Microprocesseurs et microcontrôleurs

Notes de Cours pour 1ière Année

Master électrotechnique

– Partie 1 –

I. Généralités sur les systèmes à microprocesseur

Le microprocesseur est l'aboutissement de progrès technologiques tant dans les domaines mécanique, informatique et électronique.

Quelques dates :

1645 : Pascal invente la machine à calculer entièrement mécanique (addition et soustraction)

1800 : Jacquart invente le métier à tisser avec cartes perforées.

1940 : Premier ordinateur à relais mécaniques (U. S. Navy)

1946 : Premier ordinateur à tubes à vide (1800). (grande dissipation : 150 kW, problème de rendement et de fiabilité)

1948 : Progrès de la physique quantique avec découverte de l'effet transistor.

1950 : Réalisation des premières mémoires à ferrites.

1958 : Développement du premier circuit intégré (4 à 5 tr/puce).

1964 : Ordinateur à transistors (à base de circuits TTL : 50 transistors dans une puce)

1970 : Premiers circuits L.S.I.- naissance du premier microprocesseur 4 bits avec 1000 transistors sur une puce.

1975 : Naissance du microprocesseur Motorola 6800 (8 bits) .

1980 : Apparition du microprocesseur 16 bits avec 50000 transistors sur la puce. Et sorti du premier ordinateur portable de type PC de IBM (1981)

I.1. structure d'un système à base de microprocesseur

Aujourd'hui, tous les appareils électroniques, qu'ils soient personnels (téléphones, PDA, ordinateur,...), domestiques (téléviseur lave-linge, cuisinière,...) ou professionnels (oscilloscope, émetteur TV,..) possèdent comme élément essentiel un microprocesseur ou plus exactement un microcontrôleur. Ceci est dû au fait que cet élément peut, en théorie, réaliser n'importe qu'elle action de manipulation sur les signaux. On parle alors d'un système de traitement numérique de données. Le principe d'un tel système est représenté sur la

figure I.1. Il possède une entrée (pouvant recevoir plusieurs signaux) et une sortie (pouvant délivrer également plusieurs signaux).

Le *cerveau* du système est une **unité centrale de traitement** (CPU : Central Processor Unit) pouvant réaliser différentes opérations de manipulations des données considérées. Ces opérations se limitent en réalité uniquement à des opérations arithmétiques (addition, soustraction,...) et logiques (transfert, inversion, comparaison,...). Quant aux données manipulées, elles sont obligatoirement de type numérique binaire ; c.-à-d. représentés par des **0** et des **1** (ex : '011010110').

Un tel élément (CPU) ne peut pas traiter directement les données issues du monde réel car ces données sont de type analogique, c'est-à-dire continues **dans le temps** et **dans l'espace des valeurs**. Il faut alors adjoindre au système une **interface d'entrée** capable de convertir les différents signaux (données) analogiques en signaux numériques. De même pour la restitution des résultats, le CPU fournis des données traitées sous forme numérique, il faut alors une interface de sortie afin de présenter les résultats sous une forme compatible avec le monde réel (affichage, son,...).

Il faut cependant distinguer, dans un tel système, deux parties différentes mais néanmoins complémentaires : **le logiciel** et le **matériel**.

Le matériel est tout ce que l'on peut voir du système ; c'est-à-dire les différents composants électroniques, alors que le logiciel est la partie constituée par l'ensemble des actions effectuées par le matériel ; ce sont les programmes.

Dans ce cours, nous allons nous intéresser uniquement à la partie traitement c'est-à-dire au fonctionnement de l'unité de calcul et à sa programmation. Néanmoins, et par soucis de complément, nous allons présenter le principe de base du fonctionnement de quelques unités d'entrée/sortie.

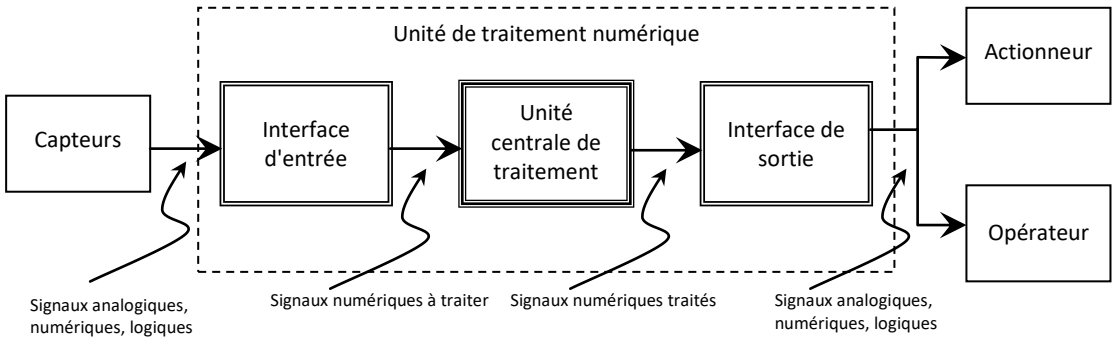


Fig.I.I : structure de base d'un système de traitement numérique

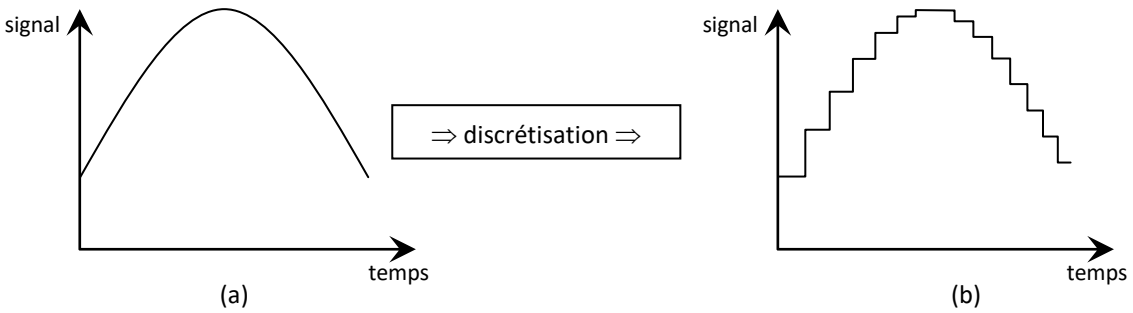


Fig.I.2 : (a)- signal continu, (b)- signal discret

I.2. Différence entre microprocesseur et microcontrôleur

Historiquement, le microprocesseur a été le premier à être utilisé dans un système de traitement automatique de l'information (Informatique). Cependant, le microprocesseur, tel qu'il a été conçu, ne contient principalement que deux unité (unité de contrôle, et unité de calcul) ce qui est loin d'être suffisant pour réaliser *l'opération* de traitement. On doit alors lui ajouter des unités *périphériques* tel que la mémoire, les unités d'entrée/sortie... Le microprocesseur est donc réservé à des systèmes évolutifs, dans lesquels on peut ajouter ou retrancher des périphériques. Ceux-ci ne doivent donc pas être inclus dans le microprocesseur. Dans un PC (personal computer), par exemple, l'ajout d'une barrette mémoire peut augmenter la puissance et la capacité du système.

En parallèle aux systèmes informatiques, très gourmands en mémoire, s'est ressenti le besoin de développer des circuits qui intègrent, en plus des unités de contrôle et de calcul, la mémoire ainsi que certains périphériques. Ces circuits appelés **microcontrôleurs** ne possèdent pas beaucoup de mémoire mais sont suffisamment performants pour être utilisés dans l'industrie pour la mesure, le contrôle, la commande,...

On vient de voir que finalement, la différence entre microprocesseur et microcontrôleur est principalement située au niveau de l'utilisation. Le microprocesseur ne peut pas être utilisé directement, vu qu'il n'intègre aucun élément lui permettant de fonctionner (mémoire, interface E/S …), alors que le microcontrôleur est généralement conçu avec la mémoire et des différentes interfaces de communication et d'acquisition.

