Chapitre 5 :
Jeu d'instructions des processeurs tms320c6x

Chapitre 5 :
Jeu d'instructions des processeurs tms320c6x
Instruction set

I. Introduction

Les algorithmes du traitement numérique du signal partagent des opérations communes :

- Le produit ou la multiplication d'un nombre important d'opérations de calculs.
- La sommation d'un nombre important d'éléments
- L'Accomplissement d'une suite d'opérations d'accumulation en somme de produits.

Les constructeurs des circuits DSP ont développé des architectures matérielles dédiées à ce genre de calcul et qui permettent l'exécution efficace de ces algorithmes en temps réel .Nous développons dans ce cours le jeu d'instruction des processeurs tms320c6x de Texas Instrument.

Les trois séries des processeurs « TMS320C62x, TMS320C64x et le TMS320C67x » partagent le même jeu d'instructions. Toutes les instructions valables pour le C62x sont également valables pour le C64x et le C67x. Cependant, comme le C67x est un processeur à virgule flottante, certaines instructions lui sont propres et ne s'exécutent pas sur les processeurs à virgule fixe. Dans ce chapitre on décrit les instructions du langage d'assemblage communes aux processeurs de signaux numériques C62x, C64x et C67x. Les opérations parallèles, les opérations conditionnelles, et les modes d'adressage sont également décrits.

II. Mode d'adressage :

Les modes d'adressage spécifient les opérandes dans une instruction et déterminent la manière avec laquelle les accès relatifs à ces opérandes seront effectués par le processeur. Il y'a deux modes d'adressage qui sont supportés par la famille des processeurs tms320c6x :L'adressage linéaire ou indirect et l'adressage circulaire. Le mode le plus utilisé est le mode d'adressage indirect de la mémoire. Le registre « AMR, Address Mode Register » du processeur permet de choisir l'un de ces deux modes **à utiliser.**

Tous les registres peuvent être utilisés en mode d'adressage linéaire. Seuls huit registres peuvent effectuer un adressage circulaire:

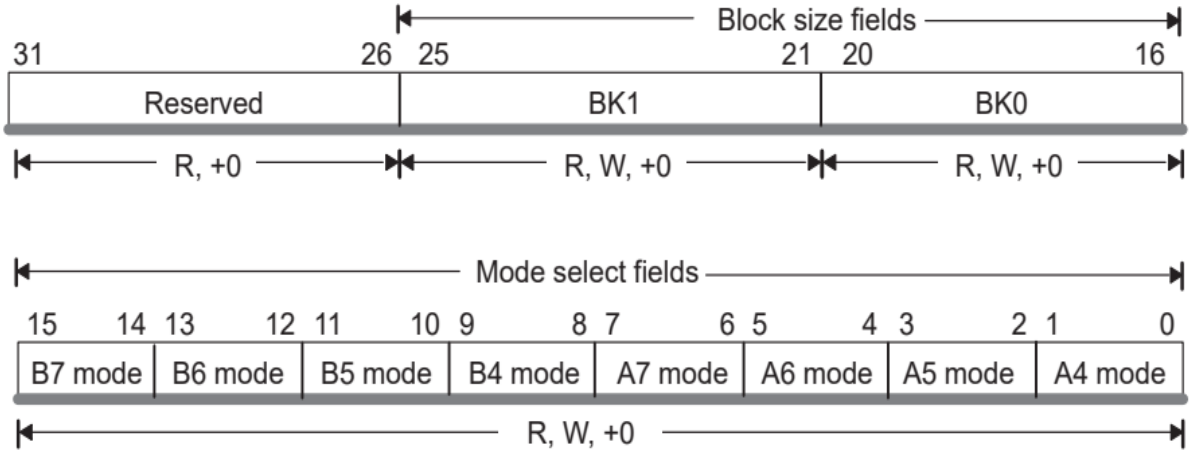
- **A4 – A7 sont utilisés par l'unité .D1 et**
- **B4 – B7 sont utilisés par l'unité .D2.**

Aucune autre unité ne peut effectuer d'adressage circulaire.

Mode		Description
0	0	Linear modification (default at reset)
0	1	Circular addressing using the BK0 field
1	0	Circular addressing using the BK1 field
1	1	Reserved

Description du **mode** dans le registre AMR

Le registre AMR dispose d'un champ à deux bits pour que chacun des huits registres ci-dessus mentionnés sélectionne l'un des deux modes appropriés. L'adressage linéaire est l'option par défaut. Dans le mode d'adressage circulaire, la spécification de la taille du bloc employé est nécessaire dans les registres de banques BK0 et BK1 du registre AMR. La Figure 5 illustre les champs de sélection ainsi que leur taille. Le tableau IV décrit la configuration adéquate au mode d'adressage désiré.



