Laboratório 4 – Análise de um processador Pipelined

Trabalho realizado por João Maçãs 99970  
Nuno Frade 100049

Exercício 1:

Q1.5:

O valor no registo x11 é igual ao endereço de memoria porém o x13 é diferente ao valor inicial do vetor b[]. O processador apresenta a arquitetura ‘pipeline’ e como estão duas instruções seguidas em que a segunda necessita do resultado da primeira, não vai haver ciclos suficientes para a segunda instrução ser executada corretamente resultando assim num conflito de dados.  
Instruções que provocaram este conflito:  
Escreve: addi x13, x11, 48  
Lê: addi x12, x13, -4

Q1.6

Conflito

Escreve: addi x11, x3, 0

Lê: addi x13, x11, 48

Operando: x11

Conflito

Escreve: addi x13, x11, 48

Lê: addi x12, x13, -4

Operando: x13

Conflito

Escreve: add x20, x13, x16

Lê: lw x21, 0(x20)

Operando: x20

Conflito

Escreve: lw x21, 0(x20)

Lê: blez x21, end

Operando: x21

Conflito

Escreve: lw x22, 0(x11)

Lê: add x22, x22, x23

Operando: x22

Conflito

Escreve: add x22, x22, x23

Lê: mul x15, x15, x22

Operando: x22

Conflito

Escreve: lw x23, 0(x12)

Lê: add x22, x22, x23

Operando: x23

Conflito

Escreve: sub x22, x12, x11

Lê: srai x22, x22, 2

Operando: x22

Conflito

Escreve: srai x22, x22, 2

Lê: add x14, x14, x22

Operando: x22

Q1.8

Apos a alteração do programa (alteração da ordem das instruções e adição de ‘nops’) os valores nos registos 11 e 13 correspondem ao inicio dos vetores correspondentes (a[], b[]). Isto porque na arquitetura pipeline, tem de haver 3 instruções a ser lidas desde o momento em que uma instrução para escrever num registo é chamada até ao fim do write-back do mesmo. Ao reordenar o código foi possível, resolver o conflito de dados anterior e fazer com que o valor nos registos esteja correto.

Q1.11

Número de Ciclos CLK: 160

Número de instruções: 140

Q1.12

Instruções uteis = 118

Instruções inúteis = 22

Q1.13

Predict not taken visto que o programa quando encontra um branch, continua a executar as instruções seguintes, até a instrução de branch estar na fase EXE (onde se encontra o branch control). Assim, o processador assume sempre que nunca é um branch até chegar ao fim da fase EXE, anulando as instruções que foram lidas pouco 2 ciclos depois da instrução de branch.

Exercício 2:

Q2.3

Para o programa funcionar corretamente houve 2 alterações a serem feitas, visto que com o forwarding podemos ir buscar o resultado de uma instrução na fase EXE para a fase ID, maioria dos conflitos de dados originalmente numa arquitetura sem forwarding foram resolvidos, à exceção de casos de load words e de branches.

Na minha abordagem, portanto, logo apos o comando “lw x23, 0(x12)” seria necessário chegar ao estágio MEM para que o valor na memoria fosse para o registo 23. Como tinha uma instrução imediatamente a seguir a necessitar do x23, acontece que este ainda teria o valor antes do load word e, portanto, foi necessário colocar um “nop” imediatamente depois.

Foi também necessário fazer uma alteração para a instrução que escreve em x21 “lw x21, 0(x20)” e que lê “blez x21, end” em que foi necessário adicionar 2 ‘nops’ para que o branch (fase EXE) antes que a primeira instrução seja lida na fase “MEM” e, portanto, que o seu valor se altere para o que pretendemos.

Q2.4

Ciclos CLK – 154

instruções executadas – 134

Q2.5

Instruções uteis = 112  
Instruções inúteis = 22

Q2.6

Exercício 3

Q3.3

Nenhumas, pois, neste tipo de arquitetura, quaisquer conflitos de dados ou de controlo que aconteciam nos outros processadores, são automaticamente resolvidos. Assim, o programa funciona perfeitamente e não necessita de alterações.

