Rapport de TP ARCH

Programmation sur processeur DLX

Table des matières

Introduction

Au cours de ce TP, nous allons implémenter les différents type optimisations logicielles vues en cours sur des processeurs de type DLX à l'aide du simulateur WinDLX. Le code sera pour la plupart implémenté en assembleur.

I. Thème 1

I. 1. Question 1

Fig 1. Nombre de cycle

```
Total:
    6267 Cycle(s) executed.
    ID executed by 4664 Instruction(s).
    2 Instruction(s) currently in Pipeline.
```

On peut voir que le programme exécute 6267 cycles.

Fig 2. Nombre d'aléa

```
Stalls:
    RAW stalls: 1024 (16.34% of all Cycles), thereof:
        LD stalls: 0 (0.00% of RAW stalls)
        Branch/Jump stalls: 0 (0.00% of RAW stalls)
        Floating point stalls: 1024 (100.00% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 0 (0.00% of all Cycles)
    Control stalls: 511 (8.15% of all Cycles)
    Trap stalls: 3 (0.05% of all Cycles)
    Total: 1538 Stall(s) (24.54% of all Cycles)
```

```
multf   f6,f2,f4        ;  5 cycles
sub     r6,r6,#1        ; 2 cycles
add     r2,r2,#4        ;2 c cycles
addf    f0,f0,f6        ; 2 cycles
```

Il y a génèration d'un aléa car on utilise f6 dont le calcul par multf n'est pas fini.

Aléa de type RAW lié à une dépendance de donnée : addf essaie de lire f6 alors que le résultat de celui n'a pas encore été écrit par multf.

I. 2. Question 2

Fig. add début de programme.

```
start:  add     r1,r0,a
        add     r2,r0,b
        lw      r10,n(r0)
        add     r3,r0,c
        add     r4,r0,r10
        lw      r11,TLig(r0)
```

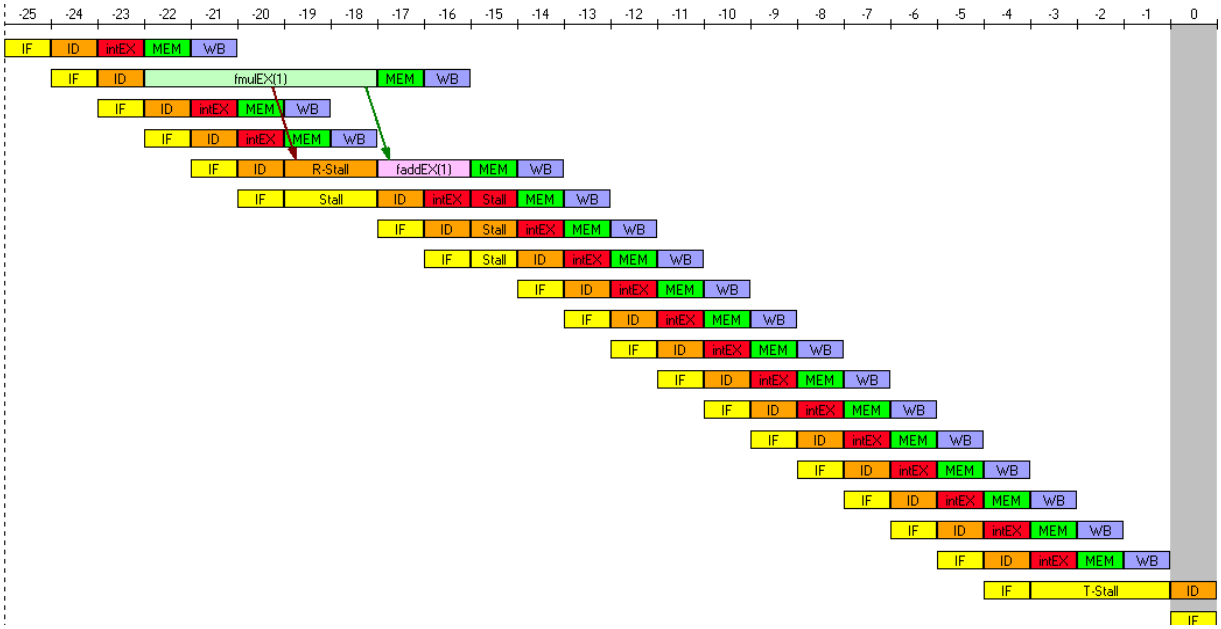
Les add situés en début de programme permettent d'initialiser les registres.

```
bnez        r5,boucle2
            nop
```

Les nop après les instructions de branchement permettent d' éviter les aléas.

I. 3. Question 3

Fig. 3 Diagramme des cycles d'horloge du code orignal.



