Université Paris Sud 2013-2014
Informatique L313

ARCHITECTURE DES ORDINATEURS
Examen Juin 2014
3 H – Tous documents autorisés – Parties indépendantes

OPTIMISATIONS DE PROGRAMME

Cette partie utilise le sous-ensemble du jeu d'instructions MIPS donné en annexe.
On suppose une version pipelinée du processeur utilisant les instructions MIPS.
La latence des instructions est donnée dans la deuxième colonne de la Table 2. On rappelle qu'une latence de **n** signifie que si une instruction **I** démarre au cycle **c**, une instruction qui utilise le résultat de I ne peut démarrer qu'au cycle c+n. (une latence de 1 signifie qu'elle peut démarrer au cycle suivant).
La Table 1 présente un programme C et le programme assembleur MIPS correspondant. Les tableaux X et Y sont rangés successivement à partir de l'adresse 0x10000000, qui est contenue au démarrage dans le registre R3. La valeur A est contenue dans le registre F0.

Question 1) Quel est le temps d'exécution (en cycles par itération) de la boucle de la table 1. Optimiser la boucle sans déroulage et donner le nouveau temps d'exécution.

float X[100], Y[100], A;
int i ;
for (i=0; i<100; i++)
Y[i]+= A*X[i]];
	ADDI R5, R3, 400
	Boucle :LWC1 F1,0(R3)
	LWC1 F2,400 (R3)
	MUL.S F1,F1,F0
	ADD.S F2,F2,F1
	SWC1 F2,400(R3)
	ADDI R3,R3,4
	BNE R3,R5, Boucle

Table 1 : Programme C et programme assembleur

Question 2) Donner le code assembleur d'une version optimisée après déroulage d'ordre 4.
Quel est maintenant le nombre cycles/itération de la boucle initiale ?

CACHES

Soit le programme suivant :

Programme P1 :

```
float X[N], Y[N], Z[N] , S ;
S=0.0;
for (i=0 ; i<N ; i++)
      S = S + X[i]+ Y[i]+ Z[i];
```

On considère un cache de données de 8 Ko, à correspondance directe, avec des lignes de 16 octets.
Les trois tableaux sont rangés successivement en mémoire à partir de l'adresse 0xA000 0000.
Les variables scalaires sont en registre.

Question 3) Si N = 128, dans quelles lignes du caches vont X[0], Y[0] et Z[0] ? Quel est le nombre total de défauts de caches pour exécuter le programme P1 ?

Question 4) Donner le nombre total de défauts de cache dans les deux cas suivants :

1. **N=2048**
2. **N=1024**

TEMPS D'EXECUTION

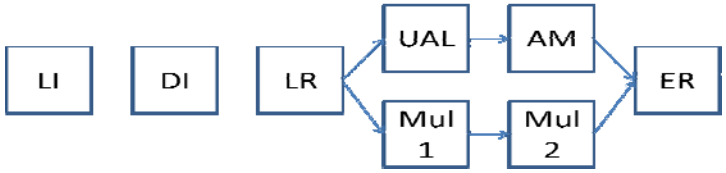
Soit le programme C suivant :

```
float X[N][N], Y[N][N], Z[N][N], A ;
for (j=0 ; j<N ;j++)
      for (i=0; i<N;i++)
            Y[i][j] = A*X[i][j];
for (i=0; i<N;i++)
      for (j=0 ; j<N ;j++)
            Z[i][j]+= Y[i][j] ;
```

Question 5) Donner une nouvelle version (sans utiliser d'instructions SIMD) du programme C permettant de réduire le temps d'exécution (sans tenir compte d'éventuelles optimisations du compilateur)

PIPELINE

La figure ci-dessous donne le pipeline du processeur ARM10 (jeu d'instructions ARM).



La signification des étapes du pipeline sont

- LI : lecture de l'instruction
- DI : décodage de l'instruction
- LR : lecture des registres
- UAL : exécution UAL ou calcul adresse
- Mul 1 et 2 : étapes de la multiplication
- AM accès cache de données
- ER : écriture du résultat.

Question 6) Donner les latences des instructions suivantes, en supposant que tous les court-circuits nécessaires sont implantés.

