Université de Paris XI
Licence Informatique / FIIFO3 / IUP3
Année 2001/2002

Architecture des ordinateurs

Travaux dirigés[1]

[1]Les sujets des travaux dirigés sur machine seront distribués au début des séances concernées.

TD 1 : Architecture logicielle

1.1 Codage des entiers

On considère un codage sur 10 bits.

- Pour a) le codage des entiers naturels b) le codage en complément en 2 :
  - Quel est l'intervalle des nombres représentables ?
  - Donner une représentation décimale des nombres codés en binaire par 0101000010 et par 1101010010.
- Coder le nombre représenté en décimal par –65.

1.2 Organisation des données en mémoire

Alignement

- Caractériser les adresses d'un demi-mot (16 bits), d'un mot et d'un double mot, lorsqu'ils sont alignés avec une mémoire :
  - adressable par octets,
  - adressable par mots de 16 bits.

Big-Endian et Little-Endian

La déclaration en C suivante définit une occupation mémoire; les valeurs sont notées en hexadécimal en commentaire. Le placement est supposé aligné, i.e un nombre adéquat d'octets est sauté pour que chaque objet soit aligné sur ses bornes naturelles. La structure est, par exemple, implantée à partir de l'adresse 0.

```
struct      {
            int          a;       /*x'11121314'*/
            double       b;       /*x'2122232425262728'*/
            char*        c;       /*x'31323334'*/
            char         d[7];    /*'A', 'B', 'C', 'D', 'E', 'F', 'G'*/
            short        e;       /*x'5152'*/
            int          f;       /*x'61626364'*/
            } data;
```

- Donner le contenu des octets mémoire concernés, en supposant que la structure *data* est implantée à partir de l'adresse 0, pour les deux cas, big endian, et little endian.

1.3 Format des instructions

Format SPARC

La figure 4.2 du poly donne les formats des instructions arithmétiques et d'accès mémoire de l'architecture SPARC. La syntaxe assembleur est OP Rd, Rs1, Rs2 ou OP Rd, Rs1, imm_{13}. Quelques codes opérations sont :

Mnémonique	Format	Opération	Code Opération 1	Code Opération 2
ADD	3	Addition	10	000000
SUB	3	Soustraction	10	000100
LD	3	Chargement	11	000000
ST	3	Rangement	11	000100

- Quel est le nombre de registres généraux ?
- Donner le codage hexadécimal des instructions :

a) ADD R4, R1, R2 b) LD R2, R10, R3
c) SUB R20, R20, 1 d) ADD R4, R1, -1

*Format PowerPC

On veut coder l'architecture logicielle suivante:

- 32 registres entiers 32 bits,
- modes d'adressages registre + immédiat 16 bits, registre + registre, et leurs versions avec mise à jour,
- 16 instructions arithmétiques et logiques,
- 6 instructions différentes de chargement et 4 de rangement.

  - Proposer un format d'instruction.

1.4 Modèles d'exécution

On considère les trois architectures suivantes : M0 est une architecture à pile ; M1 une architecture à accumulateur ; M2 une architecture RISC.
Les trois machines disposent des instructions suivantes:

M0	M1	M2
ADD (sommet <- sommet + ss-sommet)	LOAD X (accu <-X)	LOAD Ri,X
SUB (sommet <- sommet - ss-sommet)	STORE X (X <- accu)	STORE Ri, X
PUSH X (sommet <- X)	ADD X (accu <- accu + X)	ADD Ri, Rj, Rk (Ri <- Rj + Rk)
POP X (X <- sommet)	SUB X (accu <- accu - X)	SUB Ri, Rj, Rk (Ri <- Rj - Rk)

Dans la machine à pile, une instruction arithmétique (ADD ou SUB) dépile les deux opérandes, effectue l'opération et empile le résultat.
Les variables A, B, C et D sont initialement en mémoire.

- Pour chaque architecture, écrire une séquence de code optimisée correspondant au programme :

  A := B + C;
  B := A + C;
  D := A - B;

- Comparer ces architectures en taille de code statique et en nombre d'accès mémoire.

TD 2 : Instructions UAL et instructions d'accès mémoire

On utilise dans ce TD les deux architectures logicielles MEXS (à la SPARC) et MEXP (à la POWER), décrites dans le document *Architectures et jeux d'instructions MEXS et MEXP*.

