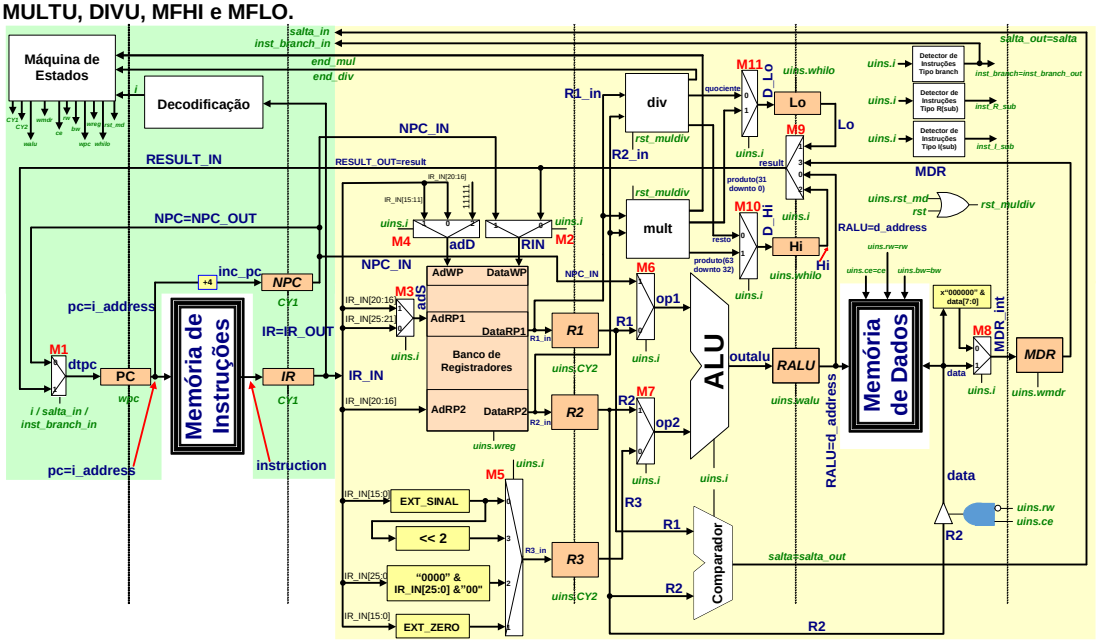
Prova G2 **Disciplina: Organização e Arquitetura de Processadores**

Professor: Ney Laert Vilar Calazans

Aluno: 06/julho/2022

Organização MIPS_S - Instruções com suporte para executar neste hardware - ADDU, SUBU, AND, OR, XOR, NOR, SLL, SLLV, SRA, SRAV, SRL, SRLV, ADDIU, ANDI, ORI, XORI, LUI, LBU, LW, SB, SW, SLT, SLTU, SLTI, SLTIU, BEQ, BGEZ, BLEZ, BNE, J, JAL, JALR, JR, MULTU, DIVU, MFHI e MFLO.



1. (3,0 pontos). Considere a organização MIPS_S cujo diagrama de blocos é ilustrado acima. Trata-se da versão vista em aula e que serviu de base para a disciplina.
   a) (1,0 ponto). Observe o trecho VHDL abaixo, que representa parte da descrição de hardware deste processador, a título ilustrativo.

```
M4: adD <= "11111"          when uins.i=JAL       else
    IR_IN(15 downto 11)           when (inst_R_sub='1' or uins.i=SLTU or uins.i=SLT
          or uins.i=JALR or uins.i=MFHI or uins.i=MFLO or uins.i=SSLL
          or uins.i=SLLV or uins.i=SSRA or uins.i=SRAV or uins.i=SSRL
          or uins.i=SRLV)                                  else
    IR_IN(20 downto 16);
```

   Suponha que existe uma falha no hardware produzido pelo VHDL acima. A falha é que o hardware que gera o sinal `adD` eternamente "gruda" este sinal no sinal `IR[15:11]`. A pergunta aqui é qual(quais) das 37 instruções será(ão) afetada(s) por esta falha. Justifique sua resposta.

   b) (2,0 pontos). Deseja-se dotar a organização multiciclo dada com a capacidade de executar a instrução BLTZ. Supondo que o hardware é alterado para dar suporte a esta instrução, diga que registradores da organização relevantes à execução da BLTZ são escritos e com que dados em cada ciclo da execução desta instrução. Mencione cada registrador relevante, tanto do Bloco de Controle como do Bloco de Dados. Para executar BLTZ, parte do hardware do processador deve ser modificado. Cite ***uma*** modificação necessária a ser realizada no Bloco de Dados e ***uma*** a ser realizada no Bloco de Controle para dar suporte à execução da instrução BLTZ.

