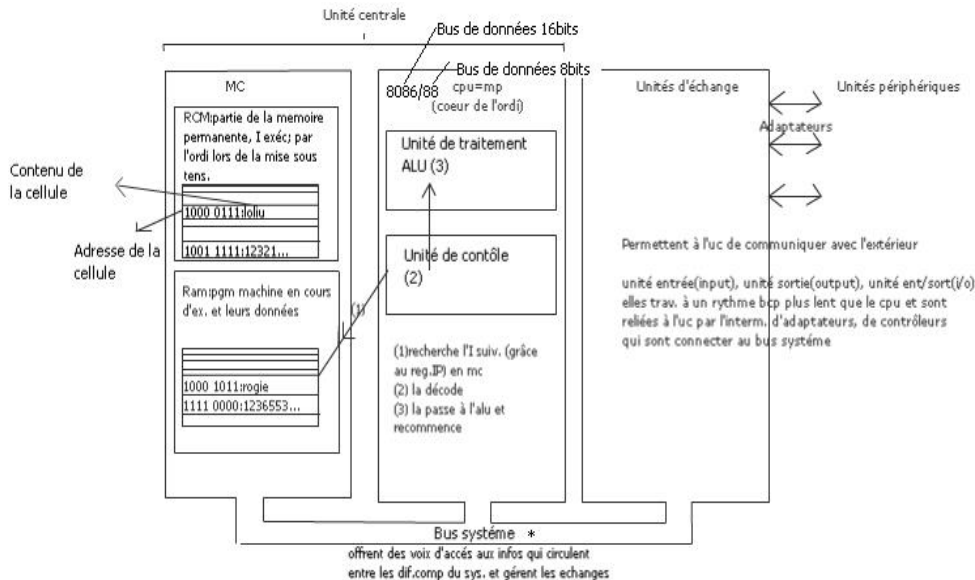


# TD 3 ADOintro. Assembleur 80x86

## Complément de cours 1 : Architecture des processeurs 80x86 (INTEL)

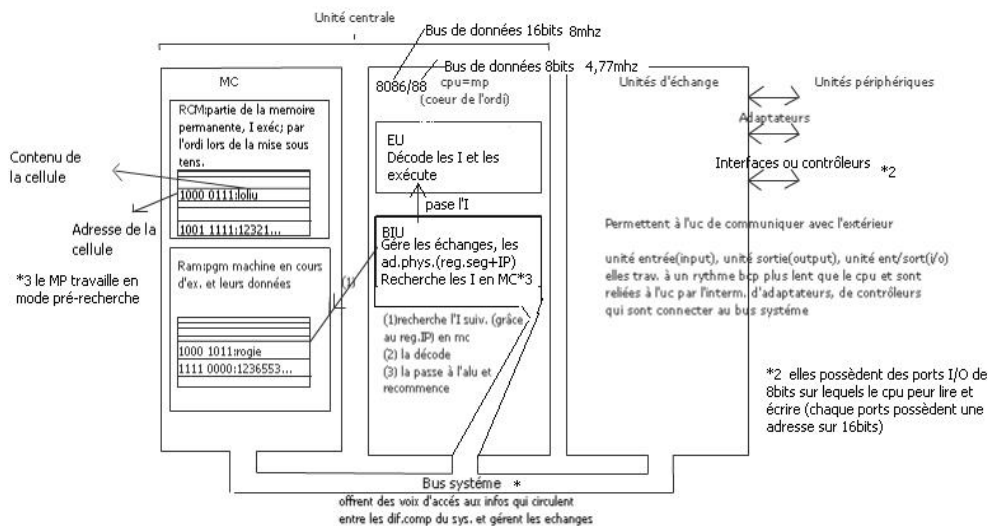
### Structure d'un ordinateur



**Bus d'adresses (20bits) :** Utilisé par le CPU (unité centrale – Central Processor Unit) pour adresser une case mémoire ou un port I/O, le CPU calcule d'abord l'adresse physique (20bits) puis la positionne sur le bus d'adresse

**Bus de données (16bits) :** transfert des informations entre la mémoire et le CPU

**Bus de commande :** circulation des signaux de commande



### Les registres

Zones mémoires processeur pour stocker des informations afin d'y accéder plus rapidement que par des accès à la mémoire centrale (RAM).

