ETUDE ET CONCEPTION D'UN PROCESSEUR MONOCYCLE : CHEMIN DE DONNEES ET CONTROLE

CARACTÉRISTIQUES DES ARCHITECTURES CISC ET RISC

CISC : Complex Instructions Set Computer. **RISC : Reduced Instructions Set Computer**

Critères	CISC	RISC
Jeu d'instructions	Complexe ; Sémantique proche des HLL ; Redondance	Réduit ; Sémantique proche du matériel ; Pas de redondance.
Format des instructions	Variable	Constant
Modes d'adressage	Riches, puissants et complexes	Très réduits, simples mais puissants
Conséquences sur la programmation en langage machine		
Programmation en assembleur	Relativement facile ; Obtention de programmes optimisés.	Relativement difficile ;
Conséquences sur la conception des compilateurs		
Ecriture des compilateurs	Difficile ; Les concepteurs n'utilisent que les instructions et les modes d'adressage généralistes ⇒ Les programmes générés sont moins optimisés	Facile ; Les concepteurs exploitent totalement les instructions et les modes d'adressage ⇒ Les programmes générés sont très optimisés
Conséquences sur l'architecture matérielle		
Unité de contrôle	Complexe ; Grande surf. de silicium ; Microprogrammée ; Exemple : 68000 (70K transistors.) 80% de la puce est utilisée par l'unité de contrôle.	Simple ; Petite surface de silicium ; Câblée ; Surface récupérée au profit des registres et de la taille des caches
Chemin de données	Non pipeliné ; Si pipeliné : complexe, irrégulier et non linéaire ; Performances d'exécution limitées.	Pipeliné ; Simple, régulier et linéaire ; Performances d'exécution accrues.

BREVE INTRODUCTION A L'ARCHITECTURE DU R3000 (MIPS) (1/2)

Les registres :

- **32 registres généraux de 32 bits (GPR : General Purpose Register) : R0, R1…R31 (R0 = 0).**
- **Registres virgule flottante (FPR : Floating Point Register) :**
  - ***Simple précision :*** 32 registres 32 bits (F0, F1, …F31).
  - ***Double précision :*** 16 registres 64 bits regroupés par paire de registres pairs-impairs (F0, F2, …F30).

Les types et formats des données :

- ***Les entiers :*** **octet, demi-mot (16 bits) et mot (32 bits).**
- ***Les réels :*** **simple précision (mot 32 bits), double précision (double mots 64 bits).**

Remarque : **les opérations sur les entiers (octet, demi-mot) sont toujours réalisées avec extension à 32 bits.**

Les modes d'adressage :

- **3 modes seulement : registres, immédiat et indirect avec déplacement signé sur 16 bits.**
- **mode avec déplacement :**
  - **Indirect par registre avec déplacement : d(Ri).**
  - **Indirect par registre : 0(Ri).**
  - **Absolu : d(R0).**

5 modes d'adressage alors que l'architecture n'en fournit que 3

L'architecture du R3000 est une architecture LOAD/STORE ==> Les références mémoires sont effectuées entre mémoire et registres (GPR, FPR) par LOAD (chargement) et STORE (rangement).

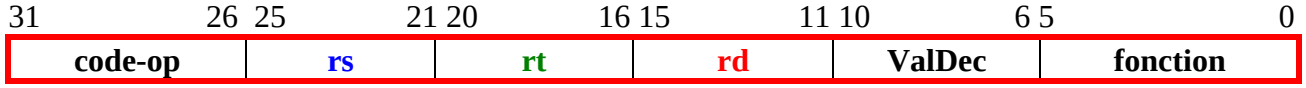
BREVE INTRODUCTION A L'ARCHITECTURE DU R3000 (MIPS) (1/2)

Les opérations de DLX : 4 grandes classes d'instructions.

- Les chargements et rangements (références mémoire) : LOAD, STORE.
- Les opérations (UAL) arithmétiques et logiques : ADD, SUB, AND, OR, XOR, …
- Les branchements et sauts.
- Les opérations arithmétiques en virgule flottante.

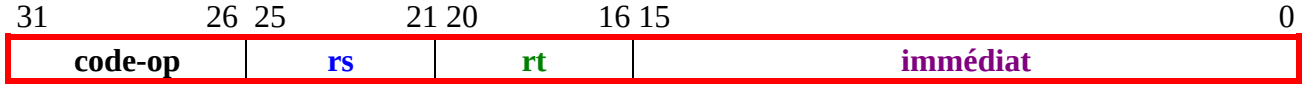
Les formats des instructions : format unique 32 bits

- *Instruction de type R :* arithmétiques et logiques de registre à registre : rd ← (rs) op (rt).



- *Instruction de type I :* référence mémoire, arithmétiques et logiques immédiat, branchements conditionnels :
  - **Références mémoire :**
    - √ Chargement : rt ← M[(rs)+(immédiat avec extension de signe)],
    - √ Rangement : M[(rs)+(immédiat avec extension de signe)] ← (rt).
  - **Branchements :**

    Compare (rs) et (rt) si la condition est vraie CP ←(CP)+(immédiat avec extension de signe <<2).
  - **Arithmétiques et logiques :**
    - √ Arithmétiques signés : rt ← (rs) op (immédiat avec extension de signe),
    - √ Arithmétiques non signés ou logiques : rt ← (rs) op (immédiat extension avec 0).



- *Instruction de type J :* sauts inconditionnels (JMP).

INTRODUCTION

Trois paramètres définissent les performances d'un processeur :

1. **le nombre d'instructions machine requis pour un programme donné,**
2. **la durée du cycle d'horloge,**
3. **le nombre de cycles d'horloge par instruction (CPI).**

- **Le premier paramètre est fonction l'architecture du jeu d'instructions et du compilateur.**
- **Les deux derniers, sont définis lors de la conception et la mise en œuvre du processeur.**

Concevoir une mise en œuvre contenant le noyau du jeu d'instructions du R3000 (MIPS) :

- **les instructions de référence mémoire : chargement d'un mot (lw), rangement d'un mot (sw)**
- **les instructions arithmétiques et logiques : add, sub, and, or, et slt,**
- **l'instruction de branchement si égal (beq)**
- **l'instruction de saut (j).**

La fréquence d'horloge et le CPI sont influencés par le choix de diverses stratégies de mise en œuvre.

Les principes fondamentaux de conception sont :

- ***"faire que les cas les plus fréquents soient les plus rapides"***
- ***"la simplicité favorise la régularité".***

DESCRIPTION GENERALE DE LA MISE EN ŒUVRE

Une grande ressemblance dans la mise en œuvre de chacune des instructions : étapes identiques et actions similaires.

Les deux premières étapes sont identiques pour toutes les instructions :

1. **Envoyer le (CP) à une mémoire qui contient le code pour extraire l'instruction.**
2. **Lire un ou deux registres en utilisant les champs de l'instruction.**

L'étape suivante à mener dépend à priori du type de l'instruction.

Or, des similitudes existent entre instructions et même entre classes d'instructions différentes ➔ toutes les classes d'instruction utilisent l'UAL après que les registres aient été lus :

- **les instructions de références mémoire pour un calcul d'adresse réelle,**
- **les instructions arithmétiques et logiques pour exécuter le code-op,**
- **les branchements pour les comparaisons.**

Enfin les actions requises pour achever chaque type d'instruction diffèrent :

- **une instruction de référence mémoire devra accéder à la mémoire**
- **une instruction arithmétique et logique devra écrire dans un registre les données issues de l'UAL,**
- **une instruction de branchement détermine l'adresse de la prochaine instruction.**

La simplicité et la régularité du jeu d'instructions simplifient la mise en œuvre en rendant similaires les exécutions de la plupart des instructions.

METHODOLOGIE DE SYNCHRONISATION (1/2)

Définitions :

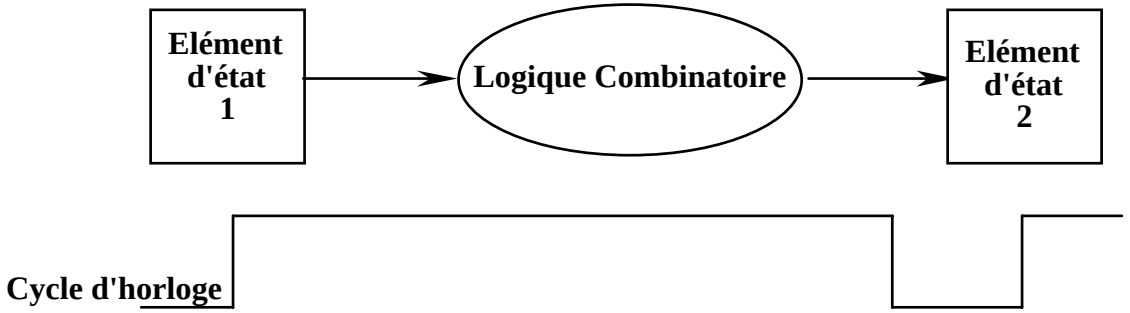
- **Les unités fonctionnelles qui traitent des données sont toutes *combinatoires*. Un élément combinatoire produit toujours le même résultat sur ses sorties pour les mêmes données sur ses entrées.**
- **Les unités fonctionnelles qui contiennent un état sont toutes *séquentielles*. Un élément contient un état s'il possède une capacité de stockage interne.**

Méthodologie de synchronisation :

➔ **But : prévenir des aléas de fonctionnement. Elle définit à quel moment les signaux peuvent être lus et écrits.**

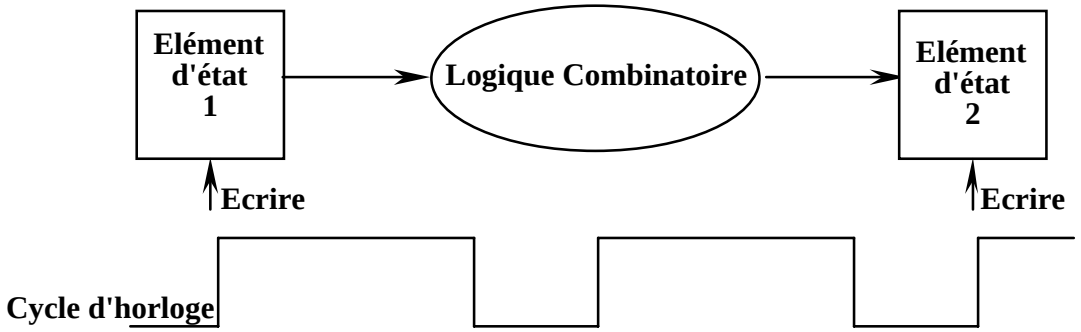
- **Considérons une méthodologie de synchronisation déclenchée par front d'impulsion.**
- **Une mise à jour d'un élément d'état n'est effectuée que sur un front d'impulsion.**
- **Les entrées de tout ensemble combinatoire doivent *provenir* (lues) d'un ensemble d'éléments d'état, et les sorties doivent être *repartir* (écrites) dans un ensemble d'éléments d'état.**
- **Les entrées sont des valeurs *issues* d'un cycle d'horloge à l'instant t_i, tandis que les sorties sont des valeurs qui seront *utilisées* dans un prochain cycle au plus tôt à l'instant t_{i+1}.**
-

Exemple 1 **: bloc combinatoire pouvant être multifonctionnel opérant en un seul cycle d'horloge**



METHODOLOGIE DE SYNCHRONISATION (2/2)

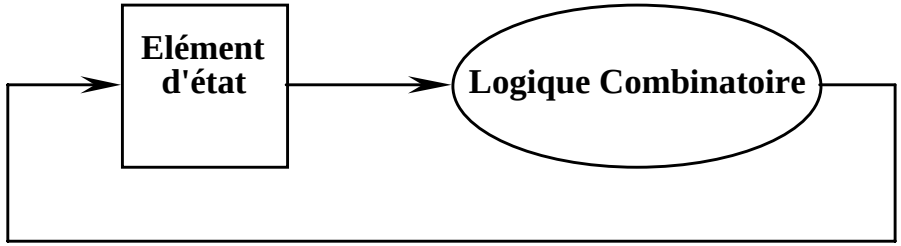
Exemple 2 **: bloc opératoire pouvant être multifonctionnel opérant en plusieurs cycles.**



- **L'écriture de l'élément d'état 2 doit être contrôlé pour que la mise à jour ne se fasse pas à chaque cycle d'horloge mais seulement pour certaines impulsions.**

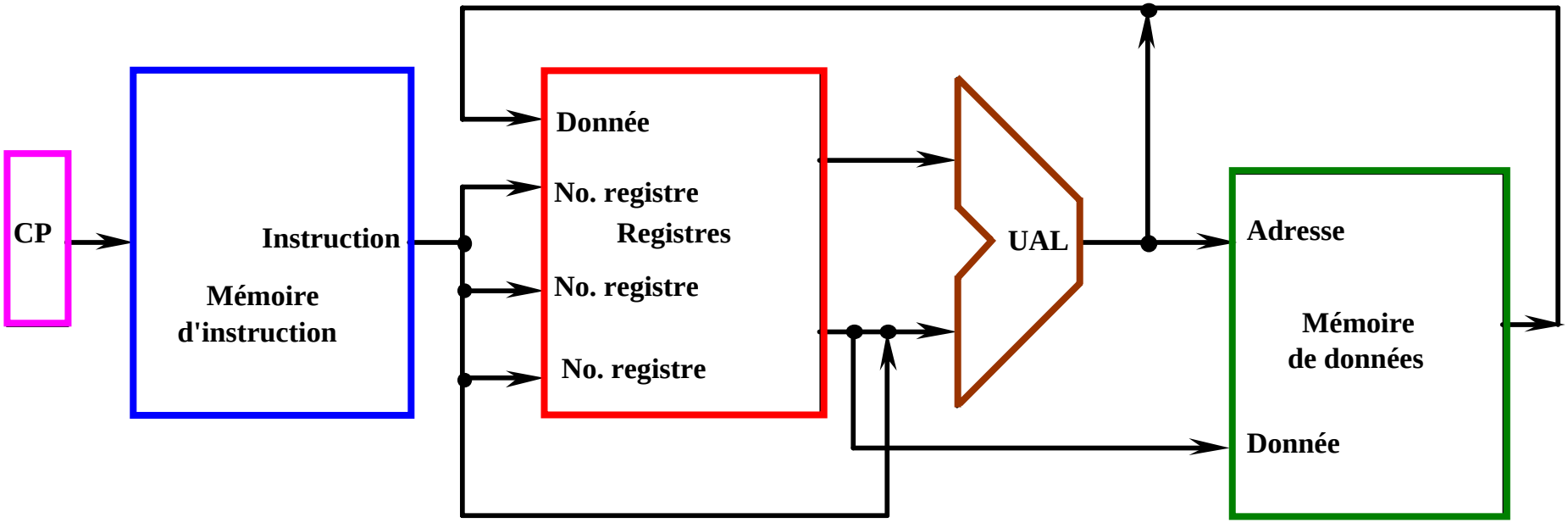
- **Les éléments d'état auront besoin de signaux d'écriture *explicites* qui doivent être coordonnés pour que la synchronisation soit cohérente.**

Exemple 3 **: Bloc opératoire multifonctionnel opérant dans un même cycle d'horloge : lecture d'un registre, envoie de sa valeur à travers le bloc combinatoire traitement et écriture dans le registre.**



LA MISE EN ŒUVRE DU SOUS-ENSEMBLE MIPS

Une mise en œuvre simple qui utilise pour chaque instruction *un cycle d'horloge unique* de longue durée et suit la forme générale de la figure synoptique suivante :



Dans ce modèle, chaque instruction débute son exécution sur un front d'horloge et l'achève au front suivant.

