INFO 301 – **2006/2010**

Architecture des Ordinateurs: Contrôle de connaissances

Programmation assembleur

décembre 2006

1 Questions

Question 1.1 : *Un pangramme est une chaîne de caractères qui contient toutes les lettres de l'alphabet. Le pangramme suivant comporte* 42 *lettres :*

`Portez ce vieux whisky au petit juge blond qui fume`

On suppose qu'une chaîne de caractères est stockée en mémoire à l'adresse `MMNN` *et que cette suite d'octets se termine bien par un* 0*.*

Écrivez un programme assembleur qui vérifie si une chaîne de caractères est un pangramme : ce programme se termine avec un 1 *dans le registre* `R4` *si c'est le cas et* 0 *sinon. Notez que seules l'espace — code ascii* 32 *— et les lettres de l'alphabet (minuscules — code ascii compris entre* 97 *et* 122 *— ou/et majuscules — code ascii compris entre* 65 *et* 90*) peuvent composer un pangramme.*

N'hésitez pas à indiquez les espaces mémoire dont vous supposez l'existence pour faire votre programme. On suppose qu'aucun registre n'est utilisé avant l'appel de votre programme et vous n'avez pas à gérer d'éventuelles erreurs comme la présence de caractères non alphabétiques dans votre chaîne.

2 Jeu d'instructions

On dispose de registres 8 bits généraux nommés de $R0$ à $R7$. On dénomme `RX0` le registre 16 bits obtenu par la concaténation de `R1` et `R0` (`R1` octet de poids fort, `RX1` par `R2` et `R3`, `RX2` par `R4` et `R5` et `RX3` par `R6` et `R7`.

Notre microprocesseur dispose des instructions suivantes :

- `ST R0, RXn` — (*pour STore*) qui stocke la valeur du registre `R0` en mémoire à l'adresse (sur 16 bits) contenue dans le registre `RXn` ;
- `LD RXn, R0` — (*pour LoaD*) qui charge dans le registre `R0` la valeur stockée en mémoire à l'adresse (sur 16 bits) contenue dans le registre `RXn` ;
- `MV` *arg*`,R`*n* — (*pour MoVe*) qui charge dans le registre `R`*n* la valeur de l'argument "*arg*". L'argument est soit une valeur immédiate, soit un registre ;
- `JMP label` — (*pour JuMP*) qui effectue un branchement à l'adresse (sur 16 bits) spécifiée par l'étiquette `label` ;
- `ADD R`*m*`, R`*n* — (*pour ADD :-)*) qui additionne la valeur du registre `R`*n* avec la valeur du registre `R`*m* et stocke le résultat dans le registre `R`*n* ;
- `SUB R`*m*`, R`*n* — (*pour SUBstract*) qui soustrait la valeur du registre `R`*m* à la valeur du registre `R`*n* et stocke le résultat dans le registre `R`*n* ;
- `AND R`*m*`, R`*n* — qui fait la conjonction bit à bit des deux registres et range le résultat dans `R`*n* ;

- `DEC Rn` — (*pour DECrement*) qui soustrait 1 à la valeur de `Rn` et stocke le résultat dans `Rn` ;
- `INC Rn` — (*pour INCrement*) qui ajoute 1 à la valeur de `Rn` et stocke le résultat dans `Rn` ;
- `NOT Rn` — qui inverse bit à bit le registre `Rn` et qui y stocke le résultat ;
- `JZ label` — (*pour Jump if Zero*) qui effectue un saut à l'adresse 16 bits spécifiée par l'étiquette `label` si l'opération précédente a donné un résultat nul ;
- `JC label` — (*pour Jump if Carry*) qui effectue un saut à l'adresse 16 bits spécifiée par l'étiquette `label` si l'opération précédente a engendrée une retenue (ou si le résultat est négatif).

3 Table de vérité, formes normales et circuit logique

Exercice 4 :

On se donne la table de vérité suivante :

a	b	c	res
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

1. Donnez une forme normale associée à cette table.
2. Dessinez un circuit logique associé à cette table.

5 Lecture et correction

Exercice 6 :

On se donne un code assembleur (voir page suivante).

1. Que fait ce code ?
2. Une erreur algorithmique a été faite par le programmeur, corrigez la.

```
.data
str:    .zero   11
i:      .long   463960
d:      .long   1
j:      .zero   4
.text .globl _start
_start: movl    i,%ebx
L1:     movl    i, %eax
        movl    $0,%edx
        movl    d,%ecx
        divl    %ecx
        push    %edx
        movl    $10,%eax
        mull    %ecx
        pop     %edx
        movl    %eax,d
        cmpl    %edx,%ebx
        jne     L1
        movl    $str, %ebx
        movl    d, %eax
        movl    $0,%edx
        movl    $10,%ecx
        divl    %ecx
        movl    %eax,d
L2:     movl    d, %eax
        movl    $0,%edx
        movl    $10,%ecx
        divl    %ecx
        movl    %eax,d
        movl    i, %eax
        movl    $0,%edx
        movl    d,%ecx
        divl    %ecx
        movl    %edx,i
        movl    j, %ecx
        movl    $0, %edx
        addl    $'0',%eax
        movb    %al, %ds:(%ebx,%ecx,1)
        inc     %ecx
        movl    %ecx,j
        movl    i, %eax
        movl    $0,%edx
        cmpl    %eax, %edx
        jne     L2
        movl    j, %eax
        addl    $str, %eax
        movl    $0, %edx
        movb    $0, %ds:(%edx,%eax,1)
        movl    $1, %eax
        movl    $0, %ebx
        int     $0x80
```