Gabarito da Prova Substitutiva de Arquitetura de Computador

Aluno: _____ No. USP: _____

1ª Questão) (1.0 ponto) Considere que o código em C da função **`int Qx(int)`** a seguir foi compilado para MIPS

```
int Qx (int i) {
  (... lógica em C ...);
}
```

Código MIPS compilado:

```
Qx:
  li $t1, 32
  li $s0, 0
  li $s1, 32
  add $s0, $s0, $a0
  L1: addiu $s0, $s0, 1
  sub $s1, $t1, $s0
  bne $s0, $t1, L1
jr $ra
```

A variável ***i*** foi armazenada em **`$a0`** e que os resultados gerados pela função **`Qx`** estarão em **`$s0`** e **`$s1`**, valores literais (variáveis) no espaço de memória global.
As instruções **`li`** e **`addiu`** correspondem, respectivamente, a um *load* imediato (constante) e um *add* imediato sem sinal (constante positiva).
a) Traduza o código MIPS para C
b) O que faz exatamente essa sequência de código em **`Qx`**?
c) Quais os valores finais armazenados em **`$s0`** e **`$s1`**? Especifique os valores em função do parâmetro ***i*** da função **`Qx`**.

a) 0.5 ponto

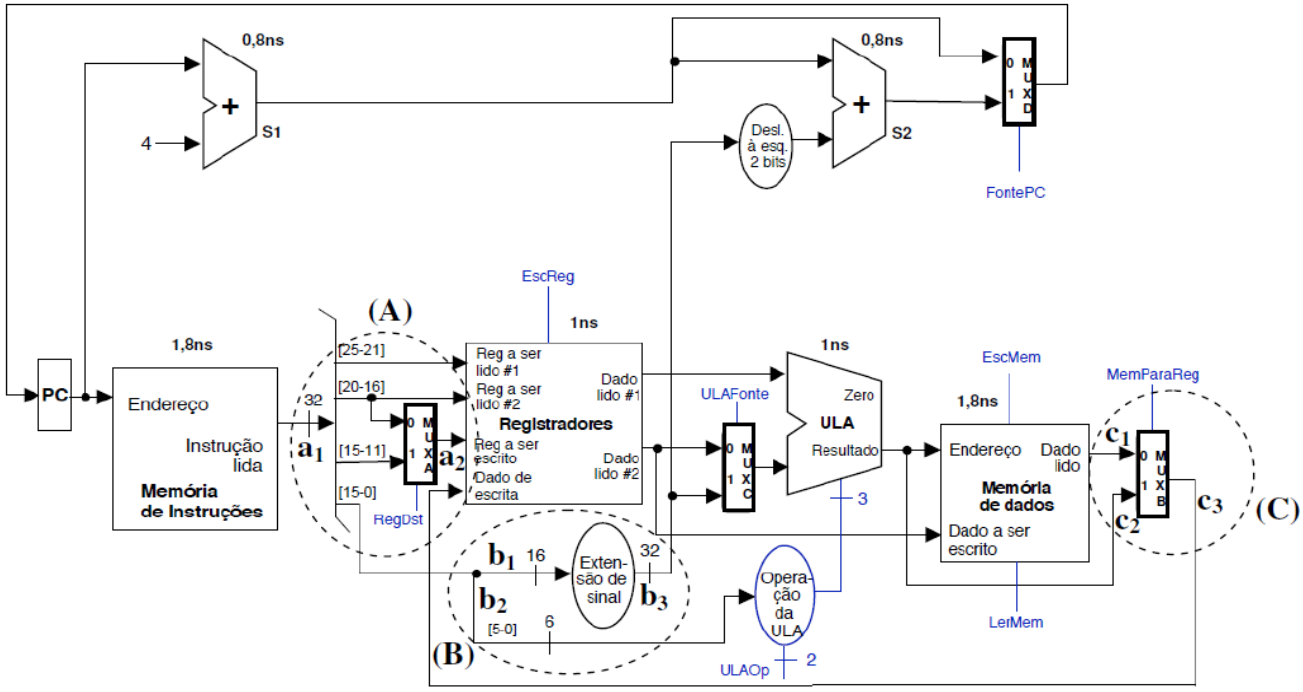
```
void Qx (int i){
        c=32;     //
        a=0      // li $s0,0
        b=32;    // li $s1,32
        a= a + i; // add $s0,$s0,$a0
        Do {       // Corresponde ao Label L1 usado no laço
           a=a+1; // addiu $s0,$s0,1
           b=c-a; // sub $s1,$t1,$s0
         } WHILE(a!=c) // bne $s0, $t1, L1

 }
```

