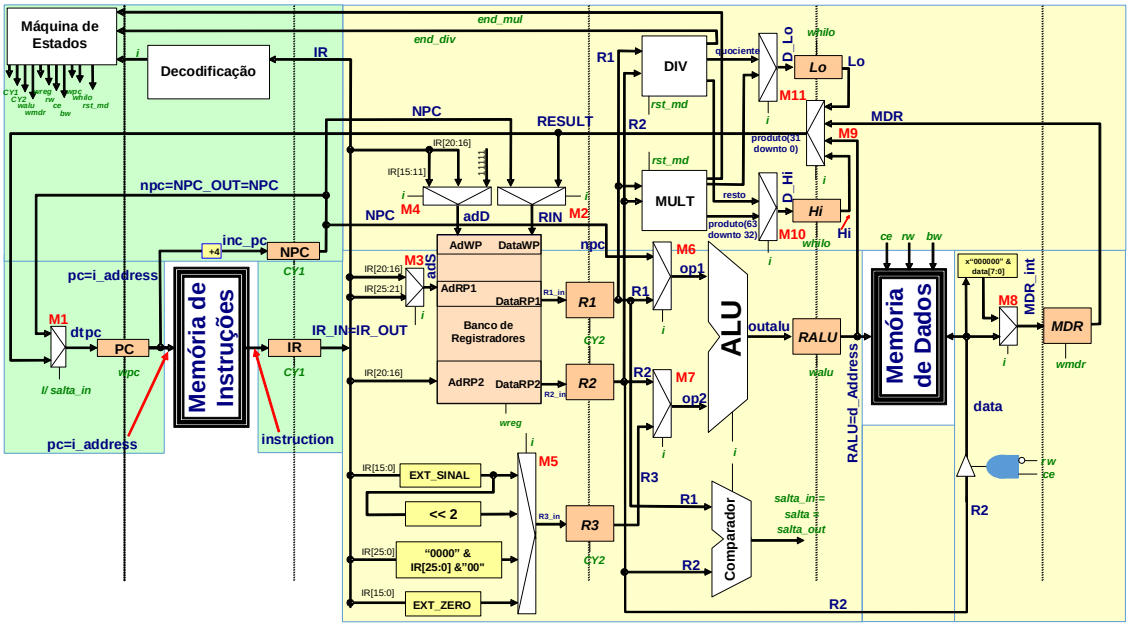
Prova P2 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 09/novembro/2022

1. [2,0 pontos] Considere o processador MIPS multiciclo a seguir (visto em aula) e marque sobre ele todos os caminhos de dados e de controle usados para executar a instrução `J`. Ressalte todos os sinais de controle efetivamente usados para executar a instrução, e indique que multiplexadores são usados para transportar informação relevante para ela. Não esqueça de identificar qual ou quais dos 11 sinais de controle gerados pela máquina de estados de controle do processador são relevantes para a execução da `J`. Use, além do diagrama de blocos fornecido abaixo, a especificação da instrução `J` que consta no Apêndice A do livro texto, a especificação textual da MIPS multiciclo e se achar necessário, a descrição VHDL da MIPS multiciclo.



2. [3 pontos] Assuma uma frequência de relógio de 500 MHz para uma organização MIPS_S (vista em aula) e com base nesta característica, responda às questões abaixo.

   (a) Qual o número de ciclos de relógio consumidos para a execução do programa abaixo na organização dada? Considere a área de dados fornecida, assumindo que a pseudo-instrução **la** leva 8 ciclos de relógio para executar, sendo equivalente a duas instruções, um **lui** seguido de um **ori**, e que a instrução **syscall** e a pseudo-instrução **li** levam, cada uma, 4 ciclos de relógio para executar. As demais instruções executam em um número de ciclos conforme descrito na Tabela 4 do documento de especificação do processador MIPS_S.

   (b) Qual o tempo de execução do programa, em **segundos**?

