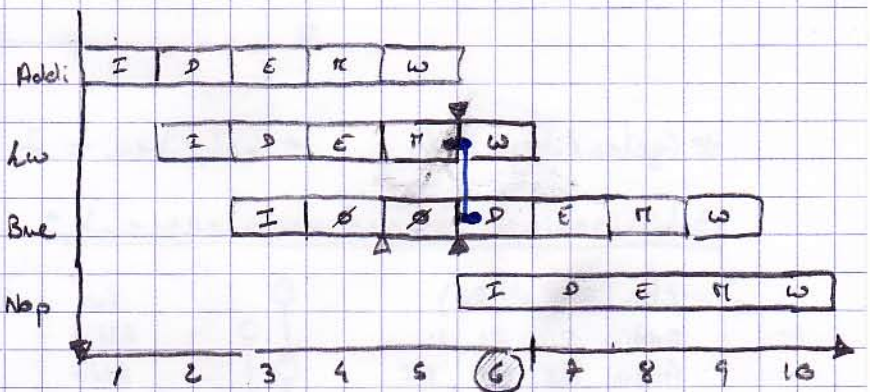
Exo 1 : Ecrire en assembleur Mips-32 la boucle principale de getlistlength.

On suppose que R4 contient l'adresse de la liste :

```
(*)
int getlistlength(struct chain *pt) {        // R4
    int i = 0;                               // R2
    while (pt != NULL) {
        i++;
        pt = pt->next;
    }
    return (i);
}
```

```
getlistlength :
    Ori R2, R0, 0
    Beq R4, R0, Endwhile
    nop
while :
    Addi R2, R2, 1
    Lw R4, 0(R4)
    Bne R4, R0, while
    nop
Endwhile :
    Jr R31
    nop
```

```
struct chain {
    struct chain* NEXT;
    void * Data ; };
```



Cycles/iterations = 6

instr/iterations = 4

$\Rightarrow$ CPI = $\frac{6}{4}$ = 1,5 cycles/inst. $\Rightarrow$ $CPI_{while} = \frac{6}{3}$ = 2 cycles/inst.

Optimisation du code :

```
Lw R4, 0(R4)
Addi R2, R2, 1
Bne R4, R0, while
nop
```

cycles/iter = 5
instr/iter = 3
$\Rightarrow$ CPI = $\frac{5}{3}$

⚠ Pas d'optimisation par déroulage car c'est un while ...

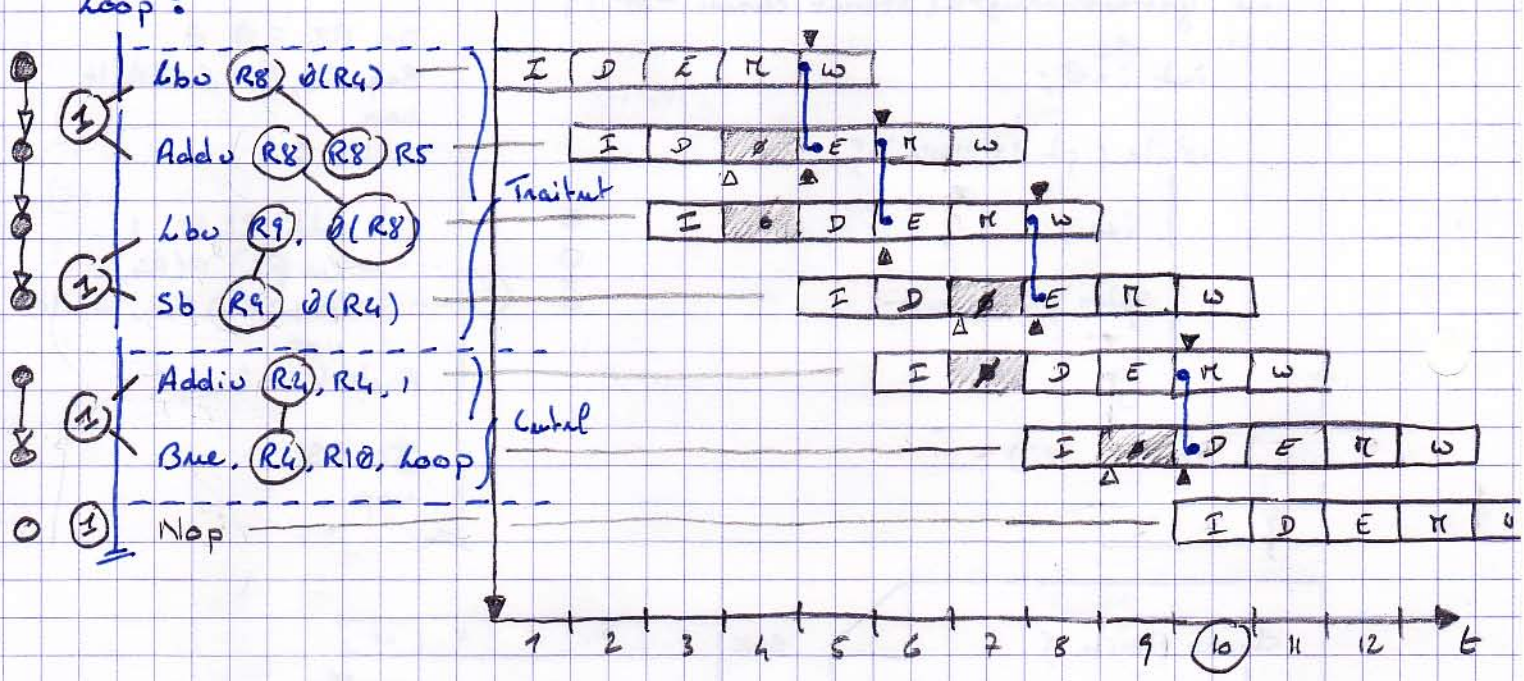
Exe 2 : Modifier le code pour qu'il s'exécute sur le Mips-32.

Analysez l'exécution de ce programme dans le Mips. Calculer le CPI et le CPI utile.

Optimisez le code en changeant l'ordre des instructions, calculez les CPI.

Optimisez le code en utilisant le "pipeline logiciel" puis calculez les CPI.