Q3.4

Ciclos CLK – 148

Instruções executadas – 111

Q3.5

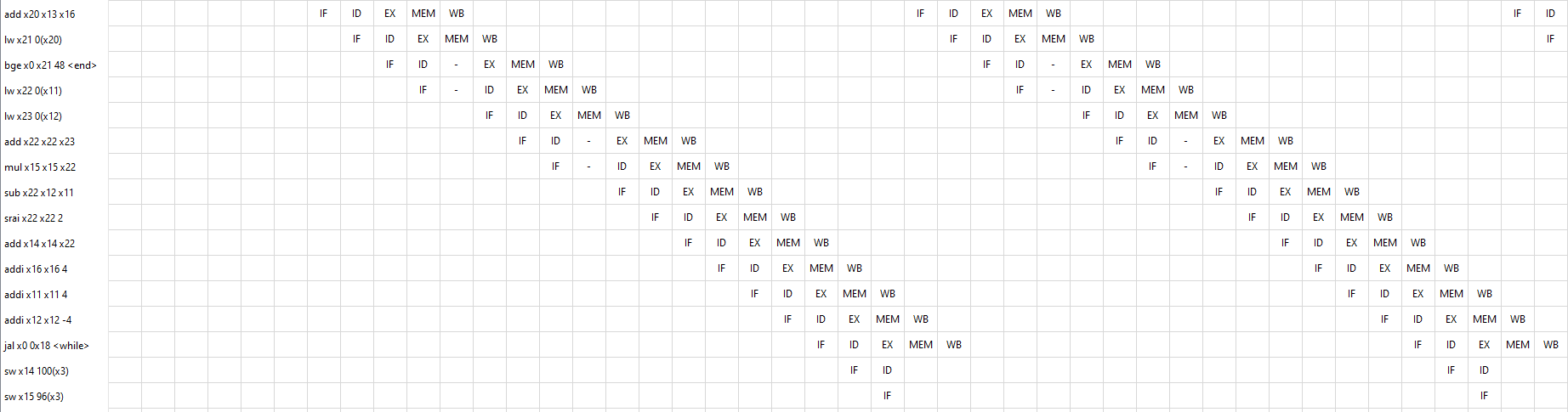
Instruções uteis = 111  
Instruções inúteis = 0

Podemos concluir que a eficiência do processador é bastante positiva visto que através dos forwardings, o numero de instruções inúteis foi eliminado.

Q3.6

O IPC é inferior a 1, devido ao numero de ciclos que estão a ser usados sem executar qualquer tipo de instrução, isto deve-se principalmente aos stalls inseridos pelo processador de forma a que o programa funcione corretamente.

Q3.9



Q3.10  
  
Apesar de não estarem devidamente representados na tabela, os conflitos de controlo vão ser as instruções começadas a ler imediatamente após a instrução de jal, visto que estas são começadas a ler, porém quando o processador finalmente assume que é uma instrução de salto, estas vão têm que ser canceladas. Através da análise da tabela em cima, o número de **stalls de controlo** são: 4  
  
Na tabela, os stalls são representados por ‘- ‘, e são contados na vertical. Sendo assim, o número de **stalls de dados** na tabela são: 4

Q3.11  
  
 Como referido na questão anterior “Q3.6”, um fator que influencia o IPC são os stalls, instruções implementadas para que o programa funcione corretamente através da correção de conflitos de dados e de forwardings. Podemos observar na figura anterior que existem stalls imediatamente após os load words pois é necessário aceder à memoria para colocar o novo valor no registo e, portanto, os stalls vão “atrasar” para que a instrução chegue ao fim da fase “MEM” e assim dar load do valor correto.

Exercício 4

Q4.2

As alterações que fiz foram todas relativas aos stalls implementados pelo forwarding ou pelo hazard detection. Assim, sempre que possível tentamos reordenar as instruções do programa de forma que não influencie o resultado, mas também de forma que conseguíssemos diminuir o número de stalls.  
 Para tal alteramos o local da instrução “addi a7, x0, 10” para o início do loop para que o quando chegasse ao ecall não fosse preciso 2 stalls. Alteramos também a instrução “addi x16, x16, 4” de sítio para que o stall aplicado, devido aos load words de x22 e x23 não fosse necessário melhorando assim a eficiência do processador.

