UNIVERSIDADE FEDERAL DE LAVRAS

Departamento de Ciência da Computação

Disciplina: **Arquitetura de Computadores II - GCC123**

2ª Lista de Exercícios

Professor: **Luiz Henrique A. Correia**

Data de entrega: 18/01/23

A lista deve ser MANUSCRITA, digitalizada e enviada para o Campus Virtual.

1. Muitas aplicações importantes fazem uso exaustivo de loops com o seguinte formato:

```
for (int i = 0; i ≤ 100; i++) {
 Y[i] := a * X[i] + Y[i];
 }
```

Por exemplo esta é a operação principal na eliminação de Gauss, conhecida com DAXPY (*Double-precision aX Plus Y*).

```
Loop:
     I1:  LD F2, 0(R1);       #carrega X[i]
     I2:  MULTD F4,F2,F0;     #multiplica a*X[i]
     I3:  LD F6, 0(R2) ;      #carrega Y[i]
     I4:  ADDD F6,F4,F6;      #soma aX[i] + Y[i]
     I5:  SD F6,0(R2);        #armazena Y[i]
     I6:  ADDI R1,R1,8;       #incrementa o índice de X
     I7:  ADDI R2,R2,8;       #incrementa o índice de Y
     I8:  SGTI R3,R1,#800; # R3 = 1 se R1 > Imediato
     I9:  BEQZ R3,Loop;       #volta ao Loop se não terminou.
```

Assuma as seguintes latências de execução das instruções:

- Operações de inteiro: 1 ciclo de clock.
- Operações de ponto flutuante: Adição 2 ciclos; Multiplicação 5 ciclos; Divisão 20 ciclos.

Além disso, considere a existência de unidades de adiantamento e de *stalls* para o pipeline.

a) Identifique todas as dependências RAW (*Read After Write*), WAW (*Write After Write*) e WAR (*Write After Read*) no código do *Loop* apresentado na formulação acima. Escreva as dependências dentro de cada iteração. Utilize a seguinte notação, por exemplo, para indicar uma dependência entre as instruções $Inst15$ e $Inst16$ através de $F3$: $Ins15 \rightarrow Inst16(F3)$.

Resposta:
RAW:
$I_1 \rightarrow I_2(F2)$
$I_4 \rightarrow I_2(F4)$
$I_4 \rightarrow I_3(F6)$
$I_5 \rightarrow I_4(F6)$
$I_8 \rightarrow I_6(R1)$
$I_9 \rightarrow I_8(R3)$
WAR:
$I_7 \rightarrow I_3(R2)$
$I_6 \rightarrow I_1(R1)$
WAW:
$I_4 \rightarrow I_3(F6)$

b) Preencha o diagrama abaixo que representa a execução das instruções em pipeline. Considere que existem unidades de adiantamento, unidades de deteção de *hazards* e que o desvio condicional é decidido no segundo ciclo. Quantos ciclos de clock são necessários para executar esse código?

Ciclos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
`LD F2,0(R1)`	I	D	X	M	W																
`MULTD F4,F2,F0`		I	D	S	X1	X2	X3	X4	X5	M	W										
`LD F6, 0(R2)`			I	S	D	X	M	W													
`ADDD F6,F4,F6`				S	I	D	S	S	S	A1	A2	M	W								
`SD F6,0(R2)`						I	S	S	S	D	X	M	W								
`ADDI R1,R1,8`							S	S	S	I	D	X	M	W							
`ADDI R2,R2,8`											I	D	X	M	W						
`SGTI R3,R1,#800`												I	D	X	M	W					
`BEQZ R3,Loop`													I	S	S	D	X	M	W		
`BEQZ R3,Loop`													I	D	X	M	W				

2. Um programa contém quatro desvios condicionais. O programa será executado milhares de vezes. Abaixo estão os resultados de cada desvio na execução do programa (T - desvio tomado e N - desvio não tomado).
Branch 1 : `T-T-T-T`
Branch 2 : `N-N-N-N`
Branch 3 : `T-N-T-N-T-N`
Branch 4 : `T-T-N-T-T-N-T`

   Assuma que o comportamento de cada desvio permanece o mesmo para cada execução do programa. Para o esquema dinâmico, assuma que cada desvio tem seu próprio *buffer* preditor e que cada buffer inicia no mesmo estado antes de cada execução. Liste as predições para o seguinte esquema de predição de desvios:

   a) Sempre Tomado.

   b) Sempre não tomado.

   c) Preditor de 1 bit, iniciado como não tomado.

   d) Preditor de 2 bits, iniciado como fracamente não tomado.