- **cette approche, est plus facile à comprendre,**
- **mais en pratique elle n'est pas utilisable en tant que telle,**
- **elle serait plus lente qu'une mise en œuvre multi-cycles.**

Une mise en œuvre multi-cycles permet aux différents instructions ou types d'instruction de prendre des nombres différents de cycles d'horloge, mais le cycle d'horloge est plus court que le cycle unique.

CHEMIN DE DONNEES COMMUN (1/2)

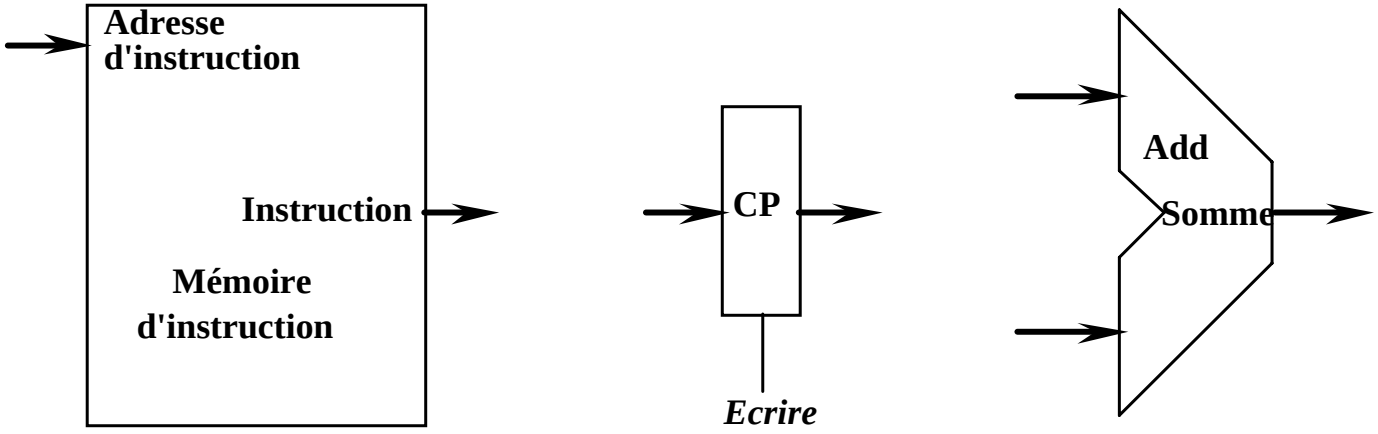
Examiner quels sont les éléments du chemin de données nécessaires pour chaque instruction et construire à partir de ces éléments les sections du chemin de données pour chaque type d'instruction.

Une étape commune à l'exécution de toutes les instructions est :

- **l'extraction (lecture) de l'instruction,**
- **la préparation de l'adresse de l'instruction suivante.**

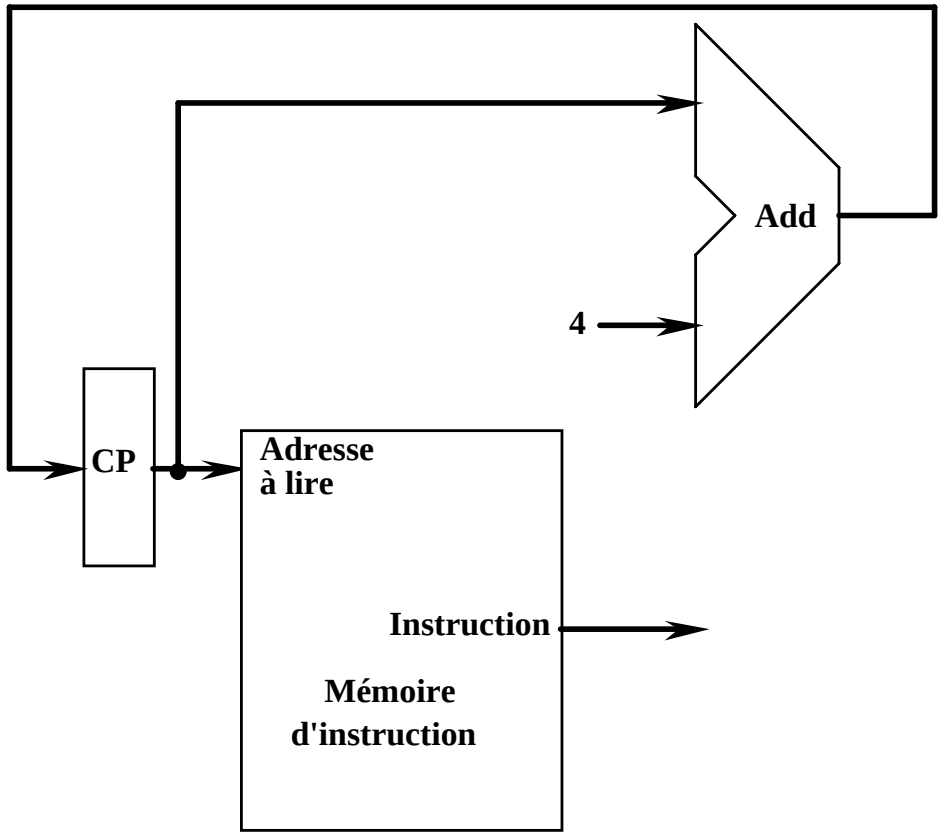
Nous avons donc besoin :

- **d'un élément d'état pour conserver et fournir les instructions à partir d'une adresse : *une mémoire,***
- **d'un élément d'état pour conserver l'adresse de l'instruction : *un compteur de programme (CP),***
- **d'un élément combinatoire pour incrémenter le CP à l'adresse de la prochaine instruction : *un additionneur*.**



CHEMIN DE DONNEES COMMUN (2/2)

- **Le chemin de données de cette étape commune est le suivant :**



CHEMIN DE DONNEES POUR INSTRUCTION AU FORMAT R (1/2)

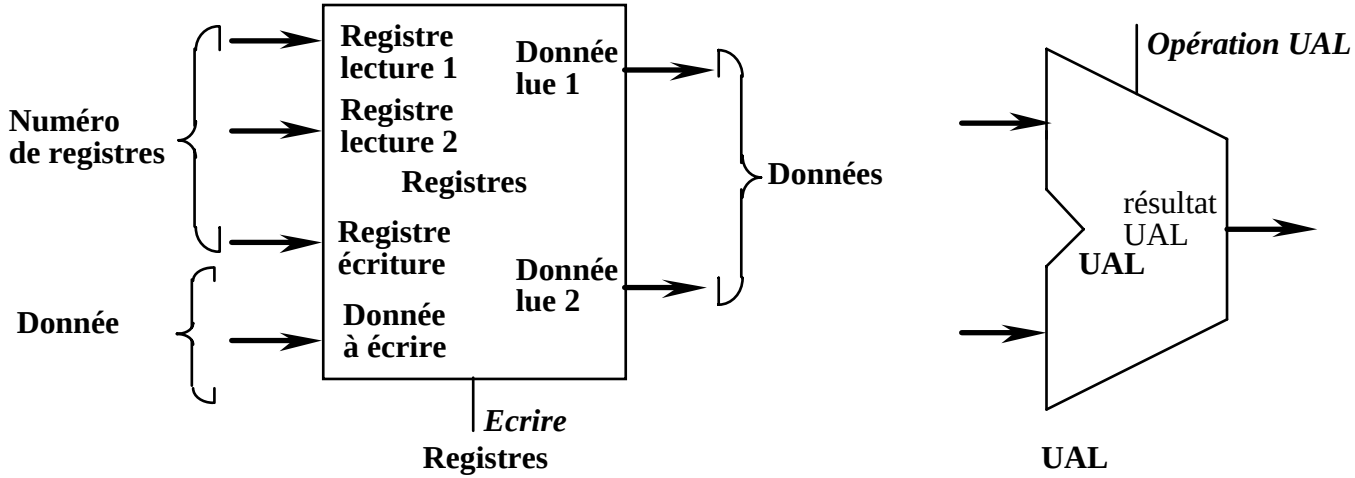
Les instructions *arithmétiques-logiques* au *format R* :

- **cette classe d'instructions inclut : *add, sub, and, or,* et *slt,***
- **elles lisent toutes 2 registres, effectuent une opération UAL sur leur contenu, et écrivent le résultat.**
- **une instance usuelle d'une telle instruction est : add R1, R2, R3 ⇒ R1 ← (R2) + (R3).**

Les éléments nécessaires pour le chemin de données sont :

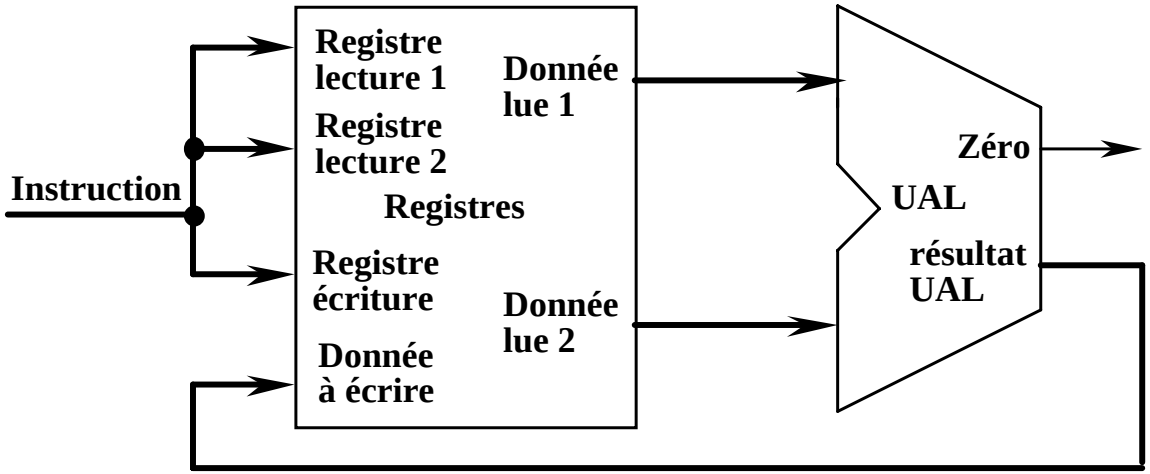
- ***une banque de registres :* structure organisée sous forme d'une banque de registres. Elle autorise simultanément 2 lectures et 1 écriture de registres en spécifiant le numéro de chacun dans la banque,**
- ***une UAL :* pour travailler sur les valeurs extraites des registres.**

Les instructions *au format R* ont 3 opérandes ➔ lire 2 mots dans la banque et y écrire un mot ➔ au total de *4 bus en entrées* (3 pour les numéros de registres et 1 pour la donnée) et *de 2 bus en sorties* (pour des données).



CHEMIN DE DONNEES POUR INSTRUCTION AU FORMAT R (2/2)

Le chemin de données des instructions *arithmétiques et logiques* après recensement des différents éléments est le suivant :



CHEMIN DE DONNEES POUR REFRENCE MEMOIRE (1/2)

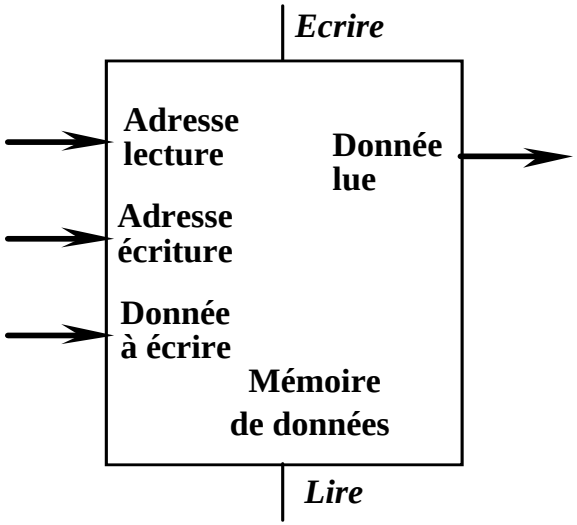
Les instructions de *référence mémoire* au *format I*, (chargement / rangement)

***Exemple : 1w R1, d(R2)* ou *sw R1, d(R2)*. *d* : déplacement signé codé sur 16 bits**

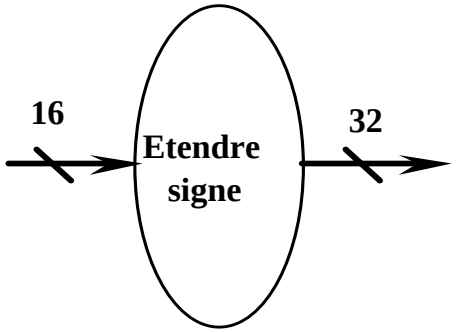
- **Elles calculent une adresse mémoire registre de base + déplacement : (R2) + (d)**
- **Si un rangement, la valeur à stocker doit être lue depuis la banque de registres (R1).**
- **Si un chargement, la valeur lue en mémoire doit être écrite dans le registre spécifié (R1) de la banque.**

Les éléments nécessaires pour le chemin de données sont :

- **La banque de registres et l'UAL, sont déjà vues pour les instructions du format R.**
- **Une unité pour l'extension de signe, du champ immédiat de l'instruction, à une valeur signée de 32 bits.**
- **Une mémoire de données ➔ 2 signaux lecture et écriture et 2 bus pour lire ou écrire les données en mémoire.**



