ELECTRONIQUE NUMERIQUE	PREM/TERM

LE MICROCONTROLEUR

1. INTRODUCTION

Le microcontrôleur est au microprocesseur ce qu'est le singe à l'homme : un cousin. Il est un peu plus bête mais en réalité beaucoup plus utilisé ; il existe tout autour de vous et vous simplifie bien la tâche lorsqu'il est dans votre téléphone portable, votre voiture (air-bag, tableau de bord...) ou votre chaîne HI-FI.

Vous l'avez compris, le microcontrôleur est taillé pour l'embarqué. Il est beaucoup plus facile à mettre en œuvre qu'un processeur (*essayez de construire votre carte mère*) et les dernières versions atteignent des performances remarquables. Il est conçu pour se suffire à lui-même ; un pentium a besoin de périphériques pour pouvoir fonctionner contrairement au microcontrôleur qui lui possède sa propre RAM, sa ROM et le programme pour exécuter les tâches, tel un système d'exploitation.

Nous voyons donc que ses possibilités sont quasi infinies et ses domaines d'application sont très larges ; c'est pourquoi il est tant apprécié des électroniciens amateurs. Dans quelque temps, vous serez capable de programmer ce type de processeur et vous pourrez créer vos propres applications. Une première partie sera réservée à une petite introduction sur l'historique et la présentation des principaux microcontrôleurs, puis suivra un rappel d'électronique et enfin les cours proprement dits. Une fois les bases assimilées, nous appliquerons nos connaissances avec de petites applications bien sympathiques (écrans LCD, pavé numérique, robotique (pour ceux qui choisiront ce thème en TPE) etc.) relativement peu chers.

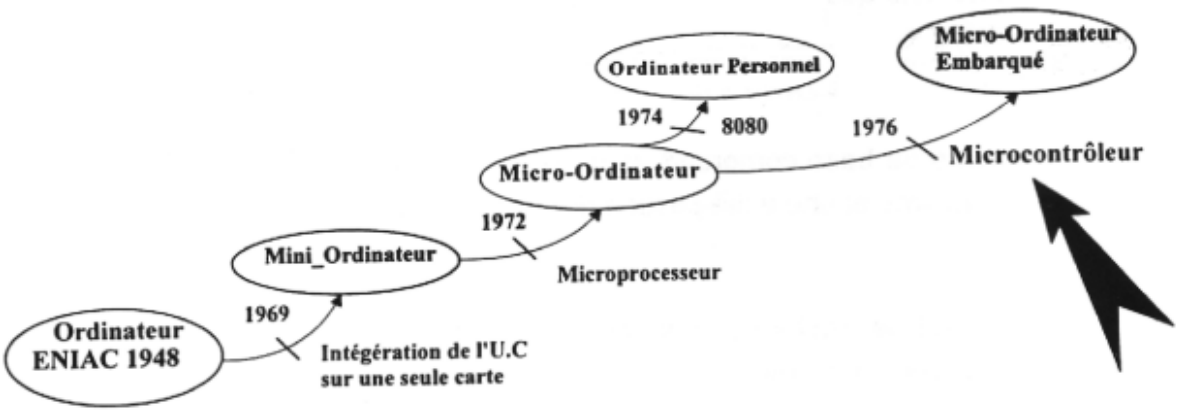
2. UN PEU D'HISTOIRE : FLASBACK

Un petit retour en arrière, dans les années 40. Le premier ordinateur fut l'ENIAC, apparut en 1945. Il coûtait 10 millions de dollars, contenait 20 000 tubes, pesait 30 tonnes, consommait 0,2 MWatts et occupait une pièce de 150m². Chacune des unités fonctionnelles tenait sur plusieurs cartes électroniques. L'évolution des technologies va permettre la miniaturisation et l'intégration de plus en plus de fonctions sur une surface de plus en plus petite.
Tout d'abord, l'intégration se fera au niveau des cartes électroniques puis au niveau des circuits intégrés.

Anecdote : Savez-vous d'où vient le mot "bug" ? "Bug" signifie punaise en anglais. A l'époque des premiers ordinateurs, les cartes électroniques chauffaient tellement que cela attirait ces satanées bestioles. Malheureusement pour elles, l'électricité n'a rien de confortable et elles grillaient facilement au contact d'un composant, provoquant de fâcheux courts-circuits. Sauf qu'aujourd'hui, les bugs sont plus souvent logiciels…

2.1. De l'ordinateur au microcontrôleur

Le diagramme ci-dessous montre les grandes étapes de l'évolution. Chaque étape a été marquée par un processeur différent



L'avènement des transistors (1958), des circuits intégrés (1968) et des circuits intégrés à haute densité (1978) a permis de construire des ordinateurs de plus en plus petits et de plus en plus puissants.
Chacune des intégrations laisse leurs traces dans les différents noms que l'on rencontre aujourd'hui : mini-ordinateur, microprocesseur, micro-ordinateur, microcontrôleur... On les doit principalement à la société Intel fondé en 1968.
Voici les grandes étapes d'évolution :

*** Mini-Ordinateur** : première intégration, cet ordinateur contient maintenant une Unité Centrale de traitement sur une seule carte.

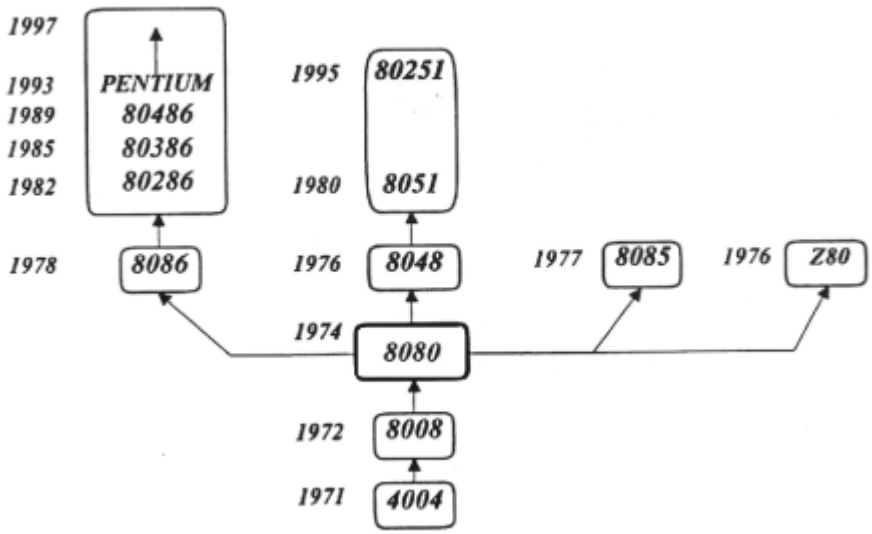
*** Microprocesseur** : deuxième intégration, l'Unité Centrale est contenue dans un seul circuit, c'est le C.P.U. . Ce fut le 4004 en 1971, le 8008 en 1972 puis le 8080 en 1974. Ce dernier eut un tel succès qu'il est considéré comme le père de nos microprocesseurs et microcontrôleurs d'aujourd'hui.

