1. Considere a execução do seguinte troço de código num processador com ISA compatível com o MicroBlaze (i.e., correspondente ao processador do laboratório). O processador tem 5 andares de *pipeline* (IF, ID, EX, MEM e WB) e endereça uma memória com palavras de 8 bits e organização *big endian*.

Código Assembly		**Resultado das instruções de controlo** (T - *Taken*; NT - *Not Taken*)			
CALC:	...				
	BGE R4,CALC	T,T,T,T,NT	T,T,T,NT	T,T,T,NT	NT
	...				
	BEQ R1,R2,NEXT	NT	T	NT	NT
	...				
NEXT:	...				
	BLT R2,CALC	T	T	T	NT
	SW 8(R5),R6				

1.0 val. (a) Considerando R5=00010128h e R6=0AB01234h, indique quais as posições de memória modificadas pela instrução `SW 8(R5),R6` e qual o seu novo valor.

5.0 val. (b) Para cada um dos mecanismos de predição de salto indicados em baixo, anote com um círculo os saltos correctamente preditos e a taxa de sucesso na predição de salto.

(Nota: considere todas as simplificações que achar convenientes anotando-as no enunciado.)

i. Preditor estático do tipo *Not Taken*

					Taxa de sucesso
BGE R4,CALC	T,T,T,T,NT	T,T,T,NT	T,T,T,NT	NT	
BEQ R1,R2,NEXT	NT	T	NT	NT	
BLT R2,CALC	T	T	T	NT	

ii. Preditor dinâmico com *Branch Predit Buffer* (BPB) de 1 bit.

					Taxa de sucesso
BGE R4,CALC	T,T,T,T,NT	T,T,T,NT	T,T,T,NT	NT	
BEQ R1,R2,NEXT	NT	T	NT	NT	
BLT R2,CALC	T	T	T	NT	

iii. Preditor dinâmico com *Branch Predit Buffer* (BPB) de 2 bits.

					Taxa de sucesso
BGE R4,CALC	T,T,T,T,NT	T,T,T,NT	T,T,T,NT	NT	
BEQ R1,R2,NEXT	NT	T	NT	NT	
BLT R2,CALC	T	T	T	NT	

Num.: **Nome:**

2. Considere a execução do seguinte troço de código num processador com ISA compatível com o MIPS.

```
120h: loop: L.D     F0,0(R1)      ; F0 ← M[R1+0]
124h:       L.D     F2,0(R2)      ; F2 ← M[R2+0]
128h:       MUL.D   F2,F0,F2      ; F2 ← F0 × F2
12Ch:       DADD.D  F2,F2,F4      ; F4 ← F2 + F4
130h:       S.D     0(R4),F2      ; M[R4+0] ← F4
134h:       DADD    R1,R1,#-8     ; R1 ← R1 - 8
138h:       DADD    R2,R2,#-8     ; R2 ← R2 - 8
13Ch:       DADD    R4,R4,#-8     ; R4 ← R4 - 8
140h:       BNE     R1,R0,loop    ; PC ← loop se R1=R0
```

Resolva as seguintes alíneas considerando todas as simplificações que achar convenientes (sempre que o fizer anote-as no enunciado).

3.0 val.

(a) Considerando um processador com 5 estágios de *pipeline* (IF, ID, EX, MEM, WB) sem *forwarding* de dados e resolução de saltos no andar de EX, indique na tabela os passos de execução de 1 iteração do troço de código.

Instrução	Ciclo de relógio					Observações
	IF	ID	EX	MEM	WB	
`loop: L.D F0,0(R1)`						
`L.D F2,0(R2)`						
`MUL.D F2,F0,F2`						
`DADD.D F2,F2,F4`						
`S.D 0(R4),F2`						
`DADD R1,R1,#-8`						
`DADD R2,R2,#-8`						
`DADD R4,R4,#-8`						
`BNE R1,R0,loop`						

3.0 val.

(b) Considerando a existência de *forwarding* de dados, indique na tabela os passos de execução de 1 iteração do troço de código.

Instrução	Ciclo de relógio					Observações
	IF	ID	EX	MEM	WB	
`loop: L.D F0,0(R1)`						
`L.D F2,0(R2)`						
`MUL.D F2,F0,F2`						
`DADD.D F2,F2,F4`						
`S.D 0(R4),F2`						
`DADD R1,R1,#-8`						
`DADD R2,R2,#-8`						
`DADD R4,R4,#-8`						
`BNE R1,R0,loop`						

Num.: **Nome:**

3.0 val.

(c) Considere um processador super-escalar com:

- agendamento dinâmico usando o algoritmo Tomasulo;
- execução especulativa (preditor de salto com uma taxa de sucesso de 100%);
- *issue* simultâneo de duas instruções;
- número suficientemente grande de estações de reserva e de entradas no ROB;
- 1 CDB e *commit* simultâneo de 2 instruções;
- unidades funcionais com as seguintes latências:
  - 1× INT ALU 1 ciclo
  - 1× LOAD/STORE 1 ciclo para cálculo do endereço + 1 ciclo para acesso à memória
  - 1× FP ADD 3 ciclos
  - 1× FP MULT 5 ciclos

Indique os passos de execução do troço de código, durante 2 iterações.
(Faça todas as simplificações que achar convenientes, indicando-as junto da resposta.)

Instrução			Ciclo de relógio					Observações
			IF	Issue	EX	CDB	Commit	
`loop:`	`L.D`	`F0,0(R1)`						
	`L.D`	`F2,0(R2)`						
	`MUL.D`	`F2,F0,F2`						
	`DADD.D`	`F2,F2,F4`						
	`S.D`	`0(R4),F2`						
	`DADD`	`R1,R1,#-8`						
	`DADD`	`R2,R2,#-8`						
	`DADD`	`R4,R4,#-8`						
	`BNE`	`R1,R0,loop`						
`loop:`	`L.D`	`F0,0(R1)`						
	`L.D`	`F2,0(R2)`						
	`MUL.D`	`F2,F0,F2`						
	`DADD.D`	`F2,F2,F4`						
	`S.D`	`0(R4),F2`						
	`DADD`	`R1,R1,#-8`						
	`DADD`	`R2,R2,#-8`						
	`DADD`	`R4,R4,#-8`						
	`BNE`	`R1,R0,loop`						

Num.: **Nome:**

3. Considere o seguinte troço de código em *C*:

```
for (i=1; i<100; i++) {
    A[i]   = B[i]*C[i]- D[i];          /* S1 */
    B[i]   = D[i] + A[i];              /* S2 */
    A[i-1] = E[i] - F[i];              /* S3 */
    D[i+1] = A[i-1] + G[i];            /* S4 */
}
```

3.0 val. (a) Assumindo que os *arrays* são distintos e não sobrepostos, apresente um grafo com todas as inter-dependências existentes.

2.0 val. (b) Diga, justificando, se o código apresentado é paralelizável ao nível de ciclo, i.e., se é possível executar cada iteração deste ciclo independentemente e em paralelo. Em caso afirmativo, reescreva-o por forma a que a sua execução possa ser efectuada em paralelo.