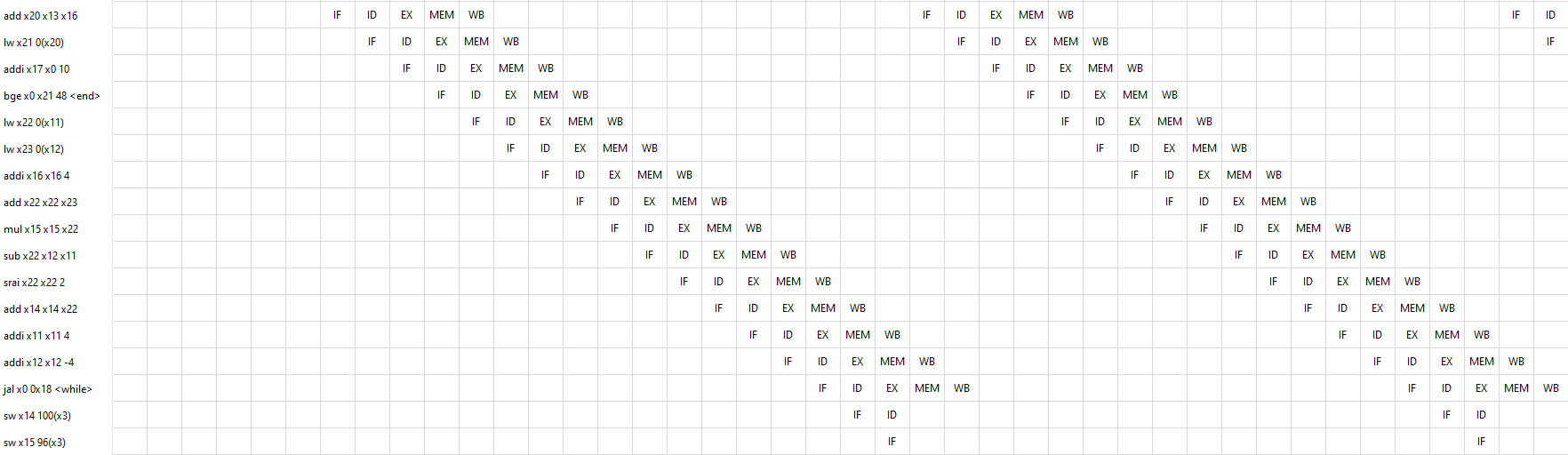
Q4.3

Ciclos CLK: 138

Instruções executadas: 118

Q4.4

Em relação ao exercício anterior, podemos verificar que o IPC aumentou. Assim, a eficiência do processador foi melhorada apenas pela reordenação do código de forma que conseguíssemos remover o maior número de stalls que existiam na versão anterior do programa (Exercício 3).

Q4.7  


Q4.8

Stalls de dados (representados por ‘- ‘): 0

Stalls de controlo (instruções começadas a ler a seguir ao jal, mas acabam por ser canceladas): 4

Q4.9

Apesar de todos os conflitos de controlo terem sido resolvidos através da reordenação do código, o processador neste tipo de arquitetura e predição de salto (predict not taken), vai ser impossível chegar a ter um IPC igual a 1. Isto deve-se ao facto de o processador assumir que todas as instruções não vão ser salto até ler na fase “EXE” uma instrução que seja. Isto leva a que algumas instruções sejam lidas a mais o que vai fazer gastar alguns ciclos. No caso do código de laboratório, no loop, a instrução ‘add x20, x13, x16’ nunca vai ser lida imediatamente após o jump and link (pois começa a ler as instruções (sw x14; sw x15) o que vai levar a que haja mais ciclos. Assim, podemos concluir que no caso do laboratório, será impossível através da reordenação do código chegar a um IPC igual a 1.

Exercício 5 – Laboratório

Q5.3

Para que o programa feito no exercício 4, tivesse menor número de ciclos possível tivemos que alterar a forma como o loop no algoritmo estava implementado, com o objetivo de o while loop fazer o menor número de iterações possível. Assim, em vez de cada iteração ser feita para cada valor de i, alteramos o programa de forma que fizesse em cada iteração para o valor de i, e i+1.No fim do ciclo tivemos que mudar a incrementação do ‘i++’ para ‘i = i+2’. Desta forma, as iterações do loop vão ser reduzidas a metade desperdiçando menos ciclos.