*** Micro-Ordinateur** : c'est le nom qui désignera désormais un ordinateur conçu autour d'un microprocesseur.

*** Microcontrôleur** : troisième intégration, l'ensemble des 3 unités (centrale, mémoires et périphériques) sont maintenant contenues dans un seul circuit. Ce fut le 8048 en 1976 puis le 8051 en 1980.

*** Micro-ordinateur embarqué** : C'est un ordinateur spécialisé et conçu autour d'un microcontrôleur. Il est capable de réagir à des évènements plus rapides.

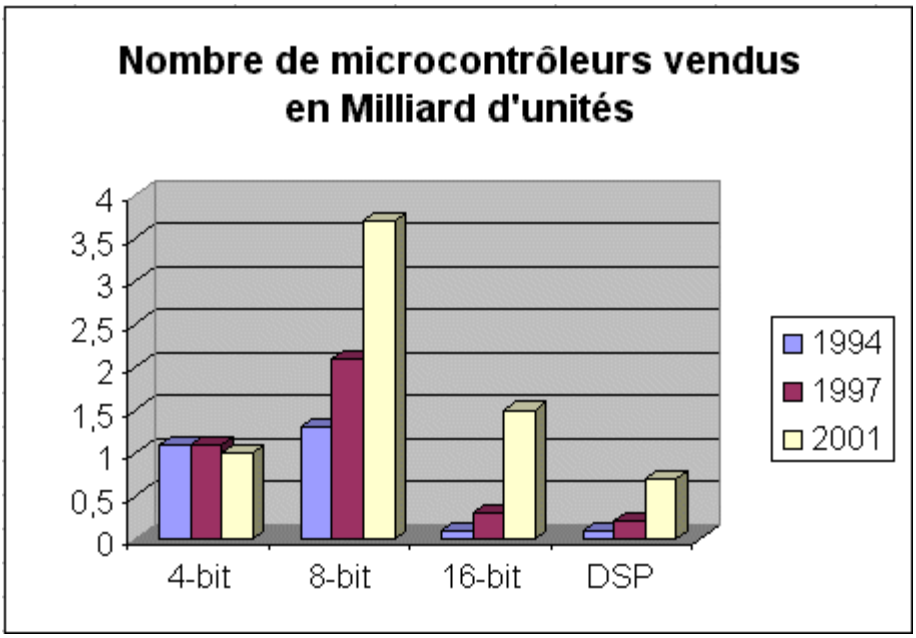
Deux noms sont à l'origine des plus fameux microcontrôleurs utilisés : Intel et Motorola, avec deux architectures différentes. En terme électronique, on parlera de cœur 8051 ou cœur 68HC11 par exemple.



L'arbre généalogique ci-dessus vous présente l'évolution de la grande famille Intel. *Une branche nous est assez familière...*

2.2. Les différents modèles

Il existe une multitude de modèles et une personne non avertie aura bien du mal à trouver ce qu'elle cherche. Mais ce vaste choix permet de trouver le processeur le plus adapté à ce que l'on veut faire. Les microcontrôleurs ne sont généralement pas très chers, comparé aux processeurs type Pentium. Les plus courants sont entre 70 et 100 F, tandis que les plus chers atteignent plus de 800 F.
Tous les microcontrôleurs se distinguent par une seule grande caractéristique : la taille de sa mémoire de calcul, exprimée en bit. C'est un peu comme les consoles en gros. Le diagramme ci-dessous vous montre l'utilisation des types de microcontrôleurs dans le monde.



Le secteur 4-bit est en voie de disparition car il revient aussi cher que de produire des 8-bit, plus performants. De plus, les produits évoluent et les options rajoutées augmentent la taille des programmes ainsi que le nombre d'entrées/sorties. Les microcontrôleurs de type 8-bit sont les plus répandus, car ils allient faible prix et puissance de calcul. Cependant, son avenir est menacé par de nouvelles architectures d'une puissance monstrueuse, les 16-bits et 32-bits de certains fabricants (Hitachi avec son célèbre H8 par exemple). Les microcontrôleurs DSP (**D**igital **S**ignal **P**rocessor) doivent leur succès à leur puissance de calcul dans le traitement du signal. C'est ce type de composant qui se trouve dans votre carte son par exemple.

2.3. A quoi sert un microcontrôleur ?

Les systèmes intelligents sont en augmentation dans tous les domaines de la vie quotidienne. Voici des exemples d'utilisation des microcontrôleurs :

* **Télécommunications** : cartes FAX et MODEM, Minitel, téléphones portables (interfaces homme machine, gestion d'écrans graphiques)...

*** Industriels** : automates programmables, contrôle de processus divers, supervision...

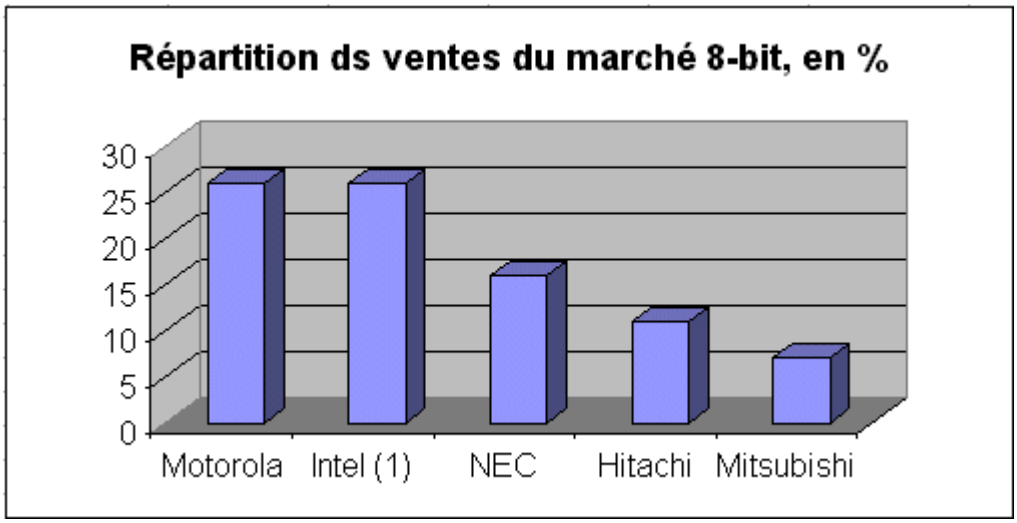
*** Commercial** : électroménager, domotique...

*** Automobile** : ABS, tableau de bord, contrôle des sièges, des vitres...

*** Militaire et spatial** : sonde, lanceurs de fusées, missile, robots...

*** Loisirs** : expo-sciences Réunion ; concours E=M6 *(beaucoup !!! J)*

Ce qui va nous intéresser ici c'est l'architecture 8-bit. C'est d'ailleurs celle la plus étudiée dans les écoles (Bac, BTS, IUT, Grandes Ecoles...). Pour clore ce chapitre, voyons un peu plus en détail la grande famille des microcontrôleurs 8-bit.