2. (2,0 pontos). Considere o trecho de programa abaixo, um misto de linguagem de montagem e código objeto do MIPS. O trecho ocupa endereços de memória que iniciam em **0x00400110**. Ele foi gerado a partir de uma descarga (do inglês, *dump*) da memória de instruções produzida pelo montador MARS. Observe as linhas abaixo e faça o que se pede nos itens a seguir.
   a) Gere os códigos intermediário e fonte correspondentes às instruções das linhas [1], [5], [8] e [9] do trecho de programa.
   b) A partir das informações existentes na linha [4], gere o código intermediário que deveria estar no lugar de `???`. Além disto, diga em que endereço de memória inicia a subrotina `insere_vertice`.

Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando cada passo.

Importante: Valores imediatos devem ser representados em decimal no código fonte e em hexadecimal no código intermediário. Endereços de memória devem ser representados como rótulos no código fonte e como valores hexadecimais no código intermediário.

```
[1]   0x00400110  0x27bdfff4  ???                      150   ???
[2]   0x00400114  0xafbf0000  sw $31,0x00000000($29)  151   sw    $ra,0($sp)
```

```
[3]    ...
[4]    0x00400120  0x0c100037  ???                        154   jal   insere_vertice
[5]    0x00400124  0x8fbf0000  ???                        155   ???
[6]    ...
[7]    0x00400130  0x8d0e0008  lw $14,0x00000008($8)   160  ins_dir:lw   $t6,8($t0)
[8]    0x00400134  0x15c00002  ???                        161   ???
[9]    0x00400138  0xad090008  ???                        162   ???
[10]   0x0040013c  0x03e00008  jr $31                     163   jr    $ra
[11]   0x00400140  0x8d080008  lw $8,0x00000008($8)    164  rec_dir:lw   $t0,8($t0)
```

3. (3,0 pontos). Abaixo aparece um programa em linguagem de montagem do MIPS. Para este programa, responda às questões que seguem:
   a) (1,0 ponto). Diga quantos bytes este programa ocupa em memória (área de instruções, área de dados e total do programa).
   b) (2,0 pontos). Calcule o tempo de execução do programa em nanossegundos, supondo que se trata de uma organização **MIPS_S** executando em uma frequência de 200MHz, e assumindo que a organização foi modificada para dar suporte à execução da instrução **syscall** em exatamente 4 ciclos de relógio.

```
1          .data
2    str:  .asciiz "Alo Mamae"
3    case: .word   0x0
4          .text
5    main:
6          la       $t0,str
7          lw       $t1,0($t0)
8          andi     $t6,$t1,0xff
9          la       $t3,case
10         lw       $t4,0($t3)
11         addiu    $t5,$zero,4
12   loop:
13         blez     $t6,end
14         addiu    $t5,$t5,-1
15         sltiu    $t2,$t6,65
16         bne      $t2,$zero,nxt_ch
17         sltiu    $t2,$t6,91
18         beq      $t2,$zero,nxt_ch
19         addiu    $t4,$t4,1
20   nxt_ch:
21         beq      $t5,$zero,endw
22         addu     $t6,$t1,$zero
23         srl      $t6,$t6,8
24         addu     $t1,$t6,$zero
25         andi     $t6,$t6,0xff
26         j        loop
27   endw:
28         addiu    $t0,$t0,4
29         lw       $t1,0($t0)
30         andi     $t6,$t1,0xff
31         addiu    $t5,$zero,4
32         j        loop
33   end:
34         sw       $t4,0($t3)
35         li       $v0,10
36         syscall
```

Dicas: (a) Cuidado com a contabilização de pseudo-instruções.

