**Q1.5**

Inicio do vetor a[] - 0x10000000

Inicio do vetor b[] - 0x10000030

Valor em x11 - 0x10000000

Valor em x13 - 0x00000030

Os valores são diferentes pois a instrução addi x13, x11, 48 tem dependência da instrução addi x11, x3, 0 logo quando addi x13, x11, 48 é executada x11 ainda tem o valor zero pois a instrução anterior não foi completada, logo em vez de x13 ter o valor 0x10000030 tem o valor 0x00000030

**Q1.6**

1º conflito  
Escreve: addi x11, x3, 0  
Lê: addi x13, x11, 48  
Operando problemático: x11

2º conflito  
Escreve: addi x13, x11, 48  
Lê: addi x12, x13, -4  
Operando problemático: x13

3º conflito  
Escreve: li x16, 0  
Lê: add x20, x13, x16  
Operando problemático: x16

4º conflito  
Escreve: add x20, x13, x16  
Lê: lw x21, 0(x20)  
Operando problemático: x20

5º conflito  
Escreve: lw x21, 0(x20)  
Lê: blez x21, end  
Operando problemático: x21

6º e 7º conflito  
Escreve: lw x22, 0(x11) e lw x23, 0(x12)  
Lê: add x22, x22, x23  
Operandos problemáticos: x22 e x23

8º conflito  
Escreve: add x22, x22, x23  
Lê: mul x15, x15, x22  
Operando problemático: x22

9º conflito  
Escreve: sub x22, x12, x11  
Lê: srai x22, x22, 2  
Operando problemático: x22

10º conflito  
Escreve: srai x22, x22, 2  
Lê: add x14, x14, x22  
Operando problemático: x22

**Q1.8**

Após fazermos as seguintes alterações (a vermelho) ao código:

.text

addi x11, x3, 0 # x11 - a's left index

lw x14, 100(x3) # x14 - n - index distance accumulator (Trocar a posição)

lw x15, 96(x3) # x15 – x (Trocar a posição)

addi x13, x11, 48 # x13 - b's left index

li x16, 0 # x16 - i (Trocar a posição)

nop

addi x12, x13, -4 # x12 - a's right index

Os valores de x11 e de x13 já estavam iguais ao início dos vetores pois agora as instruções têm espaço para serem acabadas logo não dão origem a conflitos de dados

**Q1.11**

202 ciclos

182 instruções executadas

**Q1.12**

71 instruções não uteis

111 instruções uteis

Rácio: 0.61

A execução do programa é pouco eficiente já que mais de um terço do tempo de execução é usado a fazer instruções não uteis.

**Q1.13**

Static branch prediction: predict not taken.  
Pois testámos e vimos que o simulador começa sempre a executar as instruções como se não houvesse nenhum tipo de branch e depois se o branch mudar o pc as instruções que estão na pipeline levam flush

**Q2.3**

lw x21, 0(x20) # x21 = b[i]

nop

nop

blez x21, end # if b[i] <= 0 end the loop

Introduzimos 2 nop’s entre o lw e o blez para fazer stall da operação lw, já que é conflito que não é resolvido por fowarding

lw x22, 0(x11) # x22 = a[i]

lw x23, 0(x12) # x23 = a[N-1-i]

nop

addi x16, x16, 4 # i++ (trocamos a poisção)

add x22, x22, x23 # x22 = a[i] + a[N-1-i]

Alem disso, para resolver outro conflito que não é resolvido por fowarding colocámos 1 nop entre o lw e add, e para rentabilizar a utilização do stall trocámos 1 instrução de sítio.

**Q2.4**

157 ciclos

137 instruções executadas

**Q2.5**

26 instruções não uteis

111 instruções uteis

Rácio: 0.81

A execução do programa é mais eficiente do que a anterior, mas mesmo assim deixa a desejar já que pouco menos que um quinto do tempo de execução é gasto com instruções não uteis.

**Q2.6**

Já que a frequência do processador é a mesma o Speedup será dado pelo número de ciclos de processador só com pipeline (202) a dividir pelo número de ciclos de processador com pipeline e fowarding (157). Logo o SpeedUp é 1.29

**Q3.3**

Não introduzimos nenhumas alterações pois não havia conflitos que tivessem de ser corrigidos.