Le reste est partagé entre Microchip (avec son PIC), Thomson, etc... Je pense cependant que ces chiffres sont un peu vieux car cela a beaucoup évolué ces derniers temps.
(1) : Je parle d'Intel en tant qu'inventeur du cœur 8051, car en fait plusieurs constructeurs l'utilise : Philips, Intel, Siemens, Matra MHS, Atmel, Dallas.
Ce dernier point est important : en effet, Motorola a inventé une architecture propriétaire et est donc le seul à fabriquer des microcontrôleurs type 68HC11 mais aussi 68HC05, 12, 16... l'offre dans chaque catégorie est importante, il y a vraiment beaucoup de modèles différents mais le fait que Motorola soit le seul constructeur provoque quelques fois une pénurie de composants.
Le cas de Intel est différent : il a inventé l'architecture 8051 et vit maintenant des licences qu'il vend aux autres constructeurs pour avoir le droit d'utiliser le coeur. *C'est le marketing Intel...* L'offre en 8051 est énorme et les constructeurs fabriquent des montres avec cette architecture. Ils viennent ainsi y greffer un tas de périphériques comme un convertisseur analogique/numérique, de la mémoire FLASH, ports d'entrées/sorties supplémentaires etc. On peut citer par exemple le 89C51RD+ avec ses 64Ko de mémoire Flash ou le 80C537 de Siemens, son éléphant à 8 bits comme le surnomme le constructeur. Vous pouvez retrouver la documentation et toutes les informations de chaque composant sur le site des constructeurs.
Je terminerai ce chapitre par une présentation de la famille 68HC11. Voici le tableau que l'on peut trouver sur le site de Motorola :

Product	ROM (KBytes)	RAM (Bytes)	EPROM/OTP (KBytes)	EEPROM (Bytes)	Timer	Serial	A/D	PWM	Operating Voltage (V)	Bus Frequency (Max) (MHz)
68HC11D0	-	192	-	-	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI, SPI	-	-	3.0, 5.0	3
68HC11D3	4	192	-	-	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI, SPI	-	-	3.0, 5.0	3
68HC11E0	-	512	-	-	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI, SPI	8-CH 8-Bit	-	3.0, 5.0	3
68HC11E1	-	512	-	512	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI, SPI	8-CH 8-Bit	-	3.0, 5.0	3
68HC811E2	-	256	-	2048	16-Bit, 3IC, 4OC, RTI, pulse accumulator	SCI SPI	8-CH 8-Bit	-	5	2
68HC11E9	12	512	-	512	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI, SPI	8-CH 8-Bit	-	3.0, 5.0	3
68HC11E20	20	768	-	512	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI, SPI	8-CH 8-Bit	-	5	3
68HC11F1	-	1	-	512	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI SPI	8-CH 8-Bit	-	3.0, 5.0	5
68HC11K0	-	768	-	-	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI+ SPI	8-CH 8-Bit	4-CH 8-Bit or 2-CH 16-Bit	3.0, 5.0	4
68HC11K1	-	768	-	640	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI+ SPI	8-CH 8-Bit	4-CH 8-Bit or 2-CH 16-Bit	3.0, 5.0	4
68HC11K4	24	768	-	640	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI+ SPI	8-CH 8-Bit	4-CH 8-Bit or 2-CH 16-Bit	3.0, 5.0	4
68HC11KS2	-	1	32	640	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI+ SPI	8-CH 8-Bit	-	5	4

68HC11KW1	-	768	-	640	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI+ SPI	10-CH 10-Bit	4-CH 8-Bit or 2-CH 16-Bit	5	4
MC68HC11P1	-	1	-	640	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	Triple SCI SPI	8-CH 8-Bit	4-CH 8-Bit or 2-CH 16-Bit	5	4
68HC11P2	32	1	-	640	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	Triple SCI SPI	8-CH 8-Bit	4-CH 8-Bit or 2-CH 16-Bit	5	4
68HC711D3	-	192	4	-	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI SPI	-	-	5	3
68HC711E9	-	512	12	512	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI SPI	8-CH 8-Bit	-	3.0, 5.0	4
68HC711E20	-	768	20	512	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI SPI	8-CH 8-Bit	-	5	4
68HC711KS2	-	1	32	640	16-Bit, 3/4IC, 4/5OC, RTI, pulse accumulator	SCI+ SPI	8-CH 8-Bit	-	5	4

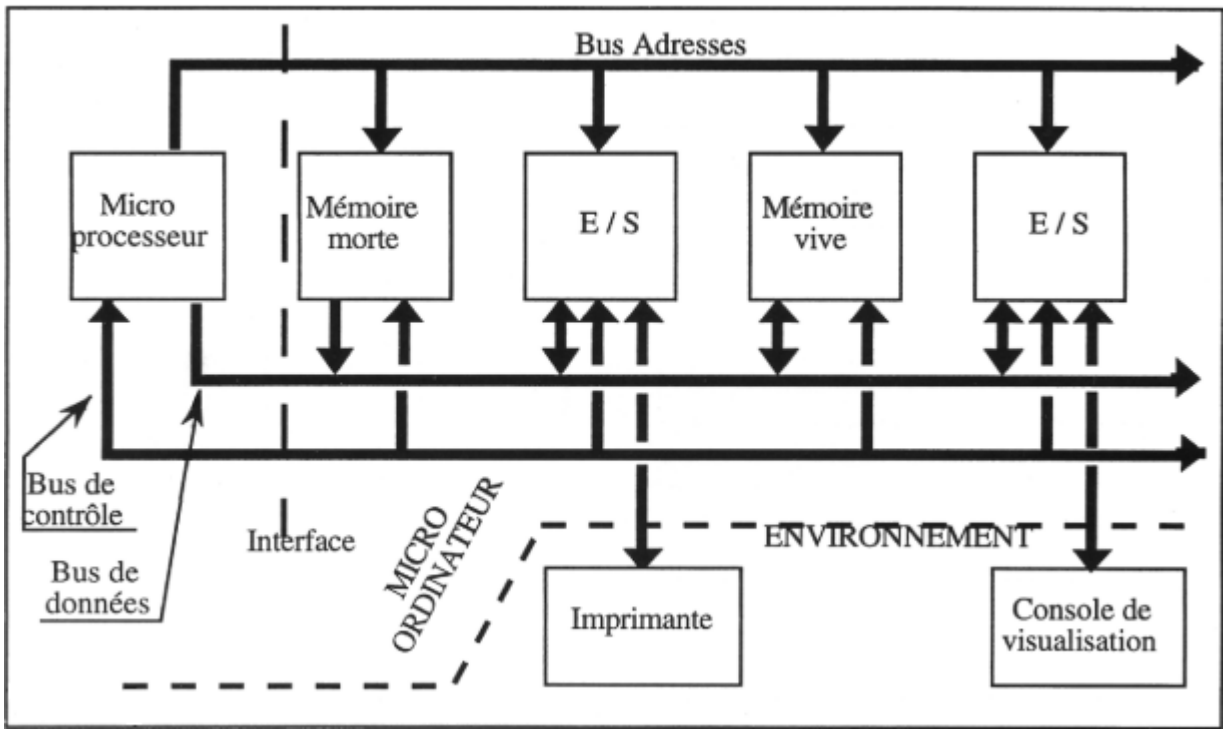
En TP cette année, vous utiliserez le 68HC11F1 qui équipe la carte Controlboy.

