Nome: N. Mec:

Grupo I

Responda às questões seguintes, **mostrando todos os passos da sua resposta**:

a) O valor dos 16 bits menos significativos, expresso em hexadecimal, do código máquina da instrução **`L1: beq $1,$1,L1`** é:

```
offset = (target_addr - (PC + 4)) >> 2
target_addr = PC
offset = (PC - (PC + 4)) >> 2 = -4 >> 2 = -1
Então o valor dos 16 lsb é: 0xFFFF
```

b) Suponha que **`$2=0xA35`** e que se pretende aceder, através da instrução **`LW`**, ao endereço de memória **`0xA31`**. Para que isso aconteça, o valor dos 16 bits menos significativos, expresso em hexadecimal, do código máquina da instrução "**`lw $3,??($2)`**" deve ser:

```
Address = 0x0A31 = $2 + offset => offset = 0x0A31 - $2
offset = 0x0A31 - 0x0A35 = -4
O valor dos 16 lsb é: 0xFFFC
```

c) O valor **`0xAC640100`** é o código máquina de uma instrução do MIPS. Apresente a instrução *Assembly* completa a que corresponde esse código (mnemónica e argumentos – consulte a tabela de códigos disponível no Grupo III):

```
0xAC640100 = 101011 00011 00100 0000000100000000
opcode = 0x2B => SW (codificada com o formato I)
Formato de codificação: SW $rt,offs($rs)

      101011 00011 00100 0000000100000000
      op=2B  $rs=3 $rs=4 ofs=0x0100
Então a instrução é: SW $4,0x0100($3)
```

d) Admita que os valores indicados no *datapath* da Figura 1 correspondem à "fotografia" tirada no decurso da execução de uma dada instrução. Observe todos os sinais e valores presentes nessa figura e responda às seguintes questões:

1) A fase de execução em que se encontra é **`Write-back`**, porque **`o sinal RegWrite está ativo, o que só acontece nesta fase`**

2) A instrução que se encontra em execução é (não necessita de colocar a instrução completa) **`LW (load word)`**, porque **`para além do RegWrite ativo, MemtoReg está a '1', i.e., o valor a escrever no Register File vem da memória. Pode ainda observar-se que RegDst está a '0' (Reg destino codificado no campo RT)`**

3) A instrução que vai ser executada de seguida encontra-se no endereço **`0x0040005C`**, porque **`é esse o valor presente à saída do registo Program Counter. A instrução está na fase WB (5ª fase) e o Programa counter já foi incrementado na fase ID (1ª fase)`**

4) Admita que o sinal "**`RegWrite`**" tinha o valor '**`0`**'. Podia então concluir-se que a instrução em execução estava na fase **`ID/Instr decode/Op fetch`**, porque **`o ALUSelA é '0', o ALUSelB é 11 e o ALUop é "00"; isto significa que a ALU está a somar o valor do PC com o offset extendido para 32 bits e deslocado 2 bits à esquerda, ou seja está a calcular o Branch Target Address, o que acontece na fase ID`**

