Chapitre I : Architecture d'un processeur : orientations et motivations

1. Introduction

La conception d'un processeur a été, depuis longtemps, considérée comme une tâche complexe. Les concepteurs doivent concevoir des processeurs, d'un côté pour satisfaire des besoins précis : personnel, industrie, science, etc… et de l'autre côté, atteindre les performances désirées avec des coûts acceptables.
Dans ce cas, le concepteur doit, avant tout, déterminer quels sont les critères et les contraintes qui s'imposent pour cette conception. Citons par exemple:

- Le jeu d'instructions
- L'organisation fonctionnelle
- Les contraintes de consommation.
- Les contraintes de surface
- La réalisation proprement dite.
- Etc……

2. Notion d'architecture avancée : Motivations

L'architecture de l'ordinateur ne cesse d'évoluer afin de satisfaire les besoins imposés par l'accroissement des domaines d'applications. En effet, lorsque l'ordinateur a été introduit dans les années quarante, l'architecture du processeur était relativement simple (architecture de Von Newman): petite mémoire, quelques registres et quelques fonctionnalités.
La programmation était une tâche fastidieuse et l'ordinateur était dépourvu de système d'exploitation. Plus tard, des langages évolués et autres formes d'abstraction ont été développées, ceci a engendrer de nouvelles visions comportementales du système.
La conception et la mise en place de nouvelles plates-formes architecturales capables de supporter de telles visions s'avère nécessaire.

2.1. Support langages évolués

L'architecture logicielle (la couche ISA) est la spécification d'un processeur, c'est la machine virtuelle que voit un compilateur, ou un programmeur en langage machine. Elle décrit les unités fonctionnelles (UAL, FPU,…), les registres visibles, les types de données manipulés et les opérations que le processeur effectue sur ces données (jeu d'instruction).
Le but de l'architecture logicielle d'un processeur est de permettre une exécution efficace des constructions des langages de haut niveau (LHN). Ces constructions ont une sémantique riche (par exemple : boucle, appel de procédure, passage de paramètres, objet,…) par rapport au langage machine.
Les architecture à concevoir ne doivent pas seulement supporter les concepts de base (structure répétitive, structure de test, branchement, objet de type simple), mais supporter aussi les capacités complexes de langage, à savoir les procédures, les modules et les structure de données complexes (tableaux, record,…..), et supporter les nouveaux concepts, issus des différentes formes d'abstraction, a savoir les objets, les TAD.
La prise en compte de l'ensemble des contraintes liées aux LHN a des conséquences majeures à tous les niveaux de l'architecture (matérielle ou logicielle)
Sur ce plan, toute un paradigme (type) de langage fut développé, entre autre :

- Les langages impératifs (procéduraux)
- Les langages applicatifs (fonctionnels)
- Les langages logiques
- Etc…..

2.2. Support système d'exploitation

L'évolution des systèmes d'exploitation a été tout le temps liée à l'évolution de l'architecture de la machine. En effet, le rôle principal d'un système d'exploitation est de cacher la complexité physique de la machine à ses utilisateurs, toute en assurant une gestion efficace des ressources de celle-ci.
La gestion des ressources dépend beaucoup de la nature du système : mono-utilisateur ou multi-utilisateur, centralisé ou réparti, mono-tâche ou multitâche, STR ou autres, et des facilités offertes par l'architecture elle-même.
Parmi les contraintes systèmes prises en considération lors de la conception des architectures, on peut citer :

- La gestion de la mémoire (segmentation, pagination et MV).
- La protection (les niveaux de privilèges, les droits d'accès,….).
- La gestion des exceptions et des interruptions.
- La gestion de processus.
- La gestion de cache.
- La gestion des E/S.
- ….

2.3. Amélioration de performances

La mesure des performances des processeurs n'est pas simple. Les performances d'une machine peuvent s'exprimer de différentes manières. Pour l'utilisateur final, le temps d'exécution d'un programme, de son début à sa fin, comprend le temps UC, les accès mémoires, les entrées-sorties et le temps utilisé pour les besoins du système d'exploitation. Il dépend des performances de l'ensemble des composantes matérielles et logicielles du système.
Il est cependant nécessaire de distinguer des performances globales de l'ordinateurs, qui tiennent compte des unités fonctionnelles (UC, mémoires, entrées-sorties), des performances de l'unité centrale.

