Examen Architecture - Janvier 2000

Tous documents autorisés.

Durée 3 heures.

Pour respecter l'anonymat, toutes les réponses seront données sur les feuilles d'examen, en respectant les formats de tables indiqués dans le texte.

Première partie : Arithmétique entière

On se propose de calculer le prédicat $p = (a < b)$, où a et b sont des entiers relatifs sur 8 bits, en utilisant une UAL qui effectue les calculs d'addition et soustraction et calcule correctement le bit d'overflow *ov*. La méthode proposée est de calculer $a - b$, et de positionner le prédicat à Vrai si le bit de signe *rn*-1du résultat est négatif.

Q1. Effectuer les opérations UAL 7-6 et 6-7 ; indiquer le bit de signe et le bit d' overflow. La méthode proposée est-elle exacte ?

Q2. Montrer que

p = !ov.*rn*-1 + ov.an-1

La preuve ne dépassera pas 5 lignes. Toute preuve plus longue ne sera pas corrigée.

Dans la suite, on considère l'architecture DSEX00 décrite en annexe.

Deuxième partie : Architecture logicielle

Q3. Décrire

- l'adressage mémoire
- les branchements
- les particularités de l'architecture par rapport aux processeurs vu en cours et TD.

Q4. Ecrire

ce 0x12345678 dans R1

- une séquence d'instructions qui place 0x12348675 dans R1
- une instruction qui réalise la macro BA dep21 (branchement inconditionnel).

Q5. Comment implémenter l'appel et le retour de sous-programme ?

Q6. En supposant que R1 = a et R2 = b, écrire le code réalisant a = max (a,b) sans branchement.

Q7. On considère le code C suivant :

```
char message[100], cle[8] ;
int i, j ;
j = 0 ;
for ( i = 0 ; i < 100 ; i++) {
        message[i] = message[i] + cle[j];
        j = j + 1 ;
        if (j == 8) j = 0 ;
}
```

Ecrire le code assembleur correspondant. On supposera que R1 contient l'adresse de message[0] et R2 l'adresse de cle[0] ; on utilisera impérativement les associations variables- registres suivantes : R11 pour message[i], R12 pour cle[i], R3 pour i et R4 pour j.
NB : un programme non commenté ne sera pas corrigé.

Troisième partie : Microarchitecture

Le chemin de données est décrit fig. 1.
L'UAL peut effectuer
Sr = Ea + Eb ou Sr = Ea ou Sr = Ea + 4

Q8. Décrire les transferts de données pour
a/ la lecture de l'instruction.
b/ ADD Rd, Ra, Rb
c/ LD Rd, Imm(Ra)
d/ ST Rd, Imm(Ra)
e/ JMP Ra, Rb

Attention : préciser clairement les transferts qui ont lieu pendant le même cycle.

Q9. Décrire le PLA des commandes (table 1). Le champ Ecr. Regs décrit les registres écrits au cours du cycle.

Instruction	UAL	Bus A	Bus B	Bus R	Ecr. Regs	Mux CP	Mux AD	Mux B	Mem
a/									
b/									
C/									
d/									
e/									
f/									

Table 1. PLA des commandes. Une ligne = 1 cycle. **On peut insérer des lignes supplémentaires**.

Q10. Définir l'automate de séquencement.

Quatrième partie : Pipeline

Toutes les instructions, sauf FMOV, FADD et FMUL, exécutent toutes les phases du pipeline suivant :

LI : lecture de l'instruction.

DI : décodage et lecture des opérandes dans le banc de registre entier.

EX : calcul pour les instructions arithmétiques (type A) ; calcul d'adresse (type M) ; calcul de la condition pour les instructions CMOV et type B ; calcul d'adresse de branchement (type B et J).

MEM : accès mémoire (type M).

RR : Ecriture du résultat dans le banc de registres entier ou flottant (type A et LD/LDB) ou flottant (LDF).

Les instructions FMOV, FADD et FMUL exécutent toutes les phases du pipeline suivant :

LI : lecture de l'instruction.

DI : décodage et lecture des opérandes dans le banc de registre flottant.

EX1, EX2, EX3 : 3 cycles d'exécution de l'opération flottante ; aucun résultat n'est disponible avant la fin de EX3.

RR : Ecriture du résultat dans le banc de registres flottants.

Les bancs de registres entiers et flottants ne permettent pas l'écriture avant la lecture.

Q11. Pour les trois séquences ci-dessous, quel est le nombre de suspensions nécessaire si aucune anticipation n'existe ?

a/
```
FLD F1, 0(R1)
FADD F3, F2, F1
```

b/
```
FADD F3, F2, F1
FMUL F5, F4, F3
```

c/
```
FADD F3, F2, F1
FST F3, 0(R1)
```

Q12. Même question si toutes les anticipations possibles sont réalisées.

Q13. Quelle est le délai de branchement ?