   (c) O que faz este programa? Diga em uma frase.

```
1.              .text
2.    main:     la          $s0, str          #
3.              la          $s1, newstr       #
4.    loop:     lbu         $t1, 0($s0)       #
5.              blez        $t1, fim          #
6.              sw          $t1, 0($s1)       #
7.              addiu       $s0,$s0, 1        #
8.              addiu       $s1,$s1, 4        #
9.              j           loop              #
10.   fim:      li          $v0,10            #
11.             syscall                       #
12.             .data
13.   str:      .asciiz     "teste de byte xy"
14.   newstr:   .word 0x0
```

3. [2 pontos] Seja dado o trecho de programa abaixo. Suponha que este trecho é executado no pipeline do MIPS_S visto em aula, supondo uma organização **sem nenhuma capacidade** de resolução de conflitos de dados, de controle ou predição de saltos, exceto pela inserção de bolhas. Responda às questões abaixo.
   (a) Qual o número **mínimo ideal de ciclos de clock** para a execução das 10 primeiras instruções deste programa no pipeline do MIPS? (do `lui $1,0x1001` até o `addiu $9,$9,0x0000`). Suponha que a instrução `blez` da quarta linha não salta).
   (b) Determine o número real de ciclos de clock para executar este mesmo trecho, com as mesmas suposições. Detalhe a execução no diagrama pipeline abaixo, indique tudo que ocorre, mostrando os estágios ocupados e as bolhas na posição correta, caso estas existam.

INSTRUÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
lui $1, 0x1001																						
ori $11, $1, 0x0084																						
lw $11, 0x0000 ($11)																						
blez $11, 0x0009																						
lw $12, 0x0000 ($8)																						
lw $13, 0x0000 ($9)																						
addu $12, $12, $13																						
sw $12, 0x0000 ($10)																						
addiu $8, $8, 0x0000																						
addiu $9, $9, 0x0000																						
addiu $10, $10, 0x0004																						
addiu $11, $11, 0xfffff																						

Convenções: **X** - bolha ***** - estágio em que um salto é executado (carga no PC)
- - estágio não usado
Estágios do pipeline: **B** (Busca), **D** (Decodificação), **E** (Execução), **M** (Memória), **W** (Write-back)

4. [3 pontos] Realize as operações pedidas abaixo sobre número racionais:
   (a) Converta o numeral **-1,59375** para o formato precisão simples (SP) do padrão IEEE-754. Assuma que o numeral está expresso na base 10, usando o padrão brasileiro de representação, onde a vírgula separa a parte inteira da parte fracionária. Após converter, diga se a conversão resultou em uma representação exata ou não do numeral original. O resultado deve ser mostrado como um código hexadecimal de 8 dígitos.
   (b) Dado o código hexadecimal **0x427B0000**, e assumindo que se trata de um numeral em precisão simples representado segundo o padrão IEEE-754, diga que número é este na base 10. Mostre a resposta usando o padrão brasileiro de representação, onde a vírgula separa a parte inteira da parte fracionária.

---------- **Aqui terminam as questões da P2** ----------

QUESTÃO EXTRA, **OPCIONAL**, PARA QUEM QUISER REFORÇAR A NOTA DA P1

1. (3 pontos) Montagem/Desmontagem de código objeto. Considere o trecho de programa abaixo, e assuma que ele começa no endereço de memória **0x00400040**.
   (1) Gere o código objeto hexadecimal correspondente à instrução das linhas [3] e [4] do trecho de programa;
   (2) Desmonte o código da linha [9] para obter a representação em linguagem de montagem da instrução que deveria estar nesta linha.
   Dica: Muita atenção ao tratamento de endereços, e cuidado com a conversão de bases
   Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

```
[1]      loop:  blez      $t1,end # Instrução inicia no endereço 0x00400040
[2]             sltiu     $t2,$t1,65
[3]             bne       $t2,$zero,nxt_ch  # Traduza para código objeto
[4]             sltiu     $t2,$t1,91        # Traduza para código objeto
[5]             beq       $t2,$zero,nxt_ch
[6]             addiu     $t4,$t4,1
[7]             addiu     $t0,$t0,4
[8]      nxt_ch:lw        $t1,0($t0)
[9]             0x08100010                  # Traduza para código fonte
```

