on ne peut pas add/sub/cmp des mémoires directement entre eux , il faut passer par les registres !!!!

Petit résumé de l'assembleur 8086

Important : les explications peuvent être inexactes et mêmes fausses, je ne suis responsable de rien, si vous trouver des fautes, ou si vous avez d'autres choses à ajouter veuillez me contacter s'il vous plait.

Definition:

- Un Immédiate : une constante exemple mov AL,5 , 5 est un immédiate
- Une mémoire : une adresse définit par une étiquette : variable_mémoire db 5
- Les registres généraux et leurs fonctions :
  - AX Accumulateur
  - BX Base
  - CX Compteur
  - DX Données
- Les registres de segment : Traduction : Fonction
  - CS : Code Segment : Contient les instructions d'un programme
  - DS Data Segment : Contient les définitions des variables, tableaux…
  - SS : Stack Segment : Contient la définition de la pile du programme (stack = pile)
  - ES : Extra Segment : ..
- Les registres d'offset :
  - IP : Instruction Pointer
  - SP : Stack Pointer
  - SI : Source Index
  - DI : Data Index
  - BP : Base Pointer

Syntaxe :

- Les commentaires sont précédés du signe ';'
- Les apostrophes et les guillemets sont équivalents. Il est cependant plus commode de réserver l'usage des guillemets aux chaînes de plusieurs caractères et celui des apostrophes aux caractères isolés.
- L'extension des fichiers d'assembleur est .asm
- La compilation se fait en tapant "TASM /m9 MONPROG" si votre source s'appelle 'MONPROG.ASM'.
- EMU8086 est INSENSIBLE à la casse

LOOP A:

*Décrémente CX (qui doit être initialisé avec une valeur >0)

*vérifie si CX != 0:

* si Oui, Exécute les instructions de la boucle (A)

* Sinon continuer

P.S : l'instruction LOOP doit être appelé après la boucle « a »

Terminaison normale du programme :

l'oublie des instructions ci dessous à la fin du programme cause l'erreur « emulator is halted »

mov ah , 4ch

int 21h

-Les principales instructions du Langage machine :

- MOV Destination, Source : Copie le contenu de Source dans Destination.
- Mouvements autorisés :
  - MOV Registre général, Registre quelconque
  - MOV Mémoire, Registre quelconque
  - MOV Registre général, Mémoire
  - MOV Registre général, Constante
  - MOV Mémoire, Constante
  - MOV Registre de segment, Registre général
- Rmq : Source et Destination doivent avoir la même taille On ne peut charger dans un registre de segment (comme DS) que le contenu d'un registre général (SI, DI et BP sont considérés ici comme des registres généraux), les registres permises sont donc AX,BX,CX,DX,SI,DI,BP
- XCHG Destination, Source : Échange les contenus de Source et de Destination.
- Mouvements autorisés :
  - XCHG Registre général, Registre général
  - XCHG Registre général, Mémoire
  - XCHG Mémoire, Registre général
- JMP MonLabel : saute à l'étiquette appelé MonLabel
- CMP A , B : compare la valeur de Destination avec celle de Source , en réalité cmp fait la soustraction de A et B (A-B) :
  - Cet opérateur sert à comparer deux nombres : A et B. C'est le registre des indicateurs (les flags) qui contient les résultats de la comparaison. Ni A ni B ne sont modifiés.
  - Indicateur (flags) affectés : AF, CF, OF, PF, SF, ZF
  - Comparaison possible :
    - REG, memory
    - memory, REG
    - REG, REG
    - memory, immediate
    - REG, immediate

- Les instructions de « saut » conditionnel :
  - JZ MonLabel: saute à MonLabel si ZF=1
    - JE équivalent à JZ
  - JA MonLabel : saute si dans la comparaison précédente le premier opérande est strictement supérieur au deuxième (les deux opérandes doivent être non-signé, s'ils sont signés, on utilisé JG)
  - JB MonLabel : saute si dans la comparaison précédente, le premier opérande est strictement inférieur au deuxième ((les deux opérandes doivent être non-signé, s'ils sont signés, on utilisé JL)
  - JNE MonLabel : saute si le premier opérande est différent (inférieur ou supérieur) du deuxième , JNZ fait la même chose
  - JAE,JBE,JGE et JLE sont pareil que JA,JB,JG et JL sauf qu'ils saute même si le résultat est égale
  - **Mnémonique** :
    - on lit toujours « «JE A,B » comme suit « Jump if A Equals B » idem pour le reste des instructions de saut
    - JZ = Jump if Zéro (i.e le résultat de la soustraction fait par cmp est nulle)
    - JG = Jump if Greater
    - JA = Jump if Above
    - JL = Jump if Lower
    - JB = Jump if Below
    - JNE = Jump if Not Equal
    - JAE = Jump if Above or Equal
    - …