Critério de correção
Colocou um return desconta 0.05
Do { }while pro while { } descontar 0.1
Definição de variável dentro do loop descontar 0.1

b) 0.25
para i<32
Adiciona 32 em $s0 e 0 para $s1
Para i>=32
Loop infinito
c) 0.25
Para i < 32
O número de interações executadas será **`32-i`**.
O valor final de **`a`** (armazenada em **`$s0`**) será: **`32`**
O valor final de **`b`** (armazenada em **`$s1`**) será: **`32-a = 0`**
Ambos os valores independem de **`i`**
Para i > 31 – Loop infinito

2ª Questão) (1.0 ponto) Suponha que a seguinte sequência de instruções deve ser executada nesta arquitetura



Endereço	Instrução	Observações
FA00:0000	**lw $s0,60($t0)**	Os registradores **$t0-$t7** são numerados de **8** a **15** (temporários) e os de **$s0-$s7** de **16** a **23.** (valores salvos/armazenados). O opcode de **lw** é **35**. O opcode da instrução **sub** é **0**, com **34** nos bits [5-0]. O valor (em hexa) de **$t0 = 8AFF:0000**. Os valores carregados em **$s0** e **$s0** após a execução da instrução **lw** serão, respectivamente, **12** e **25**. Campos de bits não utilizados/desnecessários para a classe instrução terão valor 0.
FA00:0004	**lw $s1,64($t0)**	
FA00:0008	**sub $s2,$s0,$s1**	

a) Mostre quais os valores dos conjuntos de bits (a_1 a c_3) que estão sendo enviados em cada uma das partes

$t0 = 8AFF:0000 – 1000 1010 1111 1111 0000 0000 0000 0000

	Opcode	Rs ($t0)	Rt ($s0)	Rd	Shant	Funct
lw $s0,60($t0)	35	8	16	60		
lw $s1,64($t0)	35	8	17	64		
sub $s2,$s0,$s1	0	16	17	18	0	34
Posição	31-26	25--21	20-16	15-11	11-6	5-0

	Opcode	Rs	Rt	Rd		Funct
lw $s0,60($t0)	100011	01000	10000	00000	00000	111100
lw $s1,64($t0)	100011	01000	10001	00000	00001	000000
sub $s2,$s0,$s1	000000	10000	10001	10010	00000	100010
	31-26	25-21	20-16	15-11	10-6	5-0

Extraindo o termos da instrução acima

lw $s0,60($t0)	a1	100011 10000 01000 00000 00000 111100 Corresponde a instrução com 32 bits
	a2	01000 **Registrado de destino**
	b1	00000 00000 111100
	b2	111100
	b3	0000 0000 0000 00000 00000 111100
	c1 valor 12	**Valor lido da memória (foi dado)** 0000 0000 0000 0000 0000 0000 0000 1100
	c2	**Endereço de acesso a memória** $t0+ 60 1000 1010 1111 1111 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0011 1100 = 1000 1010 1111 1111 0000 0000 0011 1100
	c3	**Valor lido da memória (foi dado)** 0000 0000 0000 0000 0000 0000 0000 1100
lw $s1,64($t0)	a1	100011 01000 10001 00000 00001 000000
	a2	10001
	b1	00000 00001 000000
	b2	000000
	b3	0000 0000 0000 00000 00001 000000
	c1 valor 25	0000 0000 0000 0000 0000 0000 0001 1001
	c2	$t0+ 60 1000 1010 1111 1111 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 0000 = 1000 1010 1111 1111 0000 0000 0001 0000