**Q3.4**  
148 ciclos

111 instruções executadas

**Q3.5**

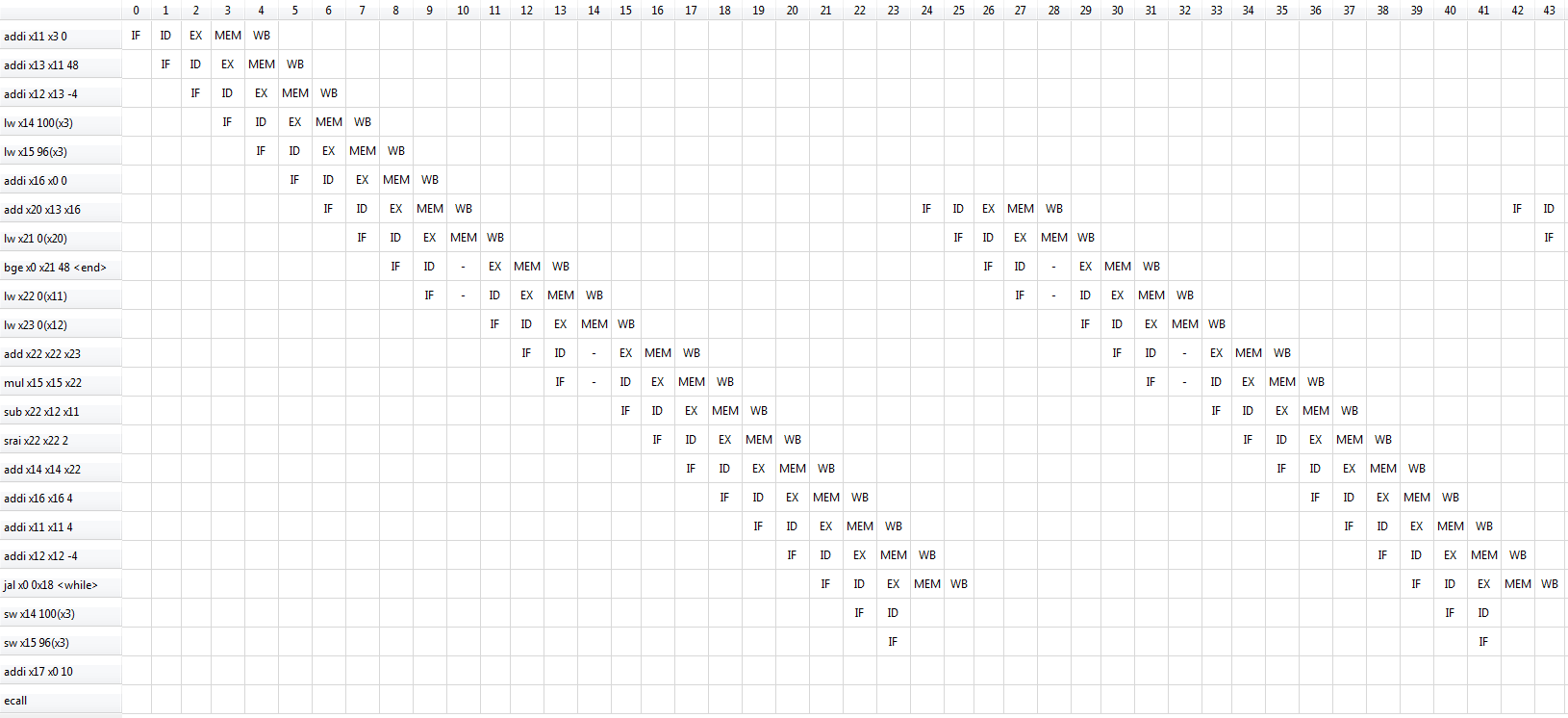
0 instruções não uteis

111 instruções uteis

Rácio: 1

A execução deste programa neste processador é extremamente eficiente já que não existem instruções não uteis.

**Q3.6**  
IPC = 111/148 = 0.75

Este número é menor que 1 pois existem Stalls derivados de conflitos de dados e alem disso existem 4 ciclos de relógio iniciais em que não são finalizadas nenhumas instruções e perdem-se instruções (que levam flush) durante as iterações do while (conflitos de controlo).   
  
