Contrôle continu
Programmation des Systèmes

Exercice 1: La Figure 1 représente la structure d'une mémoire cache ARM 940 T. En sachant que c'est une mémoire de type read-allocate, write-through:

- Quelle est la capacité de la mémoire cache?
- Ecrivez en pseudo-code les actions exécutées lorsque le processeur effectue une lecture.
- Ecrivez en pseudo-code les actions exécutées lorsque le processeur effectue une écriture.

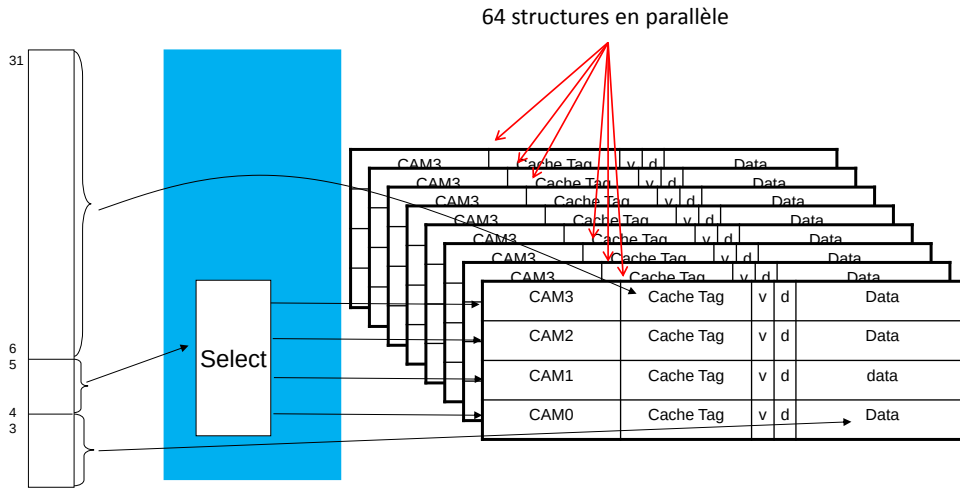

Figure 1

Exercice 2: La fonction write permet d'écrire dans un fichier, son prototype est

*ssize_t write(int fd, const void *buf, size_t count);*

où *fd* est un descripteur de fichier (file descriptor), *buf* un pointeur sur la chaîne de caractères et *count* le nombres de caractères à afficher. En utilisant les informations qui se trouvent sur les Figures 2 et 3 écrivez une routine en assembleur ARM (comme aux exercices) qui appelle la fonction write en faisant un appel système.

```
# define STDIN_FILENO 0
# define STDOUT_FILENO 1
# define STDERR_FILENO 2
...
# define _NR_setup 0
# define _NR_exit 1
# define _NR_fork 2
# define _NR_read 3
# define _NR_write 4
# define _NR_open 5
# define _NR_close 6
...
```

Figure 2: Extrait fichier *unistd.h*

Arch/ABI	Instruction	System call #	Ret val	Ret val2	Error
alpha	callsys	v0	v0	a4	a3
arc	trap0	r8	r0	-	-
arm/OABI	swi NR	-	a1	-	
arm/EABI	swi 0x0	r7	r0	r1	-
arm64	svc #0	x8	x0	x1	-
blackfin	excpt 0x0	P0	R0	-	-
i386	int $0x80	eax	eax	edx	-
ia64	break 0x100000	r15	r8	r9	r10
m68k	trap #0	d0	d0	-	-

Figure 3: Extrait du manuel *syscall* de linux

Exercice 3: Expliquez l'algorithme de division de la Figure 4. Implémentez l'algorithme sous la forme d'une routine en assembleur qui accepte les paramètres selon la convention ARM.

```
unsigned udiv(unsigned d, unsigned n){
        unsigned q=0, r=n, N=32;
        do {
                N--;
                if ((r>>N) >= d) {
                        r -= (d<<N);
                        q += (1<<N);
                }
        } while (N);
        return q;
}
```

Figure 4: Algorithme de division

Exercice 4: Ecrire une procédure **récursive** en assembleur qui calcule la suite des nombre de Fibonacci,

$$\mathcal{F}_{n+1} = \mathcal{F}_n + \mathcal{F}_{n-1},$$

avec conditions initiales $\mathcal{F}_1 = \mathcal{F}_2 = 1$.

Exercice 5: En compilant la procédure *timer* en C de la Figure 5 on obtient le programme assembleur de la Figure 6. Décrivez le code assembleur ligne par ligne, en particulier, expliquez pourquoi le programme n'est pas correct si on supprime la lecture à la ligne 6 en caractères gras.

```
void timer(int *timer1, int *timer2, int *step)
{
*timer1 += *step;
*timer2 += *step;
}
```

Figure 5: Procédure *timer* en C

```
timer:  LDR   r3, [r0,#0]
        LDR   r12, [r2,#0]
        ADD   r3, r3, r12
        STR   r3, [r0, #0]
        LDR   r0, [r1, #0]
        LDR   r12, [r2,#0]
        ADD   r0, r0, r12
        STR   r0, [r1, #0]
        MOV   pc, r14
```

Figure 6: Procédure *timer* en assembleur ARM