

Optimisation de code :

Pipeline logiciel

ARCHI 1

Karine Heydemann
karine.heydemann@lip6.fr

1 Rappel déroulage de boucle

2 Pipeline logiciel

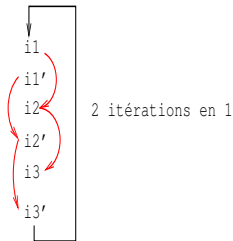
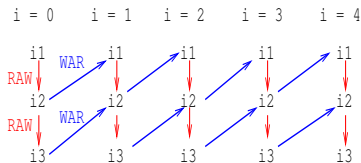
Déroutage de boucle : rappels

```
Loop: lw R4, 0(R5)      #i1
      sll R7, R4, 1  #i2
      sw R7, 0(R5)   #i3
      addiu R5, R5, 4 #i4
      bne R5, R9, Loop #i5
      nop           #i6
```

corps de la boucle

gestion de boucle

Déroutage : renommage de registres pour éliminer les dépendances WAR entre 2 (ou plus) itérations pour entrelacer les instructions des itérations



Introduction du pipeline logiciel

Loop:

lw R4, 0(R5)	#i1	corps de la boucle
sll R7, R4, 1	#i2	
sw R7, 0(R5)	#i3	
addiu R5, R5, 4	#i4	gestion de boucle
bne R5, R9, Loop	#i5	
nop	#i6	

Autre vision : sans renommage on peut réordonner des instructions de plusieurs itérations :

i1(0)

i2(0)

i1(1)

i3(0)

i2(1)

i1(2)

i3(1)

i2(2)

i1(3)

i3(2)

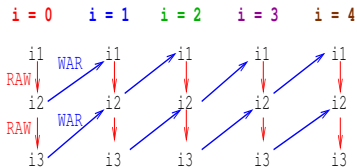
i2(3)

i1(4)

i3(3)

i2(4)

i3(4)

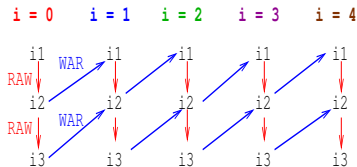


Introduction du pipeline logiciel

```

Loop:
lw R4, 0(R5)      #i1
sll R7, R4, 1     #i2
sw R7, 0(R5)      #i3
addiu R5, R5, 4   #i4
bne R5, R9, Loop  #i5
nop               #i6
    
```

Annotations: *gestion de boucle* (instructions #i4, #i5, #i6)



Autre vision : sans renommage on peut réordonner des instructions de plusieurs itérations :

```

i1(0)
i2(0)
i1(1)
    
```

meme motif :

```

i3(0)
i2(1)
i1(2)
    
```

```

i3(1)
i2(2)
i1(3)
    
```

```

i3(2)
i2(3)
i1(4)
    
```

```

i3(3)
i2(4)
i3(4)
    
```

Introduction du pipeline logiciel

```

Loop:
lw R4, 0(R5)      #i1
sll R7, R4, 1     #i2
sw R7, 0(R5)      #i3
addiu R5, R5, 4   #i4
bne R5, R9, Loop  #i5
nop               #i6
    
```

Autre vision : sans renommage on peut réordonner des instructions de plusieurs itérations :

```

i1(0)
i2(0)
i1(1)
i3(0)
i2(1)
i1(2)
i3(1)
i2(2)
i1(3)
i3(2)
i2(3)
i1(4)
i3(3)
i2(4)
i3(4)
    
```

mememotif :

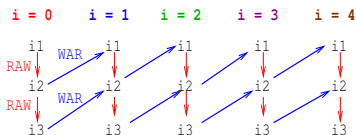
```

i3(i-2)
i2(i-1)
i1(i)
    
```

On peut reformer une boucle :

```

i3(i-2)
i2(i-1)
i1(i)
    
```



principe du pipeline logiciel :
 former une boucle avec
 des instructions appartenant
 à des itérations différentes

idee : commencer une itération
 avant que la précédente
 ait fini

Le pipeline logiciel

Même idée que le pipeline d'instructions

- On peut découper l'exécution d'une itération en N étapes
- On peut commencer l'étape M de l'itération i quand l'itération $i-1$ l'a terminée. MAIS logiciel = pas en // donc étape M de l'itération i possible quand l'itération $i-1$ a libéré les ressources nécessaires à l'étape M (registres utilisés) c-à-d a terminé l'étape suivante ($M+1$)