3. Conception d'un processeur

La conception d'un processeur est une tâche complexe et nécessite une démarche méthodique. Généralement, on utilise les outils de synthèse de haut niveau. Ces outils permettent, à partir d'une description comportementale, de générer l'architecture proprement dite. Cette architecture est composée de deux parties : Partie contrôle (PC) et Partie opérative (PO) (fig. 1). Les deux parties communiquent par le biais de commandes et comptes rendus, et fonctionnent sur la même horloge.
Les commandes, générées par la partie contrôle, sont des entrées pour la partie opérative, elles précisent l'opération à réaliser dans le cycle.
Les comptes rendues, issus de la partie opérative, sont des entrées pour la partie contrôle qui influencent le séquencement des commandes.

3.1. Partie opérative

La partie opérative, très souvent appelée chemin de données (Data Path), correspond à un ensemble blocs fonctionnelles interconnectés de manière à permettre l'exécution des opérations demandées par la partie contrôle.
Parmi les blocs composant la partie opérative (combinatoire/séquentiels), on peut trouver principalement :

- Les unités fonctionnelles pouvant chacune exécuter ou plusieurs opérations : ALU, FPU, MMU, Décaleur, Comparateur,…..
- Les unités de mémorisation : registres (CO, RI, ACC, SP, MDR, MAR, PSW,..), cache,…
- Les unités de connexion : bus, multiplexeur, porte à trois états, …

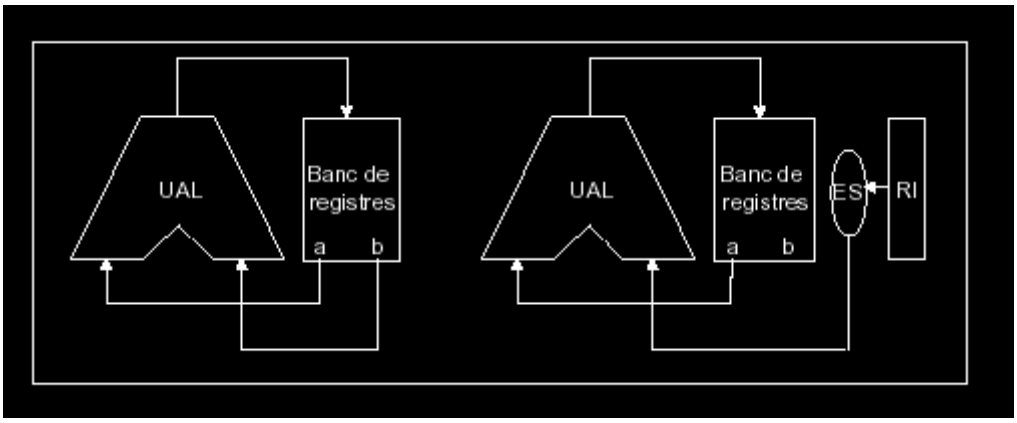

Figure 1 **:** Data Path pour l'exécution des instructions arithmétiques et logiques

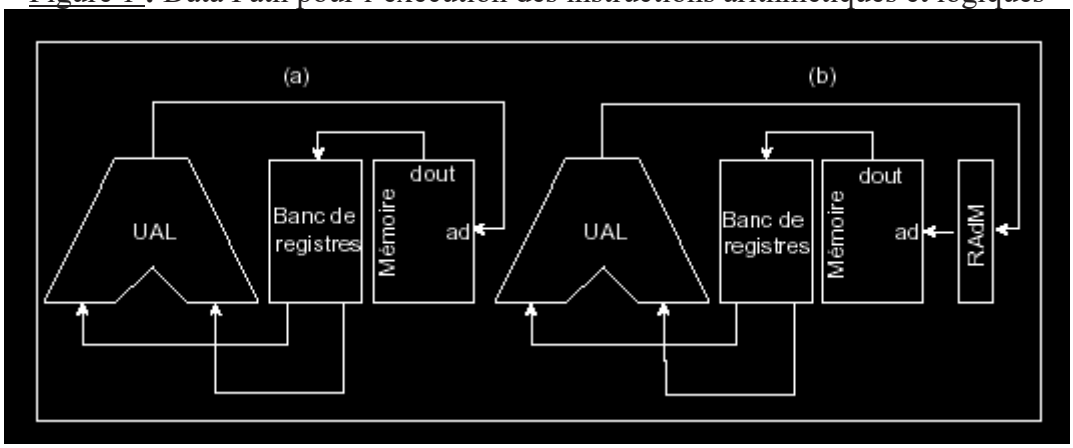

Figure 2 **:** Data Path pour le LOAD Reg-Reg (sans (a) ou avec (b) registre d'adresse)

