TD 1 – Processeurs scalaires, superscalaires et VLIW

1 Processeurs scalaires et superscalaires

1.1 Exécution de boucles

Soit un processeur à ordonnancement statique qui a les caractéristiques suivantes :
- les instructions sont de longueur fixe (32 bits)
- Il a 32 registres entiers (dont R0=0) de 32 bits et 32 registres flottants (de F0 à F31) de 32 bits.
- Il peut lire et exécuter 4 instructions par cycle.
- L'unité entière contient deux pipelines d'exécution entière sur 32 bits, soit deux additionneurs, deux décaleurs. Tous les bypass possibles sont implantés.
- L'unité flottante contient un pipeline flottant pour l'addition et un pipeline flottant pour la multiplication. - L'unité Load/Store peut exécuter jusqu'à deux chargements par cycle, mais ne peut effectuer qu'un load et un store simultanément. Elle ne peut effectuer qu'un seul store par cycle.
- Il dispose d'un mécanisme de prédiction de branchement qui permet de "brancher" en 1 cycle si la prédiction est correcte. Les sauts et branchements ne sont pas retardés.

La Table 1 donne
- les instructions disponibles
- le pipeline qu'elles utilisent : E0 et E1 sont les deux pipelines entiers, FA est le pipeline flottant de l'addition et FM le pipeline flottant de la multiplication. Les instructions peuvent être exécutées simultanément si elles utilisent chacune un pipeline séparé.

L'ordonnancement est statique. Les chargements ne peuvent pas passer devant les rangements en attente.

JEU D'INSTRUCTIONS (extrait)

Mnémo	Exemple	Latence	Pipeline	Effet
LF	LF Fi, dép.(Ra)	2	E0 ou E1	Fi <- M (Ra + dépl.16 bits avec ES)
SF	SF Fi, dép.(Ra)	1	E0	Fi -> M (Ra + dépl.16 bits avec ES)
ADD	ADD Rd,Ra, Rb	1	E0 ou E1	Rd <- Ra + Rb
ADDI	ADDI Rd, Ra, IMM	1	E0 ou E1	Rd <- Ra + IMM-16 bits avec ES
SUB	SUB Rd,Ra, Rb	1	E0 ou E1	Rd <- Ra - Rb
FADD	FADD Fd, Fa, Fb	4	FA	Fd <- Fa + Fb
FMUL	FMUL Fd, Fa, Fb	4	FM	Fd <- Fa x Fb
BEQ	BEQ Ri, dépl	1	E1	si Ri=0 alors CP <- CP+4 + depl
BNE	BNE Ri, dépl	1	E1	si Ri≠0 alors CP <- CP+4 + depl

Table 1 : instructions disponibles

La colonne latence donne la latence entre une instruction source et une instruction destination, dans le cas de dépendances de données. La valeur 1 est le cas où les deux instructions peuvent se succéder normalement, d'un cycle i au cycle i+1.

Soit le programme C suivant (SAXPY) où x et y sont des vecteurs et a est un scalaire de nombres flottants simple précision.

```
float x[1000], y[1000], a;
main ()
{
int i ;
for (i=0; i<1000 ;i++)
            y[i] = y[i] + a*x[i] ;
}
```

On utilisera les registres suivants : R1 pointe sur x[i], R2 pointe sur y[i]. R3 a été initialisé à 1000. F0 contient a. F1 et F2 recevront respectivement x[i] et y[i].

Question 1

Donner l'exécution cycle par cycle de la boucle optimisée en considérant une version scalaire du processeur et des caches parfaits. Quel est le nombre de cycles par itération de la boucle initiale ?
Donner la version déroulée (4 itérations par corps de boucle) avec la version scalaire en supposant des caches parfaits. Quel est le nombre de cycles par itération de la boucle initiale ?

Question 2

Donner l'exécution cycle par cycle de la boucle optimisée pour la version superscalaire en considérant des caches parfaits. Quel est le nombre de cycles par itération de la boucle initiale ?

Question 3

Donner la version déroulée (4 itérations par corps de boucle) avec la version superscalaire en supposant des caches parfaits. Quel est le nombre de cycles par itération de la boucle initiale ?

1.2 Instructions SIMD

On ajoute au processeur 8 registres de 128 bits S0 à S7 pouvant contenir chacun 4 flottants simple précision et les instructions SIMD données dans la Table 2. Cette table donne également les latences des instructions SIMD et le pipeline utilisé dans le cas superscalaire.

