Processeur, Assembleur, Système, ou les arcanes du PC

Eric BERTHOMIER (eric.berthomier@free.fr)

3 novembre 2009

1 Préambule

Les livres c'est bien mais quand il faut expliquer les choses simples cela devient très vite compliqué. Il est donc écrit dans les livres que les fonctions principales d'un processeur sont de faire des calculs et d'adresser la mémoire.

Bien, que de théorie en si peu de lignes ! Et je n'aime pas la théorie inutile, donc prenons notre petit pinguoin favori, Debian et attelons nous à effectuer un dialogue avec notre processeur et bien sûr son compagnon de fortune, le système LINUX.

1.1 Assembleur

Nous allons utiliser dans le reste de cet article, deux langages de programmation, l'assembleur et le C.

Le C tout le monde connaît, pour ceux qui ne connaissent pas, se rendre sur mon site et télécharger le cours [1].

Le second langage utilisé est l'assembleur, en voici une définition :

Langage de programmation de bas niveau Chaque instruction correspond à une seule Instruction machine. Chaque assembleur est spécifique d'un microprocesseur.

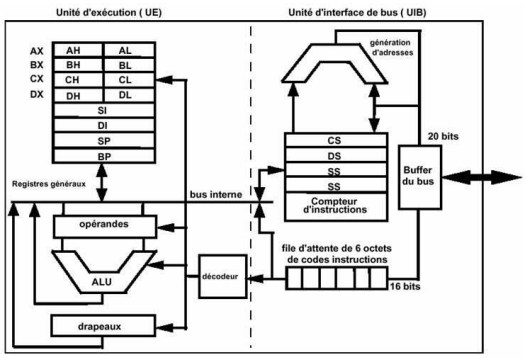
Avec l'assembleur nous sommes donc au plus proche de la machine (on peut être encore plus proche, mais restons modestes et ne nous brûlons pas les ailes comme Icare). Nous allons donc pouvoir dialoguer sans interface avec notre processeur (contrairement au C)..

1.2 Processeur

Nous allons dialoguer avec le processeur mais quel est-t'il vraiment, à part un petit rectangle qui s'adapte sur notre carte mère ? Un microprocesseur est composé des composants suivants :

[1] `http://eric.berthomier.free.fr/spip.php?article3`

- Horloge (en 4 temps : t1 t2 t3 t4)
- Unité se traitement (calculs)
  - une unité arithmétique et logique (ALU)
  - entourées de 3 registres tampons (A B C)
  - un ensemble de registres généraux
  - un registre d'adresse (RAD) et un de données (RDON)
  - 3 bus internes avec leur décodeur
  - un multiplexeur
- Unité de contrôle (séquenceur)
  - une mémoire microprogrammée
  - un registre d'adresse instruction (MCO)
  - un registre de micro-instruction (RMI)
  - un compteur incrémental (inc)
  - un mini séquenceur (MS)
  - un multiplexeur
- Bus de communications



Fin de la théorie, passons à la pratique.

2 Installation d'un éditeur de texte

Je n'ai rien contre `vi`, `emacs` ou tout autre éditeur mais lorsque l'on programme, il est parfois nécessaire d'avoir quelque chose d'un peu plus convivial. Pour ma part, j'utilise `Scite`.

```
1 su -
  apt-get install scite
  exit
```

3 Installation de l'assembleur

Nous allons utiliser l'assembleur `nasm` sur notre Linux pour pouvoir dialoguer avec le processeur. Ouvrir une console texte puis tapez

```
2 su -
  apt-get install nasm
  exit
```

Ces quelques commandes vont permettre d'installer `nasm` sur votre linux. Pour vérifier l'installation, tapez `nasm` puis Enter. Vous devriez obtenir ceci :

```
nasm: error: no input file specified
type 'nasm –h' for help
```

4 Juste un petit programme pour se faire plaisir

Passons aux choses sérieuses, ceux qui me connaissent ne seront pas étonné, on va commencer par la pratique et à partir de l'exemple expliquer le fonctionnement.

