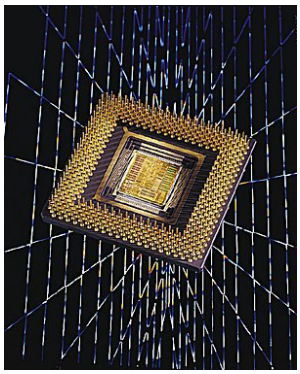
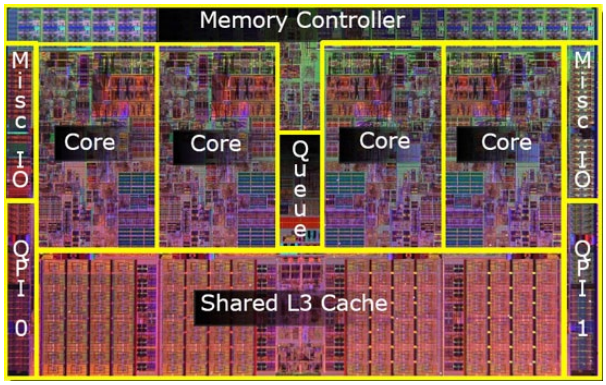
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE
DEPARTEMENT DE GENIE ELECTRIQUE ET INFORMATIQUE

4ème année AE-SE

Année 2023 - 2022

Conception d'un microprocesseur de type RISC avec pipe-line





Eric Alata

Daniela Dragomirescu

Conception d'un microprocesseur de type RISC avec cinq niveaux de pipeline

1. Objectif

Le schéma complet d'un système informatique est présenté ci-dessous.

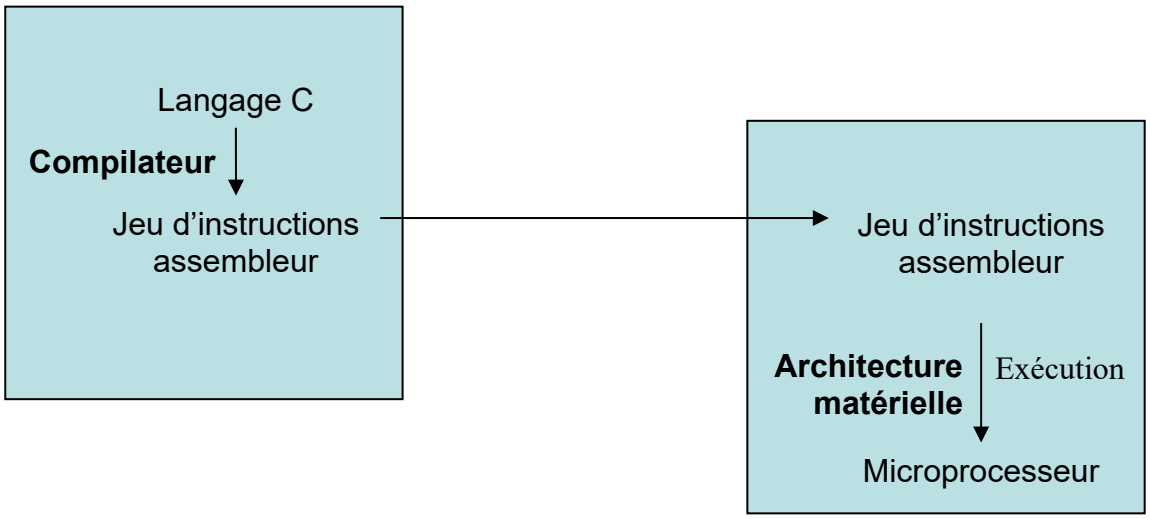


Figure 1. Schéma d'un système informatique : logiciel au matériel

L'objectif de ce Bureau d'étude est de concevoir un microprocesseur de type RISC correspondant à un jeu d'instructions assembleur donné. Ce jeu d'instructions sera orienté registre.

Note : Le cours et les Travaux Pratiques de Architecture Matérielle de 2ème et 3ème année sont un **pré-requis obligatoire** pour ces Travaux Pratiques.

1. Première partie - vous allez concevoir et implémenter en VHDL un microprocesseur avec **pipe-line** correspondant aux i**nstructions assembleur** : **addition, soustraction, multiplication, copie et affectation (sans les instructions de saut et de comparaison)**. Vous allez ensuite synthétiser et optimiser ce microprocesseur du point de vue de la fréquence de fonctionnement et vous allez ensuite l'implémenter sur le FPGA Xilinx. Cette partie est obligatoire.
2. Deuxième partie - vous allez rajouter à votre microprocesseur les unités architecturales nécessaires à l'implémentation des instructions des saut et des comparaison (par exemple : l'unité de branchement). Vous allez ensuite synthétiser et optimiser ce microprocesseur du point de vue de la fréquence de fonctionnement et vous allez ensuite l'implémenter sur le FPGA Xilinx. Cette deuxième partie est optionnelle, en fonction de votre avancement.

