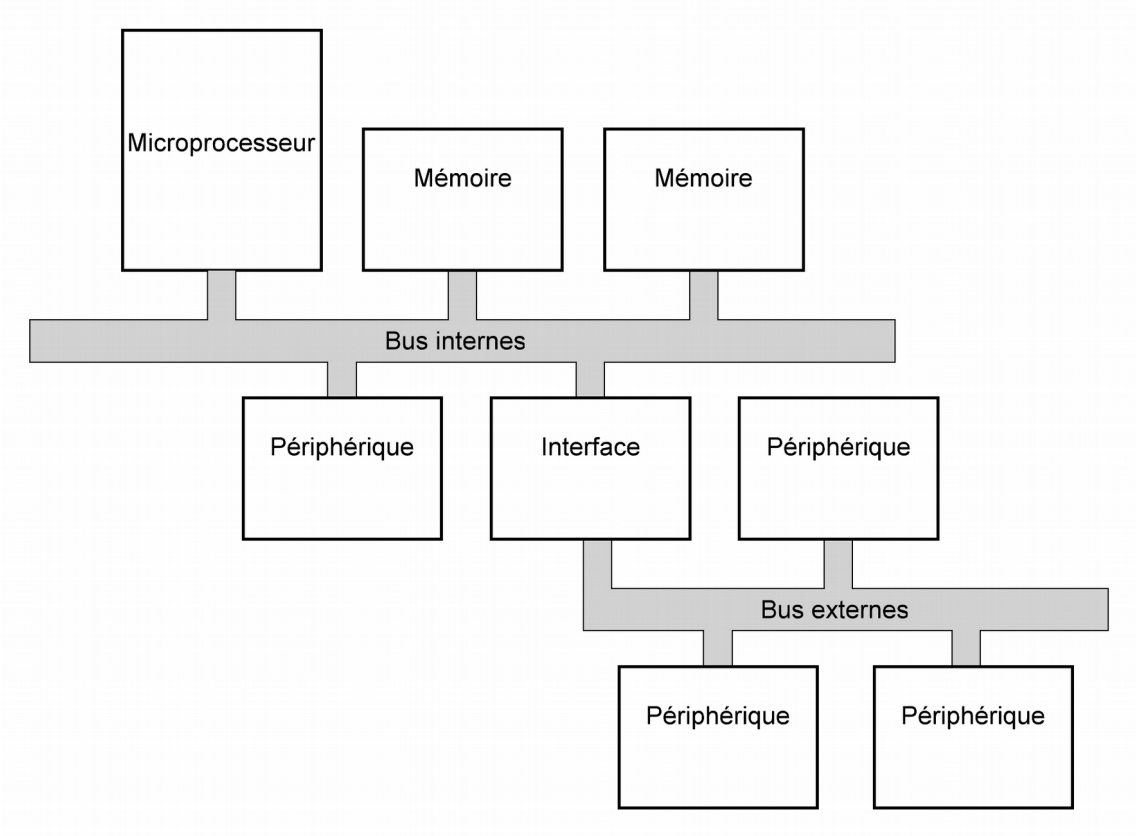
Chapitre 4
Les systèmes à microprocesseur

Version du 27/12/2017

Table des matières

I. Introduction

Un système à microprocesseur est constitué d'au moins un microprocesseur, de mémoires et de différents périphériques. Le rôle principal du microprocesseur est de contrôler l'ensemble.



Un bus est un ensemble de fils qui sert à faire circuler des informations entre les différents composants d'un ordinateur. Par exemple, un bus de huit fils pourra transférer des données codées sur 8 bits.

Un microprocesseur possède trois bus internes (un bus d'adresse, un bus de donnée et un bus de contrôle), ce qui lui permet de communiquer avec un certain nombre de mémoires et de périphériques. Toutefois, des interfaces sont fréquemment utilisées afin d'accroître les capacités de communication du microprocesseur avec le monde extérieur. Non seulement ces interfaces peuvent fournir toute la puissance électrique nécessaire pour alimenter les composants externes, mais elles permettent également de standardiser des bus externes afin d'offrir la possibilité de brancher n'importe quel équipement provenant de différents fabricants (par exemple : USB, bus PCI, bus ISA).

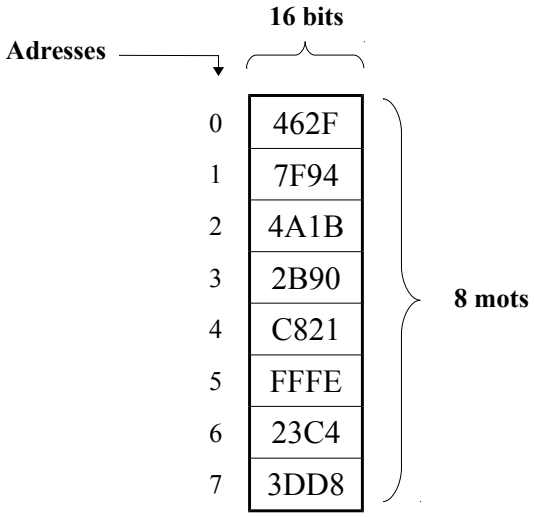
II. Les mémoires

1. Définitions

Une mémoire est un composant électronique qui permet de stoker des informations sous forme de mots binaires. Un mot binaire est une quantité déterminée de bits (par exemple, un octet est un mot de 8 bits).

On peut représenter une mémoire comme un ensemble de cases contenant chacune un mot binaire. Chaque case est identifiée par un **numéro d'adresse unique**.

Exemple d'une mémoire contenant **8 mots** de **16 bits** :



Par exemple :
- La donnée contenue dans la case mémoire d'adresse **0** est $\mathbf{462F_{16}}$.
- La donnée contenue dans la case mémoire d'adresse **4** est $\mathbf{C821_{16}}$.

L'organisation interne d'une mémoire se caractérise par les deux grandeurs suivantes : **le nombre de mots** (ou **d'adresses**) qu'elle contient et **le nombre de bits par mot**. Pour la suite du cours, nommerons respectivement ces grandeurs **profondeur** et **largeur**.

Nous pouvons également exprimer la capacité d'une mémoire en bits ou en octets. C'est-à-dire le nombre de bits ou d'octets que contient une mémoire.
- **Capacité en bits** = Profondeur × Largeur
- **Capacité en octets** = Profondeur × Largeur / 8

Notre mémoire de 8 mots de 16 bits a donc :
- une capacité de 128 bits (8 × 16) ;
- une capacité de 16 octets (8 × 16 / 8).

2. Les principales catégories de mémoire

2.1. La mémoire vive (RAM)

Les mémoires vives ou RAM (*Random Access Memory*) possèdent les caractéristiques suivantes :
- Elles sont volatiles : les données sont perdues lors d'une mise hors tension ;
- Elles sont accessibles en lecture et en écriture.

Les mémoires vives se déclinent en plusieurs sous-catégories. Nous évoquerons uniquement les deux sous-catégories principales.

2.1.1. La mémoire vive statique (SRAM)

La mémoire vive statique ou SRAM (*Static Random Access Memory*) utilise des bascules comme éléments de stockage.

Elle est très rapide et peu consommatrice d'énergie, mais reste encore très onéreuse.

La mémoire cache d'un ordinateur est généralement constituée de mémoire vive statique.

2.1.2. La mémoire vive dynamique (DRAM)

La mémoire vive dynamique ou DRAM (*Dynamic Random Access Memory*) utilise l'effet capacitif d'un transistor comme élément de stockage.

L'inconvénient de cette méthode de stockage est que l'information ne peut être conservée que très peu de temps. Il est alors nécessaire de rafraîchir régulièrement la charge électrique contenant l'information. Ceci nécessite l'ajout d'un circuit de gestion du rafraîchissement.

Son avantage est sa forte densité d'intégration. Il est possible d'intégrer sur une seule puce, une quantité de mémoire bien supérieure à celle d'une SRAM. Le coût d'une DRAM devient alors bien inférieur à cette dernière.

La mémoire principale d'un ordinateur est généralement constituée de mémoire vive dynamique.

2.2. La mémoire morte (ROM)

Les mémoires mortes ou ROM (*Read Only Memory*) possèdent les caractéristiques suivantes :
- Elles sont non volatiles : les données sont conservées lors d'une mise hors tension ;
- Elles sont accessibles en lecture seule.

Le BIOS (*Basic Input Output System*) d'un ordinateur est généralement contenu dans une mémoire morte.

