TD - LE PROCESSEUR ARM7

L'ARM7 est un processeur de marque ARM Ltd. Les processeurs ARM sont utilisés et dans de nombreuses application embarquées : smartphones, tablettes, …

Les processeurs ARM7 sont programmables avec des instructions 32 bits.

Données manipulées sont des **mots** (32 bits)**, demi-mots** (16 bits), **Octets** (8 bits)

L'architecture repose sur 17 registres :

- des registres généraux : R0 à R12,
- un registre « pointeur de pile » : SP (R13),
- un registre dit « de lien » : LR (R14),
- un registre « pointeur de programme » : PC (R15),
- un registre de statut : CPSR.

LE POINTEUR DE PILE SP

le **pointeur de pile** (SP, Stack Pointer) pointe une zone spéciale de la mémoire appelée pile où sont rangés les arguments des sous-programmes et les adresses de retour.

LE REGISTRE DE LIEN LR

le **registre de lien** Indique à partir d'où un appel de fonction a été fait. Sa copie dans R15 permet de poursuivre l'exécution du programme après celle d'une fonction;

LE POINTEUR DE PROGRAMME PC

le **pointeur de programme** (Program Counter) contient l'adresse de la prochaine instruction à exécuter ;

LE REGISTRE DE STATUT CPSR

le **registre de statut** donne l'état du microprocesseur à tout moment, il peut seulement être lu :

N : négatif
Z : zéro
C : retenue
V : dépassement

INSTRUCTIONS DE PROGRAMME

En assembleur, une instruction de programme est constituée de 4 champs :

```
<Etiquette:>  <Mnémonique code opération>  <Opérandes>  <@Commentaire>
```

```
@exemple de programme assembleur ARM.
debut:  mov   r0,#127     @lecture donnee decimale en mémoire
        mov   r1,#16      @recuperation du premier mot
        add   r2,r0,r1    @addition des valeurs
```

INSTRUCTIONS DE BASE DU PROCESSEUR ARM7

OPERATIONS ARITHMETIQUES ET LOGIQUES

```
ADD     R0,R1,R2        @R0=R1+R2 addition
ADC     R0,R1,R2        @R0=R1+R2+C addition avec retenue
SUB     R0,R1,R2        @R0=R1-R2 soustraction
SBC     R0,R1,R2        @R0=R1-R2+C-1 soustraction avec retenue
RSB     R0,R1,R2        @R0=R2-R1 soustraction inversee
RSC     R0,R1,R2        @R0=R2-R1+C-1 soust. inversee + retenue
AND     R0,R1,R2        @R0=R1 ET R2 (ET logique)
ORR     R0,R1,R2        @R0=R1 OU R2 (OU logique)
EOR     R0,R1,R2        @R0=R1 XOR R2 (OU exclusif)
BIC     R0,R1,R2        @R0=R1 ET NON R2
```

MOUVEMENT DE REGISTRES

```
MOV     R0,R1           @R0=R1
MVM     R0,R1           @R0=NON R1
MOV     pc,lr           @fin de sous-programme
```

COMPARAISON

```
CMP     R1,R2           @PSR <- R1-R2
CMN     R1,R2           @compare R1 et NON R2
```

ADRESSAGE IMMEDIAT

```
ADD     R0,R1,#1        @R0=R1+1
AND     R0,R1,#0xFF     @R0=R1 ET 0xFF
MOV     R0,#5           @R0=5
```

MULTIPLICATION

```
MUL     R0,R1,R2        @R0=R1*R2 sur 32 bits
MLA     R0,R1,R2,R3     @R0=R1*R2+R3 sur 32 bits
MLS     R0,R1,R2,R3     @R0=R1*R2-R3 sur 32 bits
```

TRANSFERT DE DONNEES

```
LDR     R0,[R1]         @Load à partir du registre
STR     R0,[R1]         @Store dans le registre
ADR     R1,Table1       @R1=adresse de Table1
LDR     R0,[R1,#4]      @R0=[R1+4]
LDR     R0,[R1,#4]!     @R0=[R1+4] puis R1=R1+4
LDR     R0,[R1],#4      @R0=[R1] puis R1=R1+4
```

BRANCHEMENT

```
B       Label
BL      label           @Branch and Link
BNE     label           @Branch si Non Equal
BEQ     label           @Branch si Equal
BGT     label           @Branch si Greater Than

Exemple de branchement :
CMP     R0,#5
BEQ     saut            @si R0=5 branchement à saut
…
saut:   …
```

*SIGNIFICATION DES INSTRUCTIONS CONDITIONNELLES

EQ : égalité (Z=1) / NE : non égal (Z=0) / GE : plus grand ou égal
LT : plus petit que / GT : plus grand que / LE : plus petit ou égal