	c3	0000 0000 0000 0000 0000 0000 0001 1001
sub $s2,$s0,$s1	a1	000000 10000 10001 10010 00000 100100
	a2	10010
	b1	10010 00000 100100
	b2	100100
	b3	0000 0000 0000 10010 00000 100100
	c1	xxxx xxxx xxxx xxxxx xxxx xxxxx xxxx xxxx
	c2 valor -13	0000 0000 0000 0000 0000 0000 0000 1101 Inverte 1111 1111 1111 1111 1111 1111 1111 0010 Soma 1 1111 1111 1111 1111 1111 1111 1111 0011 (complemento de dois)
	c3 valor -13	1111 1111 1111 1111 1111 1111 1111 0011

Cada item 0.033

3ª Questão) (2,0 pontos) Apresente a implementação em linguagem de máquina dos códigos abaixo

```
Enquanto i<100
    v[i+1] ← v[i] +1;
    se v[i+1] <> 10
        a++;
    fim se;
    i ← i+1;
    b ← c*2 -d;
fim enquanto
```

```
int fact(int n)
{
    if (n < 1)
        return 1;
    else
        return (n*fact(n-1));
}
```

<> **diferente**

* função fact passagem de parâmetros pelo registrado $a0, * a, b, c, d, i estão nos registrados $t0, $t1, $t2, $t3 e $t4

Valor 1.0 Ponto

```
Loop:  slti  $t5, $t4, 100
       beq  $t5, $zero, FIM
       sll   $t6, $t4, 2          #multiplica i por 4
       add   $t6, $t6, $s0        # soma a base
       lw     $s1, 0($t6)         #carrega v[i]
       addi   $s1, $s1, 1         #v[i]+1
       addi   $t7, $t4, 1         #i=i+1
       slti    $t7, $t7, 2        #multiplica i+1  por 4
       add   $t7, $t7, $s0        #soma a base
       sw  $s1, 0($t7)
       addi $t8, $zero, 10
       beq $s1, $t8, FSE          #IF
       addi  $t0, $t0, 1          #a = a+1
FSE:   addi  $t4, $t4, 1          # i = i + 1
       multi $t6, $t2, 2
       sub $t1, $t6,$t3
```

```
        j   Loop
FIM:
```

Esqueceu de multiplicar indice por 4
Cada instrução – 0.05

Valor 1.0 Ponto

```
fact:
        addi $sp, $sp, -12
        sw $ra, 8($sp)
        sw $fp, 4($sp)
        sw $ra, 0($sp)
        move $fp, $sp
        slti $t0, $a0, 1
        beq $t0, $zero, L1
        addi $v0, zero, 1
        addi $sp, $sp, 12
        jr $ra
L1:
        addi $a0, $s0, -1
        jal fact
        lw $a0, 0($sp)
        lw $fp, 4($sp)
        lw $ra 8($sp)
        addi $sp, $sp, 12
        mult $va0, $a0, $v0
        jr $ra
```

4ª Questão) (1,0 pontos) As arquiteturas de uso geral atuais são normalmente referenciadas como máquinas von Neumman. Quais as principais características dessa arquitetura e sua principal limitação?

Características – 0.5 ponto

Conceito de programa armazenado; 0.25

- *Dados e instruções armazenados em uma única memória de leitura e escrita.*

Endereçamento da memória por posição e não pelo tipo; 0.25
Execução sequencial de instruções; e
Único caminho entre memória e CPU.