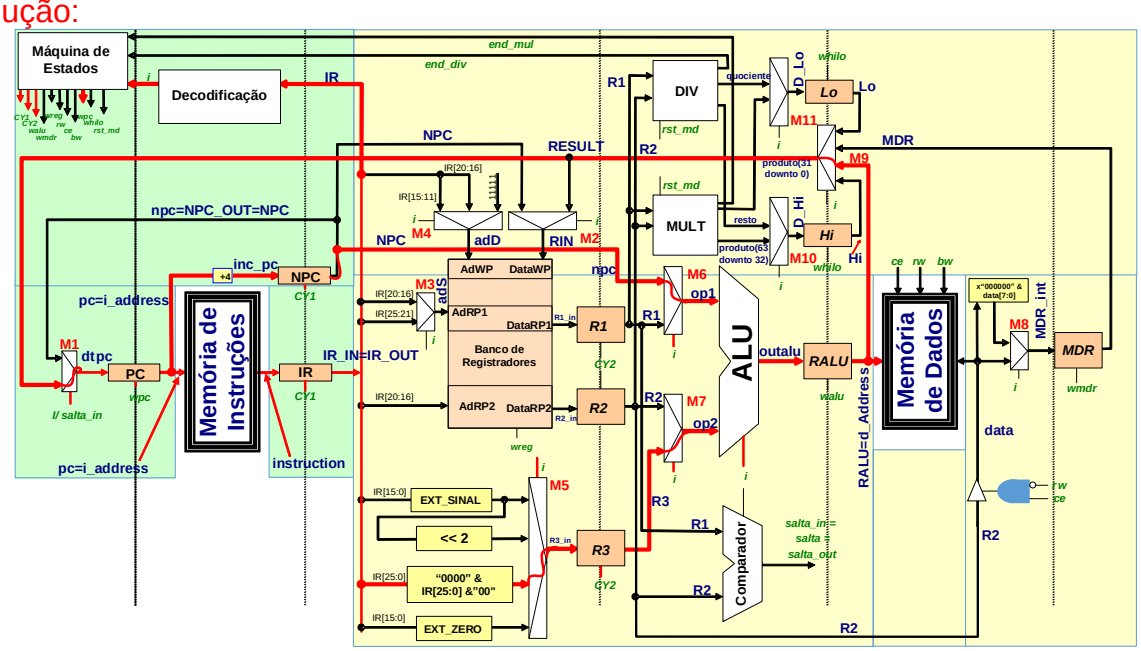
Prova P2 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: **Gabarito** 09/novembro/2022

1. [2,0 pontos] Considere o processador MIPS multiciclo a seguir (visto em aula) e marque sobre ele todos os caminhos de dados e de controle usados para executar a instrução `J`. Ressalte todos os sinais de controle efetivamente usados para executar a instrução, e indique que multiplexadores são usados para transportar informação relevante para ela. Não esqueça de identificar qual ou quais dos 11 sinais de controle gerados pela máquina de estados de controle do processador são relevantes para a execução da `J`. Use, além do diagrama de blocos fornecido abaixo, a especificação da instrução `J` que consta no Apêndice A do livro texto, a especificação textual da MIPS multiciclo e se achar necessário, a descrição VHDL da MIPS multiciclo.

Solução:



2. [3 pontos] Assuma uma frequência de relógio de 500 MHz para uma organização MIPS_S (vista em aula) e com base nesta característica, responda às questões abaixo.

