MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU

FACULTE DE GENIE ELECTRIQUE ET D'INFORMATIQUE
DEPARTEMENT D'AUTOMATIQUE

Mémoire de fin d'études

En vue de l'obtention du Diplôme de :

Master Académique
Spécialité : Génie Microélectronique

Thème

Conception d'un microprocesseur à 4 bits

Présenté par :
Melle: Kebaili Souad

Proposé et Dirigé par :
Mr: Lakhlef Ahcene

Remerciements

Je remercie vivement mon encadreur Monsieur Lakhlef Ahcene, d'avoir proposé et dirigé ce travail de ci-près, avec une qualité remarquable, j'ai grandement apprécié ses compétences, sa disponibilité et le suivi régulier a l'avancement de mon travail.

Mes sincères remerciement a tous les enseignants qui m'ont transmis les bases de la micro-électronique tout au long de ces deux années de master.

Un merci particulier à ma famille pour leur encouragement et leur soutien tout au long de ce travail.

Un grand merci à mon collègue et mes amis (es) avec qui j'ai passé de très bons moments au sein de l'université Mouloud Mammeri.

Que ceux qui se sentent oubliés trouvent ici ma profonde gratitude et les chaleureux remerciements pour leur concours dans l'accomplissement de ce travail.

Sommaire

Liste des figures

Introduction Générale

Parmi la multitude de composants d'électronique numérique, les processeurs restent les composants les plus fascinants, en particulier par leur aspect omnipotent. En effet, un processeur peut être vu comme une machine universelle pouvant effectuer n'importe quelle tâche ne dépendant que du programme qu'on lui demande d'exécuter : c'est l'interface entre le monde virtuel (les programmes et les actions qu'on leur demande d'effectuer) et le monde réel (la création et l'envoi du signal électrique vers un écran par exemple). C'est un peu la raison pour laquelle nous nous sommes tourné vers le choix d'un tel composant.

L'arrivée du microprocesseur en 1970 dans la société a permis de nombreux changements. Il a nettement révolutionné le domaine des technologies et de l'informatique. Le microprocesseur, noté **M.P.U**. (Micro processor unit) ou C.P.U. (Central Processor Unit) est un circuit intégré complexe appartenant à la famille des **VLSI** (Very large scale intégration) capable d'effectuer séquentiellement et automatiquement des suites d'opérations élémentaires, c'est l'élément de l'ordinateur, il interprète et exécute aussi les instructions d'un programme.

Des systèmes de développement sont mis en œuvre pour développer les microprocesseurs. Ils sont faits essentiellement par FPGA.

Ainsi, notre travail consiste à concevoir un microprocesseur à 4 bits avec le langage de description VHDL (VHSIC Hardware Description Language).

Le travail réalisé est décomposé en 3 chapitres : le premier est dédié à l'architecture des microprocesseurs, dans le deuxième chapitre on s'est intéressé aux différents éléments combinatoires et séquentiels constituants le microprocesseur, le troisième chapitre est consacré à la conception du microprocesseur et d'une conclusion générale et perspectives.

Chapitre 1 :
Architecture des microprocesseurs

1. Introduction

Le microprocesseur est le composant principal et essentiel de chaque carte mère.il est le cœur des systèmes automatiques, C'est lui qui gère et contrôle la totalité des procédures et des enchaînements, exécute les instructions des programmes. Sans lui, un fonctionnement opérationnel et interactif des autres composants n'est pas possible, il est relié directement ou indirectement à tous les autres composants de la carte mère. La structure interne du microprocesseur s'est considérablement développée au cours de l'évolution du PC, dans ce chapitre nous allons étudier l'architecture du microprocesseur et son fonctionnement.

2. Architecture du microprocesseur

Un microprocesseur est construit autour de deux éléments principaux :

- Une unité de commande : appelé aussi Unité de commande et de contrôle (UCC)
- Une unité de traitement

Associés à des registres chargées de stocker les différentes informations à traiter. Ces trois éléments sont reliés entre eux par des bus interne permettant les échanges d'informations comme la figure (Fig. I.1) le montre.

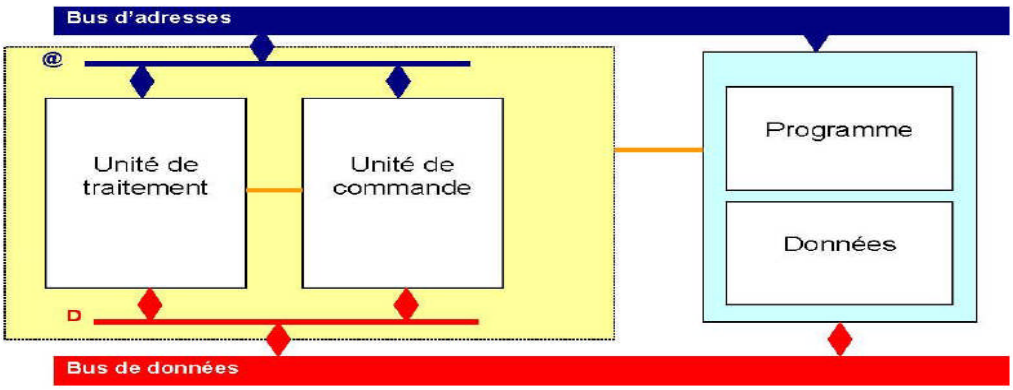


Figure I.1 : Eléments principaux constituant un microprocesseur

2.1. L'unité de commande

Elle permet de séquencer le déroulement des instructions. Elle effectue la recherche en mémoire de l'instruction. Comme chaque instruction est codée sous forme binaire, elle en

assure le décodage pour enfin réaliser son exécution puis effectue la préparation de l'instruction suivante. Pour cela, elle est composée par :

- **Le compteur de programme** : (en anglais Program Counter PC) appelé aussi compteur ordinal (CO). Le CO est constitué par un registre dont le contenu représente l'adresse de la prochaine instruction à exécuter. Il est donc initialisé avec l'adresse de la première instruction du programme. Puis il sera incrémenté automatiquement pour pointer vers la prochaine instruction à exécuter.
- **Le registre d'instruction et le décodeur d'instruction :** chacune des instructions à exécuter est rangée dans le registre instruction puis est décodée par le décodeur d'instruction.
- **Bloc logique de commande (ou séquenceur) *:*** Il organise l'exécution des instructions au rythme d'une horloge. Il élabore tous les signaux de synchronisation internes ou externes (bus de commande) du microprocesseur en fonction des divers signaux de commande provenant du décodeur d'instruction ou du registre d'état par exemple. Il s'agit d'un automate réalisé soit de façon câblée (obsolète), soit de façon microprogrammée, on parle alors de micromicroprocesseur.

Exemple :

Pour un microprocesseur ayant un format d'instructions à deux adresses, l'instruction d'addition de deux variables en mémoire A et B est représenté dans le registre d'instructions comme suit :

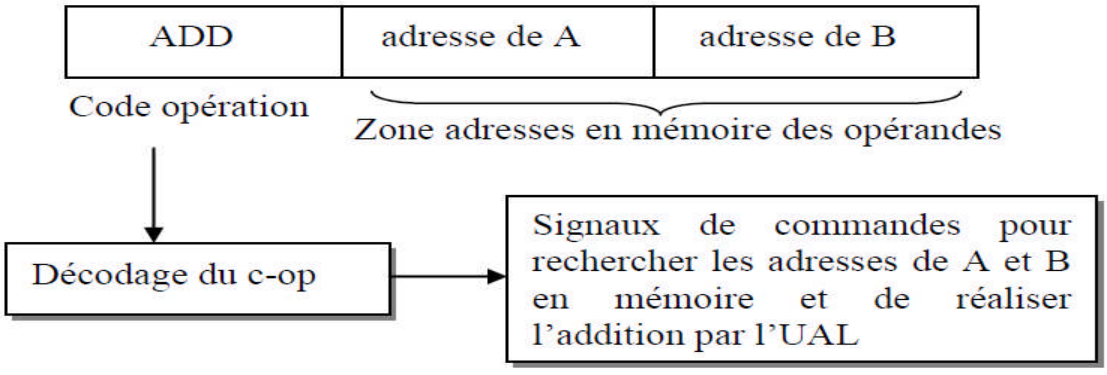


Figure I.2 : exemple d'addition pour le décodage de l'instruction

2.2. L’unité de traitement

C’est le cœur du microprocesseur. Elle regroupe les circuits qui assurent les traitements nécessaires à l'exécution des instructions. L’unité de traitement est composée de trois principales unités d’exécution, la première est l’unité arithmétique et logique (UAL) puis deux autres ont été ajoutés qui sont l’unité de calcul en virgule flottante et l’unité multimédia pour des raisons d’optimisation des performances des microprocesseurs.

2.2.1. Unité arithmétique et logique (UAL)

Elle est composée de circuits logiques tels que les additionneurs, soustracteurs, comparateurs logiques…etc., afin d’effectuer les calculs et les opérations logiques des différents instructions à exécuter, les données à traiter se présentent aux entrées de l’UAL, sont traités, puis le résultat est fourni en sortie et généralement stocké dans un registre dit accumulateur. Les informations qui concernent l’opération sont envoyées vers le registre d’état.

Le schéma suivant montre l’UAL ainsi que ses entrées et ses sorties. (Fig. I.3.)

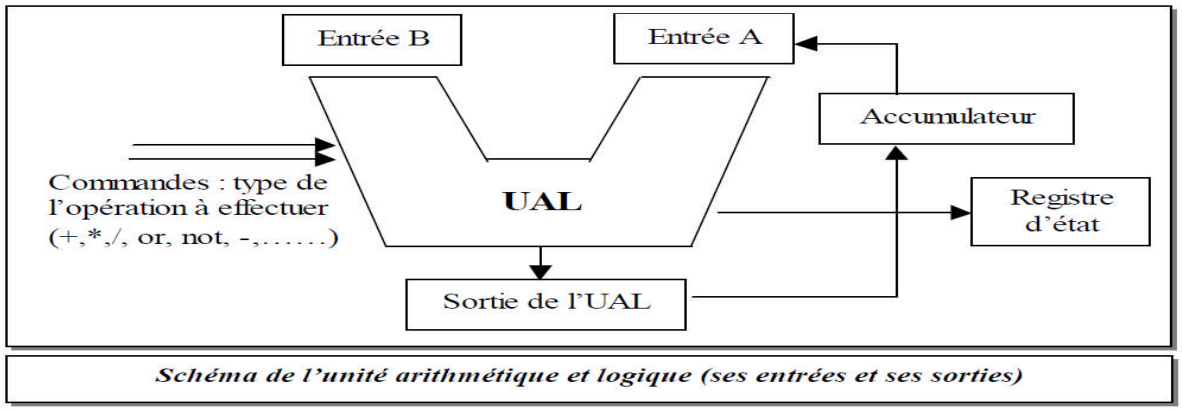


Figure I.3 : schéma d’unité arithmétique et logique (UAL)

- **Rôle de l'UAL**

Comme son nom l'indique, cette unité peut exécuter 2 types d'opérations.

- **Opérations arithmétiques :**

Elles incluent l'**addition** et la **soustraction** qui sont des opérations de base (une soustraction est une addition avec le complément à deux), la **multiplication** et la **division**.

Les données traitées sont considérées dans des représentations "entières".

- **Opérations logiques :**

Ces opérations sont effectuées **bit à bit** sur les bits de même poids de 2 mots

On trouve : ET, OU, OU exclusif…

Un autre type d'opérations concerne les **décalages** et **rotations** (droite et gauche). Ces différentes opérations diffèrent par la façon d'entrer et de sortir les bits de poids extrêmes

Elles sont utilisées par exemple pour effectuer la multiplication et la division à partir de l'addition et la soustraction.

Après chaque opération effectuée dans l'ALU, le registre d'état contient un certain nombre d'informations liées à leur exécution. On trouve généralement : **signe** du résultat, **résultat nul**, **dépassement** de capacité (retenue ou Carry (C)), etc…

2.2.2. Unité de calcul en virgule flottante

C'est une unité qui est capable de réaliser les opérations de calcul pour les réels ainsi que les calculs mathématiques et scientifiques complexes.

A l'origine la tâche de cette unité était réalisée par tout un processeur à part, en 1989 elle a été intégré dans les microprocesseurs afin d'optimiser les calculs.

2.2.3. Unité multimédia

C'est une unité qui est chargée d'accélérer l'exécution des programmes multimédia comportant des vidéos, du son, graphisme en 3D etc.…

Cette unité porte le nom de MMX pour les premiers pentiums (MultiMedia extensions) intégrants des fonctions de gestion du multimédia, de même la technologie 3DNOW pour les AMD et SSE pour les pentiums III.

Ces unités (Unité de calcul en virgule flottante, Unité multimédia) ont été créées vu la grande tendance vers la multimédia dans tous les types des programmes informatiques (jeux, logiciels sur Internet, encyclopédies…) .

La figure suivante montre le Schéma fonctionnel du microprocesseur : (Fig. I.4)

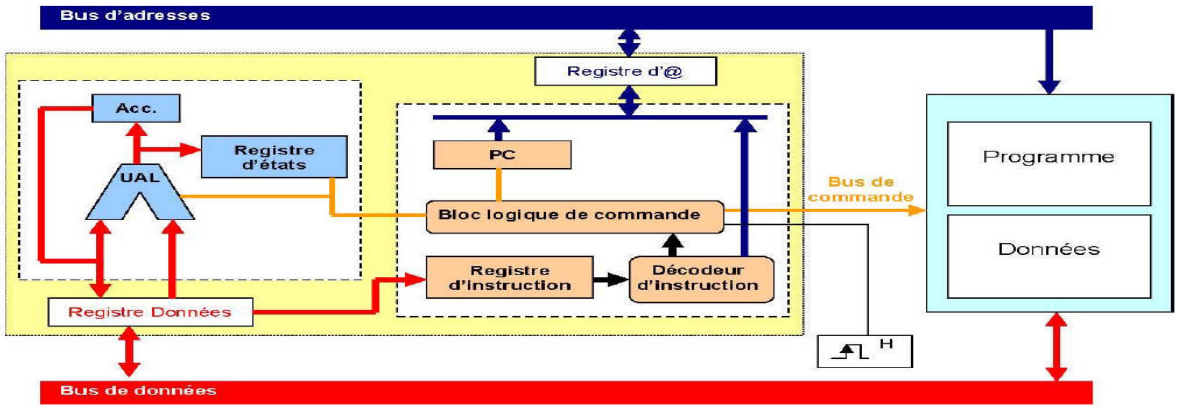


Figure I.4 : Schéma fonctionnel du microprocesseur

2.3. Les registres du microprocesseur

Un registre est une zone mémoire à l'intérieur du microprocesseur de faible taille, qui permet de mémoriser des mots mémoires ou des adresses d'une façon temporaire lors de l'exécution des instructions.