I.3. Structure interne d'un microcontrôleur

Un microcontrôleur se présente sous la forme d'un circuit intégré réunissant tous les éléments d'une structure à base de microprocesseur. Voici généralement ce que l'on trouve à l'intérieur d'un tel composant :

- Un **microprocesseur** (C.P.U.),
- La **mémoire donnée** (RAM et EEPROM),
- La **mémoire programme** (ROM, OTPROM, UVPROM ou EEPROM),
- L'**interface parallèle** pour la connexion des entrées / sorties,
- L'**interface série** (synchrone ou asynchrone) pour l'échange de données avec d'autres unités,
- Des **timers** pour générer ou mesurer des signaux avec une grande précision temporelle,
- Un **convertisseurs Analogique-Digital** (ADC) pour l'acquisition des signaux
- Un **watch dog** (chien de garde) servant à réinitialiser le système en cas de plantage.

En plus de toutes ces unités, il existe des connexions reliant toutes ces unités entre elles ; ce sont les **bus.** Il y'a trois sortes de bus dans un microcontrôleur : le bus de donnée, le bus d'adresse et le bus de contrôle ou de commande.

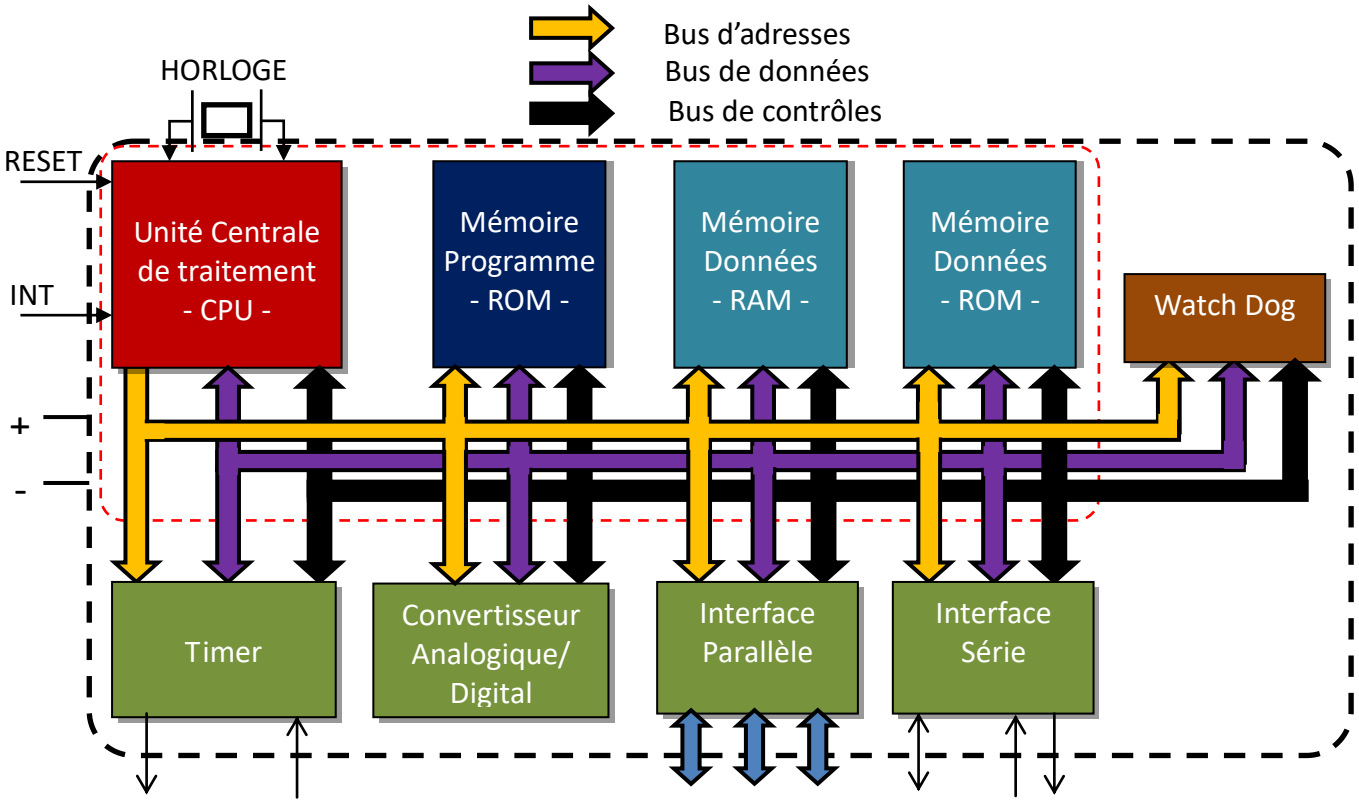


Fig. I.3 : Structure interne d'un microcontrôleur

I.3.1. Le microprocesseur (CPU)

Le microprocesseur est l'unité servant à exécuter de manière séquentielle les instructions d'un programme. Ce programme se trouvant en mémoire (voir fig I.3), Le microprocesseur doit être capable de récupérer les instructions une à une, de les analyser puis de les exécuter et enfin d'enregistrer le résultat également en mémoire. Pour cela il est

constitué d'un certain nombre d'éléments (unités) qui lui permettent de réaliser toutes ces opérations (Fig.I.4.) :

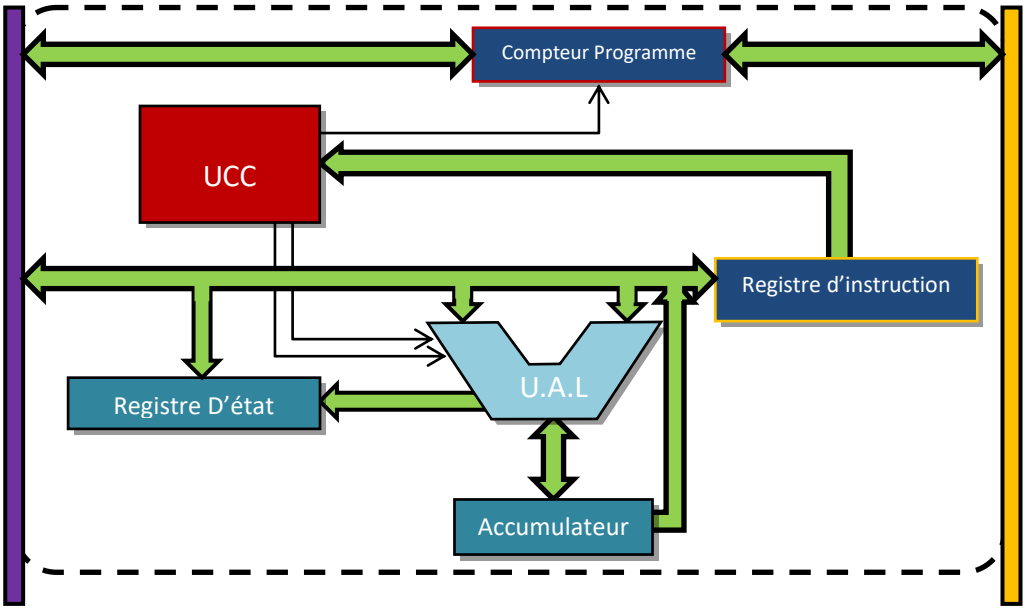


Figure I.4 : structure interne simplifiée d'un microprocesseur

- Un ou plusieurs **registres accumulateurs** contenant temporairement les *opérandes* ainsi que les résultats des opérations. Un registre est tout simplement une sorte de tableau à une seule ligne contenant un certain nombre de bit numérotés de 0 à N (Fig. I.5). ce registre est entièrement accessible à l'utilisateur c'est-à-dire qu'il peut être lu ou écrit à n'importe quel moment. Sa taille correspond à celle d'une donnée traitée par le microprocesseur.
- Des **registres auxiliaires** destinés à la sauvegarde des résultats intermédiaires. Cela évite les accès répétés en mémoire centrale accélérant ainsi l'exécution d'un programme. Ils sont aussi entièrement accessibles à l'utilisateur et leur taille est généralement égale à celle de l'accumulateur.
- Un **registre d'instruction (RI)** destiné à recevoir l'instruction en cours d'exécution. Ce registre n'est pas accessible à l'utilisateur que ce soit en lecture ou en écriture. Sa taille est celle d'une instruction.
- Un **compteur programme (PC)** ou **compteur ordinal (CO)** chargé de pointer sur la prochaine instruction à exécuter. Ce compteur est automatiquement incrémenté pour stocker l'adresse de la prochaine instruction à exécuter. Il doit donc être

suffisamment grand pour donner accès à toute la mémoire. il n'est pas accessible à l'utilisateur.