2.1 Immédiats

- Quelle est l'amplitude maximale des immédiats dans une instruction arithmétique ?
- Est-il nécessaire d'avoir une variante immédiate des instructions de soustraction ?
- Quelle est l'amplitude maximale d'un immédiat dans une instruction arithmétique ?
- Comment peut-on réaliser le chargement de 0x01000000 dans R1?
- Définir une pseudo-instruction MOV Ra, imm_{32} ; une pseudo-instruction est un ensemble de séquences d'instructions qui ne modifient pas d'autre registre que ceux specifiés dans la pseudo-instruction. La phase d'assemblage transforme la pseudo-instruction dans la séquence convenable.

2.2 Transferts Mémoire-Processeur

Le registre R10 contient 0x10000000. Le mot (32 bits) mémoire d'adresse 0x10000000 contient 0xABCD0000.

- Décrire le contenu des registres R1 à R5 après l'exécution du programme :

```
ldb R1, 0(R10)
ldsb R2, 0(R10)
ldh R3, 0(R10)
ldsh R4, 0(R10)
ld R5, 0(R10)
```

2.3 Parcours de tableaux

Soit la boucle :

```
int s, X[1000], Y[1000]
for (i = 0, i < 1000 ; i++)
  s = s + X[i] + Y[i] ;
```

- On suppose que R11 contient l'adresse de X[0], R12 l'adresse de Y[0] et R13 l'adresse de s. Ecrire en assembleur le corps la boucle (sans instructions de contrôle), en utilisant a) les modes d'adressage de MEXS b) les modes d'adressages supplémentaires de MEXP.
- Ecrire les initialisations correspondantes si X et Y sont implantés à partir des adresses 0x1000 0000 et 0x2000 0000 et la variable s à l'adresse 0x100.
- *Mêmes questions pour la boucle

```
int s, X[1000], Y[1000]
for (i = 0, i < 1000 ; i++)
  Y[i] = X[i] + X[i - 1] ;
```

TD 3 : Instructions de contrôle

On utilise dans ce TD les deux architectures MEXS et MEXP.

3.1 Conditionnelles

- Définir une pseudo-instruction CMP Ra, Rb ou CMPI Ra, imm16
- On considère les deux fragments de codes :

```
        XOR  R3, R3, R3
        CMP  R1, R2
        BLE  suite
        ADD R3, R3, 1
suite :
```

```
        XOR  R3, R3, R3
        CMP  R1, R2
        BLEU suite
        ADD R3, R3, 1
suite :
```

  Tracer l'exécution de ces codes pour les valeurs initiales a) R1 = 0x00000000 et R2 = 0x12345678 et b) R1 = 0x00000000 et R2 = 0xFFFFFFFF

- Ecrire la séquence d'instructions qui place dans un registre le maximum non signé de deux registres.

3.2 Boucles

Programme 1

Soit la boucle (voir TD2):

```
int s, X[1000], Y[1000]
for (i = 0, i < 1000 ; i++)
  s = s + X[i] + Y[i] ;
```

- On suppose que R11 contient l'adresse de X[0], R12 l'adresse de Y[0] et R13 l'adresse de s. Compléter les codes du TD 2 en ajoutant les codes de contrôle de boucle.
- *Mêmes questions pour la boucle

```
int s, X[1000], Y[1000]
for (i = 0, i < 1000 ; i++)
  Y[i] = X[i] + X[i - 1] ;
```

Pour les deux exercices suivants, on se limite à MEXS.

Programme 2

- Ecrire le programme qui recherche la première occurence du caractère 'A' dans un tableau de 10 caractères (adresse dans R11). Le programme retourne dans R1, soit l'index de la première occurence, soit -1 en cas d'échec.

***Programme 3**

- Ecrire un programme additionnant deux entiers relatifs représentés sur N octets. N est une constante, mutiple de 4, par exemple 64. Les opérandes sont initialement en mémoire dans N octets consécutifs à partir de l'octet OP1 pour l'opérande 1 et de l'octet OP2 pour l'opérande 2. Le résultat sera rangé sur N octets consécutifs à partir de l'octet RES. OVF est un octet qui sera positionné à 0 si le résultat est correct et 1 s'il y a dépassement de capacité.

TD 4 Procédures et fonctions

L'architecture logicielle utilisée est MEXS. Les programmes respecteront les conventions logicielles suivantes.