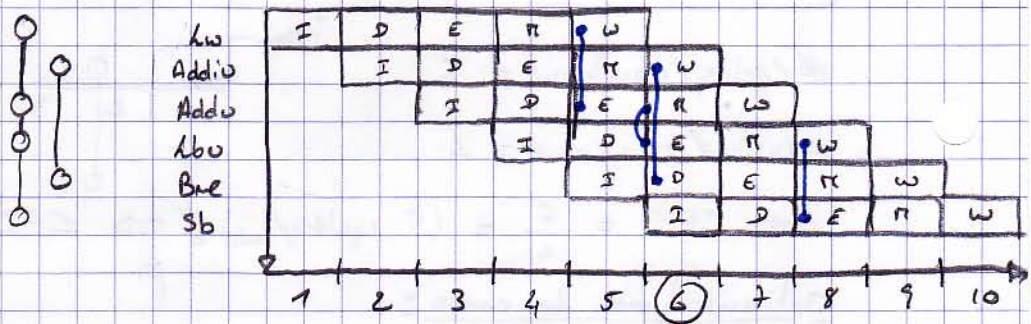
Loop :



Cycles/iter = 10 # inst/iter = 7 $\Rightarrow$ CPI = $\frac{10}{7}$ $CPI_{utile} = \frac{10}{6}$

Optimisation par ordonnancement :

```
Lbu R8, 0(R4)
Addiu R4, R4, 1
Addu R8, R8, R5
Lbu R9, 0(R8)
Bne R4, R10, Loop
Sb R9, -1(R4)
```



$\Rightarrow CPI_{utile} = 1$

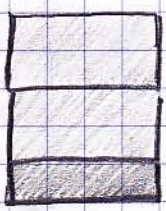
Optimisation par dénoulage de boucle :

```
Lbu R8, 0(R4)
Addu R8, R8, R5
Lbu R9, 0(R8)
Sb R9, 0(R4)

Lbu R10, 1(R4)
Addiu R6, R6, R5
Lbu R7, 0(R6)
Sb R7, 1(R4)

Addiu R4, R4, 2
Bne R4, R10, Loop
Nop
```

```
Lbu R8, 0(R4)
Lbu R10, 1(R4)
Addiu R4, R4, 2
Addiu R8, R8, R5
Addiu R6, R6, R5
Lbu R9, 0(R8)
Lbu R7, 0(R6)
Sb R9, -2(R4)
Bne R4, R10, Loop
Sb R7, -1(R4)
```

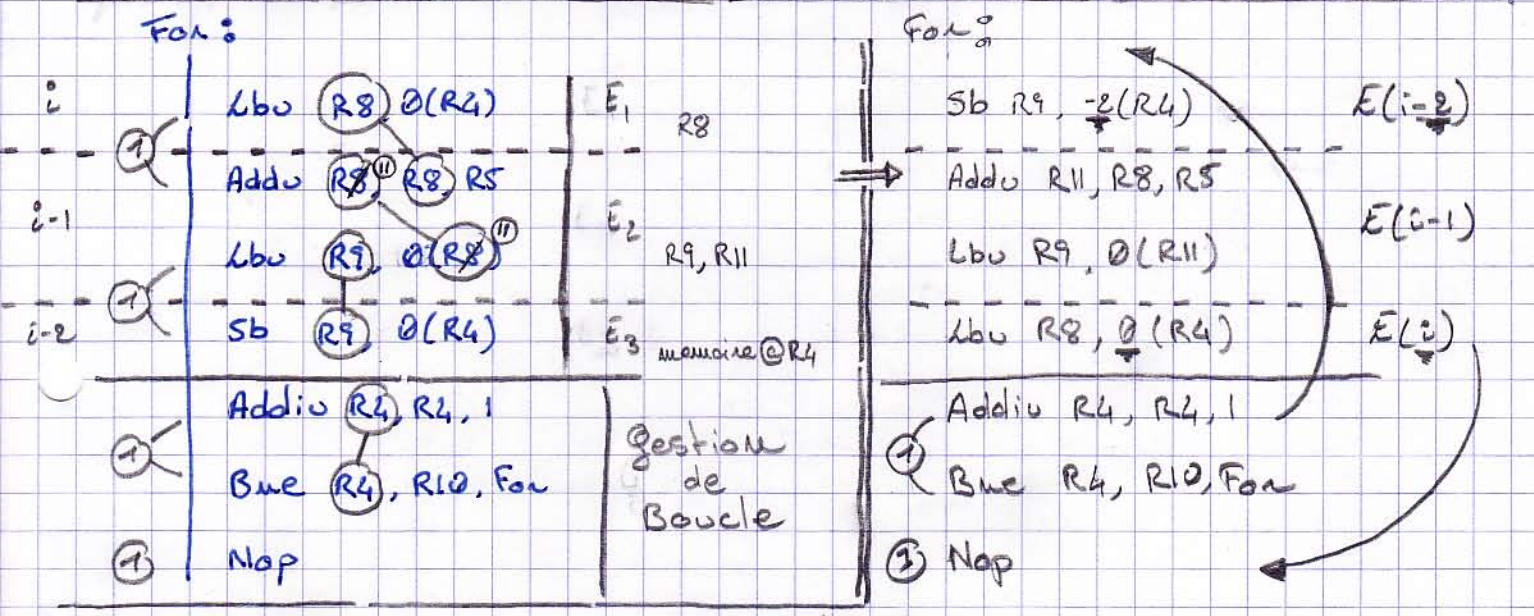


$\Rightarrow CPI_{utile} = 1$

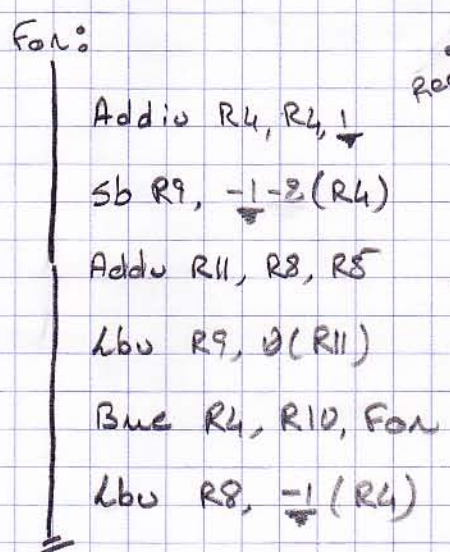
Exo 2 : Optimisez le code initiale avec le pipeline logiciel.

Algorithme de l'optimisation du "Software pipeline" :

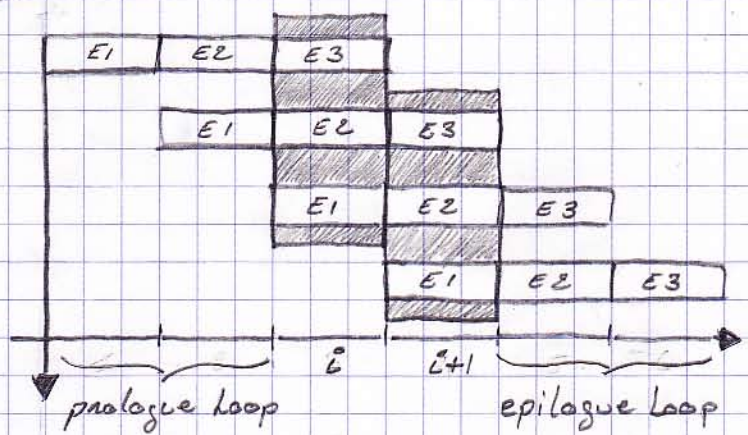
1. Separer le Traitement et la Gestion de boucle.
2. Déterminer les dependances de données dans le Traitement.
3. Indiquer les pertes de performances (cycles de gels).
4. Decouper le Traitement sur ces pertes.
5. Verifier que les règles soient bien respectés :
6. ~~Règle 1 du pipeline~~ => Ajouter des instructions de transfert.
   ~~Règle 3 du pipeline~~ => Renommer les registres.
7. Mettre le code en pipeline (réorganiser le code).
8. Effectuer le Réordonnancement de la Gestion de boucle.