Les mémoires mortes se déclinent également en plusieurs sous-catégories. Nous évoquerons uniquement les sous-catégories principales. Au sens strict du terme, il n'est pas possible d'écrire dans une ROM. Elle est programmée de façon permanente par le constructeur. Toutefois, il existe des sous-catégories de ROM qui peuvent être programmées à l'aide d'un programmateur.

2.2.1. La mémoire morte programmable (PROM)

La mémoire morte programmable ou PROM (*Programmable Read Only Memory*) ne peut être programmée qu'une seule fois. Il est impossible d'effacer son contenu. Une fois programmée, elle se comporte comme une ROM.

2.2.2. La mémoire morte effaçable et programmable (EPROM)

La mémoire morte effaçable et programmable ou EPROM (*Erasable Programmable Read Only Memory*) peut être programmée, effacée et reprogrammée un certain nombre de fois. Une fois programmée, elle se comporte comme une ROM.

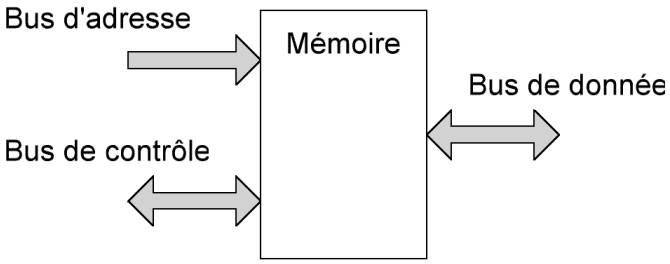
Les deux principaux types d'EPROM sont :

- **Les UV-EPROM (*UV Erasable Programmable Read Only Memory*)** qui sont effaçables par une exposition aux rayons ultraviolets. Ce type de mémoire est de moins en moins utilisé.

- **Les EEPROM (*Electrically Erasable Programmable Read Only Memory*)** qui sont effaçables électriquement.

3. Les bus

Une mémoire possède trois bus :



3.1. Le bus d'adresse

Le bus d'adresse sert à contenir l'adresse mémoire à laquelle on souhaite accéder. C'est une entrée.

Si a est le nombre de fils du bus d'adresse, alors on a la relation suivante : **profondeur = 2^a.**

3.2. Le bus de donnée

Le bus de donnée sert à transférer les données contenues dans la mémoire.

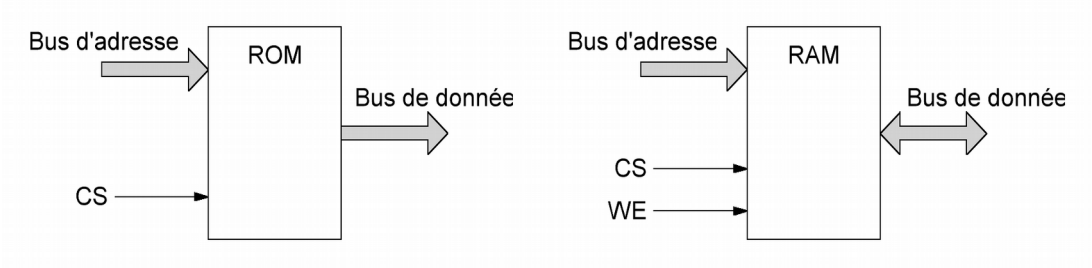
C'est un bus de sortie sur une mémoire de type ROM (la donnée peut uniquement être lue) et un bus bidirectionnel sur une mémoire de type RAM (la donnée peut être lue et écrite).

Si *d* est le nombre de fils du bus de donnée, alors on a la relation suivante : **largeur = *d*.**

3.3. Le bus de contrôle

Le bus de contrôle (ou bus de commande) sert à véhiculer les signaux de contrôle et de commande de la mémoire. Ces signaux permettent de coordonner les échanges d'informations entre la mémoire et d'autres composants. Certains signaux peuvent être unidirectionnels (en entrées ou en sorties), d'autres peuvent être bidirectionnels. Le nombre de signaux disponibles sur ce bus varie en fonction du type de mémoire et de ses fonctionnalités.

Pour la suite du cours, nous considérerons le bus de contrôle réduit à son strict minimum. Ce qui nous donne pour une ROM et pour une RAM :



3.3.1. L'entrée d'activation (CS)

Cette entrée permet d'activer ou de désactiver une mémoire. Elle est obligatoire, quel que soit le type de mémoire. Nous l'appellerons *CS* (*Chip Select*). Elle peut toutefois porter un nom différent d'une mémoire à une autre (par exemple, *CE* pour *Chip Enable*).

Quand une mémoire est désactivée, elle ignore la tension présente sur ses entrées et n'impose aucune tension sur ses sorties. On dit alors que ses sorties sont dans un état de haute impédance (il s'agit d'un troisième état qui n'est ni un état logique 0, ni un état logique 1). D'une certaine façon, c'est comme si la mémoire avait été débranchée.

3.3.2. L'entrée de sélection entre la lecture et l'écriture (WE)

Cette entrée permet de choisir si l'accès à une mémoire doit se faire en lecture ou en écriture. Elle est obligatoire sur les mémoires de type RAM et n'existe pas sur les mémoires de type ROM. Nous l'appellerons *WE* (*Write Enable*). Elle peut toutefois porter un nom différent d'une mémoire à une autre (par exemple, $\overline{R}/W$ pour $\overline{Read\ /\ Write}$).

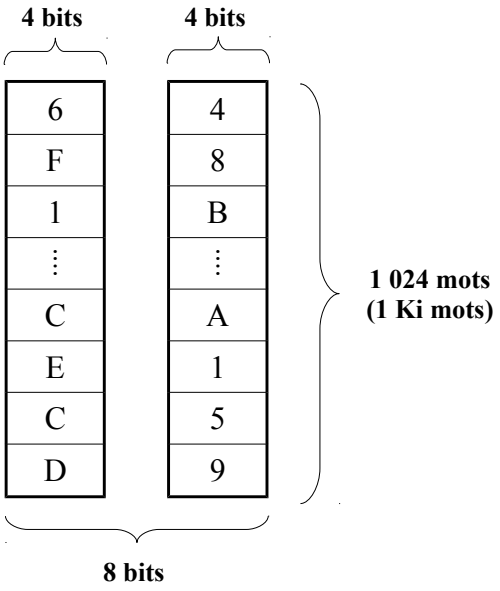
4. L'assemblage de mémoires

Il est possible d'assembler plusieurs mémoires de même type afin d'accroître la largeur et/ou la profondeur de la mémoire. C'est le type d'assemblage que l'on retrouve sur les modules SIMM et DIMM.

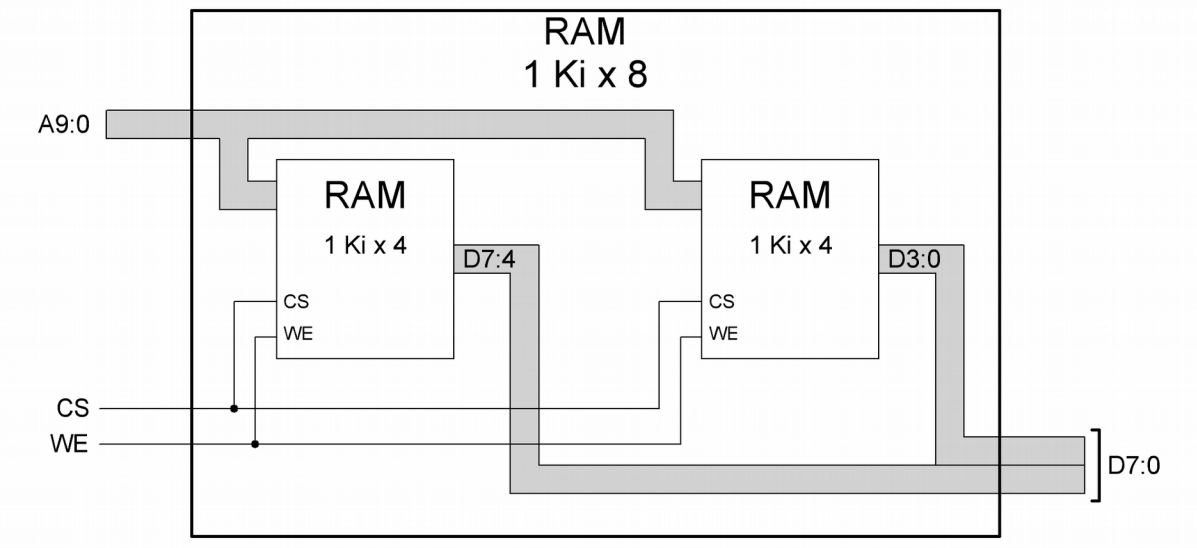
4.1. L'assemblage en parallèle

L'assemblage en parallèle permet d'accroître la largeur d'une mémoire. C'est-à-dire la taille des mots qu'elle contient. Cet assemblage augmente donc le nombre de fils du bus de donnée.

