C1	
1	
2	
3	
4	
5	
6	
7	
T	
C2	
8	
C3	
9	
F	

COMPETÊNCIAS FUNDAMENTAIS

1. As arquitecturas sequenciais, ou de ciclo único, caracterizam-se por executarem uma, e uma só, instrução por ciclo do relógio. Uma das consequências desta organização é que cada componente do *hardware*, por exemplo, ALU, só é utilizada activamente durante uma pequena fracção do ciclo do relógio. As arquitecturas encadeadas (*pipeline*) permitem uma utilização mais eficiente dos vários componentes. Explique porquê.

2. A lógica combinatória de um processador sequencial tem uma latência de 100 ps e pode ser decomposta num número arbitrário (n) de sub-blocos, cada com uma latência de 100/n ps. Considerando que a latência dos registos utilizados para armazenar o estado no fim de cada estágio é de 20 ps, qual a máxima frequência do relógio possível para a organização sequencial?

3. Nas condições do problema anterior, suponha que, devido a dependências de dados e controlo, as versões encadeadas deste processador exigem em média e por estágio do *pipeline* a injecção de um número de bolhas igual a 10% do número de instruções do programa. Por exemplo, um programa com #I instruções executado num *pipeline* com 2 estágios necessitaria de 1,2*#I ciclos do relógio, enquanto que num *pipeline* com 4 estágios necessitaria de 1,4*#I ciclos. Indique, justificando e apresentando todos os cálculos realizados, se este programa executaria mais rapidamente num *pipeline* com 5 ou 20 estágios.

Nome: _____ Número:_____

4. Pediu-se a um programador para optimizar o código apresentado na coluna esquerda da tabela abaixo. O desafio era que optimizasse o programa escrito em C, aplicando apenas optimizações que um compilador também pudesse utilizar. O resultado é apresentado na coluna da direita. Comente, justificando, se um compilador poderia aplicar autonomamente esta optimização.

Antes da optimização	Depois da optimização

`int func (int *arr) {`
`  int sum=0, i;`
`  for (i=0; i < tamanho (arr); i++)`
`  sum += arr[i];`
`  return sum;`
`}`
	`int func (int *arr) {`
	`  int sum=0, i, l;`
	`  l = tamanho(arr);`
	`  for (i=0; i < l; i++)`
	`  sum += arr[i];`
	`  return sum;`
	`}`

5. A tabela abaixo apresenta a execução de uma instrução do Y86 ao longo dos cinco estágios da organização PIPE- (note que esta informação respeita a 5 ciclos do relógio, necessários para executar a instrução). Identifique, justificando, de que instrução se trata.

Fetch	`D_icode:D_ifun ← M1[PC]`

Decode	`E_icode:E_ifun ← D_icode:D_ifun`
	`E_valA ← R[%esp] ; E_valB ← R[%esp]`
	`E_dstE ← ID(%esp) ; E_dstM ← 8`

Execute	`M_icode ← E_icode`
	`M_valE ← E_valB +4 ; M_valA ← E_valA`
	`M_dstE ← E_dstE ; M_dstM ← E_dstM`

Memory	`W_icode ← M_icode`
	`W_valE ← M_valE`
	`W_valM ← M4[M_valA]`
	`W_dstE ← M_dstE ; W_dstM ← M_dstM`

Write Back	`R[W_dstE] ← W_valE` `PC ← W_valM`

6. O código apresentado na coluna esquerda da tabela abaixo foi optimizado da forma apresentada na coluna da direita. Comente a optimização feita, justificando porque se pode esperar melhor desempenho desta 2ª versão.

Antes da optimização	Depois da optimização

`for (j=0; i < N; j++)`
`  sum += arr[i];`
	`for (j=0; i < N; j+=3) {`
	`  sum += arr[i];`
	`  sum += arr[i+1];`
	`  sum += arr[i+2];`
	`}`

Nome: _____ Número:_____

7. Considere o código apresentado na tabela abaixo. Proponha, justificando, uma alteração ao código em C que aumente as possibilidades de optimização por parte do compilador, reduzindo o número de acessos à memória.

```
int func (int *a, int n, int *res) {
  int i=0;
  *res = 0;
  while (i < n) {
    *res *= a[i];
    i++;
  }
  return (*res);
}
```

Nome: _____ Número:_____

COMPETÊNCIAS C2

8. Usando o código apresentado abaixo, identifique para cada ciclo do relógio a ocupação de cada estágio do processador, para a versão PIPE do Y86 – versão com atalhos (preencha a tabela abaixo). Justifique devidamente a sua resposta, indicando quais os sinais de atalho utilizados.

```
I1:   irmovl %10, %eax
I2:   pushl %eax
I3:   irmovl %20, %ebx
I4:   popl %ecx
I5:   addl %ebx, %ecx
```

	1	2	3	4	5	6	7	8	9	10	11	12	OBSERVAÇÕES

Nome: _____ Número:_____

COMPETÊNCIAS C3

9. Apresenta-se abaixo uma sequência de operações elementares correspondentes a um ciclo:

```
ciclo:      load 0(%ebx, %eax.0, 4)      →     t.1
            addl t.1, %ecx.0             →     %ecx.1
            addl $1, %eax.0              →     %eax.1
            cmpl %eax.1, $1000           →     cc.1
            jl-taken cc.1, ciclo
```

a. Transforme este código aplicando a técnica de loop splitting com grau 2:

b. Suponha que o código gerado na alínea anterior é executado num processador com as unidades funcionais, descritas na tabela abaixo. Apresente o diagrama de ocupação das unidades funcionais para as 2 primeiras iterações do ciclo.

Unidade	Latência	*Issue Time*
Operações inteiros	1	1
Operações inteiros + saltos	1	1
Leitura de memória (*load*)	3	1

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Nome: _____ Número:_____

Tabela Adicional para exercício 9

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Nome: _____ Número:_____

Arquitectura de Computadores
LEI – 2008/09

2º teste escrito
Janeiro.2009

Tabela Adicional para exercício 9

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Nome: _____ Número:_____

Tabela Adicional para exercício 8

	1	2	3	4	5	6	7	8	9	10	11	12	OBSERVAÇÕES

Nome: _____ Número:_____

Arquitectura de Computadores
LEI – 2008/09

2º teste escrito
Janeiro.2009

Tabela Adicional para exercício 8

	1	2	3	4	5	6	7	8	9	10	11	12	OBSERVAÇÕES

Nome: _____ Número:_____