corps de boucle



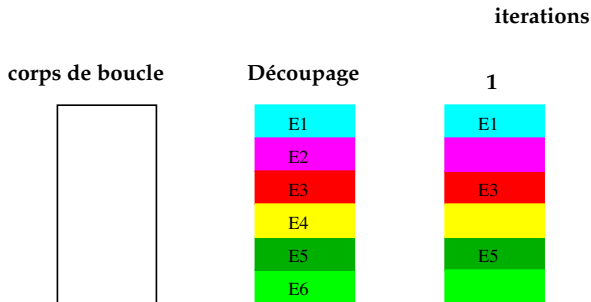
Découpage



Le pipeline logiciel

Même idée que le pipeline d'instructions

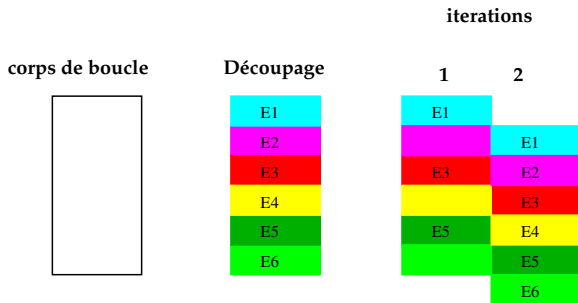
- On peut découper l'exécution d'une itération en N étapes
- On peut commencer l'étape M de l'itération i quand l'itération $i-1$ l'a terminée. MAIS logiciel = pas en // donc étape M de l'itération i possible quand l'itération $i-1$ a libéré les ressources nécessaires à l'étape M (registres utilisés) c-à-d a terminé l'étape suivante ($M+1$)



Le pipeline logiciel

Même idée que le pipeline d'instructions

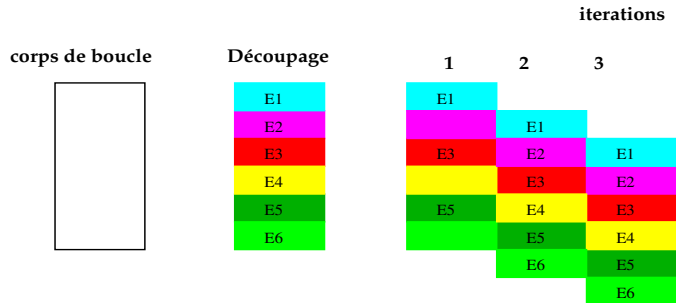
- On peut découper l'exécution d'une itération en N étapes
- On peut commencer l'étape M de l'itération i quand l'itération $i-1$ l'a terminée. MAIS logiciel = pas en // donc étape M de l'itération i possible quand l'itération $i-1$ a libéré les ressources nécessaires à l'étape M (registres utilisés) c-à-d a terminé l'étape suivante ($M+1$)



Le pipeline logiciel

Même idée que le pipeline d'instructions

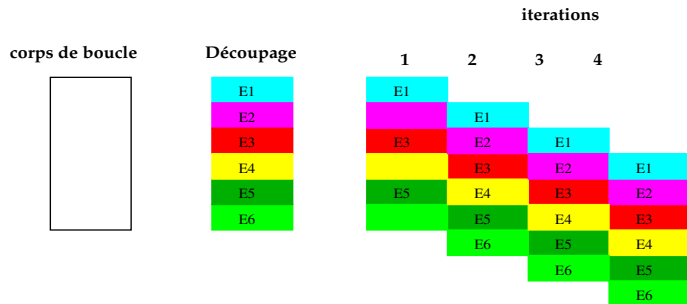
- On peut découper l'exécution d'une itération en N étapes
- On peut commencer l'étape M de l'itération i quand l'itération $i-1$ l'a terminée. MAIS logiciel = pas en // donc étape M de l'itération i possible quand l'itération $i-1$ a libéré les ressources nécessaires à l'étape M (registres utilisés) c-à-d a terminé l'étape suivante ($M+1$)



