mémoriser les données ou instructions les plus récemment accédées.
Un cache est typiquement situé entre le processeur et la mémoire, mais peut aussi être situé entre le processeur et un autre cache, etc...

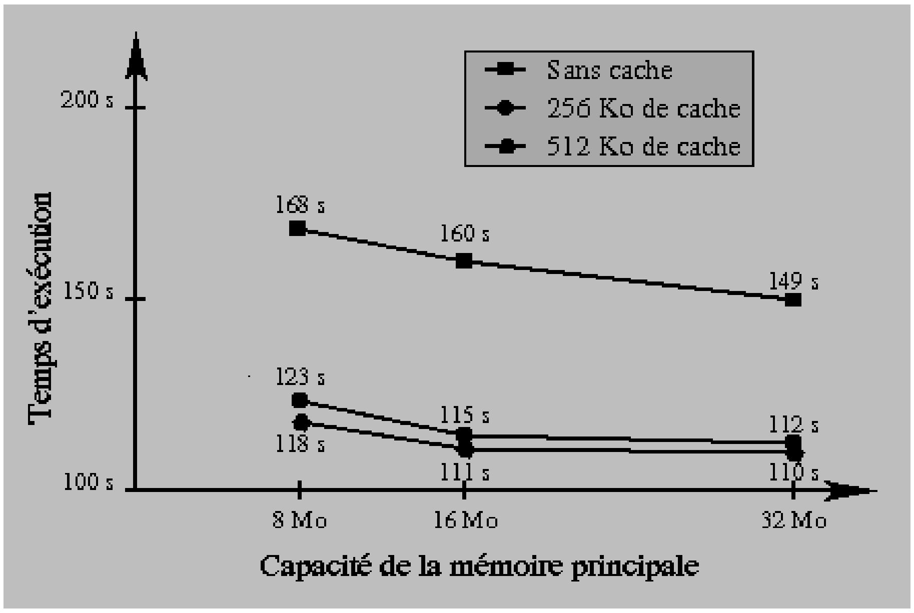
Principe Recherche d'une donnée dans le cache avant de la rechercher dans la mémoire principale :

- succès cache : la donnée est présente dans le cache
- défaut de cache : la donnée est absente du cache

Un *bloc* est un ensemble de mots d'adresses contigües. La mémoire est découpée logiquement en bloc de même taille (e.g., de 4 à 128 octets, 32 octets pour un processeur Alpha AXP 21064). Lors d'un accès à une adresse, on regarde si le bloc contenant l'adresse est dans le cache. En cas de défaut de cache, le bloc entier est copié dans le cache.
Explorons les caractéristiques d'un cache.

Nombre de caches et localisation Actuellement, la norme est à l'utilisation de multiple caches, organisés en niveau (level).
Un cache peut être situé sur la même puce que le processeur (on-chip/internal cache), ou n'être qu'accessible via un bus externe au processeur (external cache).
L'utilisation d'un cache interne permet

- d'augmenter les performances
- de laisser le bus externe disponible



Une organisation typique est : un cache interne (de niveau 1) et un cache externe (de niveau 2). Le cache de niveau 2 doit être de 10 à 100 fois plus grand que le/les caches de niveau 1 pour être intéressant.

Contenu Avec l'utilisation de multiples caches a surgi l'idée de dédier un cache au stockage des données et un cache au stockage des instructions. L'intérêt réside dans la possibilité pour le processeur de dissocier le mécanisme d'exécution des instructions d'avec celui de recherche et décodage des instructions.

Taille du cache Il doit être suffisament petit pour que son coût soit proche de celui d'une mémoire principale, et que le temps d'accès soit le plus intéressant possible, et suffisament grand pour ne pas avoir à trop accéder à la mémoire principale. Des études ont montré que les caches les plus efficaces on une taille inférieure à 512 K mots, mais leur performance dépendent beaucoup de la nature des applications traitées par la machine...

Correspondance La taille du cache est beaucoup plus petite que la taille de la mémoire. Il faut définir une stratégie de copie des blocs de données dans le cache. 3 stratégies sont possibles :

- correspondance directe : chaque bloc mémoire ne peut être placé que dans un seul bloc du cache,
- correspondance totalement associative : chaque bloc mémoire peut être placé dans n'importe quel bloc du cache

- correspondance associative par ensemble : chaque bloc mémoire peut être placé dans n'importe quel bloc du cache parmi un ensemble de n blocs.

Aujourd'hui la grande majorité des caches sont à correspondance directe ou à correspondance associative par ensemble de 2 ou 4 bloc.

Accès à un bloc du cache Les adresses mémoires peuvent être construites en fonction de la correspondance entre mémoire principale et cache. Dans ce cas, l'adresse mémoire d'un mot contient des informations sur sa présence dans un bloc et sa présence éventuelle dans le cache. Elle se décompose en deux partie :

- un numéro de bloc, qui se décompose en
  - un index, correspondant à l'emplacement de ce bloc dans le cache
  - une étiquette permetant d'identifier le bloc mémoire correspondant au bloc placé dans le cache
- un déplacement dans le bloc (le numéro du mot dans la bloc).

Ainsi, une table d'étiquette est maintenue, qui donne pour chaque bloc du cache l'étiquette du bloc mémoire placé dans ce bloc, ou le fait qu'aucun bloc mémoire n'a été copié dans ce bloc.

Algorithme de remplacement Il existe plusieurs manières de déterminer quel bloc du cache va recevoir le bloc mémoire ayant provoqué un défaut de cache.

! *Le problème ne se pose que dans le cas de la correspondace associative (totale ou par ensemble).*

Divers stratégies sont employées, principalement :

- choisir un bloc candidat de manière aléatoire
- choisir le plus ancien bloc du cache (FIFO, First In First Out)
- choisir le bloc le moins récemment utilisé (LRU Least Recently Used)
- choisir le bloc le moins fréquemment utilisé (LFU Least Frequently Used)

Les stratégie concernant l'utilisation (LFU, LRU) sont les plus efficace (vient ensuite la stratégie aléatoire). Les stratégies aléatoire et FIFO sont plus faciles à implanter.

Politique d'écriture Considérons le cas d'une opération d'écriture. Deux situations se présentent selon que le bloc dans lequel on souhaite écrire se trouve dans le cache ou non.
Dans le premier cas, on peut choisir

- d'écrire à la fois dans le bloc du cache et dans le bloc de la mémoire (écriture simultanée, ou write through)
- d'écrire uniquement dans le bloc du cache, et différer l'écriture de ce bloc en mémoire lorsque l'emplacement qu'il occupe sera désigné pour recevoir un nouveau bloc mémoire (réécriture ou write back).

? *Lorsqu'on adopte une stratégie write back, que se passe t'il lorsqu'un périphérique demande l'accès à la mémoire ?*

Dans le deuxième cas, on peut choisir

- de charger le bloc de la mémoire dans le cache puis effectuer l'opération d'écriture (écriture allouée)
- d'effectuer l'écriture directement dans la mémoire (écriture non allouée).

Une optimisation classique pour diminuer l'attente de la fin d'une écriture consiste à utiliser un tampon d'écriture, permettant au processeur de continuer à travailler dès que la donnée est écrite dans le tampon, sans attendre l'acquittement de la mémoire.

Performance On peut évaluer la peformance d'une mémoire utilisant un cache par le calcul du temps d'accès mémoire moyen :

temps d'accès mémoire moyen = temps d'accès succès
+ taux d'échec $\times$ pénalité d'échec

temps d'accès succès = temps d'accès à une donnée résidant dans le cache
taux d'échec = nombre de défaut de cache / nombre d'accès cache
Exemple : lors de l'exécution d'une instruction, le processeur prend du temps pour la décoder, accéder aux données en mémoire nécessitées par cette insstruction, et déclencher les opérations sur ces données. Voici le cas suivant :

durée d'un cycle horloge	: τ
pénalité d'échec	: 10 cycles
durée d'une instruction (sans référence mémoire)	: 2 cycles
nombre de références mémoire par instruction	: 1,33
taux d'échec	: 2%
temps d'accès succés	: négligeable

temps d'éxécution moyen d'une instruction = $(2 + 1,33 \times 2\% \times 10\times)\tau = 2,27\tau$
et dans le cas où il n'y a pas de cache, ce temps passe à :
temps d'exécution moyen d'une instruction = $(2 + 1,33 \times 10) = 15,3\tau$

5.6 Exemple : le pentium II

Le pentium II d'Intel emploie un type particulier de mémoire DRAM, la SDRAM, et deux niveau de caches.

SDRAM La SDRAM (Synchronous DRAM) est une évolution de la DRAM. La DRAM est asynchrone ; lorsque le processeur demande une opération de lecture/écriture, il doit attendre pendant le temps correspondant au temps d'accès du circuit que l'opération soit exécutée.
La SDRAM échange ses données avec le processeur en se basant sur un signal d'horloge externe. Elle dispose de registres internes pour stocker les informations à traiter (en lecture ou écriture). Elle répond après un nombre de cycles d'horloge fixé (le processeur peut faire autre chose entre temps). Elle dispose d'un mode

d'accès "en rafale" (burst mode) pour lequel seule l'adresse de début d'une séquence de mots (à lire/écrire) est donnée (ce qui élimine le décodage des adresses suivantes).

Cache Evolution cache de la famille i386 :

processeur	cache
80386	0
80486	1 cache interne 8Ko
pentium	1 cache interne donnée 8 Ko et 1 cache interne instructions 8 Ko
pentium 2	1 cache interne donnée 16 Ko et 1 cache interne instructions 16 Ko
	1 cache externe 256 Ko commun au 2 caches internes
pentium 3	1 cache interne donnée 16 Ko et 1 cache interne instructions 16 Ko
	1 cache externe 256 Ko commun au 2 caches internes

par comparaison :

processeur	cache
INTEL céléron 2	1 cache interne donnée 16 Ko et 1 cache interne instructions 16 Ko
	1 cache externe 128 Ko
AMD athlon	1 cache interne 128 Ko
	1 cache externe 256 Ko

Les caches internes du pentium 2 sont à correspondance associative par ensemble. Le cache interne donnée utilise un algorithme LRU pour le remplacement et une politique d'écriture write back. Il peut être configuré pour adopter une politique write-through.

Chapitre 6

Interconnexions

6.1 Introduction

Ce cours présente la communication entre les différentes unités fonctionnelles de la machine. On s'attarde tout d'abord sur l'arcitecture globale de communication, puis on examine les caractéristiques et le fonctionnement des bus. Le cours se clos sur l'exemple du bus PCI.

6.2 Structure d'interconnexion

Rappel : un ordinateur se compose de trois unités fontionnelles, dédiées aux opérations suivantes :

- la mémoire : lecture, écriture d'un mot de donnée à une adresse
- l'unité d'entrées-sorties (E/S) : comportement similaire à la mémoire. Chaque périphérique a une adresse (appelée numéro de port), et les opérations sont lecture écriture de mots à une adresse
- le processeur (contenant l'ALU) : lecture des instructions et des données, écriture des données, utilise des signaux de contrôle pour contrôler le sysème

Ces 3 unités commmuniquent les unes avec les autres. On appelle *structure d'interconnexion* l'ensemble des chemins les connectant.
La structure d'interconnexion doit pouvoir supporter les types de transfert suivants :

- mémoire vers processeur : lecture d'instructions
- processeur vers mémoire : écriture de données
- E/S vers processeur : lecture de données transmises par un périphériques
- processeur vers E/S : transfert de données vers un périphérique
- E/S vers mémoire ou mémoire vers E/S : utilisation du principe d'accès direct à la mémoire (Direct Memory Access) pour faire communiquer les deux unités sans passer par le processeur.

6.3 Bus

6.3.1 Définition et structure

Définition : un *bus* est un chemin *partagé* entre plusieurs unités (ou modules).

! *Un seul équipement transmet à un instant donné.*

Un bus consiste typiquement de 50 à 100 lignes transmettant des signaux représentant des 1 ou des 0 (généralement des conducteurs). Chaque ligne posséde une fonction propre. On distingue 3 groupes de fonction différentes :

- les lignes de données (appelées bus de données),

- les lignes d'adresses (bus d'adresse),
- les lignes de contrôle, dont la fonction est de controler l'accès et l'utilisation des bus d'adresses et données. Ces signaux sont de deux types :
  - des signaux de timing indiquant la validité des informations d'adresse ou de données,
  - signaux de commandes indiquant le type d'opération à effectuer.

Parmi les signaux de contrôle les plus courants, on trouve :
- memory write : ordre d'écriture de la donnée sur le bus à l'adresse mémoire indiquée,
- memory read : ordre de lecture de la donnée à l'adresse mémoire indiquée,
- I/O write : ordre d'écriture de la donnée sur le bus sur le port indiqué,
- I/O read : ordre de lecture de la donnée sur le port indiqué,
- transfer ACK : indique que les données ont été acceptées ou placées sur le bus,
- bus request : indique qu'un module demande l'accès au bus,
- bus grant : indique qu'un module a obtenu l'accès au bus,
- clock : utilisé pour synchroniser les transferts
- reset : utilisé pour la réinitialisation de tous les modules,
- des signaux gérant les interruptions (cf plus tard).