Nous sommes passés par deux étapes de modification de code, une nous laissait des aléas et la deuxièmes les enlevaient. Nous exposons ici ces deux étapes.

Première étape :

Fig. 4 Partie de code modifiées pour optimiser.

```
boucle2:    add     r6,r0,r10
            lf      f0,t
            lf      f4,0(r2)  ; Chargement des f4 et f2 avant la boucle 3
            lf      f2,0(r1)

boucle3:    multf   f6,f2,f4 ; 5 cycles
            lf      f4,0(r2)  ; Chargement des f4 et f2 apres le mult
            lf      f2,0(r1)
            add     r1,r1,#4
            sub     r6,r6,#1 ; 2 cycles
            add     r2,r2,#4 ;2 c cycles
```

Fig 5. Nobmre de cycle obtenus et aléas

```
Total:
    5947 Cycle(s) executed.
    ID executed by 4792 Instruction(s).
    2 Instruction(s) currently in Pipeline.

Hardware configuration:
    Memory size: 32768 Bytes
    faddEX-Stages: 1, required Cycles: 2
    fmulEX-Stages: 1, required Cycles: 5
    fdivEX-Stages: 1, required Cycles: 19
    Forwarding enabled.

Stalls:
    RAW stalls: 64 (1.08% of all Cycles), thereof:
        LD stalls: 64 (100.00% of RAW stalls)
        Branch/Jump stalls: 0 (0.00% of RAW stalls)
        Floating point stalls: 0 (0.00% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 0 (0.00% of all Cycles)
    Control stalls: 511 (8.59% of all Cycles)
    Trap stalls: 3 (0.05% of all Cycles)
    Total: 578 Stall(s) (9.72% of all Cycles)
```

. *Pourcentage :*

$$Pgain = \frac{6267 - 5947}{6267} * 100$$

$$Pgain = 5{,}1\%$$

Fig 6. Diagramme des cycles d'horloge du code modifié, partie 1

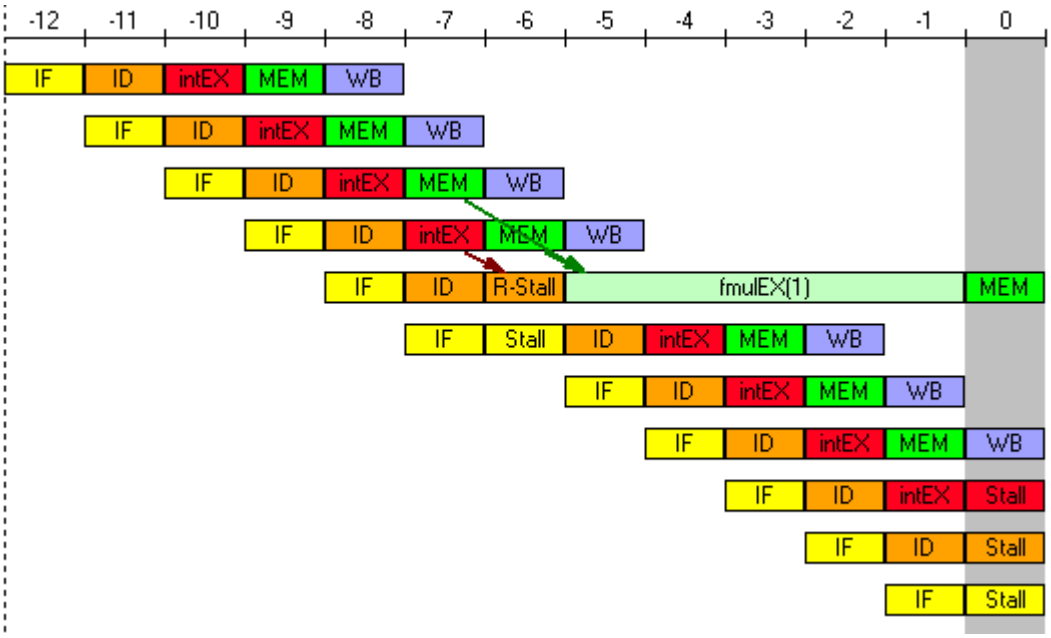


Fig 7. Information sur mul

multf f6,f2,f4	IF	ID
Adr.: boucle3 Code: 0x04443002 In Pipeline-Stage MEM First Cycle: -8 Last Cycle: ??? Total Cycles: >=9	Cycles: -8(1) Terminated successfully IMAR<-PC (=boucle3) IR<-Mem[IMAR] (=0x04443002) PC<-PC+4 (=boucle3+0x4) No Stalls required.	Cycles: -7(2) Terminated successfully A<-F2 (=0) B<-F4 (=0) 1 Stall(s) because of RAW-Hazard (F2) with lf f2,temp(r1)
fmulEX(1)	**MEM**	**WB**
Cycles: -5(5) Terminated successfully ALU<-A*B (=12) (A=3, B=4) No Stalls required. Forwarding applicated: A<-0x40400000 (lf f2,temp(r1)) B<-0x40800000 (lf f4,temp(r2))	Cycles: 0(1) In Pipeline Nothing to do. No Stalls required.	