4. (2 pontos) Dado o trecho de programa abaixo, a executar no MIPS, suponha uma implementação do MIPS sem **nenhuma** capacidade de solução de conflitos de dados. Mostre o diagrama pipeline para a execução das instruções do trecho durante **apenas os primeiros 22 ciclos de relógio**, supondo que $5 aponta para o início de um vetor de 2 posições com o conteúdo abaixo, em palavras: 125 000

```
Loop:   lw      $3, 0($5)
        beq     $0, $3, Xuxu
        add     $3, $7, $3
        addi    $5, $5, 4
        beq     $0, $0, Loop
Xuxu:   add     $2, $2,$2
        add     $4,$4, $4
        add     $6, $6, $6
        ...Resto do programa
```

Instrução/Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
lw $3, 0($5)																						
beq $0, $3, Xuxu																						
add $3, $7, $3																						
addi $5, $5, 4																						
beq $0, $0, Loop																						
add $2, $2,$2																						
add $4,$4, $4																						
add $6,$6, $6																						

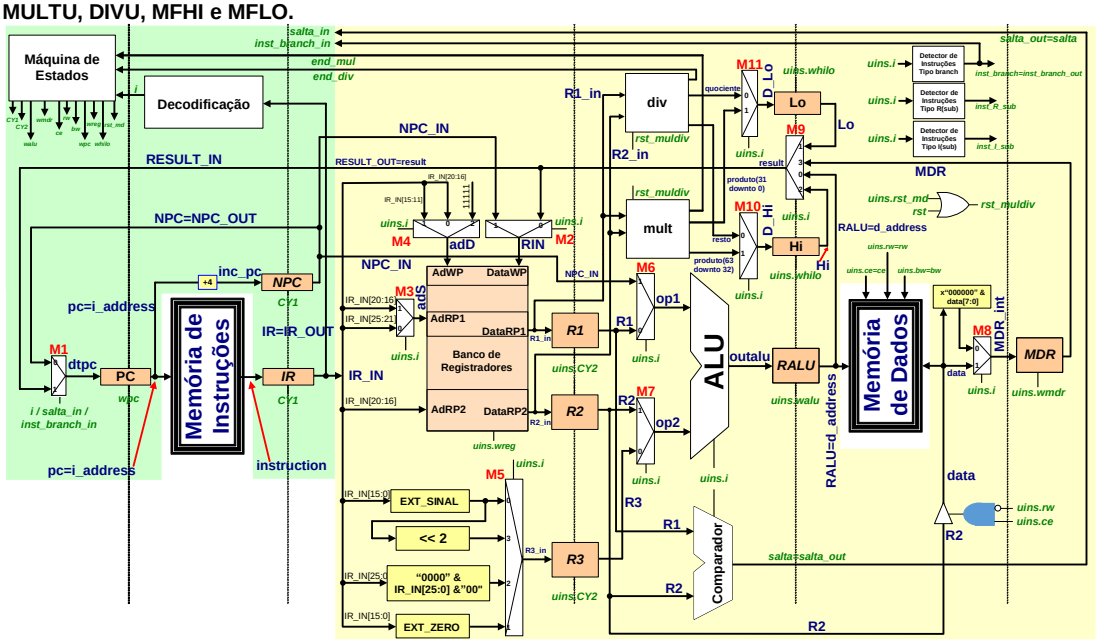
Convenções: **X - bolha** **F - flush do pipeline**
- - estágio não usado **⟶ - adiantamento ou leitura após escrita no mesmo ciclo**
*** - indica uso do estágio para salto (escrita no PC)**
B - Busca -- D - decodificação e busca de operandos -- E - execução de operação -- M - Memória -- W - write-back
Observação: Um salto, condicional ou incondicional é sempre executado no ciclo de write-back W.

Prova G2 **Disciplina: Organização e Arquitetura de Processadores**

Professor: Ney Laert Vilar Calazans

Aluno: **GABARITO** 06/julho/2022

Organização MIPS_S - Instruções com suporte para executar neste hardware - ADDU, SUBU, AND, OR, XOR, NOR, SLL, SLLV, SRA, SRAV, SRL, SRLV, ADDIU, ANDI, ORI, XORI, LUI, LBU, LW, SB, SW, SLT, SLTU, SLTI, SLTIU, BEQ, BGEZ, BLEZ, BNE, J, JAL, JALR, JR, MULTU, DIVU, MFHI e MFLO.