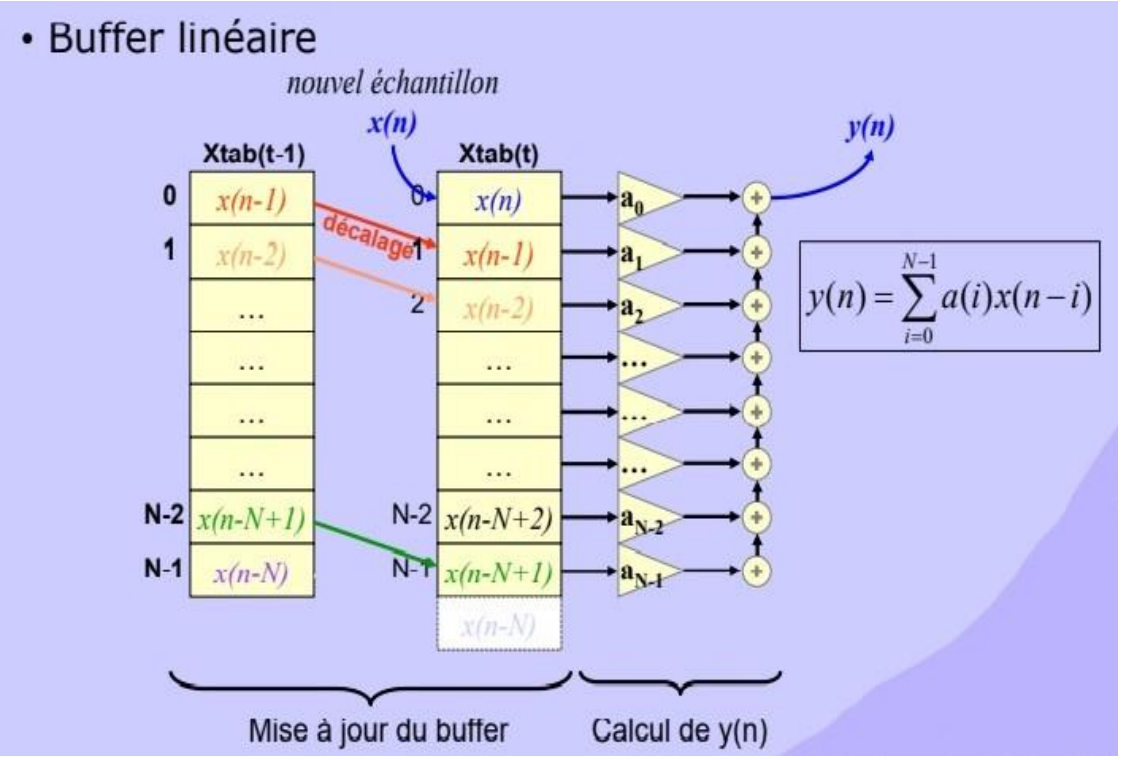
Le registre AMR

II.1- **L'adressage LINIAIRE**

Le mode d'adressage indirect permet d'accéder à une donnée par l'intermédiaire d'un registre qui contient son adresse. L'adressage indirect peut être utilisé avec ou sans déplacement. Le mode d'adressage indirect utilise un «*» conjointement avec l'un des 32 registres. Pour illustrer, considérons R comme un registre d'adresses. Le registre R représente l'un des 32 registres A0 à A15 et B0 à B15. Le registre R est un pointeur d'adresse d'une case mémoire adressable.

La notation employée dans l'adressage indirect suppose les abréviations suivantes :

a. *R est un registre contenant l'adresse de l'espace mémoire où la valeur de la donnée est stockée.
b. *R + +(d) suppose que *R est un registre contenant l'adresse de l'espace mémoire où la valeur de la donnée est stockée. Après que l'adresse mémoire ait été utilisée, la valeur du registre R est post incrémentée de telle sorte que la nouvelle adresse s'obtient en incrémentant l'adresse courante d'une valeur d. Par défaut, la valeur de d est unitaire. A la place d'une double addition, une double soustraction (—) suscite une post décrémentation de l'adresse courante à l'adresse R - d.
c. * + +R (d) préconise une pré incrémentation de d pour que l'adresse courante soit R+ d. Un double signe moins suggère une pré décrémentation de l'adresse mémoire pour avoir une adresse courante à l'emplacement R — d.
d. *+ R(d) implique une pré incrémentation de d pour avoir l'adresse courante à R+d, sans toutefois modifier le contenu de R. Un raisonnement semblable s'applique pour une pré décrémentation.



