

Annexes

Tous les registres du processeurs ont 32 bits de large : chaque registre stocke un mot binaire de 4 octets.

Registres d'usage général (calculs, résultats intermédiaires, compteurs, pointeurs ...)

r0* r1* r2* r3* r4 r5 r6 r7 r8 r9 r10 r11 r12*

* *Scratch registers* : peuvent être modifiés (sans sauvegarde/restauration) par un sous programme
r0, r1, r2 et r3 : utilisés pour passer les n 1ers arguments (n<=4) lors d'un appel de sous programme
r0 : utilisé pour passer la valeur de retour à la fin d'un appel de sous programme

Indicateurs de code condition : 4 bits désignés par NZCV

Indicateur Z (Zero) : Z=1 indique une opération dont le résultat est nul ou une égalité lors d'une comparaison

Sauf pour les instructions *compare*, il faut explicitement préciser **S (Set)** en postfixe pour qu'une instruction modifie les indicateurs de code condition en fonction de son résultat. Par défaut les codes conditions ne sont pas modifiés par une instruction de calcul.

SUB r1,r1,#1 @ Décrémente r1 mais ne modifie pas les indicateurs
SUBS r0,r0,#1 @ Décrémente r0 et met à jour les indicateurs selon la valeur résultante
@ (en particulier si r0 vaut 0 alors Z=1 sinon Z=0)

Instruction CMP et exécution conditionnelle

L'instruction *cmp* réalise la comparaison de deux opérandes en faisant une soustraction, mais aucun registre destination ne reçoit le résultat. Les indicateurs NZCV sont mis à jour.

En particulier si les 2 valeurs sont égales, l'indicateur Z passe à 1, dans le cas contraire il passe à 0.

On peut conditionner l'exécution d'une instruction à l'état des indicateurs code condition en la postfixant par EQ, NE ... :

postfixe	exécute si	description
EQ	Z=1	EQual
NE	Z=0	Not Equal

Exemples :

CMP r1,r2 @ Comparaison r1 - r2 : indicateurs code condition mis à jour
ADDEQ r0,r0,#1 @ N'incrémente r0 que si la comparaison précédente a indiqué r1==r2 (ADD if EQual)

CMP r0,#20 @ Comparaison r1 - 20 : indicateurs code condition mis à jour
BNE Boucle @ N'effectue le branchement que si le cmp a indiqué r1!=20 (Branch if Not Equal)

Instructions usuelles : exemples d'utilisation. A droite du @ en commentaire le code C équivalent.

MOV r0,r3 @ r0=r3;
MOV r1,r4 @ r1=r4;
MOV r3,#15 @ r3=15;
MOV r4,#0x2E @ r4=0x2E;

Attention: contraintes sur les valeurs immédiates #VAL : de la forme [0 ... 255] * 2²ⁿ (0≤n<12)
Pour initialiser la valeur d'un registre sans contrainte particulière utiliser la syntaxe suivante :

LDR r0,#3141593 @ r0=3141593;
LDR r0,#3*8+43; @ r0=3*8+43; // est possible de faire un calcul (avec des constantes)

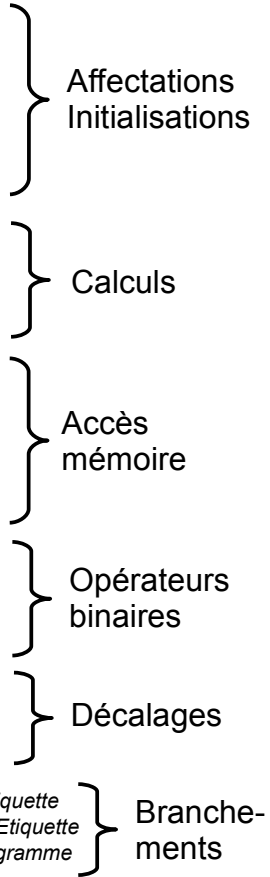
ADD r0,r1,r2 @ r0=r1+r2;
SUB r0,r1,r2 @ r0=r1-r2;
RSB r0,r1,r2 @ r0=r2-r1;
MUL r0,r1,r2 @ r0=r1*r2;