1. (3,0 pontos). Considere a organização MIPS_S cujo diagrama de blocos é ilustrado acima. Trata-se da versão vista em aula e que serviu de base para a disciplina.
   a) (1,0 ponto). Observe o trecho VHDL abaixo, que representa parte da descrição de hardware deste processador, a título ilustrativo.

```
M4: adD <= "11111"          when uins.i=JAL       else
    IR_IN(15 downto 11)           when (inst_R_sub='1' or uins.i=SLTU or uins.i=SLT
          or uins.i=JALR or uins.i=MFHI or uins.i=MFLO or uins.i=SSLL
          or uins.i=SLLV or uins.i=SSRA or uins.i=SRAV or uins.i=SSRL
          or uins.i=SRLV)                                  else
    IR_IN(20 downto 16);
```

   Suponha que existe uma falha no hardware produzido pelo VHDL acima. A falha é que o hardware que gera o sinal `adD` eternamente "gruda" este sinal no sinal `IR[15:11]`. A pergunta aqui é qual(quais) das 37 instruções será(ão) afetada(s) por esta falha. Justifique sua resposta.

   b) (2,0 pontos). Deseja-se dotar a organização multiciclo dada com a capacidade de executar a instrução BLTZ. Supondo que o hardware é alterado para dar suporte a esta instrução, diga que registradores da organização relevantes à execução da BLTZ são escritos e com que dados em cada ciclo da execução desta instrução. Mencione cada registrador relevante, tanto do Bloco de Controle como do Bloco de Dados. Para executar BLTZ, parte do hardware do processador deve ser modificado. Cite ***uma*** modificação necessária a ser realizada no Bloco de Dados e ***uma*** a ser realizada no Bloco de Controle para dar suporte à execução da instrução BLTZ.

Solução:

a) (1 ponto) Observando o comando VHDL nota-se que as 11 instruções explicitamente mencionadas entre a segunda e a quinta linha ainda continuarão a funcionar (SSLL, SLLV, SSRA, SRAV, SSRL, SRLV, SLT, SLTU, JALR, MFHI e R) pois elas usam o caminho fixado no multiplexador em falha. Além destas, funcionarão também as 6 instruções do grupo inst_R_sub (também mencionado na condição da falha), quais sejam: ADDU, SUBU, AAND, OOR e XXOR. Das restantes 20 instruções, funcionarão aquelas que não escrevem no banco de registradores (pois para estas o status deste, que seleciona em que registrador do banco escrever, é irrelevante), quais sejam: SB, SW, BEQ, BGEZ, BLEZ, BNE, J, MULTU e DIVU. Ou seja, apenas 11 (das 37) instruções não funcionarão mais, aquelas listadas acima não citadas aqui.

b) (1,2 pontos) BLTZ emprega o formato I e usa endereçamento relativo. O formato fonte da instrução BLTZ é bltz rs, rótulo e o formato do código objeto é (1 rs 0x0 offset), onde os 4 campos possuem tamanhos respectivos de 6, 5, 5 e 16 bits. A instrução é um salto condicional. Assim, efetivamente BLTZ escreve no registrador PC. Se a condição rs<0 for verdadeira o valor escrito provém de local diferente do caso contrário. A operação consiste

em: (ciclo 1) Busca (IR ← IMem[PC]; NPC←PC+4); (2) Busca de operandos (Rs←BReg[IR(25 downto 21)]; IMED←(ext_sinal(IR(15 downto 0)) <<2); (ciclo 3) Cálculo do endereço de salto e comparação ((RALU ← NPC+IMED); salta←1 se Rs<0, senão 0); (ciclo 4) salto ou incremento do PC (Se salta=1, PC←RALU, senão PC←NPC).

(0,8 pontos). As alterações a realizar no hardware são as seguintes (apenas uma em cada bloco precisa ser fornecida pelo aluno na resposta à questão):

Bloco de Controle:

i. Acrescentar o valor BLTZ no tipo inst_type:

```
type inst_type is (ADDU, SUBU, AAND, OOR, XXOR, NNOR, LW, SW,
      ORI, BLTZ, invalid_instruction);
```

ii. Decodificar a instrução, usando a condição:

```
i <= BLTZ   when instruction(31 downto 26)="000001" and instruction(20 downto
16)="00000" else # ...
```

iii. Mudar a descrição da funcionalidade do estado Salu da máquina de estados de controle:

```
when Salu =>if (i=LBU or i=LW) then
                  NS <= Sld;
            elsif (i=SB or i=SW) then
                  NS <= Sst;
            elsif (i=J or i=JAL or i=JALR or i=JR or i=BEQ
                   or i=BGEZ or i=BLEZ or i=BNE or i=BLTZ) then
                  NS <= Ssalta;
            elsif ((i=MULTU and end_mul='0') or
      (i=DIVU and end_div='0')) then
                  NS <= Salu;
            elsif ((i=MULTU and end_mul='1') or
      (i=DIVU and end_div='1')) then
                  NS <= Sfetch;
            else  NS <= Swbk;
            end if;
```

