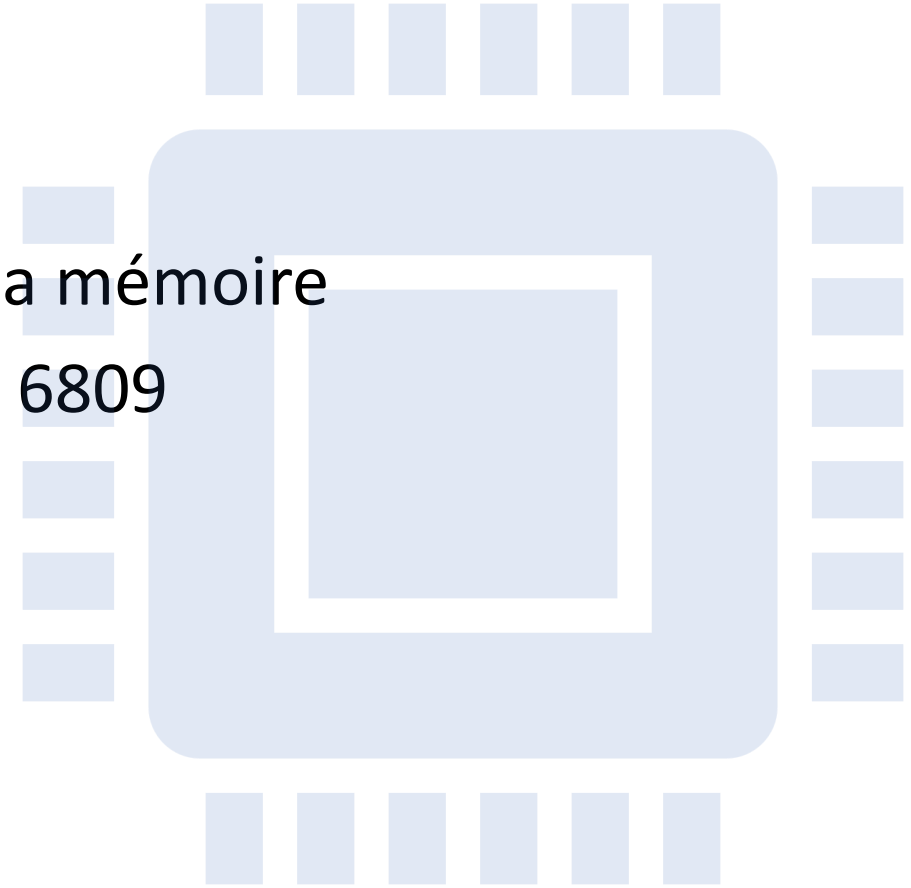
Architecture des Ordinateurs : Assembleur

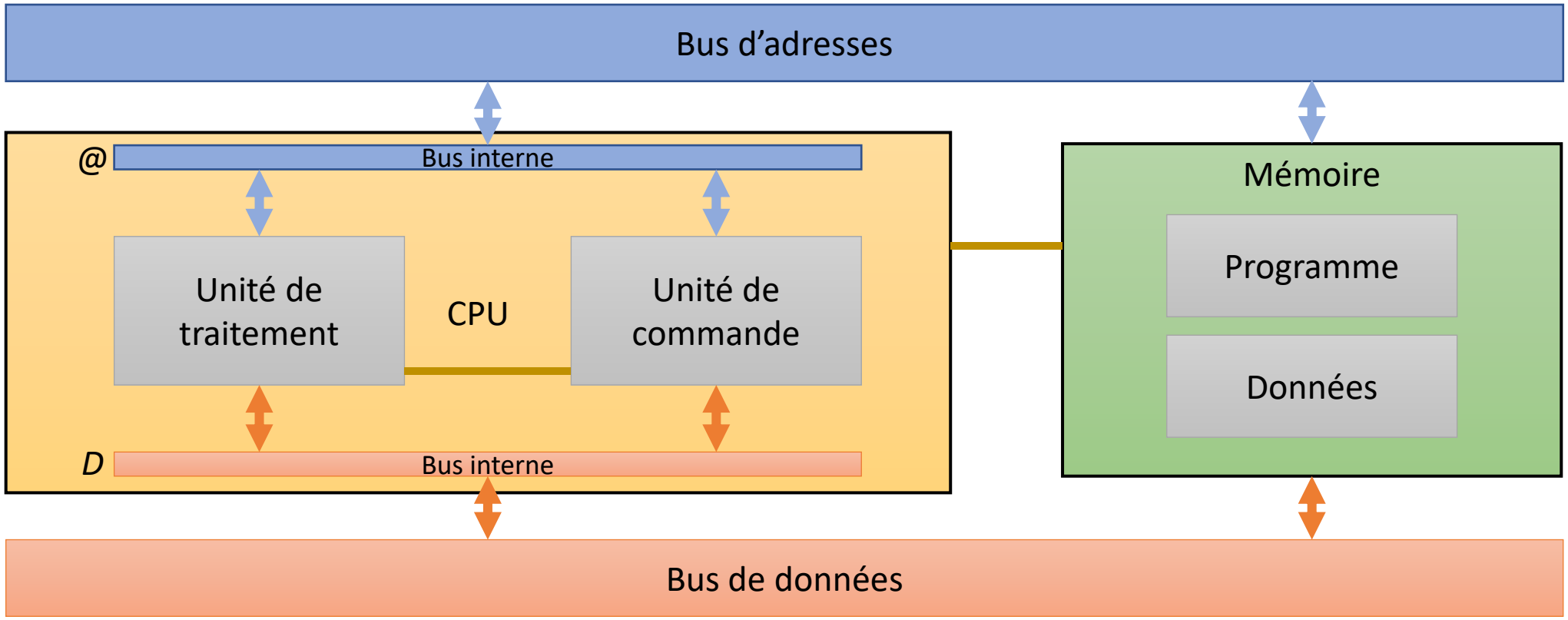
Plan

- Fonctionnement du processeur avec la mémoire
- Présentation du processeur Motorola 6809
- Assembleur Motorola 6809



Fonctionnement du processeur avec la mémoire

Architecture de base d'un CPU



L'Unité de commande

Elle permet de séquencer le déroulement des instructions.

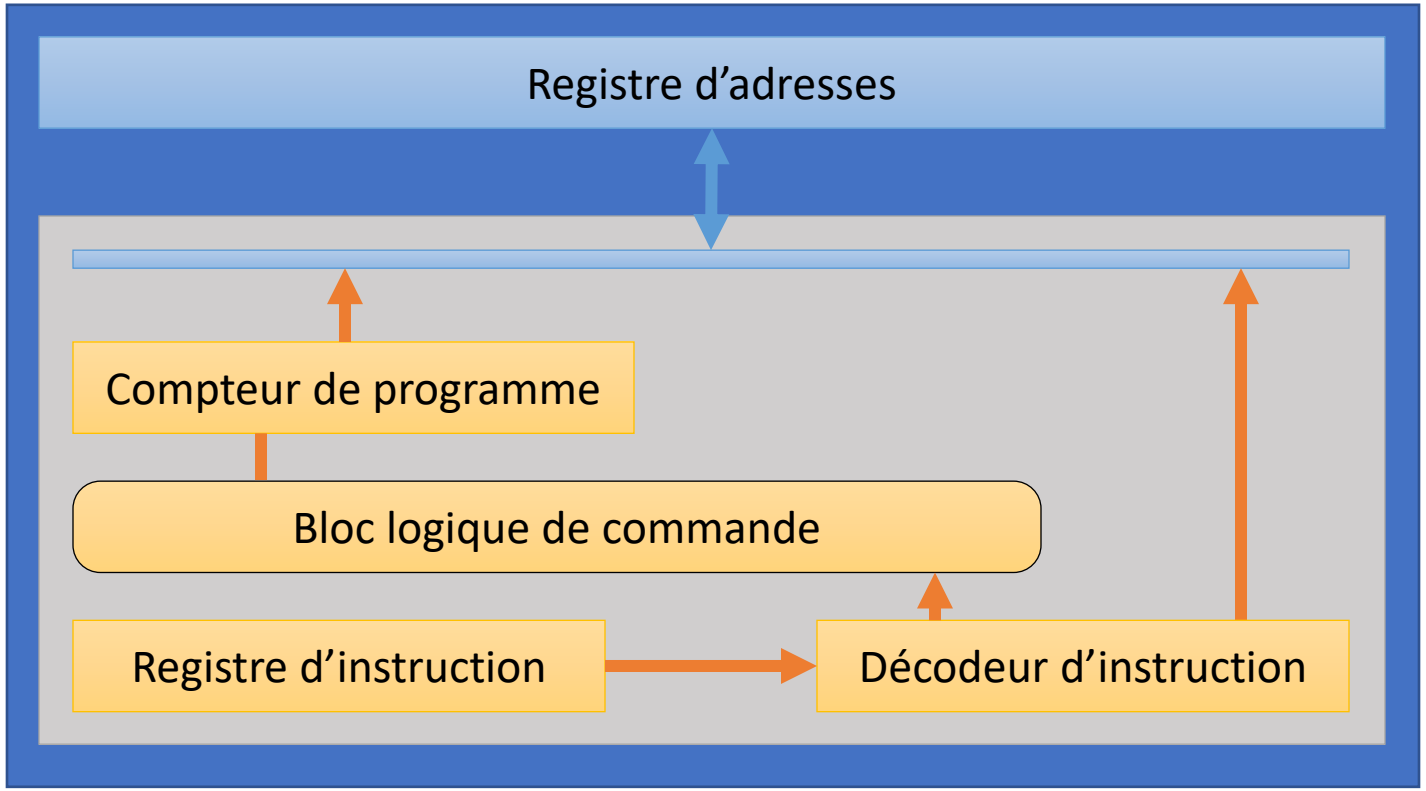
1. Elle recherche l'instruction en mémoire
2. La décode
3. L'envoie à l’unité de traitement pour exécution
4. Prépare l'instruction suivante

L'Unité de commande

L'Unité de commande est composée de :

- **Le compteur de programme** : constitue par un registre dont le contenu est initialisé avec l'adresse de la première instruction du programme. Il contient toujours l’adresse de l’instruction à exécuter.
- **Le registre d'instruction et le décodeur d'instruction** : chacune des instructions à exécuter est rangée dans le registre instruction puis est décodée par le décodeur d'instruction.
- **Bloc logique de commande (ou séquenceur)** : Il organise l’exécution des instructions au rythme d'une horloge.