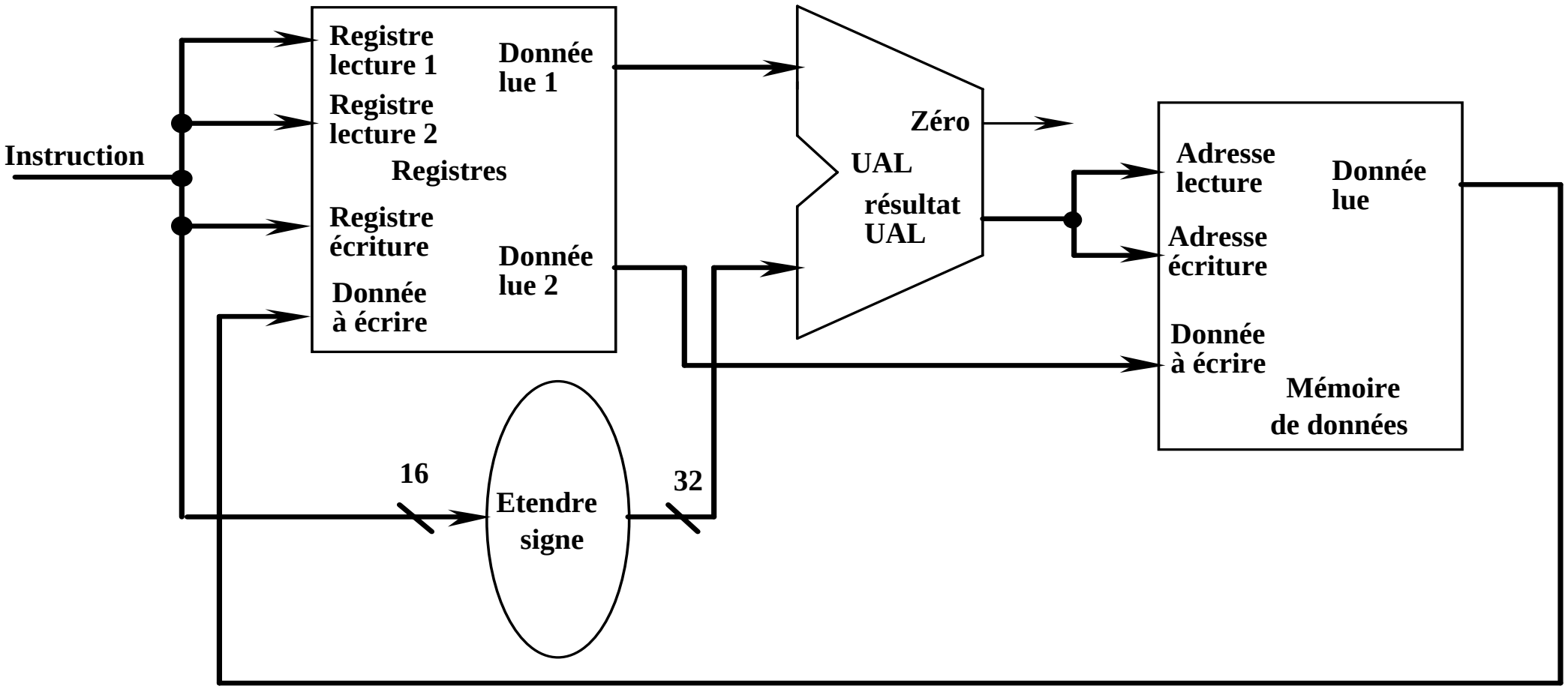
Unité mémoire de données



Unité d'extension de signe

CHEMIN DE DONNEES POUR REFRENCE MEMOIRE (2/2)

Le chemin de données des instructions de *référence mémoire* après assemblage des différents éléments est le suivant :



CHEMIN DE DONNEES POUR BRANCHEMENTS CONDITIONNELS (1/2)

Les instructions de *branchement conditionnel* au *format I*

Exemple : beq R1, R2, label

Une instruction de banchement a 3 opérandes : 2 registres dont on teste l'égalité, et un déplacement signé de 16 bits contenu dans l'instruction utilisé pour calculer l'adresse de destination du branchement.

Elles effectuent 2 opérations :

- **Calcul de l'adresse de destination du branchement : CP courant + (déplacement étendu sur 32 bits)*4**
  - **La base pour le calcul de l'adresse de branchement est l'adresse de l'instruction suivante CP+4.**
  - **Le champ immédiat est étendu sur 32 bits est décalé de 2 bits à gauche pour en faire un déplacement d'un mot ; ce décalage augmente d'un facteur 4 l'étendue effective du déplacement.**
- **Comparaison du contenu des registres.**

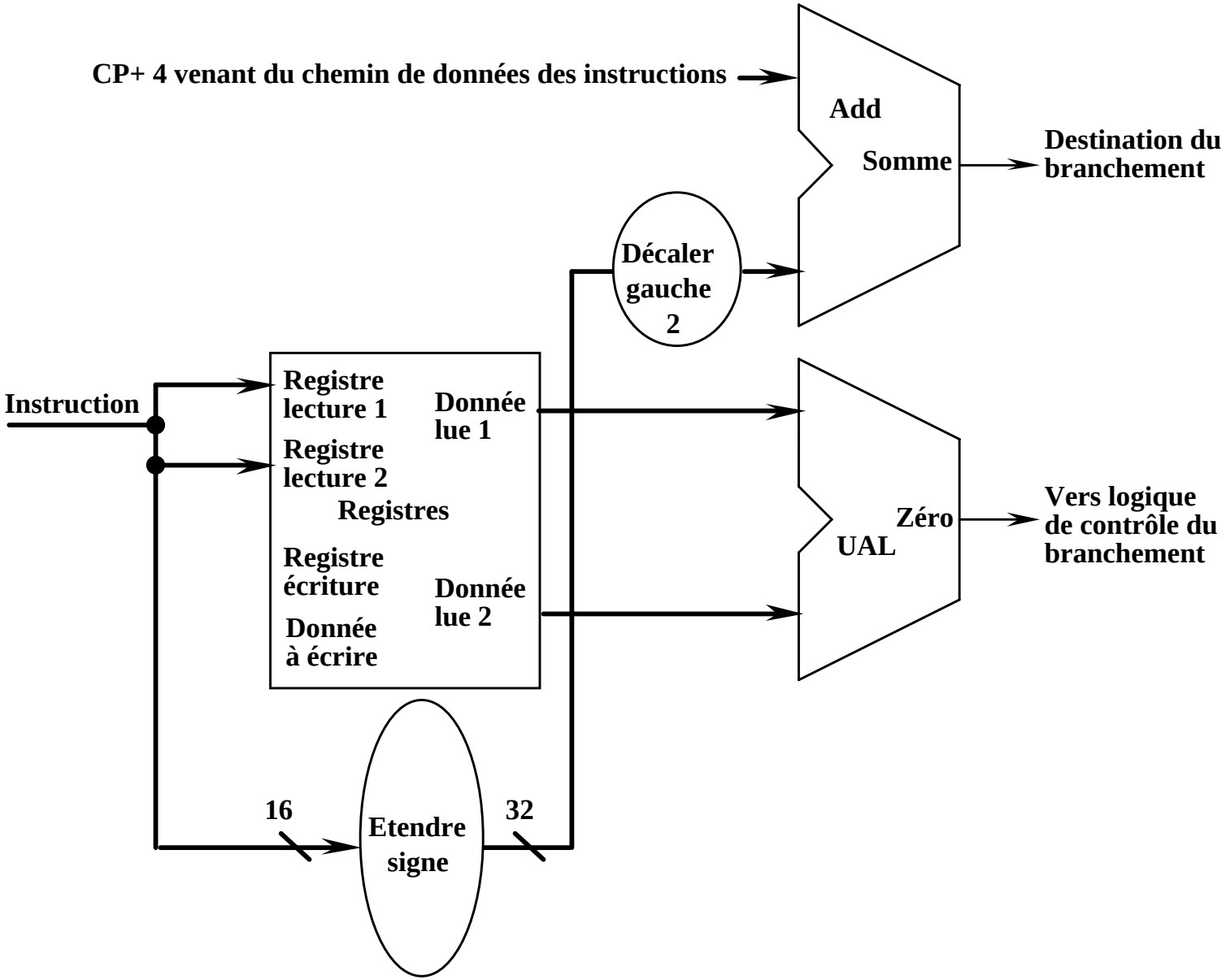
Les éléments nécessaires pour le chemin de données sont :

- **Pour le calcul d'adresse :**
  - **une unité pour l'extension de signe, du champ immédiat de l'instruction, à une valeur signée 32 bits,**
  - **une unité de décalage de 2 bits,**
  - **un additionneur,**
- **Pour la comparaison :**
  - **le banc de registres et l'UAL, déjà vus,**
  - **modification de partie extraction du chemin de données.**

Les deux registres opérandes sont envoyés à l'UAL en positionnant le contrôle pour une soustraction.
Si Z = 1, alors égalité et le branchement sera effectué sinon l'instruction suivante sera extraite.

CHEMIN DE DONNEES POUR BRANCHEMENTS CONDITIONNELS (2/2)

Le chemin de données *d'un branchement* combinant ces éléments est le suivant :



CHEMIN DE DONNEES POUR SAUT (1/2)

L'instruction de saut :

Elle opère de la manière suivante : elle remplace la valeur de CP par :

CP ← CP[31. .28] // champ adresse absolue de 26 bits de l'instruction // 00.

Après examen des chemins de données nécessaires pour chaque type d'instruction, nous allons les fusionner en un chemin de données unique et ajouter le contrôle pour compléter la mise en œuvre.

CREATION D'UN CHEMIN DE DONNEES UNIQUE (1/4)

Contraintes d'un modèle de chemin de données monocycle :

- **toute instruction est exécutée en un cycle d'horloge ;**
- **donc aucune ressource du chemin de données ne peut être utilisée plus d'une fois par instruction ;**
- **tout élément utilisé plusieurs fois dans un cycle doit être dupliqué ;**
- **donc une mémoire pour les instructions et une pour les données.**

Cette mise en œuvre simple couvre les instructions de :

- **références mémoire telles que : *(lw, sw)* ;**
- **branchements relatifs telle que : *(beq)* ;**
- **arithmétiques et logiques telles que *(add, sub, and, or, s1t)*.**

le modèle sera ensuite complété par le chemin de donnée de l'instruction de saut *(j)*.

La construction du chemin de données complet est obtenue en assemblant les segments du chemin de données de chaque classe d'instructions, en y ajoutant les lignes de contrôle nécessaires.

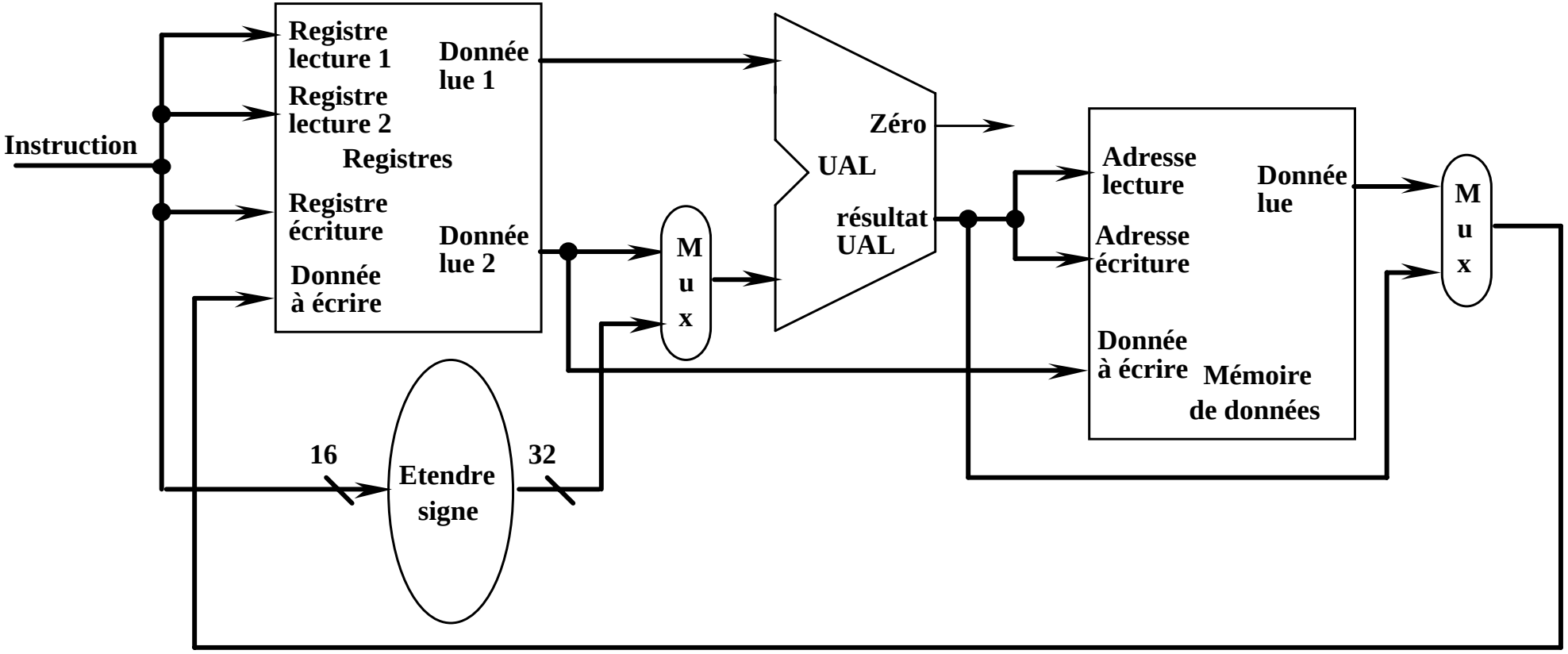
Le partage des unités fonctionnelles par des flots différents d'instructions est réalisé généralement à l'aide d'un multiplexeur.

CREATION D'UN CHEMIN DE DONNEES UNIQUE (2/4)

Fusion des chemins de données des instructions de référence mémoire et arithmétiques et logiques.