3. STRUCTURE INTERNE ET NOTIONS DE BASES

Nous allons aborder ici l'analyse du fonctionnement d'un microcontrôleur et de quoi celui-ci est composé. Il est important en assembleur de bien connaître la composition de la mémoire interne du processeur, de ses registres de configuration et des accumulateurs ; plusieurs termes qui seront expliqués un peu plus loin.

3.1. Structure d'un ordinateur.

Avant d'aborder le cas du microcontrôleur, faisons un petit parallèle avec l'ordinateur que tout le monde connaît bien. Les éléments de base sont : la carte mère, le processeur, la mémoire (sous plusieurs formes), et les connecteurs d'extension. Tous ces éléments communiquent entre eux par ce que l'on appelle des **bus**. C'est en fait un regroupement de pistes de même nature : par exemple, un bus de données de 8 bits sera constitué de 8 pistes (chacune sera un bit) et véhiculera seulement des données (une variable etc...). Le diagramme ci-dessous montre comment est interconnecté tout ça (*c'est simplifié bien évidemment*).



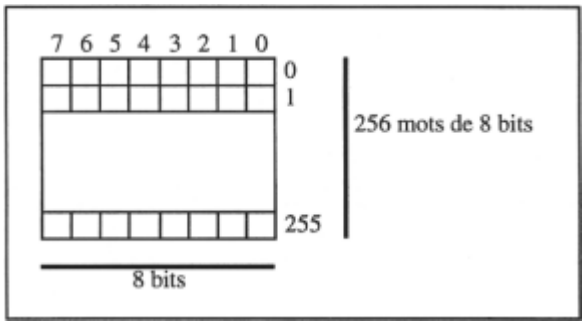
Le bus d'adresse sert à choisir où l'on va écrire une donnée dans une mémoire. Par exemple, le microprocesseur choisit d'écrire une variable dans la RAM (mémoire vive) : il va la sélectionner avec le bus de contrôle, choisir à quel emplacement à l'intérieur de la RAM il va écrire la donnée grâce au bus d'adresse et va écrire la donnée transmise par le bus de données.
Dans le schéma ci-dessus, la mémoire morte peut être un CD-ROM, le BIOS, la mémoire vive est la RAM ou un disque dur et les entrées/sorties (E/S) sont les ports PCI, parallèle (Lpt), série (Com) etc.

L'ordinateur est donc une association entre plusieurs éléments.

3.2. Composants de l'ordinateur

Nous allons maintenant voir comment sont constitués chacun des éléments décrits ci-dessus. Cette étape est importante car l'assembleur est un langage de bas niveau, ce qui signifie qu'il tient compte du matériel utilisé et de la structure interne de chaque composant.

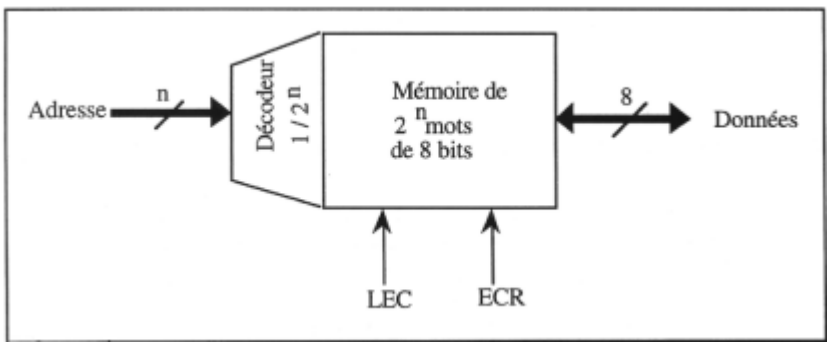
La mémoire est l'endroit où sont stockées les données. Elle peut être volatile ou non. Pour simplifier, on peut la voir comme le schéma ci-dessous.



C'est une mémoire de 256 octets, ou de 256*8 = 2048 bits.

Attention à ne pas confondre les unités : un bit, que ça soit en anglais ou en français, désigne un emplacement mémoire qui est égal à 0 ou à 1. Un octet est un ensemble de 8 bits. Et octet en anglais se dit byte... Donc quand vous regardez les caractéristiques d'une mémoire, pensez à la langue du produit : 15 Kb est différent en anglais (15 kilo-byte) et en français (15 kilo-bit). C'est pour cela que l'on voit par exemple dans certaine littérature une RAM de 32 K * 8, pour ne pas confondre.

Voyons maintenant l'extérieur de la mémoire. Nous avons vu précédemment qu'il existait un bus de données, de contrôle et d'adresse. Le schéma ci-dessous résume bien l'utilité de ces signaux.



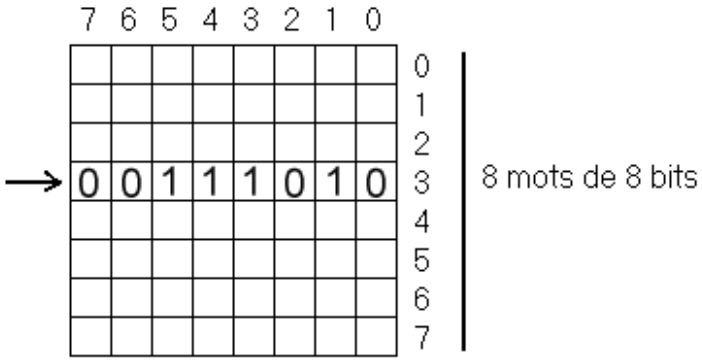
Prenons un exemple pour bien comprendre :

Si vous avez une mémoire de 8 octets, la mémoire sera vue comme un empilement de 8 mots de 8 bits. Vous souhaitez écrire une donnée de 8 bits dans cette mémoire, par exemple le chiffre (3A)hex (*rappelons que (3A)hex = (58)dec = (00111010)bin, revoyer votre cours si vous avez des problèmes avec les bases*) ; cette donnée passera par le bus de données par ses 8 pistes (*le trait qui coupe le bus et le chiffre au-dessus indique de combien de pistes est constitué le bus ; c'est ce qu'on appelle un schéma unifilaire*).

C'est là que le bus d'adresse prend toute son utilité : grâce à elle, vous pouvez choisir dans quel mot (à quel emplacement) vous souhaitez écrire la donnée.

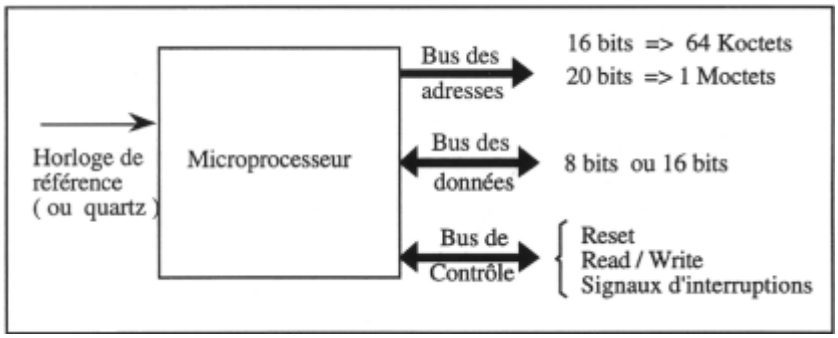
L'adresse est codée en binaire : ainsi, il faudra coder l'adresse sur 3 digits pour pouvoir obtenir l'empilement de 8 mots souhaité ci dessus, (*ce qui permet d'adresser une mémoire de 2 ^ 3=8 mots maximum*).