Deuxième étape :

Fig 8 . Diagramme des cycles d'horloge du code modifié, partie 2

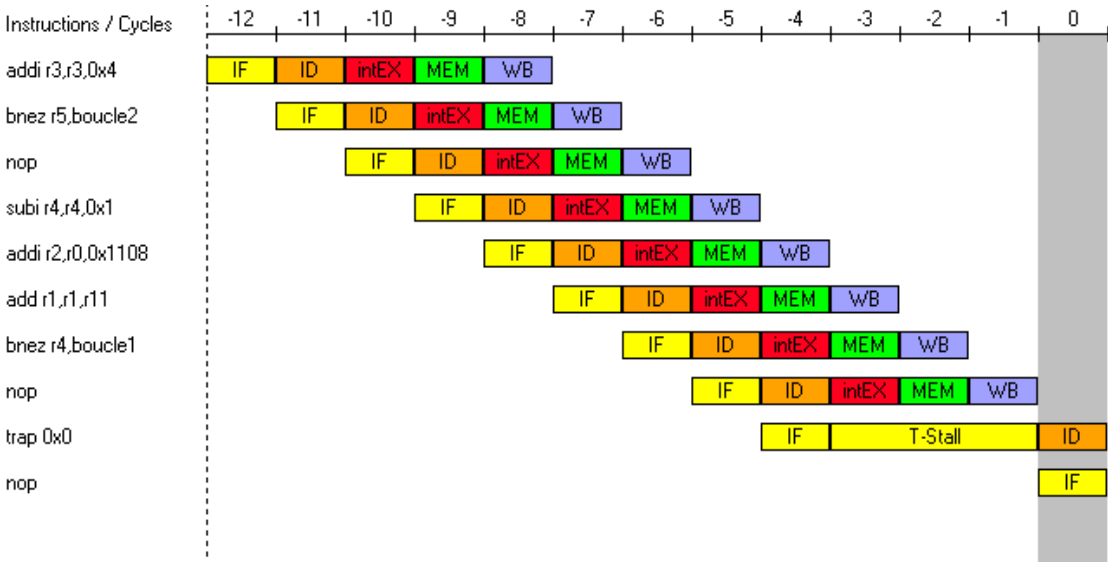


Fig. Code modifié partie 2

```
boucle2:        add     r6,r0,r10
                lf      f0,t
                lf      f4,0(r2)
                lf      f2,0(r1)

boucle3:        add     r1,r1,#4 ; 2 cycles
                multf   f6,f2,f4 ; 5 cycles
                lf      f4,0(r2)
                lf      f2,0(r1)
                sub     r6,r6,#1 ;
                add     r2,r2,#4 ;
                addf    f0,f0,f6  ;
                bnez    r6,boucle3
                nop
```

Fig 9. Nombre de cycle obtenus et aléas

```
Total:
    5371 Cycle(s) executed.
    ID executed by 4792 Instruction(s).
    2 Instruction(s) currently in Pipeline.

Hardware configuration:
    Memory size: 32768 Bytes
    faddEX-Stages: 1, required Cycles: 2
    fmulEX-Stages: 1, required Cycles: 5
    fdivEX-Stages: 1, required Cycles: 19
    Forwarding enabled.

Stalls:
    RAW stalls: 0 (0.00% of all Cycles), thereof:
        LD stalls: 0 (0.00% of RAW stalls)
        Branch/Jump stalls: 0 (0.00% of RAW stalls)
        Floating point stalls: 0 (0.00% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 0 (0.00% of all Cycles)
    Control stalls: 511 (9.51% of all Cycles)
    Trap stalls: 3 (0.06% of all Cycles)
    Total: 514 Stall(s) (9.57% of all Cycles)
```

