Université Paris Sud 2006-2007
Licence d'Informatique (L313)

ARCHITECTURE DES ORDINATEURS
Corrigé Examen Décembre 2006
3H – Tous documents autorisés

EXECUTION D'INSTRUCTIONS

On utilise un sous ensemble du jeu d'instructions ARM décrit en annexe.

Q1) En partant à chaque fois du contenu de la table 1, donner le contenu des registres et des cases mémoire modifié après exécution des instructions ARM ou séquences d'instructions

```
a)    ADD R8,R3,R4                                  R8= 80FFDCBA
b)    MUL R9,R6,R7   // multiplication entière     R9=00000600
c)    AND R10,R3,R4                                 R10= 81002200
d)    EOR R11,R3,R4  // Ou exclusif                 R11 = 7EFF98BA

e)    LDR R8, [R0]          // R8 = 1
      LDR R9,[R0, #4]       // R9 = 2
      ADD R10, R8, R9       // R10 = 3

f)    STR R6, [R1,  #-4] !        // Mem(1FFC) = 20 ; R1 = 1FFC
      STR R7, [R1,  #-4] !        // Mem(1FF8) = 30 ; R1 = 1FF8
      LDR R8, [R1],#4             // R8 = Mem (1FF8)= 30 ; R1 = 1FFC
      LDR R9, [R1], #4            // R9 = Mem (1FFC)= 20 ; R1 = 2000
      SUB R10, R8,R9              // R10 = 30 – 20 = 10

R8 = 30, R9 = 20, R10 = 10, Mem (1FF8 ) = 30 , Mem (1FFC) = 20
```

Q2) Donner l'instruction ou la suite des instructions ARM pour effectuer les actions suivantes :

a) Mettre à zéro le registre R1
b) Mettre à zéro les 1000 octets commençant à l'adresse mémoire $F0000000_H$ (en supposant que le registre R2 contient $F0000000_H$)
c) Multiplier par 17 le contenu du registre R3
d) Multiplier par 119 le contenu du registre R4

```
a)
    MOV R1, #0
b)
        MOV R0, #0
        MOV R1,#250 // 250 mots de 32 bits = 1000 octets
Boucle :STR R0, [R2],#4
        SUBS R1,R1, #1
        BGT Boucle
c)
        ADD R3, R3, R3 LSL #4
d)
        ADD R4, R4, R4 LSL #4       // x 17
        RSB  R4, R4, R4 LSL #3      // 7 x 17 = 119
```

Ou RSB R0, R4, R4 LSL # 7 // 127 * R4 dans R0
SUB R4, R0, R4 LSL #3 // 127 R4 – 8 R4 = 119 R4

Registre	Contenu (hexa)
R0	00001000
R1	00002000
R2	00001016
R3	81003210
R4	FFFFAAAA
R6	00000020
R7	00000030

Adresse (hexa)	Contenu (hexa)
00001000	00000001
00001004	00000002
00001008	00000003
0000100C	00000004
00001010	00000005
00001014	00000006

Table 1 : contenu des registres du processeur (ARM) et de cases mémoire

EXECUTION DE PROGRAMME

Q3) Que fait la suite d'instructions ARM suivante (écrire le programme C correspondant en supposant que le contenu des variables x et y a été initialement chargé dans R1 et dans R2)

```
Boucle :CMP R1, R2
        SUBGT R1,R1,R2
        SUBLT R2,R2,R1
        BNE Boucle
```

```
While (x !=y) {
        If (x > y)  x=x-y ;
        If (x < y) y=y-x ; }
```

NB : il s'agit de l'algorithme d'Euclide pour calculer le PCD

IMPLANTATION MEMOIRE

Soit la déclaration de variables C suivante

```
unsigned char  toto [17] ;
short a,b,c, d, e, f ;
double w[10], y[8][8];
float z[10], foo[4][ 5];
int ouf, cest, fini ;
```

Q4) : Si l'on suppose que la variable toto[0] est à l'adresse 1000 0000H, donnez les adresses hexadécimales des variables toto [16], a, f, y[0][0], foo[0][0], fini

toto	0	0	10000000
toto(16)	16	10	10000010
a	18	12	10000012
b	20	14	10000014
c	22	16	10000016
d	24	18	10000018

e	26	1A		1000001A
f	28	1C		1000001C
w(0)	32		20	10000020
Y(0)(0)	112		70	10000070
Z(0)	624		270	10000270
foo(0)(0)	664		298	10000298
ouf	744	2E8		100002E8
cest	748	2EC		100002EC
fini	752	2F0		100002F0

Toto[16] : 1000 0010
A : 1000 0012
F : 1000001C
Y[0][0] : 10000070
Foo [0][0] : 10000298
Fini : 100002F0

MICROARCHITECTURES ET TEMPS D'EXECUTION DE PROGRAMMES.