Réordonnancement



Resultat

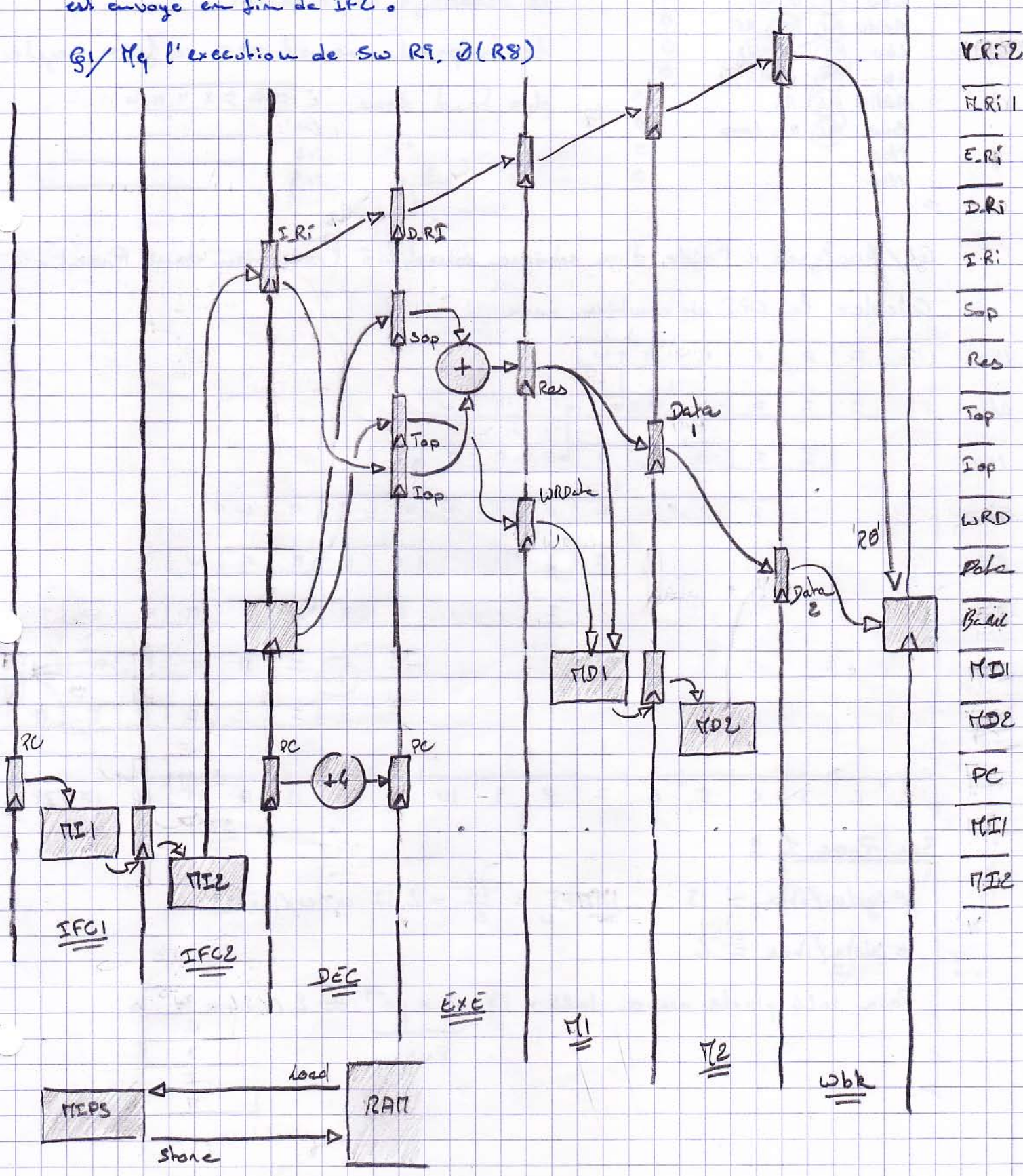


Exe 3 : On considère un processeur RISC pipeline appelé PROC qui a le même jeu d'instruction que le Mips-32.

PROC a un pipeline à 7 étages : IF1, IF2, DEC, EXE, MM1, MM2, WBK.

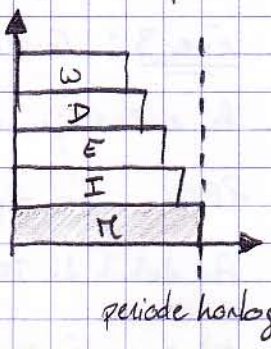
Au début de IF1, l'adresse est envoyé à la mémoire et l'instr. correspondante est envoyé en fin de IF2.

Q1/ Mq l'execution de Sw R9, 8(R8)



Q2/ Expliquez pourquoi la période d'horloge est plus longue dans PROC2

Il y a un bypass à l'étage MEM1 de Proc2, ce qui implique que l'on a un multiplexeur sur le chemin critique.