- L'adresse de retour est placée dans R31 ;
- Pour les procédures et fonctions qui ont au plus 4 paramètres, les paramètres sont passés par registres, de R2 à R5. La valeur d'une fonction est retournée dans R1.
- Le pointeur de pile d'exécution est R30, il pointe le premier octet libre du contexte courant. Les contextes sont alloués vers les adresses décroissantes.
- La sauvegarde de registres R6 à R13 est à la charge de l'appelante, celle des autres à la charge de l'appelée.

- Rappeler la définition et l'utilité des conventions logicielles.
- Comment s'effectuent l'appel et le retour de procédure ?
- Pourquoi l'immédiat dans JMPL est-il multiplié par 4 ?

4.1 Procédures terminales

Les variables locales sont stockées dans le contexte dans l'ordre de leur déclaration. On considère la procédure suivante.

```
void proc(void) {
    int u[3];
    int t[2];
/* initialisations */
    t[0] = u[0] + u[1];
    t[1] = u[0] - u[1];
}
```

- Quelle est la taille de la pile d'exécution de cette procédure ?
- Coder la procédure.
- Le langage C n'effectue aucun contrôle sur les indices de tableaux. Quel est l'état de la pile après l'exécution de la boucle :

```
for (i = 0 ; i < 5 ; i++)
   u[i] = 10 + i
```

4.2 Procédures non terminales

Soit la fonction `f` implantée à l'adresse 0x00001000 :

```
int f(int x, int y) {
    int k;
    k = x + y;
    return k;
}
```

- Coder la fonction `f`.
- On remplace dans `proc1` la ligne `t[0] =`... par `t[0] = f(u[0],u[1])`. Coder la nouvelle version de `proc1`.
- Donnez le contenu de la pile et des registres au cours de l'exécution.

4.3 *Passage de paramètres par référence

Soit la fonction `g` suivante, implantée à l'adresse 0x00002000 :

```
void g(int* v, int * w) {
    int i;
    w[0] = v[0] + v[1];
    w[1] = v[0] - v[1];
}
```

- Coder la fonction `g`.
- On remplace le corps de `proc1` par l'appel `g(u,t)`.Coder la nouvelle version de `proc1`.

TD 5 Caches

Tous les caches de ce TD sont gérés en écriture allouée et réécriture (write-back). Pour les caches associatifs, la politique de remplacement est LRU.

- Rappeler la défintion des politiques de réecriture et d'écriture allouée.

5.1 Trace d'exécution

On considère un cache de 32 octets ; la taille de la ligne est 4 octets. La mémoire est adressée par octets et les adresses sont sur 16 bits.

- Donner la taille en bits des champs de l'adresse : étiquette, numéro d'ensemble, déplacement, pour les trois cas : correspondance directe, associativité 2-way et associativité 4-way.

- On considère la trace mémoire de la table 1, qui décrit les accès mémoire successifs. On se place uniquement en correspondance directe. Pour chaque accès, indiquer s'il s'agit d'un succès ou d'un échec. Donner l'état du cache (on ne donnera que les octets modifiés) à la fin de l'exécution, en supposant que l'octet d'adresse a contient la valeur $a_{0:7}$ (exemple : l'adresse 0x00FE contient 0xFE).

Type	Adresse	Valeur
Lecture	0x0000	
Ecriture	0x0001	0x22
Lecture	0x0008	
Ecriture	0x0044	0x74
Lecture	0x004C	
Lecture	0x00A3	
Lecture	0x0002	
Lecture	0x0047	

Table 1: Trace mémoire

5.2 Accès tableaux

On considère une architecture possédant un cache de données de 8Koctets organisé en lignes de 32 octets. Les adresses sont sur 32 bits. Cet exercice sera traité avec deux types de cache différents: un cache à correspondance directe et un cache associatif par ensemble de 2 lignes.

On considère des tableaux de 4096 flottants simple précision (32 bits), implantés en mémoire aux adresses suivantes:

X	Y	Z	X1	Y1	X2	Y2
0x00010000	0x00014000	0x00018000	0x0001C000	0x00020000	0x00024000	0x00028000

- En correspondance directe, quels sont les éléments des tableaux X et Y qui peuvent occuper le mot 0 de la ligne 0, le mot 3, ligne 2 du cache ?
- Donner le nombre d'échecs et le taux d'échec dans le cache données pour les boucles suivantes (on suppose que les variables scalaires sont toujours en registre et que N = 4095):