Fonctionnement schématique Une transaction typique se compose de 3 parties :
- l'obtention du bus,
- l'envoi d'une adresse et
- l'envoi des données.

On parlera de transaction de type écriture quand un module veut transférer des données : une fois le bus obtenu, l'emetteur transmet ses données.
On parlera de transaction de type lecture quand un module veut demander des données : une fois le bus obtenu, l'emetteur transmet une requête au module destination, et attend les données.

Hiérarchie bus multiple Plus le nombre d'équipement connectés à un bus est grand, plus les performances de la machine décroissent (le bus devient un goulot d'étranglement). De plus la longueur physique du bus joue sur la vitesse de transfert de l'information : approximativement la vitesse de signal dans un conducteur correspond à la moitié de la vitesse de la lumière dans le vide (exemple pour une horloge à 200 Mhz, pour un cycle de 5ns, le signal n'a parcouru que 75cm).
Les structures d'interconnexions les plus communes sont basées sur l'utilisation d'une d'une hiérarchie de bus.
Les figures suivantes en donne quelques exemples.

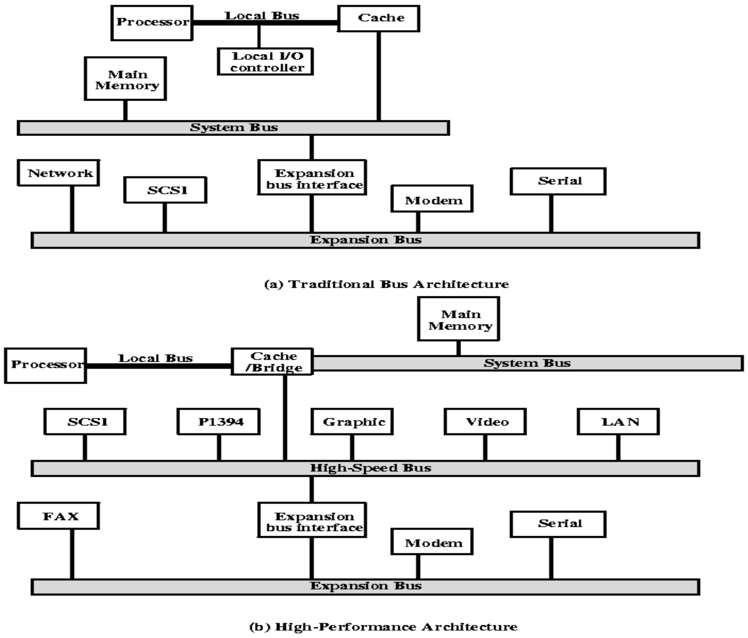


Figure 3.18 Example Bus Configurations

L'intérêt est de séparer communication processeur/mémoire de communication E/S.

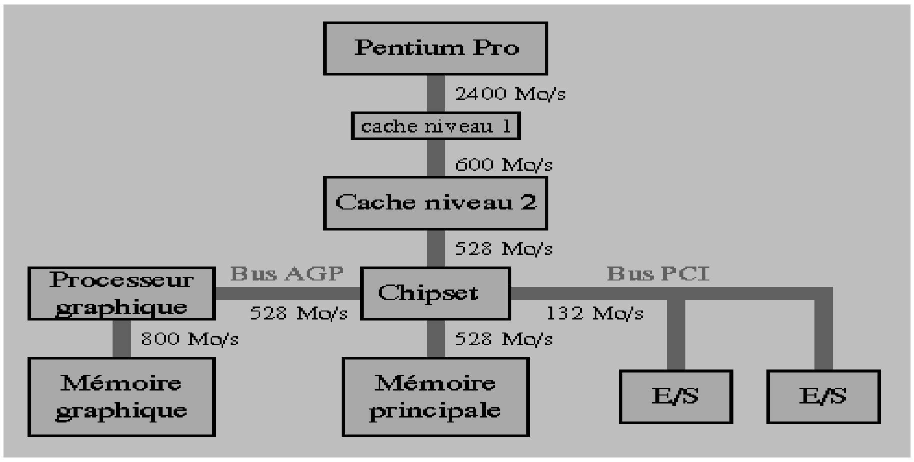
! *De manière générale, on cherchera à étager en fonction de la performance des périphériques.*

On obtient ainsi deux grandes familles de bus :

- les bus UC-mémoire, appelés *bus système*, généralement courts et rapides. Lors de la conception d'un bus système on connait les composants qui utiliseront le bus.
- les bus E/S, plus long, moins rapides, offrent une gamme étendue de débits, et font souvent l'objet de normalisation.

6.3.2 Caractéristiques

Largeur du bus Correspond au nombre d'information pouvant être envoyées en parallèle. Le débit sera d'autant plus grand que la largeur est élevée. Un bus



très large permettra l'envoi simultané de données et adresses. Un bus peu large demandera un multiplexage des donées et adresses.
La largeur du bus de données est déterminante pour les performances du système : si le bus a une largeur de 8 bits et que les instructions sont codées sur 16 bits, le processeur doit faire deux accès mémoire pendant un cycle instruction. Actuellement, la largeur d'un bus de données est de 8, 16 ou 32 bits.
La largeur du bus d'adresses détermine la taille de la capacité mémorielle de l'ordinateur. Typiquement, les bits de poids fort sont utilisées pour sélectionner soit un module d'E/S, soit la mémoire, et les bits de poids faible servent à sélectionner un port ou un emplacement mémoire.

Type de transfert de données Il existe plusieurs moyen de transferer des données :

1. écriture multiplexée : l'adresse est placée sur le bus en premier, ensuite les données,
2. lecture multiplexée : l'adresse est placée sur le bus en premier, ensuite, après un temps d'accès aux données, les données,
3. lecture/écriture non multiplexée : l'adresse et les données sont placées sur le bus en même temps,
4. lecture-modification-écriture : une donnée est lue à une adresse et est immédiatement modifiée (i.e., par une écriture à cette adresse),
5. transfert de bloc de données : une adresse est placée sur le bus, ensuite les données à écrire/lire à l'adresse placée et aux adresses consécutives.

? *Quel est l'inconvénient d'un transfert de type 5 ?*

Synchronisation La transmission peut être synchrone ou asynchrone.
Dans une communication synchrone, le signal d'horloge est transmis sur les lignes de contrôle. Un protocole est fixé en fonction de ce signal pour émettre adresses et données. Cela permet une communication rapide au prix de peu de logique de contrôle. Néanmoins cela impose à l'émetteur et au récepteur de fonctionner à la même fréquence d'horloge.
Dans une communication asynchrone, l'absence de référence à une horloge nécessite des échanges de signaux pour indiquer la progression de la communication. La communication est moins rapide, et est réalisée avec une logique plus importante. L'avantage est de permettre la communnication entre composants hétérogènes (par exemple possédant une fréquence d'horloge différente).

! *En général, le bus système fonctionnera de manière synchrone et un bus d'E/S de manière asynchrone.*

La synchronisation est détaillée dans la section suivante.

Technique d'arbitrage Un module sera appelé *maître* s'il lui est permi de démarrer une transaction sur le bus (exemple : l'UC, un module d'E/S).

! *Il ne peut y avoir qu'un seul maître à la fois.*

Un composant non maître sera dit *esclave*. Lorsqu'il existe plusieurs maîtres de bus potentiels, un mécanisme d'arbitrage est nécessaire pour désigner qui sera le prochain maître (cf section 6.5).
Une transaction faisant intervenir plusieurs esclaves (par exemple adressé à tous les caches pour les rendre cohérents) est appelée diffusion (broadcast).

Performances La performance d'un bus est définie par les critères suivants :
- la bande passante, c'est à dire la quantité d'infirmations échangées par unité de temps,
- la latence, c'est à dire le temps de réponse du bus à une requête de transfert,
- la charge, c'est à dire le nombre maximum d'unités pouvant êtres connectées au bus, et
- la longueur physique du bus.

6.4 Synchronisation des échanges

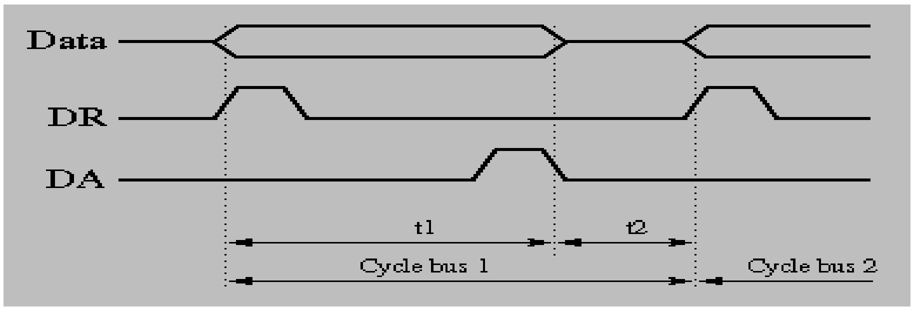
L'échange de données nécessite un timing qui peut être implanté de 3 manières différentes :
- une manière synchrone lorsque les évènements ont lieu à des moments précis dans le temps

- une manière asynchrone lorsque les évènements peuvent avoir lieu à des moements arbitraires
- une manière semi-asynchrone lorsque les évènements peuvent avoir lieu de manière asynchrone lors des différentes phases d'une horloge.

6.4.1 Communication synchrone

Le timing des opérations est contrôlé par une horloge, il n'y a pas de dialogue entre émetteur et récepteur pour le contrôle de l'échange. 2 signaux sont utilisés, qui sont générés par l'horloge :

- DR (Data Ready) : utilisé par l'émetteur, il signal que les données sont placées sur le bus
- DA (Data Accepted/Acknowledge) : utilisé par le récepteur, il signal que les données ont été reçues.



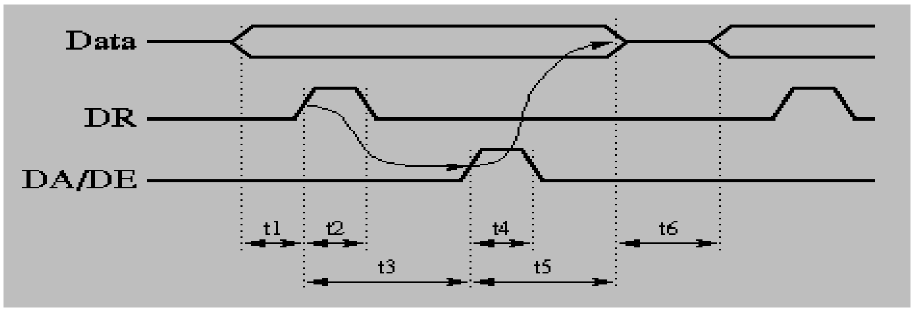
Ce type de synchronisation doit être adapté au récepteur le plus lent.

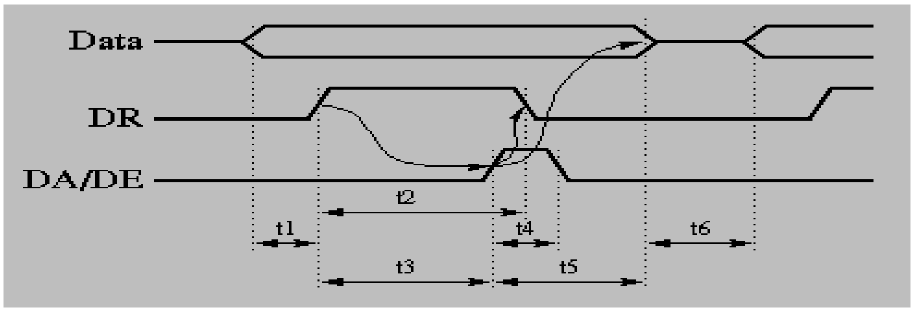
6.4.2 Communication asynchrone

Il n'y a pas besoin d'horloge fixe. Les signaux sont générés par les modules. On distingue trois type de protocoles asynchrone : non-entrelacé, semi-entrelacé, complètement entrelacé.

Transaction asynchrone non-entrelacée

Dans ce cas, il ne faut pas que t5 soit inférieur à t4 (ce qui peut arriver si l'émetteur est très rapide). Si tel est le cas, le prochain cycle de bus pourrait commencer avant la retombée du signal DA. Le protocole asynchrone semi-entrelacé permet de résoudre partiellement ce problème.