L'Unité de commande



Unité de traitement

L’Unité de traitement assure l’exécution des instructions. Elle est composée de :

- L’Unité Arithmétique et Logique (UAL) est un circuit complexe qui assure les fonctions logiques ou arithmétiques.
- Le registre d’état est généralement composé de 8 bits à considérer individuellement. Chacun de ces bits est un indicateur dont l’état dépend du résultat de la dernière opération effectuée par l'UAL. On les appelle indicateur d’état ou flag. Dans un programme le résultat du test de leur état conditionne souvent le déroulement de la suite du programme. On peut citer par exemple les indicateurs de:
  - zéro (Zero : Z)
  - signe (Sign : S)
  - débordement (Overflow : OV ou V)
- Les accumulateurs sont des registres de travail qui servent à stocker un opérande au début d'une opération arithmétique et le résultat à la fin de l’opération.

Unité de traitement

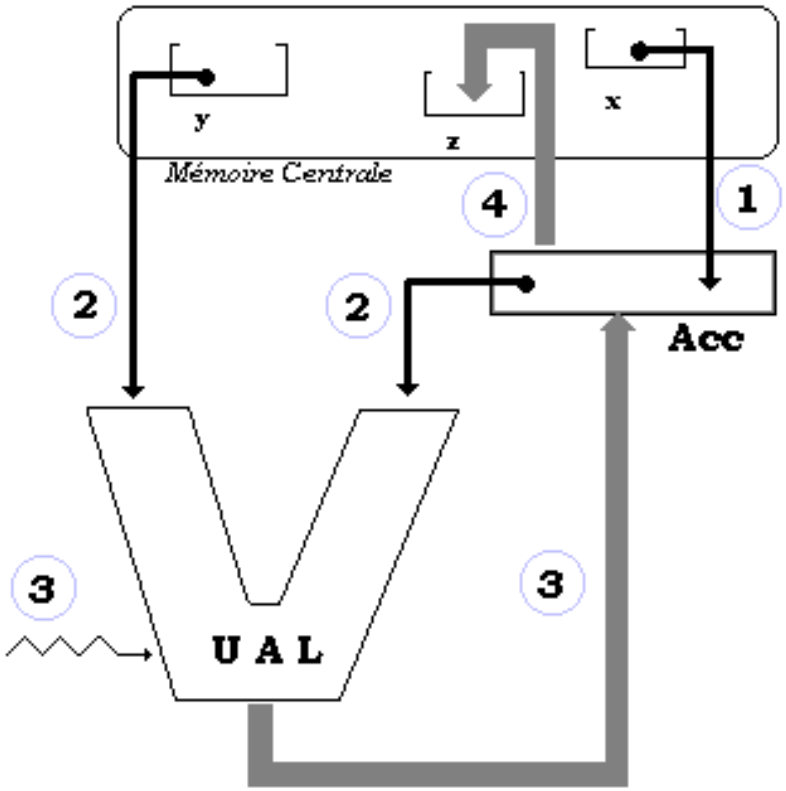
L’opération z = x + y , se décompose dans une telle machine fictivement en 3 opérations distinctes :

- LoadAcc x (chargement de l’accumulateur avec x : (1))
- Add y (préparation des opérandes x et y vers l’UAL : (2))

(Lancement commande de l’opération dans l'UAL : (3))

(transférer le résultat dans l'accumulateur :

- Store z (copie de l'accumulateur dans z : (4))



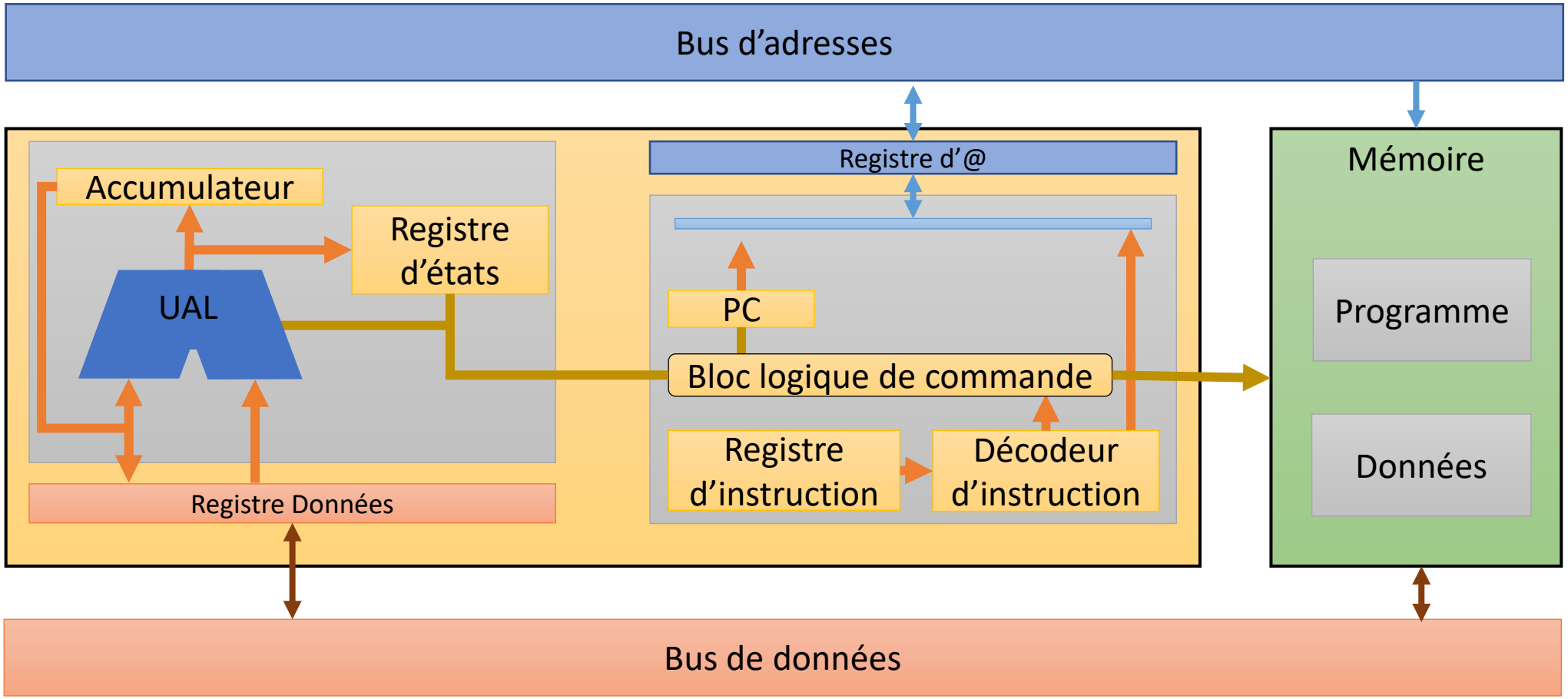
Les bus

TOUT EST RELIE AVEC 3 TYPES DE BUS

- **Le bus de données**: permet le transport des données ou les instructions du programme.
- **Le bus d'adresses**: permet d'acheminer les adresses des cases mémoires ou des ports d'E/S.
- **Le bus de commande**: permet de transmettre les ordres dans tout les sens :

  par exemple : Une lecture ou écriture dans une case mémoire.

Architecture de base d'un CPU



Le processeur

Un processeur est défini par :

- la largeur de ses registres internes de manipulation de donnees (8, 16, 32, 64, 128 bits);
- la cadence de son horloge exprimée en Hz;
- le nombre de noyaux de calcul (core) ;
- son jeu d' instructions (ISA en anglais, Instructions Set Architecture) dépendant de la famille (CISC, RISC, etc) ;

Les instructions

Définition

Le feu d'instructions décrit l’ensemble des opérations élémentaires que le microprocesseur pourra exécuter.

Types d'instruction

- **Transfert de données** pour charger ou sauver en mémoire, effectuer des transferts de registre a registre, etc.
- **Operations arithmétiques** : addition, soustraction, division, multiplication
- **Operations logiques** ET, AND, NAND, comparaison, test, etc.
- **Contrôle de séquence** : branchement, test, etc.

Les instructions

- Les instructions et leurs opérandes (paramètres) sont stockés en mémoire principale.
- La taille totale d'une instruction (nombre de bits nécessaires pour la représenter en mémoire) dépend du type d'instruction et aussi du type d’opérande.
- Chaque instruction est toujours codée sur un nombre entier d'octets afin de faciliter son décodage par le processeur.
- Une instruction est composée de deux champs :
  - Le code instruction, qui indique au processeur quelle instruction à réaliser.
  - Le champ opérande qui contient la donnée, ou la référence à une donnée en mémoire (son adresse).

Les instructions

Champ Code opération	Champ Code opérande

Format instruction

Mode d'adressage (exemples) :

Code opération	Valeur

Adressage immédiat

Code opération	Adresse de la valeur

Adressage direct

Les instructions

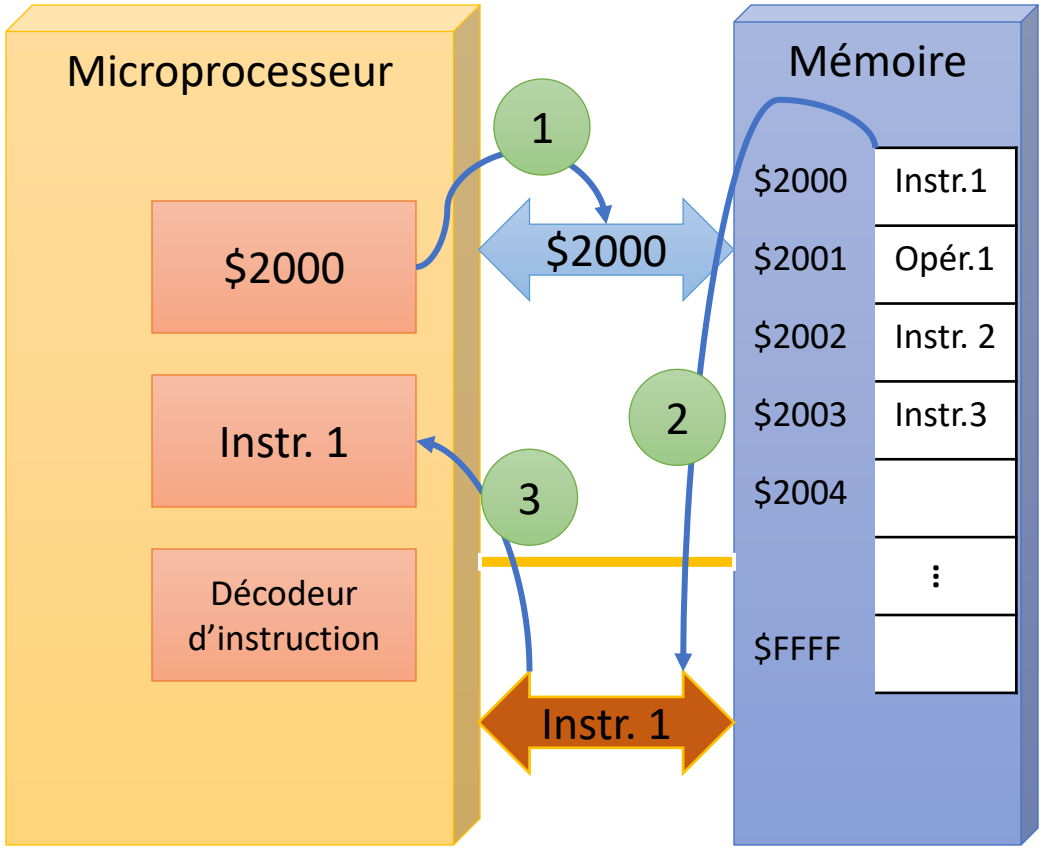
Le microprocesseur ne comprend qu'un certain nombre d'instructions qui sont codées en binaire. Le traitement d'une instruction peut être décomposé en trois phases:

1. Phase 1 : Recherche de l'instruction à traiter
2. Phase 2 : Décodage de l'instruction et recherche de l'opérande
3. Phase 3 : Exécution de l'instruction

Les instructions

Phase 1: Recherche de l'instruction à traiter (FETCH)

1. Le PC contient l'adresse de l'instruction suivante du programme. Cette valeur est placée sur le bus d'adresses par l'unité de commande qui émet un ordre de lecture.
2. Au bout d'un certain temps (temps d'accès à la mémoire), le contenu de la case mémoire sélectionnée est disponible sur le bus des données.
3. L'instruction est stockée dans le registre instruction du processeur.

