UNIVERSITE FRANCOIS-RABELAIS

TOURS

ECOLE D'INGENIEURS EN INFORMATIQUE POUR L'INDUSTRIE

Notes du cours
"Microprocesseur" de 1ère année.

Plan du cours

- Introduction

- Architecture du microprocesseur

    - architecture externe
    - architecture interne
        - le schéma fonctionnel
        - l'UAL
        - l'accumulateur
        - le compteur d'instructions
        - le registre d'adresses
        - le registre d'instructions
        - le registre d'état
        - les registres temporaires de l'UAL
        - les registres généraux
        - la logique de contrôle

- Introduction au jeu d'instructions du microprocesseur

        - le jeu d'instructions
        - le code mnémonique
        - les modes d'adressage

- la pile
- Les interruptions
- Les mémoires
- Les interfaces

        - le P.I.A. (Interface parallèle)
        - le P.T.M. (Compteur programmable)
        - l'A.C.I.A. (Interface série)

Historique

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

Quelques dates :

☞**1690** : Pascal invente la machine à calculer entièrement mécanique (addition et soustraction)

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

☞**1810** : Invention de l'orgue de barbarie (succession de cartes perforées).

☞**1940** : Premier ordinateur à relais mécaniques (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.

☞**1984** : Apparition du microprocesseur 32 bits avec un million de transistor sur la puce.

☞**1994** : Apparition du Pentium avec 3,5 millions de transistors.

Commentaires sur le graphe issu de Sciences et Vie n° spécial 1996.

C'est en 1971 que le premier microprocesseur est sorti des laboratoires d'Intel. Travaillant sur 4 bits et d'une puissance faible l'intérêt de ce nouveau composant électronique ne fut pas évident jusqu'à ce que l'idée de le transformer en calculatrice fut trouvée.

Sept ans plus tard, l'arrivée du 8088 multiplie déjà cette puissance de calcul par 200 !
Cette date correspond à la naissance des véritables micro-ordinateurs.

Arrivent ensuite les microprocesseurs 68000 et 80286 (16 bits) avec les Macintosh et P.C. que nous connaissons. Ils ont introduisent l'image et le son.

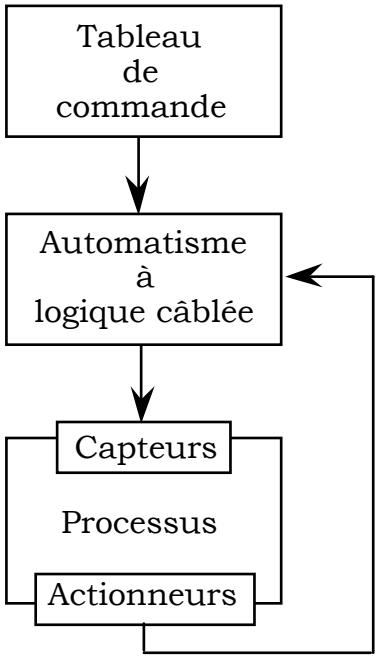
Ensuite, tout n'est plus qu'une question de course à la puissance de calcul. Chaque bond technologique apporte sont innovation.

Aujourd'hui, le multimédia puis le 3 D et le temps réel.

Demain, le monde virtuel !

Principes de base

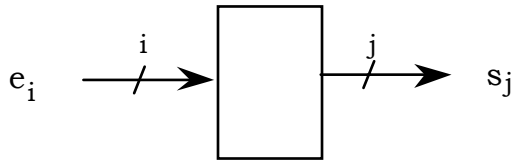
Les premiers automatismes étaient réalisés à partir de la *logique cablée* selon le synoptique suivant :



les systèmes à logique cablée sont conçus à l'aide de circuits intégrés logiques.

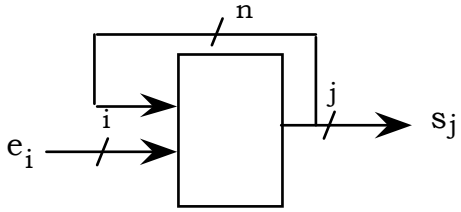
Certains de ces circuits font appel à :

La logique combinatoire



(les sorties sont définis uniquement à partir des variables d'entrée)

la logique séquentielle

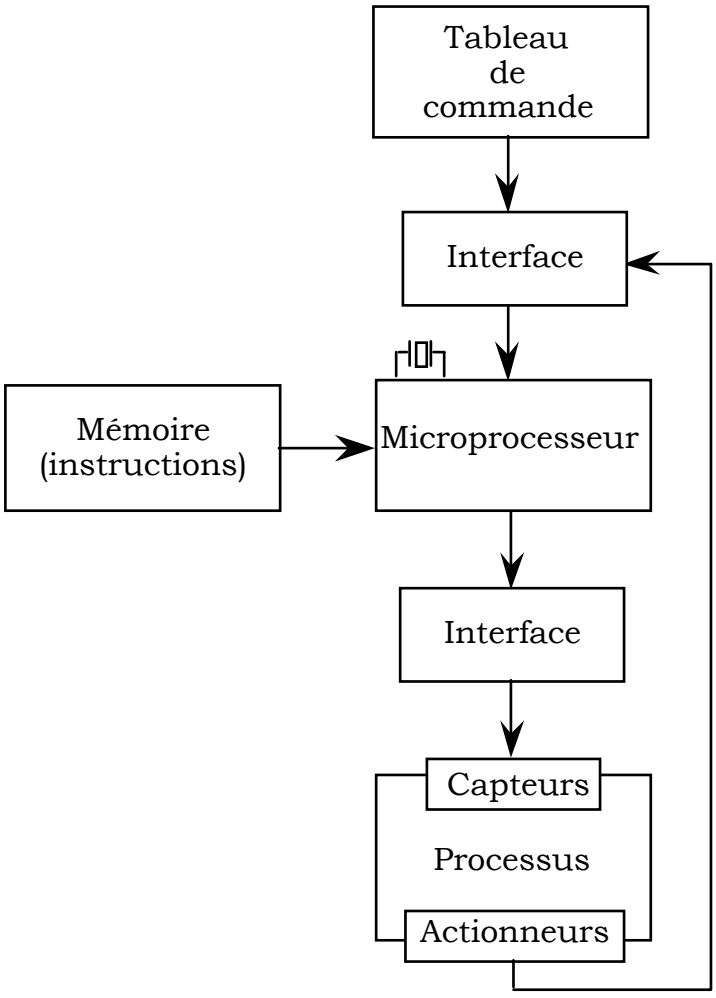


(les sorties dépendent toujours des entrées mais aussi des états antérieurs)

Le microprocesseur donne naissance au principe de la logique programmée.

Le fonctionnement n'est plus défini par un ensemble de circuits logiques, câblés entre eux, mais par une suite ordonnée d'instructions stockées en mémoire et gérées par cet élément.

Nouveau synoptique :

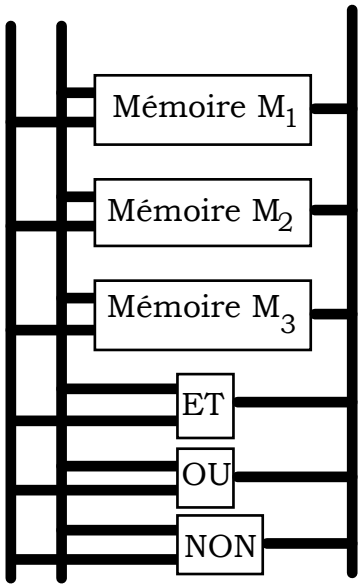


Principe de la logique programmée

Illustration à partir d'un circuit simple constitué de 3 cases mémoires et 3 portes logique ET, OU et NON.

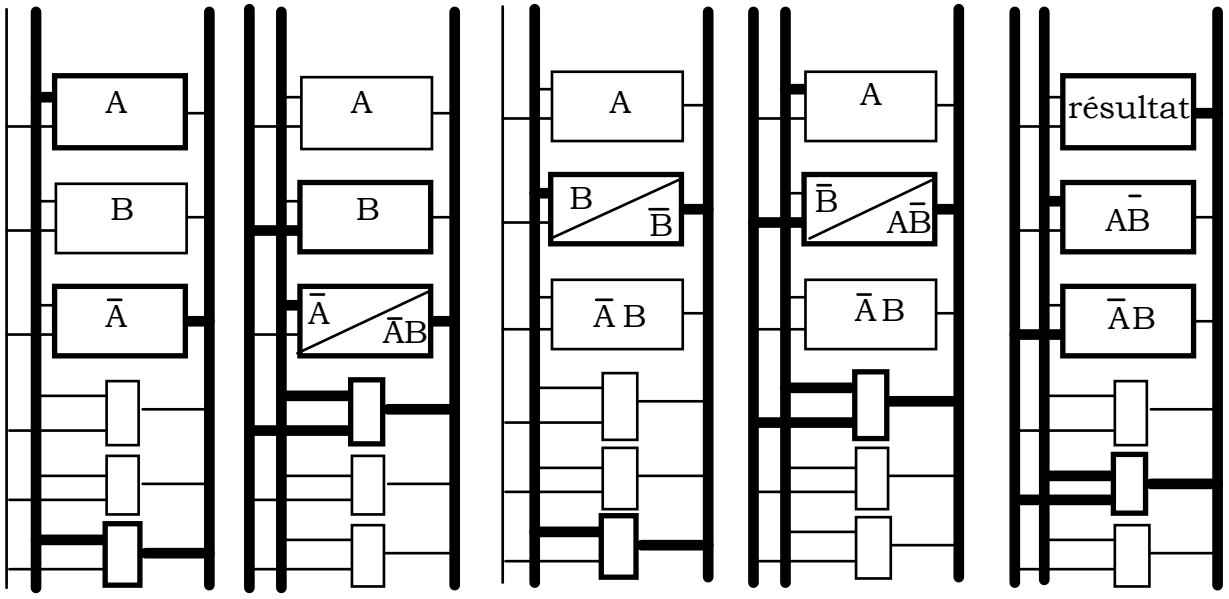
On se propose de réaliser la fonction : A exclusif B

Schéma :

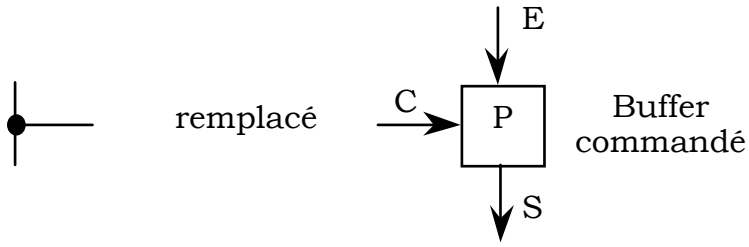


Initialisation [M1] = A et [M2] = B

Déroulement :



Réalisation

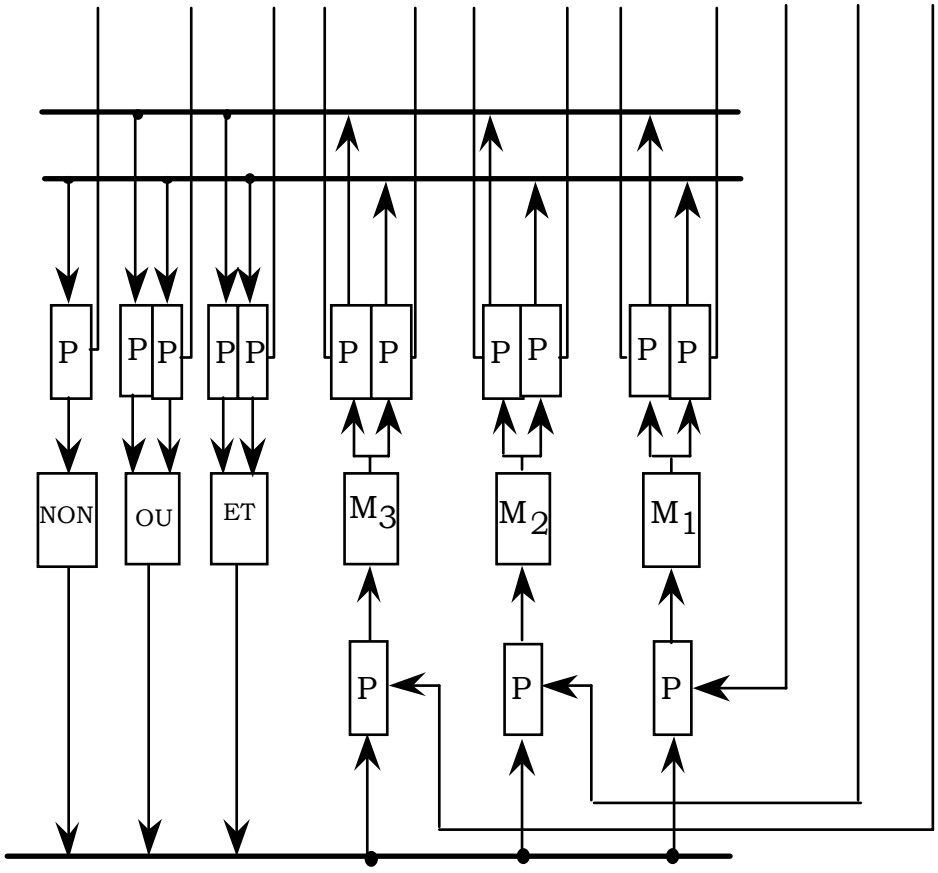


légende :

C = 0 porte bloquée

C = 1 porte transparente donc S = E

1	1	0	0	0	0	0	0	0	1	0	0	1	$\bar{A}$
2	0	0	1	0	1	1	0	0	0	0	0	1	$\bar{A}B$
3	1	0	0	0	0	0	1	0	0	0	1	0	$\bar{B}$
4	0	0	1	0	0	0	1	1	0	0	1	0	$A\bar{B}$
5	0	1	0	0	1	1	0	0	0	0	0	1	$\bar{A}B+A\bar{B}$



La notion de programme

Un programme qui réalise une fonction particulière comprend :

Une suite d'instructions.

Chaque instruction est constituée de plusieurs micro-instructions.

Chaque micro-instruction génère plusieurs micro-commandes destinées à aiguiller correctement les informations.

Dans l'exemple du ou exclusif, le programme comprend une instruction constituée de 5 micro-instructions ou phases. Chaque phase génère des micro-commandes qui, au travers des 12 fils, aiguillent correctement les informations.

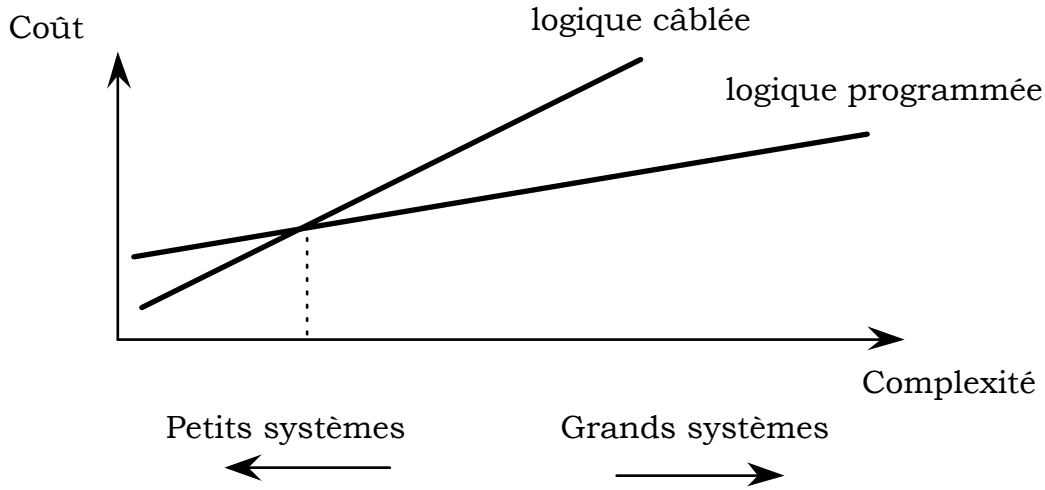
Remarques :

les phases sont commandées par une logique séquentielle synchrone.

Les micro-instructions sont stockées sous forme de mots dans une mémoire.

Choix entre les deux logiques

Courbe d'évolution du coût par rapport à la complexité.



Critères de choix :

	avantage	inconvénient
logique cablée	rapidité	spécificité
logique programmée	souplesse	lenteur

logique câblée :
Pour une configuration hardware donnée, nous avons une fonction donnée.
Il y a nécessité de modifier le hardware pour changer de fonction.

logique programmée
Le hardware est figé. C'est le programme qui crée la fonction.
Pour une suite d'instructions donnée, nous avons une fonction donnée.
Il y a nécessité de modifier le programme pour une nouvelle fonction

Le microprocesseur

Définition

le microprocesseur, noté aussi M.P.U. (Microprocessor unit) ou encore C.P.U. (Central Processing Unit) est un circuit intégré complexe appartenant à la famille des VLSI (Very large scale intégration) capable d'effectuer séquentiellement et automatiquement des suites d'opérations élémentaires.

Son rôle

Ce circuit remplit deux fonctions essentielles :

le traitement des données

On parle d'unité de traitement. Cette fonction est dédiée à l'U.A.L.
Elle concerne la manipulation des données sous formes de transfert, opérations arithmétiques, opérations logiques....

le contrôle du système

Cette fonction se traduit par des opérations de décodage et d'exécution des ordres exprimés sous forme d'instruction.

Puissance d'un microprocesseur

Définition :

La notion de puissance est la capacité de traiter un grand nombre d'opérations par seconde sur de grands nombres et en grande quantité.

Intrinsèquement la puissance se joue donc sur les trois critères suivants:

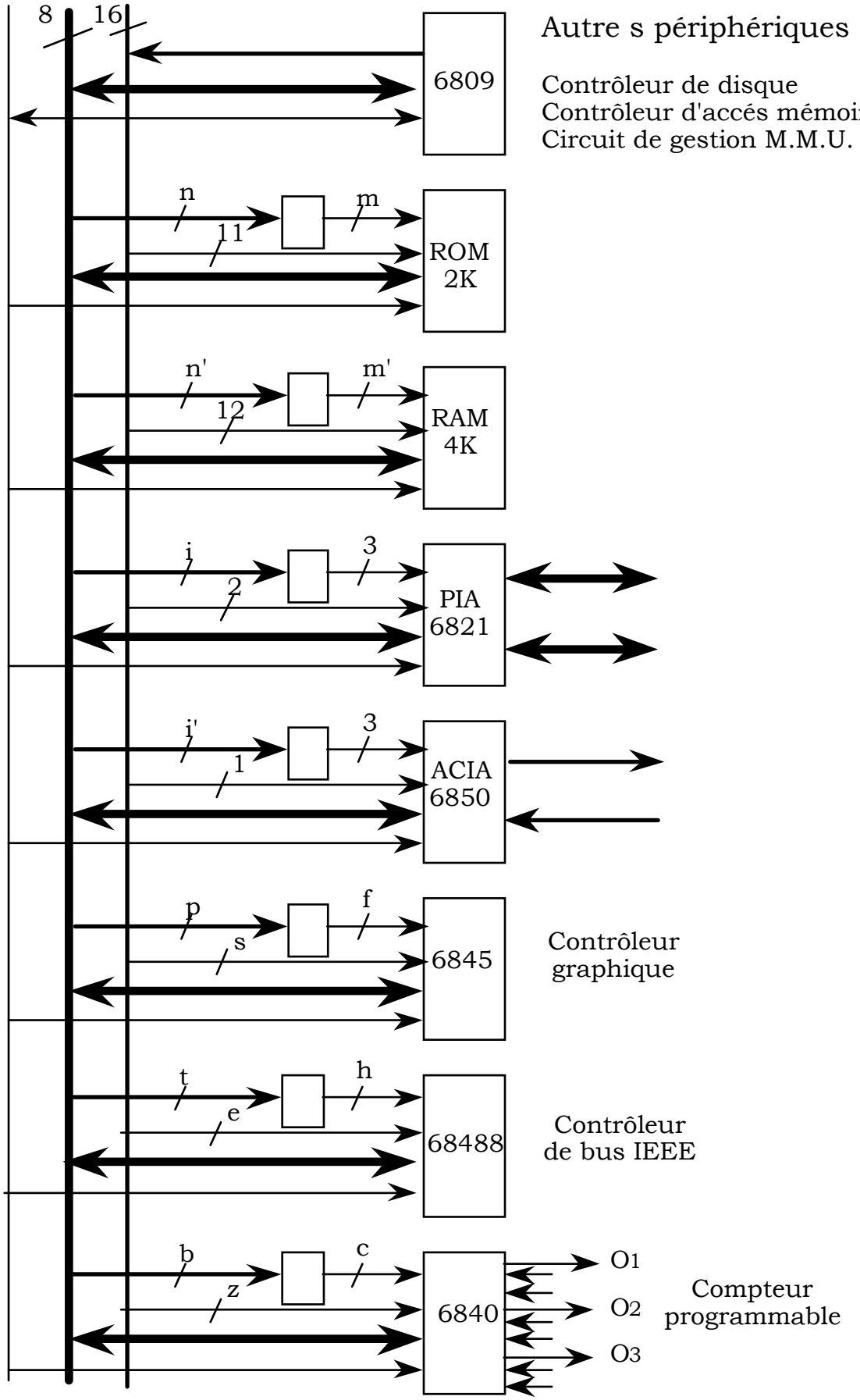
La longueur des mots : données et instructions (on parle de largeur du bus des données).

Le nombre d'octets que le microprocesseur peut adresser (on parle de largeur du bus des adresses).

La vitesse d'exécution des instructions liée à la fréquence de fonctionnement de l'horloge de synchronisation exprimée en MHZ.

l'aspect dimensionnel renseigne assez bien de la puissance du composant.

Système à base du microprocesseur 6809.



SCHEMA FONCTIONNEL D'UN MICROPROCESSEUR
(8bits)

On distingue 3 éléments logiques principaux :

- ❒ Une Unité Arithmétique et Logique (U.A.L.)
- ❒ Un Accumulateur.

et

- ❒ Des registres que l'on nomme couramment :
  - Le Compteur d'Instructions (C.I.)
  - Le Registre d'état
  - Le Registre d'Instructions (R.I.)
  - Le Registre d'Adresses (R.A.)
  - Le Registre temporaire des données

De base, il existe 6 registres fondamentaux dans une architecture de microprocesseur 8 bits.

(Des registres supplémentaires sont ajoutés pour rendre la vie plus facile aux programmeurs)

Cet ensemble est interconnecté au travers de différents bus.

On trouve trois types de bus :

- ❒ Le bus des données (bi-directionnel)
- ❒ Le bus des adresses (uni-directionnel)
- ❒ Le bus de contrôle (bi-directionnel)

Remarquer le bus interne de données qui relie tous les différents éléments du micro-processeur.

L'Unité Arithmétique et Logique

Son rôle :

Ce circuit permet de traiter et tester les données.

Toute instruction qui modifie une donnée fait toujours appel à l'UAL.

L'entrée de L'UAL est connectée au bus interne via
- p des registres "temporaires"
- p un registre particulier appelé "accumulateur".

La sortie de l'UAL est connectée uniquement à l'entrée de l'accumulateur.

Noter :

les deux entrées sont précédées par une mémoire tampon.
On les appelle encore des registres tampons ou verrou.

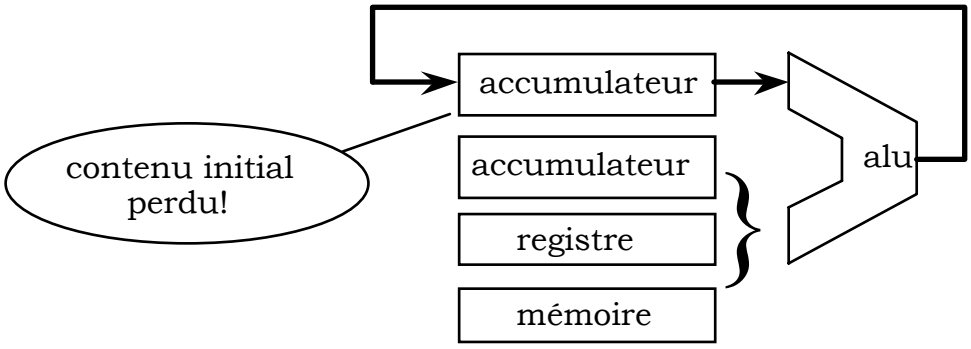
Ces registres permettent de stocker des octets aux entrées de l'U.A.L.
L'UAL étant constitué d'une logique combinatoire, elle est dépourvue de moyen propre de stockage.

Ce type de registre ne peut être manipulé par le programmeur. Il lui est totalement transparent.

L'accumulateur

C'est le registre le plus important du microprocesseur, il sert systématiquement lorsque le μp a besoin de "manipuler" des données.

La plupart des opérations logiques et arithmétiques sur les données font appel au couple "UAL - accumulateur" selon la procédure suivante:



Il en est de même pour les déplacements et transferts des données d'un endroit à un autre comme :

de mémoire à mémoire.
de mémoire à unités d'entrée-sortie (I/O).

Cette action se fait en deux temps :
source *vers* Accumulateur et ensuite Accumulateur *vers* destination.

Les instructions supportées par un accumulateur sont très nombreuses. Au niveau de la programmation, il représente une grande souplesse d'utilisation!

Les autres registres ne permettent que des opérations limitées.

Certains microprocesseur, possèdent des accumulateurs de longueur double tel D chez Motorola et HL chez Intel - dissociés en deux et généralement baptisés individuellement A et B ou H et L respectivement.

Gros avantage présenté par un μp possédant plusieurs accumulateurs : les opérations logiques et arithmétiques se font entre accumulateurs limitant ainsi les accès (transferts) avec l'extérieur.

Le Compteur d'Instructions

Appelé encore Compteur Programme (P.C.) ou Compteur Ordinal (C.O.)

Son rôle
Pointer TOUJOURS le premier octet d'une instruction.

Commentaires :
Le programme à exécuter est une succession d'instructions ordonnées (chaque instruction pouvant prend plusieurs octets!) qui se trouve rangé dans une zone mémoire, généralement à des adresses successives.

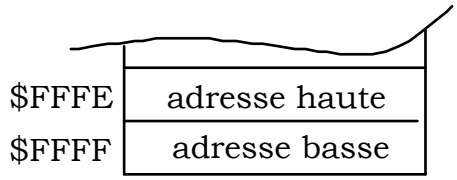
le P.C. "repère" le premier octet de chaque instruction du programme.

La taille du PC a une longueur de 16 bits ce qui lui permet d'adresser 65536 adresses mémoire soit 64 k octets (le champ mémoire).

Notons qu'il est connecté au bus interne des données.

A la mise sous tension, une valeur particulière est déposée sur le bus d'adresses (Dans le cas du 6809, cette valeur est $FFFE).

Le contenu des cases mémoires ($FFFE-$FFFF) représente en général l'adresse où se trouve le premier octet de la première instruction du programme.



Cette adresse est transmise aux circuits mémoires par l'intermédiaire du bus d'adresse via le Registre d'Adresses.

⚠ Le P.C. pointe toujours l'adresse du début de l'instruction suivante.
(A retenir, car parfois il est utile de connaître l'adresse présente.)
Il est possible de recharger le [P.C.] avec une adresse qui ne correspond pas au déroulement séquentiel du programme.
On trouve les détournements conditionnel et inconditionnel.

Le registre d'adresses

Son rôle :

Le Registre d'adresses ou (R.A.) sert d'interface entre le bus des données interne et le bus des adresses.

Il "pilote" le bus d'adresses du microprocesseur.

Commentaires :

D'une longueur de 16 bits, il est constitué de deux registres 8 bits (partie haute et partie basse).

Son contenu provient de différentes sources :

Le Compteur d'instruction
Un registre général
Un emplacement mémoire

Le contenu du registre d'adresse pointe la zone mémoire utile au microprocesseur.

⚠ Une fois que le premier octet de l'instruction en cours est décodé ...

Le contenu du Compteur d'Instructions est changé (contient l'adresse du début de l'instruction à venir).

Le contenu du Registre d'adresses change! donc

[C.I.] ≠ [R.A.]

Ce changement correspond...

soit à une incrémentation du contenu afin de lire l'information complémentaire de l'instruction en cours.

soit à un chargement d'une nouvelle valeur correspondant à une nouvelle zone mémoire utilisée temporairement par le microprocesseur (zone différente de celle où se trouve le programme).

Cette nouvelle valeur provient soit...

d'une lecture en mémoire (directement ou indirectement selon les modes d'adressage utililés)

d'un calcul (addition, ...)

Le Registre d'instructions

Sa tâche

Le registre d'instructions contient le premier octet de l'instruction en cours d'exécution.

Commentaires :

Le registre est chargé pendant le cycle de base **extraction-exécution**.

Il reçoit l'information (octet) grâce au bus de données auquel il est connecté.

L'information qu'il "capture" sur le bus des données est utilisé par le décodeur d'instructions. Suivant le protocole ci-dessous :

La donnée extraite de la mémoire est stockée dans le R.I. (c'est la phase extraction).

Ensuite ce contenu est interprété par le décodeur d'instructions qui agit alors sur la logique de contrôle (c'est la phase exécution).

Cet octet indique au microprocesseur deux choses :

Une action (une lecture, une écriture ou autre ...)

Un lieu d'action (un registre, un accumulateur, une case mémoire...)

Le résultat de cette interprétation se traduit par des niveaux logiques sur le bus de contrôle.

Le registre d'état

L'existence de ce registre permet de distinguer le simple calculateur du véritable ordinateur.

Son rôle

Stocker les résultats des tests effectués par l'U.A.L. après traitement sur les données

Commentaires :

L'existence de ces résultats permet d'écrire des programmes avec des branchements conditionnels (nouvelle adresse dans le C.I.).

En fonction de l'état des bits de ce registre le microprocesseur peut, alors, exécuter des programmes différents.

le microprocesseur prend en quelque sorte des "décisions".

Les bits les plus couramment utilisés sont :

a- *Le bit de retenue*

Ce bit est dans l'état actif lorsque le huitième bit du résultat de l'opération génére une retenue.`

b- *le bit de zéro*

Ce bit est actif lorsque l'opération a pour effet de mettre tous les bits d'un accumulateur ou d'un registre à la valeur logique 0 (très utilisé pour réaliser des compteurs).

c- *Le bit de signe*

Information qui indique que le bit le plus significatif (MSB) du contenu de l'accumulateur est un 1 logique.

Exemples d'application

Pour une soustraction - Selon les règles de l'arithmétique du complément à 2 cela signifie que le nombre est négatif.

Ces 3 bits de base sont parfois complétés par des bits supplémentaires choisis par le constructeur.

Les registres généraux

En plus des 6 registres de base que possèdent tous les microprocesseurs 8 bits, il peut en exister d'autres destinés à faciliter la tâche du programmeur. On les nomme registres généraux.

Sur notre schéma fonctionnel type, nous avons 3 registres généraux B, C et D.