Le nombre et le type de registres que possède le CPU sont une partie déterminante de son architecture et ont une influence importante sur la programmation. La structure des registres du CPU varie considérablement d'un constructeur à l'autre. Cependant les fonctions de base réalisées par les différents registres sont essentiellement les mêmes. Nous allons décrire les registres les plus importants, leur fonction et la façon dont ils peuvent être modifiés par programme.

Il existe deux types de registres : les registres généraux, et les registres d'adresses :

- ***Les registres généraux* :**

Ce sont des mémoires rapides, à l'intérieur du microprocesseur, qui permettent à l'unité d'exécution de manipuler des données à vitesse élevée. Ils sont connectés au bus données interne au microprocesseur. L'adresse d'un registre est associée à son nom (on donne généralement comme nom une lettre) A, B, C...

Ces registres permettent de sauvegarder des informations utilisées dans les programmes ou des résultats intermédiaires, cela évite des accès à la mémoire, accélérant ainsi l'exécution des programmes.

Les registres généraux sont à la disposition du programmeur qui a normalement un choix d'instructions permettant de les manipuler comme

- Chargement d’un registre à partir de la mémoire ou d’un autre registre.
- Enregistrement dans la mémoire du contenu d’un registre.
- Transfert du contenu d’un registre dans l’accumulateur et vice versa.
- Incrémentation ou décrémentation d’un registre.

➢ **Les registres d'adresses (pointeurs)**

Ce sont des registres connectés sur le bus adresses, leur contenu est une adresse en mémoire centrale. Ils existent plusieurs types On peut citer comme registres:

- Le compteur ordinal (pointeur de programme PC)
- Le pointeur de pile (Stack pointer SP)
- Les registres d'index (Index source SI et index destination DI)

2.3.1. Le compteur ordinal (pointeur de programme PC)

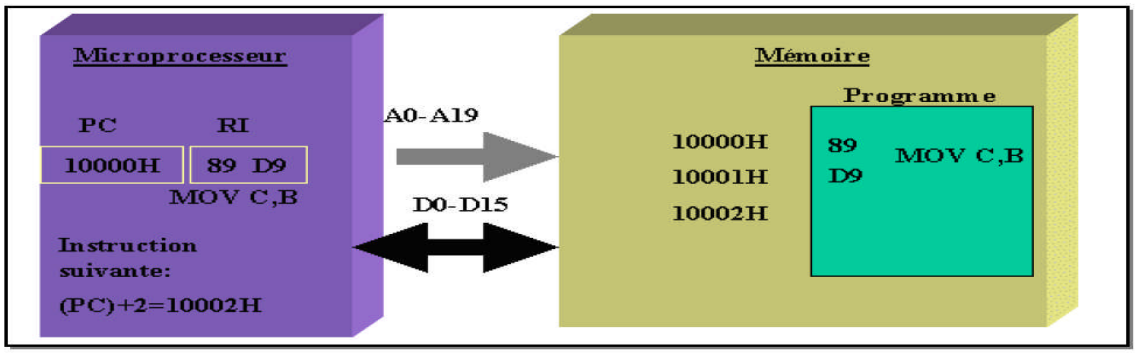


Figure I.5 : Schéma descriptif du fonctionnement du registre compteur ordinal

Il contient l'adresse de l'instruction à rechercher en mémoire. L'unité de commande incrémente le compteur ordinal (PC) du nombre d'octets sur lequel l'instruction, en cours d'exécution, est codée. Le compteur ordinal contiendra alors l'adresse de l'instruction suivante.

Exemple :

(PC)=10000H ; il pointe la mémoire qui contient l'instruction MOV C,B qui est codée sur deux octets (89 D9H) ; l'unité de commande incrémentera de deux le contenu du PC : (PC) = 10002H

(la mémoire sera supposée être organisée en octets).

2.3.2. Le registre d’instruction

Contient l’instruction qui doit être traitée par le microprocesseur, cette instruction est recherchée en mémoire puis placée dans ce registre pour être décodée par le décodeur et préparée pour l’exécution.

Une instruction est une opération élémentaire d’un langage de programmation, c’est à dire le plus petit ordre que peut comprendre un ordinateur.

2.3.3. Le registre d’adresse

C’est un registre qui contient l’adresse du mot à accéder en mémoire centrale. A chaque accès mémoire, l’adresse recherchée est stockée dans ce registre. Il a la taille d’une adresse qui est la même que celle du bus d’adresses ce qui permet de déterminer le nombre de mots mémoires adressables et l’espace mémoire adressable.

2.3.4. Le registre mot mémoire ou registre de données

Contient le mot mémoire faisant objet d’une opération de lecture ou d’écriture dans la mémoire centrale. Ce registre a la taille d’un mot mémoire qui est la même que celle des registres de travail et l’accumulateur qui est égale à la taille du bus de données.

2.3.5. Le registre accumulateur

C’est un registre de travail très important de l’UAL, dans la plus part des opérations arithmétiques et logiques, l’accumulateur contient un des opérandes avant l’exécution et le résultat après. L’accumulateur a la même taille qu’un mot mémoire.

Naturellement, le programmeur a accès à ce registre qui est toujours très sollicité pendant le traitement des données.

2.3.6. Le registre d’état (PSW program status word)

C’est un registre qui contient les différents bits appelés drapeaux (Flags) indiquant l’état d’une condition particulière dans le CPU. Ces bits peuvent être testés par un programme et donc décider des suites d’actions à prendre.

Pour exécuter correctement son travail, le séquenceur doit en outre connaître l’état d’un certain nombre d’autres composants et disposer d’informations concernant la ou les opérations qui ont déjà été exécutées (par exemple, doit on tenir compte dans l’addition en cours d’une éventuelle retenu préalable générée par une addition précédente). Le registre d’état fournit ces informations.

Les différents bits du registre d'état sont :

- ***Bit de retenu*** : c'est le bit C ou carry. ce bit est à 1 s'il y a une retenu qui est générée lors d'une opération arithmétique.
- ***Bit de parité*** : ce bit est mis à 1 si le nombre de 1 dans l'accumulateur est pair.
- ***Le bit zéro*** : il est à 1 si le résultat d'une opération arithmétique est nul.
- ***Le bit de signe*** : il est à 1 si le résultat d'une opération arithmétique est négatif.
- ***Le bit de dépassement de capacité*** : le bit n°11, O pour over flow, il est à 1 s'il y a dépassement de capacité dans les opérations arithmétiques.

2.3.7. Le registre pointeur de pile : (Stack Pointer)

- Il contient l'adresse de la pile. Celle-ci est une partie de la mémoire, elle permet de stocker des informations (le contenu des registres) relatives au traitement des interruptions et des sous-programmes
- La pile est gérée en **LIFO** : (Last IN First Out) dernier entré premier sorti. Le fonctionnement est identique à une pile d'assiette
- Le pointeur de pile SP pointe le haut de la pile, il est décrémenté avant chaque empilement, et incrémenté après chaque dépilement.
- Il existe deux instructions pour empiler et dépiler : PUSH et POP.

2.3.8. Les registres d'index (index source SI et index destination DI)

Les registres d'index peuvent être utilisés comme des registres généraux pour sauvegarder et pour compter. Mais en plus ils ont une fonction spéciale qui est de grande utilité dans la manipulation des tableaux de données.

Ils peuvent en effet être utilisés pour manipuler des adresses et permettent de mémoriser une adresse, suivant une forme particulière d'adressage, appelée adressage indexée.

Exemple :

MOV A,[SI+10000H] place le contenu de la mémoire d'adresse 10000H+le contenu de SI, dans le registre A.

2.4. Les interfaces d’entrées/sorties

Elles permettent d’assurer la communication entre le microprocesseur et les périphériques. (Capteur, clavier, moniteur ou afficheur, imprimante, modem, etc…).

2.5. Les bus

Un bus est un ensemble de fils qui assure la transmission du même type d’information. On retrouve trois types de bus véhiculant des informations en parallèle dans un système de traitement programmé de l’information :

- **Un bus de données :** bidirectionnel qui assure le transfert des informations entre le microprocesseur et son environnement, et inversement. Son nombre de lignes est égal à la capacité de traitement du microprocesseur.

- **Un bus d'adresses** : unidirectionnel qui permet la sélection des informations à traiter dans un espace mémoire (ou espace adressable) qui peut avoir **2n** emplacements, avec **n** = nombre de lignes du bus d'adresses.

- **Un bus de commande :** constitué par quelques conducteurs qui assurent la synchronisation des flux d'informations sur les bus des données et des adresses.

2.6. Les mémoires

2.6.1. Définitions

Une mémoire est un ensemble de **cellules** élémentaires qui stockent chacune un bit. Le nombre de cellules est appelé "**capacité**" de la mémoire. A chaque cellule sont associées 2 informations (Fig. I.11).

- Une **adresse** (numéro) permettant de désigner la cellule,
- Une **donnée** représentée par son état.

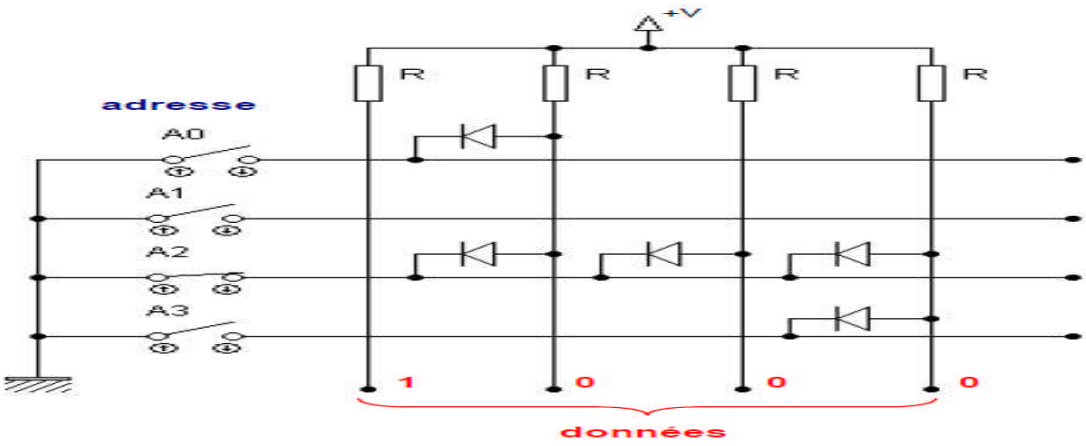


Figure I.6 : schéma de la mémoire

La possibilité d'accéder à une cellule quelconque en fournissant son adresse est appelée "**accès aléatoire**".

On distingue un mode de **lecture** permettant de connaître l'état de la cellule sélectionnée et un mode d'**écriture** permettant d'imposer son état. Ces 2 modes sont définis par l'état d'une ligne de **lecture-écriture** (Read/Write : R/W).

La mémoire se décompose souvent en :

- Une mémoire morte (*ROM = Read Only Memory*) chargée de stocker le programme. C'est une mémoire à lecture seule.
- Une mémoire vive (*RAM = Random Access Memory*) chargée de stocker les données intermédiaires ou les résultats de calculs. On peut lire ou écrire des données dedans, ces données sont perdues à la mise hors tension.

Remarque : Les disques durs, disquettes, CDROM, etc… sont des périphériques de stockage et sont considérés comme des mémoires secondaires.

2.6.2. Différentes types des mémoires

- **Les mémoires vives (RAM)**

Une mémoire vive sert au stockage temporaire de données. Elle doit avoir un temps de cycle très court pour ne pas ralentir le microprocesseur. Les mémoires vives sont en général volatiles, elles perdent leurs informations en cas de coupure d'alimentation.

Il existe deux grandes familles de mémoires RAM (Random Acces Memory : mémoire à accès aléatoire)

- Les RAM statiques

- Les RAM dynamiques

- **Les RAM statiques**

Chaque cellule est constituée par un élément **bistable .**

Une fois mise dans un certain état par l'opération d'écriture, la cellule reste dans cet état jusqu'à une écriture de l'état opposé ou jusqu'à disparition de la tension d'alimentation.

- **Les RAM dynamiques**

Dans les RAM dynamiques (DRAM), l'information est mémorisée sous la forme d'une charge électrique stockée dans un condensateur. Cette technique permet une plus grande densité d'intégration, car un point mémoire nécessite environ quatre fois moins de transistors que dans une mémoire statique. Sa consommation s'en retrouve donc aussi très réduite.

La présence de courants de fuite dans le condensateur contribue à sa décharge. Ainsi, l'information est perdue si on ne la régénère pas périodiquement (charge du condensateur).

Les RAM dynamiques doivent donc être rafraîchies régulièrement pour entretenir la mémorisation : il s'agit de lire l'information et de la recharger.

- **Les mémoires mortes (ROM)**

Pour certaines applications, il est nécessaire de pouvoir conserver des informations de façon permanente même lorsque l'alimentation électrique est interrompue. On utilise alors des mémoires mortes ou mémoires à lecture seule (ROM: Read Only Memory). Ces mémoires sont non volatiles, et contrairement aux RAM, ne peuvent être que lue.

- **La ROM**

Elle est programmée par le fabricant et son contenu ne peut plus être ni modifiée, ni effacée par l'utilisateur.

