Architecture des ordinateurs

Corrigé examen Décembre 2013
3 H. Tous documents autorisés.
X 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 ?

S = Min (A+B, 255)
Somme saturée de deux octets non signés.

Ecriture d'une fonction

Q 2) Ecrire une procédure NIOS qui vérifie si un caractère, 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 sont codés entre 30_H (0) et 39_H (9)
Test_chiffre :

```
        CMPGEI  R1,R2,0x30     // R1 = 1 si R2>=30   ou CMPGEUI
        CMPLEI R3, R2, 0x39 // R3 = 1 si R2 <=39    ou CMPLEUI
        AND R1,R1,R3
        JMP R31                 // ou RET
```

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.

Il suffit de complémenter le bit de signe (31)
Opposé : ORHI R3,R0, 0x8000
```
         XOR R1,R2,R3
         JMP R31
```
Ou (mieux) XORHI R1,R2, 0x8000
```
        JMP R31
```

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]
SF	1	SF ft, déplac(rs)	ft → MEM [rs + SIMM]
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) Quel est le temps d'exécution (en cycles par itération) du programme optimisé (mais sans déroulage de boucle) et le temps d'exécution total du programme de la table 1.

Table 1 : Programme C et programme assembleur

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,(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

Boucle 1	LF F1,(R3)
2	LF F2, 3200(R3)
3	FMUL F1,F1,F0
4	ADDI R3,R3,4
5	
6	
7	FADD F2,F2,F1
8	
9	
10	
11	SF F2,3196(R3)
12	BNEQ R3,R5, Boucle

12 cycles/itérations

Total : 9600 cycles + 2 cycles = 9602 cycles

Q 5) Quel est le temps d'exécution (en cycles par itération de la boucle initiale) avec un déroulage de boucle d'ordre 4 et le temps d'exécution total du programme de la table 1.

Boucle 1	LF F1,(R3)
2	LF F3,4(R3)
3	LF F5,8(R3)
4	LF F7,12(R3)
5	LF F2,3200(R3)
6	LF F4,3204(R3)
7	LF F6,3208(R3)

8	LF F8,3212(R3)
9	FMUL F1,F1,F0
10	FMUL F3,F3,F0
11	FMUL F5,F5,F0
12	FMUL F7,F7,F0
13	FADD F2,F2,F1
14	FADD F4,F4,F3
15	FADD F6,F6,F5
16	FADD F8,F8,F7
17	SF F2,6400(R3)
18	SF F4,6404(R3)
19	SF F6,6408(R3)
20	SF F8, 6412(R3)
21	ADDI R3,R3,16
22	BNEQ R3,R5, Boucle

22 cycles/4 itérations = 5,5 cycles/itération
Total : 5,5 * 800 + 2 = 4402 cycles.

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) ?

64 Ko = 2^{16}.
Lignes de 32 octets : 2^5.
Nombre de lignes : 2^{11} = 2048.
Déplacement : 5 bits
Index : 11 bits
Etiquette 16 bits

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] ?

A[0] va dans la ligne 0
8192 doubles = 65536 = 10000H
&A[8192] =1001 0000 = 0001 0000 0000 0001|0000 0000 000|0 0000
A[8192] va dans la ligne 0

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

A[0] et A[8192] vont dans la même ligne de cache. Il y a deux défauts en lecture et zéros défauts en écriture par itération de la boucle.
Total : 16384 défauts.

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

Les lignes sont de 32 octets, soit 4 doubles. Il faut donc accéder à tous les éléments d'une ligne avant qu'elle soit éjectée ;

Première technique logicielle.
Déroulage de boucle d'ordre 4, soit le programme

```
int A[16384], i ;
for (i = 0; i < 8192; i += 4) {
      A[i] = A[i] + 1;
      A[i+1] = A[i+1] + 1;
      A[i+2] = A[i+2] + 1;
      A[i+3] = A[i+3] + 1;
      A[i+8192] = A[i+8192] - 1;
      A[i+8193] = A[i+8193] - 1;
      A[i+8194] = A[i+8194] - 1;
      A[i+8195] = A[i+8195] - 1;
}
Il y a alors deux défauts en lecture toutes les 4 itérations.
Total : 4096 défauts
```

Deuxième technique logicielle
Décomposer la boucle en deux boucles distinctes

```
int A[16384], i ;
for (i = 0; i < 8192; i++)
      A[i] = A[i] + 1;
for (i = 0; i < 8192; i++)
      A[i+8192] = A[i+8192] - 1;

1 défaut (lecture) toutes les 4 itérations pour chaque boucle.
Total : 4096 défauts
```

Technique matérielle
Utiliser un cache associatif par ensemble (deux voies).
Plus de conflit, soit 4096 défauts

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

        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.

```
Float *X, *Y, *A;
For (i=0 ; i<128 ;i++)
        A[i] =(X[i]-Y[i])*0.5;
```

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);
}
```

Le programme lit le tableau X par groupe de 4 floats et ajoute 3.0 3.0 3.0 et 3.0 à chaque groupe de 4 floats.
Après exécution du code, le tableau X contient donc
3 3 3 3 6 6 6 6 9 9 9 9 12 12 12 12 etc.