Afin de vous guider dans la conception de ce microprocesseur, nous allons vous présenter le jeu d'instructions orientées registre que vous allez utiliser. Dans la troisième section, nous abordons l'architecture du microprocesseur que nous vous proposons d'implémenter.

2. Langage assembleur

Le jeu d'instructions orientées registre que vous allez utiliser est présenté dans le tableau 1. *Ri*, *Rj* et *Rk* représentent les numéros de registres sollicités par l'instruction. *@i* et *@j* représentent des adresses mémoire. Le format d'instruction est de taille fixe. La valeur _ est utilisée comme bourrage (**padding**), dans les instructions nécessitant moins de 3 opérandes. Autrement dit, toutes les instructions occupent 4 octets, comme illustré dans le tableau 2. Les jeux d'instructions utilisés par les microprocesseurs commerciaux ne possèdent pas tous cette caractéristique. Nous vous proposons de concevoir un jeu d'instructions orientées registre avec un format de taille fixe, afin de faciliter votre conception.

Opération	Code	Format d'instruction				Description
		OP	A	B	C	
Addition	0x01	ADD	Ri	Rj	Rk	[Ri] ← [Rj] + [Rk]
Multiplication	0x02	MUL	Ri	Rj	Rk	[Ri] ← [Rj] * [Rk]
Soustraction	0x03	SOU	Ri	Rj	Rk	[Ri] ← [Rj] - [Rk]
Copie	0x05	COP	Ri	Rj	_	[Ri] ← [Rj]
Affectation	0x06	AFC	Ri	j	_	[Ri] ← j
Chargement	0x07	LOAD	Ri	@j	_	[Ri] ← [@j]
Sauvegarde	0x08	STORE	@i	Rj	_	[@i] ← [Rj]

Tableau 1. Jeu d'instructions

Instruction :	ADD	R1	R9	R4
Hexadécimal :	0x01	0x01	0x09	0x04
Binaire :	00000001	00000001	00001001	00000100

Tableau 2. Exemple d'instruction

L'instruction de chargement permet de copier dans le registre *Ri* le contenu de la mémoire de l'adresse *@j*. L'instruction de sauvegarde permet de copier dans la mémoire à l'adresse *@i* le contenu du registre *Rj*. Ce sont les seules instructions permettant un échange avec la mémoire. Un tel processeur RISC est qualifié de processeur RISC **Load/Store**.

3. Architecture du microprocesseur RISC

Nous vous proposons d'implémenter ici une architecture de type **RISC** avec un **pipe-line à 5 étages** (figure 2). Cette architecture doit fonctionner sur **8 bits**.

Notre microprocesseur va avoir :

- Une unité arithmétique et logique, *section 3.1* ;
- Un banc de registres à double port de lecture, *section 3.2* ;
- Une mémoire d'instructions, *section 3.3* ;
- Une mémoire des données, *section 3.3* ;
- Un chemin des données, *section 3.4*;
- Une unité de contrôle, *section 3.5* ;
- Une unité de détection des aléas, *section 3.6* ;
- Architecture pipe-line sur 5 étages.

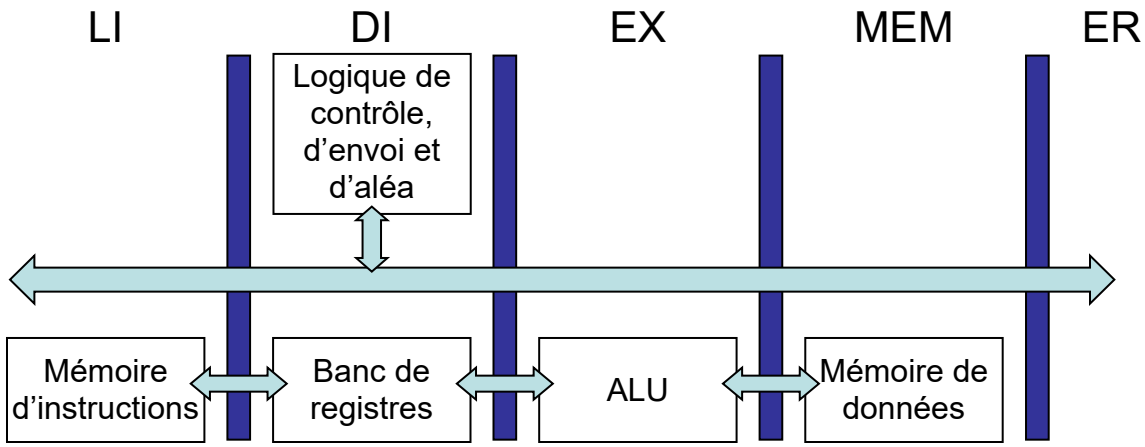


Figure 2. Architecture du microprocesseur RISC avec 5 niveaux de pipe-line

3.1 L'unité arithmétique et logique

On se propose de réaliser une UAL ayant l'interface présentée sur la figure 3. Le signal *Ctrl_Alu* informera sur l'opération à réaliser : Addition, Soustraction, Multiplication ou opérations logiques (AND, OR, XOR, NOT). Les drapeaux (**flags**) : *N*, *O* et *C* représentent, respectivement, une valeur négative en sortie ($S < 0$), un débordement de la multiplication

