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, %aeax`
- 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, %ecx
3          movl    $0, %ebx
4 bcle:    cmpl    %ecx, nb
5          je      fin
6          addl    (%eax), %ebx
7          addl    $4, %eax
8          incl    %ecx
9          jmp     bcle
```

Registres pour manipuler des adresses

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

Section 2

Structures de données

Il existe le registre RIP qui est le registre pointeur d'adresses d'instructions). Il existe également deux autres registres RSI (Re-extended Source Index) et RDI (Re-extended Destination Index) qui sont utilisés comme pointeurs vers la source ou la destination dans les opérations de copie de zones mémoires. Les registres généraux peuvent aussi servir de pointeurs.
L'adressage peut être :

- immédiat, en désignant une constante (une valeur numérique)
- registre en désignant un registre
- direct en utilisant une étiquette

Il peut aussi être indirect :

- %rsi désigne le contenu du registre rsi,
- (%rsi) désigne le contenu de la case mémoire dont l'adresse est stockée dans rsi

Avec l'ensemble des éléments vus ici, on pourrait compiler et exécuter n'importe quel programme dans un langage évolué. . .
On peut avoir des modes d'adressage indirect plus évolué :

- 5(%rsi) ici, désigne le contenu de la case mémoire dont l'adresse est 5 + l'adresse stockée dans rsi. Ce qu'on trouve avant la parenthèse ne peut être qu'une constante.

- 5(%rsi, %rax) désigne le contenu de la case mémoire dont l'adresse est 5 + l'adresse stockée dans rsi + l'adresse stockée dans rax. On nomme ici %rsi la base est %rax l'index. Les deux registres doivent être de même taille.

- 5(%rdi,%rbx,8) désigne le contenu de la case mémoire dont l'adresse est 5 + l'adresse stockée dans rdi + (8 fois l'adresse stockée dans rbx). 8 est le facteur d'échelle. C'est forcément une constante, soit 1, soit 2, soit 4 soit 8. Le scale factor correspond à la taille de l'adresse : un octet, un mot (16 bits), un mot double (32 bites), ou un mot quadruple (64 bits).

Section 3

Comparaisons et branchements conditionnels

enfin, le registre rcx étant souvent utilisé pour contenir un compteur de boucle :

- jcxz si cx contient zéro
- jecxz si ecx contient zéro
- jrcxz si rcx contient zéro

Par exemple, l'instruction pascal « if a>b then c :=a else c :=b ; », a, b et c étant des variables globales entières non signées 32 bits, peut maintenant s'écrire : exemple du max entre deux variables.
Il y a un tableau sympa sur le cours mais il est faux. à reprendre et à

refaire

Section 4

Utilisation de la pile

Le processeur dispose d'un registre spécial rsp pour Re-extended Stack Pointer, pointeur de pile étendue. La pile est un tableau de cases mémoires contigües, et %rsp pointe toujours sur le sommet de la pile (le dernier élément empilé). C'est l'instruction pushq operande qui permet d'empiler, et popq operande qui permet de dépiler.
Les opérations se font toujours par 64 bits sur les processeurs 64 bits. Sur un système 32 bits, la pile est esp et les instructions sont pushl et popl.

Sur un système 16 bits, c'est sp, pushw et popw.

Section 5

Les procédures

Les instructions call et ret

call $\text{nom}_{\text{symbolique}}$ est l'instruction pour appeler la procédure à l'adresse symbolique $\text{nom}_{\text{symbolique}}$. Les paramètres de la procédure sont généralement sur la pile. À l'appel de l'instruction call, l'adresse au sommet du registre des adresses d'instructions rip est mis sur la pile (après avoir dépilé les paramètres ? utilisation du registre rbp ?), tandis que l'adresse dans le registre rip estremplacée par l'adresse associée à $\text{nom}_{\text{symbolique}}$.

ret est l'instruction pour quitter la procédure et revenir à l'instruction suivant l'appel de la procédure. Lorsque ret est appelé, l'adresse au sommet de la pile est dépilée et posée sur rip. ret peut-être appelé avec une opérande n : ret dépile l'adresse de l'instruction suivante puis dépile n autres éléments de la pile.

Le registre rbp pour Re-extended Base Pointer sert souvent pour mémoriser les différentes adresses importantes sur la pile (lorsqu'on veut accéder directement à un élément de la pile par exemple). Il pointe généralement sur l'adresse de retour de la procédure.

Les interruptions et les exceptions

Les processeurs acceptent une programmation événementielle dans la mesure où ils peuvent être interrompus (lors d'un click, mouvement de la souris, appui sur une touche, . . .). L'instruction int permet de communiquer avec le système d'exploitation.
L'instruction int constante a un fonctionnement similaire à l'instruction call : elle provoque l'exécution d'une procédure et comme son adresse est mémorisée, alors on peut revenir à l'instruction suivant une fois la procédure finie. La constante définit le numéro du gestionnaire d'interruptions. Suivant le gestionnaire, certains registres seront utilisés pour récupérer la procédure à exécuter et ses paramètres.
Par exemple, int $0x80 appelle le gestionnaire d'interruptions numéro 80 en hexa (128 en décimal), qui est le gestionnaire d'interruption du système d'exploitation sous GNU/Linus. eax va contenir le numéro de la procédure, ebx, ecx, edx, esi, et edi contiennent les 5 paramètres suivants (dans cet ordre).
Voir l'exemple Bonjour tout le monde!

Section 6

Multiplication et division sur des entiers

Sur des entiers non signés

Multiplication

mulb <x8> multiplie le contenu du registre al avec le contenu de <x8> qui est soit l'emplacement d'un octet, soit un registre 8 bits. Le résultat est stocké dans le registre 16 bits ax.
mulw <x16> les opérandes sont ax et <x16>, idem que ci-dessus. Pour le résultat, il est stocké dans dx :ax, i.e. les 16 bits de poids forts du résultat sont dans dx et les 16 bits de poids faibles dans ax (pour des raisons historiques, eax n'existait pas dans le processeur 8086). Ça fonctionne de manière similaire pour mull <x32> et pour mulq <x64>.
Les drapeaux OF et CF sont mis à 0 tous les deux si la moitié supérieure des bits du résultat (ceux de poids fort) sont à 0, et sinon ils sont tous les deux à 1.

Division

divb <x8> effectue la division de ax par x8. le quotient est stocké dans al et le reste dans ah.

Sur des entiers non signés

- Pour la multiplication c'est l'instruction lmul :
  - avec une seule opérande, elle agit comme mul
  - avec deux opérandes, la signification est source, destination
    - ★ source est une constante, un registre ou un emplacement mémoire
    - ★ destination est un registre général qui contiendra le résultat du produit source par destination
- Pour la division c'est l'instruction ldiv qui fonctionne comme div

Section 7

Opérateurs logiques

Les instructions and, or, xor, et not sont les opérations logiques bit à bit. Sauf not, elles ont deux opérandes sur lesquelles elles effectuent l'opération logique et stocke le résultat dans la deuxième opérande. L'instruction not fonctionne de la même façon avec une seule opérande.