Ce ne sont pas des registres puissants (tel un accumulateur) puisqu'ils n'ont pas de liaison directe avec la sortie de l'U.A.L.
Ces registres peuvent néanmoins affecter le Registre d'Etat.

Parfois, ils peuvent constituer un registre 16 bits - appelé paire de registre (ex : BC chez Intel ou D chez Motorola). Ainsi, il est possible de réaliser des opérations sur un mot (ex : incrémentation de la paire).

La logique de contrôle

Appelé encore *Séquenceur* ou *Unité de contrôle* (U.C.)

Son rôle :

Permet à tous les éléments constitutifs du microprocesseur de travailler ensemble et dans l'ordre.

Commentaires :

La logique de contrôle est pilotée par le Registre d'Instruction via le décodeur d'instruction.

Cette unité joue en quelque sorte un rôle d'intendance puisqu'elle décide de la disponibilité du bus à tel ou tel élément logique.

La logique de contrôle possède une architecture complexe et très spécialisée. L'élément central est représenté par le décodeur d'instructions qui décode les informations (premier octet) stockées dans le R.I. pour générer les signaux nécessaires à l'exécution de l'instruction .

La logique de contrôle génère sur les lignes de contrôle des niveaux logiques qui activent les différents circuits environnant tels que mémoires et circuits I/O.

Cette unité fournit, à partir d'un signal de référence qui est l'horloge, tous les signaux de synchronisation utiles au bon fonctionnement de l'ensemble.

Cette horloge est crée à partir d'un oscillateur interne qui utilise un signal en provenance d'un quartz externe.

Deux actions complémentaire à noter :

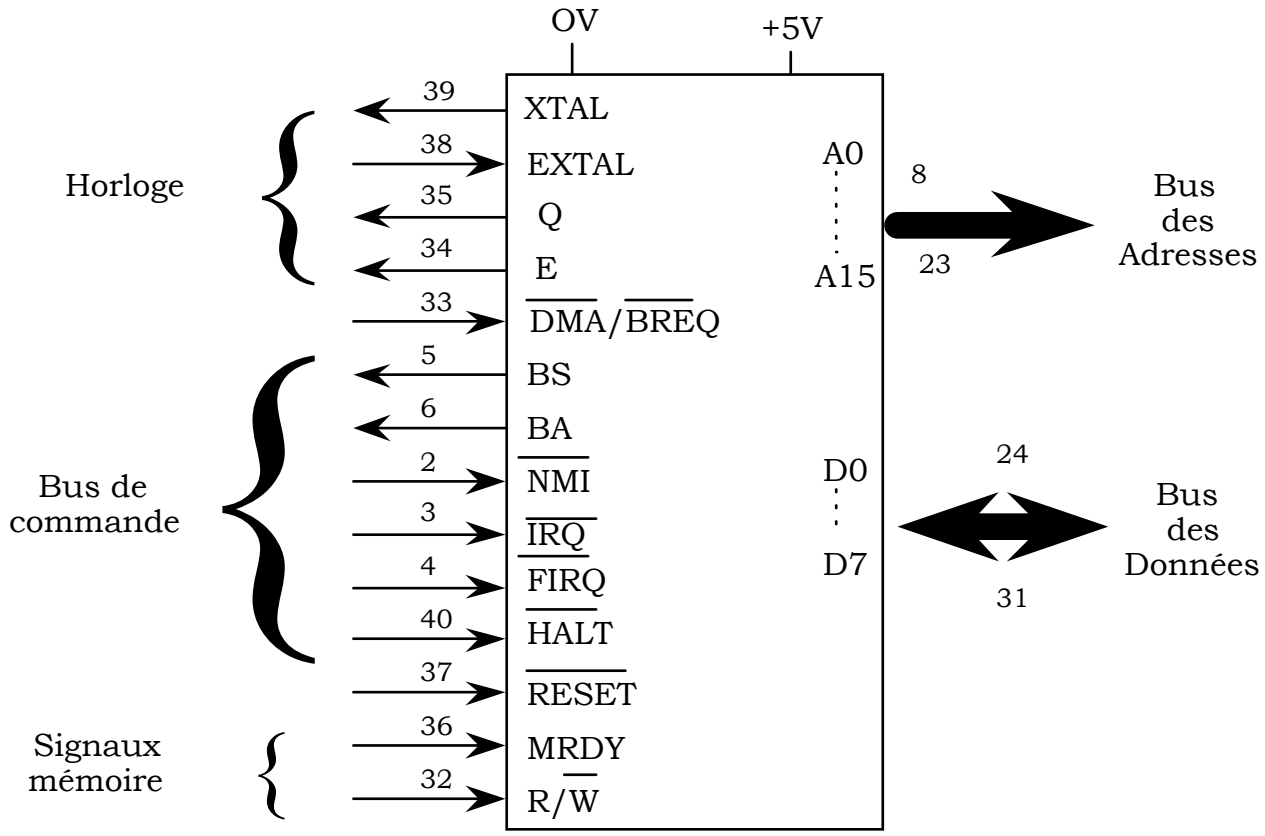
Cette unité assure :

le contrôle de mise sous tension du microprocesseur (initialisation des registres).

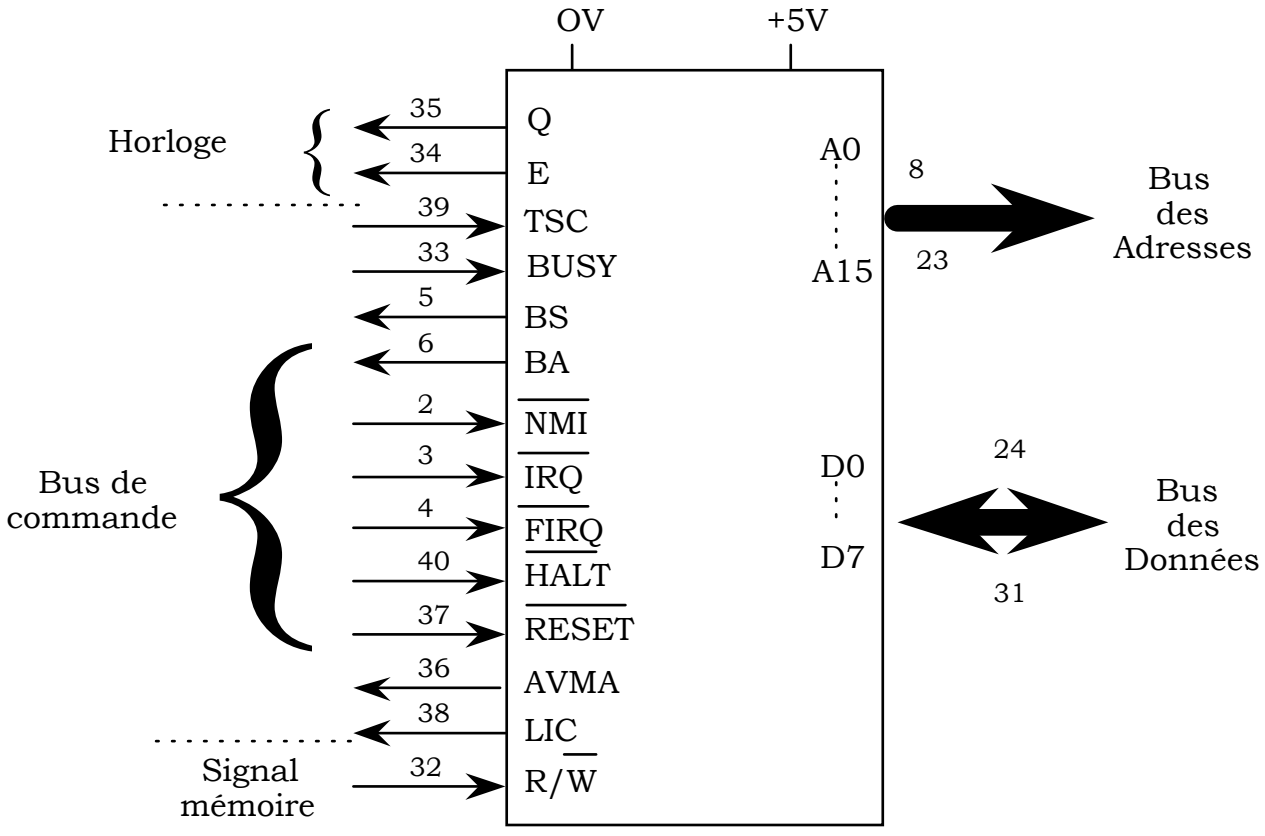
le traitement des interruptions.

Qu'est-ce qu'une interruption ?

Disons pour l'instant, que c'est une requête présentée à la logique de contrôle par des éléments extérieurs (périphériques).



Brochage du MC 6809



Brochage du MC 6809 E

Architecture du 6809

Le microprocesseur 6809 est un processeur 8 bits dont l'organisation interne est orientée 16 bits. Il est fabriqué en technologie MOS canal N et se présente sous la forme d'un boîtier DIL 40 broches. Il est mono-tension(5V).

Il existe deux versions différenciées par l'horloge.

le 6809 est rythmé par une horloge interne (f=1 MHZ, 1.5 MHZ et 2 MHZ).

le 6809E est rythmé par une horloge externe.

Ce dernier est adapté aux applications multiprocesseur. Il présente la particularité de pouvoir être synchronisé par une horloge extérieure. Compatibilité complète entre les 2 versions.

Présentation du brochage

l'alimentation (V_{SS} - V_{CC})

Le bus des données 8 bits (D_0 à D7)

Ces huit broches sont bidirectionnelles. Elles permettent la communication avec le bus des données interne du microprocesseur. Chaque broche peut "piloter" 1 charge TTL et 8 entrées de circuits appartenant à la famille 680 0. Bus en logique 3 états.

Le bus des adresses 16 bits (A0 à A15)

Ces broches unidirectionnelles transfèrent l'adresse 16 bits fournie par le microprocesseur au bus d'adresse du système.
Mêmes caractéristiques électriques que pour le bus des données. Bus en logique 3 états.

⚠ les adresses sont validées sur le front montant de Q.

Le bus de contrôle

La broche **Read/** $\overline{\text{Write}}$

Cette broche indique le sens de transfert des données sur le bus des données. Ligne à logique 3 états

R/$\overline{W}$ = 1 lecture en cours (D0 - D7 sont des entrées)
R/$\overline{W}$ = 0 écriture en cours (D0 - D7 sont des sorties)

⚠ cette ligne est validée sur le front montant de Q.

Les lignes d'état du bus

BA (Bus available) et **BS** (Bus state)

Information qui permet de connaître l'état du microprocesseur à tout moment.

BA	BS	Etat
0	0	normal
0	1	reconnaissance d'interruption
1	0	reconnaissance de synchronisation externe
1	1	arrêt bus disponible

1er cas :

Le microprocesseur est en fonctionnement normal, il gère les bus d'adresses et de données.

2ème cas :

le microprocesseur est en phase de reconnaissance d'interruption pendant deux cycles. Cet état correspond à la recherche des vecteurs d'interruption : Reset, NMI, IRQ, SW1,2 et 3.

3ème cas :

Ce signal apparaît lorsque le microprocesseur rencontre l'instruction de synchronisation externe (niveau bas sur SYNC). Il attend alors cette synchronisation sur une des lignes d'interruption. Les bus sont en haute impédance pendant ce temps.

Dernier cas :

Correspond à l'arrêt du microprocesseur (niveau bas sur HALT).

Le microprocesseur laisse la gestion des bus des données et des adresses à un circuit annexe (contrôleur de DMA). Les bus sont en haute impédance.

La ligne BA au niveau haut indique que les bus sont en haute impédance.

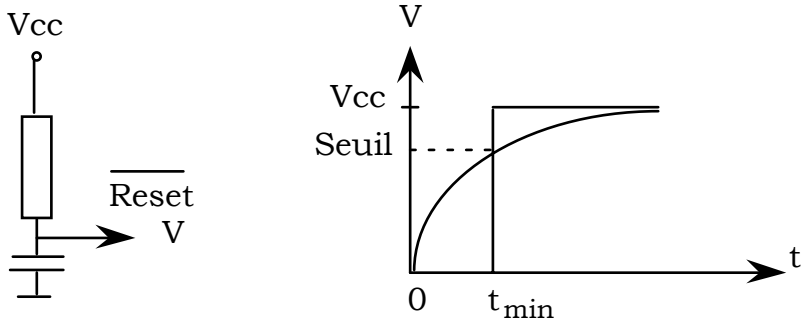
Broche d'initialisation **RESET**

Un niveau bas sur cette broche entraîne une réinitialisation complète du circuit.
Conséquences :

l'instruction en cours est arrêté
le registre de pagination (DP) est mis à zéro
les interruptions IRQ et FIRQ sont masquées
l'interrruption non masquable NMI est désarmée

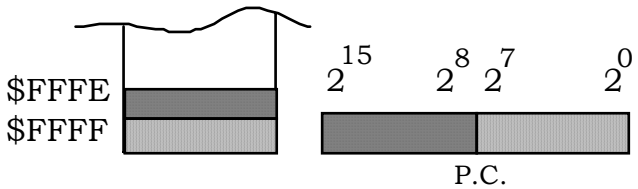
Pour être active cette ligne doit être maintenue à un niveau bas durant un temps suffisamment long (plusieurs cycles d'horloge).

schéma adopté généralement



Le P.C. est initialisé avec le contenu des vecteurs d'initialisation qui se trouvent aux adresses $FFFE et $FFFF.

Ce contenu représente l'adresse du début du programme qui sera exécuté par le microprocesseur.



la broche : **HALT** (Arrêt du microprocesseur).

Un niveau bas sur cette broche provoque l'arrêt du microprocesseur (mais à la fin de l'exécution de l'instruction en cours). Il n'y a pas perte des données. (BA = BS = 1)

Dans ce cas :

les demandes d'interruption IRQ et FIRQ sont inhibées

les demandes d'accès direct (DMA) à la mémoire sont autorisée.

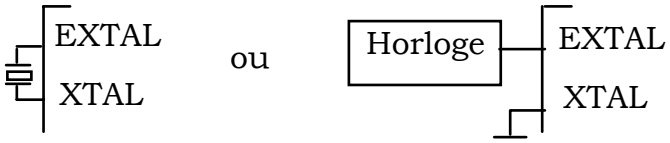
les demandes d'interruptions RESET et NMI sont prises en compte mais leur traitement est différé.

les broches d'interruption

NMI (No Masquable Interrupt)
IRQ (Interrupt Request)
FIRQ (Fast Interrupt Request)

Entrées (actives sur un niveau bas) qui peuvent interrompre le fonctionnement normal du microprocesseur sur front descendant de Q.

Entrées d'horloge **XTAL et EXTAL** (Extension crystal)



La fréquence du quartz (horloge) est quatre fois la fréquence du microprocesseur.

E_{out} représente le signal d'horloge commun au système. Il permet la synchronisation du microprocesseur avec la périphérie.

Q_{out} représente le signal d'horloge en quadrature avec E_{out}.

⚠

Les données sont lues ou écrites sur le front descendant de E_{out}.
Les adresses sont correctes à partir du front montant de Q_{out}.

Broches complémentaires du bus de contrôle :

MRDY (Memory ready) :

Cette broche de commande permet d'allonger la durée de E_{out} pour utiliser des mémoires à temps d'accès long. Active sur un niveau bas.
l'allongement est un multiple de un quart de cycle et sa valeur maximale est de 10 cycles.

$\overline{\text{DMA}}/\overline{\text{BREQ}}$ (Direct Memory Acces/Bus Request).

Cette broche permet de suspendre l'utilisation des bus du microprocesseur, pour faire de l'accès direct ou du rafraîchissement mémoire.

fonctionnement :
Pendant que Q est au niveau haut (si DMA/BREQ bas)
cela entraîne l'arrêt du microprocesseur à la fin du cycle en cours ... et non de l'instruction.

(BA = BS = 1 ce qui veut dire que tous les bus sont en haute impédance).
le circuit ayant généré cette commande dispose de 15 cycles machines avant que le microprocesseur ne reprenne le contrôle des bus.

Broches spécifiques au 6809 E

Entrées d'horloge : **EIN** et **QIN**

Ce sont deux broches dans lesquelles on applique des signaux identiques à Eout et Qout du 6809. Ces signaux doivent aussi être fournis à l'ensemble du système (signaux de synchronisation).

TSC (Tree States Control).

Cette broche a le même rôle que l'entrée DMA/BREQ précédente.
Possibilité de faire du DMA afin de réaliser des opérations de :

rafraichissement
partage de bus avec un autre microprocesseur

LIC (Last Instruction Cycle).

Cette broche de sortie est à l'état haut pendant le dernier cycle de chacune des instructions exécutées par le microprocesseur. Le cycle qui suit ce signal est donc toujours un cycle de recherche de code opératoire d'une instruction.

AVMA (Advanced Valid Memory Access)
(Contrôle des ressources communes en multiprocesseur)

C'est une broche de sortie qui signale un prochain accès au bus par le microprocesseur.
Cette sortie au niveau haut signifie que le microprocesseur utilisera les bus au cours du cycle suivant. La nature prédictive de ce signal permet d'améliorer le fonctionnement d'un système multiprocesseurs à bus partagé.
Elle permet un contrôle efficace des ressources communes d'un dispositif multiprocesseur.

BUSY : Occupation des bus :

Sortie mise au niveau haut pendant les instructions du type : lecture, écriture et exécution du premier octet d'un opérande constitué de deux octets (une adresse par exemple).

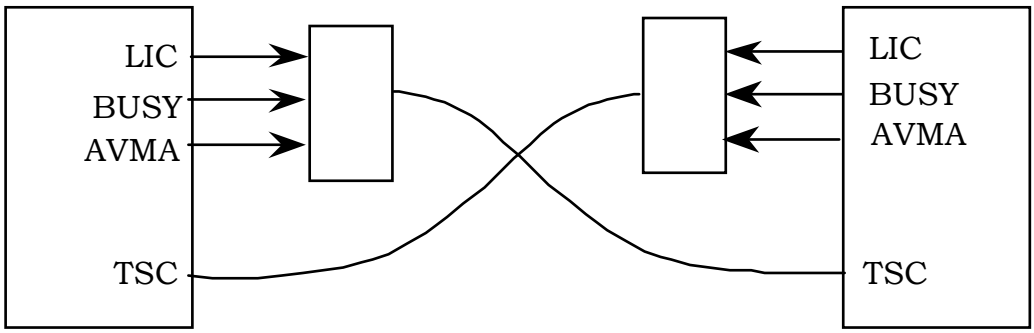
Dans un système multiprocesseur, ce signal indique le besoin pour un microprocesseur de disposer des bus au cours du prochain cycle pour assurer l'intégrité de l'opération en cours.

Cela évite l'adressage simultanée d'une même zone mémoire par 2 microprocesseurs.

⚠

On ne doit pas activer TSC quand BUSY est actif.

Exemple d'application :



LIC : fin d'exécution de l'instruction
BUSY : à besoin du bus au prochein cycle
AVMA : va utiliser les bus au prochain cycle

PRESENTATION DES DIFFERENTS REGISTRES INTERNE AU 6809

Les accumulateurs A, B ou D.

Les registres A et B sont des accumulateurs.

Ces registres sont interchangeables (même rôle/instruction) sauf pour les instructions ABX et DAA et les opérations sur 16 bits.

Certaines instructions regroupent les registres A et B pour former un seul accumulateur D de 16 bits. Dans ce cas l'accumulateur A représente l'octet de poids fort.

Les registres pointeurs

les registres d'index (registres de 16 bits)

Les registres d'index X et Y sont utilisés pour les modes d'adresse indexé.
Les données - 16 bits- contenues dans ces registres servent de pointeur de données (adresses).
Ces adresses "peuvent être modifiées" par une constante, prise comme valeur de déplacement (offset) qui permet alors de calculer une adresse effective.

[le pivot + offset] cela revient à [X] ou [Y] + le déplacement

Le contenu de ces registres peut-être incrémenté ou décrémenté pour gérer des données stockées sous forme de table.

les registres S et U (registres 16 bits).

- **le pointeur de pile S** (Système) est utilisé *automatiquement* par le microprocesseur pour mémoriser l'état de tous ces registres internes dans le cas où il doit exécuter un sous programme (d'interruption ou non).

- **le pointeur de pile U** (Utilisateur) est *géré exclusivement* par le programmeur pour effectuer, avec facilité, le passage des paramètres entre programmes et sous programmes.
(néanmoins, il peut-être utilisé pour sauvegarder un contexte mais cette fois, ce n'est pas automatique !)

Les registre U et S peuvent faire office de pointeurs - registres d'index.

Gestion de ces pointeurs :

Ces registres "pointent" toujours le haut de la zone mémoire qui leur est attribuée. (haut dans le sens adresse la plus grande).
On appelle cette zone une **pile**.

Cette pile fonctionne en mode LIFO (Last In First Out) :

Remarque :
(Le fonctionnement type premier entré - premier sorti s'apparente plus au fonctionnement d'un pipeline appelé pile FIFO (First In First Out)).

Le Compteur de Programme (PC)

C'est le registre (16 bits) qui pointe la zone où se trouve les instructions devant être exécutées.

Le registre de codes condition (CCR)

Ce registre donne à tout instant l'état des indicateurs (ou Flag) du microprocesseur. Il y a deux types d'indicateurs :

Les indicateurs sur la nature des résultats liés aux manipulations des données.

Les indicateurs liés au fonctionnement en interruption.

Présentation des différents indicateurs

$CCRb_0$ (indicateur de retenue baptisé : C)
Indique l'existence d'une retenue lors d'une opération arithmétique effectuée par l'ALU.

$CCRb_1$ (indicateur de débordement baptisé : V)
Il est mis à 1 si le résultat en complément à 2 d'une opération arithmétique déborde.

$CCRb_2$ (indicateur de résultat nul baptisé : Z)
Il est mis à 1 si le résultat de l'opération est nul.
Cet indicateur est affecté par les instructions de chargement, de stockage, des opérations arithmétiques et logiques.

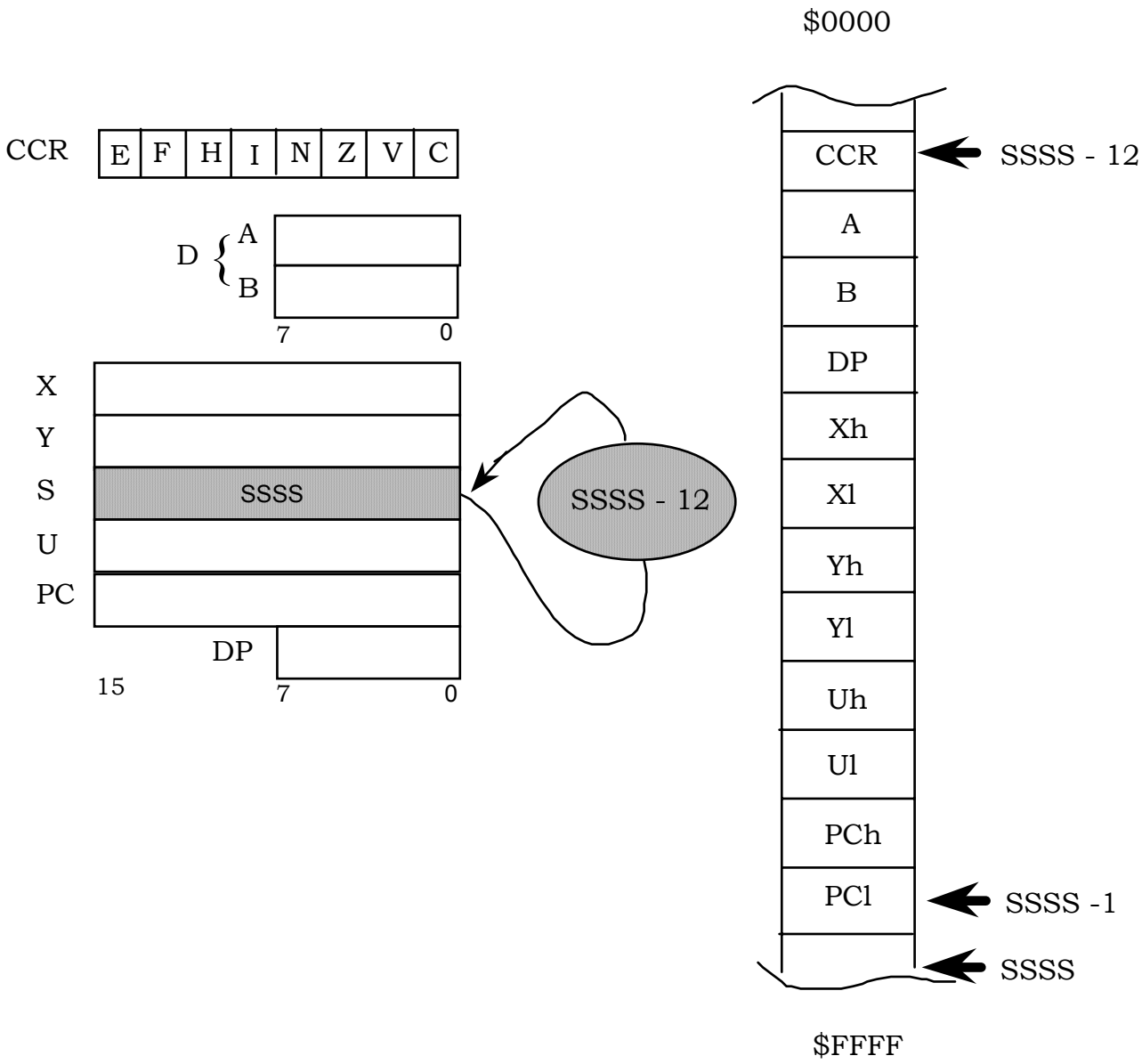
$CCRb_3$ (indicateur de résultat négatif baptisé : N)
Il recopie le bit de poids fort (MSB) de l'octet contenu dans le registre avec lequel on vient de travailler.
(Un résultat négatif en complément à 2 positionne ce flag à 1).

$CCRb_5$ (indicateur de demi-retenue (Half curry) baptisé : H).

Il représente le bit de demi-retenue. Il est utilisé par l'ALU comme indicateur de retenue entre les bits b3 et b4 (retenue du demi-octet le moins significatif) dans le cas d'une addition sur 8 bits.
Ce flag est pris en compte dans l'instruction DAA pour réaliser l'opération d'ajustement décimal.[1]

[1] Les bits $CCRb_4$, $CCRb_6$ et $CCRb_7$ sont utilisés dans des cas très particulier que l'on verra dans le chapitre "Interruption".

Exemple de sauvegarde de tous les registres avec S :

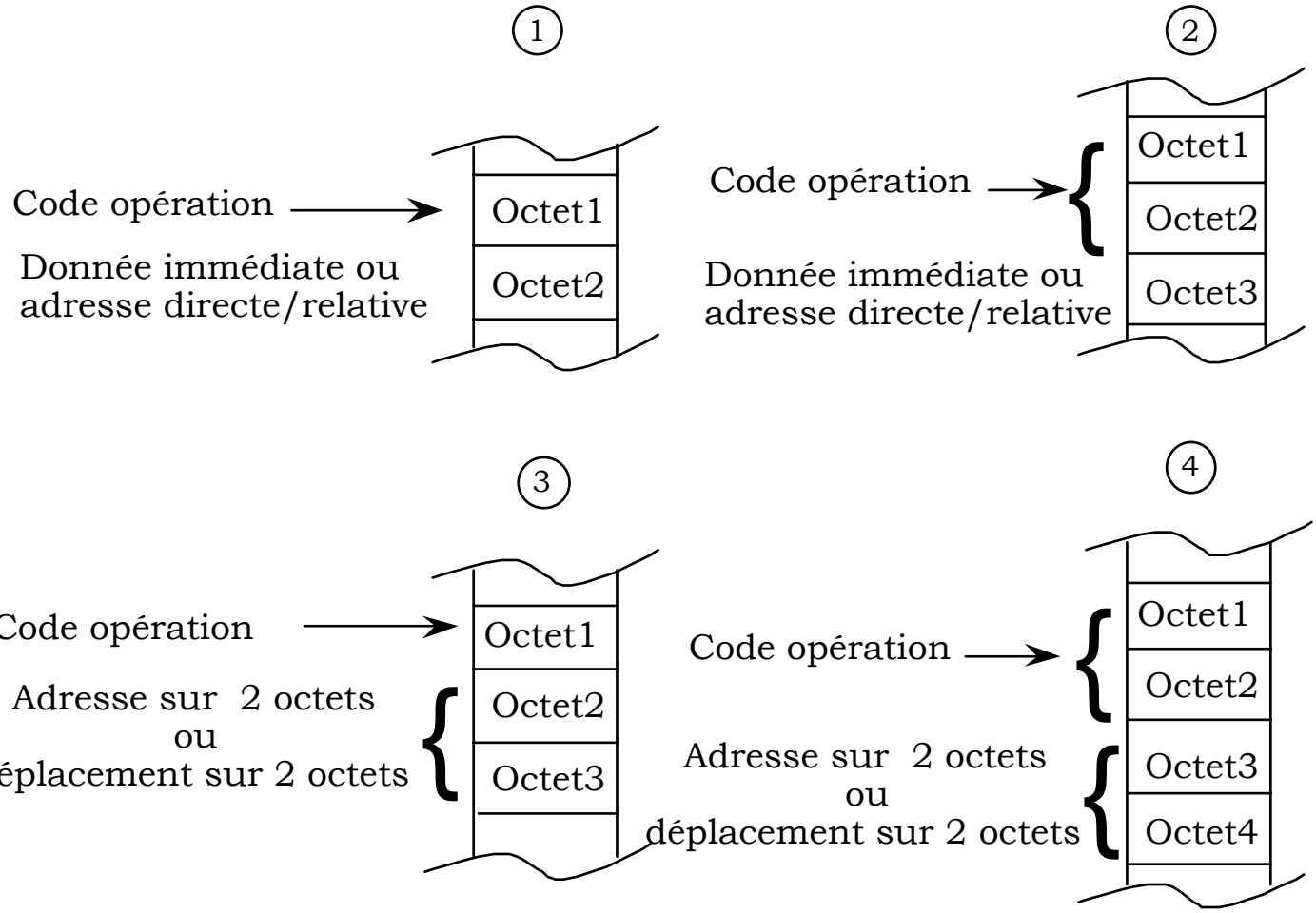


Le jeu d'instructions du 6809

Une instruction peut être simple : 1 octet
ou complexe : 4 octets

La plupart des instructions permettent
un traitement
un déplacement...
des données...
se trouvant...
en mémoire
ou dans un registre.

L'ensemble des instructions de base, compris par le Registre d'Instruction, est constitué de quelques 86 instructions. En tenant compte des variantes (modes d'adressage) il atteint 1464 instructions.



Les divers modes de codage en mémoire des instructions et modes d'adressage.

Ces instructions font référence à des données ou a des adresses de diverses façons, ces références étant les modes d'adressage dont dispose le microprocesseur.