Prenons l'exemple de deux mémoires RAM possédant chacune un bus d'adresse de 10 bits et un bus de donnée de 4 bits. Ces RAM ont donc une profondeur de 1024 mots (2^{10}) et une largeur de 4 bits par mot.



On peut très bien imaginer que ces deux mémoires d'une largeur de 4 bits par mot ne forment en fait qu'une seule mémoire d'une largeur de 8 bits par mot. Pour arriver à ce résultat, il faut câbler les mémoires de la façon suivante :



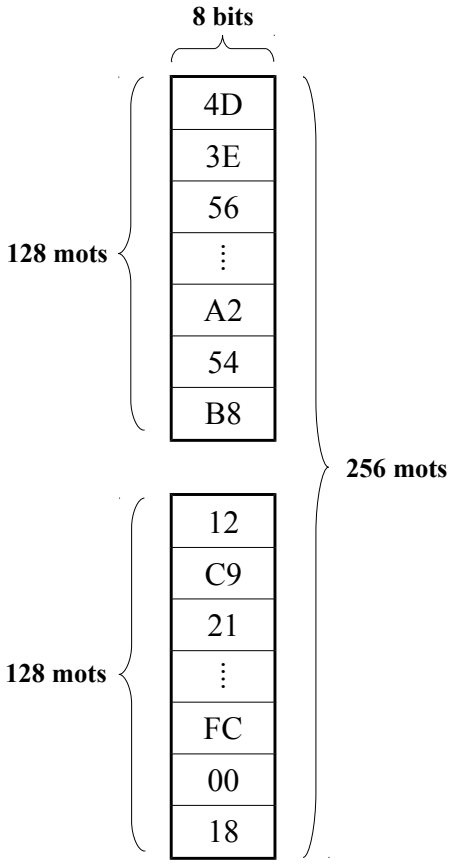
On remarque que :

- Les bus d'adresse des mémoires internes sont reliés. Ils forment le bus d'adresse de la mémoire externe ;
- Les bus de contrôle des mémoires internes sont reliés. Ils forment le bus de contrôle de la mémoire externe ;
- Les bus de donnée des mémoires internes ne sont pas reliés. Ils sont simplement juxtaposés. Cette juxtaposition forme le bus de donnée de la mémoire externe ;
- Les deux mémoires doivent être actives en même temps afin d'obtenir une donnée complète sur le bus de donnée de la mémoire externe.

4.2. L'assemblage en série

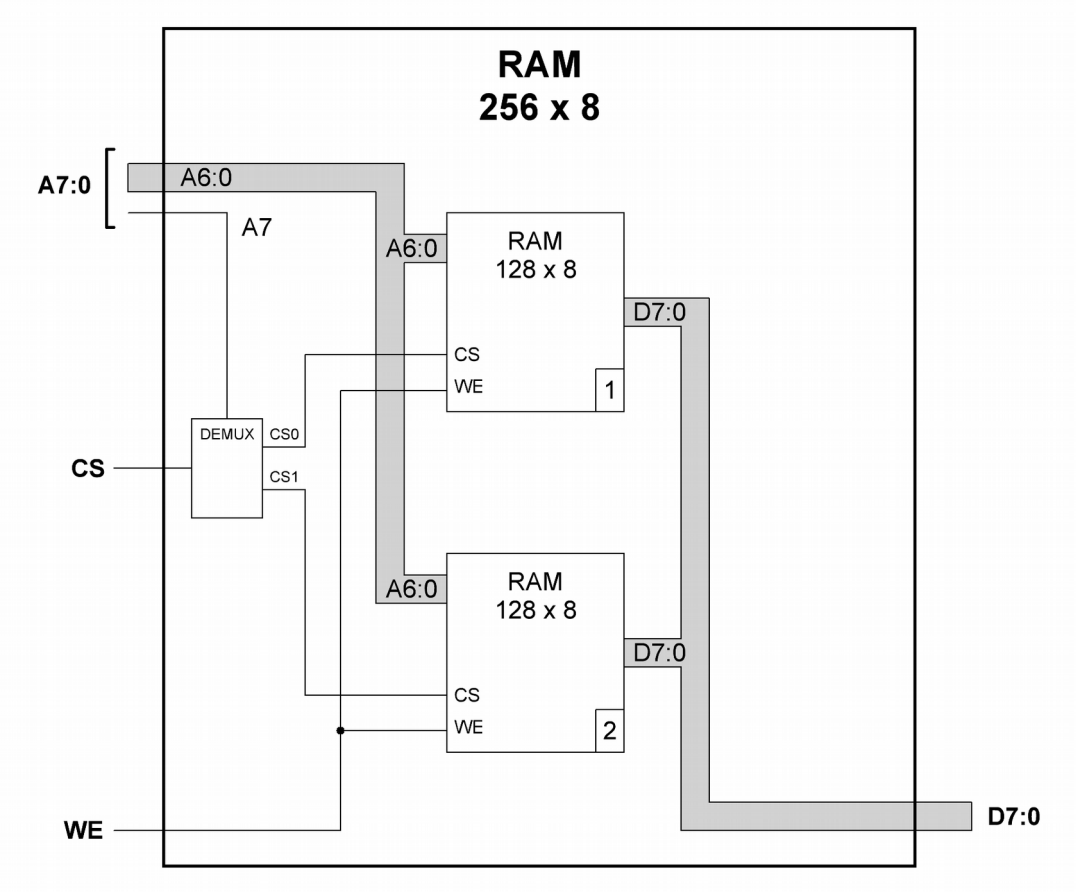
L'assemblage en série permet d'accroître la profondeur d'une mémoire. C'est-à-dire le nombre de mots qu'elle contient. Cet assemblage augmente donc le nombre de fils du bus d'adresse.

Prenons l'exemple de deux mémoires RAM possédant chacune un bus d'adresse de 7 bits et un bus de donnée de 8 bits. Ces RAM ont donc une profondeur de 128 mots (2^7) et une largeur de 8 bits par mot.



On peut très bien imaginer que ces deux mémoires d'une profondeur de 128 mots ne forment en fait qu'une seule mémoire d'une profondeur de 256 mots.

Pour arriver à ce résultat, il faut câbler les mémoires de la façon suivante :



On remarque que :

- Les bus de donnée des mémoires internes sont reliés. Ils forment le bus de donnée de la mémoire externe ;
- À l'exception des entrées *CS*, les bus de contrôle des mémoires internes sont reliés. Ils forment le bus de contrôle de la mémoire externe ;
- Les bus d'adresse des mémoires internes sont reliés. Ils forment les bits de poids faible du bus d'adresse de la mémoire externe ;
- Il faut ajouter un bit d'adresse à la mémoire externe afin d'obtenir les 256 adresses ($2^8 = 256$). Ce bit supplémentaire va servir à sélectionner l'une des mémoires internes à l'aide d'un démultiplexeur ;
- Les mémoires internes ne doivent jamais être actives en même temps afin de ne pas créer un conflit au niveau du bus de donnée. Le *CS* de la mémoire externe est donc propagé vers l'une des mémoires internes pendant que l'autre est désactivée.