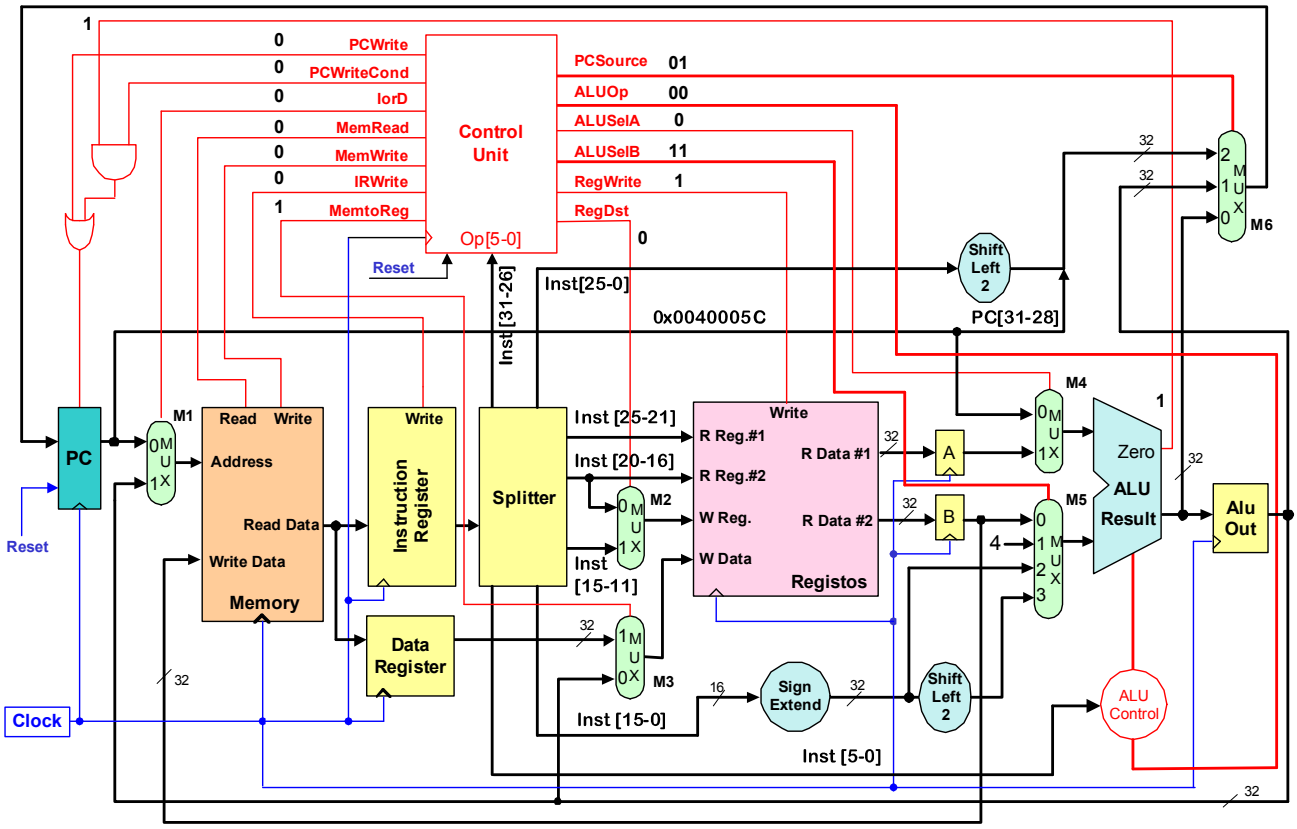


Figura 1. *Datapath multi-cycle* do MIPS.

Zona de rascunho

Nome: N. Mec:

Grupo II

Considere o *datapath* e a unidade de controlo fornecidos na Figura 1, sabendo que corresponde a uma implementação *multi-cycle* simplificada do MIPS, sem *pipelining*.

a) Preencha a tabela seguinte com o nome de cada uma das fases de execução da instrução "**`xor $1,$2,$3`**" e com o valor que tomam, em cada uma delas, os sinais de controlo ali indicados. Admita que o valor lógico "1" corresponde ao estado ativo. **Assinale as situações de "don't care" com "X"**.

Designação da Fase	Instruction Fetch/Calc. PC+4	ID/Inst Dec Op fetch	Execute	Write-Back	
PCWrite	1	0	0	0	
MemWrite	0	0	0	0	
IRWrite	1	0	0	0	
ALUOp	00	00	10	XX	
ALUSelA	0	0	1	X	
ALUSelB	01	11	00	XX	
IorD	0	X	X	X	
PCSource	00	XX	XX	XX	
MemRead	1	0	0	0	
RegWrite	0	0	0	1	
RegDst	X	X	X	1	

b) Preencha a tabela seguinte com o nome de cada uma das fases de execução da instrução "**`xor $1,$2,$3`**" (código máquina **`0x00430826`**) e com o valor que tomam, em cada uma delas, os valores do *datapath* ali indicados. Considere que os registos, no instante em que vai iniciar-se o *instruction fetch*, têm os seguintes valores: **`$1=0x145`**, **`$2=0x3A4`**, **`$3=0x75D`**, **`PC=0x0040008C`**). **Assinale as situações de "valor desconhecido" com "?"**.

Designação da fase	Instruction Fetch/Calc. PC+4	ID/Inst Dec Op fetch	Execute	Write-back	
PC	0x0040008C	0x00400090	0x00400090	0x00400090	
Instr. Register	?	0x00430826	0x00430826	0x00430826	
Data Register	?	0x00430826	?	?	
A	?	?	0x3A4	0x3A4	
B	?	?	0x75D	0x75D	
ALU Result	0x00400090	0x00402128	0x000004F9	?	
ALU Out	?	0x00400090	0x00402128	0x000004F9	
ALU Zero	0	0	0	?	