Les principales instructions du Langage machine :

- INC Dest : Incrémenter Dest d'une unité
- DEC Dest : Décrémenter Dest d'une unité
- ADD Destination , Source : Ajoute Source à Destination (résultat stocké dans la destination)
- SUB Destination , Source : soustrait Source à Destination (idem)
- MUL Source : (Source doit être un octet (byte)) Multiplie la source par la valeur contenu dans le registre AL et place le résultat dans AX
- DIV Source : (Source doit être un octet (byte)) : AX est divisé par la source , le quotient est mis dans AL et le reste dans AH
- ADD,SUB,CMP NE PEUVENT PAS FAIRE LEURS OPÉRATIONS SUR DEUX MEMOIRE (MEMOIRE = VARIABLES DEFINIT)

mov SI,OFFSET nom_de_tab:

*syntaxe : mov SI , OFFSET nom_de_votre_table_déja_créée

*utilité : permet de manipuler le tableau à travers [SI] au lieu de tab[SI]

(N'oubliez pas d'incrémenter SI pour passer à la case mémoire suivant)

Conversion d'un chiffre (0-9) de l'ASCII au décimal :

*SUB nombre_a_convertir,'0' (ou 48)

P.S : ADD nbre_a_convertir,'0' fera le contraire

Lecture Ecriture à travers l'interruption int 21h :

- Une interruption est un appel d'une routine spécial (une subséquence d'instruction effectuant un rôle spécifique).
- int 21h permet de lire ce qu'on tape au clavier / afficher à l'écran
- L'interruption 21h a plusieurs fonctions (notamment lecture du clavier/écriture à l'écran), il faut préciser la fonction voulu en modifiant la valeur du demi-registre AH :
    - Pour lire : il faut mettre ah à 1 (mov ah,1), la valeur lue sera écrite dans le demi registre AL
    - Pour écrire : il faut mettre ah à 2, la valeur à afficher doit être mis dans le demi registre DL
        - Mnémotechnique : A et 1 vient avant D et 2 (dans l'ordre alphabétique/numérique)
- **IMPORTANT :** la lecture et l'écriture s'effectuent en ASCII, par exemple :

Si j'écris dans DL 5 et je l'affiche, j'aurais le caractère qui a pour code ASCII 5 et non pas 5, pour afficher le chiffre 5 il faut lui ajouté '0' (= 48) ou écrire directement '5'

En cas de lecture si je tape au clavier 5, la valeur stocker dans AL sera le code ASCII de 5 (donc '5' ou 5+'0' ou 5 + 48)

- L'instruction int 21h modifie la valeur de AL (il faut faire attention à ça … Regardez la dernière partie du cours pour résoudre le problème)

Multiplier un nombre par 10 En binaire :

*multiplier un nombre « a » par 10 est équivalent à additionner b=Shift,a,1 (« a » décaler d'un bit à gauche) et c=Shift,a,3 (« c » décaler de 3 bit à gauche , d'habitude c'est c shift,a,2 car on a déjà décaler « a » d'un bit dans l'instruction précédente)

P.S : cela est valable quand les 3 derniers bits du poids fort est nul car sur 8 bits par exemple , si on a par exemple 1000 0000 et on fait un décalage de 1 bit à gauche on aura 0000 0000 ...

0001 0000

0000 1010

Definition d'un tab :

- nom db 'c','o','n','t','e','n','u'

  ou

- nom db "contenu"

  ou

- nom db nbre_case dup(valeur(s) à dupliquer) : permet de remplir le tableau

  ex1 : nom db 6 dup(0) <=> nom db 0,0,0,0,0,0

ex2 : nom db 6 dup(1,2) <=> nom db 1,2,1,2,1,2

- Pour déclarer un tableau vide d'une taille nbre_case
    - nom_tab DB nbre_case dup(?)

l'utilisation de mov BYTE PTR (resp WORD PTR ...) :

*l'ajout de BYTE PTR à l'instruction « mov » ex : <mov BYTE PTR destination, source> est requis quand la source n'a pas une taille précise, prenons l'exemple de la copie d'un IMMEDIATE (une constante) dans l'emplacement mémoire pointé par un registre ([SI] par exemple) : si on écrit :

mov [SI], 1 l'IMMEDIATE 1 peut être écrit sur 1 à 8 bits : 1 ou 01 ou 001 ou 0001 ...

L'assembleur ne saurait pas la taille de ce qu'il doit copier, et donc il renverrait une erreur.

En ajoutant BYTE PTR : mov BYTE PTR [SI], 1 , on FORCE la source à être traiter comme un BYTE (1 byte(eng)=1 octet (fr)=8bits)

mieux expliqué ici : http://www.c-jump.com/CIS77/ASM/Instructions/I77_0250_ptr_pointer.htm

Assume :

ASSUME segmentRegister:segmentName

- Specifie le registre de segment qui sera utiliser pour calculer les adresses effectives des labels et des variables définies dans le segment segmentName
- segmentRegister argument peut être soit CS, DS, ES, ou SS.