### Les registres généraux

**AX, BX, CX et DX** sont à usage général et servent à stocker temporairement une information de 16 bits.

Chacun de ces registres est divisible en 2 registres de 8 bits. Exemple pour AX : AH = octet de poids fort de AX. AL = octet de poids faible de AX.

**AX** (accumulateur) : instructions d'I/O et certaines opérations arithmétiques.

**BX** (registre de base) : peut être aussi utilisé comme registre d'adressage lors de l'adressage direct à la mémoire.

**CX** (compteur) : peut être aussi utilisé comme compteur de boucles

**DX** : peut être aussi utilisé pour contenir les adresses des ports pour les instructions I/O.

**Les registres d'index et pointeurs**

**SI** et **DI** peuvent être utilisés comme registres généraux de données mais le plus souvent sont utilisés avec les instructions spécialisées de manipulation des chaînes de caractères et aussi comme registres d'adressage (indexé).

**SP** et **BP** accèdent aux données de la pile (**SS : SP** = sommet de la pile et **BP** accède à des données dans la pile).

**CS** (Code Segment) et **IP** pointeur d'instruction : **CS : IP** = adresse de l'instruction suivante à exécuter. (CS contient la partie haute de l'adresse)

**Les registres de segments**

**CS, DS, ES** et **SS** permettent de calculer l'adresse Physique d'une donnée (en spécifiant la partie haute de l'adresse) à partir de son adresse logique

**Le registre des indicateurs**

**FR** est un masque de 16 bits dont 9 seulement sont significatifs; ils représentent à tout moment l'état logique du cpu (processeur – Central Processor Unit) et décrivent la manière dont se sont déroulées certaines opérations.

**Flags de contrôle qui modifient le fonctionnement du cpu**

**IF** quand il est à 1 le cpu peut être modifié par des événements extérieurs

**TF** quand il est à 1 le cpu fonctionne en mode pas à pas

**DF** permet de modifier la mise à jour des registres d'index lors des opérations de manipulation de chaînes de caractères

**Flags indicateurs d'état**

**CF** à 1 il indique la retenue lors de la dernière opération arithmétique

**PF** à 1 nombre de bits pair

**SF** à 1 résultat négatif (entier signé)

**ZF** à 1 résultat nul ou opérandes égaux

**OF** à 1 il indique qu'il y a eu débordement

**AF** à 1 indique une retenue

**Plus d'informations**

[http://www.google.fr/url?sa=t&source=web&cd=1&ved=0CB8QFjAA&url=http%3A%2F%2Fentraide-epfc.sirenacorp.be%2Fscript.redirDownload.php%3FID\\_download%3D103%26url%3Ddownload%2F2007\\_01\\_12\\_Rsumerdassembleur.doc&rct=j&q=architecture%208086&ei=4VQOTdySMJGo8QPBvM2DBw&usg=AFQjCNFof1I86l105-h5-N\\_DvadhXay-xg&cad=rja](http://www.google.fr/url?sa=t&source=web&cd=1&ved=0CB8QFjAA&url=http%3A%2F%2Fentraide-epfc.sirenacorp.be%2Fscript.redirDownload.php%3FID_download%3D103%26url%3Ddownload%2F2007_01_12_Rsumerdassembleur.doc&rct=j&q=architecture%208086&ei=4VQOTdySMJGo8QPBvM2DBw&usg=AFQjCNFof1I86l105-h5-N_DvadhXay-xg&cad=rja)

## Complément de cours 2 : Jeu d'instructions 80x86

### I ) introduction

On peut diviser les instructions du 8086/88 en 6 groupes comme suit :

- Instructions de transfert de données.
- Instructions arithmétiques.
- Instructions de bits (logiques).
- Instructions de sauts de programme.
- Instructions de chaîne de caractères.
- Instructions de contrôle de processus.
- Instructions d'interruptions.