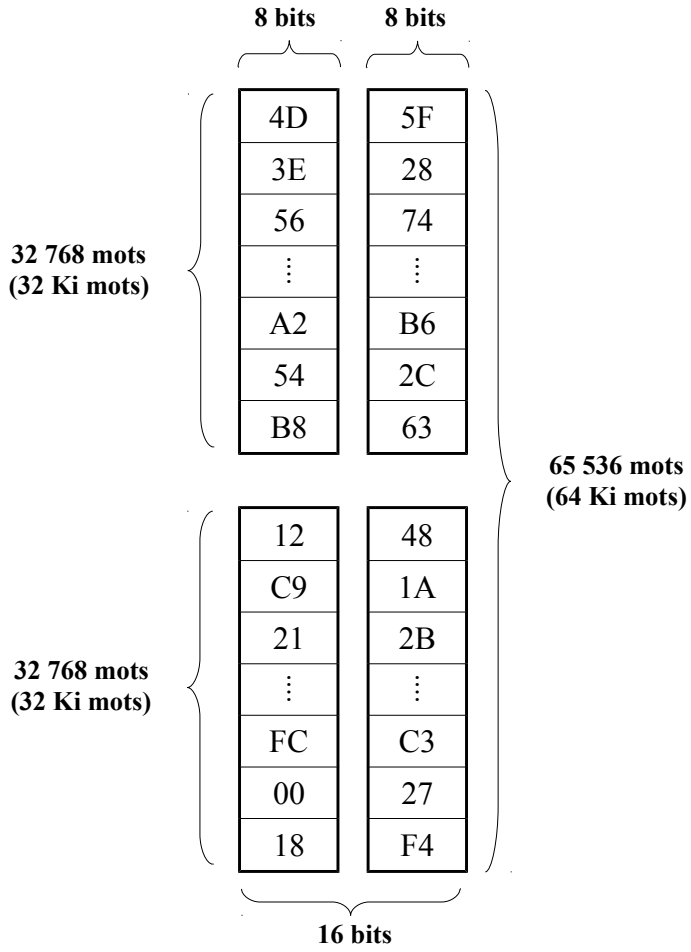
Dans le cas où le *CS* de la mémoire externe est à 1, on a les deux cas suivants :

- Si *A7* = 0 alors *CS0* = 1 et *CS1* = 0 → c'est la mémoire interne numéro 1 qui est active ;
- Si *A7* = 1 alors *CS0* = 0 et *CS1* = 1 → c'est la mémoire interne numéro 2 qui est active.

Dans le cas où le *CS* de la mémoire externe est à 0, alors toutes les mémoires internes sont désactivées.

4.3. L'assemblage en parallèle et en série

Il est possible de combiner les deux types d'assemblage. Prenons l'exemple de quatre mémoires RAM possédant chacune un bus d'adresse de 15 bits et un bus de donnée de 8 bits. Ces RAM ont donc une profondeur de 32 768 mots (2^{15}) et une largeur de 8 bits par mot.



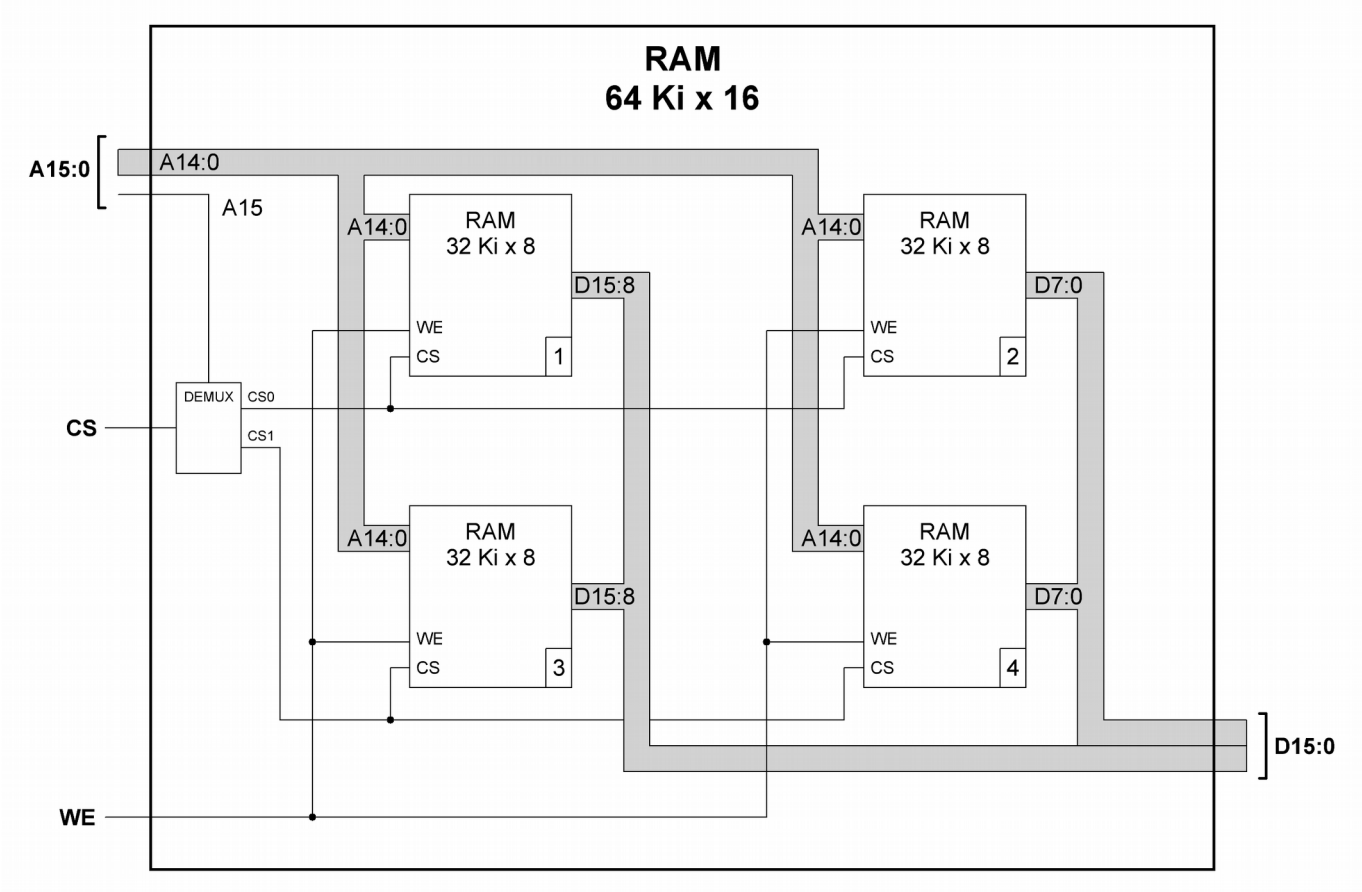
On peut très bien imaginer que ces quatre mémoires ne forment en fait qu'une seule mémoire d'une profondeur de 65 536 mots et d'une largeur de 16 bits par mot.

Les deux mémoires du haut doivent être actives en même temps afin d'obtenir une donnée complète sur 16 bits (par exemple, $4D5F_{16}$ est le contenu de l'adresse 0).

Il en est de même pour les deux mémoires du bas (par exemple, $18F4_{16}$ est le contenu de l'adresse 65 535).

Mais attention, quand les deux mémoires du haut sont actives, alors les deux mémoires du bas doivent être inactives (et inversement), faute de quoi deux données différentes seraient positionnées sur le bus de donnée. Cela provoquerait alors un conflit d'accès.

Pour arriver à ce résultat, il faut câbler les mémoires de la façon suivante :



On reconnaît assez facilement les deux types d'assemblage :
Les mémoires 1 et 2 sont en parallèle. Elles doivent être actives en même temps (elles ont le même *CS*).
Les mémoires 3 et 4 sont en parallèle. Elles doivent être actives en même temps (elles ont le même *CS*).
Les mémoires 1 et 3 sont en série. Elles ne doivent jamais être actives en même temps.
Les mémoires 2 et 4 sont en série. Elles ne doivent jamais être actives en même temps.

Dans le cas où le *CS* de la mémoire externe est à 1, on a les deux cas suivants :

- Si *A15* = 0 alors *CS0* = 1 et *CS1* = 0 → ce sont les mémoires 1 et 2 qui sont actives ;
- Si *A15* = 1 alors *CS0* = 0 et *CS1* = 1 → ce sont les mémoires 3 et 4 qui sont actives.

Dans le cas où le *CS* de la mémoire externe est à 0, alors toutes les mémoires internes sont désactivées.