- Une **unité arithmétique et logique (ALU**) permettant d'effectuer des opérations de type arithmétiques (+,-,÷,×) et logique (and, or, xor,...) entre l'accumulateur et un opérande. Le résultat obtenu est mis dans le registre spécifié dans l'instruction (Généralement l'accumulateur). Cette unité permet de réaliser des opérations sur des données d'une taille fixée par le constructeur 8, 16, 32, 64...bits. on parle alors de microprocesseurs 8, 16, 32, 64... bits respectivement.
- Une **unité de contrôle et de commande (UCC)** chargée d'organier le contrôle et le déroulement de toute l'opération d'exécution des instructions du programme. Ainsi, elle s'occupe de rechercher l'instruction en mémoire, de rechercher les données nécessaires à l'exécution de cette instruction, de préparer l'ALU à l'exécution de l'instruction, de lancer l'exécution, d'incrémenter le compteur ordinal, ...
- Un **registre code condition (CCR) ou registre d'état (SR)** (Fig. I.5) constitué d'un certain nombre de bits (flags) destinés à renseigner le programmeur sur le type de résultat obtenu à la fin de chaque d'instruction (retenu, zéro, interruption...). La figure I.5 montre un exemple de registre d'état ; celui du microcontrôleur PIC 16F877. chaque bit se met à 1 pour indiquer la nature du résultat obtenu.

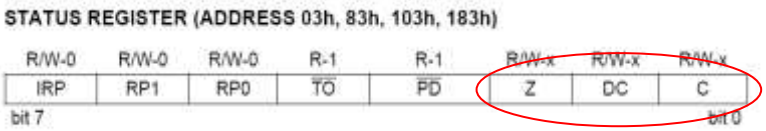


Figure I.5 : exemple de registre : le registre d'état

Bit 2 **Z : bit de zéro**

1 : le résultat de l'opération arithmétique ou logique en cours est nul

0 : le résultat de l'opération arithmétique ou logique en cours n'est pas nul

Bit 1 **DC : retenue sur quartet**

1 : une retenue est générée par le quatrième bit du résultat

0 : pas de retenue générée par le quatrième bit du résultat

Bit 0 **C : retenue**

1 : une retenue est générée par le huitième bit du résultat

0 : pas de retenue générée par le huitième bit du résultat

En marge à cela, on peut noter qu'il existe deux catégories de microprocesseur : CISC et RISC.

CISC (Complex Instruction Set Computer) : Ce microprocesseur possède un nombre important d'instructions (élémentaires). Chacune d'elles s'exécute en plusieurs *périodes d'horloges*. L'avantage est qu'une opération arithmétique complexe (puissance, racine carrée...) peut s'écrire avec un nombre limité d'instructions. Cela conduit à l'écriture de programmes cours mais lent.

RISC (Reduced Instruction Set Computer) : Ce microprocesseur possède un nombre réduit d'instructions. Chacune d'elles s'exécute en une seule *période d'horloge*. L'avantage est la rapidité d'exécution des opérations. Cela conduit à l'écriture de programmes longs mais rapides.

I.3.2 La mémoire

Ce dispositif comme son nom l'indique, sert de *réservoir* d'informations au système. La mémoire contient donc la totalité de la partie logicielle dont le système a besoin pour fonctionner ; ce qui englobe les programmes qui renseignent le système sur les différentes *actions* à réaliser, ainsi que les données sur lesquelles ces actions sont effectuées.

Il existe bien évidemment différents types de mémoires suivant la technologie de réalisation. Cependant, leur architecture externe est toujours la même.

Une mémoire est tout simplement un tableau contenant un certain nombre de cellules disposées en 2^n lignes et en m colonnes (Fig. I.6). Une ligne de cellules est appelée case mémoire et c'est la plus petite partie accessible de la mémoire (que l'on peut lire ou écrire). Le nombre de cases mémoires contenues dans la mémoire nous donne sa taille, par contre le nombre de colonnes nous renseigne sur la taille d'une case et on parle alors de mot mémoire. Un mot mémoire peut être de 1, 8 16, 32... Bits. La taille d'une mémoire est calculée soit en bits, en octet ou en mots elle est généralement une puissance de 2., Chaque case mémoire possède une adresse unique dans le système afin d'être clairement identifiée par le microprocesseur, la première case portant l'adresse 0 et la dernière dépendant de la taille de la mémoire.

La liaison entre le microprocesseur et la mémoire se fait à l'aide des trois bus précédemment cité : le bus d'adresse, composé de *n lignes,* le bus de données, composé de *m lignes* et le bus de contrôle, généralement limité à deux lignes (Fig.I.7) :