II.2- L'adressage circulaire :

Les algorithmes de filtrage numériques ou de corrélation, à titre d'exemple, en traitement du signal, nécessitent une mise à jour constante des données. Cela peut se réaliser soit d'un maniéré software ou d'une manière matérielle Hardware. L'inconvénient majeur de la première solution est la consommation énorme du temps de calcul ce qui limite directement la bande passante du système et diminue les performances du temps réel. La famille C6000 possède un espace mémoire alloué à l'adressage de type circulaire.

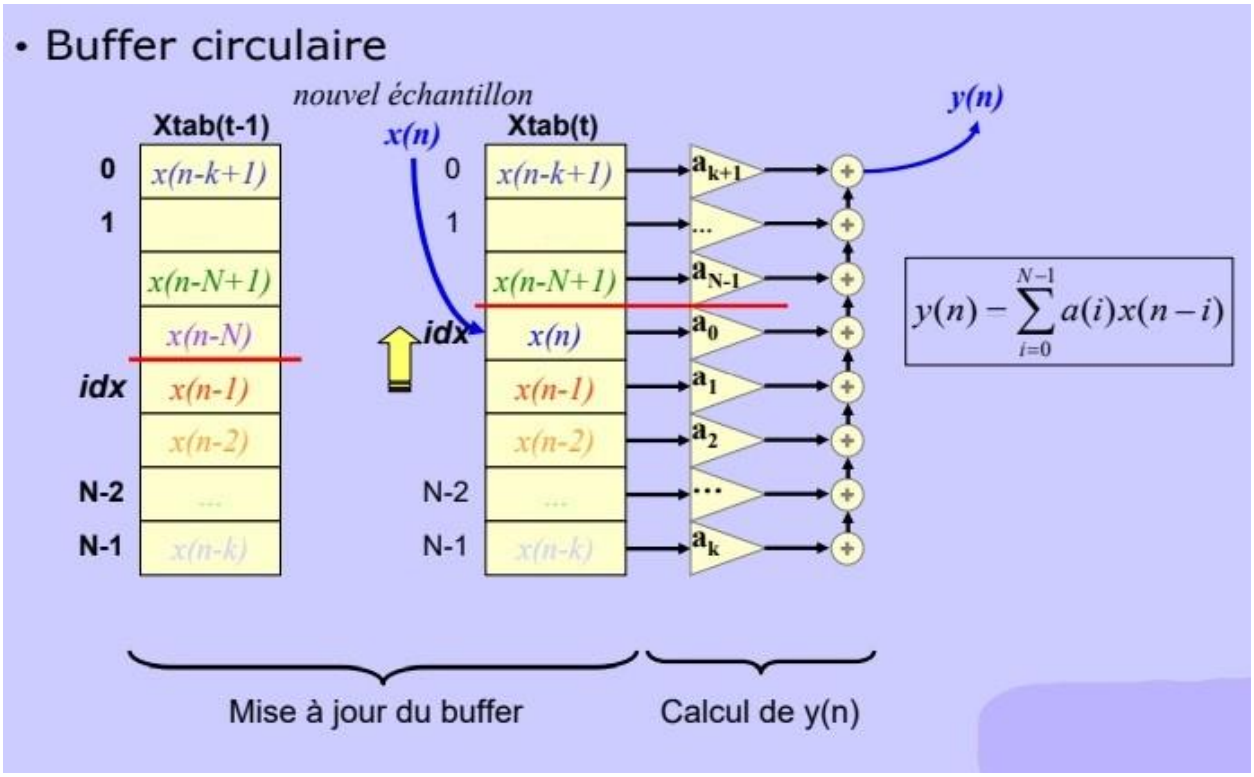
L'adressage circulaire est utilisé pour créer un tampon circulaire. Ce tampon Hardware est créé dans le matériel dans une zone de mémoire du processeur il permet de mettre à jour les échantillons en effectuant une rotation des données sans dépassement.

Les instructions:

LDB(U) /LDH(U) /LDW, STB /STH /STW, ADDAB /ADDAH /ADDAW /ADDAD et SUBAB /SUBAH /SUBAW utilisent le registre **AMR** pour déterminer le calcul à effectuer pour obtenir l'adresse de la mémoire.

