Université Paris Sud 2011-2012
Licence d'Informatique (L313)

ARCHITECTURE DES ORDINATEURS
Examen Juin 2012
3H – Tous documents autorisés
Les questions sont indépendantes

On utilise le jeu d'instructions NIOS II et le jeu d'instructions ARM.

EXECUTION D'INSTRUCTIONS NIOS

Q 1) Comment peut on exécuter avec le jeu d'instructions NIOS les actions suivantes :

a) Mise à zéro du registre R_5
b) Incrémentation du registre R_4
c) Chargement de 4500_H dans le registre R_{10}
d) Chargement de la valeur FFFF 3000_H dans le registre R_2
e) Mise à zéro de l'octet d'adresse 0000 1000_H
h) Mise à zéro des cases mémoire d'adresse 0000 8000_H à 0000 $80FF_H$

PROGRAMMATION ASSEMBLEUR

Q 2) Donner le code C correspondant au code assembleur NIOS

Le programme travaille sur un tableau d'entiers 32 bits dont l'adresse du premier élément T[0) est à l'adresse 0x4000 0000

```
      orhi r3, 0x4000
      ori r4,r3,80
      stw r0,0(r3)
      add r1,r0,r0
      addi r2,r1,1
      stw r2,4(r3)
      addi r3,r3,8
LOOP :
      add r5,r1,r2
      stw r5,0(r3)
      addi r3,r3,4
      add r1,r2,r0
      add r2,r5,r0
      bne r3,r4, LOOP
```

Q 3) Donner le contenu de la table à la fin d'exécution du programme. Que fait le programme assembleur ?

Soit le programme C suivant

```
int a, b ;
while (a != b) {
if (a>b) a=a-b;
if (a<b) b=b-a;}
```

On suppose que a et b ont été initialement chargés dans les registres r1 et r2 par des instructions ldr.

Q 4) Ecrire le code ARM de la boucle while.

CACHES

On suppose qu'un processeur a un cache données de 32 Ko, avec des blocs de 64 octets. Le cache utilise l'écriture simultanée avec écriture non allouée (il n'y a pas de défauts de cache en écriture)
Le processeur a des adresses sur 32 bits.

Q 5) Quel est pour ce cache le nombre de bits pour l'adresse dans le bloc, le nombre de bits d'index et le nombre de bits d'étiquette pour un cache à correspondance directe

Q 6) Dans quels blocs du cache vont les flottants X[0] et X[2048] qui sont aux adresses 1000 0000_H et 1000 2000_H. ?

Q 7) Quel est le nombre total de défauts de caches lors de l'exécution des boucles pour le cache à correspondance directe pour les trois programmes suivants?

On considère les programmes suivants, pour lequel le tableau X est rangé en mémoire à partir de l'adresse 1000 0000_H (adresse de X[0].)

```
float X[4096] ;
1)  for (i=0 ; i<2048 ; i++)
        S+=X[i] ;
2) for (i=0 ; i<2032 ; i++)
        S+=X[i] + X[i+16];
3) for (i=0 ; i<2048 ; i++)
      S+ =X[i] + X[i+2048];
```

OPTIMISATIONS DE PROGRAMME.

On suppose une version pipelinée du processeur utilisant les instructions ARM.
La latence de toutes les instructions arithmétique et logique est de 1 cycle, sauf pour la multiplication entière MUL (32 bits x 32 bits et résultat sur 32 bits) qui a une latence de 4.
Les instructions de chargement (LDR) ont une latence de 3 cycles ou 4 cycles (voir Table).
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 présente un programme C et le programme assembleur ARM correspondant (On suppose que R3 contient au départ l'adresse de X[0] et R4 contient l'adresse de Y[0])

Q9) Quel est le temps d'exécution (en cycles) de la boucle de la table 1. Indiquer une optimisation possible et donner le nouveau temps d'exécution ?

Q10) Quel serait le temps d'exécution (en cycles par itération de la boucle initiale) avec un déroulage de boucle d'ordre 4 ?

Programme C	Programme assembleur

int X[100], Y[100], S, i ;
for (i=0; i<100; i++)
S+=X[i]*Yi];
	MOV R5, 100
	MOV R0,#0
	Boucle :LDR R1, [R3], #4
	LDR R2, [R4], #4
	MUL R1,R1,R2
	ADD R0,R0,R1
	SUBS R5,R5,#1
	BGT Boucle

Table 1 : Code C et assembleur ARM

ANNEXE : Modes d'adressage et latences des instructions LDR

Mode d'adressage	Assembleur	Action	Latence
Déplacement 12 bits, Pré-indexé	[Rn, #déplacement]	Adresse = Rn + déplacement	3
Déplacement 12 bits, Pré-indexé avec mise à jour	[Rn, #déplacement] !	Adresse = Rn + déplacement Rn ← Adresse	4
Déplacement 12 bits, Post-indexé	[Rn], #déplacement	Adresse = Rn Rn ← Rn + déplacement	4