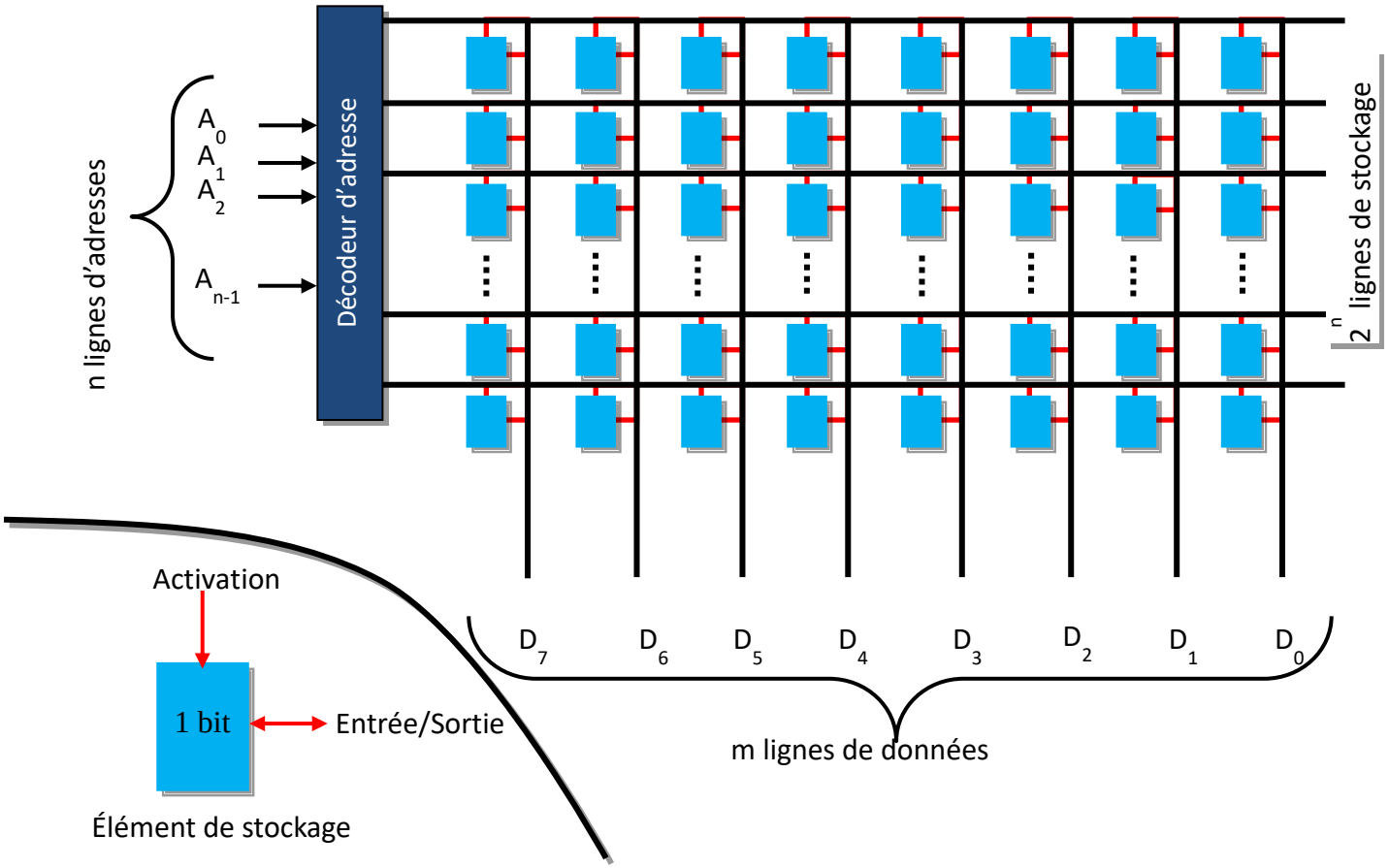


Fig. I.6 : structure matricielle d'une mémoire

$\overline{CS}$: Chip Select, qui sert à activer le boitier mémoire

$\overline{CS}$ = 0 : boitier mémoire activé

$\overline{CS}$ = 1 : boitier mémoire désactivé

R/$\overline{W}$: Read/Write, qui sert à indiquer quel type d'opération veut-on réaliser

R/$\overline{W}$ =1 : opération de lecture de la mémoire

R/$\overline{W}$ = 0 : opération d'écriture de la mémoire

L'accès en lecture de la mémoire se fait selon les étapes suivantes (Fig I.8) :

1- Le microprocesseur transmet, via le bus d'adresse, le numéro de la case dont il veut lire le contenu,

2- Le microprocesseur met la ligne R/$\overline{W}$ à l'état haut pour indiquer une opération de lecture,

3- Le microprocesseur met la ligne $\overline{CS}$ à l'état bas pour activer le boitier mémoire,

4- La mémoire activée, récupère l'adresse sur le bus d'adresse, après un temps non négligeable appelé temps d'accès à la mémoire (T_a), elle met les données sur le bus de donnée.

L'accès en écriture de la mémoire se fait selon les étapes suivantes (Fig I.8) :

5- Le microprocesseur transmet à celle-ci, via le bus d'adresse, le numéro de la case où il veut stocker un contenu,

6- Le microprocesseur met la ligne $\overline{R}/W$ à l'état bas pour indiquer une opération d'écriture,

7- Le microprocesseur met la ligne $\overline{CS}$ à l'état bas pour activer le boitier mémoire,

8- Le microprocesseur met le contenu à stoker sur le bus de données et doit le maintenir stable pendant un temps non négligeable (T_d).

9- La mémoire est activée, récupère l'adresse sur le bus d'adresse, la donnée sur le bus de donnée puis met celle-ci dans la case mémoire indiquée.

Pendant toute l'opération de Lecture et d'écriture, Le bus d'adresse, La ligne $R/\overline{W}$ ainsi que la ligne CS doivent rester stables (garder les mêmes valeurs).

Il est à noter aussi que l'opération de lecture et celle d'écriture décrites ici ne concerne qu'une seule case mémoire à la fois. Pour lire ou écrire un ensemble de donnée, il faut procéder en plusieurs fois, c'est-à-dire utiliser plusieurs cycles.

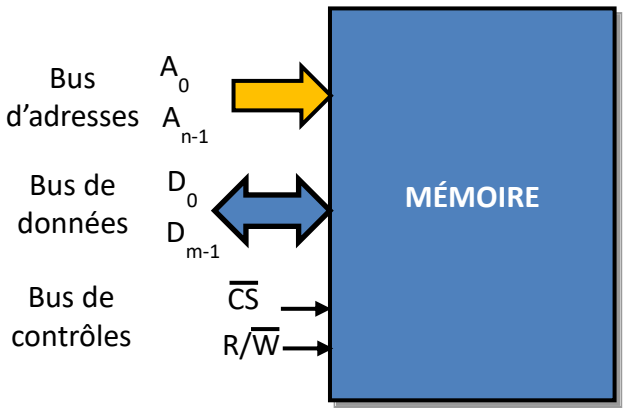


Fig. I.7 : architecture externe de la mémoire

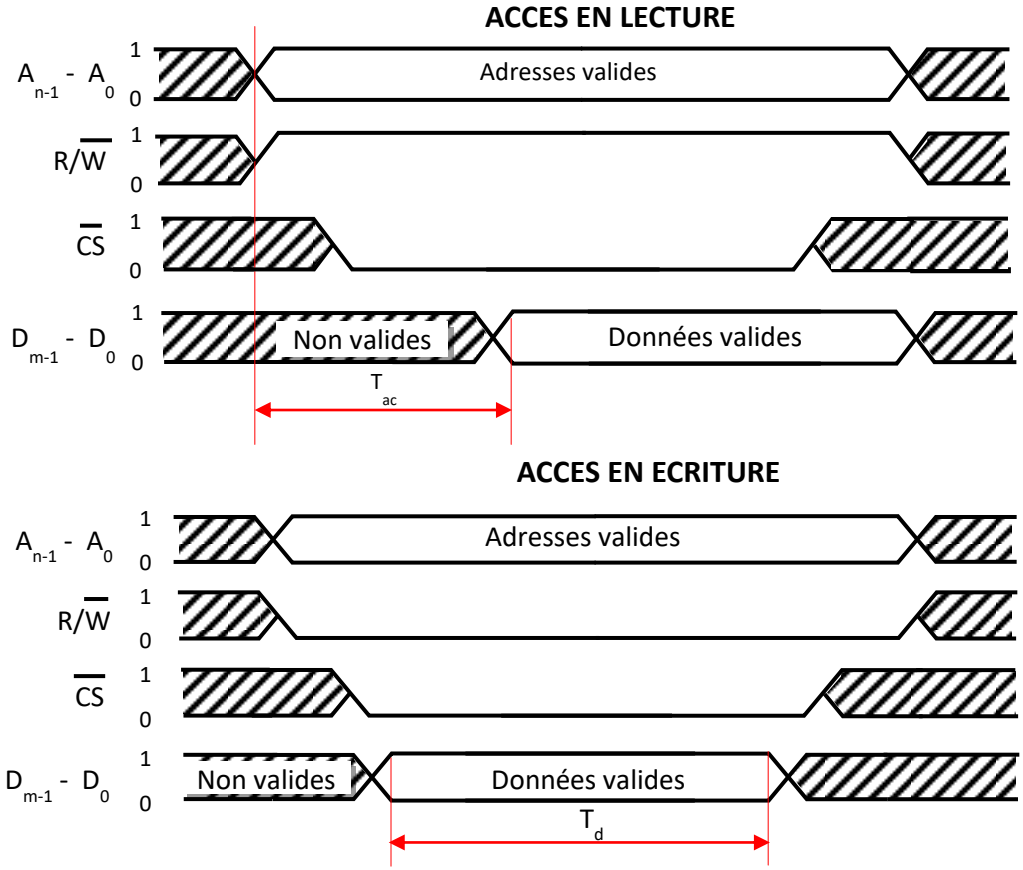


Figure I.8 : Chronogrammes d'accès en mémoire

I.3.2.1 Technologies des mémoires

Technologiquement, il existe trois types de mémoires classées suivant le type d'accès : les mémoires à accès aléatoire dites mémoire vives (RAM), les mémoires à accès en lecture seule dites mémoires mortes (ROM) et les piles.

