TD4 : Pipeline logiciel avec TMS320C6x

Introduction

L'annexe donne le schéma fonctionnel du processeur VLIW TMS320C62 de Texas Instruments, avec les instructions réparties par unité fonctionnelle et les latences des instructions. Dans un exercice, on utilisera le TMS320C67, qui a les mêmes instructions entières et a en plus des instructions flottantes dont la répartition dans les unités fonctionnelles et les latences sont également fournies en annexe.

Somme et produit scalaire flottant

Examiner l'exécution du code ci-dessous, dans lequel on suppose que sum a été initialisé à 0 :

```
LOOP:       ADDSP    x,sum,sum
  ||        LDW      *xptr++,x
  ||[cond]  B        cond
  ||[cond]  SUB      cond,1,cond
```

En utilisant le pipeline logiciel, écrire le code assembleur pour la fonction dotp "flottante". Quel est le temps d'exécution de la fonction ?

```
float dotp(float a[], float b[])
{
   int i;
   float sum;
   sum = 0;

   for(i=0; i<100; i++)
      sum += a[i] * b[i];

   return(sum);
}
```

Fonction IIR.

En utilisant le pipeline logiciel, écrire le code assembleur pour la fonction IIR. Quel est le temps d'exécution de la fonction ?

```
void iir(short x[],short y[],short c1, short c2, short c3)
{
  int i;

  for (i=0; i<100; i++) {
      y[i+1] = (c1*x[i] + c2*x[i+1] + c3*y[i]) >> 15;
      }
}
```

If then else

En utilisant le pipeline logiciel, écrire le code assembleur pour la fonction if_then. Quel est le temps d'exécution de la fonction ?

```
int if_then(short a[], int codeword, int mask, short theta)
{
  int i,sum, cond;

  sum = 0;
  for (i = 0; i < 32; i++){
       cond = codeword & mask;
       if (theta  ==  !(!(cond)))
           sum += a[i];
       else
           sum -= a[i];
       mask = mask << 1;
       }
  return(sum);
}
```

Fonction live_long

En utilisant le pipeline logiciel, écrire le code assembleur pour la fonction live_long. Quel est le temps d'exécution de la fonction ?

```
int live_long(short a[],short b[],short c, short d, short e)
{
  int i,sum0,sum1,sum,a0,a2,a3,b0,b2,b3;
  short a1,b1;

  sum0 = 0;
  sum1 = 0;
  for(i=0; i<100; i++){
       a0 = a[i] * c;
       a1 = a0 >> 15;
       a2 = a1 * d;
       a3 = a2 + a0;
       sum0 += a3;
       b0 = b[i] * c;
       b1 = b0 >> 15;
       b2 = b1 * e;
       b3 = b2 + b0;
       sum1 += b3;
       }
  sum = sum0 + sum1;
  return(sum);
}
```

Annexe

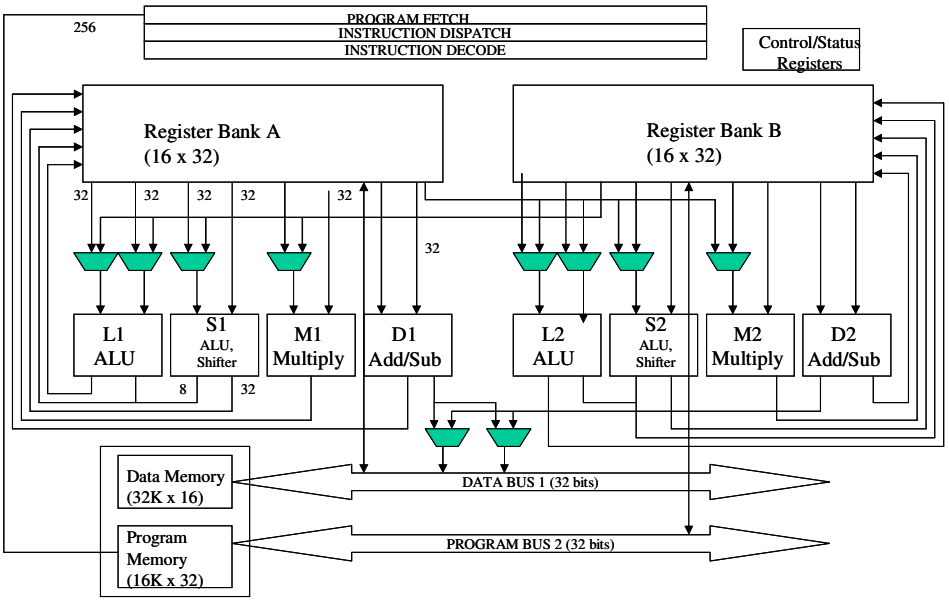

Figure 1 : Schéma fonctionnel du TMS320C62

.L Unit	.M Unit	.S Unit		.D Unit	
ABS	MPY	ADD	SET	ADD	STB (15-bit offset)‡
ADD	MPYU	ADDK	SHL	ADDAB	STH (15-bit offset)‡
ADDU	MPYUS	ADD2	SHR	ADDAH	STW (15-bit offset)‡
AND	MPYSU	AND	SHRU	ADDAW	SUB
CMPEQ	MPYH	B disp	SSHL	LDB	SUBAB
CMPGT	MPYHU	B IRP†	SUB	LDBU	SUBAH
CMPGTU	MPYHUS	B NRP†	SUBU	LDH	SUBAW
CMPLT	MPYHSU	B reg	SUB2	LDHU	ZERO
CMPLTU	MPYHL	CLR	XOR	LDW	
LMBD	MPYHLU	EXT	ZERO	LDB (15-bit offset)‡	
MV	MPYHULS	EXTU		LDBU (15-bit offset)‡	
NEG	MPYHSLU	MV		LDH (15-bit offset)‡	
NORM	MPYLH	MVC†		LDHU (15-bit offset)‡	
NOT	MPYLHU	MVK		LDW (15-bit offset)‡	
OR	MPYLUHS	MVKH		MV	
SADD	MPYLSHU	MVKLH		STB	
SAT	SMPY	NEG		STH	
SSUB	SMPYHL	NOT		STW	
SUB	SMPYLH	OR			
SUBU	SMPYH				
SUBC					
XOR					
ZERO					

† S2 only
‡ D2 only

Tableau 1 : Répartition des instructions entières dans les unités fonctionnelles

Instruction Type	Delay Slots
NOP (no operation)	0
Store	0
Single cycle	0
Multiply (16 × 16)	1
Load	4
Branch	5

Tableau 2 : Délai des instructions entières (latence = 1+délai)

.L Unit	.M Unit	.S Unit	.D Unit
ADDDP	MPYDP	ABSDP	ADDAD
ADDSP	MPYI	ABSSP	LDDW
DPINT	MPYID	CMPEQDP	
DPSP	MPYSP	CMPEQSP	
DPTRUNC		CMPGTDP	
INTDP		CMPGTSP	
INTDPU		CMPLTDP	
INTSP		CMPLTSP	
INTSPU		RCPDP	
SPINT		RCPSP	
SPTRUNC		RSQRDP	
SUBDP		RSQRSP	
SUBSP		SPDP	

Tableau 3 : Répartition des instructions flottantes.

INSTRUCTION	DELAI	LATENCE
LDDW	4	5
MPYSP	3	4
ADDSP	3	4

Tableau 4 : Délai des instructions flottantes utilisées (latence = 1 + délai)