Copiez l'intégrité de ce programme ou téléchargez le.

hello.asm

```
section .data
        ; 'Hello world!' plus un retour chariot
hello:      db 'Hello world!',10
        ; Longueur de la chaine 'Hello world!'
helloLen:   equ $-hello

section .text
        global _start

_start:
        ; Appel Systeme write
        mov eax,4
        ; Descripteur de fichier : 1 - standard output
        mov ebx,1
        ; Localisation de hello in ecx
        mov ecx,hello
        ; Longueur de la chaine a afficher
        mov edx,helloLen
        ; Appel du noyau, intr 80h
        int 80h

        ; Al=65
        mov al, 65
        ; Remplacement du premier caractère
        ; par la valeur de al
        mov [hello], al

        ; Affichage
        mov eax,4
        mov ebx,1
        mov ecx,hello
        mov edx,helloLen

        int 80h

        ; ax=32
```

```
        mov ax, 32
38      ; bx=34
        mov     bx, 34
        ; ax=ax+bx
        add ax, bx

43      ; Remplacement du premier caractère
        ; par la valeur de al
        mov [hello], al

        mov eax,4
48      mov ebx,1
        mov ecx,hello
        mov edx,helloLen

        int 80h
53
        ; Sortie propre du programme
        ; Appel Systeme exit (sys_exit)
        mov eax,1
        ; Code Erreur de retour
58      mov ebx,0
        ; Interruption Noyau
        int 80h
```

Enregistrer le programme sous `hello.asm`.
L'assembleur bien que très proche de la machine n'est pas encore du langage machine, nous allons donc compiler notre programme pour le transformer en langage assimilable complètement par notre système. Pour se faire, exécuter les 2 commandes suivantes.

```
nasm -f elf hello.asm
ld -s -o hello hello.o
```

4.1 Quelques explications

1. L'option –f elf de nasm indique que l'on désire créer un exécutable au format ELF (ELF = Executable and Linking Format) qui est le format exécutable standard sous Linux (le format a.out, par exemple esr un autre format exécutable Linux, plus ancien).
2. L'option –s de ld permet d'épurer l'exécutable.
3. L'option –o de ld permet de définir le nom du programme que l'on veut générer.

4.2 Prouve le !

```
eric@souricier:$ nasm -f elf hello.asm

3 eric@souricier:$ ld -s -o hello hello.o
eric@souricier:$ ls -al hello
-rwxr-xr-x 1 eric eric 508 sep 28 22:08 hello

eric@souricier:$ ld -o hello2 hello.o
```

```
eric@souricier:$ ls -al hello2
-rwxr-xr-x 1 eric eric 837 sep 28 22:10 hello2
```

4.3 Qu'est ce que ça fait ?

Comme vous pouvez le remarquer, vous avez maintenant 2 exécutables, hello et hello2. Exécutez l'un ou l'autre (ce sont les mêmes sources, n'oubliez pas).

```
eric@souricier:$ ./hello
Hello world!
Aello world!
Bello world!
```

Ouahh toutes ces lignes pour si peu ! Maintenant, vous comprenez que l'on utilise l'assembleur que lorsqu'il est vraiment nécessaire, dans les autres cas, nous utilisons le C ou d'autres langages évolués.

5 La même chose en C

Voici le même code en langage C. Pour le tester, il vous faut installer le compilateur C, pour se faire :

```
su -
apt-get install gcc
exit
```

Puis tapez le code source suivant :

helloC.c

```
#include <stdio.h>

int main ()
{
        char strHello[]="Hello world!\n";

        printf ("%s",strHello);

        strHello[0]='A';
        printf ("%s",strHello);

        strHello[0]=32+34;
        printf ("%s",strHello);

        return 0;
}
```

Compilez le et exécutez le :

```
eric@souricier:$ gcc -c helloC.c
eric@souricier:$ gcc -o helloC helloC.o
eric@souricier:$ ./helloC
Hello world!
Aello world!
Bello world!
```

Même résultat avec 4 fois moins de ligne. Même résultat, vous êtes sûr ? Regardons la taille de notre exécutable :

```
eric@souricier:$ ls -al helloC
-rwxr-xr-x 1 eric eric 6,2K sep 28 22:23 helloC
```

Eh oui, outil plus puissant mais taille de programme plus **IMPORTANTE**.