- **Avantages**
  - Claquage en quelques minutes
  - Coût relativement faible
- **Inconvénients**
  - Modification impossible (toute erreur est fatale)

- **EPROM ou l'UV-EPROM**

Pour faciliter la mise au point d'un programme ou tout simplement permettre une erreur de programmation, il est intéressant de pouvoir reprogrammer une PROM. La technique de claquage utilisée dans celles-ci ne le permet évidemment pas. L'EPROM (Effaçable

Programmable ROM) est une PROM qui peut être effacée.

- **Avantages**
  - Reprogrammable et non Volatile.
- **Inconvenient :**
  - Impossible de sélectionner une seule cellule à effacer.
  - Impossible d'effacer la mémoire in situ.
  - l'écriture est beaucoup plus lente que sur une RAM (environ 1000x).

- **EEPROM**

L'EEPROM (Electically EPROM) est une mémoire programmable et effaçable électriquement. Elle répond ainsi à l'inconvénient principal de l'EPROM et peut être programmée in situ.

- **Avantages :**
  - Comportement d'une RAM non Volatile.
  - Programmation et effacement mot par mot possible.
- **Inconvénients :**
  - Très lente pour une utilisation en RAM.
  - Coût de réalisation.

- **FLASH EPROM**

La mémoire Flash s'apparente à la technologie de l'EEPROM. Elle est programmable et effaçable électriquement comme les EEPROM.

- **Avantages**
  - Comportement d'une RAM non Volatile.
  - Programmation et effacement mot par mot possible.
- **Inconvénients :**
  - Très lente pour une utilisation en RAM.
  - Coût de réalisation.

2.7. Etapes d'exécution d'une instruction

Le microprocesseur ne comprend qu'un certain nombre d'instructions qui sont codées en binaire. Le traitement d'une instruction peut être décomposé en trois phases.

***Phase 1*: Recherche de l'instruction à traiter**

1. Le PC contient l'adresse de l'instruction suivante du programme. Cette valeur est placée sur le bus d'adresses par l'unité de commande qui émet un ordre de lecture.

2. Au bout d'un certain temps (temps d'accès à la mémoire), le contenu de la case mémoire Sélectionnée est disponible sur le bus des données.

3. L'instruction est stockée dans le registre instruction du processeur.

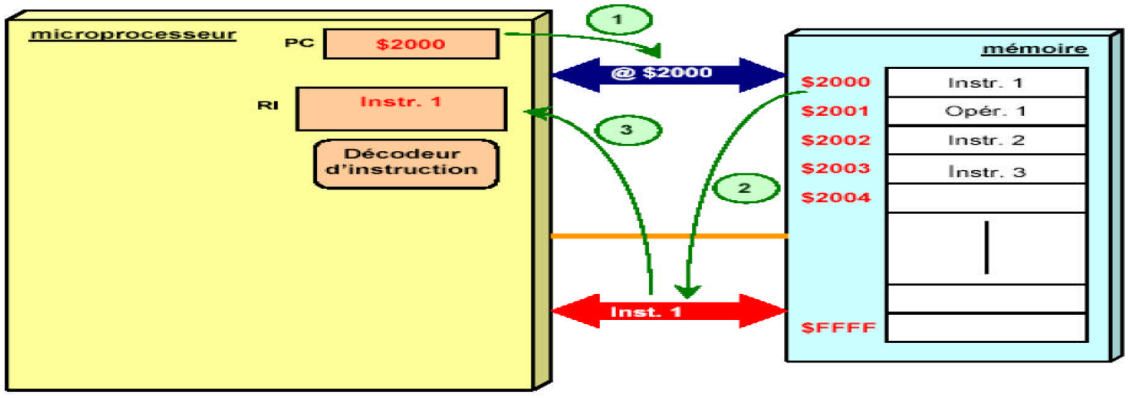


Figure I.7.a : Recherche de l'instruction à traiter

***Phase 2* : Décodage de l'instruction et recherche de l'opérande**

Le registre d'instruction contient maintenant le premier mot de l'instruction qui peut être codée sur plusieurs mots. Ce premier mot contient le code opératoire qui définit la nature de l'opération à effectuer (addition, rotation,...) et le nombre de mots de l'instruction.

1. L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.

2. Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur sur le bus de données.

3. L'opérande est stocké dans un registre.

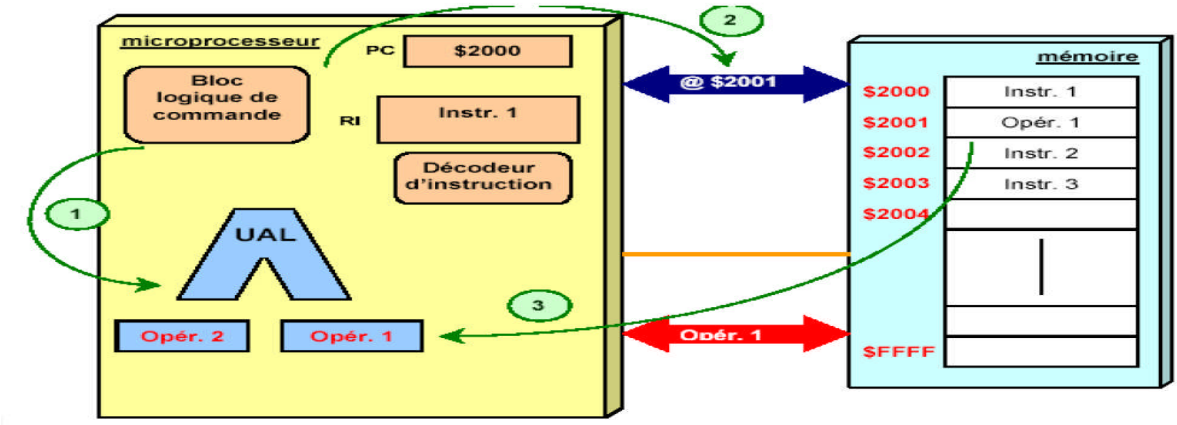


Figure I.7.b : Décodage de l'instruction et recherche de l'opérande

***Phase 3* : Exécution de l'instruction**

1. Le micro-programme réalisent l'instruction a exécuté.
2. Les drapeaux sont positionnés (*registre d'état*).
3. L'unité de commande positionne le PC pour l'instruction suivante.

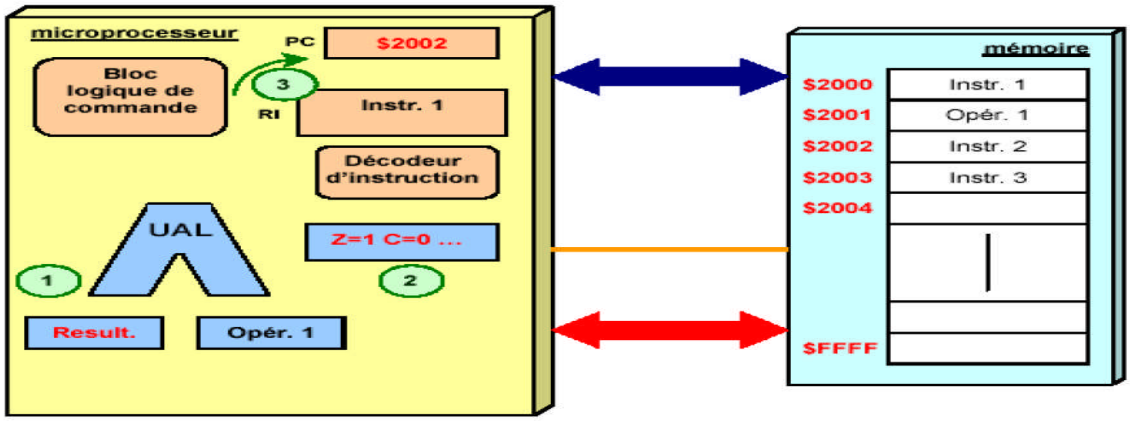


Figure I.7.c : Exécution de l'instruction

2.8. Mode d'adressage

Un mode d'adressage définit la manière dont le microprocesseur va accéder à l'opérande. Les différents modes d'adressage dépendent des microprocesseurs

Une opération se présente sous la forme générale suivante :

(Opérande source 1) F (Opérande source 2) --> Destination

On parle alors de "**machines à 3 adresses**".

Pour éviter de spécifier systématiquement ces 3 adresses à chaque opération, les opérandes sources ou/et destination sont contenus dans les registres internes de l'U.T. Ainsi, la plupart des U.T. spécifient une **seule adresse** selon :

(Registre N) F (Opérande) --> (Registre N)

Les façons de désigner les opérandes constituent les "**modes d'adressage**".

On rencontre principalement les modes suivants :

- **Adressage implicite :**

Ici les opérandes sont désignés dans le code-opération ; les données concernées se trouvent dans les **registres internes** de l'U.T.

Format : * CODE-OP *

- **Adressage immédiat :**

L'opérande est fourni explicitement, soit dans le code-opération, soit **immédiatement**

après le code-opération .

Format : * CODE-OP * * VALEUR *

- **Adressage absolu ou étendu :**

L'opérande est trouvé à **l'adresse** fournie à la suite du code-opération.

Format : * CODE-OP * * ADRESSE *

- **Adressage indirect :**

Ici, on peut distinguer 2 types :

- par **registre** : l'opérande est trouvé à l'adresse pointée par le registre, celui-ci étant désigné dans le code-opération (mode implicite).
- par la **mémoire** : l'opérande est trouvé à l'adresse contenue dans la case mémoire dont l'adresse est indiquée à la suite du code-opération.

Format : * CODE-OP * * ADRESSE DE L'ADRESSE *

- **Adressage indexé et relatif :**

L'opérande est trouvé à une adresse **calculée** de la façon suivante :

- un pointeur fournit une adresse dite "de **base**",
- la valeur indiquée à la suite du code-opération est appelée "**déplacement** (offset)",
- l'adresse de l'opérande est obtenue en **additionnant** la base au déplacement.

Ce déplacement peut être exprimé en valeur signée et le pointeur peut être un registre d'index ou le PC lui-même .

Format : * CODE-OP * * DEPLACEMENT *

- **Adressage auto-indexé :**

Le calcul de l'adresse est identique au cas précédent, mais le registre concerné est modifié.

Plusieurs cas sont possibles :

- la nouvelle base est égale à l'adresse calculée,
- le pointeur est incrémenté ou décrémenté après ou avant l'opération. Ce type d'adressage est utilisé par le mécanisme de pile.

Le SP désigne le sommet de la pile :

- **l'empilement** consiste à stocker l'information à l'adresse pointée par le SP, puis à décrémenter la valeur du SP.
- le **dépilement** consiste à incrémenter le SP, puis à lire l'information pointée par le SP.

Note : l'ordre de ces opérations peut être inversé dans certaines machines.Selon le mode d'adressage de la donnée, une instruction sera codée par 1 ou plusieurs octets.

2.9. Mécanisme des interruptions

2.9.1. Définitions

Si le déroulement d'un programme peut être suspendu, puis repris, ce programme est interruptible.

Si l'U.T. peut, par une instruction, inhiber ou autoriser la prise en compte d'interruptions, les interruptions sont masquables.

2.9.2. Différents types d'instruction

- **Avec une seule source d'interruption :**

Les 2 conditions suivantes sont à respecter :

- l'instruction en cours ne doit pas être perturbée.

- l'état de l'U.T. doit être inchangé lorsque le programme interrompu reprend son déroulement.

A la fin du programme d'interruption, l'U.T. revient à l'exécution du programme initial par l'exécution d'une instruction de "retour d'interruption (RTI)". Ceci est réalisé automatiquement en restituant le contexte, puisque ce dernier inclut le contenu du PC. Bien entendu, la zone mémoire de sauvegarde ne doit pas être modifiée par le programme d'interruption.

- **Autres types d'interruption**

On trouve :

- les commandes d'initialisation (RESET) qui sont non masquables et de priorité maximum. Elles servent à démarrer le système en plaçant dans le PC l'adresse de la première instruction à exécuter. Ici, l'opération de sauvegarde de contexte n'est pas nécessaire.
- les interruptions logicielles qui permettent la sauvegarde automatique du contexte et l'appel de fonctions prédéfinies. A chaque interruption est associé un numéro ou type désignant un vecteur (mémoire d'indirection) dans une table.

3. Conclusion :

Dans ce chapitre nous avons présenté l’architecture du microprocesseur, les différents éléments qui le constitue, ces différentes fonctions qui peuvent être réalisées. Et déduire quelle place le microprocesseur occupe-il dans notre ordinateur.

Chapitre 2 : Eléments combinatoires et séquentiels

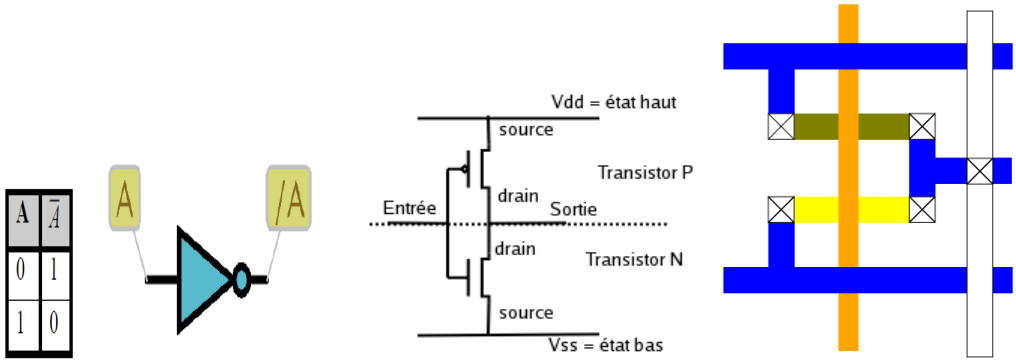
1. Introduction

Le microprocesseur est constitué des circuits logiques combinatoires et séquentiels, Dans ce chapitre nous allons étudier les différents éléments fonctionnels du microprocesseur tels que les éléments logiques et séquentiels.