Exemple si le bus d'adresse vaut (3)dec = (011)bin :



Nous avons bien le chiffre (3A)hex à la position mémoire (= à l'adresse) 3.

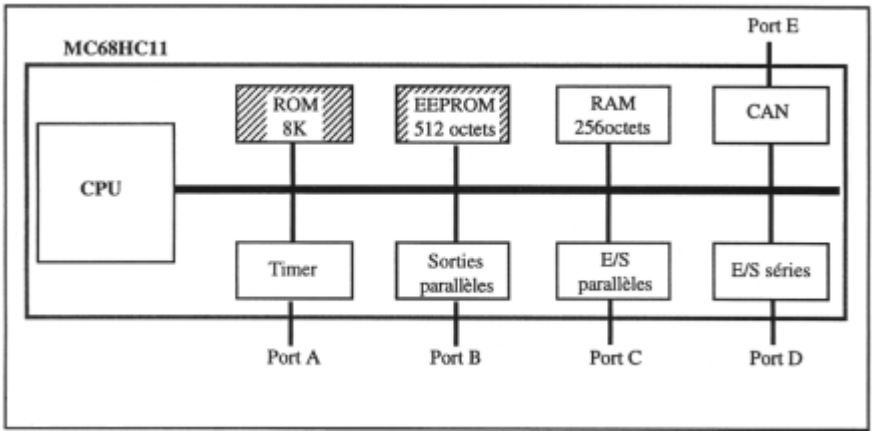
Les lignes LEC (lecture) et ECR (écriture) servent à déterminer si l'on veut écrire une donnée à une adresse ou lire une donnée à une adresse.
Parlons maintenant de l'élément de traitement de l'information : le microprocesseur. On le représente schématiquement comme cela :



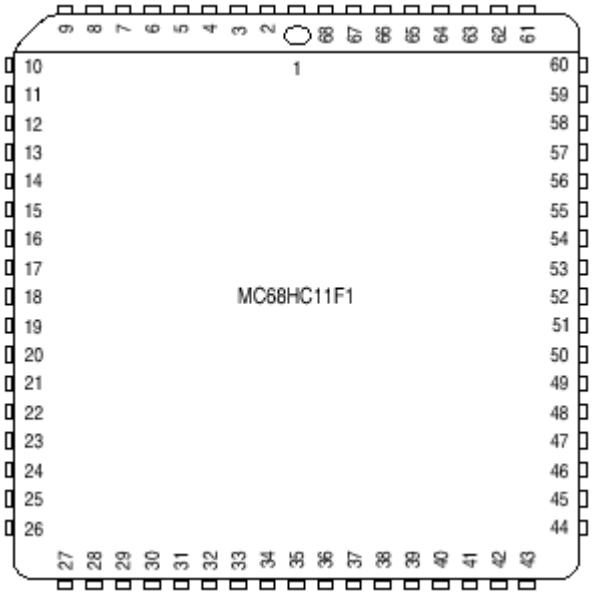
Nous retrouvons les bus de données, d'adresses et de contrôle. Ceux-ci sont générés par le microprocesseur et sont situés sur des entrées/sorties spéciales. Il est également représenté l'entrée de l'horloge : c'est généralement un quartz (pour les petites fréquences) qui fournit la fréquence de cadencement du processeur. Ainsi, plus sa fréquence est élevée, plus le processeur exécutera vite les calculs et les transferts de données. Pour les microcontrôleurs, les fréquences usuelles sont 8, 12 ou 16 MHz. Certains micros de la famille 80C51 peuvent cependant aller jusqu'à plus de 40 MHz.
Comme nous l'avons vu précédemment, la taille du bus d'adresse déterminera la taille maximale de la mémoire adressable.
Le microprocesseur est donc une association entre le matériel (hardware) et le logiciel (software) : il est donc nécessaire d'avoir un programme pour que le processeur serve à quelque chose **J**. Le programme est un ensemble logique convenablement ordonné d'opérations élémentaires que le microprocesseur sait exécuter. Le programme est toujours rangé dans la mémoire de l'ordinateur : le microprocesseur doit aller chercher les instructions au fur et à mesure pour les exécuter. Certaines instructions impliquent des calculs, d'où la présence d'une UAL (Unité Arithmétique et logique) dans les processeurs

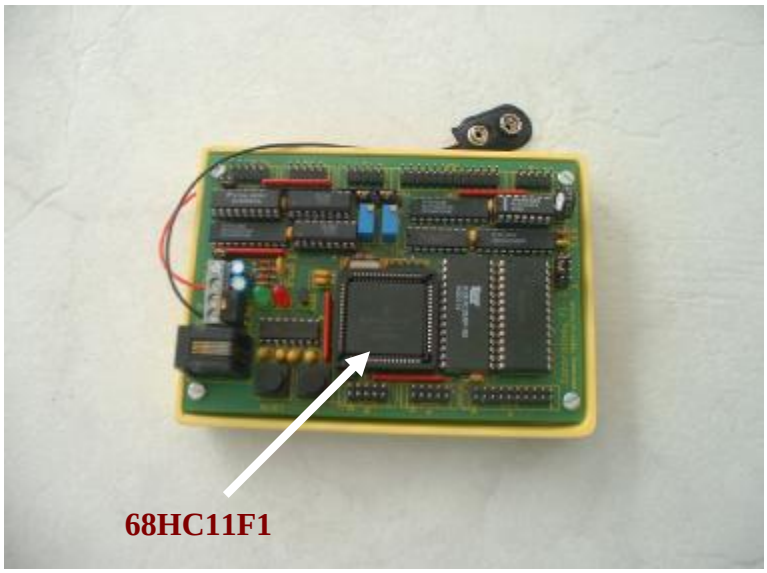
3.3. De l'ordinateur au microcontrôleur.

Nous y voilà J. Résumons tout de suite : le microcontrôleur est un ordinateur dans un seul boîtier. Mais attention, ce n'est pas un PIV 2Ghertz + 512 Mo de RAM dans un boîtier ; ses caractéristiques sont au niveau de ses applications. Ainsi, la figure ci-dessous présente schématiquement l'intérieur d'un microcontrôleur (ici le modèle 68HC11).



*...Et tout ça dans un petit boîtier de 2,5cm * 2,5cm :*





Celui-ci possède 68 broches, mais il y en a d'autres de 44, 52 etc..., tout dépend du nombre d'éléments intégrés. Décrivons un peu les éléments que nous y trouvons. Cela varie d'un microcontrôleur à l'autre.

*** CPU** : Central Processing Unit, désigne le microprocesseur

*** ROM** : Read Only Memory, est une mémoire à lecture seule qui contient le programme

*** EEPROM** : Electrically Ereasable Programmable Read Only Memory : mémoire morte électriquement effaçable/réinscriptible qui sert à sauvegarder des données

*** RAM** : Random Access Memory, mémoire servant à stocker les variables créées par le programme

* Un système de gestion du temps évolué

* Une horloge "temps réel"

* 2 interfaces séries, l'une asynchrone, l'autre synchrone

*** CAN** : Convertisseur Analogique Numérique, servant à convertir des tensions analogiques en une information numériques. Le processus contraire est le CNA. Il peut comporter plusieurs voies.

