Université de Paris XI
Licence Informatique / FIIFO3 / IUP3
Année 1999/2000

Architecture des ordinateurs

Travaux dirigés[1]

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

Partie I
Codage de l'information et stockage mémoire

Codage de l'information

1 Codage des données numériques

1.1 Nombres entiers naturels

On considère des mots de 8 bits (1 octet), convertir :

- $1001\ 0110_2$ et $1100\ 0110_2$ en notation décimale.
- 57_{10} et 248_{10} en binaire et en hexadécimal.

A quoi correspond un décalage à gauche (resp. à droite) d'une position sur un nombre binaire?

1.2 Nombres entiers relatifs

On considère une représentation complément à 2 sur un octet.

- Donner en hexadécimal l'opposé de $5A_H$, $E0_H$ et 80_H.
- Faire les additions suivantes sur un octet et noter la retenue :
  $15_H + 48_H$; $F5_H + AF_H$; $15_H + A3_H$; $72_H + F9_H$; $47_H + 3A_H$; $81_H + 95_H$.
- Dans quelles conditions le résultat des additions précédentes est-il correct ?

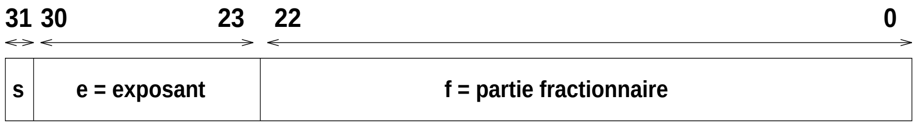
1.3 Nombres réels

Format simple précision IEEE 754

On considère la représentation à virgule flottante IEEE 754 pour les nombres réels. Cette représentation est décrite dans le tableau suivant.

Exposant	Partie fractionnaire	Valeur (E = biais)
$0 < e < e_{max}$	f	$(-1)^s \times 1,f \times 2^{e-E}$ (normalisé)
$e = 0$	$f \neq 0$	$(-1)^s \times 0,f \times 2^{-E}$ (dénormalisé)
$e = 0$	$f = 0$	+/- 0
$e = e_{max}$	$f = 0$	+/- ∞
$e = e_{max}$	$f \neq 0$	NaN

La représentation binaire sur 32 bits IEEE 754 d'un nombre flottant simple précision est présentée dans la figure suivante. Dans cette représentation, 1 bit est dedié au signe, 8 bits à l'exposant et 23 bits à la partie fractionnaire de la mantisse. L'exposant est en excès à 127 (E = 127).



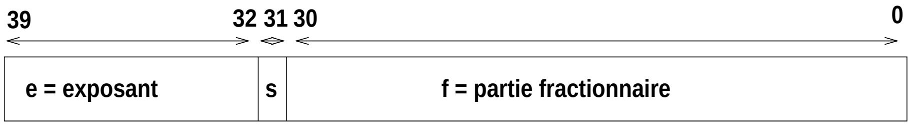
- Quels nombres simple précision correspondent aux mots de 32 bits suivants : 41300000_H, $41E00000_H$, $BF800001_H$ et $00A00000_H$.
- Donner le plus grand positif et son prédécesseur, indiquer leur écart ; le plus petit positif normalisé et dénormalisé et leurs successeurs ; le plus grand et le plus petit négatif.
- Ecrire 1 et 0,25 de façon normalisée. Coder sur 32 bits : 2,250 et 4,5625.

Format simple-précision étendu TMS C40

On considère maintenant la représentation selon le format TMS C40 qui est décrite dans le tableau suivant.

Exposant	Signe	Partie fractionnaire	Valeur
$e > -128$	0	f	$01{,}f \times 2^e$
$e > -128$	1	f	$10{,}f \times 2^e$
$e = -128$	0	0	0

L'exposant est en complément à 2 et la mantisse avec extension est interprétée en complément à 2. Le format binaire sur 40 bits est présenté dans la figure suivante.



- Donner le plus grand positif, le plus petit positif, le plus grand et le plus petit négatif.

1.4 Codage BCD

Les données non numériques sont généralement codées à l'aide de tables. Par exemple, un certain nombre de caractères (ex: lettres de l'alphabet) peuvent être représentés sur 1 octet selon le code ASCII (American Standard Code for Information Interchange). Les nombres décimaux bien que données numériques peuvent être codés de la même facon que les données non numériques.

