1. 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:       DADD    R1,R1,#-8     ; R1 ← R1 - 8
128h:       L.D     F2,0(R2)      ; F2 ← M[R2+0]
12Ch:       DADD    R2,R2,#-8     ; R2 ← R2 - 8
130h:       L.D     F4,0(R3)      ; F4 ← M[R3+0]
134h:       DADD    R3,R3,#-8     ; R3 ← R3 - 8
138h:       MUL.D   F2,F0,F2      ; F2 ← F0 × F2
13Ch:       MUL.D   F4,F4,F4      ; F4 ← F4 × F4
140h:       DSUB.D  F0,F4,F2      ; F0 ← F4 − F2
144h:       S.D     0(R4),F0      ; M[R4+0] ← F0
148h:       DADD    R4,R4,#-8     ; R4 ← R4 - 8
14Ch:       BNE     R1,R0,loop    ; PC ← loop se R1=R0
```

Resolva as seguintes alíneas fazendo todas as simplificações que considerar convenientes (sempre que o fizer, anote-as no enunciado).

2.0 val.

(a) Considere a execução do código num processador *in-order* com 5 estágios (IF, ID, EX, MEM, WB) sem qualquer mecanismo de *forwarding* de dados, de predição de saltos ou *delayed branch*. Identifique todas as dependências de dados e de controlo que geram conflitos. Indique-as directamente sobre o código.

3.0 val.

(b) Re-escreva o código de forma a resolver as dependências indicadas na alínea anterior.

Num.: **Nome:**

4.0 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/BRANCH	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 considerar convenientes, indicando-as junto da resposta.)

Instrução			Ciclo de relógio					Observações
			IF	Issue	EX	CDB	Commit	
`loop:`	`L.D`	`F0,0(R1)`						
	`DADD`	`R1,R1,#-8`						
	`L.D`	`F2,0(R2)`						
	`DADD`	`R2,R2,#-8`						
	`L.D`	`F4,0(R3)`						
	`DADD`	`R3,R3,#-8`						
	`MUL.D`	`F2,F0,F2`						
	`MUL.D`	`F4,F4,F4`						
	`DSUB.D`	`F0,F4,F2`						
	`S.D`	`0(R4),F0`						
	`DADD`	`R4,R4,#-8`						
	`BNE`	`R1,R0,loop`						
`loop:`	`L.D`	`F0,0(R1)`						
	`DADD`	`R1,R1,#-8`						
	`L.D`	`F2,0(R2)`						
	`DADD`	`R2,R2,#-8`						
	`L.D`	`F4,0(R3)`						
	`DADD`	`R3,R3,#-8`						
	`MUL.D`	`F2,F0,F2`						
	`MUL.D`	`F4,F4,F4`						
	`DSUB.D`	`F0,F4,F2`						
	`S.D`	`0(R4),F0`						
	`DADD`	`R4,R4,#-8`						
	`BNE`	`R1,R0,loop`						

2. Considere um processador RISC com barramento de dados e endereços de 32-bits e com execução especulativa e fora de ordem.

2.0 val. (a) Esboce o diagrama de estados correspondente a um preditor de saltos dinâmico com 2-bits.

2.0 val. (b) Esboce a estrutura de um *branch target buffer* BTB de dois bits, associado ao estágio de *Instruction Fetch*. Considerando que o BTB consiste numa cache de predição de salto com mapeamento directo, indique a dimensão de cada um dos campos do BTB.

2.0 val. (c) Considere que o agendamento dinâmico de instruções utiliza o algoritmo *Tomasulo*. Explique qual o mecanismo que garante que as instruções *issued* após um salto incorrectamente predito, não alteram o valor dos registos ou da memória. Justifique sucintamente.

Num.: **Nome:**

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

```
for (i=50; i>0; i--) {
    A[i+1] = B[i]*D[i] + C[i-1];       /* S1 */
    B[i-1] = D[i] - C[i];              /* S2 */
    C[i]   = 2*cos(D[i]+alpha*F[i]);   /* S3 */
    D[i]   = F[i];                     /* S4 */
}
```

3.0 val. (a) Assumindo que os vectores 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 de forma a que a sua execução possa ser efectuada em paralelo.