I. 4. Question 4

Fig. Code demandé

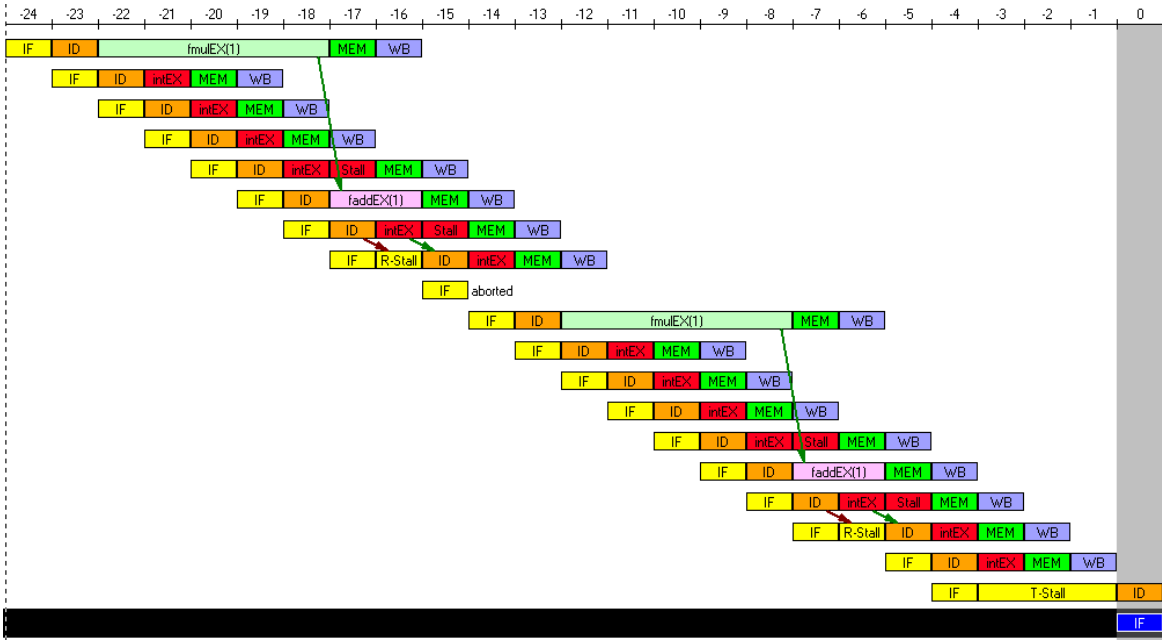
```
start:      lf f3,t(r0) ;initialisation de tmp
            lw r1,n(r0) ;initialisation de i = n
            sub r1,r1,#1
            add r3,r0,a
            add r4,r0,b
            lf  f2,0(r4)
            lf  f4,0(r3)
boucle1:
            multf f6,f4,f2
            lf  f2,0(r4)
            lf  f4,0(r3)
            add r3,r3,#4
            add r4,r4,#4
            addf f3,f3,f6
            sub r1,r1,#1
            bnez r1,boucle1  ; saut contionnel
            nop
            trap      #0
```

Fig 10. Statistique

```
Total:
    84 Cycle(s) executed.
    ID executed by 65 Instruction(s).
    2 Instruction(s) currently in Pipeline.

Hardware configuration:
    Memory size: 32768 Bytes
    faddEX-Stages: 1, required Cycles: 2
    fmulEX-Stages: 1, required Cycles: 5
    fdivEX-Stages: 1, required Cycles: 19
    Forwarding enabled.

Stalls:
    RAW stalls: 9 (10.71% of all Cycles), thereof:
        LD stalls: 2 (22.22% of RAW stalls)
        Branch/Jump stalls: 7 (77.78% of RAW stalls)
        Floating point stalls: 0 (0.00% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 0 (0.00% of all Cycles)
    Control stalls: 6 (7.14% of all Cycles)
    Trap stalls: 3 (3.57% of all Cycles)
    Total: 18 Stall(s) (21.43% of all Cycles)
```

On observe un aléa au niveau de la boucle car l'instruction de décrémentation néccéssite deux cycles et que nous avons placé celle-ci avant l'instruction conditionnelle. D'autre part, pour obtenir le registre r1 au début à n-1 au lieu de n, on génére un alée puisque le chargement prend deux cycles et qu'on tente de soustraire juste après celui-ci. De même pour le chargement juste avant le mult.

