

TD-6 : Superscalaires et VLIW

Superscalaire statique

Soit un processeur superscalaire 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.

Pour la version scalaire, une seule instruction démarre par cycle d'horloge.

JEU D'INSTRUCTIONS (extrait)

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

Table 1 : instructions disponibles

La Table 1 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 ?

Ions SIMD ?

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 ces exercices, 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.

Question 1) Que fait le programme P1 ? (on mettra en évidence l'établissement du pipeline logiciel) Quel est le temps d'exécution du programme ?

	LDW	.D1	*A4++,A2	; load ai & ai+1 from memory
	LDW	.D2	*B4++,B2	; load bi & bi+1 from memory
	MVK	.S1	50,A1	; set up loop counter
	ZERO	.L1	A7	; zero out sum0 accumulator
	ZERO	.L2	B7	; zero out sum1 accumulator
[A1]	SUB	.S1	A1,1,A1	; decrement loop counter
	LDW	.D1	*A4++,A2	; * load ai & ai+1 from memory
	LDW	.D2	*B4++,B2	; * load bi & bi+1 from memory
[A1]	SUB	.S1	A1,1,A1	; * decrement loop counter
	[A1] B	.S2	LOOP	; branch to loop
	LDW	.D1	*A4++,A2	; ** load ai & ai+1 from memory
	LDW	.D2	*B4++,B2	; ** load bi & bi+1 from memory
[A1]	SUB	.S1	A1,1,A1	; ** decrement loop counter
	[A1] B	.S2	LOOP	; * branch to loop
	LDW	.D1	*A4++,A2	; *** load ai & ai+1 from memory
	LDW	.D2	*B4++,B2	; *** load bi & bi+1 from memory
[A1]	SUB	.S1	A1,1,A1	; *** decrement loop counter
	[A1] B	.S2	LOOP	; ** branch to loop
	LDW	.D1	*A4++,A2	; **** load ai & ai+1 from memory
	LDW	.D2	*B4++,B2	; **** load bi & bi+1 from memory

```

    MPY    .M1X  A2,B2,A6      ; ai * bi
||      MPYH   .M2X  A2,B2,B6      ; ai+1 * bi+1
|| [A1] SUB    .S1   A1,1,A1      ;**** decrement loop counter
|| [A1] B      .S2   LOOP        ;*** branch to loop
||      LDW    .D1   *A4++,A2     ;***** ld ai & ai+1 from memory
||      LDW    .D2   *B4++,B2     ;***** ld bi & bi+1 from memory

    MPY    .M1X  A2,B2,A6      ;* ai * bi
||      MPYH   .M2X  A2,B2,B6      ;* ai+1 * bi+1
|| [A1] SUB    .S1   A1,1,A1      ;***** decrement loop counter
|| [A1] B      .S2   LOOP        ;**** branch to loop
||      LDW    .D1   *A4++,A2     ;***** ld ai & ai+1 from memory
||      LDW    .D2   *B4++,B2     ;***** ld bi & bi+1 from memory

LOOP:
    ADD    .L1   A6,A7,A7      ; sum0 += (ai * bi)
||      ADD    .L2   B6,B7,B7      ; sum1 += (ai+1 * bi+1)
||      MPY    .M1X  A2,B2,A6      ;** ai * bi
||      MPYH   .M2X  A2,B2,B6      ;** ai+1 * bi+1
|| [A1] SUB    .S1   A1,1,A1      ;***** decrement loop counter
|| [A1] B      .S2   LOOP        ;**** branch to loop
||      LDW    .D1   *A4++,A2     ;***** ld ai & ai+1 fm memory
||      LDW    .D2   *B4++,B2     ;***** ld bi & bi+1 fm memory
||      ; Branch occurs here

    ADD    .L1X  A7,B7,A4      ; sum = sum0 + sum1

```

Figure 1 : Programme P1

Question 2) Examiner l'exécution du code de la figure 2, 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

```

Figure 2 : Extrait de code

Question 3) Que fait le programme P2 ? Quel est son temps d'exécution ?

```

||      MVK     .S1   50,A1        ; set up loop counter
||      ZERO    .L1   A8          ; sum0 = 0
||      ZERO    .L2   B8          ; sum1 = 0
||      LDDW   .D1   A4++,A7:A6   ; load ai & ai + 1 from memory
||      LDDW   .D2   B4++,B7:B6   ; load bi & bi + 1 from memory

||      LDDW   .D1   A4++,A7:A6   ;* load ai & ai + 1 from memory
||      LDDW   .D2   B4++,B7:B6   ;* load bi & bi + 1 from memory

||      LDDW   .D1   A4++,A7:A6   ;** load ai & ai + 1 from memory
||      LDDW   .D2   B4++,B7:B6   ;** load bi & bi + 1 from memory

||      LDDW   .D1   A4++,A7:A6   ;*** load ai & ai + 1 from memory
||      LDDW   .D2   B4++,B7:B6   ;*** load bi & bi + 1 from memory
|| [A1] SUB    .S1   A1,1,A1      ; decrement loop counter

```

```

    LDDW    .D1    A4++,A7:A6    ;**** load ai & ai + 1 from memory
    LDDW    .D2    B4++,B7:B6    ;**** load bi & bi + 1 from memory
    [A1] B   .S2    LOOP          ; branch to loop
    [A1] SUB .S1    A1,1,A1        ;* decrement loop counter

