PARTE I (sem consulta)
NOTA: Não é permitida a presença de calculadoras e telemóveis na sala

1. **NOTA**: Use **no máximo 40 palavras** para responder a cada uma das 6 alíneas seguintes:

a) Diga o que entende por *Little endian.*

b) Indique quantas e quais são as principais classes de instruções presentes na generalidade dos Sets de instruções dos computadores.

c) Descreva a operação realizada pela instrução do MIPS "cvt.s.w $f0, $f2"

d) Diga quantos e quais os campos em que é decomposta uma instrução do tipo **I** do MIPS. Refira a dimensão de cada campo.

e) Indique as vantagens da estratégia de salvaguarda de registos adoptada pelo MIPS face a uma *caller saved.*

f) A gestão de *stacks*, nas arquitecturas MIPS, adopta um conjunto de regras básicas. Indique quais o.

2. Descreva, na forma de um diagrama de fluxo, o algoritmo da divisão de inteiros implementado por uma arquitectura do tipo MIPS.

3. a) Obtenha a representação binária (em vírgula flutuante, precisão simples) da quantidade armazenada no registo **$f20**, admitindo que, em decimal, o seu valor é **$7.5625 * 2^{-5}$**

b) Considere agora que o conteúdo dos registos **$f10** e **$f12** é respectivamente:

$f10 = 0 01101111 00010000000000000000000
$f12 = 1 01101100 00110000000000000000000

Determine o resultado da instrução **sub.s $f0, $f12, $f10**, realizando, em binário, os passos necessários à sua obtenção pela **FPU**. Explicite, em binário, o conteúdo do registo **$f0** após a execução da instrução.

4. Considere, na **figura 2** fornecida em anexo, o trecho de código *Assembly* ali apresentado. Admita que o valor presente em **$PC** corresponde ao *label* "**fct:**", que nesse instante o conteúdo dos registos é o indicado, e que vai iniciar-se o "*instruction fetch*" da próxima instrução. Considere ainda o *datapath* e a unidade de controlo fornecido na próxima página, no pressuposto de que corresponde a uma implementação simplificada do MIPS de execução multi-ciclo sem *pipelining*.