À noter de plus que le programme ainsi construit nécessite l'utilisation de librairies annexes pour fonctionner ![2]

```
eric@souricier:$ ldd helloC
        linux-gate.so.1 =>  (0xb7f2a000)
        libc.so.6 => /lib/i686/cmov/libc.so.6 (0xb7db5000)
        /lib/ld-linux.so.2 (0xb7f2b000)
```

6 Définitions

Afin de comprendre le programme, nous allons expliquer toutes les notions de bases utilisées par celui-ci. Ne vous inquiétez pas si vous pensez que vous êtes le seul à ne pas savoir, il n'y a que les hackers pour s'amuser avec ça !

Prenez un café, le passage va être douloureux.

6.1 Définition des registres

Les registres AX, BX, CX et DX sont des registres de 16 bits, ils se décomposent en deux sous-registres de 8 bits chacun.
h (high : partie haute) et l (low : partie basse).

Par exemple, ax se décompose en ah et al. À tout instant : AX = AH * 256 + AL.

Lorsque nous devons utiliser un nombre inférieur à 256, nous n'avons donc besoin que d'utiliser AL.

En plus des registres évoqués, il existe les registres étendus (EAX, EBX ...) sont des registres de 32 bits.

[2] Pour plus d'explications, voir le cours de C

6.2 Définition d'un caractère

Chaque caractère possède son équivalent en code numérique : c'est le code ASCII (American Standard Code for Information Interchange). Ce code numérique va de 0 à 255 et peut être codé sur 8 bits.

Un caractère peut donc être initialisé via l'utilisation de la partie haute ou basse du registre (AL ou AH par exemple).

Exemple de caractères A a pour valeur Ascii 65.
Z a pour valeur Ascii 90.

Ces valeurs n'ont de sens que par leur interprétation, c'est parceque l'on demande d'afficher le caractère *dont le code ASCII est* que l'ordinateur effectuera cet affichage, pour lui tout n'est que bits.

6.2.1 Petit exemple en C

caractereC.c

```
1  #include <stdio.h>

   int main ()
   {
           char car=65;
6
           printf ("%c\n",car);
           printf ("%d\n",car);

           return 0;
11 }
```

La compilation puis l'exécution de ce code nous donne :

```
   eric@souricier:~/$ gcc -o caractereC caractereC.c
   eric@souricier:~/$ ./caractereC
   A
4  65
```

Si vous avez bien fait attention, vous avez pu remarquer l'utilisation de \n. Ce caractère spécial aussi noté 0x0A en hexadécimal, est un retour chariot, il permet en outre (eh oui, ce caractère est un peu magique) de revenir à la ligne et d'avoir ainsi un affichage plus élégant.

6.3 Définition d'une interruption

Une interruption est un signal qui demande l'attention du CPU ou du système d'exploitation (Linux pour nous). En assembleur, une interruption est générée grâce à l'instruction `int`.

6.3.1 Interruption BIOS

Une interruption de ce type peut être provoquée par le BIOS ou être invoquée par le programme vers le BIOS ou le système.

Par exemple l'interruption 09h (IRQ1) est envoyé au BIOS lorsqu'une touche du clavier a été pressée.

Pour que mon programme puisse lire cette touche, je dois utiliser l'interruption 16h avec AH=0. Ainsi, si une touche a été frappée, le BIOS nous retournera dans AH le scancode de la touche et dans AL le code ASCII de cette dernière, dans le cas contraire, celui-ci nous retourera simplement AL=0.

Pour plus d'informations sur les interruptions BIOS :

- `ftp://ftp.embeddedarm.com/old/saved-downloads-manuals/EBIOS-UM.PDF`
- `http://lmi17.cnam.fr/~anceau/Documents/clavier.pdf`

6.3.2 int 80h

Vous remarquerez que l'interruption utilisée dans notre programme est l'interruption 80h. Cette interruption permet de faire un appel au système Linux. Dans le cadre de notre exemple, nous avons utilisé cette interruption pour faire afficher un texte sur le handle de fichier 1 soit la sortie standard (i.e. la console) et sortir proprement de notre programme (erreur=0).

6.4 Format d'un programme assembleur

Un programme assembleur est composé de 3 parties :
- Une section data qui contient les données initialisés (Exemple a=4)
- Une section text qui contient le code
- Une section bss qui contient les données non initialisés (mise a zéro) au démarrage du code.