   Responda: qual a **exatidão (%) total** das predições para cada item?

   Respostas:
   A precisão da previsão = 100% * Predição Correta / Total Branches a) Sempre tomado Branch 1: Predição: T-T-T-T, correto: 4 errado: 0 = 100

   Branch 2: Predição: N-N-N-N, correto: 0, errado: 4 = 0

   Branch 3: Predição: T-N-T-N-T-N , correto: 3, errado: 3 = 50,00

   Branch 4: Predição: T-T-N-T-T-N-T , correto: 5, errado: 2 = 71,43 Total: correto: 12, errado: 9 Precisão = 100% * 12/21= 57,14%

   b) Sempre não-tomado Branch 1: Predição: T-T-T-T, correto: 0 errado: 4 = 0

   Branch 2: Predição: N-N-N-N, correto: 4, errado: 0 = 100

   Branch 3: Predição: T-N-T-N-T-N , correto: 3, errado: 3 = 50,00

   Branch 4: Predição: T-T-N-T-T-N-T , correto: 2, errado: 5 = 28,57

   Total: correto: 9, errado: 12 Precisão = 100% * 9/21= 42,86%

   c) Preditor de 1, iniciado com NT

   Branch 1: Predição: T-T-T-T, correto: 3, errado: 1 = 75

   Branch 2: Predição: N-N-N-N, correto: 4, errado: 0 = 100

   Branch 3: Predição: T-N-T-N-T-N , correto: 0, errado: 6 = 0

Branch 4: Predição: T-T-N-T-T-N-T , correto: 2, errado: 5 =28,57

Total: correto: 12, errado: 15

Precisão = 100% * 12/27= 44,4%

d)
Branch 1: Predição: T-T-T-T, correto: 3, errado: 1 = 75

Branch 2: Predição: N-N-N-N correto: 4, errado: 0 = 100

Branch 3: Predição: T-N-T-N-T-N , correto: 2, errado: 4 = 33,33

Branch 4: Predição: T-T-N-T-T-N-T , correto: 4, errado: 3 = 57,14

Total: correto: 13, errado: 8

Precisão = 100% * 13/21= 61,9%

3. Seja trecho de código abaixo:

```
Loop:
      LD F0, 0(R2)
      LD F4, 0(R3)
      MULTD F0,F0,F4
      ADD F2,F0,F2
      ADDI R2,R2,#8
      ADDI R3,R3,#8
      SUB R5,R4,R2
      BNZ R5,Loop
```

Assuma que o valor inicial de $R4$ é $R2 + 792$ e responda:

a) Desenhe a execução do pipeline (ciclos) sem considerar *forwarding* ou hardware de adiantamento, mas assumindo a escrita e leitura de registradores no mesmo ciclo de clock. Assuma que o desvio é manipulado pelo *flushing* do pipeline. Quantos ciclos são gastos na execução deste loop?

b) Agora considere o hardware de adiantamento e assuma que os desvios são preditos como não tomados. Quantos ciclos são gastos na execução deste loop?

c) Reordene o código considerando *branch delayed* de 1 ciclo. As instruções podem ser reordenadas e seus operandos modificados, mas não realize desenrolamento do loop. Mostre o diagrama do pipeline reordenado e calcule o número de ciclos necessários para executar todo o loop.

a) Duas forma para representar, escolha a que lhe for mais conveniente.

Sem adiantamento e com flushing

Loop:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
LD F0,0(R2)	I	D	X	M	W																					
LD F4,0(R3)		I	D	X	M	W																				
MULTD F0,F0,F4			I	D	S	S	X1	X2	X3	X4	X5	X6	X7	M	W											
ADDD F2,F0,F2				I	S	S	D	S	S	S	S	S	S	S	S	A1	A2	A3	A4	M	W					
ADDI R2,R2,#8					S	S	I	S	S	S	S	S	S	S	S	D	X	M	W							
ADDI R3,R3,#8								S	S	S	S	S	S	S	S	I	D	X	M	W						
SUB R5,R4,R2																	I	D	S	X	M	W				
BNZ R5,Loop																		I	S	S	S	D	X	M	W	
.....																										
LD F0,0(R2)																						I	D	X	M	W