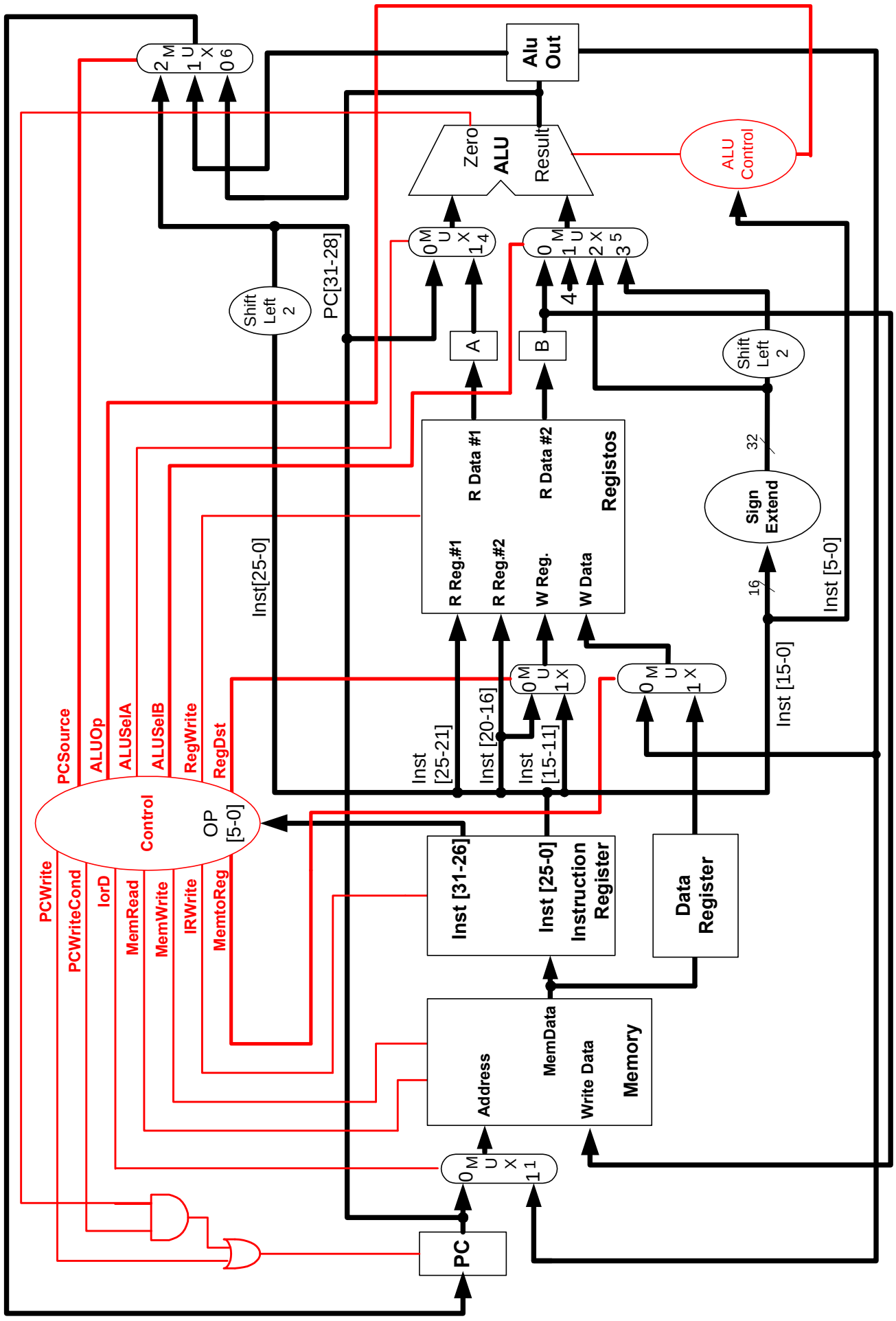
a) Escreva, nas duas colunas do lado direito da fig.2, **em hexadecimal**, o endereço em que se encontra armazenada cada uma das instruções do código fornecido, e o respectivo conteúdo, sabendo que a primeira linha corresponde à instrução presente no *label* "**fct:**".

b) Preencha a tabela fornecida em anexo com o nome de cada uma das fases de execução da **sub $8, $8, $5**, e com o valor que tomam, em cada uma delas, os sinais do *datapath* e os sinais de controlo ali indicados. Admita que o valor lógico "1" corresponde ao estado activo. ___ se esqueça de preencher o cabeçalho (nome, curso e N.M).

c) Determine quais os registos e os endereços de memória (do segmento de dados) alterados pelo código da figura 2, e qual o seu valor no momento em que vai iniciar-se o *instruction fetch* da instrução presente em "**next:"**. Justifique adequadamente a sua resposta.

d) O *datapath* fornecido não suporta a execução da instrução presente no endereço **0x0040002C**. Indique detalhadamente quais as alterações a fazer ao *datapath* fornecido para que essa instrução passe a ser suportada.

Cotações: **1a) a 1f)** – 0,5; **2** – 1.5; **3a)** – 1,5; **3b)** – 1,5; **4a)** – 1,0; **4b)** – 1.5; **4c)** – 1,0; **4d)** – 1,0


Alu Out
ALU Control
Zero
ALU
Result
Shift Left 2
PC[31-28]
A
B
4
Shift Left 2
R Data #1
R Data #2
Registos
R Reg.#1
R Reg.#2
W Reg.
W Data
32
Sign Extend
16
Inst[25-0]
Inst [5-0]
Inst [15-0]
Inst [25-21]
Inst [20-16]
Inst [15-11]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Control
OP [5-0]
PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Inst [31-26]
Inst [25-0]
Instruction Register
Data Register
Address
MemData
Write Data
Memory
PC

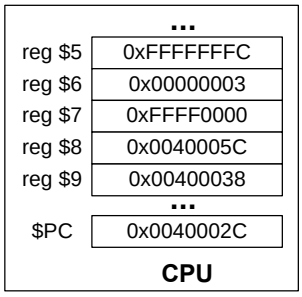
Recorrência **Arquitectura de Computadores I** **21-01-2004**