Bloco de Dados:

i. Acrescentar no sinal auxiliar inst_branch o nome da bltz:

```
-- auxiliary signals
inst_branch  <= '1' when uins.i=BEQ or uins.i=BGEZ or uins.i=BLEZ or uins.i=BNE
or uins.i=BLTZ else
                '0';
```

ii. Incrementar o comparador:

```
-- evaluation of conditions to take the branch instructions
salta <=  '1' when (  (RS=RT and uins.i=BEQ)  or
                      (RS>=0 and uins.i=BGEZ  or
                      (RS<=0 and uins.i=BLEZ) or
                      (RS<0 and uins.i=BLTZ)  or
                      (RS/=RT and uins.i=BNE)
                      )  else
          '0';
```

2. (2,0 pontos). Considere o trecho de programa abaixo, um misto de linguagem de montagem e código objeto do MIPS. O trecho ocupa endereços de memória que iniciam em **0x00400110**. Ele foi gerado a partir de uma descarga (do inglês, *dump*) da memória de instruções produzida pelo montador MARS. Observe as linhas abaixo e faça o que se pede nos itens a seguir.

c) Gere os códigos intermediário e fonte correspondentes às instruções das linhas [1], [5], [8] e [9] do trecho de programa.

d) A partir das informações existentes na linha [4], gere o código intermediário que deveria estar no lugar de `???`. Além disto, diga em que endereço de memória inicia a subrotina `insere_vertice`.

Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando cada passo.

Importante: Valores imediatos devem ser representados em decimal no código fonte e em hexadecimal no código intermediário. Endereços de memória devem ser representados como rótulos no código fonte e como valores hexadecimais no código intermediário.

```
[1]   0x00400110  0x27bdfff4  ???                      150   ???
[2]   0x00400114  0xafbf0000  sw $31,0x00000000($29)  151   sw    $ra,0($sp)
[3]   ...
[4]   0x00400120  0x0c100037  ???                      154   jal   insere_vertice
[5]   0x00400124  0x8fbf0000  ???                      155   ???
[6]   ...
```

```
[7]   0x00400130   0x8d0e0008   lw $14,0x00000008($8)   160   ins_dir:lw    $t6,8($t0)
[8]   0x00400134   0x15c00002   ???                     161    ???
[9]   0x00400138   0xad090008   ???                     162    ???
[10]  0x0040013c   0x03e00008   jr $31                  163    jr    $ra
[11]  0x00400140   0x8d080008   lw $8,0x00000008($8)    164   rec_dir:lw    $t0,8($t0)
```

Solução:

a)

```
[1]    0x00400110   0x27bdfff4   ???                    150    ???
```

Tomando os seis primeiros bits do código objeto 0x27bdfff4, obtém-se $(001001)_2$ = 9 na base 10. Com base na Figura A.10.2 do Apêndice A do livro texto da disciplina, descobre-se que se trata da instrução addiu, cujo formato fonte é addiu Rt, Rs, imm e cujo formato do código objeto é 9 Rs Rt imm, em campos de tamanhos respectivos 6, 5, 5 e 16 bits. Convertendo os valores em hexadecimal dos campos dos registradores fonte e destino, tem-se que Rt = Rs = 11101 em binário, ou 29 em decimal. O campo imm de 16 bits corresponde, em hexa a 0xFFF4 ou (1111 1111 1111 0100) em binário. Ora, como sabe-se que se trata de um número representado em complemento de 2 (do Apêndice A), usando técnicas de conversão de base, conclui-se que se trata do decimal -12. Feito isto, fica simples extrair os códigos intermediário e fonte Estes são: addiu $29,$29, 0xfff4 e addiu $sp, $sp, -12.

```
[5]    0x00400124   0x8fbf0000   ???                    155    ???
```

Como seis primeiros bits do código objeto 0x8fbf0000, têm-se (100011)2 = 35 na base 10. Com base na Figura A.10.2 do Apêndice A do livro texto da disciplina, descobre-se que se trata da instrução lw, cujo formato fonte é lw Rt, offset(Rs) e cujo formato do código objeto é 0x23 Rs Rt offset, em campos de tamanhos respectivos 6, 5, 5 e 16 bits. Destas informações se extrai facilmente os códigos intermediário e fonte. Estes são: lw $31, 0x0000 ($29) e lw $ra, 0($sp).

```
[8]    0x00400134   0x15c00002   ???                    161    ???
```

