Manipulation des données

Opérations arithmétiques et logiques

```
OPE  r_dest, r_s1, r_s2
```

`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 inversée
`RSC r0,r1,r2`	→	`r0=r2-r1-C+1`	Soustraction inversée avec retenue
`AND r0,r1,r2`	→	`r0=r1&r2`	Et binaire
`ORR r0,r1,r2`	→	`r0=r1|r2`	Ou binaire
`EOR r0,r1,r2`	→	`r0=r1^r2`	Ou exclusif binaire
`BIC r0,r1,r2`	→	`r0=r1&~r2`	Met à `0` les bits de r1 indiqués par r2

Opérations de déplacement de données entre registres

```
OPE r_dest, r_s1
```

`MOV r0,r1`	→	`r0=r1`	Déplacement
`MVN r0,r1`	→	`r0=~r1`	Déplacement et négation

Opérations de décalage

```
OPE r_dest, r_s, r_m
```

`LSL`	→	Décalage logique vers la gauche
`LSR`	→	Décalage logique vers la droite
`ASL`	→	Décalage arithmétique vers la gauche
`ASR`	→	Décalage arithmétique vers la droite
`ROR`	→	Décalage circulaire vers la droite

Opérations de comparaison

`OPE r_s1, r_s2`

`CMP r0,r1`	→ `psr` ⇐ `r0-r1`	Comparer
`CMN r0,r1`	→ `psr` ⇐ `r0+r1`	Comparer à l'inverse
`TST r0,r1`	→ `psr` ⇐ `r0&r1`	Tester les bits indiqués par r1
`TEQ r0,r1`	→ `psr` ⇐ `r0^r1`	Tester l'égalité bit à bit

Ces instructions ne modifient que les bits (N,Z,C,V) du PSR, le résultat n'est pas gardé.

La multiplication

`MUL r0,r1,r2`	→ `r0=r*r2`	multiplication
`MLA r0,r1,r2,r3`	→ `r0=r1+r2*r3`	mult. et accumulation
`MLS r0,r1,r2,r3`	→ `r0=r1-r2*r3`	mult. soustraction
`UMULL r0,r1,r2,r3`	→ `{r1,r0}=r2*r3`	mult. 64bits non signée
`SMULL r0,r1,r2,r3`	→ `{r1,r0}=r2*r3`	mult. 64bits signée
`UMLAL r0,r1,r2,r3`	→ `{r1,r0}+=r2*r3`	MAC 64bits non signée
`SMLAL r0,r1,r2,r3`	→ `{r1,r0}+=r2*r3`	MAC 64bits signée

- Opérations de décalages et rotations

```
MOV r0,r2,LSL #3    @ r0=r2<<3;
MOV r0,r2,LSR #5    @ r0=r2>>5;(1)
MOV r0,r2,ASR #5    @ r0=r2>>5;(2)
MOV r0,r2,ROR #6    @ rotation ...
```

- (1) décalage à droite non signé
- (2) décalage à droite signé

Transfert des données

Instructions pour transférer les données

Deux instructions de transfert de données entre la mémoire et les registres.

- `LDR` : charger un registre avec une donnée en mémoire
- `STR` : enregistrer la valeur du registre en mémoire

`LDR/STR`	:	mots de 32 bits (`words`)
`LDRH/STRH`	:	mots de 16 bits (`half words`)
`LDRB/STRB`	:	mots de 16 bits (`byte`)

Généralement, les adresses **doivent** être alignées :

`LDR/STR`	:	4
`LDRH/STRH`	:	2
`LDRB/STRB`	:	quelconque

Modes d'adressage

- Adressage indirect
  `LDR r0,[r1]` ($r_0 = RAM[r_1]$)
- Adressage indirect avec déplacement (offset)
  `LDR r0,[r1,#8]` ($r_0 = RAM[r_1 + 8]$)
  `LDR r0,[r1,r2]` ($r_0 = RAM[r_1 + r_2]$)
- Adressage indirect avec déplacement et pré-incrémentation
  `LDR r0,[r1,#8]!` ($r_1 = r_1 + 8$ puis $r_0 = RAM[r_1]$)
- Adressage indirect avec déplacement et post-incrémentation
  `LDR r0,[r1],#8` ($r_0 = RAM[r_1]$ puis $r_1 = r_1 + 8$)

Transferts multiples

En plus des instructions `LDR` et `STR` le jeu d'instruction ARM propose les instructions `LDM` et `STM` pour les transferts multiples.