Les principaux Objectifs de l'adressage circulaire sont donc :

- ✓ **Réduire les accès mémoire par la création d'un** Buffer circulaire : Segment mémoire contenant des données(échantillons, cœfficients…)
- ✓ Accès par pointeurs générés et Incrémentés automatiquement
- ✓ **Pas de test sur les pointeurs**
- ✓ **Pas de reset nécessaire**



Array Index	Filter Coefficient Array `h[]`	Circular Buffer Array `xcirc[]`
0	$h[0]$	$x[n - newest]$
1	$h[1]$	$x[n - newest + 1]$
⋮	⋮	⋮
		$x[n - 1]$
$newest$		$x[n]$
$oldest$		$x[n - N + 1]$
		$x[n - N + 2]$
⋮	⋮	⋮
$N - 2$	$h[N - 2]$	$x[n - newest - 2]$
$N - 1$	$h[N - 1]$	$x[n - newest - 1]$

III.Format d'une instruction :

Le format d'une instruction assembleur de la famille des tms320c6x est représenté comme suit :

Label || [ ] Instruction Unit Operands ;comments

Label :

Une étiquette, si elle est présente, représente une adresse ou un emplacement mémoire spécifique qui contient une instruction ou des données. L'étiquette doit figurer dans la première colonne.

|| :

Les barres parallèles (||) sont utilisées si l'instruction est exécutée en parallèle avec l'instruction précédente.

[Registre] :

Le champ suivant est facultatif .Il est utilisé pour rendre l'instruction associée conditionnelle. Cinq registres - A1, A2, B0, B1 et B2 - sont disponibles pour une utilisation conditionnelle sur registres. Toutes les instructions des processeurs C6x peuvent être conditionnées avec les registres A1, A2, B0,B1 et B2 en déterminant quand le registre conditionnel est nul

Instruction :

C'est la mnémonique ou l'assembleur du code de l'instruction

Unit :

Le champ Unité, peut être l'un des huit unités centrales.

Comments :

Les commentaires : seules les commentaires de la première colonne peuvent commencer par un astérisque ou un point-virgule, tandis que les commentaires commençant dans toute autre colonne doivent commencer par un point-virgule.

IV. Type d'instruction :

La famille des dsp TMS320C6x offre un langage assembleur très riche. Dans ce qui suit-on illustre une partie de la syntaxe du code assembleur de cette famille. par la présentation de quelques instructions typiques.

1. **Add/Subtract/Multiply**

(a) The instruction:

```
ADD .L1 A3,A7,A7          ;add A3 + A7 --->A7 (accum in A7)
```

Adds the values in registers A3 and A7 and places the result in register A7.

(b) The instruction:

```
SUB .S1 A1,1,A1           ;subtract 1 from A1
```

Subtracts 1 from A1 to decrement it, using the .S unit.

(c) The parallel instructions

```
        MPY .M2 A7,B7,B6  ;multiply 16 LSBs of A7,B7 ---> B6
||      MPYH .M1 A7,B7,A6 ;multiply 16 MSBs of A7,B7 ---> A6
```

2. Load/Store

(a) The instruction

```
LDH .D2 *B2++,B7    ;load (B2) --->B7, increment B2
||  LDH .D1 *A2++,A7    ;load (A2) --->A7, increment A2
```

Loads into B7 the half-word (16 bits) whose address in memory is specified/pointed by B2.
Then register B2 is incremented (post incremented) to point at the next-higher memory address.
In parallel is another indirect addressing mode instruction to load into A7 the content in memory, whose address is specified by A2. Then A2 is incremented to point at the next higher memory address.

(b) The instruction

```
STW .D2 A1,*+A4[20]    ;store A1Æ(A4) offset by 20
```

Stores the 32-bit word A1 into memory whose address is specified by A4 offset by 20 words (32 bits) or 80 bytes. The adddress register A4 is preincremented with offset, but it is not modified (two plus signs are used if A4 is to be modified).

3. **Branch/Move**.

The following code segment illustrates branching and data transfer.

```
Loop    MVK .S1 x,A4 ;move 16 LSBs of x address ÆA4
        MVKH .S1 x,A4 ;move 16 MSBs of x address ÆA4
.       ……..
.
        SUB .S1 A1,1,A1          ;decrement A1
[A1]    B .S2 Loop               ;branch to Loop if A1 # 0
        NOP 5                    ;five no-operation instructions
        STW .D1 A3,*A7           ;store A3 into (A7)
```

The first instruction moves the lower 16 bits (LSBs) of address x into registerA4. The second instruction moves the higher 16 bits (MSBs) of address x into TMS320C6x Instruction Set 73 A4, which now contains the full 32-bit address of x. One must use the instructions MVK/MVKH in order to get a 32-bit constant into a register.

