**Terceira Lista de Exercícios – Escalonamento Estático**

4.2) Considere os 4 trechos de código ao lado, cada um com 2 instruções.

```
i.   daddi  r1,r1,4          iii. sd     7(r1), r2
     ld     r2, 7(r1)             st.d   200(r7), f2

ii.  dadd   r3,r1,r2         iv.  bez    r1, place
     sd     7(r1), r2             sd     7(r1), r1
```

(a) Para os fragmentos (i-iv), identifique cada tipo de dependência que um compilador encontraria (pode não haver dependência) e aponte qual dentre fluxo de dados, re-uso de nome, ou estrutura de controle, causa a dependência. (b) Supondo execução não-especulativa, para cada fragmento discuta se um compilador poderia escalonar as duas instruções.

4.3) Considere o laço simples da Seção 4.1 (abaixo). Suponha que o número de iterações é desconhecido, embora grande. (a) Encontre o número ótimo teórico de desenroladas, com base nas latências da Fig 4.1. São necessários 2 laços distintos. (b) Qual é o número máximo real de vezes que o laço simples da Sec 4.1 pode ser desenrolado usando o código MIPS dado? Qual é o recurso limitante? Mostre como aumentar o número de vezes que o laço pode ser desenrolado ao transformar o código para que este faça uso menos intensivo do recurso limitante. De quanto esta transformação melhora o desempenho? (c) Para o CdI do MIPS, quais parâmetros adicionais limitam o número de vezes que este código pode ser desenrolado? Pista: Quando encontrar um parâmetro limitante, suponha que o recurso que ele define é ilimitado e então procure por outro parâmetro e repita enquanto necessário.

```
Laço da Sec 4.1

loop:   ld.d   f0, 0(r1)
        add.d  f4, f0, f2
        st.d   f4, 0(r1)
        addui  r1, r1, -8
        bne    r1,r2,loop
```

Fig 4.1

| instr que produz | instr que usa | latência |
|---|---|---|
| alu PF | alu PF | 3 |
| alu PF | store double | 2 |
| ld double | alu PF | 1 |
| ld double | sto double | 0 |
| ld int | qquer | 1 |
| alu int | qquer | 0 |

4.6) O laço ao lado é incomum. Primeiro, liste as dependências e então reescreva-o para que seja possível executá-lo em paralelo.

```
for (i=1; i<100; i=i+1) {
    a[i] = b[i] + c[i];       /* S1 */
    b[i] = a[i] + d[i];       /* S2 */
    a[i+1] = a[i] + e[i]; }   /* S3 */
```

4.10) Neste exercício completa-se a transformação de código iniciada no exemplo de *software-pipelining* do laço na página 330.

(a) Iniciando com a solução dada no exemplo, escreva o código para o laço completo, adicionando os trechos inicial e final. Suponha que serão executadas muitas iterações. Não é necessário mostrar o código para inicializar a variável de indução ou incrementá-la. Usando as latências da Fig 4.1 e supondo que o desvio consome 1 ciclo (*branch-delay slot*), escreva uma expressão para o calcular o tempo total de execução deste laço para incrementar todos os elementos de um vetor.

```
loop:   ld.d   f0, 0(r1)
        add.d  f4, f0, f2
        st.d   f4, 0(r1)
        addui  r1, r1, -8
        bne    r1,r2,loop
```

(b) O código original do laço pode executar somente uma iteração. Escreva o código completo para o laço com *software pipelining* que permita uma, duas ou tantas iterações quantas necessárias para cobrir o vetor. Então, para completar sua resposta, escalone e possivelmente transforme o código para que ele possa executar conm somente duas bolhas, e mostre onde estas bolhas ocorrem. Use as latências da Fig 4.1, e use desvios que sempre executam uma instrução no *branch-delay slot* de 1 ciclo.

4.11) No laço do exemplo de *software-pipelining*, considere que a latência do add.d seja 5 ciclos. O laço terá então uma bolha. Mostre como este laço pode ser modificado com *pipelining* e desenrolado para eliminar a/s bolha/s. O laço deve ser desenrolado o mínimo possível (1 vez). Mostre os trechos inicial e final.