Ordonnancement d'instructions

Les latences des instructions en cas de dépenadance sont :
- opération flottante vers opération flottante : 4 cycles (3 suspensions)
- opération flottante vers FST : 4 cycles
- FLD vers opération flottante : 3 cycles
- autres cas 1 cycle.

On suppose que le branchement s'effectue sans délai.

Les unités flottantes sont complètement pipelinées.

On considère la boucle

```
float x[1000], z [1000];
for (i = 1 ; i < 999 ; i++)
        z[i] = (x [i-1] + x [i+1])*0.25;
```

NB : float = flottant simple précision.

La boucle est compilée par le code suivant :

```
FADD    F3, F1, F2
FMUL    F3, F3, F0
FST     F3, 0(R3)
FMV     F1, F2
FLD     F12, 4(R1)
FADD    F13, F11, F12
FMUL    F13, F13, F0
FST     F13,4(R3)
ADD     R1, R1, 8
ADD     R3, R3, 8
FMV     F11, F12
CMPLE   R2, R10, R3
BEQ     R2, bcl
```

A l'entrée dans la boucle, R1 = adresse de x[2], R3 = adresse de z[1], R2 = adesse de z[998] ; F1 = x[0], F11 = x[1], F0 = 0.25.

Q12. Commenter le code assembleur et décrire les optimisations réalisées.

Q13. Calculer la performance du programme non modifié en
- cycles par itération
- MFLOPs, en supposant un temps de cycle de 5ns

Q14. Modifier le programme pour qu'il exécute une instruction à chaque cycle (sans ajouter d'instruction).

Quatrième partie : Caches

On considère un cache :
- taille 16 octets
- taille ligne 2 octets.

La mémoire est adressée par octets et les adresses sur 32 bits. Le cache est géré en écriture allouée (rappel : un défaut en écriture entraîne le chargement du bloc correspondant), en réecriture (write-back) et en LRU.

Q14. Donner la taille en bits des champs de l'adresse (voir table 2) pour les trois cas : correspondance directe (CD), associativité par ensembles de 2 blocs (2W), associativité par ensembles de 4 blocs (4W).

	etiquette	numéro d'ensemble	déplacement
CD			
2W			
4W			

Table 2 : décomposition des adresses mémoire

Q15. On considère la trace mémoire de la table 3, qui décrit les accès mémoire successifs. Pour chaque accès, indiquer s'il s'agit d'un succès ou d'un échec, et donner l'état du cache à la fin de l'exécution de la trace, en supposant que la mémoire contienne la valeur *a* à l'adresse *a* (ex : l'adresse 0x45 contient 0x45). On traitera uniquement le cas CD.

type	adresse	valeur écrite
lecture	0x00	
écriture	0x01	0xFF
lecture	0x08	
écriture	0x44	OxAB
lecture	0x4C	
lecture	0xA3	
lecture	0x00	
lecture	0x45	

Table 3 : trace d'exécution

Q16. Calculer le taux d'échec pour le code de la question 7, en CD et associativité 2-way

Q17. On dispose des données expérimentales suivantes :

	Pénalité d'échec p	Taux d'échec t
CD	4 cycles	8,00%
2-way	6 cycles	4,00%
4-way	8 cycles	2,00%

Les chargements ont un CPI de 1,2 si succès, et 1,2 + p si échec. 20 % des instructions sont des chargements.

Calculer le CPI moyen.

Mémoire virtuelle

Q18. Répondre très brièvement aux questions suivantes :

2/ Quel type d'action produit un accès mémoire illégal ?

3/ Quelle est la fonction d'un TLB ?

4/ Les TLB sont-ils gérés par matériel ou par logiciel. ?

5/ Quels sont les avantages d'une gestion matérielle des défauts de TLB ?

6/ Quels sont les avantages d'une gestion logicielle des défauts de TLB ?

7/ La table des pages est-elle en général résidente en mémoire centrale ?

8/ Les adresses d'un programme assembleur utilisateur sont-elles virtuelles ou physiques ?

9/ Quels sont les avantages de l'organisation inverse de la table des pages ?

10/ Quel problème pose l'indexation virtuelle du cache de données ?