2. Éléments de logique combinatoire

- **NON**

NON (NOT en anglais) est un opérateur unaire, qui inverse son argument. On notera généralement $NOT(A) = \overline{A}$; et sa table de vérité est :



A	$\overline{A}$
0	1
1	0

(a) (b) (c)

Figure II.1 : (a) Table de vérité du NON et dessin de la porte correspondante, (b) schéma électrique, et (c) dessin régulier

On a bien sûr : $\overline{\overline{A}} = A.$

- **ET**

ET (AND en anglais) est un opérateur à 2 entrées ou plus, dont la sortie vaut 1 si et seulement si toutes ses entrées valent 1. On le note algébriquement comme un produit, c'est à dire $S = A \times B$ ou $S = AB$. La table de vérité d'un ET à deux entrées, et le dessin usuel de la porte correspondante sont représentés dans la figure suivante :

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

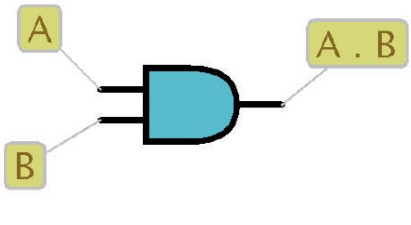


Figure II.2 : Table de vérité du ET, et symbole de la porte correspondante

De par sa définition, le ET est associatif : $A \times B \times C = (A \times B) \times C = A \times (B \times C)$. Il est également idempotent : $A \times A = A$.

➢ **OU**

OU (OR en anglais) est un opérateur à 2 entrées ou plus, dont la sortie vaut 1 si et seulement si une de ses entrées vaut 1. On le note algébriquement comme une somme, c'est-à-dire $S = A + B$. La table de vérité d'un OU à deux entrées, et le dessin usuel de la porte correspondante sont représentés dans la figure au dessous :

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

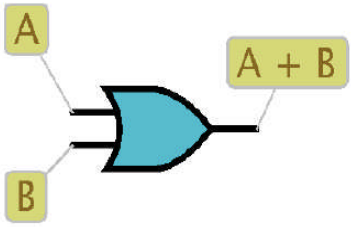


Figure II.3 : Table de vérité du OU, et symbole de la porte correspondante

De par sa définition, le OU est associatif : $A + B + C = (A + B) + C = A + (B + C)$. Il est également idempotent : $A + A = A$.

➢ **NAND**

NAND (= NOT AND) est un opérateur à 2 entrées ou plus, dont la sortie vaut 0 si et seulement si toutes ses entrées valent 1.On le note , et on a donc à deux entrées : $A \uparrow B = \overline{A \cdot B}$
La table de vérité d'un NAND à deux entrées, et le dessin usuel de la porte correspondante sont représentés dans la figure suivante :

A	B	$A \uparrow B$
0	0	1
0	1	1
1	0	1
1	1	0

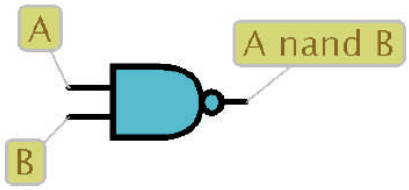


Figure II.4 : Table de vérité du NAND,et dessin de la porte correspondante

Le NAND est un opérateur dit **complet**, c'est à dire qu'il permet à lui seul d'exprimer n'importe quelle fonction combinatoire. Il permet en effet de former un NOT, en reliant ses deux entrées : $\overline{A} = A \uparrow A$

➢ **NOR**

NOR (= NOT OR) est un opérateur à 2 entrées ou plus, dont la sortie vaut 0 si et seulement au moins une de ses entrées 1. On le note ↓, et on a donc à deux entrées : $A{\downarrow}B = \overline{A+B}$

La table de vérité d'un NOR à deux entrées, et le dessin usuel de la porte correspondante sont représentés dans la figure suivante :

A	B	$A{\downarrow}B$
0	0	1
0	1	0
1	0	0
1	1	0

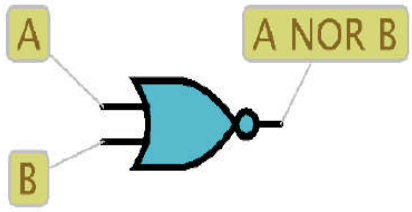


Figure II.5 : Table de vérité du NOR, et dessin de la porte correspondante

Comme le NAND, le NOR est également un ***opérateur complet***. Il permet en effet de former un NOT, en reliant ses deux entrées : $\overline{A} = A{\downarrow}A$

➢ **XOR**

XOR (= EXCLUSIVE OR) est un opérateur à 2 entrées ou plus. On peut le définir de plusieurs façons ; la définition qui permet le plus directement de démontrer ses propriétés est celle qui consiste à en faire un détecteur d'imparité : sa sortie vaut 1 si et seulement si un nombre impair de ses entrées est à 1. On le note : $\oplus$ la table de vérité d'un XOR à deux entrées, et le dessin usuel de la porte correspondante sont représentés dans la figure suivante :

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

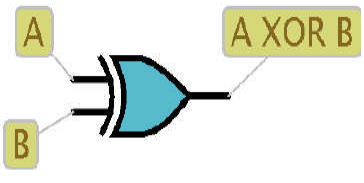


Figure II.6 : Table de vérité du XOR, et dessin de la porte correspondante

➢ **Le multiplexeur**

Le *multiplexeur* est une porte à trois entrées, qui joue un rôle d'aiguillage. Son dessin indique bien cette fonction :

SEL	S
0	A
1	B

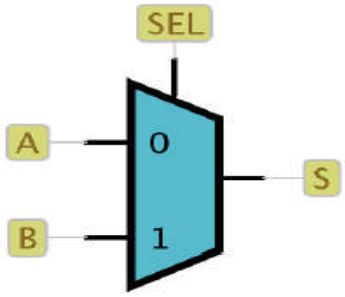


Figure II.7 : Table de vérité condensée du multiplexeur, et dessin de la porte correspondante

3. Éléments de logique séquentielle

➢ Latch RS

La notion de circuit séquentiel avec ses états internes incorpore la notion de ***mémoire*** (retour sur son passé).La cellule mémoire la plus simple est le ***latchRS*** (RS étant les initiales de Reset-Set), parfois appelé ***bistable.***

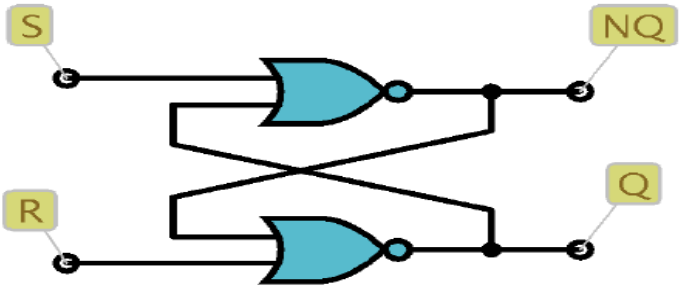


Figure II.8 : Latch RS, cellule mémoire de 1 bit

Ce circuit a clairement des connexions qui reviennent en arrière. Il s'agit d'une cellule de mémorisation de 1 bit, dont le mode d'emploi est le suivant :

1. la cellule est en mode 'lecture' lorsque les entrées *R* et *S* sont toutes les deux à 0 ; la paire $Q, \overline{Q}$ est alors nécessairement dans un des deux états (0, 1) ou (1, 0), car on peut constater que ces deux états, et seulement eux, s'auto-entretiennent lorsque *R* et *S* valent 0.

2. pour mémoriser 0 (c'est à dire que les sorties soient dans l'état $Q, \overline{Q}$ = (0, 1) lorsque la cellule reviendra dans le mode 'lecture'), il faut appliquer un 1 sur R (reset), puis le remettre à 0. Quel que soit son état antérieur, la cellule se retrouve alors dans l'état auto-stable $Q, \overline{Q}$ = (0, 1).

3. pour mémoriser 1 (c'est à dire que les sorties soient dans l'état $Q, -Q$ = (1, 0) lorsque la cellule reviendra dans le mode 'lecture'), il faut appliquer un 1 sur S (set), puis le remettre à 0.

Quel que soit été son état antérieur, la cellule se retrouve alors dans l'état auto-stable$Q, \overline{Q}$ = (1, 0).

Ce circuit est bien séquentiel : ses sorties ne dépendent pas uniquement de ses entrées. La notion d'état interne est ici clairement présente sous la forme d'un bit mémorisé.

Fronts et niveaux ; signaux d'horloge :

Le ***niveau*** d'un signal, c'est sa valeur 0 ou 1. On parle de ***front*** lors d'un changement de niveau : ***front montant*** lorsque le signal passe de 0 à 1, ***front descendant*** lorsqu'il passe de 1 à 0. En pratique, ces transitions ne sont pas tout à fait instantanées, mais leur durée est très inférieure aux temps de propagation des portes.

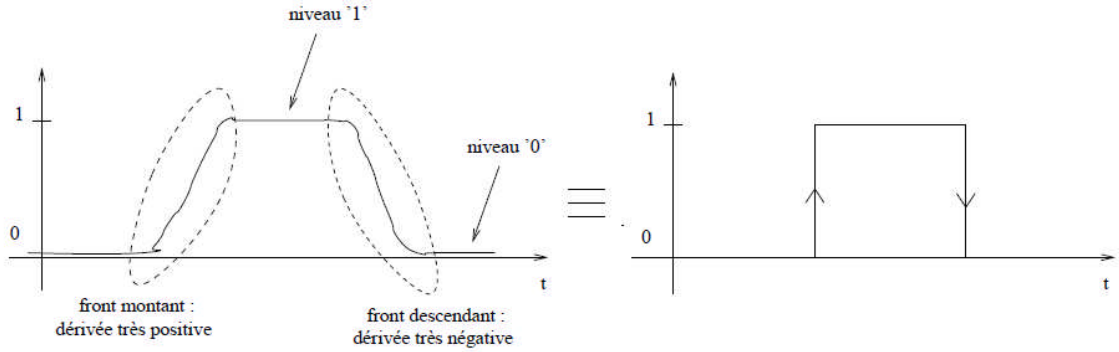


Figure II.9 : Fronts et niveaux d'un signal réel, et leur équivalent logique simplifié

Une horloge est un signal périodique, produit à partir d'un quartz ou d'un réseau RC. Il produit donc une succession périodique de fronts montants et/ou descendants. Comme en physique ou en mathématiques, on emploie les termes de *fréquence* et de *période* pour qualifier la chronologie d'un signal d'horloge ; elle a généralement une forme carrée, mais ce n'est pas obligatoire.

- **Latch RS actif sur un niveau d'horloge**

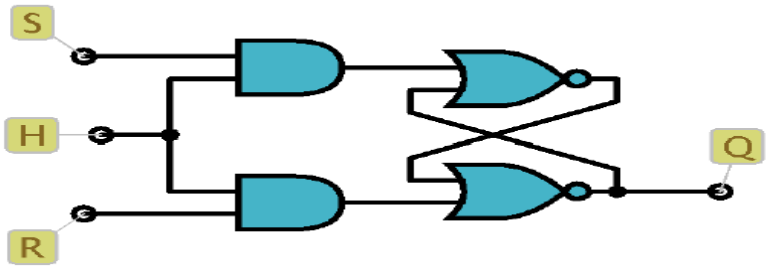


Figure II.10 : Latch RS

L'état de ce circuit est modifiable lorsque H = 1, c'est à dire sur un *niveau* du signal H. Tant que H est au niveau 1, les modifications de S et de R vont avoir un effet sur l'état du bistable. Un tel latch peut avoir certaines applications, mais il n'est pas adapté à la réalisation de circuits séquentiels synchrones, où les changements d'états de leurs différentes parties doivent

se produire de façon parfaitement synchronisée, au moment précis défini par un front d'une horloge commune. La section suivante va présenter un tel circuit.

- **Bascule active sur front d'horloge**

Nous avons donc besoin de composants séquentiels qui changent d'état sur un front d'horloge, et seulement à ce moment précis. Le latch RS vu à la section précédente ne remplissait pas cette condition, puisque le bistable pouvait changer de valeur tant que H était au niveau 1. Le circuit de la figure III.18, qui réalise ce qu'on va appeler une *bascule D* (D pour 'delay'), ne change d'état quant à lui que sur le front descendant de l'horloge H.

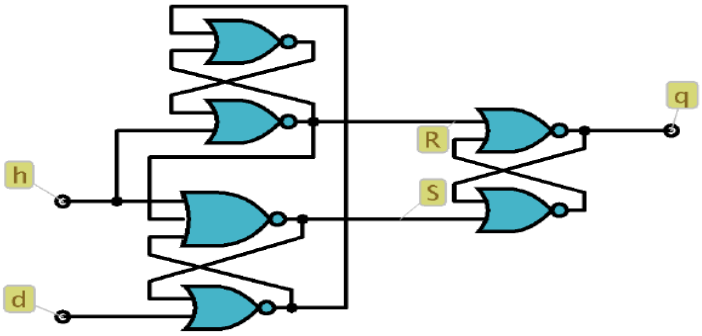


Figure II.11 : Bascule D synchrone

- **Bascule D**

Une telle bascule D est représentée couramment par le schéma de la figure au dessous :

Le signal RST, est une remise à zéro **asynchrone**, qui provoque le forçage à 0 du contenu de la bascule, sans qu'il y ait besoin d'un front d'horloge. Tant que le signal RST est actif (sur niveau 1 sur la figure), la bascule est forcée à 0 et son contenu ne peut pas évoluer. Il s'agit du même signal reset que celui qui est présent sur votre micro-ordinateur, votre téléphone portable, etc. : en l'appliquant, vous remettez dans l'état 0 tous les composants séquentiels de votre machine.