* Entrées/sorties parallèles : on les appelle les Ports. Ils sont généralement sur 8 bits

Vous voyez donc que le microcontrôleur se suffit à lui même, il n'a besoin que de très peu de composants supplémentaires (alimentations), et il permet de réaliser des systèmes électroniques très puissants (systèmes embarqués). Vous pouvez dès à présent mieux comprendre le tableau du chapitre précédent présentant les diverses versions du 68HC11.

Abordons maintenant quelques notions de base.

3.4. Notions de base

3.4.1. Les registres

Ce terme va être beaucoup utilisé dans ce cours. On désigne par registre un mot (de 8 bits généralement) qui sert aux calculs ou plus souvent à configurer un paramètre dans le microcontrôleur. Quand je parlerai de registre, je mettrai à chaque fois son aspect. En effet, chaque bit d'un registre configure quelque chose. Prenons un exemple : soit le registre de 8 bits nommé ELEVE_SI, que je représente comme cela :

bit	7	6	5	4	3	2	1	0
fonction	brun	grand	webmaster	sympas	chef	intelligent	programmeur	fort à Starcraft

Chaque bit du registre peut être mis à 1 ou à 0. Dans notre exemple, il s'agit des caractéristiques d'un individu **J**. J'ai montré le LSB et le MSB avec la numérotation des bits. Lorsque l'on **configure un registre**, cela signifie que l'on doit assigner chaque bit du registre à une valeur (0 ou 1), sachant que les bits ont tous à un état par défaut. Ici, nous savons que la personne possède un certain talent à Starcraft, donc nous mettons le bit 0 à 1 par exemple. Le reste est laissé pour l'instant à 0. L'octet à écrire dans le registre sera donc (01)hex, soit : ELEVE_SI=(01)hex.

Hop c'est fait. Tenez, cela me fait penser à une autre notion : le masquage. Lorsque vous programmerez un micro, il arrivera que vous ne vouliez pas changer tous les bits d'un coup. Un p'tit exemple : voici le registre ELEVE_SI tel que nous l'avons configuré plus haut :

bit	7	6	5	4	3	2	1	0
fonction	brun	grand	webmaster	sympas	chef	intelligent	programmeur	fort à Starcraft
valeur	0	0	0	0	0	0	0	1

Je souhaite mettre le bit nommé "intelligent" à 1 (*si l'on considère qu'il existe des élèves intelligents en S-SI* **J**), sans toucher aux autres valeurs. Ce que je vais utiliser, ce sont les "bêtes" opérateurs logiques (*si vous êtes perdu, revoyer votre cours*). Le plus facile serait de charger une nouvelle valeur, ce qui donnerait ici ELEVE_SI=(05)hex, mais cela est possible que si l'on connaît la valeur de tous les bits. Admettons que nous ne connaissons pas les autres valeurs : nous allons alors créer un masque logique afin de ne pas les modifier (les autres bits peuvent servir ailleurs). Pour mettre un bit à 1, il faut réaliser un OU (*voir les tableaux des opérateurs logiques de votre cours*) entre le bit "intelligent" et le chiffre 1. Ainsi, nous avons **0 OU 1 = 1**. *Hop, c'est fait*. Maintenant, nous allons préparer le masque des autres bits : pour laisser un bit tel qu'il est, il faut réaliser un OU entre ce bit et 0. Le masque complet pour ce registre est donc : (04)hex. Enfin, on applique ce masque au registre :

ELEVE_SI = ELEVE_SI OU (04)hex

Hop voilà, le registre ELEVE_SI devient donc :

bit	7	6	5	4	3	2	1	0
fonction	brun	grand	webmaster	sympas	chef	intelligent	programmeur	fort à Starcraft
valeur	0	0	0	0	0	1	0	1

Il est bien sûr possible de réaliser l'autre masque, pour mettre un bit à 0 : on utilise ici l'opérateur ET entre le bit et 0. Pour ne pas modifier les autres bits, on masque aussi mais avec des 1 (*voir le tableau de l'opérateur ET*).
Nous aurions eu alors :

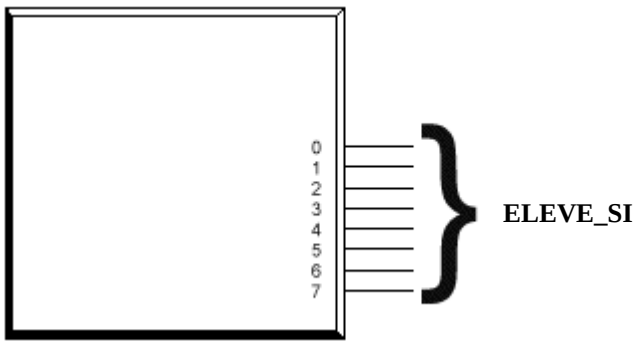
ELEVE_SI = ELEVE_SI ET (FB)hex

Compris ? **J**

3.4.2. Les entrées – sorties

Le microcontrôleur possède beaucoup de broches directement utilisables. Outre les habituelles broches d'alimentations et de configuration, il y a ce qu'on appelle des Ports. A l'instar des PC, il existe des ports série et parallèle. Les ports séries servent le plus souvent comme port de communication avec l'extérieur (un PC par exemple, pour télécharger un programme) ou pour dialoguer avec d'autres microcontrôleurs, composants esclaves ou autres modules. Quant aux ports parallèles, ils constituent la majorité des broches que l'on peut voir sur un microcontrôleur. Par exemple, un 80C51 à 44 broches peut posséder 4 ports de 8 bits, soit 32 broches utilisables. Chaque microcontrôleur a son propre brochage, c'est pour cela que l'on ne peut dire ici que des généralités. J'en dirai un peu plus lorsque je présenterai les composants ou en TP.

Cependant, on programme les ports directement par leur nom pour simplifier la programmation (il faut configurer les noms, mais nous verrons ça plus tard). Par exemple, mettons que l'on a le composant fictif suivant qui possède un port parallèle nommé ELEVE_SI (*le revoilà celui là...*)



Comme vous pouvez le voir, les ports sont composés de 8 broches, ce qui forme un octet. Chaque broche a son propre poids ; on nomme généralement chaque broche en prenant le nom du port et le poids d'une broche. Par exemple, on appellera la première broche (bit 0) du port ELEVE_SI (0), puis (1) pour le deuxième bit, (2) pour le troisième et ainsi de suite...
Une dernière chose : il existe trois status différents pour les ports.

* Port en **entrée** : il est capable de **recevoir** des informations en provenance de l'extérieur.

* Port en **sortie** : il **émet** des informations du microcontrôleur vers l'extérieur.

* Port **bi-directionnel** : il est capable, pour toutes ses broches ou certaines d'entre elles, de recevoir ou d'émettre des données.

Il s'agit là d'un status physique, c'est à dire que c'est la fabrication électronique des ports qui décide dans quel mode il sera capable de fonctionner. Certains microcontrôleurs possèdent des Port dont certaines broches sont en entrée et d'autres en sortie. Enfin, il est possible sur certains modèles de définir logiciellement dans quel mode le Port va fonctionner; C'est le cas du 68HC11F1 que vous programmerez cette année.

