Support de cours

Module Microprocesseurs et DSP

Partie :1

ARCHITECTURE D'UN ORDINATEUR

Définitions

- **Ordinateur** : Une machine de traitement automatique de l'information

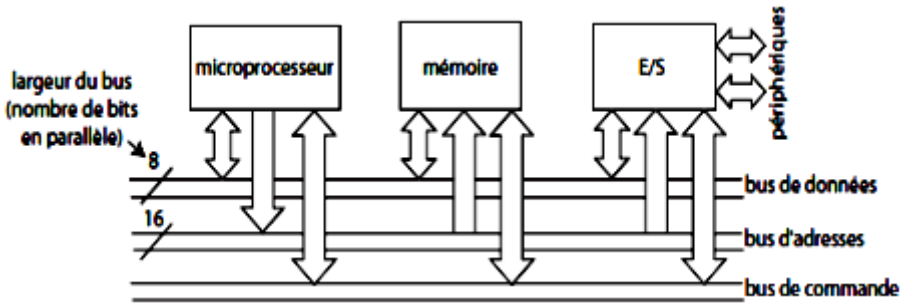
Un ordinateur est capable d'acquérir, de conserver, de traiter et restituer les informations.

- **Type d'information**: Valeurs numériques, textes, images, son, vidéos représentés sous forme de données numériques.
- **Informatique** : Science du traitement de l'information.
- **Système informatique** : Ensemble des moyens logiciels & matériels nécessaires pour satisfaire les besoins informatiques des utilisateurs.
- **Système d'exploitation :** Programme système qui gère les différentes ressources de la machine
- **Programmation** : A partir d'un problème donné, réaliser un programme dont l'exécution apporte une solution satisfaisante au problème posé.
- Un ordinateur est une machine qui doit être programmée, c'est-a-dire qu'il faut prévoir absolument tout ce qu'il doit faire et le lui expliquer précisément ! C'est le rôle du programmateur.
- **Programme :** Suite d'instructions dans un langage donnée, définissant un traitement exécutable par un ordinateur.
- **Langages de programmation** : (machine, assembleur, évolues)

Composants classiques d'un ordinateur ou unités fonctionnelles

1. **La mémoire centrale** qui contient les données et les programmes a exécuté
2. **l'unité centrale de traitement** qui exécute les programmes chargées en mémoire
3. **les unités d'entrée/sortie** qui permettent le lien et l'échange d'information avec les périphériques (clavier, écran, souris, imprimante, etc.)

Les trois unîtes sont lies par un ensemble de lignes (câbles, pistes de circuits imprimés, etc.) exploité en commun appelés bus.



Le bus

Un bus permet de transférer des données du même type sous forme parallèle entre les différents composants de l'ordinateur.

On retrouve trois types de bus :

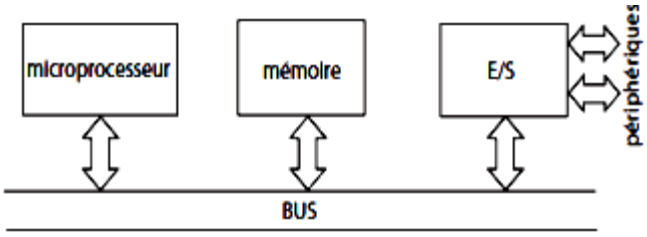
- **Bus d'adresses**: transporte les adresses mémoire auxquelles le processeur souhaite accéder (unidirectionnel) : seul le microprocesseur peut délivrer des adresses.
- **Bus de données**: assure le transfert des informations entre le processeur et son environnement. Son nombre de lignes est égal à la capacité de traitement du microprocesseur.
- **Bus de commandes**: constitué par quelques conducteurs qui transmettent les ordres de commande tel que les ordres de lecture et d'´ecriture de la mémoire et des unités E/S. (bidirectionnel).

Principes de l'ordinateur selon Von Neumann

La réalisation matérielle des ordinateurs est généralement basée sur l'architecture de **Von Neumann** :

- Machine universelle contrôlée par programme.
- Instructions du programme codées sous forme numérique binaire et enregistrées en mémoire.
- Les Instructions exécutées normalement en séquence mais pouvant être modifies par le programme lui-même.
- Existence d'instructions permettant les ruptures de séquences.

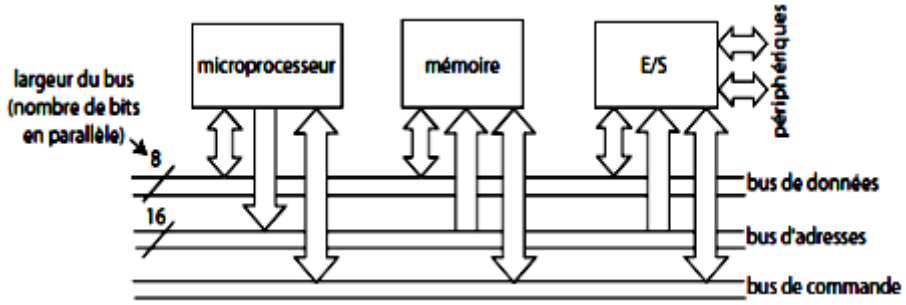
Le processeur (microprocesseur) est l'unité intelligente de traitement des informations. Son travail consiste à lire des programmes (des suites d'instructions), à les décoder et à les exécuter.
Le processeur échange des informations avec la mémoire et l'unité d'E/S, sous forme de mots binaires, au moyen d'un ensemble de connexions appelé bus.



Les processeurs peuvent être classes selon la longueur maximale des mots binai0res qu'ils peuvent échanger avec la mémoire et les E/S : microprocesseurs 8 bits, 16 bits, 32 bits, ...

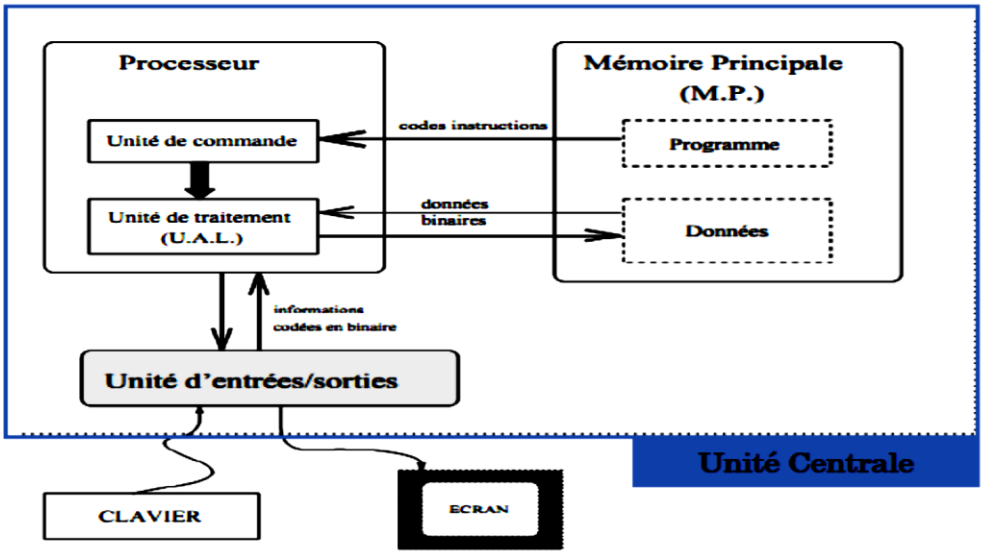
Un bus permet de transférer des données sous forme parallèle, c'st-`a-dire en faisant circuler n bits simultanément. Il est constitué par un ensemble de fils qui assure la transmission du même type d'information. On retrouve trois types de bus véhiculant des informations en parallèle dans un système de traitement programmé de l'information :

- **un bus de données** : bidirectionnel qui assure le transfert des informations entre le microprocesseur et son environnement, et inversement. Son nombre de lignes est égal à la capacité de traitement du microprocesseur.
- **un bus d'adresses**: unidirectionnel qui permet la sélection des informations à traiter dans un espace mémoire (ou espace adressable) qui peut avoir 2^n emplacements, avec n = nombre de conducteurs du bus d'adresses.
- **un bus de commande**: constitué par quelques conducteurs qui transmettent les ordres de commande tel que les ordres de lecture et d'´ecriture de la mémoire et des unités E/S.



Remarque : les bus de données et de commande sont bidirectionnels, le bus d'adresse est unidirectionnel : seul le microprocesseur peut délivrer des adresses (il existe une dérogation pour les circuits d'accès direct a la mémoire, DMA).

ARCHITECTURE DE VON NEUMANN (1946)



Unité centrale de traitement (Processeur)

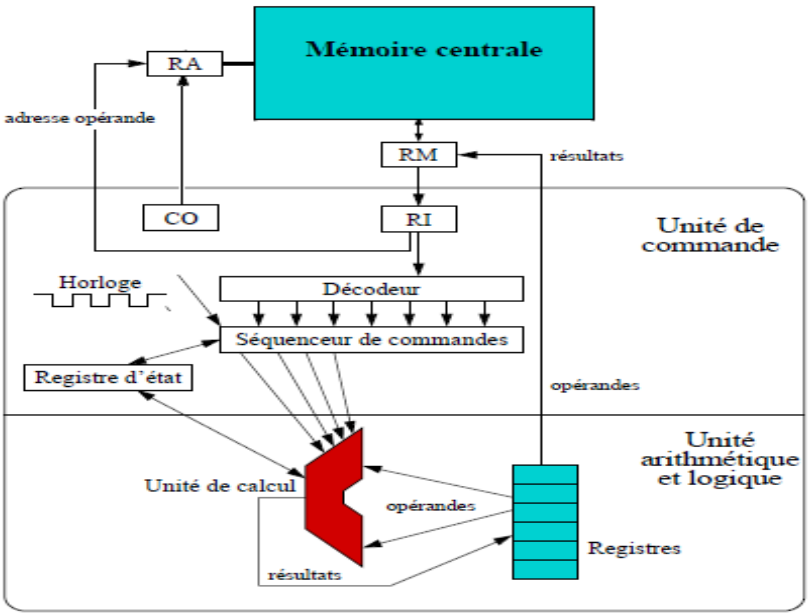
Un processeur est constitué de:
• Une unité de commande qui lit les instructions et les décode;
• Une unité de traitement (UAL - unité arithmétique et logique) qui exécute les instructions;
• D'un ensemble de mémoire appelés registres;
• D'un bus de données externe;
• D'un bus d'adresse externe;
• D'un bus de commande externe;
• D'un bus de données interne reliant l'unité de commande l'UAL et les registres.

Lorsque tous ces éléments sont regroupés sur une même puce, on parle alors de microprocesseur. La figure ci dessous donne une idée sur l'architecture interne d'un microprocesseur. Sur cette figure nous pouvons voir les 3 bus qui permettent au microprocesseur de communiquer avec l'extérieur.

- **Unité de commande**

Prends les instructions en mémoire, les décode et les passe a l'UAL en fonction des cycles horloges.
Elle contient :

- Registre d'instruction (RI) : contient l'instruction (opération + opérande) en cours d'exécution
- Compteur ordinal (CO) : registre qui adresse de la prochaine instruction à exécuter
- Décodeur : décode les instructions
- Séquenceur : active les circuits nécessaires de l'UAL
- Horloge : Une horloge est un circuit qui émet régulièrement une Suite d'impulsions calibrées (fréquence de 1 à qlq Ghz). Utilisé comme base de temps pour rythmer l'enchainement des commandes



- **Unité Arithmétique et Logique (UAL)**

Réalise électivement les différentes opérations:

- les opérations arithmétiques (+,-,*,/)
- logiques (NOT, AND, OR, XOR).

Deux registres sont associés à l'UAL : l'accumulateur et le registre d'état.

1. Accumulateur : C'est une des deux entrées de l'UAL. Il est impliqué dans presque toutes les opérations réalisées par l'UAL. Certains constructeurs ont des microprocesseurs à deux accumulateurs (Motorola : 6800).
2. Registre d'état : A chaque opération, le microprocesseur positionne un certain nombre de bascules d'état. Ces bascules sont appelées aussi indicateurs d'état ou drapeaux (status, flags). Par exemple, si une soustraction donne un résultat nul, l'indicateur de zéro (Z) sera mis à **1**. Ces bascules sont regroupées dans le registre d'état.

On peut citer comme indicateurs:

- C (carry) : retenue : Sur les opérations arithmétiques, ce bit signale la présence d'une une retenue
- AC retenue intermédiaire (auxillary carry= : Sur les opérations arithmétiques, ce bit signale une retenue entre groupes de 4 bits (Half-byte: demi-octet) d'une quantité de 8 bits.
- S signe : Ce bit est mise à **1** lorsque le résultat de l'opération est négatif (MSB: bit de plus fort poids du résultat: à **1**).
- O : Over flow débordement : Cet indicateur est mis **1**, lorsqu'il y a un dépassement de capacité pour les opérations arithmétiques
- Z : zéro, Ce bit est mis à **1** lorsque le résultat de l'opération est nul.
-P : parité: nombre de 1 pair ou impair.

Exécution d'un programme

- Chargement du programme et des données depuis un périphérique dans la mémoire centrale
- Chargement séquentiel des instructions du programme de la mémoire centrale dans l'unité de contrôle
- Analyse par l'unité de contrôle de l'instruction et passage a l'UAL pour traitement
- Traitement de l'instruction par l'UAL avec éventuellement appel à la mémoire ou aux unités d'entrée-sortie.

Exécution d'une instruction

Le microprocesseur ne comprend qu'un certain nombre d'instructions qui sont codées en binaire. Une instruction est composée de deux éléments :

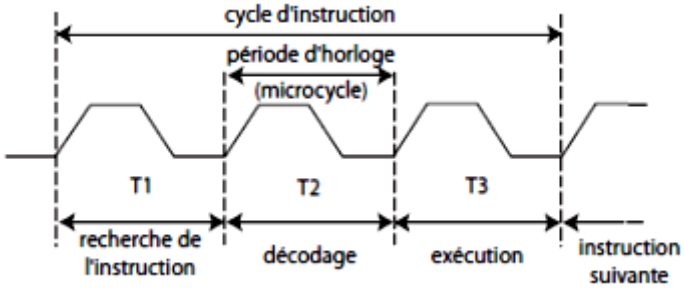
- Le code opération : C'est un code binaire qui correspond à l'action à effectuer par le processeur

- Le champ opérande : Donnée ou bien adresse de la donnée.

Exemple :

Code instruction	Code opérande
1001 0011	0011 1110

La taille d'une instruction peut varier, elle est généralement de quelques octets, elle dépend également de l'architecture du processeur.



L'exécution d'une instruction passe par plusieurs phases :

Phase 1: Recherche de l'instruction à traiter

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.

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

Le registre d'instruction contient maintenant le premier mot de l'instruction qui peut être codée sur plusieurs mots. Ce premier mot contient le code opératoire qui définit la nature de l'opération à effectuer (addition, rotation,...) et le nombre de mots de l'instruction.

1. L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.

2. Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur sur le bus de données.

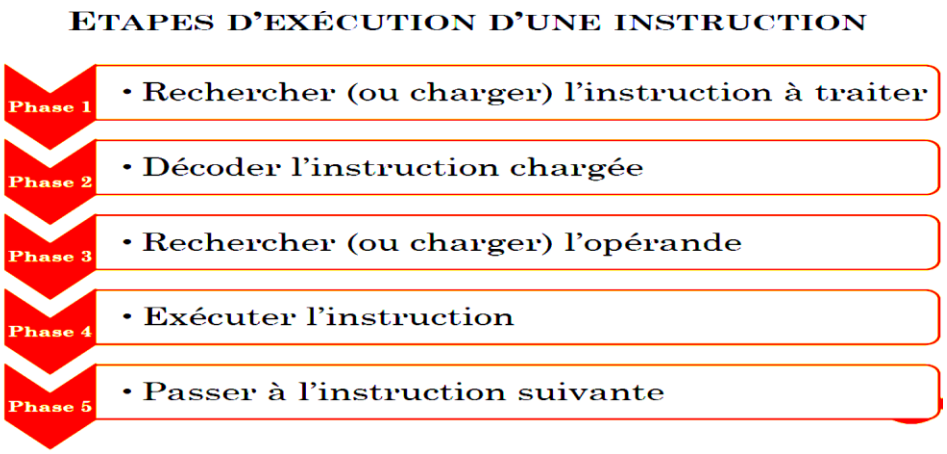
3. L'opérande est stocké dans un registre.

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.



Jeu d'instructions

Le jeu d'instructions décrit l'ensemble des opérations élémentaires que le microprocesseur pourra exécuter. Il va donc en partie déterminer l'architecture du microprocesseur à réaliser et notamment celle du séquenceur. Le nombre d'instructions du jeu d'instructions est directement lié au format du code instruction. Ainsi un octet permet de distinguer au maximum 256 instructions différentes.

Les types instructions

On distingue 6 types d'instructions :
- **les instructions de transfert de** données : charger ou sauver en mémoire, effectuer des transferts de registre à registre, etc…
- **Opérations arithmétiques** : addition, soustraction, division, multiplication
-**Opérations logiques** : ET, OU, NON, NAND, comparaison, test, etc…
- **Contrôle de séquence** : branchement conditionnel et inconditionnel, test, etc…
- **les interruptions**,
- **les instructions de contrôle du processeur et du registre d'état**.

Une instruction nécessite un certain nombre de cycles pour s'exécuter. La durée d'un cycle dépend de la fréquence d'horloge.

Temps d'exécution

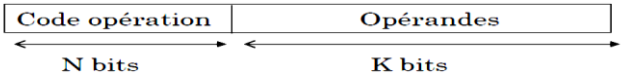
Chaque instruction nécessite un certain nombre de cycles d'horloges pour s'effectuer. Le nombre de cycles dépend de la complexité de l'instruction et aussi du mode d'adressage. Il est plus long d'accéder à la mémoire principale qu'à un registre du processeur. La durée d'un cycle dépend de la fréquence d'horloge du séquenceur.

Codage

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.

Formats des instructions en code machine

L'instruction est composée de deux champs :



- Code d'opération représentant l'action que le processeur doit accomplir.

Champ des opérandes définissant les paramètres de l'action. Un opérande peut s'agir d'une donnée ou bien d'une adresse mémoire.
La taille d'une instruction dépend du type de l'instruction et du type de l'opérande.
Les instructions et leurs opérandes sont stockés dans la mémoire.

Il existe plusieurs formats d'instruction en code machine :

1. **Instruction à trois adresses:** Il faut préciser le premier opérande, le deuxième opérande et l'emplacement du résultat

Code opération	**Adresse Operande1**	**Adresse Operande2**	**Adresse résultat**

(Adresse Operande1) **opérateur** **(Adresse Operande2) → Adresse résultat**

Exemple: ADD a,b, c: a +b → **c**
La taille de l'instruction est grande. Pratiquement ils n'existent pas d'instruction de ce type.

2. **Instruction à deux adresses:**

Il faut préciser le premier opérande et le deuxième opérande. Le résultat est implicitement mis dans le premier opérande.

Exemple: ADD a,b : a +b →**a**

Code opération	**Adresse Operande1**	**Adresse Operande2**

(Adresse Operande1) **operateur** **(Adresse Operande2) → Adresse Operande1**

3. **Instruction à une adresse**

Dans ce cas, la machine dispose d'un registre spécial appelé accumulateur (A), utilisé par défaut pour contenir le premier opérande d'une instruction avant son exécution et le résultat après.

Il faut préciser uniquement le deuxième opérande. Le premier opérande existe dans le registre accumulateur. Le résultat est mis dans le registre accumulateur.

Code opération	**Adresse Operande2**

(Accumulateur) **operateur** **(Adresse Operande2) → Accumulateur**
Exemple: ADD b : (A) +b →**A**

4. **Instruction à zéro adresse (utilisation de la pile)**

Code opération

((pointeur pile)) **operateur** **((pointeur pile -1)) → pointeur pile**

L'instruction ne contient que le code opération. Dans cette architecture, les instructions vont directement agir sur la pile.

La pile est un espace de la mémoire ou les données sont stockes les uns sur les autres c.à.d à des adresses successives en formant une pile du type LIFO (Last IN first out/dernier entrant premier sortant). L'adresse du sommet est stocké dans un registre appelé pointeur de pile (PP) ou stack pointer (SP).

- Les opérandes sont automatiquement chargés depuis la pile à partir des adresses fournies par le pointeur de pile et le résultat est à son tour empilé.
- Les opérandes sont placés au sommet de la pile, et sont adressés implicitement : le processeur n'a pas besoin de préciser leurs adresses. Les instructions arithmétiques et logiques sont vraiment très courtes.

(**pointeur pile**) = opérande au sommet de la pile
((**pointeur pile) -1**)= opérande au (sommet de la pile -1)

- Deux instructions a 1 adresse sont utilisés pour : charger la pile(PUSH) et décharger la pile (POP).

PUSH adr : charger la pile par un opérande d'adresse **adr** au sommet de la pile pointer par le contenu du registre PP :
(adr) → **sommet = (pp)**
POP adr : décharger l'opérande du sommet et le stocké a l'adresse **adr**
((pp)) → adr

Exemple:
a + b –c = d sera traduit par :
Chargé c dans la pile : le contenu de la pile est (c)

Chargé b dans la pile : le contenu de la pile est (b, c)
Chargé a dans la pile : le contenu de la pile est (a, b, c)
Addition les deux opérandes au sommet de la pile a et b : le contenu de la pile est (a+b, c)
Soustraction de deux opérandes au sommet de la pile a+b et c: le contenu de la pile est (a+b+c)
Stocké résultat **(a+b-c)** dans l'adresse d : le contenu de la pile est vide ( )
Le programme sera :

```
PUSH c  ; Empile c : (c)
PUSH b ; Empile b : (b, c)
PUSH a; Empile a : (a, b, c)
ADD     ; Additionne B et C : (a+b, c)
SOUS ; soustraire a+b et c : (a+b- c)
POP A   ; Stocke le sommet de la pile à l'adresse d et dépile
```

Les modes d'adressage mémoire

Une adresse mémoire peut être exprimée par plusieurs façons : ce sont les modes d'adressage.

Un mode d'adressage définit la manière dont le microprocesseur va accéder à l'opérande.
La forme générale d'une instruction en code machine :

Code opération	Adresse opérande

opération \ mode adressage	@

1. Le champ code opération indique :
   - le type d'opération a exécuté
   - le mode d'adressage utilisé pour accéder aux opérandes

2. Le champ code opération indique : une adresse selon le mode adresse utilisé.

Chaque processeur possède plusieurs nombre de mode d'adressage. Selon le mode d'adressage de la donnée, une instruction sera codée par 1 ou plusieurs octets.
Les différents modes d'adressage dépendent des microprocesseurs. Les modes d'adressage les plus utilisés sont : registre, immédiat, direct, indirect, indexé et relatif.

- **l'adressage par registre** :

Le champ adresse de l'instruction fait référence à un registre qui contient l'opérande.
@=R : registre identifie par une lettre (A, B,C….)
Equation :
Operande = (R)
Exemple : ADD C

- **l'adressage immédiat** :

Le champ adresse de l'instruction ne contient pas l'adresse mémoire de l'opérande mais l'opérande lui-même. Ce mode ne nécessite aucun accès mémoire.
@=DATA
Equation :
Operande = DATA
Exemple : ADI 15h
LXI B 0012h

- **l'adressage direct** :

Le champ adresse de l'instruction contient l'adresse mémoire réelle de l'opérande. Un accès mémoire est nécessaire pour accéder à l'opérande.