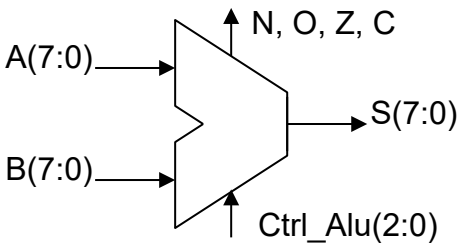


Figure 3. UAL

(**overflow** : taille de *A* MUL *B* > 8 bits) et la retenue (**carry**) de l'addition. Les opérations arithmétiques **peuvent** être réalisées à partir des fonctions correspondantes ('+', '-', '*'), **fournies dans la librairie de Xilinx.**

Le FPGA utilisé contient de multiplieurs câblés donc vous pouvez réaliser la fonction multiplication en utilisant simplement en VHDL le signe * .

3.2 Banc de registres double port de lecture

On se propose de réaliser un banc de 16 registres de 8 bits avec un double accès en lecture et un accès en écriture. Le schéma de ce registre est présenté à la figure 4. Le signal reset *RST* est actif à 0 : le contenu du banc de registres est alors initialisé à *0x00*. *@A* et *@B* permettent de lire deux registres simultanément. Les valeurs correspondantes sont propagées vers les sorties *QA* et *QB*. L'écriture de données dans un registre se fait par le biais des entrées *@W*, *W* et *DATA*. *W* spécifie si une écriture doit être réalisée. Cette entrée est active à 1, pour une écriture. Lorsque l'écriture est activée, les données présentent sur l'entrée *DATA* sont copiées dans le registre d'adresse *@W*. On considère que le reset et l'écriture se feront synchrone avec l'horloge.

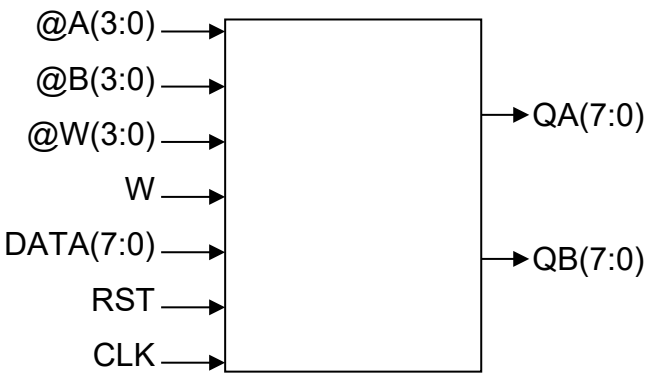


Figure 4. Banc de registres

Bypass D Q

Il arrive qu'en cours d'exécution, le processeur fasse simultanément une requête de lecture et d'écriture sur le même registre. Ceci constitue un aléa de données. Cet aléas peut être traité par l'unité d'envoie. Toutefois, afin de simplifier la conception de l'unité d'envoi, on se propose d'implémenter cette fonctionnalité directement dans le banc de registres :

Si écriture et lecture sur le même registre alors la sortie *Q*X ← *DATA*.

3.3 Banc de mémoire

Notre architecture contient deux mémoires : une mémoire pour les données et une mémoire pour les instructions. Leur structure est présentée dans la figure 5.

La mémoire des données doit permettre un accès en lecture ou en écriture. L'adresse de la zone mémoire est fournie par l'entrée @. Pour réaliser une lecture, *RW* doit être positionné à 1 et pour réaliser une écriture, *RW* doit être positionné à 0. Dans le cas d'une écriture, le contenu de l'entrée *IN* est copié dans la mémoire à l'adresse @. Le reset, *RST*, permettra d'initialiser le contenu de la mémoire à *0x00*. La lecture, l'écriture et le reset se feront synchrones avec l'horloge *CLK*.

Pour simplifier la conception, nous vous proposons une structure de la mémoire des instructions plus simple. Nous supposerons que le programme à exécuter par le microprocesseur est déjà stocké dans cette mémoire. De plus, à l'exécution, nous empêchons toute modification du contenu de cette mémoire. Elle s'apparente à une **ROM**. Elle est alors dépourvue des entrées *RST*, *IN* et *RW*. Comme pour la mémoire des données, la lecture se fera synchrone avec l'horloge *CLK*.