### II ) Les instructions de transfert de données

Elles sont divisées en 4 sous- groupes comme le montre le tableau suivant :

Usage	Nom	Fonction
Général	MOV	Transfert d'octets ou de mots
	PUSH	Chargement de la pile
	POP	Déchargement de la pile
	PUSHA	Chargement de tous les registres dans la pile
	POPA	Déchargement de tous les registres dans la pile
	XCHG	Echange d'octet ou de mot dans la pile
Entrées-sorties	XLAT	Translation d'octet
Adresses	IN	Entrée de mot ou d'octet
	OUT	Sortie de mot ou d'octet
Indicateurs	LEA	Chargement de l'adresse effective
	LDS	Chargement du pointeur avec DS
	LES	Chargement du pointeur avec ES
	LAHF	Transfert des indicateurs dans AH
	SAHF	Rangement de AH dans les indicateurs
	PUSHF	Chargement des indicateurs dans la pile
	POPF	Déchargement des indicateurs de la pile.

#### II-1 ) Les instructions d'usage général

##### II-1-1 ) MOV

Copie de données (un octet ou un mot) d'un registre vers un autre registre ou d'un registre vers une case mémoire, sa syntaxe est comme suit :

##### Exemples

Syntaxe : MOV destination, source

```
MOV  AX, BX      ; Copie du contenu d'un registre 16 bits vers un registre 16 Bits : (BX) → AX
MOV  AH, CL      ; Copie du contenu d'un registre 8 bits vers un registre 8 bits : (CL) → AH
MOV  AX, Val1    ; Copie du contenu d'une case mémoire 16 bits vers AX : (Val1) → AX
MOV  Val2, AL    ; Copie du contenu de AL vers une case mémoire d'adresse Val2 : (AL) → Val2
```

##### Remarques

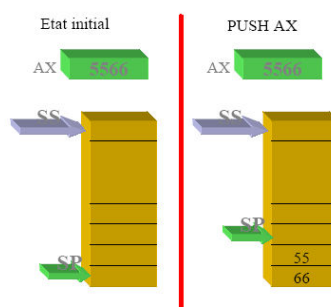
- Il est strictement interdit de transférer le contenu d'une case mémoire vers une autre case mémoire
- On n'a pas le droit aussi de transférer un registre segment vers un autre registre segment sans passer par un autre registre.

##### II-1-2 ) PUSH

Elle permet d'empiler les registres du CPU sur le haut de la pile

Syntaxe : PUSH SOURCE

Exemple :

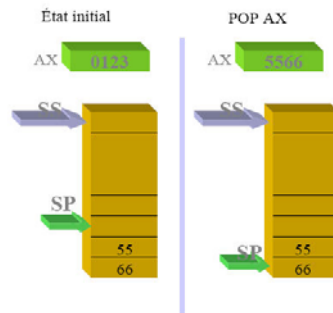


## II-1-3 ) POP

Elle permet de dépiler les registres du CPU sur le haut de la pile

Syntaxe : POP destination

## Exemple



## III ) Instructions arithmétiques

Les instructions arithmétiques peuvent manipuler quatre types de nombres :

- Les nombres binaires non signés
- Les nombres binaires signés.
- Les nombres décimaux codés binaires (DCB), non signés.
- Les nombres DCB non condensés, non signés.

Les instructions arithmétiques sont divisées en quatre sous-groupes comme le montre le tableau suivant :

Usage	Nom	Fonction
Addition	ADD	Addition sur un octet ou un mot
	ADC	Addition sur un octet ou un mot avec retenue
	INC	Incrémentation de 1
	AAA	Ajustement ASCII
	DAA	Ajustement décimal
Soustraction	SUB	Soustraction sur un octet ou un mot
	SBB	Soustraction sur un octet ( mot ) avec retenue
	DEC	Décrémentation de 1
	NEG	Mètre un octet ou un mot en négatif
	CMP	Comparaison d'octet ou mot
	AAS	Ajustement ASCII
	DAS	Ajustement décimal
Multiplication	MUL	Multiplication d'octet ou de mot <u>non signée</u>
	IMUL	Multiplication d'octet ou de mot <u>signée</u>
	AAM	Ajustement ASCII
Division	DIV	Division d'octet ou de mot <u>non signée</u>
	IDIV	Division d'octet ou de mot <u>signée</u>
	AAD	Ajustement ASCII
	CBW	Conversion d'un octet en un mot
	CWD	Conversion d'un mot en double mots