Structure d'une instruction

Octet$_i$	Octet$_{i+1}$	Octet$_{i+2}$	Octet$_{i+3}$
code opératoire	post-octet	code opérande	
ordre	facultatif	adresse	
opérations : arithmétique logique transfert		expression directe ou indirecte (modes d'adressage)	

Durée d'une instruction

L'exécution complète d'une instruction n'est pas instantanée!
L'unité de mesure est la **période** de l'**horloge** : T encore appelé "Cycle Machine".

ex: Fréquence de l'horloge E= 1 MHz donc T = 1 ms.`

La durée dépend de la complexité de l'instruction, son expression est :

$$t_{instruction} = n \,.\, T_{cycle} \quad \text{avec } 2 \leq n \leq 11.$$

Mode de fonctionnement d'une instruction
Exécuter une instruction c'est en faite réaliser le cycle

extraction-exécution

Extraction : lecture de la donnée en mémoire, t=1.T_E

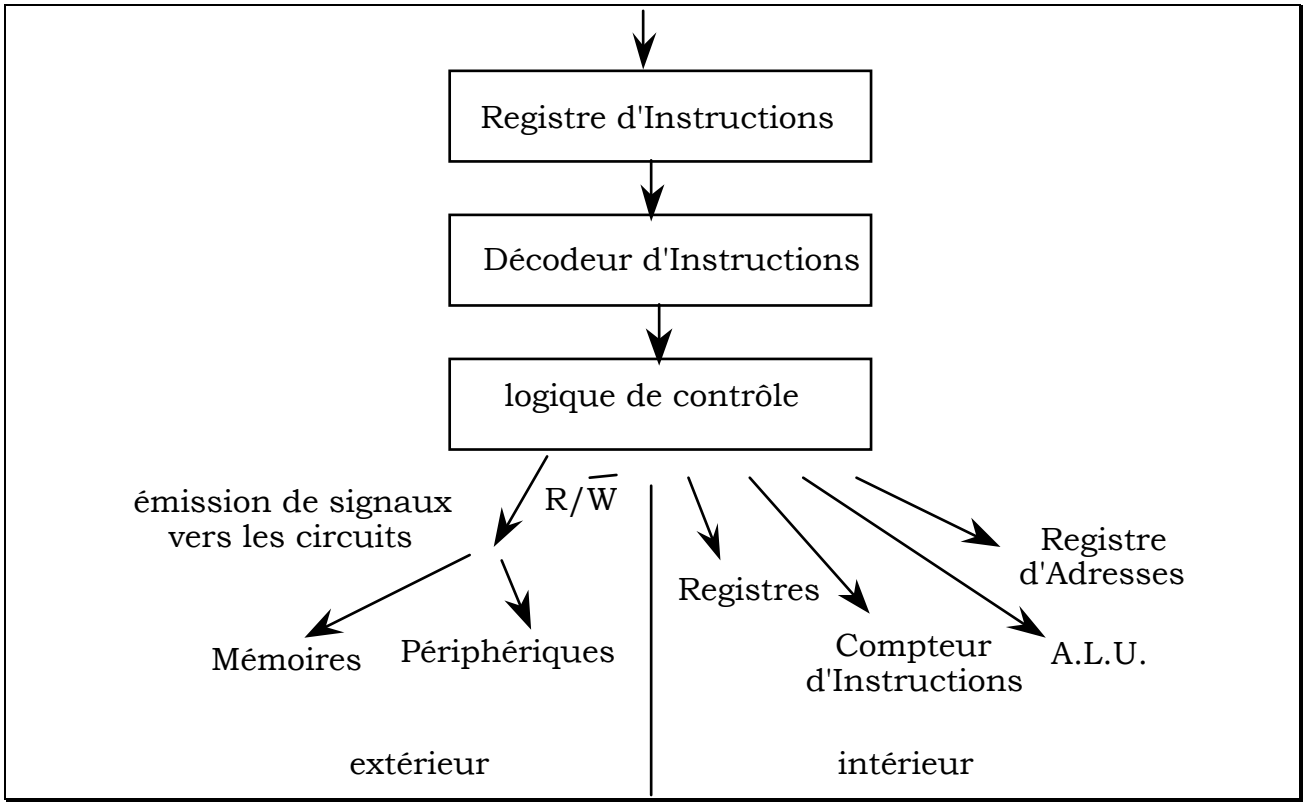
avec T_E période de l'horloge E

Il y a autant d'extractions que d'octets (code opératoire et code opérande) constituant l'instruction.

Si m octets alors t= m.T_E

Exécution : traduction et interprétation de l'octet (code opératoire) une fois extrait. Un octet donc t=1.T_E.

l'interprétation se fait selon le processus suivant :



Une période T_E supplémentaire est nécessaire pour réaliser concrètement l'opération demandée!

<u>Bilan :</u>

La durée totale de l'exécution d'une instruction est :

$m.T_E + 1.T_E + 1.T_E$ soit $(m+2)T_E$.

Le jeu d'Instructions

Les instructions sont réparties en groupes déterminés par leurs caractéristiques.

Classification

Instructions de traitement des données

- Instructions arithmétiques
  - addition (DAA, ABX, ADC)
  - multiplication (MUL)
  - soustraction (SBC, SUB)

Instructions logiques
- rotation à droite et à gauche (ROL, ROR)
- décalage à droite et à gauche (ASR, LSR et ASL, LSL)
- les fonctions logiques de base (AND, OR, EOR)
- l'incrémentation/décrémentation et complémentation (COM - NEG - NOP - INC - DEC - CLR)

Instruction de transfert de données
- transferts internes entre registres (EXG - TFR)
- transferts externes avec la mémoire (LD - ST)

Instructions de tests et de branchements
- instructions de tests sur un bit / un octet (BIT, TST)
- instruction de comparaison (CMP)
- instruction de branchement conditionnel (branchement si les indicateurs du CCR sont actifs

Instructions de branchement inconditionnel et de saut.
(provoquent la rupture de la séquence sans condition)
- saut relatif (BRA, BRN : 1 ou 2 octets)
- saut absolu (JMP : 2 octets)

Instructions d'appel et de retour de sous-programme

Appel : ☞saut absolu (JSR)

☞saut relatif (BSR)

Retour : RTS

le contenu (PC) sauvegardé dans la pile est restitué dans le registre PC. Reprise du programme à l'endroit où il a été interrompu.

Instructions opérant sur les pointeurs U, S et X, Y.

☞Load effective adress (in register) : LEA.

Permet de manipuler des données sur 16 bits. Ces données représentent généralement des adresses.

Instructions opérant sur les pointeurs S et U

☞Emplilement des registre dans la pile (PSH)

☞Dépilement des registres de la pile (PUL)

Après chaque sauvegarde/extraction, le pointeur est automatiquement décrémenté/incrémenté de 1

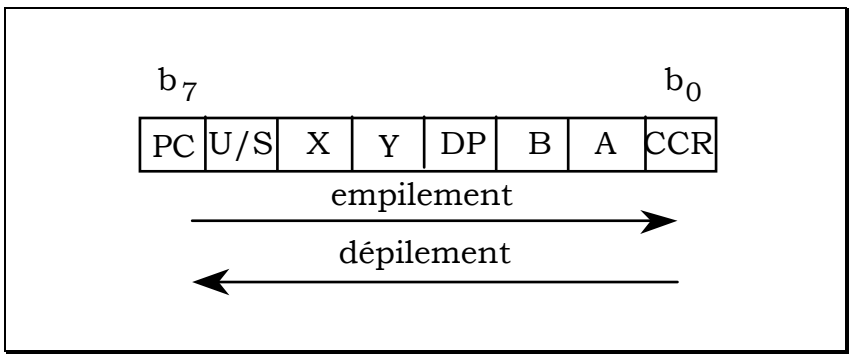
Définition d'une pile :

zone mémoire RAM gérée par des pointeurs qui permettent de transférer rapidement des données dans des cases mémoires selon un protocole bien établi.

⚠

Ordre des actions :

Toujours suivie de l'opérande qui permet de sélectionner les registres :



Les modes d'adressage du 6809

Le microprocesseur 6809 possède 59 instructions de base. Combinées avec le jeu des modes d'adressage (9 au total), elles fournissent 1464 codes opératoires différents.
(Pour le 6800 ou 6802, on avait 72 instructions de base et 193 codes opératoire)

Les modes d'adressage sont :

- l'adressage inhérent ou implicite
- l'adressage immédiat
- l'adressage étendu
- l'adressage étendu indirect
- l'adressage direct
- l'adressage par registre
- l'adressage indexé direct
- l'adressage indexé indirect
- l'adressage relatif

Au moyen des signaux qu'il génère sur le bus d'adresses, le microprocesseur a la possibilité d'adresser les divers circuits mémoires et interfaces, qui lui sont connectés au travers des bus afin d'accéder à leur contenu.

Cette accès se traduit par une opération d'adressage

Cette opération peut se faire de plusieurs façons grâce à la présence de différents modes d'adressage.

Remarque : La puissance d'un microprocesseur dépend de son jeu d'instructions mais aussi des ses modes d'adressage.

L'adressage inhérent ou implicite

L'adressage inhérent est utilisé par les instructions qui agissent seulement sur les registres internes du microprocesseur. Ici, le code opératoire de l'instruction contient toute l'information d'adressage nécessaire (adresse source ou/et adresse destination).

Exemples :

ABX, ASL, ASR, CLR, INC

Adressage par registre

Le code opératoire est immédiatement suivi dans la mémoire d'un octet qui définit un registre ou le jeu de registres devant être utilisés par l'instruction. Cet octet est appelé *post-octet*.

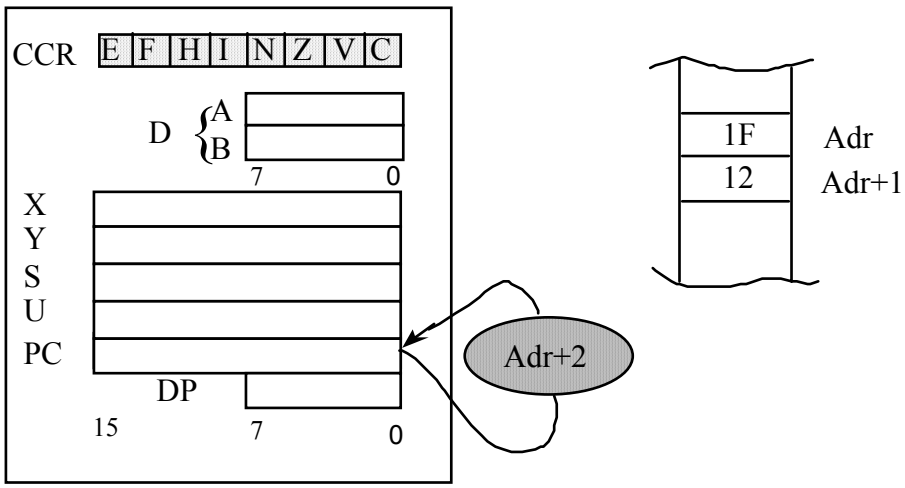
Le tableau ci-dessous présente le codage de ce post-octet :

b7	b3
b4	b0
Source	Destinataire

Post-octet transfert/échange

code	Registre
0000	D
0001	X
0010	Y
0011	U
0100	S
0101	PC
1000	A
1001	B
1010	CCR
1011	DP

Exemple : TFR X,Y (transfert de X dans Y).


CCR
E F H I N Z V C
D
A
B
7
0
X
Y
S
U
PC
DP
15
7
0
Adr+2
1F
12
Adr
Adr+1

Adressage immédiat

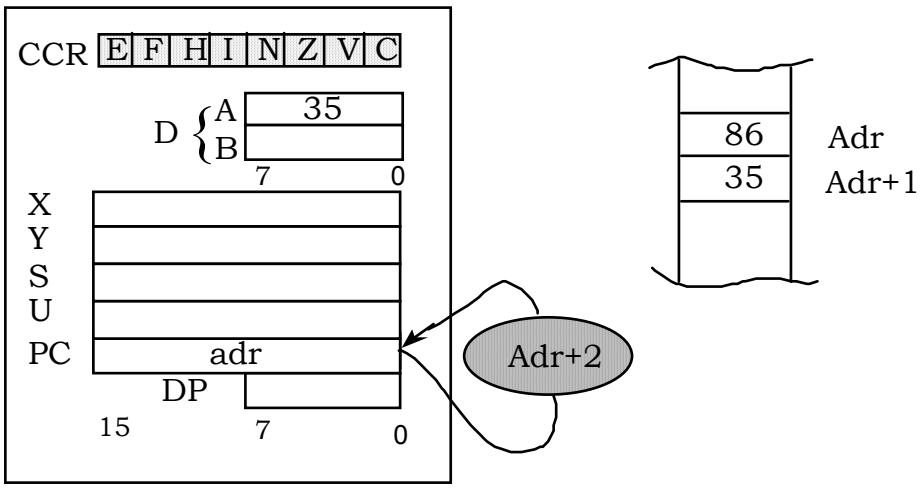
La donnée se trouve immédiatement après le code opératoire de l'instruction.
La donnée existe sous la forme de 1 ou 2 octets.

(le code opératoire est immédiatement suivi en mémoire de la donnée sur laquelle porte l'opération)

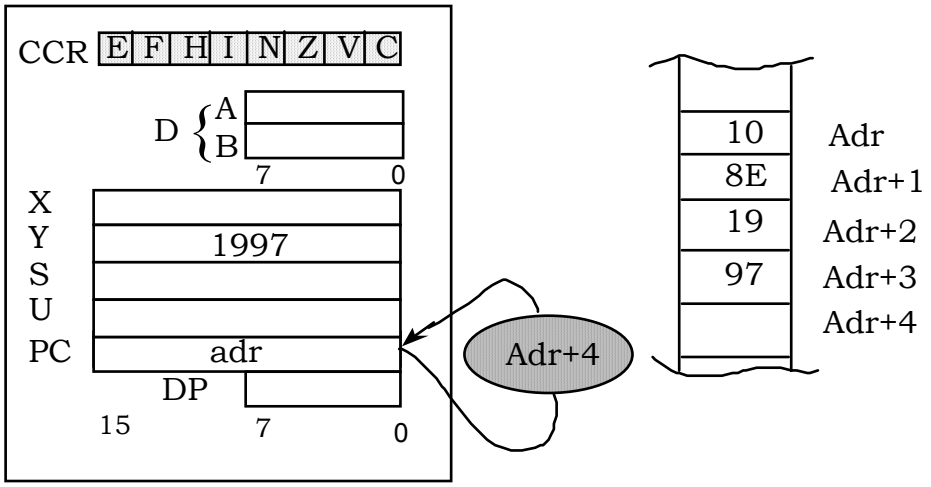
Cette adressage concerne tous les registres internes sauf le DP.

Exemples :

LDA #$35



LDY#$1997



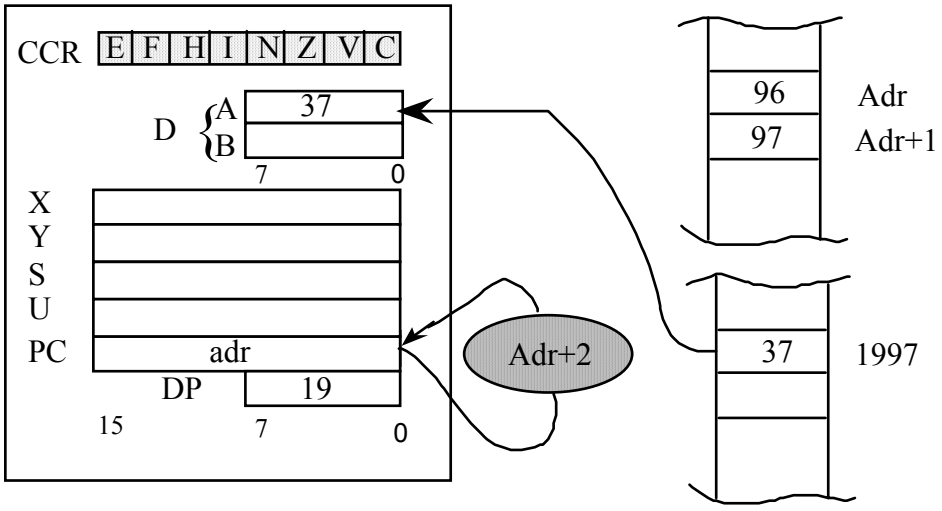
Adressage direct

On exprime le lieu de l'action par l'expression de l'adresse effective.
Le code opérande indique la partie basse de cette adresse. La partie haute de l'adresse est fournie par le contenu du Registre Direct de Page (DP).

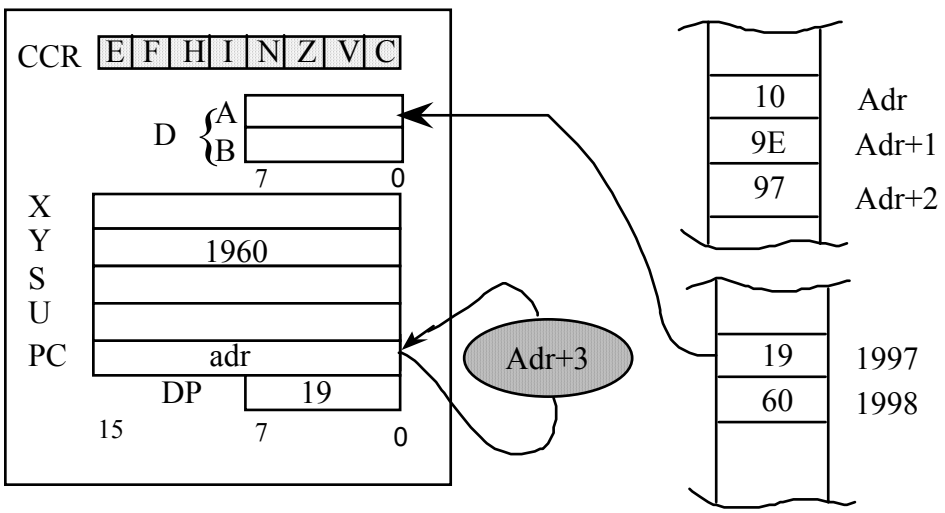
Intêret : Ce mode nécessite moins de place mémoire (1 octet donc taille mémoire réduite) par conséquent l'exécution de l'instruction est plus rapide.

Remarque : Avec ce mode, la mémoire est découpée en 256 pages de 256 octets chacune.
Ce mode est intéressant dans le cadre des systèmes d'exploitation temps réel multitâche, où on alloue à chaque tâche une page.

Exemples :
LDA $97 Charge l'Accumulateur A avec le contenu dont l'adresse est formée par [DP] et l'opérande



LDY $97 Charge le registre Y avec le contenu sur 16 bits dont les adresses sont [DP] et partie basse et partie basse+1.


CCR
E F H I N Z V C
D
A
B
7
0
X
Y
1960
S
U
PC
adr
DP
19
15
7
0
Adr+3
10
9E
97
Adr
Adr+1
Adr+2
19
60
1997
1998

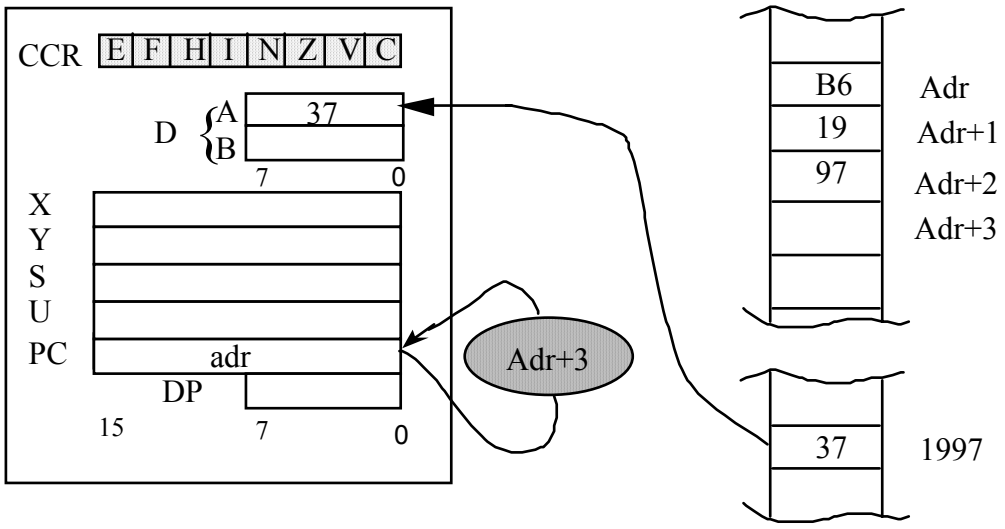
Adressage étendu(direct)

On exprime le lieu de l'action par l'expression de l'adresse effective.
Le contenu des deux octets qui suivent immédiatement le code opératoire spécifié représente l'adresse (16bits) de la donnée.
L'instruction occupe de 3 à 4 octets.

exemples :

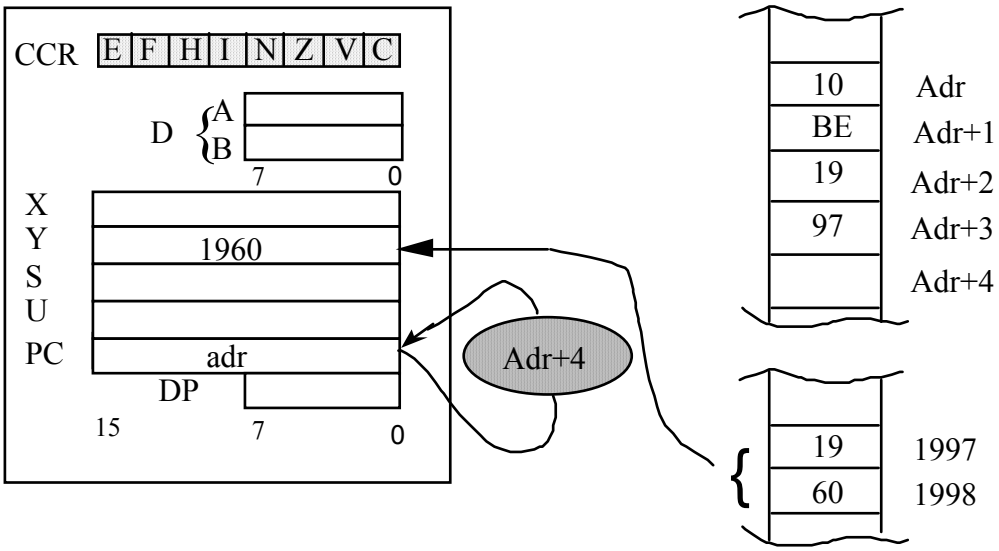
LDA $1997

Charge l'accumulateur A avec le contenu de l'adresse 1997.



LDY $1997

Charge le registre Y avec le contenu de l'adresse 1995.


CCR
E F H I N Z V C
D
A
B
7
0
X
Y
1960
S
U
PC
adr
DP
15
7
0
Adr+4
10
BE
19
97
Adr
Adr+1
Adr+2
Adr+3
Adr+4
19
60
1997
1998

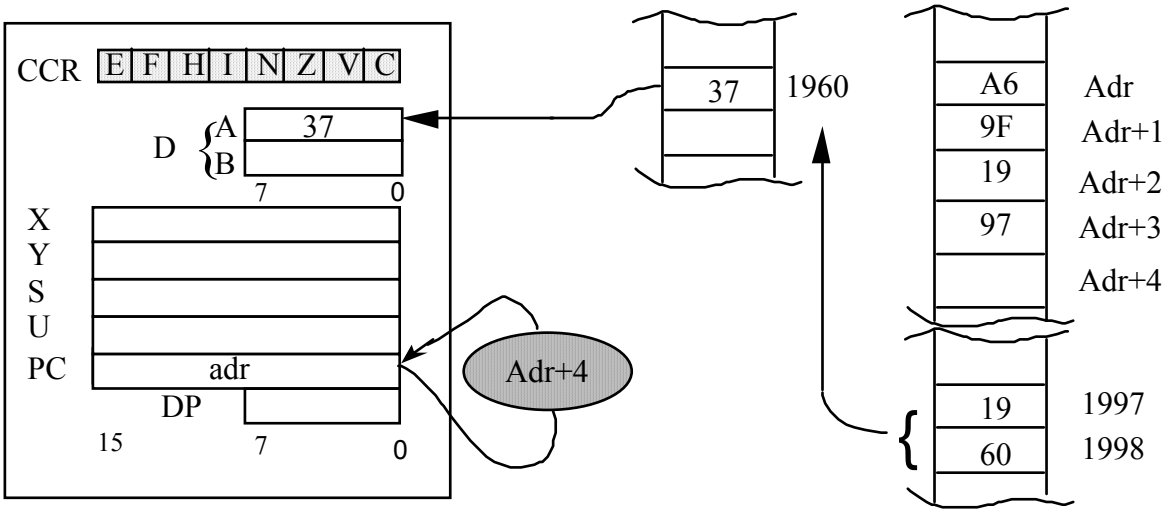
Adressage étendu indirect

Identique au mode d'adressage étendu mais on accède à la donnée en passant par une adresse intermédiaire spécifiée après le code opératoire.

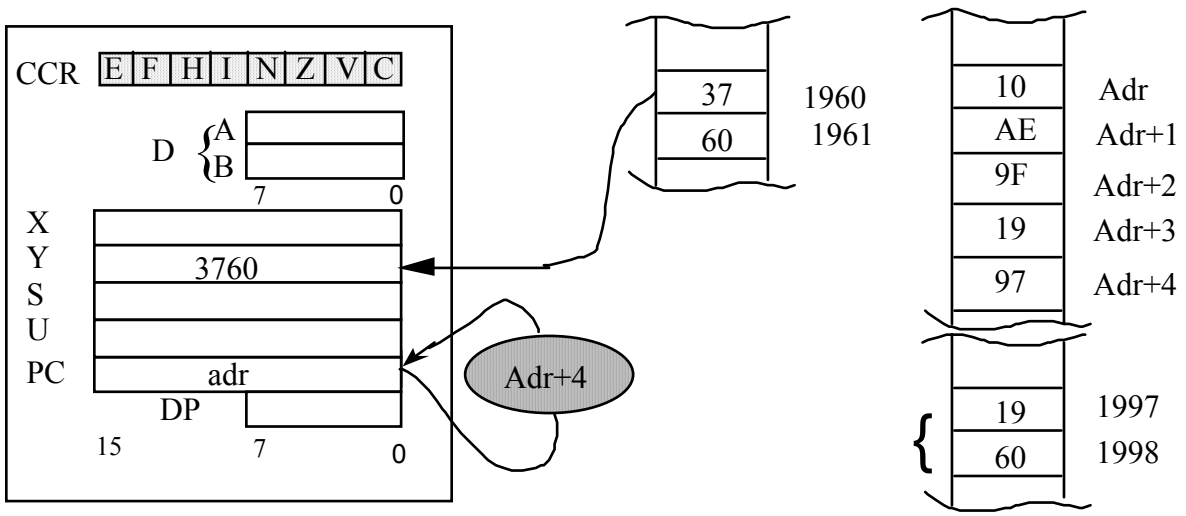
Les deux octets qui suivent le code opératoire pointent une adresse dont le contenu représente l'adresse de la donnée recherchée.
Le code opératoire est formé de deux octets - le post-octet est toujours $9F.

Exemples :

LDA [$1997] Chargement de l'accumulateur A avec le contenu dont l'adresse se trouve en 1997-1998.



LDY [$1997] Chargement du registre Y avec le contenu dont l'adresse (partie haute) se trouve en $1997.



Adressage indexé

Deux possibilités :

adressage indexé direct et indirect

Dans ce mode, les registres pointeurs (X, Y, U, S et PC) sont utilisés pour effectuer le calcul de l'adresse effective de la donnée recherchée.

Il existe 5 types d'adressage indexé.

l'adressage indexé avec déplacement nul
l'adressage indexé avec déplacement constant (non nul).
l'adressage indexé avec déplacement accumulateur
l'adressage indexé avec auto-incrémentation/décrémentation
l'adressage indexé relatif au Compteur Programme (PC)

L'octet qui suit le code opératoire (le post-octet) spécifie :
la nature de l'indexation
le type d'adressage (direct ou indirect)
le registre pointeur utilisé.

(le tableau ci-joint montre le format de ce post-octet.)

Constitution du post-octet.

b7=1

b4 : indicateur d'indirection où non
b4 = 1 : mode indirect
b4 = 0 : mode direct

b7=0

b4 représente le signe

b5 et b6 représente le registre concerné

b0,b1,b2 et b3 indiquent le mode d'adressage.

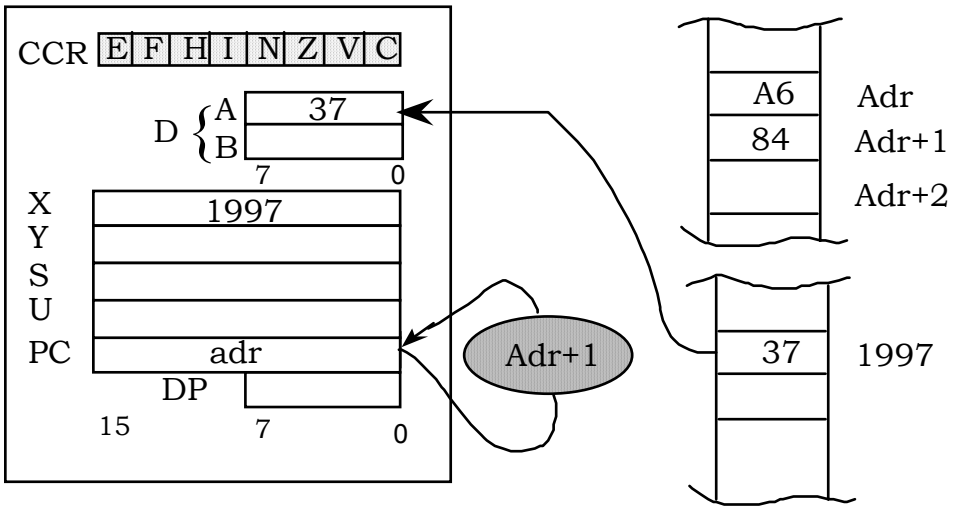
Remarquer l'octet $9F déjà vu dans le mode d'adressage indirect.

Adressage indexé à dépacement nul

Le registre pointeur contient l'adresse effectivee (A.E.) de la donnée.

Un chargement préalable du registre est impératif.

Exemple : LDA O,X ou LDA ,X



Légende de la valeur $84 soit 1 00 0 0100 du post-octet (ligne numéro 6 du tableau).

Adressage indexé à déplacement constant

L'adresse effective de la donnée est la somme du déplacement (constante en complément à deux) et du contenu du registre nommé pris comme base.

A.E. = [Registre nommé] + expression du déplacement.

Le contenu du registre n'est pas modifié.

Syntaxe :

Un code opératoire + un post-octet + un opérande.

Trois formes sont possibles selon l'expression de la valeur de la constante qui suit le post-octet.

Le déplacement s'exprime sur 4 bits + un bit de signe.
Le déplacement est compris dans l'intervalle [-16_{10} à15_{10}]
Le post-octet suffis.