Register A1 is used as a loop counter. After it is decremented with the SUB instruction, it is tested for a conditional branch. Execution branches to the label or address loop if A1 is not zero. If A1 = 0, execution continues and data in register A3 are stored in memory whose address is specified (pointed) by A7.

V. Instruction par unité fonctionnelle :

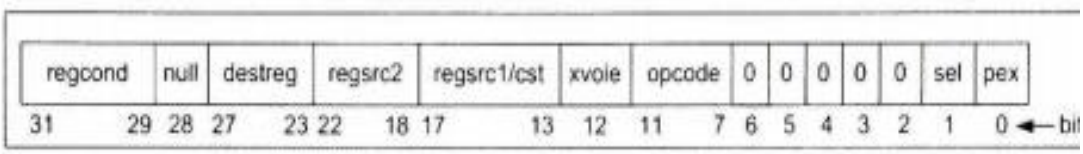
Dans cette partie du cours on présente :

- ✓ Le format de codage des instructions sur 32 bits de chaque unité fonctionnelle.
- ✓ Les instructions exécutées par chaque unité fonctionnelle.

Les abréviations qui seront utilisés par la suite ont les significations suivantes :

- regcond : champ à 3 bits spécifiant un registre conditionnel.
- null : test la présence d'une valeur nulle.
- destreg : registre de destination.
- regsrcl : registre source 1.
- regsrc2 : registre source 2.
- esta : constante a.
- cstb : constante b.
- est : constante.
- pex : exécution en parallèle.
- opcode : champ pour lequel l'opcode spécifie une instmction unique.
- sel : sélection de la voie A ou B pour la destination.
- xvoie : chemin croisé pour regsrc2.

1. L'unité .M :

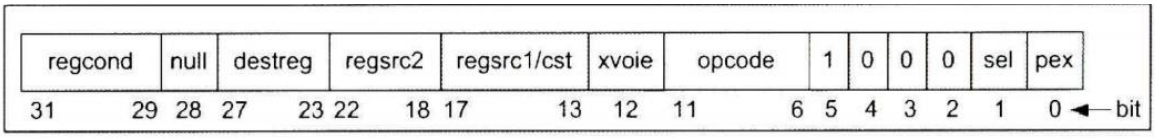


Format des instructions dans l'unité M.

Quelques instructions de l'unité M

Instruction	Délai	Description
MPY(U)	2	Multiplication signée (non signée) 16 bits LSB × 16 bits LSB
MPYDP	10	Multiplication en précision double
MPYH(U)	2	Multiplication signée (non signée) 16 bits MSB × 16 bits MSB
MPYHL(U)	2	Multiplication signée (non signée) 16 bits MSB × 16 bits LSB
MPYI	9	Multiplication sur 32 bits avec résultat sur moins de 32 bits
MPYID	10	Multiplication sur 32 bits avec résultat sur 64 bits
MPYLH	2	Multiplication signée (non signée) 16 bits LSB × 16 bits MSB
MPYSP	4	Multiplication en précision simple

2. L'unité .S :

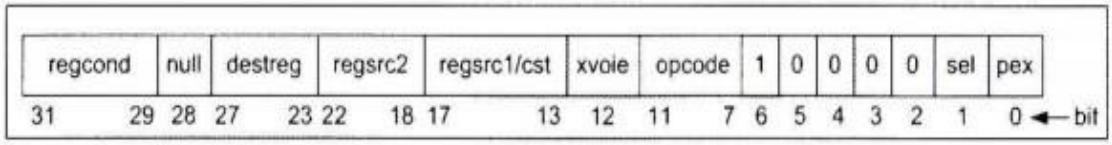


Format des instructions dans l'unité S.

