Q1.5 -> Os valores no registo x11 mantém-se inalterado, enquanto o valor de x13 é alterado de 0x10000030h para 0x0000030h. Isto deve-se, pois, a instrução que definirá o valor de x13 (addi x13, x11, 48) contém um registo que ainda não foi escrito na memória (registo x11) e isso leva a um conflito de dados. Assim, para a execução desta instrução o valor usado para x11 é o valor que estava na memória naquele momento (valor com que x11 é iniciado no programa que é 0x0000000h).

Q1.6

Instrução que escreve	**Instrução que lê**	**Operando que provoca conflito**
addi x11, x3, 0	addi x13, x11, 48	X11
addi x13, x11, 48	addi x12, x13, -4	X13
add x20, x13, x16	lw x21, 0(x20)	X20
lw x22, 0(x11)	add x22, x22, x23	X22
lw x23, 0(x12)	add x22, x22, x23	X23
add x22, x22, x23	mul x15, x15, x22	X22
sub x22, x12, x11	srai x22, x22, 2	X22
sub x22, x12, x11	add x14, x14, x22	X22
srai x22, x22, 2	add x14, x14, x22	X22
addi a7, x0, 10	ecall	X17

Q1.8 -> Depois da resolução dos conflitos de dados antes do ciclo “while” o valor dos dois registos mantém-se iguais nas duas arquiteturas. Isto já era expectável visto que se não há conflitos, então a mudança entre os dois não deve alterar os valores.

Q1.11

Número de ciclos de relógio: 145

Número de instruções executadas: 125

Q1.12 -> $\frac{\text{Instruções úteis}}{\text{instruções totais}} = \frac{115}{125} = 92\%$ de eficiência

Q1.13 -> A política de execução de salto é a política de predict not taken, ou seja, o ripes assume sempre que no IF a instrução não é um branch. Assim, quando este chegar ao exe (onde está o controlador de branch) caso haja um branch ele descarta as duas instruções novas que entraram (dá flush) e salta para o pc indicado pelo branch.

Q2.3 -> As alterações feitas foram:

➔ Reorganização do código nas linhas 41, 42 e 54, pois evita ter de se colocar mais nop. Isto deve-se, pois, o x21 tem de ter o valor guardado para fazer o branch e este só é guardado na fase da mem e, portanto, só quando x21 está no wb é que o branch pode ser executado.

➔ Adição de um nop na linha 57 para pois para o ecall o valor de x17 tem de estar a 10, logo precisa de três instruções pelo meio.

```
29 # Program section
30       .text
31                                  # NOTE: Upon start, the Global-Pointer
32       addi    x11, x3, 0         # x11 - a's left index
33       addi    x13, x11, 48       # x13 - b's left index
34       addi  x12, x13, -4      # x12 - a's right index
35       lw    x14, 100(x3)      # x14 - n - index distance accumulator
36       lw    x15, 96(x3)         # x15 - x
37       li    x16, 0            # x16 - i
38
39 while: add   x20, x13, x16       # x20 = &b[i]
40       lw    x21, 0(x20)         # x21 = b[i]
41       lw    x22, 0(x11)         # x22 = a[i]
42       lw    x23, 0(x12)         # x23 = a[N-1-i]
43       blez  x21, end         # if b[i] <= 0 end the loop
44       add   x22, x22, x23       # x22 = a[i] + a[N-1-i]
45       mul   x15, x15, x22       # x15 = x15*x22 (x *= a[i] + a[N-1-i])
46       sub   x22, x12, x11       # x22 = 4*((N-1-i)-i)
47       srai  x22, x22, 2         # x22 = x22/4
48       add   x14, x14, x22       # n += x22
49       addi  x16, x16, 4         # i++
50       addi  x11, x11, 4
51       addi  x12, x12, -4
52       jal   x0, while
53
54 end:   addi  a7, x0, 10
55       sw    x14, 100(x3)      # store n's final value
56       sw    x15, 96(x3)         # store x's final value
57       nop
58       ecall
59
```

Q2.4

Número de ciclos de relógio: 134

Número de instruções executadas: 114

Q2.5 -> $\frac{Instruções\ úteis}{instruções\ \ totais} = \frac{113}{114} = 99.1\ de\ eficiência.$ Comparando este valor com o da pergunta 1.12 podemos concluir que esta arquitetura é mais eficiente.

Q2.6 -> $\text{Speedup} = \frac{\text{Ciclos antigos}}{\text{Ciclos novos}} = \frac{145}{134} = 1.08$

Q3.3 -> Nenhuma, pois todos os problemas foram resolvidos por hardware.

