Architecture Pipeline

1. Traitement anticipé des instructions : Pipeline

Le principe du pipeline est connu depuis longtemps en informatique, puisqu'il a été utilisé sur les premiers ordinateurs scientifiques. C'est un concept introduit dans le but d'augmenter la vitesse de traitement d'un micro-processeur (RISC).
Le pipeline est l'équivalent informatique du travail en chaîne. L'instruction à effectuer demande un certain nombre d'étapes élémentaires, et chaque unité fonctionnelle travaille à partir du résultat d'une autre. Ainsi la traversée complète de la chaîne pour un produit nécessite de passer par toute les unités en succession, e ceci représente la latence du pipeline. En revanche, à chaque nouvel instant, un nouveau produit entre dans la chaîne. Le débit est donc d'un produit par unité de temps alors que la latence est de N unités de temps s'il y a n étapes de traitement.

1.1 Exemple de traitement Pipeliné

Le traitement d'une instruction peut se découper en plusieurs phases. A chacune d'elle il est possible d'associer une unité fonctionnelle. Par exemple si un micro-processeur dispose trois unités :

- Unité de recherche (R).
- Unité de décodage (D).
- Unité d'exécution (E).

Et si on suppose qu'elles remplissent leurs fonctions respectives en un cycle d'horloge. Le déroulement d'une instruction est :

- La phase « recherche » mobilise l'unité de recherche demandant un cycle d'horloge.
- La phase « décodage » occupe l'unité de décodage pendant un deuxième cycle.
- La phase « exécution » utilise l'unité d'exécution pendant un troisième cycle.

Il faut donc trois cycle d'horloge pour traiter une instruction. La prise en compte de l'instruction suivante se fait à partir du 4ème cycle, et ainsi de suite. L'exécution de N instructions nécessite donc 3*N cycles horloge.

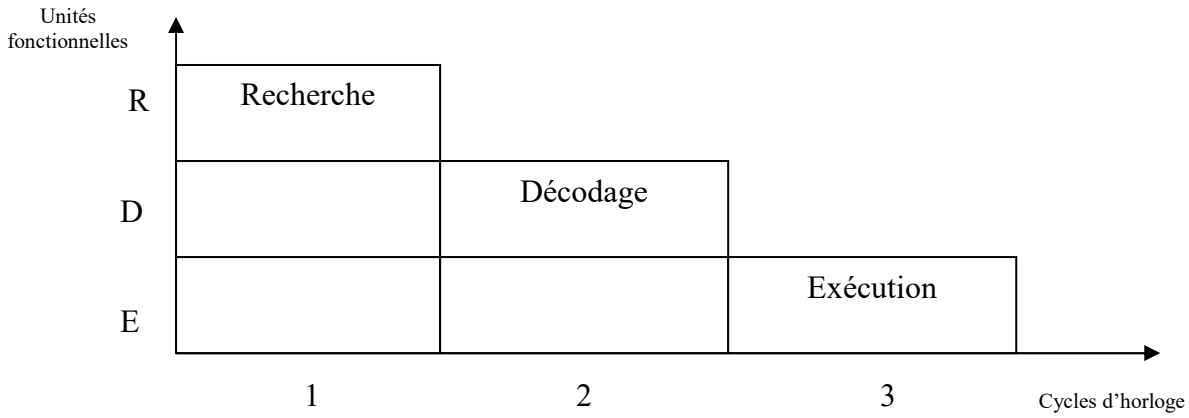

Figure 1**:** Modèle d'exécution à trois étages

Cependant nous constatons alors qu'à chaque cycle d'horloge deux des trois unités fonctionnelles sont inemployés. Il y a moyen d'accélérer les calculs en demandant à l'unité de recherche d'aller chercher l'instruction suivante dès le deuxième cycle d'horloge. Cette instruction est ensuite décodée au cycle suivant, alors que l'unité de recherche continue également.
Un processeur dans lequel ce traitement anticipé est possible est dit avec pipeline. Le compteur ordinal est incrémenté à chaque cycle d'horloge.
Dans un processeur avec pipeline il faut toujours 3 cycles pour exécuter chaque instruction, mais dès qu'une instruction est terminée il suffit d'attendre un cycle pour que la suivante soit terminée. Dans ces conditions il ne faut que N + 2 cycles d'horloge pour exécuter N instructions, donc 3 fois plus vite.