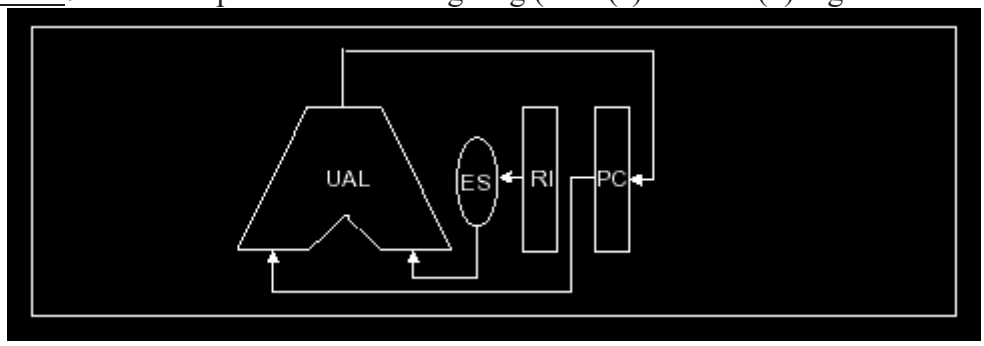

Figure 3 **:** Data Path pour les branchements

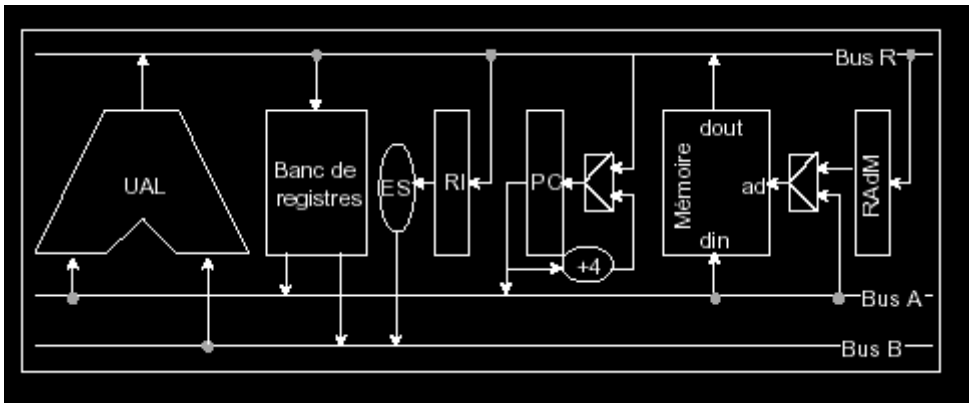

Figure 4 **:** Architecture (Data Path) à 3 bus

3.2. Partie contrôle (unité de commande)

C'est la partie intelligente qui permet de commander chacune des organes du chemin de données pour exécuter les opérations. Cette partie contrôle est un système séquentiel et dont le fonctionnement peut être modélisé par une machine d'états finis (automate).
En particulier, les signaux générés par cette partie sont :

- Les signaux de contrôle de transfert : signaux Ti de contrôle de transfert des registres vers les bus.
- Les signaux d'écriture Wi dans les registres.
- Les signaux de contrôle des opérations effectuées par les différentes opérateurs : commandes de l'UAL, commande R/W de la mémoire, commande des multiplexeurs,…

Pour exécuter les instructions, la partie contrôle doit pouvoir générer les signaux de contrôle (addition, transfert, lecture mémoire,…) selon une séquence convenable. L'implantation de la partie contrôle peut être câblée ou microprogrammée. Chaque technique a ses avantages et ses inconvénients.

Exemples de contrôle
Cycle de recherche :
Supposons que chaque instruction occupe un mot mémoire. Alors les trois étapes suivantes sont nécessaires pour que le CPU exécute une instruction.

1. MAR ← [CO]
2. Lecture mémoire ; CO ← CO + 1
3. Attente du signal MFC
4. RI ← [MDR] ;

Rangement dans un mot mémoire :

1. MAR ← @
2. MDR ← [R1]
3. Ecriture mémoire
4. Attente MFC

3.2.1. La Microarchitecture

L'architecture au-dessous de la couche machine ISA est appelée microarchitecture (micromachine ou la couche microarchitecture). Le langage correspondant est appelé langage de microprogrammation et les instructions de ce langage sont appelées micro-insturctions.