Q5.4

Ciclos CLK – 110

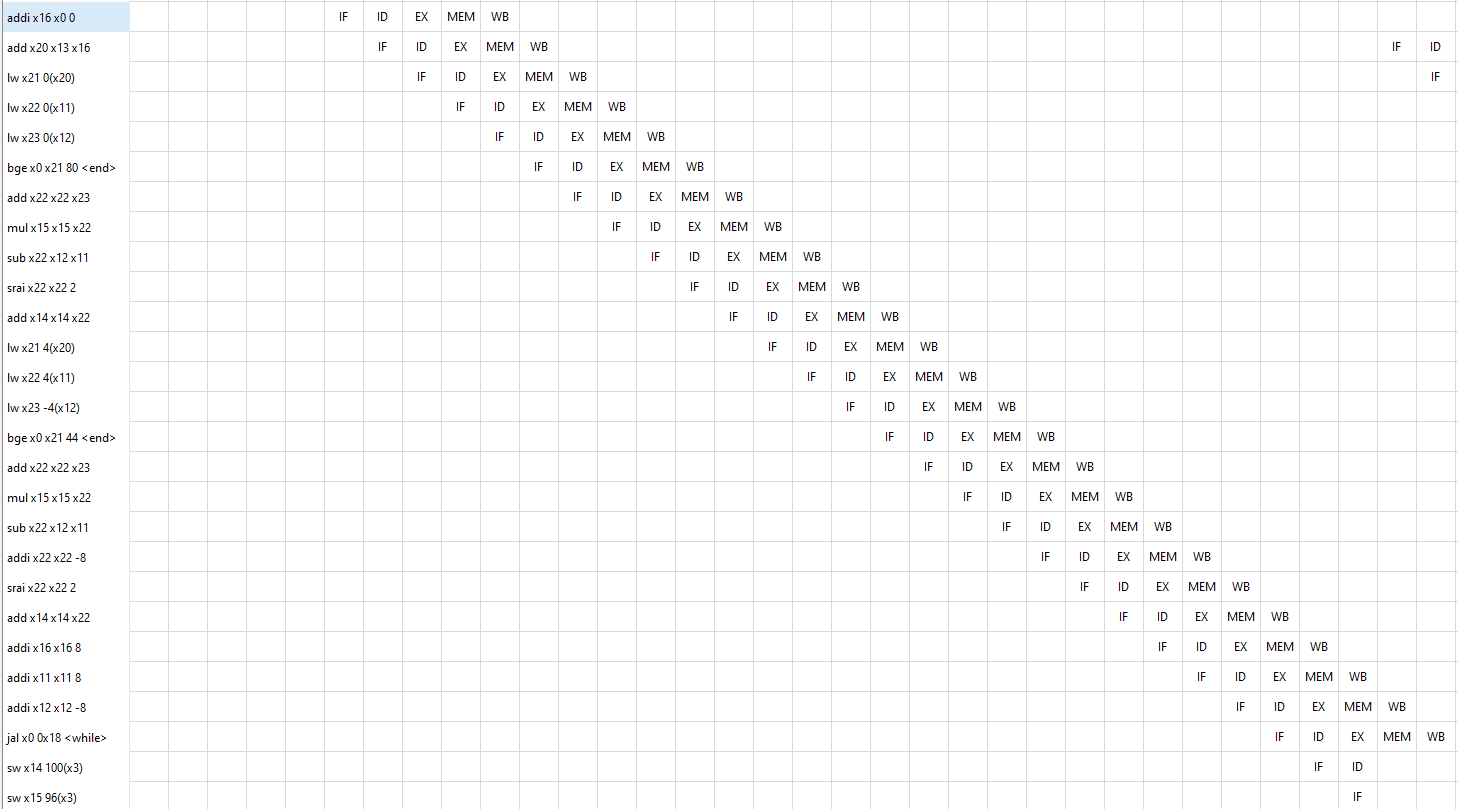
Instruções executadas – 96  
----------------------------------------------------------------------------------------------------------------------------------------------------------