`ALUOp – 00: Add, 01: Subtract, 10: R-Type, 11: Set if Less Than`

```
Fase ID: ALU_res = (0x00000826 << 2) + PC = 0x00002098 + 0x00400090
                 = 0x00402128
Fase EX: ALU_res = (0x3A4 XOR 0x75D) = 0x4F9
```

Nome: N. Mec:

Grupo III

Considere o trecho de código da tabela ao lado onde o endereço representado pelo *label* **L1** é **0x0000A204**.

a) Traduza para código máquina do MIPS as instruções abaixo indicadas (expressando o resultado em hexadecimal) e indique o endereço de memória em que se encontra cada uma. Mostre todos os passos da sua resposta e, no final, **preencha a tabela**.

Opcode	*Funct*	Instr.
0	0x20	add
0	0x26	xor
0x04		beq
0x2B		sw
0x02		j
0x08		addi
0x0A		slti
0x23		lw

```
L1:  addi $2,$0,0x14
     add  $3,$0,$0
L2:  beq  $3,$2,L3
     lw   $4,0x100($3)
     sw   $4,0x200($3)
     addi $3,$3,4
     j    L2
L3:  ...
```

sw $4,0x200($3)

Formato I; SW $rt,offs($rs)
op = 0x2B
101011 00011 00100 , 0x0200 = 1010 1100 0110 0100 , 0x0200
= AC640200

j L2

Formato J
op = 0x02
Target address = 0x0000A20C; Val_instr = 28lsb(target address) >> 2
Val_instr = (0x000A20C >> 2) = 0x0002883
Cod máquina = 000010 00000000000010100010000011
= 0000 1000 0000 0000 0010 1000 1000 0011
= 0x08002883

Endereço	Instrução	Código Máquina (hexadecimal)
0xA214	sw $4,0x200($3)	0xAC640200
0xA21C	j L2	0x08002883

b) Calcule o número total de ciclos de relógio que demora a execução completa desse trecho de código (desde o instante inicial do *instruction fetch* da primeira instrução até ao momento em que vai iniciar-se o *instruction fetch* da instrução presente em "L3:"): i) num *datapath single-cycle*; ii) num *datapath multi-cycle*. **Apresente todos os passos que justifiquem a sua resposta (a simples apresentação de valores sem justificação adequada terá cotação 0)**.

i) Single-cycle:

Inicialmente: $2=0x14, $3=0
O $3 é incrementado de 4 todos os ciclos; o loop termina quando $3 fica igual a $2, ou seja, quando $3=0x14. Assim, o ciclo executa com os seguintes valores de $3: 0x0, 0x4, 0x8, 0xC, 0x10 (5 vezes).
N° de ciclos = nr_instruções executadas
= (addi+add) + 5 * (beq+lw+sw+addi+j) + beq = 2 + 5 * 5 + 1 = 28 T

ii) Multi-cycle:

N° de ciclos de relógio por instrução:
addi: 4T, add: 4T, beq: 3T, lw: 5T, sw: 4T, j: 3T

Nr_ciclos = (Taddi+Tadd) + 5 * (Tbeq+Tlw+Tsw+Taddi+Tj) + Tbeq
Nr_ciclos = (4 + 4) + 5 * (3 + 5 + 4 + 4 + 3) + 3
= 8 + 5 * 19 + 3 = 106 T