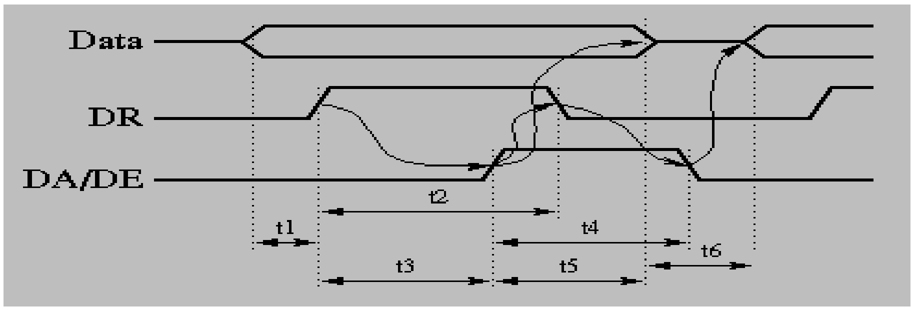




Transaction asynchrone semi-entrelacée

L'émetteur fait retomber le signal DR en réponse au signal DA. Mais le problème n'est toujours pas résolu entièrement, l'émetteur pouvant commencer une nouvelle transaction trop tôt.

Transaction asynchrone complètement entrelacée



Dans la cas d'un protocole asynchrone complètement entrelacée (aussi appelé handshake, poignée de main), l'enchainement des évènements est le suivant :

1. l'émetteur place les données sur le bus
2. l'émetteur lève le signal DR

3. le récepteur lit la donnée
4. le récepteur lève le signal DA
5. l'émetteur remet DR à zéro
6. l'émetteur retire sa donnée du bus
7. le récepteur remet DA à zéro

L'émetteur est donc forcé de débuter une nouvelle transaction lorsque le signal DA est retombé.
Ce protocole est très souvent utilisé car il permet de synchroniser des modules ayant des vitesses très différentes. Un mécanisme de *time-out* est mis en place pour éviter qu'un module ne réagissant plus soit assimilé à un module très lent. Les inconvénient de ce protocole sont que les délais de transmission des signaux de contrôle ralentissent la communication, et qu'il est sensible au bruit (le passage erroné de 1 à 0 ou de 0 à 1 d'un signal de contrôle entraine une violation du protocole).

6.4.3 Communication semi-asynchrone

Ce type de transfert emploi aussi des signaux de contrôle. Les transitions des signaux de contrôle ne peuvent avoir lieu qu'à des instants précis, déterminés par une horloge. Le temps entre deux transistions successives peut être variable. On retrouvera les modes non-entrelacé, semi-entrelacé, et entrelacé. Le principal avantage de ce protocole est qu'il est moins sensible au bruit.

? *Pourquoi ce protocole est il moins sensible aux bruits?*

6.5 Techniques d'arbitrage

L'arbitrage de bus garantit qu'à tout moment il n'y a pas plus d'un seul maître.

6.5.1 Types d'arbitrage

Arbitrage statique L'arbitrage statique consiste à rendre maître les candidats potentiels à tour de rôle, dans un ordre fixé. L'inconvénient est que si un maître ne souhaite pas faire de transaction à un moment donné, il reste maître et le bus est inutilisé (non-opération). L'avantage est la simplicité de mise en oeuvre. Ce type d'arbitrage est utilisé avec un protocole synchrone lorsqu'il y a peu de maîtres potentiels.

Arbitrage dynamique L'arbitrage dynamique permet d'allouer le bus sur demande, lorsqu'il est libre, à un maître potentiel qui en fait la demande en émettant un signal BR (Bus Request).

Lorsqu'il y a plusieurs demandes simultanées, un choix doit être fait qui peut être :

- suivant une priorité affectée de manière unique à chaque maître potentiel (les bus d'E/S utilisent souvent ce type d'arbitrage)
- de manière équitable (pour éviter qu'un maître potentiel de petite priorité voit ses demandes constamment rejetées)
- en combinant les deux premières politiques : un choix équitable départage deux demandes de même priorité.

Le bus n'est attribué que lorsqu'il est libre. La libération du bus peut avoir lieu de plusieurs manières:

- en fin de transaction,
- sur demande : le maître conserve le bus jusqu'à une nouvelle demande. Cette politique est utilisée lorsque un maître (exemple le processeur) est celui qui demande le plus souvent (exemple, par rapport à un module d'E/S),
- par préemption : un module prioritaire peut devenir maître avant la fin d'une transaction moins prioritaire.

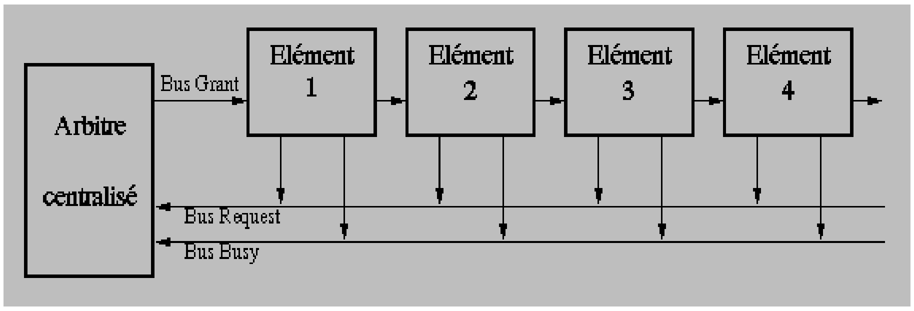
6.5.2 Méchanismes matériels d'arbitres

La réalisation matériel d'un mécanisme d'arbitrage peut être distribué, c'est à dire réparti sur l'ensemble des modules connectés au bus, ou bien centralisée, c'est à dire soit se trouver sur un seul des modules connectés au bus ou faire l'objet d'un module dédié appelé bus arbitre ou contrôleur de bus.
Cette réalisation est basée sur les trois signaux de contrôle suivants :

- BR pour Bus Request : signale une demande de bus,
- BA/BG pour Bus Acknowledge/Bus Grant : signale l'obtention du bus,
- BB pour Bus Busy : signale l'occupation du bus.

Commençons par les trois techniques d'arbitrage centralisés.

Daisy chain La structure en guirlande (daisy chain) fonctionne de la manière suivante :



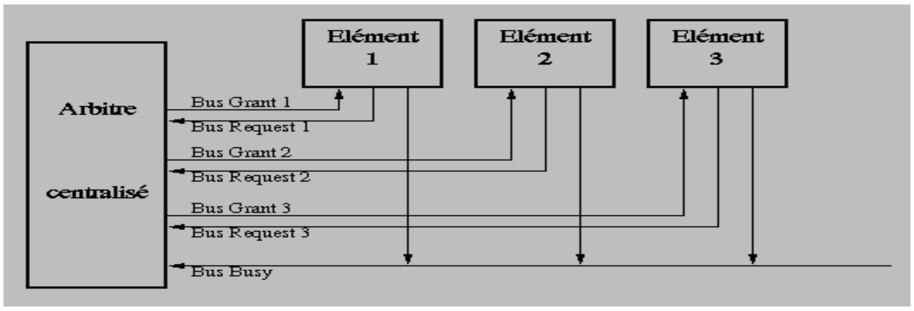
Un ordre de priorité est imposé par la guirlande

1. les éléments désirant être maître émettent un signal BR,
2. l'arbitre reçoit le ou logique des signaux BR,
3. si BR, l'arbitre envoie le signal BG à la guirlande,
4. le module demandeur de plus haute priorité lève le signal BB et fait retomber le signal BG, il devient maître du bus,
5. une fois la transaction finie, le maître fait retomber le signal BB.

Les avantages de cet arbitre sont sa simplicité de réalisation, qui est de plus quasi-indépendante du nombre de modules connectés. Les inconvénients sont la priorité statique, la lenteur de réponse de l'arbitre (proportionnelle à la longueur de la guirlande), et une grande sensibilité aux pannes.

? *Que se passe t'il si un module tombe en panne?*

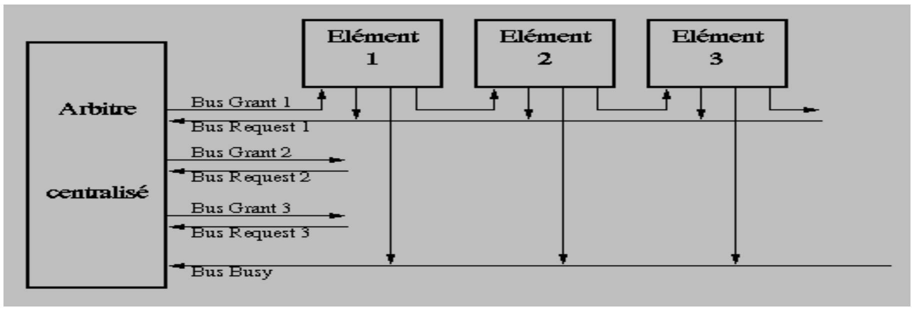
Requête-autorisation Chaque module connecté au bus dispose de lignes BG et BR propre.



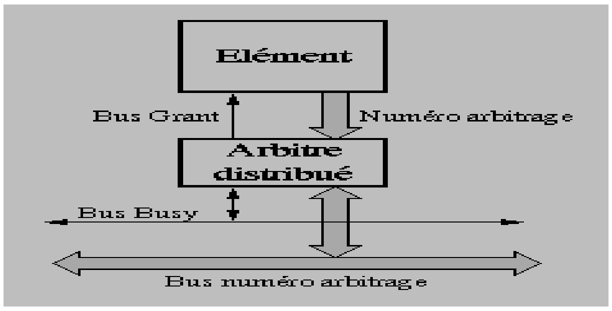
Les avantages sont l'absence de délai de réponse, de sensibilité aux pannes d'un module, et le caractère non statique de la priorité. L'inconvénient est la multiplication des lignes de contrôle.

Arbitrage mixte Cette technique est une combinaison des deux techniques précédentes.

Chaque série de modules possédant des lignes BG et BR est organisée en guirlande. Différentes stratégies d'arbitrage peuvent être implantées pour désigner le sous ensemble dont un élément sera maître du bus (cf plus bas). Pour une série de module, la priorité est fixé par la guirlande.



Arbitrage décentralisé Le défaut de l'arbitrage centralisé est la sensibilité de l'arbitre aux pannes : si l'arbitre tombe en panne, aucune allocation ne peut avoir lieu. L'arbitrage décentralisé fonctionne de la manière suivante :



Chaque module possède un numéro de priorité unique, et dialogue avec son propre arbitre. Lorsqu'un module souhaite avoir le bus,

1. il envoie son numéro de priorité à son arbitre,
2. l'arbitre place ce numéro sur le bus numéro arbitrage (un groupe de lignes de contrôle du bus réservées à cet effet),
3. le numéro qui apparait sur ces lignes est en fait le ou logique de tous les numéros de priorité des modules désirant être maître
4. ce numéro (résultat du ou logique) est comparé par chaque arbitre au numéro du module dont dépend l'arbitre,
5. si ce n'est pas celui de plus grande priorité, il est retiré, sinon il est maintenu
6. reste donc sur les lignes de contrôle du bus le numéro du module pouvant être maître,
7. l'arbitre de ce module peut donc signaler au module qu'il est le nouveau maître (signal BG) et lève le signal BB.

Ce type d'arbitre est particulièrement utilisé dans les configurations multiprocesseur, où l'échange d'informations entre processeurs se fait via un bus.

6.5.3 Stratégies d'arbitrage

Lorsque la priorité n'est pas fixée statiquement, l'arbitre doit adopter une stratégie pour départager les maîtres potentiels. Les deux notions généralement examinées sont :

- le niveau de priorité : l'arbitre choisi le module le plus prioritaire au moment où le bus est disponible,
- l'ancienneté de la demande : l'arbitre alloue le bus au module dont la demande est la plus ancienne.

Les stratégies peuvent être basées sur un seul de ces critères, ou les deux. Examinons quelques stratégies simples.

Stratégie linéaire Un numéro fixe est attribué à chaque demandeur. Les numéros sont rangés par ordre de priorité décroissante.
Exemple : 4 L 3 L 1 L 2 signifie que le module le plus prioritaire est le module 4, et le moins prioritaire le 2.
Cette stratégie est simple à réaliser, mais il y a risque de famine pour un demandeur moins prioritaire.

Stratégie circulaire La priorité circule sur les demandeurs : les numéro sont placés dans une liste circulaire, et c'est le demandeur dont le numéro est placé à droite du numéro du dernier maître du bus qui devient maître à son tour.
Exemple : 4 R 3 R 1 R 2 signifie que le module le plus prioritaire est le module 4 si le module 2 était maître du bus lors de la dernière transaction.
Cette stratégie est plus difficile à réaliser que la précédente (il faut connaitre le dernier maître) mais évite la situation de famine.

Stratégie cyclique Pour chaque demandeur, on garde l'antériorité des demandes précédentes. L'arbitre applique une stratégie linéaire sur les modules classés par ordre d'antériorité décroissante (le module le plus anciennement servi sera le plus prioritaire).
Exemple : 1 C 2 C 3 C 4, singnifie que si à un moment donné, le plus anciennement servi est l'élément numéro 4, puis l'élément numéro 1, puis l'élément numéro 3 et enfin l'élément numéro 2, l'élément le plus prioritaire sera le 4. Si, seul le numéro 3 demande le bus, il l'obtiendra, mais la priorité sera alors par ordre décroissant: 4, 1, 2 et 3.