4. Considere um processador com barramento de dados e endereços de 32-bits com três níveis de cache:

   [L1] Cache de dados separada da cache de instruções (L1-D e L1-I), cada uma com capacidade para 32KB (dividas em 4 vias);

   [L2] Cache unificada com capacidade para 64KB em cada uma das 8 vias;

   [L3] Cache unificada com capacidade para 256KB em cada uma das 16 vias.

   Considere que em todos os níveis as linhas da cache são de 32B.

2.0 val. (a) Considerando endereços de 32-bits, decomponha a palavra de endereço em etiqueta (*tag*), índice (*index*) e deslocamento (*offset*) para as caches L1-D e L2. Justifique sucintamente.

1.0 val. (b) Indique o número e a largura (número de bits) dos comparadores requeridos para a cache L2. Justifique sucintamente.

3.0 val. (c) Apresente a expressão do tempo médio de acesso aos dados. Indique o significado de cada termo da expressão.

Num.: **Nome:**

2.5 val.

(d) Determine a taxa de falhas na cache L1-D na execução do seguinte troço de código:

```
static double X[1024], Y[1024];
...
Y[0] = a0*X[0]/2;
for (i=1 ; i<1024 ; i++) {
    Y[i] = a0*X[i] + a1*X[i-1] + b1*Y[i-1];
}
```

Assuma uma política de escrita *write-back* e uma política de alocação *write-allocate*. Considere que X=00016000h e Y=00018000h, que as variáveis `i`, `N`, `a0`, `a1` e `b1` são guardadas em registos, e que o compilador não realiza qualquer optimização ao código indicado, fazendo *load* das variáveis X e Y por ordem (i.e., da esquerda para a direita) e fazendo *store* da variável Y[i] em cada iteração do ciclo.

2.5 val.

(e) Repita a alínea anterior considerando uma política de escrita *write-through* e uma política de alocação *write-not-allocate*.

5. Considere um processador de 64-bits com suporte para um espaço de endereçamento físico de 16PB e um espaço de endereçamento virtual de 8TB, endereçável byte a byte. Considere que o sistema de tradução de memória virtual é multi-nível, com páginas (e tabelas de páginas) de 8KB e entradas nas tabelas de páginas de 8 bytes.

2.0 val. (a) Determine o número de níveis necessários à tradução de endereço virtual em endereço físico.

2.0 val. (b) Represente o esquema de tradução de endereços de um processo com segmentos de programa (instruções) e dados *heap* organizados a partir do endereço 00...000h e *stack* localizada a partir do endereço F...FFh (a *stack* cresce no sentido de endereços decrescentes).

2.0 val. (c) Determine o espaço mínimo de memória necessário à tradução de endereços de um processo com programa e *heap* de 10MB, e *stack* de 15KB.

Num.: **Nome:**

6. Considere um sistema heterogéneo constituído por 1 processador *host* com 1 core e um acelerador com 10 cores SMT, cada um capaz de executar simultaneamente até 20 threads, mas a um ritmo 10x mais lento que o processador *host*.

3.0 val.

(a) Admita que pretende executar um programa no sistema heterogéneo, o qual é constituído por 4 fases de processamento:

A. código puramente sequencial com tempo de execução no *host* de 2s.

B. ciclo *for* com 2000 iterações, sem dependências entre iterações e com tempo de execução no *host* de 1s.

B. ciclo *for* com 2000×2000 iterações, sem dependências entre iterações e com tempo de execução no *host* de 50s.

D. código puramente sequencial com tempo de execução no *host* de 1s.

Admita que a execução de cada uma das fases depende apenas de dados gerados pela fase anterior e que o tempo de comunicação dos dados é:

- $T_{A\rightarrow B}^{host\rightarrow accel} = T_{A\rightarrow B}^{accel\rightarrow host} = 10ms$,
- $T_{B\rightarrow C}^{host\rightarrow accel} = T_{B\rightarrow C}^{accel\rightarrow host} = 100ms$,
- $T_{C\rightarrow D}^{host\rightarrow accel} = T_{C\rightarrow D}^{accel\rightarrow host} = 1s$.

Calcule o *speed-up* máximo que é possível obter com o sistema heterogéneo, em comparação com a execução apenas no processador *host*. Para tal, determine qual o processador (host ou acelerador) mais vantajoso para executar cada fase do programa.

Num.: **Nome:**