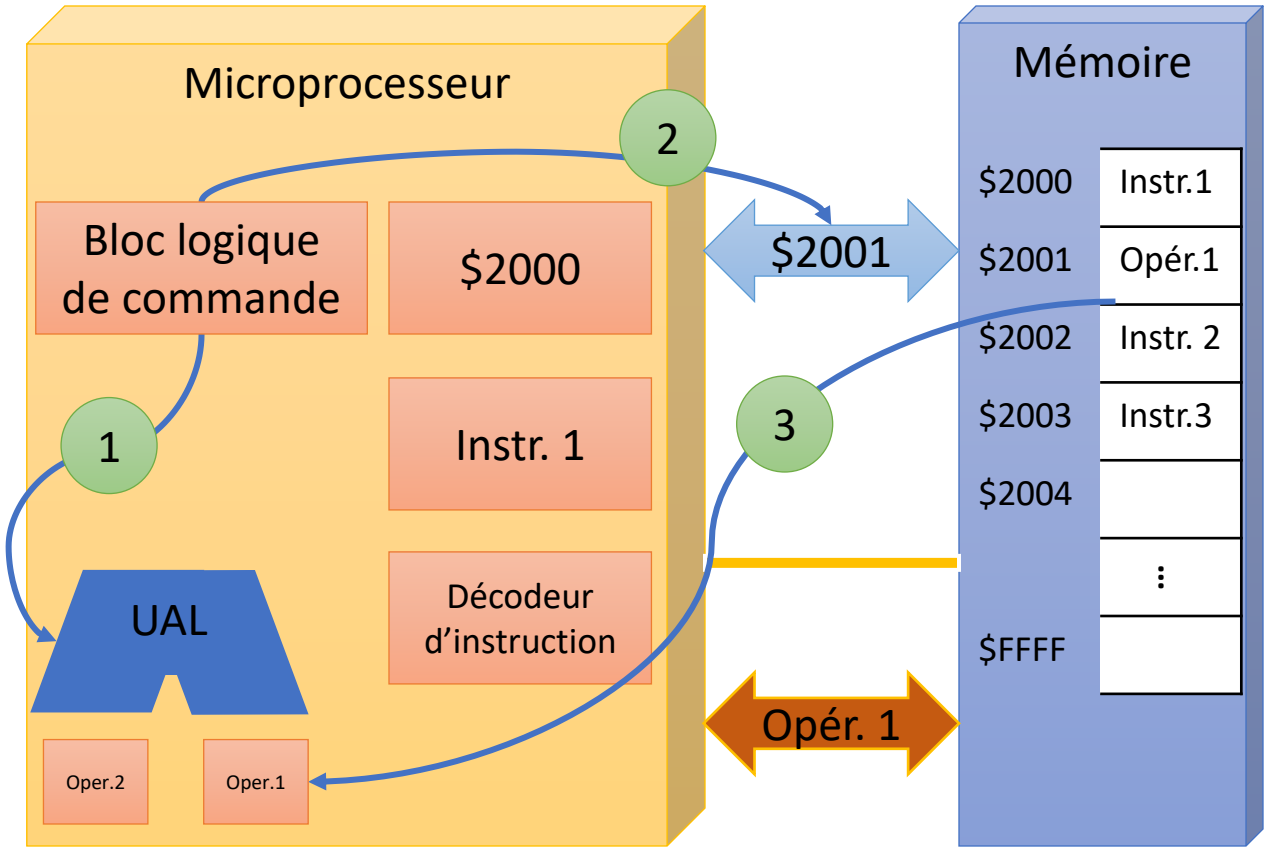


Les instructions

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

Le registre d'instruction contient l'instruction à exécuter

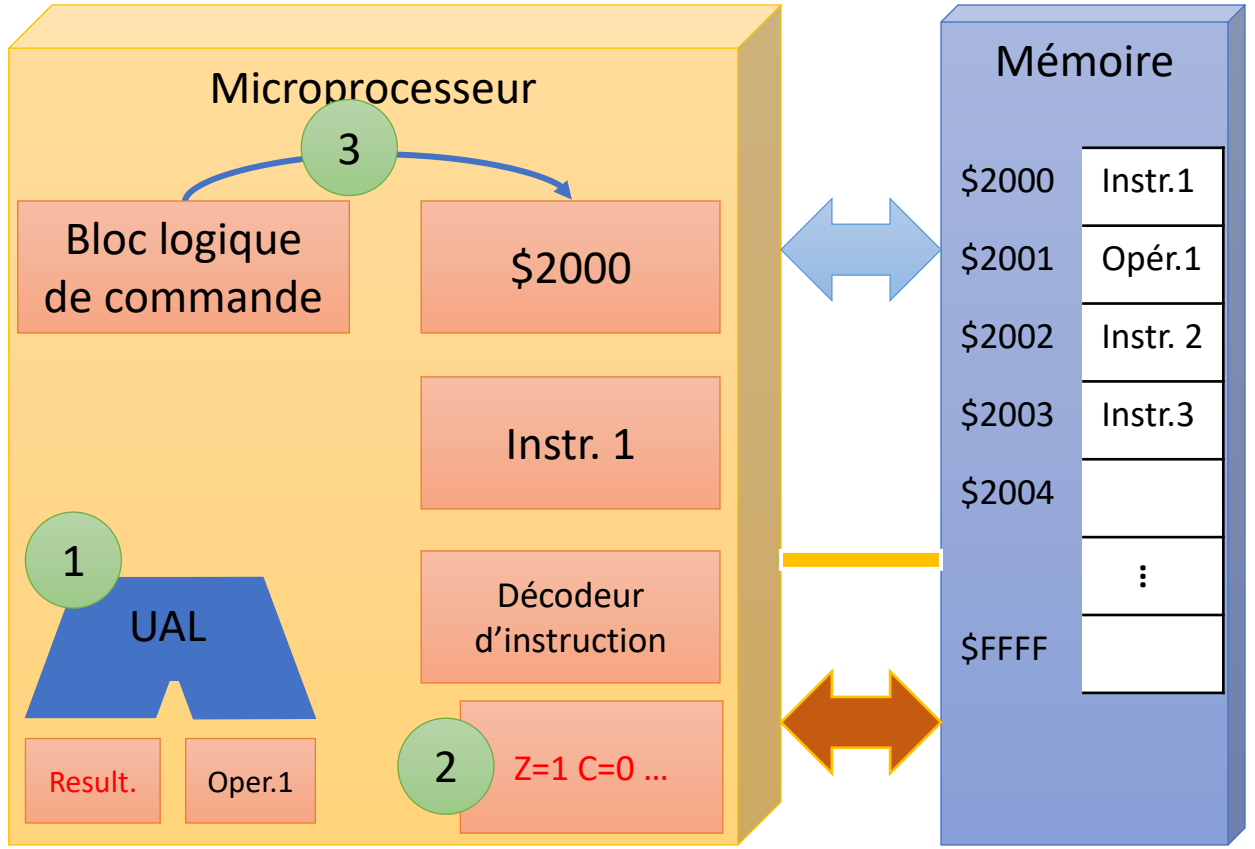
1. L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction (microprogramme).
2. Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur sur le bus de données.
3. L'opérande est stockée dans un registre.



Les instructions

Phase 3 : Exécution de l'instruction

1. Le microprogramme réalisant l'instruction est exécuté.
2. Les drapeaux sont positionnés (registre d'état).
3. L'unité de commande positionne le PC pour l'instruction suivante.



Présentation du processeur : Motorola 6809

Motorola 6809

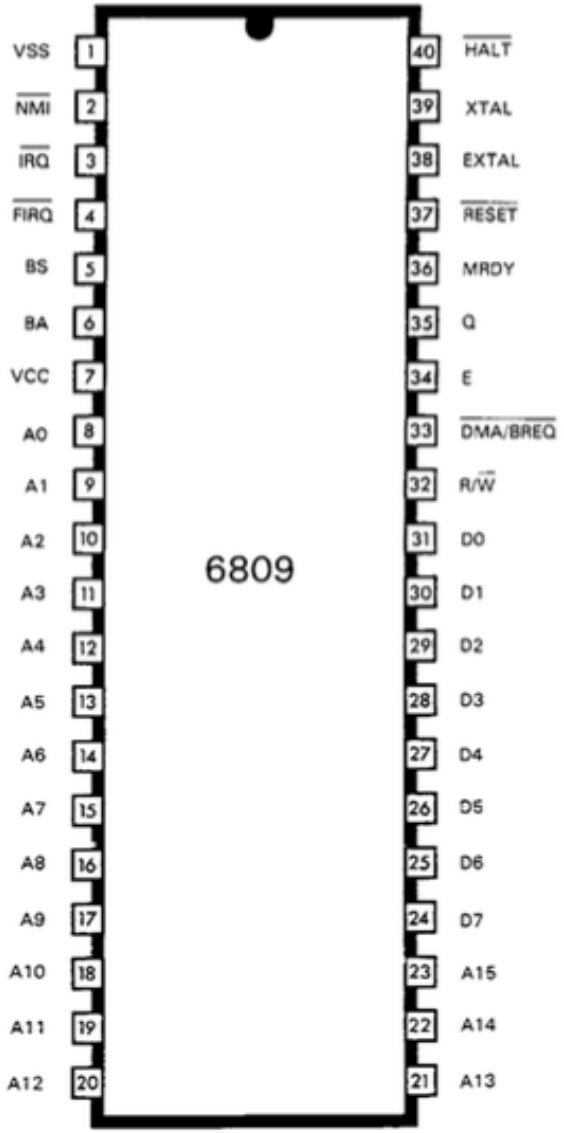
- Le 6800 est un microprocesseur 8 bits produit par Motorola et sorti peu de temps après l'Intel 8080.
- Il a 78 instructions. Il est le premier microprocesseur avec un registre d'index.
- Il se présente habituellement sous forme d'un boîtier 40 broches et contient 7000 transistors.
- Il a plusieurs descendants : 6809, beaucoup de microcontrôleurs à partir de l'architecture du 6800 : 6805, les 6807, le 68HC11 et le 68HC12.