Stratégie multiple Il s'agit simplement de mélanger plusieurs stratégies décrites ci-dessus.
Exemple : 1 R (3 C 4) R (5 L 6) signifie que 5 sera le plus prioritaire si 3 ou 4 ont été servis en dernier.

6.6 Exemple : le bus PCI

Le bus PCI (Peripheral Component Interconnect) est un standard (ses spécifications sont dans le domaine public) développé par intel , utilisé pour les communications avec les E/S (cartes graphiques, cartes réseau, controleur de disques, ...). Ses qualités (processeur-indépendant, configurable, haut débit) l'ont rendu très populaire et utilisé dans des ordinateurs personnels, des serveurs, des stations de travail.
Ses caractéristiques sont les suivantes (version 2.1) :

largeur	32 ou 64 bits
cadencé à	66 Mhz
débit max	528 Mo/s
transfert	multiplexage adresses et données en rafale
timing	semi-synchrone
arbitrage	centralisé synchrone requête-autorisation caché

Transaction Le bus PCI comprend des lignes de contrôle pour détailler type d'opération (lecture/écriture) et destinataire (mémoire, E/S). Une transaction consiste en une phase d'émission d'une adresse, suivie de une ou plusieurs phase d'émission de données. Lors d'une transaction, tous les évènements sont synchronisés sur des fronts descendants d'horloge.

Arbitrage L'arbitrage est centralisé, et repose sur deux signaux BG et BR pour chaque module. La stratégie n'est pas fixée par la spécification du bus, et l'arbitre comprend plusieurs types de stratégies (FIFO, circulaire, fixation de priorité). L'arbitrage est caché, c'est à dire qu'il est effectué pendant une transaction.

Chapitre 7

Jeu d'instructions

7.1 Introduction

On a coutume de dire que le jeu d'instruction est la partie visible de la machine par le programmeur (principalement pour l'écriture de compilateur). Il y a une correspondance directe entre le jeu d'instruction d'une machine, sa programmation et la conception de sa CPU. La connaissance du jeu d'instruction par le programmeur offre en quelque sorte une vue logique de la machine en terme de registres, de types de données, ou de fonctionnement de l'ALU.

! *Concevoir une CPU c'est en grande partie concevoir le jeu d'instructions.*

Ce cours commence par des définitions (et un rappel du chapitre 1) portant sur les instructions et leurs influence sur la CPU. Puis il est divisée en 2 grandes parties : la première partie répond à la question "que fait une instruction ?" et la seconde à la question "comment spécifier opération et opérandes d'une instruction ?"
Le pentium II sera pris en exemple.

7.2 Définition

Les opérations de la CPU sont déterminées par les instructions qu'elle exécute. L'ensemble des instructions qu'une CPU particulière peut exécuter est appelé jeu d'instruction de la CPU. Chaque type de machine possède donc un jeu d'instructions et une CPU spécifique.
Une instruction doit contenir des informations requises par la CPU pour son exécution :

- le code opération : un code (binaire) identifiant l'opération à exécuter, appelé opcode
- la référence aux opérandes sources : une ou plusieurs adresses faisant références aux opérandes paramètres de l'opération (qui peuvent provenir de la mémoire, d'un registre de la CPU, ou d'un périphérique),
- la référence à l'opérande résultat : l'opération peut produire un résultat et référencer explicitement ce résultat,
- la référence à la prochaine instruction : indique à la CPU où chercher l'instruction suivant l'exécution de l'instruction courante (elle peut être implicite, c'est le cas lorsqu'il l'instruction suivante est contigüe à l'instruction courante).

Représentation de l'instruction Chaque instruction est représentée par une séquence de bits, découpée en champs. Il y a différents formats d'instruction selon le nombre de parties réservées aux opérandes. En général, un jeu d'instructions inclue plusieurs formats d'instruction différents.
Durant l'exécution d'une instruction, celle-ci est chargée dans le registre d'instruction (IR) de la CPU, qui doit en extraire les différents champs.

Cette séquence de bit admet une représentation symbolique, où l'opcode est représenté par une abbréviation appelée *mnémonique*. Chaque opcode admet une représentation binaire fixe. Par exemple $ADD\ R, Y$ signifie ajouter la valeur contenue à l'adresse Y au contenu du registre R.
Il est donc possible de programmer en employant un tel langage, appelé *langage machine*. Aujourd'hui, la programmation ne se fait plus qu'avec des langages de haut niveau (ou au pire avec des langage d'assemblage, qui sont une légère abstraction des langages machine). Les langages machines reste cependant un bon moyen de décrire les jeux d'instruction.

! *Le jeu d'instruction doit être suffisament expressif pour coder toute instruction d'un langage de haut niveau. Ainsi, une instruction d'un langage de haut niveau correspond en fait à plusieurs instructions du langage machine.*

? *Soit l'instruction en langage de haut niveau $X = X + Y$. Comment est elle traduite en langage machine ?*

Conception du jeu d'instruction Il n'y a pas de consencus sur la conception d'un jeu d'insctructions. Les aspect fondamentaux à prendre en compte sont :

- les opérations : combien en faut il, quelle doit être leur complexité ?
- les types de données : quels sont les types de données supportés par les opérations ?
- les registres : combien en faut il et comment sont ils utilisés ?
- l'adressage : quelles sont les différentes façons de faire référence aux données ?
- le format : quelle(s) longueur(s) d'instruction adopter, quelle sera le nombre et la taille de champs ?

Ces aspects sont grandement reliés les uns aux autres, et doivent être considérés ensemble. Ils sont détaillés dans la suite.

! *La conception du jeu d'instructions influe grandement sur la conception de la CPU.*

7.3 Caractéristiques

7.3.1 Classification

On peut classifier les jeux d'instruction selon le nombre d'adresses d'opérandes manipulées (le format d'instruction), ou selon la relation entre la mémoire et les registres de la CPU.

Format d'instruction Une opération fait référence à des données sources et peut faire référence à une destination. Ces références sont en fait les adresses des données. Le format d'une instruction correspond au nombre de champs de l'instruction réservé à ces adresses. En théorie, il en faudrait 4 (pour deux opérandes, un résultat, la prochaine instruction). En pratique :

- les formats d'instruction à trois adresses ils sont peu courants, car donne lieu à des instructions longues ;
- les formats d'instruction à deux adresses nécessitent que l'une des adresses fasse office de source et de destination (ce qui doit être géré si la donnée source doit rester disponible) ;
- les instructions employant basé sur un format à une seule adresse emploient systématiquement un registre CPU, l'accumulateur, pour garder une opérande et le résultat ;
- Les instructions n'employant aucune adresse utilisent systématiquement une pile, ou stack (i.e., une structure de liste gérée de manière LIFO) pour stocker opérandes et résultats.

? *Quel est l'impact du format dans le codage de $X = X + Y$?*

Le choix du nombre d'adresse est fait en considérant le fait suivant : moins il y a d'adresses, plus les instruction sont courtes, moins la CPU est complexe, mais les instructions seront plus nombreuses, donc les programmes plus lents à exécuter. Disposer de plusieurs adresses est conjoint à la possibilité d'utiliser plusieurs registres.

! *Actuellement la plupart des machines utilisent un jeu d'instructions mélangeant les formats 2 adresses et 3 adresses.*

Relation entre mémoire et registres Les registres sont des éléments de mémorisation rapide interne à la CPU. En disposer permet de minimiser les accès à la mémoire et d'accélérer l'exécution des instructions. Ainsi le jeu d'instruction peut être basé sur différentes stratégies de stockage des données :

- registre-registre (chargement-rangement) : deux instructions sont dédiées au chargement de registre à partir de la mémoire et au rangement de registre dans la mémoire. Donne lieu à des instructions simples dont l'exécution est rapide (nombre de cycles fixe) mais le nombre d'instructions générées est important,
- registre-mémoire : les instructions peuvent manipuler les données directement dans les registres ou dans la mémoire. Le code généré est plus compact mais les instructions sont plus difficiles à décoder, et le nombre de cycles pour l'exécution est variable,
- mémoire-mémoire : il y a accès systématique à la mémoire, qui peut devenir un goulôt d'étranglement (concerne les anciennes machines).

! *La tendance actuelle est l'emploi du chargement-rangement.*

7.3.2 Types d'instructions

Le nombre d'opcodes varie d'une machine à l'autre. On peut cependant donner les catégories principales suivantes :

- transfert de données : ces instructions spécifient l'emplacement des opérandes source et destination, la taille des données à transférer, ainsi que la manière d'accéder à ces donnée (le mode d'adressage) ;
- les opérations arithmétique : toutes les machines comprennent les opérations de base sur des entiers signés (fixed point numbers), la plupart incluent des instructions pour les opérations sur des flottants, d'autres instructions sont parfois disponibles ;
- logique : servent aux manipulations de données bit à bit (masque), aux décalages,
- conversion : changement du format d'une donnée (e.g., DCB vers binaire) ;
- E/S : transfert de données avec les E/S ;
- contrôle système : instructions réservées à un mode d'exécution privilégié (e.g., pour l'utilisation du système d'exploitation) ;
- transfert de contrôle : instructions impliquant un changement explicite du compteur ordinal, pour exécuter un branchement conditionnel, ou une répétition.

7.3.3 Types d'opérandes

Les données sur lesquelles les instructions opèrent peuvent être rangées dans les catégories suivantes :

- les adresses, qui peuvent être considérées comme des entiers non signés sur lesquels des calculs sont possibles ;
- les nombres : entiers (fixed-point), flottants, DCB. Ils peuvent avoir un rôle spécifique (e.g., compteur, de taille de champs) ;
- les caractères, sous forme de codes ASCII ;
- des données logiques, c'est à dire des séquences de bits ayant chacun un sens (e.g., un tableau de booléens).

Certaines machines emploient des types de données plus évoluées (liste ou chaîne de caractères).

7.3.4 Exemple : le pentium II

Types de données Le pentium II utilise la convention little endian, c'est à dire que l'octet de plus faible poids est stocké à la plus petite adresse. Il travaille sur des données de longueur 8 (octet), 16 (mot), 32 (double mot) ou 64 bits (quadruple mot) qui sont qualifiées de génériques, et peuvent contenir une séquence de bits arbitraire. Plus précisément, le pentium II supporte des types de données spécifiques (manipulés par des opérations particulières) :

- général : contenu arbitraire sur 8, 16, 32 ou 64 bits ;
- entier : binaire signé en complément à 2 sur 8, 16 ou 32 bits ;
- ordinal : entier non signé sur 8, 16 ou 32 bits ;
- BDC : entier entre 0 et 9 sur 8 bits ;
- packed bcd : 2 chiffres BDC (de 0 à 99) sur un octet ;
- pointeur : une adresse sur 32 bits ;
- champs binaire : séquence de bits considérés chacun de manière ndépendante (bornée à 2^{32} bits);
- chaîne d'octet : séquence d'octets, mots ou doubles mots (bornée à 2^{32} octets) ;
- flottants : correspond à un ensemble de types utilisés par l'unité arithmétique en virgule flottante, manipulées par des opérations dédiées. Ces types sont :
  - entier mot : binaire en complément à 2 sur 16 bits ;
  - entier court : binaire en complément à 2 sur 32 bits ;
  - entier long : binaire en complément à 2 sur 64 bits ;
  - packed BCD : 1 bit de signe, 18 octet BDC ;
  - simple précision, double précision et précision étendue (sur 80 bits, 64 bits de mantisse et 15 bits d'exposant) : standard IEEE 754.

Types d'opération Le pentium II comprend un jeu d'opérations complexe, incluant de nombreuses opérations spécialisées. L'idée était d'optimiser la traduction de code de haut niveau en langage machine. Ainsi, on trouve les catégories d'instructions communes à la majorité des machines, et d'autres catégories particulières au pentium II :

- les opérations de transfert de contrôle donnent lieu à un type d'opérations supportant des opérations de langage de haut niveau ;
- un ensemble d'instruction est dédié à la gestion de la mémoire et du cache interne (ces instructions sont exécutées seulement à partir de l'OS) ;
- des instructions de branchement sont définies pour fonctionner conjointement au test de registres de condition (registres contenant des informations sur des opérations arithmétique ou de comparaison).

En 1996, Intel a introduit un ensemble de 57 nouvelles instructions dédiées aux opérations multimédia, appelé MMX. Ces instructions permettent d'effectuer une même opération sur plusieurs éléments à la fois (principe SIMD). Les données vidéo et audio sont composées de large tableaux de petits types de données (8 ou 16 bits, parfois 32) alors que les instructions conventionnelles sont calibrées pour des types de données plus grands (32 ou 64 bits). Ainsi, trois nouveaux types de données ont été introduits en MMX, chacun de 64 bits découpés en champs contenant chacun un entier :

- packed bytes : 8 octets ;
- packed word : 4 mots ;

– packed doubleword : 2 double mot.

La majorité des nouvelles instructions consiste en une opération parallèle sur des octets, mots ou doubles mots.