- **Notion de micro-opérations (MOs)**

Une MO est une opération primitive et indivisible de la micro-architecture. Chaque MO représente une affectation d'une expression à une ressource de mémorisation de type registre ou mémoire.
Chaque MO est validée par un ensemble de signaux de contrôle (microcommandes) issus de l'unité de commande.

- **Exécution des MOs**
  - Séquentielle
  - Parallèle.

Condition pour l'exécution de deux MOs (MOi & MOj) en parallèle : condition de Bernstein :
$[Ei \cap Sj = \phi]$ et $[Ej \cap Si = \phi]$ et $[Si \cap Sj = \phi]$ et $[Oi \cap Oj = \phi]$ avec :
E : ensemble des entrées de la MO
S : ensemble des sorties de la MO
O : ensemble des opérateurs de MO
Si les deux Mos ne satisfont pas la condition précédente on parlera d'exclusion mutuelle entre MOi et MOj.

- Temps d'exécution d'une MO : on considère pour simplifier que toutes les Mos ont le même temps d'exécution.
- Rôle de l'unité de commande : ordonner l'exécution des MOs.
- L'utilisation du « ; » indique que les MOs s'exécutent en parallèle.

Exemple1 : réalisation d'un opération arithmétique ou logique (ADD R1 R2 R3 : R3 ← R1 + R2)

1. $R1_{OUT}$; X_{IN}
2. $R2_{OUT}$;
3. Add ; Y_{IN}
4. Y_{OUT} ; $R3_{IN}$
5. END

Exemple2 : réalisation d'une opération de chargement (LOAD dir R1 50)

1. RI_{OUT} ; MAR_{IN}
2. Lecture
3. Attente MFC
4. MDR_{OUT} ; $R1_{IN}$
5. END

Remarque : ($R1_{OUT}$, X_{IN}), $R2_{OUT}$, Add , Y_{IN} , (Y_{OUT} , $R3_{IN}$), END sont des MOs.

Exemple3 : réalisation du cycle de recherche

1. CO_{OUT} ; MAR_{IN} ; RAZ X ; $R_0 \leftarrow 1$
2. READ ; ADD ; Y_{IN}
3. Y_{OUT} ; CO_{IN} ; Attente MFC
4. MDR_{OUT} ; RI_{IN}

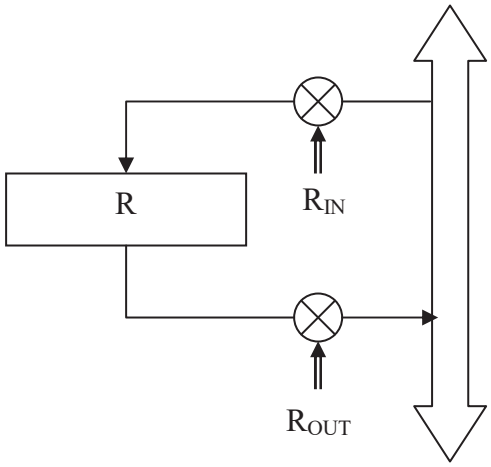

Figure 5 **:** Barrière de contrôle d'E/S des registres

3.2.2. Contrôle câblé

C'est la technique utilisée dans quelques architectures avancées et plus particulièrement dans les architectures RISC. Le séquencement des commandes est assurée par des composants hardware, telles que les bascules, les PLA,…

- Dans ce type de contrôle, chaque étape de contrôle (de l'instruction) nécessite un cycle horloge (période de temps) pour une exécution correcte (sauf le MFC de la mémoire).
- Chaque période de temps doit être assez longue pour permettre de terminer les fonctions spécifiées à l'étape correspondante.
- Si on suppose que toutes les périodes de temps aient une durée égale. Donc le contrôleur nécessaire peut être réalisé à partir d'un compteur commandé par une horloge.
- Chaque état de ce compteur correspond à l'un des étapes de contrôle.
- Les signaux de contrôle nécessaires sont uniquement déterminés par les informations suivantes :
  - le contenu de compteur de contrôle
  - le contenu du registre d'instruction RI (COP , mode d'adressage , Registres ,…).
  - Le contenu du code condition et des drapeaux d'état (PSW tel que CF, OF et MFC).