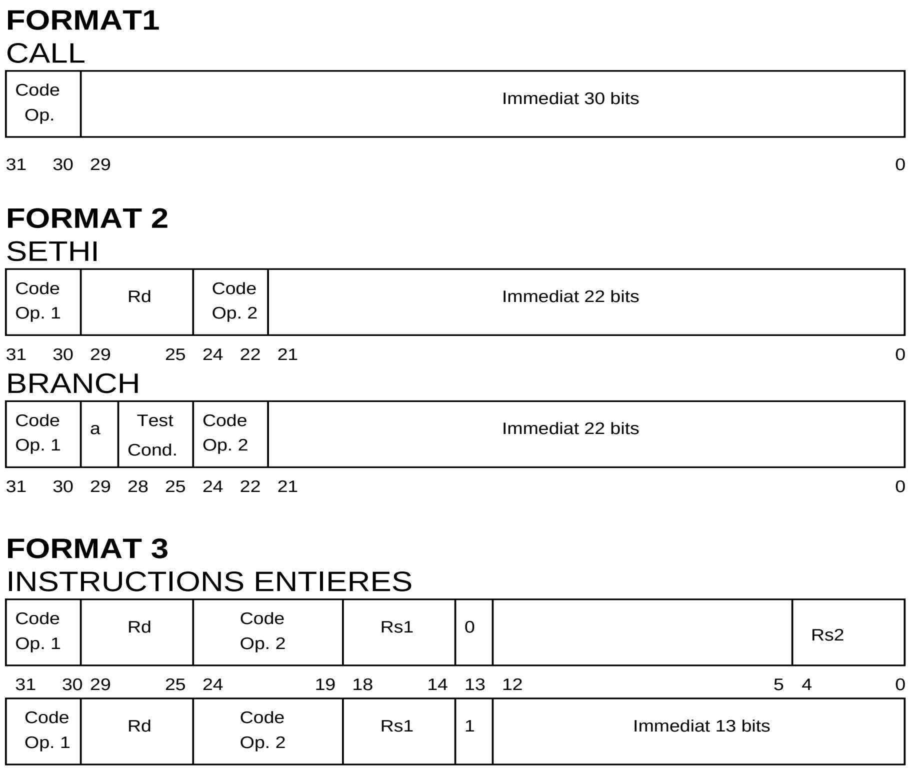
Dans la représentation décimale codée binaire (BCD), chaque chiffre (0 à 9) est représenté individuellement sur 1 octet avec le quartet fort (4 bits de poids fort) égal à 0 et le quartet faible (4 bits de poids faible) codant en binaire la valeur du chiffre. Cette représentation peut être utilisée lorsque tous les chiffres significatifs des nombres doivent être conservés.

- Si A et B sont deux octets qui représentent en BCD respectivement 2 et 5, quel est le résultat de A + B? Même question si A représente 9.

- Montrer que l'addition modulo 10 de deux nombres représentés en BCD par les octets A et B peut se réaliser en ajoutant A et B (addition binaire) et en ajoutant 6 si le résultat est supérieur ou égal à 10_{10}.

2 Codage des instructions

Soit le format des instructions SPARC :



Le processeur considéré a 32 registres généraux. Les codes opération des instructions utilisées dans cet exercice sont présentés dans le tableau suivant.

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

Le format 3 a les syntaxes suivantes:

- OP Rd, Rs1, Rs2 (Rd <- Rs1 OP Rs2),

- OP Rd, Rs1, imm$_{13}$ (Rd <- Rs1 OP ES(imm$_{13}$)).
- OPmem Rd, Rs1, Rs2 (chargement : Rd <- Mem(Rs1 + Rs2), rangement : Rd -> Mem(Rs1 + Rs2)).
- OPmem Rd, Rs1, imm$_{13}$.

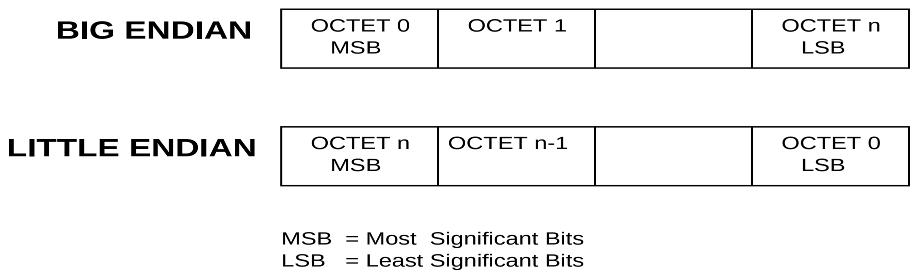
- Quel serait le codage (en hexadécimal) des instructions suivantes:
  - LD R1, R10, 12
  - LD R2, R10, R3
  - ADD R4, R1, R2
  - SUB R20, R20, 1
  - ST R4, R7, 22

Organisation des données en mémoire

3 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.

4 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.