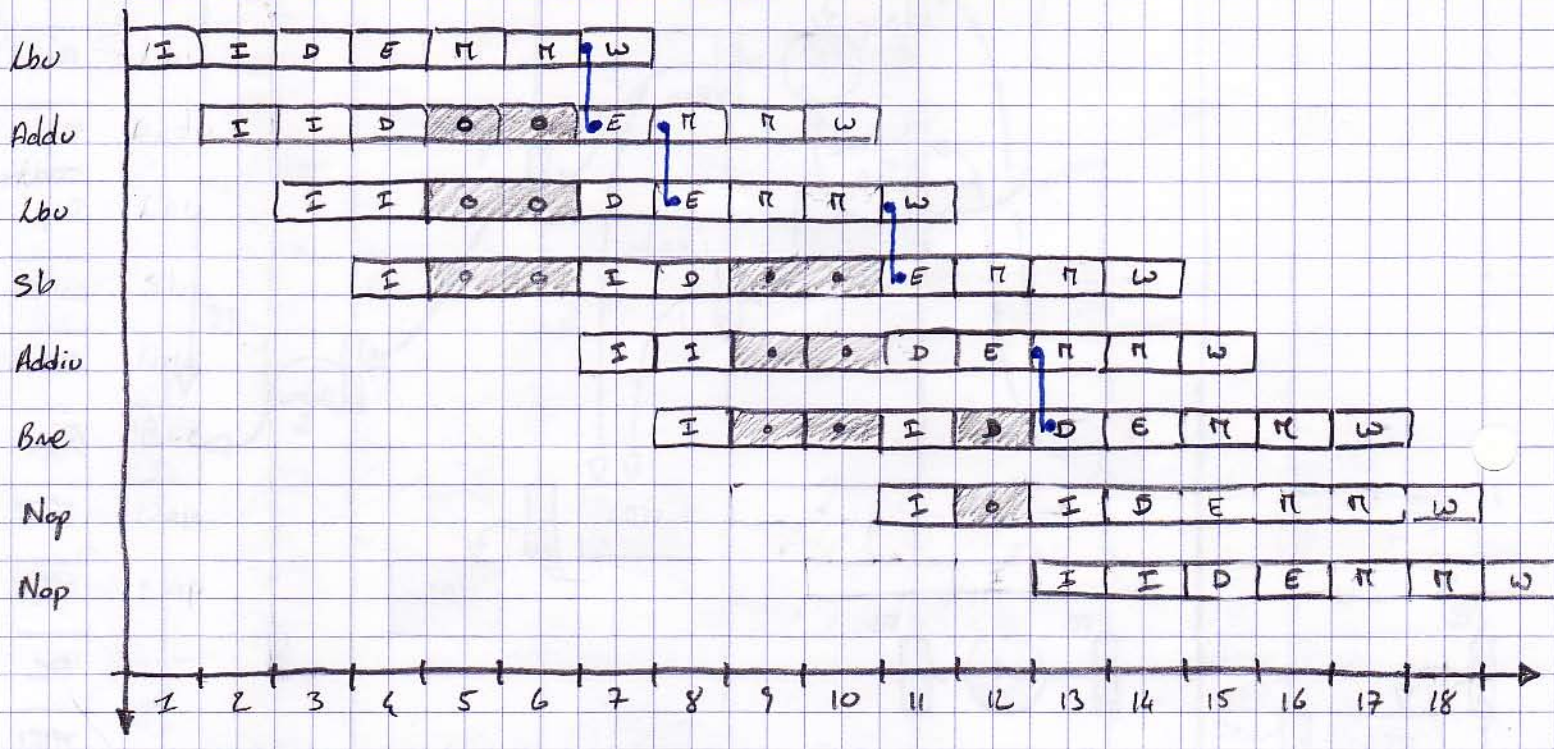


Q3/ On reprend le code de l'exercice précédent. Modifier le code de ce programme pour qu'il soit exécutable sur Proc.

```
Loop :
Lbu  R8, 0(R4)
Addu R8, R8, R5
Lbu  R9, 0(R8)
Sb   R9, 0(R4)
Addiu R4, R4, 1
Bne  R4, R10, Loop
Nop
Nop
```

Le décodage et le calcul de la prochaine adresse se fait 1 cycle plus tard donc

Q4/ Analysez à l'aide d'un schéma simplifié l'exécution dans Proc1. Calculez le CPI et combien dure



Sur Proc 1 :

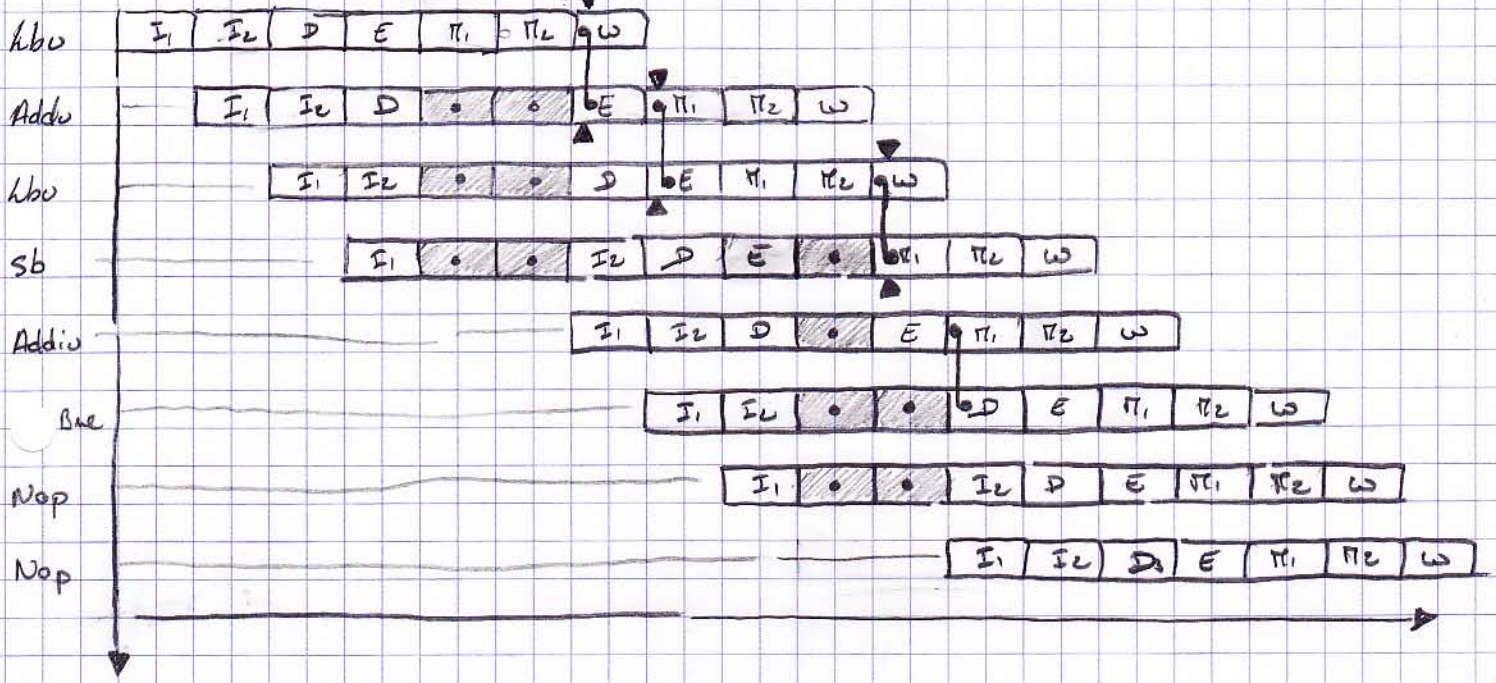
#cycles/iter = 13 $\quad CPI_U = \frac{13}{6} = 2,17$ cycles/instr

#instr$_U$/iter = 6

Pour 1024 pixels on a $1024 \times 13 \times \underbrace{2 \times 10^{-9}}_{2ns} \approx 2,6624 \times 10^{-5}$ s

Exe 3 : Suite

Q5/ Analysez, à l'aide d'un schéma simplifié, l'execution dans Proc2.
Calculez le CPI et le temps de traitement pour 1024 pixels.



Sur Proc2 :

#cycles/iter = 12 $CPI_U = \frac{12}{6} = 8$ cycles/instr$_U$

#instr$_U$/iter = 6

Pour 1024 pixels on a $1024 \times 12 \times \underbrace{2,1 \times 10^{-9}}_{2,1\ ns} \approx 2,58048 \times 10^{-5}$ s

$\Rightarrow P = \frac{F}{CPI}$ Donc $\frac{F_1}{CPI_1}$ et $\frac{F_2}{CPI_2}$ on a $F_1 > F_2$, $CPI_1 > CPI_2$.