B1:

```
do i = 0, N
  S = S + X(i) + Y(i)
enddo
```

B2:

```
do i = 0, N
  S1 = S1 + X1(i) + Y1(i)
  S2 = S2 + X2(i) + Y2(i)
enddo
```

B3:

```
do i = 0, N
  S1 = S1 + X1(i) + Y1(i)
enddo
do i = 0, N
   S2 = S2 + X2(i) + Y2(i)
enddo
```

B4:

```
do i = 0, N
  S1 = S1 + X(i) + Y(i)
  S2 = S2 + X(i) + Z(i)
enddo
```

TD 6 Micro-Architecture

6.1 Architecture 3 bus

DSEX99 est une architecture 32 bits, avec
- 32 registres entiers 32 bits (R0 est câblé à 0) ;
- un registre de lien LR ;
- 32 registres flottants 32 bits ;
- la mémoire est organisée par octets et les adresses sur 32 bits.

DSEX99 a de instructions UAL en format registre-registre et registre-immédiat (avec ES des 16 bits de poids faible de l'instruction). Les autres instructions considérées ici sont :

LD Rd, (Ra)	Rd <- Mem [Ra]
LDU Rd, (Ra)	Rd <- Mem [Ra] ; Ra <- Ra + 4
ST Rd, (Ra)	Mem [Ra] <- Rd
STU Rd, (Ra)	Mem [Ra] <- Rd ; Ra <- Ra + 4
CALL imm12	LR <- PC ; PC <- PC + ES(imm12)
BLR	PC <- LR

Le code opération est contenu dans les bits de poids fort de l'instruction, avec les codes de la table 2.

Instruction	Code opération
arithmétique, reg-reg	000x
arithmétique, reg-imm	010x
LD	0010
LDU	0110
ST	0011
STU	0111
CALL	1000 si condx vraie
BLR	1100

Table 2: Codes opérations de DSEX99

On considère une micro-architecture non-pipelinée implémentant DSEX99. Le chemin de données est décrit fig.1. L'UAL peut effectuer :

Sr = Ea + Eb ou Sr = Ea ou Sr = Ea + 4 ou Sr = Ea + 0

L'unité Uadd effectue +4 sur son entrée.

- Décrire les transferts de données associés à la lecture de l'instruction et à l'exécution des instructions : ADD Rd, Ra, Rb ; LD Rd, Ra ; LDU Rd, Ra ; ST Rd, Ra ; STU Rd, Ra ; BLR ; CALL depl

- Décrire les entrées et le contenu du PLA des commandes.
- Définir l'automate de séquencement.