5 Directives assembleur SPARC

On considère le segment de données:

```
.seg "data"
.align 4
a :  .word 2
b :  .byte 8
c : .skip 5
.align 4
d : .word 15, 22,  84
e : .byte 0x48
f :  .byte 2
.align 4
g :  .word 8
h : .word 15, 22,  1084
i : .word 0x48
```

- Décrire le contenu de la mémoire à partir du début du segment de données. L'architecture SPARC utilise le big endian.

6 Transferts Mémoire-Processeur

On considère le programme SPARC:

```
sethi %hi(nomb),%l0
ldub [%l0 + %lo(nomb)], %l1
! load unsigned byte
ldsb [%l0 + %lo(nomb)], %l2
! load signed byte
lduh [%l0 + %lo(nomb)], %l3
```

```
! load unsigned halfword
ldsh [%l0 + %lo(nomb)], %l4
! load signed halfword
ld [%l0 + %lo(nomb)], %l5
! load word
ldd [%l0 + %lo(nomb)], %l6
! load doubleword

.seg "data"
        .align 4
nomb:
.word 0xABCD0000, 0x12345678
```

Les registres sont de 32 bits. Le chargement d'un double mot dans un registre pair (l2n) affecte l2n et l2n+1, avec MSW (*Most Significant Word*) dans l2n.

- Décrire le contenu des registres %l1 à %l7 après l'exécution du programme précédent.

Partie II
Modèles d'exécution et programmation assembleur

Modèles d'exécution

7 Comparaison machines à pile, à accumulateur et chargement-rangement

Considérons les trois machines suivantes:

M0 **machine à pile** (résultat au sommet): dans une machine à pile, une instruction arithmétique (ADD ou SUB) dépile les opérandes, effectue l'opération et empile le résultat.

M1 **machine à accumulateur**:

M2 **machine chargement-rangement:** dans une machine de ce type, les accès mémoire ne sont possibles que par les instructions de chargement (LOAD) et de rangement (STORE).

Les trois machines disposent des instructions suivantes:

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

Les variables A, B, C et D sont initialement en mémoire.

- Ecrire les séquences de codes suivants pour les machines M0, M1, M2.
  - A := B + C;
  - A := B + C;
    B := A + C;
    D := A - B;
- Comparer ces machines en taille de code statique et en nombre d'accès mémoire.

Programmation assembleur

8 Jeu d'instructions

8.1 Valeurs et immédiats

Considérons les formats des instructions SPARC présentés dans la section 2 (codage des instructions).

- Quel est l'intervalle maximal d'un immédiat dans une instruction arithmétique, dans un branchement, dans un appel de procédure ?
- Ecrire la séquence d'instructions correspondant à l'initialisation du registre %l0 à 00000012_H ; à 12345678_H.

8.2 Modes d'adressage

Le processeur considéré dans cet exercice posséde 32 registres généraux de 32 bits (R0 câblé à 0). Les différents modes d'adressage sont décrits dans le tableau suivant sur l'exemple de l'instruction de chargement (LD ; idem ST). Les instructions ADD et SUB ont la syntaxe suivante :

- OP Rd, Rs1, Rs2 (Rd <- Rs1 OP Rs2),

- OPI Rd, Rs1, imm (Rd <- Rs1 OP ES(imm)).

Mode adressage	Mnémonique	Syntaxe	Action
Reg+imm	LD	LD Rd, Ra, imm	Rd <- Mem(Ra+ES(imm))
Reg+Reg	LDX	LDX Rd, Ra, Rb	Rd <- Mem(Ra+Rb)
Reg+ imm avec Mise à jour	LDU	LDU Rd, Ra, imm	Rd <- Mem(Ra+ES(imm)) Ra <- Ra + ES(imm)
Reg+Reg avec Mise à jour	LDUX	LDUX Rd, Ra, Rb	Rd <- Mem(Ra+Rb) Ra <- Ra + Rb

Soient les deux boucles :

```
B1                          B2
Pour i = 0, 999             Pour i = 1, 999
  s = s + X(i) + Y(i)         Y(i) = X(i) + X(i-1)
```

On suppose que les vecteurs X et Y sont implantés à partir des adresses $1000\ 0000_H$ et $2000\ 0000_H$. La variable s est placée à l'adresse 100_H. Les vecteurs X et Y sont des entiers.

- Ecrire le corps de chacune des boucles en assembleur (sans instruction de contrôle), en utilisant successivement les quatre modes d'adressage.

8.3 Format des instructions

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,
- les variantes des instructions LOAD et STORE comme dans la section *Transferts Mémoire-Processeur* i.e. 6 instructions différentes de chargement et 6 de rangement.

  - Proposer un format d'instruction.

9 Conditionnelles et boucles

On utilise dans cette section les deux architectures MEXS (à la SPARC) et MEXP (à la POWER), dont le jeu d'instruction est décrit dans la section *Architectures et jeux d'instructions MEXS et MEXP*.