Le pipeline logiciel

Même idée que le pipeline d'instructions

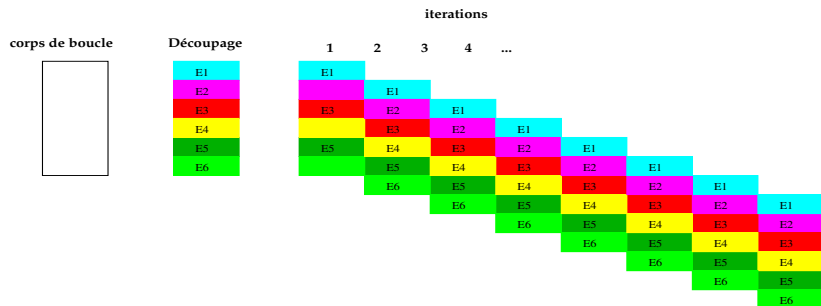
- On peut découper l'exécution d'une itération en N étapes
- On peut commencer l'étape M de l'itération i quand l'itération $i-1$ l'a terminée. MAIS logiciel = pas en // donc étape M de l'itération i possible quand l'itération $i-1$ a libéré les ressources nécessaires à l'étape M (registres utilisés) c-à-d a terminé l'étape suivante ($M+1$)



Le pipeline logiciel

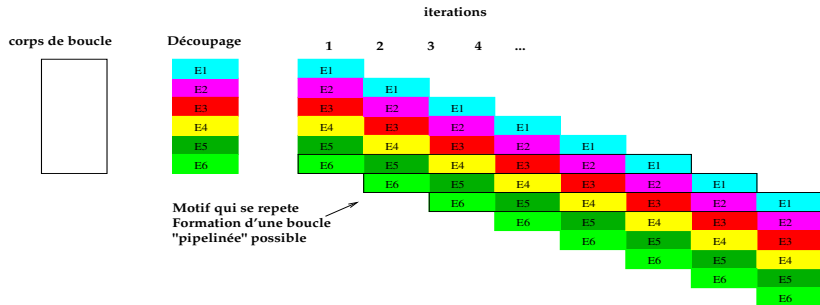
Même idée que le pipeline d'instructions

- On peut découper l'exécution d'une itération en N étapes
- On peut commencer l'étape M de l'itération i quand l'itération $i-1$ l'a terminée. MAIS logiciel = pas en // donc étape M de l'itération i possible quand l'itération $i-1$ a libéré les ressources nécessaires à l'étape M (registres utilisés) c-à-d a terminé l'étape suivante ($M+1$)



Le pipeline logiciel

- Motif qui se répète, une étape différente sur N itérations successives
- Formation d'une boucle pipelinée possible
- Un bout d'itération initiale exécutée dans la même itération (cf. en parallèle pour pipeline matériel)



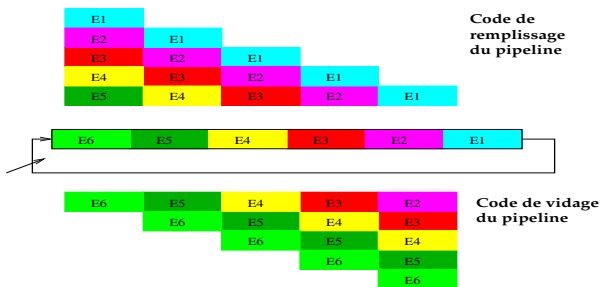
Le pipeline logiciel

- Boucle ordonnancant une étape de N itérations successives
- Ajout de code de remplissage/vidage du pipeline nécessaire

Découpage



Repliage du code
Formation d'une
boucle "pipelinée"



Pipeline logicielle : exemple

```
# a[i] dans R6
# a[N] dans R10

loop:
  lw R4, 0(R6) 'lecture elem i'
  sll R5, R4, 1 'multiplication par 2'
  sw R5, 0(R6) 'stockage elem i'
  addiu R6, R6, 4
  bne R10, R6, loop
```

Étape préalable : analyser la boucle pour déterminer les dépendances, cycles perdus ou limitation de performance (dépend de l'architecture cible)

Etape 1 : découpages d'une itération en étapes

- Considérer le corps et couper en étapes
- Couper là où il y a des cycles de gel, des dépendances, des limitations de performances (dépend de l'architecture cible)

```
# a[i] dans R6
# a[N] dans R10

loop:
    lw R4, 0(R6) 'lecture elem i'
    sll R5, R4, 1 'multiplication par 2'
    sw R5, 0(R6) 'stockage elem i'
    addiu R6, R6, 4
    bne R10, R6, loop
```


Etape 1 : découpages d'une itération en étapes

- Considérer le corps et couper en étapes
- Couper là où il y a des cycles de gel, des dépendances, des limitations de performances (dépend de l'architecture cible)
- Dépendances `lw` → `sll` → `sw`

```
E1 : lw R4, 0(R6)
```

```
E2 : sll R5, R4, 1
```

```
E3 : sw R5, 0(R6)
```