@=adresse : sur p bits

Equation :

Operande = (@)

Exemple : LDA 0024h

STA 0024h

- **l'adressage indirect :**

Le champ adresse de l'instruction contient une adresse mémoire qui contient l'adresse effective de l'opérande. Un accès mémoire est nécessaire pour accéder à l'adresse de l'opérande, un deuxième accès mémoire pour accéder à l'opérande lui-même

@=adresse : sur p bits

Equation :

Operande = ((@))

- **L'adressage relatif** :

Le champ adresse de l'instruction contient une adresse mémoire relative calculé par rapport à une adresse de référence qui est stocké dans un registre de l'unité arithmétique et logique (UAL) :

1. L'adressage est dite indexé si la référence se trouve dans le registre index (RI)
2. L'adressage est dite basé si la référence se trouve dans le registre de base (RB)
3. L'adressage par rapport à l'adresse courante si la référence est le contenu du compteur ordinal (PC)

Equation :

Opérande = (@) + référence

1. Référence= (RI)
2. Référence= (RB)
3. Référence= (PC)

Les mémoires

Une mémoire est un circuit à semi-conducteur permettant

- d'enregistrer des informations binaires (instructions et données),
- de les conserver
- de restituer ces informations

Il y a :

- écriture lorsqu'on enregistre des informations en mémoire,
- lecture lorsqu'on récupère des informations précédemment enregistrées.

Organisation de l'information binaire

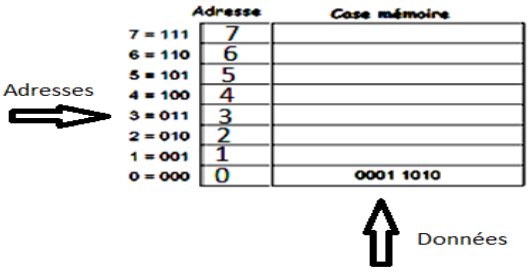
Une information en binaire est une suite de bits de 1 et 0 qu'on peut organiser sous forme de

- Bit : Unité de base. C'est le plus petit élément de stockage
- Octet (ou byte) : groupe de 8 bits
- Le caractère (7, 8 ou 16 bits), codage selon un standard (ASCII, Unicode ...)
- Mot : groupement d'octets (8, 16, 32, 64 ...) : Unité d'information adressable en mémoire
- Enregistrement : bloc de données
- Fichier : ensemble d'enregistrements

Organisation d'une mémoire

Les données binaires dans une mémoire sont organisés sous forme de mots de plusieurs bits sur les quelles on peut faire des opérations de lecture et d'écriture.

Chaque mot est caractérisé par un numéro (valeur numérique binaire) ou adresse qui indique son emplacement à l'intérieur de la mémoire.



Mot-mémoire

La mémoire est constituée de cellules. Chaque cellule correspond à un mot-mémoire.
Physiquement, une mémoire est constituée d'un grand nombre de registres de même taille.
Chaque registre reçoit un mot donc **la taille du registre définit la taille du mot.**
La longueur de ce mot constitue une caractéristique importante de l'architecture d'un ordinateur.

Chaque registre représente alors une case mémoire qui peut contenir un seul mot. Le nombre de cases mémoires pouvant être très élevé, il est alors nécessaire de pouvoir les identifier par un numéro. Ce numéro est appelé adresse. Chaque donnée devient alors accessible grâce à son adresse.

Adresses mémoire

Avec une adresse de n bits il est possible de référencer au plus 2^n cases mémoire. Chaque case est remplie par un mot de données (sa longueur m est toujours une puissance de 2).

- Le nombre de fils d'adresses d'une mémoire définit donc le nombre de cases mémoire que comprend la mémoire.
- Le nombre de fils de données définit la taille des données que l'on peut sauvegarder dans chaque case mémoire.

En plus du bus d'adresses et du bus de données, un boîtier mémoire comprend une entrée de commande qui permet de définir le type d'opération que l'on effectue avec la mémoire (lecture/écriture) et une entrée de sélection qui permet de mettre les entrées/sorties du boîtier en haute impédance.

On peut donc schématiser un circuit mémoire par la figure suivante où l'on peut distinguer :

- les entrées d'adresses
- les entrées de données
- les sorties de données
- Une entrée de sélection de lecture ou d'écriture. (R/W)

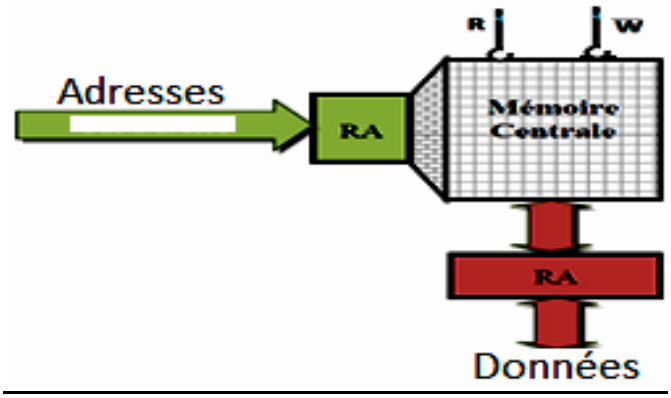
Une opération de lecture ou d'écriture de la mémoire suit toujours le même cycle :

1. sélection de l'adresse
2. choix de l'opération à effectuer (R/W)
3. lecture ou écriture de la donnée

Remarque : Les entrées et sorties de données sont très souvent regroupées sur des bornes bidirectionnelles.

La mémoire communique avec son milieu extérieur à travers deux registres :

- Registre adresse (RA), qui contient l'adresse du mot mémoire à lire ou a écrire.
- Registre mot (RM), qui contient le contenu du mot mémoire à lire ou a écrire.



Exemple

Si la mémoire comporte 256($256=2^8$) mots de 32 bits, le registre d'adresse doit avoir 8bits tandis que le registre mot doit avoir 32 bits.

Ces registres sont utilises pour exécuter les 2 opérations élémentaires en mémoire :

Lecture : le registre d'adresse contient l'adresse du mot à lire qui est copie dans le registre mot.

Écriture : le registre d'adresse contient l'adresse du mot dans lequel le contenu du registre mot va être écrit.

La capacité d'une mémoire

S'exprime en fonction du nombre de mots-mémoire ainsi que du nombre de bits par mot. Mais en général seulement la taille en octet est comptée.

Unité de mesure de la capacité de mémoire ($2^{10}=1024\approx1000=1$Killo)

- Kilo (Ko) = $2^{10}=1024$ octets
- Mega (Mo) = $2^{20}=1048576$ octets
- Giga (Go) = $2^{30}=1073741824$ octets
- Tera (To) = $2^{40}=1099511627776$ octets

Caractéristiques d'une mémoire

- La capacité : c'est le nombre total de bits que contient la mémoire. Elle s'exprime aussi souvent en octet.
- Le format des données : c'est le nombre de bits que l'on peut mémoriser par case mémoire. On dit aussi que c'est la largeur du mot mémorisable.
- Le temps d'accès : c'est le temps qui s'écoule entre l'instant où a été lancée une opération de lecture/écriture en mémoire et l'instant où la première information est disponible sur le bus de données.
- Le temps de cycle : il représente l'intervalle minimum qui doit séparer deux demandes successives de lecture ou d'écriture. Le débit : c'est le nombre maximum d'informations lues ou écrites par seconde.
- Volatilité : elle caractérise la permanence des informations dans la mémoire. L'information stockée est volatile si elle risque d'être altérée par un défaut d'alimentation électrique et non volatile dans le cas contraire.

Les types d'accès mémoire

L'accès à une mémoire peut se faire avec

- Accès direct

On accède directement à n'importe quelle information dont on connaît l'adresse et que le temps mis pour obtenir cette information ne dépend pas de l'adresse. On dira que l'accès à une telle mémoire est aléatoire ou direct.

- Accès séquentiel

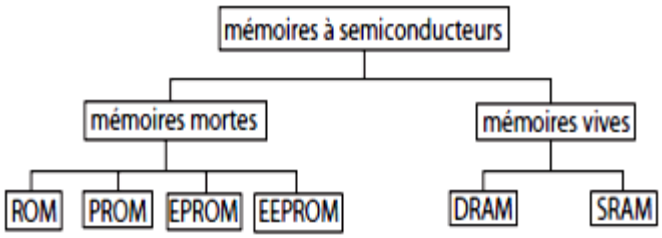
Le temps d'accès est variable selon la position de l'information recherchée. Exemple de la bande magnétique ou il faut dérouler la bande en repérant tous les enregistrements jusqu'à ce que l'on trouve celui que l'on désire.

- Accès semi-séquentiel : combinaison des accès direct et séquentiel.

Pour un disque magnétique par exemple l'accès à la piste est direct, puis l'accès au secteur est séquentiel.

Différents types de mémoire

- Les mémoires vives (RAM)
- Les mémoires mortes (ROM)



1. **Les mémoires vives RAM** ((Random Acces Memory : mémoire à accès aléatoire)

Une mémoire vive sert au stockage temporaire de données. Elle doit avoir un temps de cycle très court pour ne pas ralentir le microprocesseur. Les mémoires vives sont volatiles : elles perdent leurs informations en cas de coupure d'alimentation. Il existe deux grandes familles de mémoires RAM) :

- Les RAM statiques
- Les RAM dynamiques

- **Les RAM statiques**

Le bit mémoire d'une RAM statique (SRAM) est composé d'une bascule. Chaque bascule contient entre 4 et 6 transistors.

- **Les RAM dynamiques**

Dans les RAM dynamiques (DRAM), l'information est mémorisée sous la forme d'une charge électrique stockée dans un condensateur (capacité grille substrat d'un transistor MOS)

Avantage :

Cette technique permet une plus grande densité d'intégration, car un point mémoire nécessite environ quatre fois moins de transistors que dans une mémoire statique. Sa consommation s'en retrouve donc aussi très réduite.

Inconvénients :

La présence de courants de fuite dans le condensateur contribue à sa décharge. Ainsi, l'information est perdue si on ne la régénère pas périodiquement (charge du condensateur). Les RAM dynamiques doivent donc être rafraîchies régulièrement pour entretenir la mémorisation : il s'agit de lire l'information et de la recharger. Ce rafraîchissement indispensable a plusieurs conséquences :

- il complique la gestion des mémoires dynamiques car il faut tenir compte des actions de rafraîchissement qui sont prioritaires.
- la durée de ces actions augmente le temps d'accès aux informations.

D'autre part, la lecture de l'information est destructive. En effet, elle se fait par décharge de la capacité du point mémoire lorsque celle-ci est chargée. Donc toute lecture doit être suivie d'une réécriture.

Conclusions

En général les mémoires dynamiques, qui offrent une plus grande densité d'information et un coût par bit plus faible, sont utilisées pour la mémoire centrale, alors que les mémoires statiques, plus rapides, sont utilisées lorsque le facteur vitesse est critique, notamment pour des mémoires de petite taille comme les caches et les registres.

2. **Les memoires mortes ROM** (ROM: Read Only Memory).)

Pour certaines applications, il est nécessaire de pouvoir conserver des informations de façon permanente même lorsque l'alimentation électrique est interrompue. On utilise alors des mémoires mortes ou mémoires à lecture seule Ces mémoires sont non volatiles.

Ces mémoires, contrairement aux RAM, ne peuvent être que lue. L'inscription en mémoire des données reste possible mais est appelée programmation. Suivant le type de ROM, la méthode de programmation changera. Il existe donc plusieurs types de ROM :

ROM; PROM; EPROM; EEPROM; FLASH EPROM.

- **LA ROM**

Elle est programmée par le fabricant et son contenu ne peut plus être ni modifié, ni effacé par l'utilisateur.

- **La PROM** (Programmable ROM).

C'est une ROM qui peut être programmée une seule fois par l'utilisateur. La programmation est réalisée à partir d'un programmateur spécifique.

- **L'EPROM** ou UV-EPROM (Erasable Programmable ROM)

Est une PROM reprogrammable et dont le contenu peut être effacer par l'exposition d'une vingtaine de minutes à un rayonnement ultraviolet Cet effacement est reproductible plus d'un millier de fois.

- **L'EEPROM** (Electically EPROM)

L'EEPROM est une mémoire programmable et effaçable électriquement. Elle se comporte comme une RAM non Volatile. La Programmation et l'effacement mot par mot est possible.

- **La FLASH EPROM**

La mémoire Flash s'apparente à la technologie de l'EEPROM. Elle est programmable et effaçable électriquement comme les EEPROM.

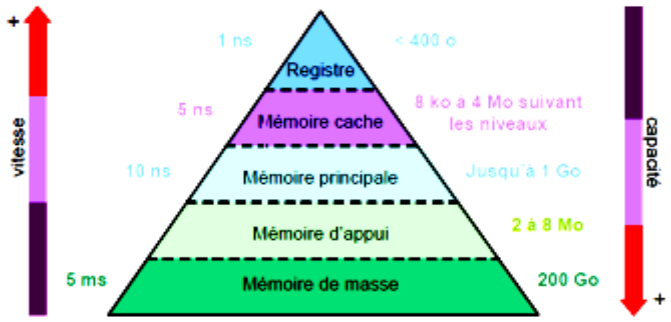
Critères de choix d'une mémoire

Les principaux critères à retenir sont :

- capacité
- vitesse
- consommation
- coût

Notion de hiérarchie mémoire

La vitesse d'accès à la mémoire est un critère important dans la mesure les performances d'un système.
Une mémoire idéale serait une mémoire de grande capacité, capable de stocker un maximum d'informations et possédant un temps d'accès très faible afin de pouvoir travailler rapidement sur ces Informations. Mais il se trouve que les mémoires de grande capacité sont souvent très lente et que les mémoires rapides sont très chères.
Afin d'obtenir le meilleur compromis coût-performance, on définie donc une hiérarchie mémoire. On utilise des mémoires de faible capacité mais très rapide pour stocker les informations dont le microprocesseur se sert le plus et on utilise des mémoires de capacité importante mais beaucoup plus lente pour stocker les informations dont le microprocesseur se sert le moins. Ainsi, plus on s'éloigne du microprocesseur et plus la capacité et le temps d'accès des mémoires vont augmenter.



- Les registres sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.
- La mémoire cache est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.
- La mémoire principale est l'organe principal de rangement des informations. Elle contient les programmes (instructions et données) et est plus lente que les deux mémoires précédentes.
- La mémoire d'appui sert de mémoire intermédiaire entre la mémoire centrale et les mémoires de masse. Elle joue le même rôle que la mémoire cache.
- La mémoire de masse est une mémoire périphérique de grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations. Elle utilise pour cela des supports magnétiques (disque dur, ZIP) ou optiques (CDROM, DVDROM).

I. Représentation des informations en binaire

Rappels

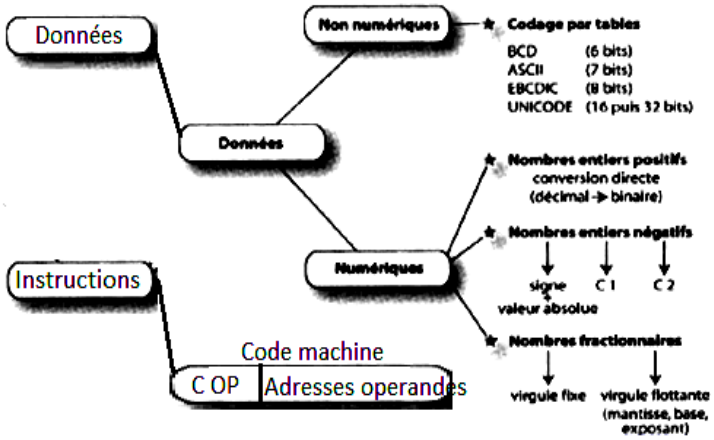
Tout peut être représenté par des 0 et des 1 : c'est la codification ou la numérisation.
Les informations traitées par un microprocesseur sont de différents types (nombres, instructions, images, vidéo, etc…) mais elles sont toujours représentées sous un format binaire. Seul le codage changera suivant les différents types de données à traiter.
En binaire, une information élémentaire est appelé bit et ne peut prendre que deux valeurs différentes:0 ou 1. Elles sont représentées physiquement par 2 niveaux de tensions différents (5V pour le 1 et 0V pour le 0) Une information plus complexe sera codée sur plusieurs bits. On appelle cet ensemble un mot.
Un mot de 8 bits est appelé un octet

Différents types d'informations

Les informations traitées par ordinateur sont de deux types :

1. Instructions
2. Données

- Numériques : Nombres (entiers, réels positifs ou négatifs)
- Non numérique : alphanumériques Chaines de caractères (texte)
-



Résumé des représentations binaires des informations

Codage de l'information

Le codage d'une information consiste à établir une correspondance entre la représentation externe (A ou 36) et sa représentation interne qui est une suite de bits (01000001, 100100, ...).

Intérêt du système binaire

C'est système de numération utilisant la base 2. Toutes les informations sont codées avec des 0 et des 1.

1 bits : 2^1 possibilités = 0, 1
2 bits : 2^2 possibilités = 00, 01, 10, 11
3 bits : 2^3 possibilités = 000, 001, 010, 011, 100, 101, 110, 111
n bits : 2^n possibilités
Un mot = un ensemble de bit avec un poids 2^n; 2^{n-1} ; 2^1; 2^0

Avantage :

- Facile à réaliser techniquement. En électronique ces 2 états corresponde a l'existence ou non d'une tension (+5V=1 et 0V=0).
- Operations fondamentales faciles a effectué (circuits logiques)
- Arithmétique binaire peut être réalisée à partir de la logique symbolique

I. Instructions

Elles sont écrites en langage machine. Les instructions sont composées par plusieurs champs :

- Le code de l'opération a effectué
- Les opérandes de l'opération

Le codage dépend du processeur. Le décodage est réalisé par l'unité de commande et le nombre d'instructions est limité {processeur CISC/RISC (Complex/Reduced Instruction-Set Computer)}

Code opération	Adresses opérandes

II. Données

Données non numériques : (codage assez simple car aucune opération arithmétique ou logique ne sera appliquée sur ces données, une table de correspondance suffit. Des codages binaires normalises ont été établis (BCD, ASCII, Unicode, etc.)

- Code ASCII (American Standard Code for Information Interchange) : chaque caractère est codé sur 8bits(7 bits : (128 caractères) et 1 bit utilise pour le contrôle de parité)
- Code BCD sur 6bits : chaque caractère est codé sur 6bits

Données numériques

Codage complexe qui doit faciliter la mise en place de circuits réalisant les opérations arithmétiques.

- Nombres entiers non signés : 0 ; 1 ; 315

- Nombres entiers signés : +1 ; -1255 ; +25
- Nombres fractionnaires : +3,1415 ; -0,5

A. Entiers non signés

Plusieurs codes binaires existent pour représenter un nombre entier non signé :

- Binaire pur
- Décimaux codes binaire (BCD) : Chaque chiffre d'un nombre est code individuellement en son équivalent binaire sur 4 bits (15 = 0001'0101)
- Code excédent 3 : BCD+3 a chaque chiffre
- Code 2 dans 5 : Chaque chiffre d'un nombre est code individuellement dur 5bits dont deux sont des 1
- Code biquinaire : Chaque chiffre d'un nombre est code individuellement sur 7bits sur les 5 premiers bits un seul doit être égale a 1 et sur les deux derniers bits un seul doit être égale a 1.

Avantage :
Operations d'entrées / sorties plus faciles
Inconvénient
Operations arithmétiques compliquées

B. Entiers signés

Plusieurs façons de représenter un nombre signé:

- Valeur absolue signée
- Complément `a 1 (ou logique)
- Complément `a 2 (ou arithmétique)

1. **Valeur absolue signée** : Les nombres sont codes comme des entiers positifs sur n bits mais on sacrifie un bit (celui de poids fort) pour coder le signe :

Bit n−1 : pour le signe (signe + = 0, signe - = 1)
Bits n−2...0 : pour la valeur absolue
Exemple : représenter +6 et -6 sur 4 bits : +6= 0110 / -6 = 1110
Avantage : Symétrie : autant de chiffres négatifs que de positifs
Inconvénients :

- 2 représentations du 0 : 0000000 et 10000000
- Bit de signe doit être traite de façon particulière => opérations arithmétiques compliquées.

2. **Complément `a 1** : Les nombres positifs sont codes comme précédemment, les négatifs sont obtenus en remplaçant tous les bits `a 1 par 0 et vice-versa. Bit n−1 pour le signe (signe + = 0 , signe - = 1) Bits n−2...0 pour les positifs et leurs compléments :

Exemple : +6 = 0110 / -6 = 1001

Avantages : Symétrie, Autant de négatifs que de positifs, Une soustraction se réduit à l'addition de son complément

Inconvénient : 2 représentations du 0 : 0000000 et 11111111 , Bit de retenue à reporter lors de l'addition

3. Complément `a 2

On obtient le complément à 2 en ajoute 1 au complément `a 1.
Les nombres positifs sont codes comme précédemment, les négatifs sont remplacés par leurs complément à 2.
Exemple : 6 = 0110, le complément à 2 de -6 : $C_2(6)= C_1(6)+1= 1001 + 1 = 1010$
Avantage : Une seule représentation du zéro Une soustraction se réduit `à l'addition de son complément, Pas de report du bit de retenue pour l'addition

Inconvénient : Dissymétrie : plus de négatifs que de positifs

C. Nombres fractionnaires

Deux méthodes sont utilisées : la représentation en virgule fixe et la représentation en virgule flottante.

- Représentation en virgule fixe

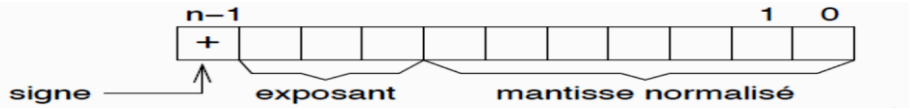
L'ordinateur ne possède pas de virgule au niveau de la machine. Les nombres fractionnaires sont traités comme des entiers avec une virgule virtuelle gérer par le programme.

- Représentation en virgule flottante : un nombre fractionnaires peut être représenté sous forme d'un nombre sous la forme d'un produit de 2 facteurs $N = M \times B^E$

B : base, M : mantisse (nombre purement fractionnaire i.e., 0,xxx), E : exposant

Exemple : $-0{,}23643\times10^{+3} = -236{,}43$. Exposant et mantisse peuvent être signes.

L'exposant est codé en représentation décalée ou biaise (ou en excédent) sur k bits(le décalage est de 2^{k-1} et la mantisse en valeur absolue signée sur $(n-k-1)$ bits.



Operations arithmétiques en virgule flottante

- multiplication : additionner les exposants, multiplier les mantisses et ré-normaliser le résultat division : soustraire les exposants, diviser les mantisses et ré-normaliser le résultat
- Addition : de-normaliser la plus petite valeur d'exposant, additionner les mantisses, ré normaliser le résultat
- Soustraction : de-normaliser la plus petite valeur d'exposant, soustraire les mantisses, ré normaliser le résultat

Il peut être nécessaire d'arrondir la mantisse (i.e., perte de précision).

Dépassement de capacité et sous-passement de capacité peuvent se produire si l'exposant devient trop grand ou trop petit.

Support de cours

Module Microprocesseurs et DSP

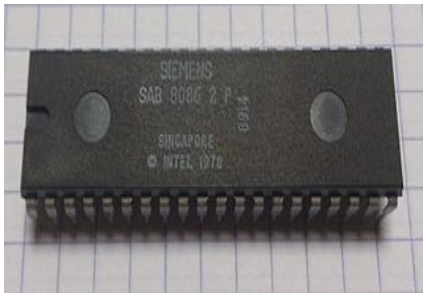
Partie :2

Etude et programmation d'un microprocesseur 16 bits

Le Microprocesseur 8086

Le microprocesseur Intel 8086 est un microprocesseur 16 bits, développé par Intel en 1978. C'est le premier microprocesseur de la famille Intel 80x86 (8086, 80186, 80286, 80386, 80486, Pentium, ...). C'est un processeur CISC 16 bits. Il se présente sous la forme d'un boîtier DIP (Dual In-line Package) à 40 broches.
Ses caractéristiques principales sont les suivantes :

- Bus de données d'une largeur de 16 bits.
- Bus d'adresses de 20 bits, ce qui permet d'adresser un total de 1 mégaoctet de mémoire.
- 14 registres de 16 bits



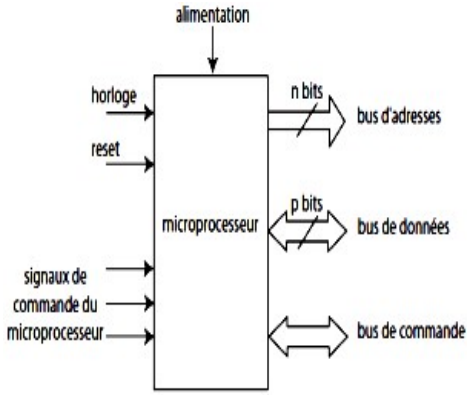


Schéma fonctionnel d'un microprocesseur

I. Description matérielle d'un microprocesseur

Un microprocesseur se présente sous la forme d'un circuit intégré muni d'un nombre généralement important de broches. Exemples :
• Intel 8085, 8086, Zilog Z80 : 40 brooches, DIP (Dual In-line Package) ;
• Motorola 68000 : 64 broches, DIP ;
• Intel 80386 : 196 broches, PGA (Pin Grid Array).