Motorola 6809

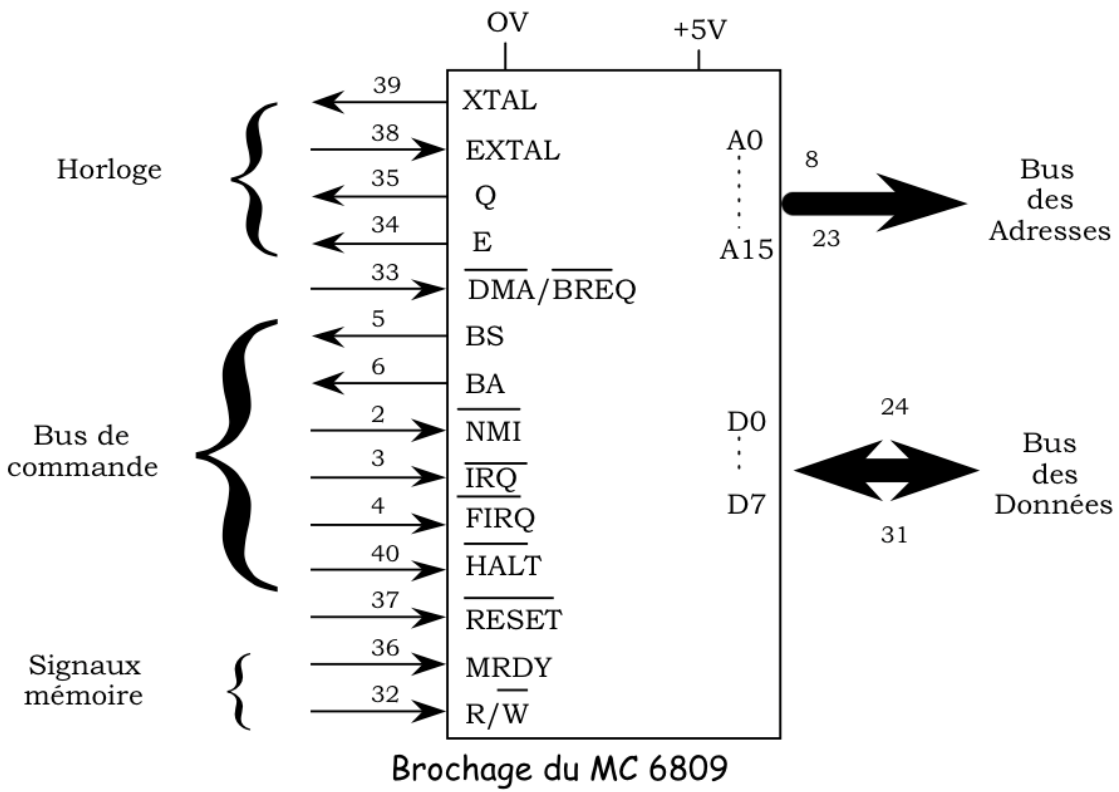
Le motorola 6809 comporte :

- Un bus Data sur 8 bits
- Un bus d'adresse sur 16 bits permettant un adressage mémoire de 64 KiloBits.
- Deux accumulateurs de 8 bits "A" et "B" transformables en 1 accumulateur de 16 bits "D"
- Deux registres d'index de 16 bits "X" et "Y"
- Deux registres pointeur de pile "U" et "S"
- Un pointeur de page "DP" de 8 bits servant à l'adressage direct de la mémoire
- Un registre d'état "CC" sur 8 bits
- Le compteur de programme sur 16 bits "PC" (CO) pointant toujours sur l'adresse que le microprocesseur doit exécuter.

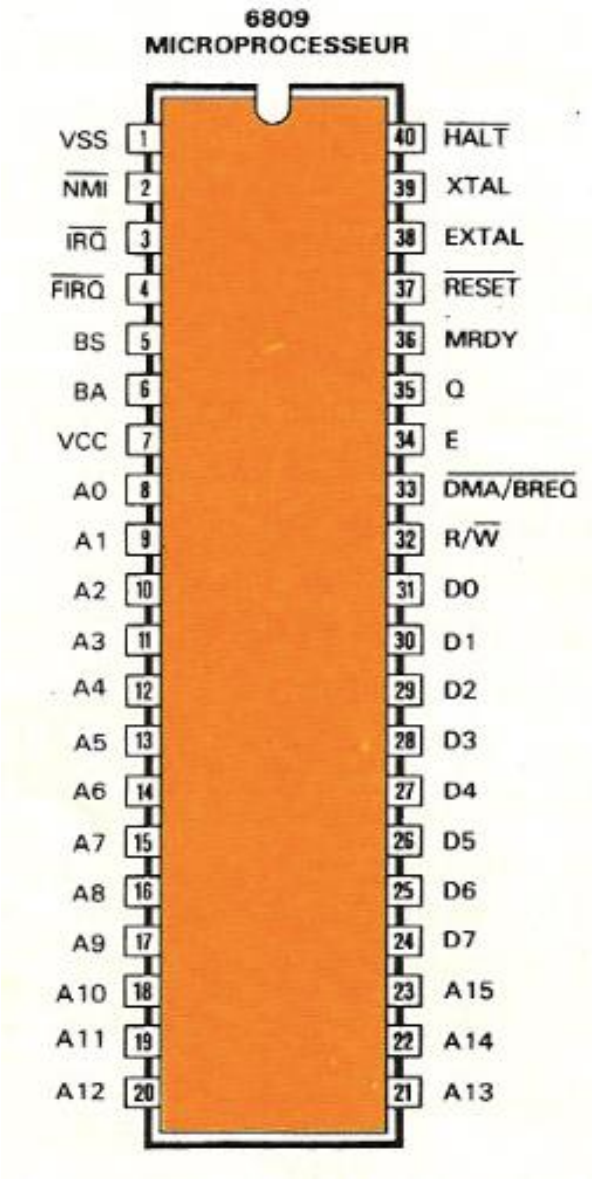
Les broches du processeur



Les broches du processeur



Brochage du MC 6809

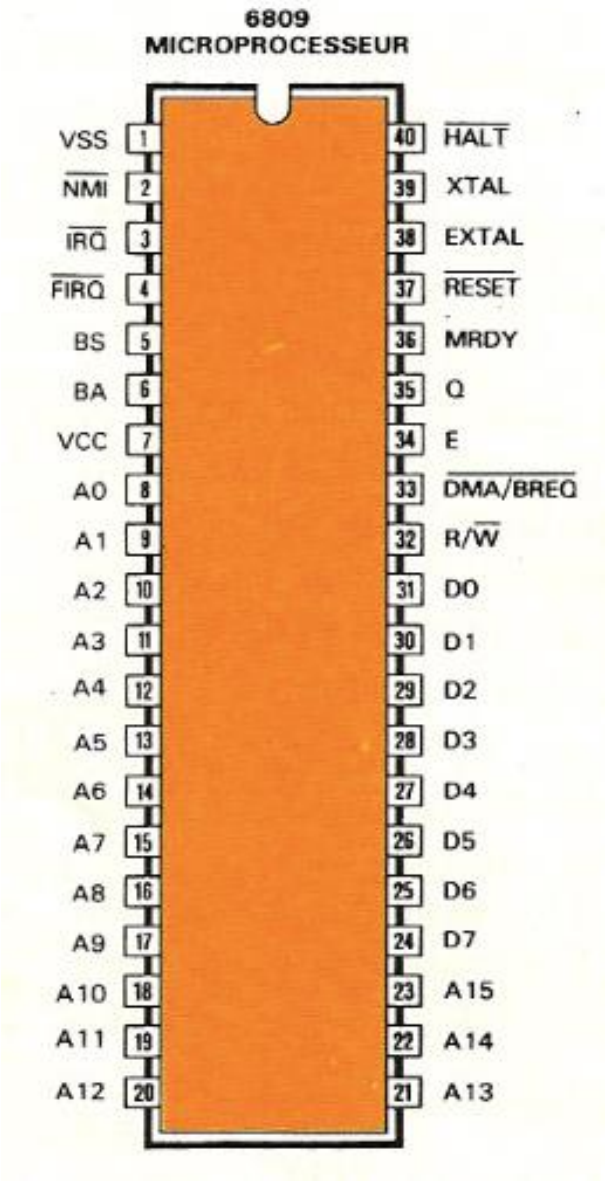


Les broches du processeur

- Le microprocesseur comporte 40 broches et il est alimenté en 5V uniquement.

- Il possède 3 bus indépendants :

  - Bus de données sur 8 bits (D0 à D7).
  - Bus d'adresse sur 16 bits (A0 à A15).
  - Bus de contrôle de 10 bits.



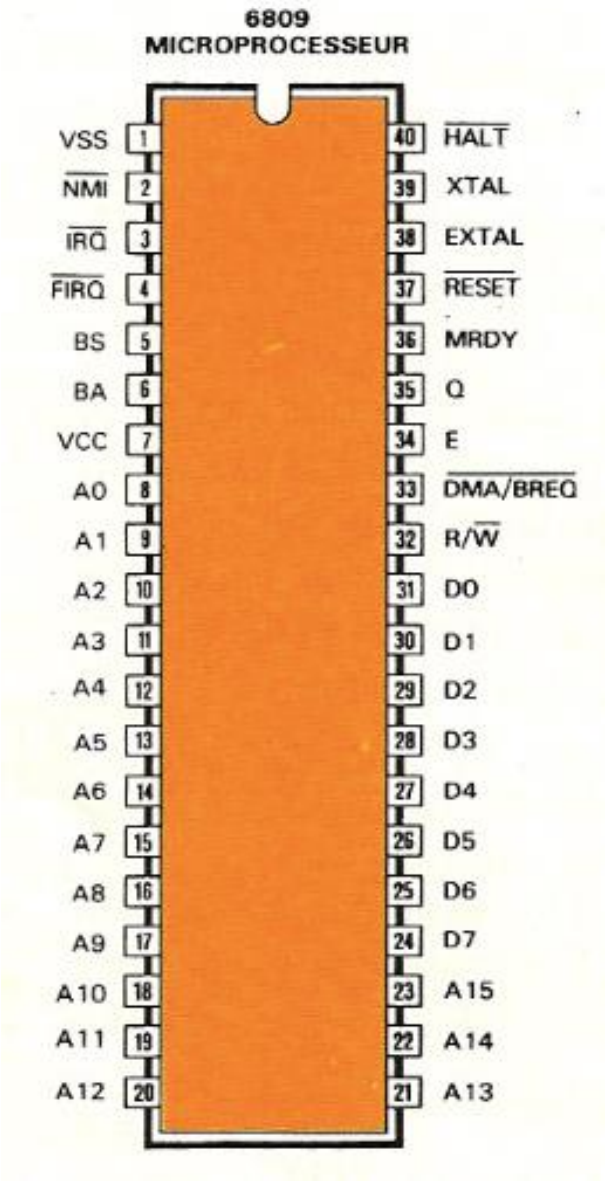
Les broches du processeur

Le signal lecture-écriture R/W (read/write) détermine le sens du transfert des données.