Sem adiantamento e com flushing

Loop:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
LD F0,0(R2)	I	D	X	M	W																					
LD F4,0(R3)		I	D	X	M	W																				
MULTD F0,F0,F4			I	S	S	D	X1	X2	X3	X4	X5	X6	X7	M	W											
ADDD F2,F0,F2				S	S	I	S	S	S	S	S	S	S	S	D	A1	A2	A3	A4	M	W					
ADDI R2,R2,#8							S	S	S	S	S	S	S	S	I	D	X	M	W							
ADDI R3,R3,#8								S	S	S	S	S	S	S		I	D	X	M	W						
SUB R5,R4,R2																	I	S	D	X	M	W				
BNZ R5,Loop																		S	I	S	S	D	X	M	W	
.....																										
LD F0,0(R2)																						I	D	X	M	W

O loop será executado 792/8 (tamanho da palavra) = 99 vezes.
Cada loop gasta 25 ciclos para ser completado, e a segunda iteração começa no ciclo 22.
Assim, tem-se: (98 x 22) + 1 x 25 = 2181 ciclos de clock.

b)

Loop:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	26
LD F0,0(R2)	I	D	X	M	W																						
LD F4,0(R3)		I	D	X	M	W																					
MULTD F0,F0,F4			I	D	S	X1	X2	X3	X4	X5	X6	X7	M	W			Ciclo 17 as duas instruções não usam a FU de M (veja Aula 8)										
ADDD F2,F0,F2				I	S	D	S	S	S	S	S	S	A1	A2	A3	A4	M	W									
ADDI R2,R2,#8					S	I	S	S	S	S	S	S	D	X	M	W											
ADDI R3,R3,#8							S	S	S	S	S	S	I	D	X	M	W										
SUB R5,R4,R2														I	D	X	M	W									
BNZ R5,Loop															I	S	S	D	X	M	W						
Instrução N (não Loop)																	S	I	D	X	M	W	Flush na Instrução (Aula6)				
LD F0,0(R2)																			I	D	X	M	W				

O loop será executado 99 vezes (792/8).
Se não considerarmos o desvio como T, cada loop gastará 21 ciclos para ser completado, e a segunda iteração começa no ciclo 18.
Assim, tem-se: (18 x 98) + 21 = 1785.

Considerando a predição de desvio for NT haverá um *flushing* do pipeline para as primeiras 98 iterações, com uma penalização de 1 ciclo durante as 98 iterações, sendo que a segunda iteração começará no ciclo 19 gastando: (98x19) + 21 da última iteração quando a predição estará correta => 1883.

c)

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Loop:																											
	LD F0,0(R2)	I	D	X	M	W																					
	LD F4,0(R3)		I	D	X	M	W																				
	ADDI R2,R2,#8			I	D	X	M	W																			
	MULTD F0,F0,F4				I	D	X1	X2	X3	X4	X5	X6	X7	M	W												
	ADDI R3,R3,#8					I	D	X	M	W																	
	SUB R5,R4,R2						I	D	X	M	W																
	ADDD F2,F0,F2							I	D	S	S	S	S	A1	A2	A3	A4	A5	M	W							
	BNZ R5,Loop								I	S	S	S	S	D	X	M	W										

	LD F0,0(R2)									S	S	S	S	I	D	X	M	W									

ou
O loop será executado 99 vezes (792/8).
Cada loop gasta 18 ciclos para ser completado, e a segunda iteração começa no ciclo 14.
Assim, tem-se: (14x 98) + 18 = 1390.
O loop será executado 99 vezes (792/8).
Cada loop gasta 19 ciclos para ser completado, e a segunda iteração começa no ciclo 13.
Assim, tem-se: (13x 98) + 19 = 1293.

4. Dado o código do Loop abaixo que calcula $Y[i] = a \times X[i] + Y[i]$, faça o desenrolamento do loop em 4 vezes para escaloná-lo, eliminando o overhead das instruções no loop.

```
Loop:
      LD F0, 0(R1)
      MULTD F0,F0,F2
      LD F4, 0(R2)
      ADDD F0,F0,F4
      SD F0, 0(R2)
      ADDI R1,R1,#-8
      ADDI R2,R2,#-8
      BNEQZ R1,Loop
```

Resposta:
Corpo 1:

```
Loop:
      LD F0, 0(R1)
      MULTD F0,F0,F2
      LD F4, 0(R2)
      ADDD F0,F0,F4
      SD F0, 0(R2)
```

