Architecture des ordinateurs

L3 Informatique 2013-2014

TD/TP 5 - Assembleur : les bases

Points abordés

- Écrire des programmes en assembleur
- Comprendre les instructions
- Utiliser ddd

Exercice 1. (Premier exemple)

1. Copiez le programme suivant dans un fichier `exemple.asm`.

```
section .data

section .text
        global _start

_start:
        mov ax,128
        mov bx,453
        add ax,bx
        mov cl,al

        mov eax,1
        mov ebx,0
        int 80h
```

2. Compilez-le avec les instructions `nasm -f elf64 -g -o exemple.o exemple.asm`, puis `ld -o exemple exemple.o`.
3. Ouvrez le programme dans ddd avec `ddd exemple`.
4. Ouvrez la fenêtre d'affichage des registres et le panneau des outils de commande.
5. Placez un point d'arrêt à la ligne mov ax,128 en cliquant à gauche de cette instruction.
6. Exécutez le reste du programme pas à pas (commande Stepi) et observez l'effet de chaque instruction sur les registres.
7. Observez le code assembleur produit par ddd. Quelles sont les différences avec le code que vous avez entré ?

Exercice 2. (Stockage en mémoire)

Copiez le programme suivant dans un fichier `memoire.asm` et compilez-le.

```
section .data

  plop: dd 0

section .text
        global _start

_start:
        mov ecx,plop
        mov al,0xff
        add ah,0xdd
        mov [ecx], ax

        mov eax,1
        mov ebx,0
        int 80h
```

1. Exécutez ce programme pas à pas avec ddd et observez l'état de la mémoire (en l'affichant en hexadécimal). Comment sont stockées les données dans la mémoire ?
2. Dans quel ordre sont stockés les octets d'un mot 32 bits ?

Exercice 3. Le nombre limité d'instructions dont nous disposons permet de créer des programmes complexes. Dans chacun des exemples suivants, s'efforcer d'obtenir un code aussi efficace que possible.

1. Écrire un code assembleur qui charge une valeur t dans ax, calcule $8t + 2$ et place cette valeur dans cx. Faire de même pour calculer $3t - 2$, $13t + 145$ et $-2t + 4$.
2. Écrire un code assembleur qui charge deux valeurs dans ax et bx et met max(ax,bx) dans cx et min(ax,bx) dans dx.
3. Écrire un code assembleur qui charge deux valeurs dans les registres al et bl et calcule, dans cx, le résultat de leur produit.
4. Écrire un code assembleur qui déclare dans le segment de données une table de valeurs (que l'on initialisera directement dans les dw) et qui calcule le max et le min de cette table, ces résultats étant au final mis dans ax et bx.
5. La suite de Syracuse est une suite dont le premier terme u_0 est arbitraire et dont le terme u_{n+1} s'obtient en fonction de u_n comme suit :
   – $u_{n+1} = \frac{u_n}{2}$ si u_n est paire,
   – $u_{n+1} = 3u_n + 1$ sinon.
   La *conjecture de Collatz*, formulée dans les années 1930 et toujours ouverte, énonce que quelque soit le nombre u_0 que l'on fixe, il existe un entier n tel que $u_n = 1$.
   (a) Écrire un code assembleur qui place une valeur dans eax (représentant u_0), calcule les termes successifs $u_1, u_2, \ldots$ dans eax et met dans ebx le plus petit n tel que $u_n = 1$.
   (b) Écrire un code C qui fait de même. Ouvrez-le avec ddd (pensez à le compiler avec l'option -g), observez le code assembleur produit et comparez-le au votre.
   (c) Comparez les temps de calculs de vos deux programmes (pour voir la différence, modifiez vos programmes pour qu'ils effectuent le calcul sur beaucoup de valeurs).