

Guide de référence rapide: l'assembleur Pentium

Jean Goubault-Larrecq

11 octobre 2002

Les registres : `%eax %ebx %ecx %edx %esi %edi %ebp %esp`.

Ils contiennent tous un entier de 32 bits (4 octets), qui peut aussi être vu comme une adresse. Le registre `%esp` est spécial, et pointe sur le sommet de pile ; il est modifié par les instructions `pushl`, `popl`, `call`, `ret` notamment.

Il y a aussi d'autres registres que l'on ne peut pas manipuler directement. (L'instruction `info registers` sous `gdb` ou `ddd` vous les montrera.) Le plus important est `eip`, le *compteur de programme* : il contient en permanence l'adresse de la prochaine instruction à exécuter.

- `addl <source>, <dest>` $\langle \text{dest} \rangle = \langle \text{dest} \rangle + \langle \text{source} \rangle$ (addition)

Ex : `addl $1, %eax` ajoute 1 au registre `%eax`.

Ex : `addl $4, %esp` dépile un élément de 4 octets de la pile.

Ex : `addl %eax, (%ebx, %edi, 4)` ajoute le contenu de `%eax` à la case mémoire à l'adresse `%ebx + 4 * %edi`.

(Imaginez que `%ebx` est l'adresse de début d'un tableau `a`, `%edi` est un index `i`, ceci stocke `%eax` dans `a[i]`.)

- `andl <source>, <dest>` $\langle \text{dest} \rangle = \langle \text{dest} \rangle \& \langle \text{source} \rangle$ (et bit à bit)

- `call <dest>` appel de procédure à l'adresse `<dest>`

Équivalent à `pushl $a`, où `a` est l'adresse juste après l'instruction `call` (l'adresse *de retour*), suivi de `jmp <dest>`.

Ex : `call printf` appelle la fonction `printf`.

Ex : `call *%eax` (appel indirect) appelle la fonction dont l'adresse est dans le registre `%eax`. Noter qu'il y a une irrégularité dans la syntaxe, on écrit `call *%eax` et non `call (%eax)`.

- `cltd` conversion 32 bits -> 64 bits

Convertit le nombre 32 bits dans `%eax` en un nombre sur 64 bits stocké à cheval entre `%edx` et `%eax`.

Note : `%eax` n'est pas modifié ; `%edx` est mis à 0 si `%eax` est positif ou nul, à -1 sinon.

À utiliser notamment avant l'instruction `idivl`.

- `cmp <source>, <dest>` comparaison

Compare les valeurs de `<source>` et `<dest>`. Utile juste avant un saut conditionnel (`je`, `jge`, etc.). À noter que la comparaison est faite dans le sens inverse de celui qu'on attendrait. Par exemple, `cmp <source>, <dest>` suivi d'un `jge` ("jump if greater than or equal to"), va effectuer le saut si $\langle \text{dest} \rangle \geq \langle \text{source} \rangle$: on compare `<dest>` à `<source>`, et non le contraire.

- `idivl <dest>` division entière et reste

Divise le nombre 64 bits stocké en `%edx` et `%eax` (cf. `cltd`) par le nombre 32 bits `<dest>`. Retourne le quotient en `%eax`, le reste en `%edx`.

- `imull <source>, <dest>` multiplie `<dest>` par `<source>`, résultat dans `<dest>`

- `jmp <dest>` saut inconditionnel : `eip = <dest>`

- `je <dest>` saut conditionnel

Saute à l'adresse `<dest>` si la comparaison précédente (cf. `cmp`) a conclu que $\langle \text{dest} \rangle = \langle \text{source} \rangle$, continue avec le flot normal du programme sinon.

- `jg <dest>` saut conditionnel

Saute à l'adresse `<dest>` si la comparaison précédente (cf. `cmp`) a conclu que $\langle \text{dest} \rangle > \langle \text{source} \rangle$, continue avec le flot normal

du programme sinon.