Corpo 2:

```
      LD F6, -8(R1)
      MULTD F6,F6,F2
      LD F8, -8(R2)
      ADDD F6,F6,F8
      SD F6, -8(R2)
```

Corpo 3:

```
      LD F10, -16(R1)
      MULTD F10,F10,F2
      LD F12, -16(R2)
      ADDD F10,F10,F12
      SD F10, -16(R2)
```

Corpo 4:

```
      LD F14, -24(R1)
      MULTD F14,F14,F2
```

```
LD F16, -24(R2)
ADDD F14,F14,F16
SD F14, -24(R2)
```
Controle:
```
ADDI R1,R1,#-32
ADDI R2,R2,#-32
BNEQZ R1,Loop
```

5. O seguinte código em C irá calcular o produto escalar de dois vetores A e B de 100 entradas.
,

```
double dotProduct = 0;
for (int i = 0; i < 100; i++) {
   dotProduct += A[i]*B[i];
   }
```

Assuma para o loop as seguintes convenções:

- O registrador $R7$ armazena o valor 100.
- O registrador $dotProduct$ ($F10$) é inicializado em 0.
- O endereço base para o vetor $A[i]$ está em ($R5$).
- O endereço base para o vetor $B[i]$ está em ($R6$).

Questões a serem realizadas:

a) Escreva o código para o MIPS.

b) Realize o desenrolamento de loop em **três** vezes.

c) Escreva o código de forma otimizada.

Resposta:

dotProduct:
l.d $f10, 0($5) # $f10 <- A[i]
l.d $f12, 0($6) # $f12 <- B[i]
mul.d $f10, $f10, $f12 # $f10 <- A[i]*B[i]
add.d $f8, $f8, $f10 # $f8 <- $f8 + A[i]*B[i]
addi $5, $5, 8 # incrementa i para A[i]
addi $6, $6, 8 #incrementa i para B[i]
addi $7, $7, -1 # decrementa o contador do loop
bgtz $7, dotProduct # Continua se o contador do loop > 0

Desenrolado: Renomear os registradores e mudar o controle de loads e stores

dotProduct:
l.d $f10, 0($5) # $f10 <- A[i]
l.d $f12, 0($6) # $f12 <- B[i]
mul.d $f10, $f10, $f12 #$f10 <- A[i]*B[i]
add.d $f8, $f8, $f10 #$f8 <- $f8 + A[i]*B[ki]

l.d $f14, 8($5) # $f14 <- A[i + 1]
l.d $f16, 8($6) # $f16 <- B[i +1]

```
mul.d $f18, $f14, $f16 # $f18 <- A[i+1]*B[i+1]
add.d $f8, $f8, $f18 #$f8 <- $f8 + A[i+1]*B[i+1]

l.d $f20, 16($5) # $f20 <- A[i + 2]
l.d $f22, 16($6) # $f22 <- B[i +2]
mul.d $f24, $f20, $f22 # $f24 <- A[i+2]*B[i+2]
add.d $f8, $f8, $f24 #$f8 <- $f8 + A[i+2]*B[i+2]

l.d $f26, 24($5) # $f26 <- A[i + 3]
l.d $f28, 24($6) # $f28 <- B[i +3]
mul.d $f30, $f26, $f28 # $f24 <- A[i+3]*B[i+3]
add.d $f8, $f8, $f30 #$f8 <- $f8 + A[i+3]*B[i+3]

addi $5, $5, 32
addi $6, $6, 32
addi $7, $7, -4
bgtz $7, dotProduct ; Continue loop if $7 > 0
```

6. Considere um BTB que tem penalidades de 0, 2 e 2 ciclos para predição correta de desvio, predição incorreta e erro de buffer (*miss buffer*), respectivamente. Considere o projeto de BTB que distingue desvios condicionais e incondicionais, armazenando o endereço alvo para um desvio condicional e uma instrução alvo para um desvio incondicional.

   a) Qual é a penalidade em ciclos de clock quando um desvio incondicional é encontrado no buffer?

   b) Determine a melhoria de um *Branching Folding* para desvio inconcionais. Assuma 90% de taxa de acerto (*hit rate*), uma frequência de desvios incondicionais de 5% e 2 ciclos de penalidade para erro de buffer (*miss buffer*). Quanto foi a melhoria obtida por este aprimoramento? Quão alto deve ser a taxa de acerto para esta melhoria proporcione um ganho de desempenho?