Fig 11. Résultats



I. 5. Question 5

On modifie le code pour réduire les aléas

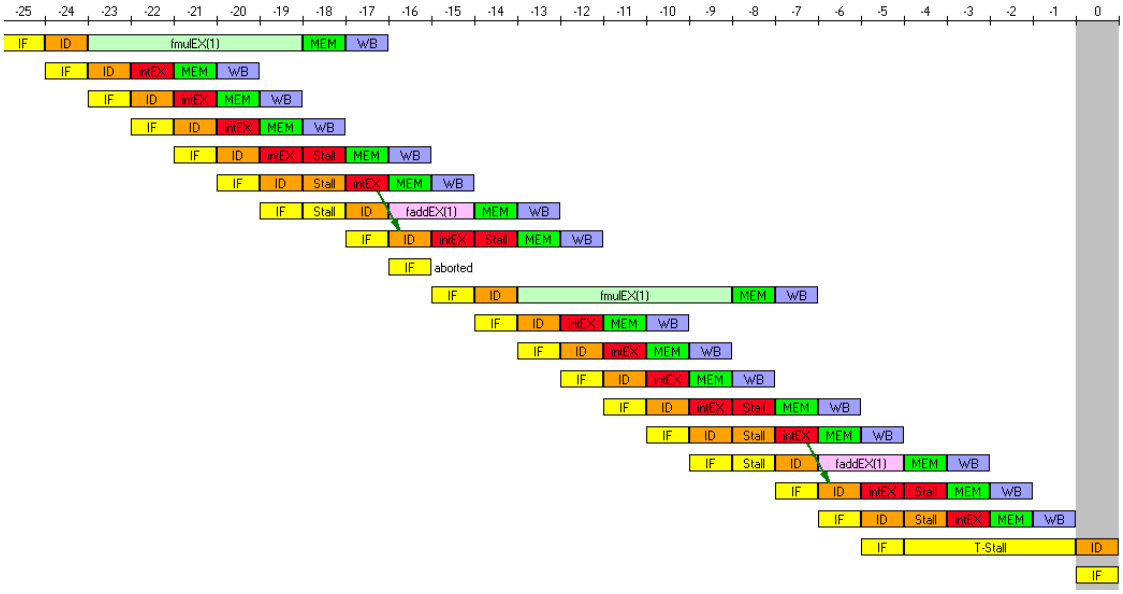
Fig 12. Code obtenu

```
start:      lf f3,t(r0) ;initialisation de tmp
            lw r1,n(r0) ;initialisation de i = n
            add r3,r0,a
            add r4,r0,b
            lf  f2,0(r4)
            lf  f4,0(r3)
            sub r1,r1,#1

boucle1:    multf f6,f4,f2
            lf  f2,0(r4)
            lf  f4,0(r3)
            add r3,r3,#4    ; deplacement dans a
            add r4,r4,#4    ; deplacement dans b
            sub r1,r1,#1
            addf f3,f3,f6   ; tmp = tmp + a[i]*b[i]
            bnez r1,boucle1  ; saut contionnel
            nop
            trap     #0
```

Fig 13. Statistiques

```
Total:
    83 Cycle(s) executed.
    ID executed by 65 Instruction(s).
    2 Instruction(s) currently in Pipeline.

Hardware configuration:
    Memory size: 32768 Bytes
    faddEX-Stages: 1, required Cycles: 2
    fmulEX-Stages: 1, required Cycles: 5
    fdivEX-Stages: 1, required Cycles: 19
    Forwarding enabled.

Stalls:
    RAW stalls: 0 (0.00% of all Cycles), thereof:
        LD stalls: 0 (0.00% of RAW stalls)
        Branch/Jump stalls: 0 (0.00% of RAW stalls)
        Floating point stalls: 0 (0.00% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 0 (0.00% of all Cycles)
    Control stalls: 6 (7.23% of all Cycles)
    Trap stalls: 4 (4.82% of all Cycles)
    Total: 10 Stall(s) (12.05% of all Cycles)
```

Fig 14. Résultats



I. 6. Question 6

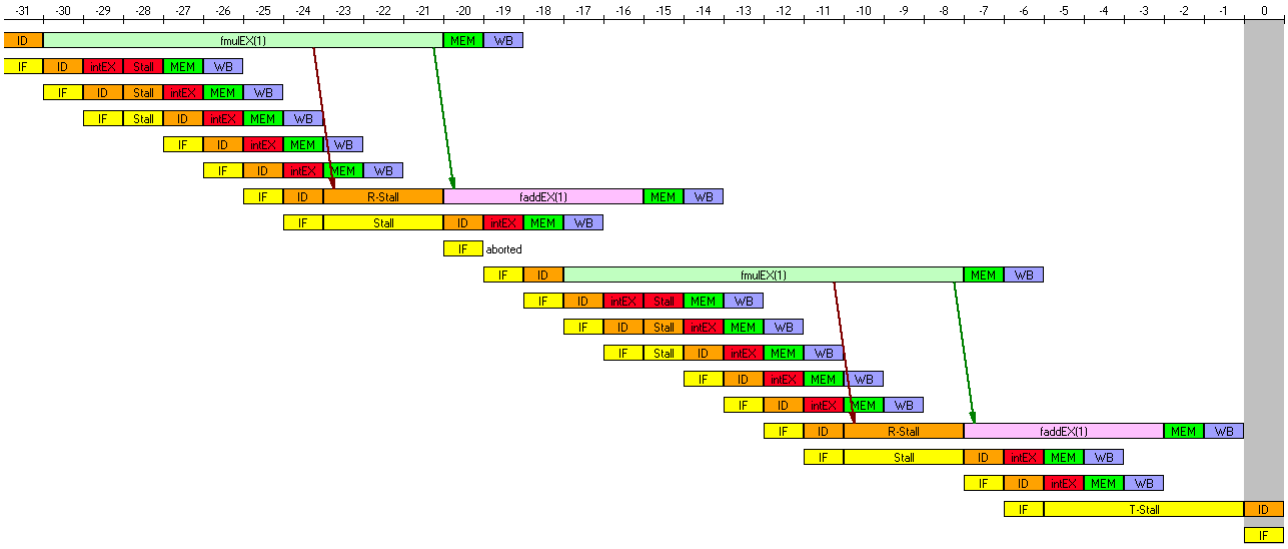
Fig 15. Code optimisé pour P2

```
start:      lf f3,t(r0) ;initialisation de tmp
            lw r1,n(r0) ;initialisation de i = n
            add r3,r0,a
            add r4,r0,b
            lf  f4,0(r3)
            lf  f2,0(r4)
            sub r1,r1,#1 ; n-1

boucle1:    multf f6,f4,f2
            lf  f4,0(r3)      ;
            lf  f2,0(r4)      ;
            add r3,r3,#4    ; deplacement dans a
            add r4,r4,#4    ; deplacement dans b
            sub r1,r1,#1
            addf f3,f3,f6   ; tmp = tmp + a[i]*b[i]
            bnez r1,boucle1  ; saut contionnel tant que > 0
            nop
            trap      #0
```

