CIRC2 : Assembleur, Numération, et Circuits

Licence informatique – semestre 4

Section 1

Adressages en mode indirect et indirect indexé

Les différents modes d'adressage

- mode d'adressage immédiat :
  `movb $10, %al`
- mode d'adressage direct :
  `movl nb, %eax`
- mode d'adressage registre
- mode d'adressage indirect :
  `movl (%ebx), %eax`
- mode d'adressage indirect basé sur le pointeur de registre
- mode d'adressage indirect indexé

Manipuler une liste

Si on veut représenter une liste :

```
1         .data
2 tab:    .int    345, 123, 278, 4548, 193452
3 nb:     .int    5
```

alors :

- `tab` contient l'adresse du premier élément de la liste
- `tab+4` contient l'adresse du deuxième élément de la liste
- `tab+2*4` contient l'adresse du troisième élément de la liste
- ...

Somme des éléments d'une liste

```
1 main:    movl    $tab, %eax
2          movl    $0, %esi
3          movl    $0, %ebx
4 bcle:    cmpl    %ecx, nb
5          je      fin
6          addl    (%eax), %ebx
7          addl    $4, %eax
8          incl    %esi
9          jmp     bcle
```

Deux nouveaux registres généraux pour manipuler des adresses

Deux registres généraux qui peuvent être utilisés dans toutes les opérations de transfert de valeur :

- `RSI Re-extended Source Index` registre utilisé comme pointeur vers une source
- `RDI Re-extended Destination Index` registre utilisé comme pointeur vers une destination

Les autres registres généraux peuvent aussi servir de pointeur vers une source ou une destination (cf. exemple précédent).

De nouveaux modes d'adressage indirect indexé (1)

```
movl 4(%esi), %eax
```

- calcule l'adresse `%rsi + 4`
- copie le contenu de cette adresse dans `%rax`
- on ne peut trouver qu'une constante ou une adresse devant les parenthèses (pas de registre)
- la valeur de `%rsi` n'est pas changée

De nouveaux modes d'adressage indirect indexé (2)

Forme générale :

```
Adresse_Ou_Decalage(%Base_Ou_Decalage,%Index,Fact_echelle)
```

qui calcule l'adresse

```
Adresse_Ou_Decalage + %Base_Ou_Decalage
        + %Index * Fact_echelle
```

- `Fact_echelle` ne peut prendre qu'une valeur parmi : 1, 2, 4 ou 8
- Chaque élément est optionnel

La somme des éléments d'une liste (2)

```
1 main:    movl    $tab, %eax
2          movl    $0, %esi
3          movl    $0, %ebx
4 bcle:    cmpl    %ecx, nb
5          je      fin
6          addl    (%eax,%esi,4), %ebx
7          incl    %ecx
8          jmp     bcle
```

Calculer la taille d'un tableau

```
1       .data
2 tab:      .int      345, 123, 278, 4548, 193452
3 nb:       .int      .-tab       # donne la taille du tableau
4 ch:       .asciz    "Bonjour !"
5 len:      .int      .-ch        # mais ici la taille en octet
6                                 # c'est la taille en caracter
```

Section 2

Les interruptions

Les interruptions

Une interruption du fonctionnement normal du processeur peut-être provoquée :

- par un composant physique (horloge, périphérique d'E/S, mémoire, ...)
- par le processeur lui-même

C'est un signal qui provoque l'interruption :

- division par zéro
- référence mémoire en dehors de l'espace autorisé
- dépassement de capacité
- fin d'une opération d'entrée-sortie
- défaillance matérielle ...

Les interruptions (2)

Le processeur peut provoquer lui-même une interruption. C'est l'instruction `int` suivi d'un numéro de service.
Le numéro de service `0x80` lance un appel à une fonction du noyau Linux :

```
1           int    $0x80
```

Dans les registres, avant l'appel à `int`, on transmet :

- la fonction à exécuter (représentée par un numéro)
- ses paramètres

Quelques fonctions standards

%eax	Fonction	%ebx	%ecx	%edx	sortie (dans %eax)
1	quitter `sys_exit`	code erreur (0 pas d'erreur)			
3	lecture `sys_read`	numéro flux entrée (0 clavier)	adresse ch. de car.	nb car. lus	nb car. lus
4	écriture `sys_write`	numéro flux sortie (1 écran)	adresse ch. de car.	nb car. écrits	nb car. écrits

Lorsque l'interruption est terminée, l'exécution du programme assembleur se poursuit normalement.

Exemple 1

```
1           .data
2  bonjour:
3           .ascii  "Bonjour ! "
4  nbcar:   .int    .-bonjour
5           .text
6           .globl  main
7  main:    # afficher la chaine de caracteres
8           movl    $4, %eax          # fonction sys_write
9           movl    $1, %ebx          # sur la sortie standard
10          movl    $bonjour, %ecx    # adresse de la chaine
11          movl    nbcar, %edx       # nombre de caracteres
12          int     $0x80             # interruption
```

Exemple 2

```
1           .data
2 nb:       .int    140
3 chaine:   .fill   140,1,' '
4 prompt:   .asciz  "Saisissez votre message (140 car. max) :"
5 len_mess:
6           .int    .-prompt
7
8           .text
9           .globl  main
10 main:    # afficher le prompt
11          movl    $4, %eax            # fonction sys_write
12          movl    $1, %ebx            # sur la sortie standard
13          movl    $prompt, %ecx       # adresse de la chaine
14          movl    len_mess, %edx      # nombre de caracteres
15          int     $0x80               # interruption
```

Exemple 2 - suite

```
1          # lire le message
2          movl    $3, %eax            # fonction sys_read
3          movl    $0, %ebx            # au clavier
4          movl    $chaine, %ecx       # adresse de la chaine
5          movl    nb, %edx            # nombre de caracteres
6          int     $0x80               # interruption
7
8          # repeter le message
9          movl    %eax, %edx          # nombre de caracteres
10         movl    $4, %eax            # fonction sys_write
11         movl    $1, %ebx            # sur la sortie standard
12         movl    $chaine, %ecx       # adresse de la chaine
13         int     $0x80               # interruption
14
15         # quitter le programme
16         movl    $1, %eax            # fonction sys_exit
17         movl    $0, %ebx            # sur la sortie standard
18         int     $0x80               # interruption
```