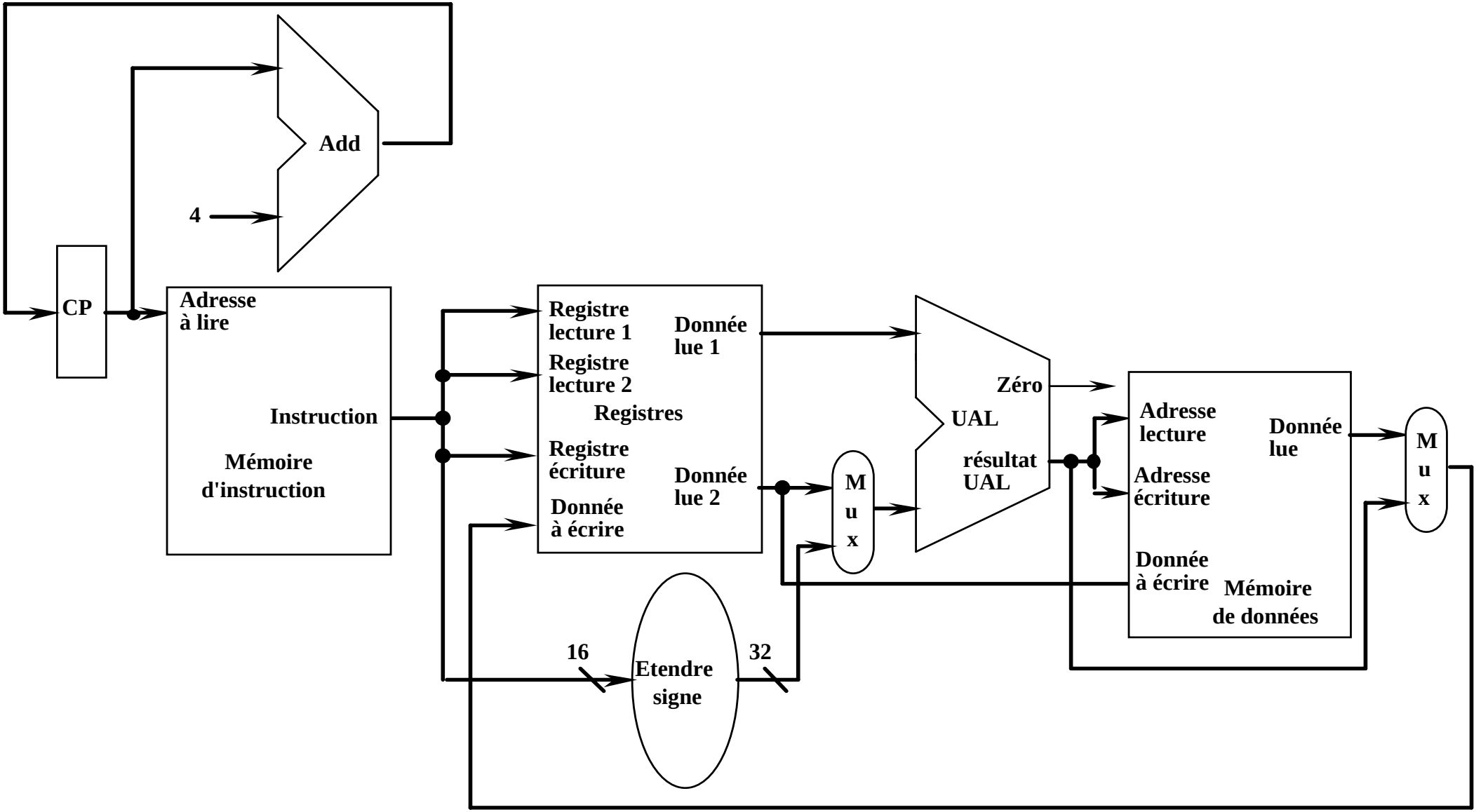
une grande similitude des deux chemins de données ; les différences majeures sont :

- **La 2ème entrée de l'UAL est : soit un registre (format R), soit les 32 bits de l'unité d'extension de signe (Format I).**
- **La valeur écrite dans le registre résultat provient de l'UAL (instruction type R) ou de la mémoire (chargement).**



CREATION D'UN CHEMIN DE DONNEES UNIQUE (3/4)

Fusion du résultat de la combinaison précédente avec la partie extraction du chemin de données.

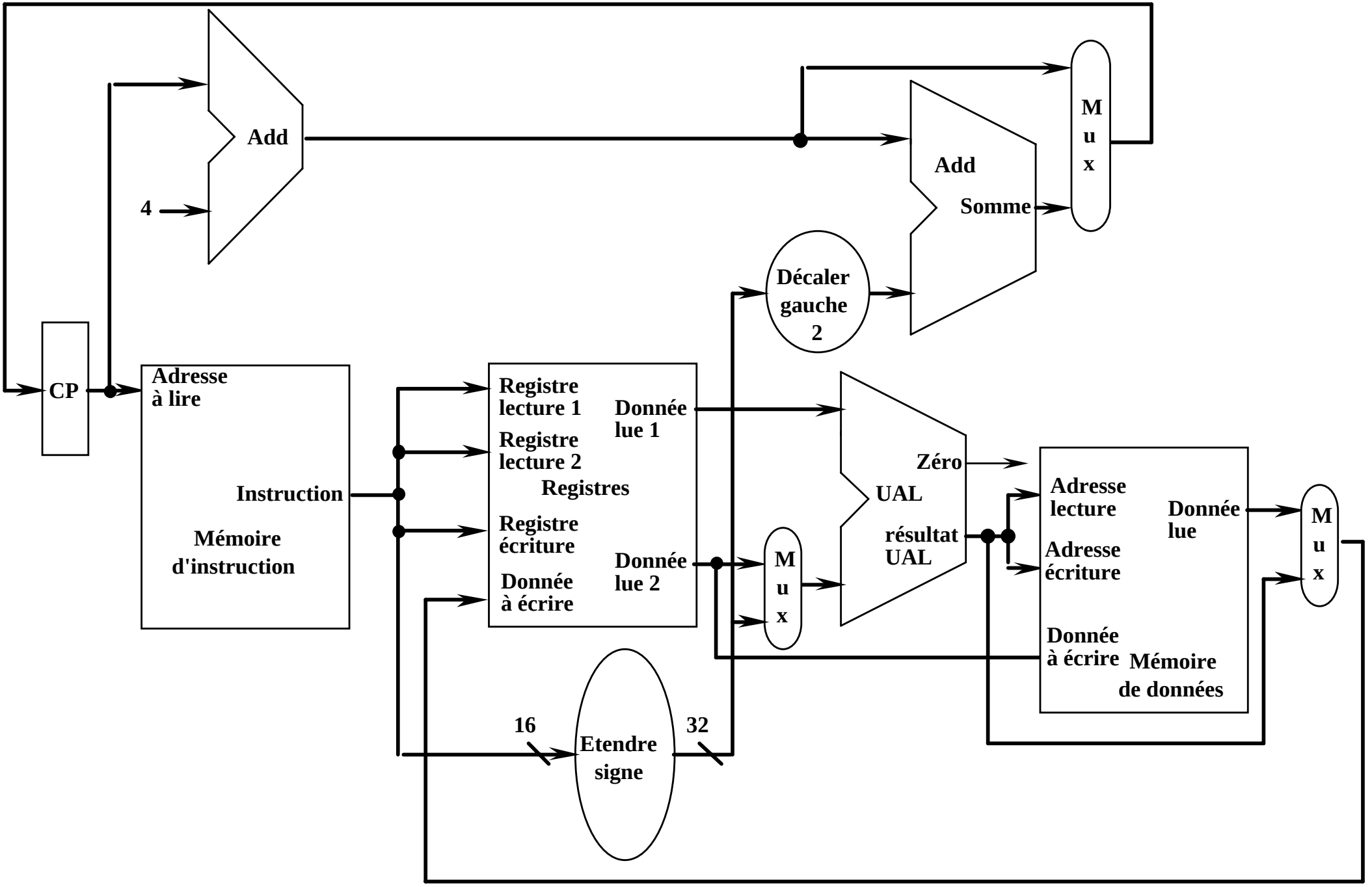


CREATION D'UN CHEMIN DE DONNEES UNIQUE (4/4)

Fusion du résultat de la combinaison précédente avec le chemin de données du branchement.

- **Un multiplexeur supplémentaire est nécessaire pour choisir soit l'adresse de l''instruction suivante (CP+4) soit l'adresse de destination du branchement à écrire dans le CP.**

- **Le CP sera écrit avec une de ces deux valeurs à chaque cycle d'horloge, donc aucun un signal de contrôle d'écriture explicite n'est nécessaire.**


Add
4
M
u
x
Add
Somme
Décaler
gauche
2
CP
Adresse
à lire
Instruction
Mémoire
d'instruction
Registre
lecture 1
Registre
lecture 2
Registres
Registre
écriture
Donnée
à écrire
Donnée
lue 1
Donnée
lue 2
M
u
x
Zéro
UAL
résultat
UAL
Adresse
lecture
Adresse
écriture
Donnée
à écrire
Donnée
lue
Mémoire
de données
M
u
x
16
Etendre
signe
32

LE CONTROLE DE L'UAL (1/3)

- **Entrées de contrôle de L'UAL et les fonctions qu'elle doit effectuer selon le type de l'instruction.**

Entrée de contrôle de l'UAL	Fonction
000	Et
001	Ou
010	Addition
110	Soustraction
111	Positionner si inférieur

- **Positionnement des bits de contrôle de l'UAL en fonction des bits UALOp (codant la classe de l'instruction) et du code fonction pour une instruction de type R. Le champ code-op détermine le positionnement des bits de UALOp.**

Code-op instruction	UALOp classe	Opération de l'instruction	Code fonction	Actions de l'UAL désirées	Entrée de contrôle de l'UAL
LW	00	Chargement mot	xxxxxx	addition	010
SW	00	Rangement mot	xxxxxx	addition	010
BEQ	01	Branchement si =	xxxxxx	soustraction	110
type R	10	Addition	100000	addition	010
type R	10	Soustraction	100010	soustraction	110
type R	10	ET	100100	et	000
type R	10	OU	100101	ou	001
type R	10	Positionner si <	101010	soustraction	111

LE CONTROLE DE L'UAL (2/3)

Table de vérité des 3 bits de contrôle UAL en fonction de UALOp et du code de fonction.

UALOp		Code de fonction						Entrées de contrôle de l'UAL		
UALOp1	UALOp0	F5	F4	F3	F2	F1	F0	Opération 2	Opération 1	Opération 0
0	0→X	x	x	x	x	x	x	0	1	0
0 →X	1→X	x	x	x	x	x	x	1	1	0
1	x	x	x	0	0	0	0	0	1	0
1	x	x	x	0	0	l	0	1	1	0
1	x	x	x	0	1	0	0	0	0	0
1	x	x	x	0	1	0	1	0	0	1
1	x	x	x	1	0	1	0	1	1	1

Optimisation de la table de vérité ➔ tirer profit des termes indifférents.

Par exemple, le bit de faible poids du contrôle de l'UAL (Opération 0) est positionné par les deux dernières entrées de la table de vérité.

La table de vérité pour Opération 0 = 1.

UALOp		Code de fonction					
UALOp1	UALOp0	F5	F4	F3	F2	F1	F0
1	x	x	x	x	x	x	1
1	x	x	x	1	x	x	x

Opération 0 = UALOp1 (F0 + F3)

La table de vérité pour Opération l = 1.

UALOp		Code de fonction					
UALOp1	UALOp0	F5	F4	F3	F2	F1	F0
0	x	x	x	x	x	x	x
1	x	x	x	x	0	x	x

Opération 1= UALOp1\+F2

$\overline{UALOp_1} + UALOp_1 . \overline{F2}$

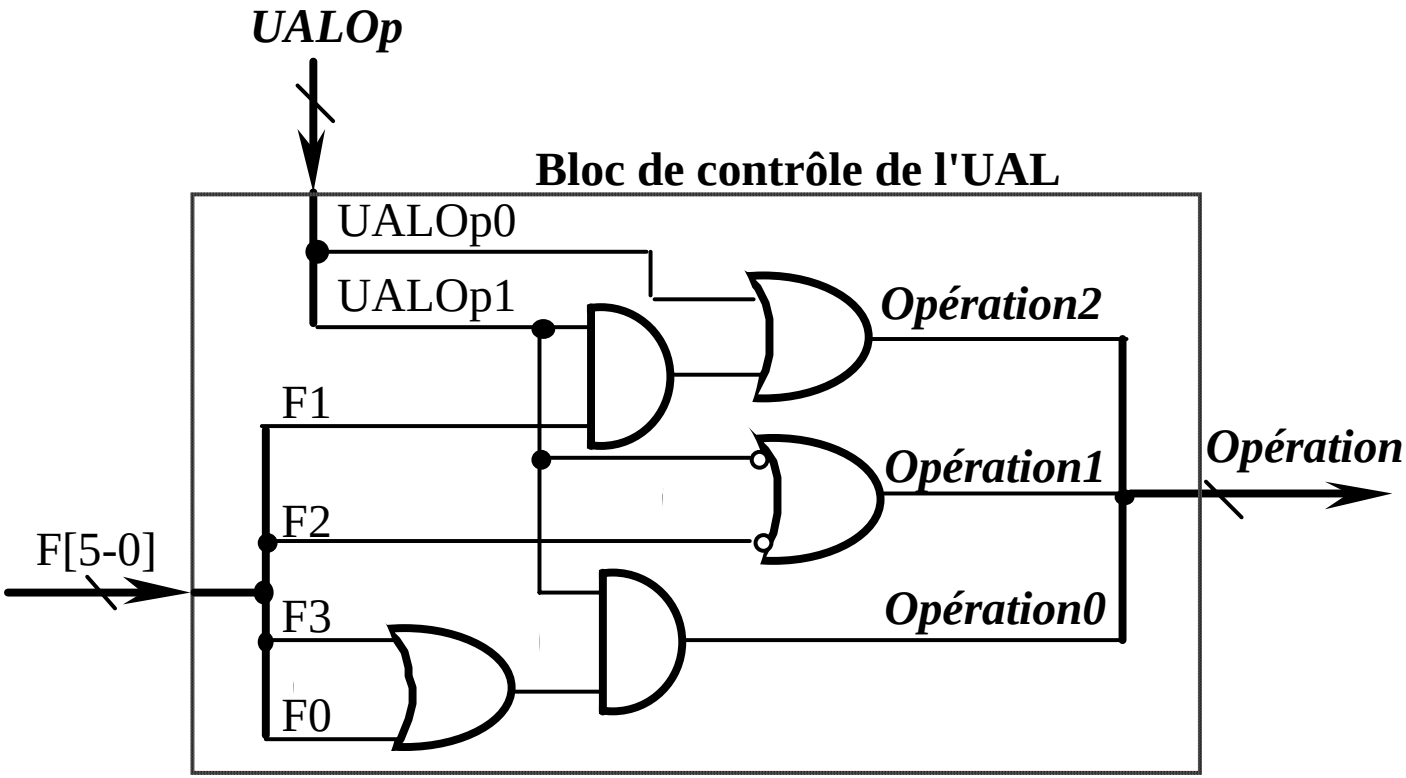
LE CONTROLE DE L'UAL (3/3)

- **La table de vérité pour Opération 2 = 1.**

UALOp		Code de fonction					
UALOp1	UALOp0	F5	F4	F3	F2	F1	F0
x	1	x	x	x	x	x	x
1	x	x	x	x	x	1	x

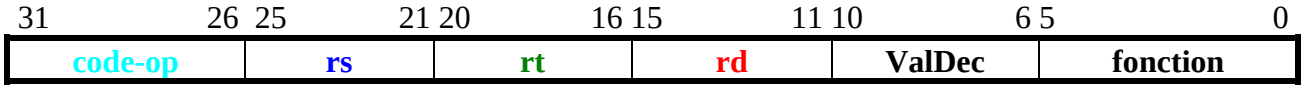
Opération 2 = UALOp0 + UALOp1.F1

A partie de cette simplification, nous pouvons construire la logique du bloc de contrôle de l'UAL.