Fig 16. Statistique du code

```
Total:
    53 Cycle(s) executed.
    ID executed by 33 Instruction(s).
    2 Instruction(s) currently in Pipeline.

Hardware configuration:
    Memory size: 32768 Bytes
    faddEX-Stages: 2, required Cycles: 5
    fmulEX-Stages: 2, required Cycles: 10
    fdivEX-Stages: 1, required Cycles: 19
    Forwarding enabled.

Stalls:
    RAW stalls: 10 (18.87% of all Cycles), thereof:
        LD stalls: 0 (0.00% of RAW stalls)
        Branch/Jump stalls: 0 (0.00% of RAW stalls)
        Floating point stalls: 10 (100.00% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 0 (0.00% of all Cycles)
    Control stalls: 2 (3.77% of all Cycles)
    Trap stalls: 5 (9.43% of all Cycles)
    Total: 17 Stall(s) (32.08% of all Cycles)

Conditional Branches):
    Total: 3 (9.09% of all Instructions), thereof:
        taken: 2 (66.67% of all cond. Branches)
        not taken: 1 (33.33% of all cond. Branches)

Load-/Store-Instructions:
    Total: 10 (30.30% of all Instructions), thereof:
        Loads: 10 (100.00% of Load-/Store-Instructions)
        Stores: 0 (0.00% of Load-/Store-Instructions)

Floating point stage instructions:
    Total: 6 (18.18% of all Instructions), thereof:
        Additions: 3 (50.00% of Floating point stage inst.)
        Multiplications: 3 (50.00% of Floating point stage inst.)
        Divisions: 0 (0.00% of Floating point stage inst.)

Traps:
    Traps: 1 (3.03% of all Instructions)
```

Fig 17. Résultat



I. 7. Question 7

Fig. 18 Code avec déroulage de boucle 2

```
start:      lf f3,t(r0) ;initialisation de tmp
            lw r1,n(r0) ;initialisation de i = n
            add r3,r0,a
            add r4,r0,b
            lf  f4,0(r3)
            lf  f2,0(r4)

            lf  f8,-4(r3)     ;
            lf  f10,-4(r4)    ;

            sub r1,r1,#2

boucle1:    multf f6,f4,f2
            multf f11,f8,f10

            lf  f4,0(r3)      ;
            lf  f2,0(r4)      ;

            lf  f8,-4(r3)     ; deplacement des indices
            lf  f10,-4(r4)    ;

            add r3,r3,#4      ; deplacement dans a
            add r4,r4,#4      ; deplacement dans b

            addf f3,f3,f6     ; tmp = tmp + a[i]*b[i]
            sub r1,r1,#2      ; deplacement pas de 2
            addf f3,f3,f11    ; tmp = tmp + a[i]*b[i]

            bnez r1,boucle1   ; saut contionnel
            nop
            trap      #0
```