7.4 Structure d'une instruction

Dans la conception d'un jeu d'instruction, un paramètre important est le nombre de registres et leur utilisation. Les CPU modernes comportent de multiples registres non spécifiques disponibles pour le programmeur.

7.4.1 L'adressage

L'instruction fait référence à des données. Or la taille d'un champ adresse dans une instruction est petite, alors qu'il faut pouvoir adresser un espace mémoire important. Ainsi une variété de technique d'adressage, c'est à dire de manières d'obtenir une adresse effective, ont été proposées.

Modes d'adressage Voici les modes d'adressage les plus communs :

- l'adressage immédiat : l'opérande est donnée dans l'instruction. Pas de référence à la mémoire, mais la taille de l'opérande est limitée ;
- l'adressage direct : l'adresse mémoire de l'opérande est donnée dans l'instruction. Simple, mais la taille de l'espace adressable est limitée ;
- l'adressage registre : l'adresse du registre interne (généralement sur 3 ou 4 bits) dans lequel est contenue l'opérande est donnée dans l'instruction. Pas de référence à la mémoire, mais la taille de l'espace adressable est limitée ;
- l'adressage indirect par mémoire : l'adresse de l'opérande est le contenu de l'adresse mémoire donnée dans l'instruction. Implique plusieurs références à la mémoire ;
- l'adressage indirect par registre : l'adresse de l'opérande est le contenu du registre dont l'adresse est donnée dans l'instruction.
- adressage déplacement : l'adresse de l'opérande est donnée en deux parties, une base dans un registre (peut être le Program Counter) et un déplacement (à ajouter à la base) dans l'instruction. Flexible, mais complexe.

On s'aperçoit qu'il y a antagonisme entre :

- espace adressable et flexibilité d'une part, et
- nombre de références mémoire et complexité du calcul d'adresse d'autre part.

Les modes d'adressage immédiat, indirect par registre et déplacement sont les plus utilisés. Toutes les machines disposent de plusieurs modes d'adressage. Pour un jeu d'instruction, il y a deux manières de différencier les modes d'adressage utilisés :

- par l'utilisation d'un ou plusieurs bits indiquant quel est le mode employé. Ces bits sont appelés *spécificateur d'adresse*. Cette méthode est adoptée lorsqu'on souhaite disposer d'un grand nombre de mode d'adressage, et peut donner lieu à une taille d'instruction variable ;
- par l'utilisation d'opcodes différents, ce qui permet de conserver une taille d'instruction fixe (ce qui facilitera la réalisation matérielle).

7.4.2 Format d'instruction

Le format d'une instruction définit les champs de l'instruction (opcode et adresses) et l'adressage employé.

Taille d'instruction Plus le jeu d'instructions est complexe (en termes de nombre d'opcodes, de types d'opérandes et de mode d'adressage), plus facile (compacte) sera la programmation. Mais un jeu d'instruction complexe implique une longueur d'instruction importante, ce qui implique une consommation importante en espace. La longueur d'une instruction doit être considérée conjointement à

- la taille et l'organisation de la mémoire. Ainsi la mémoire pourra être adressable à la taille de l'instruction ;
- la structure d'interconnexion. Ainsi la longueur de l'instruction sera un multiple de la taille du bus de données ;
- la complexité et la vitesse de la CPU. Par exemple, si la CPU exécute les instructions plus vite qu'elle ne les charge à partir de la mémoire, la mémoire devient un goulot d'étranglement ; la solution peut être de réduire la taille de l'instruction.

Pour un jeu d'instruction, la taille de l'instruction peut demeurer fixe ou être variable. Une taille d'instruction variable autorise une large gamme d'opcodes (de tailles différentes), et une plus grande flexibilité d'adressage. Mais cela accroit la complexité de la CPU.

Allocation des bits L'allocation des bits pour les différents champs de l'instruction dépend des facteurs suivants :

- le nombre d'opérandes ;
- le nombre d'opérations : pour une taille d'instruction fixée, plus le nombre d'opcodes est grand plus la capacité d'adressage découlant des champs adresse sera réduite. Si on souhaite à la fois une taille d'instruction raisonnable, une capacité d'adressage raisonnable et un nombre d'opcodes important, on peut utiliser une taille d'opcode variable. Ainsi, pour des instructions utilisant peu de paramètres ou une capacité d'adressage moindre, quelques bits sont rajoutés au champs opcode.

- le nombre de modes d'adressage : si le mode d'adressage n'est pas indiqué implicitement (certaines instructions sont réservées à un mode d'adressage particulier), il l'est explicitement via un ou plusieurs bits dédiés. Chaque adresse d'opérande peut comporter le mode d'adresage qui la concerne ;
- l'utilisation des registres : un ensemble de registres est disponible dans la CPU pour recevoir des données et des adresses. Ceci constitue un petit espace d'adressage, qui ne nécessite que quelques bits d'adresse (maximum 5). Ces registres peuvent être séparés en groupe (un pour les données, un autre pour les adresses), dont l'identification est déportée au niveau de l'opcode ;
- l'espace adressable : il est directement relié au nombre de bit d'adresses à utiliser et donc au modes d'adressage permis ;
- la façon d'adresser la mémoire : adresser une mémoire à l'octet demande plus de bit d'adresse qu'adresser la même mémoire au mot.

7.4.3 Quelques exemples

Les paragraphes suivant montrent comment les choix de conception de jeu d'instructions ont été fait pour différentes machines.

Alpha Le processeur ALpha de DEC contient 32 registres (R0 à R31) de 64 bits pour la manipulation des entiers et 32 registres (F0 à F31) de 64 bits pour la manipulation des flottants. Les registres R31 et F31 sont à lecture seule et contiennent la représentation de 0.
Les instructions ont une taille fixe de 32 bits. Il y a 4 formats d'instruction possibles :

- les instructions spécifiques au système d'exploitation : opcode sur 6 bits + un nombre sur 26 bits ;
- les instructions de branchement : opcode sur 6 bits + adresse registre sur 5 bits + déplacement sur 21 bits ;
- les instructions de transfert de données : opcode sur 6 bits + adresse registre sur 5 bits + adresse registre sur 5 bits + déplacement sur 16 bits ;
- les instructions utilisées pour les opérations de calcul entier ou flottant : opcode sur 6 bits + adresse registre sur 5 bits + adresse registre sur 5 bits + extension de l'opcode sur 11 bits + adresse registre sur 5 bits.

SPARC Le processeur SPARC de SUN possède un grand nombre de registres (peut être supérieur à 520). Néanmoins seuls 32 registres sont visibles simultanéments, divisés en 4 groupes de registres spécifiques. Les instructions ont une taille fixe de 32 bits. 3 format d'instructions différents sont utilisés. Le format est codé sur 2 bits.

- le premier ne correspond qu'à une seule opération de branchement, l'appel de sous-programme. Cette instruction ne comprend donc pas d'opcode mais simplement un déplacement sur 30 bits ;

- le deuxième a un opcode sur 3 bits. Cet opcode distingue 2 types d'instruction
  - les instructions de branchement : une condition de saut sur 4 bits (comparée au contenu d'un registre de condition), un déplacement sur 22 bits
  - une instruction permettant de charger une donnée dans les poids fort d'un registres : adresse registre sur 5 bits + 22 bits de donnée immédiate ;
- le troisième format (3 adresses) correspond à toutes les autres opérations : une adresse registre de destination sur 5 bits, une adresse registre source sur 5 bits, une extension de l'opcode sur 6 bits. Selon cette extension :
  - une extension du code opération pour les opérations flottante sur 9 bits + une adresse registre source sur 5 bits
  - une donnée immédiate sur 13 bits
  - un espace d'adressage mémoire sur 8 bits + une adresse registre source sur 5 bits.

Pentium II Le pentium II comprend 9 modes d'adressage, pouvant utiliser un registre ségment, un registre de base et un registre d'index :
- immédiat : l'opérande est inclue dans l'instruction
- registre : l'opérande est contenue dans un registre
- déplacement : l'adresse est le contenu du registre ségment + un déplacement inclu dans l'instruction
- indirect par registre : l'adresse est le contenu du registre ségment + contenu du registre de base
- base et déplacement : idem précédent + déplacement
- base indexé avec déplacement : idem précédent + registre index
- base indexé avec déplacement : idem précédent + registre index × un facteur
- indexé avec déplacement : idem précédent - base
- relatif : l'adresse est le ontenu du compteur ordinal + un déplacement contenu dans l'instruction

Le pentium comprend une grande variété de formats, l'opcode étant le seul élément réccurrent. Le mode d'adressage est donnée par l'opcode. L'instruction comporte :
- un préfixe de 0, 1, 2, 3 ou 4 octet,
- un opcode de 1 ou 2 octets,
- un spécificateur d'adresse de 0, 1 ou 2 octets,
- un déplacement de 0, 1, 2 ou 4 octets,
- un champ pour l'adressage immédiat de 0, 1, 2 ou 4 octets

Les raisons de cette variété sont l'efficacité d'exécution de langages de haut niveau, et le respect de la compatibilité ascendante de la famille 8086 dont le pentium est issu.

7.5 Conclusion

On oppose traditionnellement

- CISC (Complex Instruction Set Computer), les machines possédant un jeu d'instruction complexe, en nombre d'instruction, de mode d'adressages, de registres, etc... (comme le pentium) à
- RISC (Reduced Instruction Set Computer), les machines possédant un jeu d'instruction réduit (comme le SPARC ou l'ALPHA) et donc une architecture correpondant à ce jeu d'instruction réduit.

Un jeu d'instructions complexe est il préférable à un jeu d'instruction réduit RISC ? En fait, il est très difficile de comparer des machines CISC à des machines RISC. Actuellement, les deux technologies ont quelque peu convergé : les performances atteintes font que les systèmes RISC deviennent plus complexes, et la recherche de performances supérieures pousse les concepteurs CISC à envisager des problèmes traditionnellement associés aux machines RISC.

Chapitre 8

CPU

8.1 Introduction

Ce cours complète le précédent en étudiant la structure de la CPU. Le modèle RISC est utilisé comme support réccurrent de la présentation.

8.2 Organisation de la CPU

8.2.1 Organisation du processeur

La CPU possède :
- une ALU pour le traitement des données,
- une unité de commande qui contrôle le mouvement des données et des instructions, ainsi que les opérations de l'ALU,
- des registres spécifiques pour stocker temporairement les données et les instructions
- un bus interne pour interconnecter ces différentes composants.

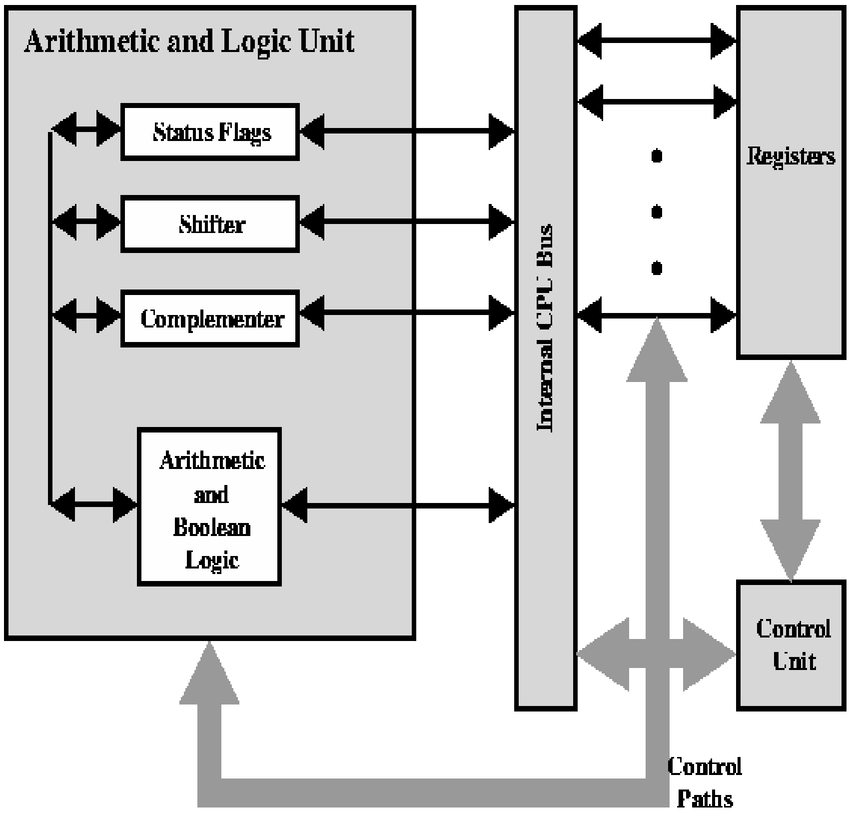
8.2.2 Organisation des registres

L'exécution d'une instruction demande le stockage temporaire de données. La mémoire interne de la CPU est découpée en 2 types de registres :
- les registres visibles par l'utilisateur qui permettent au programmeur d'optimiser les références à la mémoire
- les registres de contrôle et de statuts, utilisés par l'unité de commandes pour controler l'activité de la CPU et par des programmes du système d'exploitation pour controler l'exécution des programmes.