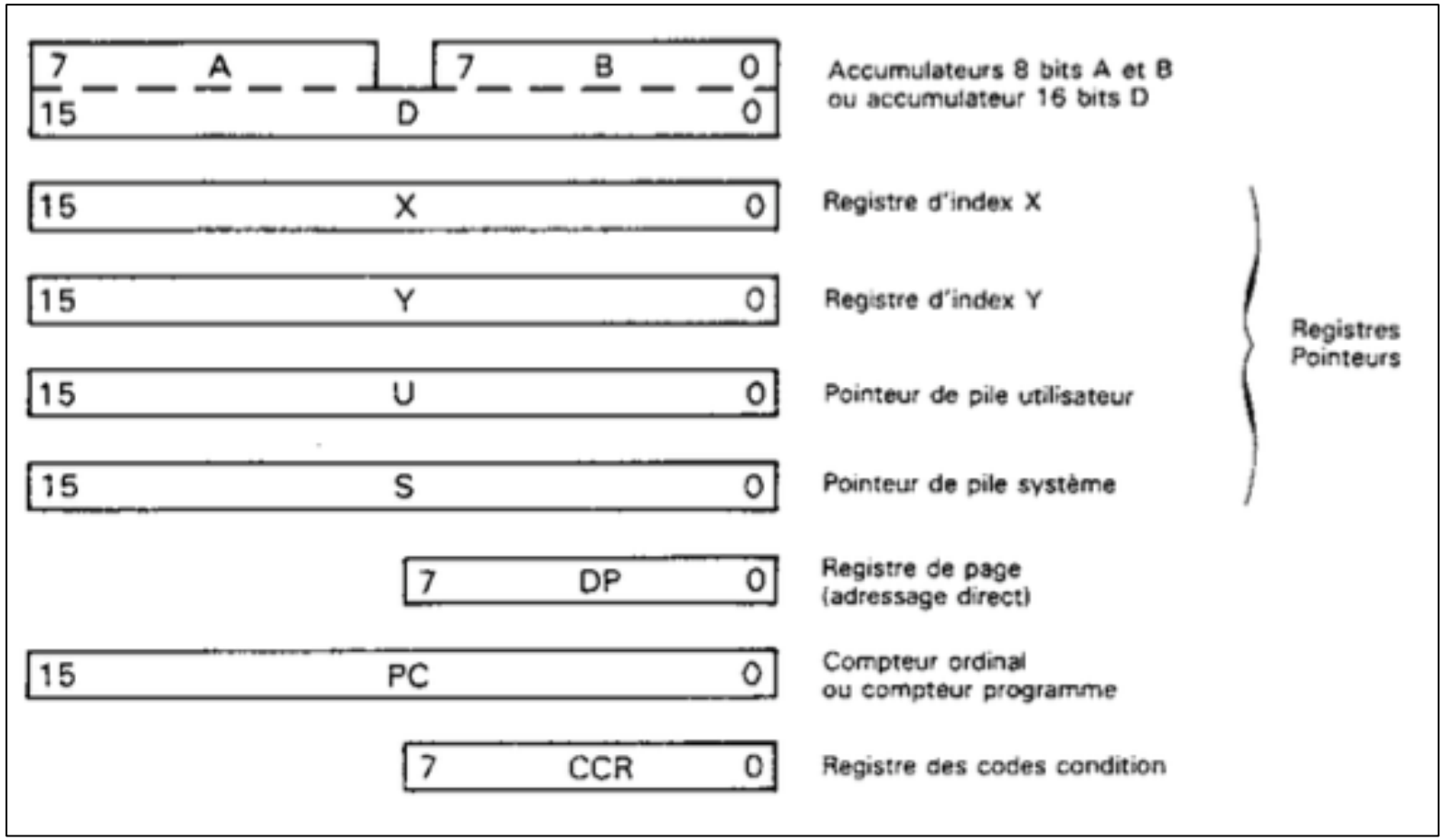
- Lorsque R/W = 1, le 6809 fait une lecture.
- Lorsque R/W = 0, le 6809 fait une écriture sur le bus de données.

Les lignes d'état du bus BA (Bus Available) et BS (Bus State) renseignent les périphériques du 6809 sur la disponibilité des bus de données et d'adresse.

- BA = BS = 0 : les bus de données et d'adresse sont disponibles
- BA = 0, BS = 1 : le 6809 vient de recevoir une interruption.
- BA = 1, BS = 0 : le 6809 vient de rencontrer dans le programme l'instruction SYNC.
- BA = BS = 1 : Interruption par le signal HALT sur 6809

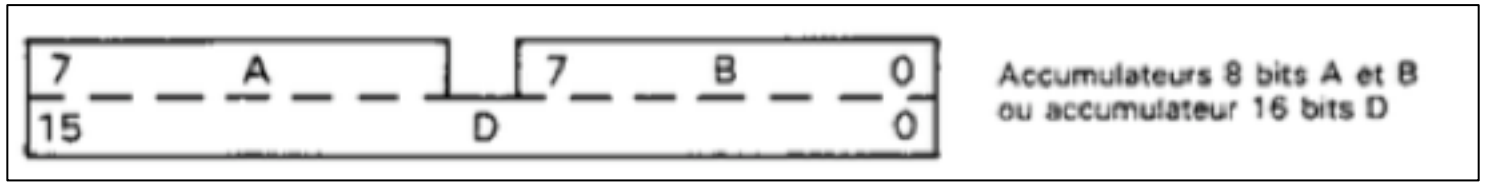


Registres


7 A 7 B 0
15 D 0
Accumulateurs 8 bits A et B
ou accumulateur 16 bits D
15 X 0
Registre d'index X
15 Y 0
Registre d'index Y
15 U 0
Pointeur de pile utilisateur
15 S 0
Pointeur de pile système
Registres
Pointeurs
7 DP 0
Registre de page
(adressage direct)
15 PC 0
Compteur ordinal
ou compteur programme
7 CCR 0
Registre des codes condition

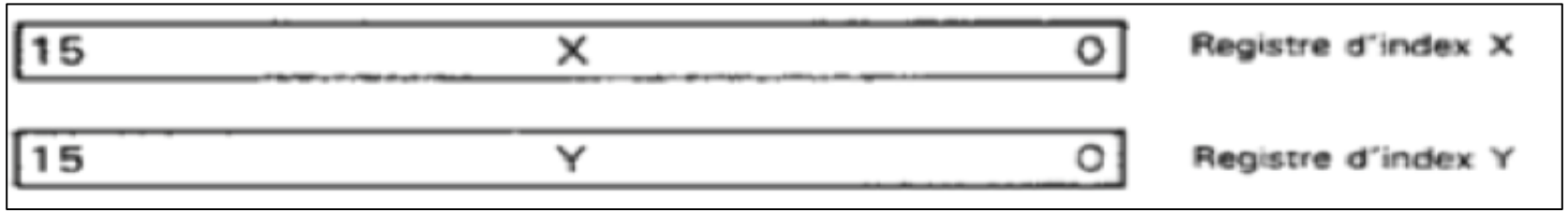
Registres

- Le 6809 possède plusieurs registres utilisés pour la manipulation et le traitement des données :
- 2 accumulateurs de 8 bits A et B transformables en 1 accumulateur de 16 bits, D.
- Ils sont utilisés pour les instructions arithmétiques, logiques et de chargement de données 8 bits (ou 16 bits) en mémoire. Ils sont pour cela entièrement identiques.
- L'accumulateur D est en fait la concaténation de A et B, le registre A représentant les poids forts (bits 8 à 15) et B les poids faibles (bits 0 à 7).



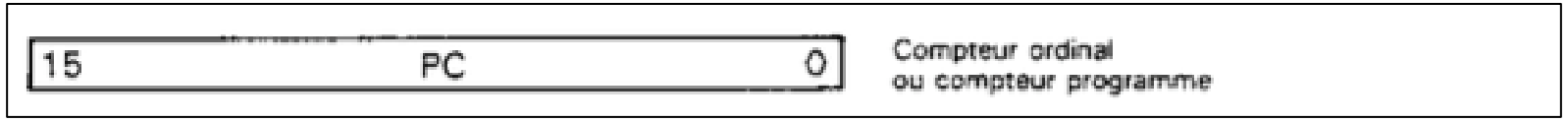
Registres

2 registres d'index X et Y, sur 16 bits. Ils permettent d'adresser tout l'espace mémoire (un programme peut accéder à n'importe quelle adresse mémoire dans un espace de 64 kilooctets (2^16 octets), ce qui correspond à l'adresse maximale que peut gérer un processeur doté de registres de 16 bits.) avec en plus la capacité d'être pré-décrémenté ou post-incrémenté pour faciliter le traitement de variables en tables (le registre peut être modifié avant ou après l'accès à la mémoire).



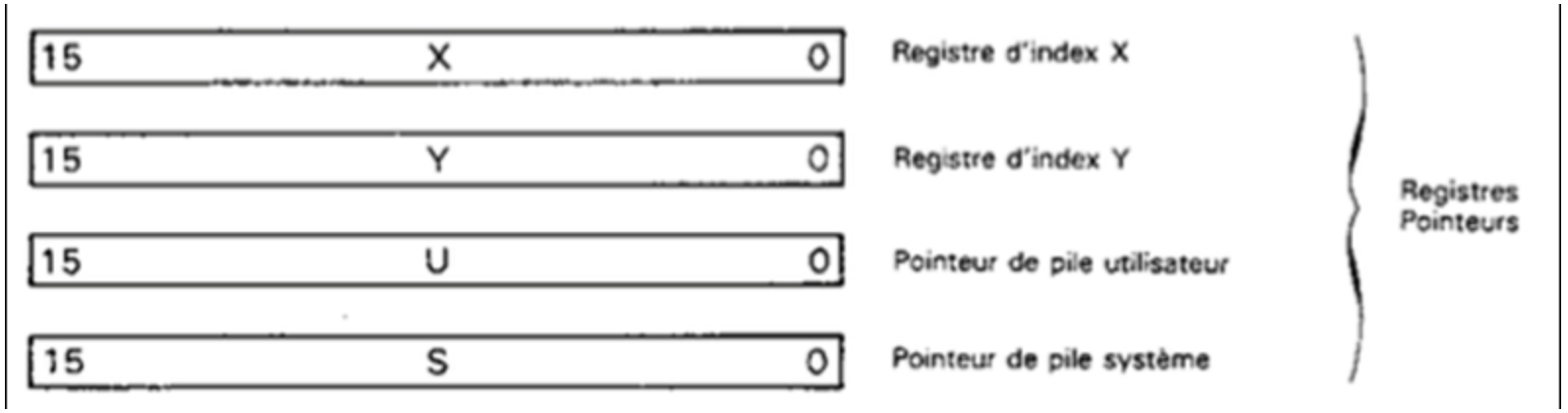
Registres

- Un compteur programme PC (Program Counter), sur 16 bits, pointe l'adresse mémoire à laquelle le 6809 doit exécuter une instruction.
- un registre spécial utilisé pour stocker l'adresse mémoire de l'instruction suivante à exécuter.