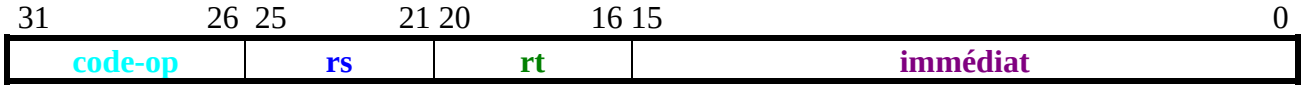


CONCEPTION DE L'UNITE CENTRALE DE CONTROLE (1/5)

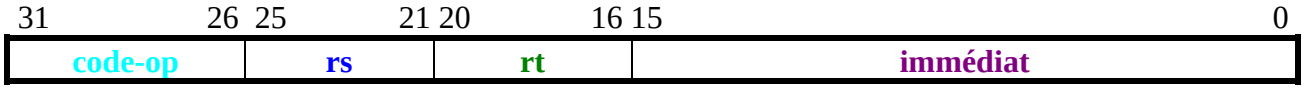
Pour comprendre comment il faudrait ajouter des bus dans le chemin de données pour acheminer les champs d'instruction, il est utile de revoir les formats pour les 3 types d'instruction.



Instruction de type R (code-op : 0)



Instructions de chargement/rangement ou arithmétiques et logiques en mode immédiat ; par exemple : **(code-op pour lw/sw=35/43)**



Instructions de branchement (code-op pour beq =4)

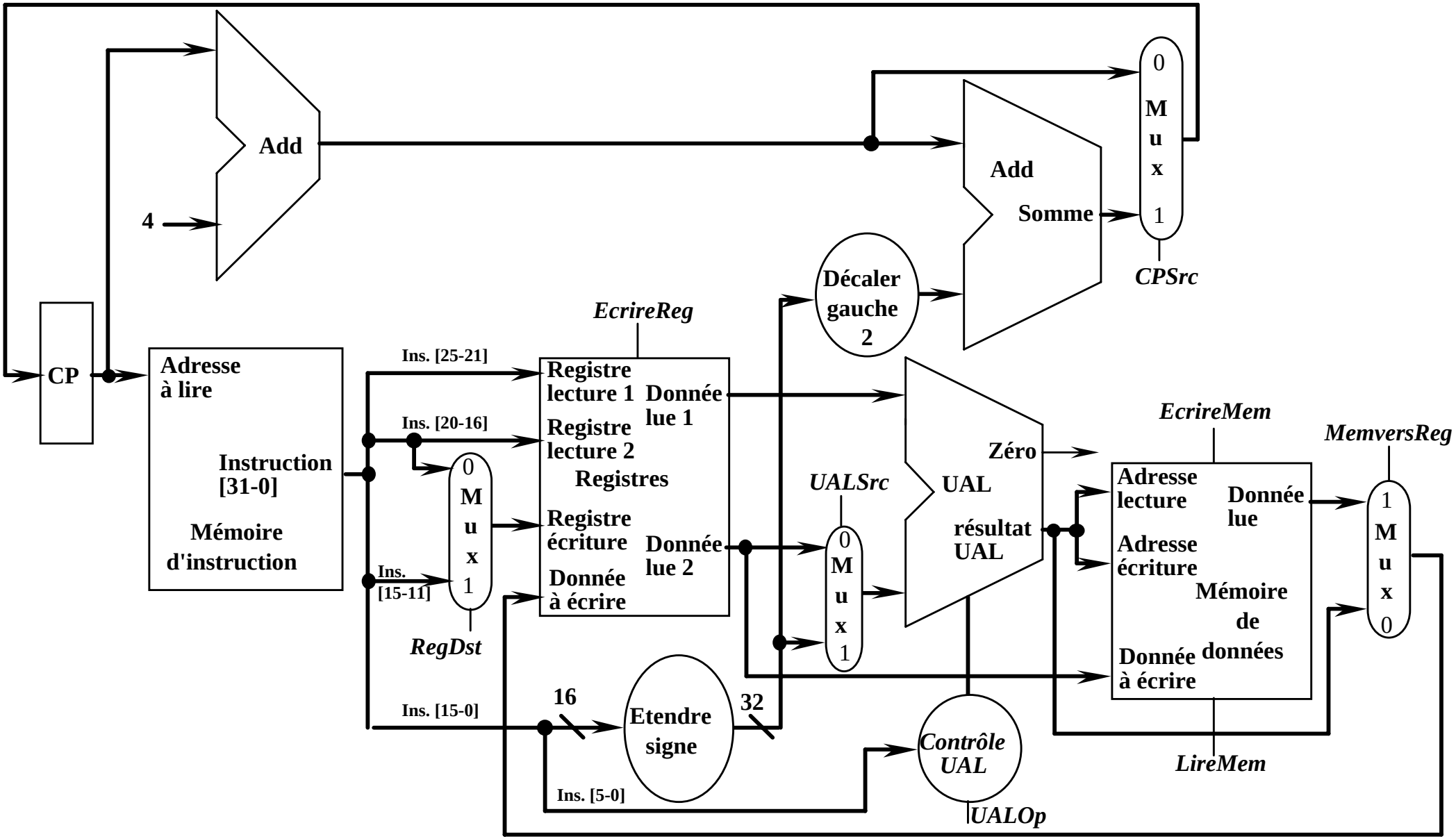
- **Le code opération est défini par code-op[31-26]. Nous référençons à ces bits par Op[5-0].**
- **Les n° des 2 registres à lire sont définis par rs[25-21] et rt[20-16]. (type R, branchement et rangement.**
- **Le registre de base pour les références mémoire est toujours rs[25-21].**
- **Le déplacement de 16 bits est toujours immédiat[15-0]. (branchements et les références mémoire).**
- **Le n° du registre à écrire est défini par : rd[15-11] pour type R et rt[20-16] pour instructions de chargement et arithmétiques et logiques en immédiat ➔ nécessite un multiplexage**

A partir de ces informations, nous pouvons ajouter au chemin de données :

- **les étiquettes des champs d'instruction,**
- **le multiplexeur supplémentaire pour aiguiller l'adresse du registre à écrire,**
- **les signaux d'écriture pour les éléments d'état et les signaux de contrôle pour les multiplexeurs.**

CONCEPTION DE L'UNITE CENTRALE DE CONTROLE (2/5)

Identification des lignes de contrôle et des champs d'instruction nécessaires pour le chemin de données.



CONCEPTION DE L'UNITE CENTRALE DE CONTROLE (3/5)

Description du fonctionnement des signaux de contrôle : 7 lignes de contrôle à 1 bit + les 2 lignes UALOp.

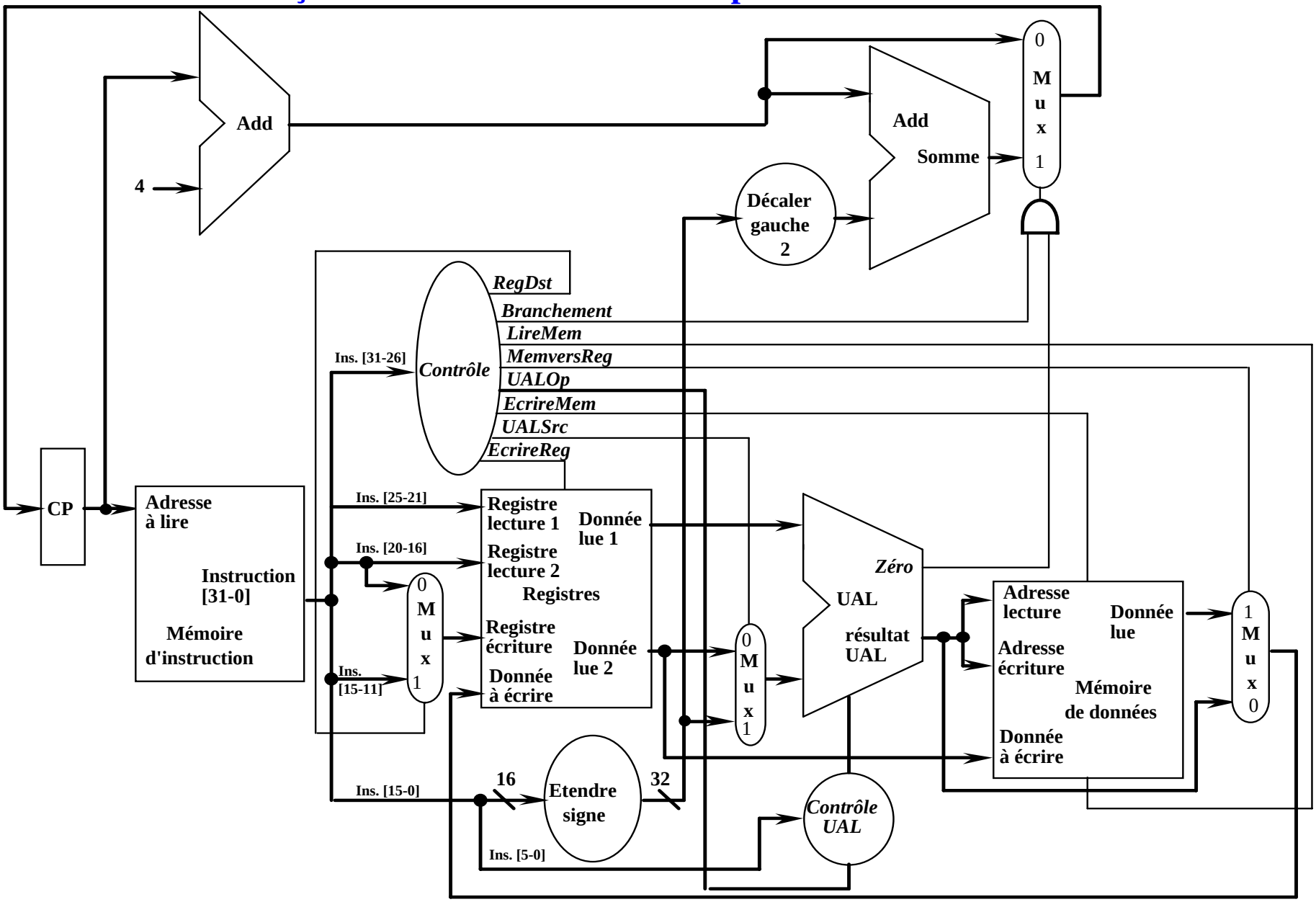
Nom du signal	Effet lorsque signal est inactif	Effet lorsque le signal est actif
LireMem	Aucun	Lecture mémoire : contenu placé sur «Donnée lue»
EcrireMem	Aucun	Ecriture mémoire : donnée placée sur «Donnée à écrire»
UALSrc	2nd opde UAL ← 2ème sortie de la banque de registres	2nd opde l'UAL ← unité d'extension de signe
RegDst	le n° de registre à écrire ← rt.	le n° de registre à écrire ← rd.
ÉcrireReg	Aucun	Ecriture d'un registre par valeur placée sur «Donnée à écrire».
CPSrc	Le CP ← CP + 4.	Le CP ← destination du branchement.
MemversReg	Donnée à écrire dans un registre ← UAL.	Donnée à écrire dans un registre ← Mémoire de données.

- L'unité de contrôle positionne tous les signaux de contrôle en fonction du code-op, sauf CPSrc.
- CPSrc doit être positionnée si l'instruction est un branchement (beq) ➔ décision que peut prendre l'unité de contrôle en fournissant un signal appelons le ***Branchement,*** et si la sortie Zéro de l'UAL est vraie.

CPSrc = *Branchement* . Zéro

CONCEPTION DE L'UNITE CENTRALE DE CONTROLE (4/5)

L'unité de contrôle reçoit en entrée 6 bits du code-op et fournit en sortie 9 bits de contrôle



CONCEPTION DE L'UNITE CENTRALE DE CONTROLE (5/5)

Positionnement des lignes de contrôle en fonction du code-op

Instruction	RegDst	UALSrc	MemversReg	EcrireReg	LireMem	EcrireMem	Branchement	UALOp1	UALOp0
Format R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Déroulement de l'exécution d'une instruction

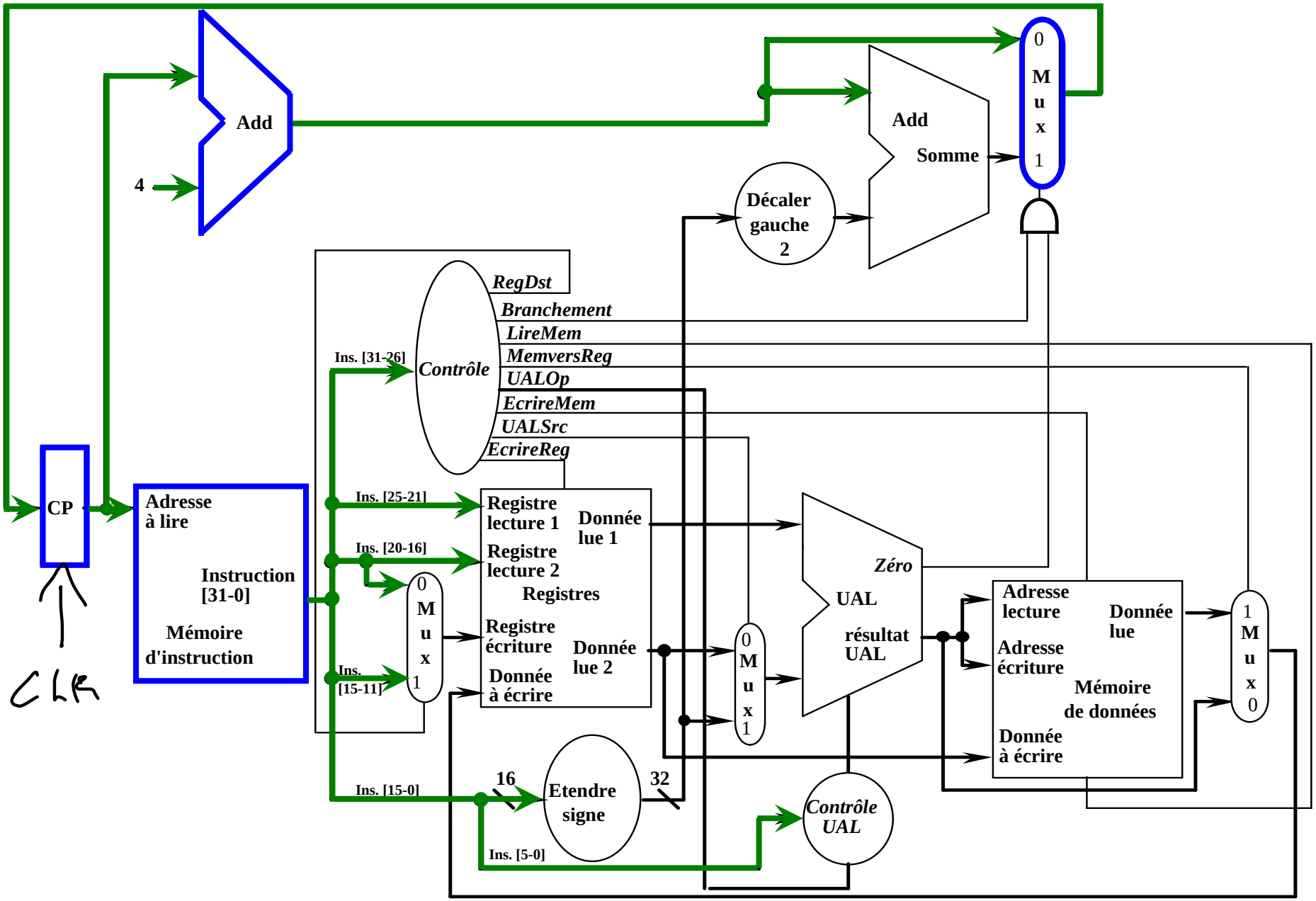
- **Avant de construire la logique de l'unité contrôle, observons comment chaque instruction utilise le chemin de données en mettant en valeur les signaux de contrôle.**