Como seis primeiros bits do código objeto 0x15c00002, têm-se (000101)2 = 5 na base 10. Com base na Figura A.10.2 do Apêndice A do livro texto da disciplina, descobre-se que se trata da instrução bne, cujo formato fonte é bne Rs,Rt, rótulo, e cujo formato do código objeto é 5 Rs Rt offset, em campos de tamanhos respectivos 6, 5, 5 e 16 bits. Destas informações se pode extrair os códigos intermediário e fonte. Como o offset é o valor 2, o endereço para onde se salta (quando o salto é tomado) é para duas linhas abaixo da linha que segue a instrução bne, onde se encontra o rótulo rec_dir. Os códigos intermediário e fonte são, então: bne $14, $0, 0x0002 e bne $t6, $zero, rec_dir.

```
[9]    0x00400138   0xad090008   ???                    162    ???
```

Como seis primeiros bits do código objeto 0xad090008, têm-se (101011)2 = 43 na base 10. Com base na Figura A.10.2 do Apêndice A do livro texto da disciplina, descobre-se que se trata da instrução sw, cujo formato fonte é sw Rt, offset(Rs), e cujo formato do código objeto é 0x2B Rs Rt offset, em campos de tamanhos respectivos 6, 5, 5 e 16 bits, respectivamente. Destas informações se extrai facilmente os códigos intermediário e fonte. Estes são: sw $9, 0x0008($8) e sw $t1, 8($t0).

b)

```
[4]    0x00400120   0x0c100037   ???          154   jal    insere_vertice
```

A instrução jal tem os seguintes formatos fonte e objeto (de acordo com o Apêndice A): jal alvo e 3 alvo, em campos de tamanhos respectivos 6 e 26 bits, respectivamente. Aqui, alvo é o rótulo/endereço para onde saltar, e a instrução usa modo de endereçamento pseudo-absoluto. Neste modo, o operando corresponde a um endereço de memória que pode ser calculado a partir do código objeto da seguinte forma: toma-se os 26 bits menos significativos do código objeto, acrescenta-se dois bits 0 à direita deste, e quatro bits do valor armazenado no PC no momento da execução do jal (o endereço da linha abaixo do jal) à esquerda dos 26 bits. O código assim obtido é: 0000 00000100000000000000110111 00, ou em hexa 0x004000dc. Este último é o endereço onde inicia a rotina insere_vertice. Obviamente, o código intermediário é então jal 0x004000dc.

3. (3,0 pontos). Abaixo aparece um programa em linguagem de montagem do MIPS. Para este programa, responda às questões que seguem:
   c) (1,0 ponto). Diga quantos bytes este programa ocupa em memória (área de instruções, área de dados e total do programa).
   d) (2,0 pontos). Calcule o tempo de execução do programa em nanossegundos, supondo que se trata de uma organização **MIPS_S** executando em uma frequência de 200MHz, e assumindo que a organização foi modificada para dar suporte à execução da instrução **syscall** em exatamente 4 ciclos de relógio.

```
1           .data
2    str:   .asciiz "Alo Mamae"
3    case:  .word   0x0
4           .text
5    main:
6           la      $t0,str
7           lw      $t1,0($t0)
8           andi    $t6,$t1,0xff
9           la      $t3,case
10          lw      $t4,0($t3)
11          addiu   $t5,$zero,4
12   loop:
13          blez    $t6,end
14          addiu   $t5,$t5,-1
15          sltiu   $t2,$t6,65
16          bne     $t2,$zero,nxt_ch
17          sltiu   $t2,$t6,91
18          beq     $t2,$zero,nxt_ch
19          addiu   $t4,$t4,1
20   nxt_ch:
21          beq     $t5,$zero,endw
22          addu    $t6,$t1,$zero
23          srl     $t6,$t6,8
24          addu    $t1,$t6,$zero
25          andi    $t6,$t6,0xff
26          j       loop
27   endw:
28          addiu   $t0,$t0,4
29          lw      $t1,0($t0)
30          andi    $t6,$t1,0xff
31          addiu   $t5,$zero,4
32          j       loop
33   end:
34          sw      $t4,0($t3)
35          li      $v0,10
36          syscall
```

Dicas: (a) Cuidado com a contabilização de pseudo-instruções.