**Q3.9**



**Q3.10**

Stall de conflitos de dados RAW: 4 assinalados com



Stall de conflitos de controlo: 2 assinalados com



**Q3.11**   
Tal como justificado anteriormente na pergunta Q3.6 a métrica IPC não é 1 pois existem ciclos de relógio em que tem de ser efetuados Stalls devido a conflitos

**Q4.2**

while:

add x20, x13, x16 # x20 = &b[i]

lw x21, 0(x20) # x21 = b[i]

lw x22, 0(x11) # x22 = a[i] (Trocamos a ordem e colocamos mais em cima pois torna o stall útil e prepara um variável que será usada mais tarde)

blez x21, end # if b[i] <= 0 end the loop

lw x23, 0(x12) # x23 = a[N-1-i]

addi x16, x16, 4 # i++ (Trocamos a ordem e colocamos mais em cima pois evita um stall em que não se faz nada)

add x22, x22, x23 # x22 = a[i] + a[N-1-i]

mul x15, x15, x22 # x15 = x15\*x22 (x \*= a[i] + a[N-1-i])

**Q4.3**

134 ciclos

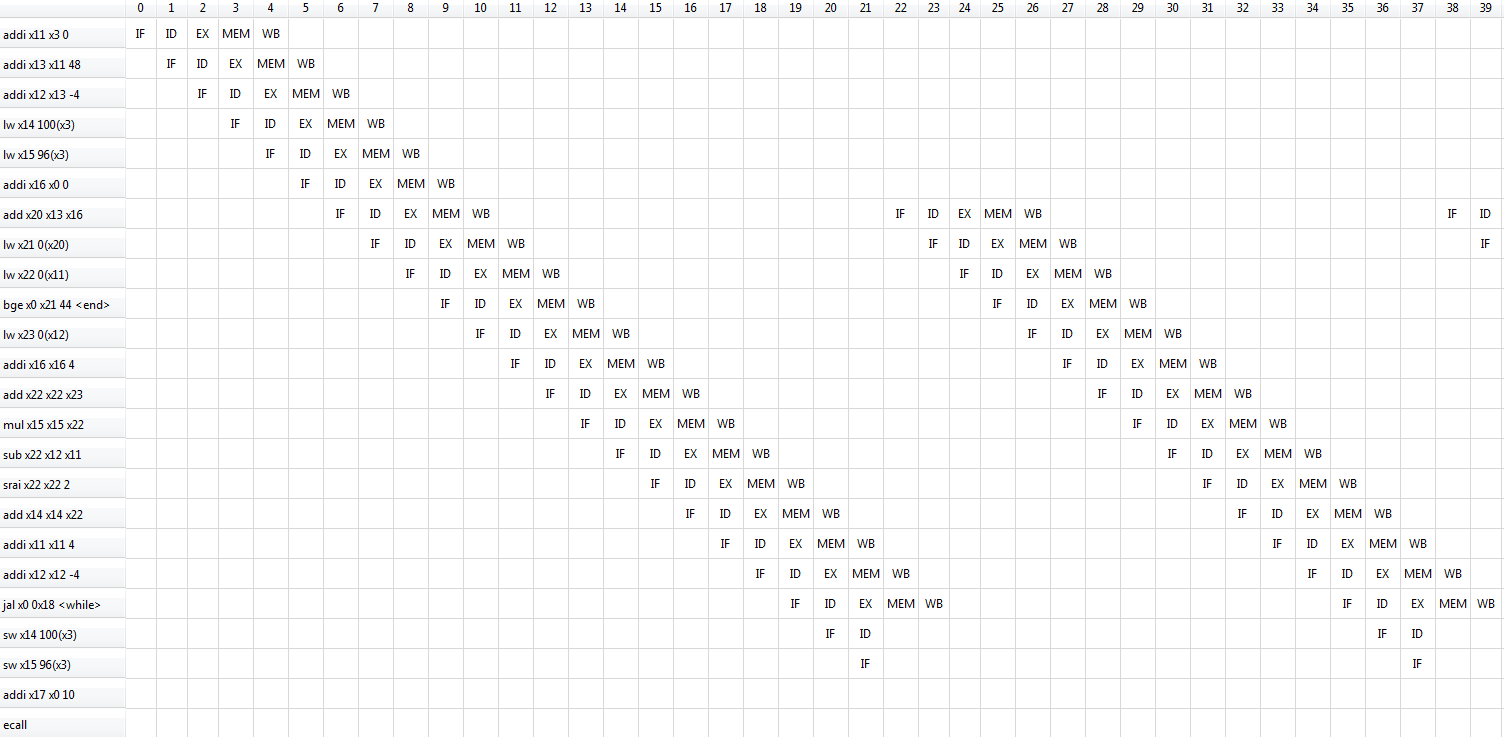
112 instruções executadas

**Q4.4**

IPC = 112/134 = 0.836

O IPC aumentou em relação ao calculado anteriormente

**Q4.7**





**Q4.8**

Stall de conflitos de dados RAW: 0

Stall de conflitos de controlo: 2 assinalados com