	Instruction	Producteur	Consommateur
a)	ADD	ADD **R1**,R2,R3	SUB R5,R6,**R1**
b)	ADD	ADD **R1**,R2,R3	STR **R1**,[R4 +4] !
c)	ADD	ADD **R1**,R2,R3	STR R4,[**R1** +4] !
d)	ADD	ADD **R1**,R2,R3	MUL R5,R6,**R1**
e)	LDR	LDR **R1,** [R2], #4	SUB R5,R6,**R1**
f)	MUL	MUL **R1**,R2,R3	SUB R5,R6,**R1**

PROGRAMMATION ASSEMBLEUR

Soit le programme assembleur ARM ci-dessous

```
      LDR r2,=A         @chargement adresse de A
      MOV r1,#9
      LDR r3,[r2],#4
      MOV r4,r3
LOOP: LDR r5,[r2],#4
      CMP r5,r3
      MOVLT r3,r5
      CMP r5,r4
      MOVGT r4,r5
      SUBS r1,r1,#1
      BGT LOOP
      LDR r6,=X
      STR r3,[r6]
      LDR r6,=Y
      STR r4,[r6]
      SWI 0x11 @ Stop program execution

.data
A: .word 14, -50,132, 10, -2000, 3456,76, -123, 45,-300, 345
X: .word 0
Y: .word 0
```

Question 7) Que fait le programme assembleur ARM ? Quel est le contenu des cases mémoire d'adresse X et Y après exécution du programme ? ***La réponse doit tenir en moins de 5 lignes.***

SIMD

Soit le programme ci-dessous utilisant des intrinsics SIMD, comme dans le TP n°10.

```
#define N 500
#define PADD(a,b) _mm_add_ps(a,b)   // addition SIMD 4 floats
#define PMUL(a,b) _mm_mul_ps(a,b)   // multiplication SIMD 4 floats
#define DUP4(a)  _mm_set1_ps(a)     // 4 fois le float a dans 128 bits

typedef union VEC{
      float float_word[N];
      __m128 quad_word[N/4];
} VEC ;
VEC X,Z;
float A;
__m128 SS, temp;

main(){
 int j;
      SS = DUP4(A);
      for (j=0;j<N/4;j++){
            temp= PMUL (SS, X.quad_word[j]);
            Z.quad_word[j]= PADD (Z.quad_word[j, temp); }
```

Question 8) Quel calcul effectue le programme sur les vecteurs X[N] et Z[N] ? Donner le programme C équivalent. (*La réponse doit tenir en moins de 5 lignes).*

ANNEXE MIPS

Les figures donnent la liste des instructions MIPS disponibles.
La signification des abréviations est la suivante :
IMM correspond aux 16 bits de poids faible d'une instruction.
SIMM est une constante sur 32 bits, avec 16 fois le signe de IMM, suivi de IMM (extension de signe). ADBRANCH est l'adresse de branchement, qui est égale à NCP+ SIMM (NCP est l'adresse de l'instruction qui suit le branchement)

Mnémonique	Latence	Syntaxe	Action
ADDI	1	ADDI rt, rs, IMM	rt ← rs + SIMM avec exception sur débordement
BNE	1	BNE rs,rt, IMM.	si rs ≠ rt, branche à ADBRANCH
LWC1	2	LWC1 ft, IMM(rs)	ft ← MEM [rs +SIMM]
SWC1	1	SWC1 ft, IMM.(rs)	ft → MEM [rs + SIMM]
ADD.S	3	ADD.S fd, fs,ft	fd ← fs + ft (addition flottante simple précision)
MUL.S	5	MUL.S fd, fs,ft	fd ← fs * ft (multiplication flottante simple précision)
SUB.S	3	SUB.S fd, fs,ft	fd ← fs - ft (soustraction flottante simple précision)
DIV.S	12	DIV.S fd,fs,ft	fd ← fs / ft (division flottante simple précision)

Table 2 : Instructions entières et flottantes MIPS utilisées (NB : les branchements ne sont pas retardés)

ANNEXE ARM

On rappelle que le processeur ARM a 15 registres de 32 bits. Les immédiats sont signés.
R15 est le compteur de programme.

Instruction	Assembleur	Effet
CMP	CMP Ri,Rj	Compare Ri et Rj et positionne les codes condition CC
MOV	MOV Ri,Rj	Ri ← Rj
MOVLT	MOVLT Ri,Rj	Ri ← Rj si CC = LT (<)
MOVGT	MOVGT Ri,Rj	Ri ← Rj si CC = GT (>)
SUBS	SUBS Ri, Rj, #N	Ri ← Rj - N ; positionne Code condition
B	B adresse cible	Branchement inconditionnel
BNE	BNE adresse cible	Branchement si le résultat si CC différent de zéro
BL	BL adresse cible	Branchement et adresse de retour dans R14
MUL	MUL Ri, Rj, Rk	Ri reçoit Rj * Rk
LDR	LDR Rd, [Rs], #N	Rd reçoit Mem (Rs) et Rs = Rs +N
STR	STR Rd, [Rs], #N	Mem (Rs) reçoit Rd et Rs = Rs +N

Table 3 : Instructions ARM utilisées