(a) Qual o número de ciclos de relógio consumidos para a execução do programa abaixo na organização dada? Considere a área de dados fornecida, assumindo que a pseudo-instrução **la** leva 8 ciclos de relógio para executar, sendo equivalente a duas instruções, um **lui** seguido de um **ori**, e que a instrução **syscall** e a pseudo-instrução **li** levam, cada uma, 4 ciclos de relógio para executar. As demais instruções executam em um número de ciclos conforme descrito na Tabela 4 do documento de especificação do processador MIPS_S.

(b) Qual o tempo de execução do programa, em **segundos**?

(c) O que faz este programa? Diga em uma frase.

Solução: (a)

```
1.          .text                                   Ciclos
2.    main: la          $s0, str            #      8
3.          la          $s1, newstr         #      8
4.    loop: lbu         $t1, 0($s0)         #      5
5.          blez        $t1, fim            #      4
6.          sw          $t1, 0($s1)         #      4
7.          addiu       $s0,$s0, 1          #      4
8.          addiu       $s1,$s1, 4          #      4
9.          j           loop                #      4
10.   fim:  li          $v0,10              #      4
11.         syscall                         #      4
12.         .data
13.   str:  .asciiz     "teste de byte xy"
14.   newstr:.word      0x0
```

A área de programa ocupa efetivamente as 9 linhas numeradas de 2 a 11. As linhas 2, 3, 10 e 11 executam apenas 1 vez cada uma, perfazendo um total de 8+8+4+4= 24 ciclos para serem executadas.

As linhas 4-10 são um laço cujo número de execuções depende da área de dados do programa. A área de dados ocupa as linhas 13 e 14.

A linha 13 contém uma cadeia definida pela diretiva `.asciiz`. Como implícito na funcionalidade desta diretiva, a cadeia `str` ocupa em memória uma quantidade de bytes dada pelo número de caracteres dela acrescido de 1, devido à inserção automática do caractere ASCIIE NULL (0x00) para indicar o fim da cadeia. Ora, a cadeia `"teste de byte xy"` possui 16 caracteres imprimíveis (13 letras e 3 brancos), que somado ao caractere NULL acrescido pela diretiva, totaliza 17 caracteres. A linha 14 possui apenas uma variável que ocupa 4 bytes em memória (designada pelo nome `newstr`). A partir destas informações se pode calcular o número de vezes que o laço executa e com isto determinar o número de ciclos gasto para executá-lo.

As linhas 2 e 3 do programa apenas carregam, respectivamente em $s0 e $s1 os endereços da cadeia `str` e da variável `newstr`. A partir daí, entra-se no laço, que inicia lê o caractere apontado por $s0 (na linha 4) para o registrador $t1 e testa a condição de saída do laço na linha 5 (esta condição será verdadeira apenas quando o caractere lido for o NULL ou seja 0x00, exatamente o último da cadeia). Logo, descobre-se assim que a última vez que o laço for executado ele gastará apenas 5+4= 9 ciclos para executar (o tempo de executar `lbu` seguido do `blez`). Se na linha 5 o salto não for executado, executam-se todas as quatro linhas restantes do laço (da 6 a 9). Nestas linhas se escreve todo o conteúdo do registrador $t1 na memória apontada por $s1 ($t1 tem 4 bytes, onde o byte menos significativo contém o caractere da cadeia que acabou de ser lido e os outros três bytes estão zerados pela execução da instrução lbu). Em seguida, avança-se os ponteiros da cadeia `str` e da cadeia `newstr`. Note-se que a ponteiro da primeira avança de 1 em 1 e o da segunda avança de 4 em 4. O número de ciclos gasto para uma execução não-final do laço então é o tempo de executar todas as instruções do laço, ou seja: 5+4+4+4+4+4= 25 ciclos.

Do exposto acima, como a cadeia possui 17 caracteres, o tempo gasto para executar o laço com os dados é 16*25 + 9 = 409 ciclos. Somados aos 24 ciclos da parte fora do laço, o número total de ciclos para executar o programa é, então 24+409= **433 ciclos**.