Technologies de fabrication : NMOS, PMOS, CMOS.

II. Description des Pins

Vcc et GND : assure l'alimentation électrique du microprocesseur.
CLK : entrée destinée à recevoir le signale de l'horloge système, qui cadence le fonctionnement du microprocesseur. Ce signal provient d'un générateur d'horloge
READY : permet la synchronisation des mémoires et périphériques lents avec le CPU 8086. Il est élaboré à partir du RDY qui est validé par AEN.
RESET: un signale de remise à l'état initiale est généré à partir d'un signal externe RES qui se synchronise avec l'horloge CLK. Ce signale doit rester à l'état haut pendant au moins quatre cycles d'horloge, pour permettre au CPU 8086 de s'initialiser correctement, en commençant l'exécution à partir de l'adresse FFFF:0000.
MN/$\overline{MX}$: Sélectionne entre l'un des deux modes de fonctionnement du 8086.

- Mode minimum: le CPU 8086 fonctionne d'une manière autonome et en monoprocesseur et il génère par lui même les signaux de bus de commande.
- Mode maximum : Ces signaux sont produit par un contrôleur de bus 8288, ce qui lui permet d'opérer dans un environnement multiprocesseur.

$\overline{TEST}$: entrée de synchronisation entre le CPU 8086 et le coprocesseur
NMI, INTR: entrées de demande d'interruption. INTR: interruption normale, NMI (Non Masquable Interrupt) : interruption prioritaire.
$\overline{INTA}$ (Inerrupt Acknowledge) : indique que le microprocesseur a pris en compte l'interruption.
HOLD: entrée de demande d'accès au bus.
HLDA: indique que le microprocesseur a pris en compte la demande d'accès au bus.

S0 à S7: Signaux d'état indiquant le type d'opération sur le bus.
A16/S3 à A19/S6: 4 bits de poids fort du bus d'adresses, multiplexés avec 4 bits d'état.
AD0 à AD15: 16 bits de poids faible du bus d'adresses, multiplexés avec 16 bits de données, d'où la nécessité d'un multiplexage pour obtenir séparément les bits d'adresse et de données.
$\overline{RD}$ (Read): signal de lecture d'une donnée.
$\overline{WR}$ (Write): Signal d'écriture d'une donnée.
M/$\overline{IO}$ (Memory /Input-Output): indique que le CPU 8086 adresse la mémoire ou les unités d'entrées/sorite.
$\overline{DEN}$(Data ENable): indique que la donnée est disponible sur le bus de données.
ALE (Adress Latch Enable) : indique que l'adresse est disponible sur le bus d'adresses.
DT/$\overline{R}$ (Data Transmit/ Receive): indique le sens de transfert des données.
$\overline{BHE}$ (Bus High Enable): Signale de validation de l'octet du poids fort du bus de données.

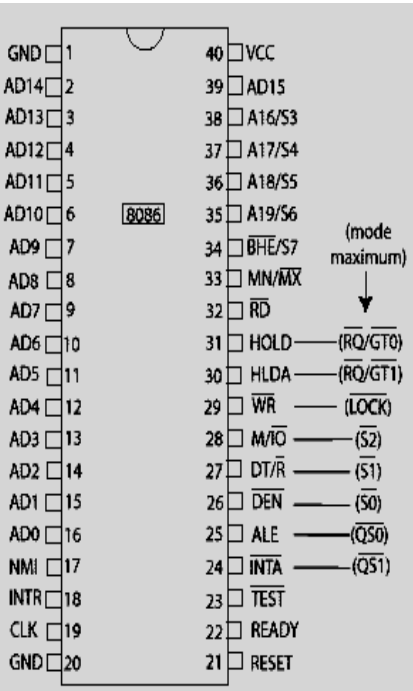


Figure 1: brochage du CPU 8086

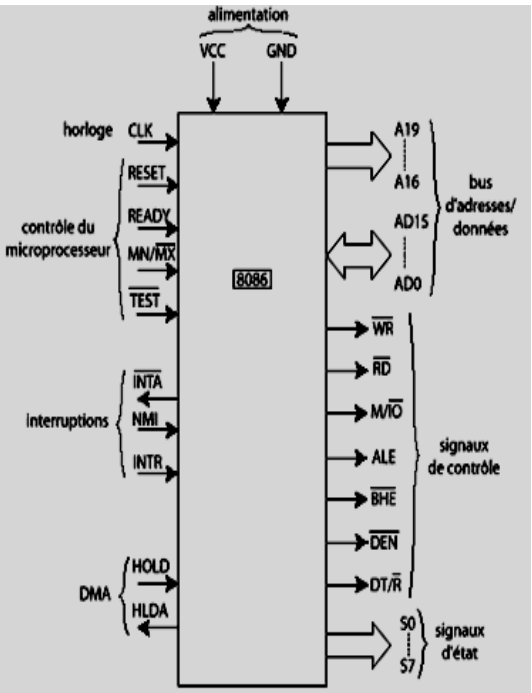


Figure 2: Schéma fonctionnel du CPU 8086

III. Démultiplexage du bus adresses/données (A/D)

Le démultiplexage des signaux AD0 àAD15 (ou A16/S3 à A19/S6) se fait en mémorisant l'adresse lorsque celle-ci est présente sur le bus A/D, à l'aide de circuits **verrous** (latch circuit) 74373, formé de bascules D. La commande de mémorisation de l'adresse est générée par le microprocesseur : c'est le signal **ALE.**

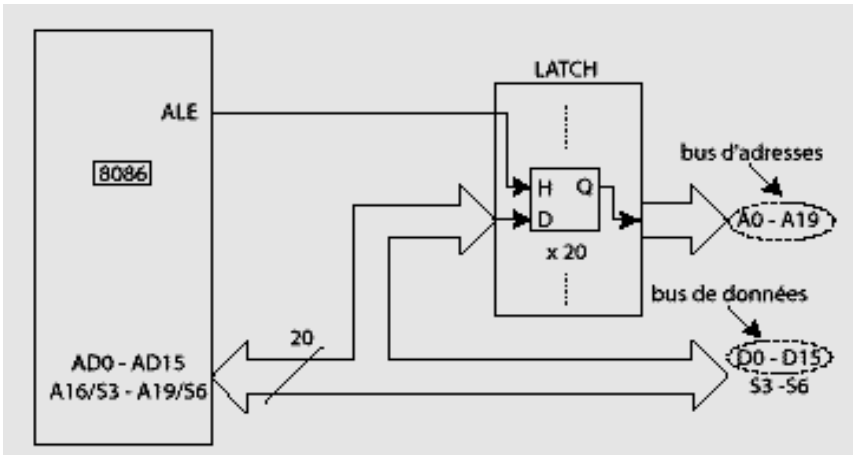


Figure 3: Principe de Circuit de démultiplexage A/D

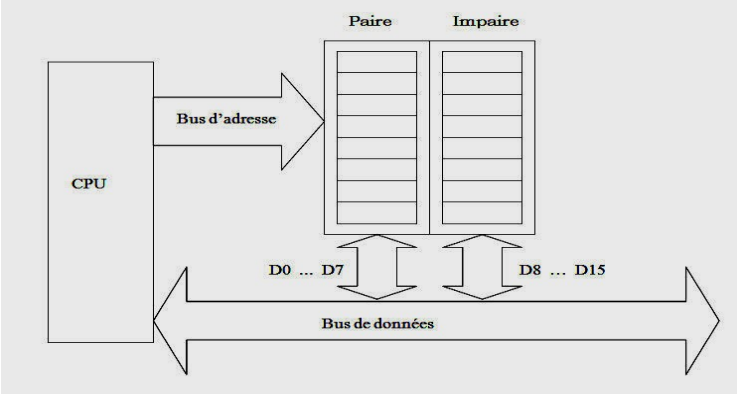
Si ALE = 1, le verrou est transparent (Q = D) ;

Si ALE = 0, mémorisation de la dernière valeur de D sur les sorties Q.

IV. Interfaçage de la mémoire

Le CPU 8086 possède un bus d'adresses de 20 lignes, permettant d'adresser un espace de 1Moctets, organisé en deux banques de 512 Ko :

- la banque **inférieure** (ou **paire**);
- la banque **supérieure** (ou **impaire**).



Ces deux banques, figure
, sont sélectionnées par :

- A0 pour la banque paire qui contient les octets de poids faible;
- BHE pour la banque impaire qui contient les octets de poids fort.

Seuls les bits A1 à A19 servent à désigner une case mémoire dans chaque banque de 512 Ko. Le microprocesseur peut ainsi lire et écrire des données sur 8 bits ou sur 16 bits selon A0 et BHE, tableau 1.

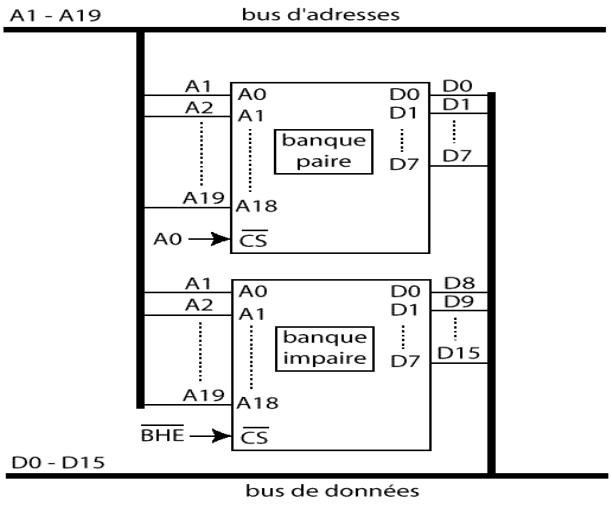


Figure 4a: Organisation de la mémoire

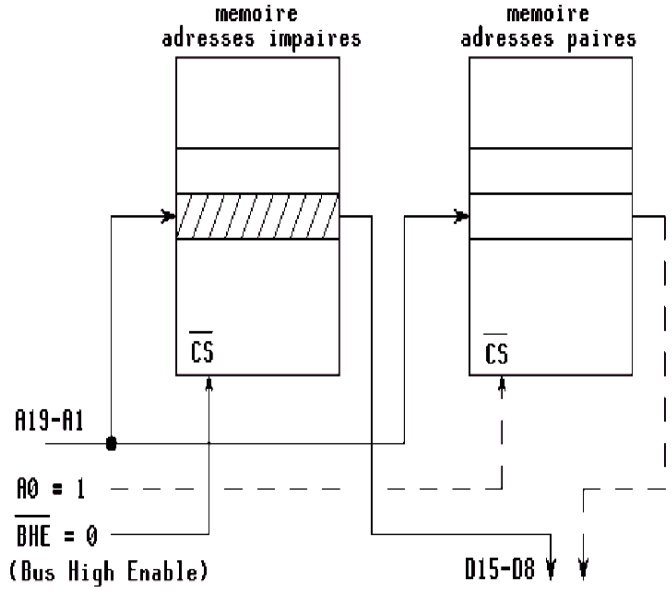
BHE	A0	Octets transférés
0	0	Les deux octets (mot complet)
0	1	Octet de poids fort (adresse impaire)
1	0	Octet de poids faible (adresse paire)
1	1	Aucun octet

Tableau 1.

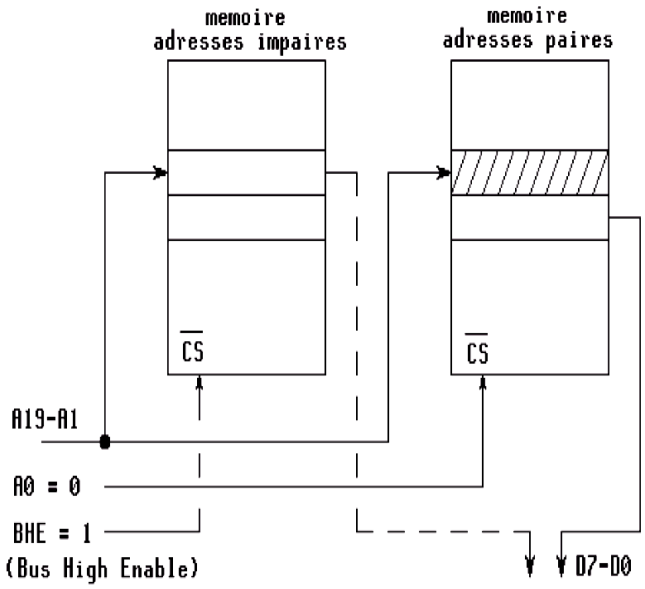
La lecture (écriture) d'un mot d'adresse paire nécessite un seul accès, par contre celle d-un mot d'adresse impaire nécessitera deux accès, les deux octets du mot ne sont pas alignés.
Le signal BHE/ permet de sélectionner le banc impair, pour cela il prendra la valeur 0
Le tableau suivant montre l'utilité du signal BHE/

Pour lire (écrire) un mot de 16 bits qui se trouve à une adresse impaire, deux accès mémoire sont nécessaires. Le premier cycle mémoire correspond à la lecture du poids faible de la donnée, tandis que le second cycle permet la lecture du poids fort de la donnée, Ceci est illustré par le schéma suivant :

Lecture d'un mot situé à une adresse impaire,
1er cycle : Lecture du premier octet situé à une adresse impaire,:
2eme cycle : Lecture de l'octet suivant situé à une adresse paire



1er cycle



2eme cycle

Le 8086 ne peut lire une donnée sur 16 bits en une seule fois, uniquement si l'octet de poids fort de cette donnée est rangé à une adresse impaire et l'octet de poids faible à une adresse paire (alignement sur les adresses paires), sinon la lecture de cette donnée doit se faire en deux opérations successives, d'où une augmentation du temps d'exécution du transfert dû à un mauvais alignement des données.

V. Organisation interne du CPU 8086

Le CPU 8086 est constitué de deux unités, figure 5, fonctionnant en parallèle :

- l'**unité d'exécution** (EU : Execution Unit) ;
- l'**unité d'interface de bus** (BIU : Bus Interface Unit).

L'unité d'exécution (EU) exécute les instructions arithmétique ou logique contenues dans la file d'attente.
L'unité d'interface de bus (BIU) recherche les instructions en mémoire et stocke par anticipation 6 octets instructions dans une **file d'attente.**
Les deux unités fonctionnent simultanément, ainsi, pendant que l'unité d'exécution traite une instruction, la BIU recherche dans la mémoire les octets composant la prochaine instruction à exécuter. Ce mode de fonctionnement, dit **pipeline**, accélère le processus d'exécution d'un programme, en réduisant le temps mort de l'exploitation du système de bus.

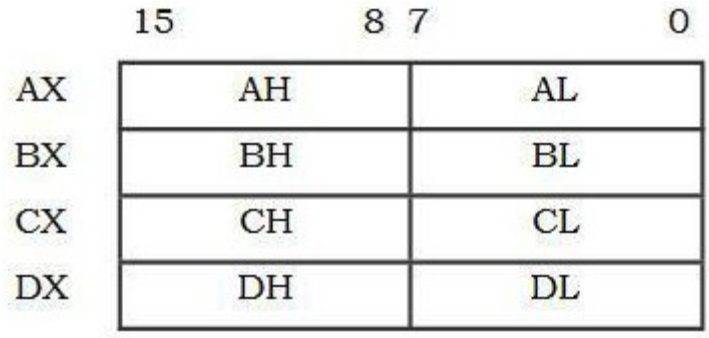
V.1. Les registre du CPU 8086

Le jeu de registres contient l'ensemble des registres du microprocesseur. Un registre est une petite partie de mémoire intégrée au microprocesseur, dans le but de recevoir des informations spécifiques, notamment des adresses et des données stockées durant l'exécution d'un programme. Il existe plusieurs types de registres. Certains d'entre eux sont affectés à des opérations d'ordre général et sont accessibles au programmeur à tout moment. Nous disons alors qu'il s'agit de registres généraux. D'autres registres ont des rôles bien plus spécifiques et ne peuvent pas servir à un usage non spécialisé.

Le CPU 8086 contient 14 registres répartis en 4 groupes :

1.. Registres généraux :
4 registres sur 16 bits.
AX =(**AH**,**AL**); **BX** =(**BH**,**BL**); **CX** =(**CH**,**CL**); **DX** =(**DH**,**DL**).

Ils peuvent être également subdivisés en deux registres de 8 bits supérieurs et inférieurs, et être référencés sous ***x*H** et ***x*L:** par exemple AH et AL, pour AX. En d'autre termes **AX**=256***AH**+**AL**.



Ils servent à contenir temporairement des données. Ce sont des registres généraux mais ils peuvent être utilisés pour des opérations particulières.
AX comme accumulateur ;
BX comme registre de base pour l'adressage basé ;
CX comme compteur pour traitement en boucle;
DX Sert aussi pour quelques opérations mathématiques.

- Registre AX : (Accumulateur)

Toutes les opérations de transferts de données avec les entrées-sorties ainsi que le traitement des chaînes de caractères se font dans ce registre, de même les opérations arithmétiques et logiques.
Les conversions en BCD du résultat d'une opération arithmétique (addition, soustraction, multiplication et la division) se font dans ce registre.

- **Registre BX : (registre de base)**

Il est utilisé pour l'adressage de données dans une zone mémoire différente de la zone code : en général il contient une adresse de décalage par rapport à une adresse de référence.). (Par exemple, l'adresse de début d'un tableau). De plus il peut servir pour la conversion d'un code à un autre.

- **Registre CX : (Le compteur)**

Lors de l'exécution d'une boucle on a souvent recours à un compteur de boucles pour compter le nombre d'itérations, le registre CX a été fait pour servir comme compteur lors des instructions de boucle.
Remarque :
Le registre CL sert en tant que compteur pour les opérations de décalage et de rotation, dans ce cas il va compter le nombre de décalages (rotation) de bits à droite ou à gauche.

- **Registre DX :**

On utilise le registre DX pour les opérations de multiplication et de division mais surtout pour contenir le numéro d'un port d'entrée/sortie pour adresser les interfaces d'E/S.

2. Registres de pointeurs et d'index (déplacement):
4 registres sur 16 bits.
a) Pointeurs :
- SP : (Stack Pointer), pointeur de pile, pointe sur le sommet de la pile.
- BP : (Base Pointer), pointeur de base, pointe sur l'adresse de base de la pile
b) Index :

- **SI** : Source Index ;
- **DI** : Destination Index.

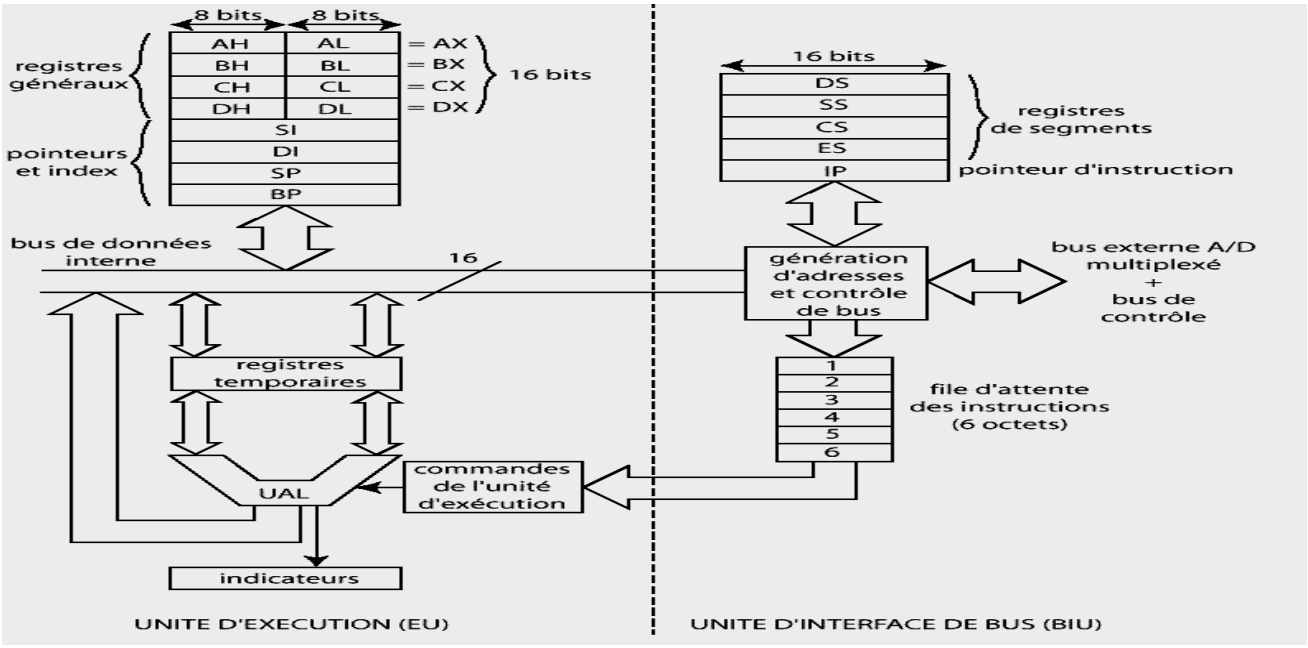


Figure 5. Les unités fonctionnelles du microprocesseur 8086.

Ils sont utilisés pour les transferts de chaînes d'octets entre deux zones mémoire. Les registres pointeurs et d'index contiennent des adresses de cases mémoire et sont appelés aussi registre de déplacement.
Ces registres sont plus spécialement adaptés au traitement des éléments dans la mémoire. Ils sont en général munis de propriétés d'incrémentation et de décrémentation.

- **le pointeur SP de pile** (Stack Pointer : SP). Ce registre permet de pointer la pile pour stocker des données ou des adresses selon le principe du "Dernier Entré Premier Sorti" ou "LIFO"

(Last In First Out).

- **L'idexe SI** : (source indexe) :

Il permet de pointer la mémoire il forme en général un décalage (un offset) par rapport à une base fixe (le registre DS), il sert aussi pour les instructions de chaîne de caractères, en effet il pointe sur le caractère source

- **L'indexe DI** : (Destination indexe) :

Il permet aussi de pointer la mémoire il presente un décalage par rapport à une base fixe (DS ou ES), il sert aussi pour les instructions de chaîne de caractères, il pointe alors sur la destination

- **Les pointeurs SP et BP** : (Stack pointer et base pointer)

Ils pointent sur la zone pile (une zone mémoire qui stocke l'information avec le principe filo : voir plus loin), ils presentent un décalage par rapport à la base (le registre SS). Pour le registre BP il a un rôle proche de celui de BX, mais il est généralement utilisé avec le segment de pile.

3.Les registres segment:

Le 8086 a quatre registres segments de 16 bits chacun : CS (code segment, DS (Data segment), ES (Extra segment) et SS (stack segment), ces registres sont chargés de sélectionner les différents segments de la mémoire en pointant sur le début de chacun d'entre eux. Chaque segment de mémoire ne peut excéder les 65535 octets.
CS : (Code Segment), registre de segment de code ;
DS : (Data Segment), registre de segment de données ;
SS : (Stack Segment), registre de segment de pile ;
ES : (Extra Segment), registre de segment supplémentaire pour les données ;

Les registres de segments, associés aux pointeurs et aux index, permettent au CPU 8086 d'adresser l'ensemble de la mémoire.