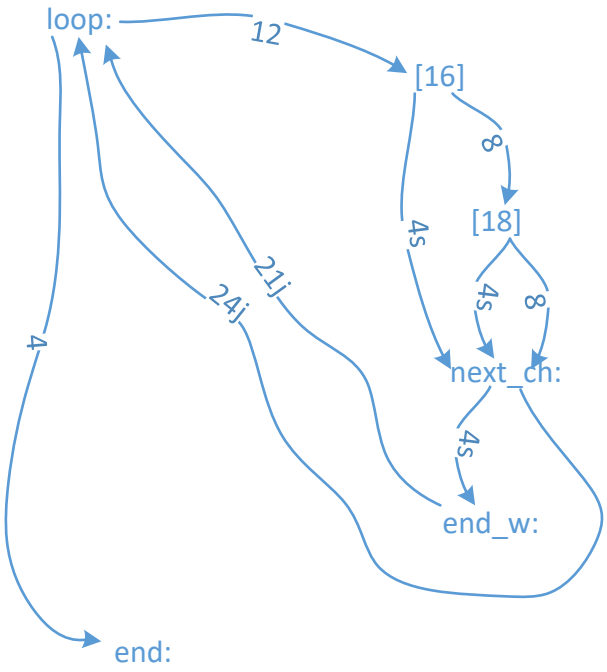
Solução:

```
1     .data                          #      Bytes        Cycles
2   str: .asciiz    "Alo Mamae"      #      10/12        -
3   case: .word   0x0                #      4            -
4     .text                          #
5   main:                            #
6     la      $t0,str                #      8            8
7     lw      $t1,0($t0)             #      4            5
8     andi    $t6,$t1,0xff           #      4            4
9     la      $t3,case               #      8            8
10    lw      $t4,0($t3)             #      4            5
11    addiu   $t5,$zero,4            #      4            4
12  loop:                            #
13    blez    $t6,end                #      4            4
14    addiu   $t5,$t5,-1             #      4            4
15    sltiu   $t2,$t6,65             #      4            4
16    bne     $t2,$zero,nxt_ch       #      4            4
17    sltiu   $t2,$t6,91             #      4            4
18    beq     $t2,$zero,nxt_ch       #      4            4
19    addiu   $t4,$t4,1              #      4            4
20  nxt_ch:                          #
21    beq     $t5,$zero,endw         #      4            4
22    addu    $t6,$t1,$zero          #      4            4
23    srl     $t6,$t6,8              #      4            4
24    addu    $t1,$t6,$zero          #      4            4
25    andi    $t6,$t6,0xff           #      4            4
26    j       loop                   #      4            4
27  endw:                            #
28    addiu   $t0,$t0,4              #      4            4
29    lw      $t1,0($t0)             #      4            5
30    andi    $t6,$t1,0xff           #      4            4
31    addiu   $t5,$zero,4            #      4            4
32    j       loop                   #      4            4
33  end:                             #
34    sw      $t4,0($t3)             #      4            4
35    li      $v0,10                 #      4            4
36    syscall                        #      4            4
```

a) No campo de comentários acima está representada a ocupação em bytes de cada linha. Note que algumas diretivas não ocupam memória (.data e .text), o mesmo ocorrendo nas linhas apenas com rótulos que se referem a uma instrução abaixo deles (linhas 5, 12, 20, 27 e 33). A cadeia str tem de fato 10 bytes, mas questões de alinhamento em palavras de memória deixa os próximos dois bytes não usados. Aqui pode-se considerar 10 bytes como certo, mas a ocupação é de fato de 12 bytes. As demais linhas sempre ocupam 4 bytes, exceto aquelas que possuem a diretiva la que corresponde a duas instruções (lui+ori) e correspondem, cada linha, a 8 bytes de memória. Assim, o total de bytes ocupado é 12+4=16 bytes de dados e 2*8+25*4 = 116 bytes de programa, perfazendo um total de 132 bytes.

b) O primeiro passo é calcular o número de ciclos que leva para executar o programa. Este problema é bem trabalhoso para este programa, devido à quantidade de opções de caminho possíveis (observe-se o número de saltos condicionais do programa, além do número de saltos incondicionais). O que este programa faz é de fato contar o número de letras maiúsculas do vetor str. Para tanto identifica-se uma letra maiúscula como um caracter ASCIIE entre 65 (letra