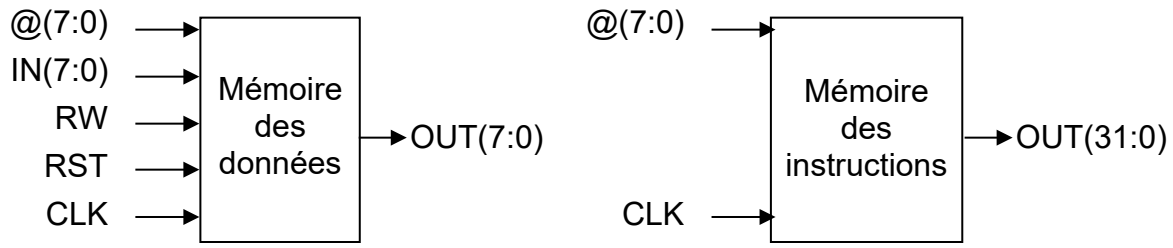


Figure 5. Mémoire des données et mémoire des instructions

3.4 Chemin des données

Nous allons à présent commencer l'intégration des différents éléments précédemment développés afin de concevoir une première version du processeur. Nous allons créer un chemin des données (présenté dans la figure 2). Afin de vous guider dans la création de ce chemin des données, nous vous conseillons de procéder par étapes. Autrement dit, nous vous conseillons de commencer par créer un chemin de données prenant en compte uniquement une des instructions. Vous pourrez alors tester votre implémentation afin de corriger les erreurs éventuelles. Ensuite, vous pourrez intégrer une nouvelle instruction. Vous bouclerez ainsi tant qu'il reste des instructions à implémenter. La suite de cette section vous présente un exemple de démarche.

1 - Nous commençons par implémenter l'instruction *AFC*. Cette instruction permet de copier une constante dans un registre destination. Sachant que la modification d'un registre ne doit être effectuée qu'au niveau de l'étage 5, l'identifiant du registre destination et la valeur à lui affecter doivent être propagées jusqu'à cet étage. Le chemin suivi par cette instruction est donc le suivant :

1.1. Au niveau du premier étage, les différents champs constituant l'instruction sont placés dans le premier pipe-line (*LI/DI*) ;
1.2. Au niveau du second étage, aucune sélection de valeur dans le banc de registres n'est nécessaire : le code de l'opération en cours, la valeur à copier et le registre destination sont simplement propagés dans le pipe-line (*DI/EX*) ;
1.3. Au niveau du troisième étage, aucune exécution arithmétique n'est nécessaire pour la copie : le code de l'opération en cours, la valeur à copier et le registre destination sont simplement propagés dans le pipe-line (*EX/Mem*) ;
1.4. Au niveau du quatrième étage, aucune modification de la mémoire n'est nécessaire pour la copie : le code de l'opération en cours, la valeur à copier et le registre destination sont simplement propagés dans le pipe-line (*Mem/RE*) ;
1.5. Au niveau du cinquième étage, nous disposons de toutes les informations nécessaires pour mener à bien la copie : valeur à copier et registre destination. Cette modification se fait en utilisant l'entrée écriture du banc de registres.

Le schéma correspondant est donné en illustration 1. Dans cette illustration, *OP* représente le code d'opération et *A*, *B* et *C* les opérandes. Notons, au passage, que le chemin emprunté par cette opération nécessite, au niveau de l'étage 5, d'utiliser un composant situé schématiquement au niveau de l'étage 2. Mais, en aucune façon, le chemin en question ne passe deux fois par un même pipe-line.

2 - Sur la base de ce schéma, l'instruction *COP* peut être ajoutée. La différence par rapport au schéma précédent est l'obligation, lors du passage à l'étage 2, de propager la valeur associée au registre source et non l'identifiant de ce registre. Pour effectuer ces modifications sans pour autant altérer le bon fonctionnement de l'instruction *AFC*, nous utilisons un multiplexeur. En fonction du code de l'opération, il propage le paramètre *B* ou la valeur contenue dans le registre pointé par le paramètre *B*. Le schéma obtenu est donné en illustration 2.

3 - Remarquons que les illustrations précédentes incluent une unité arithmétique et logique sans les exploiter. Nous allons donc inclure les instructions arithmétiques pour rattacher cette UAL au chemin de données. En utilisant à nouveau des multiplexeurs pour déterminer les éléments à propager, nous obtenons l'illustration 3.

4 - En procédant de la même manière pour l'implémentation des instructions *LOAD* et *STORE*, nous obtenons les illustrations 4 et 5.