Fig. 19 Résultats

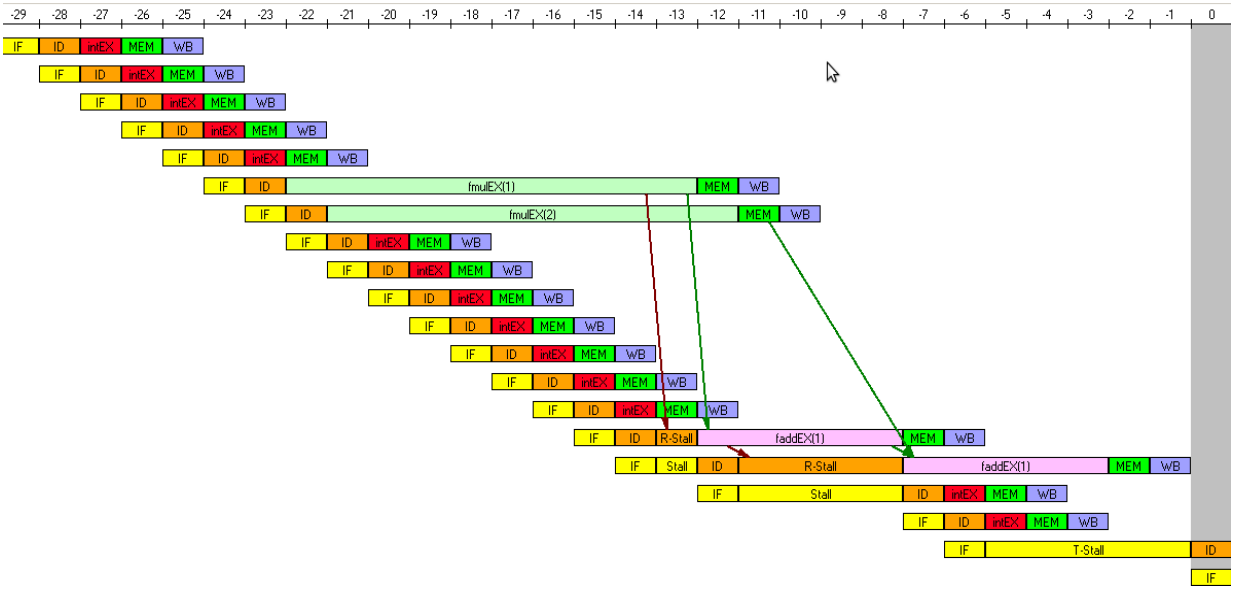


Fig 20. Statistiques

```
Total:
    34 Cycle(s) executed.
    ID executed by 23 Instruction(s).
    2 Instruction(s) currently in Pipeline.

Hardware configuration:
    Memory size: 32768 Bytes
    faddEX-Stages: 2, required Cycles: 5
    fmulEX-Stages: 2, required Cycles: 10
    fdivEX-Stages: 1, required Cycles: 19
    Forwarding enabled.

Stalls:
    RAW stalls: 5 (14.70% of all Cycles), thereof:
        LD stalls: 0 (0.00% of RAW stalls)
        Branch/Jump stalls: 0 (0.00% of RAW stalls)
        Floating point stalls: 5 (100.00% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 0 (0.00% of all Cycles)
    Control stalls: 0 (0.00% of all Cycles)
    Trap stalls: 5 (14.70% of all Cycles)
    Total: 10 Stall(s) (29.41% of all Cycles)

Conditional Branches):
    Total: 1 (4.35% of all Instructions), thereof:
        taken: 0 (0.00% of all cond. Branches)
        not taken: 1 (100.00% of all cond. Branches)

Load-/Store-Instructions:
    Total: 10 (43.48% of all Instructions), thereof:
        Loads: 10 (100.00% of Load-/Store-Instructions)
        Stores: 0 (0.00% of Load-/Store-Instructions)

Floating point stage instructions:
    Total: 4 (17.39% of all Instructions), thereof:
        Additions: 2 (50.00% of Floating point stage inst.)
        Multiplications: 2 (50.00% of Floating point stage inst.)
        Divisions: 0 (0.00% of Floating point stage inst.)

Traps:
    Traps: 1 (4.35% of all Instructions)
```

Une addition durant 5 cycles et une multiplication durant 10 cycles, il nous est difficile de optimiser le code plus que cela : il faudrait avoir des instructions à insérer pour être calculés aux moments problématiques demandant un temps avant d'accéder aux données, cependant le programme étant relativement court nous n'avons plus d'instruction pouvant être placées, l'ajout de nop n'étant pas une solution puisqu'augmentant le nombre de cycle. On observe cependant que le déroulage de boucle nous fais gagner des cycles puisqu'il réduit du nombre d'instructions utilisées pour la boucle.

