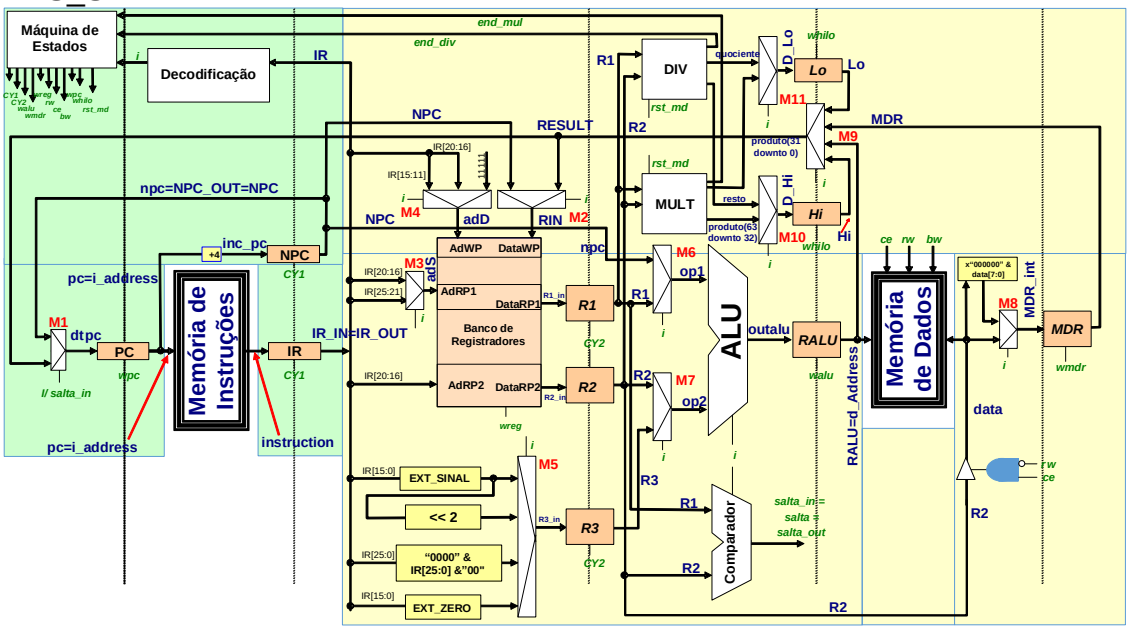
Prova PS **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 29/junho/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 **BLTZ**. Ressalte também todos os sinais de controle efetivamente usados para executar a instrução. 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 **BLTZ**. Use, além do diagrama de blocos fornecido abaixo, a especificação da instrução **BLTZ** que consta no Apêndice A do livro texto, a descrição VHDL da MIPS_S, bem como a especificação textual da MIPS_S.



2. [4 pontos] Assuma uma frequência de relógio de 400 MHz para uma organização MIPS_S (vista em aula) e com base neste fato, responda às questões abaixo sobre esta organização.

   (a) Qual o número de ciclos de relógio consumidos para a execução do programa abaixo nesta organização (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 leva 4 ciclos de relógio para executar).

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

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

   (d) Diga se o programa contém alguma subrotina. Em caso afirmativo, identifique-a no código.

```
1.              .text
2.              .globl      main
3.      main:   la          $t0,N
4.              lw          $t0,0($t0)
5.              la          $t1,S
6.              beq         $t0,$zero,end
7.              addiu       $t2,$zero,0
8.              addiu       $t3,$zero,1
9.              sw          $t2,0($t1)
10.             addiu       $t0,$t0,-1
11.             beq         $t0,$zero,end
12.             addiu       $t1,$t1,4
13.             sw          $t3,0($t1)
14.             addiu       $t0,$t0,-1
15.             beq         $t0,$zero,end
16.     loop:   addiu       $t1,$t1,4
17.             addu        $t4,$t2,$t3
18.             sw          $t4,0($t1)
19.             addu        $t2,$zero,$t3
```

```
20.           addu        $t3,$zero,$t4
21.           addiu       $t0,$t0,-1
22.           bne         $t0,$zero,loop
23.     end:  li          $v0,10
24.           syscall
25.
26.           .data
27.     N:    .word       10
28.     S:    .space      40
```