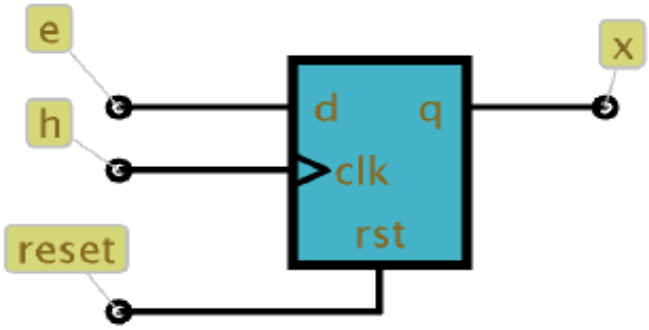
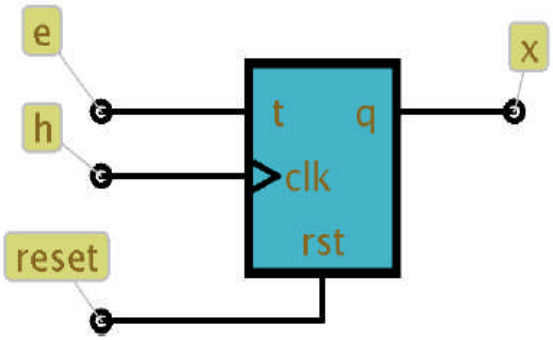


Figure II.12: Représentation habituelle d'une bascule D synchrone

➢ **Bascule T**

Tout comme la bascule D, la ***bascule T*** (trigger) mémorise également un bit de façon synchrone, mais elle a des modalités différentes d'utilisation. Son schéma et sa table de transitions synthétique sont donnés dans la figure suivante :

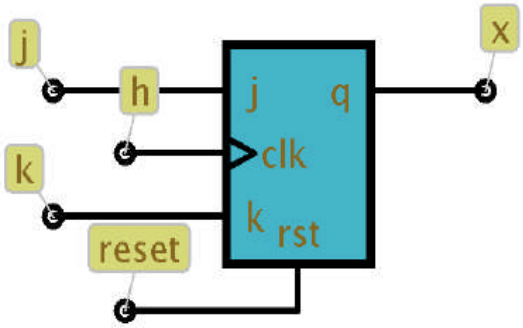


avant		après
T	Q	Q
0	x	x
1	x	$\bar{x}$

Figure II.13 : Bascule T synchrone. Lorsque l'entrée T est à 0 elle ne change pas d'état ; lorsque l'entrée T vaut 1 son état s'inverse

➢ **Bascule JK**

Comme les bascules D et T, la ***bascule JK*** (Jack-Kilby) mémorise un bit de façon synchrone. Son schéma, sa table de transitions simplifiée sont donnés dans la figure suivante :



avant			après
J	K	Q	Q
0	0	x	x
0	1	-	0
1	0	-	1
1	1	x	$\bar{x}$

Figure II.14: Bascule JK synchrone, table de transitions simplifiée. Selon les valeurs de ses entrées J et K, elle permet le forçage de son état à 0 ou à 1, mais aussi la conservation ou l'inversion de son état

Elle fonctionne de façon analogue au latch RS, J jouant le rôle de S et K de R. Lorsque J et K sont à 0, l'état de la bascule ne change pas au front d'horloge. Si on veut faire une mise à 1ou une mise à 0, on applique 1sur J (respectivement sur K) puis on applique un front d'horloge. De plus, mettre à la fois J et K à 1 est autorisé, et l'état de la bascule s'inverse au front d'horloge.

- **Choix des bascules**

La bascule D est clairement adaptée aux situations où on doit stocker un bit présent. La bascule T est adaptée aux situations qui se posent en termes de changement ou d'inversion. La bascule JK semble plus versatile, et elle est d'ailleurs souvent appelée *bascule universelle*. Par analogie avec les portes combinatoires, on pourrait se demander si cette bascule JK est 'plus puissante' que les bascules D ou T, c'est à dire s'il est possible de fabriquer avec elle des circuits que les autres ne pourraient pas faire. La réponse est non. Ces trois types de bascules ont une puissance d'expression équivalente : elles mémorisent un bit, et elles ne diffèrent que par les modalités de stockage de ce bit. Par exemple, on vérifie facilement qu'on peut fabriquer une bascule JK avec une bascule T et réciproquement et la figure suivante le démontre :

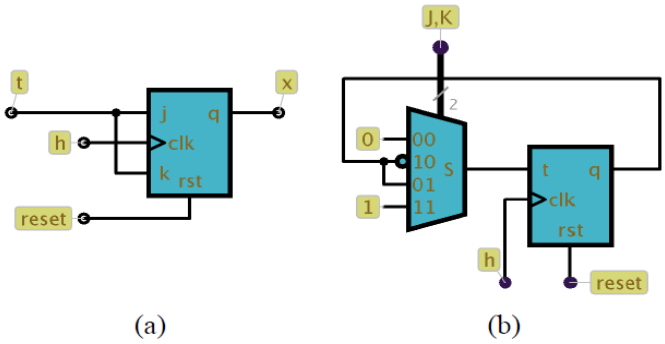


Figure II.15 : (a) construction d'une bascule T avec une bascule JK, (b) construction d'une bascule JK avec une bascule T

4. Circuits séquentiels réutilisables

- **Compteur binaire**

Un ***compteur binaire*** sur *n* bits incrémente à chaque front d'horloge une valeur de *1* bits. On pourrait d'abord penser à créer un registre de *n* bits avec des bascules D, puis à relier leurs entrées à un circuit d'incrémentation, mais il y a une solution plus directe et efficace : un bit est recopié avec inversion si tous les bits à sa droite valent 1, et le bit de poids faible est toujours inversé. Cette description en termes d'inversion conduit à une conception à base de bascules T, dans laquelle l'entrée T d'une bascule est reliée au ET de tous les bits qui sont à sa droite. La figure suivante montre un exemple d'un tel compteur sur 4 bits.

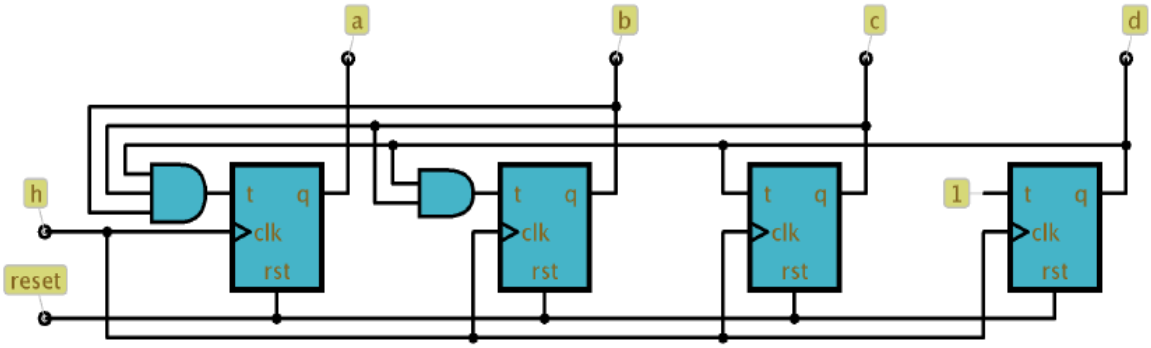


Figure II.16 : *Compteur binaire 4 bits. Un bit est inversé lorsque tous les bits situés à sa droite valent 1*

On peut également créer un circuit décompteur selon la même méthode : un bit est inversé lorsque tous les bits qui sont à sa droite valent 0 cela est représenté dans la figure suivante:

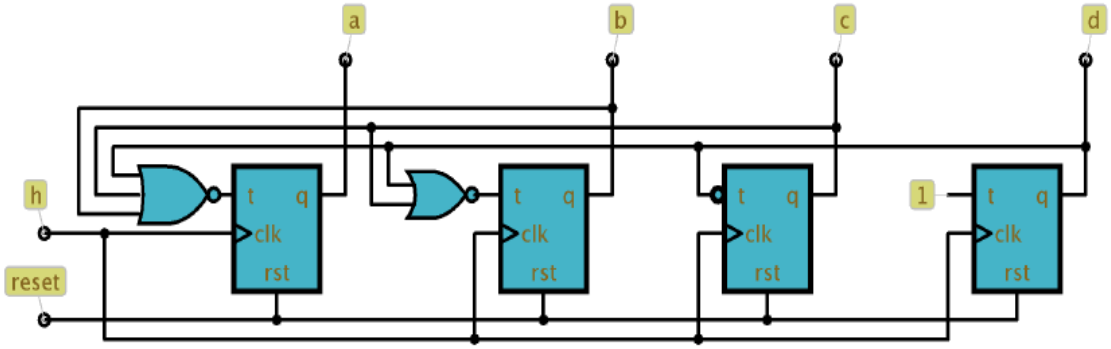


Figure II.17 :Décompteur binaire 4 bits.Un bit est inversé lorsque tous les bits situés à sa droite valent 0

- **Ajout d'une entrée de validation**

Ces circuits comptent ou décomptent en permanence, et il n'y a aucun moyen de suspendre et de reprendre leur comptage.On va maintenant ajouter une entrée de validation *en* (pour 'enable') qui ne permettra au compteur de compter que si elle vaut '1'. Lorsqu'elle vaudra '0', le compteur restera figé sur la dernière valeur produite, malgré l'arrivée de fronts d'horloge sur *h*.

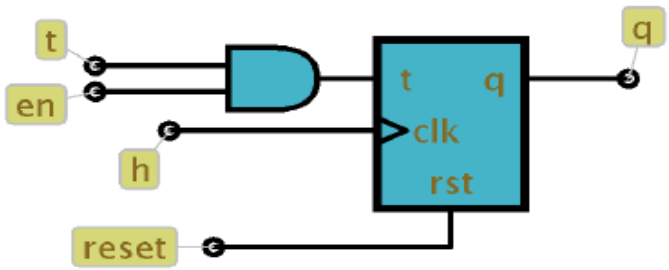


Figure II.18 : Bascule T avec entrée de validation

- **Ajout d'une remise à zéro synchrone**

On souhaite maintenant ajouter une entrée *sclr* de remise à zéro synchrone, qui remet le compteur à zéro *au prochain front d'horloge*. Elle n'est bien sûr pas de même nature que l'entrée de reset asynchrone, qui n'est là que pour l'initialisation. Par ailleurs il ne faut pas essayer d'utiliser ce reset asynchrone pour implémenter notre remise à zéro synchrone. La règle d'or des circuits séquentiels synchrones purs, c'est de relier entre-elles toutes les horloges et de relier entre eux tous les signaux de reset asynchrone, sans chercher à les modifier.

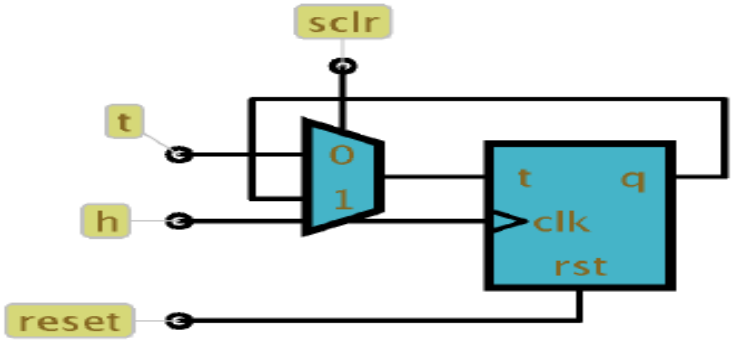


Figure II.19: Bascule T avec remise à zéro synchrone

5. Éléments fonctionnels d'un processeur

Dans cette section on va décrire les modules combinatoires et séquentiels les plus communément utilisés dans les architectures d'ordinateurs :

- **Décodeurs**

Un ***décodeur*** est un circuit possédant *n* entrées et 2*n* sorties numérotées comme la montre la figure au dessous.

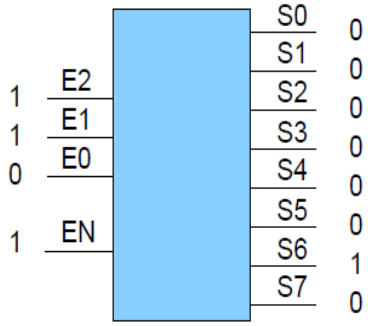


Figure II.20 : Décodeur 3 vers 8.Avec la valeur 6 en entrée (110 en binaire), la sortie numéro 6 est activée

À tout moment, une et une seule sortie est active : celle dont le numéro correspond à la valeur binaire présente sur les *n* entrées. Le décodeur traduit donc la valeur d'entrée en une information de position spatiale.

- **Multiplexeurs**

Ces sont des circuits d'aiguillage pour les signaux logiques. Un ***multiplexeur*** possède 2^n entrées de données, n entrées de commandes, et une seule sortie. On indique sur la commande le numéro (en binaire) de l'entrée de donnée qui va être aiguillée en sortie. On a en figure au dessous un exemple de multiplexeur 8 vers 1.

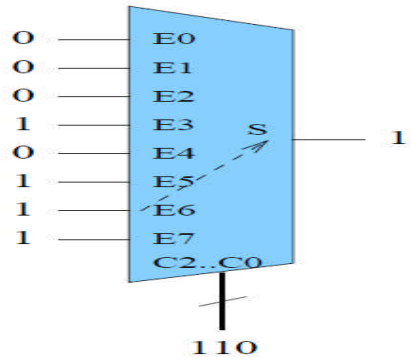


Figure II.21 : Multiplexeur 8 vers 1.L'entrée numéro 6 est aiguillée vers la sortie

6. Conclusion

Lors de ce chapitre on a pu voir les différents éléments combinatoires et séquentiels constituant un microprocesseur capable d'effectuer des instructions très simples.

Chapitre 3 : Conception du processeur

1. Introduction

L'électronicien a toujours utilisé des outils de description pour représenter des structures logiques ou analogiques. Le schéma structurel que l'on utilise depuis si longtemps et si souvent n'est en fait qu'un outil de description graphique. Aujourd'hui, l'électronique numérique est de plus en plus présente et tend bien souvent à remplacer les structures analogiques utilisées jusqu'à présent. Ainsi, l'ampleur des fonctions numériques à réaliser nous impose l'utilisation d'un autre outil de description. Il est en effet plus aisé de décrire un compteur ou un additionneur 64 bits en utilisant l'outil de description VHDL plutôt qu'un schéma.

Dans ce chapitre, nous allons concevoir un microprocesseur de 4 bits avec le langage de description VHDL.

