

## Examen Décembre 09 - Architectures Avancées

3H – Tous documents autorisés

### PIPELINES

Pour les processeurs P0 et P1, les phases du pipeline ont la signification classique :

LI : lecture d'instruction

DI : Décodage

LO : Lecture registres

CAB : Calcul de l'adresse de branchement

CA : Calcul d'adresse mémoire

AD : Accès Donnée (cache donnée)

EX : Exécution

RR : Rangement du résultat.

### **Pipelines du processeur P0**

UAL	LI	DI/LO	EX	RR		
Load/Store	LI	DI/LO	CA	AD	RR	
Flottant	LI	DI	LO	EX1	EX2	RR
Branch	LI	DI/CAB				

### **Pipelines du processeur P1**

UAL	LI	DI	LO	EX1	EX2	RR		
Load/Store	LI	DI	LO	CA	AD	RR		
Flottant	LI	DI	LO	EX1	EX2	EX3	EX4	RR
Branch	LI	DI	LO	CAB				

On suppose que tous les "bypass" nécessaires existent.

### **Q 1) Donner pour le processeur P0**

- la latence des instructions entières :
- la latence des instructions de chargement :
- la latence des instructions flottantes :
- la pénalité pour un branchement mal prédit :

### **Q 2) Donner pour le processeur P1**

- la latence des instructions entières :
- la latence des instructions de chargement :
- la latence des instructions flottantes :
- la pénalité pour un branchement mal prédit :

### **SIMD IA-32**

Soit le programme P1.

```
unsigned char X[512], max;  
  
max = X[0];  
for (i=1 ; i<512 ; i++)  
    if (X[i] >max) max=X[i] ;
```

**Q 3) Donner la version SIMD IA-32 du programme P1, en utilisant les instructions IA-32 données en annexe (on supposera que le vecteur X est aligné sur des frontières de mots de 128 bits)**

**Instructions spécialisées sur processeur NIOS.**

On veut implémenter sous forme d'instructions spécialisées des instructions de réduction de type max sur le contenu d'un registre de 32 bits.

- L'instruction RMAXUB (Rs) considérera les quatre octets non signés contenus dans le registre Rs et fournira le maximum non signé des quatre octets dans l'octet bas du registre Rd, les trois octets de poids fort étant mis à zéro.
- L'instruction RMAXB (Rs) considérera les quatre octets signés dans le registre Rs et fournira le maximum (signé) dans l'octet bas du registre Rd, les trois octets de poids fort contenant le signe de l'octet bas (extension de signe).

**Q 4) Donner le code VHDL (entité et architecture) pour les instructions RMAXUB et RMAXH. Quel nombre de cycles d'horloge processeur utilisera chacune de ces instructions ?**

**Pipeline logiciel avec TMS 320C62x**

Les deux premières instructions avant le prologue d'un programme sont

```
MVK 100,A1
MV B4,B6
```

La boucle de pipeline logiciel après le prologue travaille sur deux tableaux X[100] et Y[100]  
LOOP :

```
.....LDH      .D1      *A4++,A2          ; A4 pointe sur X[i]
||             LDH      D2      *B4++,B2..... ;B4 pointe sur Y[i]

.....STH      D2      B5, *B6++          ;
||.....ADD     .L2X    A2,B2,B5          ;
|| [ A1] SUB   .S1    A1,1,A1           ;
|| [ A1] B     S2     LOOP              ;
```

Ce pipeline logiciel correspond à un programme C dont le début est

```
Type X[100], Y[100];
for (i=0 ; i<100 ;i++){
    Corps de boucle ; }
```

**Q 5) Quel est le type des variables X[100] et Y[100] Donner le corps de boucle du programme C original.**

**Q 6) Quel est le nombre de cycles par itération lorsque le pipeline logiciel est établi ? Pourquoi ne peut-on diminuer ce nombre de cycles ?**

**Q 7) Donner le code C correspondant à la boucle de pipeline logiciel suivante qui travaille également sur X[100] et Y[100]. Quel est le nombre de cycles par itération ?**

NB : l'instruction ADD2 effectue l'addition SIMD sur les mots de 16 bits signés contenus dans les registres de 32 bits.

```
MVK 100,A1
MV B4,B6
```

LOOP :

```
.....LDW      .D1      *A4++,A2          ; A4 pointe sur X[i]
||             LDW      D2      *B4++,B2..... ;B4 pointe sur Y[i]

.....STW      D2      B5, *B6++          ;
||.....ADD2    L2X    A2,B2,B5          ;
```

```

|| [ A1]   SUB   .S1   A1, 2, A1   ;
|| [ A1]   B     S2   LOOP       ;
    
```

**Cohérence des caches**

Le protocole MESI (Illinois) est un protocole avec réécriture et invalidation qui a quatre états : modifié (M), exclusif (E), partagé (S) et invalide (I). Son diagramme de transitions est donné en figure 1.

On considère l'organisation suivante :

Deux processeurs P1 et P2 utilisent le protocole MESI. Ils ont des caches à correspondance directe, chacun de taille 4 mots. Il faut 4 cycles pour un transfert de bloc d'un cache à l'autre. Une lecture ou écriture avec succès (hit) prend 1 cycle. Une invalidation prend 2 cycles. Un transfert de bloc de la mémoire vers un cache prend 8 cycles.