- **Plutôt que de considérer l'exécution comme une seule étape (cycle unique), il est plus simple de voir l'exécution comme une succession d'étapes, en concentrant notre attention sur la partie du chemin de données associée à chaque étape.**

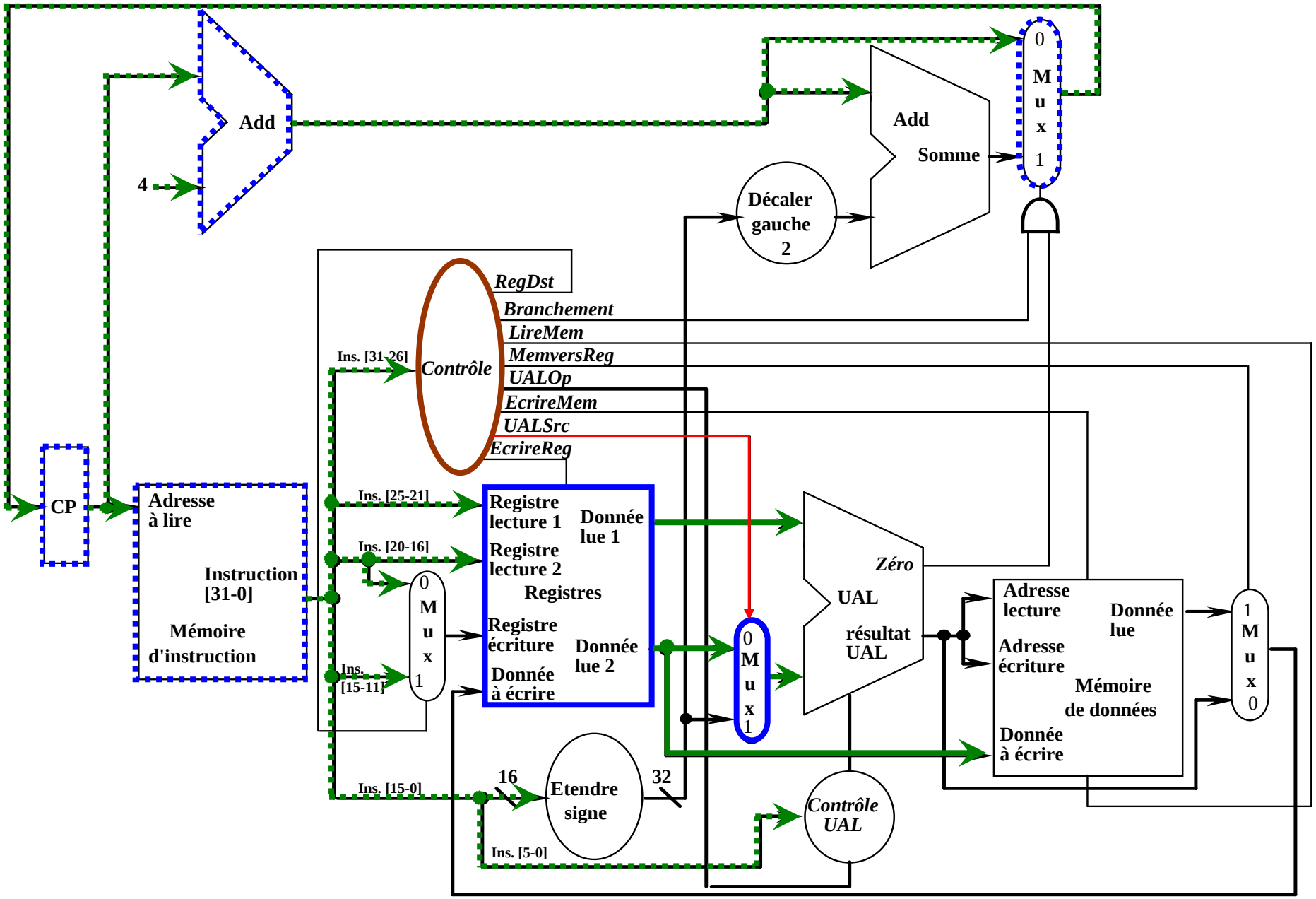
Cas d'une instruction de type R

- ***add Rx, Ry, Rz .***
- **Il semble alors que l'addition s'exécute virtuellement en une succession de 4 étapes**

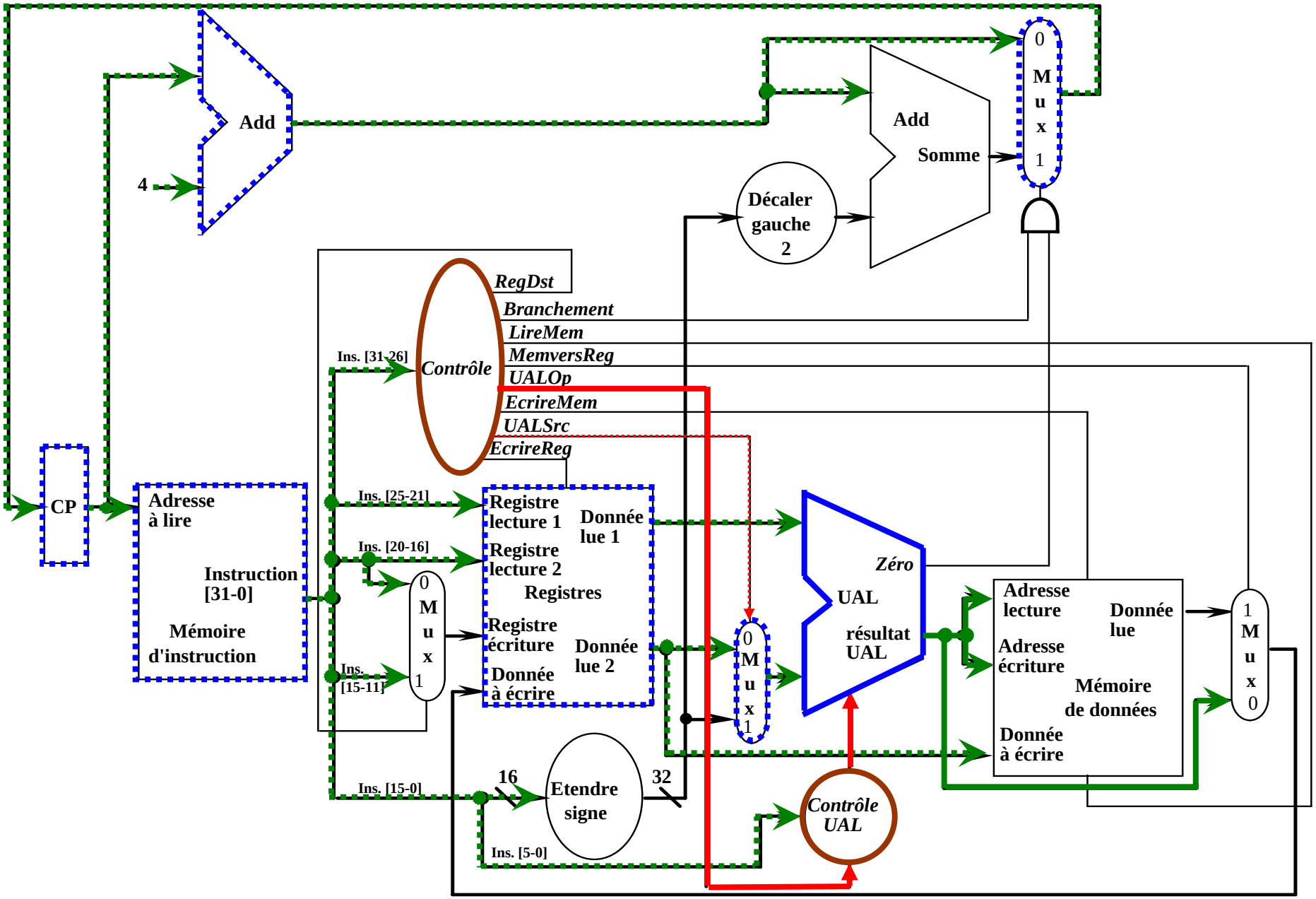
ETAPE 1 : EXTRACTION OU LECTURE DE L'INSTRUCTION



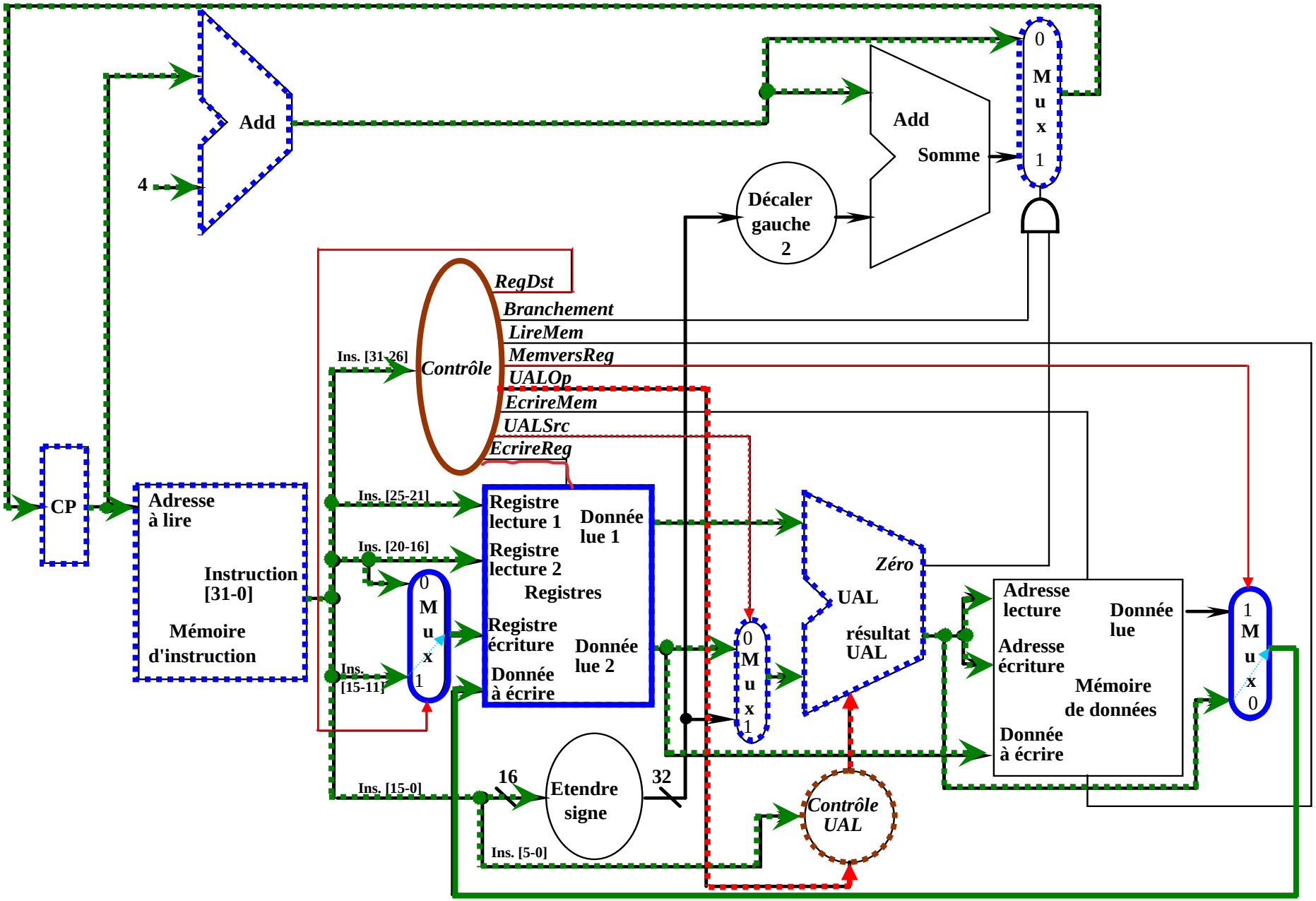
ETAPE 2 : LECTURE REGISTRES ET DECODAGE



ETAPE 3 : EXECUTION (UAL OPERE A PARTIR DU CONTROLE UAL)