    LDDW    .D1    A4++,A7:A6    ;***** load ai & ai + 1 from memory
    LDDW    .D2    B4++,B7:B6    ;***** load bi & bi + 1 from memory
    MPYSP   .M1X   A6,B6,A5       ; pi = a0 b0
    MPYSP   .M2X   A7,B7,B5       ; pil = a1 b1
    [A1] B   .S2    LOOP          ;* branch to loop
    [A1] SUB .S1    A1,1,A1        ;** decrement loop counter

    LDDW    .D1    A4++,A7:A6    ;***** load ai & ai + 1 from memory
    LDDW    .D2    B4++,B7:B6    ;***** load bi & bi + 1 from memory
    MPYSP   .M1X   A6,B6,A5       ;* pi = a0 b0
    MPYSP   .M2X   A7,B7,B5       ;* pil = a1 b1
    [A1] B   .S2    LOOP          ;** branch to loop
    [A1] SUB .S1    A1,1,A1        ;*** decrement loop counter

```

```

    LDDW    .D1    A4++,A7:A6    ;***** load ai & ai + 1 from memory
    LDDW    .D2    B4++,B7:B6    ;***** load bi & bi + 1 from memory
    MPYSP   .M1X   A6,B6,A5       ;** pi = a0 b0
    MPYSP   .M2X   A7,B7,B5       ;** pil = a1 b1
    [A1] B   .S2    LOOP          ;*** branch to loop
    [A1] SUB .S1    A1,1,A1        ;**** decrement loop counter

    LDDW    .D1    A4++,A7:A6    ;***** load ai & ai + 1 from memory
    LDDW    .D2    B4++,B7:B6    ;***** load bi & bi + 1 from memory
    MPYSP   .M1X   A6,B6,A5       ;*** pi = a0 b0
    MPYSP   .M2X   A7,B7,B5       ;*** pil = a1 b1
    [A1] B   .S2    LOOP          ;**** branch to loop
    [A1] SUB .S1    A1,1,A1        ;***** decrement loop counter

LOOP:
    LDDW    .D1    A4++,A7:A6    ;***** load ai & ai + 1 from memory
    LDDW    .D2    B4++,B7:B6    ;***** load bi & bi + 1 from memory
    MPYSP   .M1X   A6,B6,A5       ;**** pi = a0 b0
    MPYSP   .M2X   A7,B7,B5       ;**** pil = a1 b1
    ADDSP   .L1    A5,A8,A8       ; sum0 += (ai bi)
    ADDSP   .L2    B5,B8,B8       ; sum1 += (ai+1 bi+1)
    [A1] B   .S2    LOOP          ;***** branch to loop
    [A1] SUB .S1    A1,1,A1        ;***** decrement loop counter
; Branch occurs here

    ADDSP   .L1X   A8,B8,A0       ; sum(0) = sum0(0) + sum1(0)

    ADDSP   .L2X   A8,B8,B0       ; sum(1) = sum0(1) + sum1(1)

    ADDSP   .L1X   A8,B8,A0       ; sum(2) = sum0(2) + sum1(2)

    ADDSP   .L2X   A8,B8,B0       ; sum(3) = sum0(3) + sum1(3)

    NOP                                ; wait for B0

    ADDSP   .L1X   A0,B0,A5       ; sum(01) = sum(0) + sum(1)

    NOP                                ; wait for next B0

    ADDSP   .L2X   A0,B0,B5       ; sum(23) = sum(2) + sum(3)

    NOP                                3

    ADDSP   .L1X   A5,B5,A4       ; sum = sum(01) + sum(23)

    NOP                                3
;

```

Annexe

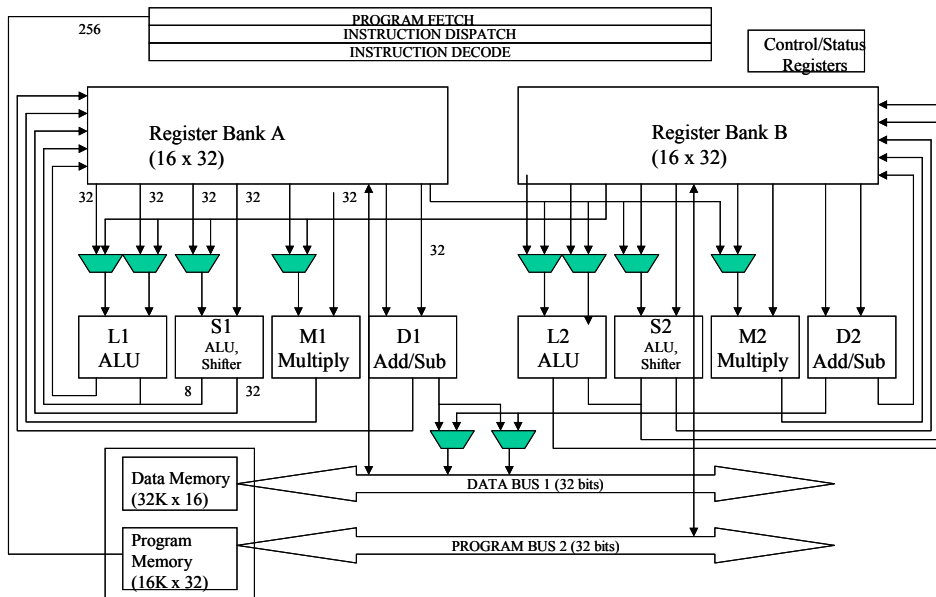


Figure 3 : Schéma fonctionnel du TMS320C62

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)

.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 2 : 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 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)