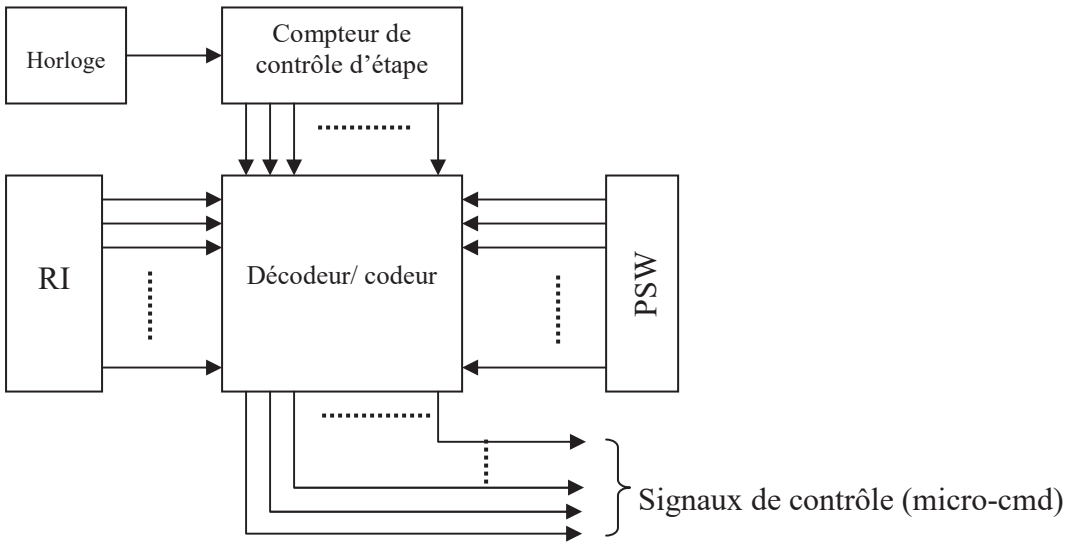

Figure 6 **:** Organisation de l'unité de commande câblée

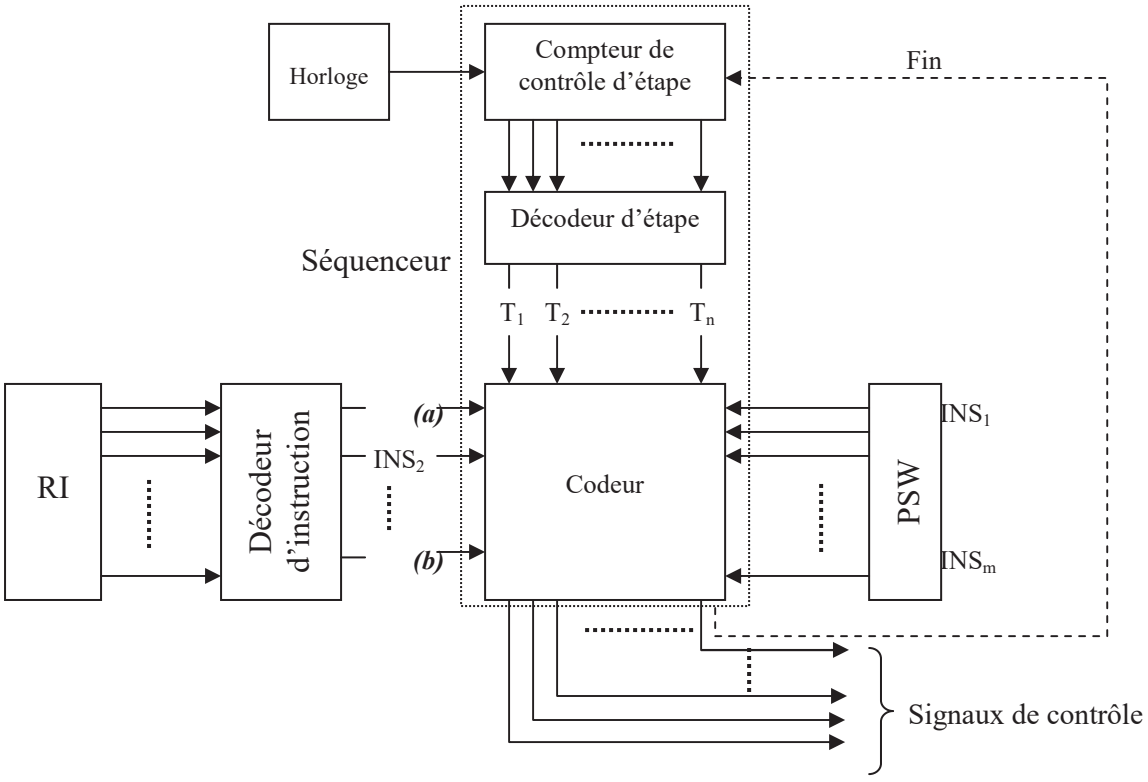

Figure 7 **:** Organisation de l'unité de commande câblée (codeur séparé)