## III-1 ) Addition

## III-1-1 ) ADD: (Addition)

Syntaxe : ADD Destination, source

Elle permet d'additionner le contenu de la source (octet ou un mot) avec celui de la destination le résultat est mis dans la destination

Destination <----- Destination + source

## Exemples

ADD AX, BX ; AX = AX + BX (addition sur 16 bits) ADD AL, BH  
; AL = AL + BH (addition sur 8 bits )

ADD AL, [SI] ; AL = AL + le contenu de la case mémoire pointée par SI

ADD [DI], AL ; le contenu de la case mémoire pointée par DI est additionnée avec AL, le résultat est mis dans la case mémoire pointée par DI

## III-1-3 ) INC : (Incrémentation)

Syntaxe : INC Destination

Elle permet d'incrémenter le contenu de la destination : Destination <----- Destination + 1

## Exemples

INC AX ; AX = AX + 1 (incrémenter sur 16 bits).

INC AL ; AL = AL + 1 (incrémenter sur 8 bits).

INC [SI] ; [SI] = [SI] + 1 le contenu de la case mémoire pointée par SI sera incrémenté

**III-2 ) Soustraction****III-2-1 ) SUB : (Soustraction)**

Syntaxe : SUB Destination, source

Elle permet de soustraire la destination de la source (octet ou un mot) le résultat est mis dans la destination

Destination <----- Destination -- source

**Exemples**

SUB AX,BX ; AX = AX - BX (Soustraction sur 16 bits )  
 SUB AL,BH ; AL = AL - BH ( Soustraction sur 8 bits )  
 SUB AL,[SI] ; AL = AL - le contenu de la case mémoire pointée par SI  
 SUB [DI],AL ; le contenu de la case mémoire pointée par DI est soustraite de AL , le résultat est mis  
 ; dans la case mémoire pointée par DI

**III-2-3 ) DEC : (Décrémentement)**

Syntaxe : DEC Destination

Elle permet de décrémenter le contenu de la destination

Destination <----- Destination - 1

**Exemples**

DEC AX ; AX = AX - 1 (décrémentement sur 16 bits).  
 DEC AL ; AL = AL -1 (décrémentement sur 8 bits).  
 DEC [SI] ; [SI] = [SI] - 1 le contenu de la case mémoire pointée par SI sera décrémenté

**III-2-5 ) CMP : (Comparaison)**

Syntaxe : CMP Destination , Source

Elle soustrait la source de la destination , qui peut être un octet ou un mot , le résultat n'est pas mis dans la destination , en effet cette instruction touche uniquement les indicateurs pour être tester avec une autre instruction ultérieure de saut conditionnel

Les indicateurs susceptibles d'être touché sont : AF, CF, OF, PF, SF, ZF

Donc cette instruction va nous permettre de comparer deux nombres comme le montre le tableau suivant :

	Opérande non signé				Opérande signé			
	OF	SF	ZF	CF	OF	SF	ZF	CF
Source < destination	-	-	0	0	0/1	0	0	-
Source = destination	-	-	1	0	0	0	1	-
Source > destination	-	-	0	1	0/1	1	0	-

**III-3 ) La multiplication :****III-3-1 ) MUL : (Multiplication pour les nombres non signés)**

MUL effectue une multiplication non signée de l'opérande source avec l'accumulateur :

Syntaxe : MUL Source

- Si la source est un octet alors elle sera multipliée par l'accumulateur AL le résultat sur 16 bits sera stocké dans le registre AX.
- Si la source est un mot alors elle sera multipliée avec l'accumulateur AX le résultat de 32 bits sera stocké dans la paire des registres AX et DX