a- Les mémoires mortes

Les mémoires mortes, aussi appelées ROM (Read Only memory : Mémoires à lecture seule), sont des mémoires dont le contenu ne peut être modifié ou, dans certain cas, modifiable mais en suivant une séquence particulière appelée programmation. Ce type de mémoire est généralement destiné à la sauvegarde de données sensibles dont la modification n'est pas souhaitée. Elle peut également servir pour la sauvegarde du programme que le système va exécuter (mémoire programme). Il existe plusieurs type de ces mémoires suivant leur mode de programmation :

- ROM dont le contenu est programmé lors de sa fabrication (par le constructeur).
- PROM (Programmable ROM) programmable électriquement une seule fois par le développeur (appelée aussi OTPROM : Once Time PROM),

- EPROM (Erasable PROM) programmable électriquement et effaçable aux rayons ultraviolets (appelée aussi UVPROM),
- EEPROM (Electrically EPROM) programmable et effaçable électriquement.

L'avantage de ce type de mémoires est que les données se trouvent définitivement enregistrées et ce, même en absence de l'alimentation électrique. L'inconvénient, par contre, est, d'une part le temps d'accès (T_a) qui est important et, d'autre part, le nombre de cycles écriture/lecture qui est limité ; on parle alors de durée de vie de la mémoire.

b- Les mémoires vives

Contrairement aux mémoires mortes, les mémoires vives appelée aussi RAM (Random Access Memory : Mémoire à accès Aléatoire), dont le temps d'accès(T_a) est très faible, peuvent être lues et écrites sans aucune contrainte. Ceci permet leur utilisation pour l'enregistrement des données générée par le microprocesseur pendant des calculs. Cependant, ce genre de mémoire perd totalement sont contenu lors d'une coupure de l'alimentation électrique.

Il en existe deux types :

- RAM statique : dont le contenu est stable durant le fonctionnement.
- RAM dynamique : dont le contenu se vide durant le fonctionnement. Elle nécessite alors une opération spéciale appelée rafraîchissement afin de mettre à jours les données.

Sur le schéma de la figure I.3 on trouve une mémoire programme constitué d'une mémoire morte et une mémoire donnée constituée à la fois d'une mémoire morte et d'une mémoire vive (c.f. § I.4)

c- Les piles

Figure I.9 : Les Piles

Les piles sont un type de mémoire à accès séquentiel, c'est-à-dire qu'elle ne possède pas de bus d'adresse. Il existe deux types des piles (Figure I.9) :

Les piles FIFO (First In First Out – Premier Entré Premier Sorti) : Ce genre de piles, les donnée sont stockées l'une après l'autre à travers un Bus de données d'entrée et elles sortent à travers un autre bus de données, de telle sorte que la première donnée entrée est celle qui sort en premier.

Les piles LIFO (Last In First Out – Dernier Entré Premier Sorti) : Ce genre de pile est utilisé dans les microprocesseurs pour stocker l'adresse de retour lors des interruptions ou lors d'appels de fonctions (sous programmes). Les données entrent et sortent à travers le même bus de telle sorte que la dernière donnée entrée est celle qui sort en premier.

I.3.3 Les Bus

Dans un système à base de microprocesseur, les différents *signaux* sont véhiculés par des lignes regroupées sous formes de nappes de fils ou de pistes de cuivre sur le circuit imprimé et desservent la totalité du système reliant ainsi tous ses composants (voir figure I.3). Ces lignes appelées bus (pour des raisons évidentes : un bus est un moyen servant à transporter des voyageurs d'un point à un autre d'une ville en passant par plusieurs stations) sont aux nombres de trois type : le bus d'adresses, le bus de données et le bus de commandes.

Le bus d'adresse est utilisé afin de véhiculer l'adresse (l'endroit) en mémoire d'une instruction ou d'une donnée. Il est aussi utilisé pour indiquer l'adresse d'un périphérique ; ce dernier étant pour le microprocesseur une simple case mémoire ou un ensemble de cases mémoires. La largeur (la taille) du bus d'adresse doit être suffisante pour accéder à toute la mémoire présente sans oublier les différents périphériques. Le bus d'adresse est un bus monodirectionnel ce qui veut dire qu'il transporte les adresses dans un seul sens : **du microprocesseur vers la mémoire ou les périphériques.**

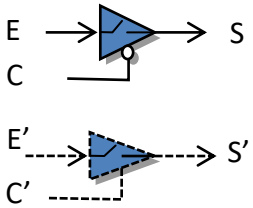
Le bus de données, quant à lui, est destiné à transporter les données, c'est à dire le contenu de la mémoire vers le microprocesseur ou l'inverse ; c'est un bus bidirectionnel. Sa taille est égale à celle d'une donnée traitée par l'ALU.

Le bus de commandes est un bus spécifique dans le sens que chacune de ses lignes possède un rôle bien précis. Par exemple on peut trouver dans ce bus une **ligne d'écriture** utilisée uniquement quand le microprocesseur veut écrire en mémoire, on peut aussi trouver une **ligne de validation** qui permet au microprocesseur d'indiquer à la mémoire (ou

au périphérique) que les données sont valides... Le nombre de ligne de ce bus varie suivant les constructeurs et les modèles de microprocesseurs.

Il est important de signaler ici que dans un système à base de microprocesseur, à un instant donné, seul deux éléments sont en *connexion*, c'est-à-dire qu'ils peuvent échanger des données : le microprocesseur et une autre unité (voir fig I.3). Pour rendre cela possible et ainsi éviter des *conflits* de transfert, les lignes des différents bus possèdent **trois états** : les deux états logiques (haut et bas) et un état **haute impédance** où le bus est hors connexion.

L'élément de base qui constitue le bus est la porte trois états. Une telle porte est représentée sur la figure I.10 avec et sans inversion de l'entrée de commande. Les deux tables de vérité indiquent le fonctionnement des deux types de portes



C	S
0	Déconnectée
1	E

C'	S'
0	E'
1	Déconnectée

Figure I.10 : Portes trois états

Le bus d'adresse est donc constitué à chacun de ses terminaisons par une porte dont l'entrée de commande est inversée de telle sorte que le bus interne de la mémoire n'est connecté au bus externe que si le boitier est activé (Figure I.10).

Le bus de commande par contre est un bus bidirectionnel, c'est-à-dire que les données peuvent transiter dans les deux sens. Dans ce cas, une combinaison de deux portes en tête-bêche est nécessaire pour contrôler le bus. (Figure I.10)

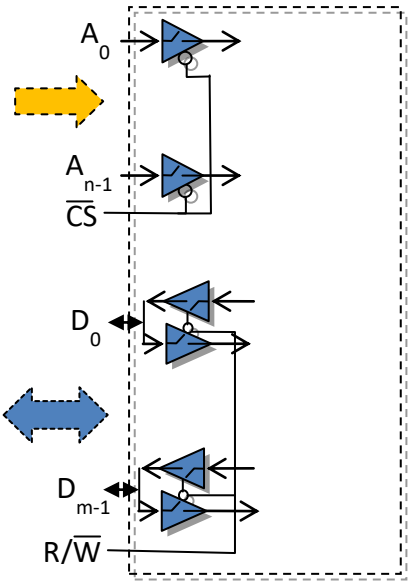


Figure I.11 : Représentation des bus internes d'un boitier mémoire

I.4. Architectures des microprocesseurs

Il existe deux architectures différentes pour les microprocesseurs. La première est connue sous le nom d'**architecture Von Neumann** (1946) du nom du chercheur qui l'a mise au point. Elle se caractérise essentiellement par un seul processeur (unité centrale) qui travaille de manière séquentielle sur des informations en mémoire centrale qui constituent à la fois les données et les programmes : Un seul bus de donnée sert à transférer les instructions et les données. Le CPU a donc besoins de deux accès, au minimum, à la mémoire centrale pour exécuter une instruction. L'avantage de cette architecture est sa simplicité de réalisation. Elle est à la base de la réalisation des PC actuels (Figure I.12).