Fig. 21 Code avec déroulage de boucle 8

```
start:      lf f3,t(r0) ;initialisation de tmp
            lw r1,n(r0) ;initialisation de i = n
            add r3,r0,a
            add r4,r0,b
            lf  f4,0(r3)       ;
            lf  f2,0(r4)       ;
            lf  f8,-4(r3)      ;
            multf f6,f4,f2     ;
            lf  f10,-4(r4)     ; 1
            lf  f12,-8(r3)     ; 2
            multf f11,f8,f10   ; 3
            lf  f13,-8(r4)     ; 4
            lf  f14,-12(r3)    ; 5
            lf  f15,-12(r4)    ; 6
            lf  f16,-16(r3)    ; 7
            lf  f17,-16(r4)    ; 8
            lf  f18,-20(r3)    ; 9
            lf  f19,-20(r4)    ; 10
            multf f24,f12,f13 ; 0
            addf f3,f3,f6      ; tmp = tmp + a[0]*b[0] ; 1
            lf  f20,-24(r3)    ; 2
            multf f25,f14,f15 ; 3
            addf f3,f3,f11     ; tmp = tmp + a[1]*b[1] ; 4
            lf  f21,-24(r4)    ; 5
            lf  f22,-28(r3)    ; 6
            lf  f23,-28(r4)    ; 7
            ;nop              ; 8
            ;nop              ; 9
            ;nop              ; 10
            multf f26,f16,f17 ; 0
            addf f3,f3,f24     ; tmp = tmp + a[2]*b[2] ; 1
            ;nop              ; 2
            multf f27,f18,f19 ; 3
            addf f3,f3,f25     ; tmp = tmp + a[3]*b[3] ; 4
            ;nop              ; 5
            ;nop              ; 6
            ;nop              ; 7
            ;nop              ; 8
            ;nop              ; 9
            ;nop              ; 10
            multf f28,f20,f21 ; 0
            addf f3,f3,f26     ; tmp = tmp + a[4]*b[4] ; 1
```

```
;nop              ; 2
multf f29,f22,f23 ; 3
addf f3,f3,f27    ; tmp = tmp + a[5]*b[5] ; 4
;nop              ; 5
;nop              ; 6
;nop              ; 7
;nop              ; 8
;nop              ; 9
;nop              ; 10
addf f3,f3,f28    ; tmp = tmp + a[6]*b[6] ;0
;nop              ; 1
;nop              ; 2
addf f3,f3,f29    ; tmp = tmp + a[7]*b[7] ; 3

trap      #0
```

On a placé les nop en commentaire pour observer le nombre d'aléa provoqué suite à la non disponibilité des ressources et au manque d'instruction pouvant être intercallées permettant d'éviter ces aléas.

Fig 22. Résultats

```
Total:
    65 Cycle(s) executed.
    ID executed by 37 Instruction(s).
    2 Instruction(s) currently in Pipeline.

Hardware configuration:
    Memory size: 32768 Bytes
    faddEX-Stages: 2, required Cycles: 5
    fmulEX-Stages: 2, required Cycles: 10
    fdivEX-Stages: 1, required Cycles: 19
    Forwarding enabled.

Stalls:
    RAW stalls: 14 (21.54% of all Cycles), thereof:
        LD stalls: 0 (0.00% of RAW stalls)
        Branch/Jump stalls: 0 (0.00% of RAW stalls)
        Floating point stalls: 14 (100.00% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 5 (7.69% of all Cycles)
    Control stalls: 0 (0.00% of all Cycles)
    Trap stalls: 11 (16.92% of all Cycles)
    Total: 30 Stall(s) (46.15% of all Cycles)

Conditional Branches):
    Total: 0 (0.00% of all Instructions), thereof:
        taken: 0 (0.00% of all cond. Branches)
        not taken: 0 (0.00% of all cond. Branches)

Load-/Store-Instructions:
    Total: 18 (48.65% of all Instructions), thereof:
        Loads: 18 (100.00% of Load-/Store-Instructions)
        Stores: 0 (0.00% of Load-/Store-Instructions)

Floating point stage instructions:
    Total: 16 (43.24% of all Instructions), thereof:
        Additions: 8 (50.00% of Floating point stage inst.)
        Multiplications: 8 (50.00% of Floating point stage inst.)
        Divisions: 0 (0.00% of Floating point stage inst.)

Traps:
    Traps: 1 (2.70% of all Instructions)
```

On observe un nombre de cycle plus important que le déroulage de boucle par 2 suite à la complexité du déroulage manuel et des aléas générés par le mauvais ordonnancement des instructions. On peut donc conclure que sur ce proccesseur, un déroulage de boucle important n'amène aucune optimisation.

I. 8. Question 8

Avec un multiplieur pipeline, la limitation d'entrée n'est pas effective, le démarrage mettrait autant de temps que lors des précédentes question mais au fur et à mesure, le délais de sortie des informations serait réduit permettant avec un déroulage de boucle de réduire les cycles d'éxecution.

Conclusion

Au cours de ce TP nous avons pus mettre en pratique le cours dispensé sur l'optimisation et nous avons pus nous rendre compte à quel point l'optimisation dépendait non seulement des ressources matérielles sur lequel on implémentait nos algorithmes mais aussi des différents choix d'implémenations mis en oeuvre.