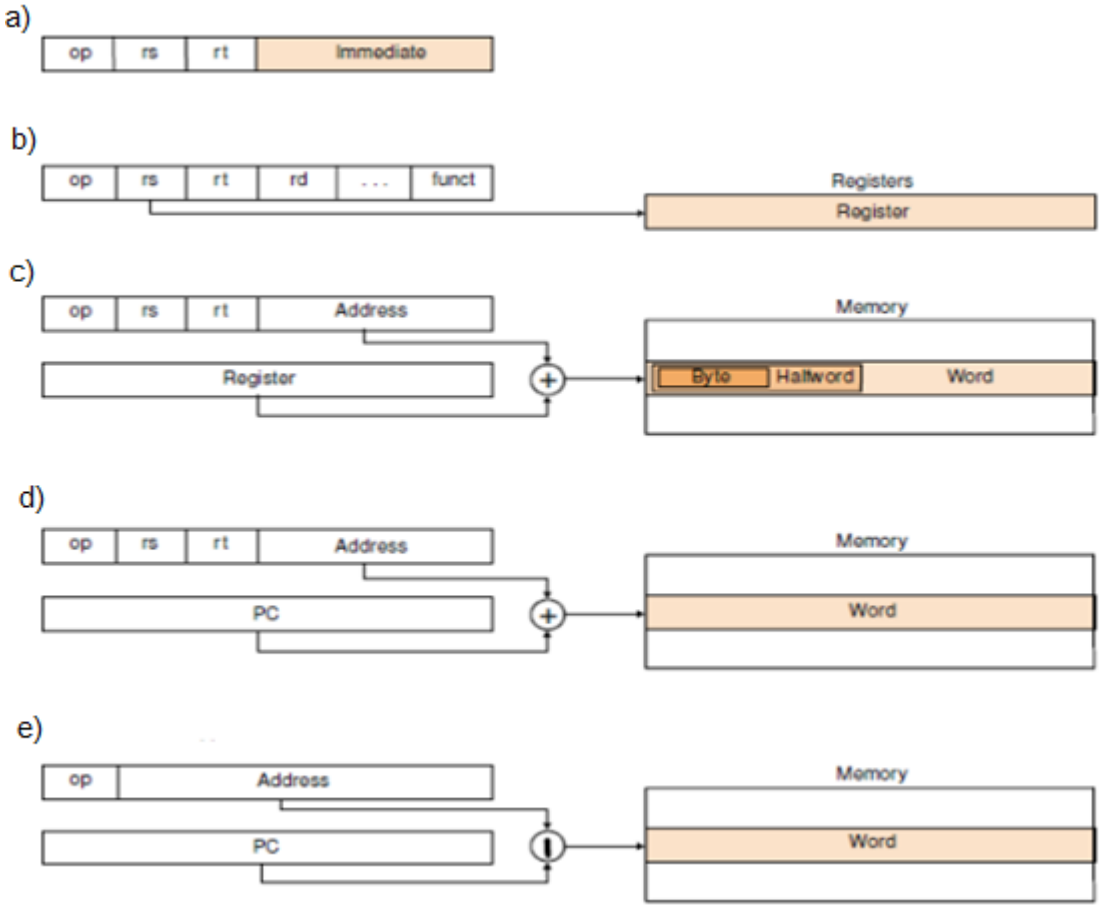
Limitação – 0.5 ponto

Tráfego intenso no barramento do sistema

- *Principal rota de informação: CPU e memória (ponto crítico)*
- *Constante fluxo de dados e instruções*

Gera desperdício de tempo (CPU em espera)
Agrava-se gradativamente pelo aumento do gap de velocidade entre a memória e a CPU

5ª Questão) (1,5 pontos) Explique como funcionam as 5 formas de endereçamento básicas da arquitetura MIPS mostradas. Compare as em termos de complexidade de execução da busca/utilização dos operandos:



Cada item – 0.3

a) *Endereçamento imediato: o operando é uma constante dentro da própria instrução. Operações aritméticas e de comparação: ADDI R1, R2, #A*

b) *Endereçamento a registrador: O operando está em um registrador e a instrução contem o número do registrador. ADD R3, R3, R7*

c) *Endereçamento via registrador-base ou via deslocamento: operando está no endereço de memória obtida com a soma do conteúdo do registrador-base com a constante armazenada na instrução (Const ($reg)). Instruções de acesso à memória*

d) *Endereçamento relativo ao PC: operando está no endereço formado pela soma do conteúdo do PC com a constante obtida na própria instrução. Instrução branch. o número de instruções a serem puladas a partir da instrução é especificado na instrução*

e) *Endereçamento pseudo-direto: o endereço de desvio é formado pela concatenação dos 28 bits (26 bits estendido) obtido da instrução com os bits mais significativos do PC*