(b) O tempo de execução é calculado depois de se descobrir o tempo gasto por 1 ciclo de relógio. Com a frequência dada, 500MHz e a fórmula f=1/T, onde T é o período do clock (exatamente o que queremos, o tempo gasto para executar um ciclo do clock), fica simples computar a informação. MHz significa milhões de Herz ou 10^6Hertz e 1 Hz= 1 ciclo/s, donde T é medido em s/ciclo. Logo, T=1/f = 1/(500*10^6) s/ciclo = 1/(5*10^8) s/ciclo = 10^{-8}/5 s/ciclo = 2*10^{-9} s/ciclo. Logo, cada ciclo de relógio dura exatamente 2 bilionésimos de segundo. Multiplicando este valor pelo número de ciclos que o programa leva para executar (resposta do item (a)), tem-se que o programa executa em 433*2*10-9 segundos, ou **866*10^{-9} s** ou, em notação científica **8,66*10^{-7} s**, ou ainda, em notação de engenharia **0,866 *10^{-6} s**.

(c) Este programa realiza a operação de descompactar uma cadeia de caracteres, distribuindo cada caractere da cadeia original `str` em uma palavra de memória de uma nova cadeia `newstr`. Esta operação é útil para acelerar operação de sistemas que possuem memórias lentas para acesso a byte comparado com o tempo de acesso a palavras de 32 bits.

3. [2 pontos] Seja dado o trecho de programa abaixo. Suponha que este trecho é executado no pipeline do MIPS_S visto em aula, supondo uma organização **sem nenhuma capacidade** de resolução de conflitos de dados, de controle ou predição de saltos, exceto pela inserção de bolhas. Responda às questões abaixo.
    (a) Qual o número **mínimo ideal de ciclos de clock** para a execução das 10 primeiras instruções deste programa no pipeline do MIPS? (do `lui $1,0x1001` até o `addiu $9,$9,0x0000`). Suponha que a instrução `blez` da quarta linha não salta).

Solução: Para executar as 10 primeiras instruções o número mínimo ideal (NMI) seria gastar 5 ciclos para terminar a primeira instrução e 1 ciclo para terminar cada instrução seguinte, ou seja: NMI=5+9= **14 ciclos**.

(b) Determine o número real de ciclos de clock para executar este mesmo trecho, com as mesmas suposições. Detalhe a execução no diagrama pipeline abaixo, indique tudo que ocorre, mostrando os estágios ocupados e as bolhas na posição correta, caso estas existam.

INSTRUÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
lui $1, 0x1001	B	D	E	-	W																									
ori $11, $1, 0x0084		B	D	X	X	X	E	-	W																					
lw $11, 0x0000 ($11)			B	X	X	X	D	X	X	X	E	M	W																	
blez $11, 0x0009							B	X	X	X	D	X	X	X	E	-	-													
lw $12, 0x0000 ($8)											B	X	X	X	D	E	M	W												
lw $13, 0x0000 ($9)															B	D	E	M	W											
addu $12, $12, $13																B	D	X	X	X	E	-	W							
sw $12, 0x0000 ($10)																	B	X	X	X	D	X	X	X	E	M	W			
addiu $8, $8, 0x0000																					B	X	X	X	D	E	-	W		
addiu $9, $9, 0x0000																									B	D	E	-	W	
addiu $10, $10, 0x0004																														
addiu $11, $11, 0xfffff																														

Convenções: **X** - bolha ***** - estágio em que um salto é executado (carga no PC)
- - estágio não usado

Estágios do pipeline: **B** (Busca), **D** (Decodificação), **E** (Execução), **M** (Memória), **W** (Write-back)