`LDMIA r0,{r1,r2,r3}`	$(r_1 = RAM[r_0])$
	$(r_2 = RAM[r_0 + 4])$
	$(r_3 = RAM[r_0 + 8])$
`STMIA r0,{r1-r3}`	$(RAM[r_0] = r_1)$
	$(RAM[r_0 + 4] = r_2)$
	$(RAM[r_0 + 8] = r_3)$

Il existe 4 suffixes possibles pour les instructions de transferts multiples :

- `IA` pour la post-incrémentation (*Increment After*)
- `IB` pour la pré-incrémentation (*Increment Before*)
- `DA` pour la post-décrémentation (*Decrement After*)
- `DB` pour la pré-décrémentation (*Decrement Before*)

Pour que la valeur du registre d'adresse soit modifiée il faut ajouter (**!**)

```
LDMIA r0!,{r1-r3}
```

Branchements

Il existe deux instructions de branchement :

`B adresse`	Aller à l'adresse
`BX registre`	Aller à l'adresse pointée par le registre et éventuellement changer de mode (ARM/THUMB interworking)

Ces instructions modifient le compteur programme **`"pc" (r15)`**

`BL(X)`	Pour sauvegarder l'adresse de retour dans **`"lr" (r14)`**

L'adresse de retour est celle de l'instruction suivant le `BL`.

- Pour revenir d'un branchement `BL` il suffit de remettre `lr` dans `pc`
  - `BX lr`
  - `MOV pc,lr` (deprecated)
- pour les instructions `B` et `BL` l'adresse est stockée comme un immédiat qui représente un offset par rapport à la position actuelle :
  - l'offset est forcement limité
- pour l'instruction `BX` le mode (ARM/Thumb) est déterminé en fonction du bit de poids faible de l'adresse normalement non utilisé (*voir interworking*)

Exécution conditionnelle

`CMP r0,r1`	comparer r_0 à r_1
`SUBGE r0,r0,r1`	si $r_0 \geq r_1$ alors $r_0 = r_0 - r_1$
`SUBLT r0,r1,r0`	si $r_0 < r_1$ alors $r_0 = r_1 - r_0$
`SUBS r0,r1,r2`	$r_0 = r_1 - r_2$
`BEQ address`	aller à adresse si le résultat est nul

CMP r0,r1
BEQ adress : Branchement si r0=r1
BHI adress :Branchement si r0>r1

Suffix	Description	Flags tested
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Minus	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 or Z=1
GE	Greater or equal	N=V
LT	Less than	N!=V
GT	Greater than	Z=0 & N=V
LE	Less than or equal	Z=1 or N=!V
AL	Always	

Exemple

```
Start:
    MOV  r0,#0  @ mise zero de r0
    MOV  r2,#10 @ charger la valeur 10 dans r2
Loop:
    ADD  r0,r0,r2,LSL #1 @ r0=r0+2*r2
    SUBS r2,r2,#1        @ r2--
    BNE  Loop
    B    Start
```

- Définir une constante symbolique (équivalent au #define du C)

```
.EQU    MACONSTANTE, 0x0200
```