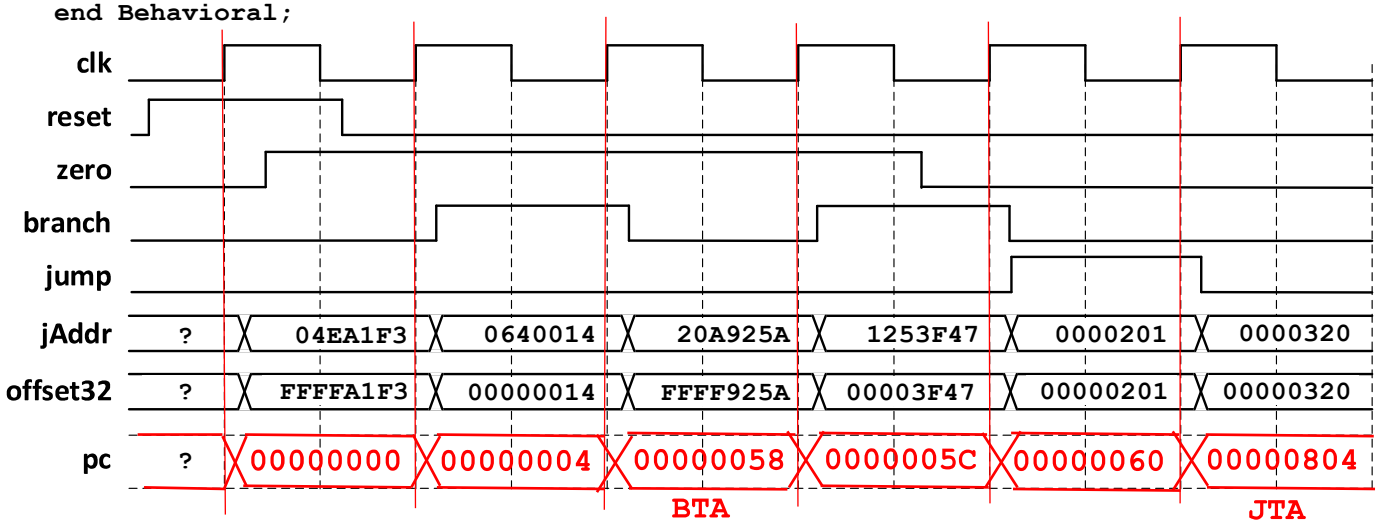
Nome: N. Mec:

Grupo IV

a) O código VHDL que se apresenta de seguida corresponde a uma possível implementação do módulo "PC_update" para a arquitetura ***single-cycle*** do MIPS que implementou nas aulas práticas, responsável pela manutenção e atualização do valor do *Program Counter*. Complete o diagrama temporal da figura seguinte, calculando o valor de saída ("**pc**") para todos os ciclos de relógio ali apresentados (note que os valores de "**jAddr**" e "**offset32**" estão representados em hexadecimal).

```
entity PC_update is
  port(clk, reset, branch, jump, zero : in std_logic;
       offset32  : in std_logic_vector(31 downto 0);
       jAddr     : in std_logic_vector(25 downto 0);
       pc        : out std_logic_vector(31 downto 0));
end PC_update;

architecture Behavioral of PC_update is
  signal s_pc, s_offsetSL2 : unsigned(31 downto 0);
  signal s_pc4 : unsigned(31 downto 0);
begin
  s_offsetSL2 <= unsigned(offset32(29 downto 0)) & "00";
  s_pc4 <= s_pc + 4;
  process(clk)
  begin
    if(rising_edge(clk)) then
      if(reset = '1') then
        s_pc <= (others => '0');
      else
        if(jump = '1') then
          s_pc <= s_pc4(31 downto 28) & unsigned(jAddr) & "00";
        elsif(branch = '1' and zero = '1') then
          s_pc <= s_pc4 + s_offsetSL2;
        else
          s_pc <= s_pc4;
        end if;
      end if;
    end if;
  end process;
  pc <= std_logic_vector(s_pc);
end Behavioral;
```



BTA = (PC+4) + (14 << 2) = 00000008 + 50 = 0x00000058

JTA = 4msb(pc+4) concat. (jAddr << 2) = 0201 << 2 = 0x0804

Grupo V

a) Complete o código VHDL seguinte com a implementação de um registo de N bits com *reset* síncrono e *enable*. A ação de *reset* não deve depender do sinal *enable*.

```
entity RegisterN is
   generic ( N : positive                          );
   port ( clk     : in  std_logic ;
          enable  : in  std_logic ;
          reset   : in  std_logic ;
          din     : in  std_logic  vector(N-1 downto 0);
          dout    : out std_logic  vector(N-1 downto 0)  );
end RegisterN;

architecture behav of RegisterN is
begin
.....
.....
    process(clk)
    begin
.....
        if(rising_edge(clk)) then
.....
            if(reset = '1') then
.....
                dout <= (others => '0');
.....
            elsif(enable = '1') then
.....
                dout <= din;
.....
            end if;
.....
        end if;
.....
    end process;
.....
.....
.....

end behav;
```

Zona de rascunho