Registres

- 2 registres pointeurs de pile U et S.
  - U (User) est utilisé uniquement par le programmeur.
  - S (System) par le système pour les opérations de sauvegarde en cas d'interruption ou de saut à un sous-programme (Adresse de retour).
- La pile est un emplacement ou le microprocesseur sauvegarde le contenu de ses registres internes pendant un certain temps.
- Elles opèrent en mode dernier entré-premier sorti (LIFO : Last In - First Out).
- Le pointeur de pile pointe vers la dernière entée effective de la pile.

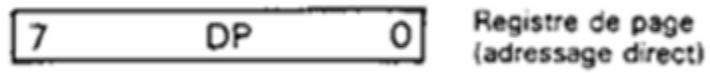


Registres

- Un registre de "page mémoire" DP (Direct Page), sur 8 bits, est utilisé pour adresser des pages en mémoire.

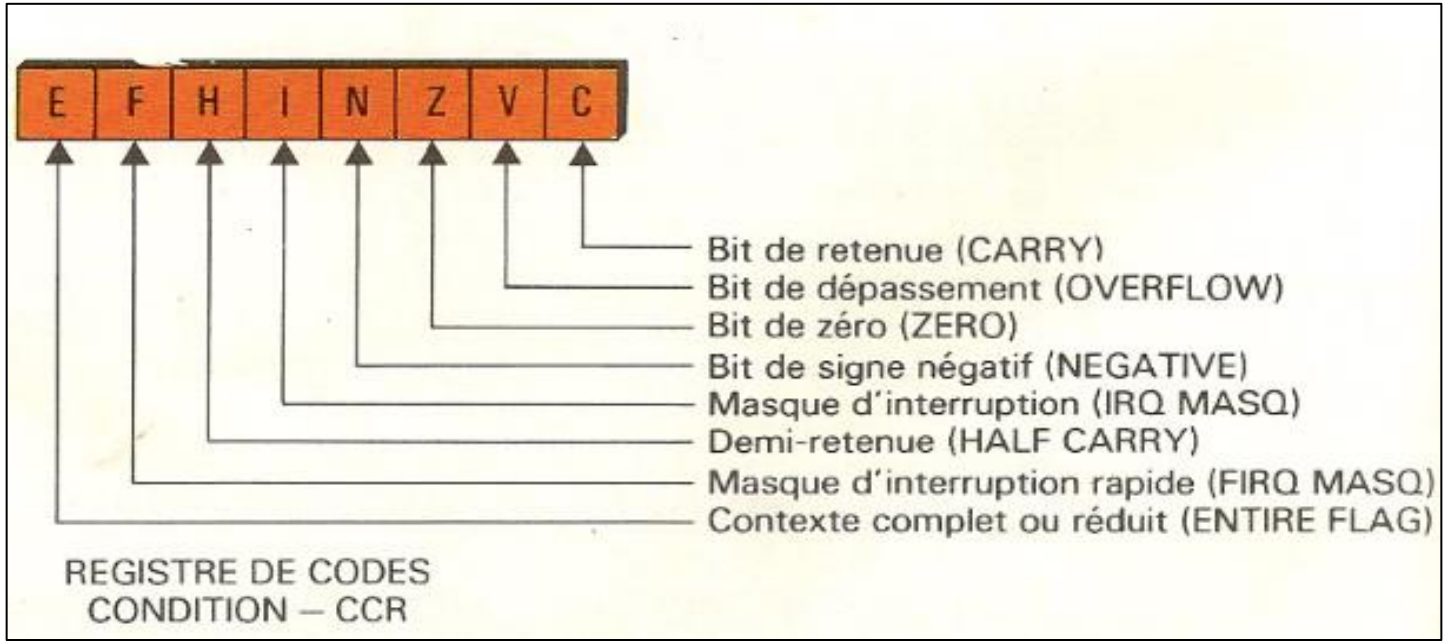
- Une page est un bloc de 256 bits. Aussi les emplacements mémoire 0 à 255 forment la page 0 de la mémoire. Le 6809 possède un bus d'adresse de 16 bits, cela donne 256 pages.

- Le registre DP permet une exécution plus rapide des programmes. Il est automatiquement remis à 00 par un RESET.



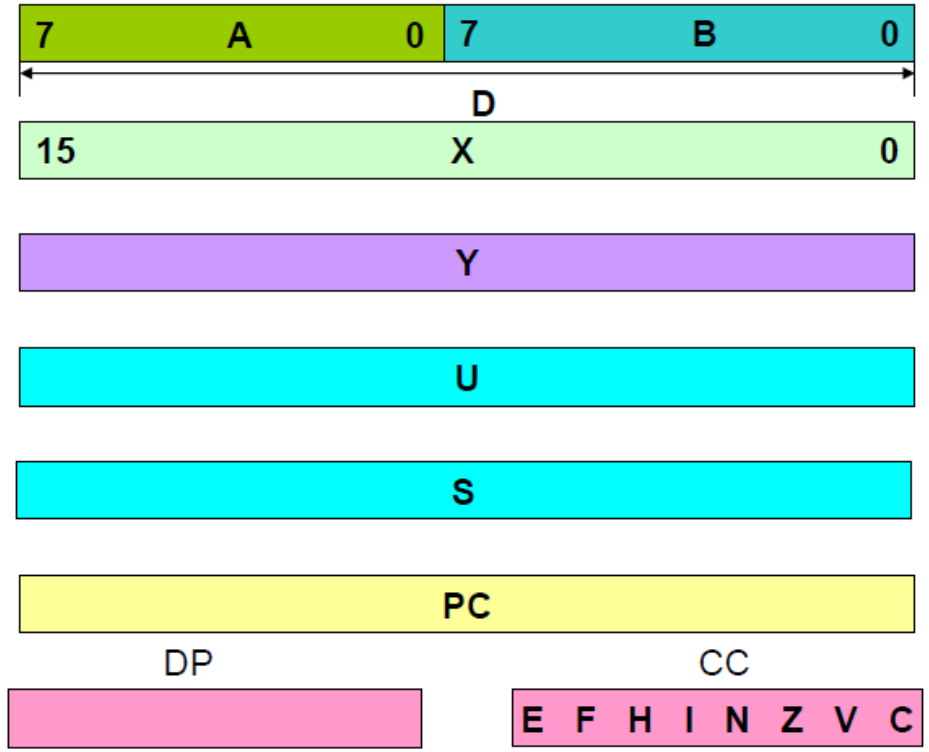
Registres

- Un registre "codes conditions" CC, affichant en permanence l'état du 6809 résultant d'une instruction. Le contenu du registre d'état est essentiellement conditionné par l'UAL.

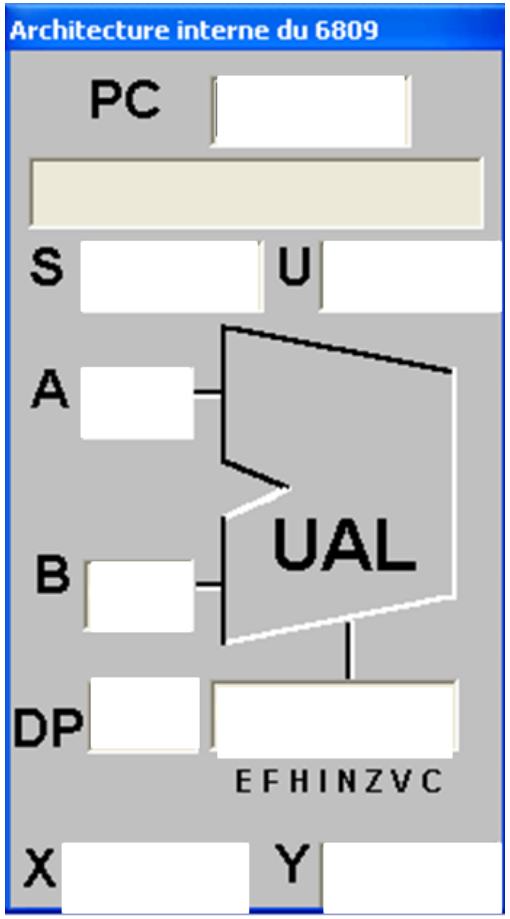


Registres

- Deux accumulateurs A et B
- Deux registres indexe X,Y
- Deux registres de pile : U,S
- Un compteur ordinal PC
- Un registre d'état CC et un DP (8bits)



Registres



Assembleur : Motorola 6809

Assembleur : Motorola 6809

- **Introduction**
- **Structure d’une instruction en assembleur**
- **Les principales instructions**
- **Les modes d’adressage en assembleur**

Introduction

- Le programmeur a le choix entre différents langages (assembleur, Java, Rust, C, etc.). La machine ne comprend que le langage machine (i.e., instructions binaires) !
- L'assembleur (i.e., langage d'assemblage) est le premier langage non binaire accessible au programmeur
- Code mnémoniques et symboles
- Permet d'exploiter au maximum les ressources de la machine
- Dépend de la machine : Chaque fabricant voire chaque processeur a son propre Assembleur

Introduction

- Le langage assembleur ou assembleur est le langage de programmation.
- C'est une version lisible par un humain du langage machine, obtenu en remplaçant les valeurs entières du langage machine par des mnémoniques (instruction du langage assembleur).
- Pour un même langage machine, il peut exister différents langages assembleur : variation sur la syntaxe.
- assembleur : programme qui transforme du langage assembleur en langage machine.

