Université Paris Sud
L3 Informatique
D. Etiemble

Architecture des ordinateurs

Examen Décembre 2012
3 H. Tous documents autorisés.
4 parties indépendantes

1. PROGRAMMATION ASSEMBLEUR

On utilise le jeu d'instructions NIOS

Programme assembleur

Les variables A, B et S ont situées aux adresses 0x 0000 1000, 0x 0000 1001 et 0x 0000 1002
Soit le programme NIOS

```
        LDBU R1, 0x1000 (R0)
        LDBU R2, 0x1001(R0)
        ADD R1,R1,R2
        ANDI R1,R1,0xFF
        BGE R1,R2,FIN
        ADDI R1,R0,0xFF
FIN :   STB R1,0x1002(R0)
        BEQ R0,R0, -4
```

Q 1) Que contient la variable S à la fin de l'exécution du programme ?

Ecriture d'une fonction

Q 2) Ecrire une procédure NIOS qui vérifie si un caractère codé en ASCII, contenu dans l'octet de poids faible de R2, est un chiffre, et renvoie 1 dans R1 si vrai et 0 sinon. On suppose que R2 a été chargé par l'instruction LDBU. On rappelle que les chiffres (0 à 9) sont codés de 0x30 à 0x39.

Ecriture d'une fonction

Q 3) Ecrire une fonction NIOS qui prend le contenu du registre R2 (un nombre flottant en simple précision) et renvoie dans R1 l'opposé du nombre flottant.

NB : le format flottant simple précision est rappelé en annexe 1.

2. EXECUTION DE BOUCLES

On ajoute au jeu d'instructions NIOS II des instructions flottantes simple précision (32 bits) (Figure 1) et 32 registres flottants F0 à F31 (F0 est un registre normal).
Les additions, soustractions et multiplications flottantes sont pipelinées. Une nouvelle instruction peut démarrer à chaque cycle. Les latences sont de 2 cycles pour LF et de 4 cycles pour les instructions flottantes.
Les branchements ne sont pas retardés.

LF	2	LF ft, déplac(rs)	ft ← MEM [rs +SIMM(déplac)]
SF	1	SF ft, déplac(rs)	ft → MEM [rs + SIMM(déplac)]]
FADD	4	FADD fd, fs,ft	fd ← fs + ft (addition flottante simple précision)
FMUL	4	FMUL fd, fs,ft	fd ← fs * ft (multiplication flottante simple précision)
FSUB	4	FSUB fd, fs,ft	fd ← fs - ft (soustraction flottante simple précision)

Figure 1: Instructions flottantes ajoutées (Ce ne sont pas les instructions NIOS)

Q 4) On considère le programme de la table 1. Les tableaux X, Y et Z sont rangés successivement en mémoire. L'adresse de X[0] est initialement dans R3. Optimiser ce programme sans déroulage de boucle. Quel est son temps d'exécution (en nombre de cycles par itération) du programme optimisé ? Quel est son temps d'exécution total ?

float X[800], Y[800], Z[800], A;
int i ;
for (i=0; i<800; i++)
Z[i] = A* X[i] + Y[i];
	LF F0, A //F0 ← A
	ADDI R5, R3, 3200
	Boucle :LF F1,0(R3)
	LF F2,3200 (R3)
	FMUL F1,F1,F0
	FADD F2,F2,F1
	SF F2,6400(R3)
	ADDI R3,R3,4
	BNEQ R3,R5, Boucle

Table 1 : Programme C et programme assembleur

Q 5) Donner le code assembleur correspondant à un déroulage de boucle d'ordre 4. Quel est alors le temps d'exécution (en nombre de cycles par itération de la boucle initiale) et le temps d'exécution total de ce nouveau programme.

3. CACHES

Soit un cache de 64 Ko à correspondance directe, avec des lignes de 32 octets. Le processeur a des registres de 32 bits et des adresses de 32 bits. Le cache est à réécriture (write back) et allocation d'écriture (il y a des défauts de caches en écriture).

Q 6) Quels sont les nombres de bits nécessaires pour l'index, l'étiquette et le déplacement (adresse dans la ligne) ?

Soit le code C.

```
double A[16384], i ;
for (i = 0; i < 8192; i++) {
      A[i] = A[i] + 1.0;
      A[i+8192] = A[i+8192] – 1.0;
}
```

L'adresse de A[0] est 0x1000 0000

Q 7) Dans quelles lignes du cache vont les flottants A[0] et A[8192] ?

Q 8) Quel est le nombre de défauts par itération et le nombre total de défauts ?

Q 9) Donner deux techniques logicielles (modification du code) et une technique matérielle (modification du cache) qui permettraient d'obtenir le nombre minimal de défauts compte tenu des défauts de démarrage. Quel est alors le nombre total de défauts de cache ?

4. SIMD

Soit le programme C SIMD ci-dessous, qui utilise les intrinsics définis en annexe 2.

__m128 *XS,*YS,*AS,C,T1,T2 ; //mots de 128 bits contenant des flottants simple précision.

```
C= setps (0.5) ;
for (i=0 ; i<32 ; i++)
{
        T1= lf16 ( XS[i]);
        T2 = lf16 (YS[i]);
        T1=subps(T1,T2);
        T1= mulps(T1,C);
        st16 (AS[i], T1); }
```

Q 10) Donner le programme C scalaire équivalent qui travaille sur des tableaux de flottants *X, *Y, *A.

Q 11) On suppose que le tableau X est initialisé à 3.0 (X[i] = 3.0 pour tout i). Quel est le contenu du tableau X après exécution du code suivant :

```
T1= lf16 XS[0];
for (i=1 ; i<32 ; i++)
{
        T2 = lf16 (XS[i]);
        T2 = addps(T1,T2);
        T1 = T2 ;
        st16 (XS[i], T2);
}
```

5. ANNEXE 1 : Rappel sur le format des nombres flottants simple précision

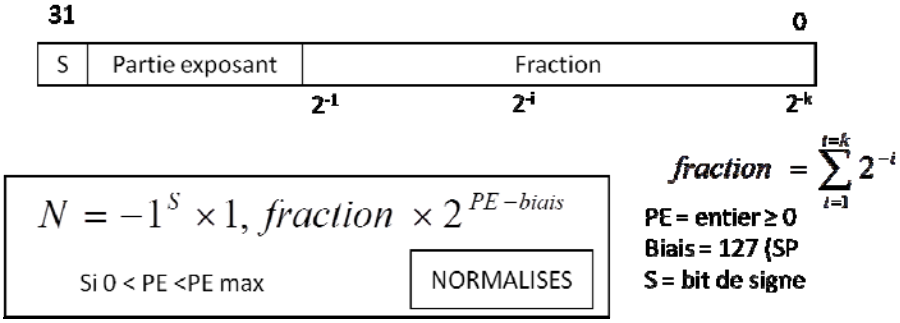

Figure 2 : Rappel sur le format flottant simple précision.

6. ANNEXE 2 : Instructions SIMD IA-32 utilisables

#define lf16(a)	_mm_load_ps(&a)	chargement aligné de 4 floats
#define st16(a, b)	_mm_store_ps(&a, b)	rangement aligné de 4 floats
#define addps	_mm_add_ps (a,b)	Addition des 4 floats de a et b
#define subps	_mm_sub_ps (a,b)	Soustraction des 4 floats a - b
# define mulps	_mm_mul_ps (a,b)	Multiplication des 4 floats de a et b
# define setps	_mm_set1_ps (float w)	Met w dans les 4 floats du mot