- Le bloque décodeur/codeur est un circuit combinatoire qui gère les sorties de contrôle désirées, à partir de toutes ses entrées.
- Le décodeur d'étapes fournit une ligne indépendante pour chaque état, en période de temps, de la séquence de contrôle.

- La sortie du décodeur d'instructions se compose d'une ligne indépendante pour chaque instruction machine. Une ligne de sortie (INS_1, INS_2 , …, INS_m) est mise à 1 et les autres sont mises à 0.
- Tous les signaux d'entrée du codeur doivent être combinés pour générer chaque signal de contrôle : X_{IN}, CO_{OUT}, ADD, END,….
- La structure de codeur est réalisée par des fonctions logiques de type :
  $Y_{IN} = T_1 + T_6.ADD + T_5. BR + ….$
- Cas général, chaque MO générée par l'unité de commande est une fonction logique de type :
  $S = f[(T_1,T_2,..T_n), (INS_1,INS_2,….INS_m), H, OF, CF, IF, MFC, ….]$

3.2.3. Contrôle microprogrammé

- La microprogrammation consiste à remplacer le séquenceur câblé par un séquenceur programmé.
- En effet, une instruction en langage machine, après décodage, donne lieu à un microprogramme composée d'une suite de micro-instructions. Une micro-instruction correspond à un ensemble de commandes (actions) faisant fonctionner une fois la partie opérative.
- Une micro-instruction est une représentation codée d'un ou plusieurs micro-opérations faisant fonctionner une fois (pendant une phase ou un cycle) la partie opérative.
- Le microprogramme peut être stocké dans une mémoire (ROM, EPROM) appelée mémoire de contrôle ou mémoire de microprogrammes.
- Il faut ajouter des mécanismes pour l'exécution séquentielle des micro-instructions, toute en permettant des branchements conditionnels et inconditionnels : Un micro-compteur ordinal (MPC) et un micro-registre d'instruction (MIR).
- Le format de la micro-instruction peut ressembler à celui utilisée dans les langages machines, c'est à dire un code opération et des opérandes, dans ce cas on parle de la microprogrammation verticale, par opposé à la microprogrammation horizontale.