$\Rightarrow$ Proc2 est plus performant pour l'instant.

Q6-7/ Ré-ordonnez le code pour Proc1 et Proc2 afin d'éviter les cycles de gel. Calculez leur CPI_U et la donnée traitement pour 1024 pixels.

Init :

```
(2) Lbu  R8, 0(R4)
    Addu R8, R8, R5
(2)(1) Lbu  R9, 0(R8)
    Sb   R9, 0(R4)
(1) Addiu R4, R4, 1
    Bne  R4, R10, Loop
(1) Nop
(1) Nop
```

Réordonnancement :

```
    Lbu  R8, 0(R4)
(1) Addiu R4, R4, 1
    Addu R8, R8, R5
    Lbu  R9, 0(R8)
    Bne  R4, R10, Loop
(1) Nop
    Sb   R9, -1(R4)
```

Idem pour Proc2

Proc1 :
#C = 8 et #I_U = 6
$\Rightarrow CPI_U = \frac{8}{6} = 1,33$ c/I
$\Rightarrow 1024 \times 8 \times (2 \times 10^{-9}) = 1,6384 \times 10^{-5}$

Proc2 :
#C = 8 et I_U = 6
$\Rightarrow CPI_U = \frac{8}{6} \approx 1,33$ c/I
$\Rightarrow 1024 \times 8 \times 2,1 \times 10^{-9}$
$= 1,72039 \times 10^{-5}$ secondes

Exe 4 : On a un RISC pipliné à 9 étages : IF_1, IF_2, DEC_1, DEC_2, EXE_1, EXE_2 MEM_1, MEM_2, WBK. Chaque étage a été découpé en deux sauf WBK. Le calcul de l'adresse suivante dans l'étage DEC_2.

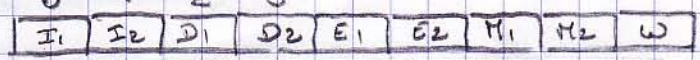
On souhaite étudier le fonctionnement de ce processeur. Le code suivant est la boucle principale d'une fonction calculant la valeur absolue des elements d'un tab d'entier.

A l'entrée de la boucle R4 contient l'adresse du tableau, le registre R9 contient l'adresse de la fin du tableau.

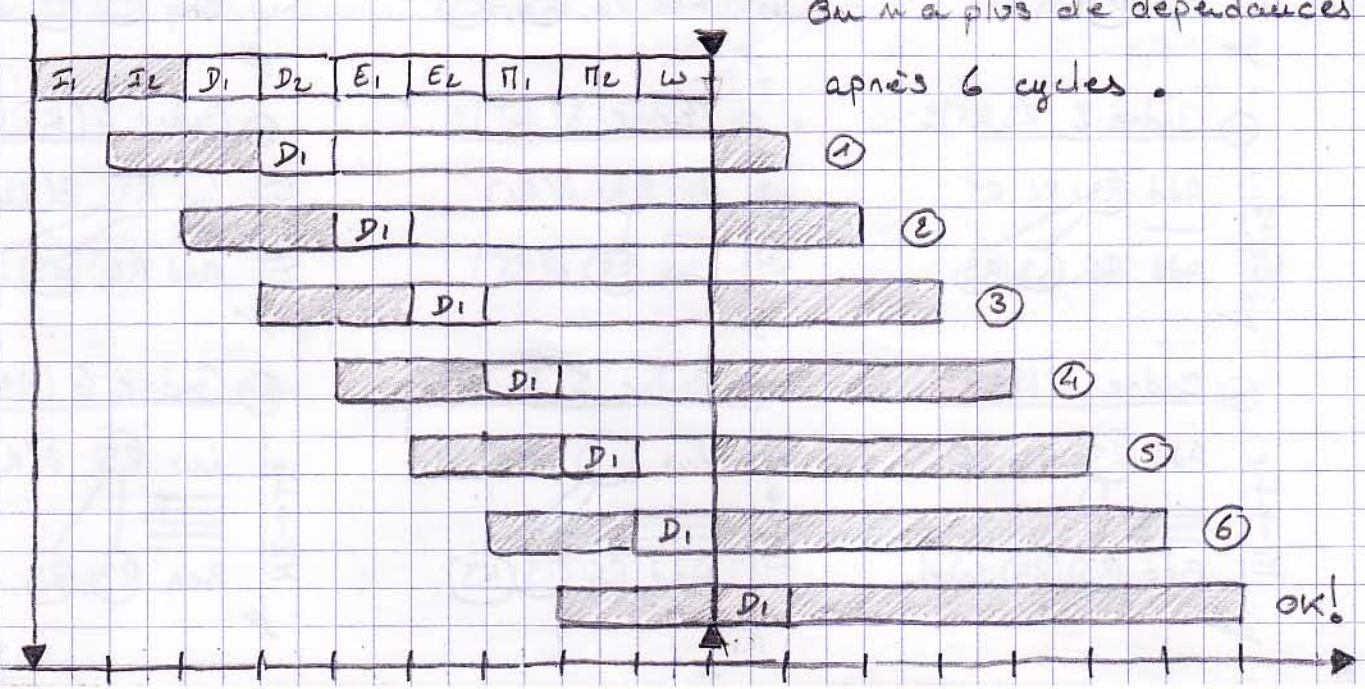
```
Loop:
    lw   R8, 0(R4)
    Bgez R8, Endif
    Sub  R8, R0, R8
    Sw   R8, 0(R4)
Endif:
    Addiu R4, R4, 4
    Bne  R4, R9, Loop
```

Q1/ Le découpage du pipeline en 9 étages a-t-il un impact sur le compilateur ?

Oui, on a 3 Delayed Slots pour la position du calcul de l'adresse dans ce pipeline à 9 étage :

