ARCHITECTURE DES ORDINATEURS
Corrigé Examen Décembre 2011
3H – Tous documents autorisés
Les questions sont indépendantes

On utilise le jeu d'instructions ARM.

PROGRAMMATION ASSEMBLEUR

PREMIERE PARTIE

Soit le code C

```
int a, b;
short c;
unsigned char d;   // Les adresses de a, b, c, d sont initialement
                   // dans R1, R2, R3, R4

a= (int) c;
b= (int) d
```

Les instructions nécessaires et les modes d'adressage du jeu d'instructions ARM sont rappelés en annexe.

Question 1) Ecrire le code assembleur ARM correspondant au code C

```
LDRSH R5, [R3]
STW R5, [R1]
LDRB R5, [R4]
STW R5, [R2]
```

DEUXIEME PARTIE

Soit le code assembleur ARM

```
            MOV R3, 6
            MOV R1,#0
            STR R1, [R4], 4
            MOV R2, #1
            STR R2, [R4], 4
            SUBS R3, R3 , #2
   Boucle:  ADD R5 ,R1,R2
            MOV R1,R2
            MOV R2, R5
            STR R5, [R4], 4
            SUBS R3, R3 , #1
            BGT Boucle
```

Question 2) En supposant que R4 contient au départ l'adresse d'un tableau T[6], donner le contenu du tableau en fin d'exécution.

Ecrit dans un tableau T[6] les 6 premiers nombres de Fibonnaci.
T[0] = 0 ;
T[1] = 1 ;
T[2] = 1 ;
T[3] = 2;
T[4] = 3 ;
T[5] = 5 ;

TROISIEME PARTIE

On suppose que les registres ARM R0, R1 et R2 contiennent respectivement les variables (int) x, y et s.
Soit les programmes assembleurs P1 et P2

```
P1)                                     P2)
      MOV R2,#0                               MOV R2,#0
      CMP R0,#4                               CMP R0,#7
      CMPEQ R1,#5                             CMPNE R1,#5
      MOVEQ R2,#3                             MOVEQ R2,#5
```

```
Programme P1
if (x==4 && y==5) S=3;
      Else S=0;

Programme P2
if (x!=7 || y==5) S=5 ;
      Else S=0 ;
```

```
Programme 1

      MOV R2,#0   //
      CMP R0,#4   // EQ si R0=4, sinon faux
      CMPEQ R1,#5 // compare R1 et 5 si R0=4 sinon NOP
      MOVEQ R2,#3 // R3=3 si R0=4 et R1=5. Sinon NOP (donc R2=0)
```

```
Programme 2
Cas R=7
      MOV R2,#0
      CMP R0,#7 // EQ si R0=7
      CMPNE R1,#5 est un NOP

      MOVEQ R2,#5 // R2 = 5 (EQ est vrai)
```

Cas R0 !=7

```
      MOV R2,#0
      CMP R0,#7 // NE car R0 !=7
      CMPNE R1,#5 // compare R1 et 5

      MOVEQ R2,#5 // si R1=5 alors R2=5
```

D'ou le OU des deux conditions R0!=7 et R1=5

Question 3) Donner le code C correspondant aux programmes P1 et P2

CACHES

On suppose que le processeur utilisé a un cache données de 64 Ko, avec des lignes de 64 octets.
Le cache utilise la réécriture avec écriture allouée (**il y a des défauts de cache en écriture**)
Le processeur a des adresses sur 32 bits.
Le programme travaille sur deux tableaux de doubles X[N] et Y[N]
&X[0] est $F000\ 0000_H$
&Y[0] =&X[N-1] +8

Question 4) Quel est pour ce cache le nombre de bits pour l'adresse dans la ligne, le nombre de bits d'index et le nombre de bits d'étiquette dans les deux cas suivants : a) correspondance directe, b) associativité quatre voies (quatre lignes par ensemble) avec remplacement par LRU.

Adresse dans la ligne : 6 bits
Il y a 1024 lignes
Correspondance directe
- Adresse dans bloc : 6 bits
- Index : 10 bits
- Etiquette : 16 bits

Associativité 4 voies
- Adresse dans le bloc : 6 bits
- Index : 8 bits
- Etiquette : 18 bits

Question 5) Dans quelles lignes vont les flottants X[0] et Y[0] en correspondance directe pour N=128, N= 1024 et N=8192 ?