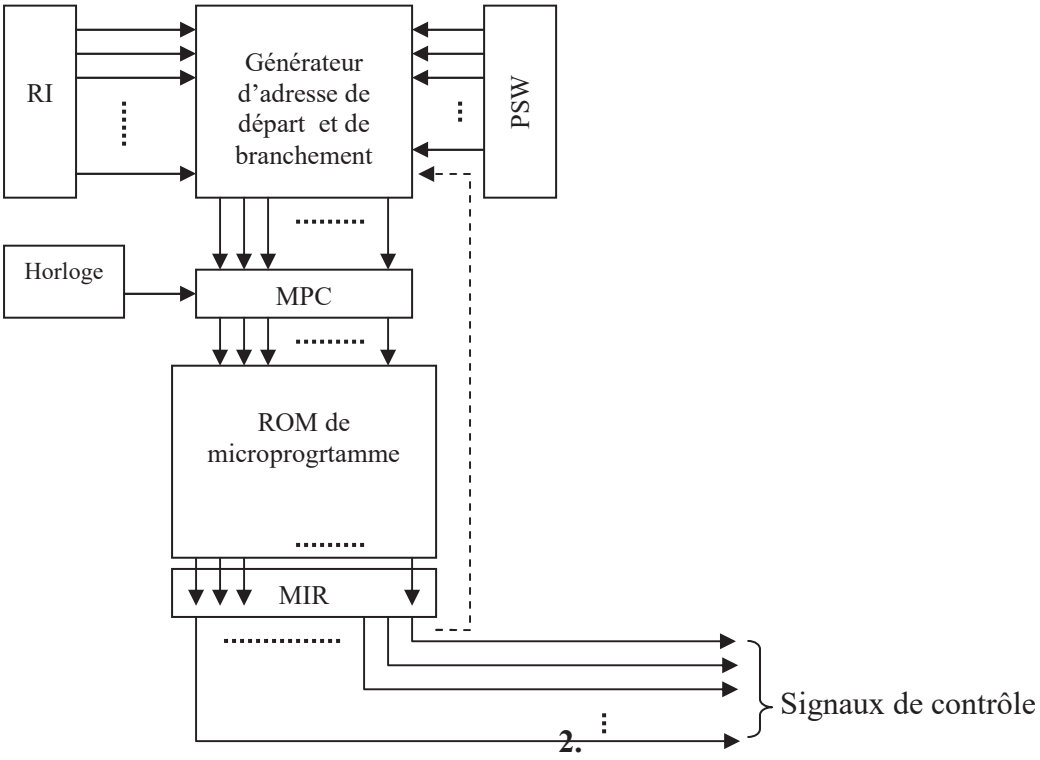

Figure 8 **:** Organisation de l'unité de commande microprogrammée

- **Format des micro-instructions**

COP	Adr	Cond	Add	Sub	And	RD	WR	MAR_{IN}	MAR_{OUT}	$R1_{IN}$	$R1_{OUT}$	IR_{IN}	IR_{OUT}	MDR_{IN}	MDR_{OUT}

Une micro-instruction contient un certain nombre de champs, chacun contrôlant un circuit de la partie opérative. Par exemple :

- Les bits : Add, Sub, And,… sont spécifiques à l'UAL.
- Les bits RD, WR sont spécifiques à la mémoire.
- Les autres bits sont spécifiques aux commandes d'E/S des différents registres.

Cependant, l'incrémentation ou la modification (branchement) du MPC est généralement incluse explicitement dans la micro-instruction. Ceci signifie qu'une section de la micro-instruction appelée Séquenceur est dédiée à cette tâche. Sa fonction est de déterminer l'adresse de prochaine micro-instruction à exécuter.

Le séquenceur est divisé en trois champs :

- Un champ COP qui contient le code opération d'un micro-saut conditionnel.
- Un champ Cond qui représente la condition.
- Un champ Adr qui exprime l'adresse de la mémoire de contrôle où le branchement se fait si la condition est vérifiée

- **La microprogrammation horizontale (contrôle direct) « modèle de Wilkes »**

- Le principe de la microprogrammation horizontale, consiste à coder directement les commandes (MOs) de la micro-instruction (un bit pour chaque MO). Dans ce cas le codage de la micro-instruction nécessite plus de bits.
  - Avantage : rapidité d'exécution.
  - Inconvénient : micro-instructions très longues (plus de 100 bits / MI).

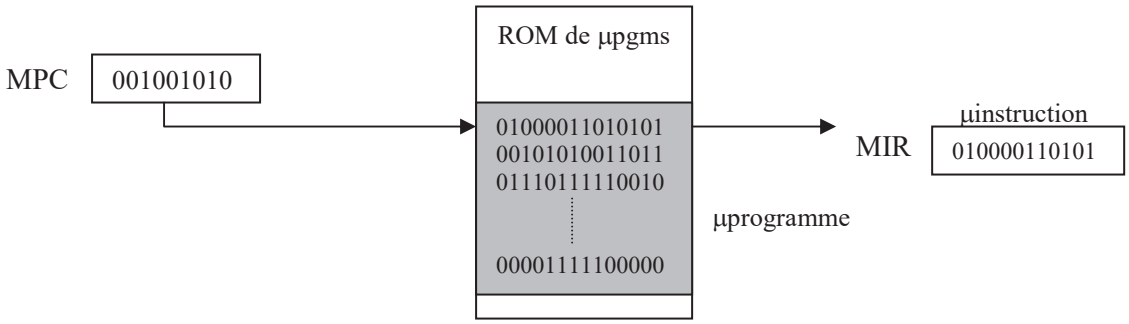

Figure 9 **:** microprogrammation horizontale

- **La microprogrammation verticale (contrôle par champs codés)**

- La méthode verticale consiste à coder les micro-instructions par des champs codés.
- Permet un gain considérable de mémoire (micro-instructions plus courtes).
- Inconvénients : ralentissement d'exécution dû à la phase de décodage des champs.