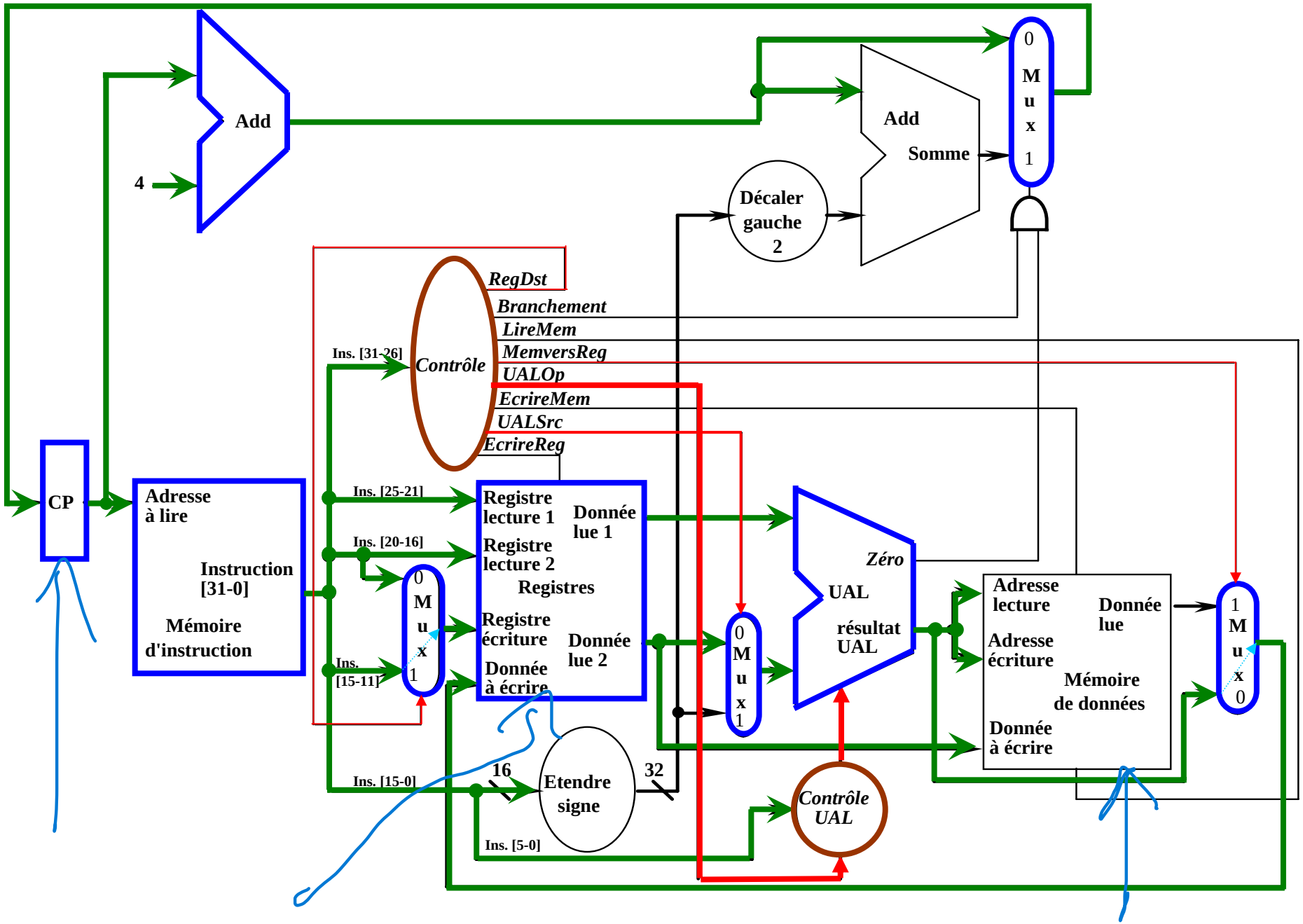
ETAPE 4 : ECRITURE RESULTAT



REMARQUES

- √ **Cette mise en œuvre est combinatoire.**
- √ **Il ne s'agit pas vraiment d'une séquence de 4 étapes distinctes.**
- √ **Le chemin de données opère en réalité en un seul cycle d'horloge.**
- √ **Les signaux du chemin de données peuvent varier de manière imprévisible durant le cycle.**
- √ **Les signaux se stabilisent approximativement dans l'ordre donné par ces étapes car le flot d'informations suit cet ordre.**
- √ **L'étape 4 montre donc non seulement l'action de celle-ci, mais aussi, avant tout, l'opération effectuée par le chemin de données complet lorsque le cycle d'horloge prend effectivement fin.**

CHEMIN DE DONNEES ET DE CONTROLE POUR FORMAT R



DEROULEMENT D'INSTRUCTIONS DE REFERENCE MEMOIRE

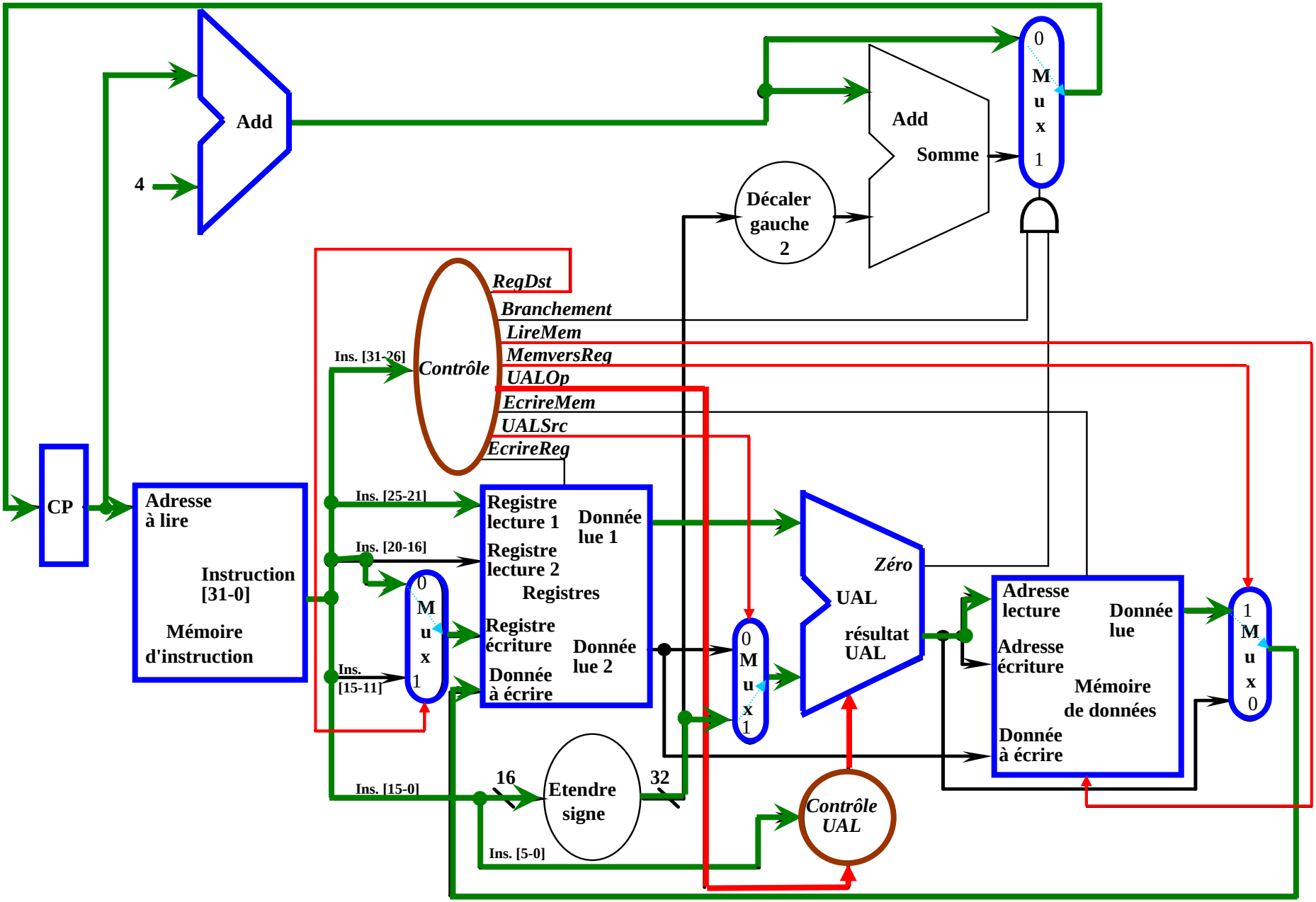
Cas du chargement d'un mot :

lw Rx,d(Ry)

L'instruction de chargement semble opérer en 5 étapes :

1. **Lecture de l'instruction et incrémentation du CP.**
2. **Lecture du registre de base Ry.**
3. **Calcul de l'adresse par L'UAL : Ry + d étendu sur 32 bits.**
4. **Accès mémoire à l'adresse calculée à l'étape 3.**
5. **Ecriture de la donnée lue à l'étape 4 dans le registre Rx (champ rt).**

CHEMIN DE DONNEES ET DE CONTROLE POUR LW



DEROULEMENT D'INSTRUCTIONS DE BRANCHEMENT

Cas du branchement si égal :

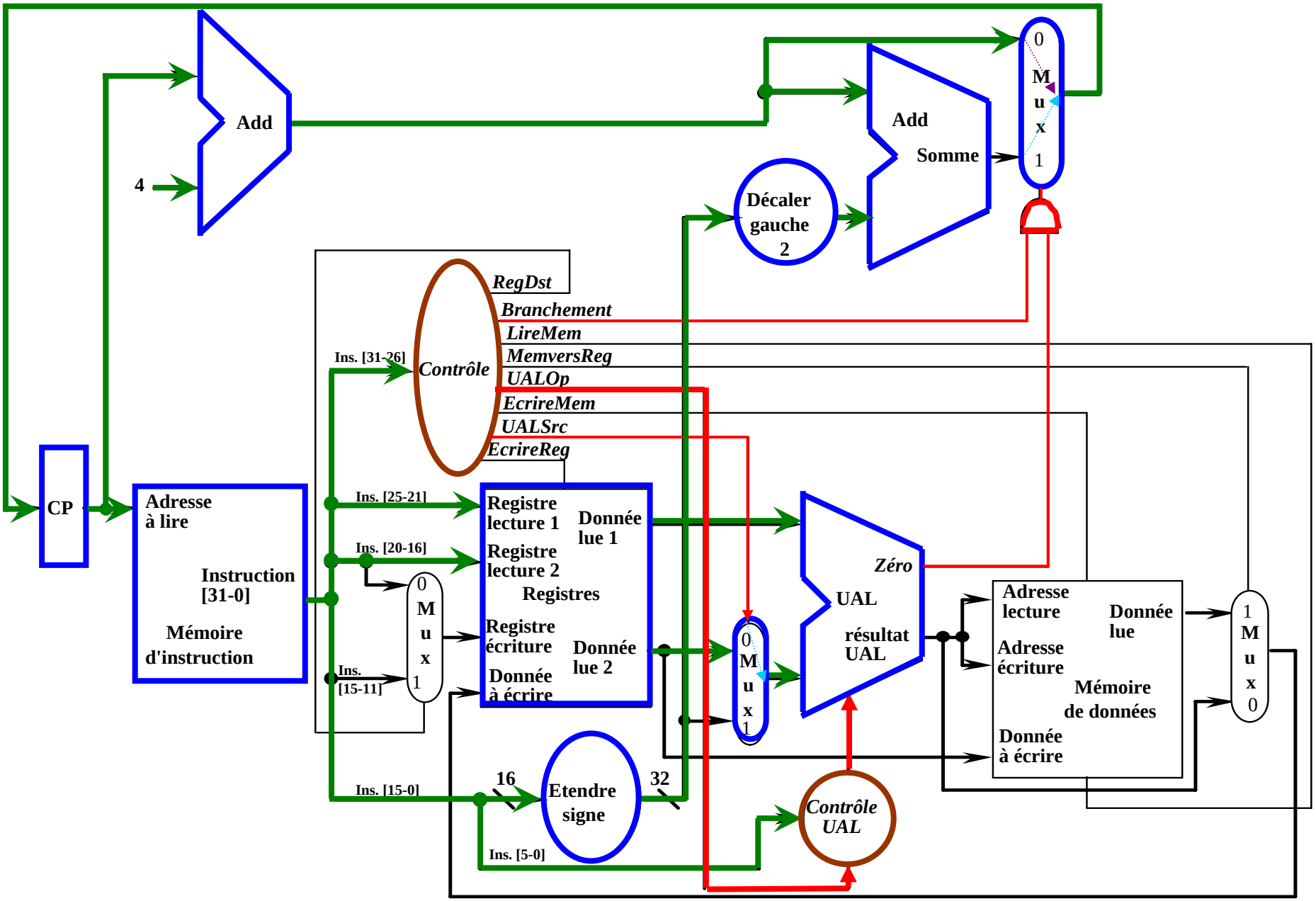
beq Rx,Ry,ETQ.

Similaire à une instruction au format R ➔ mais l'UAL est utilisée positionner les indicateurs d'état afin de déterminer la prochaine valeur de CP : CP+4 ou adresse de destination du branchement.