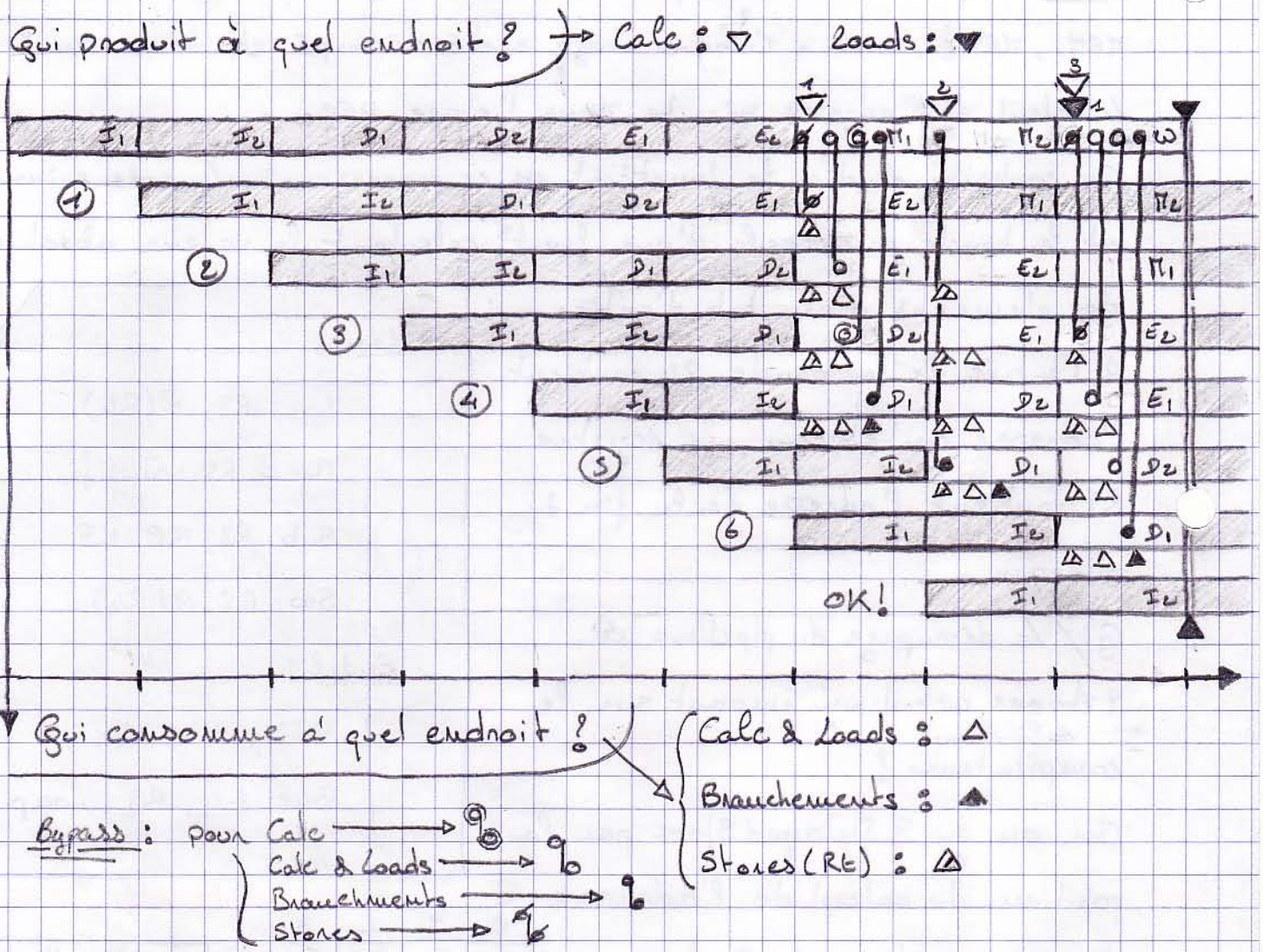


Q2/ Dans le processeur Mips, il existe des dependences d'ordre 1, 2 et 3. Dans P9, qu'elles sont les dependances de données ?



On n'a plus de dependances après 6 cycles.

Q3/ Quels sont les bypass nécessaires à l'exécution des instructs dans ce PP ? Illustrer pour chaque bypass son utilisation à l'aide d'un exemple de suite d'instructions.



Ordre 1 (RT) :
E2 ← E2
Add (R3) R4, R5
Sw (R3), 0(R5)

Ordre 3 (RS, RT) :
E2 ← D2
Add (R3), R4, R5
Add R6, (R3), (R3)

Ordre 4 (RS, RT) :
E2 ← D1
Add (R3) R4, R5
Beq (R3), (R3), label

Ordre 2 (RS, RT) :
E2 ← E1
Add (R3), R4, R5
Add R6, (R3), (R3)

Ordre 3 (RT) :
M2 ← E2
Lw (R3), 0(R4)
Sw (R3), 0(R5)

Ordre 4 (RS, RT) :
M2 ← E1
Lw (R3), 0(R4)
Add R6, (R3), (R3)

Ordre 5 (RS, RT) :
E1 ← D1
Add (R3) R4, R5
Beq (R3), (R3) label

Ordre 5 (RS, RT) :
M2 ← D2
Lw (R3) 0(R4)
Add R6, (R3) (R3)

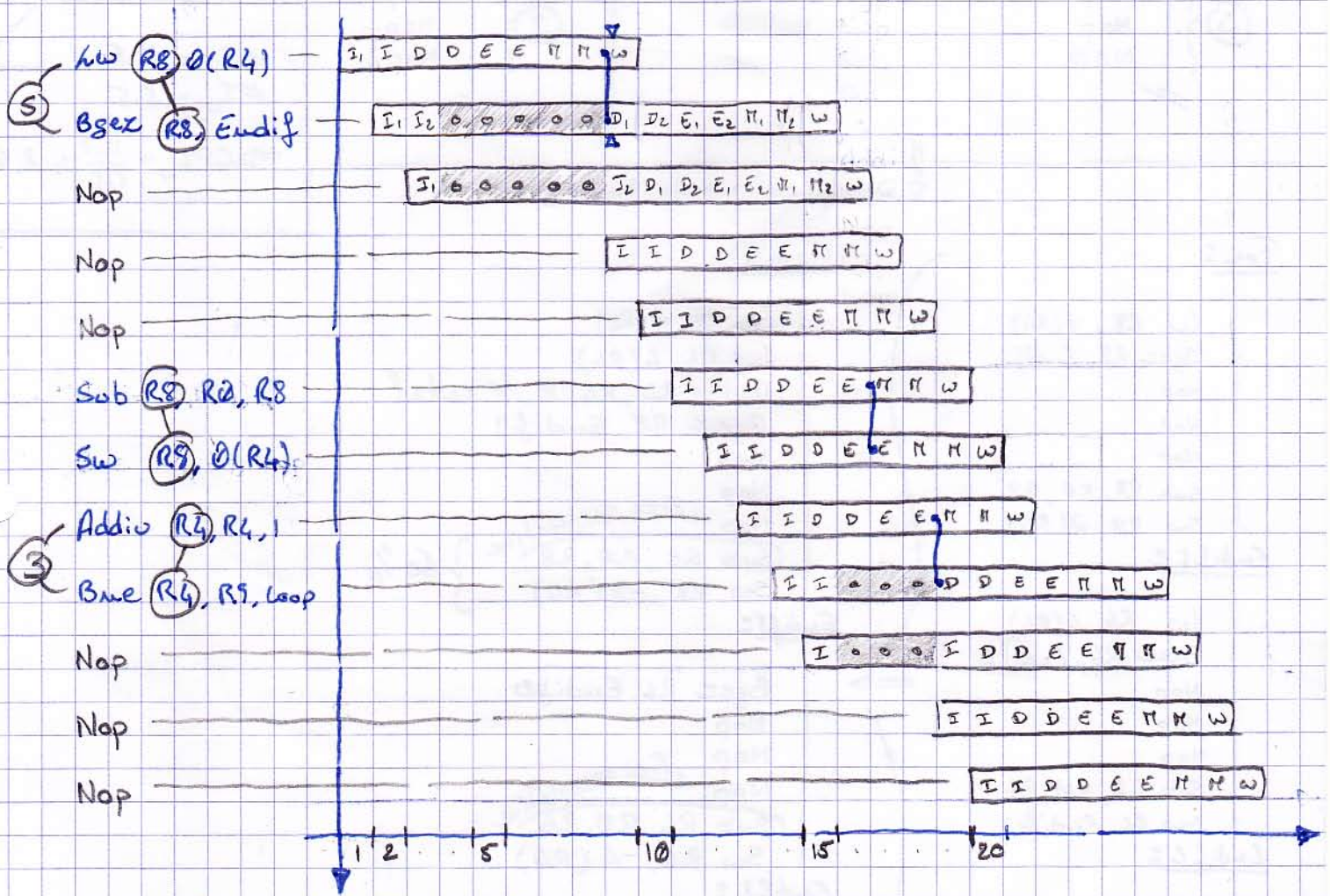
Ordre 6 (RS, RT) :
D1 ← M2
Lw (R3), 0(R4)
Beq (R3), (R3), label