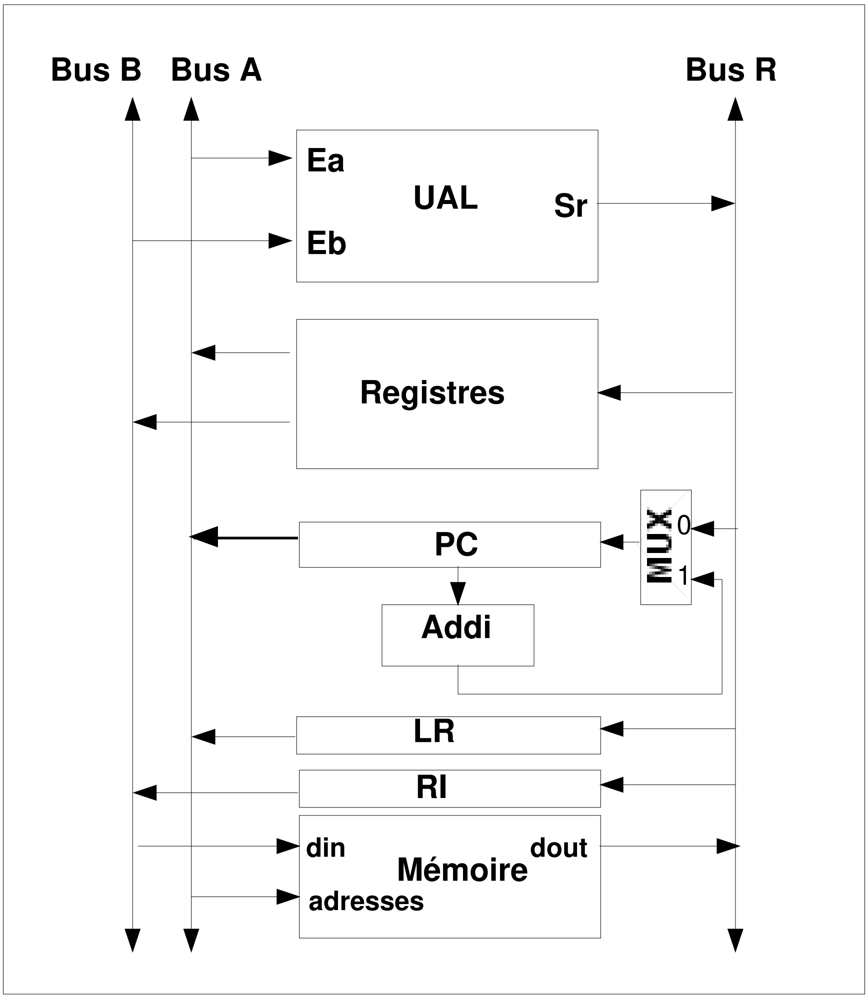

Figure 1: Chemin de données

6.2 *Microarchitecture pipelinée

On considère maintenant une implantation pipelinée de DSEX99. Les phases du pipeline sont **dans cet ordre** :
LI : lecture de l'instruction
DI : décodage et lecture des opérandes
MEM : accès mémoire
EX : calcul pour les instructions arithmétiques ; autoincrémentation
RR : Ecriture du banc de registres
Par rapport au pipeline classique vu en cours, les phases MEM et EX sont inversées.

- Modifier le chemin de données de la question précédente pour permettre l'exécution pipelinée
  - des instructions arithmétiques
  - des chargements et rangements sans autoincrémentation
  - des chargements et rangements avec autoincrémentation

TD 7 Pipeline

7.1 Ressources matérielles

On considère le pipeline classique vu en cours.

- Quel problème pose une architecture à cache unique instructions et données ?

Dans la suite, on supposera toujours que le cache de données et le cache d'instructions sont séparés (architecture Harvard)

7.2 Aléas de données et de contrôle

On considère une architecture RISC possédant 4 types d'instructions:

- A: arithmétiques et logiques,
- C: chargement: lecture mémoire,
- R: rangement: écriture en mémoire,
- B: branchement. Les branchements conditionnels se font sur code condition.

Les instructions d'accès mémoire utilisent les modes d'adressage registre-registre et registre-immédiat.

- Donner tous les cas possibles d'aléas de données entre deux instructions consécutives, et un exemple pour chaque cas.

On considère deux implantations pipelinées, l'une avec le pipeline d'instructions P1 et l'autre avec le pipeline d'instructions P2. Les différentes opérations élémentaires des pipelines sont données dans la table 3.

Dans les deux cas, on supposera que le banc de registres permet l'écriture avant la lecture. En conséquence, un registre écrit dans la phase RR est disponible dans le même cycle pour une lecture, sans mécanisme d'anticipation. Pour chacun des deux pipelines :

- On suppose qu'il n'existe pas de mécanisme d'anticipation (bypass). Pour chacun des deux pipelines, indiquer le nombre de suspensions nécessaires entre deux instructions successives (sauf branchements).
- Même question en supposant que tous les mécanismes d'anticipation possibles ont été implémentés.
- Quelle est la taille du délai de branchement ?

Pipeline P1	
LI	Lecture d'instructions
DI	Pour toutes les instructions, décodage de l'instructions et lecture des opérandes registre. Pour les types C et R, calcul de l'adresse mémoire. Pour le type B, calcul de l'adresse de branchement.
EX	Pour le type A, opération UAL. Pour le type R, lecture du 3ème opérande. Pour les types C et R, accès mémoire.
RR	Rangement du résultat arithmétique (A) ou écriture (C) dans le banc de registres.
Pipeline P2	
LI1	Début de lecture de l'instruction. Demande l'adresse.
LI2	Fin lecture de l'instruction.
DI	Décodage de l'instruction et lecture des opérandes registre.
EX	Pour le type A, opération UAL . Pour le type R, lecture du 3ème opérande . Pour les types C et R, calcul de l'adresse mémoire. Pour le type B, calcul de l'adresse de branchement.
M1	Pour les types C et R, début accès mémoire. Demande l'adresse uniquement.
M2	Pour les types C et R, fin accès mémoire.
RR	Rangement du résultat arithmétique (A) ou écriture (C) dans le banc de registre.

Table 3:

7.3 Performance

On considère deux implantations pipelinées d'une même architecture, comme dans la question précédente, avec les caractéristiques données dans la table 4.