   Resposta:

   a) Armazenar a instrução alvo (target) de um desvio incondicional efetivamente remove uma instrução de desvio condicional. Se existe um acerto (Hit) na busca de uma instrução no BTB e a instrução alvo estiver disponível, então esse tipo de instrução é introduzido na decodificação no lugar de uma instrução de desvio. A penalidade é -1 ciclo, em outras palavras, ele terá um ganho de desempenho de 1 ciclo.

   b) Se o BTB armazena somente o endereço alvo do desvio incondicional, a busca tem de recuperar a nova instrução. Isso tem uma CPI de: 5%x (90%x0 + 10%x2) = 0,01 A CPI dos desvios incondicionais representa a frequência de 5%.

   Se o BTB armazena a instrução alvo, a CPI será: 5% (90% (–1) + 10% 2) ou –0,035 O sinal negativo mostra uma redução do valor da CPI total.

   Para o percentual de acerto ser o mesmo, ou seja empatar. A taxa de acerto deve ser maior que zero, logo: 5% (x% (–1) + 10% 2) > 0 -0,05 x + 0,01 > 0 x > 0,01/0,05 –> x > 0,2 ou maior que 20% o valor seria de pelo menos 20%

7. Realize o escalonamento dinâmico para os algoritmos **Score Board** e **Tomasulo** para o código abaixo, considerando a latência para Instruções de inteiros e LD – 1 ciclo; ADDD – 2 ciclos; MULTD – 6 ciclos e DIVD – 12 ciclos, para o código:

```
LD F6, 32(R2)
LD F2, 44(R3)
MUL.D F0,F2,F4
SUB.D F8,F2,F6
DIVD F10,F0,F6
ADDD F6,F8,F2
```

Instruction		*j*	*k*	*Issue*	*Read operands*	*Execution complete*	*Write Result*	
LD	F6	32	R2	1	2	3	4	LD – 1
LD	F2	44	R3	5	6	7	8	ADD -2
MULTD	F0	F2	F4	6	9	15	16	MULTD -6
SUB.D	F8	F2	F6	7	9	11	12	DIVD – 12
DIVD	F10	F0	**F6**	8	17	29	30	
ADDD	**F6**	F8	F2	13	14	16	18	WAR em F6

Instruction status:

Instruction		*j*	*k*	-2**Issue*	-2**Execution complete*	-2**Write Result*
LD	F6	32	R2	1	2	3
LD	F2	44	R3	2	3	4
MULTD	F0	F2	F4	3	10	11
SUB.D	F8	F2	F6	4	6	7
DIVD	F10	F0	**F6**	5	23	24
ADDD	**F6**	F8	F2	6	9	10

8. O escalonamento dinâmico permite que instruções sejam executadas fora de ordem. Para tornar esse escalonamento possível é ampliado o número de unidades funcionais de execução, de modo a aumentar as chances de manter o pipeline cheio. O método de escalonamento dinâmico funciona até mesmo se o compilador não puder escalonar o código. Considere o código abaixo e que as latências de execução são: LD, SD – 1 ciclo, ADD – 2 ciclos e MULTD – 3 ciclos.

```
LD F0, 0(R2)
MULTD F8,F0,F2
ADD.D F2,F10,F2
SD F8,8,R3
```

Usando as mesmas figuras do exercício anterior, realize o escalonamento dinâmico para os algoritmos:

a) Score Board

b) Tomasulo

Instruction status				***Issue***	***Read operands***	***Execution complete***	***Write Result***
Instruction		***j***	***k***				
LD	F0	0	R2	1	2	3	4
MULD	F8	F0	F2	2	5	8	9
ADD	F2	F10	F2	3	4	6	7
SD	F8	8	R3	5	10	11	12

Please add the following required packages to your document preamble: multirow [normalem]ulem

Instruction status:				***Issue***	***Execution complete***	***Write Result***
Instruction		***j***	***k***			
LD	F0	0	R2	1	2	3
MULD	F8	F0	F2	2	6	7
ADD	F2	F10	F2	3	5	6
SD	F8	8	R3	4	8	9

References

[1] Hennessy, John L. and Patterson, David A.. Arquitetura de computadores: uma abordagem quantitativa. Vol. 5. Elsevier Brasil, 2014.

[2] Missouri State University. MARS (MIPS Assembler and Runtime Simulator): An IDE for MIPS Assembly Language Programming. `http://courses.missouristate.edu/kenvollmar/mars/download.htm`. Acessado em fevereiro de 2022.