Etape 2 : Former le nouveau corps de boucle

- Nouveau corps : 1 étapes pour n itérations successives si découpage en n étapes
- Respecter l'ordre des instructions : itérations les plus anciennes en 1er
- Si 1ère étape correspond à itération i : $E_n(i - n + 1), E_{n-1}(i - n + 2), \dots, E_2(i - 1), E_1(i)$
- $R6 = a[i]$ est incrémenté à chaque itération, il faut ajuster si besoin les instructions qui l'utilisent

```
E3(i-2) : sw R5, '??' (R6)
```

```
E2(i-1) : sll R5, R4, 1
```

```
E1(i) : lw R4, '??' (R6)
```

Etape 2 : Former le nouveau corps de boucle

- R6 incrémenté de 4 à chaque itération, il faut ajuster les instructions qui l'utilisent
- R6 contient l'adresse de l'élément i

```
E3(i-2) : sw R5, '-8'(R6) 'elem i-2 se trouve -2x4'  
          'octets avant elem i en memoire'  
E2(i-1) : sll R5, R4, 1  
E1(i)   : lw R4, '0'(R6) 'elem i, adresse dans R6'
```

Etape 3 : Mettre la gestion de boucle

- On doit s'arrêter au bon moment
- Arrêt lorsque l'on a lu le dernier élément (étape 1)
- Arrêt à l'adresse de l'élément N et démarrage à $i=2$
- Il faut que le registre R6 contiennent l'adresse de du 2ème élément avant la boucle

```
loop: 'de 2 a N-1'  
    sw R5, -8(R6) 'E3(i-2)'  
    sll R5, R4, 1 'E2(i-1)'  
    lw R4, 0(R6) 'E1(i)'  
    addiu R6, R6, 4  
    bne R6, R10, loop '#R10 contient @tab[N]'  
    nop
```

Etape 3 : Mettre le prologue et epilogue

- On doit s'arrêter au bon moment
- Arrêt lorsque l'on a lu le dernier élément (étape 1)
- Arrêt à l'adresse de l'élément N et démarrage à $i=2$
- Il faut que le registre R6 contienne l'adresse de du 2ème élément avant la boucle

```
'avant : E1(0), E2(0), E1(1)'  
      'R6 = @tab[2]'  
loop: 'de 2 a N-1'  
      sw R5, -8(R6) 'E3(i-2)'  
      sll R5, R4, 1 'E2(i-1)'  
      lw R4, 0(R6) 'E1(i)'  
      addiu R6, R6, 4  
      bne R6, R10, loop '#R10 contient @tab[N]'  
      nop  
'apres : E3(N-2), E2(N-1), E3(N-1)'
```

Réordonnement

- Réordonner les instructions pour éviter les nop et cycles de gel
- Plus de dépendance RAW intra-iteration des instructions du corps de boucle
- Dépendances RAW inter-itération

```
'avant : E1(0), E2(0), E1(1)
      R6 = @tab[2]'
```

```
loop : 'i de 2 a N-1'
      sw R5, '-8'(R6) 'E3(i-2)'
```

```
      sll R5, R4, 1 'E2(i-1)'
```

```
      lw R4, 0(R6) 'E1(i)'
```

```
      addiu R6, R6, 4
```

```
      bne R6, R10, loop '#R10 contient @tab[N]'
```

```
      nop
```

```
'apres : E3(N-2), E2(N-1), E3(N-1)'
```

Réordonnement

- Remplissage du delayed slot : `lw` dans le delayed slot
- Élimination du cycle de gel `addiu` → `bne` : remonter le `addiu`
- 1 itération = 5 cycles sur MIPS32
- Et en SS2 ?

```
'avant : E1(0), E2(0), E1(1)
      R6 = @tab[2]'
loop : 'i de 2 a N-1'
      sw R5, '-8'(R6) 'E3(i-2)'
      addiu R6, R6, 4
      sll R5, R4, 1 'E2(i-1)'
      bne R6, R10, loop '#R10 contient @tab[N]'
      lw R4, '-4'(R6) 'E1(i)'
'apres : E3(N-2), E2(N-1), E3(N-1)'
```