9.1 Instructions arithmétiques

- Quelle est l'amplitude maximale d'un immédiat dans une instruction arithmétique?
- Comment peut-on réaliser le chargement de $0100\ 0000_H$ dans R1?
- Définir une macro-instruction MOV Ra, imm_{32}.

9.2 Conditionnelles

- Ecrire la séquence d'instructions qui place dans un registre le maximum de deux registres.
- En code EBCDIC, les caractères 0 à 9 sont représentés sur 8 bits par les configurations hexadécimales, $F0_H$ à $F9_H$. Si A et B sont chacun un chiffre EBCDIC, écrire en assembleur la séquence d'instructions qui place dans la variable C la représentation EBCDIC du plus grand des deux. L'adresse de A est $A0000000_H$, celle de B est $B0000000_H$ et celle de C, $C0000000_H$.

9.3 Boucles

Programme 1

Soient les deux boucles:

```
B1                          B2
Pour i = 0, 999             Pour i = 1, 999
  s = s + X(i) + Y(i)       Y(i-1) = X(i) + X(i-1)
```

On suppose que les vecteurs X et Y sont des entiers (32 bits), implantés respectivement à partir des adresses 10000000_H et 20000000_H. La variable s est placée à l'adresse 100_H.

- Ecrire le code des deux boucles (reprendre la section 10 et rajouter la partie contrôle, gestion de la boucle).

Programme 2

- Ecrire le programme qui recherche la première occurence du caractère 'A' dans un tableau de 10 caractères.

Programme 3

- Ecrire un programme additionnant deux entiers relatifs représentés sur N octets. N est une constante, mutiple de 4. 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.

10 Architectures et jeux d'instructions MEXS et MEXP

Les deux architectures ont :

- un chemin de données 32 bits,
- 32 registres 32 bits avec R0 câblé à 0,
- un registre code condition RCC avec les drapeaux usuels,
- une mémoire organisée par octets, avec 4Goctets adressables en big endian.

10.1 Jeu d'instructions de MEXS

Mnémonique	Syntaxe	Description
ADD	ADD Rd, Ra, Rb	Rd <-Ra + Rb
ADDcc	ADDcc Rd, Ra, Rb	Rd <-Ra + Rb RCC affecté
ADDX	ADDX Rd, Ra, Rb	Rd <-Ra + Rb + CF
ADDXcc	ADDXcc Rd, Ra, Rb	Rd <-Ra + Rb + CF RCC affecté
SUB	SUB Rd, Ra, Rb	Rd <-Ra - Rb
SUBcc, SUBX, SUBXcc	-	-
AND	AND Rd, Ra, Rb	Rd <-Ra AND Rb
ANCcc	ANDcc DD Rd, Ra, Rb	Rd <-Ra AND Rb RCC affecté
OR	OR Rd, Ra, Rb	Rd <-Ra OR Rb
ORcc	ORcc Rd, Ra, Rb	Rd <-Ra OR Rb RCC affecté
XOR	XOR Rd, Ra, Rb	Rd <-Ra XOR Rb
XORcc	XORcc Rd, Ra, Rb	Rd <-Ra XOR Rb RCC affecté
LDB	LDB Rd, (Ra + Rb)	$Rd_{0:7}$ <- Mem8 (Ra + Rb) $Rd_{8:31}$ <- 0
LDSB	LDSB Rd, (Ra + Rb)	$Rd_{0:7}$ <- Mem8 (Ra + Rb) $Rd_{8:31}$ <- ES($Rd_{0:7}$)
LDH	LDH Rd, (Ra + Rb)	$Rd_{0:15}$ <- Mem16 (Ra + Rb) $Rd_{16:31}$ <- 0
LDSH	LDSH Rd, (Ra + Rb)	$Rd_{0:15}$ <- Mem16 (Ra + Rb) $Rd_{16:31}$ <- ES($Rd_{0:15}$)
LD	LD Rd, (Ra + Rb)	Rd <- Mem32 (Ra + Rb)
STB	STB (Ra + Rb), Rd	$Rd_{0:7}$ -> Mem8 (Ra + Rb)
STH	STH (Ra + Rb), Rd	$Rd_{0:15}$ -> Mem16 (Ra + Rb)
ST	ST (Ra + Rb), Rd	Rd -> Mem32 (Ra + Rb)
ADDI (1)	ADDI Rd, Ra, imm_{16}	Rd <- Ra + ES (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 de imm_5 avec remplissage à 0
SRA	SRA Rd, imm_5, Ra	Rd <- Ra décalé à droite de imm_5 avec extension de signe
Bxx(voir tableau suivant)	Bxx imm_{20} ou Bxx etiq	si condxx, PC <- PC + ES(imm_{20}) $\times$ 4
JMPL	JMPL Rd, Ra, Rb	Rd <- PC PC <- Ra + Rb
JMPLI	JMPLI Rd, Ra, imm_{16}	Rd <- PC PC <- Ra + ES (imm_{16}) $\times$ 4
SETHI	SETHI Rd, imm_{22}	$Rd_{10:31}$ <- imm_{22} $Rd_{0:9}$ <- 0

(1) Variante immédiate de toutes les instructions Arithmétiques et Logiques et des Chargements et Rangements.