DOUBLE TEST AVEC UNE INSTRUCTION CONDITIONNELLE

En python :

```
if R0==R1 and R2==R3:
          R4=R4+1
```

En assembleur:

```
CMP R0,R1
CMPEQ R2,R3
ADDEQ R4,R4,#1
```

LE SIMULATEUR « ARMSIM# »

Le simulateur libre « ARMSim# » a été développé par l'université de Victoria (Canada). Il permet :

- d'importer un programme écrit dans un éditeur de texte,
- de déboguer les programmes d'assemblage ARM
- de surveiller l'état du système pendant qu'un programme s'exécute. Les informations de surveillance incluent les états es différents registres et les cycles d'horloge.

Lien de téléchargement sur le site de la classe : http://www.info-nsi.fr/nsi-assembleur.html

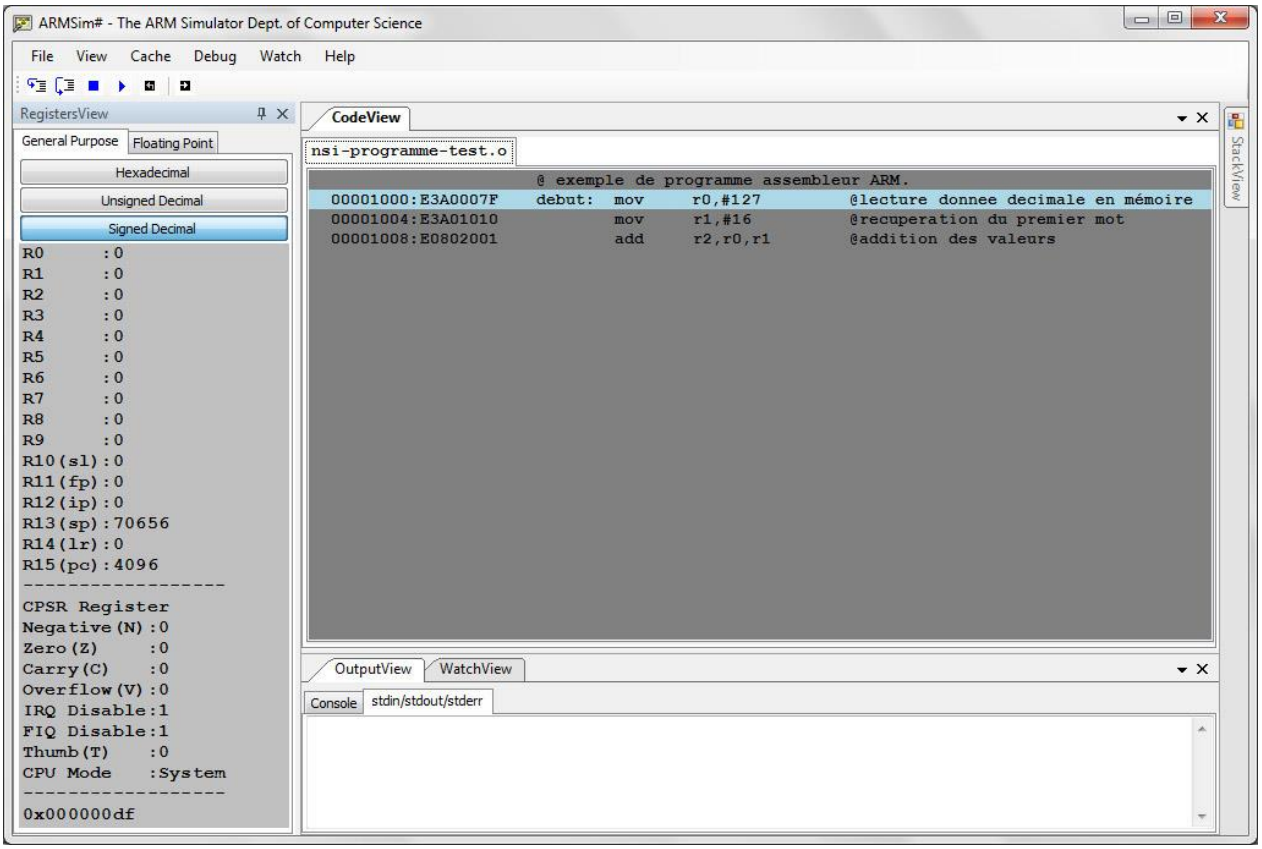
Marche à suivre pour tester un programme :

Etape 1 : saisie du code assembleur dans un éditeur de texte (Notepad++ par exemple)