Pipeline logicielle : exemple 2

```
' a[] dans R6 '  
' a[N] dans R10 '  
  
loop:  
  lw R4, 0(R6) 'lecture val elem i'  
  sll R5, R4, 1 'val * 2'  
  add R5, R5, R4 'val * 2 + val == val * 3'  
  sw R5, 0(R6) 'stockage elem i'  
  addiu R6, R6, 4  
  bne R10, R6, loop
```

- Analyse du corps + découpage

Pipeline logicielle : exemple 2

```
' a[] dans R6 '  
' a[N] dans R10 '  
  
loop:  
  lw R4, 0(R6) 'lecture val elem i'  
  sll R5, R4, 1 'val * 2'  
  add R5, R5, R4 'val * 2 + val == val * 3'  
  sw R5, 0(R6) 'stockage elem i'  
  addiu R6, R6, 4  
  bne R10, R6, loop
```

- Analyse du corps + découpage
- Dépendances RAW : lw → sll → add → sw
- Découpage en 4 étapes

Pipeline logicielle : exemple 2

- Dépendances RAW : lw → sll → add → sw
- Découpage en 4 étapes
- Boucle pipelinée :

```
' a[i] dans R6 '  
' a[N] dans R10 '  
  
loop:  
  sw R5, '-12'(R6) 'E4(i-3) stockage elem i-3'  
  add R5, R5, R4 'E3(i-2) elem i-2 * 3'  
  sll R5, R4, 1 'E2(i-1) elem i-1 * (2 + 1)'  
  lw R4, 0(R6) 'E1(i) lecture elem i'  
  
  addiu R6, R6, 4  
  bne R10, R6, loop
```

La boucle est elle correcte ?

Pipeline logicielle : exemple 2

```
' a[] dans R6 '  
' a[N] dans R10 '  
loop:  
  sw R5, '-12'(R6) 'E4 : stockage elem i-3'  
  add R5, R5, R4 'E3 : elem i-1 * (2 + 1)'  
  sll R5, R4, 1 'E2 : elem i-2 * 2'  
  lw R4, 0(R6) 'E1 : lecture elem i'  
  
  addiu R6, R6, 4  
  bne R10, R6, loop
```

- La boucle est elle correcte ?
 - E1 produit R4 utilisé en E2 et E3
 - E2 produit R5 utilisé en E3
 - E3 utilisé R5 et R4 et produit R5 pour E4
 - Explications au tableau...

Pipeline logicielle : exemple 2

```
' a[] dans R6 '  
' a[N] dans R10 '  
loop:  
  sw R5, '-12'(R6) 'E4 : stockage elem i-3'  
  add R5, R5, R4 'E3 : elem i-1 * (2 + 1)'  
  sll R5, R4, 1 'E2 : elem i-2 * 2'  
  lw R4, 0(R6) 'E1 : lecture elem i'  
  
  addiu R6, R6, 4  
  bne R10, R6, loop
```

- La boucle est elle correcte ?
- E1 produit R4 utilisé en E2 et E3
- E2 produit R5 utilisé en E3
- E3 utilisé R5 et R4 et produit R5 pour E4
- Explications au tableau...

Pipeline logicielle : exemple 2

```
' a[] dans R6 '  
' a[N] dans R10 '  
loop:  
  sw R5, '-12'(R6) 'E4 : stockage elem i-3'  
  add R5, R5, R4 'E3 : elem i-1 * (2 + 1)'  
  sll R5, R4, 1 'E2 : elem i-2 * 2'  
  lw R4, 0(R6) 'E1 : lecture elem i'  
  
  addiu R6, R6, 4  
  bne R10, R6, loop
```

- La boucle est elle correcte ?
- E1 produit R4 utilisé en E2 et E3
- E2 produit R5 utilisé en E3
- E3 utilisé R5 et R4 et produit R5 pour E4
- Explications au tableau...

Pipeline logicielle : exemple 2

```
' a[] dans R6 '  
' a[N] dans R10 '  
loop:  
  sw R5, '-12'(R6) 'E4 : stockage elem i-3'  
  add R5, R5, R4 'E3 : elem i-1 * (2 + 1)'  
  sll R5, R4, 1 'E2 : elem i-2 * 2'  
  lw R4, 0(R6) 'E1 : lecture elem i'  
  
  addiu R6, R6, 4  
  bne R10, R6, loop
```

- La boucle est elle correcte ?
- E1 produit R4 utilisé en E2 et E3
- E2 produit R5 utilisé en E3
- E3 utilisé R5 et R4 et produit R5 pour E4
- Explications au tableau...