Structure d’une instruction en assembleur

- Le langage machine est le langage directement interprétable par le processeur.
- Le langage est défini par un ensemble d'instructions que le processeur exécute directement
- Chaque instruction correspond à un nombre (codé selon le cas sur un octet. un mot de 16 bits, ... : le format de l'instruction) et se décompose en:
  - une partie codant l'opération à exécuter appelé opcode ou code opération
  - une partie pour les opérandes

Code opération	Opérandes

Structure d'une instruction en assembleur

- Un programme en langage machine est une suite de mots codant opérations et opérandes

Adresse	Programme
0x2024	00F1
0x2026	00AA
0x2028	00F1
0x202A	0B28
0x202C	00F1
0x202E	0C91
0x2030	0001

- Chaque processeur possède son propre langage machine.

Structure d’une instruction en assembleur

- Le jeu d'instructions est l'ensemble des opérations élémentaires qu'un processeur peut accomplir.
- Le type de jeu d'instructions d'un processeur détermine son architecture.
- Deux types d'architectures

  RISC (Reduced Instruction Set Computer) PowerPC, MIPS, Sparc

  CISC (Complex Instruction Set Computer) Pentium

Structure d’une instruction en assembleur

Cycle d'exécution d'une instruction

1. Récupérer (en mémoire) l'instruction à exécuter : RI ← Mémoire[PC] L'instruction à exécuter est présente en mémoire à l'adresse contenue dans le compteur de programme PC et est placé dans le registre d'instruction RI.
2. Le compteur de programme est incrémenté : PC ← PC + 4, Par défaut, la prochaine instruction à exécuter est la suivante en mémoire (sauf si l'instruction est un saut)
3. L instruction est décodée . On identifie les opérations qui vont devoir être réalisées pour exécuter l'instruction
4. L'instruction est exécutée : elle peut modifier les registres (opérations arithmétiques - lecture en mémoire). la mémoire (écriture). le registre PC (instructions de saut)

Structure d’une instruction en assembleur

Langage C	Assembleur M6809	Langage Machine : Circuits M6809
var1=var2+1 Printf ("Résultat=%d",var1)	LDA var2 ADDA # 1 STA var1 JSR impression (impression identifie un sous programme)	1011 0110 0010 0000 0000 0000 1011 1011 0000 0001 1011 0110 0010 0000 0000 0001 1011 0011 0000 0000 0000 0000

Structure d’une instruction en assembleur

- **Une instruction est composée de deux parties distinctes et consécutives :**
  - **Un opérateur (obligatoire) ;**
  - **Un ou plusieurs opérandes (facultatif).**

Opérateur	Opérandes

Les principales instructions

- Chargement de registre : **LDA, LDB**

Exemple : LDA $0043 => charger l'accumulateur A par le contenu de l'adresse mémoire $0043. Le symbole $ indique que c'est en hexadécimal.

- Chargement de la mémoire : **STA, STB**

Exemple : STA $0043 => charger l'adresse mémoire $0043 par le contenu de l'accumulateur A

- Echanger le contenu des registres: **EXG R1, R2**

Exemple : EXG A,B => Echanger le contenu des deux accumulateurs A et B

- Transférer le contenu d'un registre : **TFR R1, R2**

Exemple : TFR A,B => Transfère le contenu de l'accumulateur A vers l'accumulateur B.

Exercices

1. Écrire un programme en M6809 qui charge le contenu de l'adresse mémoire $0040 dans l'accumulateur A, échange le contenu de A avec celui de B, puis stocke le contenu de B dans l'adresse mémoire $0041.
2. Écrire un programme en M6809 qui charge le contenu de l'adresse mémoire $0050 dans l'accumulateur A, transfère le contenu de A dans B, puis charge le contenu de l'adresse mémoire $0051 dans l'accumulateur A et stocke le résultat dans l'adresse mémoire $0052.

Les principales instructions

- Opération d'Addition: **ADDA, ADDB**

ADDA $0034 => Additionner le contenu de l'accumulateur A avec le contenu de la case mémoire d'adresse $0034.

- Opération de soustraction : **SUBA, SUBB**

SUBA $0043 => soustraire le contenu de l'accumulateur A du contenu de la case mémoire d'adresse $0034

- Comparaison: **CMPA, CMPB**

CMPA $0043 => Comparer le contenu de l'accumulateur A avec le contenu de l'adresse mémoire

- Opération de Multiplication : **MUL**

MUL => Multiplie le contenu de A et B. le résultat est mis dans D.

Exercices

1. Écrire un programme en M6809 qui charge le contenu de l'adresse mémoire $0034 dans l'accumulateur A, ajoute 10 à ce contenu et stocke le résultat dans l'adresse mémoire $0035.
2. Charger la valeur $0A dans l'accumulateur A, puis ajouter le contenu de la case mémoire d'adresse $0020 et stocker le résultat dans l'accumulateur B. Ensuite, soustraire le contenu de la case mémoire d'adresse $0021 du contenu de l'accumulateur B et stocker le résultat dans l'accumulateur $0022.
3. Transférer le contenu de l'accumulateur A vers l'accumulateur B, puis ajouter le contenu de la case mémoire d'adresse $0010 au contenu de l'accumulateur B et stocker le résultat dans la case mémoire d'adresse $0030.
4. Multiplier le contenu de l'accumulateur A avec le contenu de l'accumulateur B, puis stocker le résultat dans la case mémoire d'adresse $0080.

Les principales instructions

- **LDX, LDY** => charger les registres X ou Y par le contenu de l'adresse mémoire mentionnée

- **STX/STY** => Charger le contenu de l'adresse mémoire mentionnée dans les registres X ou Y

- **ABX** => Ajouter le contenu de l'accumulateur B à X

- **CMPX, CMPY** => Comparer X ou Y avec le contenu de la mémoire

Exercices

- Chargez le contenu de l'adresse mémoire $0078 dans A, puis transférez le contenu de A à B, puis chargez le contenu de B à X stockez le résultat dans l'adresse mémoire $0080.
- Chargez le contenu de l'adresse mémoire $0045 dans X, puis chargez le contenu de l'adresse mémoire $0030 dans Y, Stockez les dans des cases mémoires différentes.
- Comparer le contenu de l'adresse mémoire $50 avec le contenu de l'accumulateur X.

Les modes d’adressage en assembleur

Adressage inhérent : Le code opératoire contient toute l’information nécessaire à l’exécution de l’instruction.

- **CLRA** => Efface le contenu de l'accumulateur A.
- **ABX** => Ajoute le contenu de l’accumulateur B à X

Adressage immédiat: L'opérande (1 ou 2 octets) manipulé se trouve directement après l'opérateur (symbole #) et n'est pas spécifié par l'intermédiaire d'une adresse

- **LDA** #$12 => L'accumulateur est chargé avec la donnée correspondant à 12 en hexadécimal.
- **ADDB** #%10001001 => Addition de la valeur binaire 10001001 à B

Les modes d’adressage en assembleur

Adressage étendu : Il permet d’atteindre toute la mémoire mais avec un opérande 2 octets (le symbole # est absent).

- LDA $0035 => L'accumulateur A reçoit le contenu de l'adresse 35.
- STA $0035 => Le contenu de A est chargé à l'adresse $35.
- ADDA $0035 => Le contenu de $35 est ajouté à celui de A et le résultat est mis dans A.

Les modes d’adressage en assembleur

Adressage indexé : Ce mode se rapproche quelque peu de l'adressage étendu. Mais, il utilise un registre d’index (X ou Y) de 16 bits qui spécifie une base à laquelle on ajoute un déplacement signé de 5, 8 ou 16 bits. Tout dépend donc du contenu de cet index.

- LDX #$0010 => le registre X reçoit la valeur hexadécimale 10 (adressage immédiat).
- LDA, X => charge A avec le contenu de l’adresse pointée par la valeur de X.
- LDA, X+ => Charge A avec le contenu de l’adresse pointée par X, mais incrémente le contenu de X après la fin de l’instruction (Par analogie : LDA, -X).
- LDA $05,X => L'accumulateur est chargé avec une valeur située à l'adresse égale à la somme du déplacement indiqué (soit $5) et du contenu de X (soit $10) => LDA $15

Exercices

1. Utilisez l'adressage inhérent pour effacer le contenu de l'accumulateur B.
2. Ajoutez la valeur hexadécimale 1A au contenu de l'accumulateur A en utilisant l'adressage immédiat.
3. Ajoutez le contenu de la case mémoire d'adresse $0045 au contenu de l'accumulateur B en utilisant l'adressage étendu.
4. Chargez le contenu de la case mémoire d'adresse $00C0 dans l'accumulateur A en utilisant l'adressage étendu.
5. Chargez le contenu de la case mémoire pointée par la valeur de X (initialisée à $0020) dans l'accumulateur B en utilisant l'adressage indexé.
6. Soustrayez la valeur binaire 01011010 du contenu de l'accumulateur A en utilisant l'adressage immédiat.
7. Chargez le contenu de la case mémoire pointée par la valeur de X (initialisée à $0015) dans le registre Y en utilisant l'adressage indexé.
8. Ajoutez le contenu de la case mémoire pointée par la valeur de Y (initialisée à $00A0) à l'accumulateur A en utilisant l'adressage indexé
9. Comparez le contenu de l'accumulateur A avec le contenu de la case mémoire d'adresse $0030 en utilisant l'adressage étendu.

Effacer le contenu des registres A et B

- L'instruction CLRA est utilisée pour effacer le contenu du registre A en M6809.

```
CLRA   ; Efface le contenu du registre A
```

- L'instruction CLRB est utilisée pour effacer le contenu du registre B en M6809.

```
CLRB   ; Efface le contenu du registre B
```

Les instructions CLRA et CLRB sont souvent utilisées pour initialiser les registres avant de commencer une opération ou un calcul.

Il est important de noter que CLRA et CLRB ne modifient pas les autres registres ou les bits de statut.

En utilisant ces instructions, vous pouvez vous assurer que les registres A et B sont initialisés avec une valeur connue et que vous pouvez commencer votre calcul ou votre opération en toute confiance.

Instruction INC

L'instruction **INC** (increment) est utilisée pour ajouter 1 à une valeur stockée dans un registre ou une adresse mémoire.

- Syntaxe :

  INCA ; Incrémente le registre A

  INCB ; Incrémente le registre B

  INC adr ; Incrémente la valeur stockée à l'adresse adr

- Remarques :
  - L'instruction INC ne modifie pas le flag de carry (retenue).
  - Si la valeur stockée à l'adresse mémoire est la valeur maximale possible pour son type de données, l'instruction INC la remettra à zéro.

Décalages arithmétiques et logiques

Les instructions ASL et LSL

- Les instructions ASL (arithmetic shift left) et LSL (logical shift left) font la même chose : elles décalent tous les bits vers la gauche.
- Lorsque le bit de poids fort est décalé hors du registre, un zéro est placé à la place du bit manquant. Par conséquent, le contenu du bit de position 0 est déplacé vers la position 1, et un zéro est placé à la position 0
- Exemple : si le nombre binaire 10100101 est décalé de 1 bit à gauche avec ASL ou LSL, on obtient 01001010.