Quand on écrit une donnée sur un port, le CPU va transformer un ordre logiciel en une donnée bien physique : un niveau de tension. Ainsi, lorsqu'une broche d'un port est "mise à 1" (expression qui signifie que l'on a placé dans le programme cette broche à l'état 1), la tension sur cette borne sera de 5V. Et bien sûr, lorsque l'on placera une broche à 0 dans un programme, la tension sur cette borne sera de 0V. Cette tension, qui a pour valeur soit 0V, soit 5V, est normalisée. Ainsi, la plupart des composants électroniques possèdent cette référence.
A l'inverse, une tension de 5V qui apparaît sur une broche se traduit par un niveau logique '1', si l'on lit l'état de cette broche dans un programme.

3.4.3. Les types de mémoire

Nous en avons brièvement parlé plus haut mais il est important de revenir plus sérieusement sur cette notion.

Commençons avec la **RAM** (**R**andom **A**ccess **M**emory). Presque tous les microcontrôleurs en possèdent en interne. Elle a le même rôle que sur PC, c'est une mémoire temporaire et volatile (les données disparaissent lorsqu'il n'y a plus d'alimentation électrique). La RAM embarquée peut aller de 256 octets à 1Ko ; une RAM de 256 octets peut donc contenir 256 variables de 8 bits, et une RAM de 1Ko contiendra 1024 variables de un octet au maximum. Cette valeur est suffisante pour programmer de petites applications. Dans d'autres cas, il est possible d'ajouter une mémoire RAM externe au microcontrôleur : c'est un composant d'une vingtaine de broches qui peut faire 32Ko par exemple (*ce qui devient très important*).
Donc, dès que vous créerez une nouvelle variable dans votre programme, elle sera logée dans la RAM. La RAM est accessible en lecture et en écriture.

La mémoire **ROM** : acronyme de **R**ead **O**nly **M**emory, cette mémoire n'est accessible qu'en lecture. Elle contient généralement le programme de l'utilisateur. Une fois programmée, une ROM ne peut pas être effacée ; il faut donc être sûr de son programme sous peine de devoir jeter la mémoire. Certains microcontrôleurs disposent d'une ROM interne ; c'est donc une programmation définitive.

L'**EPROM** : **E**reasable **P**rogrammable **R**ead **O**nly **M**emory. Cette mémoire possède les mêmes propriétés que la ROM sauf que l'on peut l'effacer à l'aide d'un Effaceur d'EPROM. Cet appareil se charge d'insoler la mémoire avec des rayons UV pour l'effacer ; dès lors, il est possible de la reprogrammer. Ce type de composant est reconnaissable par une fenêtre sur le boîtier d'où l'on peut voir le cœur de la mémoire. L'EPROM est accessible en lecture uniquement, et contient généralement le programme à exécuter.

Enfin, nous avons l'**EEPROM** : **E**lectrically **E**reasable **P**rogrammable **R**ead **O**nly **M**emory. Cette mémoire est très intéressante car il est possible de l'effacer et de la programmer électriquement ; inutile donc d'investir dans un programmateur coûteux. Cette mémoire peut contenir le programme ou encore une variable qu'il est nécessaire de sauvegarder avant une coupure d'alimentation.

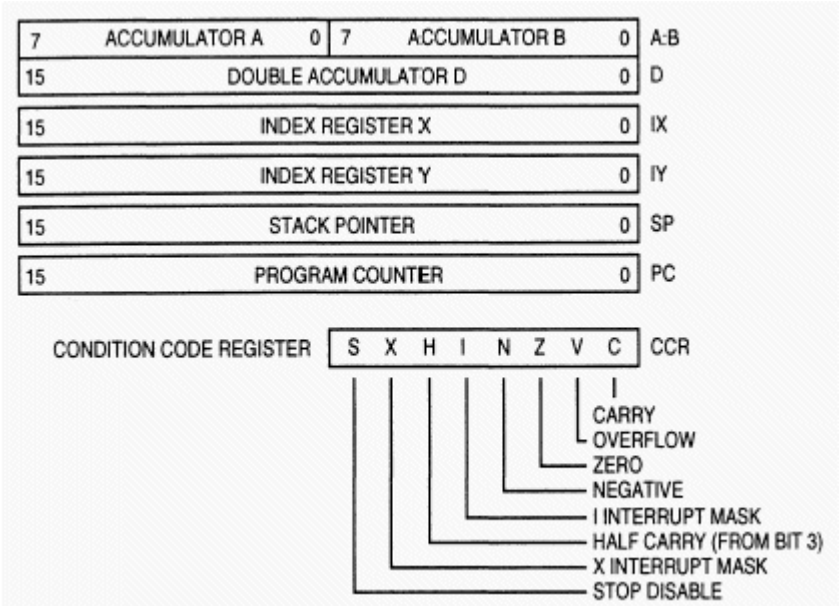
4. STRUCTURE DU 68HC11

Aïe ! Les choses sérieuses commencent **J.** Comme je l'ai dit précédemment, il est important de connaître la structure interne du composant que l'on programme. Je vais vous présenter ici une version du 68HC11 : le 68HC11F1.

4.1. les éléments du CPU

4.1.1. Les registres principaux.

Maintenant que vous savez ce qu'est un registre, nous allons parler des plus importants : les registres de calcul. Ces zones mémoire sont immuables et sont identiques quelque soit la version du 68HC11. Voici le schéma présentant en détail les registres :



*** A** et **B** : ce sont deux registres de 8 bits à usage général que l'on appelle **"accumulateur"**. Ces deux registres peuvent être utilisés "bout à bout" dans certaines instructions pour former un registre de 16 bits appelé **D** (**A** est le poids fort et **B** le poids faible). Ces deux registres servent pour les calculs ou pour stocker des données par exemple.

*** X** et **Y** : ce sont deux registres de 16 bits qui peuvent servir de registre à usage général, ou bien de registre d'index (le I devant signifie Index) pour ce mode d'adressage. Ces termes seront expliqués plus tard. Pour l'instant, souvenez-vous que vous pouvez y stocker des données 16 bits. On parle de IX ou de X, c'est pareil.

*** SP** : Un registre pointeur de pile utilisé pour la gestion de la structure de la pile (**S**tack **P**ointer)

*** PC** : C'est le registre 16 bits "compteur programme" qui contient l'adresse de l'instruction courante dans la mémoire. (**P**rogram **C**ounter)

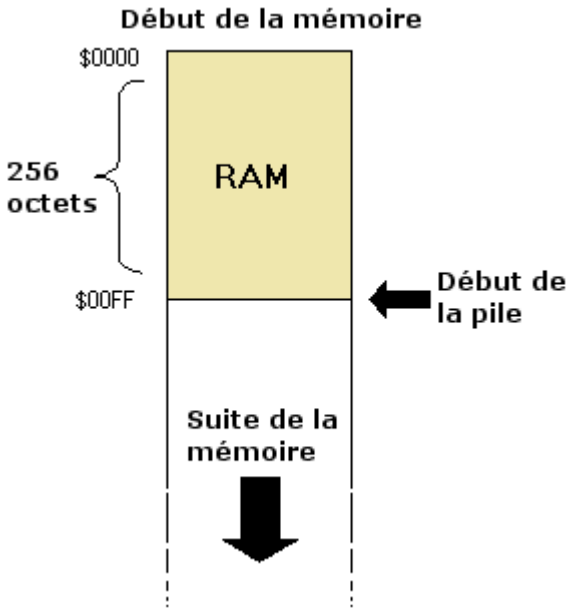
*** CCR** : C'est un registre de 8 bits qui contient les indicateurs d'état pour les calculs entre autres. (**C**ondition **C**ode **R**egister)

Notes sur l'algorithmie : quelques notations vont être utilisées ici qui seront également valable pour l'assembleur.