- **Le registre CS (code segment) :** Il pointe sur le segment qui contient les codes des instructions du programme en cours.

Remarque :
Si la taille du programme dépasse les 65535 octets alors on peut diviser le code sur plusieurs segments (chacun ne dépasse pas les 65535 octets) et pour basculer d'une partie à une autre du programme il suffit de changer la valeur du registre CS et de cette manière on résout le problème des programmes qui ont une taille supérieure à 65535 octets.

- **Le registre DS (Data segment) :**

Le registre segment de données pointe sur le segment des variables globales du programme, bien évidemment la taille ne peut excéder 65535 octets (si on a des données qui dépassent cette limite, on utilise la même astuce citée dans la remarque précédente mais dans ce cas on change la valeur de DS).

- **Le registre ES (Extra segment)** :

Le registre de données supplémentaires ES est utilisé par le microprocesseur lorsque l'accès aux autres registres est devenu difficile ou impossible pour modifier des données, de même ce segment est utilisé pour le stockage des chaînes de caractères.

- **Le segment SS (Stack segment)** :

Le registre SS pointe sur la pile : la pile est une zone mémoire ou on peut sauvegarder les registres ou les adresses ou les données pour les récupérer après l'exécution d'un sous programme ou l'exécution d'un programme d'interruption, en général il est conseillée de ne pas changer le contenu de ce registre car on risque de perdre des informations très importantes (exemple les passages d'arguments entre le programme principal et le sous programme)

4. **. Registre pointeur d'instruction et registre d'état :**

2 Registre sur 16 bits.

- **Le registre IP** : (Le compteur de programme) :

Instruction Pointer ou Compteur de Programme, contient l'adresse de l'emplacement mémoire où se situe la prochaine instruction à exécuter. Autrement dit, il doit indiquer au processeur la prochaine instruction à exécuter. Le registre IP est constamment modifié après l'exécution de chaque instruction afin qu'il pointe sur l'instruction suivante.

- **Le registre d'état (Flag) :** Ce registre est manipulé bit par bit. Il offre 16 bits dont seulement 9 sont utilisés:

Le registre d'état FLAG sert à contenir l'état de certaines opérations effectuées par le processeur. Par exemple, quand le résultat d'une opération est trop grand pour être contenu dans le registre cible (celui qui doit contenir le résultat de l'opération), un bit spécifique du registre d'état (le bit OF) est mis à 1 pour indiquer le débordement.
Remarque : Drapeaux (flags)
Les drapeaux sont des indicateurs qui annoncent une condition particulière suite à une opération arithmétique ou logique.
Le registre d'état du 8086 est formé par les bits suivants :

Remarque :
X : bit non utilise.
CF (Carry Flag) :
Retenue : cet indicateur et mis à 1 lorsque il y a une retenue du résultat à 8 ou 16 bits. il intervient dans les opérations d'additions (retenue) et de soustractions (borrow) sur des entiers naturels. Il est positionné en particulier par les instructions ADD, SUB et CMP
(comparaison entre deux valeurs).
CF = 1 s'il y a une retenue après l'addition ou la soustraction du bit de poids fort des opérandes. Exemples (sur 8 bits pour simplifier) :

PF (Parity Flag) :
Parité : si le résultat de l'opération contient un nombre pair de 1 cet indicateur est mis à 1, sinon zéro.
AF (Auxiliary Carry) :
Demie retenue : Ce bit est égal à 1 si on a une retenue du quarter de poids faible dans le quarter de poids plus fort.
ZF (Zero Flag) :
Zéro : Cet indicateur est mis à 1 quand le résultat d'une opération est égal à zéro. Lorsque l'on vient d'effectuer une soustraction (ou une comparaison), ZF=1 indique que les deux opérandes étaient égaux. Sinon, ZF est positionné à 0.
SF (Sign Flag) :
SF est positionné à 1 si le bit de poids fort du résultat d'une addition ou soustraction est 1 ; sinon SF=0. SF est utile lorsque l'on manipule des entiers signés, car le bit de poids fort donne alors le signe du résultat. Exemples (sur 8 bits) :

OF (Overflow Flag) :
Débordement : si on a un débordement arithmétique ce bit est mis à 1.c a d le résultat d'une opération excède la capacité de l'opérande (registre ou case mémoire), sinon il est à 0.
DF (Direction Flag) :
Auto Incrémentation/Décrémentation : utilisée pendant les instructions de chaîne de caractères pour auto incrémenter ou auto décrémenter le SI et le DI.
IF(Interrupt Flag) :
Masque d'interruption : pour masquer les interruptions venant de l'extérieur ce bit est mis à 0, dans le cas contraire le microprocesseur reconnaît l'interruption de l'extérieur.
TF (Trap Flag) :
Piége : pour que le microprocesseur exécute le programme pas à pas du.
Remarque :
Les instructions de branchements conditionnels utilisent les indicateurs (drapeaux), qui sont des bits spéciaux positionnés par l'UAL après certaines opérations. Chaque indicateur est manipulé individuellement par des instructions spécifiques

R	**R**	**R**	**R**	**O**	**D**	**I**	**T**	**S**	**Z**	**U**	**A**	**U**	**P**	**U**	**C**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

En peut regrouper les bits du registre d'état en deux catégories:
a) Indicateurs d'état:

CF : (carry flag), indicateur de retenue. Mis à 1 si un calcul produit une retenue ;
PF : (prity flag), indicateur de parité. Mis à 1 si les 4 bits de poids faible du résultat contiennent un ;
AF : (auxillary flag), indicateur de retenue auxiliaire. Mis à 1 si un calcul en BCD non compacté produit une retenue. ;
ZF : (zero flag), indicateur de zéro. Mis à 1 si le résultat d'un calcul (i.e. le contenu de l'accumulateur) vaut 0.
SF : (signe flag), indicateur de signe. Prend la valeur du bit de poids fort de l'accumulateur après un calcul. ;
OF : (overflow flag), indicateur de dépassement. Mis à 1 si une opération provoque un dépassement de capacité.

b) Indicateurs de contrôle:
TF : (trace(trap) flag), indicateur d'exécution pas à pas. Mis à 1, il force le processeur à fonctionner pas à pas ;
IF : (interruption flag), indicateur d'autorisation d'interruption. Mis à 1, il autorise les interruptions. S'il vaut 0, il les empêche. ;
DF : (direction flag), indicateur de direction. Fixe le sens (incrémentation et décrémentation) dans lequel seront effectuées les opérations de traitement de chaînes de caractères. STD le met à 1, CLD à 0.
R : bit réservé ;
U : bit indéfini ;

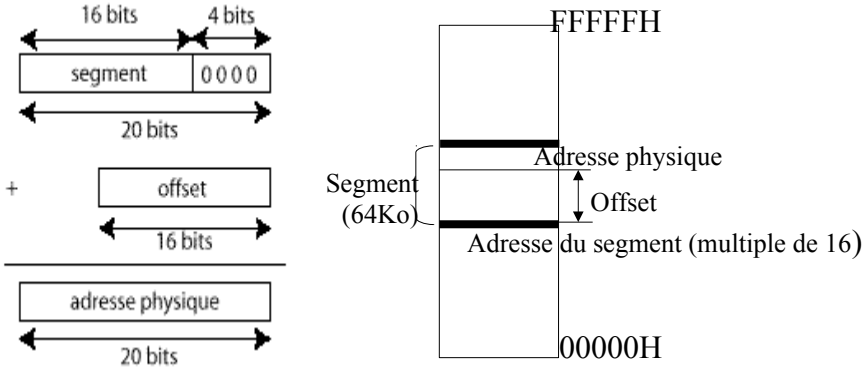
V. Gestion de la mémoire par le CPU 8086

L'espace mémoire adressable par le 8086 est de 1 Mo. Le pointeur d'instruction fait 16 bits donc il y a possibilité d'adresser 2^{16} = 64 Ko (ce qui ne couvre pas la mémoire). La stratégie de gestion de la mémoire consiste à diviser l'espace mémoire adressable par le CPU 8086 en **segments**. Un segment est une zone mémoire de 64 Ko définie par son adresse de départ qui doit être un multiple de 16.
Pour désigner une case mémoire parmi les 2^{16} contenues dans un segment, il suffit d'une valeur sur 16 bits.
Ainsi, une case mémoire est repérée par le CPU 8086 au moyen de deux quantités sur 16 bits :
1) l'adresse d'un segment ;
2) un déplacement ou **offset** (appelé aussi **adresse effective**) dans ce segment.
Cette méthode de gestion de la mémoire est appelée **segmentation de la mémoire**.

- La donnée d'un couple (segment,offset) définit une **adresse logique**, notée sous la forme **segment : offset**.
- L'adresse d'une case mémoire donnée sous la forme d'une quantité sur 20 bits (5 digits hexa) est appelée **adresse physique**, figure 7, car elle correspond à la valeur envoyée réellement sur le bus d'adresses A0 - A19.

L'adresse physique se calcule par l'expression : **Adresse physique = 16 × segment + offset**

On a la multiplication par $16 = 2^4$ revient à effectuer un décalage de 4 positions vers la gauche donc le calcule s'effectue comme suit :



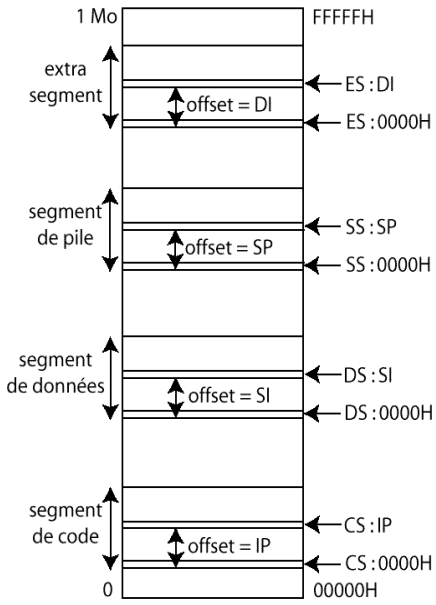


Figure 7a. Principe de calcul des adresses mémoire

Exemple :

Pour CS=1000 et IP = 2006

Adresse physique =

10000
\+ 2006
= 12006

Le CPU 8086 peut accéder à 4 type de segments, figure 8, dont leur adresses se trouve dans les registres de segment:

1. **Segment de code**: contient les instructions du programme. Son adresse est dans le registre CS qui est associé au pointeur d'instruction IP, ainsi la prochaine instruction à exécuter se trouve à l'adresse logique CS:IP.

2. **Segment de données** : contient les données manipulées par le programme. Son adresse se trouve dans le registre de DS

3. **Segment supplémentaire:** peut contenir des données. Les registres de segments DS et ES peuvent être associés à un registre d'index. Exemple: DS : SI, ES : DI.

4. **Segment de pile** : contient la pile de sauvegarde. Le registre SS peut être associé au registres de pointeur de pile (SS:SP; SS:BP)

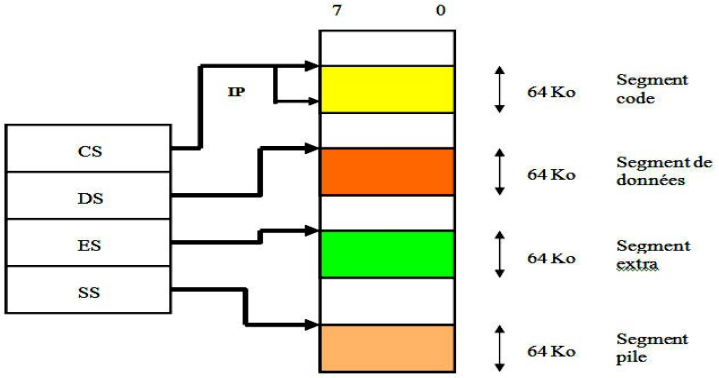


Figure 7b. Principe de calcul des adresses mémoire

Support de cours

Module Microprocesseurs et DSP

Partie :3

Etude et programmation d'un microprocesseur 16 bits

Le Microprocesseur 8086

Programmation en langage d'assemblage du CPU 8086

Le 8086 est programmable dans un langage d'assemblage comportant des instructions utilisant les registres, les flags, la mémoire et d'autres éléments de l'ordinateur.
L'assembleur permet de contrôler directement la CPU. Cela permet d'avoir une total maîtrise du système et surtout permet de faire des programmes rapides par rapport aux langages de haut niveau (C++, Basic, Pascal ...). En effet, bien que ces langages permettent de faire des programmes facilement et rapidement, ils n'optimisent pas le code d'exécution. Cela engendre donc des programmes (beaucoup) plus volumineux.

Organisation d'une ligne de programme d'assemblage
Le format d'une ligne de programme est :

Etiquette : mnémonique opérandes ; commentaire

- **Etiquette :** (optionnel) représente l'adresse de l'instruction. Elle permet d'y faire référence ailleurs dans le programme (dans les instructions de branchement). Elles sont formées de chiffres, de lettres et des caractères.
- **Mnémonique**: Souvent c'est la compression d'un mot ou d'une expression en anglais présentant l'action de l'instruction. Par exemple l'instruction MUL permet la multiplication (MULtiply). Elles sont constituées de 2à 6 lettres comme JZ et LOOPNZ.
- **Opérandes** sur lesquelles on veut agir. Ils peuvent être des registres, des emplacements de mémoire ou des constantes (valeurs immédiates). On peut trouver des instructions à opérande implicite : MUL BH.

En générale dans une instruction : **Mnémonique op1, op2 .**
op1 : représente la destination. C'est l'endroit ou le résultat sera stocké.
op2 : représente la source. C'est la donnée qui sera éventuellement combinée avec op1 pour calculer le résultat de l'instruction.

Dans la plupart des instructions, la source et la destination doivent être de même taille.
On peut trouver des instructions à opérande implicite : MUL BH.
On ne peut pas utiliser tous les types d'opérandes n'importe où. Les types autorisés selon l'instruction et nombre d'opérandes sont donnés sur le tableau 2.

	2 opérandes	**1 opérande**
Avec tous les opérateurs	R, R R, I R, M M, R M, I	 R M
Pour MOV, on a aussi	R, S M, S S, R S, M	
Pour PUSH et POP, on a		S

Où :
S un registre de segment (CS, DS, ES, SS)
R un registre ordinaire (yX, yH, yL, SI, DI, BP, SP)
I un immédiat (78h, 54, ...)
M un des modes d'adressage de la mémoire

Les modes d'adressage de la mémoire

–En adressage direct, on indique l'adresse d'un emplacement en mémoire principale en hexadécimal entre crochets.
Exemple : MOV AX, [A340]

–En adressage relatif, on indique simplement l'adresse (hexa). L'assembleur traduit automatiquement cette adresse en un déplacement (relatif sur un octet).

Exemple: JNE 0108

–En adressage indirect, on indique une expression - en fonction du contenu des registres- indiquant l'adresse où se trouve l'adresse d'un emplacement en mémoire principale.
Pour l'adressage indirect, on peut utiliser exclusivement les registres : BX,BP, SI et DI.
La forme générale de l'adressage indirect est :
[registre de base +/- registre index +/- déplacement]
tel que : registre de base = BX, BP
registre d'index = SI, DI
déplacement = une constante

Les modes d'adressage qui en découlent sont :
[déplacement]
[reg. base] : [BX] ;
[reg. index] : [SI], [DI];
[reg. Base +/- reg. index] : [BX +/- SI], [BX +/- DI], [BP +/- SI], [BP +/- DI] ;
[reg. Base +/- déplacement] : [BX +/- déplacement], [BP +/- déplacement] ;
[reg. Index +/- déplacement] : [SI +/- déplacement], [DI +/- déplacement] ;
[reg. Base +/- reg. Index +/- déplacement] : [BX +/- SI+/- déplacement], [BX +/- DI +/- déplacement], [BP +/- SI +/- déplacement], [BP +/- DI +/- déplacement].

Noter bien que
Les expressions utilisant BX pointent sur la RAM et celles utilisant BP pointent sur la pile.
BX est différent de [BX]: BX correspond au contenu du registre qui peut être une adresse ou autre, alors que [BX] est le contenu de l'emplacement mémoire dont l'adresse est dans BX

Exemple:
MOV SI, 4
MOV AL, [SI]
……
MOV AH, [BX+5]
……
MOV Al, [BX+SI]

Jeu d'instruction

Voici un aperçu des instructions les plus couramment utilisées

Instructions de transfert

Entre registre et mémoire,
MOV a, b a := b
Entre registre et la pile
PUSH a pile := a
PUSHF pile := registre d'état
POP a a := pile
POPF registre de drapeaux := pile
Echange entre registres
XCHG a, b temp := b b := a a := temp
Entre l'unité entrée/sortie et microprocesseur
OUT port, a port := (port entre 0 et FFFF)
IN a, port a := port (port entre 0 et FFFF)
Les adresses sont dans DX. Seul les registres AL et AX sont autorisés pour ces transferts.

```
MOV DX, 100h        MOV DX, 100h
IN AL, DX           MOV AL, 0Fh
                    OUT DX, AL
```

Remarque : pas de transfert entre mémoire et mémoire.

Instructions arithmétiques

- ADD a, b a := a + b sans retenue
- ADC a, b a := a + b avec retenue
- SUB a, b a := a - b sans retenue
- SBB a, b a := a - b avec retenue
- INC a a := a + 1
- DEC a a := a - 1
- NEG a a := -a

- MUL a : AX := AL* a si a est un octet
- DX:AX := AX * a si a est un mot
- IMUL a idem à MUL, mais en signé

- DIV a : AL := AX / a AH := reste si a est un octet
- IDIVa idem à DIV, mais en signé
- AX := DX:AX / a DX := reste si a est un mot

Instructions logiques

AND	a, b	a := a et b
OR	a, b	a := a ou b
XOR	a, b	a := a xor b
NOTa		a := complément à 1 de (a)
NEGa		a := complément à 2 de (a)

Remarque :

1) L'instruction AND permet de masquer des valeurs :

```
MOV AL, 5Dh
AND AL, 0Fh;    ne retient que les 4bits de poids faible de AL
```

2) L'instruction OR peut être utilisée pour forcer des bits à 1

```
MOV AL, 5Dh
OR AL, 0Fh
```

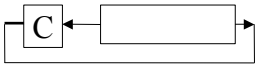
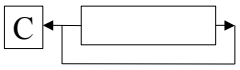
3) L'instruction XOR permet d'inverser les bits d'un registre ou de les mettre plus rapidement à zéro que ne le fait un MOV.

Exemple:

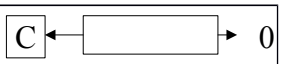
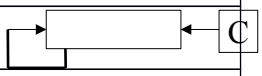
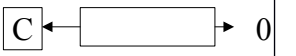
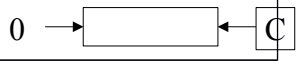
```
XOR AX, AX;          remise à zéro rapide
MOV BL, 10110001b
NOT BL;              BL=01001110b
XOR BL, 0FFh;        BL reprend sa valeur initiale
```

Instructions de décalages, de rotations

a) Rotation :

RCL	a, b	rotation à gauche de b bits de a en passant par CF	
RCR	a, b	rotation à droite de b bits de a en passant par CF	
ROL	a, b	rotation à gauche de b bits de a sans passer par CF	
ROR	a, b	rotation à droite de b bits de a sans passer par CF	

b) Décalage:

SAL	a, b	déplacement à gauche de b bits de a	
SAR	a, b	déplacement à droite de b bits de a	
SHL	a, b	déplacement à gauche de b bits de a	
SHR	a, b	déplacement à droite de b bits de a	

Instructions de test et de comparaison

CMP	a, b	effectue a - b
TEST a,b		effectue a and b
SCASB		AL - ES:DI
SCASW		AX - ES:DI

Instructions de branchement (saut)

Branchement inconditionnel

JMP	a	saut à l'adresse a (saut libre)
LOOP	a	saut à l'adresse a (antérieur)

Exemple :

```
MOV     CX, 100         ; boucle de 100
eti:
        ici, passe 100 fois
        LOOP    eti             ; décrémente CX et boucle si CX ≠ 0
```

Branchement conditionnel

JO a saut à l'adresse a si OF = 1
JC a saut à l'adresse a si CF = 1
JZ a saut à l'adresse a si ZF = 1
JS a saut à l'adresse a si SF = 1
JP a saut à l'adresse a si PF = 1
JNO a saut à l'adresse a si OF = 0
JNC a saut à l'adresse a si CF = 0
JNZ a saut à l'adresse a si ZF = 0
JNS a saut à l'adresse a si SF = 0
JNP a saut à l'adresse a si PF = 0
JE a saut à l'adresse a si a = b (idem à JZ)
JNE a saut à l'adresse a si a ≠ b (idem à JNZ)
JG a saut à l'adresse a si a > b (idem à JNLE)(en non signé: JA, JNBE)
JGE a saut à l'adresse a si a >= b (idem à JNL)(en non signé: JAE, JNB)
JL a saut à l'adresse a si a < b (idem à JNGE)(en non signé: JB, JNAE)
JLE a saut à l'adresse a si a <= b (idem à JNG)(en non signé: JBE, JNA)
JCXZ a saut à l'adresse a si CX=0
LOOPZ a saut à l'adresse a si ZF=0
LOOPNZ a saut à l'adresse a si ZF=1

Instructions de traitement de chaîne caractères

STD Met à 1 DF

CLD clear direction_flag, Met à 0 DF

REP instruction répète CX fois l'instruction (décrémente CX à chaque passage)

Exemple :

```
MOV CX, 7
REP MOVSB    ; exécute movsb 7 fois
```

REPZ instruction répète CX fois l'instruction tant que le ZF vaut zéro

Exemple:

```
MOV CX, 7
REPZ CMPSB   ; répète 7 fois
```

MOVSB, MOVSW Copie de string:

```
ES:[DI] := DS:[SI]
if direction_flag = 0 then
  SI := SI + size;      (size = B, W)
  DI := DI + size;
Else
  SI := SI - size;
  DI := DI - size;
endif;
```

CMPSB, CMPSW Compare chaîne de caractères:

```
CMP DS:[SI], ES:[DI]
if direction_flag = 0 then
  SI := SI + size;      (size = B, W)
  DI := DI + size;
Else
  SI := SI - size;
  DI := DI - size;
endif;
```

Exemple:

```
CLD
MOV CX, 7       ; longueur à comparer
REPZ CMPSB
JNZ  pas_egal
...
pas_egal:
...
```

LODSB, LODSW

```
AL/AX := [DS:SI]
if direction_flag = 0 then
  SI := SI + size;      (size = B, W)
Else
  SI := SI - size;
endif;
```

STOSB, STOSW

```
[ES:DI] := AL/AX
if direction_flag = 0 then
  DI := DI + size;      (size = B, W ou D)
Else
  DI := DI - size;
endif;
```

SCASB, SCASW cherche Ax dans string ES:DI

```
CMP AX/AL, [ES:DI]
if direction_flag = 0 then
  DI := DI + size;      (size = B, W ou D)
Else
  DI := DI - size;
endif;
```

Exemple:

```
CLD
MOV CX, 7       ; longueur du string
MOV AL, 'H'     ; on cherche H
REPNE  SCASB
```

JNZ pas_trouve

Instructions Divers

Conversion : CBW AX := AL “étendu” (converti en Byte en Word)