3. [2,5 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
```

4. [1,5 pontos] Converta o numeral -28,25 para o formato de 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. Em outras palavras, há um erro de representação resultante da conversão ou não?

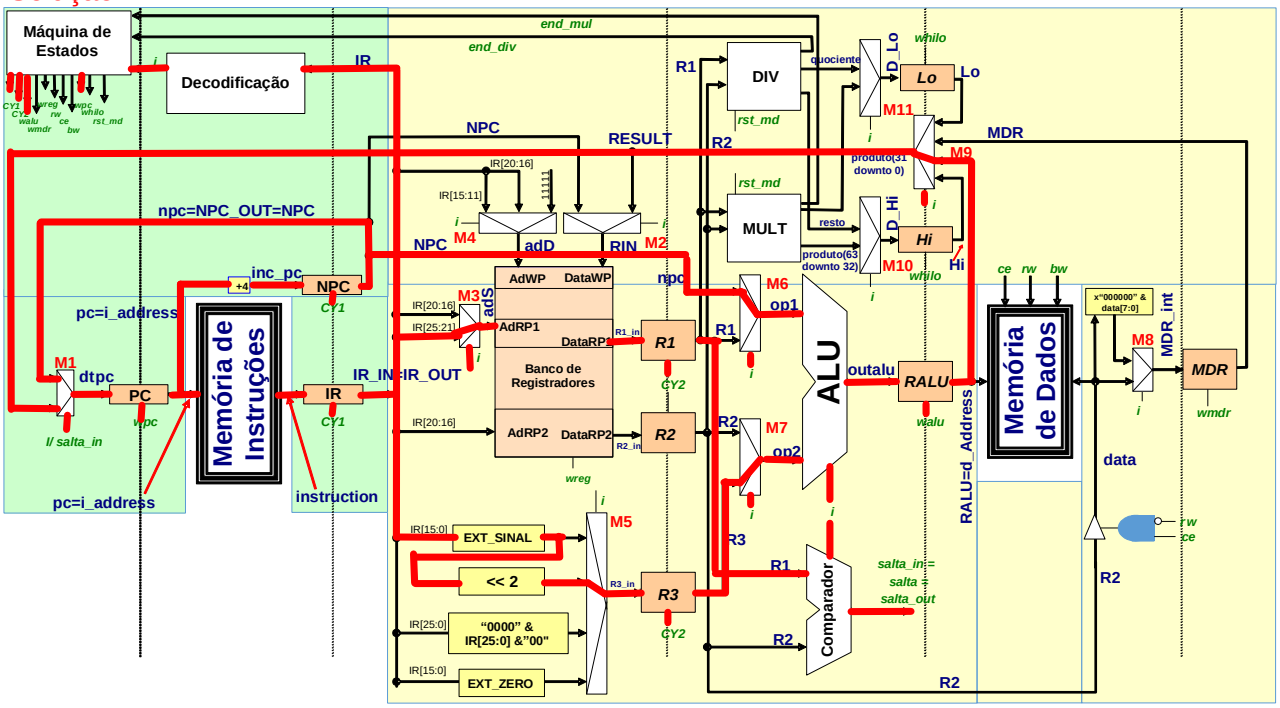
Prova PS **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: **Gabarito** 29/junho/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 BLTZ. Ressalte também todos os sinais de controle efetivamente usados para executar a instrução. 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 BLTZ. Use, além do diagrama de blocos fornecido abaixo, a especificação da instrução BLTZ que consta no Apêndice A do livro texto, a descrição VHDL da MIPS_S, bem como a especificação textual da MIPS_S.

Solução:



2. [4 pontos] Assuma uma frequência de relógio de 400 MHz para uma organização MIPS_S (vista em aula) e com base neste fato, responda às questões abaixo sobre esta organização.
(a) Qual o número de ciclos de relógio consumidos para a execução do programa abaixo nesta organização (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** leva 4 ciclos de relógio para executar).
(b) Qual o tempo de execução do programa, em **segundos**.
(c) O que faz este programa? Diga em uma frase.
(d) Diga se o programa contém alguma subrotina. Em caso afirmativo, identifique-a no código.

Solução:

```
1.          .text
2.          .globl      main
3.    main: la          $t0,N              8#  Programa para gerar os N primeiros
                                             #     elementos da Série de Fibonacci
4.          lw          $t0,0($t0)         5# Busca o número de elementos a gerar
5.          la          $t1,S              8# Gera ponteiro para onde armazenar série
6.          beq         $t0,$zero,end      4# Se é para gerar 0 elementos, nada a fazer
7.          addiu       $t2,$zero,0        4# Senão gera os dois primeiros elementos,
8.          addiu       $t3,$zero,1        4# da série de Fibonacci, 0 e 1
9.          sw          $t2,0($t1)         4# armazena o primeiro
10.         addiu       $t0,$t0,-1         4# Testa se era para gerar apenas 1
11.         beq         $t0,$zero,end      4# Se sim, termina aqui
```