**Q4.9** Dados o tipo de política de predição de salto adotada por este simulador vão sempre existir stalls de conflitos de controlo logo o IPC nunca será bem 1

**Q5.2**

Com loop unrolling

62 ciclos

50 instruções executadas

Sem loop unrolling

102 ciclos

82 instruções executadas

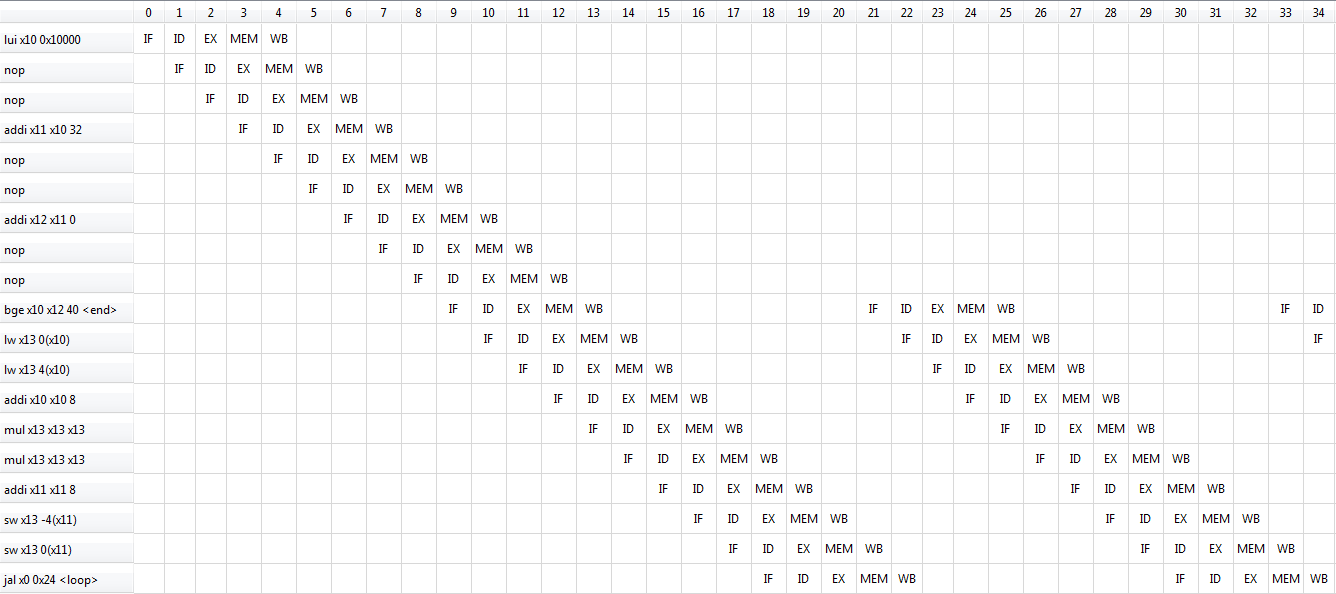
**Q5.3**

IPC (com loop unrolling) = 50/62 = 0.806

IPC (sem loop unrolling) = 82/102 = 0.804

Ficou ligeiramente superior mais por muito pouco

**Q5.6**



**Q5.7**

Devido ao tipo de predição do processador (static predict not taken) foram introduzidas 2 bolhas após o primeiro jal devido ao conflito de dados gerado.

**Q5.8**

Tal como justificado anteriormente o IPC será menor que 1 pois existem stalls derivados de conflitos de controlo e alem disso existem 4 ciclos de relógio iniciais em que não são finalizadas nenhumas instruções logo não serão executadas o máximo de instruções possíveis por ciclo.

**Q5.9**

Já que a frequência do processador é a mesma o speedup é dado pelo número de ciclos sem loop unrolling a dividir pelo número de ciclos com loop unrolling ou seja:

102/62=1.65