Exemple : LDA -2,X Code A6
1E soit 0 00 1 1110

Le déplacement s'exprime sur 7 bits + un bit de signe.
Le déplacement est compris dans l'intervalle [-128_{10} à127_{10}]
Un octet supplémentaire après le post-octet est nécessaire.

Exemple : LDA 53,X Code A6
88 soit 1 00 0 1000
35 (53_{16})

Le déplacement s'exprime sur 15 bits + un bit de signe.

Le déplacement est compris dans l'intervalle [-32768$_{10}$ à+32767$_{10}$]
Deux octets supplémentaires après le post-octet sont nécessaires.

Exemple : LDA $997,X	Code	A6
		89 soit 1 00 0 1001
		09
		97

Adressage indexé avec déplacement accumulateur.

Ce mode est semblable au mode précédent excepté que la valeur du déplacement (exprimé en complément à 2) se trouve dans un accumulateur afin d'être ajouté au contenu du registre pointeur nommé pour former l'A.E. de la donnée.

Les contenus de l'accumulateur et du pointeur ne sont pas modifiés par cette addition.

C'est le post-octet qui spécifie l'accumulateur utilisé (pas d'octet suppémentaire).

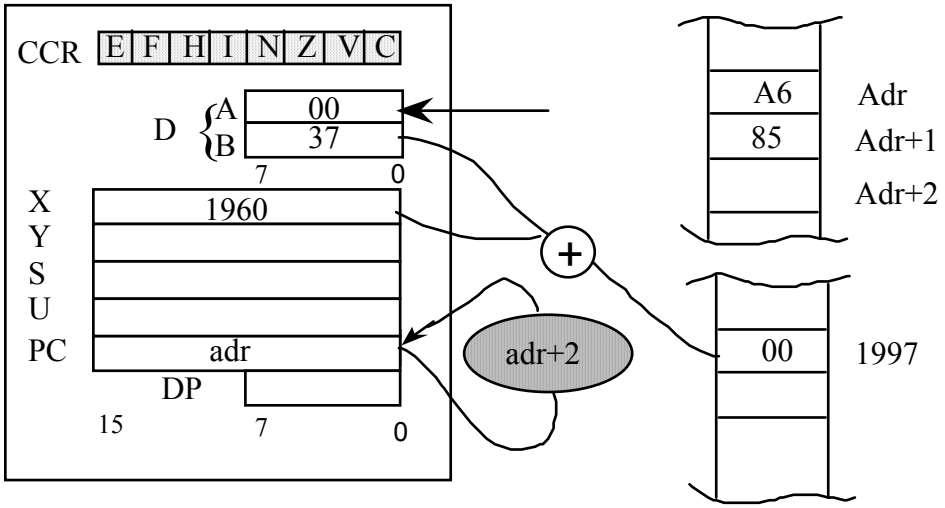
Intêret

la valeur du déplacement est calculée par le programme en cours d'exécution, en fonction des événements.

Exemples :

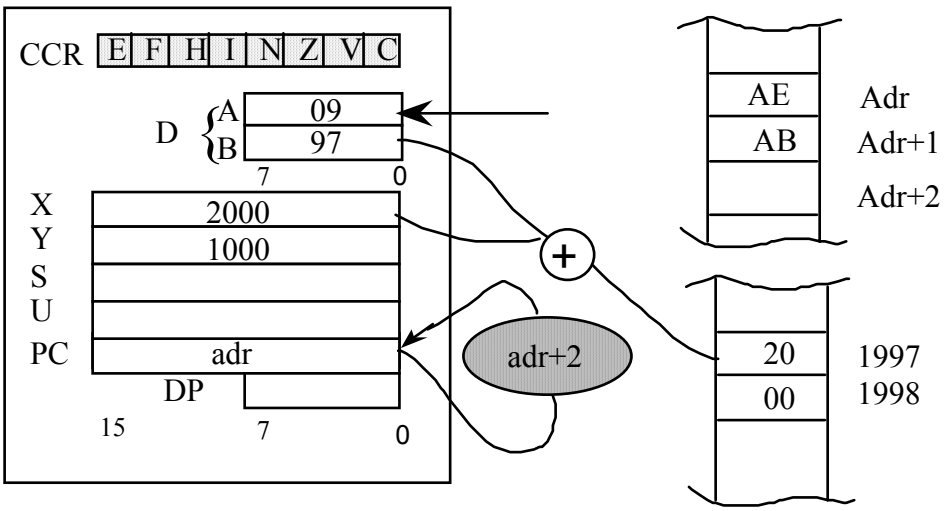
LDA B,X avec post-octet = $85 soit 1 00 0 0101.

Chargement du registre A avec le contenu se trouvant à l'adresse exprimée par la somme des contenus des registres B et X.



LDX D,Y avec post-octet = $AB soit 1 01 0 1011.

Charge le registre X avec le contenu se trouvant à l'adresse exprimée par la somme des contenus des registres D et X ainsi que D+1 et X.



Adressage indexé
avec auto-incrémentation/décrémentation

Le registre concerné contient l'adresse de la donnée.

En plus de l'utilisation en mode indexé avec déplacement nul, il est possible de modifier le contenu du registre selon le mécanisme suivant :

La pré-décrémentation

Le pointeur est décrementé de un ou deux avant utilisation.

La post-incrémentation

Le pointeur est incrémenté de un ou de deux après utilisation.

Dans tous les cas, le contenu du registre est modifié.

Intêret :

Une incrémentation/décrémentation permet de gérer des tables de données 8 bits (octets).

Deux incrémentations/décrémentations permettent de gérer des tables de données de 16 bits (mots ou adresses)

Les aspects pré-décrementation et post-incrémentation sont très utiles pour créer des piles logicielles supplémentaires dont le comportement est identique à celui des piles S et U (hardware)

(gestion efficace de blocs-mémoire organisés en données 8 bits ou 16 bits)

Exemples :

LDA 0,-X Code A6
82 soit 1 00 0 0010

Chargement de l'accumulateur A avec le contenu se trouvant à l'adresse exprimée par le contenu de X décrémenté de 1.

LDY 0,--X Code 10
AE
83 soit 1 00 0 0011

Chargement du registre Y avec le contenu se trouvant à l'adresse exprimée par le contenu de X décrémenté deux fois.

LDA 0,X+ Code A6
80 soit 1 00 0 0000

Chargement de l'accumulateur A avec le contenu se trouvant à l'adresse exprimée par le contenu de X. Après le chargement, le contenu de X est incrémenté de 1.

LDD 0,X++ Code EC
81 soit 1 00 0 0001

Chargement de l'accumulateur D avec le contenu se trouvant à l'adresse exprimée par le contenu de X. Après le chargement, le contenu de X est incrémenté deux fois.

Adressage indexé
relatif au compteur-programme (PC)

Un programme qui contient des instructions exprimées dans le mode étendu ou direct, faisant référence à des adresses situées à l'intérieur de sa zone d'implantation en mémoire ne pourra pas être exécuté si l'on transfère son code objet dans une autre zone mémoire.
On dit, dans ce cas, que le programme n'est pas translatable.

Cet inconvénient peut-être évité en utilisant le mode d'adressage indexé avec dépacement relatif au PC.

Syntaxe :

code opératoire destination-(adr+4), PC

l'expression de "destination-(adr+4)" est codée en complément à 2.

Exemples :

Chargement du registre A avec le contenu se trouvant à l'adresse $1997.

Code	$1000	adr	A6	
		adr+1	8D	soit 1 00 0 1101
		adr+2	09	
		adr+3	93	

Justification :

Si adr=$1000 alors destination ($1997) est égale à $1997-$1004 soit $993.
Il faut tenir compte de la longueur de l'instruction puisque le PC pointe toujours l'instruction suivante.

LDA $FO,PC

chargement de l'accumulateur A avec le contenu mémoire dont l'adresse est la valeur de PC + $FO.

Code	adr	A6	
	adr+1	8C	soit 1000 1100
	adr+2	F0	

La donnée se trouve à l'adresse (adr+3)-$10 donc en amont du programme.
Avantage du mode indexé relatif au PC :

Si l'ensemble du programme est chargé à une autre adresse, le même code objet permet toujours de retrouver la donnée dans le corps du programme et d'obtenir ainsi une exécution correcte.
Le même traitement appliqué à l'ensemble des instructions du programme, le rendra entièrement translable et totalement indépendant de son implantation physique dans la mémoire. Les adresses des données étant toutes referencées par rapport à la valeur réelle du PC au moment de l'exécution.

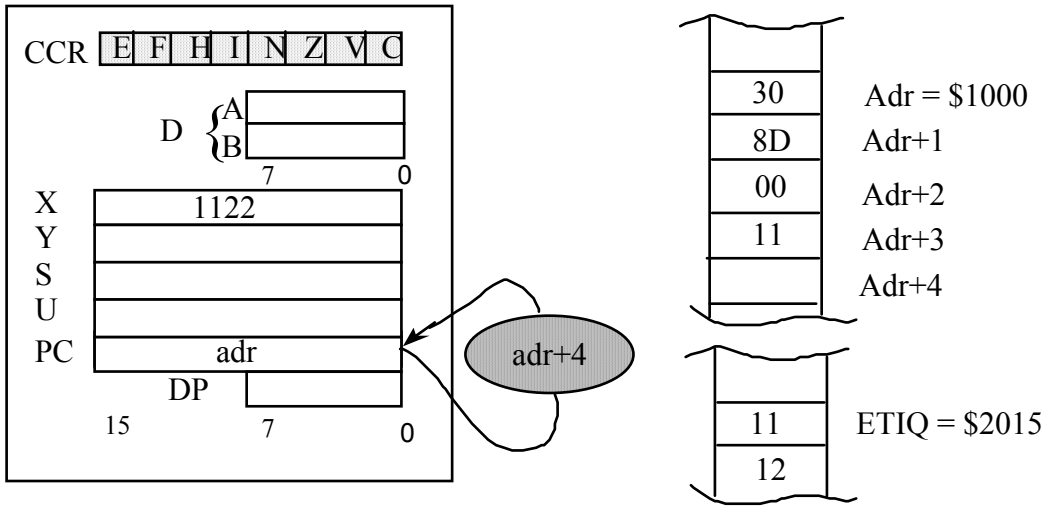
L'utilisation d'une étiquette facilite l'emploi de ce mode.

Illustration :

LEAX ETIQ, PC

Chargement du registre X avec le contenu dont l'adresse est exprimée par la valeur issue de l'opération ETIQ-(adr+4).

Code du post-octet : $8D soit 1 00 0 1101



Déplacement = $2015 -($1000+4) soit 11

Adressage indexé indirect.

Dans ce mode, l'Adresse Effective est contenue à l'emplacement indiqué par le contenu du registre d'index utilisé auquel on additionne le déplacement.

Ici, l'A.E. transite par une adresse intermédiaire.

Syntaxe :

A.E(H)={[Registre] + expression du déplacement}.
A.E(L)={[Registre] + expression du déplacement+1}.

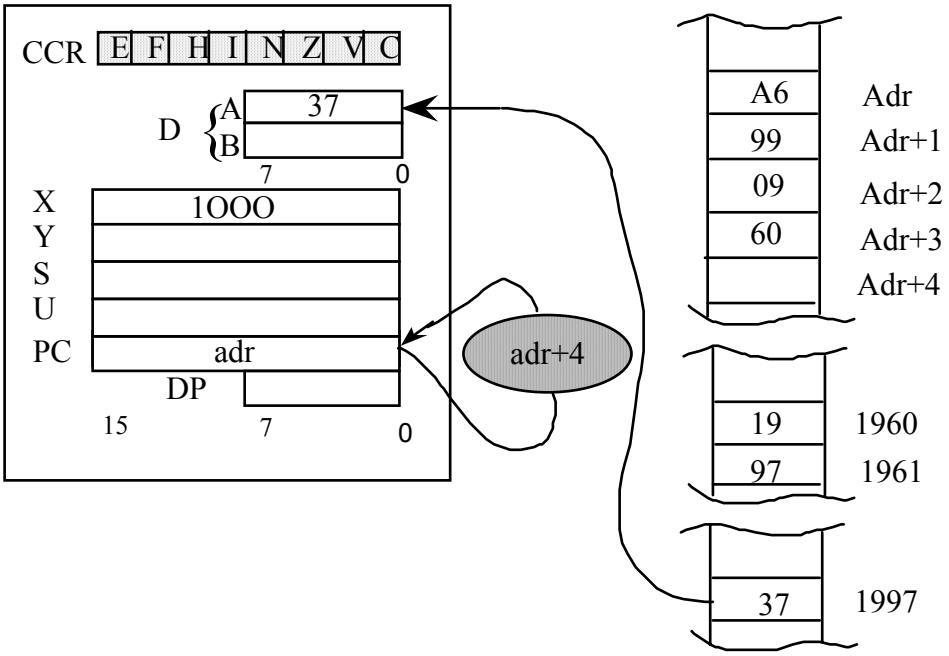
Tous les modes d'adressage indexé peuvent travailler en indirection sauf les modes :

auto-incrémentation/décrémentation par 1
déplacement constant codé sur 5 bits

Illustration :

LDA [$960,X]

Chargement de A avec le contenu mémoire dont l'adresse est le contenu de [X+960, X +961].



Quelques exemples :

1- Adressage indexé indirect avec déplacement nul.

LDA [0,X]

Chargement de A avec le contenu mémoire dont l'adresse est [X, X +1].

2- Adressage indexé indirect avec déplacement constant.

LDA [$35,X]

Chargement de A avec le contenu mémoire dont l'adresse est [X+$35, X +$36].

3- Adressage indexé indirect avec déplacement accumulateur.

Le déplacement exprimé en complément à deux est contenu dans A, B ou D.

LDU [D, Y]

Chargement du registre U avec le contenu mémoire dont l'adresse est [Y+D] et [Y+D+1]

4- Adressage indexé indirect avec double auto-incrémentation/décrémentation.

ADDA [, U++]

Addition du contenu de A et du contenu-mémoire dont l'adresse est [U] et [U+1], suivi de deux incrémentations.

5- Adressage indexé indirect relatif au compteur programme.

Les déplacements sont codés sur 8 ou 16 bits en complément à 2.

LDA [$1997, PC]

Chargement de A avec la valeur dont l'adresse est la valeur issue de l'opération PC + $1997.

Adressage relatif

Technique d'adressage utilisée avec les instructions de branchement conditionnel ou non.

Constitution

	code opératoire	code opérande
adr. relatif court	1 octet	1 octet
adr. relatif long	1 octet + $10	2 octets

Le code opératoire représente le test et le type de branchement (court ou long : $10) à réaliser.

Le code opérande représente le déplacement signé qui est ajouté au contenu du PC

Dans le mode d'adressage relatif court, le déplacement est exprimé sur un octet en complément à deux

$-128 \leq$ déplacement $\leq +127$

Dans le mode d'adressage relatif long, le déplacement est exprimé sur deux octets en complément à deux

$-32768 \leq$ déplacement $\leq +32767$

Réalisation de l'Adresse Effective :`

$$\text{Adresse Effective} = PC + 2(4) + \text{déplacement}$$

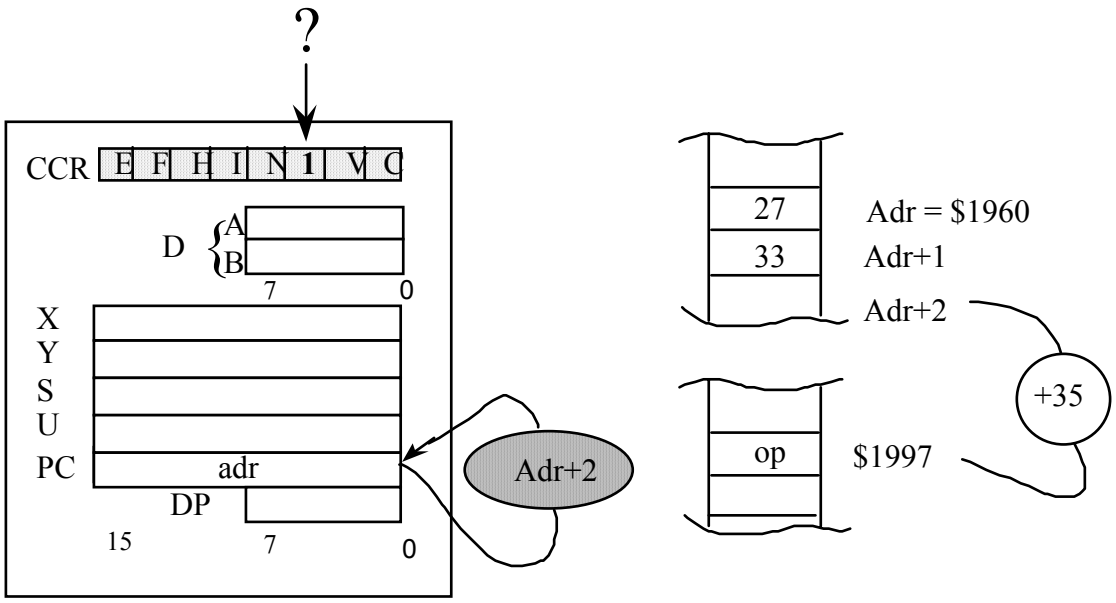
$$(PC+2) - \text{valeur} \leq \text{déplacement} \leq (PC+2) + \text{valeur}$$

Dans la syntaxe assembleur il faut rajouter un (L) comme préfixe dans le cas d'un saut long.

Quelques exemples :

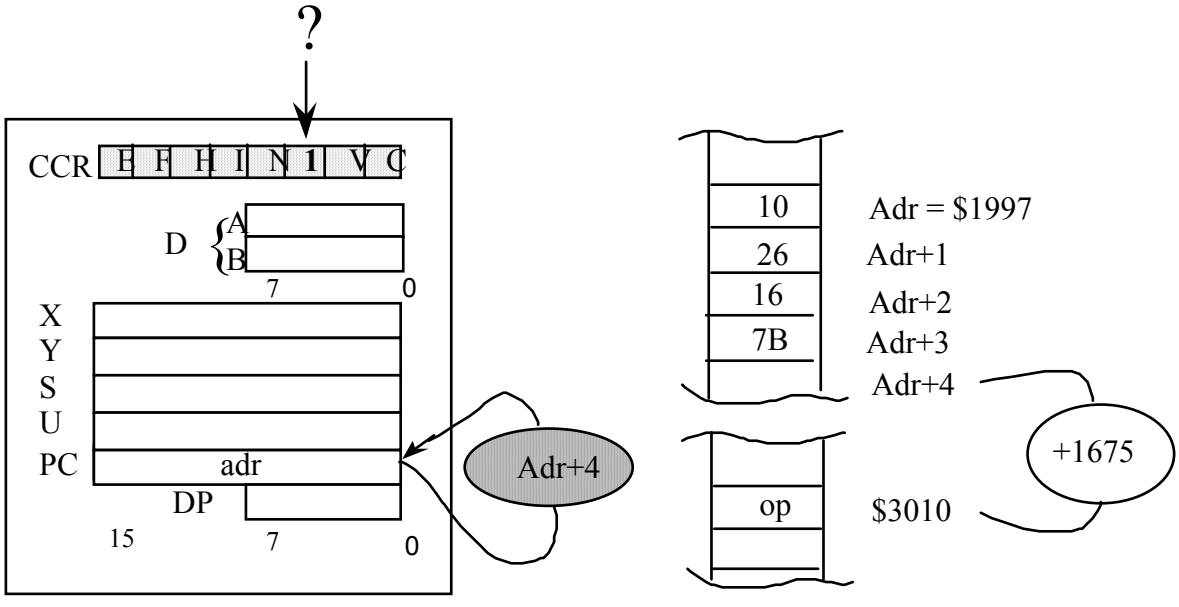
Adressage relatif court avec un déplacement positif.

BEQ LOOP branchement si Z=1



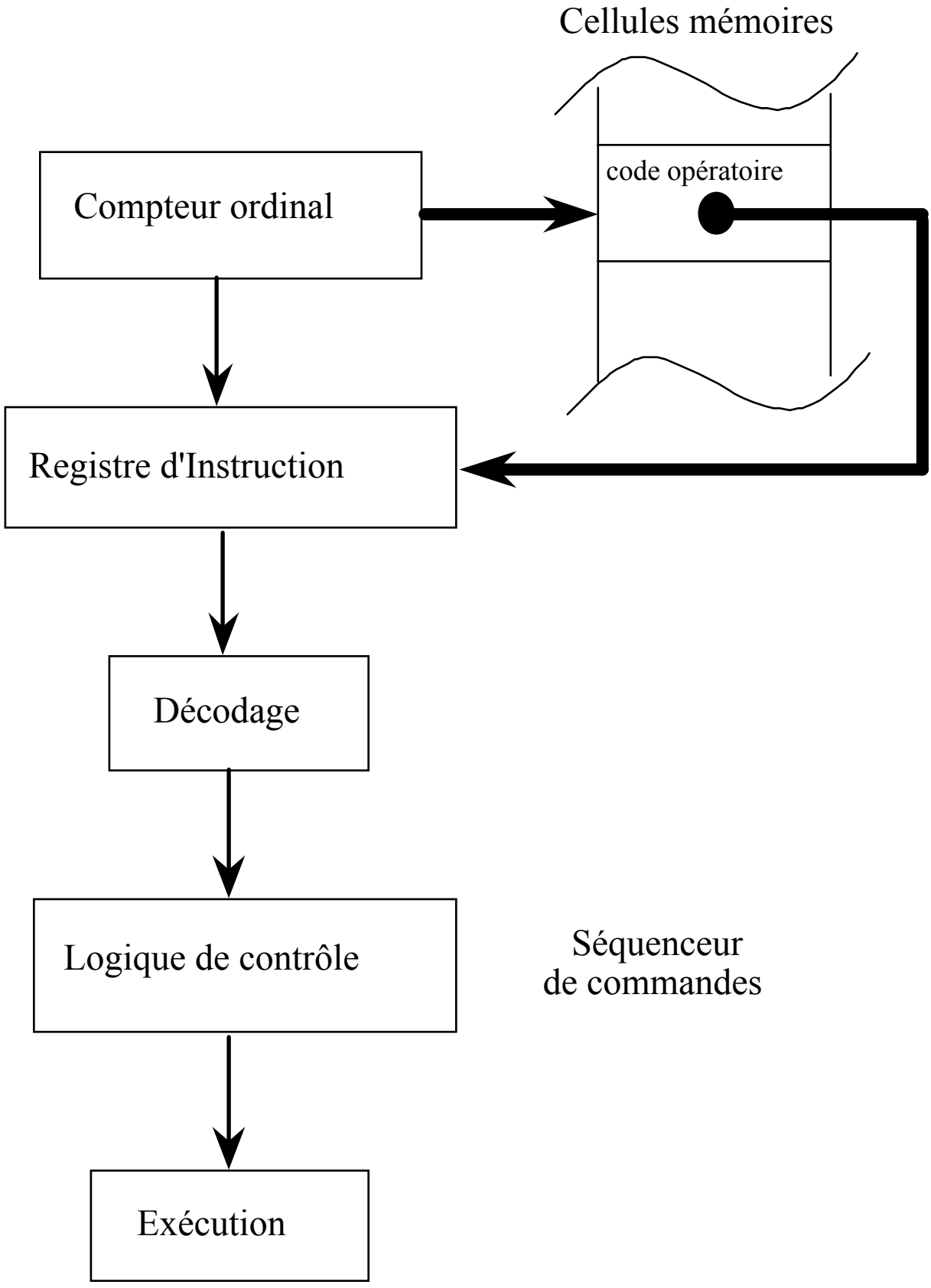
Valeur = $1997-($1960+2) soit $35

Adressage relatif long avec un déplacement positif.



Valeur = $3010-($1997+4) soit $1675

Principe de fonctionnement



Fonctionnement séquentiel

La réalisation de toute instruction s'effectue selon le cycle *extraction-exécution* qui se décompose comme suit :

1er temps : Phase d'extraction

Recherche de l'instruction suivi du **décodage**

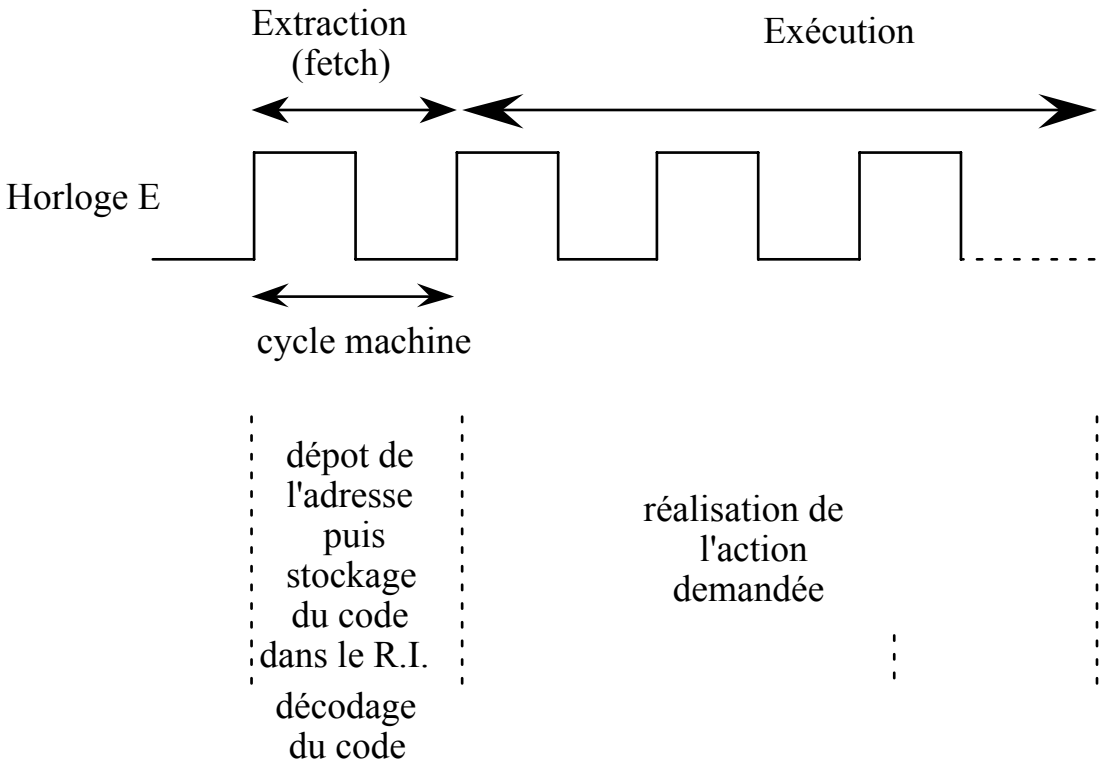
2ème temps : Phase d'exécution

Exécution du code opératoire

Cette étape est plus ou moins longue suivant la complexité de l'instruction.

Chaque opération élémentaire constituant ce cycle est validée par un signal d'horloge.

Illustration :


Extraction
(fetch)
Exécution
Horloge E
cycle machine
dépot de
l'adresse
puis
stockage
du code
dans le R.I.
décodage
du code
réalisation de
l'action
demandée

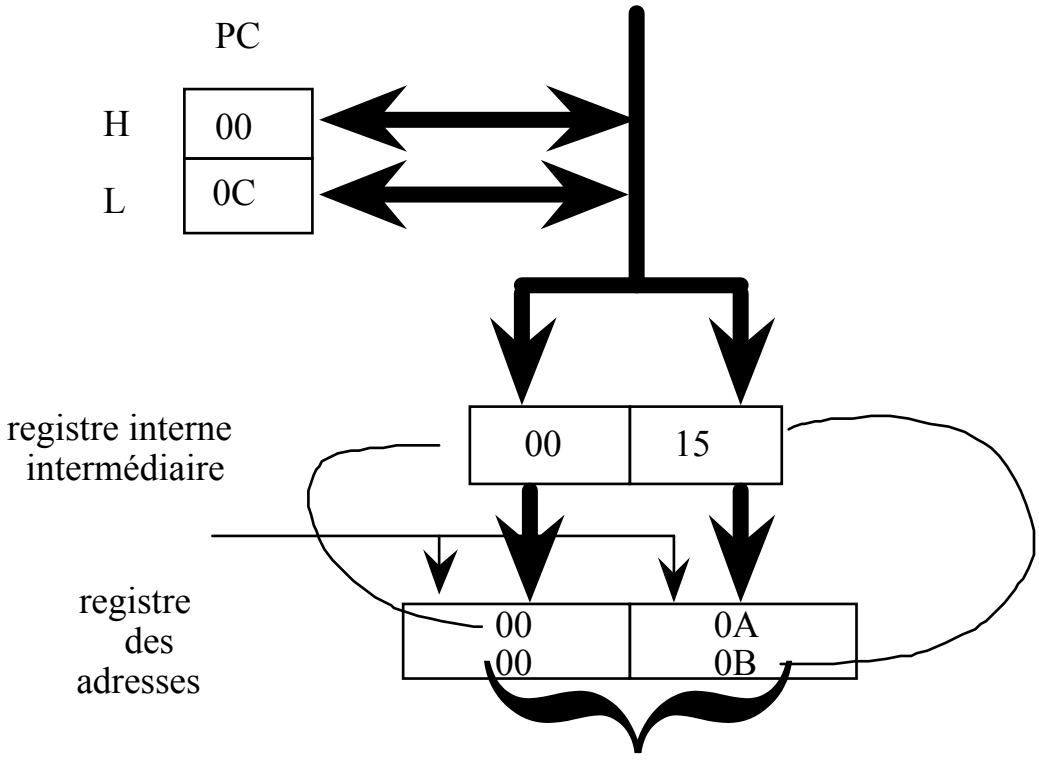
¿ Premier exemple :

$0000	LDA	#$10	86	10
$0002	ADDA	#$29	8B	29
$0004	NOP	12		

¡ Deuxième exemple :

			#	~
$0000	LDA	$0014	3	5
$0003	ADDA	$0016	3	5
$0006	STA	$0018	3	5
$0009	LDA	$0015	3	5
$000C	ADCA	$0017	3	5
$000F	STA	$0019	3	5
$0012	BRA	$FE	2	3

Commentaire :



Apport du 6809 dans la famille des microprocesseurs 8 bits.

Son architecture et ses différents modes d'adressage permettent d'aborder des concepts logiciels tels que :

La génération de programmes translatales, réentrant et stucturés.

Présentation de ces concepts

1- Définition d'un programme translatable

Ce type de programme fonctionne quelque soit l'adresse à laquelle il est implanté en mémoire.

Possible grâce aux modes d'adressage relatif (BSR, BRA) et indexé par rapport au PC (LDA valeur, PC).

L'avantage de ce mode de programme est qu'il est utilisable dans n'importe quelle application.

Le principe du programme relogeable existe aussi. Dans ce cas, (à la différence du cas précédent) les adresses relatives sont transformées en adresses absolues après édition des liens.

2- Définition d'un programme réentrant.