LDR r0,[r1] @ r0 prend la valeur du mot mémoire pointé par r1
LDRH r0,[r1] @ r0 prend la valeur du demi-mot (2 octets) mémoire pointé par r1
LDRB r0,[r1] @ r0 prend la valeur de l'octet mémoire pointé par r1
STR r0,[r1] @ Le mot mémoire pointé par r1 prend la valeur r0
STRH r0,[r1] @ Le demi-mot (2 octets) mémoire pointé par r1 prend la valeur r0
STRB r0,[r1] @ L'octet mémoire pointé par r1 prend la valeur r0

AND r0,r1,r2 @ r0=r1&r2; ET
ORR r0,r1,r2 @ r0=r1|r2; OU
EOR r0,r1,r2 @ r0=r1^r2; OU Exclusif
MVN r0,r2 @ r0=~r2; NON

MOV r0,r2,LSL #3 @ r0=r2<<3; Logical Shift Left : décalage binaire à gauche
MOV r0,r2,LSR #5 @ r0=r2>>5; Logical Shift Right : décalage binaire à droite
ADD r0,r1,r2,LSL #3 @ r0=r1+(r2<<3); Décalage et opération arithmétique ou binaire

B monEtiquette @ Branchement non conditionnel : l'exécution reprend au niveau de monEtiquette
BNE autreEtiquette @ Branchement conditionnel : si indicateur Z est à 0 alors exécuter à autreEtiquette
BX lr @ Branchement spécial de retour à l'appelant : marque la fin d'un sous-programme



Exemples de codes assembleurs et sous-programmes

Correspondances		
Décimal	Binaire	Hexa
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Boucle "répéter n fois"

En assembleur pour obtenir une boucle "répéter n fois" il est plus simple d'avoir un compteur qui décompte.
Exemple : répéter 10 fois

```
LDR  r0,=10          @ initialiser compteur à 10 (si nécessaire)

maBoucle:           @ étiquette de début
-----
-----           @ code à répéter 10 fois
-----
SUBS  r0,r0,#1      @ décrémenter compteur...
BNE  maBoucle      @ boucler si compteur ≠ 0

-----           @ code après la boucle...
-----
```

Sous-programme asmMemset : remplit une zone mémoire avec une valeur constante

```
// PROTOTYPE ET APPEL DEPUIS LE C
// s adresse du début de zone mémoire
// c valeur de remplissage
// n nombre d'octets consécutifs à remplir
void asmMemset(char *s, char c, int n);

int main(){
    char tab[3];
    asmMemset(tab,65,3); // Exemple : init des 3 cases avec valeur 65
    ...

@ SOUS-PROGRAMME ASSEMBLEUR
asmMemset: @ Début du sous-programme
           @ r2 arrive déjà initialisé, on attaque directement la boucle "répéter r2 fois"
msBoucle: @ Début de boucle

        STRB  r1,[r0] @ Mettre r1 (valeur de remplissage) à l'adresse pointée par r0
        ADD  r0,r0,#1 @ Pointer case suivante

        SUBS  r2,r2,#1 @ Gestion boucle
        BNE  msBoucle

        BX   lr      @ Fin du sous-programme, retour à l'appelant
```

Sous-programme asmSegment : dessine un segment horizontal de 10 pixels

```
@ Sous-programme ne prenant pas de paramètre
@ Dessine un segment blanc de 10 pixels de long en partant du centre de l'écran (vu au TP2)

asmSegment: @ Début du sous-programme

LDR  r2,=0x06000000+2*120+480*80 @ Adresse du pixel de départ : milieu de l'écran
LDR  r3,=0x7FFF @ Couleur des pixels à colorier
LDR  r0,=10 @ Compteur de boucle "répéter 10 fois"

BoucleSeg: @ Etiquette de début de boucle

        STRH  r3,[r2] @ Colorier le pixel courant
        ADD  r2,r2,#2 @ Adresse du pixel suivant (un pixel à droite)

        SUBS  r0,r0,#1 @ gestion boucle
        BNE  BoucleSeg

        BX   lr      @ Fin du sous-programme, retour à l'appelant
```

Sauver et restaurer les registres modifiés (ne concerne pas r0 à r3)

```
@ Sous-programme devant modifier les valeurs de r0,r1,r2,r3,r4,r5,r6

asmSousProg:
    PUSH  {r4,r5,r6} @ Sauver les registres modifiés (sauf r0 à r3)
    ... @ Corps du sous programme (r0 à r6 peuvent être modifiés)
    POP  {r4,r5,r6} @ Restaurer les registre
    BX   lr @ Fin du sous-programme, retour à l'appelant
```