7 Le programme

Maintenant nous pouvons reprendre notre programme.

7.1 .data

Dans cette partie, nous avons déclaré 2 zones de mémoire,

- l'une contenant la chaîne Hello world ! suivi d'un retour chariot (0x0A = 10)
- l'autre contenant la longueur de cette chaîne

7.2 .text

Cette partie contient le code à exécuter.

7.2.1 Affichage

La majeure partie du code consiste à écrire la chaine de caractère que nous avons stocké précédemment. Aussi pour cela nous utilisons le code suivant :

affichage

```
        ; Appel Systeme write
        mov eax,4
        ; Descripteur de fichier : 1 - standard output
        mov ebx,1
        ; Localisation de hello in ecx
        mov ecx, hello
        ; Longueur de la chaine a afficher
        mov edx, helloLen
        ; Appel du noyau, intr 80h
        int 80h
```

Pour effectuer un affichage l'interruption 80h est invoquée avec le paramétrage suivant :

- eax = 4, codé de la façon suivante : `mov eax,4`
- ebx = handle du fichier sur lequel il faut écrire, ici 1 donc la sortie standard, codé de la façon suivante : `mov ebx, 1`
- ecx = adresse mémoire de la chaîne à afficher
- edx = taille de la chaîne à afficher.

Une fois tous ces paramètres indiqués, il ne reste plus qu'à appeler l'interruption.

7.2.2 Modification directe de Hello World !

La suite du code consiste à modifier le contenu de la chaîne de caractères. Il est nécessaire de savoir que pour modifier le contenu d'une adresse mémoire et non une adresse mémoire il faut utiliser la syntaxe suivante :

initialisation

```
mov al, valeur
mov [hello], al
```

Notez les **[]** autour de `hello` qui indique le contenu de l'adresse mémoire.

7.2.3 Modification après addition

Ici le but est de montrer les capacités de calcul du processeur. au lieu de mettre la valeur 66 directement dans `[hello]`, nous la faisons calculer.

Le calcul se fait de la façon suivante :

- ax contient une des opérandes
- bx contient l'autre opérande
- ax contient le résultat de l'opération après addition (add)

Le remplacement dans la chaîne se fait de la même façon qu'auparavant. Nous utilisons AL comme registre car un caractère est codé sur 1 octet soit 8 bits soit AL ou AH.

7.2.4 Sortie du programme

Un programme doit se terminer en retournant un code erreur égal à 0 lorsqu'il se termine correctement et différent de 0 autrement. Cette sortie "sans erreur" est effectuée par l'appel de l'interruption 80h avec `EAX=1` (commande Exit) et `EBX=0` (Code erreur).

7.3 Le programme en C

Je n'expliquerai pas le programme C, il fait la même chose que le programme Assembleur.

8 Conclusions

Au travers de cet exemple, vous avez pu effleurer le dialogue avec le processeur et le système si on peut oser dire ! J'espère que ceci vous aura permis de mieux comprendre comment fonctionne un système d'exploitation. Un second article devrait vous permettre de voir la différence de vitesse entre l'assembleur et le C. À venir, patience.

9 Remerciements

Merci à mes relecteurs Laurent et Daniel.
Merci à ceux qui osent me poser des questions sans quoi je n'écrirai plus ...

10 Bibliographie

– L'image du microprocesseur :
`http://www.technologuepro.com/`
– La définition de l'Assembleur :
`http://www.pixelle.org/dictionnaire_informatique.php?define=assembleur`
– Linux Assembly Tutorial :
`http://www.cin.ufpe.br/~if817/arquivos/asmtut/index.html`
– Pour se rappeler l'addition en assembleur :
`http://fr.wikiversity.org/wiki/Assembleur/Le_langage_assembleur`
– ELF :
`http://cs.mipt.ru/docs/comp/eng/os/linux/howto/howto_english/elf/elf-howto-1.html`
– Int 80h :
`http://myweb.stedwards.edu/laurab/cosc2331/int80h.html`
– Format d'un programme assembleur :
`http://cerebral-vortex.net/index.php?id=61`
– Définition d'une interruption :
`http://www.iprezo.org/index.php?page=int`


YES WE CAN
אוגוסט פינגוין
2009