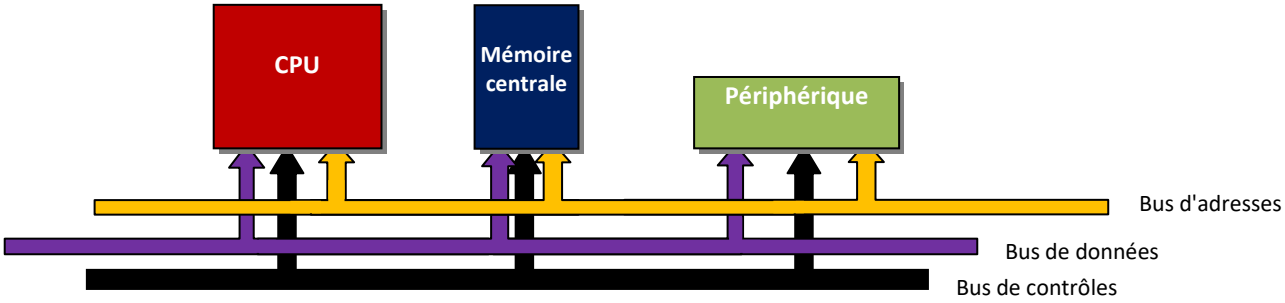


Fig.I.12. : Architecture Von Neumann

La seconde est connue sous le nom d'**architecture Harvard** (1940). C'est la première architecture à être utilisée pour les ordinateurs. Elle est caractérisée par la séparation des données et programmes sur deux mémoires différentes. L'accès à chacune s'effectue par un bus séparé. Un accès simultané aux instructions et aux données est donc possible, ce qui implique une exécution plus rapide (un seul accès aux deux mémoires permet d'exécuter une instruction). Cette architecture est abandonnée sur les ordinateurs personnels en raison de sa complexité matérielle (nécessité de plusieurs bus), mais elle est avantageuse pour les systèmes de traitement numérique de signaux. Le 4004 (premier processeur Intel) relève de cette architecture, de même que la plupart des microcontrôleurs et processeurs DSP.

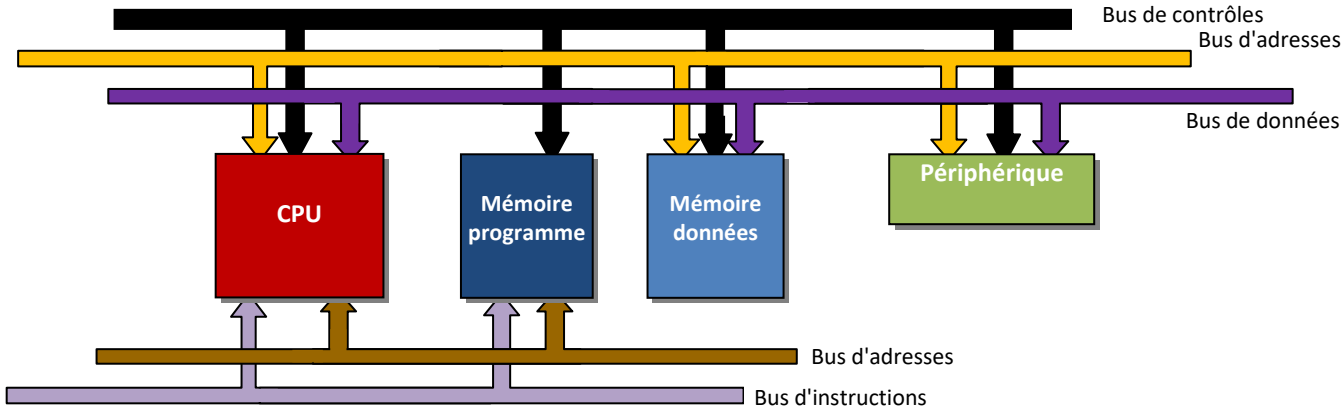


Fig.I.13 : Architecture Harvard

I.5. Format d'une instruction en mémoire

Les instructions exécutées par le microprocesseur sont situées en mémoire sous forme d'une suite de codes binaires de format fixe de telle sorte qu'elles puissent renseigner sur un certain nombre d'informations. Pour cela, une instruction est divisée en deux champs distincts : le champ code opération et le champ opérande (Fig.I.14)

Fig.I.14 : représentation d'une instruction

Le code opération est unique dans une instruction et est codé sur un certain nombre de bit. Ce code doit indiquer :

- la nature de l'opération (addition, soustraction, comparaison,…),
- Le nombre d'opérandes,
- L'adressage des données ; c'est-à-dire la manière dont on doit accéder aux données.

Il existe plusieurs types d'adressage :

- **Adressage immédiat** : où l'opérande indique la donnée elle même
- **Adressage direct** : où l'opérande indique l'adresse de la donnée. Un déplacement en mémoire donnée est donc nécessaire pour récupérer la donnée.
- **Adressage indirect** : où l'opérande indique l'adresse en mémoire ou se trouve l'adresse de la donnée. Ici on doit effectuer deux (02) déplacements en mémoire pour chercher la donner : une première fois pour chercher l'adresse de la donnée et une seconde fois pour chercher la donnée.
- **Adressage relatif** : où l'opérande indique un décalage (positif ou négatif) à additionner à l'adresse de la prochaine instruction à exécuter pour trouver l'adresse ciblée. Cet adressage est uniquement utilisé dans les branchements.
- **Adressage implicite** : où le champ opérande n'existe pas. Le code opération suffit de lui-même pour exécuter l'instruction

Le champ opérande peut contenir plusieurs opérandes qui peuvent être les données elles-mêmes, les adresses des données, le nom d'un registre, un incrément, ou rien, suivant le type d'adressage (voir le code opération)

Chaque type de microprocesseur possède sont propre codage des instructions. Certains microprocesseurs acceptent des instructions de longueur variable qui peuvent être codés sur plusieurs mots mémoire. Dans ce cas, une instruction est enregistrée en mémoire sur plusieurs cases (Fig.I.15-a). Dans le cas où le microprocesseur utilise des instructions de longueur fixe, on doit utiliser une mémoire dont le mot correspond exactement à la taille d'une instruction (Fig.I.15-b)

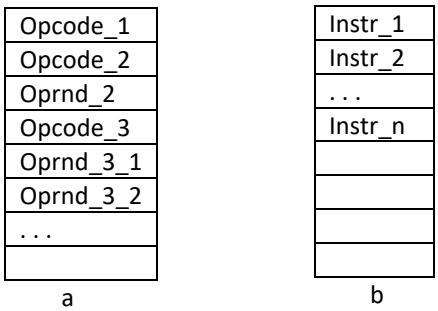


Fig.I.15 : différentes façons de mémoriser un programme

I.6. cycle d'exécution d'une instruction

Le traitement par le microprocesseur d'une instruction en mémoire nécessite un certain temps qui est compté en nombre de cycles d'horloge. Ce temps est appelé **cycle d'exécution d'une instruction**. Les instructions simples nécessitent un cycle plus court que les instructions complexes. Le temps d'exécution de l'instruction la plus rapide est appelé **cycle machine.** Le cycle d'exécution d'une instruction peut être décomposé en trois phases.