Exo 4 : Suite

Q4/ Quelles sont les situations provoquant des cycles de gels ? Illustrez chaque cas à l'aide d'une suite d'instruction :
Toutes les instructions qui sont pas couvertes par des bypass provoquent des cycles de gels.

Q5/ Modifiez le code de la boucle pour qu'il soit executable sur le processeur P9.

Q6/ Analysez l'execution d'une iteration de la boucle avec un schéma simplifié.



Q7/ Calculez le CPI et le CPI_u sachant que 50% des Wbr sont negatif :

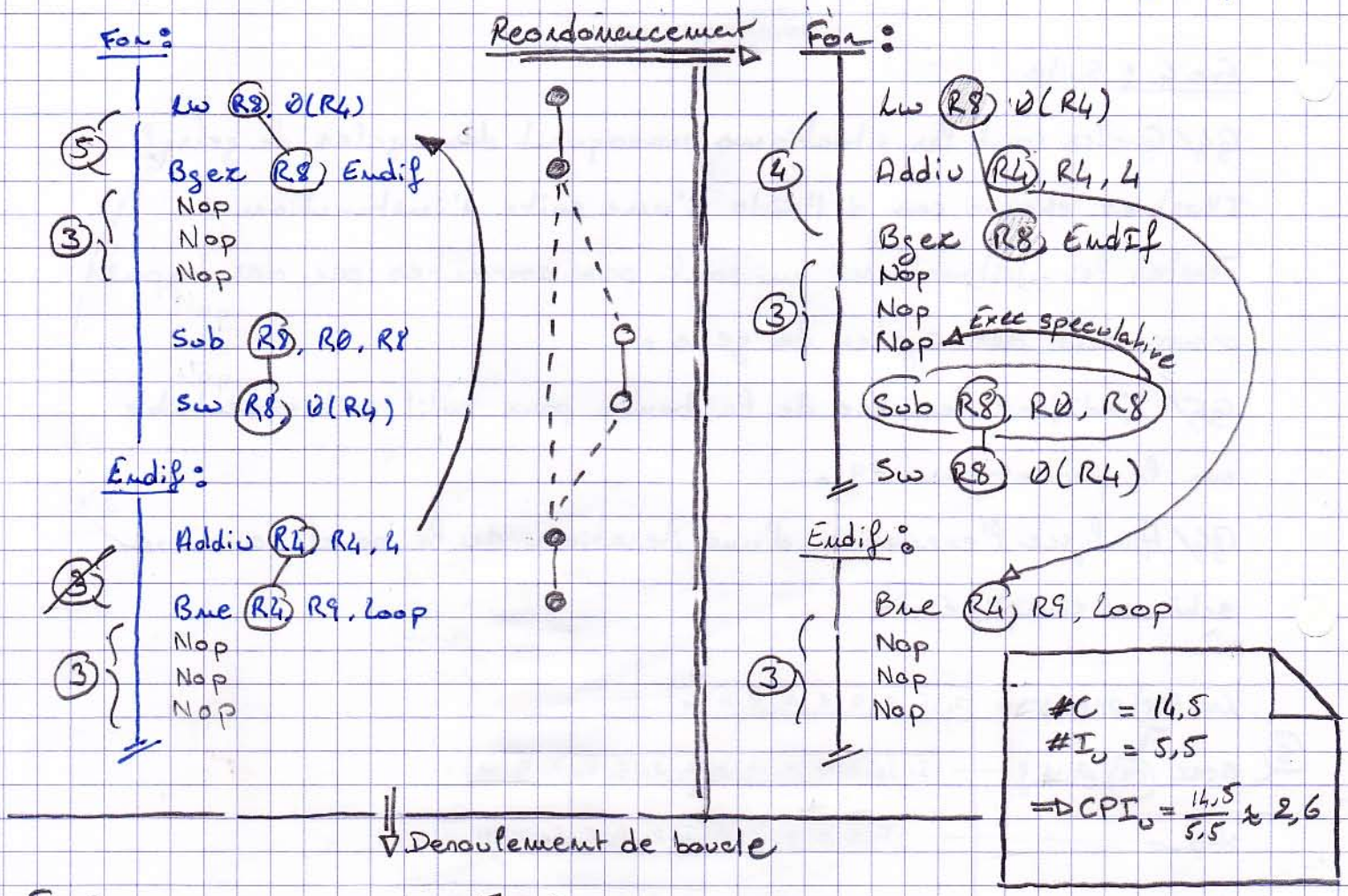
B^t echoue :
- #C = 20
- #I = 12
- $\#I_u$ = 6