A) e 90 (letra Z). Uma complicação do programa é que ele usa instruções LW, que carregam 4 caracteres por vez em um registrador do MIPS. Assim, depois de tratar 4 caracteres, se busca mais 4 caracteres e assim por diante. A segunda coluna do campo de comentários acima contém o cálculo de número de ciclos para cada linha. Inicialmente. As linhas 6-11 e 34-36 são executadas sempre uma única vez, perfazendo 34 ciclos (linhas 6-11) + 12 ciclos (linhas 34-36), um total de 46 ciclos. O laço entre as linhas 13-32 é executado uma vez para cada um dos 10 caracteres de str. A figura abaixo apresenta a estrutura de controle do laço principal do programa. Rótulos representam números de ciclos gastos para chegar no rótulo ou linha de destino da seta. Os sufixos s e j correspondem a salto (condição de salto verdadeira em instruções de salto condicional) e salto incondicional (resultado da execução de uma instrução J). Cada letra maiúscula é tratada em 12+8+8=28 ciclos, cada letra minúscula é tratada em 12+8+4=24 ciclos, enquanto brancos e caracteres de pontuação em geral são tratados em 12+4+4=20 ciclos. Após tratar cada caractere, a manutenção do laço se faz em 24 ciclos para caracteres que não são o último de uma palavra, ou em 21 ciclos, quando o caractere é o último da palavra. O caractere NULL (último de str) é tratado em apenas 4 ciclos e com ele sai-se do laço loop. Contabilizando o total de ciclos do laço, tem-se: Duas letras maiúsculas que não são últimos caracteres de uma palavra → 2*(28+24) = 104ciclos; 5 letras minúsculas que não são últimos caracteres de uma palavra → 5*(24+24) = 240ciclos; 1 caractere espaço em branco no final de uma palavra → 20+21=41ciclos; 1 letra minúscula no final de uma palavra → 24+21=45ciclos. Somando-se estes valores parciais com o número de ciclos de tratar o NULL tem-se o número de ciclos total do laço, que é: 104+240+41+45+4=434ciclos. O total ciclos para executar o programa é então: 46+434=480ciclos.Uma frequência de 200MHz corresponde a um período de 5ns. Logo, o tempo de execução solicitado é: 480*5ns=2.400ns.



4. (2 pontos) Dado o trecho de programa abaixo, a executar no MIPS, suponha uma implementação do MIPS sem **nenhuma** capacidade de solução de conflitos de dados. Mostre o diagrama pipeline para a execução das instruções do trecho durante **apenas os primeiros 22 ciclos de relógio**, supondo que $5 aponta para o início de um vetor de 2 posições com o conteúdo abaixo, em palavras: 125 000

```
Loop:   lw      $3, 0($5)
        beq     $0, $3, Xuxu
        add     $3, $7, $3
        addi    $5, $5, 4
        beq     $0, $0, Loop
Xuxu:   add     $2, $2,$2
        add     $4,$4, $4
        add     $6, $6, $6
        ...Resto do programa
```

Instrução/Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
lw $3, 0($5)	B	D	E	M	W								B	D	E	M	W					
beq $0, $3, Xuxu		B	D	X	X	X	E	-	W					B	D	X	X	X	E	-	W	
add $3, $7, $3			B	X	X	X	D	E	-	W					B	X	X	X	D	E	-	F
addi $5, $5, 4							B	D	E	-	W								B	D	E	F
beq $0, $0, Loop								B	D	E	-	W								B	D	F
add $2, $2,$2									B	D	E	-	F									B
add $4,$4, $4										B	D	E	F	F								
add $6,$6, $6											B	D	F	F	F							
Rdp												B	F	F	F	F						

Convenções: **X - bolha** **F - flush do pipeline**

- - estágio não usado **⟶ - adiantamento ou leitura após escrita no mesmo ciclo**

*** - indica uso do estágio para salto (escrita no PC)**

B - Busca -- D - decodificação e busca de operandos -- E - execução de operação -- M - Memória -- W - write-back

Observação: Um salto, condicional ou incondicional é sempre executado no ciclo de write-back W.

Solução: A solução está no diagrama pipeline acima. Como não há nenhuma capacidade de solução de conflitos, tudo se resolve inserindo bolhas, até que o conflito se resolva. Note-se que a primeira vez que a segunda linha é executada, a condição do salto avalia como falsa (125 é diferente de 0) e a segunda vez ela avalia como verdadeira.