- La phase de recherche de l'instruction.
- La phase d'analyse et de recherche d'opérandes
- La phase d'exécution de l'opération

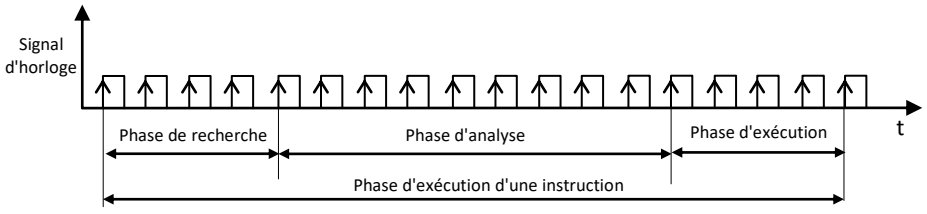


Fig.I.15. : Représentation du cycle d'exécution d'une instruction

I.6.1.la phase de recherche

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.

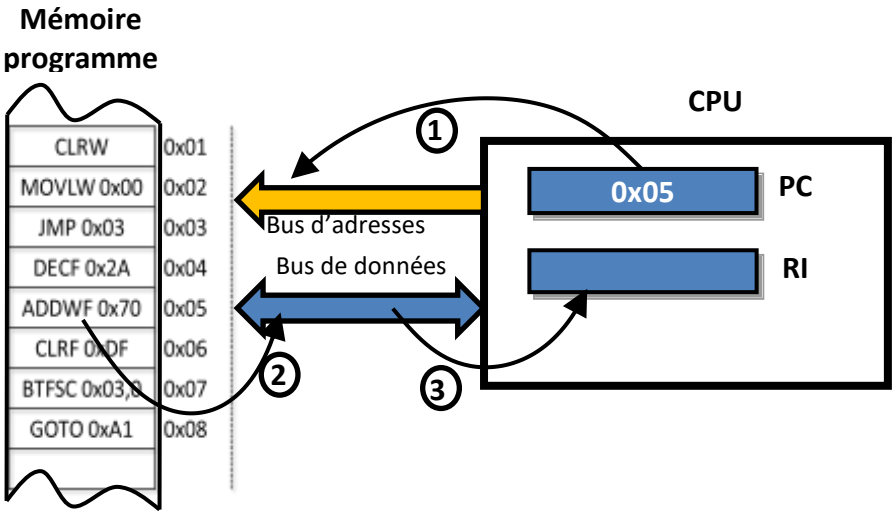


Fig.I.16. : phase de recherche

I.6.2. la phase de décodage et recherche d'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.
4. le compteur programme est incrémenté pour réaliser le cycle de recherche suivant.

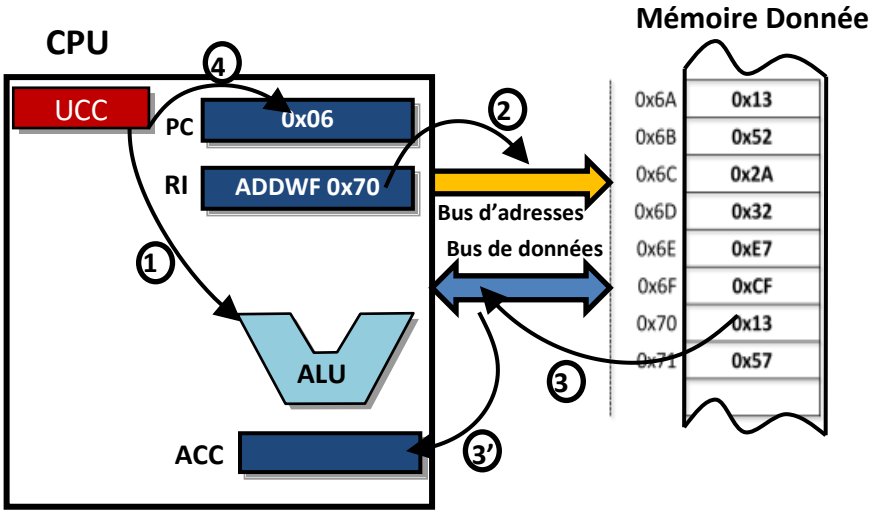


Fig.I.17. : Phase de décodage

I.6.3. Exécution de l'instruction

1. Le microprogramme réalisant l'instruction est exécuté. S'il s'agit d'une instruction de branchement, le champ adresse est alors transféré dans le PC (1')
2. Les indicateurs du registre d'état (RE) sont mis à jour.

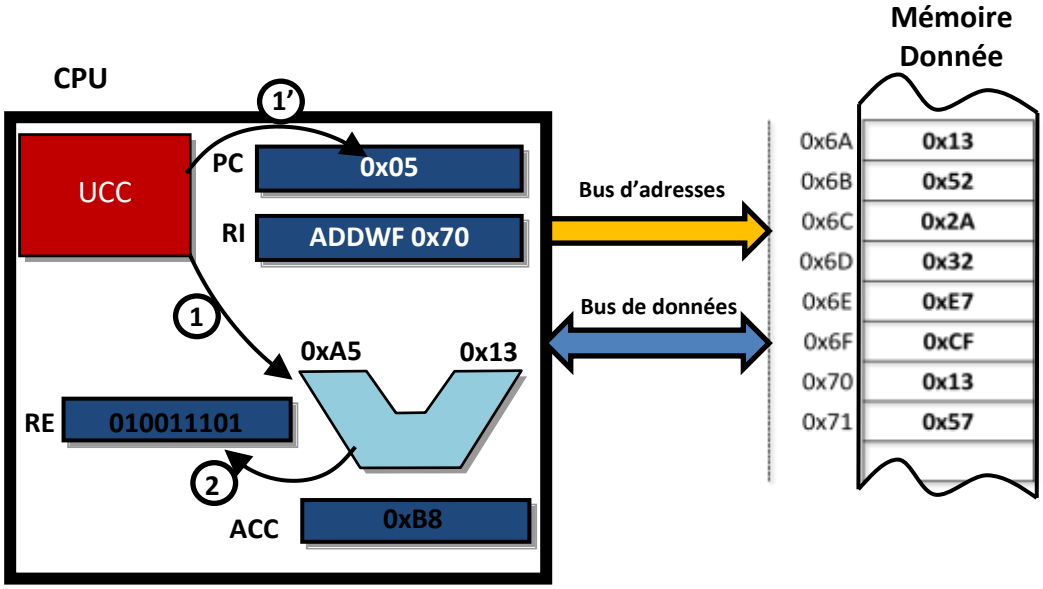


Fig.I.18. : Phase d'exécution

I.7 Le vecteur RESET et le vecteur d'interruption

I.7.1 le vecteur RESET

Le démarrage des composants programmables (microprocesseurs, microcontrôleurs,…) se fait à l'aide d'une séquence automatique que le constructeur a rendu possible. Le but c'est d'amener le microprocesseur, initialement éteint, à exécuter le programme se trouvant à une adresse quelconque de la mémoire.

Quand on branche le microprocesseur à une source d'énergie électrique et qu'on lui fournit un signal d'horloge, il effectue la séquence automatique suivante qui lui permet de démarrer.

1- Au premier cycle d'horloge, le compteur ordinal (P.C) est initialisé à une valeur nulle, ou à sa valeur maximale (suivant le constructeur). Il pointe alors, respectivement, soit en début de mémoire, soit à la fin. Cet endroit de la mémoire est appelé vecteur RESET, car c'est là que se branche le compteur ordinal à chaque démarrage.