Exemple: Si `MaVar` vaut 4, exécute une tâche:

exemple en C

```
if (MaVar == 4) {
  // exécute une tâche…
}

// le programme continue…
```

assembleur

```
LDR R0, MaVar      ; Met la valeur de MaVar dans R0
CMP R0, 4          ; Change les drapeaux comme R0-4

BNE PasEgal
; execute une tâche…

PasEgal
; le programme continue…
```

exemple en C

```
for (int i = 0; i < 5; ++i) {
  // tâche à l'intérieur de la boucle
}
// le programme continue…
```

assembleur

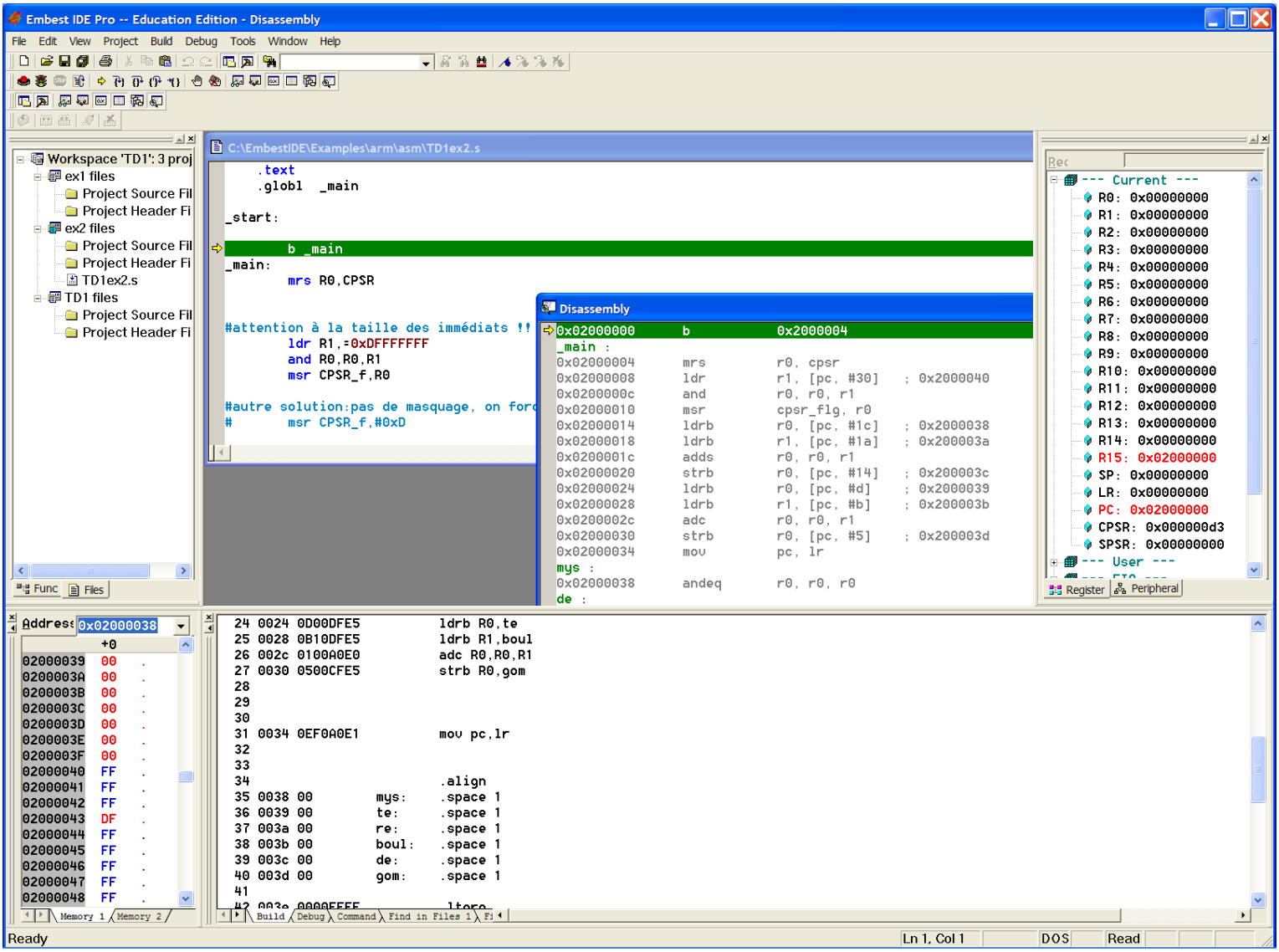
```
InitDeBoucle
    MOV R4,#5

DebutDeBoucle
    ; tâche à l'intérieur de la boucle
    SUBS R4, R4, #1      ; R4 = R4 - 1, change les drapeaux
    BNE DebutDeBoucle    ; Condition d'arrêt

; le programme continue…
```