	Pipeline A	Pipeline B
Nombre d'étages	6	10
Taux d'instructions produisant 3 suspensions	0%	5%
Taux d'instructions produisant 2 suspensions	5%	10%
Taux d'instructions produisant 1 suspension	10%	20%

Table 4:

- Calculer IPC pour chacun des deux pipelines.
- On suppose que A est implanté avec une fréquence de 500MHz. Evaluer le temps d'exécution d'un programme de 10^4 instructions.
- Quelle doit être la fréquence de B pour que B soit plus performant que A ?

TD 8 Ordonnancement d'instructions

8.1 Dépliement de boucles

Soit le programme C suivant :

```
float X[N], Y[N], S
for (i = 0 ; i < N ; i++)
  S = S + X[i]*Y[i] ;
```

Dans toute la suite, on suppose que R1 contient l'adresse de X et R2 celle de Y.
On considère l'architecture MEXP augmentée de :
- 32 registres flottants 32 bits F0 à F31,
- les instructions de la table 5.

Mnémonique	Syntaxe	Description
FLD	FLD Fd, Ra, Rb	Fd <- Mem (Ra + Rb)
	FLD Fd, imm$_{16}$(Ra)	Fd <- Mem (Ra + ES(imm$_{16}$))
FST	FST Fd, Ra, Rb	Fd -> Mem (Ra + Rb)
	FST Fd, imm$_{16}$(Ra)	Fd -> Mem (Ra + ES(imm$_{16}$))
FLDU FSTU	Variantes avec mise a jour	
FADD	FADD Fd, Fa, Fb	Fd <- Fa + Fb
FMUL	FMUL Fd, Fa, Fb	Fd <- Fa * Fb

Table 5:

Les latences sont :
- chargement flottant vers toute instruction flottante et rangement flottant : 3 cycles,
- addition flottante vers toute instruction flottante et rangement flottant : 4 cycles,
- multiplication flottante vers toute instruction flottante et rangement flottant : 5 cycles,
- autres cas: 1 cycle.

Il n'y a pas de délai de branchement.

- Ecrire un code optimisé pour le corps de la boucle. Quelle est sa performance en opérations flottantes par cycle ? En MFLOPS si on suppose un temps de cycle de 2ns ? On ne tiendra pas compte des initialisations.

- Dérouler 3 fois la boucle (soit quatre corps de boucle) et optimiser le code réultant. Mêmes questions.

8.2 Pipeline logiciel

- Tracer quatre itérations du code ci-dessous en supposant qu'initialement R1 contient l'adresse de X[2] et R2 celle de Y[2].

```
boucle :  FADD F3, F3, F4
          FMUL F4, F1, F2
          FLDU F1,4(R1)
          FLDU F2,4(R2)
          BCA boucle
```

- Compléter ce code pour qu'il réalise la boucle de l'exercice 8.1.
- Pourquoi ce type d'optimisation de code est-il appelé pipeline logiciel ?

Architectures logicielles MEXS et MEXP

Les deux architectures ont :
- un chemin de données 32 bits,
- 32 registres 32 bits avec R0 câblé à 0 ; l'écriture dans R0 ne modifie aucun registre général.
- un registre code condition RCC avec les drapeaux usuels,
-une mémoire organisée par octets, avec 4Goctets adressables en big endian.

Dans les tableaux suivants, $\Vert$ désigne la concaténation des chaînes de bits.