Un programme réentrant est un programme qui peut-être utilisé à n'importe quel niveau de priorité (Pour cela il doit être interruptible).

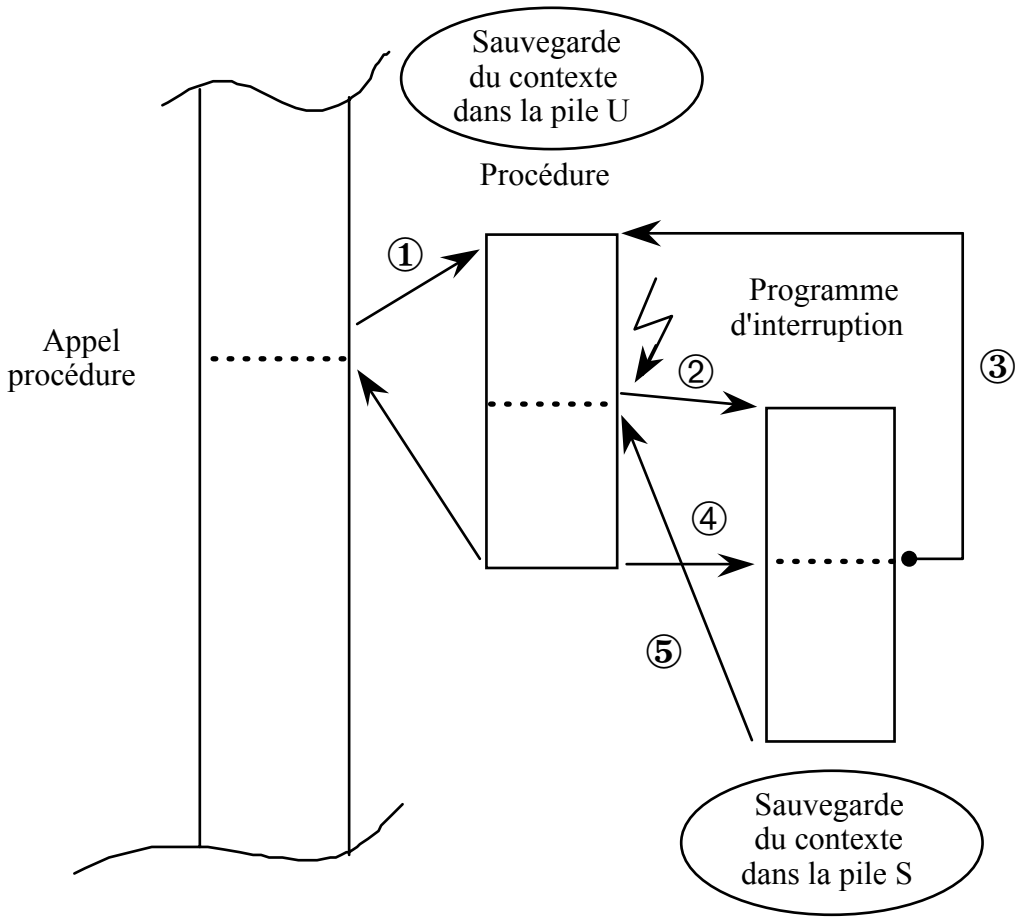
Dans ce cas, le programme d'interruption peut utiliser la même séquence de programme que celle qu'il vient d'interrompre. Il ne doit pas perturber les informations et résultats de la tâche interrompue. nécessité d'utiliser deux zones de sauvegarde des données.
Le concept de réentrance est possible avec un microprocesseur possédant plusieurs pointeurs de pile pour gérer plusieurs zones de sauvegarde.

Rendu possible avec l'existence des instructions de manipulation de pile hardware PSH et PUL et logicielle STA, -X et LDA, X+.

Equivalence :

LDX	,U++	équivalent à	PULU	X
STX	,--U	équivalent à	PSHU	X
LDA	,U+	équivalent à	PULU	A
STA	,-U	équivalent à	PSHU	A

Illustration :


Sauvegarde
du contexte
dans la pile U
Procédure
①
Programme
d'interruption
Appel
procédure
②
③
④
⑤
Sauvegarde
du contexte
dans la pile S

Les langages de programmation

Un langage est un ensemble de règles ou commandes utilisées par un programmeur afin de réaliser un programme écrit tout d'abord sous la forme d'un algorithme et ensuite traduit par un organigramme.

Le langage évolué

Les critères pris en compte lors du choix d'un langage sont :

le temps de programmation
la maintenance
la transportabilité

Un programme écrit dans un langage évolué donné peut être implanté "facilement" sur diverses machines sans réécriture. (compilateurs appropriés)

Un programme écrit dans un langage évolué est traduit en binaire de deux manières différentes.
En faisant appel soit à un *interpréteur* soit à un *compilateur.*
Tous deux sont des progammes puissants de traduction.

Un **interpréteur** agit pendant l'exécution du programme. Chaque ligne du programme est traduite en instruction binaire juste avant son exécution.
A mesure qu'une ligne est exécutée, la traduction de la ligne suivante commence.

Un compilateur traduit d'abord le programme en langage hexadécimal et ensuite le code généré est sauvegardé sous la forme d'un nouveau programme binaire. (ressemble beaucoup à un assembleur).

Le compilateur présente l'inconvénient par rapport à l'interpréteur, de nécessité beaucoup plus de mémoire mais par contre il est beaucoup plus rapide à l'exécution.

L'apprentissage d'un langage évolué rapporte strictement rien en ce qui concerne la connaissance de l'architecture fondamentale d'un microprocesseur.

Par contre la programmation langage machine ou assembleur nécessite une bonne connaissance de l'architecture du microprocesseur.

Quelles sont les raisons pour lesquelles on choisit un langage moins évolué?

Deux raisons à cela :

¨ Gain en place mémoire :

Le langage évolué nécessite pour effectuer la même tâche beaucoup plus d'instructions machines.

Exemple :
une simple instruction du type WRITELN peut faire appel à une centaine d'instruction.
Ainsi plus d'espace-mémoire est nécessaire.

¡ Gain de temps à l'exécution :

Plus d'instructions de base implique un temps d'exécution plus lent à moins de disposer de microprocesseur puissant (rapide) avec un grand champ mémoire.

Le langage d'assemblage

Ce langage est l'intermédiaire entre un langage évolué comme C ou Pascal et le langage machine.

Le langage d'assemblage est lié au microprocesseur.

Chaque processeur possède son propre ensemble d'instructions

Ses avantages :

minimise le code
meilleur contrôle sur le matériel.
[bien adapté à l'écriture des routines d'entrées/sorties]

L'assembleur est un petit compilateur !

Ce langage s'appuie sur l'abréviation de terme anglais pour chaque instruction.

Exemple de la fonction de remise à zéro (CLEAR) : CLR

Vocabulaire

¿ Le langage d'assemblage

C'est un langage constitué d'instructions et de pseudo-instructions.
Un programme écrit dans ce type de langage est une suite de lignes symboliques écrites dans un ordre logique.

¡ L'assembleur

C'est un programme de traduction qui traite chaque ligne du programme source pour fournir le code machine. Il détecte les erreurs. Le résultat de son travail s'appelle le programme objet.

¬ Macro-assembleur

C'est un assembleur qui permet de traiter des macro-instructions : ensemble d'instructions regroupées sous formes de fonction.

√ Cross-assembleur

C'est un assembleur qui permet de développer en langage d'assemblage, une application utilisant un microprocesseur donné à l'aide d'un système possédant un microprocesseur différent. Le système, dit hôte, engendre du code machine pour le microprocesseur cible.

Intéressant pour les petits systèmes car dans ce cas, on bénéficie des avantages d'un système plus important : Editeur, Système d'exploitation, etc...

ƒ Désassembleur :

C'est un logiciel qui travaille à l'inverse de l'assembleur. A partir d'un programme machine, il construit le langage mnémonique.

Exemple : $86 traduction de LoaD accumulateur A

L'assembleur 6809

Le langage d'assemblage est un langage de programmation utilisée dans les systèmes de développement, dispositifs d'aide à l'écriture de programmes.

Il permet de traduire un programme source écrit en langage symbolique (ou mnémonique) en un langage binaire.

Cette opération appelée "*Assemblage*" est réalisée par un programme approprié appelé "*Assembleur*".

Il génére directement un programme exécutable par le microprocesseur encore appelé "*programme objet*".

Comme tout langage, ce dernier possède une syntaxe et un vocabulaire propres qu'il est indispensable de connaître afin d'établir le programme source.

Syntaxe du langage assembleur

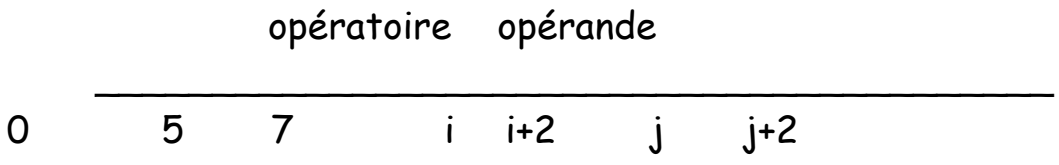
(règles d'écriture)

Chaque ligne d'instruction du programme comporte quatre parties appelées "*champs*".

Chaque champ est séparé par un blanc (espace).

Illustration :

Etiquette Code Code Commentaire


opératoire
opérande
0
5
7
i
i+2
j
j+2

Définition des différents champs

¿ Etiquette

Ce champ n'est pas obligatoire.
Une étiquette est un symbole d'au plus 6 caractères alpha-numériques, commençant obligatoirement par une lettre de l'alphabet.

Dans le cas, où le champ étiquette est vide au moins un blanc doit précéder le champ suivant.

Si le 1er caractère de la ligne est une *, la ligne est alors considérée comme un commentaire

Le rôle de l'étiquette est de permettre de répérer la position d'une instruction dans le programme.

¡ Le champ opération

Il contient soit un code mnémonique de l'opération à effectuer soit une directive d'assemblage.

¬ Le champ opérande

Il complète le champ opération et contient la "donnée" nécessaire à l'exécution de l'instruction.

Sa syntaxe est variée et dépend du mode d'adressage attribué à l'instruction.
On trouve : des étiquettes (des noms)
des symboles (nom de registre)
des nombres
des expressions : Combinaison des 3 éléments ci dessus.

Les nombres peuvent être représentés dans différentes bases décimale, octale(@), hexadécimale ($) et binaire (%).

L'identificateur de la base s'effectue grâce à un suffixe ou un préfixe dans le cas de certain assembleur (H).

Sans indication particulière, un nombre est interprété comme un décimal.

ƒ Les expressions

Elles se composent d'une suite de nombres et de symboles séparés par des opérateurs logiques, arithmétiques et des parenthèses.

Elles servent à spécifier une valeur selon les règles de la logique et de l'arithmétique.

ƒ Le champ commentaire

facultatif, il permet de documenter le programme.

Tous les caractères d'écriture disponibles dans l'éditeur sont utilisables.

Les directives d'assemblage

Une directive d'assemblage (ou pseudo-instruction) est une commande que le programmeur utilise pour donner des directives à l'assembleur et qui agit sur le processus d'assemblage.

On distingue 5 catégories de directive.

¿ Affectation de symboles : EQU et SET
¡ Gestion de l'espace mémoire : ORG et RMB
¬ Inscription de constantes en mémoire : FCB, FDB et FCC
√ Assistance à la programmation : END et OPT
ƒ Présentation des listings : TITLE et PAGE

EQU : affectation d'une valeur à un symbole ou étiquette

Syntaxe : Symbole EQU Expression

Rôle :

Cette directive affecte au symbole situé dans le champ étiquette la valeur (8 ou 16 bits) de l'expression placée dans le champ opérande.

Les symboles utilisés avec EQU ne peuvent pas être redéfinis dans la suite du programme.

SET : affectation temporaire d'une valeur à un symbole.

Rôle :

SET est semblable à EQU excepté que l'affectation est temporaire. Les symboles définis avec cette directive peuvent être à nouveau définis par la suite dans le même programme.

Exemples :

```
ARG   SET   $2          ARG=$2
ARG   SET   ARG*$2      ARG=$4
ARG   SET   ARG*ARG     ARG=$10(16₁₀)
```

ORG : Origine (initialisation du compteur programme)

Syntaxe : ORG Expression

Rôle :

Définition d'une adresse d'origine.

Charge le PC à la valeur spécifiée par l'expression située dans le champ opérande.

Remarque : Plusieurs ORG peuvent-être utilisés dans un même programme source.

On stucture ainsi l'espace mémoire en blocs :

le programme principal
les sous programmes
la gestion des interfaces

Chaque bloc est initialisé par un ORG.

RMB : Reserve Memory Bytes (Réservation d'octets en mémoire)

Syntaxe : Symbole RMB Expression

Rôle :

Provoque lors de l'assemblage un "saut" du PC, d'un nombre d'octets égal à la valeur de l'expression.

Le but étant de réserver une zone mémoire pour un usage particulier.

FCB : Form Constant Byte (définition d'une constante d'un octet)

Syntaxe : Symbole FCB expr1, expr2...

Rôle :

Inscrit la valeur exprimée sur 8 bits des expression du champ opérande dans les cases mémoires définies par la valeur du PC.
Pour chaque écriture en émoire : PC = PC+1

Remarque : Les expressions peuvent être numériques ou alphanumériques. Le symbole ' placé devant le caractère, le désigne comme étant alphabétique, dans ce cas, le code ASCII est mis en mémoire.
Exemples :

```
            ORG     $4000                   $4000   01
    DATA    FCB     1,$62+$48,'A,'B                 AA
            FCB     $FF                             41
            FCB     '4,$4                           42
            FCB     DIX+2                           FF
    DIX     EQU     $A                              34
            LDX     DATA                            04
            END                                     0C
                                                    8E
                                                    40
                                                    00
```

FDB : Form Double Byte Constant (définition d'une constante de deux octets)

Syntaxe : Symbole FDB Exp1, Exp2,...

Rôle :

Identique à FCB mais cette fois, l'expression (valeur numérique et symbole) représente une valeur sur 16 bits.
le PC est incrémenté de 2 pour chaque mot écrit.

Exemples :

```
            ORG     $4000                   $4000   19   00
    DATA    FDB     $1997,$5432+$ABCD               97   05
            FDB     'A,'B                           FF   00
```

```
        FDB     '5,$5
        FDB     10,$10
DIX     FDB     FIN
        FDB     0
FIN     EQU     4
```

```
FF  0A
00  00
41  10
00  00
42  04
00  00
35  00
```

FCC : Forme Constant Character string (définition d'une constante chaîne de caractères

Syntaxe : Symbole FCC délimiteur *chaîne* délimiteur

Rôle :

Inscrit en mémoire le code ASCII des caractères situés dans le champ opérande entre les délimiteurs. Et PC = PC+1 pour chaque caractère inscrit.

RemarqueS :

Les deux délimiteurs doivent être identiques et peuvent être n'importe lequel des caractères imprimables (on prend en général //).
L'expression FCC /A/
est équivalent à FCB 'A

Illustration :

```
MESS  FCC   /FIN/          $2000  46   49
      FCC   'LUNDI'               49   4D
      FCC   #MARDI#               4E   41
      FCC   252                   4C   52
                                  55   44
                                  4E   49
                                  44   35
```

END : fin d'assemblage

Syntaxe : END

Rôle :

Cette directive marque la limite du programme source.
Les lignes situées après cette directive sont ignorées par l'assembleur.

PAGE : Saut de page

Syntaxe : PAGE

Rôle :

Fait avancer le papier de l'imprimante au début de la page suivante.

NAME : nom du programme

Syntaxe : NAME PMAK

Rôle :

Cette directive permet de donner un nom au programme qui apparait en tête de page.

OPT : Option d'assemblage

Syntaxe : OPT ABS

Rôle :

Cette directive fournit des compléments d'informations lors de l'assemblage. Les options sont écrites dans le champ opérande et sont séparées par des virgules.

Différentes options :

ABS : ABSolu, le programme est écrit avec un adressage absolu.

REL : RELogeable, le programme est écrit avec un adressage relatif.

CRE : permet d'obtenir la table dite des références croisées.

L : demande l'impression du listing après opération d'assemblage terminé.

L'opération d'assemblage

Elle s'effectue généralement en deux étapes (2 passes).

1ère passe :

Consiste en une lecture de l'ensemble du programme source.
Au cours de cette phase, 3 analyses sont effectuées :

Analyse lexicographique :

Compare le code opératoire à une table interne (fichier comprenant les codes).

Analyse syntaxique :

Cette analyse détecte : les étiquettes manquantes
les codes opératoires erronés
les opérandes manquants

Analyse sémantique :

Cette analyse permet de dire si une ligne syntaxiquement correcte à un sens.
Détecte un symbole non défini par ailleurs.

L'assembleur construit la table des symboles employés. Il regroupe tous les symboles définis par le programmeur en leur donnant une valeur à chacun d'eux (Création de la table des références croisés).

Il génére un code intermédiaire qui servira au second passage.

Les directives d'assemblage sont également interprétées.

2ème passe :

L'assembleur génère le code objet en tenant compte des adresses et des données réelles qui figurent dans la table des symboles.

3ème passe :

Cette phase fournit le listing avec son code objet.

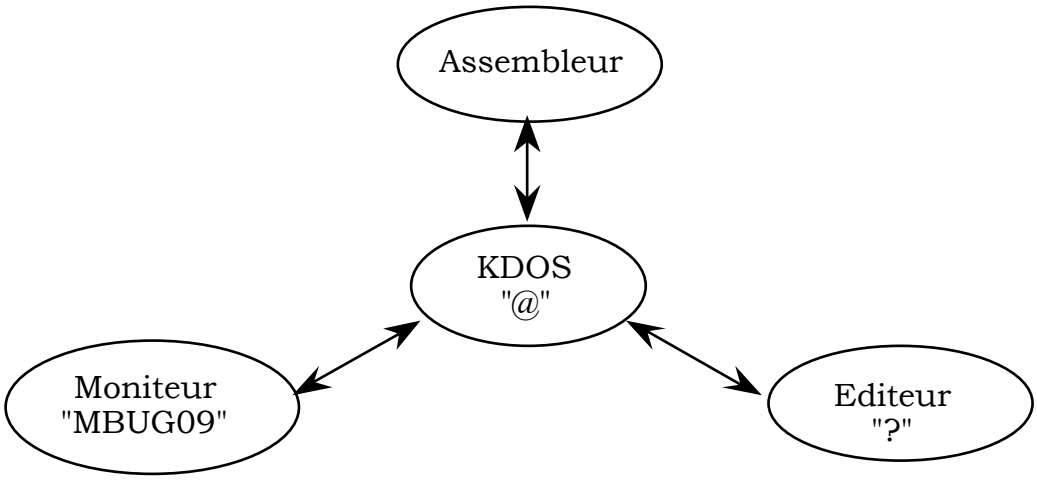
Plan adopté pour l'écriture d'un programme

```
                    OPT     ABS
Définition des variables
          DEBUT     EQU     $4000
          TIRQ      EQU     $5000
          TFIRQ     EQU     $6000
          MEM       EQU     $2000
          PIA       EQU     $EF20
Réservation des cases mémoires
                    ORG     MEM
          MEM1      RMB     1
          MEM2      RMB     2
          PILE      RMB     100
          PPILE     EQU     *
Réservation des ports
                    ORG     PIA
          DDRA      RMB     1
          CRA       RMB     1
Programme principal
                    ORG     DEBUT
                    LDS     #PPILE
                      "
                      "
                      "
                      "
                    BRA     *
Traitement des interruptions IRQ
                    ORG     TIRQ
                      "
                      "
                    RTI
Traitement des interruptions FIRQ
                    ORG     TFIRQ
                      "
                      "
```

```
                    RTI
Initialisation des vecteurs d'interruption
                    ORG     $FFF6
                    FDB     TFIRQ
                    FDB     TIRQ
                    FDB     TSWI
                    FDB     TNMI
                    FDB     DEBUT
                    END
```

Les principales commandes du systèmes EUROMAK

Environnement :



Le système d'exploitation se charge en RAM après la commande g (ou MDOS)

Toutes commandes KDOS doit comprendre la source et la destination du fichier concerné (par défaut 0).

Deux lecteurs de disquette :

disque 1 : On utilise le label 0, on y trouve la disquette système, toujours protégée en écriture.

disque 2 : avec le label 1, on y trouve la disquette des programmes utilisateurs : fichier.ASM et fichier.LO

Exemples de commande système : CAT (pour catalogue)

CAT équivalent à CAT : 0 [lecture des fichiers du disque 1]
CAT : 1 [lecture des fichiers du disque 2]

Appel de l'Editeur : EDITE

Exemples de commande :

BUILD fichier.ASM:1 [si le fichier n'existe pas encore]
LOAD fichier.ASM: 1 [si le fichier existe déjà]

Q équivalent à S(ave) : sauvegarde du programme à l'endroit précisé par la commande BUILD.
Q (une deuxième fois) sortie de l'éditeur. Retour à KDOS.

Appel de l'assembleur : RASM

Exemple de commande :

RASM fichier.ASM:1;L,CRE
Si pas d'erreur, création du fichier.LO.

Chargement du fichier assemblé
LOAD fichier.LO:1

Une fois chargé, on se trouve automatiquement dans le moniteur.
Excution du programme exécutable : G $adr

Ce type de concept facilite la mise au point et la maintenance d'un programmme.

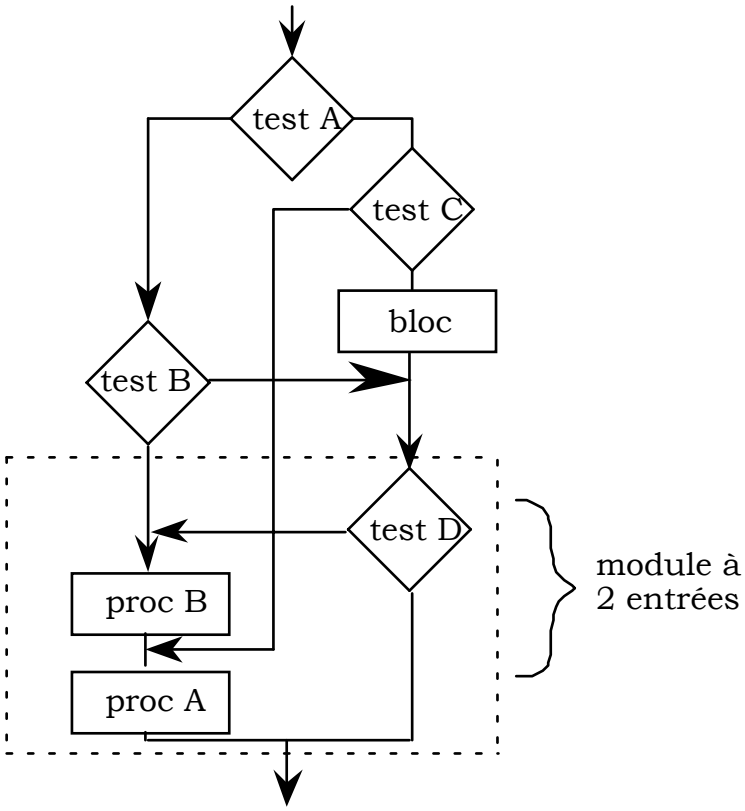
Une bonne compréhension du programme est possible dans le cas où le programme est écrit comme une suite de modules simples.

Avec pour chaque module une entrée et une sortie.

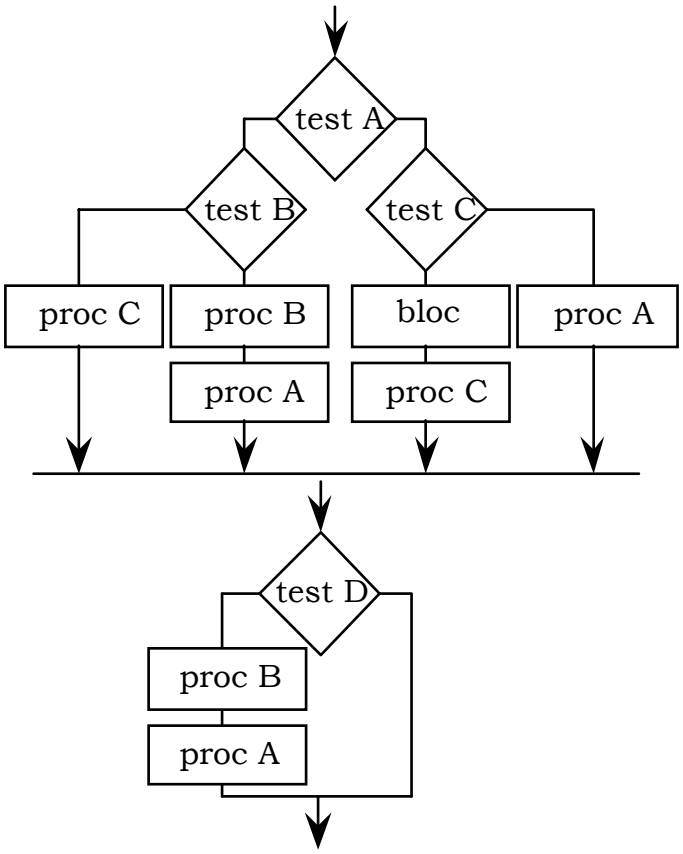
Il peut être vu comme un sous-programme ou comme une macro-instruction.

Ainsi le programme principal apparaît comme une suite d'appels de sous-programme dans lequel il n'y a pas de rupture de séquence ni saut systématique.

Exemple d'un programme non structuré.


test A
test C
bloc
test B
test D
proc B
proc A
module à
2 entrées

Transformation



Avec Proc C :

Ecriture du programme en langage pascal :

Définition des procédures :

Procédure PROC_A ;

Procédure PROC_B ;

Procédure PROC_C

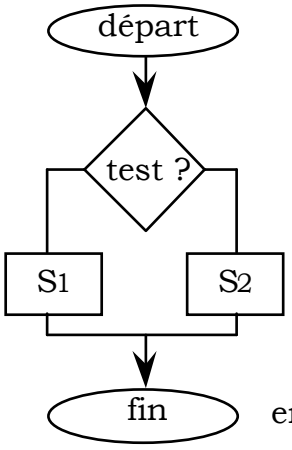
Le programme

```
Begin
    if test_A
    then    if test_C
            then    begin
                    BLOC
                    PROC_C
                    end
            else PROC_A
    else    if test B
            then PROC_C
            else    begin
                    PROC_B
```

```
                    PROC_A
                    end
end.
```

Stucture conditionnelle de base

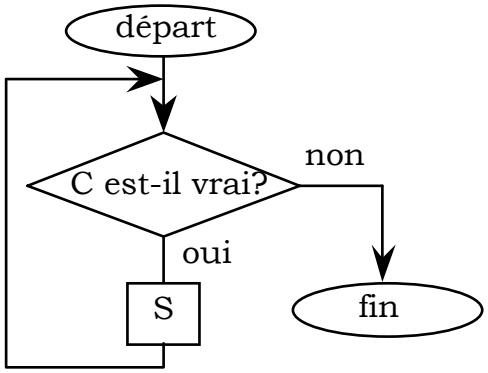
Organigramme de la stucture "*Si alors sinon*" [if then, else]



end if

Organigramme de la stucture de boucle

tant que c(condition), *faire S* (instruction) [do while]

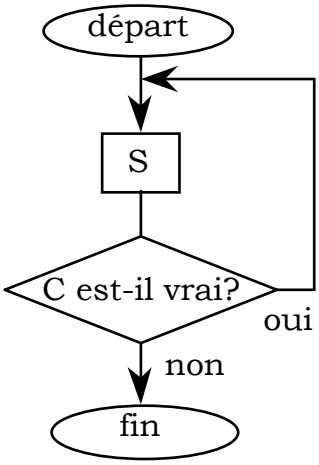


C est testé avant l'exécution de S.

Conséquence : S peut très bien ne jamais être exécutée.

Une variante :

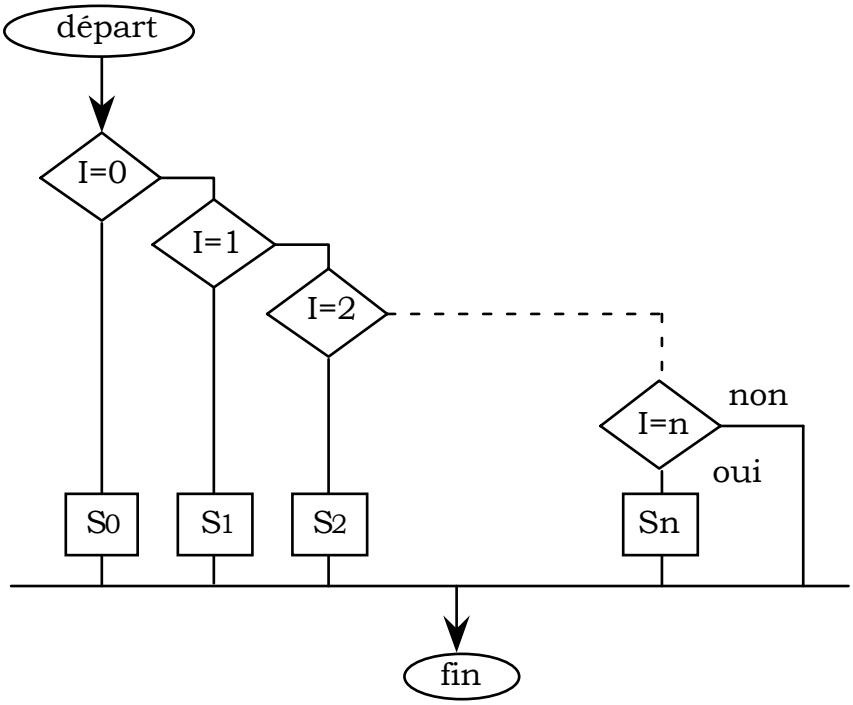
exécuter S, jusqu'à C [do until]



Ici, le test de la condition est réalisé à la fin de la boucle.

Conséquence : La boucle est exécutée au moins une fois.

Structure du *case of* :



Avantages de la programmation structurée :

Séquences des opérations simples
Nombre de structures limité
Terminologie standard
Facilité de lecture

(description possible des organigrammes)

Accroît la productivité du programmeur

On démontre que le jeu donné des structures est complet c.a.d que tout programme peut-être rédigé à l'aide des structures de base vues ci-dessus.

Inconvénients :

Exécution lente et nécessité de beaucoup de mémoire.
Obligation d'une série de boucles emboitées.

bibliographie : Z80 (627) Programmation en langage assembleur - édition Radio SECF de Laure A. Levent hal

Le fonctionnement en interruption

Définition

Une interruption est une *procédure* qui permet de suspendre l'exécution d'un programme au profit d'un autre, avec la possibilité de reprendre l'exécution du programme initial là où il avait été suspendu.

Interruptions "matérielles"

Le microprocesseur est capable de répondre à des sollicitations extérieures. Ainsi peut-il traiter des informations en "temps réel".
Pour cela, le microprocesseur est "couplé" au monde extérieur par des lignes sur lesquelles s'effectue, de façon indéterministe, un échange de messages sous forme de signaux logiques !