https://youtu.be/9xKp1RH1y1Y

https://youtu.be/BFMc5Ba9_cQ


Embest IDE Pro -- Education Edition - Disassembly
File Edit View Project Build Debug Tools Window Help
Workspace 'TD1': 3 proj
ex1 files
Project Source Fil
Project Header Fi
ex2 files
Project Source Fil
Project Header Fi
TD1ex2.s
TD1 files
Project Source Fil
Project Header Fi
Func Files
C:\EmbestIDE\Examples\arm\asm\TD1ex2.s
.text
.globl _main
_start:
b _main
_main:
mrs R0,CPSR
#attention à la taille des immédiats !!
ldr R1,=0xDFFFFFFF
and R0,R0,R1
msr CPSR_f,R0
#autre solution:pas de masquage, on forc
msr CPSR_f,#0xD
Disassembly
0x02000000 b 0x2000004
_main :
0x02000004 mrs r0, cpsr
0x02000008 ldr r1, [pc, #30] ; 0x2000040
0x0200000c and r0, r0, r1
0x02000010 msr cpsr_flg, r0
0x02000014 ldrb r0, [pc, #1c] ; 0x2000038
0x02000018 ldrb r1, [pc, #1a] ; 0x200003a
0x0200001c adds r0, r0, r1
0x02000020 strb r0, [pc, #14] ; 0x200003c
0x02000024 ldrb r0, [pc, #d] ; 0x2000039
0x02000028 ldrb r1, [pc, #b] ; 0x200003b
0x0200002c adc r0, r0, r1
0x02000030 strb r0, [pc, #5] ; 0x200003d
0x02000034 mov pc, lr
mys :
0x02000038 andeq r0, r0, r0
de :
--- Current ---
R0: 0x00000000
R1: 0x00000000
R2: 0x00000000
R3: 0x00000000
R4: 0x00000000
R5: 0x00000000
R6: 0x00000000
R7: 0x00000000
R8: 0x00000000
R9: 0x00000000
R10: 0x00000000
R11: 0x00000000
R12: 0x00000000
R13: 0x00000000
R14: 0x00000000
R15: 0x02000000
SP: 0x00000000
LR: 0x00000000
PC: 0x02000000
CPSR: 0x000000d3
SPSR: 0x00000000
--- User ---
Register Peripheral
Address: 0x02000038
+0
02000039 00
0200003A 00
0200003B 00
0200003C 00
0200003D 00
0200003E 00
0200003F 00
02000040 FF
02000041 FF
02000042 FF
02000043 DF
02000044 FF
02000045 FF
02000046 FF
02000047 FF
02000048 FF
Memory 1 Memory 2
24 0024 0D00DFE5 ldrb R0,te
25 0028 0B10DFE5 ldrb R1,boul
26 002c 0100A0E0 adc R0,R0,R1
27 0030 0500CFE5 strb R0,gom
31 0034 0EF0A0E1 mov pc,lr
34 .align
35 0038 00 mys: .space 1
36 0039 00 te: .space 1
37 003a 00 re: .space 1
38 003b 00 boul: .space 1
39 003c 00 de: .space 1
40 003d 00 gom: .space 1
Build Debug Command Find in Files 1
Ready
Ln 1, Col 1
DOS
Read