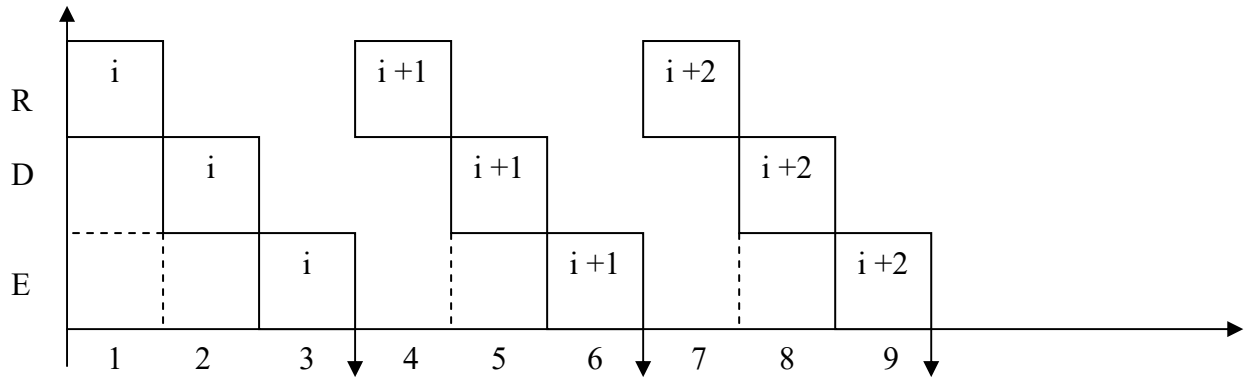

Figure 2**:** Modèle d'exécution classique à trois étages

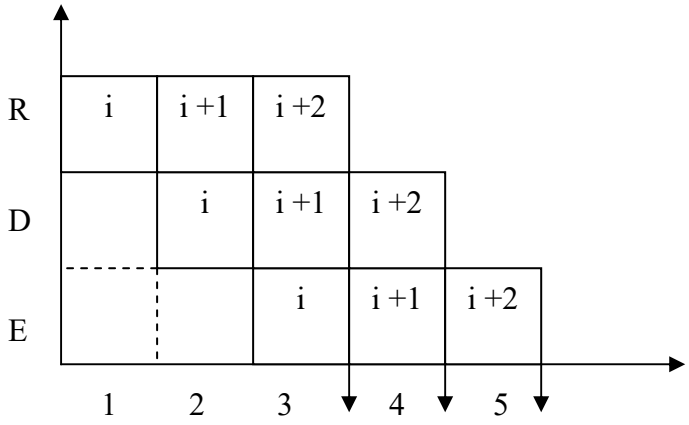

Figure 3**:** Modèle d'exécution en pipeline à trois étages

Pour améliorer les performances, il faut augmenter le nombre d'étages (unités) du pipeline
Exemple le processeur Pentium d'Intel contient 5 étages :

- Prefetch (PF)
- Decode 1 (D1)
- Decode 2 (D2)
- Execute (EX)
- Write back (WB)

- Pour augmenter le nombre d'étage, il faut éclater les fonctions (exemple Décodage = D1 et D2).
- Accélération du pipeline est le rapport entre le temps d'exécution sans pipeline et le temps d'exécution avec pipeline.

1.2 Problème d'utilisation des pipelines (Les Aléas)

- Aléas de données.
- Aléas des ruptures de séquence (Branchement, Appel/Retour de procédure, Interruption).

1.2.1 Aléas de données

- Selon la complexité la durée de traitement de toutes les instructions n'est pas nécessairement identique. Surtout les cas d'accès mémoire.
- Si on considère un processeur avec une quatrième UF chargée de la lecture en mémoire des opérandes.
  1. UF de recherche (R)
  2. UF de décodage (D)
  3. UF de lecture (L)
  4. UF d'exécution (E)
     - L'UF de lecture n'est pas toujours sollicité par les instructions.
     - Considérons le traitement de deux instructions I1 et I2.