Bxx	Description
Ba	Toujours
Bne	$\neq 0$
Be	$= 0$
Ble	≤ 0
Bl	< 0
Bleu	≤ 0 non signé
Bcs	CF $= 1$, < 0 non signé
Bvs	OF $= 1$
Bcc	CF $= 0$, ≥ 0 non signé
Bvc	OF $= 0$
Bge	≥ 0
Bg	> 0
Bgu	≥ 0 non signé

10.2 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 imm**: CTR <- Ra ou imm.
- **BCxx imm20 (ou etiq)**: CTR <- CTR - 1 ; branchement si (condxx ET (CTR $\neq$ 0)).
  Mêmes variantes que pour MEXS.
- **LDUxx et STUxx** : après l'accès mémoire, Ra <- Adresse Effective.
  Mêmes variantes que pour MEXS.

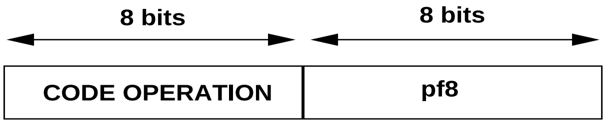
Partie III

Micro-architecture, pipeline et cache

Micro-Architecture

11 Architecture 1 bus

Le chemin de données du processeur DOLD a une largeur de 16 bits, et une mémoire de 64 Kmots de 16 bits. Toutes les instructions sont sur 16 bits et le code opération sur 8 bits (CODOP).



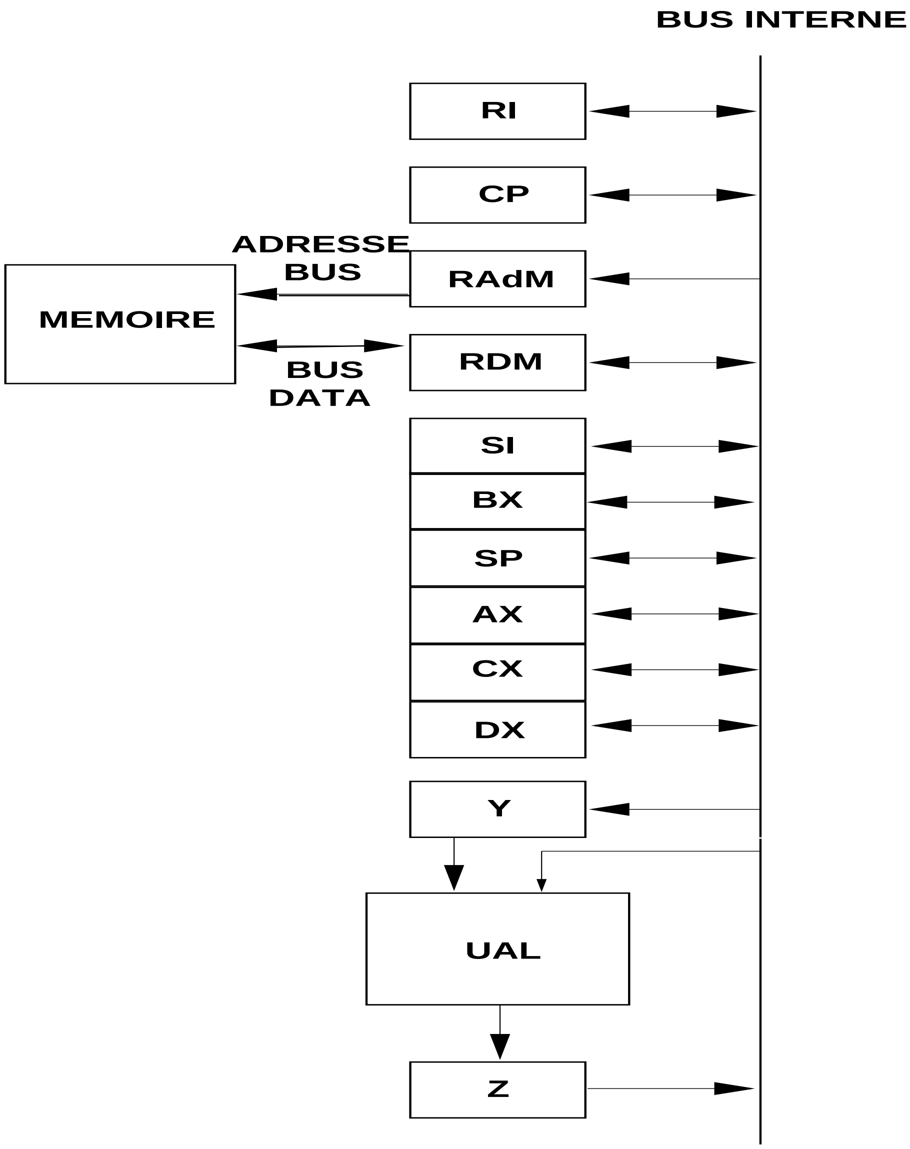
Les registres sont:

