

Langage d'assemblage : Syntaxe, directives Ensimag 1A Apprentissage

Matthieu Moy

Matthieu.Moy@imag.fr

mai 2012



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 /* Un octet, de valeur 0xFF */
lulu: .int $5000, suite /* Deux entiers sur 32 bits,
                        de valeur 5000, puis l'adresse
                        de l'etiquette suite */
```



Programme source

- Ensemble de sections
- `data` (`.rodata`, `.bss`) pour les données
- `text` (`.text`) pour les instructions
- Chaque section est une suite de lignes :
 - ▶ Pour les instructions :
[etiquette:] *code-op opérandes*
Exemple : `addl $42, %eax`
 - ▶ Pour les données :
[etiquette:] *def-de-donnée suite de valeurs*
Exemple : `x: .int 42`
- des commentaires
- des directives d'assemblage



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 `*/`
- Alternative : `#` jusqu'à la fin de la ligne



Sommaire

- 1 Syntaxe du langage d'assemblage
- 2 Directives d'assemblage
- 3 Mécanismes d'adressage



Modèle mémoire (assembleur gnu)

Les directives : `.text`, `.data`, `.section`

```
un: .section .data
    .int 1
    ...
    .section .bss
    .lcomm tab,10
tab: skip 10
    ...
    .text
main: pushl %ebp
```

Code : text
rodata
Données initialisées : data
Données non initialisées : BSS
Pile



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`
- ⚠ les types d'opérandes valides dépendent des instructions



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



Les 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 (i.e. l'adresse de l'endroit où elle est définie)



Définition de données initialisées

- [*etiquette*]: *.type-de-donnée val1, val2, val, ...*
- .type-de-donnée* = byte | hword | long | quad | asciz
- Exemple :

```
.data
xi: .long 0xaabbccdd, xi, -4500
xb: .byte 0xf, 35, 'c'
message: .asciz "Hello World"
```



Exemple

```
$ gcc exemple.S -m32 -Wa,-a
2 0000 DDCCBBAA xi: .long 0xaabbccdd, xi, -4500
2      00000000
2      6CEFFFFF
3 000c 3F2363 xb: .byte 0xf, 35, 'c'
4 000f 48656C6C message: .asciz "Hello World"
4      6F20576F
4      726C6400
5
6      .lcomm tab1, 10
7      .lcomm tab2, 10
8
9      .text
10     .globl main
11     main:
12     pushl %ebp
13
14     DEFINED SYMBOLS
15     example.S:2      .data:0000000000000000 xi
16     example.S:3      .data:0000000000000000c xb
17     example.S:4      .data:0000000000000000f message
18     example.S:5      .bss:0000000000000000 tab1
19     example.S:6      .bss:00000000000000010 tab2
20     example.S:9      .text:0000000000000000 main
```



Exportation de symbole

- Motivation : pouvoir définir dans un module du code et des données utilisables dans un autre module (produit par un compilateur ou par un programmeur).
- Directive .globl (ou .global)
 - .globl *étiquette1, ...*
 - ⇒ rendre les étiquettes *étiquette1, ...* (définies dans le module courant) visible depuis l'extérieur du module
 - Toute étiquette référencée dans le module courant sans y être définie est considérée comme externe, donc définie dans un autre module d'assemblage (pas d'erreur à l'assemblage).
 - L'édition de liens (plus tard) fait le lien entre symboles indéfinis et symboles globaux des autres modules.
 - Exemple :

```
.globl main // chaque programme comporte un
// 'main' appelle par le systeme
```



Directives d'assemblage

- Directives d'assemblage : commandes fournies par l'assembleur 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 non-initialisées

- Section .bss dédié aux données non-initialisées
 - Ne prend pas de place dans le fichier binaire
 - Alloué (et initialisé à 0) au chargement
- Directives d'assemblages :
 - .lcomm *nom, taille* (pratique pour les tableaux)
 - [*etiquette*]: .skip *taille* (à utiliser dans la section .bss)



Définition de constantes

- symbole* = *expression*
- Comme le #define du langage C
- associe de façon définitive la valeur d'*expression* au symbole défini par le champ *symbole*.
- Remplacement *syntactique* et *statique* : on pourrait le faire avec « rechercher/remplacer » dans un éditeur de texte !
- Syntaxes alternatives :
 - .set *symbole, expression*
 - .equ *symbole, expression*
 - #define *symbole expression*



Introduction aux modes d'adressages

- Exemples
 - movl \$42, %eax (\$42 = valeur immédiate, %eax = registre)
 - movl toto, %ebx (toto = valeur située à l'adresse toto)
 - ~~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 « LdB Modes d'adressages » :

http://ensiwiki.ensimag.fr/index.php/LdB_Modes_d%27adresses