Le squelette d'un code assembleur (à apprendre :D)

```
assume CS:code Ds:data
Data segment
;declaration des variables
Data ends
main:
Code segment
mov ax,data
mov ds,ax
;le reste du code..
Code ends
end main
```

Les instructions de gestion de la pile

- Déclaration de la pile :

```
Pile SEGMENT STACK
 dw taille dup (?)
Pile ENDS
```

  - vous pouvez remplacer taille par la valeur que vous voulez (entre 1 et 256 octets)
  - vous pouvez laisser le « ? » ou le remplacer par des 0 ou la valeur que vous voulez (c'est la valeur initiale des octets de la pile)
- PUSH Source : Empile le mot (deux octets, 16 bits) Source (l'ajoute à la tête de la pile)

- POP Destination : Dépile le dernier mot (deux octets, 16 bits) ajouté à la pile et le met dans Destination
- PUSH/POP NE MANIPULE QUE DES MEMOIRES/REGISTRES 16BIT

L'adressage indexé et/ou basé :

- il est possible d'utiliser des registres de *base* ou d'*index* pour adresser un octet.
- On appelle « *base* » les registres BX et BP et « *index* » les registres SI et DI. La différence est que la base est censée être fixe tandis que l'index varie automatiquement lorsqu'on utilise certaines opérations, telles que MOVSB.

Les procédures:

- la procédure est une simple fonction sans retour , définit avec une étiquette (label) suivit des instructions et finit par un « ret » , par convention, on ajoute « proc near » après le nom de la procédure , pour améliorer la lisibilité et réduire la procédure en offset (donc la procédure ne pourra être appelé qu'à l'intérieur du segment actuel) et on ajoute aussi un endp (end procédure) comme suit : nom_étiquette endp
- la procédure peut être appelé par seulement les procédure du même segment (si on ajoute « near ») avec l'instruction CALL nom_proc

sources :
documentation emu8086
http://benoit-m.developpez.com/assembleur/tutoriel/

```
nom_procédure proc near
;instructions
ret
nom_procédure endp
```

Remarques :

- on peut pas « mov » une mémoire à une autre il faut passer à travers un register :
  - A db 'a'
  - B db ?
  - Mov B ,A ; impossible
  - Il faut faire (exemple):
  - Mov AH,B
  - MOV A,AH

- Pour remettre un registre à zéro, il est préférable de faire “XOR AX, AX” que “MOV AX, 0”. En effet, le résultat est le même, mais la taille et surtout la vitesse d'exécution de l'instruction sont très largement optimisées.
- INC modifie le ZF, elle le met à 1

Problèmes communs :

- **Problème 1 : Quand j’essaie d’afficher ce que j’affiche un retour à la ligne (par exemple) et j’essaie d’afficher le contenu de AL, je n’ai pas le résultat attendu**
    - **Solution** : Vous devez mettre la valeur de AL dans un autre registre ou mémoire avant d’exécuter l’interruption 21h, sinon elle sera perdue
    - **Explication** : L’instruction int 21h modifie la valeur de AL (il faut faire attention à ça …) si vous essayez par exemple de lire une valeur écrite en clavier (elle sera stocké dans AL), d’afficher (par exemple) un retour à la ligne (c’est 10 en ASCII) et après vous essayer d’afficher la valeur déjà lu vous n’aurez plus la valeur lu au clavier, mais vous aurez la valeur que vous avez affiché (on peut dire que int 21h effectue l’instruction mov AL,DL …)
- **LES FAUTES QUE JE COMMET SOUVENT** :
    - **Oublier de préciser les frontières de chaque segment**
    - **Oublier de mettre DATA à DS (à travers AX)**
    - **Oublier le main et le end main**
    - **MOV , SUB , ADD … ne peuvent pas opérer sur deux registres, il faut avoir au moins un registre.**
    - **tableau[INDEX] INDEX peut être seulement un registre de 16 bits**
    - **quand on met l’offset d’un tableau dans un registre (SI par exemple , comme suit : mov SI,OFFSET tab_name) , on accés à la case actuelle par [SI] et non pas avec SI, et on incrémente SI pour passer à la case suivant du tab_name**
    - **On peut mettre l’offset d’un tableau dans n’importe quelle registre mais on ne peut utiliser que les registres comme index de tableau (exemple tab[registre] ou [registre])**
    - **pour mettre un caractère dans un adresse pointé par BX il faut ajouter BYTE PTR pour préciser que le caractère est sur 8 bits (il peut bien être sur 16 bits , dans ce cas on utilise WORD au lieu de BYTE)**