**En conclusion :**

Multiplication	Opérande 1	Opérande 2	Résultat
Octet x Octet	AL	Registre ou memoire	AX
Mots x Mots	AX	Registre ou memoire	DX AX
Mots x Octet	AL= Octet, AH=0	Registre ou memoire	DX AX

**III-4 ) La division****III-4-1 ) DIV : (Division des nombres non signés)**

Syntaxe : DIV Source

Elle effectue une division non signée de l'accumulateur par l'opérande source :

**Exemples**

- Si l'opérande est un octet : alors on récupère le quotient dans le registre AL et le reste dans le registre AH.
- Si l'opérande est un mot : alors on récupère le quotient dans le registre AX et le reste dans le registre DX

a)  
 MOV AH,00h  
 MOV AL,33H  
 MOV DL,25H  
 DIV DL

b)  
 MOV AX,500H  
 MOV CX,200H  
 DIV CX

## IV ) Les instructions logiques ( de bits )

Ils sont divisés en trois sous-groupes comme le montre le tableau suivant :

Usage	Nom	Fonction
Logique	NOT	Inversion logique sur un octet ou un mot
	AND	Et logique
	OR	Ou logique
	XOR	Ou exclusif
	TEST	Et logique sans résultat, affecte uniquement les indicateurs du registre des flags.
Décalages	SHL	Décalage logique à gauche
	SAL	Décalage arithmétique à gauche
	SHR	Décalage logique à droite
	SAR	Décalage arithmétique à droite
Rotation	ROL	Rotation à gauche
	ROR	Rotation à droite
	RCL	Rotation à gauche à travers le bit de retenue
	RCR	Rotation à droite à travers le bit de retenue

Syntaxe des instructions de rotation et de décalage :

Exemple : Décalage logique à droite

```
SHR destination, compteur
```

## Exemple

```
SHR AX,1 ; décale le contenu de AX, noté (AX) d'un cran vers la droite :
          ; réalise une division par 2 de (AX)
          ; l'ancien LSB est perdu à cause du décalage; le nouveau MSB est un 0.
```

## V-1 ) Branchement inconditionnel

## V-1-1 ) CALL : notion de procédure :

La notion de procédure en assembleur correspond à celle de fonction en langage C, ou de sous-programme dans d'autres langages.

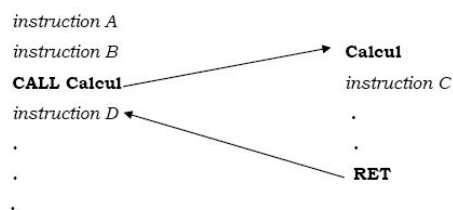


FIG. – Appel d'une procédure.

La procédure est nommée calcul. Après l'instruction B, le processeur passe à l'instruction C de la procédure, puis continue jusqu'à rencontrer RET et revient à l'instruction D.

Une procédure est une suite d'instructions effectuant une action précise, qui sont regroupées par commodité et pour éviter d'avoir à les écrire à plusieurs reprises dans le programme.

Les procédures sont repérées par l'adresse de leur première instruction, à laquelle on associe une étiquette en assembleur.

L'exécution d'une procédure est déclenchée par un programme *appelant*. Une procédure peut elle-même appeler une autre procédure, et ainsi de suite.

Instructions CALL et RET

L'appel d'une procédure est effectué par l'instruction CALL.

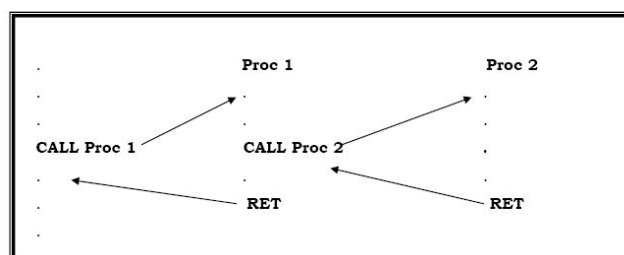
CALL *adresse\_debut\_procedure*

L'adresse est sur 16 bits, la procédure est donc dans le même segment d'instructions. CALL est une nouvelle instruction de branchement inconditionnel. La fin d'une procédure est marquée par l'instruction RET :

## V-1-2 ) RET :

RET ne prend pas d'argument ; le processeur passe à l'instruction placée immédiatement après le CALL.