III. Le décodage d'adresse

1. Introduction

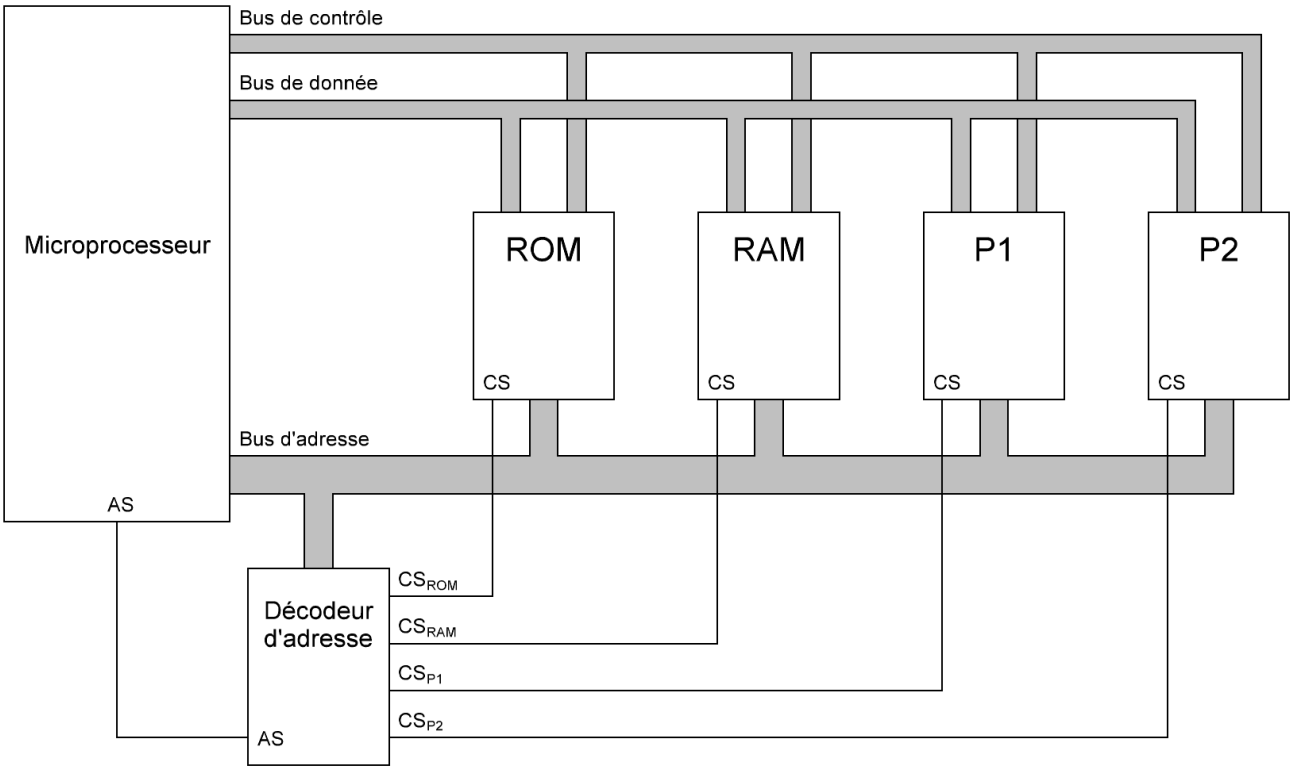
L'objectif du décodage d'adresse est de faire communiquer un microprocesseur avec différents composants (mémoires, cartes, périphériques, etc.).

Afin d'illustrer les principales techniques de décodage d'adresse que nous aborderons dans la suite de ce cours, nous allons nous servir d'un exemple concret :

On souhaite faire communiquer un microprocesseur avec quatre composants : une mémoire de type ROM, une mémoire de type RAM et deux périphériques quelconques que l'on notera **P1** et **P2**.

Composant	Bus d'adresse	Bus de donnée	Capacité d'adressage
Microprocesseur	20 fils	8 fils	1 Mio
ROM	16 fils	8 fils	64 Kio
RAM	15 fils	8 fils	32 Kio
P1	10 fils	8 fils	1 Kio
P2	4 fils	8 fils	16 octets

La communication entre le microprocesseur et les différents composants se fait à l'aide des bus de donnée, d'adresse et de contrôle. La sélection d'un composant nécessite un décodeur d'adresse.



Les sorties du décodeur d'adresse sont les entrées *CS* de tous les composants. Ses entrées sont le bus d'adresse et le signal *AS* (*Address Strobe*). Ce dernier est généré par le microprocesseur (il fait partie de son bus de contrôle). Il est actif (*AS* = 1) quand l'adresse présente sur le bus d'adresse du microprocesseur est valide. Par conséquent, quand il est inactif (*AS* = 0) aucun des composants ne doit être activé. C'est pourquoi le décodeur d'adresse doit prendre ce signal en compte.

Tous les composants souhaitant échanger des informations avec le microprocesseur doivent posséder une entrée d'activation *CS*, car un seul composant à la fois doit accéder aux bus. Si plusieurs composants accèdent aux bus en même temps, cela produira un conflit d'accès.

Le microprocesseur utilise son bus d'adresse pour sélectionner un composant.

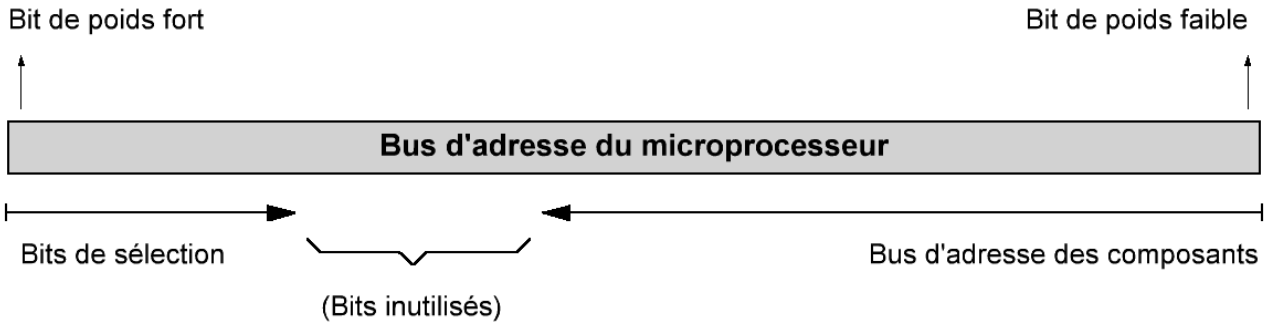
Le décodage d'adresse consiste donc à décoder l'adresse présente sur le bus d'adresse du microprocesseur et à activer un seul composant en fonction de la valeur de cette adresse.

Un décodeur d'adresse doit donc être mis en place. Ce dernier positionnera sur ses sorties la valeur des différents signaux d'activation *CS* en fonction de l'adresse présente sur le bus d'adresse du microprocesseur.

Il existe de nombreuses techniques de décodage d'adresse. Nous nous limiterons à deux de ces techniques : **le décodage linéaire** et **le décodage par zone**. L'apprentissage de ces deux types de décodage suffira à comprendre les principes et les concepts de base du décodage d'adresse.

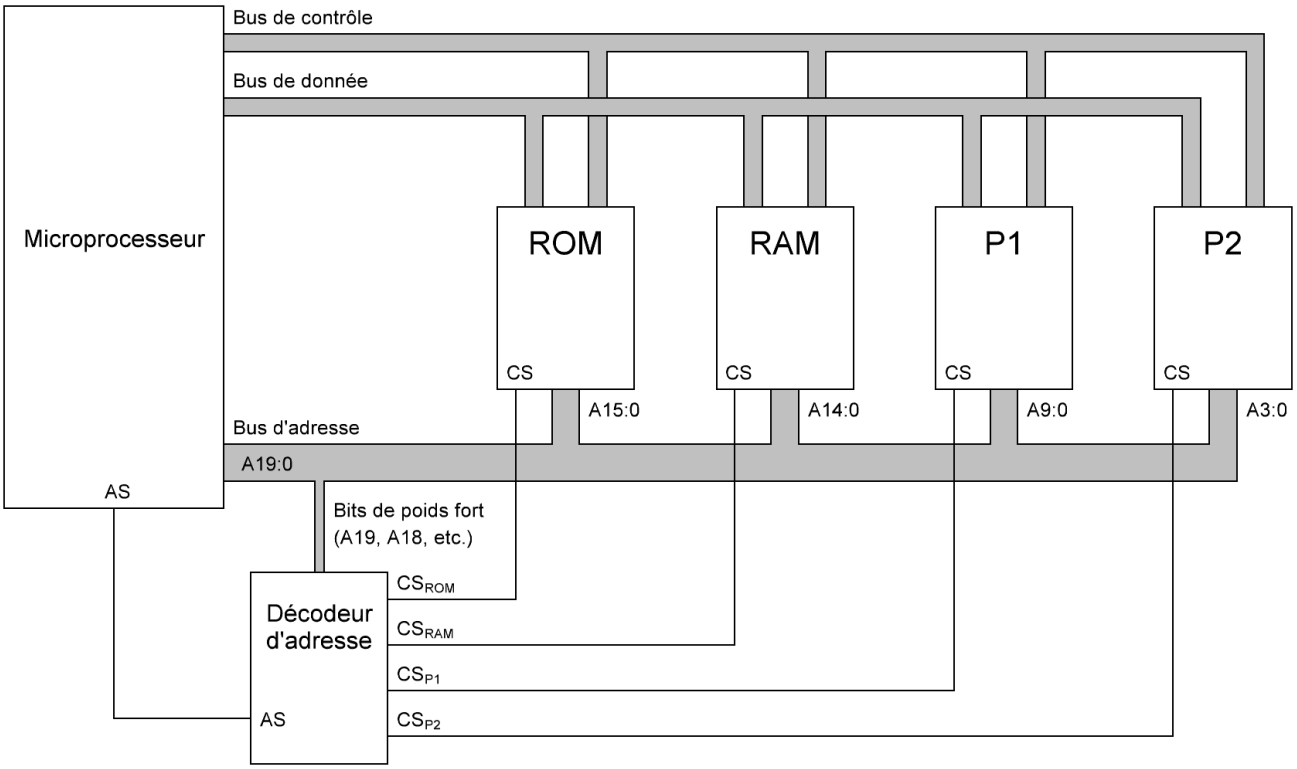
Nous allons commencer par aborder les points communs à ces deux types de décodage :