2- Au second cycle d'horloge, le microprocesseur est prêt. Il charge alors l'instruction qui se trouve en début de mémoire (ou à la fin), l'exécute, puis passe à l'instruction suivante et ainsi de suite.

Pour forcer le microprocesseur à se brancher vers le programme de l'utilisateur, qui peut se trouver à une adresse quelconque de la mémoire, il suffit de mettre, dans le vecteur RESET, une instruction de branchement vers la première instruction du programme utilisateur Fig.I.19.

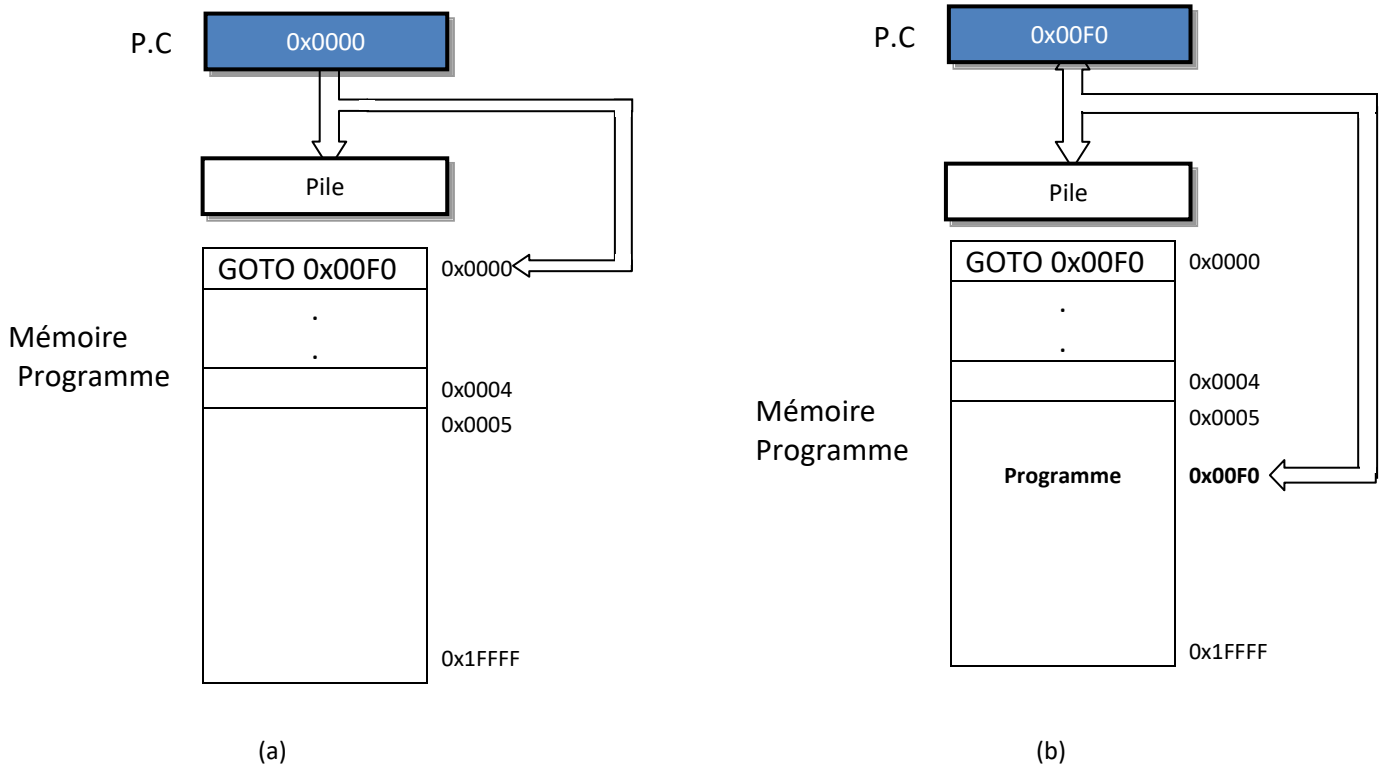
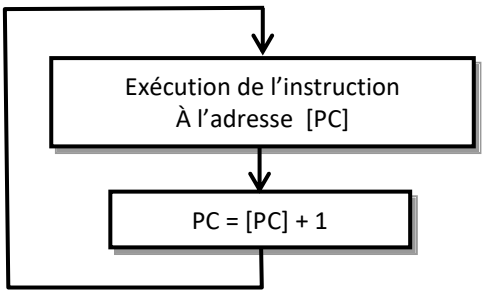


Fig.I.9 : Démarrage du CPU (PIC 16F877), (a) branchement au vecteur RESET, (b) saut vers le programme principal

Le CPU continu alors de fonctionner suivant l'organigramme suivant :



Il faut remarquer le fonctionnement du CPU ne s'arrête pas et qu'il n'existe pas d'instruction "*fin de programme*". Mais on peut suspendre l'exécution d'un programme de manière momentanée ; dans le cas de l'exécution pas à pas, d'une mise en veille ou dans le cas, le plus générale, d'une *interruption*.

I.7.2 Le vecteur d'interruption

La communication entre un périphérique externe (modem par exemple) et un ordinateur (système) se fait par interruption : quand le modem veut transmettre des données à l'ordinateur il demande une "*autorisation*" (une requête d'interruption) à travers une ligne spéciale destinée à cet effet. Cette "*autorisation*" s'appelle interruption. Ce signal va donc s'adresser directement au CPU pour lui demander d'arrêter d'exécuter le programme principal et passer à l'exécution d'un programme appelé programme de gestion d'interruption qui permettra la gestion du transfert des données entre l'ordinateur et le modem.

Cette façon de faire permet au CPU d'exécuter plusieurs programmes différents, chacun relatif à un périphérique, mais seulement quand le périphérique le lui demande. Lorsque plusieurs périphériques demandent simultanément une interruption, le CPU procède selon un ordre de priorité préalablement établit.

A la fin du programme de gestion d'interruption, le CPU revient pour continuer le programme principal à l'endroit exact où il l'a quitté.

Le problème c'est que le programme d'interruption se trouve à un endroit quelconque de la mémoire dont uniquement l'utilisateur connait l'adresse. Le constructeur a donc prévu une séquence automatique (comme dans le cas du RESET) pour amener le CPU à exécuter ce programme d'interruption. Cette séquence fait en sorte d'amener le CPU à se brancher à une adresse spéciale appelée vecteur d'interruption où une instruction de branchement vers le programme d'interruption est placée par l'utilisateur. La séquence complète est donnée par l'algorithme suivant

1- Terminer l'instruction en cours, stoker le résultat
2- Masquer (interdire) les interruptions dont la priorité est plus faible que celle en cours
3- Stoker le contenu du compteur ordinal (PC) dans la pile (adresse de retour)
4- Se brancher au vecteur d'interruption

L'utilisateur doit donc placer une instruction vers le programme d'interruption dont lui seule connaît l'adresse.

Dans le cas du PIC16F877 le vecteur d'interruption se trouve à l'adresse 0x0004.

A la fin du programme d'interruption le CPU doit retourner au programme principal à l'endroit exact où il l'a quitté. Cela est possible car avant de quitter le programme principal, le contenu du PC (adresse de la prochaine instruction) a été stocké dans la pile.

Une instruction spéciale (RETFIE : Return From Interrupt) est donc placée à la fin du programme d'interruption pour informer le CPU qu'il droit retourner au programme principal. La séquence automatique de retour au programme principal est donnée par l'algorithme suivant :

1- Exécution de l'instruction RETFIE
2- Autoriser les interruptions
3- récupérer l'adresse de retour à partir de la pile et la charger dans le PC

Avec les deux séquences précédentes, le CPU peut gérer plusieurs périphériques sans que cela ne gêne l'exécution du programme principal ; le CPU quitte et revient au programme principal comme si aucune interruption n'a été observée.