PLF	PLF Si, dép.(Ra)	2	E0	Charge quatre flottants simple précision à partir de l'adresse (Ra + dépl.16 bits avec ES).
PST	PST Si, dép.(Ra)	2	E0	Range en mémoire à partir de l'adresse (Ra + dépl.16 bits avec ES) quatre flottants simple précision
PADD	PADD Sf,Sa, Sb	4	FA	SF <- Sa + Sb
PSUB	PSUB Sf ,Sa, Sb	4	FA	SF <- Sa - Sb
PMUL	PMUL Sf, Sa, Sb	4	FM	SF <- Sa x Sb
PMULS	PMULS Sf, Sa, Fb	4	FM	SF <- Sa x Fb (chaque élément de Sa est multiplié par Fb et rangé dans l'élément correspondant de SF)

Table 2 : Instructions SIMD

Question 4

Pour une version scalaire du processeur, en supposant des caches parfaits, donner l'exécution cycle par cycle de la boucle SAXPY en utilisant des instructions SIMD. Quel est le temps d'exécution par itération de la boucle initiale ?

Question 5

Pour la version superscalaire du processeur, en supposant des caches parfaits, donner l'exécution cycle par cycle de la boucle SAXPY en utilisant des instructions SIMD. Quel est le temps d'exécution par itération de la boucle initiale ?

Question 6

Pour la version superscalaire du processeur, en supposant des caches parfaits, donner l'exécution cycle par cycle de la boucle SAXPY en utilisant des instructions SIMD et un déroulage d'ordre 4 (16 itérations de la boucle initiale par corps de boucle). Quel est le temps d'exécution par itération de la boucle initiale ?

2 Prédiction de branchement

Soit le programme C suivant :

```
long a=0, b=0, n=0, p=0;
main()
{
int i, n, p;
for (i=0; i<24; i++)
      {
      a = (a+1)%2
      b= (b+1)%3
      if (a>=b)
```

```
            n++;
            else p++;
      }
}
```

On considère le branchement conditionnel correspondant au if(a>=b). Le branchement est pris (P) si (a<b) et non pris (NP) autrement.
On associe un prédicteur à ce branchement. On utilise soit un prédicteur 1 bit, soit un compteur 2 bits. Le compteur 2 bits a 4 états : fortement non pris (FNP), faiblement non pris (fNP), faiblement pris (fP) et fortement pris (FP) auxquels on peut associer les valeurs 0, 1, 2 et 3.
Dans tous les cas, les prédicteurs sont initialisés à NP (ou FNP pour le compteur 2 bits)

Question 7) Pour les 24 itérations de la boucle, quel est le nombre de prédictions correctes dans les cas suivants :

a) on utilise un prédicteur 1 bit par branchement
b) on utilise un prédicteur 2 bits par branchement
c) on utilise l'historique des 2 derniers branchements, avec un prédicteur 1 bit par configuration du registre d'historique
d) on utilise l'historique des 3 derniers branchements, avec un prédicteur 1 bit par configuration.

a																							
b																							
B																							
a)																							
b)																							
c)																							
d)																							

3 Processeurs VLIW

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.

3.1 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
```

Question 8) 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);
}
```

3.2 Fonction IIR.

Question 9) 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;
       }
}
```

3.3 Annexe

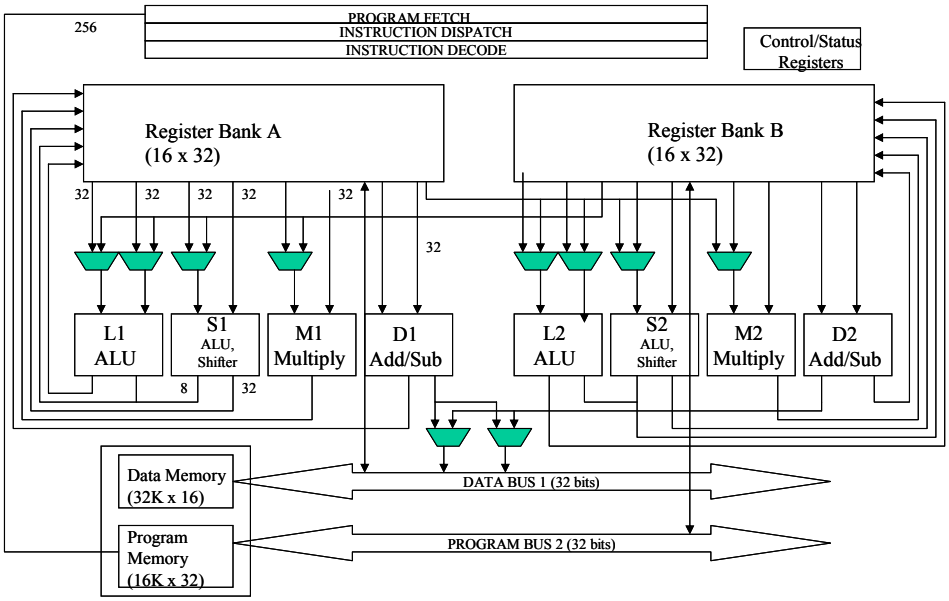

Figure 1 : Schéma fonctionnel du TMS320C62

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

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

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

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

.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 3 : Répartition des instructions entières dans les unités fonctionnelles

.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 4 : Répartition des instructions flottantes.