Third Lab Assignment:
Instruction Level Parallelism

2. Procedure

2.1 Simple execution, without data forwarding techniques

f)

Clock cycles	174	**Stalls: - Data**	101
Instructions	61	**- Structural**	0
Average CPI	2.852	**- Branch Taken**	8

g)

A branch prediction policy que é adotada pelo simulador WinMIPS64 consiste em static branch prediction, ou seja, não tem em conta o histórico das branch predictions e executa sempre a mesma previsão, dadas as mesmas circunstâncias. Conseguimos perceber isso a partir da execução do programa, já que o número de RAW Stalls é muito elevado, coisa que não acontece quando ativamos a opção de Data Forwarding. Nesse caso, a branch prediction policy já seria dinâmica.

2.2 Application of data forwarding techniques

c)

Clock cycles	136	**Stalls: - Data**	63
Instructions	61	**- Structural**	9
Average CPI	2.230	**- Branch Taken**	8

d)

- $Speedup = \frac{CPI_{old}}{CPI_{new}} = \frac{2.852}{2.230} = 1{,}278924$

2.3 Application of data forwarding techniques

a) Attach a copy of the new assembly program.

```
loop:   lw      $12, 0($1)      ; $12 = A[i]
        daddi   $5, $5, 1       ; i++
        dmul    $12, $12, $9    ; $12 = $12*$9 ;; $12 = A[i]*mult
        daddi   $1, $1, 8       ;
        dadd    $9, $9, $12     ; $9 = $9 + $12  ;; mult = mult + A[i]*mult

        bne     $6, $5, loop   ; Exit loop if i == N

        sw      $9, mult($0)   ; Store result
        halt
```

c)

Clock cycles	118	**Stalls: - Data**	36
Instructions	61	**- Structural**	9
Average CPI	1.934	**- Branch Taken**	8

d)

- $Speedup = \frac{CPI_{old}}{CPI_{new}} = \frac{2.852}{1.934} =$

2.4 Source code optimization: loop unrolling

a) Attach a copy of the new assembly program.

```
loop:   lw      $12, 0($1)      ; $12 = A[i]
        daddi   $5, $5, 1       ; i++
        dmul    $12, $12, $9    ; $12 = $12*$9 ;; $12 = A[i]*mult
        daddi   $1, $1, 8       ;
        dadd    $9, $9, $12     ; $9 = $9 + $12 ;; mult = mult + A[i]*mult

        lw      $12, 0($1)      ; $12 = A[i]
        daddi   $5, $5, 1       ; i++
        dmul    $12, $12, $9    ; $12 = $12*$9 ;; $12 = A[i]*mult
        daddi   $1, $1, 8       ;
        dadd    $9, $9, $12     ; $9 = $9 + $12 ;; mult = mult + A[i]*mult

        lw      $12, 0($1)      ; $12 = A[i]
        daddi   $5, $5, 1       ; i++
        dmul    $12, $12, $9    ; $12 = $12*$9 ;; $12 = A[i]*mult
        daddi   $1, $1, 8       ;
        dadd    $9, $9, $12     ; $9 = $9 + $12 ;; mult = mult + A[i]*mult

        bne     $6, $5, loop    ; Exit loop if i == N

        sw      $9, mult($0)    ; Store result
        halt
```

c)

Clock cycles	106	**Stalls: - Data**	36
Instructions	55	**- Structural**	9
Average CPI	1.927	**- Branch Taken**	2

d)

- $Speedup = \frac{CPI_{old}}{CPI_{new}} = \frac{2.852}{1.927} =$

2.5 Source code optimization: branch delay slot

a) Attach a copy of the new assembly program.

```
loop:   lw      $12, 0($1)      ; $12 = A[i]
        daddi   $5, $5, 1       ; i++
        dmul    $12, $12, $9    ; $12 = $12*$9 ;; $12 = A[i]*mult

        dadd    $9, $9, $12     ; $9 = $9 + $12 ;; mult = mult + A[i]*mult

        bne     $6, $5, loop    ; Exit loop if i == N
        daddi   $1, $1, 8       ;

        sw      $9, mult($0)    ; Store result
        halt
```

d)

Clock cycles	119	**Stalls: - Data**	45
Instructions	61	**- Structural**	9
Average CPI	1.951	**- Branch Taken**	0

e)

- $Speedup = \frac{CPI_{old}}{CPI_{new}} = \frac{2.852}{1.951} =$