Registres visibles par l'utilisateur Une registre utilisateur est un registre référençable par les instructions exécutées par la CPU. On trouve différentes catégories :
- données : ne peuvent pas être employés pour le calcul d'adresses ;
- adresses : souvent dévolus à un mode d'adressage particulier (contenant des valeurs de base ou d'index) ;
- conditions (flags) : constitués d'une suite de bits indépendants dont chacun est positionné en fonction du résultat d'une opération ;
- autres : n'ont pas de fonction spécifique.

Le choix des registres pour une CPU est basé sur les questions suivantes :
- quelle proportion de registres spécialisés adopter ?
- quel nombre de registres adopter ?
- quelle taille de registres adopter ?

? *Quels sont les impacts des réponses à ces questions ?*



Registres de contrôle et de statuts En général, ces registres ne sont pas visibles par l'utilisateur. 4 registres sont essentiels à l'exécution d'une instruction, ils sont utilisés pour l'échange avec la mémoire principale :

- le compteur ordinal (PC, pour Program Counter) : contient l'adresse de la prochaine instruction à exécuter ;
- le registre d'instruction (IR) : contient l'instruction du fetch le plus récent ;
- le registre d'adresse mémoire (MAR) : contient une adresse mémoire. Est directement connecté au bus d'adresse ;
- le registre tampon mémoire (MBR) : contient un mot de données à écrire en mémoire ou un mot lu récemment. Est directement connecté au bus de données. Fait le lien avec les registres visibles par l'utilisateur.

Des registres supplémentaires peuvent être intercalés entre l'ALU et les registres utilisateurs/le MBR.

Un registre PSW (Program Status Word) contient des informations de status. Parmi les plus fréquentes :

- signe : le bit de signe du résultat de la dernière opération arithmétique
- zéro : à 1 lorsque le résultat est 0
- retenue : à 1 lorsqu'une opération a générée une retenue
- égal : à 1 si le résultat d'une comparaison est une égalité
- débordement : à 1 lorsqu'une opération a provoqué un débordement
- interruption : indique si le fonctionnement normal peut être interrompu
- superviseur : indique un mode privilégié

En plus du PSW, un registre peut faire office de pointeur sur une zone mémoire contenant des informations supplémentaires.
L'implantation de registres de contrôle peut aussi être faite en fonction de la prise en compte du système d'exploitation.

8.3 Cycle de l'instruction

Le cycle de l'instruction fait référence à la façon dont la machine exécute une instruction.

8.3.1 Cycle normal

Ce travail est effectué par la CPU et consiste en :

- recherche de l'instruction (fetch) : l'instruction est lue depuis la mémoire
- interprétation de l'instruction (decode) : l'instruction est décodée pour déterminer à quelle action elle correspond
- exécution (execute) :
  - recherche des données (fetch data) : l'exécution de l'instruction peut demander la lecture de données dans la mémoire ou depuis un module d'E/S
  - traitement des données (process data) : l'exécution de l'instruction peut demander des opérations arithmétiques ou logiques sur les données
  - écriture des données (write data) : l'exécution de l'instruction peut demander l'écriture du résultat dans la mémoire ou depuis un module d'E/S

Toutes les machines ont un schéma d'exécution similaire, qui consiste donc à exécuter la boucle suivante :

- répéter
  - fetch
  - decode
  - execute

Flots de données La séquence exacte des évènements se produisant durant un cycle dépend de l'architecture de la CPU. Considérons le cas générique d'une CPU possédant un registre MAR, un registre MBR un compteur ordinal PC et un registre d'instruction IR.

- fetch :
  - $MAR \leftarrow PC$
  - l'unité de contrôle demande une lecture de la mémoire principale
  - le résultat de la lecture est placé dans MBR
  - $IR \leftarrow MBR$
  - $PC \leftarrow PC + 1$
- decode :
  - l'unité de contrôle détermine si le contenu de IR utilise un adressage indirect ; si c'est le cas, cycle indirect :
    - $MAR \leftarrow$ les N bits de poids faibles de MBR
    - l'unité de contrôle demande une lecture de la mémoire principale
    - le résultat de la lecture est placé dans MBR
    - $MAR \leftarrow MBR$
    - l'unité de contrôle demande une lecture de la mémoire principale
    - le résultat de la lecture est placé dans MBR
- execute : dépend des instructions.

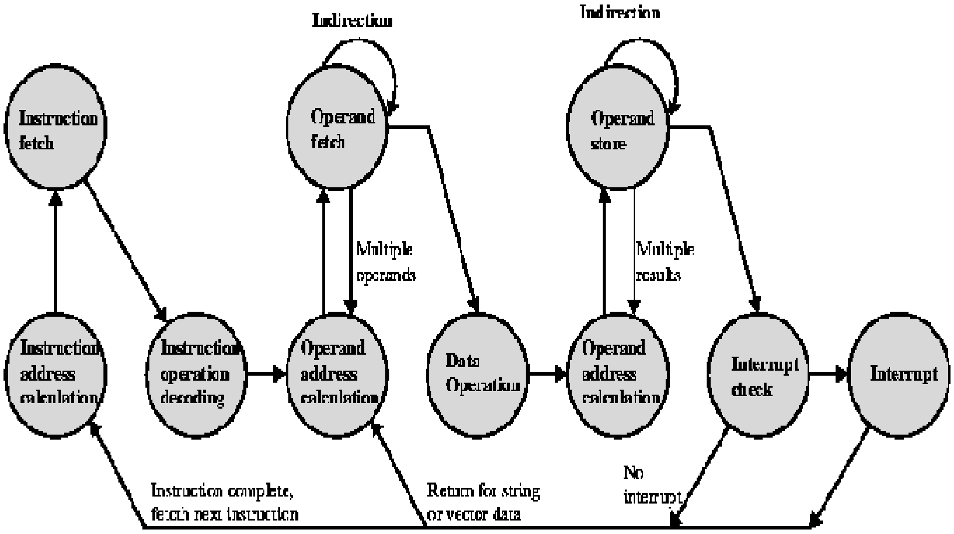
! *Les cycles fetch et decode sont simples et prévisibles, le cycle execute est imprévisible.*

8.3.2 Interruptions

Chaque ordinateur permet un mécanisme selon lequel un module (E/S, mémoire) peut interrompre le déroulement normal de ce cycle. Si une interruption survient, l'état courant (principalement le compteur ordinal) est sauvegardé et l'interruption est traitée (i.e., l'exécution est détournée sur une suite d'instructions traitant l'interruption, appelée routine d'interruption).
Le cycle d'interruption est simple et prévisible :

- $MBR \leftarrow PC$
- $MAR \leftarrow$ une adresse mémoire ou sauvegarder PC
- l'unité de contrôle demande une écriture dans la mémoire principale
- $PC \leftarrow$ adresse de la routine d'interruption

Le nouveau cycle commence par le fetch de la première instruction de la routine d'interruption.



8.4 Reduced Instruction Set Computers

8.4.1 Caractéristiques

Une raison majeur de l'avènement des RISC est la constatation que les compilateurs produisent surtout des instructions simples (les instructions complexes des CISC sont donc rarement exploitées et peuvent souffrir d'être recodées par des instructions plus simples).
Les caractéristiques communes aux architectures RISC sont les suivants :

- vers une instruction par cycle : les instructions peuvent être réalisées par matériel ;
- des opérations registre-registre dans une achitecture de type chargement-rangement ;
- des modes d'adressage simples et peu nombreux, les modes d'adressages complexes pouvant être réalisés à partir des modes d'adressages simples ;
- des formats d'instructions simples et peu nombreux, taille d'instruction fixe, possédant des champs fixes.

Le tout résulte en des instructions et une unité de contrôle simples.

8.4.2 Exemple de machine RISC

Considérons la machine RISC suivante.

Jeu d'instructions et registres La machine utilise 3 formats d'instructions (on ne s'intéresse pas au troisième format correspondant aux instructions de saut).

COP	Rs1	Rd	Immédiat
6 bits	5 bits	5 bits	16 bits

COP	Rs1	Rs2	Rd	Fonction
6 bits	5 bits	5 bits	5 bits	11 bits

COP	Déplacement
6 bits	26 bits

Les instructions sont réparties dans 4 groupes :

- accès mémoire : instruction de chargement-rangement ;
- opération registre-registre : calcul ou échange de données n'impliquant que des registres ;
- opérations registre-immédiat : calcul impliquant un registre et une donnée immédiate ;
- branchement.

Les instructions et les données sont lues dans deux caches internes dédiés (adressables à l'octet). Les registres internes sont les suivants :

- RI : registre d'instruction
- CP : compteur ordinal
- NCP : adresse de l'instruction suivante
- A, B, IMM entrées d'ALU
- SALU : sortie ALU
- COND : registre de condition/statut
- DMC : sortie mémoire de données
- des registres utilisateurs

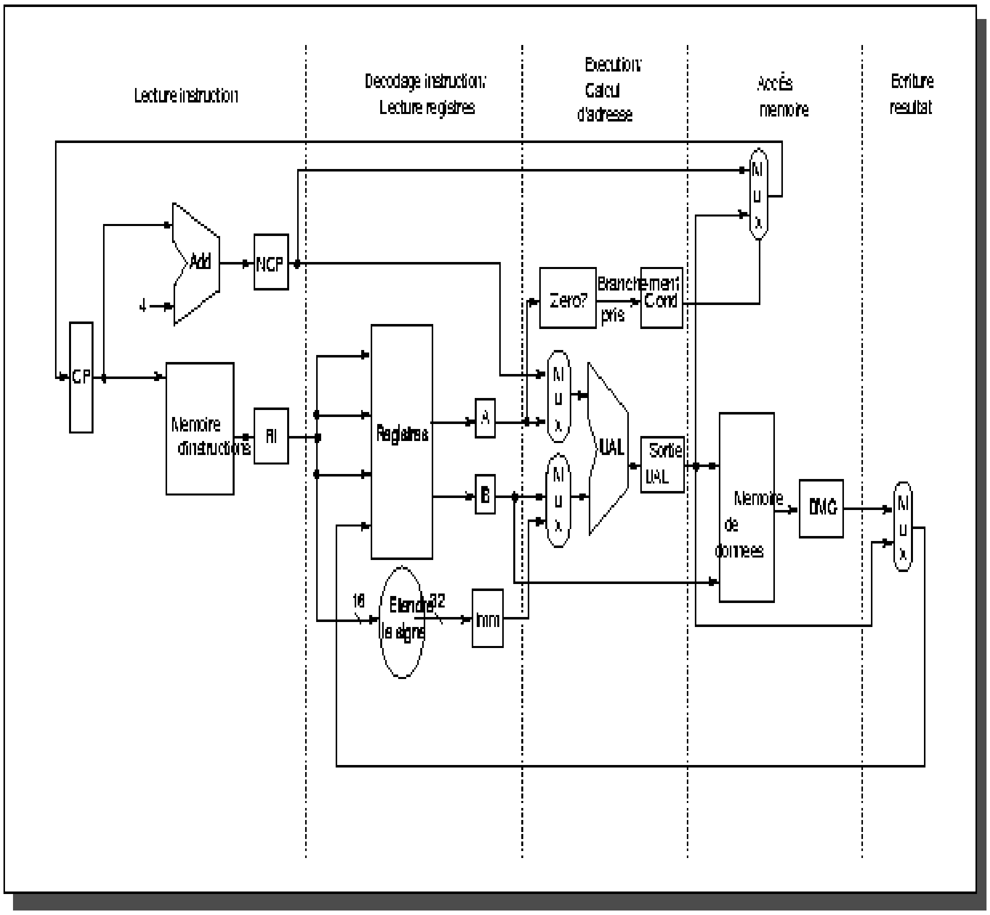
Cycle d'instruction Détaillons le cycle d'instruction exécuté par cette machine (le traitement de l'opcode n'est pas détaillé).

1. **fetch (LI)**
   - RI $\leftarrow$ mémoire d'instruction(PC)
   - NCP $\leftarrow$ CP + 4
2. **decode (DI)**
   - A $\leftarrow$ registre($RI_{25..21}$)

   - B $\leftarrow$ registre($RI_{20..16}$)
   - IMM $\leftarrow$ $RI_{15..0}$
3. **execute/calcul de l'adresse effective (EX)** selon l'opcode, une des 4 opérations suivantes est réalisée :
   - accès mémoire
     - SALU $\leftarrow$ A + IMM
   - opération registre-registre
     - SALU $\leftarrow$ A op B
   - opération registre-immédiat
     - SALU $\leftarrow$ A op IMM
   - branchement
     - SALU $\leftarrow$ NCP + IMM
     - COND $\leftarrow$ A op 0
4. **accès mémoire/branchement (MEM)**
   - chargement
     - DMC $\leftarrow$ mémoire données(SALU)
     - CP $\leftarrow$ NCP
   - rangement
     - mémoire données(SALU) $\leftarrow$ B
     - CP $\leftarrow$ NCP
   - branchement
     - si COND alors CP $\leftarrow$ SALU sinon CP $\leftarrow$ NCP
   - autres instructions
     - CP $\leftarrow$ NCP
5. **écriture du résultat (ER)**
   - chargement
     - registre($RI_{20..16}$) $\leftarrow$ DMC
   - opération registre-registre
     - registre($RI_{15..11}$) $\leftarrow$ SALU
   - opération registre-immédiat
     - registre($RI_{20..16}$) $\leftarrow$ SALU

? *Quels sont les modes d'adressage employés ?*

L'exécution d'une instruction prendra de 4 à 5 cycles, les plus rapides étant les instructions de branchement. Pour une machine de ce type, on estime qu'elles constituent 12 % des instructions exécutés. La durée moyenne d'une instruction est donc de $12\% \times 4 + 88\% \times 5 = 4,88$ cycles.