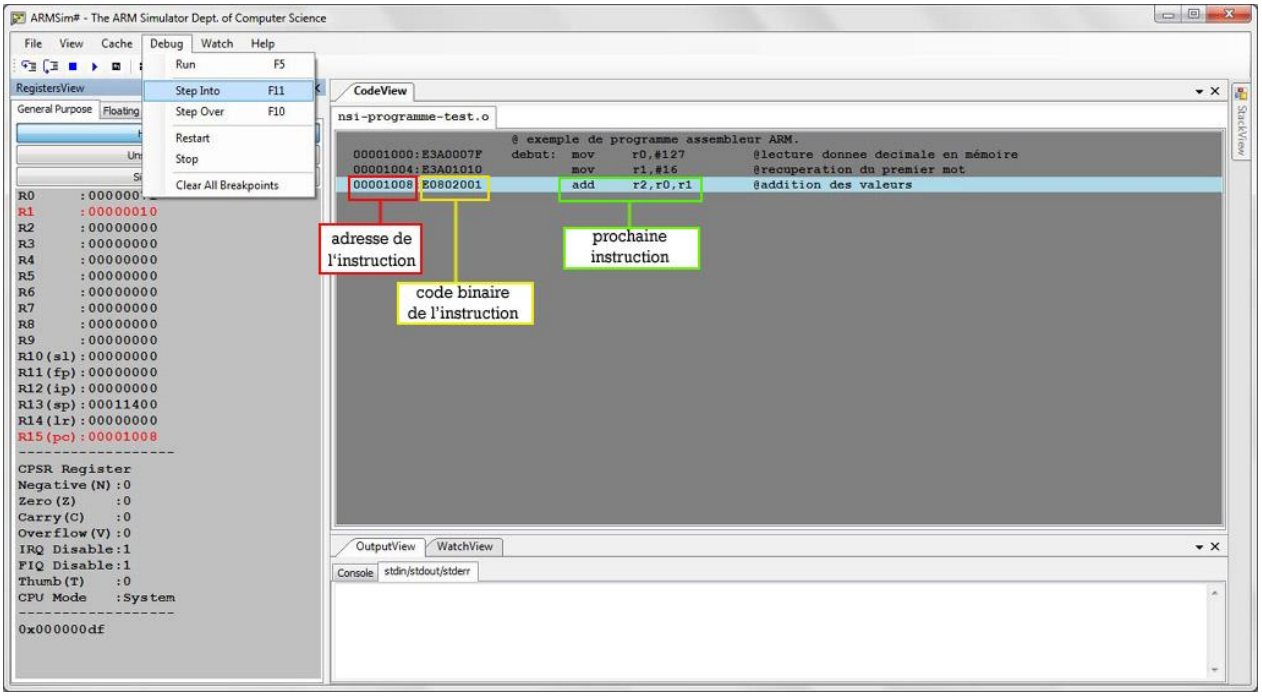
```
@ exemple de programme assembleur ARM.
debut:  mov     r0,#127   @lecture donnee decimale en mémoire
        mov     r1,#16    @recuperation du premier mot
        add     r2,r0,r1  @addition des valeurs
```

L'extension du fichier source doit être .s (nsi-programme-test.s dans notre exemple)

Etape 2 : chargement du programme dans ARMSim# (commande load du menu File)



Etape 3 : Execution du programme, pas à pas (F11) ou jusqu'à l'arrêt (F5)



La colonne de gauche permet de visualiser l'état des registres pendant l'exécution du programme (en hexadecimal ou decimal).

Faire fonctionner le programme d'exemple et observer l'évolution des registres pendant son exécution.

Ecrire le code hexadécimal :

. . . . . .

. . . . . .

. . . . . .

Ecrire le code binaire :

. . . . . .

. . . . . .

. . . . . .

Exercice 1

Faire fonctionner le programme suivant, traduire en langage python et compléter les commentaires.

Que fait ce programme ? Comment évoluent les registres pendant son exécution ?

```
@ ** programme 1 **
  mov   r0,#20    @ ........................................
  mov   r1,r0     @ ........................................
tantque:
  sub   r0,r0,#1  @ ........................................
  add   r1,r1,r0  @ ........................................
  cmp   r0,#0     @ ........................................
  bne   tantque   @ ........................................
```

```
version python :      ........................................
                      ........................................
                      ........................................
                      ........................................
                      ........................................
                      ........................................
                      ........................................
                      ........................................
```

```
Que fait ce programme ? Comment évoluent les registres ?
.............................................................
.............................................................
.............................................................
.............................................................
.............................................................
.............................................................
.............................................................
```

Exercice 2

Ecrire un programme qui calcule le produit de r0 par r1 et range le résultat dans r2, sans utiliser l'instruction de multiplication.

Etudier les deux cas où les deux nombres seraient positifs ou non.