En voici les définitions :
ppv : prend pour valeur, équivalent au = en C.
$MEM_8[X]$: position mémoire de 8 bits désignée (ou pointée) par X. Cette mémoire est la RAM ou la pile.
$F300 : le signe $ permet de dire que le nombre est en hexadécimal. Cette écriture est propre à Motorola ; Intel l'écrit F300h.
@26 : le signe @ permet de dire que le nombre est en octal.
%11001010 : le signe % permet de dire que le nombre est en binaire.
'A : le signe ' permet de dire que ce qui suit est UN caractère ASCII (PAS une chaîne de caractère).

4.1.2. La structure de la pile

La pile est une notion assez importante car vous la rencontrerez également dans d'autres langages. Cette notion peut simplifier la programmation mais est une source fréquente d'erreurs.
La pile est en fait une simple zone de la RAM. Voici la structure du début de la mémoire interne :



5. L'ASSEMBLEUR 68HC11 : LE JEU D'INSTRUCTION

Les instructions peuvent paraître un peu bizarres, pour peut que l'on soit habitué à des langages plus évolués comme le C ou le BASIC. En fait, les instructions sont formées à l'aide d'initiales ; ainsi, **LDAA** signifie LoaD Accumulator A, soit charge dans l'accumulateur A. Encore un autre : **ASL** signifie Arithmetic Shift Left. Voilà, et ainsi de suite pour les autres.

Le microcontrôleur ne comprend que le langage assembleur. Toutefois, il existe des compilateurs qui permettent de convertir des instructions réalisées dans des langages évolués en code assembleur (*OUF ! Et heureusement !* ***J***). En TP vous n'aurez presque pas à programmer en assembleur, mais il est toujours bon de savoir les instructions.

5.1. Classification des instructions du 68HC11

Notes : M désigne un opérande qui est obtenu par un mode d'adressage direct, immédiat, étendu ou indexé. La notation M:M+1 signifie que l'on manipule une donnée 16 bits sur deux adresses consécutives ; M désigne une adresse de 8 bits et M+1 la suivante dans la mémoire. Le signe = signifie Prend Pour Valeur (**ppv**), soit = en C eu en Basic ou := en pascal : c'est une affectation et non une comparaison. r est le reste d'une division et q son quotient.
Enfin, sachez que **branch** signifie branche, bifurque, va à etc : c'est pour indiquer un changement de lecture dans le déroulement du programme.

Arithmétiques			
Addition	**Soustraction**	**Incrémentation**	**Décrémentation**
ABA A=A+B ADDA A=A+M ADDB B=B+M ADDD D=D+M:M+1 ADCA A=A+M+C ADCB B=B+M+C ABX X=X+B ABY Y=Y+B	SBA A=A-B SUBA A=A-M SUBB B=B-M SUBD D=D-M:M+1 SBCA A=A-M-C SBCB B=B-M-C	INC M=M+1 INCA A=A+1 INCB B=B+1 INX X=X+1 INY Y=Y+1 INS SP=SP+1	DEC M=M-1 DECA A=A-1 DECB B=B-1 DEX X=X-1 DEY Y=Y-1 DES SP=SP-1
Opposé		**Divers**	
NEG M=-M NEGA A=-A NEGB B=-B		DAA Decimal Adjust FDIV D/X, D=r, X=q IDIV D/X, D=r, X=q MUL D=A*B	

Logiques			
Et/Ou/Xor/Not	**Indicateurs**	**Mise à 1/0 de bit(s)**	**Rotations**
ANDA A=A and M ANDB B=B and M ORAA A=A or M ORAB B=B or M EORA A=A xor M EORB B=B xor M COM M=not(M) COMA A=not(A) COMB B=not(B)	CLC C=0 SEC C=1 CLV V=0 SEV V=1	BSET M=M+masque BCLR M=M and not(masque)	ROL M rotate left ROLA A rotate left ROLB B rotate left ROR M rotate right RORA A rotate right RORB B rotate right
Décalages logiques		**Décalages arithmétiques**	
LSL M shift left LSLA A shift left LSLB B shift left LSLD D shift left LSR M shift right LSRA A shift right LSRB B shift right LSRD D shift right		ASL M shift left ASLA A shift left ASLB B shift left ASLD D shift left ASR M shift right ASRA A shift right ASRB B shift right	

Transferts		
Chargement de registres	**Remise à zéro**	**Rangement de registres**
LDAA A=M LDAB B=M LDD A=M,B=M+1 LDX X=M:M+1 LDY Y=M:M+1 LDS SP=M:M+1	CLR M=0 CLRA A=0 CLRB B=0	STAA M=A STAB M=B STD M=A,M+1=B STX M:M+1=X STY M:M+1=Y STS M:M+1=SP

Inter-registres 8 bits	Inter-registres 16 bits
TAB B=A TBA A=B TAP CCR=A TPA A=CCR	TSX X=SP+1 TSY Y=SP+1 TXS SP=X-1 TYS SP=Y-1 XGDX Echange D avec X XGDY Echange D avec Y

Ruptures de séquence		
Inconditionnelles	**Avec test direct d'indicateur**	**Avec test de bits**
JMP Jump BRA Branch always BRN Branch never JSR Jump to subroutine BSR Branch to subroutine RTS Return from subroutine	BCC Branch if C clear BCS Branch if C set BEQ Branch if equal to 0 BNE Branch if not equal to 0 BPL Branch if plus BMI Branch if minus BVC Branch if V clear BVS Branch if V set	BRCLR Branch if bit(s) clear BRSET Branch if bit(s) set
Conditionnelles (nb. signés)	**Conditionnelles (nb. non signés)**	
BGT Branch if greater than 0 BGE Branch if greater than 0 or equal BLT Branch if less than 0 BLE Branch if less than 0 or equal	BHI Branch if higher BHS Branch if higher or same BLO Branch if lower BLS Branch if lower or same	

Comparaisons/tests	
Comparaisons	**Tests**
CBA A-B CMPA A-M CMPB B-M CPD D-M:M+1 CPX X-M:M+1 CPY Y-M:M+1	BITA A and M BITB B and M TST M-0 TSTA A-0 TSTB B-0

Gestions diverses	
Interruptions	Pile et divers
CLI I=0 SEI I=1 SWI Software IT WAI Wait for IT RTI Return from IT	PSHA Push A PSHB Push B PSHX Push X PSHY Push Y PULA Pull A PULB Pull B PULX Pull X PULY Pull Y NOP no operation STOP stop internal clocks