Réalisation

Un intérêt des machines RISC est qu'elle autorise un pipeline plus efficace.

8.5 Pipeline

Pour améliorer les performances de cette machine on cherche à diminuer le temps d'exécution d'une instruction. L'idée est de faire travailler à la chaîne les différentes unités fonctionnelles.

8.5.1 Pipeline de base

Pipeline à 3 étages Soit l'exécution d'une instruction en 3 phases *indépendantes* (fetch, decode, execute) réalisée chacune en 1 cycle d'horloge. La mise en pipeline consiste à faire travailler en parallèle les 3 unités responsables des 3 phases.

Ainsi, au cycle i, on recherche l'instruction i, on décode l'instruction $i-1$ et on exécute l'instruction $i-2$. Au cycle $i+1$, on recherchera l'instruction $i+1$, on décodera l'instruction i et on exécutera l'instruction $i-1$.

cycle d'horloge	1	2	3	4	5	6
étage fetch	inst 1	inst 2	inst 3	inst 4	inst 5	inst 6
étage decode		inst 1	inst 2	inst 3	inst 4	inst 5
étage execute			inst 1	inst 2	inst 3	inst 4

Le temps moyen d'exécution d'une instruction est divisé par 3. On parle de pipeline à 3 étages. Le temps d'exécution de n instructions est de 2+n cycles d'horloge. Le délai de 2 cycles est appelé temps de latence du pipeline.

Pipeline à 5 étages Appliquons ce principe à la machine RISC décrite plus haut. Les 5 phases doivent être rendues indépendantes. Toute information issue d'un étage et devant être utilisée par l'étage suivant doit être mémorisée dans un registre servant d'interface entre les étages. On obtient ainsi le fonctionnement décrit par le tableau suivant :

cycle d'horloge	1	2	3	4	5	6
étage LI	inst 1	inst 2	inst 3	inst 4	inst 5	inst 6
étage DI		inst 1	inst 2	inst 3	inst 4	inst 5
étage EX			inst 1	inst 2	inst 3	inst 4
étage MEM				inst 1	inst 2	inst 3
étage ER					inst 1	inst 2

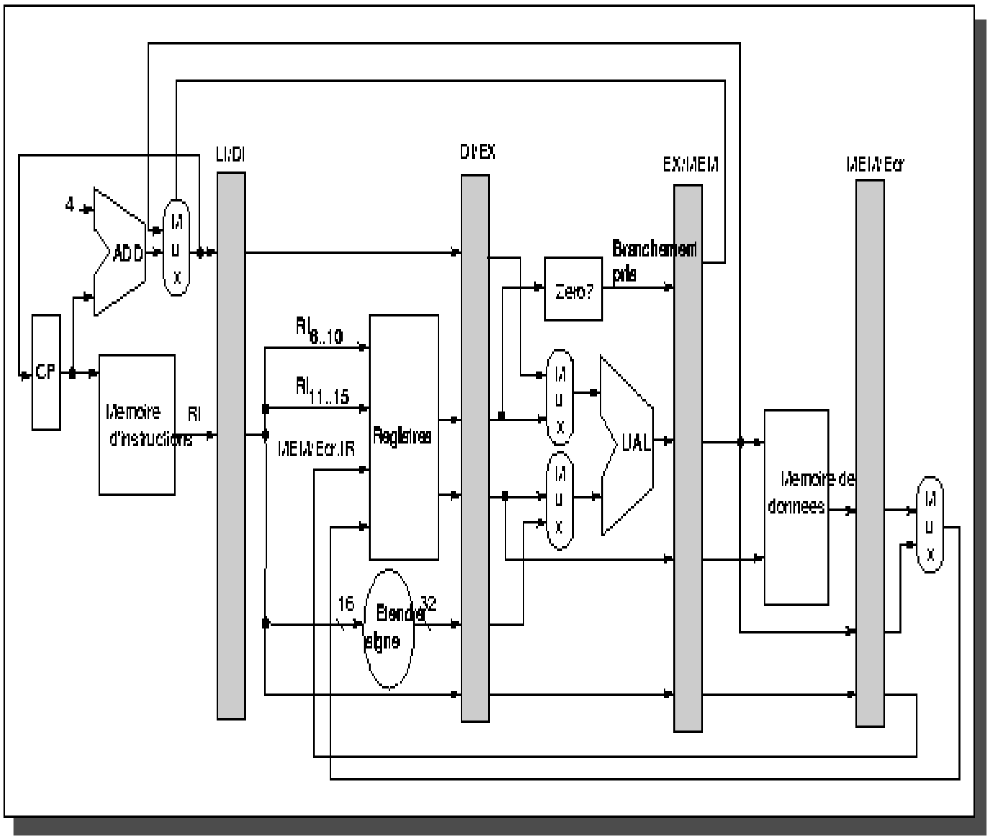
Quelques remarques :

1. l'emploi de deux caches spécialisés se révèle judicieux : chaque mémoire est sollicitée à chaque cycle. Cela n'aurait pas été possible si un seul cache contenant données et instructions avait été utilisé ;
2. le bloc registres doit permettre deux lectures (registres sources pour l'étage DI) et une écriture (résultat pour l'étage ER) dans le même cycle. Un problème survient si l'écriture concerne un registre accédé en lecture ;
3. le multiplexeur de sélection du registre CP a été déplacé de l'étage MEM vers l'étage LI. L'exécution à l'étage LI peut dépendre de la présence dans l'étage MEM d'une instruction de branchement compliquant le fonctionnement du pipeline.

8.5.2 Les aléas

Un aléa est une situation qui empêche l'instruction suivante de s'exécuter au cycle d'horloge prévu. On distingue plusieurs sortes d'aléas.

Aléa structurel Il apparait lorsqu'un conflit de ressources ne peut pas être géré par le matériel. C'est le cas de la mémoire unique pour instructions et données. Dans ce cas, le pipeline est suspendu durant un cycle pour lever l'aléa.



Aléa de données Il apparait lorsqu'il y a dépendance de données entre les instructions. Soit par exemple les deux instructions

1. r2 ← mémoire(120)
2. r3 ← r2 + r1

Au moment où l'instruction 2 a besoin de l'information produite par l'instruction 1, l'instruction 2 est à l'étage DI et l'instruction 1 est à l'étage EX.

Aléa de contrôle Il concerne les instructions de modification du registre PC. Le cas trivial est l'instruction de branchement conditionnel. Jusqu'à l'exécution d'une telle instruction, il est impossible de savoir si un branchement est effectué ou non. Par exemple, pour la machine RISC, l'adresse de l'instruction suivante n'est connue qu'au niveau de l'étage EX.

cycle d'horloge	1	2	3	4	5	6	7	8
instruction 1	LI	DI	EX	MEM	ER			
instruction 2		LI	DI	EX	MEM	ER		
instruction 3			LI	DI	EX	MEM	ER	
instruction 4				LI	DI	EX		
instruction 5					LI	DI		
instruction 6						LI		
instruction 7							LI	DI
instruction 8								LI

! *Cet aléa est le principal facteur de dégradation de performance dans une architecture pipeline.*

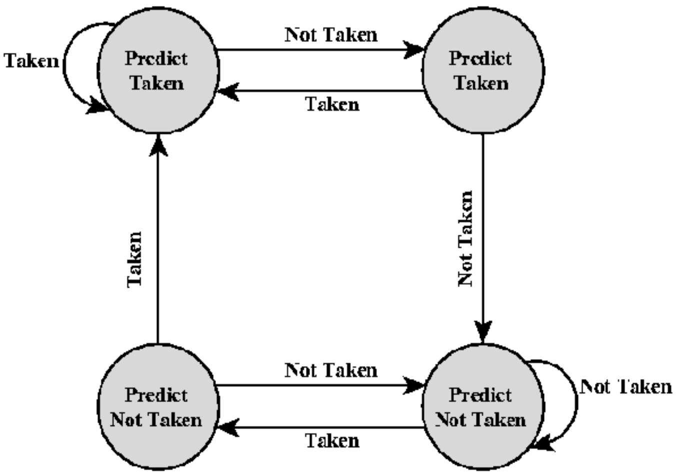
Plusieurs solutions ont été proposées pour gérer les branchements dans une architecture pipeline :
- dupliquer l'architecture de pipeline pour traiter les deux cas du branchement (pris ou pas) ;
- précharger l'instruction (ou la suite d'instruction) correspondant à l'adresse de branchement (quitte à ne pas l'utiliser) ;
- se baser sur une prédiction des branchements
  - supposer qu'un branchement ne sera jamais/toujours pris ;
  - supposer que certains opcodes favorisent le branchement ;
  - se baser sur un historique des branchements ;
- générer des instruction NOP (No OPeration) après l'instructions de branchement le temps que le branchement puisse se conclure.

8.6 Améliorations

8.6.1 Architecture superscalaire

Les architecture superscalaires disposent de plusieurs unités fonctionnelles (unités pour les opérations entières, unités pour les opérations flottantes, ...). Ainsi les instructions les plus communes (arithmétiques, chargement, rangement, branchement) peuvent être exécutés simultanément et indépendament dans différents pipelines. De plus, les instructions peuvent être exécutées dans un ordre différent de l'ordre du programme.

Par exemple, si on exécute la séquence d'instructions
- DIV x,y



- ADDF z,w
- SUB u,v

les instructions se termineront dans l'ordre SUB, ADDF et DIV.

8.6.2 Architecture VLIW

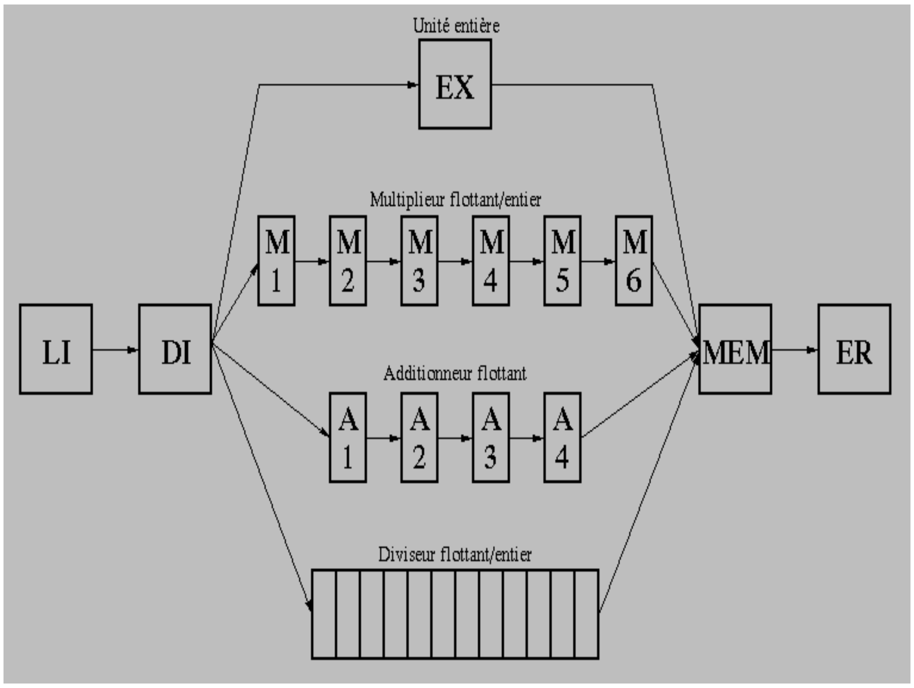
Le séquencement et l'ordonancement des opérations sont réalisés par le compilateur. Une instruction est composée de plusieurs champs, chaque champs commande une opération sur une unité de la machine. L'instruction peut être très longue, d'où le nom de Very Long Instruction Word (jusqu'à 1024 bits).
Imaginons une machine VLIW disposant

- d'une mémoire qui permet deux opérations simultanées (lecture ou écriture)
- un additionneur
- un soustracteur
- un multiplieur
- un diviseur

Les instructions d'une telle machine pourront comporter 6 champs : 2 pour les opérations de chargement/rangement en mémoire et 4 pour les opérations arithémtiques.
Soit par exemple le programme à exécuter :

- A = J + K
- B = L × M
- C = N - B
- D = A / B

Les instructions VLIW pourront être :



LOAD Rj,J	LOAD Rk,K				
LOAD Rl,L	LOAD Rm,M	Ra = Rj + Rk			
STORE Ra,A	LOAD Rn,N			Rb = Rl × Rm	
STORE Rb,B			Rc = Rn - Rb		Rd = Ra / Rb
STORE Rc,C	STORE Rd,D				

Ce qui fait 5 instructions pouvant s'exécuter chacune en 1 cycle d'horloge. Ce principe est adopté dans la nouvelle architecture 64 bits développée par Intel et HP.