- RI: registre instruction,
- CP: compteur de programme,
- RAdM: registre adresse mémoire,
- RDM: registre donnée mémoire,
- SI: registre d'index,
- SP: pointeur de pile,
- AX: accumulateur, BX: base, CX: compteur et DX: supplémentaire,
- X, Y: registres intermédiaires.

Les modes d'adressage sont:

Mode adressage	Opérande	pf8
Registre	valeur dans un registre spécifié dans CODOP	inutilisé
Immédiat	valeur dans l'instruction	opérande
Direct page 0	adresse dans l'instruction	adresse
Registre indirect	adresse dans un registre spécifié dans CODOP	inutilisé
Indexé	Adresse = SI + déplacement	déplacement
Basé	Adresse = BX + déplacement	déplacement
Relatif	Adresse = CP + déplacement	déplacement

L'architecture interne est représentée dans la figure suivante.

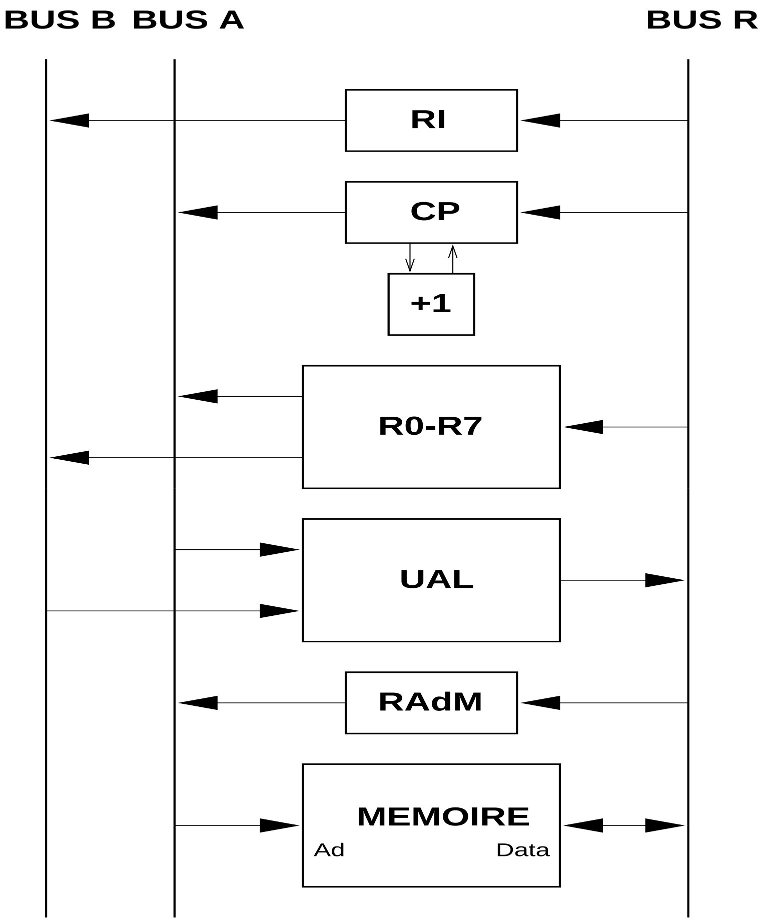


Les échanges avec la mémoire (lecture ou écriture) se font en un cycle avec le contenu actuel du registre d'adresse (RAdM). L'adresse mémoire doit être placée dans le registre d'adresse dans un cycle précédent. Ces transferts peuvent avoir lieu concuremment avec un transfert sur le bus interne.

- Décrire les transferts de données élémentaires qui interviennent pendant:
  - la lecture de l'instruction,
  - le rangement de l'opérande (contenu de registre ou de case mémoire) dans le registre Y suivant les différents types d'adressage,
  - l'exécution des instructions ADD, SUB, XOR (AX),
  - l'exécution des instructions JMP, PUSH BX, POP CX.