Quelques instructions de l'unité S

Instruction	Délai	Description
ABSDP	2	Valeur absolue des nombres en précision double
ABSSP	1	Valeur absolue des nombres en précision simple
ADD	1	Addition non saturée signée en point fixe
ADDK	1	Addition signée en utilisant une constante sur 16 bits
AND	1	Et logique
B	6	Branchement
CMPEQDP	2	Égalité en précision double
CMPEQSP	1	Égalité en précision simple
CMPGTDP	2	Inégalité supérieure en précision double
CMPGTSP	1	Inégalité supérieure en précision simple
CMPLTDP	2	Inégalité inférieure en précision double
CMPLTSP	1	Inégalité inférieure en précision simple
EXT	1	Extraction et extension de signe
MV	1	Déplacement d'une registre à l'autre
MVK	1	Déplacement d'une constante de 16 bits signée et extension de signe
MVKH	1	Déplacement d'une constante de 16 bits dans les MSB d'un registre
NOT	1	Non logique
OR	1	Ou logique
RCPDP	2	Approximation réciproque en précision double (division)
RCPSP	1	Approximation réciproque en précision simple (division)
RSQRDP	2	Approximation réciproque de la racine carrée en précision double
RSQRSP	1	Approximation réciproque de la racine carrée en précision simple
SPDP	2	Conversion précision double en précision simple
SPINT	2	Conversion précision simple à entier
SET	1	Mise à un
SHL(R)	1	Décalage à gauche (droite)
SSHL	1	Décalage à gauche avec saturation
STB	1	Stockage d'une valeur en bytes
STH	1	Stockage d'une valeur sur 16 bits
STW	1	Stockage d'une valeur sur 32 bits
SUB(U)	1	Soustraction non saturée signée (non signée) en point fixe
XOR	1	Ou exclusif
ZERO	1	Mise à zero

3. L’unité .D :

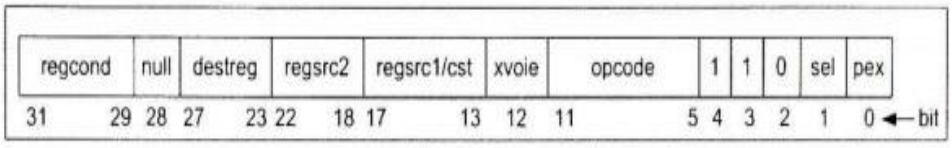


Format des instructions dans l’unité D.

Quelques instructions de l’unité D

Instruction	Délai	Description
ADD(U)	1	Addition non saturée signée (non signée) en point fixe
ADDAB	1	Addition en bytes
ADDAH	1	Addition sur 16 bits
ADDAW	1	Addition sur 32 bits
LDB	5	Chargement en bytes
LDH	5	Chargement sur 16 bits
LDDW	5	Chargement sur 32 bits d’un mot en précision double
LDW	5	Chargement sur 32 bits
MV	1	Déplacement d’une registre à l’autre

4- L’unité .L :



Format des instructions dans l’unité L.

Quelques instructions de l'unité L

Instruction	Délai	Description
ABS	1	Valeur absolue avec saturation
ADD(U)	1	Addition non saturée signée (non signée) en point fixe
ADDDP	7	Addition en précision double
ADDSP	4	Addition en précision simple
AND	1	Et logique
CMPEQ	1	Égalité en point fixe
CMPGT(U)	1	Inégalité supérieure signée (non signée) en point fixe
CMPLT(U)	1	Inégalité inférieure signée (non signée) en point fixe
DPTRUNC	4	Conversion précision double vers entier avec troncature
MV	1	Déplacement d'un registre à l'autre
NEG	1	Négation en point fixe
NOT	1	Non logique
OR	1	Ou logique
SADD	1	Addition d'entiers avec saturation
SAT	1	Saturation en point fixe
SPTRUNC	4	Conversion précision simple vers entier avec troncature
SUB(U)	1	Soustraction non saturée signée (non signée) en point fixe
SUBDP	7	Soustraction en précision double
SUBSP	4	Soustraction en précision simple
XOR	1	Ou exclusif
ZERO	1	Mise à zero

VI.Références bibliographiques:

1. R. Chassaing, D. Reay, Digital Signal Processing and Applications with the TMS320C6713 and
TMS320C6416 DSK, John Wiley & Sons, 2008.

2. Caroline Petitjean université de Rouens cour s dsp2006/2007

4. Steven A Tretter, Communication System Design Using DSP Algorithms, Springer 2008.

5. N. Dahnoun, Digital Signal Processing Implementation using the TMS320 C6000 DSP platform, Prantice Hall, 2000.

6. N. Kehtarnaz, N. Kim, Real Time Digital Signal Processing Based on TMS320C6000, Newnes, 2004.

7. N. Kehtarnaz, M. Keramat, DSP System Design using TMS320C6000, Prentice Hall, 2006.