```
12.       addiu       $t1,$t1,4          4# Senão,avança ponteiro da série em geração
13.       sw          $t3,0($t1)         4# armazena o segundo elemento
14.       addiu       $t0,$t0,-1         4# decrementa contador de elementos
15.       beq         $t0,$zero,end      4# Se zerou contador, acabou o trabalho
16. loop: addiu       $t1,$t1,4          4# Senão,entra no laço, avançando o ponteiro
17.       addu        $t4,$t2,$t3        4# Gera próximo elemento da série
18.       sw          $t4,0($t1)         4# Armazena o que gerou agora
19.       addu        $t2,$zero,$t3      4# Move penúltimo para $t2
20.       addu        $t3,$zero,$t4      4# Move último gerado para $t3
21.       addiu       $t0,$t0,-1         4# Decrementa contador
22.       bne         $t0,$zero,loop     4# Volta para o loop, se ainda não gerou N
23. end:  li          $v0,10             4# Senão, finaliza o programa.
24.       syscall                        4#
25.
26.       .data
27. N:    .word       10                 # Número de elementos a gerar
28. S:    .space      40                 # Local onde armazenar os N elementos
```

a) (1,5 pontos) O programa contém apenas um laço com 13 linhas preparatórias executadas antes dele (linhas 3 a 15) e duas instruções de fechamento do programa (linhas 23 e 24). As instruções fora do laço (antes e depois) executam em 8+5+8+10*4 + 4+4=69 ciclos de clock. O laço ocupa as linhas 16 a 22 e é executado sempre de forma completa, N-2 vezes, pois os dois primeiros elementos são tratados antes de entrar no laço. Uma execução total do laço gasta 7*4=28 ciclos. Como N é 10, o laço é executado exatamente 8 vezes. Agora, o número de ciclos para executar o programa completo pode ser facilmente determinado: num_ciclos= 69 + 28*8 = 293 ciclos.

b) (0,5 pontos) Como a frequência de execução dada é de 400MHz, um ciclo dura (1/(400 * 10^6))s ou seja, 2,5ns ou 2,5x10^{-9} s. Logo, o tempo total de execução do programa, que gasta 293 ciclos de relógio (clock) para executar é 293*(2,5x10^{-9} s) ou seja, *tempo total de execução do programa=7,325x10^{-7}*s.

c) (1,5 pontos)O programa calcula os N primeiros elementos da Série de Fibonacci e os armazena na memória a partir do endereço de memória associado ao rótulo S.

d) (0,5 pontos) O programa não possui sub-rotinas, pois não existe nele uso de instruções jal/jalr (para entrar na subrotina) e jr $ra ou instrução equivalente para voltar da subrotina.

5. [2,5 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:

(1,0 pontos) **Linha [3]** - Dado o formato da instrução bne (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.

(0,5 pontos) **Linha [4]** - Dado o formato da instrução sltiu (0xB rs rt imediato), e sabendo que o formato em linguagem de montagem é sltiu rt,rs, 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.

(1,0 pontos) 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.

3. [1,5 pontos] Converta o numeral -28,25 para o formato de 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. Em outras palavras, há um erro de representação resultante da conversão ou não?

Solução: O formato SP usa 32 bits. O número -28,25 é negativo, logo o campo de sinal do significando é 1. O módulo do número, quando convertido para binário, ponto fixo, fornece $(28,25)_{10} = (11100,01)_2$. Ao normalizar o resultado, tem-se $(28,25)_{10} = (11100,01)_2 = (1,110001 * 2^4)_2$. Da forma normalizada é fácil obter o expoente: 4+127(a polarização do expoente)=131, que convertido para base 2 em 8 bits fornece 10000011. O significando em 23 bits é obtido omitindo-se os caracteres “1,” e completando com 0s os bits menos significativos, o que fornece o seguinte: 11000100000000000000000. O numeral completo no formato IEEE-754 SP é então:

$(1100\ 0001\ 1110\ 0010\ 0000\ 0000\ 0000\ 0000)_2 = 0xC1E20000$

Claramente, esta é a representação exata de $(-28,25)_{10}$, ou seja, não há erro gerado pela conversão, pois toda a parte fracionária (depois de normalizada) cabe nos 23 bits do significando.