&X[0] = 1111 0000 0000 0000 | 0000 0000 00|00 0000
N= 128
&Y[0] = 1111 0000 0000 0000 | 0000 0100 00|00 0000
X[0] va dans la ligne 0 et Y[0] va dans la ligne 16
N=1024
&Y[0] = 1111 0000 0000 0000 | 0010 0000 00|00 0000
X[0] va dans la ligne 0 et Y[0] va dans la ligne 128
N= 8192
&Y[0] = 1111 0000 0000 0001 | 0000 0000 00|00 0000
X[0] va dans la ligne 0 et Y[0] va dans la ligne 0

Question 6) Quel est le nombre total de défauts de caches lors de l'exécution des boucles P1, P2 et P3 pour les deux cas suivants : a) correspondance directe, b) associativité quatre voies

Programme P1
```
double X[8192], Y[8192] ;
for (i=0 ; i<8192 ; i++)
      Y[i] =  X[i];
```

Programme P2
```
double X[1024], Y[1024] ;
for (i=0 ; i<1024 ; i++)
      Y[i] = X[i] + Y[i] ;
```

Programme P3

```
double X[1024], Y[1024] ;
 for (i=0 ; i<1020 ; i++)
       Y[i] =X[i] + X[i+4];
```

1)
double X[9192], Y[8192] ;

```
for (i=0 ; i<8192 ; i++)
```

Y[i] = X[i];
Un bloc contient 8 doubles.
Il y a 2 défauts de cache par itération en CD *(car X[0] et Y[0] vont dans la ligne 0)* et 2 défauts toutes les 8 itérations en 4-voies (*X[0] et Y[0] vont dans une voie différente*)
CD = 16 384
4 voies = 2048

2)
Double X[1024], Y[1024] ;

```
for (i=0 ; i<1024 ; i++)
```

Y[i] = X[i] + Y[i] ;
En CD, il y a 2 défauts de cache toutes les 8 itérations, soit un total de 256 *(car X[0] et Y[0] vont dans des lignes différentes et il n'y a pas de défaut en écriture car la ligne a été chargée par la lecture)*
En 4 voies, il y a 2 défauts de cache toutes les 8 itérations, soit un total de 256 (pas de problème car 4 voies)

3) Double X[1024], Y[1024] ;

```
for (i=0 ; i<1020 ; i++)
```

Y[i]°=X[i] + X[i+4];

Même situation qu'en 2). Les chargements X[i] et X[i+4] utilisent une même ligne lorsque i=0,1,2,3 mod (8) et une ligne et la ligne suivante lorsque i=4,5,6,7 mod (8). Les lignes X sont donc chargées une seule fois, comme dans le cas n°2.
Il n'y a pas de conflits entre X et Y en CD

En CD, il y a 2 défauts de cache toutes les 8 itérations, soit un total de 256
En 4 voies, il y a 2 défauts de cache toutes les 8 itérations, soit un total de 256

EXECUTION DE BOUCLES

On ajoute au jeu d'instructions ARM 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 3 cycles pour les instructions flottantes. Les branchements ne sont pas retardés.

LF	2	LF ft, adresse	ft ← MEM [adresse]
SF	1	SF ft, adresse	ft → MEM [adresse]
FADD	3	FADD fd, fs,ft	fd ← fs + ft (addition flottante simple précision)
FSUB	3	FSUB fd, fs,ft	fd ← fs - ft (soustraction flottante simple précision)

FMUL	3	FMUL fd, fs,ft	fd ← fs * ft (multiplication flottante simple précision)

Figure 1: Instructions flottantes ajoutées (Ce ne sont pas les instructions ARM). Les instructions LF et SF ont les mêmes modes d'adressage que les instructions entières

Question 7) Sans optimisation, quel est le temps d'exécution (en cycles par itération) et le temps d'exécution total du programme de la table 1. (Au démarrage R3 contient l'adresse du tableau X et R4 contient l'adresse du tableau Y)

Table 1 : Programme C et programme assembleur

float X[100], Y[100], S;
int i ;
for (i=0; i<100; i++)
S+=X[i]*X[i] + Y[i]*Y[i];
	MOV R5, #100
	FSUB F0, F0, F0
	Boucle :LF F1, [R3], 4
	LF F2, [R4], 4
	FMUL F1,F1,F1
	FMUL F2,F2,F2
	FADD F0,F0,F1
	FADD F0,F0,F2
	SUBS R5,R5,#1
	BGT Boucle
	SF F0, (adresse de S)

	`MOV R5,#100`
	`FSUBF0,F0,F0`
`Boucle 1L`	`LF` **`F1`**`, [R3], 4`
2	`LF` **`F2`**`, [R4], 4`
3	`FMUL F1,`**`F1`**`,F1`
4	`FMUL F2,`**`F2`**`,F2`
5	
6	`FADD F0,F0,F1`
7	
8	
9	`FADD F0,F0,F2`
10	`SUBS R5,R5,#1`
11	`BGT Boucle`
	SF F0, (adresse de S)

11 cycles/itérations

Temps total : 11*100+3 = 1103 cycles

Question 8) Optimiser la boucle de la table 1 et donner le nouveau temps d'exécution.

	`MOV R5,#100`
	`FSUB F0,F0,F0`
	`FSUB F3,F3,F3`
`Boucle 1L`	`LF F1, [R3], 4`
2	`LF F2, [R4], 4`
3	`FMUL F1,F1,F1`
4	`FMUL F2,F2,F2`

5	`SUBS R5,R5,#1`
6	`FADD F0,F0,F1`
7	`FADD F3,F3,F2`
8	`BGT Boucle`
	`FADD F0,F0,F3`
	SF F0, (adresse de S)

8 cycles/itérations

Temps total : 8*100 + 7 = 807 cycles.