12 Architecture 3 bus

Soit le processeur RISC 16 bits DER/94, dont le chemin de données est représenté dans la figure suivante.



Tous les registres sont sur 16 bits:

- RI: registre instruction,
- CP: compteur de programme,
- R0-R7: 8 registres généraux (R0 câblé à 0),
- RAdM: registre adresse mémoire.

La mémoire est commune pour les instructions et les données. Le format d'instruction est fixe, selon la figure suivante.

F1: reg-reg

00xx	a	d	b	xxx
4	3	3	3	3

F2: reg-imm

01xx	a	d	immediat
4	3	3	6

F3: branchement

1xxx	a	immediat
4	3	9

Les instructions sont représentées dans le tableau suivant.

Mnémonique	Code opération	Action
ADD	0000	Rd <- Ra + Rb
SUB	0001	Rd <- Ra - Rb
ADI	0100	Rd <- Ra + ES(imm_6)
SUBI	0101	Rd <- Ra - ES(imm_6)
LD	0010	Rd <- Mem (Ra + Rb)
LDI	0110	Rd <- Mem (Ra + ES(imm_6))
ST	0011	Rd -> Mem (Ra + Rb)
STI	0111	Rd -> Mem (Ra + ES(imm_6))
Bxxx	1xxx	CP <- CP + ES(imm_9) si condx vraie

On considère que l'exécution d'une instruction se déroule en deux phases: lecture de l'instruction, avec incrémentation de CP, puis exécution effective de l'instruction.

- Préciser les transferts de données élémentaires associés à chacune de ces phases, pour chaque mode d'adressage.
- On veut définir la partie contrôle du processeur. Définir son organisation de principe.

Pipeline

13 Aléas des pipelines

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.

Cette architecture a deux implantations, 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 le tableau suivant.

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.
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.
M2	Pour les types C et R, fin accès mémoire.
RR	Rangement du résultat arithmétique (A) ou écriture (C) dans le banc de registre.

13.1 Mémoires instructions-données séparées

On suppose que dans les deux pipelines, les caches d'instructions et de données sont séparées (1 port d'accès pour chacun).

- 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 suivant leur type (sauf B). On ne considérera pas les problèmes de propagation de cette suspension aux instructions suivantes.
- Même question en supposant que tous les mécanismes d'anticipation possibles ont été implémentés.
- Quelle est la taille du délai de branchement pour P1 et pour P2?
- Quelles sont les ressources matérielles nécessaires aux deux pipelines ?

13.2 Mémoire instructions-données unifiée

On considère un cache commun instructions et données (1 seul port d'accès).

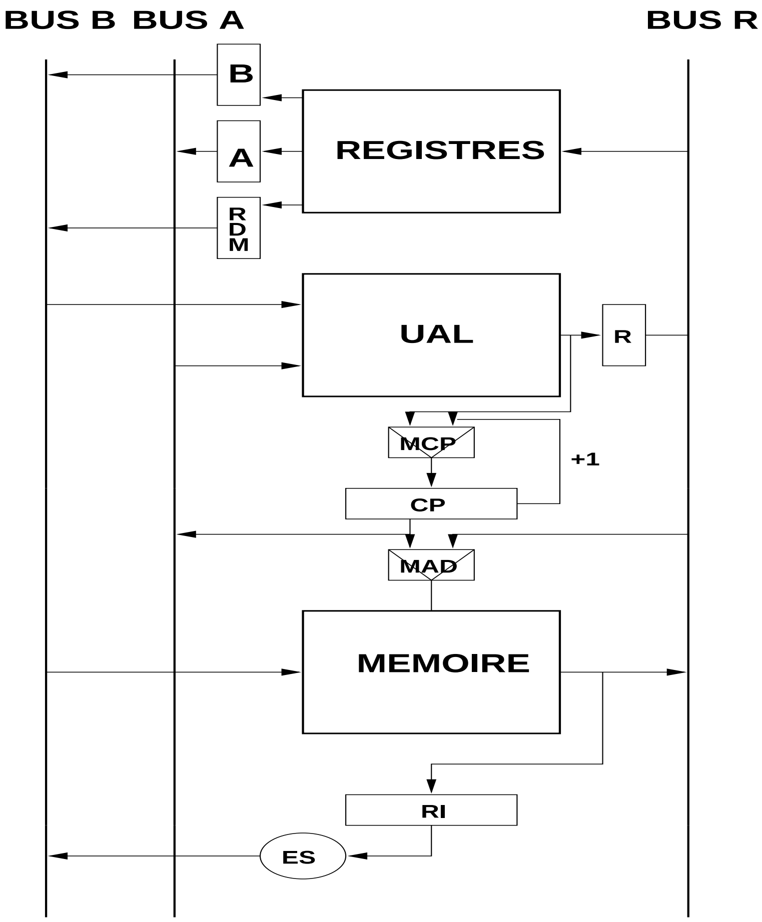
- En supposant que les mécanismes d'anticipation sont mis en oeuvre, indiquer le nombre de suspensions nécessaires entre deux instructions succéssives.