Autres instructions spécifiques au microprocesseur

NOP pas d'opération

Support de cours

Module Microprocesseurs et DSP

Partie :4

Etude et programmation d'un microprocesseur 16 bits

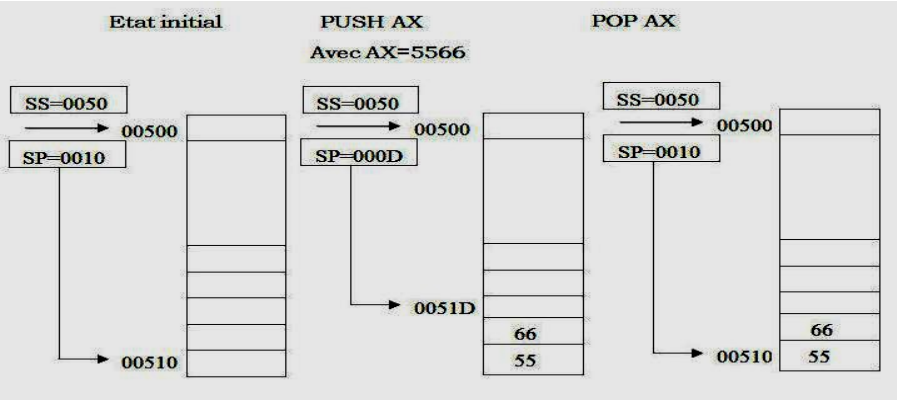
Le Microprocesseur 8086

Gestion de la pile

La programmation d'un CPU nécessite l'usage (d'au moins) une pile surtout pour le passage des paramètres aux sous-programmes, en sauvegardant temporairement le contenu des registres.
La pile est une zone mémoire, définie dans un segment de mémoire particulier (segment de pile) fonctionnant en mode LIFO (Last In, First Out: dernier entré premier sorti). C'est pour cela que tout ce qu'on déposera sur la pile via l'instruction push devra obligatoirement en être retiré dès que possible dans l'ordre inverse à l'ordre de dépôt via l'instruction pop.
Exemple:

```
push ax
push bx
push cx
[...]
pop cx
pop bx
pop ax
```

Le CPU possède deux registres dédiés à la gestion de la pile, SS et SP. SS est normalement initialisé au début du programme et reste fixé par la suite. Le registre SP contient le déplacement du sommet de la pile. Donc, l'adresse logique du dernier élément (sommet de la pile) est posée dans SS:SP. Lorsque l'on ajoute un élément à la pile, l'adresse contenue dans SP est décrémentée de 2 octets (car un emplacement de la pile fait 16 bits de longueur).
La pile et le code (programme) croissent au sens inverse pour diminuer le risque de collision entre code et pile dans le cas où celle-ci est placée dans le même segment que le code (SS=SC).
Le schéma suivant montre comment une valeur est stocker dans la pile (pushed) et comment elle est récupérée (poped) :



Procédures (sous-programme)

Pour éviter la répétition d'une même séquence d'instructions plusieurs fois dans un programme, on rédige cette séquence une seule fois à une adresse et on l'appelle lorsqu'on a besoin. Le programme appelant est le **programme principal**. La séquence d'instructions appelée est le **sous-programme** ou **procédure (subroutine).**

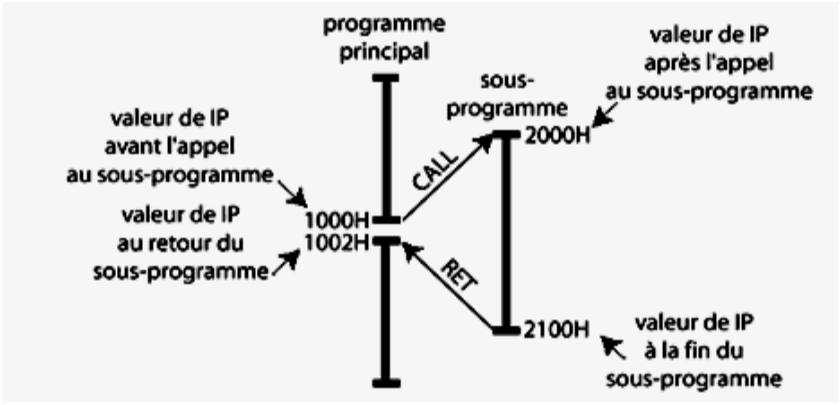


Figure 12. Principe de l'appelle d'un sous-programme.

L'appelle d'une procédure revient à effectuer un saut -comparable à celui effectué par un jmp- vers un sous-programme dont l'adresse est donné en opérande. La différence réside dans le fait qu'il est possible après l'exécution du sous-programme de revenir par l'instruction ret à l'instruction qui suivait le call. Pour ce faire, l'instruction call empile sur le sommet de la pile l'adresse de l'instruction suivant le call afin que l'instruction ret puisse dépiler cette adresse, puis y sauter.

Remarque:
- Lors de l'appel à un sous-programme, Il est très important de remettre la pile dans l'état où on l'avait trouvé avant que ne soit atteinte une instruction ret car les instructions call et ret utilisent la pile pour mémoriser l'adresse à laquelle devra se faire le retour.

Passage de paramètres

Utilisation de la pile pour le passage de paramètres
Pour transmettre des paramètres à une procédure, on peut les placer sur la pile avant l'appel de la procédure, puis celle-ci les récupère en effectuant un adressage basé de la pile en utilisant le registre BP.
Exemple : soit une procédure effectuant la somme de deux nombres et retournant le résultat dans le registre AX :

programme principal :

```
        mov ax,200
        push ax ; empilage du premier paramètre
        mov ax,300
        push ax ; empilage du deuxième paramètre
        call somme ; appel de la procédure somme
```

procédure somme :

```
somme proc
        push bp ; sauvegarde de BP
        mov bp,sp ; faire pointer BP sur le sommet de la pile
        mov ax,[bp+4] ; récupération du deuxième paramètre
        add ax,[bp+6] ; addition au premier paramètre
        pop bp ; restauration de l'ancienne valeur de BP
        ret 4 ; retour et dépilage des paramètres
somme endp
```

L'instruction ret 4 permet de retourner au programme principal et d'incrémenter le pointeur de pile de 4 unités pour dépiler les paramètres afin de remettre la pile dans son état initial, figure 13.

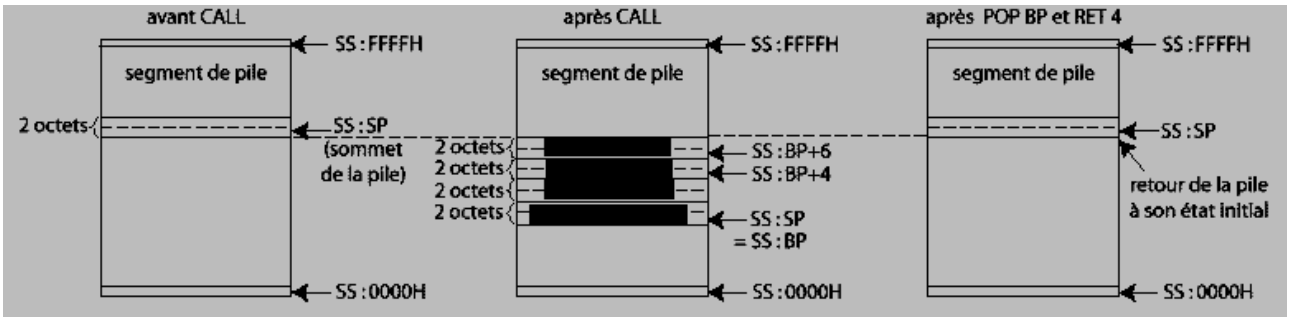


Figure 13. L'état de la pile.

Remarque:
- Gardez bien à l'esprit tout de même que, s'il est vrai que les fonctions permettent une meilleure lisibilité et une meilleure maintenance du code, elle le ralentit également. Si c'est donc la plus grande rapidité possible que vous recherchez et que votre code n'est après tout pas si long que ca, n'hésitez pas à écrire le même code trois ou quatre fois au lieu de l'encapsuler dans une fonction. Le programme généré sera bien sûr plus long, mais aussi plus rapide. **A vous de voir**.
- Toute opération concernant la pile est plus lourde que les opérations avec les registres, c'est pourquoi il est préférable lorsque cela est possible de sauvegarder les valeurs dans des registres au lieu de les empiler/dépiler.
- Il existe deux conventions pour l'empilement/dépilement des arguments passés à une fonction :
    - la convention C (c'est à l'appelant de nettoyer la pile — cela permet une gestion plus aisée des arguments variables);
    - la convention Pascal (c'est à la fonction appelée de nettoyer la pile).
- Les instructions call, ret, ainsi que toutes les opérations impliquant la pile sont très coûteuses en cycles processeur.

Jeu d'instructions du 8086

1. **Transfert de données** :

Général	
MOV	Déplacement d'un octet ou d'un mot
PUSH	Ecriture d'un mot au sommet de la pile
POP	Lecture d'un mot au sommet de la pile
XCHG	Echange d'octets ou de mots
XLAT ou XLATB	Traduction d'un octet à l'aide d'une table
Entrées/Sorties	
IN	Lecture d'un port d'E/S
OUT	Ecriture d'un port d'E/S
Transfert d'adresses	
LEA	Chargement d'une adresse effective
LDS	Chargement d'un pointeur utilisant DS
LES	Chargement d'un pointeur utilisant ES
Transfert des flags	
LAHF	Transfert des 5 flags bas dans AH
SAHF	Transfert de AH dans les 5 flags bas
PUSHF	Sauvegarde des flags sur la pile
POPF	Restauration des flags à partir de la pile

2. **Instructions logiques :**

Logique	
NOT	Complément à 1 d'un octet ou d'un mot
AND	ET logique de deux octets ou de deux mots
OR	OU logique de deux octets ou de deux mots
XOR	OU exclusif logique de deux octets ou de deux mots
TEST	Comparaison, à l'aide d'un ET, d'octets ou de mots
Décalages	
SHL/SAL	Décalage à gauche arithmétique ou logique (octet ou mot)
SHR	Décalage logique à droite d'un octet ou d'un mot
SAR	Décalage arithmétique à droite d'un octet ou d'un mot
Rotations	
ROL	Rotation à gauche d'un octet ou d'un mot)
ROR	Rotation à droite d'un octet ou d'un mot
RCL	Rotation à gauche incluant CF (octet ou mot)
RCR	Rotation à droite incluant CF (octet ou mot)

1. **Instructions arithmétiques** :

Addition	
ADD	Addition d'octets ou de mots
ADC	Addition d'octets ou de mots avec retenue
INC	Incrémentation de 1 d'un octet ou d'un mot
AAA	Ajustement ASCII de l'addition
DAA	Ajustement décimal de l'addition

Soustraction	
SUB	Soustraction d'octets ou de mots
SBB	Soustraction d'octets ou de mots avec retenue
DEC	Décrémentation de 1 d'un octet ou d'un mot
NEG	Complémentation à 2 d'un octet ou d'un mot (changement de signe)
CMP	Comparaison d'octets ou de mots
AAS	Ajustement ASCII de la soustraction
DAS	Ajustement décimal de la soustraction
Multiplication	
MUL	Multiplication non signée d'octets ou de mots
IMUL	Multiplication signée d'octets ou de mots
AAM	Ajustement ASCII de la multiplication
Division	
DIV	Division non signée d'octets ou de mots
IDIV	Division signée d'octets ou de mots
AAD	Ajustement ASCII de la division
CBW	Conversion d'octet en mot
CWD	Conversion de mot en double mot

2. **Instructions sur les chaines de caractères** :

Préfixes	
REP	Répétition tant que CX n'est pas nul
REPE ou REPZ	Répétition tant qu'il y a égalité et que CX n'est pas nul
REPNE ou REPNZ	Répétition tant qu'il n'y a pas égalité et que CX n'est pas nul
Instructions	
MOVS ou MOVSB/MOVSW	Déplacement de blocs d'octets ou de mots
CMPS ou CMPSB/CMPSW	Comparaison de blocs d'octets ou de mots
SCAS ou SCASB/SCASW	Exploration d'un bloc d'octets ou de mots
LODS ou LODSB/LODSW	Tranfert d'un octet ou d'un mot dans AL ou AX
STOS ou STOSB/STOSW	Chargement d'un bloc d'octets ou de mots par AL ou AX

3. **Instructions de branchements** :

Branchements inconditionnels	
CALL	Appel de procédure
RET	Retour d'une procédure
JMP	Saut inconditionnel
Contrôles d'itérations	
LOOP	Bouclage tant que $CX \neq 0$
LOOPE ou LOOPZ	Bouclage tant que $CX \neq 0$ et $ZF = 1$ (égalité)
LOOPNE ou LOOPNZ	Bouclage tant que $CX \neq 0$ et $ZF = 0$ (inégalité)
JCXZ	Saut si CX est nul

Interruptions	
INT	Interruption logicielle
INTO	Interruption si OF = 1 (overflow)
IRET	Retour d'une interruption

4. **Instructions de branchements conditionnels** :

Sauts conditionnels	
JA ou JNBE (1)	Saut si « supérieur » (si CF + ZF = 0)
JAE ou JNB (1)	Saut si « supérieur ou égal » (si CF = 0)
JB ou JNAE (1)	Saut si « inférieur » (si CF = 1)
JBE ou JNA (1)	Saut si « inférieur ou égal » (si CF + ZF = 1)
JC	Saut en cas de retenue (si CF = 1)
JE ou JZ	Saut si « égal » ou « nul » (si ZF = 1)
JG ou JNLE (2)	Saut si « plus grand » (si (SF ⊕ OF) + ZF = 0)
JGE ou JNL (2)	Saut si « plus grand ou égal » (si SF ⊕ OF = 0)
JL ou JNGE (2)	Saut si « plus petit » (si SF ⊕ OF = 1)
JLE ou JNG (2)	Saut si « plus petit ou égal » (si (SF ⊕ OF) + ZF = 1)
JNC	Saut si « pas de retenue » (si CF = 0)
JNE ou JNZ	Saut si « non égal » ou « non nul » (si (ZF = 0)
JNO	Saut si « pas de dépassement » (si OF = 0)
JNP ou JPO	Saut si « parité impaire » (si PF = 0)
JNS	Saut si « signe positif » (si SF = 0)
JO	Saut si « dépassement » (si OF = 1)
JP ou JPE	Saut si « parité paire » (si PF = 1)
JS	Saut si « signe négatif » (si SF = 1)

(1) concerne des nombres non signes. (2) concerne des nombres signes.

5. **Instructions de contrôle du 8086** :

Opérations sur les flags	
STC	Met le flag de retenue à 1
CLC	Efface le flag de retenue
CMC	Inverse l'état du flag de retenue
STD	Met le flag de direction à 1 (décrémentation)
CLD	Met le flag de direction à 0 (incrémentation)
STI	Autorise les interruptions sur INTR
CLI	Interdit les interruptions sur INTR
Synchronisation avec l'extérieur	
HLT	Arrêt du microprocesseur (sortie de cet état par interruption ou reset)
WAIT	Attente tant que $\overline{TEST}$ n'est pas à 0
ESC	Préfixe = instruction destinée à un coprocesseur
LOCK	Préfixe = réservation du bus pour l'instruction
Pas d'opération	
NOP	Pas d'opération

Support de cours

Module Microprocesseurs et DSP

Partie :5

Etude et programmation d'un microprocesseur 16 bits : Le Microprocesseur 8086

Dans cette partie, nous allons etudie en detail :

1. les instructions de traitements de chaines de données.
2. Les operations d'entrées/sorties
3. Les codes machines des instructions du 8086

I. instructions de traitements de chaines de données.

- Une chaine peut atteindre 128K octets;
- La chaine Source est référencée par DS:SI:
- La chaine Destination est référencée par ES:DI:
- L'indicateur D indique le sens de l'accès aux éléments d la chaine:
  - D=1 ⇒ Droite à Gauche (Fin au Début)
  - D=0 ⇒ Gauche à Droite (Début à la Fin).
- Le bit D est positionné par les instructions STD et CL

LODSB/W	Chargement d'un élément de chaine depuis la mémoire dans un registre
STOSB/W	Écriture d'un élément de chaine en mémoire
MOVSB/W	Transfert d'un élément entre deux chaines
SCASB/W	Comparaison entre une valeur et un élément de chaine
CMPSB/W	Comparaison entre deux éléments de chaine

- Un élément de la chaine peut être soit un octet (caractère) [B] ou un mot [W].

MOVSB/W: transfert un élément de la chaine source (DS:SI) vers une élément de la chaine destination (ES:DI).

	D = 0	D = 1

MOVSB	MOV R, [DS:SI]
	MOV [ES:DI], R
	INC SI
	INC DI
		MOV R, [DS:SI]
		MOV [ES:DI], R
		DEC SI
		DEC DI

MOVSW	MOV R, [DS:SI]
	MOV [ES:DI], R
	ADD SI, 2
	ADD DI, 2
		MOV R, [DS:SI]
		MOV [ES:DI], R
		SUB SI, 2
		SUB DI, 2

MOVSB/W: transfert un élément de la chaine source (DS:SI) vers une élément de la chaine destination (ES:DI).

	D = 0	D = 1

MOVSB	MOV R, [DS:SI]
	MOV [ES:DI], R
	INC SI
	INC DI
		MOV R, [DS:SI]
		MOV [ES:DI], R
		DEC SI
		DEC DI

MOVSW	MOV R, [DS:SI]
	MOV [ES:DI], R
	ADD SI, 2
	ADD DI, 2
		MOV R, [DS:SI]
		MOV [ES:DI], R
		SUB SI, 2
		SUB DI, 2

- **STOSB/W:** Écriture d'un élément de chaine en mémoire

	D = 0	D = 1
STOSB	MOV [ES:DI], AL INC DI	MOV [ES:DI], AL DEC DI
STOSW	MOV [ES:DI], AX ADD DI, 2	MOV [ES:DI], AX SUB DI, 2

MOVSB/W: transfert un élément de la chaine source (DS:SI) vers une élément de la chaine destination (ES:DI).

	D = 0	D = 1

MOVSB	MOV R, [DS:SI]
	MOV [ES:DI], R
	INC SI
	INC DI
		MOV R, [DS:SI]
		MOV [ES:DI], R
		DEC SI
		DEC DI

MOVSW	MOV R, [DS:SI]
	MOV [ES:DI], R
	ADD SI, 2
	ADD DI, 2
		MOV R, [DS:SI]
		MOV [ES:DI], R
		SUB SI, 2
		SUB DI, 2

- **SCASB/W:** Comparaison entre une valeur et un élément de chaîne

	D = 0	D = 1
SCASB	CMP [ES: DI], **AL** INC DI	CMP [ES: DI], **AL** DEC DI
SCASW	CMP [ES: DI], **AX** ADD DI, 2	CMP [ES: DI], **AX** SUB DI, 2

110

CMPSB/W: compare un élément de la chaine source (DS:SI) avec une élément de la chaine destination (ES:DI).

	D = 0	D = 1

CMPSB	MOV **R**, [DS:SI]
	CMP [ES:DI], **R**
	INC SI
	INC DI
		MOV **R**, [DS:SI]
		CMP [ES:DI], **R**
		DEC SI
		DEC DI

CMPSW	MOV **R**, [DS:SI]
	CMP [ES:DI], **R**
	ADD SI, 2
	ADD DI, 2
		MOV **R**, [DS:SI]
		CMP [ES:DI], **R**
		SUB SI, 2
		SUB DI, 2

111

Les indicateurs de FLAGS **ne sont pas modifiés** par le opérations (incrémentation et décrémentation) modifian la valeur des index (SI, DI) qui se font automatiquemen dans les instructions de traitement de chaines d caractères. Donc, on ne peut pas utiliser les instruction de branchement après ces instructions pour détecter l fin des chaines.

	D = 1
STOSB ~~JE~~	MOV [ES:DI], **AL** DEC DI **JE**

112

REP, REPE, REPNE permettent de répéter une instruction de traitement de chaines de données. Le registre CX indique le nombre d'itérations à effectuer.

REP ne peut préfixer que les instructions **LODS, STOS, MOVS.**

REPE et **REPNE** ne peuvent préfixer que les instructions **CMPS** et **SCAS.**

113

PROGRAMMATION EN ASSEMBLEUR
TRAITEMENT DES CHAÎNES DE DONNÉES

- **REP LODSB, REP LODSW, REP STOSB, REP STOSW, REP MOVSB, REP MOVSW:** à chaque itération, les actions suivantes sont exécutées :

1. Tester le compteur (CX). S'il est à zéro, aller à 2, sinon:
   a. Exécuter l'opération sur la chaîne (LODS, STOS ou MOVS)
   b. Décrémenter le compteur CX
2. Continuer l'exécution du programme

114

REPNE SCAS compare au plus CX éléments de la chaîne pointée par ES:DI avec la valeur du registre AL ou AX selon le cas. Les itérations sont poursuivies tant que **les éléments de la chaîne sont différents à la valeur du registre et** tant que **le compteur n'est pas nul**. Dès que l'une de ces conditions n'est plus vérifiée, l'instruction REPE SCAS est terminée

117

REPNE CMPS compare au plus CX éléments de la chaine pointée par ES:DI avec ceux de la chaine pointée par DS: SI. Les itérations sont poursuivies tant que **les éléments des deux chaines sont différents et** tant que **le compteur n'est pas nul**. Dès que l'une de ces conditions n'est plus vérifiée, l'instruction REPE CMPS est terminée

118

II. Les operations d'entrées/sorties

PROGRAMMATION EN ASSEMBLEUR
ENTRÉES / SORTIES

Pour faire des entrées/sorties (essentiellement avec l'écran et le clavier). on passe par des interruptions (des sous-programmes préexistants dans la machine) du BIOS. la plus importante est l'**interruption 21h**.

L'appel se fait via l'instruction **INT 21h**.

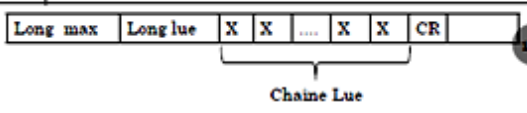
Le registre AH contient le numéro de la fonction qu'on veut utiliser.

Instructions	Fonctionnalité
MOV AH. 4CH INT 21	Quitter le système d'exploitation.
MOV AH. 01H INT 21	Lire un caractère au clavier et le renvoyer **(son code ASCII)** dans le registre AL.
MOV AH. 02H INT 21	Afficher le caractère qui se trouve dans le registre DL à l'écran.

- **Exemple:** afficher le caractère b à l'écran.

```
MOV DL. 'b'
MOV AH. 2
INT 21h
```

Instructions	Fonctionnalité
MOV AH. 0AH INT 21	Lire une chaîne de caractère à partir du clavier •L'adresse du premier caractère de la chaîne est DS:DX •Il faut préciser dans le premier octet de la chaîne. la longueur maximale à ne pas dépasser. •Le deuxième octet indique le nombre de caractère effectivement saisi. •La chaîne se termine par retour chariot. • La longueur de la chaîne doit être supérieure ou égale



- **Exemple:** Copier une chaîne lue dans une autre

```
Ch1 DB 15 DUP(13)
Ch2 DB 15 DUP(13)
.....
MOV AH. 0AH
LEA DX. Ch1
INT 21h
MOV CH. 0
MOV CL. [Ch1 + 1]
LEA SI. Ch1+2
LEA DI. Ch2+2
REP MOVSB
```

Instructions	Fonctionnalité
MOV AH. 09H INT 21	Afficher une chaîne de caractère à l'écran: •L'adresse du premier caractère de la chaîne est DS:DX. •Le dernier caractère de la chaîne doit être le caractère $.

```
msg DB 'hello world', '$'
.....
MOV AH. 9
LEA DX. msg
INT 21h
```

III. Les codes machines des instructions.

CODE MACHINE DES INSTRUCTIONS

Les instructions 8086 sont présentées dans la machine sous forme d'une suite de bits.

Elles comportent de 1 à 6 octets:

- Le premier octet est obligatoire
- Les autres octets sont facultatifs (ils dépendent du code de l'opération)

De manière générale, le code machine d'une instruction comportent le code de l'opération (COP), le code du registre utilisé (REG), le code du mode d'adressage utilisé (MOD), l'adresse de la case mémoire (R/M), etc.