- **jge** $\langle \text{dest} \rangle$ saut conditionnel
Saute à l'adresse $\langle \text{dest} \rangle$ si la comparaison précédente (cf. **cmp**) a conclu que $\langle \text{dest} \rangle \geq \langle \text{source} \rangle$, continue avec le flot normal du programme sinon.
- **jle** $\langle \text{dest} \rangle$ saut conditionnel
Saute à l'adresse $\langle \text{dest} \rangle$ si la comparaison précédente (cf. **cmp**) a conclu que $\langle \text{dest} \rangle \leq \langle \text{source} \rangle$, continue avec le flot normal du programme sinon.
- **jle** $\langle \text{dest} \rangle$ saut conditionnel
Saute à l'adresse $\langle \text{dest} \rangle$ si la comparaison précédente (cf. **cmp**) a conclu que $\langle \text{dest} \rangle \leq \langle \text{source} \rangle$, continue avec le flot normal du programme sinon.
- **leal** $\langle \text{source} \rangle, \langle \text{dest} \rangle$ chargement d'adresse effective
Au lieu de charger le contenu de $\langle \text{source} \rangle$ dans $\langle \text{dest} \rangle$, charge l'adresse de $\langle \text{source} \rangle$.
Équivalent C : $\langle \text{dest} \rangle = \&\langle \text{source} \rangle$.
- **movl** $\langle \text{source} \rangle, \langle \text{dest} \rangle$ transfert
Met le contenu de $\langle \text{source} \rangle$ dans $\langle \text{dest} \rangle$. Équivalent C : $\langle \text{dest} \rangle = \langle \text{source} \rangle$.
Ex : **movl %esp, %ebp** sauvegarde le pointeur de pile **%esp** dans le registre **%ebp**.
Ex : **movl %eax, 12(%ebp)** stocke le contenu de **%eax** dans les quatre octets commençant à **%ebp + 12**.
Ex : **movl (%ebx, %edi, 4), %eax** lit le contenu de la case mémoire à l'adresse **%ebx + 4 * %edi**, et le met dans **%eax**. (Imaginez que **%ebx** est l'adresse de début d'un tableau **a**, **%edi** est un index **i**, ceci stocke **a[i]** dans **%eax**.)
- **negl** $\langle \text{dest} \rangle$ $\langle \text{dest} \rangle = -\langle \text{dest} \rangle$ (opposé)
- **notl** $\langle \text{dest} \rangle$ $\langle \text{dest} \rangle = \sim \langle \text{dest} \rangle$ (non bit à bit)
- **orl** $\langle \text{source} \rangle, \langle \text{dest} \rangle$ $\langle \text{dest} \rangle = \langle \text{dest} \rangle \mid \langle \text{source} \rangle$ (ou bit à bit)
- **popl** $\langle \text{dest} \rangle$ dépilement
Dépile un entier 32 bits de la pile et le stocke en $\langle \text{dest} \rangle$.
Équivalent à **movl (%esp), %eax** suivi de **addl \$4, %esp**.
Ex : **popl %ebp** récupère une ancienne valeur de **%ebp** sauvegardée sur la pile, typiquement, par **pushl**.
- **pushl** $\langle \text{source} \rangle$ empilement
Empile l'entier 32 bits $\langle \text{source} \rangle$ au sommet de la pile.
Équivalent à **movl %eax, -4(%esp)** suivi de **subl \$4, %esp**.
Ex : **pushl %ebp** sauvegarde la valeur de **%ebp**, qui sera rechargée plus tard par **popl**.
Ex : **pushl %eax** permet aussi d'empiler les arguments successifs d'une fonction. (Note : pour appeler une fonction C comme **printf** par exemple, il faut empiler les arguments en commençant par celui de droite.)
- **ret** retour de procédure
Dépile une adresse de retour a , et s'y branche. Lorsque la pile est remise dans l'état à l'entrée d'une procédure f , ceci a pour effet de retourner de f et de continuer l'exécution de la procédure appelante.
Équivalent à **popl %eip**... si cette instruction existait (il n'y a pas de mode d'adressage permettant de manipuler **%eip** directement).
- **subl** $\langle \text{source} \rangle, \langle \text{dest} \rangle$ $\langle \text{dest} \rangle = \langle \text{dest} \rangle - \langle \text{source} \rangle$ (soustraction)
Ex : **subl \$1, %eax** retire 1 du registre **%eax**.
Ex : **subl \$4, %esp** alloue de la place pour un nouvel élément de 4 octets dans la pile.
- **xorl** $\langle \text{source} \rangle, \langle \text{dest} \rangle$ $\langle \text{dest} \rangle = \langle \text{dest} \rangle \wedge \langle \text{source} \rangle$ (ou exclusif bit à bit)