- I1 écrivant en mémoire et I2 lisant en mémoire.
- Dans ce cas, un conflit de traitement se produira entre les deux instructions

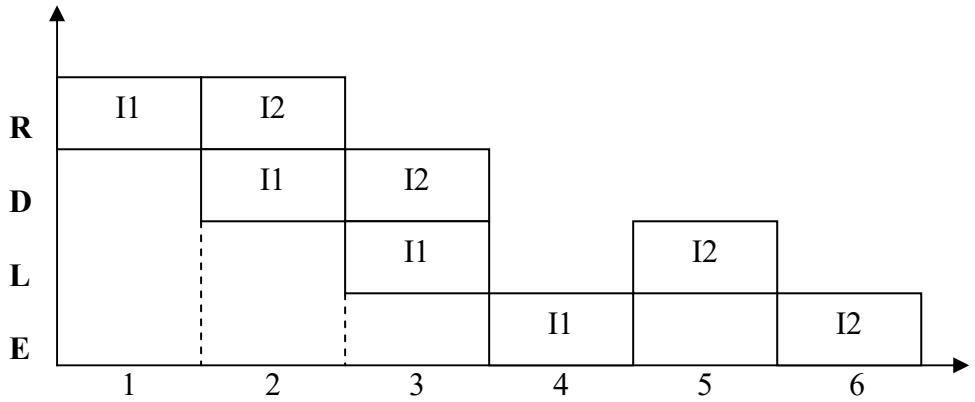

Figure 4**:** exemple d'aléa de données

- L'unité de lecture pour le besoin de l'instruction I2 aurait dû fonctionner au $4^{ème}$ cycle d'horloge.
- Mais simultanément l'unité d'exécution aurait été le train d'écrire en mémoire.
- Le traitement de l'instruction I2 est donc pénalisé d'un cycle.
- Dans cet exemple le temps d'accès à la mémoire est 1 cycle mais réellement ce temps peut correspondre à plusieurs cycles horloge et peut être variable en plus.
- Les deux instructions I1 et I2 peuvent faire appel au même bloc mémoire ou à deux blocs différents.
- En fonction de l'architecture de la mémoire que les temps d'accès pour les deux instructions peuvent être identiques dans un cas et différents dans l'autre.
- Il peut y avoir dépendance entre deux instructions, la seconde ayant besoin d'une adresse calculée par la première par exemple.

- La gestion de ces problème est généralement résolue par les compilateurs qui peuvent insérer des instructions NOP (Non opération = opération vide) ou permuter des instructions du programme en tenant compte des spécificités du processeur (architecture RISC).

1.2.2 Aléas de Branchement

1..2.1. Branchement

- Il faut gérer les instructions dont le traitement a été anticipé.
- Il existe deux types de branchement (simple et retardé) :
  1. Pour le branchement simple le traitement anticipé est interrompu et la chaine (Pipe) est vidée avant d'exécuter le branchement.
  2. Il faut ensuite attendre à nouveau trois cycles pour obtenir le résultat.
  3. Pour le branchement retardé on termine le traitement commencé avant d'exécuter le branchement.
- Une bonne utilisation de ces deux branchements permet d'optimiser le fonctionnement du processeur.
- On peut trouver dans certains processeurs une unité qui cherche à anticiper la destination d'un branchement, de manière statistique (et même probabiliste). Par exemple en fin de boucle.
- Souvent il existe plusieurs techniques de gestions des aléas de branchement :
  1. **Suspension automatique du pipeline** : vider la chaine en cas de branchement.
  2. **Branchement différé** : Exécution des instructions qui suivent immédiatement le branchement, cependant le compilateur doit changer l'ordre des instructions du programme et insérer des NOP de façon à assurer le bon déroulement du programme
  3. **Prédiction de branchement** : On tente de prédire (modèle aléatoire) si le branchement sera effectuer ou non. Si la prédiction est juste, aucun nettoyage du pipeline n'est requis, sinon une pénalité de branchement survient à cause de nettoyage du pipeline. Il existe deux types de prédictions :

- Prédiction statique : Basée sur la nature de branchement arrière ou avant (Exemple : le processeur PowerPC 601).
- Prédiction dynamique : repose sur les informations sur le comportement passé des branchements, de façon à prédire leur comportement future. Cette méthode nécessite une table de prédiction (Branch Target Buffer), utilisé surtout pour les appels de procédures.

- **Cas d'interruption**

Le traitement des interruptions, pose également quelques problèmes. Pour simplifier la gestion et la reprise du programme interrompu, les instructions engagées dans le pipeline sont traitées avant la prise en compte de l'interruption.