TP N° 1 : PREMIERS PROGRAMMES

Exercice 1 : premier programme et arm-elf-insight

1. Saisir le programme suivant (que vous appellerez `ex1.s` et que vous placerez dans un répertoire `L3-archi`) dans un éditeur de texte :

```
.global  _start
_start:  mov      r0,#3       @ r0 = 3
         mov      r1,#1       @ r1 = 1
         add      r0,r0,#3    @ r0 ← r0 + 3
         sub      r2,r1,r0    @ r2 ← r1 - r0
         adr      r3,tab1     @ r3 ← tab3
exit:    b        exit        @ boucle infinie

tab1:    .int     2,3,-1
tab2:    .fill    4,2,12
chaine:  .asciz   "bonjour"
tab3:    .byte    13,-3,'A'
```

2. Pour générer un exécutable pour ce programme :
   - copier le fichier `makefile` disponible sur moodle dans le répertoire `L3-archi`
   - modifier dans le `makefile` le nom du programme à compiler (2eme ligne du `makefile`)
   - générer l'exécutable : pour cela taper la commande `make.` Si tout se passe bien le fichier exécutable `ex1.elf` a été créé.
3. Exécuter le programme grâce au debugger. Pour cela :
   - Lancer le debugger taper la commande `arm-elf-insight&`
   - Dans le menu "File", sélectionner "Open ..." et choisir le nom de l'exécutable (ex : `ex1.elf`).
   - Dans le menu "Run", sélectionner "Run". La première fois, la fenêtre "Target Selection" s'ouvre.
   - Dans la fenêtre "Target Selection" (qui s'ouvre à la première exécution, et qu'on peut rappeler en choisissant "Target settings" dans le menu "File") : choisir Target = Simulator
4. Exécuter le code pas à pas. Tester les fonctionnalités du debugger :
   - visualisation des registres : regarder l'évolution des registres r0,r1, r2 et r3
   - visualisation de la mémoire : examiner le contenu de la mémoire à l'adresse tab1, tab2, chaine et tab3. Expliquer les valeurs observées ,
   - utilisation de points d'arrêt, ...

Exercice 2 : codage d'instruction ARM

En utilisant le document fourni en annexe :

1. Coder en hexadécimal les instructions ARM :
   - `sub r2, r1, r0`
   - `add r2, r1, r0, LSL #5`
   - `add r0, r0, #130`
2. Retrouver ces codes dans l'exécutable désassemblé (`ex1.dis`) et en mémoire.
3. Quelle est l'instruction ARM dont le codage hexadécimal est : `E1899008 ?`

Exercice 3 : load et store

1. Écrire un programme qui à partir de deux entiers stockés dans deux variables en mémoire `var1` et `var2`, en fait la somme et stocke le résultat dans une variable en mémoire `res`.
2. On considère un tableau `tab` en mémoire contenant 3 entiers. Écrire un programme qui met le contenu de la case 1 dans la case 2, puis le contenu de la case 0 dans la case 1 puis met à 0 la case 0.
3. On considère un tableau `tab` en mémoire contenant les octets 0xA0, 0xD0, 0x74 et 0x24. Écrire un programme qui permute la première et la dernière case du tableau.
4. Écrire un programme qui met en minuscule une chaîne de 3 caractères `chaine3` écrite en majuscule et stockée en mémoire.

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 :

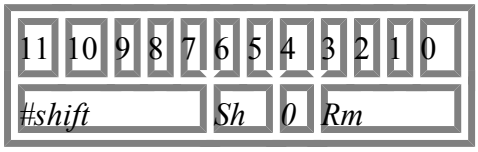
- *cond* : condition d'exécution de l'instruction. Les différentes valeurs possibles sont les suivantes :

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 :

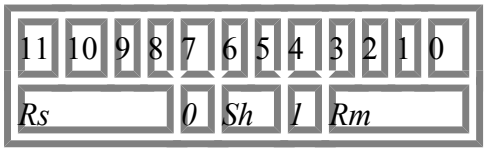
valeur	**condition**		**valeur**	**condition**
0000	AND		1000	TST
0001	EOR		1001	TEQ
0010	SUB		1010	CMP
0011	RSB		1011	CMN
0100	ADD		1100	ORR
0101	ADC		1101	MOV
0110	SBC		1110	BIC
0111	RSC		1111	MVN

- *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 :



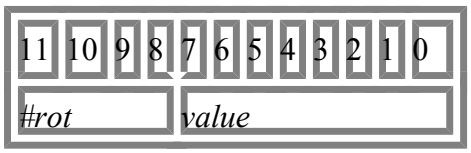
    - *#shift* : nombre de positions de décalage
    - *Sh* : type de décalage (00 = LSL, 01 = LSR, 10 = ASR, 11 = ROR)
    - *Rm* : numéro du registre

  - opérande registre, décalage éventuel spécifié par un registre :



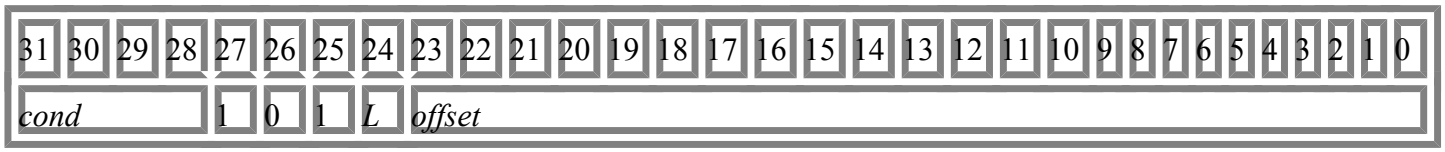
    - *Rs* : numéro du registre contenant le nombre de positions de décalage
    - *Sh* : type de décalage (00 = LSL, 01 = LSR, 10 = ASR, 11 = ROR)
    - *Rm* : numéro du registre

  - 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
```