A partir des sollicitations extérieures, le microprocesseur doit changer "d'état" en fonction des priorités relatives de l'opération en cours et de celle qui est demandée. Selon le cas, il interrompt ou non le déroulement "normal" du programme.

Contrainte :

Si la demande est prise en compte, le microprocesseur doit être capable de traiter *rapidement* cette demande externe.

Le microprocesseur 6809 posséde pour cela quatre lignes, baptisées "entrée d'interruption" matérielles qui sont par ordre de priorité :

RESET	Ré-initialisation du microprocesseur.
NMI	Non Masquable Interrupt.
FIRQ	Fast Interrupt Request.
IRQ	Interrupt Request.

Ces entrées sont actives sont un niveau "bas".

Rôle du Registre de Code Condition

E	F		I				

a- Fonction du flag I

+ Masque concernant l'interruption IRQ.

Tableau

Etat du flag I	IRQ
I=1	Inhibée
I=0	Autorisée

Lorsque ce flag est mis à 1, le microprocesseur ne prend pas en compte les demandes d'interruption arrivant sur le ligne IRQ.

Avec ce type d'interruption, le contexte total du microprocesseur (12 octets) est sauvegardé sur la pile S.

b- Fonction du flag F

+ Masque concernant l'interruption FIRQ.

Tableau

Etat du flag F	FIRQ
F=1	Inhibée
F=0	Autorisée

Lorsque ce flag est mis à 1, le microprocesseur ne prend pas en compte les demandes d'interruption arrivant sur le ligne FIRQ.

Ici, le contexte partiel du microprocesseur (3 octets) est sauvegardé sur la pile S. (seuls les contenus des registres PC et CCR sont concernés.

Méthode de changement de l'état des flags I et F.

On peut forcer l'état des deux bits à l'aide des instructions suivantes :

ANDCC d'une part et ORCC d'autre part !

Tableau

Instruction	Etat du flag
ANDCC #$EF	CCRb_4=0
ORCC #$10	CCRb_4=1
ANDCC #$BF	CCRb_6=0
ORCC #$40	CCRb_6=1

c- Fonction du flag E

Ce flag est positionné par le microprocesseur afin d'indiquer le type de sauvegarde devant être réalisée (partielle ou complète).

Il fait office en quelque sorte de mémoire interne.

Tableau

Etat du flag E	Etat de la sauvegarde
E=1	complète
E=0	partielle

Ce flag est utilisé (sous forme de test), systématiquement lors de l'instruction RTI, par le microprocesseur, pour déterminer quelle action à mener afin de revenir proprement dans le programme suspendu (nombre de registres a dépilés).

Interruptions "logicielles

Hormis les demandes d'interruptions matérielles, il existe d'autres causes d'interruption de programme. Ce sont les interruption "soft". Elles se traduisent par des instructions à par entière que l'on place dans les programmes. Elles représentent des évènement "déterministes".

 La sauvegarde du contexte est totale.

On en dénombre trois, qui sont :

SWI	(SoftWare Interrupt)
SWI2	(SoftWare Interrupt n°2)
SWI3	(SoftWare Interrupt n°3)

Les interruptions de synchronisation

Pour finir, le microprocesseur peut se mettre en attente d'évènements extérieurs afin de synchroniser son évolution sur l'apparition de ces derniers.

Il y a une "pré-sauvegarde" complète du contexte.

Pour cela, on trouve les instructions suivantes :

CWAI	(attente d'interruption).
SYNC	(attente de synchronisation).

Quelques remarques importantes

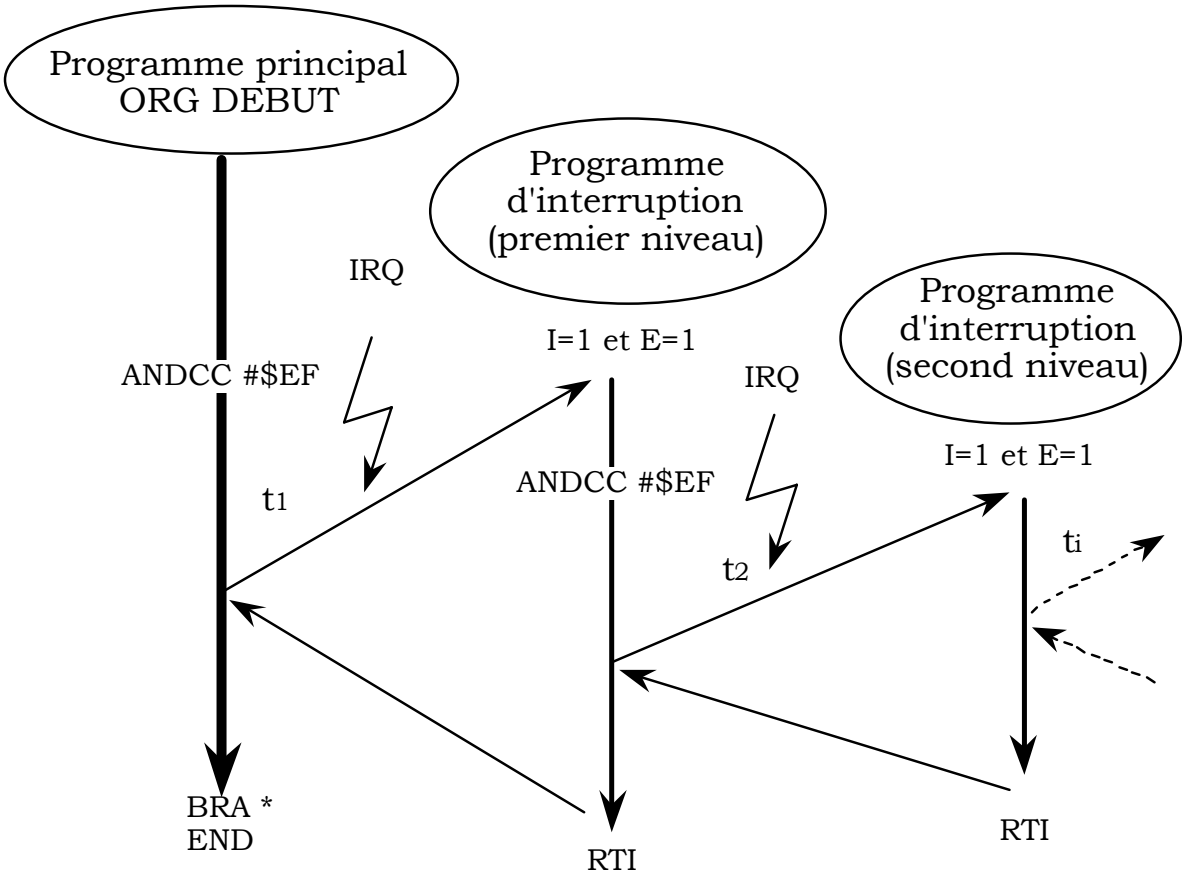
Les lignes NMI, FIRQ, IRQ et RESET ainsi que l'instruction SWI positionnent automatiquement le flag CCRb4 à 1 (masquage de l'IRQ).

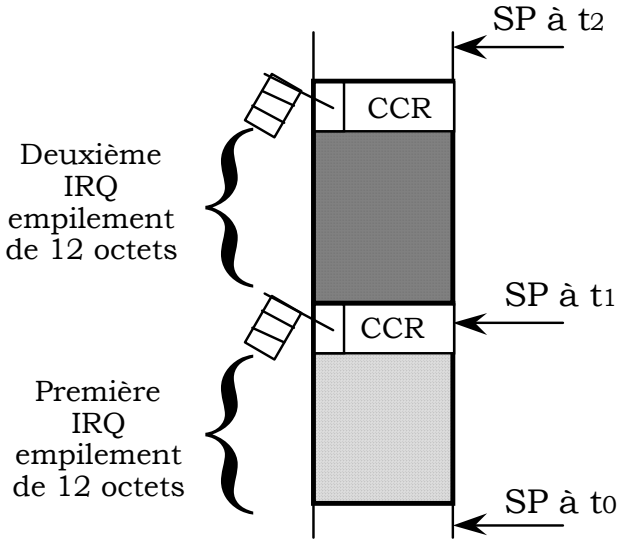
Les lignes NMI, FIRQ et RESET ainsi que l'instruction SWI positionnent automatiquement le flag $CCRb_6$ à 1 (masquage de la FIRQ).

Rien n'empêche de repositionner à 0 ces flags si on le souhaite!

Les bits I et F étant positionnés à 1 lors de l'exécution d'un programme d'interruption, il est possible, néanmoins d'autoriser la prise en compte de nouvelles interruptions en les repositionnant à O.

Illustration :





Zone mémoire RAM gérée par la pile

Tableau récapitulatif

Interruptions

Niveaux de priorité	Entrée	Instruc-tion	Vecteur	Nature de l'interruption	Sauvegarde sur la pile	Masquage réalisée
1	RESET		$FFFE $FFFF	Initialisation	Aucune	NMI I=F=1
2	NMI		$FFFC $FFFD	Non masquable	Totale E=1	I=F=1
3		SWI	$FFFA $FFFB	Logicielle	Totale E=1	I=F=1
4	FIRQ		$FFF6 $FFF7	Rapide et masquable	Partielle E=0	I=1
5	IRQ		$FFF8 $FFF9	Masquable	Totale E=1	Aucun
6		SWI2	$FFF4 $FFF5	Logicielle	Totale E=1	Aucun
6		SWI3	$FFF2 $FFF3	Logicielle	Totale E=1	Aucun

Instructions d'interruption

Instruction	Interruption concernée	Fonction	Sauvegarde	Dépilement
CWAI	NMI, FIRQ et IRQ	Attente d'interruption	Totale E=1	
SYNC	NMI, FIRQ et IRQ	Synchronisation externe	Aucune	
RTI				Total si E=1 Partiel si E=0

Traitement des interruptions

Le traitement d'une interruption se fait selon un protocole bien établi que nous allons détaillé.

Dès la mise au niveau bas de la broche adhoc, ou bien dès la rencontre de l'instruction SWI, le microprocesseur entame une "procèdure implicite", programmée dans la silice, dont les organigrammes respectifs sont représentés plus loin.

On remarque que chaque organigramme se termine par un chargement du PC avec la valeur contenue dans le vecteur de l'interruption demandée. Ceci provoque un saut à l'adresse correspondante.

Dans un système simple et figé, où la demande d'interruption ne peut provenir que d'une seule source, et où l'on n'envisage qu'un seul programme d'interruption, il est possible d'implanter directement le début de ce programme à l'adresse contenue dans le vecteur d'interruption. La sortie de la procédure implicite aboutira directement à l'exécution du programme.

Dans ce cas général où plusieurs "utilisateurs" peuvent demander l'interruption et où un "utilisateur" au moins veut pouvoir choisir entre plusieurs programmes d'interruption différents, la "procédure implicite" doit aboutir d'abord à un programme de gestion de l'interruption.

Le programme de gestion d'une interruption IRQ par exemple, doit assurer les opérations suivantes :

⇨ d'abord identifier le demandeur

⇨ faire respecter une hiérarchie dans les demandeurs lorsqu'il y a des appels simultanés. La priorité entre demandes simultanées de différentes interruptions est réalisée en partie par le jeu des masques.

⇨ brancher le microprocesseur (par un changement du PC) au programme d'interruption souhaité par le demandeur.

Méthodologie de gestion des périphériques d'entrées-sorties

Les transferts E/S peuvent s'effectuer selon différents modes. Deux grands modes apparaissent :

le mode programmé et le mode interruptible.

Le mode programmé

Les transferts s'effectuent à l'initiative du microprocesseur.. C'est lui (c.a.d. le programme) qui décide du moment des actions. Ces transferts peuvent être néanmoins inconditionnels ou non.

Mode inconditionnel

Dans ce cas, le transfert est exécuté quelque soit l'état du périphérique. Ceci suppose que ce dernier est toujours prêt à envoyer la donnée ou disponible pour recevoir celle-ci.
(Possible pour des périphériques rapides).

Mode conditionnel

Dans ce cas, au transfert s'ajoute des échanges de signaux de commande entre un interface (le PIA) et le périphérique. ceci afin d'assurer un minimum de synchronisation.
(technique de la poignée de main ou handshaking).

Principes

Avant d'effectuer un transfert, le microprocesseur teste l'état du périphérique et ne réalise l'opération qu'après s'être assuré de sa diponibilité.

L'état du périphérique (prêt ou occupé, registre de réception plein ou registre de transmission vide...) est généralement indiqué par un bit d'état appelé flag.

Dans le cas où il y a beaucoup d'interfaces il exécute des boucles d'attente sur des événements extérieurs en passant son temps à tester les indicateurs d'état de chacun (technique du sondage ou polling).

Conséquence de cette technique le microprocesseur est immobilisé dans cette tâche. Très pénalisant !
Le remède...

Le mode interruptible

Dans ce cas, les transferts d'E/S s'effectuent à la demande du périphérique.

(Exemple du clavier : le microprocesseur ne peut pas savoir à quel moment l'utilisateur appuyera sur une touche du clavier).

Celui-ci envoie par l'intermédiaire du circuit d'interface une demande d'interruption au microprocesseur. Si celle-ci n'est pas masquée ou inhibée, le microprocesseur termine l'exécution de l'instruction en cours, puis se branche dans le programme d'IT grâce au vecteur d'interruption.

Cette méthode permet de décharger le microprocesseur des opérations de polling.

Avantage :

Les périphériques peuvent fonctionner à leur propre vitesse sans pénaliser pour autant le microprocesseur.

Questions :

Que faire s'il y a plusieurs périphériques susceptibles de faire des demandes d'IT?

Dans ce cas, il faut pouvoir identifier le périphérique demandeur .

Trois possibilités :

1- On utilise les différentes entrées d'IT du microprocesseur si cela est possible.

2- On réunit les différentes sorties de demande d'interruption avec un câblage en "ou câble". Cela oblige alors de faire une recherche du demandeur par scrutation.

3- On utilise des circuits spéciaux qui répondent au mieux à ce problème avec prise en compte des priorités au niveau des demandes simultanées.

Cas 1

Pas de problème.

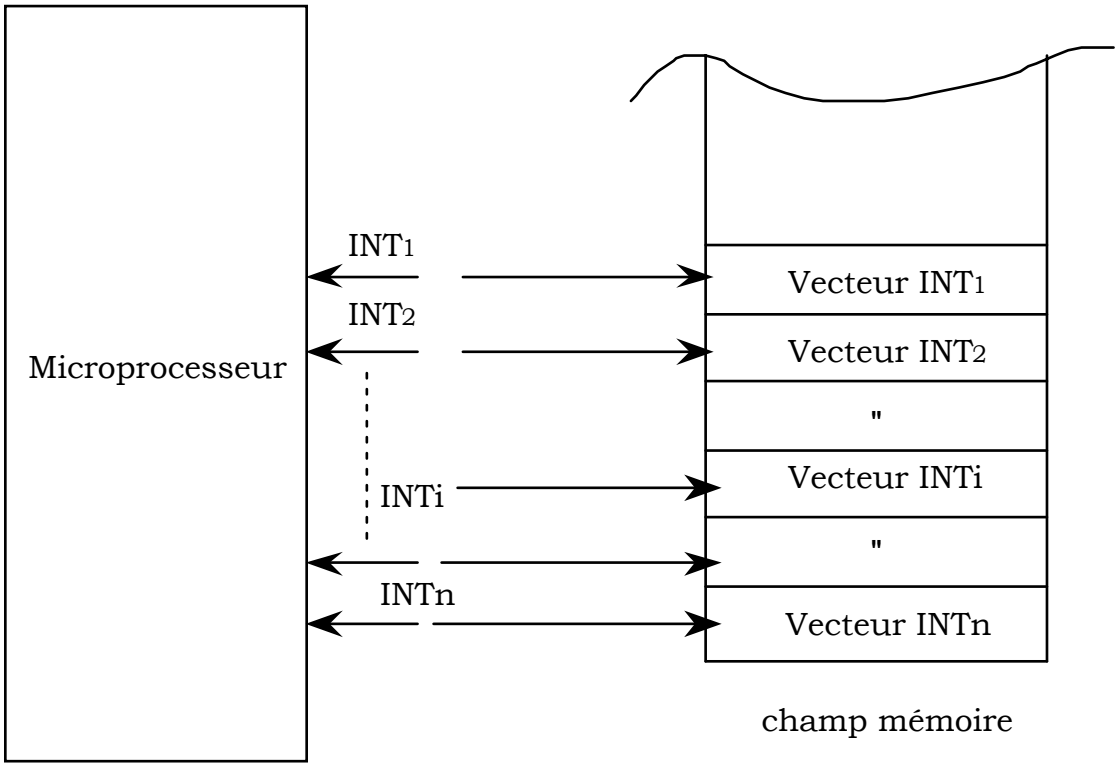
Chaque périphérique possède son propre vecteur d'interruption.

On sait quel est le mécanisme mis en place lors de la prise en compte d'une telle demande :

⇨sauvegarde du contexte integral ou partiel.

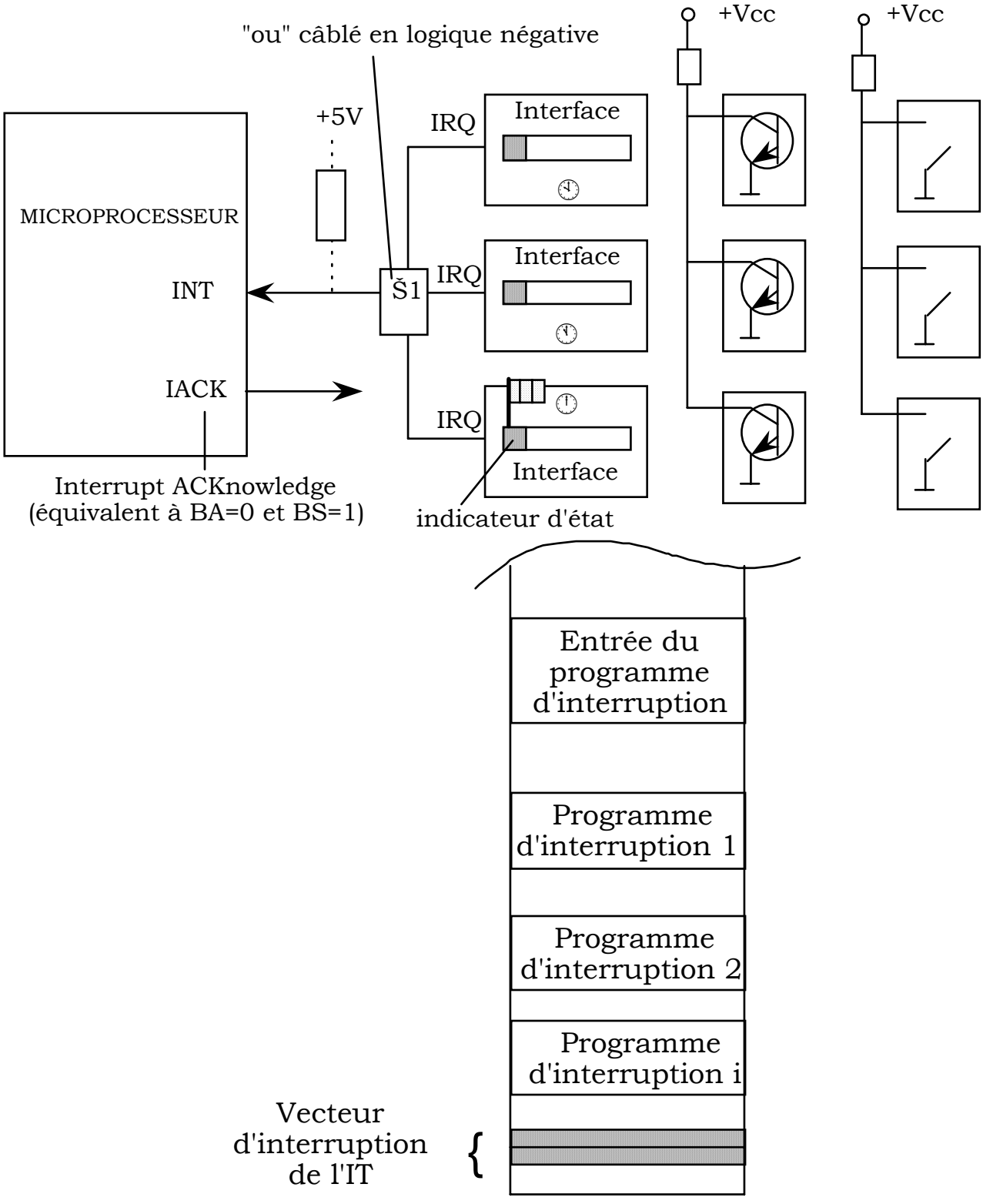
⇨chargement du PC avec le contenu de vecteur d'interruption spécifique.

Schéma :



cas 2

On considère une seule entrée de demande d'interruption sur le microprocesseur avec plusieurs interfaces. On est amené alors à adopter le schéma suivant :



Gestion des priorités

Cas 3

Avec la structure du "ou câblé", le programme d'interruption doit comporter un mécanisme de recherche (de scrutation) pour connaitre quel est le contrôleur qui a provoqué la demande.

Ainsi apparaît la notion de Priorité!

La gestion des priorités peut se faire de façon logicielle ou matérielle.

Méthode logicielle

Après prise en compte de l'interruption (le contenu de vecteur allant dans le PC), la première tâche du programme d'interruption est de tester chaque indicateur d'état des interfaces afin de savoir qui a effectué la demande.

Ici l'ordre de scrutation détermine le niveau de priorité des interfaces entre eux.

Dans cet esprit, il est nécessaire de gérer une table dans laquelle se trouve les adresses (pointeurs d'entrée) des différents sous-programmes d'IT.

On peut "voir" ces adresses comme des vecteurs d'IT de second niveau propre à chaque interface.

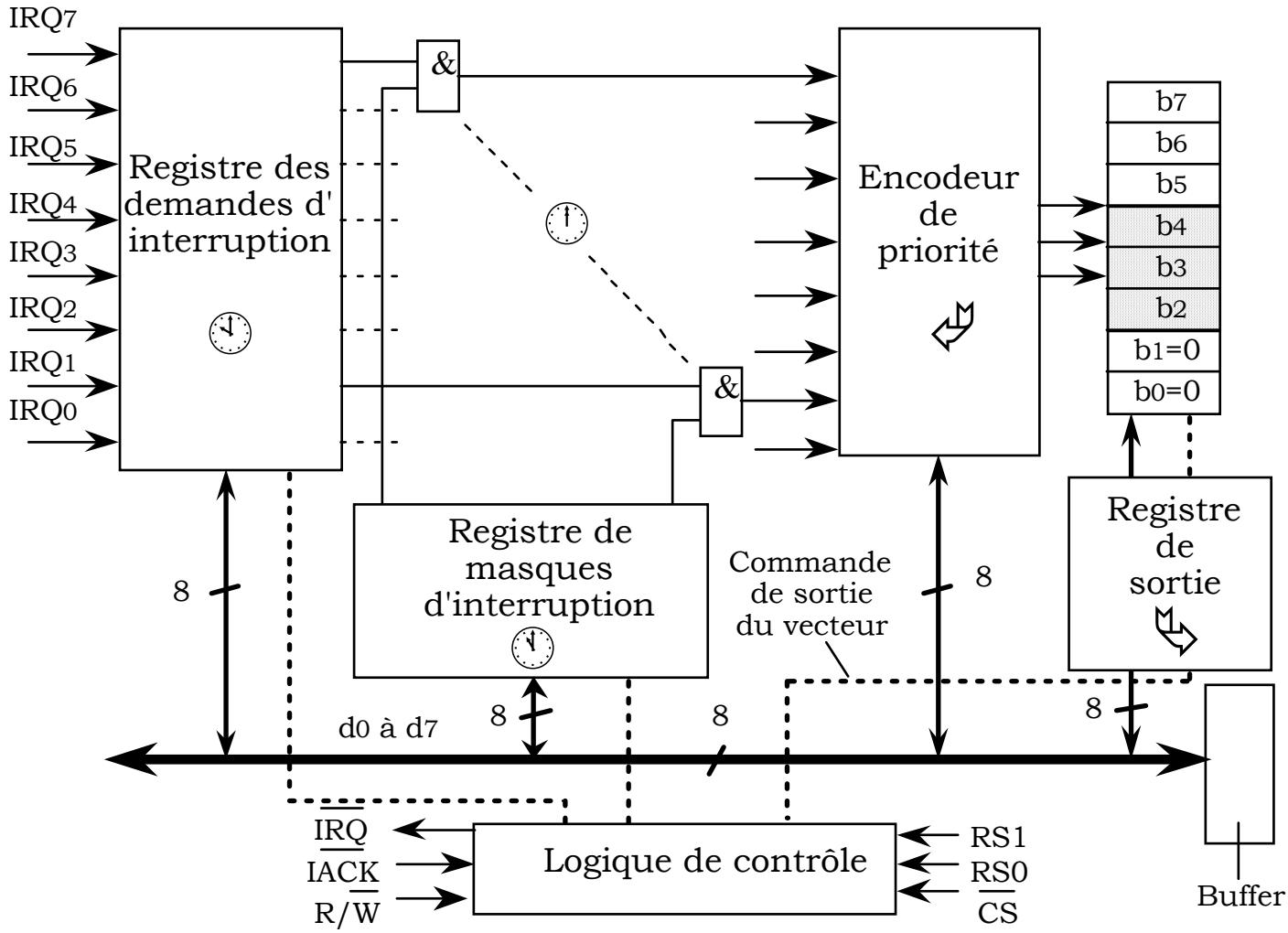
Méthode matérielle

On peut établir la hiérarchie entre les interfaces en utilisant une logique supplémentaire qui permet d'identifier directement l'origine de la demande.
C'est à dire que l'on crée directement le vecteur d'IT de second niveau.

Avantage :

Prise en compte rapide de la demande d'IT en provenance d'un interface (la phase de recherche disparaît...).

Première illustration matérielle :



Commentaires :

1 Registre d'entrée capable de stocker huit demandes d'IT.

2 Registre interne de masque d'IT (initialisé par le microprocesseur).

3 Ensemble de ET pour bloquer ou non les demandes d'IT à partir des informations du registre de masques.

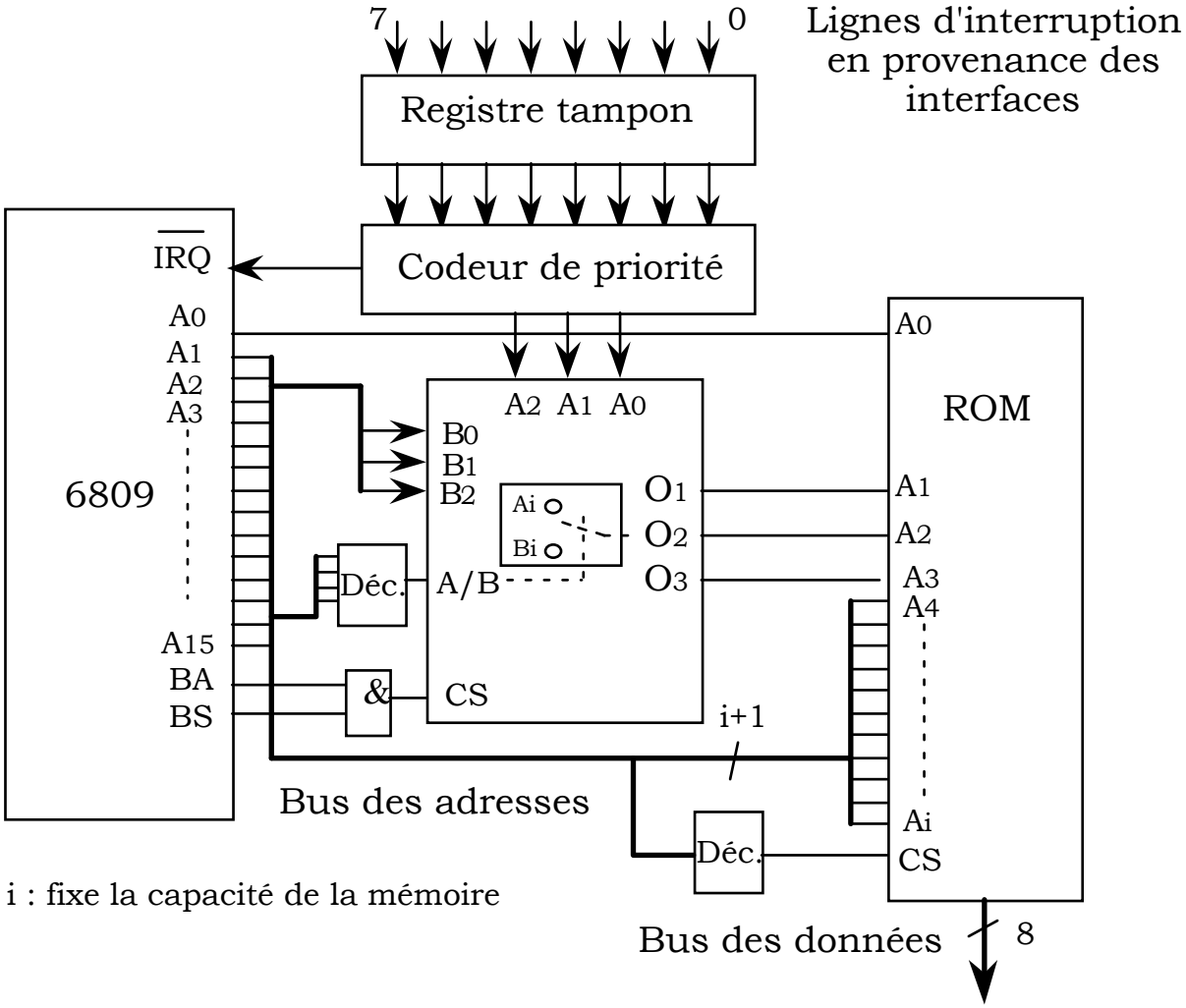
3 Codeur de priorité qui fournit le codage de la demande la plus prioritaire parmi celles actives.

4 Registre de sortie qui fournit l'octet constituant une partie de l'adresse du vecteur. L'autre partie étant définie par ailleurs (b_5, b_6 et b_7 étant initialisés par le microprocesseur).

Seconde illustration

Vectorisation par décodeur de priorité et Mux.

Ici, les sorties du codeur de priorité sont multiplexées avec les lignes d'adresse A_1, A_2 et A_3.



Commentaires :

La ligne A_0 est reliée directement à la mémoire car on agit sur les adresses.
Le multiplexeur est actif seulement en phase de reconnaissance d'IT (BA=0 et BS=1) et lorsque le vecteur d'IT est déposé sur le bus d'adresse ($FFF8 et $FFF9 dans notre exemple).
Le reste du Mux se traduit par l'apparition d'un code ($O_1O_2O_3$).
Ainsi, le microprocesseur accède directement au vecteur d'IT correspondant à la demande. Il n'y a donc plus besoin de la scrutation logicielle des interfaces.