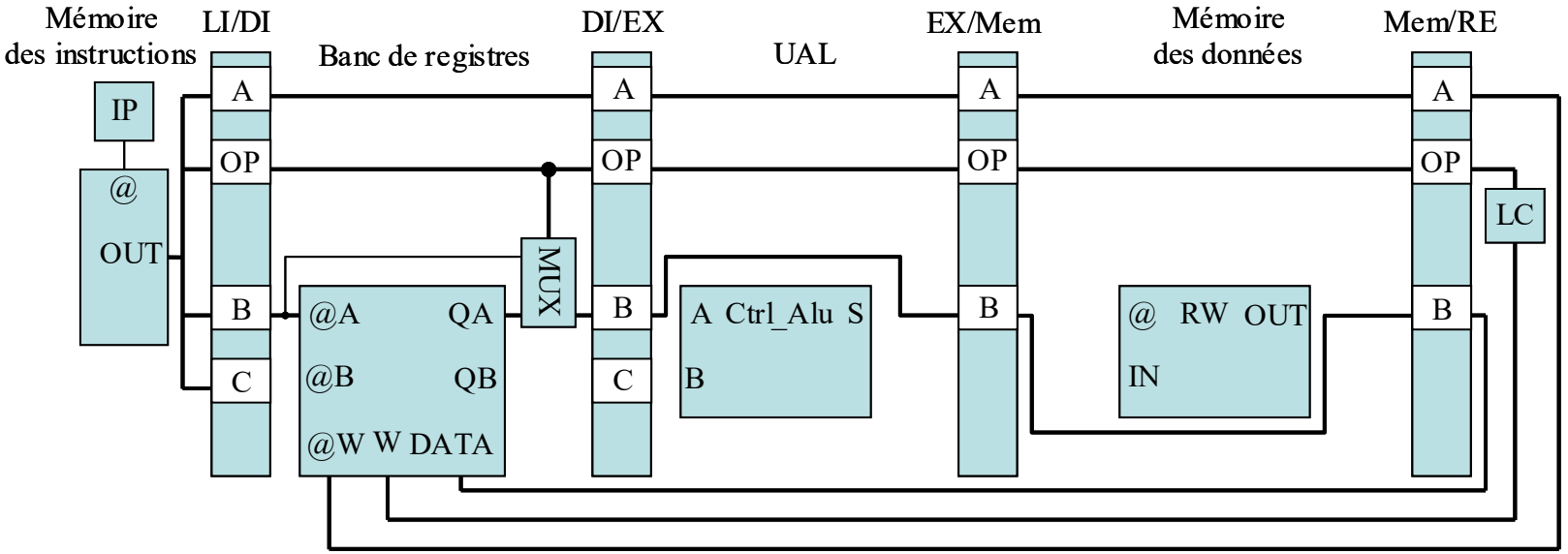


Illustration 2 : Instructions AFC, COP

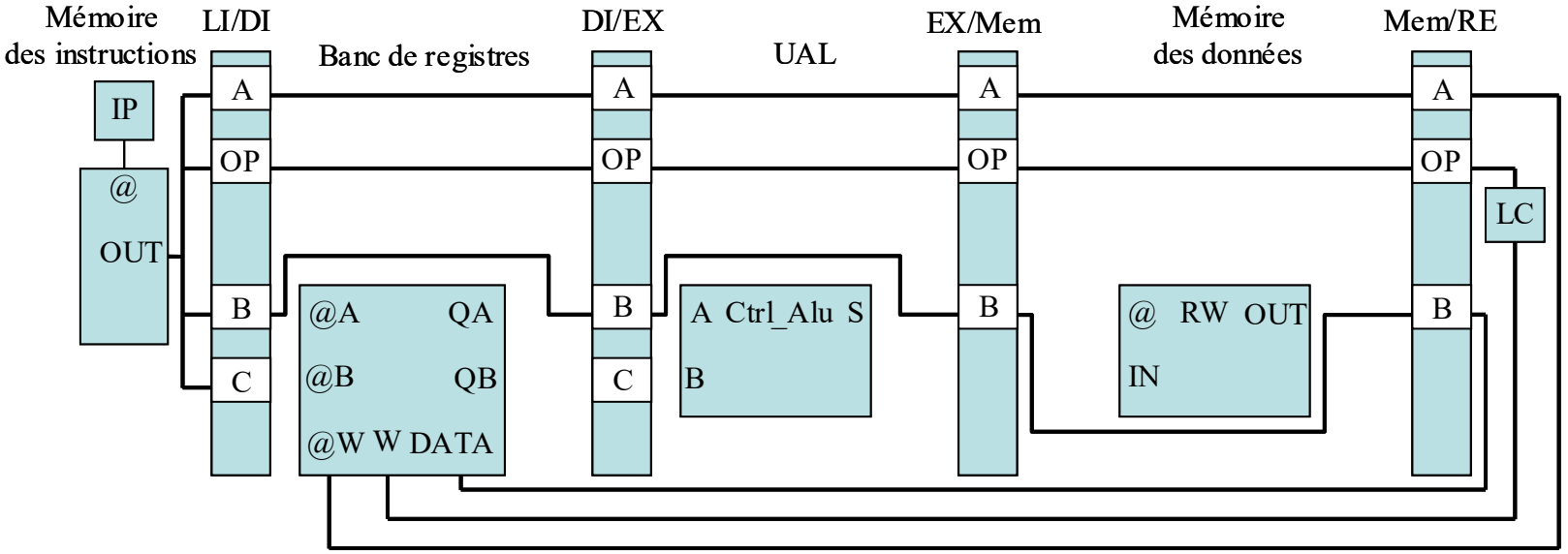


Illustration 1 : Instruction AFC

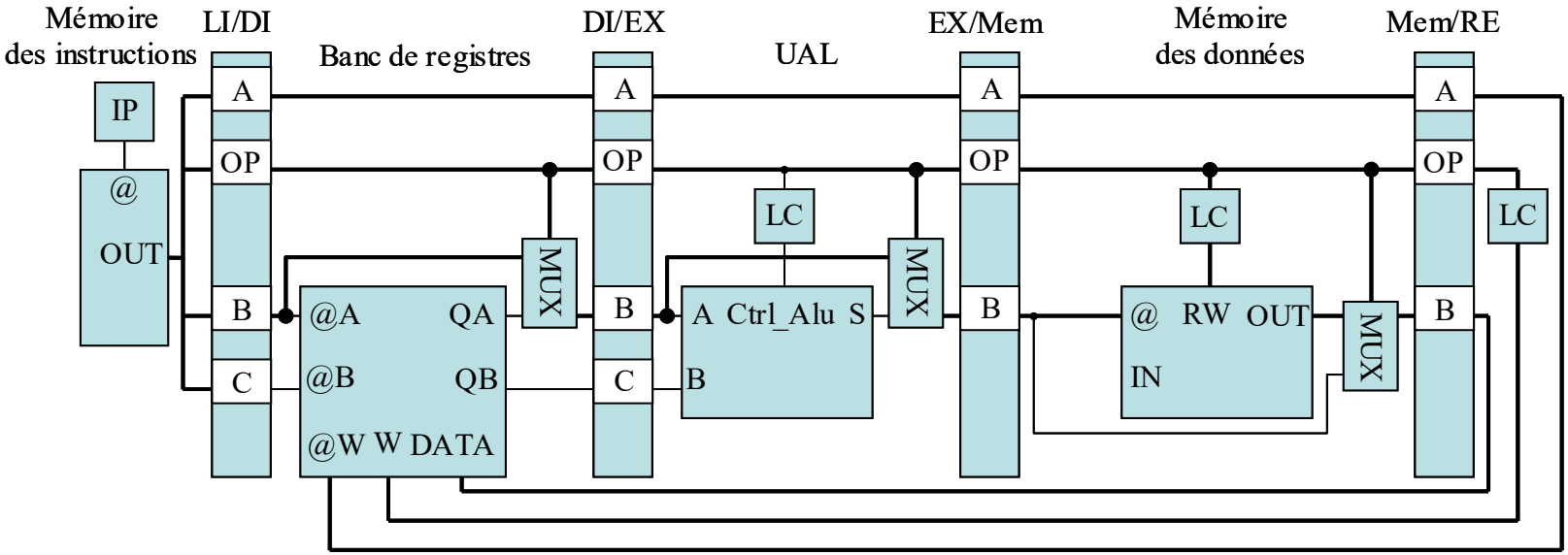


Illustration 4 : Instructions AFC, COP, ADD, MUL, DIV, SOU, LOAD

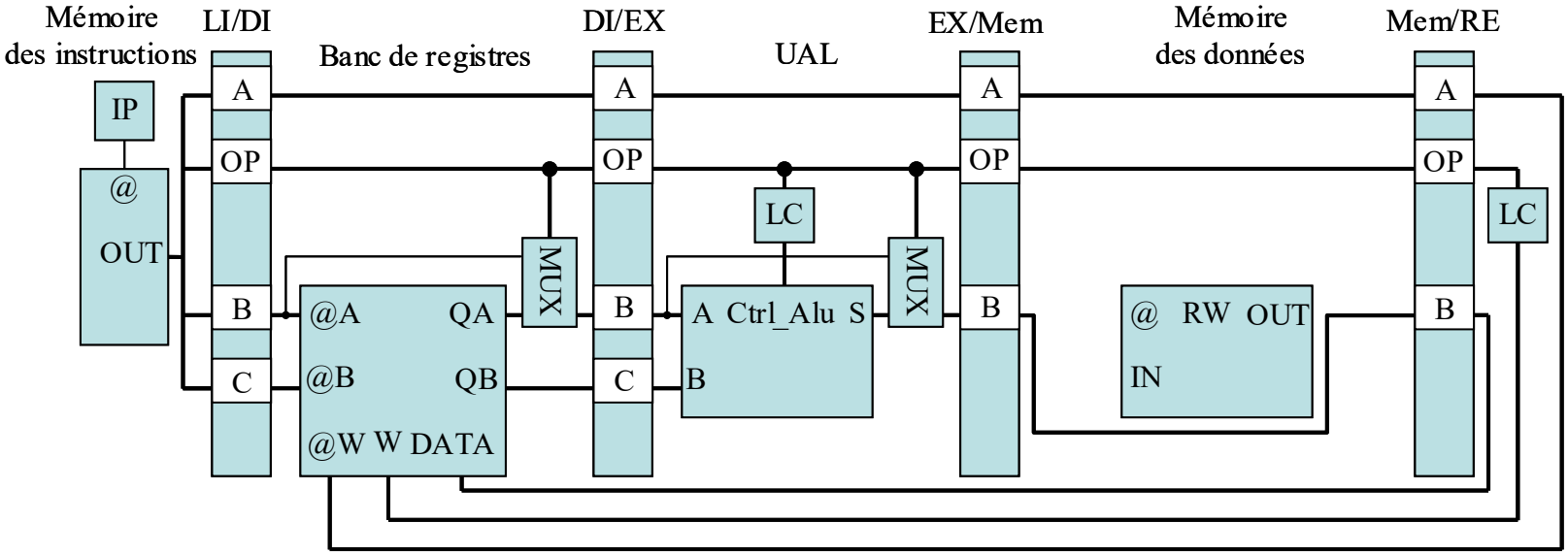


Illustration 3 : Instructions AFC, COP, ADD, MUL, DIV, SOU

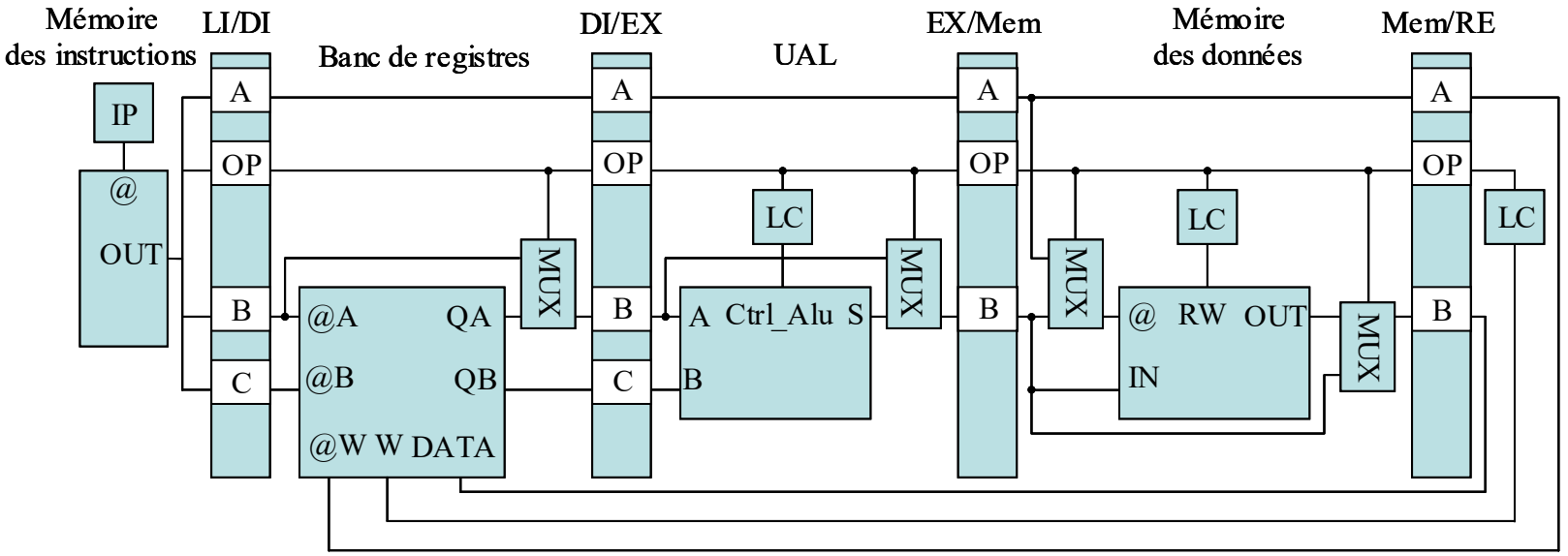


Illustration 6 : Chemin des donnée

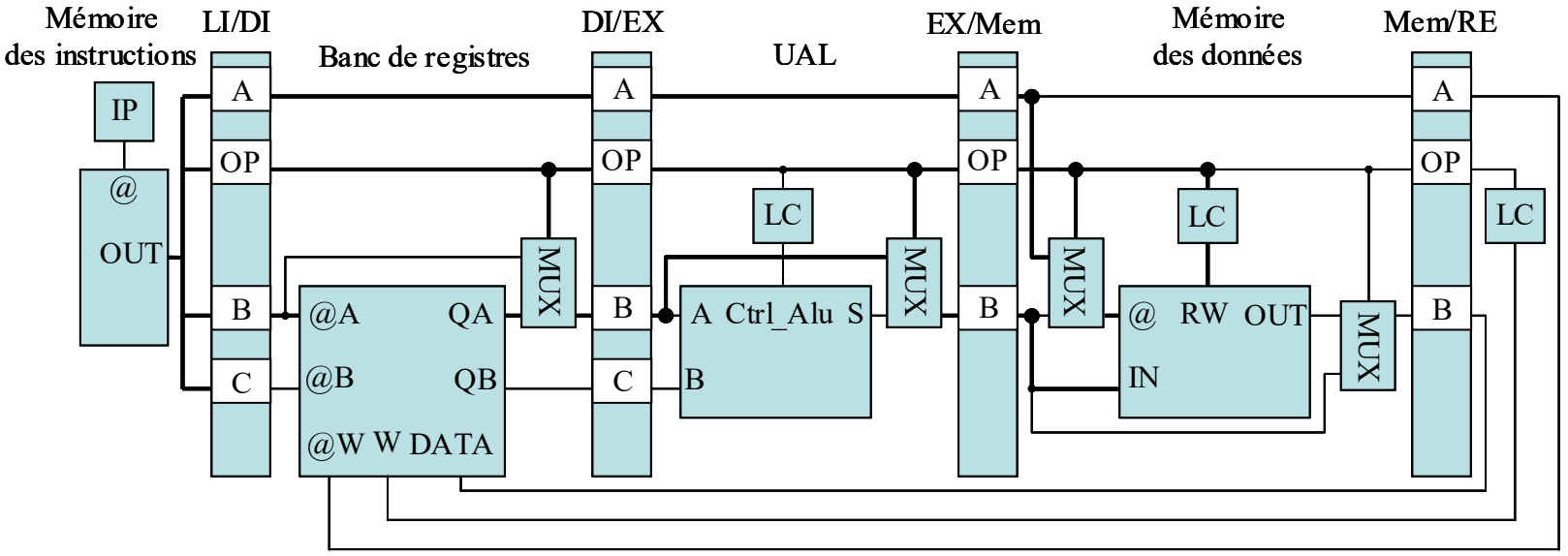


Illustration 5 : Instructions AFC, COP, ADD, MUL, DIV, SOU, LOAD, STORE

Unité de contrôle

Parallèlement au chemin de données, vous aurez à compléter la procédure de contrôle dont le rôle est de définir l'état des signaux de commandes des multiplexeurs, des bancs de mémoire et autres éléments propres à chaque étage du pipeline. Ce contrôle est fonction des instructions en cours d'exécution.