Le bus d'adresse du microprocesseur est divisé en deux parties :
- Les bits de poids fort servent à la sélection des composants ;
- Les bits de poids faible servent à contenir les bus d'adresse des composants.



Selon les configurations (nombre de composants, taille du bus d'adresse des composants, etc.), certains bits du bus d'adresse du microprocesseur peuvent être inutilisés. Lorsque le cas se présentera, nous positionnerons ces bits à 0.

Nous pouvons maintenant détailler un peu plus le schéma de câblage en précisant la répartition des bits de poids fort et de poids faible du bus d'adresse du microprocesseur.



- Les bits de poids faible du bus d'adresse du microprocesseur sont reliés aux bus d'adresse des différents composants ;
- Les bits de poids fort du bus d'adresse du microprocesseur sont les entrées du circuit logique de décodage. Ce sont les bits de sélection. Le nombre de bits de sélection dépend de la technique de décodage utilisée.

2. Le décodage linéaire

Le principe du décodage linéaire est d'associer un bit de sélection par composant (en partant du bit de poids fort).

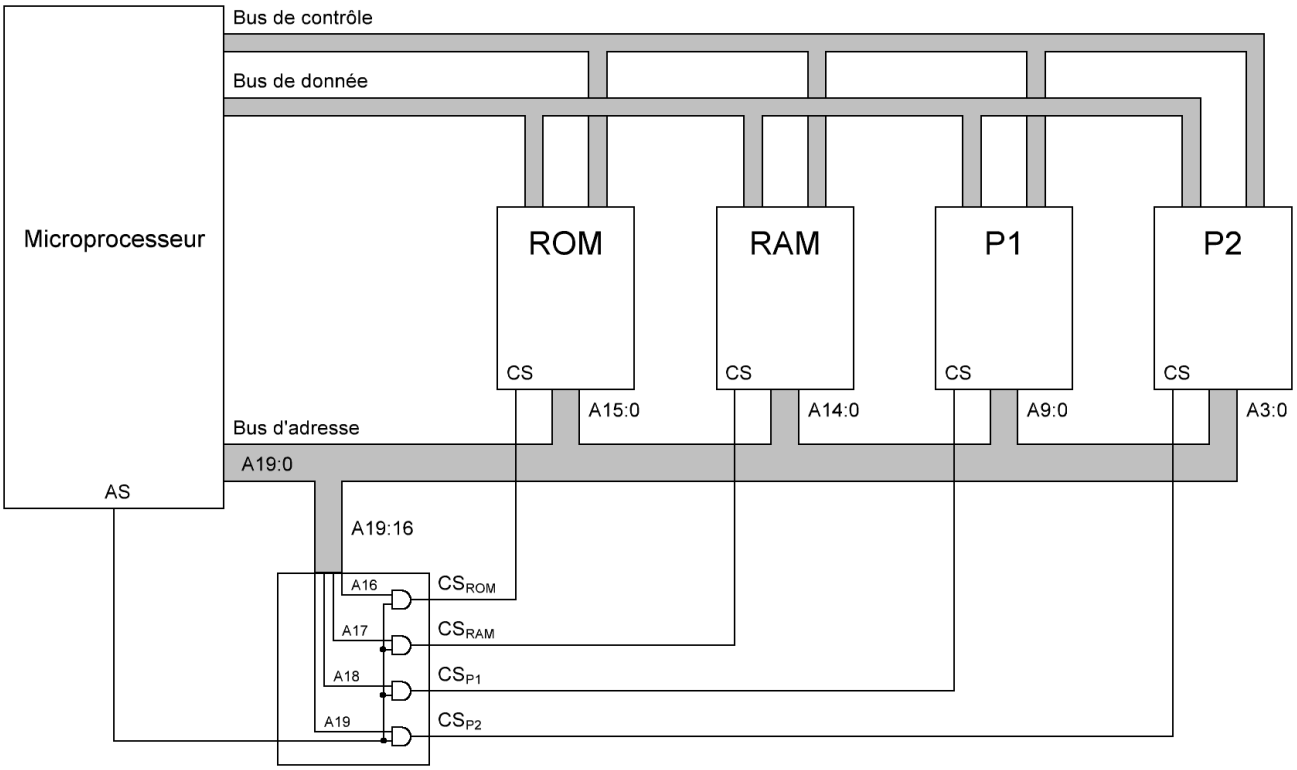
Dans notre exemple, quatre composants nécessitent quatre bits de sélection : *A19*, *A18*, *A17* et *A16*. Chacun de ces bits doit être associé à un composant. Dans l'absolu, le choix d'associer tel composant à tel bit de sélection est arbitraire. Le concepteur fera un choix en fonction de ses besoins et de ses contraintes.

Bit de sélection	Composant associé	
A16	ROM	← Si *A16* = 1, alors la ROM est activée.
A17	RAM	← Si *A17* = 1, alors la RAM est activée.
A18	P1	← Si *A18* = 1, alors P1 est activé.
A19	P2	← Si *A19* = 1, alors P2 est activé.

2.1. Décodeur d'adresse

La logique d'un décodage linéaire est très simple. Le *CS* d'un composant doit être mis à 1 quand son bit de sélection est à 1. Aussi, comme cela a été dit précédemment, quand la valeur du bus d'adresse est invalide (*AS* = 0), aucun composant ne doit être activé.

CS_{ROM} = AS.A16
CS_{RAM} = AS.A17
CS_{P1} = AS.A18
CS_{P2} = AS.A19



2.2. Représentation de l'espace mémoire

La représentation de l'espace mémoire d'un microprocesseur indique pour tous les composants, les adresses auxquelles ces derniers sont accessibles par le microprocesseur.

Il faut donc déterminer l'adresse la plus basse et l'adresse la plus haute de chaque composant dans l'espace mémoire.

Il faut pour cela déterminer les bits de poids fort (sélection) et les bits de poids faible (adressage du composant) du bus d'adresse du microprocesseur pour chaque composant. S'il existe des bits inutilisés, qui ne servent ni à l'adressage, ni à la sélection, ils seront positionnés à 0.

Par exemple pour la ROM :

- *A16* est à 1 pour sélectionner la ROM ;
- *A17*, *A18* et *A19* sont à 0 pour désactiver les autres composants ;
- Pour son adresse basse : on positionne ses 16 bits d'adresse à 0 ;
- Pour son adresse haute : on positionne ses 16 bits d'adresse à 1.