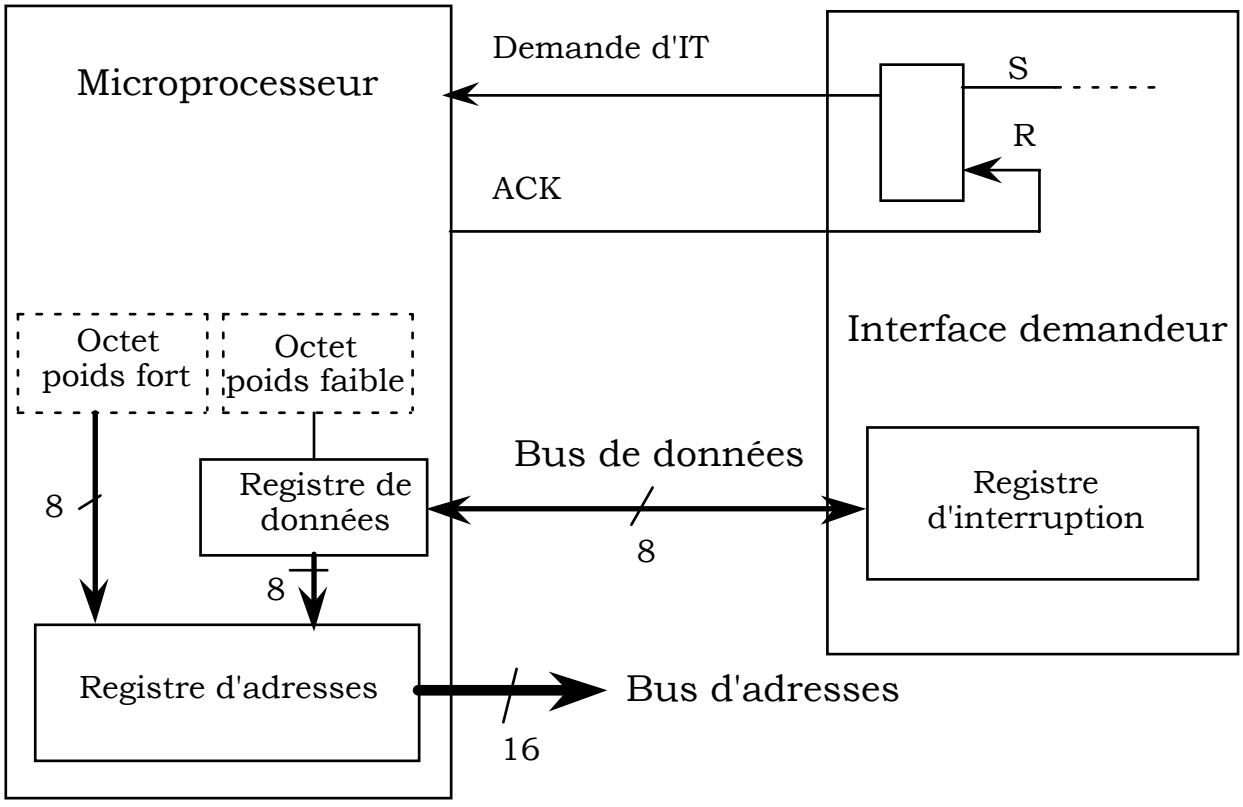
Autre structure (orientée plutôt INTEL)

Une solution, plus souple et plus économique en circuits, est possible lorsque l'interface comporte un registre d'interruption qui peut-être programmé par le microprocesseur (au même titre qu'un registre de contrôle).

Lors de la prise en compte de l'IT, le contenu de ce registre fourni la partie basse du vecteur (le choix est donc plus grand).

La partie haute du vecteur est imposé par le microprocesseur.

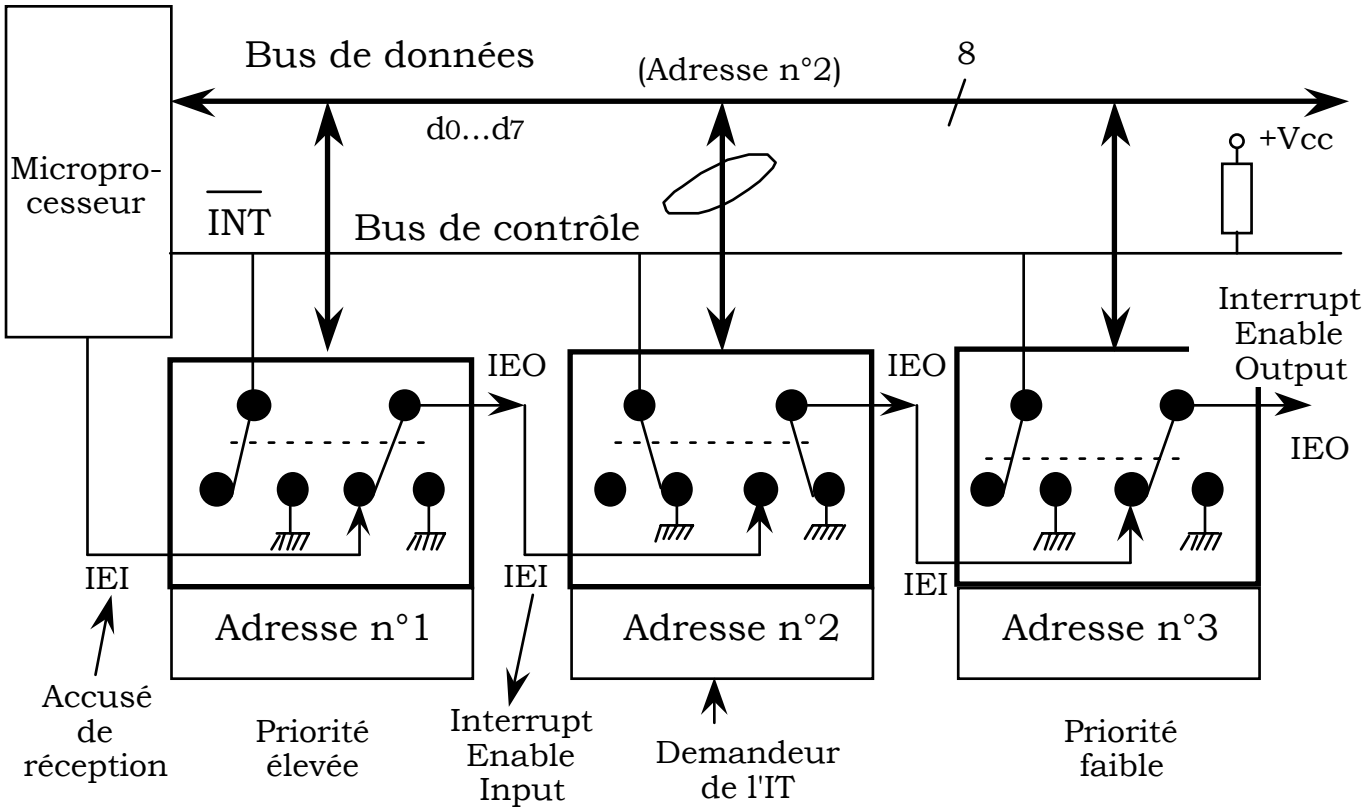
Schéma de principe :



Avec ce type d'interface, la gestion des IT peut se faire suivant le principe de la "***Daisy chain*** ".

Grâce à ce chaînage de priorité, le microprocesseur dispose rapidement du vecteur. L'interface de plus forte priorité dépose sur le bus des données l'octet bas du vecteur.

Schéma de principe :



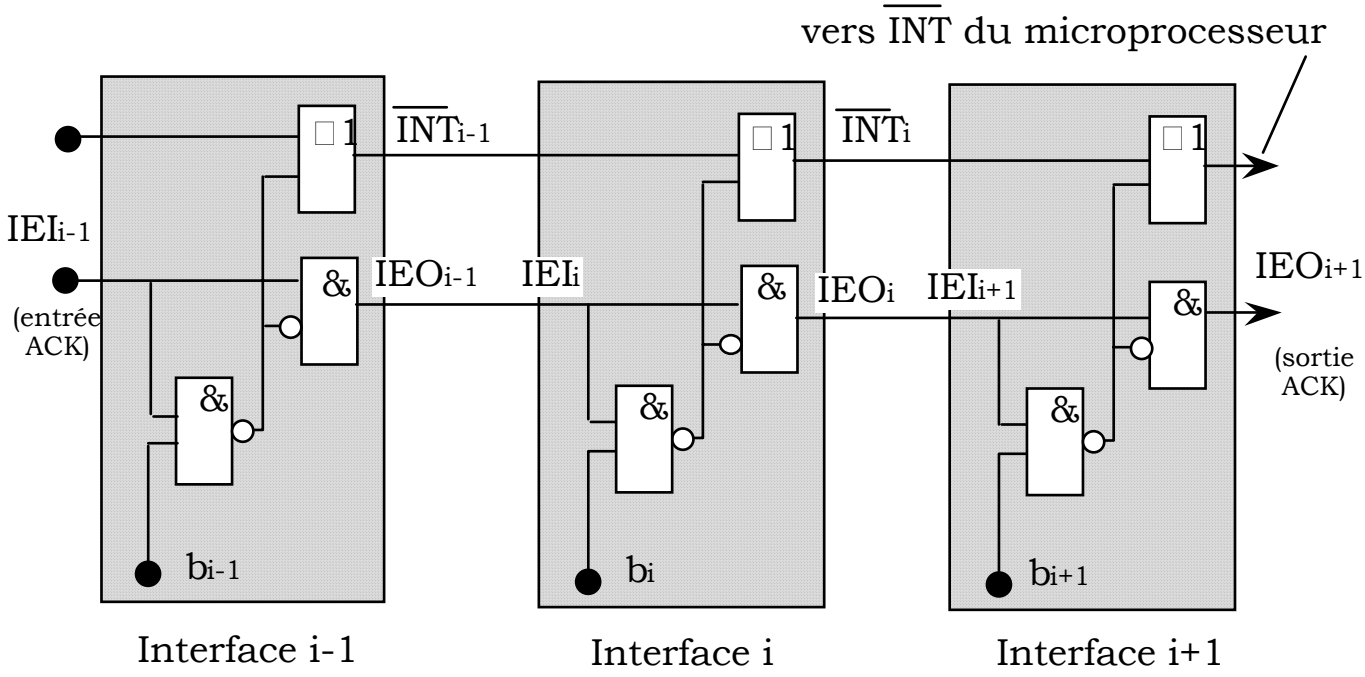
Légende :

Un dispositif inactif, c'est à dire ne demandant pas d'IT, propage le signal présent sur son entrée IEI vers sa sortie IEO.

Lorsqu'un interface est actif :
a- Il impose IEO à bas (il n'y a plus propagation de l'information présenté sur IEI).
b- Il attend l'acquittement du microprocesseur pour présenter l'adresse (octet haut) du programme d'interruption sur le bus de données.

Ainsi, seul le premier dispositif actif de la chaîne peut répondre au microprocesseur.
La priorité est obtenue dans ce cas par la position de l'interface dans la chaîne!

Schéma de principe



Equations :

$$IEO_i = IEI_i \cdot IEI_i \cdot b_i$$
$$INT_i = INT_{i-1} + IEI_i \cdot b_i$$

Commentaires

Au repos : $INT_i = 0$ et $b_i = 0$
$IEI_i = IOI_{i+1} = 1$

Soit $b_i = 1$ alors INT_i et INT haut (envoi d'une interruption vers le microprocesseur).

Conséquence :

L'accusé se réception ACK = IEI_{i-1}=1 implique IEO_{i-1}=IEI_i=1 (le signal ACK est pris en compte par l'étage i).

Mais IEO_i =0 (plus de signal ACK pour les étages suivants i+1.
IEO_{i+n} toujours à bas quelque soit le signal b_{i+n}.

Si maintenant b_{i-1}=1 alors le signal INT toujours bas.

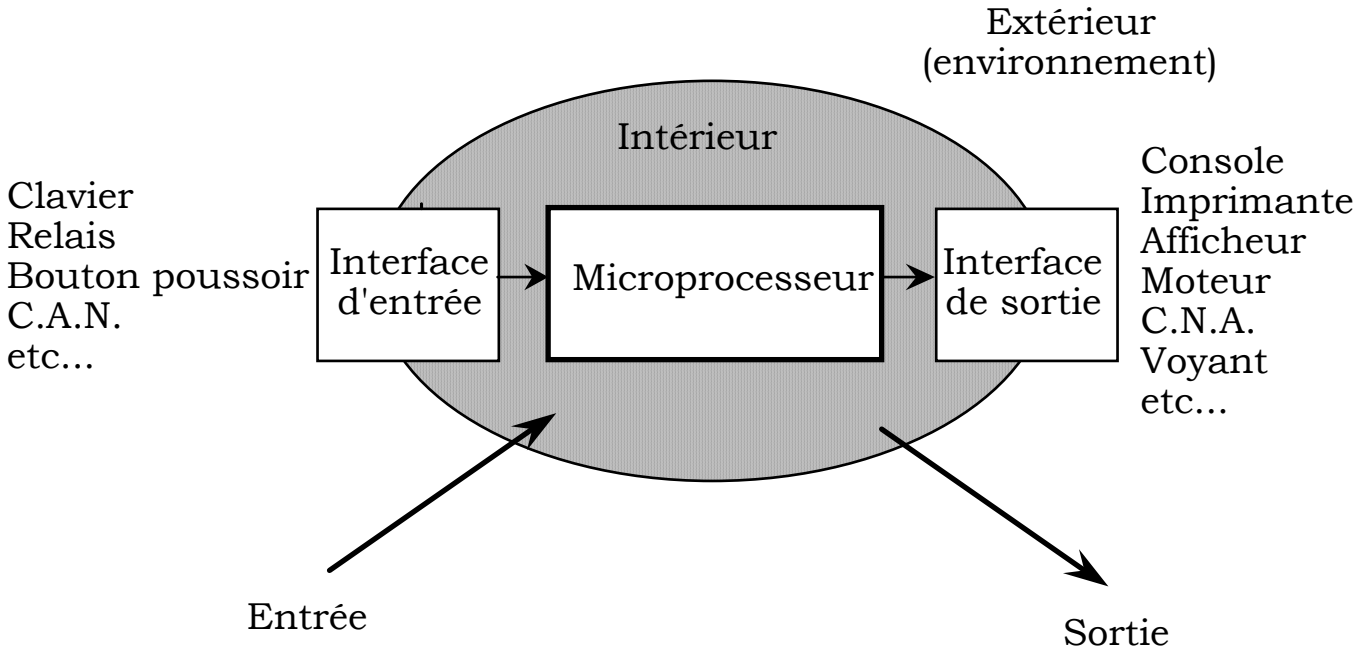
L'accusé de réception (signal ACK) est pris en compte par l'étage i-1 mais plus par l'étage i et encore moins par les autres.

L'interface i-1 est prioritaire sur l'interface i et i+n.

NOTIONS D'ENTREE ET DE SORTIE

Un microprocesseur avec son programme est "sourd" et "aveugle". Cependant il lui est nécessaire de communiquer avec l'extérieur pour réaliser une action.

Schéma :



Définition d'un signal d'entrée.

C'est un signal venant de l'extérieur sur lequel le microprocesseur peut vérifier (par lecteur) son niveau logique.
Il ne peut en aucun cas modifier l'état de celui-ci.

Définition d'un signal de sortie.

C'est un signal allant vers l'extérieur sur lequel le microprocesseur contrôle le niveau logique.
Il peut mettre ce signal aux niveaux haut et bas.

L'INTERFACAGE AVEC L'ENVIRONNEMENT

Le microprocesseur ne travaille pas uniquement avec ses accumulateurs, ses registres et sa mémoire.

Il lui faut communiquer avec l'extérieur.

Pour cela, on utilise des unités d'échange pour assurer le dialogue entre le microprocesseur et la périphérie.
Ces échanges étant assurés par l'intermédiaire des bus de données, de contrôle et d'adresses.

Ces unités d'échange (appelées interfaces) occupent un certain nombres de cases mémoires dans le champ mémoire du microprocesseur. Ils sont programmables pour s'adapter aux périphériques connectés.

Plusieurs types d'interfaces existent sur le marché. En résumé, on trouve deux catégories :

Les interfaces passifs :

Ces unités ou *coupleur* permettent l'interfaçage matériel entre des périphériques, divers et variés, et le bus de données.
La gestion des signaux d'échange incombe au microprocesseur

Dans le cas où le nombre de transferts est faible, le microprocesseur assure cette tâche assez rapidement.

La situation est critique dans le cas où l'on doit gérer un périphérique avec des fonctions complexes. L'interfaçage avec ce type d'unité implique un développement matériel et logiciel plus important. (d'où temps microprocesseur plus long).

Aussi trouve-t-on pour contourner cet inconvénient des unités dites spécialisées (appelées contrôleurs). Ces contrôleurs répondent mieux à la spécificité de certains périphériques comme les disques souples ou durs, les écrans vidéo ou bien encore, facilite la mise en oeuvre de protocole tel que celui du G.P.I.B.- I.E.E.E.

Ces unités une fois programmées allège considérablement le travail du microprocesseur qui peut faire bien autre chose.

Les interfaces actifs :

Pour les périphériques vraiment complexes, on trouve des unités d'échange appelées interfaces dites *intelligentes*.
Ici, le microprocesseur ne perd plus de temps pour gérer les échanges. Ceux-ci sont pris complétement en charge par l'unité.
Bien souvent, ils sont dotés d'un microprocesseur eux-mêmes. Ils ont à charge généralement de gérer des coupleurs simples ou spécialisés.

Certains de ces contrôleurs intelligents sont plus spécialisés que d'autres pour accomplir une tâche bien précise et très complexe. Citons par exemple la gestion d'écran graphique.

Remarque :

Notons que le co-processeur qui dispense le microprocesseur de toutes tâches de calcul peut être considéré comme une interface intelligent spécialisée !

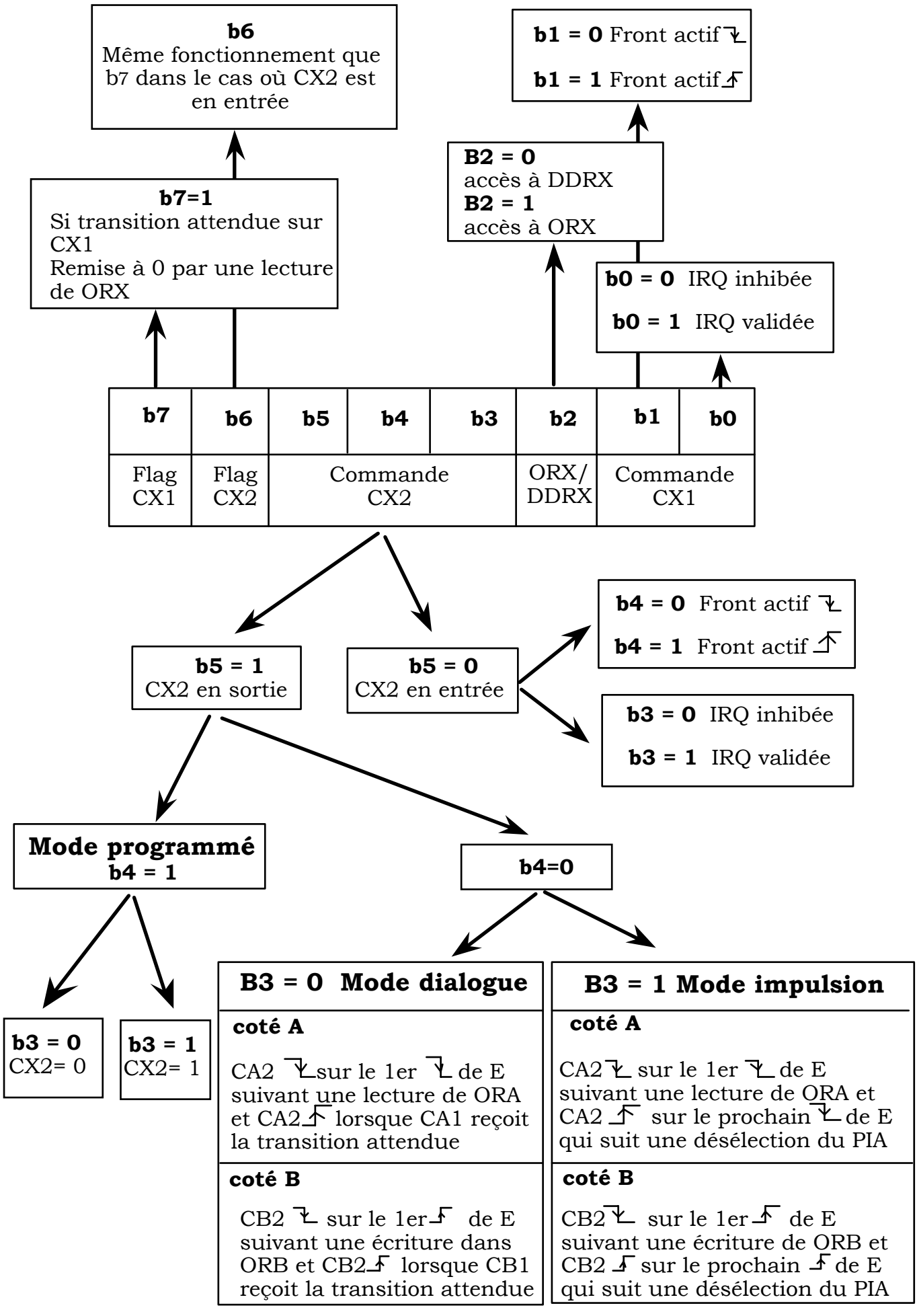
Tableau :

Interface passif	simple
	spécialisé
Interface actif (intelligent)	universel
	spécialisé

Nous allons découvrir les unités simples suivantes :

P.I.A., A.C.I.A., et P.T.M.

et spécialisées : le contrôleur de gestion d'écran.


b6
Même fonctionnement que b7 dans le cas où CX2 est en entrée
b1 = 0 Front actif
b1 = 1 Front actif
B2 = 0
accès à DDRX
B2 = 1
accès à ORX
b7=1
Si transition attendue sur CX1
Remise à 0 par une lecture de ORX
b0 = 0 IRQ inhibée
b0 = 1 IRQ validée
b7 b6 b5 b4 b3 b2 b1 b0
Flag CX1
Flag CX2
Commande CX2
ORX/ DDRX
Commande CX1
b4 = 0 Front actif
b4 = 1 Front actif
b5 = 1
CX2 en sortie
b5 = 0
CX2 en entrée
b3 = 0 IRQ inhibée
b3 = 1 IRQ validée
Mode programmé
b4 = 1
b4=0
b3 = 0
CX2= 0
b3 = 1
CX2= 1
B3 = 0 Mode dialogue
coté A
CA2 sur le 1er de E suivant une lecture de ORA et CA2 lorsque CA1 reçoit la transition attendue
coté B
CB2 sur le 1er de E suivant une écriture dans ORB et CB2 lorsque CB1 reçoit la transition attendue
B3 = 1 Mode impulsion
coté A
CA2 sur le 1er de E suivant une lecture de ORA et CA2 sur le prochain de E qui suit une désélection du PIA
coté B
CB2 sur le 1er de E suivant une écriture de ORB et CB2 sur le prochain de E qui suit une désélection du PIA

Etude de l'interface série asynchrone
A.C.I.A. 6850

I- Généralités

A.C.I.A. : Asynchronous communication interface adapter.
Adaptateur pour la communication asynchrone.

Appartient à la famille des U.A.R.T.

Ce circuit programmable permet la communication série asynchrone selon la procédure START-STOP.

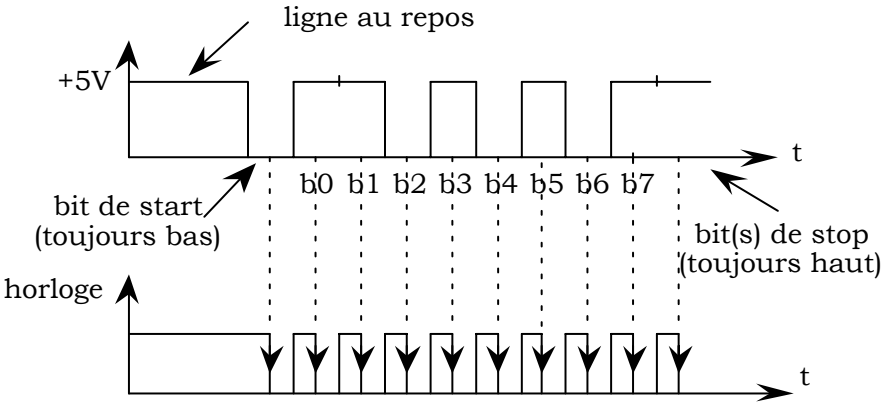
Procédure très utilisée pour de faibles débits d'informations (50 ->19 200 bits/seconde : vitesse de travail de bon nombre de périphériques).

Format du mot :

Il comprend entre 5 et 8 bits, l'ensemble étant précédé d'un bit de start et suivi de 1 ou 1,5 ou 2 bits de stop.

Le bit de start est synchronisé sur une horloge mais la suite des caractères est asynchrone.

Illustration :

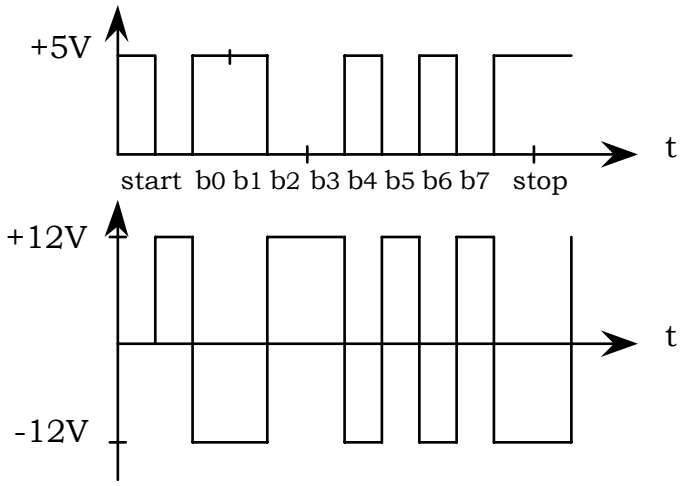


Le circuit travaille en parallèle (bus données 8 bits) coté microprocesseur et en série coté périphérique (télétype, clavier, imprimante, Modem, etc...

Les niveaux logiques délivrés par l'ACIA sont compatibles TTL. Pour configurer les signaux au standard RS-232C, il est nécessaire d'adjoindre des circuits, de conversion de niveaux (TTC⇨R232), les 1489 et (RS232⇨TTC), les 1488.

Il faut distinguer les niveaux fournis par l'ACIA : TTC (logique positive) et ceux délivrés par la ligne.

Illustration :



Généralement, les infos sont véhiculées, dans le code ASCII,

⇨ soit par boucle de courant recommandé pour les milieux fortement perturbés :

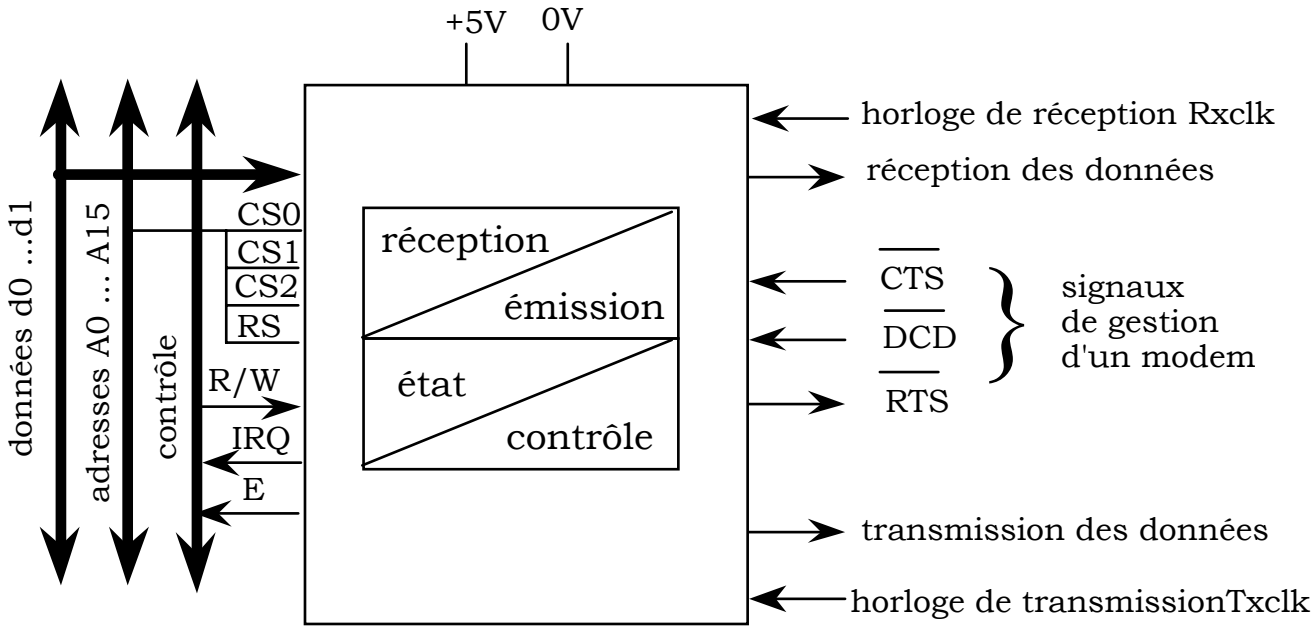
niveau 0 : circuit ouvert
niveau 1 : 20 mA

⇨ soit par liaison RS 232 (20Kbits)
RS 423 (100 Kbits)
RS 422 (10 mégabits)

niveau 1 : -12V
niveau 0 : +12V

Présentation

Schéma :



Remarque : il n'y a pas d'entrée Reset.

Ce circuit intègre :

- ⇨ un émetteur de données asynchrone
- ⇨ un récepteur de données asynchrone
- ⇨ une logique de commande Modem
- ⇨ des entrées d'horloge séparées pour l'émission et la réception.

L'émission et la réception peuvent fonctionner simultanément (full-duplex) et avec des vitesses différentes.

Ce circuit comprend quatre registres internes :

- ⇨ 1 registre de transmission
- ⇨ 1 registre de réception.
- ⇨ 1 registre de contrôle
- ⇨ 1 registre d'état

Etude matérielle

Ce circuit possède 24 broches.

Coté microprocesseur :

⇨ Le bus des données : d_0 ... d_7

Assure l'échange des données autre le microprocesseur et l'ACIA.
Lorsque le boîtier n'est pas sélectionné ces lignes bidirectionnelles sont en haute impédance.

⇨ Le bus de contrôle :

La ligne E : Signal d'activation des échanges.