B1 et B2 sont des blocs mémoire qui vont dans le même bloc de cache dans chaque processeur (comme résultat de la correspondance directe). B1 contient les mots W,X,Y,Z et B2 contient les mots P,Q,R,S.

**Q 8) Remplir la table ci-dessous qui contient la séquence des accès mémoire successifs pour les processeurs P1 et P2.** Les accès mémoire sont strictement séquentiels, dans l'ordre ci-dessous. On suppose que les caches sont initialement vides. Quand une action bus est nécessaire, son temps d'exécution s'ajoute au temps nécessaire pour la requête processeur (lecture ou écriture).

	Accès mémoire	Action sur caches	Etat P1	Etat P2	Coût (cycles)
1-	P1 : Lecture Z				
2-	P2 : Lecture W				
3-	P1 : Lecture W				
4-	P1 : Ecriture Z				
5-	P2 : Lecture W				
6-	P1 : Lecture P				
7-	P2 : Ecriture P				
8-	P1 : Lecture P				
9-	P1 : Lecture Q				
10-	P2 : Lecture W				
11-	P1 : Ecriture Q				
12-	P2 : Lecture X				
13-	P1 : Lecture R				

**Q9) Déterminer le temps total pour exécuter la séquence d'accès mémoire.**

**Parallélisation OpenMP**

**Q 10) Donner une version OpenMP du programme P1 de la question Q3)**

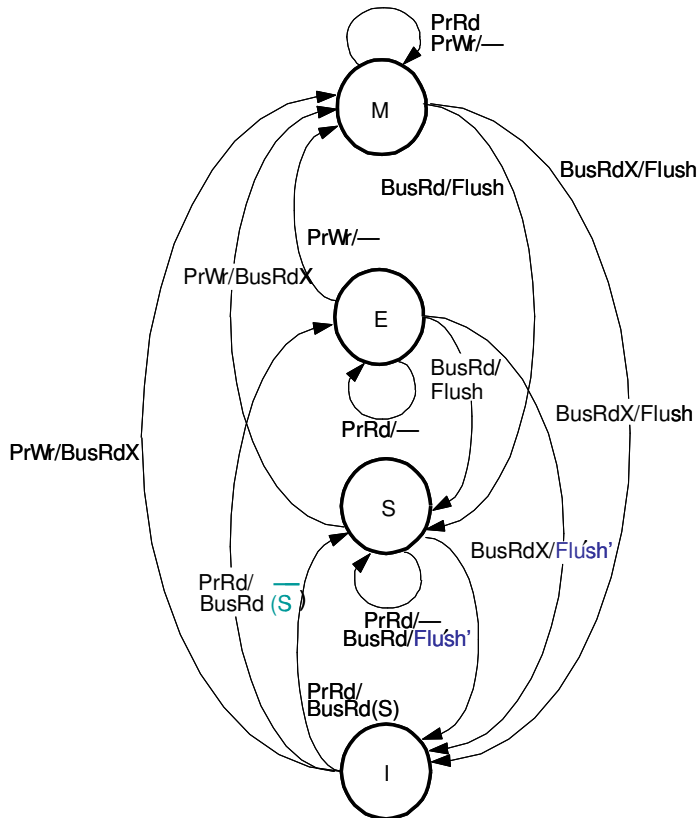


Figure 1 : protocole MESI

### **ANNEXE : Instructions SIMD IA-32 utilisables**

#define ld16(a)	_mm_load_si128(&a)	chargement aligné
#define st16(a, b)	_mm_store_si128(&a, b)	rangement aligné
#define or(a,b)	_mm_or_si128(a,b)	ou logique
#define xor(a,b)	_mm_xor_si128(a,b)	ou exclusif
#define maxbu(a,b)	_mm_max_epu8(a,b)	max 8 bits non signés
#define minbu(a,b)	_mm_min_epu8(a,b)	min 8 bits non signés
#define addh(a,b)	_mm_add_epi16(a,b)	addition 16 bits signée
#define addhu(a,b)	_mm_add_epu16(a,b)	addition 16 bits non signée
#define subh(a,b)	_mm_sub_epi16(a,b)	soustraction 16 bits signée
#define subbu(a,b)	_mm_subs_epu8(a,b)	Soustraction 8 bits non signés avec saturation
#define b2hl(a,b)	_mm_unpacklo_epi8(a,b)	8 octets bas entrelacés vers 8 shorts
#define b2hh(a,b)	_mm_unpackhi_epi8(a,b)	8 octets haut entrelacés vers 8 shorts
#define h2b(a,b)	_mm_packus_epi16(a,b)	8 shorts (a) dans 8 octets bas - 8 shorts (b) dans 8 octets haut
#define srl128(a,v)	_mm_srli_si128(a,v)	décalage droite des 128 bits de a de v octets
#define sll128(a,v)	_mm_slli_si128(a,v)	décalage gauche des 128 bits de a de v octets
#define srai16(a,v)	_mm_srai_epi16(a,v)	déc. droite des 8x 16 bits de a de v bits
#define sll16(a,v)	_mm_slli_epi16(a,v)	déc. gauche des 8x16 bits de a de v bits
#define cmpgt8(a,b)	_mm_cmpgt_epi8(a,b)	comparaison octets signés. Si octet (a) > octet (b) octet résultat = FF <sub>H</sub> sinon 00 <sub>H</sub>