RET est aussi une instruction de branchement : le registre IP est modifié pour revenir à la valeur qu'il avait avant l'appel par CALL. Comment le processeur retrouve-t-il cette valeur ? Le problème est compliqué par le fait que l'on peut avoir un nombre quelconque d'appels imbriqués, comme sur la figure suivante :



L'adresse de retour, utilisée par RET, est en fait sauvegardée sur la pile par l'instruction CALL. Lorsque le processeur exécute l'instruction RET, il dépile l'adresse sur la pile (comme POP), et la range dans IP.

L'instruction CALL effectue donc les opérations :

- Empiler la valeur de IP. A ce moment, IP pointe sur l'instruction qui suit le CALL.
- Placer dans IP l'adresse de la première instruction de la procédure (donnée en argument).

Et l'instruction RET :

- Dépiler une valeur et la ranger dans IP.

**Remarque 1 :**

Si la procédure appartient au même segment que le programme principal elle est dite de type NEAR sinon elle est dite de type FAR, la différence entre eux c'est que dans le premier cas le processeur doit empiler une seule valeur dans la pile c'est le registre IP mais dans le deuxième cas il faut empiler le registre IP ainsi que le registre segment CS et bien sur il les dépile pendant le retour de la procédure.

**Remarque 2 : Passage de paramètres**

En général, une procédure effectue un traitement sur des données (paramètres) qui sont fournies par le programme appelant, et produit un résultat qui est transmis à ce programme. Plusieurs stratégies peuvent être employées :

1. *Passage par registre* : les valeurs des paramètres sont contenues dans des registres du processeur. C'est une méthode simple, mais qui ne convient que si le nombre de paramètres est petit (il y a peu de registres).
2. *Passage par la pile* : les valeurs des paramètres sont empilées. La procédure lit la pile.

**V-1-3 ) JMP : (Saut inconditionnel)**

Syntaxe : JMP cible

Si le JMP est de type NEAR alors IP = IP + Déplacement

Si le JMP est de type FAR alors CS et IP sont remplacé par les nouvelles valeurs obtenues à partir de l'instruction.

JMP transfère, sans condition, la commande à l'emplacement de destination. L'opérande Cible peut être obtenu à partir de l'instruction elle-même (JMP direct) ou à partir de la mémoire ou à partir d'un registre indiqué par l'instruction.

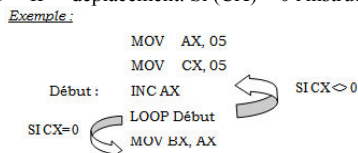
**V-2 saut conditionnel**

JC : (Saut si retenue)	Si CF=1 alors IP = IP + déplacement
JE/JZ : (Saut si égal/Si zéro)	Si ZF=1 alors IP = IP + déplacement
JNC : (Saut si pas de retenue)	Si CF=0 alors IP = IP + déplacement
JNE/JNZ : (Saut si non égal ) Non zéro)	Si ZF=0 alors IP = IP + déplacement
JNO : (Saut si pas de débordement)	Si OF=0 alors IP = IP + déplacement
JNP/JPO : (Saut si pas de parité/ Si parité impaire)	Si PF=0 alors IP = IP + déplacement
JNS : (Saut si pas de signe)	Si SF=0 alors IP = IP + déplacement
JO : (Saut si débordement)	Si OF=1 alors IP = IP + déplacement
JP/JPE : (Saut si parité (paire))	Si PF=1 alors IP = IP + déplacement
JS : (Saut si signe (négatif))	Si SF=1 alors IP = IP + déplacement

**V-3 ) Les instructions de boucle**

**V-3-1) LOOP : (boucle) :**

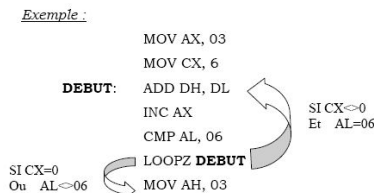
Elle décrémente le contenu de CX de 1. Si (CX) ≠ 0 alors IP = IP + déplacement. Si (CX) = 0 l'instruction suivante est exécutée.