La ligne R/W : ligne de contrôle du transfert des données avec le microprocesseur.
R/W = 1 ; seuls les buffers de sorties sont activés
(possibilité de lire un registre).
R/W = O ; seuls les buffers d'entrée sont activés
(possibilité d'écriture dans un registre)
Cette ligne est utilisée également comme ligne supplémentaire pour l'adressage des registres internes (cf plus loin).

On remarque l'absence de la ligne RESET - Ce qui signifie que l'initialisation se fait par soft.

⇨ Le bus des adresses :

CSO, CS1 ET /CS2 (Chip Select)

Ces 3 lignes sont reliées via le décodeur au bus des adresses du microprocesseur pour sélectionner le boîtier, la sélection est validée lorsque la combinaison est : (110)

RS (Register select)

Cette entrée permet de sélectionner les registres internes (2 octets mémoire).
Elle est utilisée conjointement avec la ligne R/W de sorte que l'on puisse choisir un registre parmi les 4 disponibles. (Voir plus loin tableau récapitulatif).

IRQ (Interrupt request)

Ligne de sortie a drain ouvert (pas de R de rappel) active sur un niveau bas - reliée aux entrées IRQ ou NMI du microprocesseur ou au PIC (6828 : priority interrupt control).

Coté extérieur

1- Les lignes "horloge"

Txclk : horloge de transmission

Sert de synchronisation (référence) pour la transmission des données sur la ligne Txdata.
Le registre de décalage concernant la transmission est synchronisé sur un front descendant de ce signal.

Rxclk : horloge de réception

Sert de synchronisation pour la réception des données sur la ligne Rxdata.
Le registre de décalage (chargement et décalage), spécifique à la réception, est piloté par un front montant de ce signal.

Les vitesses de transmission et réception peuvent varier de O à 500 Kbits/s.
Il y a possibilité de diviser ces deux horloges par 16 ou 64.

2- Les lignes de contrôle d'un périphérique type modem.

a- Ligne CTS (Clear To send)

Entrée permettant de savoir si le modem est prêt à recevoir des infos.
Un niveau bas signifie que le modem est prêt .
Un niveau haut signifie que le modem est absent (ou non prêt).
Influence sur le bit (TDRE) du SR
Remarque : S'il n'y a pas de modem (ou autre) mettre toujours cette entrée au niveau bas.

b- Ligne DCD (Data carrier detect)

Détection de porteuse ou perte de la porteuse.

Entrée permettant de savoir si la porteuse au niveau du modem est présente. Une absence de celle-ci inhibe la réception, cela se traduit par un niveau haut sur cette entrée. (défaut sur la ligne !)
Un niveau bas signifie une présence de la porteuse au niveau du modem, état normal.
remarque : non utilisée, elle doit être toujours au niveau bas !

Cette entrée peut générer une interruption IRQ, si CR7 = 1 et si un front montant est apparu sur DCD.

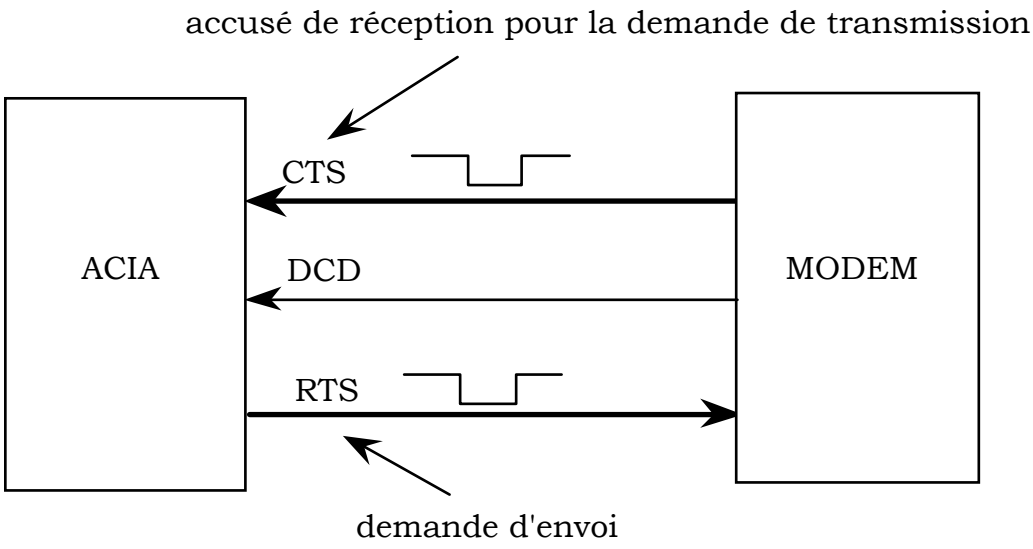
c- La sortie RTS (Request To Send)

Demande d'émission.

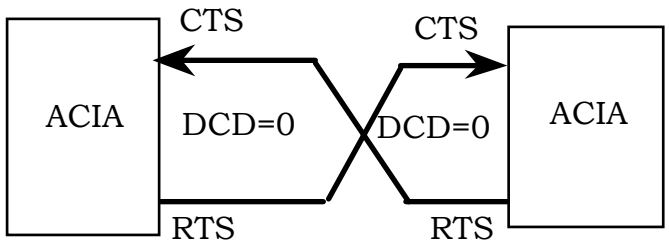
Cette sortie permet de solliciter le modem (ou autre) pour une transmission par le microprocesseur (émission demandée par le microprocesseur).
L'état de cette sortie dépend du mot écrit dans le CR. La demande d'émission se traduit par RTS au niveau bas.

Exemples d'utilisation :

a- Modem commandé par un ACIA



b- Liaison Acia - Acia



Organisation interne

⇨ **le TDR** (Transmit Data Register)

Registre de transmission dans lequel on place le mot (8 bits) à transmettre. l'écriture du mot a lieu sur le front descendant de E.

Fonctionnement :

Si pas de transmission en cours, alors le contenu du TDR est transféré dans le registre à décalage automatiquement après une instruction d'écriture.
Si par contre, un caractère est en cours de transmission, le transfert TDR dans le registre à décalage est différé que l'opération de décalage est en cours (b_0 en 1^{er})

Grâce au double registre, le caractère suivant peut-être écrit dans TDR même si le caractère précédent est encore en cours de transmission dans le registre à décalage.

Dès que le transfert à lieu - Un bit du registre d'état est positionné à "1" (voir plus loin)

⇨ **Le RDR** (Réception Data Register)

Registre de réception dans lequel on reçoit le mot (8 bits) en provenance du périphérique.

Fonctionnement :

Quand un caractère complet est reçu, il est automatiquement transféré du registre de décalage de réception dans le RDR. A ce moment là, un bit dans le registre d'état est

positionné à "1". Le caractère peut alors être lu par le microprocesseur.

Tant que le bit dans le registre d'état est à 1, le transfert automatique est suspendu !

Du fait du double registre, la lecture peut-être différée tant que la réception du mot suivant n'est pas terminé.

C'est le bit b_0 de la donnée qui est reçu en 1er.

⇨ **Le registre d'état SR** (Status Register)

Permet au microprocesseur de connaître à tout instant l'état d'une transmission ou d'une réception.

⇨ **Le registre de contrôle CR** (Control Register)

Permet de configurer le circuit en adéquation avec le périphérique.
Reçoit les paramètres de fonctionnement sous forme d'un mot de contrôle (8 bits) écrit sur le front descendant de E - aussi bien pour la transmission que la réception.

Ces registres ont la particularité d'être soit en écriture soit en lecture uniquement. Etant donné que l'ACIA est vu par le microprocesseur comme deux cases-mémoire, le complément du décodage est réalisé par la ligne R/W barre.

Les combinaisons entre RS et R/W permettent de sélectionner l'ensemble des registres selon le schéma suivant :

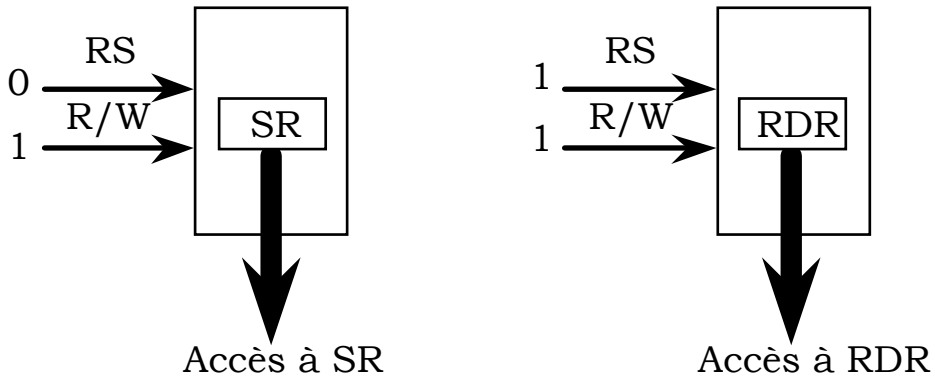
	RS	
R/W	0	1
0	CR	TDR
1	SR	RDR

à écriture seulement.
à lecture seulement.

Tableau récapitulatif (avec A_0=RS) :

	R/W	RS	Observation
adr	0	0	CR
adr+1	0	1	TDR
adr	1	0	SR
adr+1	1	1	RDR

Illustrations :



Présentation du registre de contrôle

CR0 et CR1 :

Déterminent le rapport de division sur les signaux d'entrée Rxclk et Txclk.
La combinaison (11) génère un reset logiciel (Master reset)) indispensable avant toute programmation du CR.

Le Master reset : initialisation locale (propre à l'ACIA)

Conséquences :

Remise à 0 du SR excepté les bits liés aux conditions externes :

Initialise le TDR et RDR (contenu nul)

Aucune modification néanmoins des autres bits du CR.

CR2, CR3 et CR4 :

Choix du format de l'octet à transmettre où à recevoir conformément au tableau ci-joint.

Remarques :

On a la possibilité d'insérer un bit de parité.
Si l'on a choisi une parité paire, le nombre total de bits au niveau haut, y compris le bit de parité, doit être pair.
Si l'on a choisi une parité dite impaire, le nombre total de bits au niveau haut, y compris le bit d'imparité, doit être impair.

Une erreur de parité est signalée si au cours de la transmission un parasite fait apparaître ou disparaître un bit au niveau haut.

Une erreur de format est signalé si le bit de stop n'existe pas.

CR5 et CR6 :

Ces deux bits contrôlent à la fois la sortie RTS et la demande d'interruption en transmission.

Autorisent ou non une interruption provoquée par le fait que le TDR est vide (TDRE = 1 voir plus loin).

Pourquoi ?

D'une façon générale, l'envoi d'un caractère vers la périphérie est toujours précédé d'un test sur TDR pour s'assurer que celui-ci est vide. Ce test pour être obtenu par :

1- Une scrutation permanente du bit dans le SR (dans ce cas on choisit CR6 et CR5 = 00).

2- Une interruption générée par l'ACIA vers le microprocesseur chaque fois que le TDR est vide (dans ce cas, on choisit CR6 et CR5 = 01).

Remarque :

L'ACIA met systématiquement CR6 et CR5 = 01 à la mise sous tension pour éviter l'envoi d'information intempestives.

CR5 et CR6 = 11 permet d'envoyer un break sur la ligne de sortie c'est-à-dire un niveau bas.

CR7 (Ne concerne que le récepteur).
Ce bit sert de masque d'interruption concernant les événements suivants :

1- Registre de réception plein

2- Front montant sur l'entrée DCD indiquant la perte de la porteuse.

Ce bit mis à 1 traduit ces événements par une demande d'interruption IRQ bas.

Etude des bits du registre d'état.

SR0

La lecture du bit b_0 indique si le registre de Réception RDR est plein (SR0 = 1)

Remise à 0 par : ⇨ Une lecture du RDR
⇨ Un Master Reset
⇨ Une entrée DCD haut (plus de porteuse donc plus de donnée!)

Remarque : SR0 = 0 --> RDR vide --> on attend !

SR1

Concerne le registre de Transmission.

SR1 = 1 : lorsque la donnée est disponible dans le TDR.

Remise à 0 par : ⇨ Une écriture dans le TDR
⇨ Un Master reset
⇨ Une entrée CTS haut (par prêt à recevoir)
Remarque : SR1 = 0 --> TDR est plein --> on attend !

SR2 :

Indique l'état électrique de la ligne DCD barre.

SR2 = 1 si DCD = 1 (absence de la porteuse)
Ceci peut générer une interruption IRQ si autorisée (CR7 = 1)

Ce bit est remis à 0 par :
⇨ Une lecture du SR suivie par une lecture du RDR après avoir DCD retour à bas bien sûr !

SR3

Indique l'état de l'entrée CTS

SR3 = 1 signifie que CTS est haut (modem pas prêt à recevoir)

Conséquence : forçage de SR1 à 0 (TDR plein) ce qui signifie que la transmission est interrompue !
Ce bit n'est pas affecté par un Master reset.

SR4

Indique une erreur de formatage (FE)

SR4 = 1 : indique un problème de synchronisation qui se caractérise par une absence du (des) bit(s) de stop.

Mise à 0 de SR4 après :

- ⇨ une lecture du RDR après disparition du problème
- ⇨ un MASTER RESET
- ⇨ un niveau bas sur DCD

SR5

Indique un débordement (OVRN)

SR5 = 1 signifie que plusieurs caractères ont été reçus avant la lecture du caractère précédent.

Si problème résolu, mise à 0 de SR5 après :

- ⇨ Lecture du RDR.
- ⇨ Un niveau haut sur DCD ou un Master RESET.

SR6

Indique une erreur de parité (PE)

SR6 = 1, indique que la parité reçue avec le caractère est incorrect.

SR6 est remis à zéro après :

- ⇨ Une lecture du RDR si pb résolu !
- ⇨ Un MASTER RESET.

SR7

SR7=1 : Indique qu'il y a une demande d'interruption.

Cette demande peut provenir des événements suivants :

- ⇨ RDR plein (SR0 = 1) si CR7 = 1
- ⇨ TDR vide (SR1 = 1) si CR5 = CR6 = 1
- ⇨ DCD haut si CR7 = 1

Remise à 0 de ce bit après :

- ⇨ par une lecture de RDR
- ⇨ par une écriture de TDR
- ⇨ par une lecture de SR et du RDR (DCD)

Etude Logicielle

Programmer un ACIA consiste à écrire dans son registre de contrôle un octet pour définir le mode de fonctionnement.
A la mise sous tension, la logique interne d'initialisation met l'ACIA dans un état d'attente pour éviter des transitions intempestives en sortie. l'Acia sort de cet état par l'initialisation programmée "Master reset" - qui lui permet de prendre en compte le chargement du CR avec le mot approprié.

Principes de fonctionnement

En général, le programme de transmission commence par une interrogation du bit SR1 appelé TDRE (TDR vide) - Ce bit est mis à 1 lorsque le caractère précédemment chargé dans le TDR est transféré dans le registre à décalage de transmission
(rajout des bits de start, stop et parité) pour y être décalé au rythme de Txclk et apparaître sous forme série sur la ligne TxData.
Le TDR est alors libre d'accueillir un nouveau mot sans risquer "d'écraser" le mot précédent.

Pour procéder à l'écriture d'un nouveau mot, le microprocesseur doit donc se mettre en polling sur TDRE, ou exploiter la demande d'interruption générée lors du passage au niveau 1 de TDRE.

Les données sont reçues en série par l'entrée RxData en provenance du périphérique, à une fréquence sous multiple de l'horloge de transmission. Le rapport est programmé à l'initialisation de l'ACIA. suivant l'un des 3 modes.

Ensuite transféré dans le RDR avec élimination des bits de start, de stop et de parité.

Une séquence de réception commence par une lecture de SR pour tester le bit RDRF = 1, soit par une mise en polling sur ce bit, soit à la suite d'une demande d'interruption. Dès que RDRF = 1, la réception du mot est terminée et celui-ci peut être lu par le microprocesseur.

La parité du mot reçu est contrôlée lors de cette réception. Le registre d'état indique si le registre de réception contient un caractère avec d'éventuelles erreurs de parité, de format ou d'overflow.

Gestion des interruptions

En transmission :

Si CR5 = 1 et CR6 = 0, la ligne IRQ est activée au niveau bas dès que le TDR est déclaré vide par le passage au niveau 1 de TDRE.
Le programme de traitement de cette interruption a évidemment comme but d'écrire le caractère suivant dans le TDR, ce qui entraîne en plus la remise à zéro de TDRE donc le retour à l'état de repos de la ligne IRQ ainsi que la remise à zéro du bit SR7 du registre d'état.
Ce fonctionnement est maintenu tant que l'ACIA n'est pas sous contrôle d'un Master Reset et tant que la ligne CTS est maintenue au niveau bas par le modem.
Rappelons que lorsque le modem ne veut plus transmettre de caractère, il met CTS au niveau haut ce qui force TDRE au niveau bas et inhibe le processus ci-dessus.

En réception :

Si CR7 vaut 1, la ligne IRQ est activé au niveau bas à chaque fois que le RDR est plein (RDRF=1), le programme d'interruption est chargé de lire le caractère reçu, ce qui en outre remet RDRF à zéro ainsi que la ligne d'interruption au repos.
Cette opération est déclenchée également si une perte de la porteuse est signalée (DCD=1) ou si le bit OVRN (SR5) passe au niveau haut. Les flags (PE) et (FE) ne déclenchent pas d'interruption et c'est à la charge du programme d'interruption lui-même de contrôler que le mot reçu ne comporte pas d'erreur en testant les deux bits SR4 et SR6.

Remarques :

Ce fonctionnement en mode interruption laisse le microprocesseur libre de faire autre chose entre chaque écriture et/ou chaque lecture.
En effet les fréquences usuelles de fonctionnement de l'ACIA sont :

Pour le Minitel 75 b/s en réception et 1200 b/s en émission

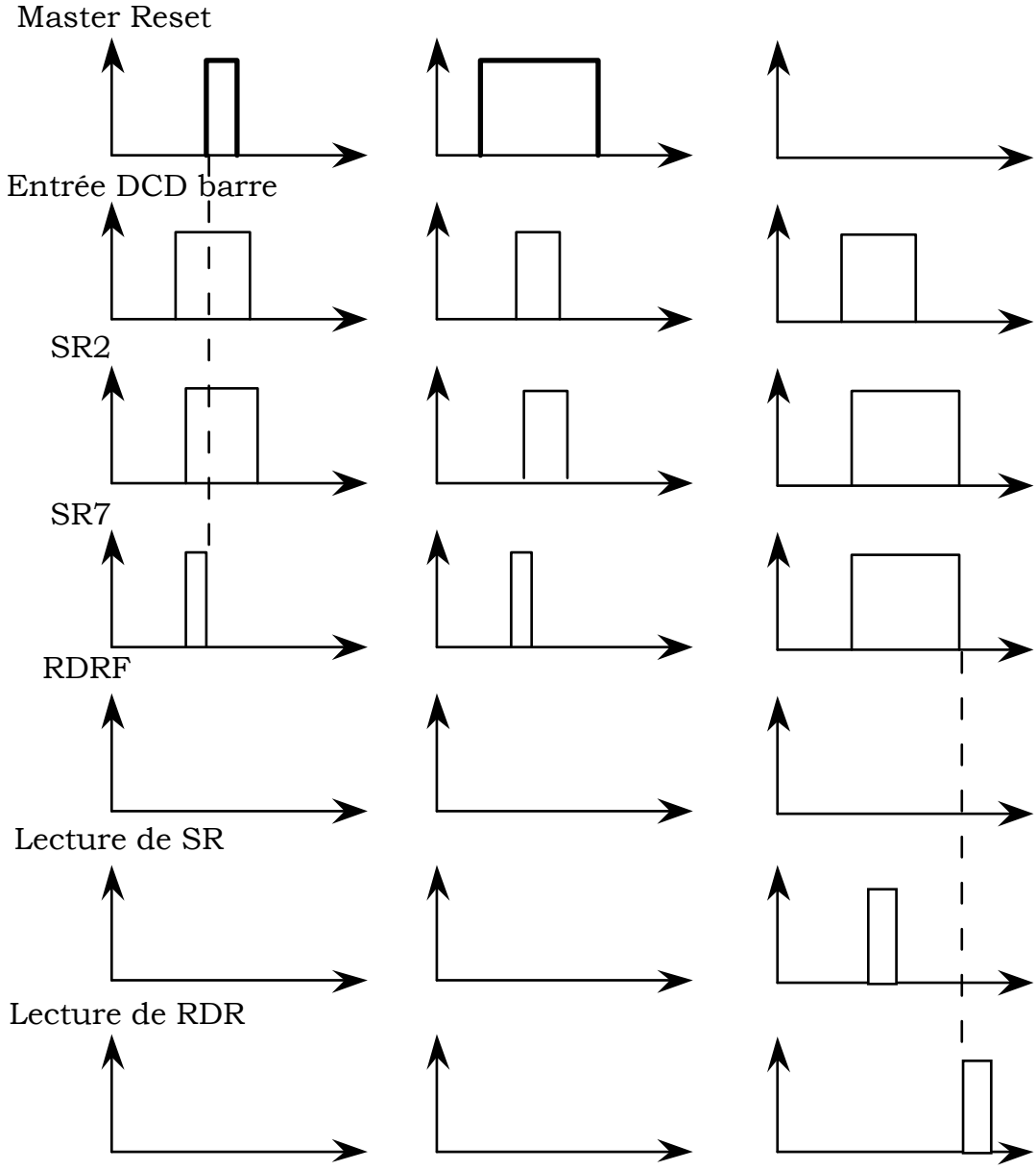
Pour un télétype, 300 b/s

Pour une imprimante série lente, 1200 b/s

Pour des systèmes plus rapides, 2400, 4800, 9600 ou 19200 b/s, ce qui dans ce dernier cas, requiert 52 micro-secondes par bit soit environ 500 microsecondes par mot. Durée pendant laquelle le microprocesseur peut travailler

Illustrations de la remise à 0 des bits SR2 et SR7.

Génération d'une interruption en réception si CR7=1 et RDRF=0. Remise à O des flags SR2 et SR7 par une lecture de SR suivie une lecture de RDR si l'entrée DCD barre revenu à "bas".

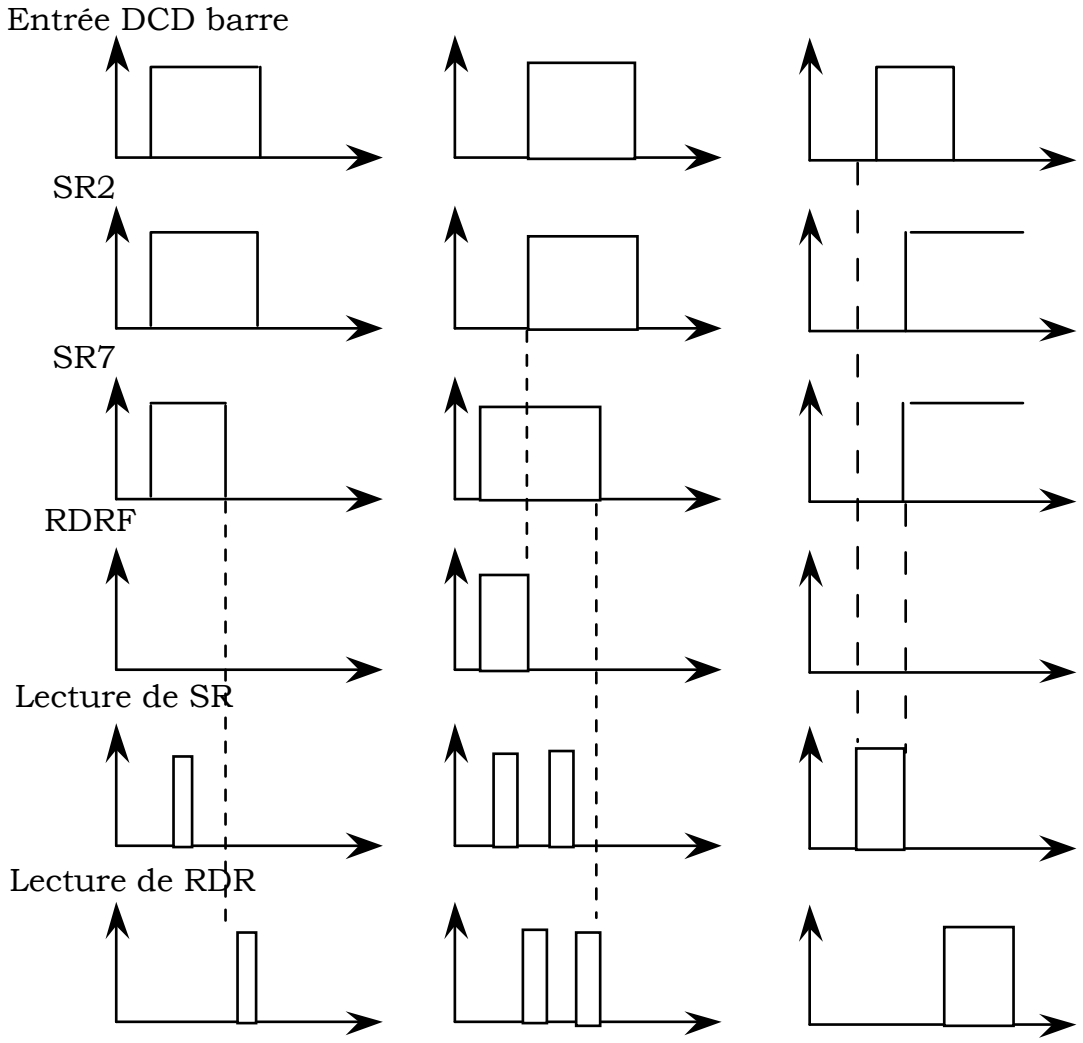


Le master Reset remet le bit SR7 à 0. Il avait été mis à 1 par la perte de la porteuse.

DCD barre est haut pendant un Master Reset. SR2 refléte l'état de DCD barre.

SR7 n'est pas affecté.

Après qu'une interruption soit apparue, suite à DCD barre haut, une lecture de RDR remet SR2 et SR7 à 0 si toutefois l'entrée DCD barre a retrouvé son niveau bas. Avec Master Reset toujours à "bas".



Après qu'une interruption soit apparue, suite à l'entrée DCD barre haut, une lecture de SR suivie d'une lecture de RDR remet SR7 à 0 mais le bit SR2 ne revient à 0 que lorsque la ligne DCD barre a retrouvé son niveau bas.

Une perte de la porteuse a été

enregistrée.
Précédemment les lectures de SR et RDR ont eu lieu suite à RDRF =1. Le bit SR7 reste à 1. Quand RDRF retombe, une deuxième lecture de SR et de RDR remet SR7 à 0. Là encore SR2 suit l'entrée DCD.
Si une transition positive de DCD barre se produit pendant une lecture de SR et RDR, elle n'est pas prise en compte avant le front descendant du signal de lecture.

ETUDE DU PIC 6828

rôle

Du point de vue de l'utilisateur, ce contrôleur comporte) 2 registres

On t(rouve le registre demande (R R)
et le registre de masque.

a - le registre de demande comporte 8 qui représente 8 entrées "demandes d'interruption"
INO à IN7.
ces entrées sont hiérarchisées de telle façon que IN 7. soit la plus prioritaire alors que INO est la moins prioritaire.

b - le registre de masque.

Permet la mise en place de la hiérarchie.
Il comprend le dont es combinaisons binaires indiquent les numéros des entrées.

Il inhibe la prise en compte des demandes d' de niveau inférieur à son contenu.

Fonctionnement :

- ce circuit ne permet qu'un seul mode de fonctionnement
- Les priorités sont fixés et déterminés à l'avance.

Une possède donc aucun registre de contrôle ni d'état.
Le registre de masque est le seul registre programmable.

Son initialisation est particulière !

Une instruction d'écriture à l'adresse 1111 1111 111X XXXO charge ce registre avec la valeur indiquée xxxx

Cette valeur varie de OOOO à 1OOO

avec OOOO indiquant : aucun niveau masqué
et 1OOO indiquant : tous niveaux d' masqués.

adresses correspondantes sont comprises dans l'intervalle suivant :
($ F F E O, $ F F F O)

La programmation de ce registre n'est pas fait le de donnée.
(n'importe quelle valeur à ce moment là).

Schéma :

Ai reliés au adresses up
Zi reliés au adresses de la mémoire(cteur transmis
-- par le contrôleur vers la mémoire)
INI signaux de demande d'I T en provenance de la périphérie.

le signal I R Q est la demande d'I T adressée au up

Niveau bas si une entrée INi , de niveau de priorité égal ou supérieur au masque est activée (bas)

Le 6828 doit être activé par $ F F F 8 et $ F F F 9
Une solution :
A15 A14 et A1O, A9, A8, A7, A6 et A5 décodés

Comme l'indique le schéma le contrôleur d'IT doit être inserré entre les A1 à Au du d'adresses up et celui de la mémoire ROM.

Ainsi suite à une demande d'I T, le up dépose sur le d'adresses le mot $F F F 8 qui en fait adresse le P I C lequel positionne les Z1, Z2, Z3, et Z4 qui font office d'adresse pour la mémoire.

Le groupe Z1, Z2, Z3 et Z4 formant un vecteur relatif à la demande d'I T intervenu.

A ce moment là, le d'adresses de la mémoire contient la valeur :

1111 1111 111Z4 Z3Z2Z10 car Ao = 0

Reste à mettre, dans cette case mémoire et la suivante l'octet haut et bas respectivement de l'adresse du sous-programme d'IT à exécuter.

Tableau des adresses générées :

Par ordre de décroissance du niveau de priorité

Remarque :

Le P I C possède une sortie STRECM qui permet de prolonger la période de l'horloge du up dans le cas où l'accès à la ROM est peu rapide.

Car par rapport au cycle du up l'adresse "fabriquée" n'est validé qu'après un temps de retard dû au PIC.

Il faut donc faire attention à la synchronisation !

chronogramme de traitement d'I T par le P I C :

Processus suivi :

Une entrée INJ passe à O
le contrôle active IRQ qui passe à O
Le processeur exécute les opérations préalables au traitement de l'instruction : sauvegarde, I = 1 etc.

Le up dépose $FFF8 sur le d'adresse.
Le contrôleur génère le vecteur Zi relatif à INj

Le microprocesseur lit le contenu de la deuxième case mémoire relative à INj (A_0 = 1)

Le up dépose $ F F F 9 sur le bus d'adresses.

Le contrôleur est adressé et le signal S T R E T C H est activé
Le contrôleur génère le vecteur Zi relatif à INj
Le microprocesseur lit le contenu de la deuxième case mémoire relative à INj (Ao = 1) et se branche à l'adresse ainsi obtenue.

le signal S T R E T C H retourne à l'état haut.

Et enfin ! le microprocesseur exécute le sous-programme relatif à l'entrée INj

Remarque :

Lorsque le contrôleur n'est pas adressé (adresse autre que le vecteur IRQ par ex), il laisse passer les d'adresses A4, A3, A2 et A1.

Ce circuit joue donc le rôle d'un sélecteur avec 8 entrées et 4 sorties - comme le montre le schéma suivant :