2. Introduction au langage de description VHDL

2.1. Définition

VHDL est le sigle de *VHSIC hardware description language* ; VHSIC, de *very-high-speed integrated circuits*, une initiative de la Défense américaine dans les années 1980 visant la construction de circuits intégrés très rapides. Le VHDL est un langage de description du matériel utilisé en électronique. En tant que standard, il est indépendant du logiciel utilisé pour la compilation, la programmation des composants, la simulation, etc.

2.2. Utilité

VHDL est un langage de description matériel. À la différence des langages informatiques classiques, VHDL ne vise pas une exécution procédurale, son but est de permettre la description de tout système électronique, d'en valider le fonctionnement avant de passer à la mise en œuvre matérielle.
Tout en étant d'un très haut niveau d'abstraction en électronique (il est indépendant de la technologie utilisée : FPGA, CPLD, ASIC, etc.). Cette abstraction permet d'ailleurs de le simuler sur ordinateur avant de programmer la moindre puce.

Utiliser ce langage évite de dériver des tables de vérité des spécifications, que l'on simplifie alors en équations booléennes à implémenter. Il permet d'expliciter la fonction à implémenter sans se soucier de ces détails, bien que l'implémentation en pratique se fera à l'aide de portes logiques (le compilateur se chargera de passer du code aux portes).

3. Conception du microprocesseur

3.1. UAL (unité arithmétique et logique)

Le fonctionnement de l'UAL est représenté dans la figure ci-dessous :

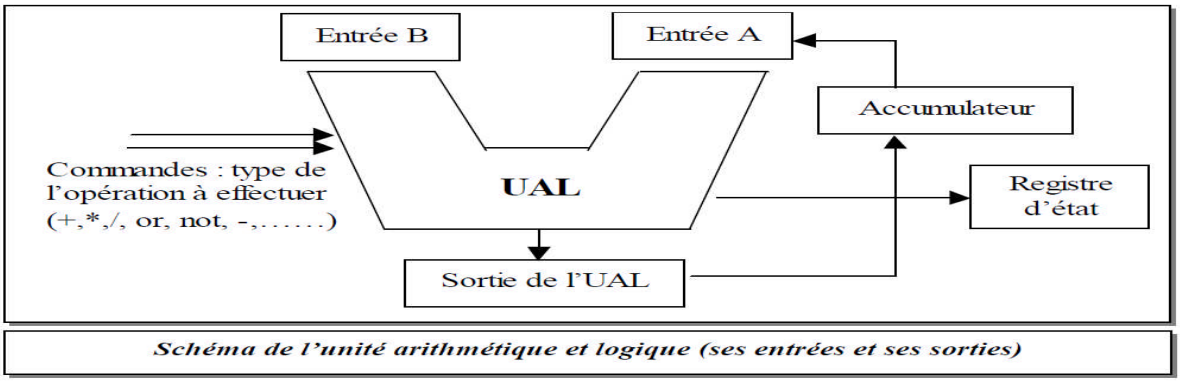


Figure III.1 : Description fonctionnelle du l'Unité Arithmétique et Logique 4 bits

L'UAL est contrôlée par trois entrées de sélection (S0 à S2) d'un multiplexeur 8 vers 1, qui sélectionnent l'opération à effectuer.

S_2	S_1	S_0	Function (F)
0	0	0	A+B
0	0	1	A-B
0	1	0	A-1
0	1	1	A+1
1	0	0	A $\wedge$ B
1	0	1	A $\vee$ B
1	1	0	NOT A
1	1	1	A $\oplus$ B

Figure III.2 : Table de vérité de l'UAL

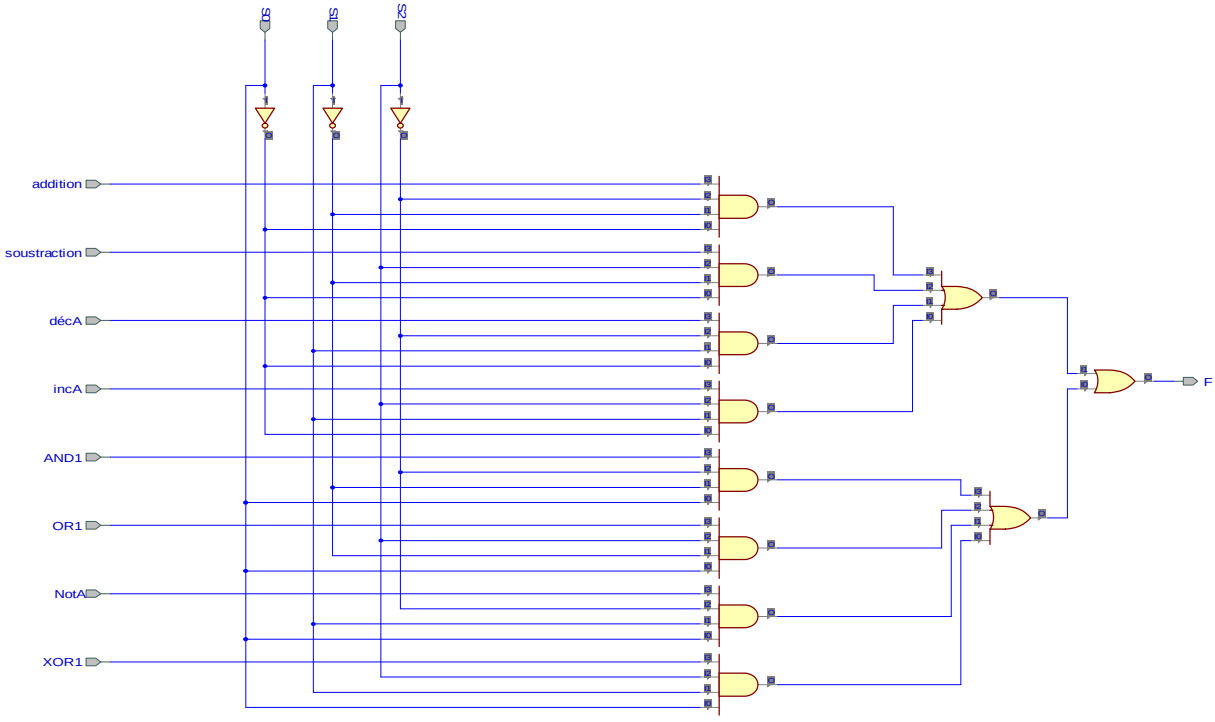


Figure III.3 : schéma fonctionnel d'un multiplexeur 8 vers 1en VHDL

Exemple : Additionneur /soustracteur 4 bits en VHDL

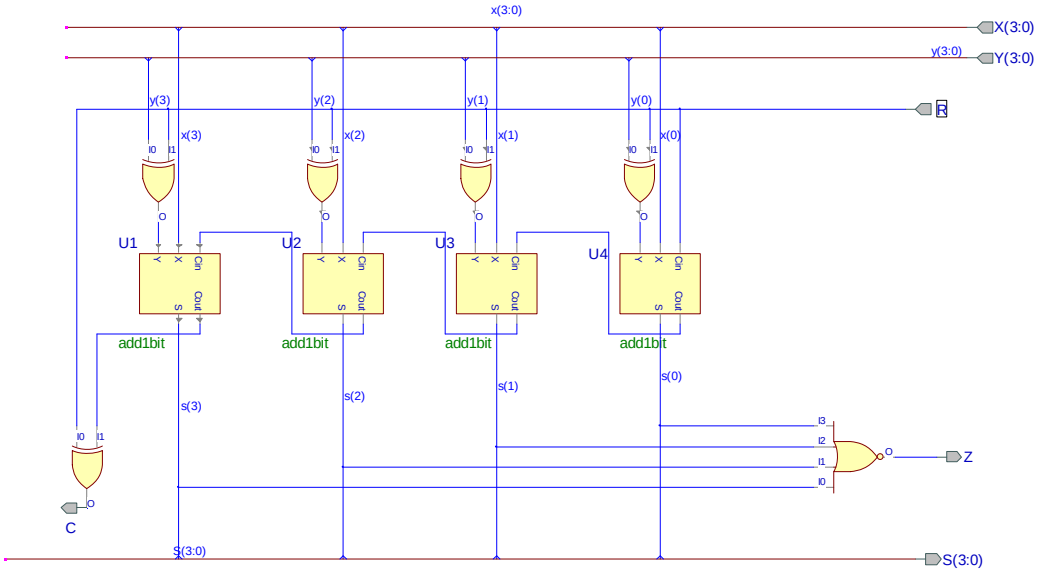


Figure III.4 : schéma fonctionnel d'un additionneur /soustracteur 4 bits en VHDL

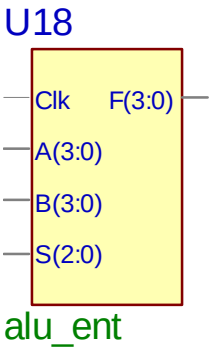
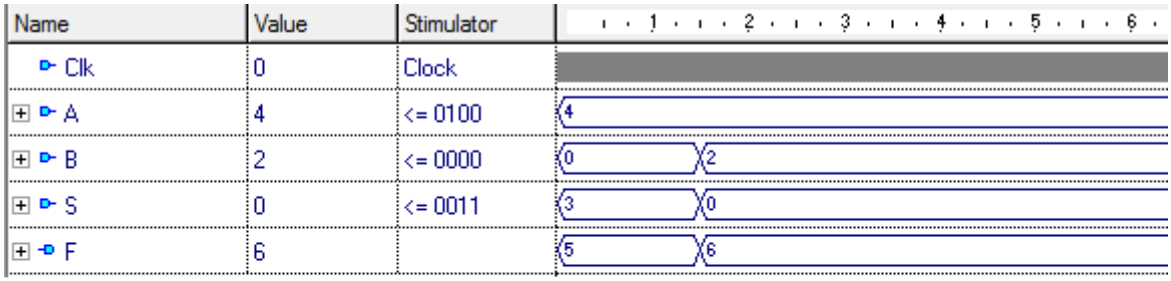


Figure III.5 : Bloc fonctionnel d'une UAL en VHDL

- **Simulation**



3.2. Les Registres

Un registre est une zone mémoire à l'intérieur du microprocesseur de faible taille, qui permet de mémoriser des mots mémoires ou des adresses d'une façon temporaire lors de l'exécution des instructions.

Les registres sont constitués de plusieurs bascules de mémorisation, la bascule D est la plus utilisée.

- **La Bascule D**

Ce type de bascule est très utilisé dans les mémoires intermédiaires situées à l'entrée des divers organes des microprocesseurs, là où l'information doit être recopié systématiquement avant qu'elle ne pénètre dans l'organe à qui elle est destinée.

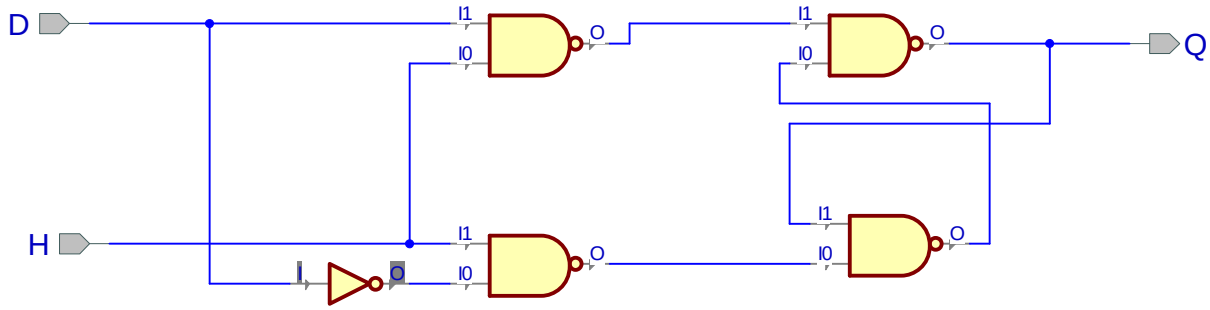


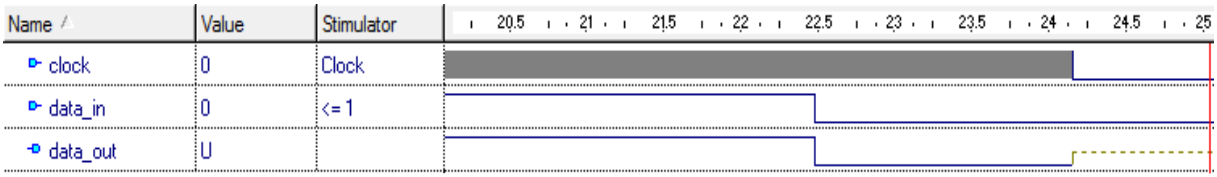
Figure III.6 : schéma fonctionnel de la bascule D en VHDL

Si H=0 quel que soit la valeur de D, Q et Q' restent inchangés.

- Si H=1, D=1 Q=1 et Q'=0
- Si H=1, D=0 Q=0 et Q'=1

T-1	T
D	Q
0	0
1	1

- **Simulation**



A base de la bascule D on va concevoir un registre qui permet de lire ou d'écrire un mot mémoire.