Gestion des aléas

Par la suite, vous serez confrontés à des situations d'aléas. Il existe deux types d'aléas : les aléas de données et les aléas de branchement. Nous nous intéresserons aux aléas de données. Afin d'illustrer ces situations, considérons le programme suivant :

```
…
NOP
NOP
[R1] ← 12
[R2] ← [R1]
NOP
NOP
…
```

Supposons que le registre *R1* contient *14* et le registre *R2* contient *8*. A l'issu de ce programme, la valeur affectée au registre *R2* doit être *12*. Or, lorsque l'écriture associée à la première instruction sera réalisée (étage 5), la valeur associée au registre *R1* dans l'étage 4 ne sera pas encore *12* mais *14*. La figure 9 présente ce cas de figure. Pour résoudre ce problème, nous vous conseillons d'utiliser une entité de gestion des aléas, qui aura pour rôle d'injecter des « bulles » en cas de détection d'aléas. Ces bulles permettent de temporiser l'exécution des étages inférieurs lors de la détection d'un conflit avec les étages supérieurs. Pour l'exemple précédent, lorsque la première instruction se situe au niveau du second pipe-line – et la seconde instruction au niveau du premier pipe-line – la comparaison entre les données du premier pipe-line et du second pipe-line permet de détecter cet aléa :

$$OP_{DI\_EX} = AFC \text{ and } OP_{LI\_DI} = COP \text{ and } A_{DI\_EX} = B_{LI\_DI}$$

Pour ce cas, l'horloge du premier pipe-line est inhibée, le temps que la première instruction finisse. Attention toutefois que le « vide » engendrée par cette temporisation n'altère pas l'état