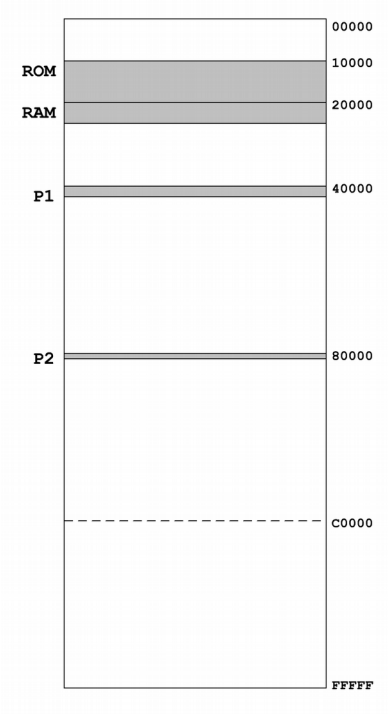
Ce qui nous donne pour tous les composants :

ROM basse : **0001 0000 0000 0000 0000**$_2$ = 10000_{16}
ROM haute : **0001 1111 1111 1111 1111**$_2$ = $1FFFF_{16}$

RAM basse : **0010** 0000 0000 0000 **0000**$_2$ = 20000_{16}
RAM haute : **0010** 0111 1111 1111 **1111**$_2$ = $27FFF_{16}$

P1 basse : **0100** 0000 0000 **0000 0000**$_2$ = 40000_{16}
P1 haute : **0100** 0000 0011 **1111 1111**$_2$ = $403FF_{16}$

P2 basse : **1000** 0000 0000 0000 **0000**$_2$ = 80000_{16}
P2 haute : **1000** 0000 0000 0000 **1111**$_2$ = $8000F_{16}$



Par exemple, si le microprocesseur positionne l'adresse $207F2_{16}$ sur son bus d'adresse, alors c'est l'adresse $7F2_{16}$ de la RAM qui est sélectionnée :

Bus d'adresse du microprocesseur

	A19 P2	A18 P1	A17 RAM	A16 ROM	A15	A14:0
207F2 →	0	0	1	0	0	000 0111 1111 0010
	↓				↓	↓
	Sélection de la RAM				Bit inutilisé	Bus d'adresse de la RAM : 7F2

2.3. Conclusion

Quelques caractéristiques du décodage d'adresse linéaire :

- Il est facile à mettre en œuvre : la logique de décodage ne contient que des fils ;
- Le nombre de composants connectables est vite limité : associer un fil par composant n'est pas très optimisé ;
- Il ne protège pas contre les conflits d'accès.

Remarque :
Le principal défaut du décodage linéaire est qu'il ne protège pas contre les conflits d'accès. C'est-à-dire que si le microprocesseur positionne sur son bus d'adresse une valeur qui contient plusieurs bits de sélection à 1, alors plusieurs composants seront sélectionnés en même temps.

Il est toutefois possible de modifier facilement la logique de décodage afin de protéger les composants. Il suffit d'activer le *CS* d'un composant uniquement si les autres sont désactivés. C'est-à-dire si le bit de sélection qui leur est associé est à 0. Ce qui donne :

$CS_{ROM} = AS.\overline{A19}.\overline{A18}.\overline{A17}.A16$
$CS_{RAM} = AS.\overline{A19}.\overline{A18}.A17.\overline{A16}$
$CS_{P1} = AS.\overline{A19}.A18.\overline{A17}.\overline{A16}$
$CS_{P2} = AS.A19.\overline{A18}.\overline{A17}.\overline{A16}$

3. Le décodage par zone

Le principe du décodage par zone est de diviser l'espace mémoire en un certain nombre de zones de même taille. Une zone correspond à une combinaison des bits de sélection. Les bits de sélection sont toujours les bits de poids fort.

Par exemple :
- 1 bit de sélection donne 2 zones ;
- 2 bits de sélection donnent 4 zones ;
- *n* bits de sélection donnent 2^n zones.

1 bit de sélection	A19
Zone 0	0
Zone 1	1

2 bits de sélection	A19:18
Zone 0	00
Zone 1	01
Zone 2	10
Zone 3	11

Dans notre exemple, le plus grand bus d'adresse est celui de la ROM qui possède 16 fils. Il reste donc quatre fils disponibles pour la sélection. Ce qui nous laisse les quatre possibilités suivantes :
- 1 fil de sélection → 2 zones de 512 Kio ;
- 2 fils de sélection → 4 zones de 256 Kio ;
- 3 fils de sélection → 8 zones de 128 Kio ;
- 4 fils de sélection → 16 zones de 64 Kio.

Le décodage par zone permet d'associer un composant par zone. Puisque dans notre exemple nous avons quatre composants, il nous faudra au moins quatre zones ; c'est-à-dire au moins deux fils de sélection. Il est possible d'avoir plus de zones que de composants. Les zones restantes seront inutilisées.

Il faut maintenant associer les composants à leur zone. Dans l'absolu, le choix du nombre de zones et celui d'associer tel composant à telle zone sont arbitraires. Le concepteur fera un choix en fonction de ses besoins et de ses contraintes. Des zones vides peuvent laisser la possibilité d'ajouter d'autres composants par la suite et donc de faire évoluer plus facilement le système.

Pour la suite de l'exemple, nous choisirons de diviser l'espace mémoire en huit zones et d'associer les composants aux quatre premières zones :

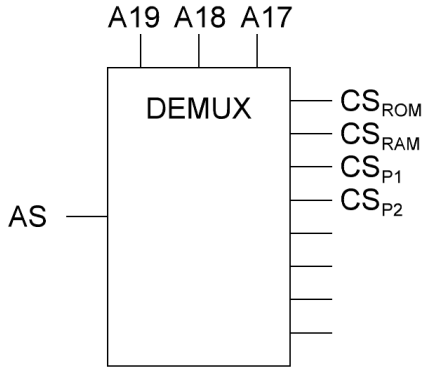
Zone	A19	A18	A17	Composant
0	0	0	0	ROM
1	0	0	1	RAM
2	0	1	0	P1
3	0	1	1	P2
4	1	0	0	*zone inutilisée*
5	1	0	1	*zone inutilisée*
6	1	1	0	*zone inutilisée*
7	1	1	1	*zone inutilisée*

3.1. Décodeur d'adresse

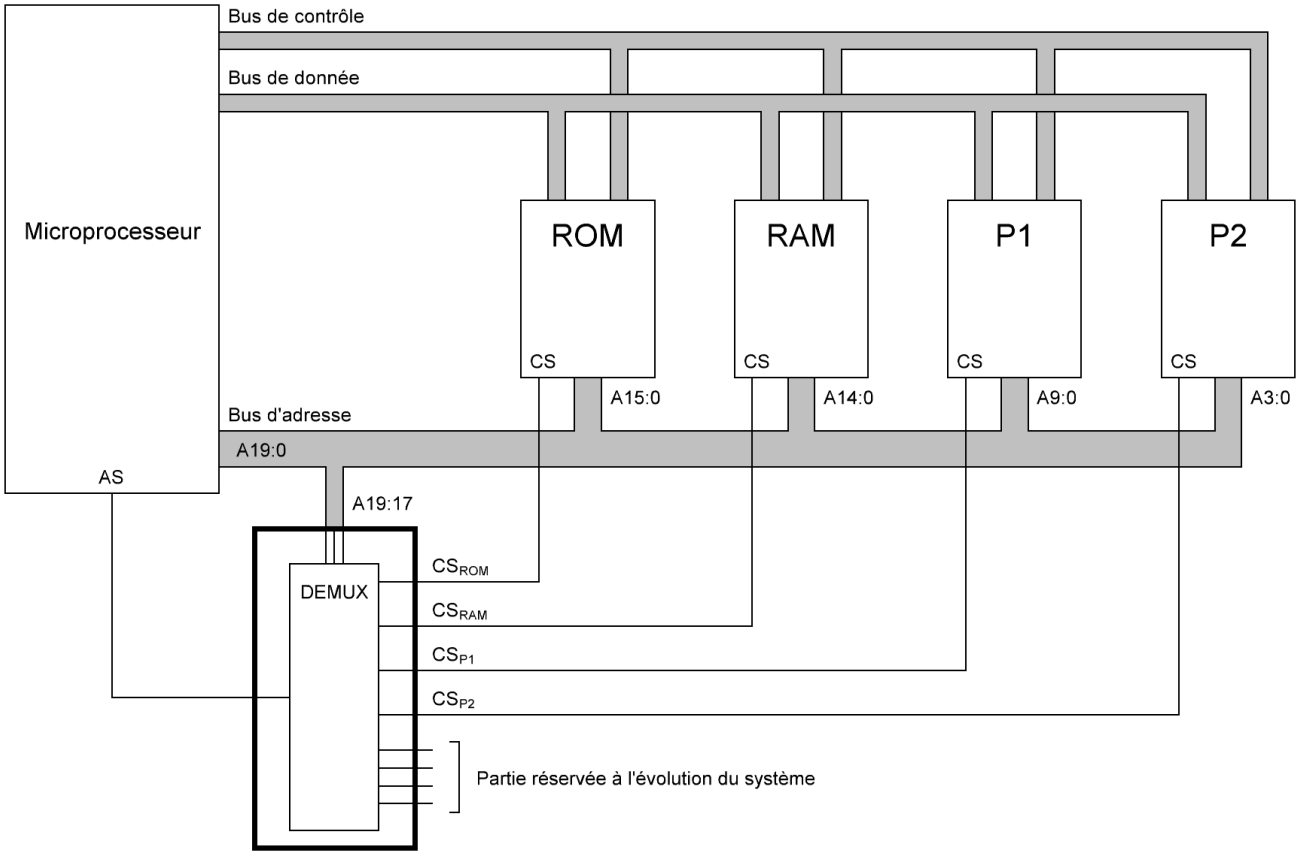
Le *CS* de chaque composant doit être activé lorsque la combinaison de bits associée au composant est présente sur les bits de sélection. Aussi, comme cela a été dit précédemment, quand la valeur du bus d'adresse est invalide (*AS* = 0), aucun composant ne doit être activé.