6ª Questão) (1,5 ponto) Escreva os códigos em linguagem de máquina para computar a expressão **X=(A+B×C) / (A–B×C)** para as duas arquiteturas com instruções em linguagem de máquina disponíveis e formas de endereçamento para operações aritméticas conforme a tabela a seguir. Em seguida, determine o número de bytes de instruções buscados na memória e o número de bytes de dados trocados entre a memória e a CPU.

Forma de Endereçamento	
Registrador-Memória	Registrador
`Load R1,M`	`Load R1,M`
`Store R1,M`	`Store R1,M`
`Add R3,R2,M`	`Add R3,R2,R1`
`Sub R3,R2,M`	`Sub R3,R2,R1`
`Mult R3,R2,M`	`Mult R3,R2,R1`
`Div R3,R2,M`	`Div R3,R2,R1`

OBSERVAÇÕES:

1. **Considere** possíveis **otimizações** no código
2. M é um endereço de memória 16 bits
3. Ri é um registrador de 4 bits
4. Os *opcodes* possuem 8 bits
5. As instruções devem ter **comprimentos múltiplos de 4 bits**
6. Note que a linguagem **não é MIPS**

Tabela REG – REG → 0.5 ponto
Buscado – 0.15
Trocado – 0.1
Tabela REG – MEM → 0.5 ponto
Buscado – 0.15
Trocado – 0.1

Considerando Add R1, R2, E # E ← R1+R2

Assembly REG-MEM	Significado	Bits Instrução	Bits Dados
Load R1, B	R1 = B	28	16
Load R2, C	R2 = C	28	16
Mult R1, R2, D	D = B x C	32	16
Load R1, A	R1 = A	28	16
Load R2, D	R2 = B x C	28	16
Add R1, R2, E	E = (A+BxC)	32	16
Sub R1, R2, F	F = (A-BxC)	32	16
Load R1, E	R1 = (A+BxC)	28	16
Load R2, F	R2 = (A-BxC)	32	16
Div R1, R2, X	X= (A+BxC)/ (A-BxC)	32	16

Considerando Add R1, R2, E # R1 ← R2+E

Assembly REG-MEM	Significado	Bits Instrução	Bits Dados
Load R1, B	R1 = B	28	16
Mult R1, R1, C	R1 = R1 x C = B x C	32	16
Store R1, D	D = R1 = B x C	28	16
Load R1, A	R2 = A	28	16
Add R2, R1, D	R2 = A + BxC	32	16
Sub R3, R1, D	R3= A - B*C	32	16
Store R3, D	D =R3 = A – Bx C	28	16
Div R3, R2, D	R3 =(A+BxC)/(A-BxC)	32	16
Store R3, X	X = (A+BxC)/(A-BxC)	28	16

Considerando Add R1, R2, R3 # R1 ← R1+ R3

Assembly REG-REG	Significado	Bits Instrução	Bits Dados
Load R1, B	R1 = B	28	16
Load R2, C	R2 = C	28	16
Load R3, A	R3 = A	28	16
Mult R4, R1, R2	R4 = (BxC)	20	0
Add R5, R3, R4	R5 = (A+BxC)	20	0

Sub R4, R3, R4	R4 = (A-BxC)	20	0
Div R3, R5, R4	R3 = (A+BxC)/(B-AxC)	20	0
Store R3, X	X=(A+BxC)/(D-ExF)	28	16

Duas instruções de multiplicação descontar 0.1
Não atribui os valores a M (A, B, C) descontar 0.1
Usar registrados R1A – descontar 0.05
Esqueceu uma instrução – descontar 0.1
Instrução errada – descontar 0.1
Troca entre Bits Instrução e Bits Dados – considera apenas 0.1 (se estiver tudo certo)