Le branchement si égal semble opérer en 4 étapes :

1. **Lecture de l'instruction et incrémentation du CP.**
2. **Lecture des registres source Rx et Ry.**
3. **UAL compare Rx et Ry et additionneur calcule l'adresse : (CP+4) + (ETQ étendu sur 32 bits)<<2.**
4. **l'indicateur Z (sortie Zéro de l'UAL) détermine la source de CP.**
   - **Z = 0 ; CP ← adresse de l'instruction suivante, calculée à l'étape 1**
   - **Z = 1 ; CP ← adresse de destination du branchement, calculée à l'étape 3**

CHEMIN DE DONNEES ET DE CONTROLE POUR BEQ



CONSTRUCTION DE L'UNITE CENTRALE DE CONTROLE (1/3)

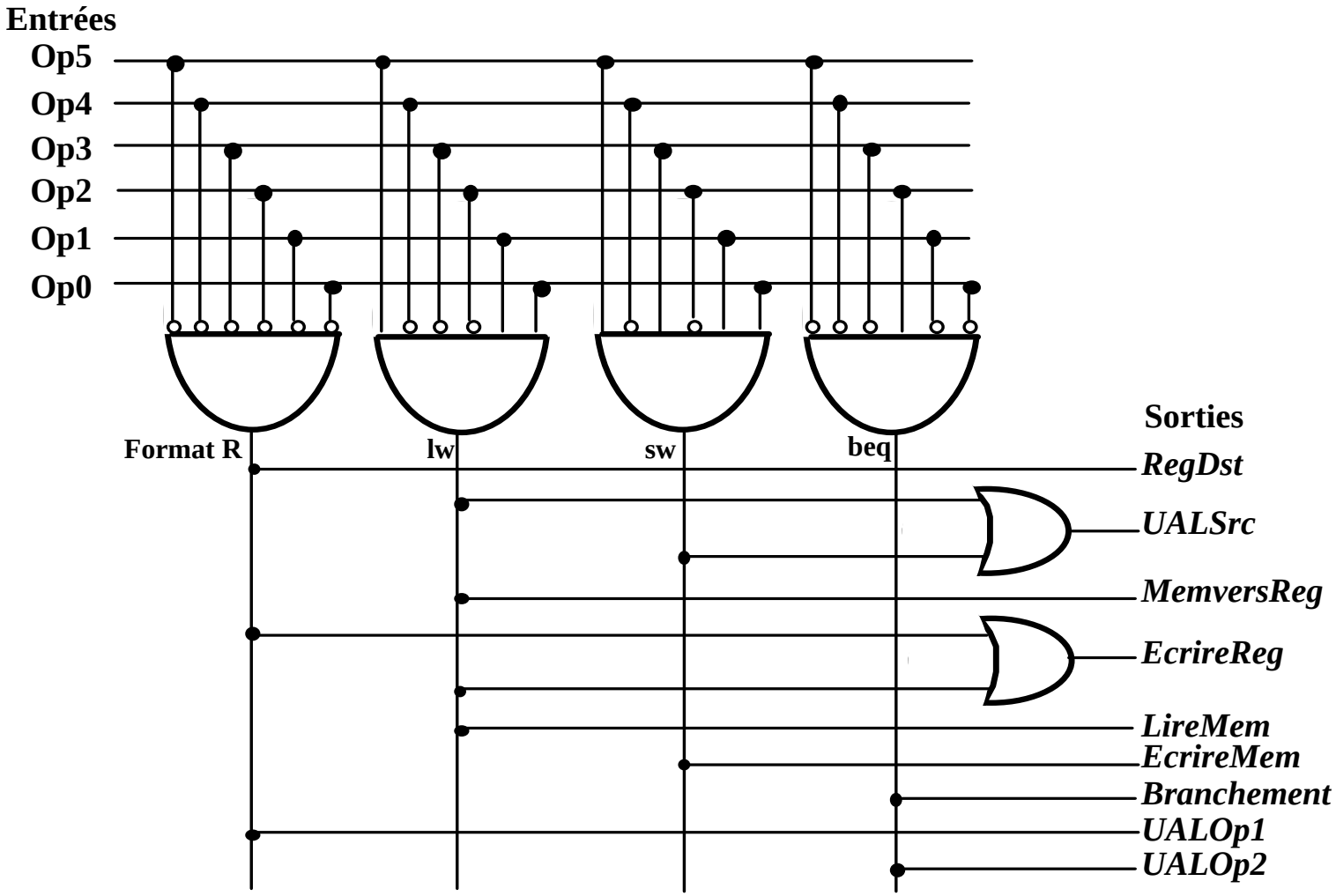
Etat des lignes de contrôle en fonction du code-op, que nous avons appelés Op[5-0].

Nom	Code-op en Hexa.	Code-op en binaire					
		Op5	Op4	Op3	Op2	Op1	Op0
Format R	00	0	0	0	0	0	0
lw	23	1	0	0	0	1	1
sw	2B	1	0	1	0	1	1
beq	04	0	0	0	1	0	0

		Format R	lw	sw	beq
Entrées	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Sorties	RegDst	1	0	X	X
	UALSrc	0	1	1	0
	MemversReg	0	1	X	X
	ÉcrireReg	1	1	0	0
	LireMem	0	1	0	0
	EcrireMem	0	0	1	0
	Branchement	0	0	0	1
	UALOp1	1	0	0	0
	UALOp0	0	0	0	1

CONSTRUCTION DE L'UNITE CENTRALE DE CONTROLE (2/3)

- **Nous pouvons réaliser la logique de l'unité de contrôle directement avec des portes logiques classiques.**
- **Ceci est raisonnable car ici la fonction de contrôle n'est ni trop complexe ni trop grande.**
- **Par contre si la majorité des 64 codes-op étaient utilisées ➔ + de lignes de contrôle, et + de portes logiques chacune pourrait avoir un nombre important d'entrées. ➔ un RLP de ET et de OU aurait été nécessaire.**



CONSTRUCTION DE L'UNITE CENTRALE DE CONTROLE (3/3)

Cas de l'instruction de saut :

- **Etendre la mise en œuvre pour inclure l'instruction de saut.**
- **Décrire comment positionner les nouvelles lignes de contrôle.**

L'instruction de saut similaire à un branchement, mais elle n'est pas conditionnelle.

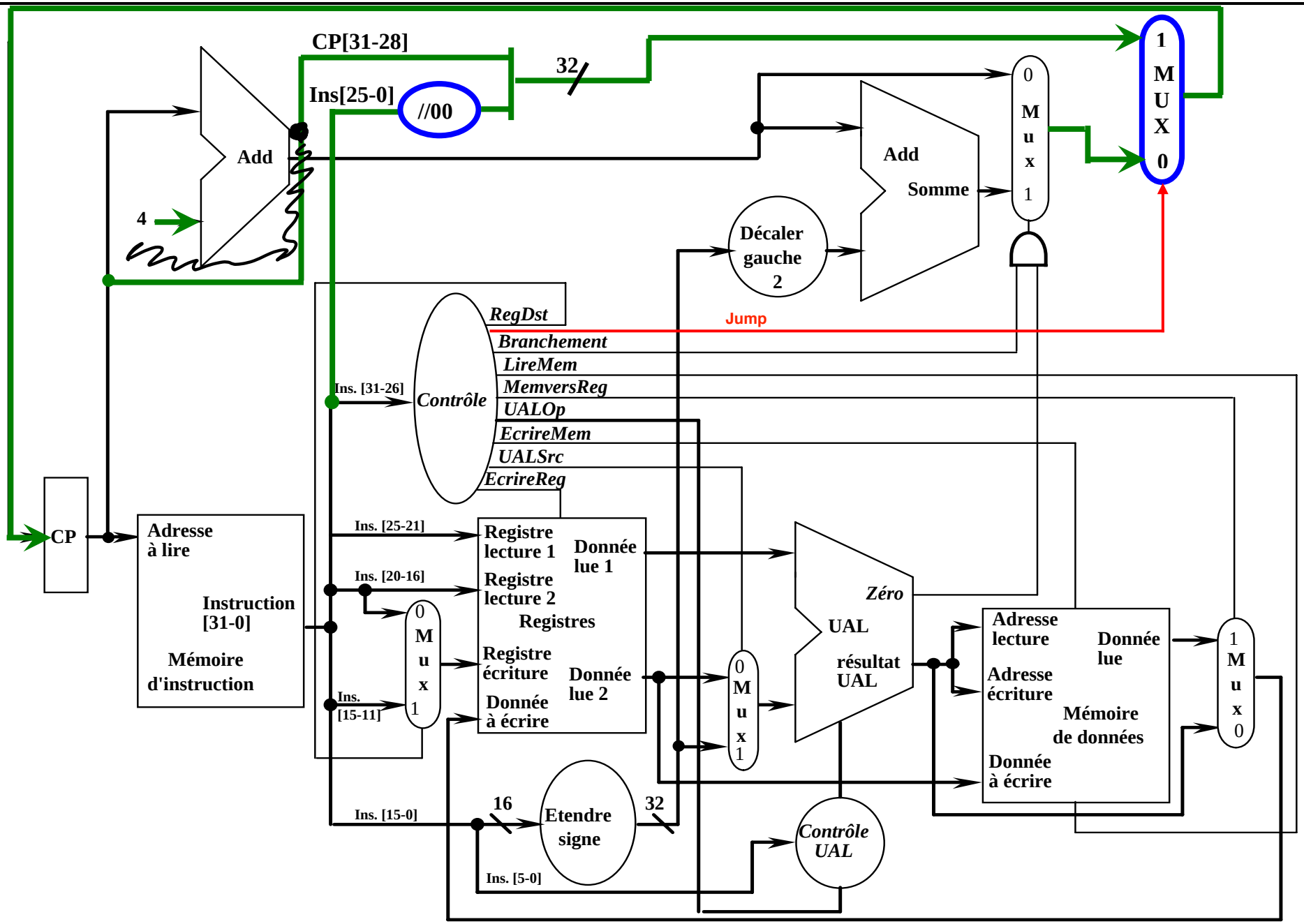
L'adresse de destination de saut est déterminée différemment :

- **Les 2 bits de poids faible de l'adresse sont toujours à 00 (comme pour un branchement).**
- **Les 26 bits suivants de cette adresse proviennent des 26 bits de poids faible l'instruction.**
- **Les 4 bits de poids forts de l'adresse qui restent proviennent du CP courant + 4.**

Le contrôle nécessite un multiplexeur supplémentaire pour sélectionner la source du CP : soit l'adresse (CP+4), soit celle de destination du branchement soit le celle de destination du saut.

Un signal de contrôle est nécessaire pour contrôler le multiplexeur supplémentaire. Ce signal de contrôle, appelé *saut*, n'est activé que lorsque l'instruction est un saut (code-op= 2).

CHEMIN DE DONNEES ET DE CONTROLE COMPLET



EVALUATION DE LA PERFORMANCE DANS LE CAS D'UNE MISE EN ŒUVRE A CYCLE UNIQUE (1/4)

- **Dans le modèle à cycle unique le cycle d'horloge est le même pour toutes les instructions ➔ CPI = 1.**

- **Le cycle d'horloge est défini par le chemin le plus long dans la machine ➔ Le chargement (5 unités fonctionnelles).**

- **Bien que CPI = 1, la mise en œuvre à cycle unique n'est pas performante**

Exemple : **supposons que le temps d'opération pour les principales unités fonctionnelles vaut :**

- **Unités mémoire : 10 ns,**
- **UAL et additionneurs : 10 ns,**
- **Banc de registres (lecture ou écriture) : 5 ns.**

MUX, unité de contrôle, unité d'extension signe, accès au CP et les conducteurs ne génèrent pas de délais

Quelle mise en œuvre parmi les suivantes sera la plus rapide et de combien ?

1. Une mise en œuvre où chaque instruction opère en un cycle d'horloge de longueur constante.

2. Une mise en œuvre où chaque instruction s'exécute en un cycle d'horloge de longueur variable,

EVALUATION DE LA PERFORMANCE DANS LE CAS D'UNE MISE EN ŒUVRE A CYCLE UNIQUE (2/4)

Utilisons la répartition d'instructions d'un benchmark (gcc) pour établir les performances de ces alternatives.

Noyau R3000	Mnémonique	gcc
Addition	add	0%
Addition immédiat	addi	0%
Addition non signé	addu	8%
Addition imm. non signé	addiu	16%
Soustraction	sub	0%
Soustraction non signé	subu	1%
ET	and	2%
ET immédiat	andi	2%
OU	or	2%
OU immédiat	ori	0%
Décalage logique à gauche	sll	8%
Décalage logique à droite	srl	2%
Chargement poids fort imm.	lui	2%
Chargement mot	lw	22%
Rangement mot	sw	11%
Branchement si égal	beq	8%
Branchement si différent	bne	8%
Saut	j	0%
Saut avec lien	jal	1%
saut par registre	jr	1%
Positionner si inférieur	slt	3%
Positionner si < immédiat	slti	1%
Positionner si < non signé	sltu	1%
Position. si < imm. non signé	sltiu	1%
Total		100%

EVALUATION DE LA PERFORMANCE DANS LE CAS D'UNE MISE EN ŒUVRE A CYCLE UNIQUE (3/4)

On va Comparer les temps d'exécution UC pour le benchmark dans les deux mises en œuvre

Temps d'exécution UC=Nombre d'instructions x CPI x Temps de cycle. Or CPI = 1 (cas d'un cycle unique)
Temps d'exécution UC = Nombre d'instructions x Temps de cycle.

Il faut calculer le temps de cycle d'horloge pour les deux mises en œuvre.

- **Chemin critique pour les différents types d'instruction, est le suivant :**

Type d'instruction	Unités fonctionnelles utilisées par le type d'instruction				
Format R	Extraction instruc.	Accès à un registre	UAL	Accès à un registre	
Chargt. mot	Extraction instruc.	Accès à un registre	UAL	Accès à la mémoire	Accès à un registre
Rangt. mot	Extraction instruc.	Accès à un registre	UAL	Accès à la mémoire	
Branchemt.	Extraction instruc.	Accès à un registre	UAL		
Saut	Extraction instruc.				

√ **A partir de ces chemins critiques, on obtient la longueur requise par chaque type instruction**

Type d'instruction	Mémoire d'instructions	Registre lecture	Opération UAL	Mémoire de données	Registre d'écriture	Total
Format R	10	5	10	0	5	30 ns
Chargt. 1 mot	10	5	10	10	5	40 ns
Rangt. 1 mot	10	5	10	10		35 ns
Branchement	10	5	10	0		25 ns
Saut	10					10 ns

Le cycle d'horloge pour une machine ayant un cycle unique pour toutes les instructions est de 40 ns.
Une machine avec un cycle d'horloge variable aura un cycle compris entre 10 ns et 40 ns.

EVALUATION DE LA PERFORMANCE DANS LE CAS D'UNE MISE EN ŒUVRE A CYCLE UNIQUE (4/4)

- **Distribution des fréquences d'instructions : 22% chargements, 11% rangements, 49% format R, 16% branchements, et 2% sauts.**

- **Le temps moyen d'exécution avec une horloge variable est de : 40 x 22% + 35 x 11% + 30 x 49% + 25 x 16% + 10 x 2% = 31,6 ns**

Le temps de cycle de la mise en œuvre à horloge variable est plus court ➔ plus rapide.

Déterminons le rapport des performances :

$$\frac{\text{Performances UC}_{\text{cycle variable}}}{\text{Performances UC}_{\text{cycle unique}}} = \frac{\text{Temps exécution UC}_{\text{cycle unique}}}{\text{Temps exécution UC}_{\text{cycle variable}}} = \frac{\text{NI x Cycle horloge UC}_{\text{cycle unique}}}{\text{NI x Cycle horloge UC}_{\text{cycle variable}}}$$

$$\frac{\text{Performances UC}_{\text{cycle variable}}}{\text{Performances UC}_{\text{cycle unique}}} = \frac{\text{Cycle horloge UC}_{\text{cycle unique}}}{\text{Cycle horloge UC}_{\text{cycle variable}}} = \frac{40}{31,6} = 1,27$$

La mise en œuvre à horloge variable serait 1,27 fois plus rapide.

Problème : **la mise en œuvre d'une horloge à durée variable pour chaque instruction est extrêmement difficile et le surcoût engendré par une telle approche peut être plus important que le gain éventuel qu'elle apporte.**

Solution **: l'amélioration de performance d'une mise en œuvre à cycle unique peut être obtenue de 2 manières :**

- √ **Une mise en œuvre d'un chemin de données et de contrôle avec un cycle constant mais plus court ;**
- √ **Une mise en œuvre d'un chemin de données et de contrôle pipeliné.**