$CS_{ROM} = AS.\overline{A19}.\overline{A18}.\overline{A17}$
$CS_{RAM} = AS.\overline{A19}.\overline{A18}.A17$
$CS_{P1} = AS.\overline{A19}.A18.\overline{A17}$
$CS_{P2} = AS.\overline{A19}.A18.A17$

Cette logique de décodage représente celle d'un démultiplexeur (3 lignes de sélection, 8 sortie).



Pour terminer, on obtient le schéma de câblage suivant :



3.2. Représentation de l'espace mémoire

Le principe est le même que pour le décodage linéaire. Il faut déterminer l'adresse la plus basse et l'adresse la plus haute de chaque composant dans l'espace mémoire.

Il faut pour cela déterminer les bits de poids fort (sélection) et les bits de poids faible (adressage du composant) du bus d'adresse du microprocesseur pour chaque composant. S'il existe des bits inutilisés, qui ne servent ni à l'adressage ni à la sélection, ils seront positionnés à 0.

Par exemple pour la ROM :

- *A16* = 0, car il est inutilisé ;
- *A17*, *A18* et *A19* sont à 0 pour la sélectionner ;
- Pour son adresse basse : on positionne ses 16 bits d'adresse à 0 ;
- Pour son adresse haute : on positionne ses 16 bits d'adresse à 1.

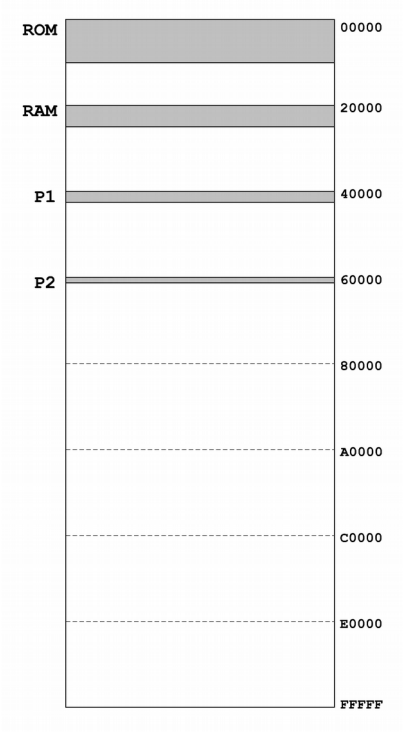
Ce qui nous donne pour tous les composants :

ROM basse : **0000 0000 0000 0000 0000**$_2$ = 00000$_{16}$
ROM haute : **0000 1111 1111 1111 1111**$_2$ = 0FFFF$_{16}$

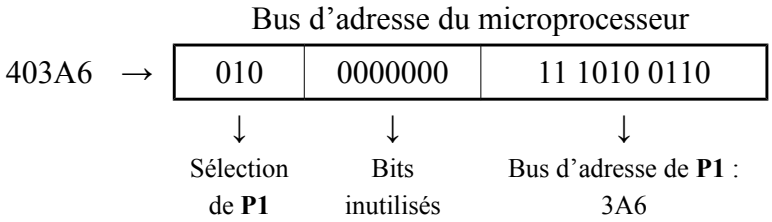
RAM basse : **0010 0000 0000 0000 0000**$_2$ = 20000$_{16}$
RAM haute : **0010 0111 1111 1111 1111**$_2$ = 27FFF$_{16}$

P1 basse : **0100 0000 0000 0000 0000**$_2$ = 40000$_{16}$
P1 haute : **0100 0000 0011 1111 1111**$_2$ = 403FF$_{16}$

P2 basse : **0110 0000 0000 0000 0000**$_2$ = 60000$_{16}$
P2 haute : **0110 0000 0000 0000 1111**$_2$ = 6000F$_{16}$



Par exemple, si le microprocesseur positionne l'adresse 403A6$_{16}$ sur son bus d'adresse, alors c'est l'adresse 3A6$_{16}$ de **P1** qui est sélectionnée :



3.3. Conclusion

Quelques caractéristiques du décodage d'adresse par zone :

- Il est facile à mettre en œuvre : la logique de décodage est celle d'un décodeur ;
- Le nombre de composants connectables est plus important que celui d'un décodage linéaire ;
- Plus le nombre de zones est élevé, plus la taille des composants est réduite (et inversement) ;
- Aucun risque de conflit d'accès.

4. Effet miroir et redondance

Lorsqu'un ou plusieurs bits d'adresse du microprocesseur ne sont pas utilisés, un composant peut se retrouver à différents emplacements dans l'espace mémoire du microprocesseur.

Par exemple, la ROM utilisée précédemment (lors du décodage d'adresse par zone) possède les adresses haute et basse suivantes :
ROM basse : $\mathbf{0000\ 0000\ 0000\ 0000\ 0000}_2 = 00000_{16}$
ROM haute : $\mathbf{0000\ 1111\ 1111\ 1111\ 1111}_2 = 0FFFF_{16}$

Nous pouvons remarquer que *A16* est inutilisé.

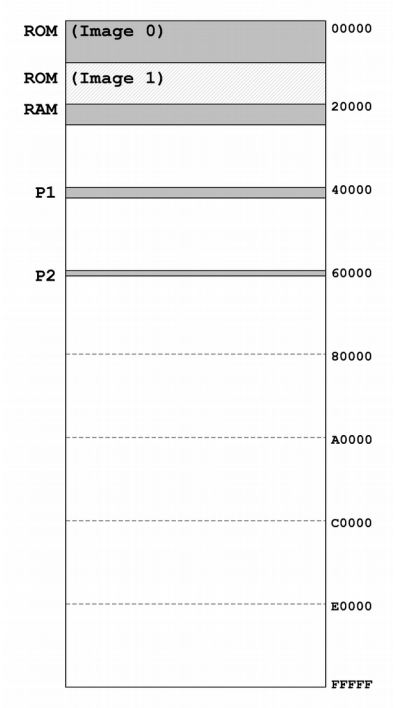
Jusqu'à présent, les bits inutilisés étaient mis à 0, mais nous pouvons tout à fait les mettre à 1. En fait, quelle que soit la valeur de ces bits, le composant activé et l'adresse sélectionnée ne changent pas.

Par exemple, si le bit *A16* est positionné à 1:
ROM basse : $\mathbf{0001\ 0000\ 0000\ 0000\ 0000}_2 = 10000_{16}$
ROM haute : $\mathbf{0001\ 1111\ 1111\ 1111\ 1111}_2 = 1FFFF_{16}$

Par conséquent, la ROM se trouve également en mémoire de l'adresse 10000_{16} à l'adresse $1FFFF_{16}$. Il s'agit d'une image redondante de la ROM.

Pour accéder à l'adresse $ABCD_{16}$ de la ROM, le microprocesseur peut donc positionner sur son bus d'adresse soit $0ABCD_{16}$ soit $1ABCD_{16}$.

Si l'on représente les deux images de la ROM dans l'espace mémoire, voici ce que nous obtenons :



Nous pouvons déterminer le nombre d'images pour chaque composant.

Le nombre d'images est simplement le nombre de combinaisons que peuvent prendre les bits d'adresse inutilisés du microprocesseur.

Bits inutilisés = 20 bits d'adresse – 3 bits de sélection – bits d'adresse du composant

ROM : 20 – 3 – 16 = 1 bit inutilisé → $\mathbf{2^1}$ **= 2 images**
RAM : 20 – 3 – 15 = 2 bit inutilisés → $\mathbf{2^2}$ **= 4 images**
P1 : 20 – 3 – 10 = 7 bit inutilisés → $\mathbf{2^7}$ **= 128 images**
P2 : 20 – 3 – 4 = 13 bit inutilisés → $\mathbf{2^{13}}$ **= 8 192 images**

L'espace mémoire devient alors :