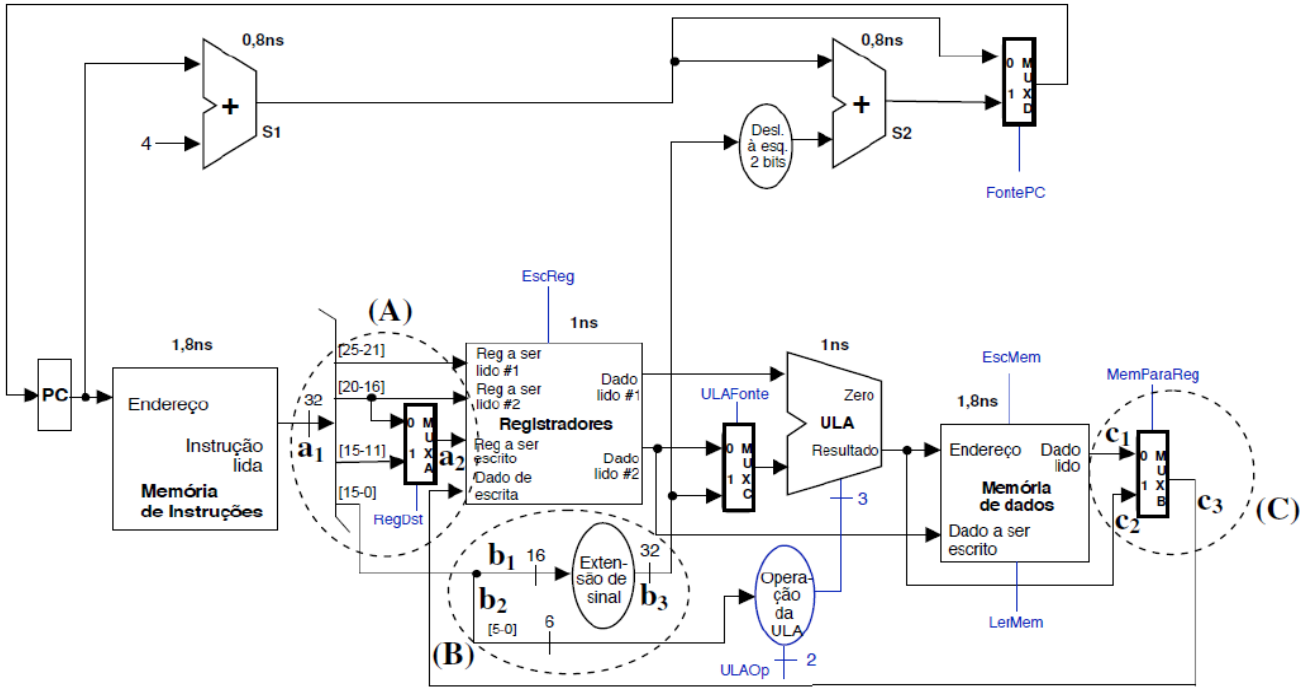
7ª Questão) (2,0 pontos) Considere a organização do bloco de dados multiciclo abaixo, que acomoda a execução de um subconjunto da arquitetura do conjunto de instruções do processador MIPS. Considere também as modificações que foram necessárias realizar nesta. Em seguida, responda às questões abaixo.
a) Utilize a figura da questão 1 e descreva todos os caminhos do bloco de dados e sinais de controle efetivamente usados pela instrução BEQ Rs, Rt, Rótulo e JUMP Rótulo. Isto significa descrever em texto todos os caminhos por onde passa informação útil relevante à execução da instrução, ou seja, os dados e endereços que esta realmente necessita manipular.

Beq Rs, Rt

Primeiro ciclo a instrução é buscada na memora e o PC é incrementado, o sinal FontePC é setado igual a 0. No segundo ciclo, a instrução é decodificada, o sinal de EscReg e RegDst é setado igual a 1 e X respectivamente. No terceiro ciclo, a subtração entre o conteúdo dos registradores Rs, Rt é realizada e os sinais de controle ULAOp e ULAFonte é setado igual a 01 a 0, respectivamente. Se o conteúdo destes registradores são iguais , a ULA gera um sinal igual a 1. Este sinal deve entrar numa porta and juntamente o sinal de controle (branch) vindo da unidade de controle. A saída da porta and é usada para setar multiplexador, tal que a saída deste seja a soma do PC+4 com os 16 bits da instrução estendido. Este valor é usado para atualizar o PC. Os outros sinais de controle (EscMem, LerMem, MemParaReg) são setados para 0.