Nome:_____

Curso:_____ **Nº Mecanográfico:**_____

```
fct:  lui   $8, 0x1001
      ori   $8, $0, 0x1
loop: sw    $7, 0x0B($8)
      addi  $6, $6, -1
      sub   $8, $8, $5
      bne   $6, $0, loop
next: ...
```

OpCode	Funct	Operação
0	0x20	add
0	0x21	addu
0	0x22	sub
0x03		jal
0x05		bne
0x08		addi
0x0D		ori
0x0F		lui
0x23		lw
0x2b		sw



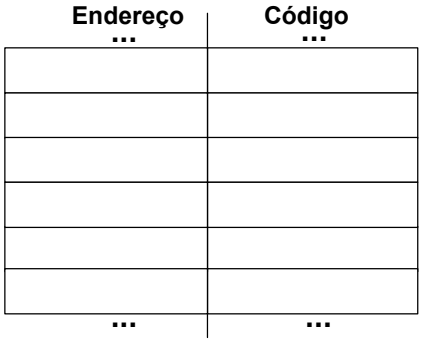


Figura 2 (Problema 4)

	Fase 1	Fase 2	Fase 3	Fase 4	Fase 5
Nome da fase					
Datapath					
A					
B					
Data Register					
ALU Result					
ALU Out					
ALU Zero					
Controlo					
PCSource					
ALUSelA					
ALUSelB					
IorD					
MemtoReg					
RegDst					
PCWrite					
PCWriteCond					
RegWrite					
MemRead					
MemWrite					
ALUOp					
IRWrite					

Arquitectura de Computadores I

Recorrência

Parte Prática 21/01/2004

NOTE BEM: Leia atentamente todas as questões, comente o código usando, preferencialmente, a linguagem C e respeite a convenção de passagem de parâmetros e salvaguarda de registos. Respeite rigorosamente os aspectos estruturais e a sequência de instruções indicadas no código original fornecido, bem como as indicações sobre quais os registos a usar para cada variável.

O programa que se segue controla uma balança electrónica. Internamente possui uma tabela de preços indexada pelo código do produto. Os preços armazenados na tabela representam o custo por grama de cada produto.

Considere que a função `RegistarProdutos` tem o seguinte protótipo:

```
float RegistarProdutos(unsigned int* nProdutos);
```

e que o programa principal tem a seguinte codificação em C:

```
static float tabelaPrecos[] = {1.2, 5.3, 4.7, 8.9, 2.5};

void main(void)
{
      static unsigned int numProdutos;
      float custoTotal;

      while (1)
      {
            custoTotal = RegistarProdutos( &numProdutos );

            print_str("\nNr. de produtos: ");   /* Syscall */
            print_int(numProdutos);             /* Syscall */
            print_str("\nCusto total: ");       /* Syscall */
            print_float(custoTotal);            /* Syscall */
      }
}
```

a) Defina, no segmento de dados, as constantes, as variáveis declaradas como `static`, e codifique em *assembly* do MIPS a função `main`.

b) Considere que as funções `Custo` e `LerPeso` têm os seguintes protótipos:

```
float Custo(unsigned int codProd, unsigned int peso);
unsigned int LerPeso(void);
```

Traduza para *assembly* do MIPS a função `RegistarProdutos` cujo código em C é:

```
float RegistarProdutos( unsigned int* nProdutos )
{
	unsigned int codProd, peso;
	float total = 0.0;

	(*nProdutos) = 0;
	codProd = 0;
	while( codProd != 99 )
	{
		print_str("Código do produto (99 para terminar):"); /* Syscall */
		codProd = read_int();
		if( codProd < 5 )
		{
			peso = LerPeso();

			total = total + Custo(codProd, peso);

			(*nProdutos)++;
		}
		else if( codProd != 99 )
		{
			print_str( "Código errado." );
		}
	}
	return total;
}
```

c) Escreva em C a função `Custo` que determina, e devolve, o custo de uma dada quantidade em gramas (`peso`) do produto cujo código é `codProd`. Traduza para *assembly* do MIPS a função `Custo`. Note que o preço por grama de cada produto é obtido por indexação da tabela `tabelaPrecos` através do código de produto.

```
float Custo(unsigned int codProd, unsigned int peso);
```

Tenha em atenção que o valor de retorno da função é do tipo `float` e deve ser devolvido através do registo `$f0`.