L'exécution de l'instruction MOV BX, AX sera faite après l'exécution de la boucle 5 fois.

**V-3-2) LOOPE / LOOPZ : (boucle si égale ou si égale à zéro) :** Le registre CX est décrémenter de 1 automatiquement

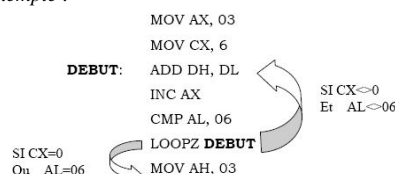
Si CX est différent de zéro et ZF=1 alors IP = IP + déplacement



**V-3-3 ) LOOPNE / LOOPNZ : (boucle si égale ou si égale à zéro) :** Le registre CX est décrémenter de 1 automatiquement

Si CX est différent de zéro et ZF=0 alors IP = IP + déplacement

Exemple :



## VII ) Les instructions de commande du processeur

Ces instructions agissent sur le processeur et ses indicateurs (Flags) ils sont en nombre de 12 comme le montre le tableau suivant

Type	Nom	Fonction
Indicateur (FLAGS)	STC	Met à 1 la retenue CF
	CLC	MET à 0 la retenue CF
	CMC	Complémente la retenue
	STD	Met à 1 la direction DF
	CLD	Met à 0 la direction DF
	STI	Met à 1 l'autorisation d'interruption
	CLI	Met à 0 l'autorisation d'interruption
Synchronisation	HLT	Halte jusqu'à interruption ou RESET
	WAIT	Attente jusqu'à broche TEST passe à 0
	ESC	Pour un coprocesseur
	LOCK	Verrouillage des bus pendant la prochaine instructions
Sans opération	NOP	Pas d'opération

VII-1 ) Indicateurs :

VII-1-1/ STD :

Met CF à 1 ; les registres d'indexation SI et/ou DI sont alors automatiquement décrémenter par les instructions de chaîne de caractère.

VII-1-2 ) STI :

Met IF à 1, permettant ainsi au CPU de reconnaître des demandes d'interruption masquables apparaissant sur la ligne d'entrée INTR.

VII-2 ) Synchronisation :

VII-2-1 ) HALT :

Maintient le processeur dans un état d'attente d'un RESET ou d'une interruption externe non masquable ou masquable (avec IF=1).

VII-2-2 ) WAIT :

Met le CPU en état d'attente tant que sa ligne de TEST n'est pas active. En effet toutes les cinq périodes d'horloge le CPU vérifie est ce que cette entrée est active ou non, si elle est active le processus exécute l'instruction suivante à WAIT.

VII-2-3 ) ESC :

L'instruction Escape fournit un mécanisme par lequel des coprocesseurs peuvent recevoir leurs instructions à partir de la suite d'instructions du 8086.

VII-2-4 ) LOCK :

Elle utilise dans les systèmes Multiprocesseur en effet elle permet le verrouillage du bus vis-à-vis des autres processeurs.

VII-3 Sans opération :

VII-3-1 ) NOP (No operation) :

Le CPU ne fait rien on peut s'en servir pour créer des temporisations.

Plus d'informations :

[http://www.technologuepro.com/microprocesseur/chap4\\_microprocesseur.htm](http://www.technologuepro.com/microprocesseur/chap4_microprocesseur.htm)



## Complément de cours 3 : Interruptions système

### Interruptions DOS

*.Lecture d'un caractère (unique) au clavier avec écho* (avec écho signifie que le caractère tapé au clavier apparaît sur l'écran après la frappe)

Pas de nécessité de validation avec la touche "Entrée" : le caractère est acquis dès sa frappe au clavier.