Algumas observações que justificam algumas das bolhas:

a) A segunda instrução (ori ...) deve esperar a escrita no ciclo 5 antes de poder buscar seu operando $1 corretamente, no ciclo 6, ficando 4 ciclos no estágio D (gerando 3 bolhas);
b) A terceira instrução (lw $11 ...) fica presa do ciclo 3 ao 6 no estágio B, pois só depois que a segunda avança para o estágio E no ciclo 7 ela pode entrar no estágio D;
c) A quarta instrução (blez ...) tem que esperar que o registrador $11 seja escrito 13 para no ciclo 14 busca seu valor para realizar a comparação;
d) A oitava instrução (sw ...) precisa esperar que $12 seja escrito no ciclo 23 antes de poder buscar seu valor no ciclo 24; por isto ela fica 4 ciclos presa no estágio D. Lembrar que o valor do registrador que será escrito na memória (que deve no estágio D ser escrito com o valor do registrador $12 escrito pela sétima instrução) será usado no estágio M, depois de se calcular o endereço de escrita no estágio E.

4. [3 pontos] Realize as operações pedidas abaixo sobre número racionais:
   (a) Converta o numeral **-1,59375** para o formato precisão simples (SP) do padrão IEEE-754. Assuma que o numeral está expresso na base 10, usando o padrão brasileiro de representação, onde a vírgula separa a parte inteira da parte fracionária. Após converter, diga se a conversão resultou em uma representação exata ou não do numeral original. O resultado deve ser mostrado como um código hexadecimal de 8 dígitos.

Solução: O primeiro passo é converter $(-1,59375)_{10}$ para a base 2. Usamos a técnicas vista no curso de introdução a CC/EC no primeiro semestre. A parte inteira vale 1, que é o mesmo valor em decimal e binário, logo $(-1,59375)_{10}$ = (-1,**f**) e falta apenas definir a parte fracionária **f** em binário, a partir de parte fracionário decimal. Operamos por multiplicações sucessivas pela base 2, guardando a cada passo o valor à esquerda da vírgula e continuando com a parte fracionária resultante da multiplicação:

0,59375 *2 = **1**,1875
0,1875 *2 = **0**,375
0,375 *2 = **0**,75
0,75 *2 = **1**,5
0,5 *2 = **1**,0

Os cinco dígitos assim produzidos, unidos ao (-1,) são o número racional convertido para a base 2, qual seja $(1,10011)_2$. Note-se que o número em questão está normalizado (ou seja, $1,10011 = 1,10011 * 2^0$. Agora basta aplicar a forma geral de números SP do padrão IEEE-754, ou seja, $(-1)^s * 1.f * 2^{(e-127)}$. A representação em 32 bits fica então:

s=1; **e**=127, o número que ao subtrair 127 dele gera o expoente do número, 0. Logo, o campo **e** deve valer 127 na base 10 ou, em 8 bits 01111111, → **e**=01111111; fração é conjunto de 23 bits que se obtém ao remover 1, do número (que está implícito na forma normalizada) e preencher as posições que faltam com 0s à direita para perfazer 23 bits, ou seja,

f=10011000000000000000000. Juntando os três campos se tem o valor de 32 bits que representa o número decimal original no padrão IEEE-754 SP, que é:
1 01111111 10011000000000000000000.
Reagrupando os bits em grupos de quatro bits e convertendo cada grupo para hexadecimal, obtém-se a resposta final: **0xBFCC0000**.

(b) Dado o código hexadecimal **0x427B0000**, e assumindo que se trata de um numeral em precisão simples representado segundo o padrão IEEE-754, diga que número é este na base 10. Mostre a resposta usando o padrão brasileiro de representação, onde a vírgula separa a parte inteira da parte fracionária.

Solução: O primeiro passo é recuperar os campos **s**, **e**, **f**, a partir do código hexadecimal. Ficamos assim:

s = 0
e = 10000100
f = 11110110000000000000000

Logo, trata-se de um número positivo (s=0) com e=132 e significando igual a 1,1111011. O expoente real é 132-127 = 5. Logo, em binário o número racional em questão é $1{,}1111011 * 2^5$, ou $(111110{,}11)_2$. Convertendo este número para a base 10 tem-se:

- Parte inteira: 111110 ou $1*2^5+1*2^4+1*2^3+1*2^2+1*2^1+0*2^0 = 32+16+8+4+2+0= 62$.
- Parte fracionária: 0,11 ou 0,5+0,25 = 0,75
- **Resposta final: 62,75**

