

# Syntaxe, directives assembleur

Ensimag, filière  
apprentissage, 1A  
2011



# Exemples

---

## □ Instructions:

```
iter:    cmpw $0,%ax    # compare word
         je    fin      # jump if equal
         shrw $1,%ax    # shift right word
         jnc   suite    # jump if no carry
         add   %dx,%ax   # add
suite:   shlw $1,%dx    # shift left word
         jmp   iter     # jump inconditionnel
fin:
```

## ■ Données :

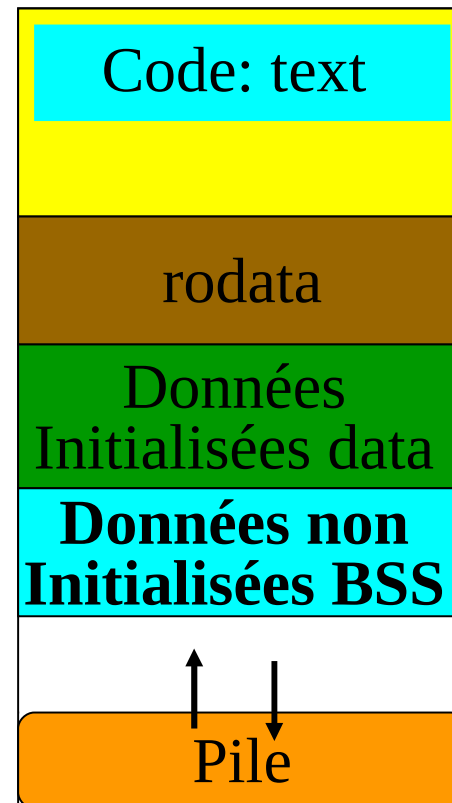
```
toto:    .byte 0xff
lulu:    .int   $5000, suite
```

# Modèle mémoire (assembleur Gnu)

## Les directives .text, .data, .section

```
        .section .data
un:      .int 1
....    ...

        .section .bss
        .lcomm tab,10
tab1:    skip 10
---
        .text
main:    pushl %ebp
```





# Programme source

---

- Ensemble de sections
  - .data (.rodata, .bss) pour les données
  - .text pour les instructions
- Chaque section est une suite de lignes
- Pour les instructions
  - [etiquette:] code\_op opérandes
- Pour les données
  - [etiquette:] def\_de\_donnée suite de valeurs
- des commentaires
- des directives d'assemblage



# Représentation symbolique des instructions

---

- Code de l'opération
  - La dernière lettre correspond à la longueur des opérandes
  - Exemple : shr, subl, movb
- Représentation symbolique des opérandes
  - Registre. Ex: %eax
  - Adresse en mémoire, dénotée par un mode d'adressage Ex: 4(%ecx)
  - Valeur immédiate Ex: \$0x45ab
- Attention : les types d'opérandes valides dépendent des instructions



# Les commentaires

---

- Définition : il s'agit de textes non interprétés par l'assembleur et qui sont fournis par le programmeur pour augmenter la lisibilité de son programme.
- Comme en C (avec des fichiers \*.S, S majuscule) :
  - soit sur une ligne tout ce qui suit **//** jusqu'à la fin de ligne
  - soit ce qui est entre les deux couples de caractères **/\*** et **\*/**



# Les étiquettes

---

- Une étiquette (identificateur suivi de « : ») sert à désigner l'adresse d'un emplacement de mémoire
- On peut l'utiliser dans un champ opérande

```
toto:      movw %eax,lulu
```



# Étiquettes

---

- Déclaration d'étiquette :  
toto: <quelquechose>
  - Ne génère pas de code
  - Définie toto comme l'adresse de <quelquechose>
- Utilisation d'étiquette :  
movl toto, %eax
  - L'étiquette est remplacée par sa valeur





# Directives d'assemblage

---

- Directives d'assemblage : commandes fournies par l'assembleurs qui ne correspondent à aucune instruction du processeur.
- Elles permettent entre autres :
  - La définition de données.
  - La définition de constantes ou de symboles.
  - Les sections (.text et .data)




# Définition de données

---

- Définition de données initialisées
  - [étiquette] .<DNAME> val1,val2,..., valn  
<DNAME> = byte | hword | long | quad | asciz
    - xi: .long 0xaabbccdd, xi, -4500
    - xb: .byte 0x3f, 35, 'c'
    - message: .asciz "Hello World"
- Définition de données « non initialisées »
  - .lcomm nom, taille
  - [étiquette] .skip taille (,valeur)

# Directives assembleur de définition de données



---

```
$ gcc exemple.S -m32 -Wa,-a
1          .data
2 0000 DDCCBBAA xi:      .long 0xaabbccdd, xi, -4500
2          00000000
2          6CEEFFFF
3 000c 3F2363  xb:      .byte 0x3f, 35, 'c'
4 000f 48656C6C message: .string "Hello World"
4          6F20576F
4          726C6400
5          .lcomm tab,10
6          .text
7          .globl main
8 0000 55      main:    pushl %ebp
```



# Définition de constantes

---

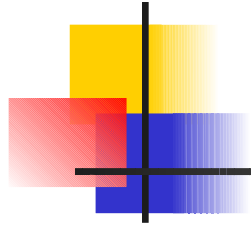
- Comme le `#define` du langage C
    - Symbole = expression  
associe de façon définitive la valeur d'*expression* au  
symbole défini par le champ *symbole*.
- X = 1024  
movl \$X, %eax // Strictement équivalent à « movl \$1024, %eax »
- Remplacement syntaxique : on pourrait le faire avec  
« rechercher/remplacer » dans un éditeur de texte !
  - (syntaxes alternatives : « .set x, y » ou « .equ x, y » ou encore  
« #define X Y »)



# Exportation de symbole

---

- Motivation : pouvoir définir (resp. utiliser) dans un module d'assemblage du code et des données utilisables (resp. définis) dans un autre module d'assemblage produit par un compilateur ou par un programmeur.
- Directive `.globl` (ou `.global`)
  - `.globl <étiquette>,...` les étiquettes du champ opérande définies dans le module courant sont rendues visibles à l'extérieur de ce module.
  - Toute étiquette référencée dans le module courant sans y être définie est considérée comme externe, donc définie dans un autre module d'assemblage.
  - Exemple:  
**`.globl main`** // chaque programme comporte un  
// « main » appelé par le système



# Mécanismes d'adressage



# Introduction (1)

---

- Exemples :
  - `movl $42, %eax`
  - `movl toto, %ebx`
  - ~~`movl 2*(toto + %ebx), %eax`~~
- Les opérandes des instructions ne peuvent pas être des expressions quelconques.
- Expressions autorisées comme opérandes = mode d'adressages



# Modes d'adressages principaux du pentium

---

- Cf. EnsiWiki :

- [http://ensiwiki.ensimag.fr/index.php/LdB\\_Modes\\_d%27adresses](http://ensiwiki.ensimag.fr/index.php/LdB_Modes_d%27adresses)