La figure 1 donne le chemin de donnée d'un processeur NON pipeliné (jeu d'instructions ARM)

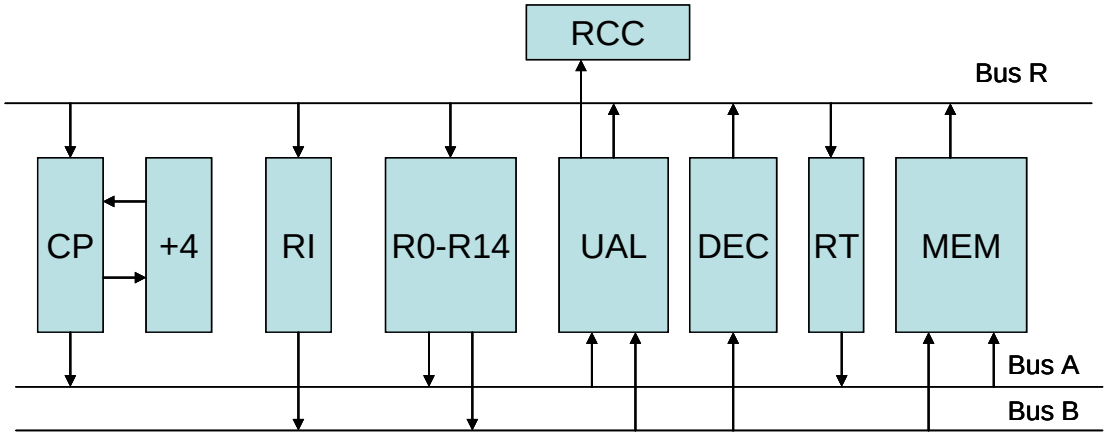

Figure 1 : microarchitecture non pipelinée.

Soient la liste des actions élémentaires qui peuvent s'exécuter en un cycle
LI : RI ←MEM(CP) et CP ← CP+4
DEC : RT←Décalage (Rs2)
UAL1 Rd ←Rs1 opération Rs2
UAL2 Rd ← Rs1 opération immédiat // immédiat est non signé sur 8 bits
UAL3 Rd ←Rs1 opération RT
CA1 RT← Rs1 + déplacement // déplacement sur 12 bits, extension de signe sur 32 bits
CA2 (RT et Rd) ← Rs1 + déplacement
CA3 RT←Rs1+Rs2
CA4 CP←CP + déplacement

LM1 Rd ← MEM(RT)
LM2 Rd ← MEM(Rs1)
EM1 MEM (Rs1) ← Rs2
EM2 MEM (RT) ← Rs2
NOP Décodage : condition fausse

Q5) : Donner le temps d'exécution de chacune des instructions suivantes (en précisant la suite des actions élémentaires):

a) ADD R2,R1,R0
b) ADD R3, R1,#4
c) ADD R4, R1, R2 LSL#4
d) LDR R6, [R1,#4]
e) LDR R7, [R1,#4] !
f) LDR R8, [R1],#4
g) BEQ déplacement (condition vraie)
h) BEQ déplacement (condition fausse)

Instruction					Total
ADD R2,R1,R0	LI	UAL1			2
ADD R3, R1,#4	LI	UAL2			2
ADD R4, R1, R2 LSL#4	LI	DEC	UAL3		3
LDR R6, [R1,#4]	LI	CA1	LM1		3
LDR R7, [R1,#4] !	LI	CA2	LM1		3
LDR R8, [R1],#4	LI	LM2	CA2		3
BEQ déplacement (V)	LI	CA4			2
BEQ déplacement (F)	LI	NOP			2

CACHES.

On suppose que le processeur utilisé a un cache données de 16 Ko, avec des blocs de 64 octets.
Le processeur a des adresses sur 32 bits.
On considère le programme suivant

```
double X[4096], Y[2048];
for (i=1 ; 0<2048 ; i++)
      Y[i] = X[i+2048] – X[i] ;
```

Les tableaux X et Y sont rangés successivement en mémoire à partir de l'adresse 1000 0000H (adresse de X[0].)

Q6) 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'étiquettes

a) s'il est à correspondance directe
b) s'il est associatif quatre voies (quatre blocs par ensemble)

Le cache a 256 blocs de 64 octets. Il y a 6 bits pour l'adresse dans le bloc.
Pour la correspondance directe, il y a 8 bits d'index et 32 – 14 = 18 bits d'étiquette
Pour l'associativité 4 voies, il y a 6 bits d'index et 20 bits d'étiquette

Q7) Quel est le nombre total de défaut de caches lors de l'exécution du programme pour les deux cas suivants : a) correspondance directe, b) associativité quatre voies (quatre blocs par ensemble) si le cache est à écriture simultanée (write through) et écriture non allouée?