Bits	7	6	5	4	3	2	1	0	
1er octet	COP						D	W	Le code d'opération
2ème octet	MOD		REG			R/M			Le mode d'adressage
3ème octet	[Optionnel]								Les bits de poids faibles ADR, DEP ou VAL
4ème octet	[Optionnel]								Les bits de poids forts de ADR, DEP ou VAL
5ème octet	[Optionnel]								Les bits de poids faible de VAL
6ème octet	[Optionnel]								Les bits de poids faible de VAL

1er octet: Le code d'opération

7	6	5	4	3	2	1	0
COP						D	W

- COP: présente le code proprement dit de l'instruction
- D: désigne la destination du résultat

D	
0	Résultat dans mémoire ou Opération entre deux registres
1	Résultat dans un registre

- W: indique si l'opération est sur 8 bits ou sur 16 bits

W = 0	W = 1
8 bits	16 bits

2ème octet: Le mode d'adressage

7	6	5	4	3	2	1	0
MOD		REG			R/M		

- MOD: indique le mode d'adressage

MOD	Mode d'adressage
00	Directe Indirecte avec DEP = 0
01	Indirect déplacement court (sur 8 bits)
10	Indirect déplacement long (sur 16 bits)
11	Par registre ou Immédiat

2ème octet: Le mode d'adressage

7	6	5	4	3	2	1	0
MOD		REG			R/M		

- REG: désigne le registre constituant un opérande

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

2ème octet: Le mode d'adressage

7	6	5	4	3	2	1	0
MOD		REG			R/M		

- R/M: précise l'adresse constituant l'instruction (direct ou indirect) ou le code de registre destinataire (par registre)

R/M		R/M	
000	[BX+SI+DEP]	100	[SI+DEP]
001	[BX+DI+DEP]	101	[DI+DEP]
010	[BP+SI+DEP]	110	[BP+DEP] Direct
011	[BP+DI+DEP]	111	[BX+DEP]

CODE MACHINE
DES INSTRUCTIONS DE TRANSFERT

Mnémonique	1er octet			
MOV				
R/M, R/M	1000 10dw	MOD REG R/M	(Adr ou dep)	
M, Val	1100 011w	MOD 000 M	(Adr ou dep)	valeur
R, Val	1011 w REG	Valeur		
AX/AL, M_direct	1010 000w	M_direct		
M_direct, AX/AL	1010 001w	M_direct		
XCHG				
R/M, R/M	1000 011w	MOD REG R/M	(Adr ou dep)	
R, AX	10010 REG			
LEA R, Var	10001101	MOD REG 110	Offset de Var	

CODE MACHINE
DES INSTRUCTIONS ARITHMÉTIQUE

Mnémonique	1er octet			
ADD				
R/M, R/M	0000 00dw	MOD REG R/M	(Adr ou dep)	
R/M, Val	1000 00sw	MOD 000 R/M	(Adr ou dep)	Valeur
AL/AX, Val	0000 010w	Valeur		
SUB				
R/M, R/M	0010 10dw	MOD REG R/M	(Adr ou dep)	
M, Val	1000 00sw	MOD 101 R/M	(Adr ou dep)	Valeur
R, Val	1000 00sw	11 101 REG	Valeur	
AL/AX, Val	0010 110w	Valeur		
CMP				
R/M, R/M	0011 10dw	MOD REG R/M	(Adr ou dep)	
R/M, Val	1000 00sw	MOD 111 R/M	(Adr ou dep)	Valeur
AL/AX, Val	0011 110w	Valeur		

s=1 dans le cas où une extension de signe 8 bits vers 16 bits est effectuée sur la donnée valeur avant l'opération.

CODE MACHINE
DES INSTRUCTIONS DE DÉCALAGE

Mnémonique	1er octet	2ème octet	
SAR			
R/M,1	1101 000w	MOD 111 R/M	(Adr ou dep)
R/M,CL	1101 001w	MOD 111 R/M	(Adr ou dep)
SHL/SAL			
R/M,1	1101 000w	MOD 100 R/M	(Adr ou dep)
R/M,CL	1101 001w	MOD 100 R/M	(Adr ou dep)
SHR			
R/M,1	1101 000w	MOD 101 R/M	(Adr ou dep)
R/M,CL	1101 001w	MOD 101 R/M	(Adr ou dep)

CODE MACHINE
DES INSTRUCTIONS DE DÉCALAGE

Mnémonique	1er octet	2ème octet	
ROL			
R/M,1	1101 000w	MOD 000 R/M	(Adr ou dep)
R/M,CL	1101 001w	MOD 000 R/M	(Adr ou dep)
ROR			
R/M,1	1101 000w	MOD 001 R/M	(Adr ou dep)
R/M,CL	1101 001w	MOD 001 R/M	(Adr ou dep)
RCL			
R/M,1	1101 000w	MOD 010 R/M	(Adr ou dep)
R/M,CL	1101 001w	MOD 010 R/M	(Adr ou dep)
RCR			
R/M,1	1101 000w	MOD 011 R/M	(Adr ou dep)
R/M,CL	1101 001w	MOD 011 R/M	(Adr ou dep)

CODE MACHINE
DES INSTRUCTIONS DE BRANCHEMENT

Mnémonique	1er octet	2ème octet
JMP		
Etiq	1110 1001	DepRel_16
Short Etiq	1110 1011	DepRel_8
JS	0111 1000	DepRel_8
JNS	0111 1001	DepRel_8
JC	1110 0010	DepRel_8
JNC	0111 0011	DepRel_8
JE/JZ	0111 0100	DepRel_8
JNE/JNZ	0111 0101	DepRel_8

Mnémonique	1er octet	2ème octet
JA	0111 0111	DepRel_8
JAE	0111 0011	DepRel_8
JB	0111 0010	DepRel_8
JBE	0111 0110	DepRel_8
JG	0111 1111	DepRel_8
JGE	0111 1101	DepRel_8
JL	0111 1100	DepRel_8
JLE	0111 1110	DepRel_8
LOOP	1110 0010	DepRel_8
LOOPZ	1110 0001	DepRel_8
LOOPNZ	1110 0000	DepRel_8
JCXZ	1110 0011	DepRel_8

AUTRES INSTRUCTIONS

Instructions d'action sur les indicateurs :

Mnémonique	1er octet	Action
CLC	1111 1000	C←0
STC	1111 1001	C← 1
CMC	1111 0101	C ← NOT C
CLD	1111 1100	D←0 (Gauche à droite)
STD	1111 1101	D← 1 (Droite à gauche)

Instructions de Conversion

Mnémonique	1er octet
CBW	1001 1000
CWD	1001 1001

Instruction d'une interruption

Mnémonique	1er octet	2eme octet
INT Num	1100 1101	Numéro d' interruption (21h)

CODE MACHINE
DES INSTRUCTIONS DE TRAITEMENT DES CHAÎNES

Mnémonique	1er octet	
MOVS	1010 010w	w = 0 pour MOVSB, LODSB, STOSB, SCASB, CMPSB w = 1 pour MOVSW, LODSW, STOSW, SCASW, CMPSW
LODS	1010 110w	
STOS	1010 101w	
SCAS	1010 111w	
CMPS	1010 011w	
REP/REPZ/REPE	1111 0011	
REPNZ/REPNE	1111 0010	

CHAPITRE

Les processeurs numériques du siganl (DSP)

Un DSP pour Digital Signal Processor en anglais est un processeur spécialisé dans le traitement numérique du signal. Son architecture, ses instructions, ses modes d'adressage sont optimisés pour traiter une grande quantité de données en parallèle à chaque cycle d'horloge.

Comme un microprocesseur classique, un DSP est mis en œuvre en lui associant de la mémoire (RAM, ROM) et des périphériques.

Le DSP fonctionne sous 2 modes :

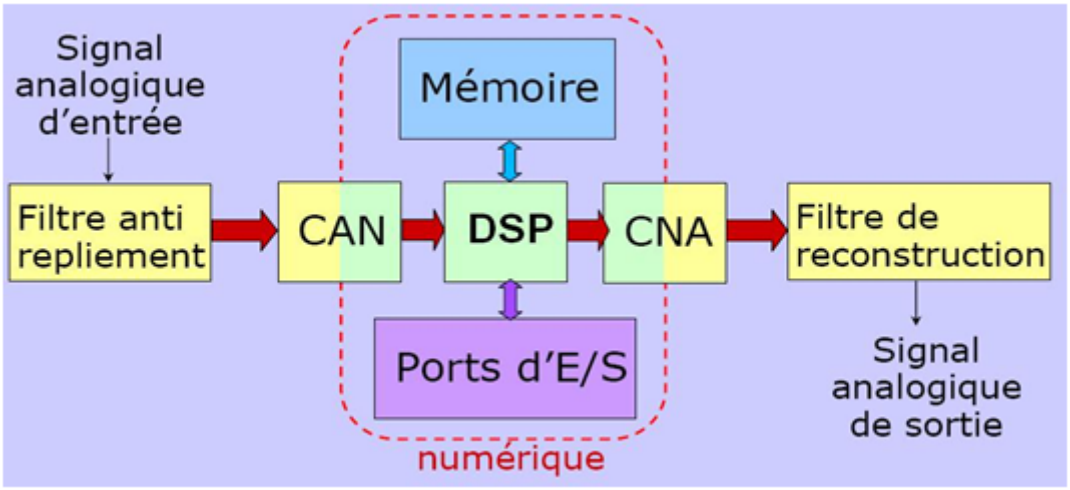
1. mode microcontrôleur: fonctionne sur sa mémoire programme interne rapide
2. mode microprocesseur: fonctionne sur une mémoire programme externe

Système de traitement numérique du signal à base de DSP

Le traitement numérique du signal est l'application d'opérations mathématiques sur des signaux représentés sous forme numérique :

- Représentation des signaux sous forme de séquences d'échantillons
- Les signaux numériques sont obtenus à partir de signaux physiques via des récepteurs (ex : microphones) et des convertisseurs analogiques-numériques (CAN)
- Les signaux numériques sont reconvertis en signaux physiques par des convertisseurs numériques-analogiques (CNA)
- Processeurs de traitement du signal (DSP) : système électronique qui traite des signaux numériques

D'une manière générale l'architecture d'un système de traitement numérique du signal peut être représentée par la Figure suivante :



Chaîne typique d'un système de traitement numérique du signal

Le filtre anti-repliement et de reconstruction sont des filtres passe-bas

Avantage du traitement numérique

- obtention d'1 meilleure précision et donc réalisation de différents filtres
- Permet d'enchaîner des algorithmes de traitement beaucoup plus complexes (ex transformée de Fourier
- Grande résistance aux bruits
- Précisions arbitraire

- Stabilité dans le temps
- Stockage des données

Utilité de l'utilisation d'un DSP

- Flexibilité de la programmation
- Stabilité
- Redondance

Les applications des DSP

Les applications sont nombreuses dans les domaines suivants :

- Télécommunications : Modem, multiplexeurs, récepteurs de numérotation DTMF, télécopieurs, codeurs de parole GMS, ...),
- Interfaces vocales : Codeur vocaux, reconnaissance automatique de la parole, synthèse vocale ...
- Militaire : Guidage missiles, navigation, communications cryptée, radar, ...
- Multimédias et grand public : Compression des signaux audio (CD), compression des images, cartes multimédias pour PC, synthèse musicale, jeux, ...
- Médical : Compression d'image médicale (IRM, échographie...), traitements des signaux biophysiques (ECG, EEG,...), implants cochléaires, équipement de monitoring.
- Electronique automobile : Equipement de contrôle moteur, aide à la navigation, commande vocale, détection de cliquetis pour avance à l'allumage, ...
- Automatisation et contrôle de processus : Surveillance et commande de machines, contrôle de moteurs, robots ...
- Instrumentation : Analyseur de spectre, générateurs de fonction, interprétation de signaux sismiques, ...

Les principaux fabricants de DSP sont :
Texas Instruments, Analog Devices, Motorola, Zilog, Lucent, Nec, Zoran, Zsp, Microchip

Les DSP à virgule fixe, Les DSP à virgule flottante

Il est possible de distinguer deux familles de DSP :

- Les DSP à virgule fixe : les nombres (les données) manipulés sont représentés en virgule fixe
- Les DSP à virgule flottante : les nombres (les données) manipulés sont représentés en virgule flottante

Virgule fixe

• Précision fixée

• Dynamique réduite

• Calcul plus rapide

• Processeur moins cher

Virgule flottante

• Représentation en mantisse et exposant:

$v = \text{mantisse} \times 2^{\text{exposant}}$

• Dynamique étendue

• Programmation plus simple

• Processeur plus complexe et plus cher

Opération MAC

L'opération MAC (multiplication suivit d'une addition) est l'élément clé dans la majorité des algorithmes de traitement de signal, pour cela l'architecture des DSP est optimisée pour performer cette l'opération. Voici quelques algorithmes typiques comme:

Finite Impulse Response Filter (FIR)

$$y(n) = \sum_{k=0}^{N} a(k) * x(n-k)$$

Infinite Impulse Response Filter(IIR)

$$y(n) = \sum_{k=0}^{N} a(k) * x(n-k) + \sum_{k=0}^{M} b(k) * y(n-k)$$

Convolution

$$y(n) = \sum_{k=0}^{N} x(k) * h(n-k)$$

Discrete Fourier Transform (DFT)

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp[-j(2\Pi/N)nk]$$

Le DSP est prévu pour effectuer le plus rapidement possible, en principe en un seul cycle d'horloge, l'opération multiplication/addition sur des grandeurs numériques :

MR=X*Y +R : c'est l'opération MAC

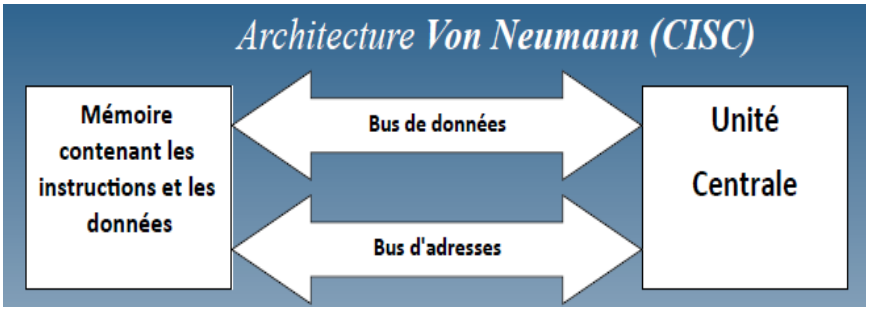
Où X et Y sont soit des données, soit des constantes et R une donnée, une constante ou un résultat précédent. MR est alors le résultat de l'opération arithmétique.

Si le DSP fonctionne en virgule fixe avec des données sur 16 bits, le résultat MR est alors sur 32 bits (ou plus, selon l'architecture). Si l'utilisateur ne conserve que les 16 bits de poids fort, le calcul est alors effectué en simple précision. Si les 32 bits sont utilisés, on parle de double précision : le temps de calcul est alors plus long.

Si le DSP fonctionne en virgule flottante avec des données en 32 bits, le résultat MR est alors sur 40 bits (ou plus, selon l'architecture). L'utilisateur ne prend en compte que les données de 32 bits en ignorant les bits de poids faibles de la mantisse.

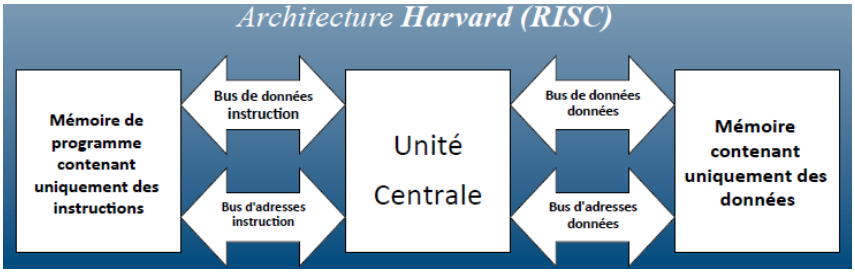
Architecture des DSP

- **Cas d'un microprocesseur : Architecture Von Neumann**



Dans l'architecture de la machine de Von Neumann, le programme et les données sont enregistrés sur la même mémoire. Chaque instruction contient le code de l'opération à effectuer et l'adresse de la donnée à utiliser, il faut donc souvent plusieurs cycles d'horloge pour exécuter une instruction.

- **Cas d'un DSP : Architecture Harvard**



Dans l'architecture dite de Harvard, on sépare la mémoire de programme de la mémoire des données : l'adressage de ces mémoires est indépendant. Cela leur permet ainsi de lire une instruction tout en chargeant des opérandes.
Un DSP est souvent relié à plusieurs mémoires, ou à des mémoires multiports, il est capable d'effectuer plusieurs accès mémoires simultanés.

Exemple de DSP

- Analog Devices

ADSP 21xx	fixe	16 bits	33 Mips
ADSP210x x	flottant	32	40

- Lucent

DSP16xx		fixe	16	70
DSP32xx		Flottant	32	20

- Motorola

DSP5600x	fixe	24	40
DSP561xx	fixe	16	30
DSP563xx	fixe	24	80
DSP96002	flottant	32	20

- Texas Instrument

TMS320C1 x	fixe	16	9
TMS320C2 x	fixe	16	12
TMS320C2 xx	fixe	16	40
TMS320C3	flottant	32	25

x			
TMS320C4x	flottant	32	30
TMS320C5x	fixe	16	50
TMS320C8x	fixe	8/16	50

Présentation du DSP TMS320C6701

Le TMS320C6701 est un membre de la catégorie C67x, appartenant à La plateforme TMS320C6000 des processeurs de signaux numériques fait partie de la famille TMS320 qui comporte les processeurs TMS320C62x à arithmétique fixe et TMS320C67x à arithmétique flottante.

Description générale du DSP TMS320C6701

Ce DSP permet le traitement par paquets de 8 instructions en parallèle par 8 unités fonctionnelles. Il est organisée en deux blocs d'unités fonctionnelles. Le premier contient les unités fonctionnelles .L1, .S1, .M1, .D1 et 16 registres constituant le chemin A (path A). Le deuxième contient quatre autres unités .L2,.S2, .M2, .D2 et 16 registres constituant ainsi le chemin B (path B).

Caractéristiques principales du TMS320C6701

Le DSP TMS320C6701 comportant 2 CPUs, où chacune dispose des caractéristiques suivantes :

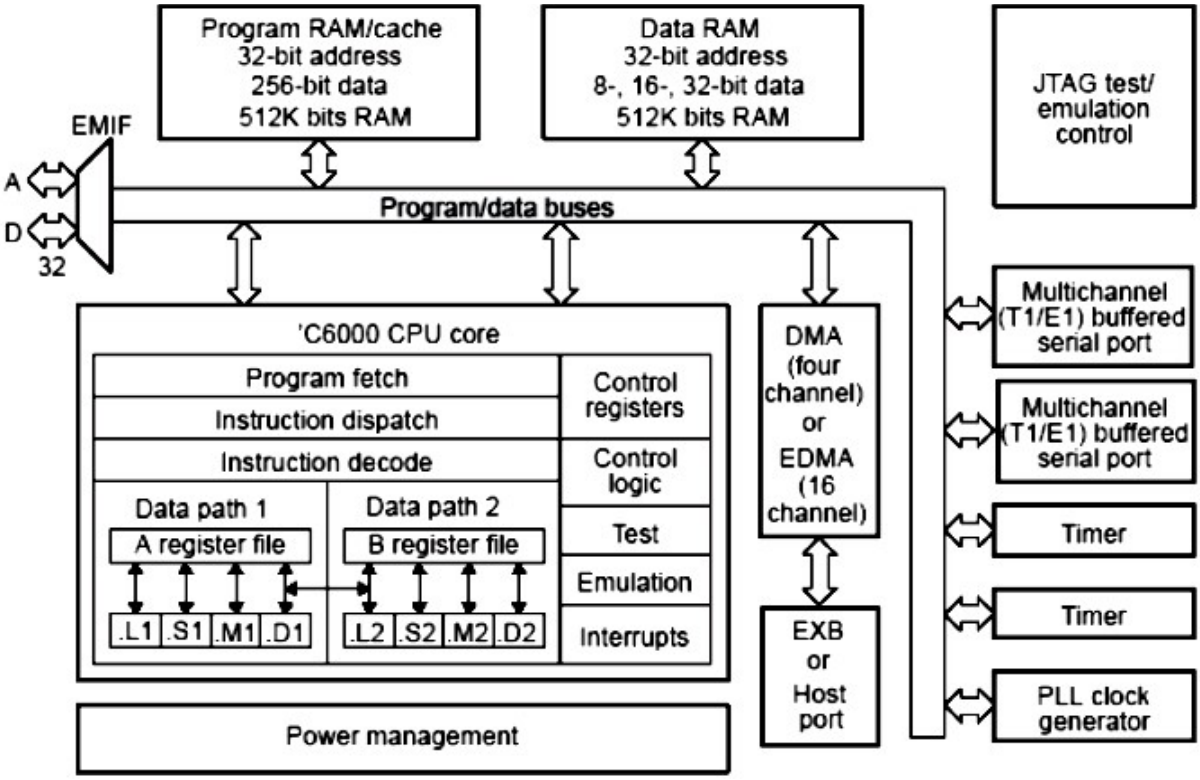
• 16 Registres de 32 bits ;
• 04 UAL et multiplieurs ;
• 334 MIPS ;
• 600 MFLOP ;
• temps de cycle de 5 ns ;
• bus d'adresse de 32 bits ;
• format de données 32 bits.

Mémoire/périphérique :

- 64K octets L1P de mémoire cache de programme,
- 64K octets L1D de mémoire cache de données,
- 64 K octets L2 de mémoire cache RAM.

• 32 bits interface de mémoire externe (EMIF) ;
• contrôleur d'accès mémoire direct (EDMA) ;
• un port parallèle de 16 bits (HPI) ;
• 2 bus série (McBSP) ;
• 2 timers de 32 bits ;
• générateur d'horloge par PLL.

Architecture interne du TMS320C6701



Les unités fonctionnelles au niveau de la CPU exécutent des opérations de logique, de décalage, de multiplication, etc. Les deux unités (.D1 et .D2) sont responsables de tous les transferts de données entre les registres dossiers et les mémoires. Les quatre unités fonctionnelles du même chemin de données ont un seul bus de données relié aux registres de l'autre côté de l'unité centrale de traitement, de sorte que ces unités puissent échanger les données avec le registre dossier du côté opposé.

- Unité centrale de traitement (CPU)

L’unité centrale de traitement contient :

• Unité de recherche de programmes ;

• Unité d'expédition d'instructions ;

• Unité de décodage d’instructions ;

• 32 registres de 32 bits ;

• Deux chemins de données, chacune avec quatre unités fonctionnelles ;

• Registres de contrôle ;

• Logique de contrôle ;

- Les Mémoires internes

Le C6701 a une architecture basée en partie sur les mémoires caches. Des mémoires de niveau 1, séparées en mémoires programme et données (L1P et L1D). Ces espaces sont accessibles à tout moment par l'unité centrale de traitement. Le contrôleur de la mémoire cache de programme (L1P) interface l'unité centrale de traitement au L1P. Un chemin de 256-bits forme un jet (stream) continu de 8 instructions de 32 bits pour une exécution maximum. Le contrôleur de la mémoire cache de données (L1D) fournit l'interface entre l’unité centrale

de traitement et L1D. Ce dernier permet l'accès simultané par les deux côtés de l'unité centrale de traitement. Quand un manque apparaît au niveau de L1D ou de L1P, la demande passe au contrôleur L2. Ce contrôleur facilite les fonctions suivantes :
• L'arbitrage nécessaire à l'accès mémoire interne entre CPU et contrôleur EDMA.
• L'accès des données d'unité centrale de traitement à l'EMIF.
• Les accès d'unité centrale de traitement aux périphériques sur chip.
• Envoi des demandes à EMIF pour des données de L2 manquantes.