Pipeline logicielle : exemple 2

```
' a[] dans R6 '  
' a[N] dans R10 '  
loop:  
  sw R5, '-12'(R6) 'E4 : stockage elem i-3'  
  add R5, R5, R4 'E3 : elem i-1 * (2 + 1)'  
  sll R5, R4, 1 'E2 : elem i-2 * 2'  
  lw R4, 0(R6) 'E1 : lecture elem i'  
  
  addiu R6, R6, 4  
  bne R10, R6, loop
```

- La boucle est elle correcte ?
- E1 produit R4 utilisé en E2 et E3
- E2 produit R5 utilisé en E3
- E3 utilisé R5 et R4 et produit R5 pour E4
- Explications au tableau...

Pipeline logicielle : retour sur le découpage

- Chaque étape doit produire une valeur dans un registre différent des autres : si ce n'est pas renommer les registres
- Chaque étape doit utiliser des valeurs de l'étape précédente et produire pour l'étape suivante uniquement : sinon, ajout d'instructions pour faire passer la valeur via des registres différents à chaque étape intermédiaire

```
' a[i] dans R6, a[N] dans R10'  
loop:  
  sw 'R7', '-12'(R6) 'E4(i-3) stockage elem i-3'  
  add 'R7', R5, 'R9' 'E3(i-2) elem i-1 * (2 + 1)'  
  'ori R9, R4, 0' 'E2(i-1) sauvegarde de R4'  
  sll R5, R4, 1 'E2(i-1) elem i-2 * 2'  
  lw R4, 0(R6) 'E1(i) lecture elem i'  
  addiu R6, R6, 4  
  bne R10, R6, loop
```


Pipeline logicielle : quid des dépendances de contrôle ?

Exemple 3

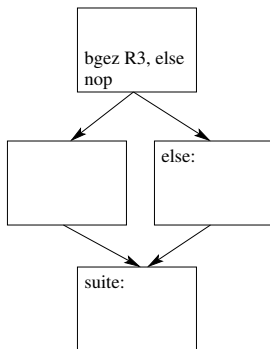
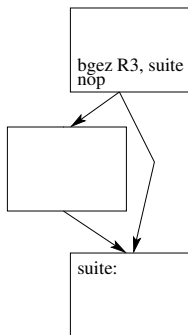
```
loop:
    lw R3, 0(R4)
    bgez R3, suite
    nop
    sub R3, R0, R3
    sw R3, 0(R4)
suite:
    addiu R4, R4, 4
    bne R4, R10, loop
    nop
```

Exemple 4

```
loop:
    lw R3, 0(R4)
    bgez R3, else
    nop
    addi R3, R3, -1
    j suite
else:
    addi R3, R3, 1
suite:
    sw R3, 0(R4)
    addiu R4, R4, 4
    bne R4, R10, loop
    nop
```

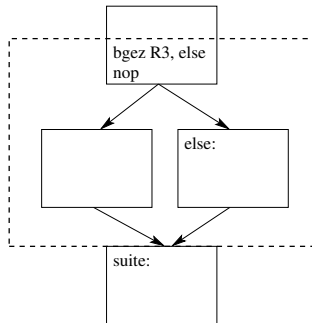
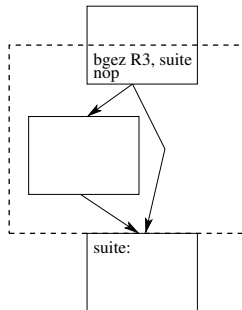
Pipeline logicielle : quid des dépendances de contrôle ?

- Où peut on couper l'itération ?



Pipeline logicielle : quid des dépendances de contrôle ?

- On ne peut séparer un branchement des instructions dont l'exécution en dépend, on ne peut dire à une itération suivante si on doit les exécuter ou non



Résumé : comment découper une itération ?

- Ne considérer que le corps de la boucle
- Les instructions dépendantes d'un branchement (dépendance de contrôle) doivent être dans la même étape que le branchement
On ne coupe que dans les blocs dominants
- Couper là où il y a perte de performance, dépendance de type RAW/cycle de gel (lié à l'architecture)
- Une fois le découpage réalisé, vérifier :
 - Chaque étape fournit des valeurs **uniquement** à la suivante, sinon ajouter des instructions pour faire passer les valeurs ou revoir le découpage en étapes
 - Chaque étape écrit dans des registres différents des autres étapes, sinon pertes/incohérences de valeur.