Programmation, TD 3: exercices en assembleur Pentium

10 octobre 2002

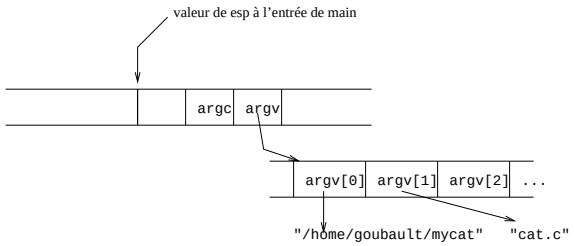
Le but de ce TD est d'écrire quelques programmes simples en assembleur. Vous pouvez les tester en salle machine comme suit. Créez un fichier `td_asm.s` tout d'abord. Le suffixe "`.s`" indique à `gcc` que le contenu est de l'assembleur. Écrivez ceci dedans (ou bien récupérez-le depuis `http://www.lsv.ens-cachan.fr/~goubault/CoursProgrammation/td_asm.s`):

```
.text
        .align 4
.globl main
        .type    main,@function
.hello:
        .string "hello.\n"
        .align 4
main:
        pushl %ebp
        movl %esp,%ebp
        pushl $.hello
        call printf
        addl $4, %esp
        movl %ebp,%esp
        popl %ebp
        ret
```

Vous remarquerez que les opcodes ont un "`l`" de plus à la fin par rapport au cours : il faut écrire `pushl` et non `push`, `movl` et non `mov`. La raison est que l'assembleur que vous utiliserez demande à ce que vous lui disiez la taille des données que vous voulez empiler, resp. déplacer : le suffixe `l`, pour "long", dit qu'on veut déplacer 32 bits, soit 4 octets.

1. Que fait ce programme ? Essayez de le déduire en lisant le source ci-dessus. (Note : les incantations du genre `.text`, `.globl main`, ou `.align 4` sont nécessaires, mais l'explication est technique. Souvenez-vous juste qu'il faut écrire un `.align 4` après chaque déclaration d'une zone de chaîne de caractères par la directive `.string`.)

   Vérifiez que ceci fait ce que vous pensiez : compilez-le en tapant `gcc -o td_asm td_asm.s`, et exécutez `./td_asm`. Si vous ne voyez pas pourquoi il fait ce qu'il fait, lancez `gdb td_asm`, ou mieux `ddd td_asm`, et regardez ce qui se passe en pas à pas.
2. (La notion de *plantage*.) Enlevez la ligne `addl $4, %esp`. Que fait maintenant le programme ? Expliquez pas à pas ce qui se passe après le retour de la fonction `printf`.
3. Modifiez le programme ci-dessus pour qu'il affiche le nombre d'arguments (`argc`, dans les exemples que nous avons vus en C). En C, on écrirait `printf ("%d.\n", argc)` pour le faire. Souvenez-vous que l'état de la pile (pointée par `%esp`) à l'entrée de `main`, est le suivant :



4. L'appel C `sscanf (argv[1], "%d", &i)`, où `i` est une variable de type `int` (un entier machine, signé, sur 32 bits) lit la chaîne de caractères `argv[1]`, la convertit en nombre et met ce nombre dans `i`. Par exemple, si `argv[1]` est la chaîne `"123"`, `i` recevra le nombre `123`.

   Modifiez le programme ci-dessus pour qu'il calcule le nombre en argument plus un. Par exemple :

   ```
   $ ./td_asm 123
   124
   $ ./td_asm -45
   -44
   ```

   Note : l'opcode de l'addition est `addl`.

   Note : pour en savoir plus sur `sscanf`, tapez `man sscanf`.

5. Que se passe-t-il si on appelle le programme précédent avec une syntaxe non prévue, par exemple en tapant l'une des commandes suivantes ?

   ```
   ./td_asm                    ./td_asm bonjour
   ./td_asm 45bonjour          ./td_asm 45 bonjour
   ```

6. Écrivez un programme qui calcule la fonction de Fibonacci F_n, où n est le premier argument (donc, représenté sous forme de chaîne caractères dans `argv[1]`), et affiche sa valeur à l'aide de `printf`. On rappelle que $F_0 = 1$, $F_1 = 1$, $F_{n+1} = F_n + F_{n-1}$ pour $n \geq 1$. Pour indication, voici la table des quelques premières valeurs de F_n :

n	2	3	4	5	6	10	20	30	40	50
F_n	2	3	5	8	13	89	10946	1346269	165580141	20365011074

   Vous écrirez pour cela une fonction `fib`, en ajoutant à la fin de la fonction `main` les lignes :

   ```
   fib:
         movl 4(%esp), %eax
   ```

   et en complétant. Notez que `movl 4(%esp), %eax` récupère dans le registre `%eax` la valeur du paramètre de la fonction `fib`, c'est-à-dire le n dont on veut calculer F_n. Le résultat F_n sera mis dans `%eax` avant de retourner de `fib`.

   Finalement, modifiez `main` pour qu'il appelle successivement `sscanf` pour lire la valeur de l'argument, `fib` pour calculer F_n, et `printf` pour afficher la valeur calculée.

   Testez votre programme en tapant :

   ```
   ./td_asm 2          ./td_asm 3          ./td_asm 10
   ./td_asm 30         ./td_asm 40         ./td_asm 50
   ```

   Que concluez-vous ?

   Note : pour calculer F_{n-1} et F_{n+1}, vous pouvez appeler `fib` (par `call fib`).

   Note : N'oubliez pas le `ret` pour retourner à la fin du calcul de `fib`.

   Note : quand le premier appel à `fib` retourne F_{n-1}, pensez à stocker F_{n-1} quelque part ! Si vous le laissez dans `%eax`, le second appel à `fib` écrasera cette valeur par d'autres...

Note : si vous n'arrivez pas à écrire les programmes demandés, que vous êtes perdus, mais que vous pensez que vous sauriez les écrire en C... écrivez-les en C, puis invoquez `gcc -S` sur votre source C, cela vous produira l'assembleur que je vous demande. Mais ne le faites pas sur tous les exercices, sinon vous n'apprendrez jamais à écrire un peu d'assembleur vous-même !