- Les registres

Abréviation	Nom De Registre	Description
AMR	Addressing mode register	Indique si l'adressage utilisé est linéaire ou circulaire pour chacun des huit registres. Il contient également des tailles pour l'adressage circulaire.
CSR	Control status register	Contient le bit d'autorisation d'interruption global, bits de contrôle de la mémoire cache, et d'autres types de contrôle et bits d'état.
PCE1	Program counter, E1 phase	Contient l'adresse du paquet à chercher qui est dans l'étape de pipeline E1.
FADCR	Floating-point adder configuration registre	Spécifie le mode underflow, mode d'arrondissage, NaNs, et d'autres exceptions pour l'unité L.
FAUCR	Floating-point auxiliary configuration register	Spécifie le mode underflow, mode d'arrondissage, NaNs, et d'autres exceptions pour l'unité S.
FMCR	Floating-point multiplier configuration register	Spécifie le mode underflow, mode d'arrondissage, NaNs, et d'autres exceptions pour l'unité M.

* Registres dossiers à usage général
Il y'a deux registres dossiers à usage général (A et B) dans les chemins de données du C6000. Pour les DSP C62x/C67x, chacun de ces registres dossier contient 16 registres de 32 bits (A0–A15 pour le registre dossier A et B0–B15 registres dossier B) .

* Registres de contrôle
Le tableau donnent une description des registres de contrôle de la famille C6000 et particulièrement celle de C67x.

Les périphériques
Les processeurs C6701 disposent de plusieurs périphériques intégrés auxquels sont associés des registres de contrôle mappés en mémoire. Les périphériques disponibles dépendent de la version du circuit et comprennent :
- Interface de mémoire externe (EMIF),
- un contrôleur d'accès mémoire direct (EDMA) à 17 canaux,
- 02 Timers de 32 bits,
- 02 Ports série (McBSP),

un générateur d'horloge par PLL,
- Le contrôleur EDMA (Enhanced Direct Memory Access)
Le TMS320C6701 élabore des transferts de données depuis un élément interne ou externe, en utilisant soit la CPU ou l'EDMA Figure 5.2. Typiquement les transferts depuis les périphériques internes au DSP sont faits par L'EDMA, libérant ainsi la CPU pour des tâches de calcul.
L'EDMA inclut plusieurs perfectionnements par rapport au DMA classique, vu le nombre canaux fournis avec priorités programmables, et la capacité de lier et enchaîner des transferts de données. L'EDMA permet le mouvement de données vers et depuis tous les espaces mémoires accessibles, y'compris la mémoire interne (L2 SRAM), les périphériques, et la

mémoire externe. Il prévoit deux types de transferts des données, à savoir, les transferts à une dimension (1D) et à deux dimensions (2D).
Les événements EDMA sont capturés dans le registre d'événements (Un événement est un signal de synchronisation qui déclenche un canal EDMA pour commencer un transfert). Si les événements se produisent simultanément, ils sont résolus par l'encodeur d'événement.
L'EDMA a la capacité d'exécuter vite les transferts effectifs en acceptant une demande de transfert DMA rapide (QDMA) de la CPU. Un transfert QDMA convient le mieux pour les applications qui exigent des transferts rapides de données tels que les demandes de données dans les algorithmes à boucles.

- Les temporisateurs
Le TMS320C6701 a deux temporisateurs (timers) à usage général qui ont pour objectifs:
1) La mesure la durée d'un événement.
2) Le comptage des événements.
3) La génération des interruptions vers la CPU.
4) L'envoi des événements de synchronisation à l'EDMA.

Le contrôleur d'EDMA comporte :
1) Registres d'événements et de traitement d'interruptions
2) Encodeur d'événements
3) RAM paramètres
4) Hardware de génération d'adresses

- La liaison série (McBSP) (multichannel buffered serial port)
Le C6701 contient deux modules de liaison série (McBSPo et McBSP1).Ces liaisons série permettent des communications en full-duplex, un flot continu des données, une indépendance dans le timing et le format des données à l'émission et la réception, une interface directe avec les CODEC,CNA et CAN, et une synchronisation par horloge interne ou externe.

- Contrôleur PLL
Le contrôleur PLL est capable de générer différentes horloges pour différentes parties du noyau de DSP, l'interface externe de mémoire, McBS, périphériques, et d'autres modules à l'intérieur du DSP.

Jeu d'instructions
Le TMS320C6701 présente une liste très riche en instructions utilisées dans sa programmation en assembleur. On se limite de décrire quelques instructions.
- Instructions d'addition
• Instructions pour les entiers
* L'instruction ADD
Syntaxe ADD (.unit) src1, src2, dst
Ou ADDU (.L1 ou .L2) src1, src2, dst
Ou ADD (.D1 ou .D2) src2, src1, dst
.unit = .L1, .L2, .S1, .S2
Example: ADD .L2X A1, B1, B2
* L'instruction ADDAB
Syntaxe ADDAB (.unit) src2, src1, dst
.unit = .D1 ou .D2
• Addition de réelle double précision
Syntaxe ADDDP (.unit) src1, src2, dst
.unit = .L1 or .L2

Example: ADDDP .L1X B1:B0, A3: A2, A5: A4
- Instructions de multiplication
• Instructions pour les entiers
Syntaxe MPY (.unit) src1, src2, dst
Ou MPYU (.unit) src1, src2, dst
.unit = .M1 ou .M2
Exemple : MPYU .M1 A1, A2, A3
• Addition de réelle double précision
Syntaxe MPYDP (.unit) src1, src2, dst
.unit = .M1 ou .M2
Exemple : MPYDP .M1 A1:A0, A3:A2, A5:A4

Modes d’adressages

Les modes d'adressage sur le C671 sont : linéaires et circulaire en utilisant BK0 ou BK1. Le mode est indiqué par le registre de mode d'adressage, ou (AMR). Tous les registres peuvent exécuter l'adressage linéaire. Seulement huit registres peuvent exécuter l'adressage circulaire (A4–A7 sont employés par l'unité .D1, et B4–B7 employés par l'unité .D2).

CHAPITRE

Les processeurs numériques du siganl (DSP)

Partie : 2

Présentation du DSP TMS320C6701

Le TMS320C6701 est un membre de la catégorie C67x, appartenant à La plateforme TMS320C6000 des processeurs de signaux numériques fait partie de la famille TMS320 qui comporte les processeurs TMS320C62x à arithmétique fixe et TMS320C67x à arithmétique flottante.

Description générale du DSP TMS320C6701

Ce DSP permet le traitement par paquets de 8 instructions en parallèle par 8 unités fonctionnelles. Il est organisée en deux blocs d'unités fonctionnelles. Le premier contient les unités fonctionnelles .L1, .S1, .M1, .D1 et 16 registres constituant le chemin A (path A). Le deuxième contient quatre autres unités .L2,.S2, .M2, .D2 et 16 registres constituant ainsi le chemin B (path B).

Caractéristiques principales du TMS320C6701

Le DSP TMS320C6701 comportant 2 CPUs, où chacune dispose des caractéristiques suivantes :

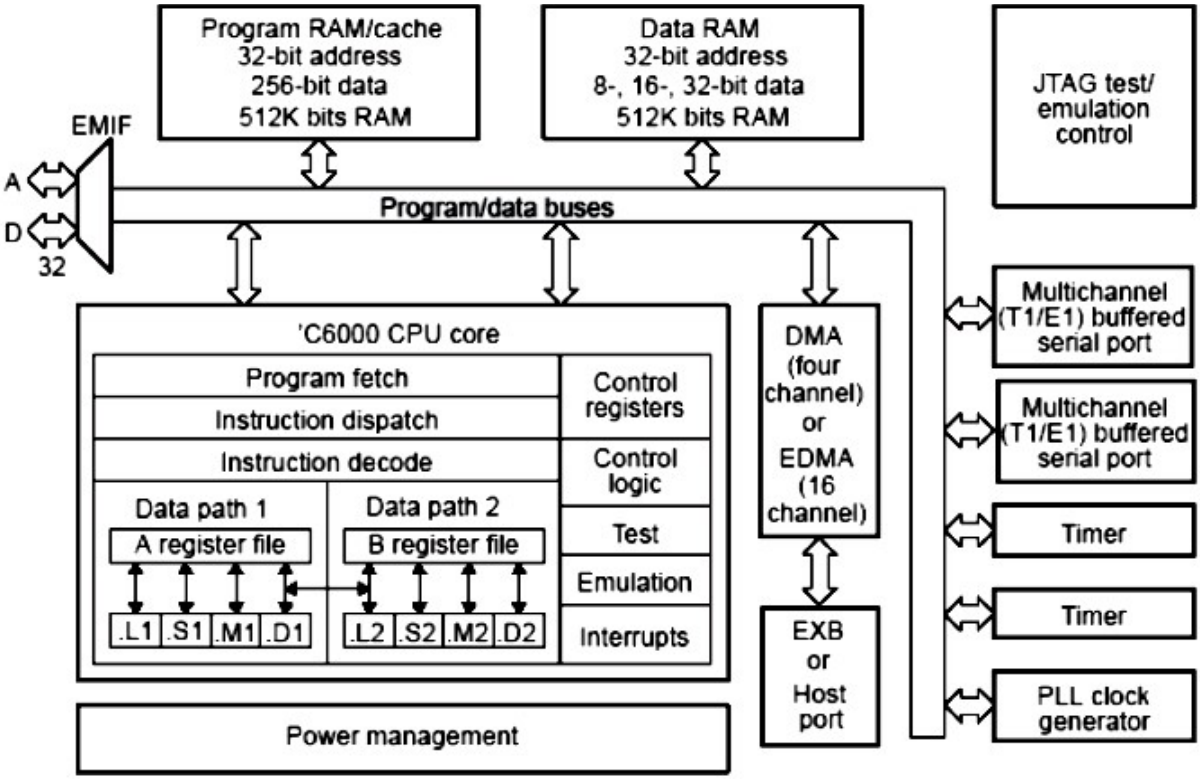
• 16 Registres de 32 bits ;
• 04 UAL et multiplieurs ;
• 334 MIPS ;
• 600 MFLOP ;
• temps de cycle de 5 ns ;
• bus d'adresse de 32 bits ;
• format de données 32 bits.

Mémoire/périphérique :

- 64K octets L1P de mémoire cache de programme,
- 64K octets L1D de mémoire cache de données,
- 64 K octets L2 de mémoire cache RAM.

• 32 bits interface de mémoire externe (EMIF) ;
• contrôleur d'accès mémoire direct (EDMA) ;
• un port parallèle de 16 bits (HPI) ;
• 2 bus série (McBSP) ;
• 2 timers de 32 bits ;
• générateur d'horloge par PLL.

Architecture interne du TMS320C6701



Les unités fonctionnelles au niveau de la CPU exécutent des opérations de logique, de décalage, de multiplication, etc. Les deux unités (.D1 et .D2) sont responsables de tous les transferts de données entre les registres dossiers et les mémoires. Les quatre unités fonctionnelles du même chemin de données ont un seul bus de données relié aux registres de l'autre côté de l'unité centrale de traitement, de sorte que ces unités puissent échanger les données avec le registre dossier du côté opposé.

- Unité centrale de traitement (CPU)

L’unité centrale de traitement contient :

• Unité de recherche de programmes ;

• Unité d'expédition d'instructions ;

• Unité de décodage d’instructions ;

• 32 registres de 32 bits ;

• Deux chemins de données, chacune avec quatre unités fonctionnelles ;

• Registres de contrôle ;

• Logique de contrôle ;

- Les Mémoires internes

Le C6701 a une architecture basée en partie sur les mémoires caches. Des mémoires de niveau 1, séparées en mémoires programme et données (L1P et L1D). Ces espaces sont accessibles à tout moment par l'unité centrale de traitement. Le contrôleur de la mémoire cache de programme (L1P) interface l'unité centrale de traitement au L1P. Un chemin de 256-bits forme un jet (stream) continu de 8 instructions de 32 bits pour une exécution maximum. Le contrôleur de la mémoire cache de données (L1D) fournit l'interface entre l’unité centrale

de traitement et L1D. Ce dernier permet l'accès simultané par les deux côtés de l'unité centrale de traitement. Quand un manque apparaît au niveau de L1D ou de L1P, la demande passe au contrôleur L2. Ce contrôleur facilite les fonctions suivantes :
• L'arbitrage nécessaire à l'accès mémoire interne entre CPU et contrôleur EDMA.
• L'accès des données d'unité centrale de traitement à l'EMIF.
• Les accès d'unité centrale de traitement aux périphériques sur chip.
• Envoi des demandes à EMIF pour des données de L2 manquantes.

- Les registres

Abréviation	Nom De Registre	Description
AMR	Addressing mode register	Indique si l'adressage utilisé est linéaire ou circulaire pour chacun des huit registres. Il contient également des tailles pour l'adressage circulaire.
CSR	Control status register	Contient le bit d'autorisation d'interruption global, bits de contrôle de la mémoire cache, et d'autres types de contrôle et bits d'état.
PCE1	Program counter, E1 phase	Contient l'adresse du paquet à chercher qui est dans l'étape de pipeline E1.
FADCR	Floating-point adder configuration registre	Spécifie le mode underflow, mode d'arrondissage, NaNs, et d'autres exceptions pour l'unité L.
FAUCR	Floating-point auxiliary configuration register	Spécifie le mode underflow, mode d'arrondissage, NaNs, et d'autres exceptions pour l'unité S.
FMCR	Floating-point multiplier configuration register	Spécifie le mode underflow, mode d'arrondissage, NaNs, et d'autres exceptions pour l'unité M.

* Registres dossiers à usage général
Il y'a deux registres dossiers à usage général (A et B) dans les chemins de données du C6000. Pour les DSP C62x/C67x, chacun de ces registres dossier contient 16 registres de 32 bits (A0–A15 pour le registre dossier A et B0–B15 registres dossier B) .

* Registres de contrôle
Le tableau donnent une description des registres de contrôle de la famille C6000 et particulièrement celle de C67x.

Les périphériques
Les processeurs C6701 disposent de plusieurs périphériques intégrés auxquels sont associés des registres de contrôle mappés en mémoire. Les périphériques disponibles dépendent de la version du circuit et comprennent :
- Interface de mémoire externe (EMIF),
- un contrôleur d'accès mémoire direct (EDMA) à 17 canaux,
- 02 Timers de 32 bits,
- 02 Ports série (McBSP),

un générateur d'horloge par PLL,
- Le contrôleur EDMA (Enhanced Direct Memory Access)
Le TMS320C6701 élabore des transferts de données depuis un élément interne ou externe, en utilisant soit la CPU ou l'EDMA Figure 5.2. Typiquement les transferts depuis les périphériques internes au DSP sont faits par L'EDMA, libérant ainsi la CPU pour des tâches de calcul.
L'EDMA inclut plusieurs perfectionnements par rapport au DMA classique, vu le nombre canaux fournis avec priorités programmables, et la capacité de lier et enchaîner des transferts de données. L'EDMA permet le mouvement de données vers et depuis tous les espaces mémoires accessibles, y'compris la mémoire interne (L2 SRAM), les périphériques, et la

mémoire externe. Il prévoit deux types de transferts des données, à savoir, les transferts à une dimension (1D) et à deux dimensions (2D).
Les événements EDMA sont capturés dans le registre d'événements (Un événement est un signal de synchronisation qui déclenche un canal EDMA pour commencer un transfert). Si les événements se produisent simultanément, ils sont résolus par l'encodeur d'événement.
L'EDMA a la capacité d'exécuter vite les transferts effectifs en acceptant une demande de transfert DMA rapide (QDMA) de la CPU. Un transfert QDMA convient le mieux pour les applications qui exigent des transferts rapides de données tels que les demandes de données dans les algorithmes à boucles.

- Les temporisateurs
Le TMS320C6701 a deux temporisateurs (timers) à usage général qui ont pour objectifs:
1) La mesure la durée d'un événement.
2) Le comptage des événements.
3) La génération des interruptions vers la CPU.
4) L'envoi des événements de synchronisation à l'EDMA.

Le contrôleur d'EDMA comporte :
1) Registres d'événements et de traitement d'interruptions
2) Encodeur d'événements
3) RAM paramètres
4) Hardware de génération d'adresses

- La liaison série (McBSP) (multichannel buffered serial port)
Le C6701 contient deux modules de liaison série (McBSPo et McBSP1).Ces liaisons série permettent des communications en full-duplex, un flot continu des données, une indépendance dans le timing et le format des données à l'émission et la réception, une interface directe avec les CODEC,CNA et CAN, et une synchronisation par horloge interne ou externe.

- Contrôleur PLL
Le contrôleur PLL est capable de générer différentes horloges pour différentes parties du noyau de DSP, l'interface externe de mémoire, McBS, périphériques, et d'autres modules à l'intérieur du DSP.

Jeu d'instructions
Le TMS320C6701 présente une liste très riche en instructions utilisées dans sa programmation en assembleur. On se limite de décrire quelques instructions.
- Instructions d'addition
• Instructions pour les entiers
* L'instruction ADD
Syntaxe ADD (.unit) src1, src2, dst
Ou ADDU (.L1 ou .L2) src1, src2, dst
Ou ADD (.D1 ou .D2) src2, src1, dst
.unit = .L1, .L2, .S1, .S2
Example: ADD .L2X A1, B1, B2
* L'instruction ADDAB
Syntaxe ADDAB (.unit) src2, src1, dst
.unit = .D1 ou .D2
• Addition de réelle double précision
Syntaxe ADDDP (.unit) src1, src2, dst
.unit = .L1 or .L2

Example: ADDDP .L1X B1:B0, A3: A2, A5: A4
- Instructions de multiplication
• Instructions pour les entiers
Syntaxe MPY (.unit) src1, src2, dst
Ou MPYU (.unit) src1, src2, dst
.unit = .M1 ou .M2
Exemple : MPYU .M1 A1, A2, A3
• Addition de réelle double précision
Syntaxe MPYDP (.unit) src1, src2, dst
.unit = .M1 ou .M2
Exemple : MPYDP .M1 A1:A0, A3:A2, A5:A4

Modes d’adressages

Les modes d'adressage sur le C671 sont : linéaires et circulaire en utilisant BK0 ou BK1. Le mode est indiqué par le registre de mode d'adressage, ou (AMR). Tous les registres peuvent exécuter l'adressage linéaire. Seulement huit registres peuvent exécuter l'adressage circulaire (A4–A7 sont employés par l'unité .D1, et B4–B7 employés par l'unité .D2).

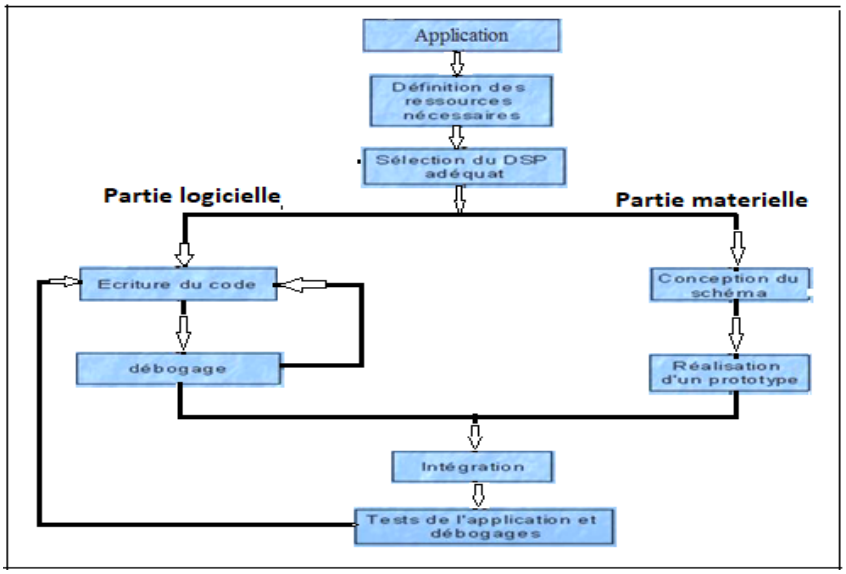
CHAPITRE Les processeurs numériques du siganl (DSP)

Partie : 3

Développement d'applications sur DSP

I. Méthodes et outils de développements des applications avec DSP

La partie matérielle :
Elle inclut le DSP et la création d'une chaîne d'acquisition et de restitution du signal (des signaux) à traiter.
Partie logicielle : elle s'appuie sur des outils classiques adaptés aux spécificités des DSP. L'approche est différente de celle utilisée pour la partie matérielle.



Définition des ressources nécessaires
Cette phase doit permettre d'évaluer les besoins nécessaires à la mise en œuvre du système de traitement numérique du signal voulu. Elle consiste notamment à définir les spécifications de la chaîne d'acquisition et de restitution du signal.

La sélection du DSP le plus adapté
La sélection d'un DSP se base avant tout sur la puissance de traitement nécessaire, et sur le résultat de benchmarks réalisant des fonctions représentatives des traitements à réaliser.

Structure matérielle de développement

- Un environnement (ou système) de développement pour DSP peut être scindé en deux parties principales: Un environnement de développement pour créer et mettre en forme le logiciel de l'application (création du source, utilisation des bibliothèques, assemblage).
- Un environnement de développement utilisant des outils spécifiques pour tester et déboguer le logiciel de l'application (simulateur, module d'évaluation, émulateur).

Le simulateur
Le simulateur est un programme particulier exécuté par un PC ou une station de travail. Son rôle consiste à simuler le plus exactement possible le fonctionnement du DSP cible. L'interface utilisateur du simulateur permet de consulter les mémoires, tous les registres internes du DSP, ses entrées/sorties, etc.
Le simulateur exécute chaque instruction DSP comme le ferai le DSP lui-même, et en répercute les résultats dans les mémoires et les registres simulés.

Le module d'évaluation
Le module d'évaluation se présente sous la forme d'une carte électronique incorporant le DSP cible et le minimum des ressources nécessaires à sa mise en œuvre, telles que des mémoires externes, le cas échéant une liaison série RS232, et une alimentation.

L'émulateur temps réel

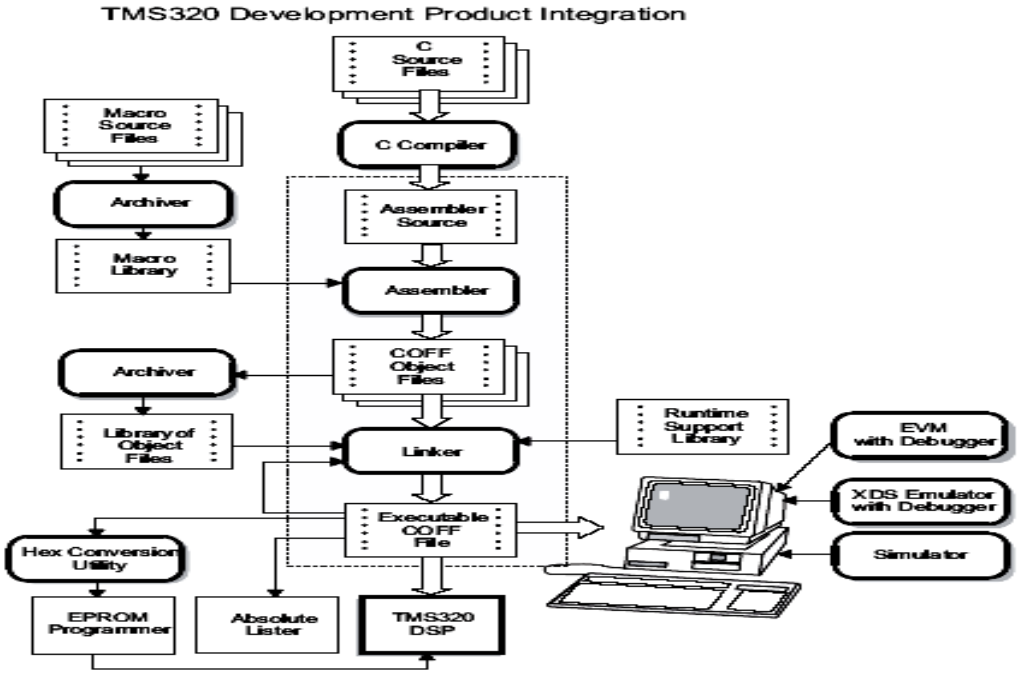
L'émulateur temps réel est l'outil privilégié pour développer des applications DSP. C'est l'outil le plus souple et le plus performant, car il ne souffre pas des limitations d'un simulateur ou d'un module d'évaluation. Son rôle consiste à émuler en temps réel le fonctionnement du DSP au sein même du prototype de l'application à développer. Toutes les ressources du DSP cible sont libres pour tester non seulement le code du programme de l'application, mais également le fonctionnement du prototype.