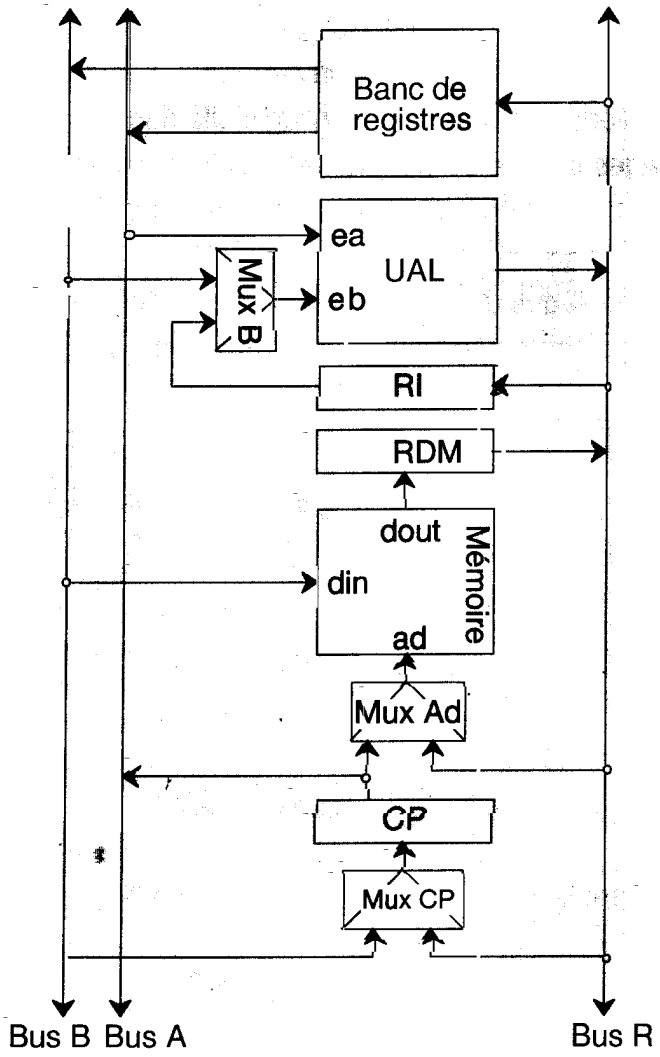

Figure 1 : Chemin de données

Annexe : Spécification de l'architecture DSEX00

DSEX00 est une architecture 32 bits, avec

- 32 registres entiers 32 bits. R0 est câblé à 0.
- 32 registres flottants 32 bits.
- La mémoire est organisée par octets en Big Endian et les adresses sont sur 32 bits.

Dans la table suivante, || désigne la concaténation : 0x12 || 0x34 = 0x1234

Mnémonique	Syntaxe	Description
ADD	ADD Rd, Ra, Rb ou imm8	Rd := Ra + Rb ou Rd := Ra + 0x000000 || imm8
SUB	SUB Rd, Ra, Rb ou	Rd := Ra - Rb ou Rd := Ra - 0x000000 || imm8
AND	AND Rd, Ra, Rb	Rd := Ra AND Rb bit à bit
OR	OR Rd, Ra, Rb	Rd := Ra OR Rb bit à bit
XOR	XOR Rd, Ra, Rb	Rd := Ra XOR Rb bit à bit
SLL	SLL Rd, imm5,Ra	Rd := Ra décalé à gauche de imm5 positions
SLR	SLR Rd, imm5,Ra	Rd := Ra décalé à gauche de imm5 positions avec remplissage à 0
SAR	SRA Rd, imm5,Ra	Rd := Ra décalé à gauche de imm5 positions avec extension de signe
CMPpred (voir table A2)	CMPxx Rd, Ra, Rb	si pred (Ra,Rb) Rd := 1 sinon Rd := 0
CMOVcond (voir table A3)	CMOVcond Rd, Ra, Rb	si cond (Ra) alors Rd := Rb
Bcond (voir table A3)	Bcond Ra, imm21	Si cond (Ra), CP := CP + ES (imm21)*4
BSR	BSR Ra, imm21	Ra:= CP, CP := CP + ES (imm21)*4
JMP	JMP Rd, Ra	Rd:= PC ; PC := Ra
LD	LD Rd, imm16(Ra)	Rd := Mem32 [Ra + ES(imm16)]
LDB	LDB Rd, imm16(Ra)	Rd := ES(Mem8 [Ra + ES(imm16)])
ST	ST Rd, imm16(Ra)	Mem32 [Ra + ES(imm16)] := Rd
STB	STB Rd, imm16(Ra)	Mem8 [Ra + ES(imm16)] := Rd
FLD	FLD Fd, imm16(Ra)	Fd := Mem32 [Ra + ES(imm16)]
FST	FST Fd, imm16(Ra)	Mem32 [Ra + ES(imm16)] := Fd
FADD	FADD Fd, Fa, Fb	Fd := Fa + Fb
FMUL	FMUL Fd, Fa, Fb	Fd := Fa * Fb
FMOV	FMOV Fd, Fa	Fd := Fa

LA LA Rd, Ra,imm16 Rd := Ra + ES(imm16)

LAH LA Rd, Ra,imm16 Rd := Ra + (imm16 || 0x0000)

pred	test
EQ	Ra = Rb
LE	Ra ≤ Rb signé
LT	Ra < Rb signé
LEU	Ra ≤ Rb non signé
LTU	Ra < Rb non signé

cond	test
EQ	Ra = 0
GE	Ra ≥ 0
GT	Ra > 0
LE	Ra ≤ 0
LT	Ra < 0