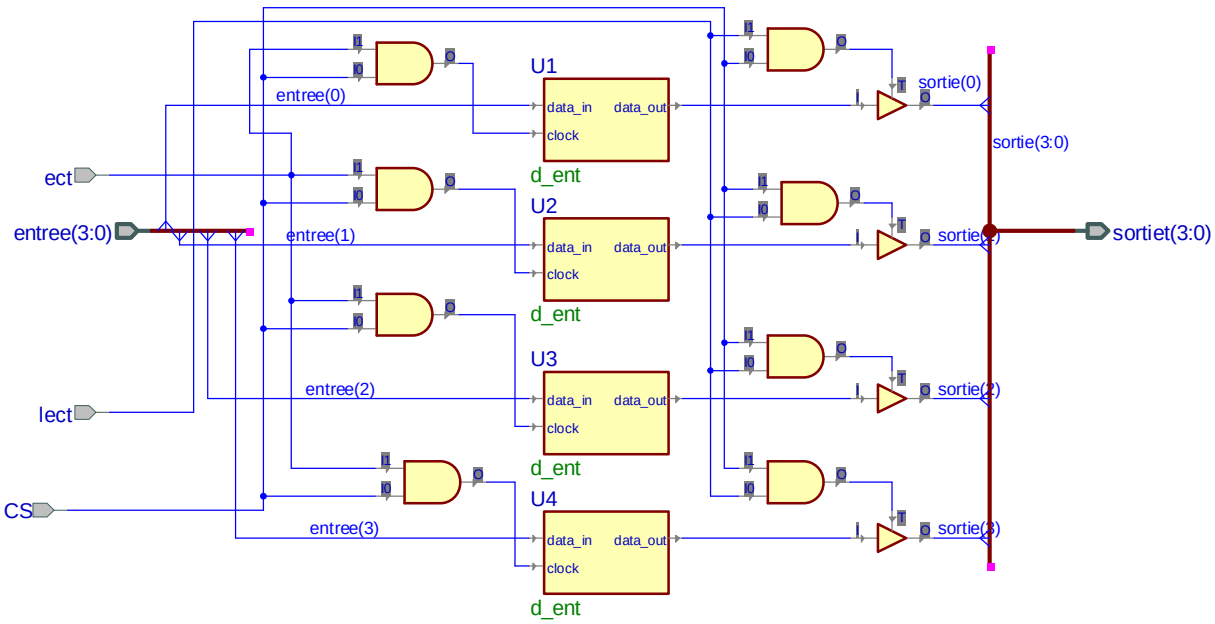
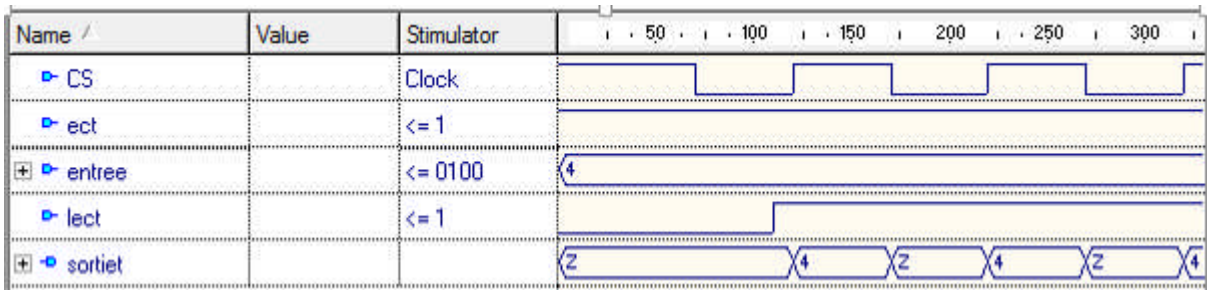


Figure III.7 : schéma fonctionnel d'un registre 4 bits en VHDL

Le registre est constitué de 4 bascules D commandées avec trois signaux, le premier est le signal ect (écriture) (activé à 1) qui permet aux bascules de mémoriser les données présente à ses entrées, quand le signal clock de la bascule est en front montant. Le deuxième est le signal lect (lecture) (activé à 1) qui permet la lecture des données mémorisées dans les bascules et les transférées à la sortie du registre. Et le troisième signal c'est le signal de commande(CS) qui sélectionne dans quelle bascule où on va écrire ou lire.

- **Simulation**



3.3. Décodeur 4 vers 16

Un décodeur est un circuit à (n) entrées et (m) sorties, qui permet de sélectionner une seule sortie parmi plusieurs. Le décodeur fait correspondre à un code binaire présent sur ses (n) entrées sur une seule sortie active.

Le numéro de la sortie active est équivalent au numéro présent sur la combinaison des entrées. (combinaison N°1 →sortie N°1 activée).

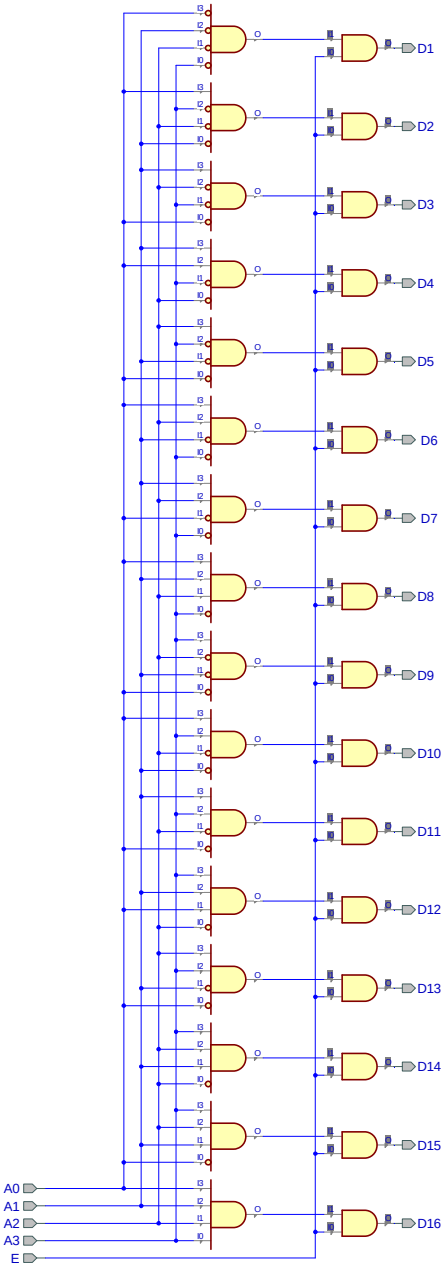
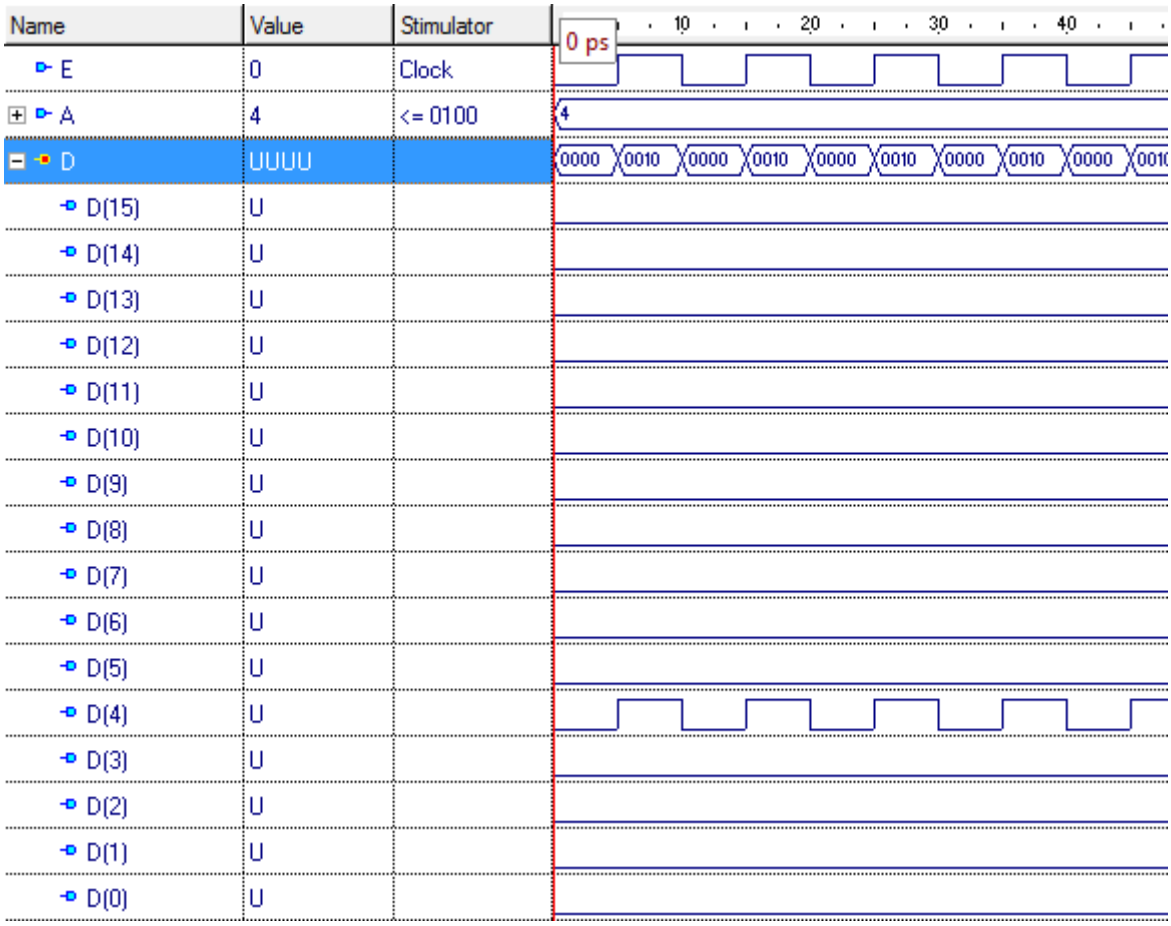


Figure III.8 : schéma fonctionnel d'un décodeur 4 vers 16 en VHDL

Avec A0…A3, les entrées du décodeur, et D1…D16, ces sorties et E l'entrée de validation du décodeur.

➢ **Simulation**



3.4. Compteur Ordinal (CO)

Appelé aussi compteur de programme .Le CO est constitué d'un registre dont le contenu représente l'adresse de la prochaine instruction à exécuter. Il est donc initialisé avec l'adresse de la première instruction du programme. Puis il sera incrémenté automatiquement pour pointer vers la prochaine instruction à exécuter.

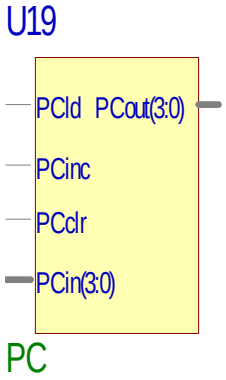
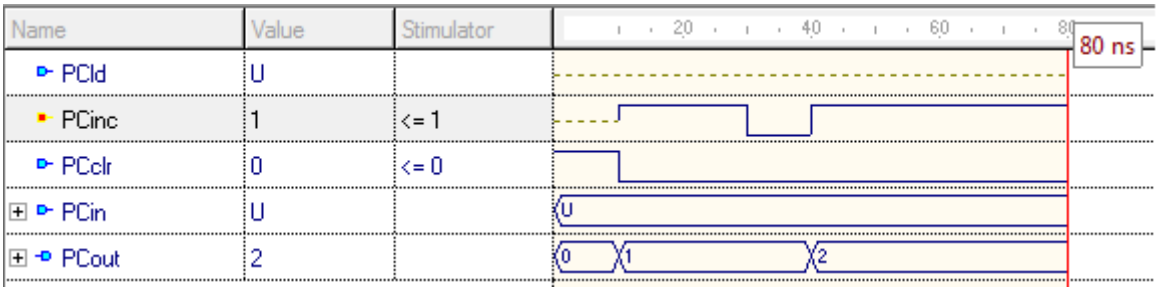


Figure III.2 : Bloc fonctionnel d'un compteur ordinal en VHDL

➢ **Simulation**



3.5. Banc de registres

En général, un processeur comporte plusieurs registres regroupés dans un banc de registres; ces registres sont numérotés, et le processeur spécifie quel registre il veut lire ou écrire en indiquant le numéro du registre correspondant.

Le numéro de registre peut être considéré comme une adresse, et le banc de registres comme une petite mémoire.

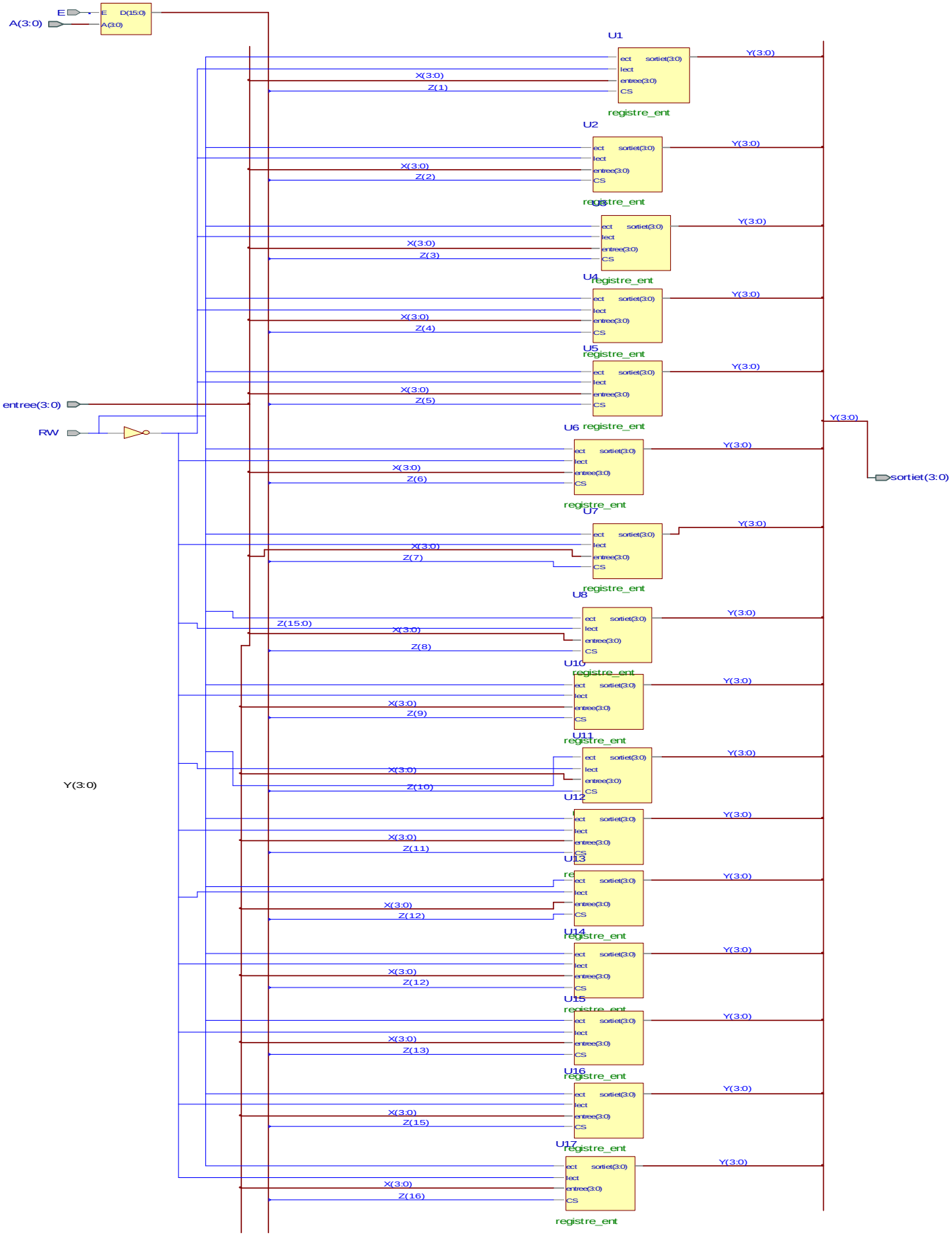


Figure III.3 : schéma fonctionnel d'un banc de registres de 16 registres en VHDL

Le schéma est mené d'un signal de commande write/read qui permet de sélectionner quelle opération effectuée (lecture/écriture).Et d'une entrée de sélection (CS) commandée par un décodeur 4 vers16 qui va déterminer le registre où on va lire ou écrire.