9 Jeu d'instructions de MEXS

Mnémonique	Syntaxe	Description
ADD	ADD Rd, Ra, Rb	Rd <-Ra + Rb
	ADD Rd, Ra, imm_{16}	Rd <-Ra + ES(imm_{16})
ADDcc	ADDcc Rd, Ra, Rb	Rd <-Ra + Rb, RCC affecté
	ADDcc Rd, Ra, imm_{16}	Rd <-Ra + ES(imm_{16}), RCC affecté
ADDX	ADDX Rd, Ra, Rb	Rd <-Ra + Rb + CF
	ADDX Rd, Ra, imm_{16}	Rd <-Ra + ES(imm_{16}) +CF
ADDXcc	ADDXcc Rd, Ra, Rb	Rd <-Ra + Rb + CF, RCC affecté
	ADDXcc Rd, Ra, imm_{16}	Rd <-Ra + ES(imm_{16})+ CF, RCC affecté
SUB	SUB Rd, Ra, Rb	Rd <-Ra - Rb
SUBcc, SUBX, SUBXcc	-	-
AND	AND Rd, Ra, Rb	Rd <-Ra AND Rb
	AND Rd, Ra, imm_{16}	Rd <-Ra AND 0x0000 $\Vert$ imm_{16}
OR	OR Rd, Ra, Rb	Rd <-Ra OR Rb
	OR Rd, Ra, imm_{16}	Rd <-Ra OR 0x0000 $\Vert$ imm_{16}
XOR	XOR Rd, Ra, Rb	Rd <-Ra XOR Rb
	XOR Rd, Ra, imm_{16}	Rd <-Ra XOR 0x0000 $\Vert$ imm_{16}
SLL	SLL Rd, imm_5, Ra	Rd <- Ra décalé à gauche de imm_5
SRL	SRL Rd, imm_5, Ra	Rd <- Ra décalé à droite (logique) de imm_5
SRA	SRA Rd, imm_5, Ra	Rd <- Ra décalé à droite (arithmétique) de imm_5
LDB	LDB Rd, (Ra + Rb)	Rd <- 0x000000 $\Vert$ Mem8 (Ra + Rb)
	LDB Rd, imm_{16}(Ra)	Rd <- 0x000000 $\Vert$ Mem8 (Ra + ES(imm_{16}))
LDSB	LDSB Rd, (Ra + Rb)	Rd <- ES(Mem8 (Ra + Rb))
	LDSB Rd, imm_{16}(Ra)	Rd <- ES(Mem8 (Ra + ES(imm_{16})))
LDH	LDH Rd, (Ra + Rb)	Rd <- 0x0000 $\Vert$ Mem16 (Ra + Rb)
	LDH Rd, imm_{16}(Ra)	Rd <- 0x0000 $\Vert$ Mem16 (Ra + ES(imm_{16}))
LDSH	LDSH Rd, (Ra + Rb)	Rd <- ES(Mem16 (Ra + Rb))
	LDSH Rd, imm_{16}(Ra)	Rd <- ES(Mem16 (Ra + ES(imm_{16})))
LD	LD Rd, (Ra + Rb)	Rd <- Mem32 (Ra + Rb)
	LD Rd, imm_{16}(Ra)	Rd <- Mem32 (Ra + ES(imm_{16}))

Mnémonique	Syntaxe	Description
STB	STB (Ra + Rb), Rd	$Rd_{0:7}$ -> Mem8 (Ra + Rb)
	STB imm_{16}(Ra), Rd	$Rd_{0:7}$ -> Mem8 (Ra + ES(imm_{16}))
STH	STH (Ra + Rb), Rd	$Rd_{0:15}$ -> Mem16 (Ra + Rb)
	STH imm_{16}(Ra), Rd	$Rd_{0:15}$-> Mem16 (Ra + ES(imm_{16}))
ST	ST (Ra + Rb), Rd	Rd -> Mem32 (Ra + Rb)
	ST imm_{16}(Ra), Rd	Rd -> Mem32 (Ra + ES(imm_{16}))
Bxx(voir tableau suivant)	Bxx imm_{20} ou Bxx etiq	si condxx, PC <- PC + ES(imm_{20}*4)
JMPL	JMPL Rd, Ra, Rb	Rd <- PC ; PC <- Ra + Rb
JMPLI	JMPL Rd, Ra, imm_{16}	Rd <- PC ; PC <- Ra + ES (imm_{16}*4)

Bxx	Description
Ba	Toujours
Bne	$\neq$
Be	$=$
Ble	$\leq$
Bl	$<$
Bleu	$\leq$ non signé
Bcs	CF = 1
Bvs	OF = 1
Bge	$\geq$
Bg	$>$
Bgu	$>$ non signé

10 Jeu d'instructions de MEXP

L'architecture MEXP est une extension de MEXS. Elle comprend un registre CTR, utilisable comme compteur de boucle et un mode d'adressage avec mise à jour. Les instructions supplémentaires sont :

- **SCTR Ra** ou **SCTR imm$_1$6**: CTR <- Ra ou 0x0000‖ imm$_1$6.
- **BCxx imm20 (ou etiq)**: CTR <- CTR - 1 ; branchement si (condxx ET (CTR $\neq$ 0)).
  Mêmes variantes que dans les instructions de branchement.
- **LDUxx et STUxx** : après l'accès mémoire, Ra <- Adresse Effective.
  Mêmes variantes que pour les accès mémoire sans mise à jour.