Les instructions ASR et LSR

- Les instructions ASR (arithmetic shift right) et LSR (logical shift right) font la même chose : elles décalent tous les bits vers la droite.
- Cependant, lors d'un décalage arithmétique, le bit de signe est décalé avec les autres bits. Cela signifie que si le bit de signe est un 1, un 1 sera inséré à la place du bit de signe lors du décalage. Si le bit de signe est un 0, un 0 sera inséré à la place du bit de signe
- Exemple : si le nombre binaire 10100101 est décalé de 1 bit à droite avec ASR, on obtient 11010010

TD1

Assembleur : Motorola 6809

Complément à un

Les instructions **COMA** et **COMB** sont utilisées pour effectuer une opération de complément à un sur les registres A et B respectivement. Complémenter à un signifie que chaque bit de la valeur binaire est inversé, c'est-à-dire que chaque 0 devient 1 et chaque 1 devient 0.

Exemple :

Supposons que la valeur de A soit 10100110 en binaire (ou A6 en hexadécimal). Après avoir exécuté l'instruction COMA, la valeur de A deviendra 01011001 en binaire (ou 59 en hexadécimal), car chaque bit a été inversé.

De même, si la valeur de B est 00111010 en binaire (ou 3A en hexadécimal), après avoir exécuté l'instruction COMB, la valeur de B deviendra 11000101 en binaire (ou C5 en hexadécimal).

Ces instructions peuvent être utilisées dans des opérations de négation ou pour inverser des bits spécifiques d'un registre.

Complément à 2

Technique utilisée en informatique pour représenter des nombres entiers signés sous forme binaire. Il est dérivé du complément à 1 et permet d'effectuer des opérations arithmétiques sur des nombres signés en utilisant les mêmes algorithmes que pour les nombres non signés.

Étapes :

- Trouver le complément à 1 du nombre binaire en question.
- Ajouter 1 au résultat obtenu à l'étape 1 pour obtenir le complément à 2.

Exemple :

Supposons que nous voulions trouver le complément à 2 du nombre binaire 00000101. Voici les étapes à suivre :

Trouver le complément à 1 :

- Inverser tous les bits du nombre pour obtenir 11111010.

Ajouter 1 au complément à 1 :

- 11111010 + 00000001 = 11111011.

Le complément à 2 du nombre binaire 00000101 est donc 11111011, qui correspond au nombre signé -5 en décimal.

Le complément à deux pour représenter les nombres signés

- En notation binaire signée (nombres positifs et négatifs), le bit de poids le plus élevé est utilisé pour représenter le signe du nombre : 0 pour les nombres positifs et 1 pour les nombres négatifs.
- Cependant, cela implique que les opérations arithmétiques sur les nombres signés doivent être traitées différemment en fonction du signe des nombres, ce qui peut compliquer les calculs.
- Le complément à deux est une méthode qui permet de représenter les nombres signés en utilisant uniquement des nombres positifs.
- Pour cela, il faut inverser tous les bits du nombre négatif et ajouter 1.
- Cette méthode permet d'effectuer les opérations arithmétiques de la même manière que pour les nombres non signés, ce qui simplifie les calculs.

Exemple d'utilisation

- Pour soustraire deux nombres signés A et B, on peut inverser tous les bits de B pour obtenir -B en notation complément à deux, puis ajouter A à -B en utilisant l'addition classique.
- Le résultat final sera automatiquement en notation complément à deux et représentera le résultat correct de la soustraction.

Exercice

Dans le langage assembleur M6809, soustraire le nombre signé -15 du nombre signé 10 en utilisant la notation complément à deux.

Étapes:

- Convertir 10 en binaire signé :
  - Le bit de poids fort (MSB) est 0, ce qui indique un nombre positif.
  - La représentation binaire est donc simplement 00001010.

- Convertir -15 en binaire signé avec la notation complément à deux :
  - Convertir 15 en binaire non signé : 00001111.
  - Inverser tous les bits : 11110000.
  - Ajouter 1 : 11110001.
  - Le résultat est donc 11110001.

- Additionner les deux nombres signés en utilisant l'addition classique :
  - 00001010 (10)

Les branchements en assembleur M6809

- Les branchements sont des instructions permettant de modifier le flot d'exécution d'un programme en sautant à une autre adresse.
- Ils sont utilisés pour implémenter des structures de contrôle de flux, comme des boucles, des branchements conditionnels, ou des appels de sous-routines.
- Les branchements peuvent être conditionnels ou inconditionnels.
- Les branchements inconditionnels sautent toujours à une adresse spécifique dans le programme.
- Les branchements conditionnels sautent à une adresse spécifique dans le programme en fonction d'une condition.

Branchement conditionnel M6809

- Les instructions de branchement conditionnel du M6809 permettent de modifier le flux d'exécution du programme en fonction de l'état des drapeaux (flags) du processeur.

- Les instructions disponibles sont :
  BHI (Branch if Higher) : branche si le drapeau Carry est à zéro et le drapeau Zéro est à zéro.
  BHS (Branch if Higher or Same) : branche si le drapeau Carry est à zéro.
  BLO (Branch if Lower) : branche si le drapeau Carry est à un.
  BLS (Branch if Lower or Same) : branche si le drapeau Carry est à un ou le drapeau Zéro est à un.

- Exemple d'utilisation :

```
CMPA #10 ; comparer A à 10
BHI LABEL1 ; si A > 10, aller à LABEL1
BLS LABEL2 ; si A <= 10, aller à LABEL2
END
```

Branchements dans l'assembleur m6809 : BSR, BRA, BRN

- L'instruction BSR (Branch Subroutine) est un branchement inconditionnel qui permet de sauter à une sous-routine (une portion de code qui peut être appelée plusieurs fois).
- L'instruction BSR est suivie d'une étiquette qui indique l'adresse de la sous-routine à appeler, comme dans l'exemple suivant :

```
BSR SUBROUTINE ; sauter à la sous-routine SUBROUTINE
```

- L'instruction BRA (Branch Always) est un branchement inconditionnel qui permet de sauter à une adresse spécifique dans le programme.
- L'instruction BRA est suivie d'une étiquette qui indique l'adresse à laquelle sauter, comme dans l'exemple suivant :

```
BRA LOOP ; sauter à l'étiquette LOOP
```

Différence - **Retour au point d'appel :**

  - **BSR** : Nécessite une instruction **RTS** dans la sous-routine pour revenir au point d'appel initial.
  - **BRA** : Pas de retour automatique, le flux reste sur l'adresse cible jusqu'à la prochaine instruction de saut.

- L'instruction BRN (Branch Never) est un branchement inconditionnel qui ne saute jamais.
- L'instruction BRN est souvent utilisée pour remplacer une instruction non utilisée ou pour ajouter une instruction d'attente dans une boucle infinie, comme dans l'exemple suivant :

```
WAIT:

BRN WAIT ; attendre indéfiniment
```

TD2

Assembleur : Motorola 6809

Branchements dans l'assembleur m6809 : BEQ

Description :

- BEQ permet de sauter à une nouvelle instruction si le contenu d'un registre ou d'une adresse mémoire est égal à zéro.
- Si la condition est vraie (le contenu est égal à zéro), le programme saute à l'adresse spécifiée dans l'instruction BEQ.
- Si la condition est fausse (le contenu n'est pas égal à zéro), le programme continue l'exécution à la ligne suivante sans sauter.

Syntaxe :

BEQ adresse

Exemple : Vérifie si le contenu de A est égal à zéro

LDA #$00 ; Charge le registre A avec la valeur zéro
BEQ skip ; Sauter à l'adresse de "skip" si A est égal à zéro
; Code à exécuter si A n'est pas égal à zéro
skip: ; Code à exécuter si A est égal à zéro

Branchements dans l'assembleur m6809 : **BNE**

Description :

- BNE permet de sauter à une nouvelle instruction si le contenu d'un registre ou d'une adresse mémoire n'est pas égal à zéro.
- Si la condition est vraie (le contenu n'est pas égal à zéro), le programme saute à l'adresse spécifiée dans l'instruction BNE.
- Si la condition est fausse (le contenu est égal à zéro), le programme continue l'exécution à la ligne suivante sans sauter.

Syntaxe :

BNE adresse

Exemple :

```
; Vérifie si le contenu de A n'est pas égal à zéro
LDA #1 ; Charge le registre A avec la valeur un
BNE skip ; Sauter à l'adresse de "skip" si A n'est pas égal à zéro
; Code à exécuter si A est égal à zéro
skip: ; Code à exécuter si A n'est pas égal à zéro
```

- Pour faciliter l'écriture du programme dans un tel cas, on ne spécifie pas explicitement l'adresse à laquelle le PC doit se rendre mais une étiquette.
    - Exemple :

```
LDB  $AA
Retour:
DECB
BNE  Retour
```

- Dans cet exemple, la première ligne charge B avec le contenu de l'adresse $AA, puis on décrémente le contenu de B à la troisième ligne et la quatrième ligne signifie que si ce contenu ne devient pas égal à zéro, alors on doit revenir à la ligne indiquée par l'étiquette : Retour
- Remarque : Une étiquette peut être n'importe quelle chaîne de caractère commençant par un caractère de l'alphabet

TD3

Assembleur : Motorola 6809

Mnémoniques	Opérations réalisées
Branchements simples	
BEQ, LBEQ	Branchement si égal à zéro (bit Z)
BNE, LBNE	Branchement si différent de zéro (bit Z)
BMI, LBMI	Branchement si négatif (bit N)
BPL, LBPL	Branchement si positif (bit N)
BCS, LBCS	Branchement si retenue (bit C)
BCC, LBCC	Branchement si pas de retenue (bit C)
BVS, LBVS	Branchement si dépassement (bit V)
BVC, LBVC	Branchement si pas de dépassement (bit V)
Branchements signés	
BGT, LBGT	Branchement si supérieur à zéro
BGE, LBGE	Branchement si supérieur ou égal à zéro
BLT, LBLT	Branchement si inférieur à zéro
BLE, LBLE	Branchement si inférieur ou égal à zéro
Branchements non signés	
BHI, LBHI	Branchement si plus grand que
BHS, LBHS	Branchement si plus grand ou égal à
BLO, LBLO	Branchement si plus petit que
BLS, LBLS	Branchement si plus petit ou égal à
Autres branchements	
BSR, LBSR	Branchement à un sous-programme
BRA, LBRA	Branchement inconditionnel
BRN, LBRN	Branchement n'ayant jamais lieu

BEQ : Branchement si égal à zéro

BNE : Branchement si différent de zéro

BMI : Branchement si négatif

BPL : Branchement si positif

BCS : Branchement si retenue

BCC : Branchement si pas de retenue

BVS : Branchement si dépassement de capacité

BRA : Branchement toujours

……