Q3.4

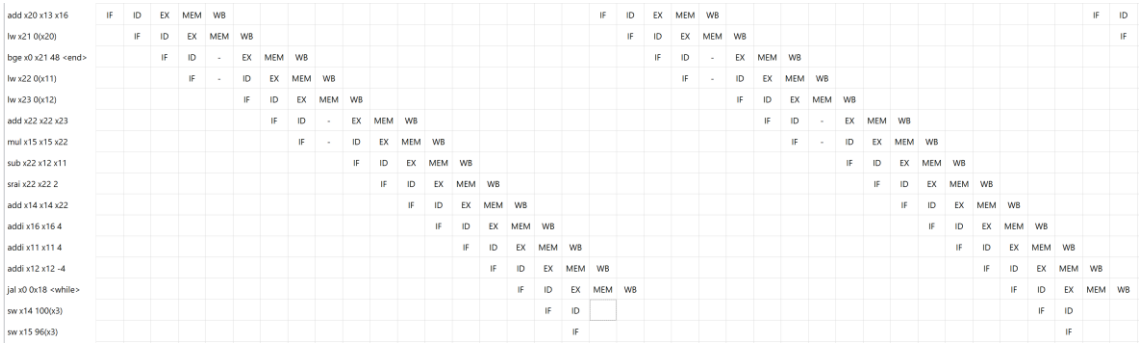
Número de ciclos de relógio: 148

Número de instruções executadas: 111

Q3.5 -> $\frac{\text{Instruções úteis}}{\text{instruções totais}} = \frac{111}{111} = 100\%$ de eficiência. Comparando este valor com o da pergunta 1.12 e o da pergunta 2.5 podemos concluir que esta arquitetura é mais eficiente.

Q3.6 -> $\text{IPC} = \frac{\text{Instruções executadas}}{\text{Ciclos de relógio}} = \frac{111}{148} = 0.75$ instuções por ciclo. Este valor é menor que 1 pois existem stall pelo meio o que faz com que as instruções demorem mais que um ciclo a ser concluídas. Para além disso esse número também diminui com os flush dos branch e dos jal.

Q3.9



Q3.10

Stall conflitos de dados raw = 4

Stall conflito de controlo = 0

Q3.11 O IPC é inferior a 1 porque há instruções em que acontece um flush devido aos saltos dos branch e dos jal e porque existe um stalls entre o lw x21 0(x20) e o branch. Este stall deve-se ao facto de o load para x21 só ir buscar o valor à memória na parte da MEM e depois só pode ser usado em WB. Logo o EX do bge tem de ser dado quando o load já está no WB.

Q4.2 -> As alterações feitas foram:

Reorganização do código nas linhas 42, 43 e 59, pois assim evita-se de usar stall para esperar o x21 para o branch e o x17 para o ecall.

```
29 # Program section
30         .text
31
32         addi    x11, x3, 0
33         addi    x13, x11, 48
34         addi    x12, x13, -4
35
36         lw      x14, 100(x3)
37         lw      x15, 96(x3)
38         li      x16, 0
39
40 while:  add     x20, x13, x16
41         lw      x21, 0(x20)
42         lw      x22, 0(x11)
43         lw      x23, 0(x12)
44         blez    x21, end
45
46         add     x22, x22, x23
47         mul     x15, x15, x22
48
49         sub     x22, x12, x11
50         srai    x22, x22, 2
51         add     x14, x14, x22
52
53         addi    x16, x16, 4
54
55         addi    x11, x11, 4
56         addi    x12, x12, -4
57         jal     x0, while
58
59 end:    addi    a7, x0, 10
60         sw      x14, 100(x3)
61         sw      x15, 96(x3)
62         ecall
```

Q4.3

Número de ciclos de relógio: 133

Número de instruções executadas: 113

Q4.4-> $\text{IPC} = \frac{\text{Instruções executadas}}{\text{Ciclos de relógio}} = \frac{113}{133} = 0.85$ instuções por ciclo. Comparando este valor com o anterior podemos ver que este sobe em 0.1, devido a já não ser necessário o uso de alguns stalls.

Q4.7

add x20 x13 x16		IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB									IF	ID
lw x21 0(x20)			IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB										IF
lw x22 0(x11)				IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB									
lw x23 0(x12)					IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB								
bge x0 x21 40 <end>						IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB							
add x22 x22 x23							IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB						
mul x15 x15 x22								IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB					
sub x22 x12 x11									IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB				
srai x22 x22 2										IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB			
add x14 x14 x22											IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB		
addi x16 x16 4												IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB	
addi x11 x11 4													IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB
addi x12 x12 -4														IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB
jal x0 0x18 <while>															IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB
addi x17 x0 10																IF	ID													IF	ID		
sw x14 100(x3)																	IF														IF		

Q4.8

Stall conflitos de *raw* = 0

Stall conflito de controlo = 0

Q4.9 -> O IPC continua inferior a 1 porque há instruções em que acontece um flush devido aos saltos dos *branch* e dos *jal*.

Q6.2

Número de ciclos de relógio: 105

Número de instruções executadas: 93

Q6.3 -> $\text{IPC} = \frac{\text{Instruções executadas}}{\text{Ciclos de relógio}} = \frac{93}{105} = 0.886$. Este parâmetro aumenta em 0.036, pois ao fazermos um loop unrolling diminuímos a quantidade de instruções de salto no programa, logo perdemos menos instruções para *flush* ao longo do programa.

Q6.6

Q6.7

Stall conflitos de *raw* = 0

Stall conflito de controlo = 0

Q6.8 -> O IPC continua inferior a 1 porque há instruções em que acontece um flush devido aos saltos dos *branch*.

Q6.9-> $\text{Speedup} = \frac{\text{Ciclos antigos}}{\text{Ciclos novos}} = \frac{133}{105} = 1.27$