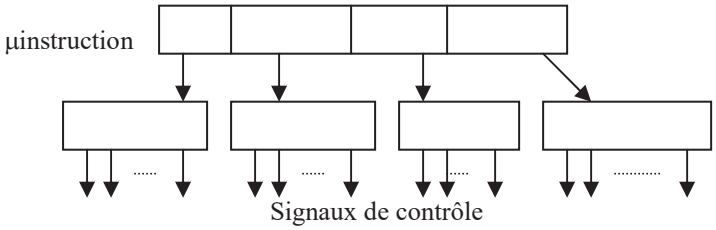

Figure 10 **:** microprogrammation verticale

3.2.3.1 Techniques de décodage des instructions

- Le décodage dans le cas de la micro-programmation consiste à faire une correspondance entre le code opération d'une insturction (couche ISA) et l'adresse de début du micro-programme associé dans la mémoire de contrôle.
- Il existe plusieurs méthodes pour cette correspondance.

• **Arbre décodage**

- Cette méthode consiste à faire l'examen des bits du COP et parcourir un arbre binaire pour trouver l'adresse de début du microprogramme correspondant.
- Inconvénient : nécessité d'autant de comparaisons pour chaque bit du COP.

• **Décodage par table de jump**

- Cette méthode consiste à adopter une table de saut :

```
DECODE : ADR := JTAB + RI
         JMP ADR
JTAB :   JMP OPCODE1
         JMP OPCODE2
         ……
         JMP OPCODEn
```

- Dans ce cas, trois micro-instructions sont exécutées : la première additionnant le code opération à l'adresse de début de la table, le saut à l'adresse obtenue et la localisation de l'instruction de saut correspondante.
- Inconvénient : nécessité d'utilisation d'une référence indirecte pour contenir l'adresse calculée.

• **Décodage par blocs**

- Cette méthode consiste à envoyer le COP sur les bits de poids fort de l'adresse de la mémoire de contrôle et d'envoyer des 0 sur les autres bits.

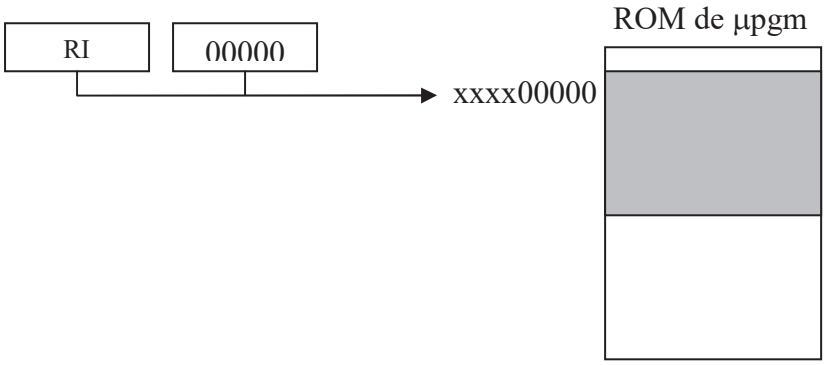

Figure 11 **:** décodage par bloc

- Inconvénient : méthode assez rigide : arrangement obligatoire des microprogrammes aux adresses mémoire équidistantes, ce qui cause un gaspillage considérable de l'espace mémoire

• **Décodage par mappage**

- C'est la technique la plus utilisée.
- Consiste à utiliser une mémoire de mapping (correspondance); le COP est envoyé sur les lignes d'adresse de cette mémoire qui remplace le décodeur câblé. On obtient sur les lignes de données l'adresse de début du microprogramme correspondant.

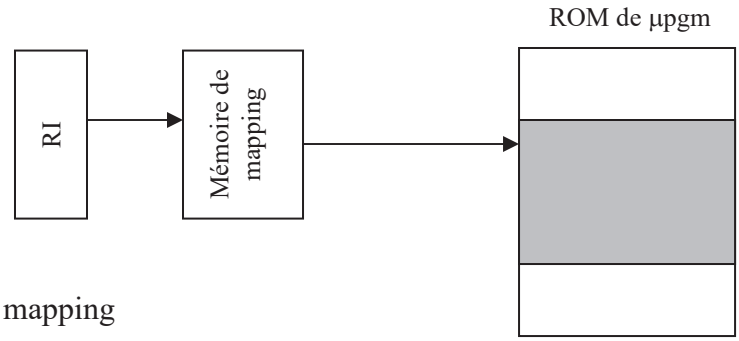

Figure 12 **:** mémoire de mapping