CONTRÔLE INTERMÉDIAIRE – L3 INFO : ARCHITECTURE

Durée =2h ; documents autorisés : polycopié « Assembleur ARM », cours, TPs

Exercice 1 (4 points)

En utilisant le document fourni en annexe :

- Quelle est l'instruction ARM dont le codage hexadécimal est : `E1899008 ?`
- Coder en hexadécimal l'instruction ARM : `ADD r1, r2, r3, LSL r4`

Exercice 2 (8 points)

1. Écrire une fonction `longueur(chaine)` qui calcule la longueur d'une chaîne de caractères `chaine`. On rappelle qu'une chaîne de caractères se termine par le caractère nul (`'\0'`). Le passage de paramètres devra se faire par la pile.
2. Écrire une fonction `renverse(chaine1, chaine2)` qui écrit à l'envers une chaîne de caractères `chaine1` et stocke le résultat dans `chaine2`. `chaine1` et `chaine2` sont donc les adresses de ces deux chaînes. On utilisera obligatoirement la fonction `longueur` écrite ci-dessus. Le passage de paramètres devra se faire par la pile.

   Écrire également le programme principal qui teste la fonction `renverse`.

   Exemple :

   si `chaine1 = « salut »`, après l'appel de `renverse` on aura `chaine2 = « tulas »`

Exercice 3 (8 points)

Écrire une fonction `puissance(a,n)` qui prend en paramètre d'entrée deux entiers `a` et `n` (que l'on supposera strictement positifs) et renvoie en paramètre de sortie a^n. L'algorithme **obligatoirement** utilisé, appelé puissance indienne, est le suivant :

```
int puissance (int a,int n){
     if (n==0)return 1 ;
     if (n%2==0){
          x=puissance(a,n/2) ;
          ret=x*x ;
     }else{
          x=puissance(a,n-1) ;
          ret=x*a
     }
     return ret ;
}
```

Exemple : `puissance(3,5)=243`

Les paramètres seront passés par la pile et la fonction sera obligatoirement **récursive**.

Annexe : Codage des instructions en assembleur ARM

Codage d'une instruction arithmétique

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				0	0	#	*op*				*S*	*Rn*				*Rd*				*operand2*											

Les champs variables ont la signification suivante :

- *cond* : condition d'exécution de l'instruction. Les différentes valeurs possibles sont les suivantes :

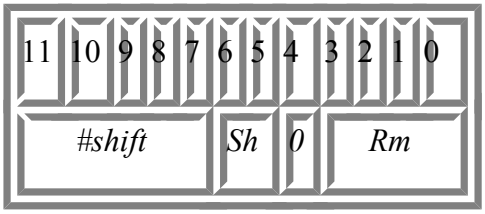
valeur	**condition**		**valeur**	**condition**
0000	EQ		1000	HI
0001	NE		1001	LS
0010	CS		1010	GE
0011	CC		1011	LT
0100	MI		1100	GT
0101	PL		1101	LE
0110	VS		1110	{AL}
0111	VC			

  Comme nous n'utilisons pas l'exécution conditionnelle (l'instruction doit toujours être exécutée), on choisira AL.
- # : format du 2nd opérande. S'il s'agit d'une valeur immédiate, ce champ est à 1. Sinon, il est à 0.
- *op* : code opération. Les différentes valeurs possibles sont les suivantes :

valeur	**condition**		**valeur**	**condition**
0000	AND		1000	TST
0001	EOR		1001	TEQ
0010	SUB		1010	CMP
0011	RSB		1011	CMN

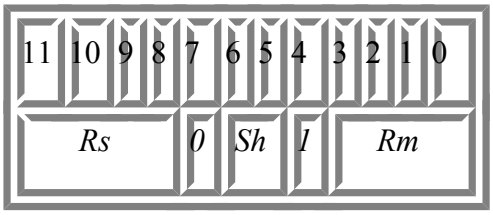
0100	ADD		1100	ORR
0101	ADC		1101	MOV
0110	SBC		1110	BIC
0111	RSC		1111	MVN

- *S* : est à 1 si les codes conditions doivent être mis à jour.
- *Rn* : numéro du registre opérande 1
- *Rd* : numéro du registre destination
- *operand2* : second opérande. Trois formats sont possibles :
  - opérande registre, décalage éventuel spécifié par une constante :



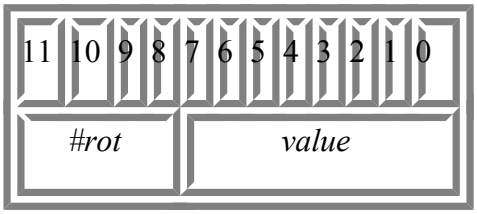
    - *#shift* : nombre de positions de décalage
    - *Sh* : type de décalage (00 = LSL, 01 = LSR, 10 = ASR, 11 = ROR)
    - *Rm* : numéro du registre

  - opérande registre, décalage éventuel spécifié par un registre :



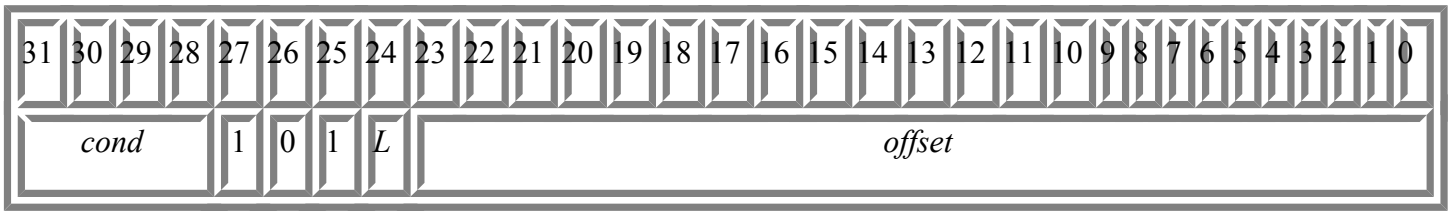
    - *Rs* : numéro du registre contenant le nombre de positions de décalage
    - *Sh* : type de décalage (00 = LSL, 01 = LSR, 10 = ASR, 11 = ROR)
    - *Rm* : numéro du registre

  - opérande immédiat (de la forme valeur_8_bits * 2^2*n*) :



    - *#rot* : *n*
    - *value* : valeur sur 8 bits

Codage d'une instruction de branchement



Les champs variables ont la signification suivante :

- *cond* : condition du branchement. Les différentes valeurs possibles sont les mêmes que pour les instructions arithmétiques (voir ci-dessus).
- *L* : Si l'adresse suivante doit être mémorisée dans r14 (adresse de retour), ce champ est à 1. Sinon, il est à 0.
- *offset* : déplacement (*d*) divisé par 4. L'adresse cible du branchement sera calculée comme :

```
adresse cible = adresse du branchement (PC) + 8 + d
```