des registres. Pour éviter ceci, il faut injecter un *NOP* (*NO-OPERATION*), qui constitue la bulle. La résolution des autres cas d'aléas suit le même raisonnement.

t	Entrée LI/DI	Entrée DI/EX	Entrée EX/Mem	Entrée Mem/RE	Entrée écriture banc de registre	Contenu banc de registres
1 0	OP = AFC A = R1 B = 12	NOP	NOP	NOP	NOP	R1 = 14 ; R2 = 8
1 0	OP = COP A = R2 B = R1	OP = AFC A = R1 B = 12	NOP	NOP	NOP	R1 = 14 ; R2 = 8
1 0	NOP	OP = COP A = R2 **B = 14**	OP = AFC A = R1 B = 12	NOP	NOP	R1 = 14 ; R2 = 8
1 0	NOP	NOP	OP = COP A = R2 B = 14	OP = AFC A = R1 B = 12	NOP	R1 = 14 ; R2 = 8
1 0	NOP	NOP	NOP	OP = COP A = R2 B = 14	OP = AFC A = R1 B = 12	**R1 = 12** ; R2 = 8
1 0	NOP	NOP	NOP	NOP	OP = COP A = R2 B = 14	R1 = 12 ; **R2 = 14**

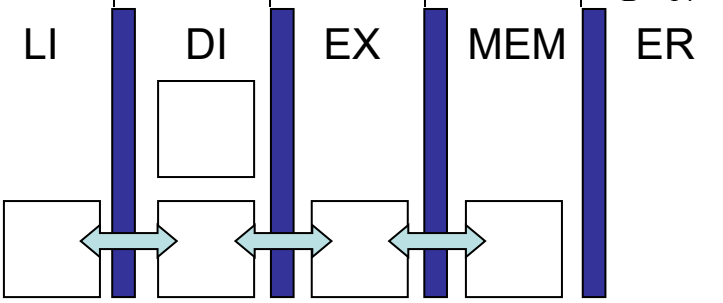


Figure 6 : Exemple d'aléa de données