Chapitre 9

Unité de commande

9.1 Introduction

Ce cours examine les opérations et l'organisation de l'unité de commande. Cette unité régit l'ordinateur en se basant seulement sur les instructions à exécuter et la nature du résultat des opérations arithémtiques et logiques. Elle ne prend en considération ni les données traitées, ni les résultats produits. Et elle contrôle la machine seulement via quelques signaux de contrôle.
Ce cours poursuit l'analyse descendante entreprise depuis le cours sur le jeu d'instruction.

9.2 Micro-opérations

L'exécution d'un programme consiste en l'exécution séquentielle d'instructions. Chaque instruction est exécutée durant un cycle d'instruction. Un cycle d'instruction se décompose en plusieurs phases (fetch, decode, ...). Chaque phase du cycle d'instruction est constitué d'une ou plusieurs étapes simples appelées *micro-opérations*.
Les micro-opérations sont les opérations fonctionnelles atomiques du processeur. Elles peuvent être regroupées pour s'exécuter en parallèle à condition

- de respecter la dépendance des données
- d'éviter les conflits

Ce concept est expliqué sur un exemple simple.
Supposons que nous disposons des registres PC, IR, MAR et MBR décrit au cours précédent. Le cycle de l'instruction comprend les cycles de fetch, indirect, execute et interrupt.
Commençons par détailler les cycles prévisibles. Chaque cycle implique une petite séquence fixée de micro-opérations.

Fetch Le cycle fetch consiste en 3 étapes et comprend 4 micro-opérations :

t1 MAR $\leftarrow$ PC
t2 MBR $\leftarrow$ mémoire
PC $\leftarrow$ PC + T
t3 IR $\leftarrow$ MBR

où T est la taille d'une instruction et les ti représentent des unités de temps de valeur égale (le compteur ordinal aurait pu être incrémenté lors de t3).

Indirect Pour ce cycle, on suppose un format une adresse. En cas de cycle indirect, le contenu de IR est directement affecté pour finalement contenir une adresse directe.

t1 MAR $\leftarrow$ IR(adresse)
t2 MBR $\leftarrow$ mémoire
t3 IR(adresse) $\leftarrow$ MBR(adresse)

où IR(adresse) identifie le champs adresse du registre IR. Le registre IR est maintenant dans le même état que si l'adressage indirect n'avait pas été utilisé.

Interrupt Après le cycle execute, un test est effectué pour savoir si une interruption est survenue. Si oui, un cycle d'interruption est déclenché, qui peut comprendre les micro-opérations suivantes :

t1 MBR $\leftarrow$ PC
t2 MAR $\leftarrow$ adresse de sauvegarde
PC $\leftarrow$ adresse de routine d'interruption
t3 mémoire $\leftarrow$ MBR

Execute Le cycle execute n'est pas prévisible. Pour une machine avec n différents opcodes, n différentes séquences de micro-opérations peuvent se produire.
Par exemple, considérons l'instruction ADD R1,X. Elle peut correspondre à la séquence de micro-opérations suivante :

t1 MAR $\leftarrow$ IR(adresse)
t2 MBR $\leftarrow$ mémoire
t3 R1 $\leftarrow$ R1 + MBR

Considérons maintenant deux exemples plus complexes. Le premier concerne l'instruction ISZ X, qui incrémente X de 1 et qui saute l'instruction suivante si le résultat est 0. Elle peut correspondre à la séquence de micro-opérations suivante :

t1 MAR $\leftarrow$ IR(adresse)
t2 MBR $\leftarrow$ mémoire
t3 MBR $\leftarrow$ MBR + 1
t4 mémoire $\leftarrow$ MBR
si MBR = 0 alors PC $\leftarrow$ PC + T

où T est la taille d'une instruction. Pour l'instruction BSA X, qui sauvegarde l'adresse de l'instruction suivante dans X et demande de poursuivre l'exécution par l'instruction située à l'adresse X + T, on a :

t1 MAR $\leftarrow$ IR(adresse)
MBR $\leftarrow$ PC
t2 PC $\leftarrow$ IR(adresse)
mémoire $\leftarrow$ MBR
t3 PC $\leftarrow$ PC + T

où T est la taille de l'instruction. Cette instruction est typiquement utilisée pour les appels de sous-routines.
On voit bien qu'une séquence de micro-opérations est nécessaire pour chaque opcode.

Le cycle d'instruction Pour coordonner les séquences de micro-opérations entre elles, un nouveau registre est nécessaire qui contiendra l'état du processeur en terme de cycle. Appelons ce registre ICC (Instruction Cycle Code). Dans notre exemple, seuls 2 bits composent ce registre :

00	fetch
01	indirect
10	execute
11	interrupt

Le registre ICC est modifié en fin de chacun des 4 cycles.

9.3 Contrôle du processeur

Nous nous intéressons maintenant aux activités à implanter dans l'unité de commande pour exécuter les séquences de micro-opérations. L'unité de commande doit exécuter 2 tâches basiques :

- le séquencement des micro-opérations
- l'exécution des micro-opérations

Ces micro-opérations appartiennent toutes aux catégories suivantes :

- transfert de données entre registres
- transfert de données d'un registre vers une interface externe (e.g., bus)
- transfert de données d'une interface externe (e.g., bus) vers un registre
- opération sur l'ALU en utilisant des registres pour les opérandes sources et le résultat

Interactions entre l'unité de commande et l'extérieur Les entrées de l'unité de commande sont

- l'horloge : une micro-opération s'exécute en un top d'horloge ;
- le registre d'instruction : l'opcode permet de déterminer quelle séquence de micro-opérations doit être exécutée lors d'un cycle exécute ;
- le registre de condition : utilisé pour déterminer le statut du processeur et pour les instructions conditionnelles ;
- des signaux de contrôle du bus de contrôle : par exemple pour détecter une interruption ou recevoir les acquittements des unités externes.

Les sorties de l'unité de commande sont

- des signaux de contrôle internes au processeur, de 2 types :
  - des signaux déclenchant des transferts entre registre
  - des signaux déclenchant des opérations sur l'ALU
- des signaux de contrôle transportés par le bus de contrôle, de 2 types :
  - à destination de la mémoire
  - à destination des modules d'E/S

Exemple Ainsi, pendant un fetch, les signaux de contrôle envoyés seront les suivants :

- un signal autorisant le chargement de MAR par le contenu de PC
- simultanément
  - un signal plaçant le contenu du MAR sur le bus d'adresse

  - l'activation d'une ligne lecture mémoire du bus de contrôle
  - un signal autorisant le chargement de MBR par le bus de données
  - un signal déclenchant une opération incrémentant le PC (ce qui peut être fait indépendament de l'ALU si on considère une logique dédié)
- un signal autorisant le chargement de IR par MBR

Ensuite, le contenu de IR est analysé pour savoir quel cycle doit suivre (indirect ou execute). Les cycles indirect et interrupt fonctionnent de manière similaire. Nous verrons le cycle execute plus tard...

cycle	timing	signal de contrôle
fetch	t1 : MAR ← PC	C_2
	t2 : MBR ← mémoire	C_5, C_r
	PC ← PC + T	
	t3 :IR ← MBR	C_4
indirect	t1 : MAR ← IR(adresse)	C_8
	t2 : MBR ← mémoire	C_5, C_r
	t3 : IR(adresse) ← MBR(adresse)	C_4
interrupt	t1 : MBR ← PC	C_1
	t2 : MAR ← adresse de sauvegarde	
	PC ← adresse de routine d'interruption	
	t3 : mémoire ← MBR	C_{12}, C_w

Organisation L'organisation typique de la CPU comprend un bus interne (ou un ensemble de bus internes) pour la circulation interne des différentes informations (données, instructions, adresses). Ce bus relie tous les registres internes et l'ALU. Chaque signal de contrôle interne émis par l'unité de commande (exceptés les signaux déclanchant des opérations sur l'ALU) contrôle en fait l'accès à ce bus.

9.4 Implantation de l'unité de commande

Les techniques d'implantation de l'unité de commande se partage en 2 catégories :
- implantation matérielle
- implantation micro-programmée

9.4.1 Implantation matérielle

Dans ce cas là, l'unité de commande est essentiellement un circuit combinatoire élaborant les signaux de contrôle.
L'opcode passe par un décodeur dont chaque sortie correspond à une séquence de mirco-opérations particulière, donc à une séquence de signaux de contrôle particulière.
Le signal d'horloge sert à alimenter un compteur donnant chaque période ti. Ce compteur est réinitialisé en fin de chaque cycle.

La logique utilisée correspond aux équations booléennes des signaux de sorties en fonction du cycle, de la valeur délivrée par le compteur et de l'opcode, c'est à dire de la valeur fournie par le décodeur.

9.4.2 Implantation micro-programmée

Dans un processeur moderne beacoup plus complexe que l'exemple qui a servi dans ce cours, le nombre d'équation booléenne nécessaires à l'implantation de l'unité de commande est prohibitif pour la construction d'un circuit combinatoire. La microprogrammation constitue une solution simple à ce problème. C'est une technique très commune dans les processeurs CISC contemporains.

Micro-instructions Les micro-opérations peuvent être considérées comme des opérations déclenchées par des instructions d'un langage. Ces instructions sont appelées micro-instructions et un programme à base de micro-instructions est appelé micro-programme. Une instruction et une micro-instruction partagent quelques caractéristiques communes :

- elles sont décomposées en champs
- elles sont rangées dans une mémoire à une adresse précise.

Un découpage en champs est par exemple :

- un mot de contrôle correspondant à l'activation des signaux binaires
  - 1 bit pour chaque ligne de contrôle interne
  - 1 bit pour chaque ligne de contrôle du bus de contrôle
- l'adresse de la micro-instruction à exécuter ensuite si une condition est remplie
- la condition de branchement

Une telle micro instruction est interprétée de la manière suivante :

1. déclencher la/les micro-opéations en positionnant les signaux de contrôle en fonction du mot de contrôle (un 1 active un signal, un 0 n'active pas ou désactive un signal)
2. si la condition indiquée par les bits de condition est fausse alors exécuter la micro-instruction à l'adresse suivante
3. si la condition indiquée par les bits de condition est vraie alors exécuter la micro-instruction dont l'adresse est mentionnée dans le champs adresse.

Les micro-instructions sont organisées en séquences dans une mémoire de contrôle. Chaque séquence définit une routine correspondant à

- un sous-cycle du cycle d'instruction
- un opcode pour le cycle exécute

⋮ ⋮ saut vers indirect ou execute	routine du cycle fetch
⋮ ⋮ saut vers execute	routine du cycle indirect
⋮ ⋮ saut vers fetch	routine du cycle interrupt
saut vers routine d'opcode	routine du cycle execute
⋮ ⋮ saut vers fetch ou interrupt	routine ADD
⋮ ⋮ saut vers fetch ou interrupt	routine AND
⋮	⋮
⋮ ⋮ saut vers fetch ou interrupt	routine SUB

Réalisation Implanter une unité de commande micro-programmée revient simplement à exécuter le programme de la mémoire de contrôle. Par analogie à une CPU, ceci requiert :

- la mémoire de contrôle stockant les micro-instructions
- un registre d'adresse de contrôle contenant l'adresse de la prochaine micro-instruction à lire
- un registre de contrôle contenant la micro-instruction. La partie de ce registre contenant le mot de contrôle est directement reliée aux lignes correspondant aux signaux de contrôle
- un séquenceur qui charge le register d'adresse de contrôle et envoie des ordres de lecture à la mémoire de contrôle.

! *Lire une instruction revient à exécuter cette instruction.*

Ainsi, en un top d'horloge, l'unité de commande fonctionne de la manière suivante :

1. le séquenceur envoie une commande de lecture à la mémoire de contrôle

2. la micro-instruction dont l'adresse est spécifiée par l'adresse contenue dans le registre d'adresse de contrôle est chargée dans le registre de contrôle
3. le contenu du registre de contrôle permet de générer les signaux de contrôle et donne l'adresse de la micro-instruction suivante au séquenceur
4. le séquenceur charge une nouvelle adresse dans le registre d'adresse de contrôle en fonction de l'adresse délivrée par le registre de contrôle et des conditions provenant de l'ALU. Cette adresse peut être
   - l'adresse courante + 1
   - l'adresse du registre de contrôle
   - l'adresse d'une routine correspondant à l'opcode du registre d'instruction