Adresse de X[0] : 1000 0000H
Adresse de X[2048] = 1000 0000 + 2048*8 octets= $1000\ 4000_H$
Adresse de Y [0] = 1000 0000 + 4096 *8 octets = $1000\ 8000_H$

En correspondance directe, X[0] va dans le bloc 0 du cache, X[2048] va dans le bloc 0 et Y[0] vont dans le bloc 0 du cache
Il y a donc 2 défauts de cache par itération pour la lecture de X[i+2048] et X[i] et un défaut par itération en écriture pour Y[0], soit 3 défauts par itération.

Avec l'associativité 4 voies, il y a 2 échecs toutes les 8 itérations en lecture et 1 échec (réécriture) toutes les 8 itérations en écriture, soit 3/8 défauts par itération.

a) Correspondance directe
réécriture : 3 * 2048 = 6144 défauts

b) Associatif quatre voies
réécriture : 2048*(3/8) = 768 défauts

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])

Q8) Quel est le temps d'exécution (en cycles) de la boucle ci-dessus. Indiquer une optimisation possible et donner le nouveau temps d'exécution ?

12 cycles.
En remontant l'instruction SUBS, on obtient 11 cycles.
Si, en plus, on utilise LDR R2, [R4] (3 cycles) et ADD R4,R4,#4, on gagne un cycle supplémentaire et on obtient 10 cycles

Q9) Quel serait le temps d'exécution (en cycles par itération de la boucle initiale) avec un déroulage de boucle d'ordre 4 ?

Il n'y a plus de suspensions : 18 cycles pour 4 itérations, soit 4,5 cycles/itération

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

1 Loop :LDR R1, [R3], #4	1 Loop :LDR R1, [R3], #4	1 Loop :LDR R1, [R3], #4	1 Loop :LDR R1, [R3], #4
2 LDR R2, [R4], #4	2 LDR R2, [R4], #4	2 LDR R2, [R4]	2 LDR R2, [R4], #4
3	3 SUBS R5,R5,#1	3 SUBS R5,R5,#1	3 LDR R7, [R3], #4
4	4	4 ADD R4,R4,#4	4 LDR R8, [[R4], #4
5	5	5 MUL R1,R1,R2	5 LDR R9, [R3], #4
6 MUL R1,R1,R2	6 MUL R1,R1,R2	6	6 LDR R10, [R4], #4
7	7	7	7 LDR R11, [R3], #4
8	8	8	8 LDR R12, [R4], #4
9	9	9 ADD R0,R0,R1	9 MUL R1,R1,R2
10 ADD R0,R0,R1	10 ADD R0,R0,R1	10 BGT Loop	10 MUL R7,R7,R8
11 SUBS R5,R5,#1	11 BGT Loop		11 MUL R9,R9,R10
12 BGT Loop			12 MUL R11,R11,R12
			13 ADD R0,R0,R1
			14 ADD R0,R0, R7
			15 ADD R0,R0, R9
			16 ADD R0,R0,R11
			17 SUBS R5,R5,#1
			18 BGT Boucle

ANNEXE : Sous ensemble du jeu d'instructions ARM utilisable

On rappelle que le jeu d'instructions ARM a 16 registres de 32 bits, de R0 à R15. R0 est un registre normal. Le compteur de programme CP est R15. R14 reçoit les adresses de retour des fonctions.
Toutes les instructions sont à exécution conditionnelle. Par exemple, SUBGT R1, R2, R3 signifie que la soustraction est exécutée si la condition GT (plus grand) contenue dans le registre code condition est vrai, et l'addition se transforme en NOP si la condition est fausse. SUBLT correspond à la condition LT (plus petit). Les conditions possibles sont GT, LT, EQ (égal), NE (différent), GE (plus grand ou égal), LE (plus petit ou égal), etc.
Les instructions arithmétiques et logiques ne positionnent pas le registre code condition, sauf si bit S est positionné. Par exemple, SUB R1,R2,R3 ne positionne pas le registre code condition alors que SUBS R1,R2,R3 positionne le code condition. L'instruction CMP positionne le registre code condition.
On rappelle que le contenu du deuxième opérande peut être le contenu d'un registre décalé de n positions. Ex : ADD R1, R2, R3 LSL #4 correspond à R1 <= R2 + 16 * R3
Les modes d'adressage des instructions LDR avec les latences correspondantes sont données dans la table 3. La table 4 donne les instructions utilisables.

Table 2 : Modes d'adressage et latences des instructions LDR et STR (mots de 32 bits)

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

Table 3 : Instructions utilisables

		Positionnent les codes condition
Arithmétiques et logiques	ADD, SUB, ORR, EOR, AND, MUL, MOV	NON
Arithmétiques et logiques	CMP, ADDS, SUBS, ORRS, EORS, ANDS	OUI
Branchement	BGT, BEQ, BLT, BGE, BNE,BLE	
Instructions mémoire	LDR et STR (32 bits)	