3.6. Microprocesseur à 4 bits

Notre microprocesseur 4 bits qu'on a conçu est un microprocesseur simplifié qui sert à expliquer le fonctionnement d'un microprocesseur et les différents blocs qui le constituent.

Il nous permet de comprendre le mécanisme de l'exécution de l'instruction.

Le traitement de l'instruction est décomposé en trois phases.

- Recherche de l'instruction :

Le contenu du compteur ordinal est placé sur le bus d'adresse et une case mémoire est sélectionnée.

La mémoire reçoit l'ordre de lecture, les données sont alors transférées vers le registre d'instruction.

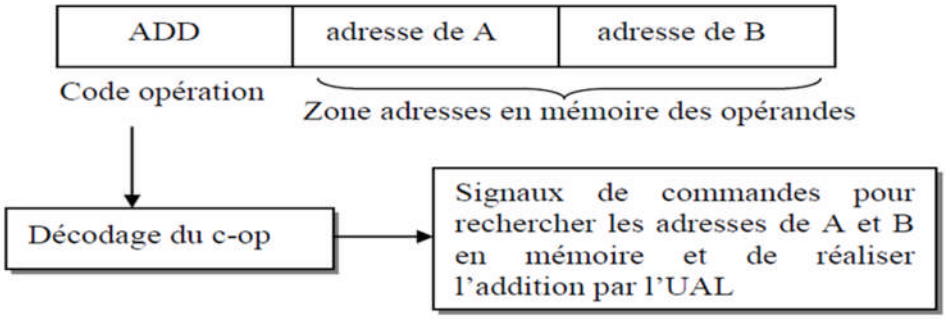
- Décodage de l'instruction :

Le registre d'instruction contient maintenant le premier mot de l'instruction. Ce premier mot contient le code opératoire qui définit la nature de l'opération à effectuer (addition, soustraction,...) et le nombre de mots de l'instruction.

1. L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.

2. Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur sur le bus de données.

3. L'opérande est stocké dans un registre.



- Exécution de l'instruction :

1. L'UAL réalisent l'instruction a exécuté.

2. L'unité de commande positionne le PC pour l'instruction suivante.

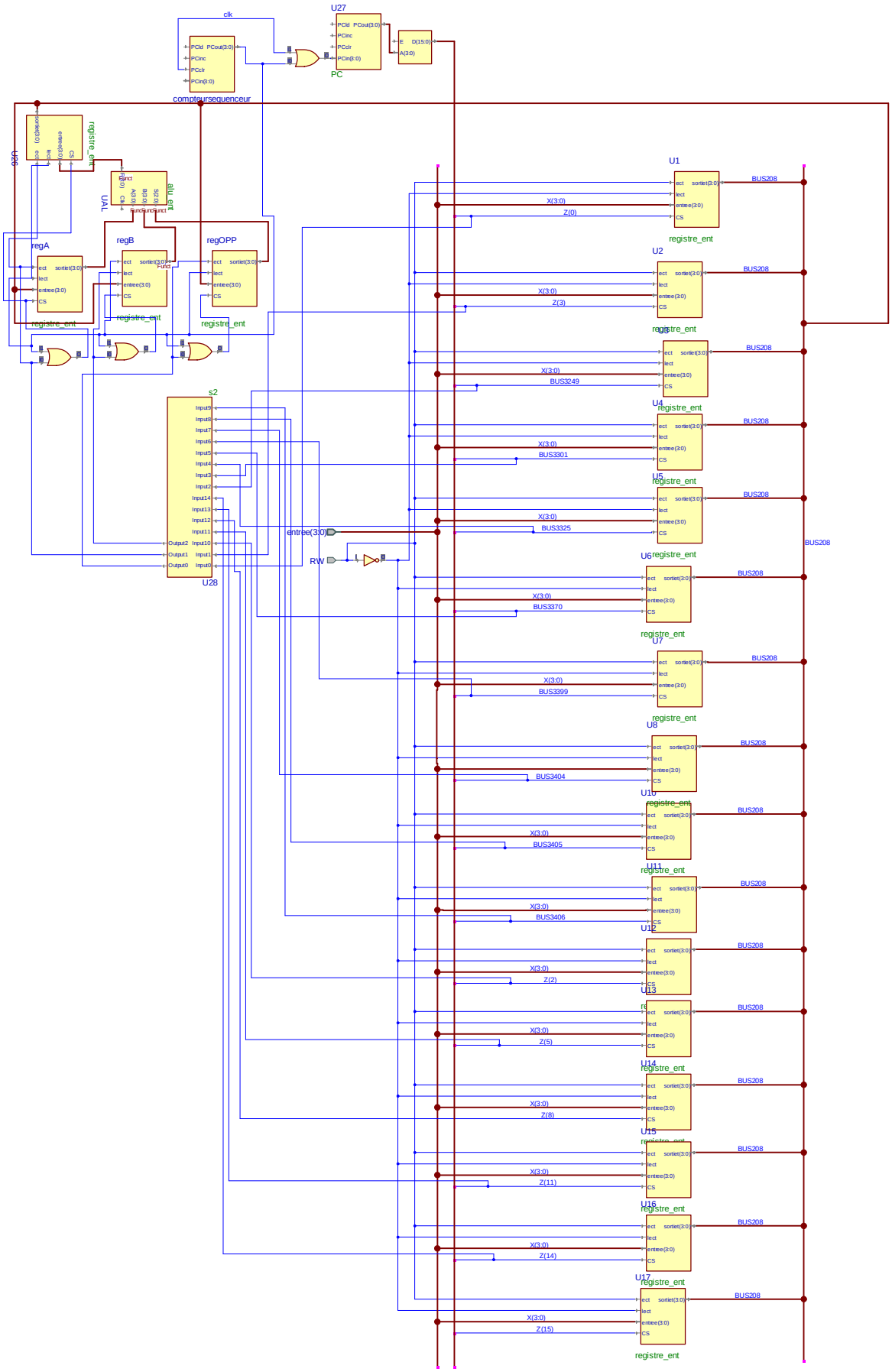


Figure III.4 : schéma fonctionnel d’un microprocesseur à 4 bits en VHDL

4. Conclusion

L’objectif de ce travail est de donner une vue d’ensemble et complète d’un circuit intégré totalement opérationnel. Le microprocesseur 4 bits que nous avons réalisé, est un circuit conçu pour être le plus simple et le plus pédagogique possible. On a utilisé le langage de description VHDL pour le concevoir qui a été d’une utilité remarquable.

Conclusion Générale

Les microprocesseurs ont révolutionné la société à travers l'apport technologique. Ils ont permis de développer l'informatique, mais aussi la communication grâce à leur système de sauvegarde et aux appareils électroniques qui en découlent. Au quotidien, on trouve des microprocesseurs un peu partout autour de nous et il serait difficile de vivre sans, car la naissance du microprocesseur a été un tournant majeur du XXe siècle. L'existence de cet objet pourtant si petit a eu un énorme impact sur la société et l'a définitivement fait évoluer.

Ce mémoire présente l'architecture d'un microprocesseur à 4 bits ainsi que les différents aspects de sa conception. La description du microprocesseur a été réalisée avec le langage de description de matériel VHDL.

En plus de connaissance théoriques, ce travail nous a permis d'acquérir des compétences pratiques en architecture des microprocesseurs qui est le cœur des machines informatiques.

Cette étude nous a montré que la conception d'un microprocesseur était difficile et très enrichissante car elle nous a permis de découvrir le monde de la conception des circuits intégrés qui est toujours un domaine de recherche. Comme nous avons aussi constaté que ce projet pouvait faire figure d'un cours pédagogique pour une introduction aux microprocesseurs.

Sur la base de notre travail, des travaux futurs sont envisageables tels que l'ajout de certaines instructions au jeu (sauvegarde des résultats dans la mémoire, multiplication, division etc.….), augmentation de nombre de bits du microprocesseur ainsi qu'une version programmable en « VHDL » qui pourrait être implémentée sur circuits reprogrammables « FPGA ».

Annexes

Annexe 1

Code VHDL d'une UAL (Unité arithmétique et logique)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity alu_ent is
port(   Clk : in std_logic;
        A,B : in signed(3 downto 0);
        S : in unsigned(2 downto 0);
        F : out signed(3 downto 0)
        );
end alu_ent;
architecture alu_arch of alu_ent is
signal t1,t2,t3: signed(3 downto 0) := (others => '0');
begin
t1 <= A;
t2 <= B ;
F <= t3;
process(Clk)
begin
   if(rising_edge(Clk)) then
      case S is
         when "000" =>
            t3<= t1 + t2;  --addition
         when "001" =>
            t3<= t1 - t2; --subtraction
         when "010" =>
            t3<= t1 - 1;  --sub 1
         when "011" =>
            t3<= t1 + 1; --add 1
```

```
            when "100" =>
               t3<= t1 and t2; --AND gate
            when "101" =>
               t3<= t1 or t2;  --OR gate
            when "110" =>
               t3<= not t1 ;  --NOT gate
            when "111" =>
               t3<= t1 xor t2; --XOR gate
            when others =>
               NULL;
         end case;
      end if;
end process;
 end alu_arch;
```

Annexe 2

Code VHDL d’une bascule D

```
library ieee ;
use ieee.std_logic_1164.all;
use work.all;
entity d_ent is
port(   data_in:        in std_logic;
        clock:          in std_logic;
        data_out:       out std_logic
);
end d_ent;
architecture d_arch of d_ent is
begin
   process(data_in, clock)
   begin
        if (clock='1' and clock'event) then
           data_out <= data_in;
        end if;
   end process;
end d_arch;
```

Annexe 3

Code VHDL d'un décodeur 4 vers 16

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity \dec 4 vers 16-ent\ is
        port(
                E : in STD_LOGIC;
                A : in STD_LOGIC_VECTOR(3 downto 0);
                D : out STD_LOGIC_VECTOR(15 downto 0)
            );
end \dec 4 vers 16-ent\;
architecture \dec 4 vers 16-arch\ of \dec 4 vers 16-ent\ is
begin
  process (A, E)
begin
if (E='0') then
D <= "0000000000000000";
else
case A is
when "0000" =>

D <= "0000000000000001";
when "0001" =>
D <= "0000000000000010";
when "0010" =>
D <= "0000000000000100";
when "0011" =>

D <= "0000000000001000";
when "0100" =>
D <= "0000000000010000";
```

```
when "0101" =>
D <= "0000000000100000";
when "0110" =>
D <= "0000000001000000";
when "0111" =>
D <= "0000000010000000";
when "1000" =>
D <= "0000000100000000";
when "1001" =>
D <= "0000001000000000";
when "1010" =>
D <= "0000010000000000";
when "1011" =>
D <= "0000100000000000";
when "1100" =>
D <= "0001000000000000";
when "1101" =>
D <= "0010000000000000";
when "1110" =>
D <= "0100000000000000";
when "1111" =>
D <= "1000000000000000";
when others =>
D <= "0000000000000000";
end case;
end if;
end process;
end \dec 4 vers 16-arch\;
```

Annexe 4

Code VHDL d'un compteur ordinal

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity PC_ent is
port(
        PCld:   in std_logic;
        PCinc: in std_logic;
        PCclr:  in std_logic;
        PCin:   in std_logic_vector(3 downto 0);
        PCout: out std_logic_vector(3 downto 0)
);
end PC_ent;
architecture pc_arch of PC_ent is
signal tmp_PC: std_logic_vector(3 downto 0);
begin
    process(PCclr, PCinc, PCld, PCin)
    begin
            if PCclr='1' then
                tmp_PC <=(others =>'0');
            elsif (PCld'event and PCld = '1') then
                tmp_PC <= PCin;
            elsif (PCinc'event and PCinc = '1') then
                tmp_PC <= tmp_PC + 1;
            end if;
    end process;
    PCout <= tmp_PC;
end pc_arch;
```

Bibliographie

1_Architecture des ordinateurs (Cours Dumartin) [T.Dumartin].

2_Cours et travaux pratiques de microprocesseur [M.Cattoen].

3_Concevoir son microprocesseur (structure des systèmes logiques) [Jean-Christophe Buisson].

4_Architecture des ordinateurs (cours DEUG MIAS) [Frédéric Vivien].

5_Architecture des Ordinateurs (cours Licence Informatique USTL) [David Simplot].

6_Architecture des systèmes à microprocesseurs [Maryam Siadat et Camille Diou].

Sites web

1_Fonctionnement des composants du PC.

http://www.vulgarisation-informatique.com/composants.php

2_Cours d'initiation aux microprocesseurs et aux microcontrôleurs.

http://www.polytech-lille.fr/~rlitwak/Cours_MuP/sc00a.htm

3_Le cours hardware d'YBET informatique.

http://www.ybet.be/hardware/hardware1.htm

4_Introduction au langage VHDL.

http://tcuvelier.developpez.com/tutoriels/vhdl/introduction-langage/