Lecture d'un caractère frappé au clavier; après appel de l'interruption, le code ASCII du caractère est stocké dans le registre AL.

Paramètre d'entrée : AH = 1

Appel d'interruption : INT 21H

Paramètre de sortie : AL ; (AL) = code ASCII du caractère

### Interruptions BIOS

*.Affichage d'un caractère à l'écran*

Paramètres d'entrée : AH = 0EH

AL = Code ASCII du caractère à afficher à l'écran

Appel d'interruption : INT 10H

Paramètres de sortie : aucun

## Complément de cours 4 : Codage ASCII

### Codage ASCII des caractères

Le code ASCII du caractère '0' est 30h

Le code ASCII du caractère '1' est 31h

Le code ASCII du caractère '2' est 32h

...

Le code ASCII du caractère '9' est 39h

d'où l'algorithme de conversion *chiffre* → *code ascii* : **code ascii = chiffre + 30h.**

**Exercice 1 : Programmation ASSEMBLEUR (ASM)**

a) Installer le simulateur 8086 : Assembler with Microprocessor Simulator 8086 (fichier **emu8086v408r.exe**) téléchargeable à l'adresse :

[http://pcwin.com/Software\\_Development/Debugging/Assembler\\_with\\_Microprocessor\\_Simulator\\_8086/index.htm](http://pcwin.com/Software_Development/Debugging/Assembler_with_Microprocessor_Simulator_8086/index.htm) (rubrique *download*)

b) A l'aide de l'émulateur 8086 (menu *emulate*) et des compléments de cours, réaliser et vérifier le fonctionnement des programmes suivants : (il est conseillé d'utiliser les registres généraux AX, BX, CX, DX ainsi que la pile)

**Calcul de la somme S des N premiers entiers**

1- Sans Entrées/Sorties :  $0 < N < 100$  (N entier)

**1a-** Par itération :  $S = \text{somme de } i \text{ (de } i = 1 \text{ à } N)$  : chargement de N en dur et visualisation du résultat via l'émulateur

**1b-** Par formule de Gauss :  $S = N(N+1)/2$  : chargement de N en dur et visualisation du résultat via l'émulateur

*Test :  $N = 99; S = 4950 = 1356h$*

2- Avec Entrées/Sorties :  $0 < N < 4$  (N entier)

**2a-** Par itération :  $S = \text{somme de } i \text{ (de } i = 1 \text{ à } N)$  : lecture de N au clavier et affichage du résultat à l'écran (utilisations des interruptions système)

*Conseil : se placer en clavier Anglais pour l'acquisition de N au clavier*

**2b-** Par formule de calcul :  $S = N(N+1)/2$  : lecture de N au clavier et affichage du résultat à l'écran (utilisations des interruptions système)

*Conseil : se placer en clavier Anglais pour l'acquisition de N au clavier*

*Test :  $N = 3; S = 6 = 0006h$*

3- *Facultatif* : Avec Entrées/Sorties :  $0 < N < 10$  (N entier)

**3a-** Par itération :  $S = \text{somme de } i \text{ (de } i = 1 \text{ à } N)$  : lecture de N au clavier et affichage du résultat à l'écran (utilisations des interruptions système)

*Conseil : se placer en clavier Anglais pour l'acquisition de N au clavier*

*Test :  $N = 9; S = 45 = 002Dh$*

**Calcul de la somme S2 des carrés des N premiers entiers**

4- Calcul de la somme S2 des N premiers carrés des entiers

Sans Entrées/Sorties : N (entier) :  $0 < N < 9$  :

**4a-** Par itération :  $S2 = \text{somme de } (i*i) \text{ (de } i = 1 \text{ à } N)$  : chargement de N en dur et visualisation du résultat via l'émulateur

**4b-** Par formule :  $S2 = N(N+1)(2N+1)/6$  : chargement de N en dur et visualisation du résultat via l'émulateur

*Test1 :  $N = 3; S2 = 14 = 000Eh$*

*Test 2 :  $N = 8; S2 = 204 = 00CCh$*