PIPELINE

Un processeur a les pipelines suivants :

Instructions arithmétiques	LI1	LI2	DI	LR	EX1	EX2	ER	
Lecture mémoire (Load)	LI1	LI2	DI	LR	EX1	AM1	AM2	ER

La signification des différents étages est

- LI1 et LI2 : phases d'acquisition (lecture) de l'instruction
- DI : décodage de l'instruction
- LR : lecture des registres
- EX1 : calcul adresse mémoire, adresse de branchement, début exécution (EX1).
- EX2 : fin exécution (EX2) des instructions arithmétiques
- ER : écriture du résultat dans un registre
- AM1 et AM2 : phases accès cache données

Question 9 : Donner les latences entre instruction producteur et instruction consommateur en nombre de cycles pour les quatre cas ci-dessous en supposant que tous les bypass nécessaires soient implémentés. (NB : la valeur n signifie que deux instructions dépendantes peuvent se suivre aux cycles i et i+n)

	Instruction producteur	Instruction consommateur	Latence scalaire
1	ADD **R1**,R2,R3	ADD R4, **R1**, R6	2
2	SUB **R1**,R2,R3-	ST **R1**, 0 (R5)	1
3	LDW **R1**, 4(R2)	XOR R8,**R1**,R7-	3
4	LDW **R1**, 4(R2)	ST **R1**, 0 (R5)	2

1)

LI1	LI2	DI	LR	EX1	**EX2**	ER			
	LI1	LI2	DI	LR		**EX1**	EX2	ER	

2)

LI1	LI2	DI	LR	EX1	**EX2**	ER			
	LI1	LI2	DI	LR	EX1	**AM1**	AM2	ER	

3)

LI1	LI2	DI	LR	EX1	AM1	**AM2**	ER			
	LI1	LI2	DI	LR			**EX1**	EX2	ER	

4)

LI1	LI2	DI	LR	EX1	AM1	**AM2**	ER		
	LI1	LI2	DI	LR	EX1		**AM1**	AM2	ER

PREDICTION DE BRANCHEMENT

Question 10) Pour chacun des branchements suivants (considérés chacun séparément), proposer un type de prédicteur en justifiant parmi *Toujours pris / Prédicteur 1 bit / Prédicteur 2 bits*. Si plusieurs prédicteurs conviennent, on choisira celui qui utilise le moins de matériel.

a) Branchement B1,
P,P,P,P,P,*N*,*N*,*N*,*N*,*N*,*N*,*N*,P,P,P,P,P,P,P,P,P,P,*N*,*N*,*N*,*N*,*N*,P,P,P,P,P,P
Prédicteur 1 bit : branchements pris et non pris, avec longues suites identiques.

b) Branchement B2
P,P,P,P,*N*,P,P,P,P,P,P,P,*N*,P,P,P,P,P,P,P,*N*,*N*,*N*,*N*,*N*,P,*N*,*N*,*N*,*N*,*N*,P,*N*,*N*,*N*,*N*,*N*
Prédicteur 2 bits. Longue suite avec même comportement, interrompu par un seul branchement avec comportement opposé

c) Branchement B3
P,P,P,P,P,*N*,P,P,P,P,P,P,*N*,P,P,P,P,P,P,P,P,*N*,P,P,P,P,P,P,P,P,*N*,P,P,P,P,P,P,P
Prédiction statique pris. Le branchement est toujours pris, sauf exception

ANNEXE : instructions et modes d'adressage ARM utilisés

LDR	Chargement d'un mot dans un registre	Rd ← Mem32 (adresse)
STR	Rangement d'un mot	Mem32 (adresse) ← Rd
LDRB	Chargement d'un octet non signé	Rd ← Zéro|Mem8 (adresse)
LDRSB	Chargement d'un octet signé	Rd ← Extension signe|Mem8 (adresse)
STRB	Rangement d'un octet	Mem8 (adresse) ← Rd
LDRH	Chargement 16 bits non signés	Rd ← Zéro|Mem16 (adresse)
LDRSH	Chargement 16 bits signés	Rd ← Extension signe|Mem16(adresse)
STRH	Rangement 16 bits	Mem16 (adresse) ← Rd
MV	Transfert	Rd ← Rs
SUBS	Soustraction + positionnement code cond.	Rd ← Rs1 – Rs2 ; Positionne CC
BGT	Branchement si plus grand	CP ← NCP + déplacement si GT = vrai

Mode	Assembleur	Action
Déplacement 12 bits, Pré-indexé	[Rn, #deplacement]	Adresse = Rn + déplacement
Déplacement 12 bits, Pré-indexé avec mise à jour	[Rn, #deplacement] !	Adresse = Rn + déplacement Rn ← Adresse
Déplacement 12 bits, Post-indexé	[Rn], #deplacement	Adresse = Rn Rn ← Rn + déplacement