Structure logicielle de développement

Les deux principales méthodes pourv écrire un programme DSP consistent à utiliser un assembleur dédié ou un langage de haut niveau.

Programmation des DSP : système de développement

Les phases d'un développement typique d'une application, sont présentées sur l'organigramme de la figure suivante. Elle montre le système de développement complet d'un DSP de Texas Instruments



Déroulement typique de génération de code exécutable d'une application sur 'DSP' sous 'CCS'

1. **Le compilateur C**

Le compilateur traduit du code source C en code source assembleur du processeur.

2. **L'assembleur**

L'*assembleur* traduit les fichiers source en fichiers objet, ceux-ci étant alors en langage machine. Les fichiers sources contiennent des instructions (indiquées par leur *code mnémonique* et non par leur code binaire ou *opcode*) et directives assembleur. Les directives assembleur permettent de contrôler le processus d'assemblage, par exemple en offrant un assemblage conditionnel fonction de tel ou tel paramètre (version du processeur ou configuration de sa mémoire).

Le code objet produit est *relogeable*, i.e. il ne fait référence à aucune adresse absolue. Il pourra donc être placé et exécuté dans des zones de la mémoire du processeur inconnues au moment de l'assemblage.

L'assembleur effectue les opérations suivantes :

- test de l'exactitude du programme;
- génération de la table de symboles contenant la valeur (relative) de tous les symboles et de toutes les étiquettes définis par l'utilisateur dans le programme;
- mise à jour d'un compteur de position qui indique où doit être placée (relativement) en mémoire l'instruction ou la donnée suivante;

- transformation des noms symboliques (codes mnémoniques) des instructions et opérandes en leurs valeurs binaires (génération du programme objet);
- génération d'un listing (option).

L'assembleur ne produit donc pas de code directement exécutable. Le format des fichiers objet (langage machine) est bien défini. Dans le cas Texas, il s'agit du COFF (« Common Object File Format »).

3. L'éditeur de liens

L'éditeur de liens (*linker*) lie les fichiers objet générés par l'assembleur et produit un seul fichier exécutable.
Les différents fichiers objets générés par l'assembleur faisant très souvent référence les uns aux autres, le linker résout ces problèmes en remplaçant les références à des symboles externes par des adresses bien définies. Pour cela, il dispose d'informations complètes concernant la configuration de la mémoire du processeur (map file). Toutefois, le linker est également capable de générer un code exécutable relogeable.
Le code exécutable doit encore être chargé en mémoire du processeur, cette tâche n'**incombant pas** au linker mais à un programme de chargement séparé, dépendant de l'application (loader). Dans certains cas, le fichier exécutable est encore converti en un code compatible avec des programmateurs d'EPROMs.

4. Les outils de debugging

Pour débugger le code, les outils principaux sont l'*émulateur* et le *simulateur*. L'émulateur est à même d'émuler par hardware le fonctionnement du processeur en s'y substituant sur la carte.
Le simulateur permet une émulation exclusivement par logiciel, ce qui exclut par exemple l'accès aux périphériques et rend le test des programmes correspondants plus délicat.
Dans un cas comme dans l'autre, des fonctions indispensables au debugging sont à disposition, comme le fonctionnement pas par pas, la définition de points d'arrêt, la visualisation en tout temps des registres du processeur et de la mémoire, etc. On peut à la fois travailler avec le code assembleur et le code C.
Le simulateur constitue le minimum indispensable au développement d'un projet.
Toutefois, le programmeur doit souvent avoir recours à l'émulateur, outil en général beaucoup plus coûteux même si son prix n'est plus comparable à ce qu'il était il y quelques années, i.e. avant que les microprocesseurs ne soient systématiquement dotés de l'interface JTAG.

5. L'archiver

L'*archiver* permet de grouper des fichiers dans une seule librairie. Le code source assembleur peut alors faire référence à des routines se trouvant dans la librairie, le nom de celle-ci étant spécifié lors de l'assemblage ou du lien. On peut par exemple écrire en assembleur plusieurs routines mathématiques et les réunir ensuite en un seul groupe logique par le moyen de l'archiver.

II. Evaluation des performances des algorithmes sur les DSP

Pour évalue les performances des algorithmes sur les DSP, on peut utiliser Code Composer Studio (CCS) par exemple pour les DSP produits par Texas Instruments.

Environnement logiciel

Code Composer Studio (CCS)

Code Composer Studio (CCS) fournit plusieurs outils pour faciliter la construction et la mise au point des programmes de DSP. Il comprend un éditeur de code source, un compilateur de langage C/C++, un assembleur de code re-localisable, un éditeur de liens, et un environnement d'exécution qui permet de télécharger un programme exécutable sur une carte cible, de l'exécuter et de le déboguer au besoin. CCS comprend aussi des outils qui permettent l'analyse en temps réel d'un programme en cours d'exécution et des résultats produits. Finalement, il fournit un environnement de gestion de fichiers qui facilite la construction et la mise au point des programmes.

Le logiciel Code Composer Studio(CCS) est une plate-forme de développement qui inclut les éléments suivants,

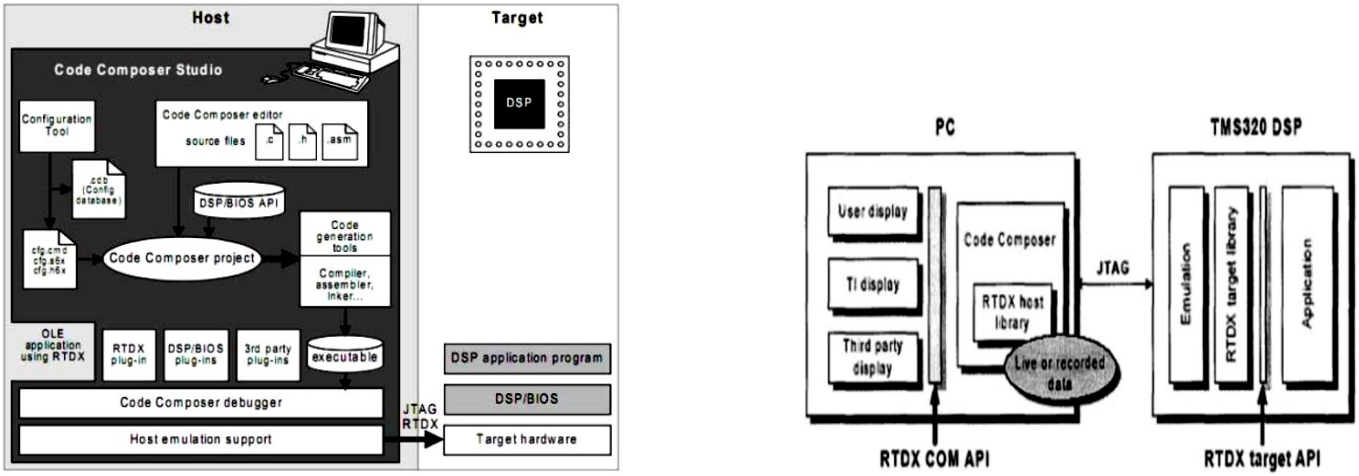


Plate-forme de développement du logiciel Code Composer Studio

1. Environnement de développement intégré (IDE) : il permet l'édition ('built'), et la correction ('debug') des programmes destinés au DSP.

2. Outils de génération du code pour les dsp de TI : ces outils sont le compilateur, l'assembleur et l'éditeur de lien :

   - Le compilateur C/C++ permet de compiler le programme source (xxx.c) pour le convertir en assembleur (xxx.asm),
   - L'assembleur reçoit le fichier xxx.asm et le convertit en langage machine ou fichier objet (xxx.obj),
   - L'éditeur de liens (linker) qui combine les fichiers objet et les fichiers librairies et le fichier xxx.cmd pour produire un fichier exécutable avec une extension .out, c'est ce fichier qui sera chargé sur le DSP pour être exécuter.

3. DSP/BIOS: c'est un outil d'analyse en temps réel, pour s'en servir, on doit créer un fichier de configuration 'xxx.cdb', où seront définis les objets utilisés par l'outil DSP/BIOS. Ce fichier permet aussi de faciliter l'organisation de la mémoire et la gestion du vecteur des interruptions, en offrant la possibilité de les faire sur un environnement visuel via la section de gestion de la mémoire MEM (Memory Section Manager), et via HWI (Hardware Interrupt Service Routine Manager) pour les interruptions.

4. JTAG (Joint Team Action Group) et RTDX (Real Time Data Exchange) le RTDX permet un échange de données en temps réel entre l'hôte (PC par exemple) et la destination, il permet aussi l'analyse et la visualisation des données au cours de l'exécution du programme, alors que le lien JTAG est utilisé pour atteindre l'émulateur (qui se trouve à l'intérieure du DSP), c'est ce dernier qui permet au CCS de contrôler en temps réel l'exécution du programme.

5. Simulateur intégré : le logiciel Code Composer Studio offre la possibilité de tester des programmes pour DSP sans utiliser la carte DSK à l'aide d'un simulateur intégré.

Création du fichier exécutable

La réalisation d'une application avec CCS se fait par la création d'un projet (un fichier avec extension .pjt) suivi de la configuration des différentes options nécessaires à son exécution. Dans un projet, on retrouve plusieurs fichiers regroupés selon leurs types dans des répertoires différents, ces répertoires sont :

1. Source : contient les fichiers sources du projet, ces fichiers peuvent être des programmes en langage C, C++, ou/et des programmes en assembleur * .asm.
2. Include: contient les fichiers de l'en-tête *.h.
3. Libraries : contient les fichiers librairies *.lib.

Pour créer le fichier exécutable qui sera charge sur le DSP, il faut passer par plusieurs étapes :

A. Création d'un répertoire de travail

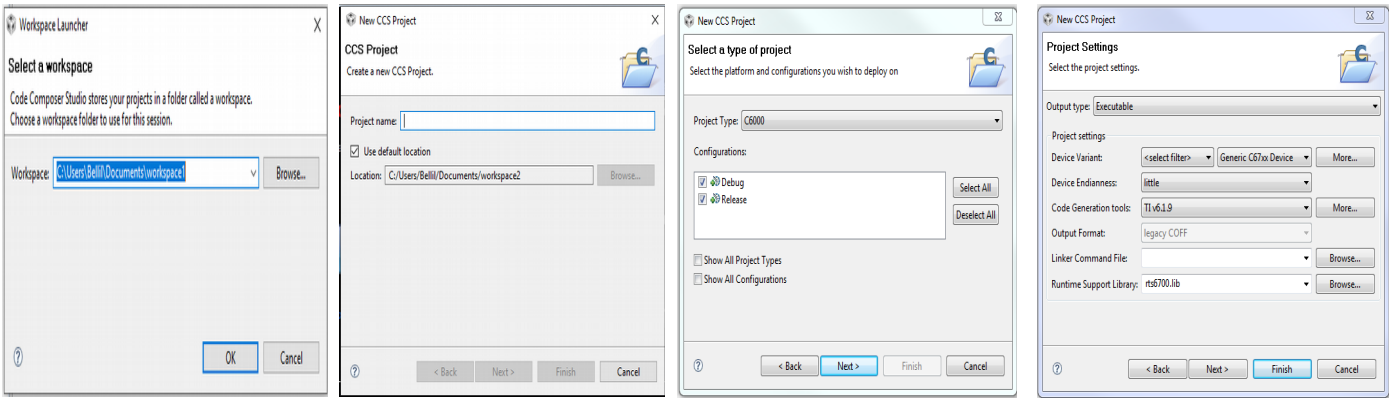
Le répertoire de travail est l'endroit où vos projets seront sauvegardés. Lors de la première utilisation de CCS, il devra se créer un répertoire de travail sur votre espace personnel sur le disque C.
Le changement du répertoire de travail peut s'effectuer à partir la commande du menu «File→ Switch Workspace» puis il faut entrer le nom de l'espace de travail dans la fenêtre.

B. Création d'un projet

Le processus de développement de code de CCS commence par la création d'un projet qui facilite l'intégration des fichiers requis pour produire un fichier exécutable. Le répertoire projet contient des références aux fichiers source (*.c, *.asm) aux fichiers d'en-tête (*.h), au fichier de commande pour l'éditeur de liens (*.cmd), et aux fichiers de bibliothèque (*.lib). Le projet permet aussi de spécifier les paramètres du compilateur, de l'assembleur et de l'éditeur de liens afin de produire le fichier exécutable.

Après l'ouverture du logiciel, on commence par créer un projet, vous choisissez l'élément de menu « File →New→CCS Project » de la barre menu.

- vous introduisez un nom de projet dans le champ « Project name » (votre nom, par exemple).
- Poursuivre (NEXT) avec le type du projet en sélectionnant : « Project Type : C6000» pour le TMS320C6701.
- Poursuivre (NEXT) et sélectionner dans «Device Variant: Generic C67xx Device» pour le TMS320C6701 et «Runtime Support Library : RTS6700.lib».
- Poursuivre (NEXT) et sélectionner dans la fenêtre «Project Templates» l'item «Empty Projects / Empty Project Device».



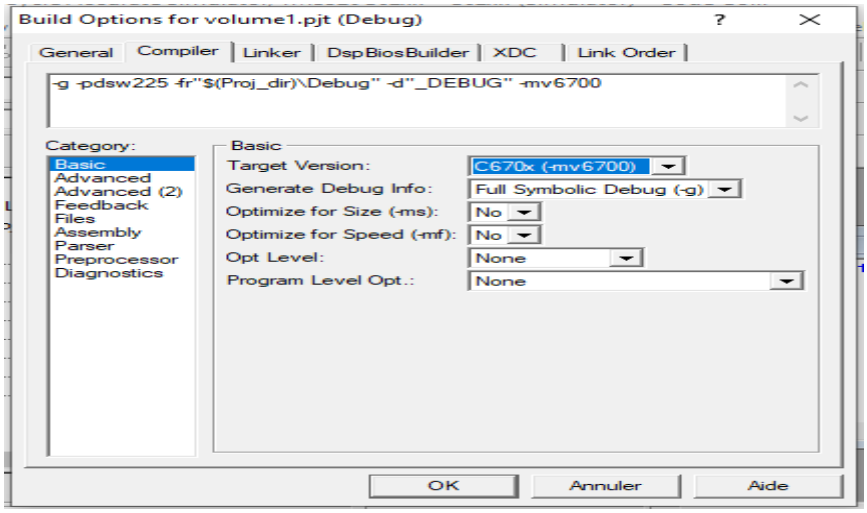
C. Création un fichier source

CCS fournit un éditeur intégré qui permet la création des fichiers sources. Une fenêtre d'éditeur apparaît en choisissant l'élément de menu « File → New File ». Il faut écrire le programme principal en C ou en Assembleur.

D. Ajout de fichiers de support au projet

Pour le bon fonctionnement du programme, en plus des fichiers sources, des fichiers supports pour le DSP TMS320C6701et la carte cible DSK6701 devront être inclus au projet sous forme de librairie et répertoire «Include».

- Pour joindre tous les fichiers supports, on doit cliquer dans le menu principal sur « Project », puis « add files to project », et finalement sur « open ». Joindre aussi le fichier de commande (.cmd ou .lcf). Ce type de fichier divise la mémoire de la carte en des sections.
- Avant de générer le code, il faut configurer les options de compilation et de lien (Project build options). (Voir figure).



E. Génération d'un fichier exécutable

Après avoir ajouté tous les fichiers sources, fichier de commande et fichier de bibliothèque au projet, vous pouvez construire le projet et créer un fichier exécutable pour le DSP-cible.
Pour cela, choisissez l'élément de menu «Project→Build Active Project». Cette fonction permet de compiler, assembler, et joindre tous les fichiers dans le projet. Un le fichier exécutable avec l'extension .out est généré. Ce dernier est ensuite chargé sur le DSP (File Load) et finalement, il ne reste qu'à exécuter le programme (Debug Run).

Les Fichiers supports

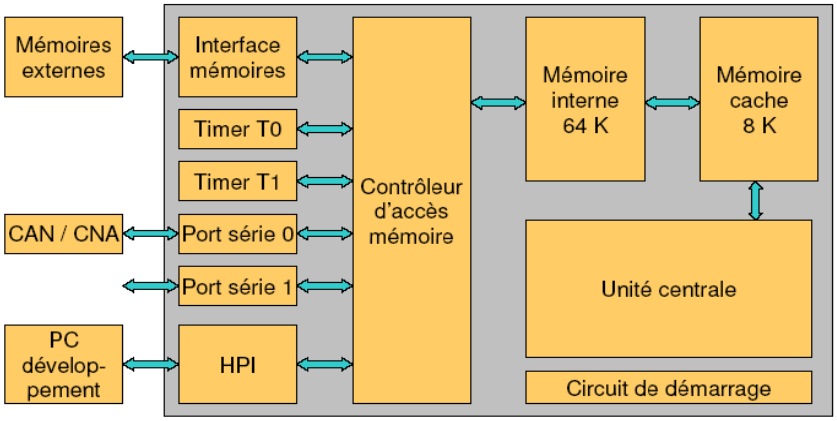
1. C6701dskinit.c: contient les fonctions initialisant le DSK, le codec, les ports séries et les entrées/sorties. Il n'est pas inclus avec CCS.
2. C6701dskinit.h: fichier en-tête avec des fonctions prototypes. Des fonctionnalités telles que celles utilisées pour sélectionner l'entrée micro au lieu d'entrée- ligne (par défaut), le gain d'entrée, et ainsi de suite, elles sont toutes obtenues à partir de ce fichier.
3. C6701dsk.cmd: fichier de commande. Ce fichier peut être modifié lors de l'utilisation de la mémoire externe à la place de la mémoire interne.
4. Vectors_intr.asm: une version modifiée d'un « vector file » inclus avec le CCS pour gérer les interruptions. Douze interruptions, INT4 à INT15, sont disponibles, INT11 est sélectionnée pour ce « vector file». Elles sont utilisées pour les programmes d'interruption pilotés.
5. Vectors_poll.asm: c'est un « vector file» pour les programmes utilisant le polling.
6. rts6700.lib, dsk6701bsl.lib, csl6701.lib: run-time de la carte, et bibliothèques supports pour la le dsp, respectivement. Ces trois fichiers d'extension .lib sont localisés dans : C6000\cgtools\lib, C6000\dsk6701\lib, et c6000\bios\lib, respectivement.

Environnement matériel et la plate-forme expérimentale

Les cartes d'évaluation

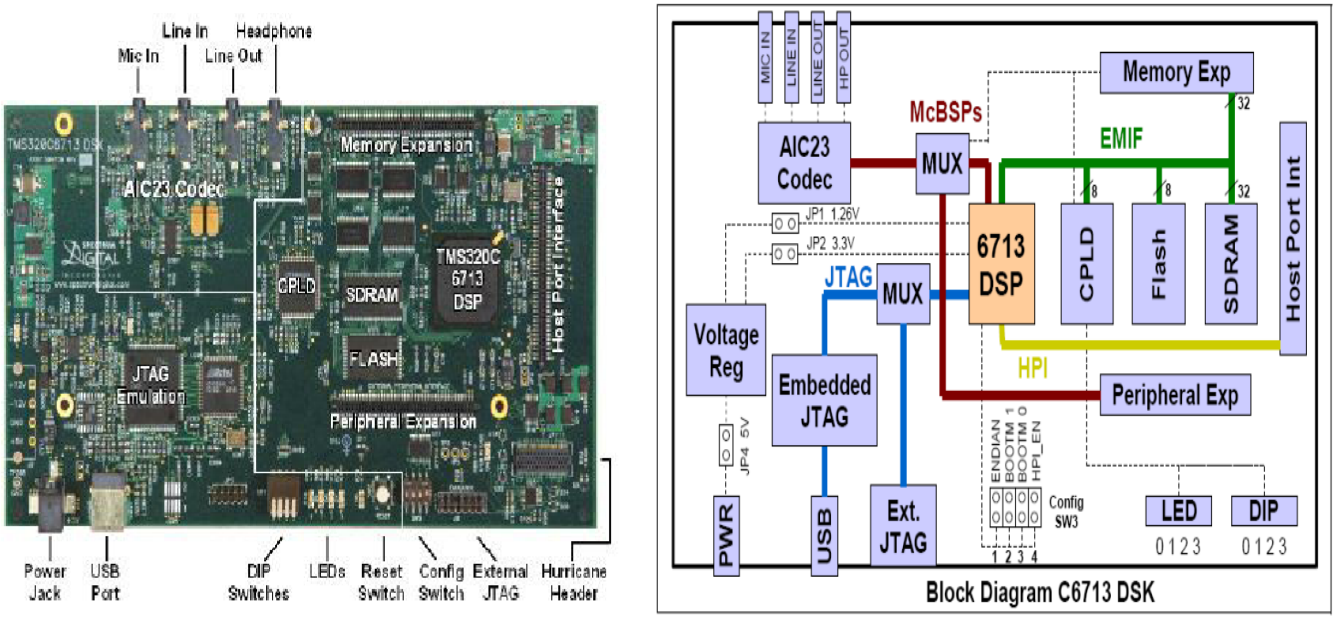
Elles regroupent les diverses parties constituant la chaîne de traitement de signal en disposant d'outils de support matériels et logiciels nécessaires pour le développement et l'implantation en temps réel, des algorithmes destinés au traitement de signal.
Les cartes intègrent des interfaces de communication, des interfaces analogiques (voix et données), et des ports d'entrée/sortie, dans une seule carte pour former un système d'évaluation complet.
Les constituants principaux sont:

- Un DSP
- Une interface de périphériques parallèle.
- Mémoires 'SDRAM'
- Mémoires 'ROM'.
- Un circuit d'interface analogique
- Un port d'entrée sortie.
- Un support d'émulation 'JTAG' intégré.



Les cartes d'évaluation DSK de Texas instruments
Ce sont des cartes électroniques conçues autours d'un processeur de traitement de signal de Texas instruments. Elles peuvent être relie à un micro-ordinateur contenant l'outil de développement software CCS, simplifiant la saisie et le débogage des algorithmes destinés aux traitements.

Description générale (DSK C6713)



Carte d'évaluation DSK C6713 de Texas instruments

Caractéristiques du DSK C6713

- DSP TMS320C6713 à virgule flottante fonctionnant à 225 MHz.
- Codec stéréo de 16 bits (AIC23) : fonctionne comme un CAN pour les entrées analogiques et comme un CNA pour les sorties numériques du DSP. Il dispose de quatre connecteurs permettent des entrées et des sorties analogiques:
  - MIC IN : entrée de microphone,
  - LINE IN: entrée analogique “ligne”
  - LINE OUT: sortie analogique “ligne”
  - HEADPHONE: sortie écouteurs/casque (multiplexée avec l’entrée ligne).
- Mémoire SDRAM 16 Mbytes
- Mémoire flash de 512 Kbytes de mémoire flash 4 LED et 4 interrupteurs (DIP switches)
- Configuration de la carte par programmation du CPLD
- Deux connecteurs de 80 broches permettent des extensions de périphériques et mémoire externes.
- Un émulateur JTAG intégré jumelé avec la connectivité USB

Implantation des algorithmes sur un kit DSK

L’implantation d’un algorithme sur le kit est réalisée comme suit :

- Le code source est écrit en langage C sur le PC, en utilisant les utilitaires de l’interface CCS.
- Le code est compilé, assemblé et lié, donnant un fichier de format objet ‘COFF ’.
- Le fichier objet est chargé sur la carte ‘DSK’ à travers le port parallèle.
- Initialisation du kit par PC, l’exécution du programme devient autonome.