Q5.5

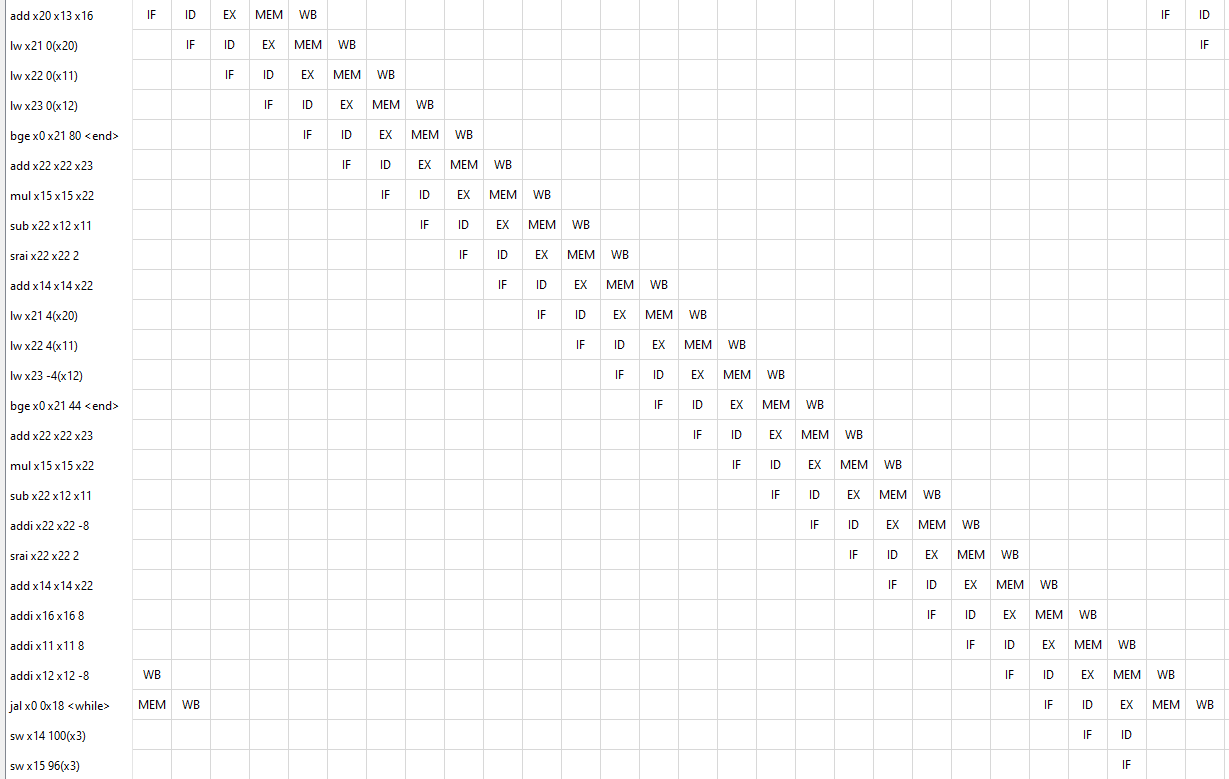
Podemos concluir que a eficiência do processador foi bastante positiva.

Q5.6

O IPC continua a ser menor que 1 devido aos conflitos de controlo que vêm apos o jal, que não poderão ser evitados devido ao tipo de predição de salto implementado (Predict not taken).

Q5.9  
1º Iteração  


2º Iteração



Q5.10

Todos os stalls de dados conseguiram ser evitados tal como no exercício 4.   
Número de Stalls de dados: 0

Devido à predição de salto implementada, nas duas primeiras iterações vai continuar a haver conflito de dados logo após o jal.  
Número de Stalls de controlo: 4  
  
Porém o número de stalls de controlo vai ser menor visto que o loop executa metade das vezes que executava no exercício 4, levando assim a um menor número de ciclos.

Q5.11  
 Como já repetido nas perguntas anteriores, o IPC vai continuar a ser menor que um devido aos conflitos de controlo após o jal. Estes são inevitáveis, porém com a nossa alteração do programa conseguimos fazer com que fossem reduzidos ao máximo levando assim a um IPC mais próximo de 1.