B^t reussit :
- #C = 18
- #I = 10
- $\#I_u$ = 4

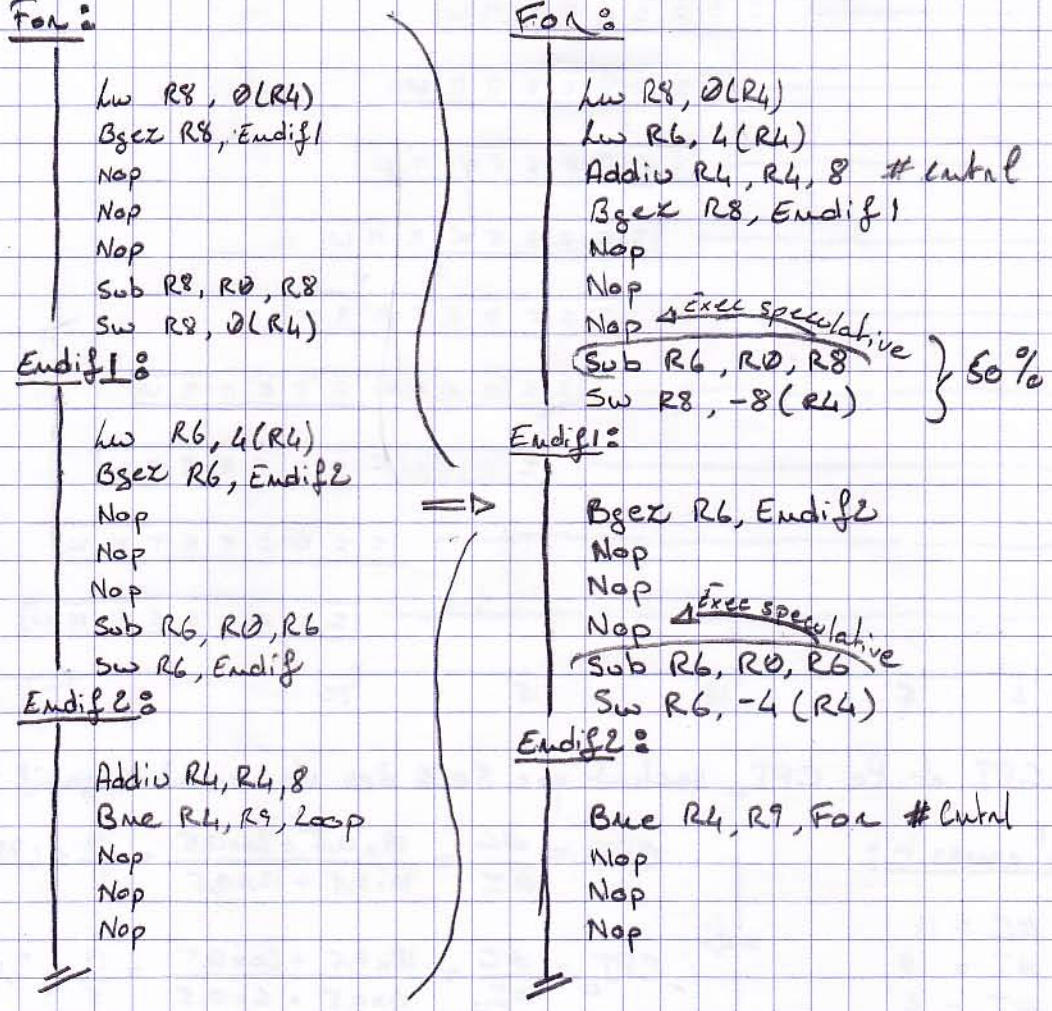
$$CPI = \frac{\#C}{\#I} = \frac{18 \times 0,5 + 20 \times 0,5}{10 \times 0,5 + 12 \times 0,5} = \frac{19}{11} \approx 1,73$$

$$CPI_u = \frac{\#C}{\#I_u} = \frac{18 \times 0,5 + 20 \times 0,5}{4 \times 0,5 + 6 \times 0,5} = \frac{19}{5} \approx 3,8$$

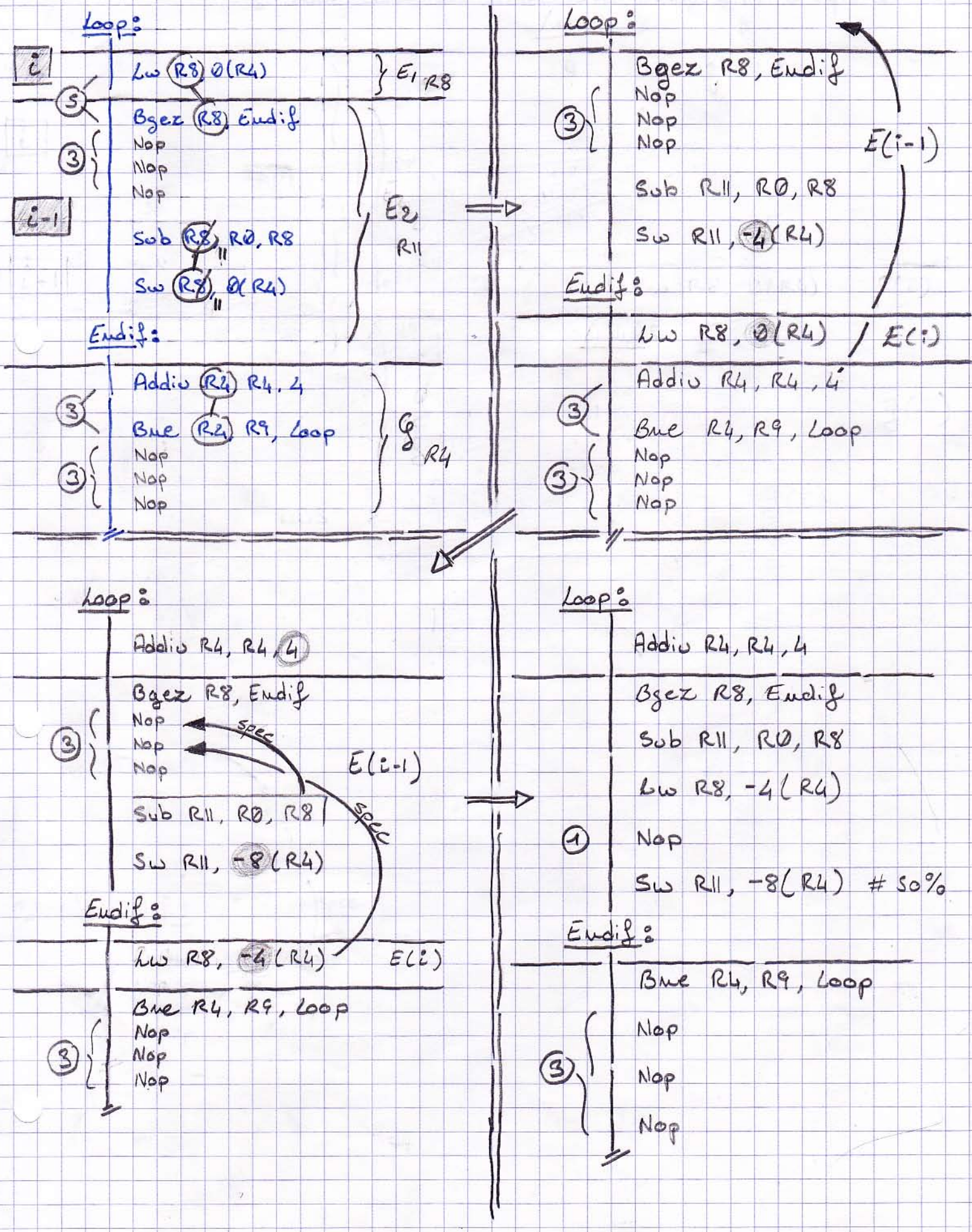
§8/ Réordonnez le code de manière à éviter au max les cycles perdus :



Déroulement de boucle



Exe 4 : Q9/ Optimisez le code par "software pipeliné" puis par spéculation :



Q10/ Le temps de cycle du processeur P9 est egal à 0,6 fois le temps de cycle du processeur Mips-32 pipeliné. Comparez le temps necessaire à l'execution d'une iteration de la boucle entre le Mips et P9. Quelle conclusion en tirez-vous ?