JUMP Rótulo

Primeiro ciclo a instrução é buscada na memora e o PC é incrementado, o sinal FontePC é setado igual a 0. No segundo ciclo, a instrução é decodificada, o sinal de EscReg e RegDst é setado igual a 1 e X respectivamente. No terceiro ciclo, o endereço contendo 26 bits deslocado de 2 bits é concatenado com os bits mais significativos do PC. Os sinais de controle ULAOp e ULAFonte pode ser X, já os sinais de controle (EscMem, LerMem, MemParaReg) devem ser setados para 0.



b) Diga qual operação é executada pela unidade lógica-aritmética (ALU) no terceiro ciclo de relógio da instrução BEQ Rs, Rt, Rótulo e JUMP Rótulo, justificando sua resposta.

Como já descrito no ítem anterior a ULA faz uma soma de suas entradas op1 e op2, para obter o endereço para onde saltar, caso o salto seja executado.

c) Qual a função da ULA na execução da instruções em cada ciclo das instruções na organização multiciclo.

BEQ Rs, Rt, Rótulo

Primeiro ciclo *- Incrementa o PC*

Segundo ciclo *- Cálculo o endereço de destino do desvio*

Terceiro ciclo *- depende da instrução*

Referência à memória - calcula o endereço de memória

Instrução R - realiza a operação especificada pelo código de função com dois valores lidos do banco de registradores

Desvio condicional - realiza a comparação de igualdade entre os dois registradores lidos na etapa anteriorr

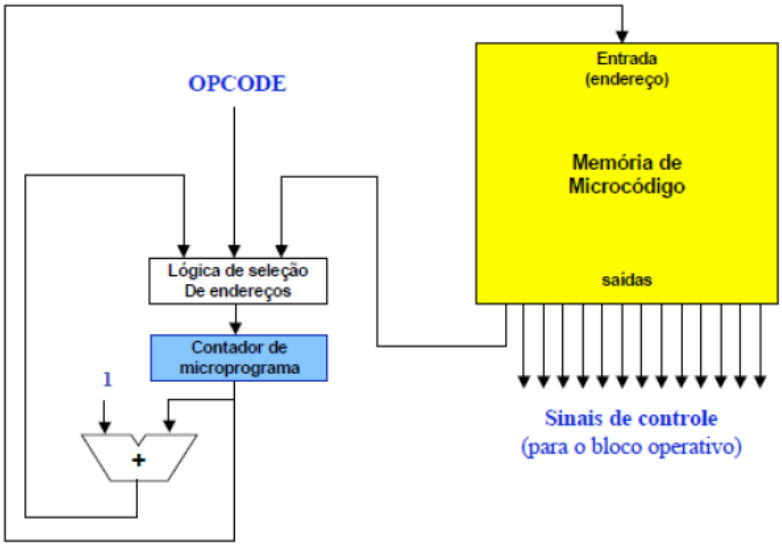
JUMP Rótulo - não realiza nenhuma operação

Quarto ciclo *- Acesso a memória ou conclusão da instrução tipo R*

Quinto ciclo *- Conclusão da leitura da memória*

d) Explique ***detalhadamente*** como funciona uma unidade de controle microprogramado.

È uma técnica de implementação de controladores lógicos que tem como objetivo administrar o correto funcionamento das atividades dentro e fora da CPU. Segue a figura que corresponde a implementação de um controle microprogramado. Basta explicar como funciona.


OPCODE
Entrada
(endereço)
Memória de
Microcódigo
saídas
Lógica de seleção
De endereços
Contador de
microprograma
1
+
Sinais de controle
(para o bloco operativo)