14 Micro-architecture pipelinée

On considère une architecture à cache commun instructions et données, dont le jeu d'instructions est le suivant.

Mnémonique	Code opération	Action
ADD	0000	Rd <- Ra + Rb
SUB	0001	Rd <- Ra - Rb
ADI	0100	Rd <- Ra + ES(imm)
SUBI	0101	Rd <- Ra - ES(imm)
LD	0010	Rd <- Mem (Ra + Rb)
LDI	0110	Rd <- Mem (Ra + ES(imm))
ST	0011	Rd -> Mem (Ra + Rb)
STI	0111	Rd -> Mem (Ra + ES(imm))
BA	1000	CP <- CP + ES(imm)

et un pipeline à cinq étages LI, DI, EX, MEM et RR. On veut implémenter ce jeu d'instructions et ce pipeline sur la micro-architecture suivante.



- Définir l'ensemble des états possibles du pipeline. Montrer qu'il ne peut il y avoir simultanément une phase MEM et une phase RR et modifier le chemin de données pour qu'elles puissent être réalisées simultanément.
- Décrire les transferts élémentaires durant chaque phase. Compléter les tableaux suivants, i.e. les connexions bus, les opérations UAL activées, et les commandes d'écriture de registres, multiplexeurs et mémoire.

Etage	Instruction	Bus A	Bus B	ES	Bus R	Op. UAL
LI	toutes					
EX	ADD					
-	SUB					
-	LD					
-	ADDI					
-	SUBI					
-	LDI					
-	ST					
-	STI					
-	BA					
MEM	LD					
-	LDI					
-	ST					
-	STI					
RR	aritm.					

Etage	Instruction	Reg. écrit	MCP	MAD	Mémoire	Registres
LI	toutes					
EX	ADD					
-	SUB					
-	LD					
-	ADDI					
-	SUBI					
-	LDI					
-	ST					
-	STI					
-	BA					
MEM	LD					
-	LDI					
-	ST					
-	STI					
RR	aritm.					

- Définir l'automate de séquencement, à partir des états possibles du pipeline.

Caches

On considère une architecture possédant un cache de données de 8Koctets organisé en blocs de 32 octets. Cet exercice sera traité avec deux types de cache différents: un cache à correspondance directe et un cache associatif par ensemble de 2 blocs avec stratégie de remplacement LRU (*Last Recently Used*).

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

X	Y	Z	X1	Y1	X2	Y2
1 0000_H	1 4000_H	1 8000_H	1 $C000_H$	2 0000_H	2 4000_H	2 8000_H

1. Quels sont les éléments des tableaux X et Y qui peuvent occuper le mot 0 du bloc 0, le mot 3, bloc 2 du cache (cas d'un cache à correspondance directe et d'un cache associatif par ensemble de 2 blocs) ?
2. Combien de défauts 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
```

Partie IV
Programmation et optimisation

15 Dépliement de boucles

On considère le programme FORTRAN suivant:

```
real X(0: N), Y(0:N)

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

On considère l'architecture MEXP augmentée de:

- 32 registres flottants 32 bits F0 à F31,
- les instructions de chargement et rangement flottantes, ainsi qu'addition et multiplication flottantes en simple précision.

LFS	LFS Fd Ra Rb	Fd <- Mem (Ra + Rb)
STFS	STFS Ra Rb Fd	Fd ->Mem (Ra + Rb)
reg-imm (LFSX)	reg-reg indexé (LFSU)	reg-imm indexé (LFSUX)
FADDS	FADDS Fd Fa Fb	Fd <- Fa + Fb
FMULS	FMULS Fd Fa Fb	Fd <- Fa * Fb

Les latences sont:

- chargement flottant: 3 cycles,
- addition flottante: 4 cycles,
- multiplication flottante: 5 cycles,
- autres instructions: 1 cycle.

Il n'y a pas de délai de branchement. On traitera les 2 cas: unités flottantes non pipelinées et pipelinées.

- Ecrire un code optimisé sans dérouler la boucle. Quelle est sa performance en opérations flottantes par cycle? En MFLOPS si on suppose un temps de cycle de 50ns? On ne tiendra pas compte des initialisations.
- Dérouler 3 fois la boucle (soit quatre corps de boucle). Mêmes questions.