---------- **Aqui terminam as questões da P2** ----------

QUESTÃO EXTRA, **OPCIONAL**, PARA QUEM QUISER REFORÇAR A NOTA DA P1

1. (3 pontos) Montagem/Desmontagem de código objeto. Considere o trecho de programa abaixo, e assuma que ele começa no endereço de memória **0x00400040**.
   (1) Gere o código objeto hexadecimal correspondente à instrução das linhas [3] e [4] do trecho de programa;
   (2) Desmonte o código da linha [9] para obter a representação em linguagem de montagem da instrução que deveria estar nesta linha.
   Dica: Muita atenção ao tratamento de endereços, e cuidado com a conversão de bases
   Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

```
[1]       loop:  blez       $t1,end # Instrução inicia no endereço 0x00400040
[2]              sltiu      $t2,$t1,65
[3]              bne        $t2,$zero,nxt_ch  # Traduza para código objeto
[4]              sltiu      $t2,$t1,91        # Traduza para código objeto
[5]              beq        $t2,$zero,nxt_ch
[6]              addiu      $t4,$t4,1
[7]              addiu      $t0,$t0,4
[8]       nxt_ch:lw         $t1,0($t0)
[9]              0x08100010                   # Traduza para código fonte
```

Solução:

Linha [3] - Dado o formato da instrução bne rs, rt, rótulo (5 rs rt Offset), tem-se o seguinte código binário 000101 (constante 5 em 6 bits), concatenado com 01010 ($t2=$10 em 5 bits), concatenado com 00000 ($zero=$0 em 5 bits), concatenado com 0000000000000100 (o deslocamento, em palavras, da instrução abaixo da bne até a instrução que contém o rótulo nxt_ch, ou seja +4). O resultado é o vetor 0001 0101 0100 0000 0000 0000 0000 0100 em binário, ou **0x15400004** em hexadecimal. Este é o código de 32 bits da instrução **bne** em questão.

Linha [4] - Dado o formato da instrução sltiu rt, rs, immed (0xB rs rt imediato), tem-se o seguinte código binário 001011 (constante B em 6 bits), concatenado com 01001 ($t1=$9 em 5 bits, o Rs), concatenado com 01010 ($t2=$10 em 5 bits, o Rt), concatenado com 0000000001011011 (91 decimal convertido para binário em 16 bits). O resultado é o vetor 0010 1101 0010 1010 0000 0000 0101 1011 em binário, ou **0x2D2A005B** em hexadecimal. Este é o código de 32 bits da instrução **sltiu** em questão.

Linha [9] - Dado o código hexadecimal **0x08100010**, sabe-se que os 6 primeiros bits definem o código da instrução, ou pelo menos a classe desta. Neste caso, o valor dos 6 primeiros bits em binário é 000010, ou 0x2 em hexadecimal. Assim, consultando-se a documentação do ISA do MIPS, descobre-se que se trata da instrução **j**. Esta instrução possui o formato (**2 target**), onde os campos possuem, da esquerda para a direita os tamanhos 6 e 26, respectivamente. O cálculo do endereço de salto implica tomar os 26 bits mais à direita do código objeto, concatenando 2 bits em 0 à direita deste e colocando

os quatro bits mais significativos do valor do PC no momento da execução como os quatro bits mais significativos do endereço a gerar. Como no momento da execução o PC aponta para a próxima instrução a executar o valor deste é 0x00400064. Assim o endereço de salto da instrução **j** será a concatenação de 0000 (4 bits do PC no momento da execução), 00 0001 0000 0000 0000 0001 0000 (valor do campo **target**) e 00 (os dois bits constantes do endereço de qualquer instrução). Reagrupando de quatro em quatro bits e expressando o endereço em hexadecimal, tem-se 0x0400040, que é o endereço do rótulo loop. Logo, o código fonte da instrução da linha [18] é **j loop**.