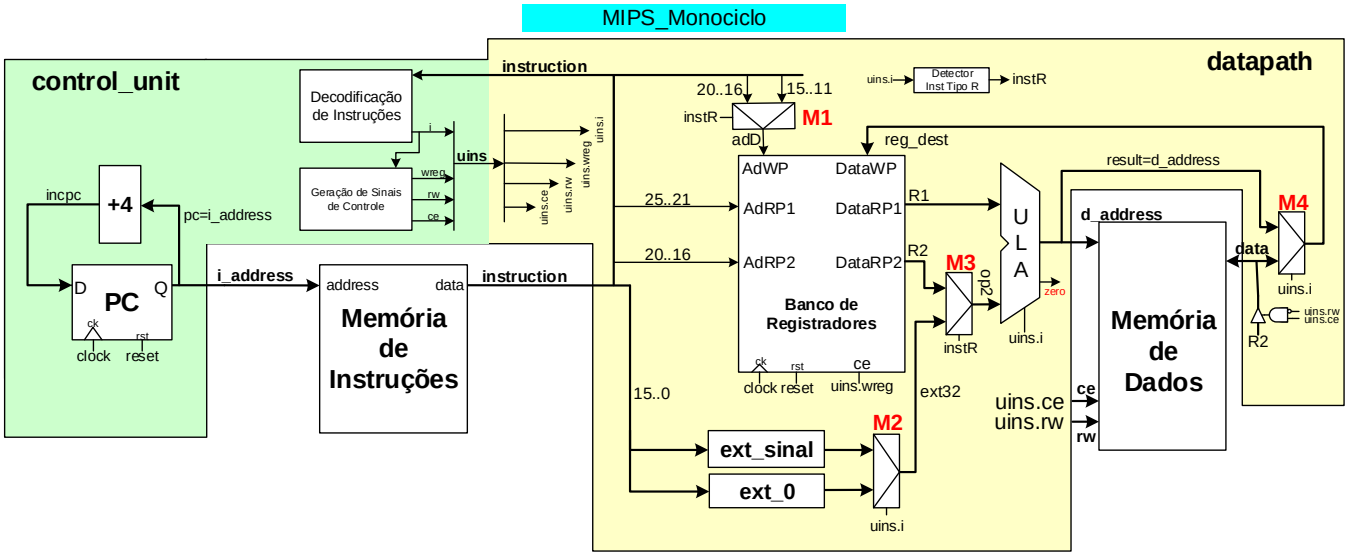
Prova P2 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 21/junho/2019



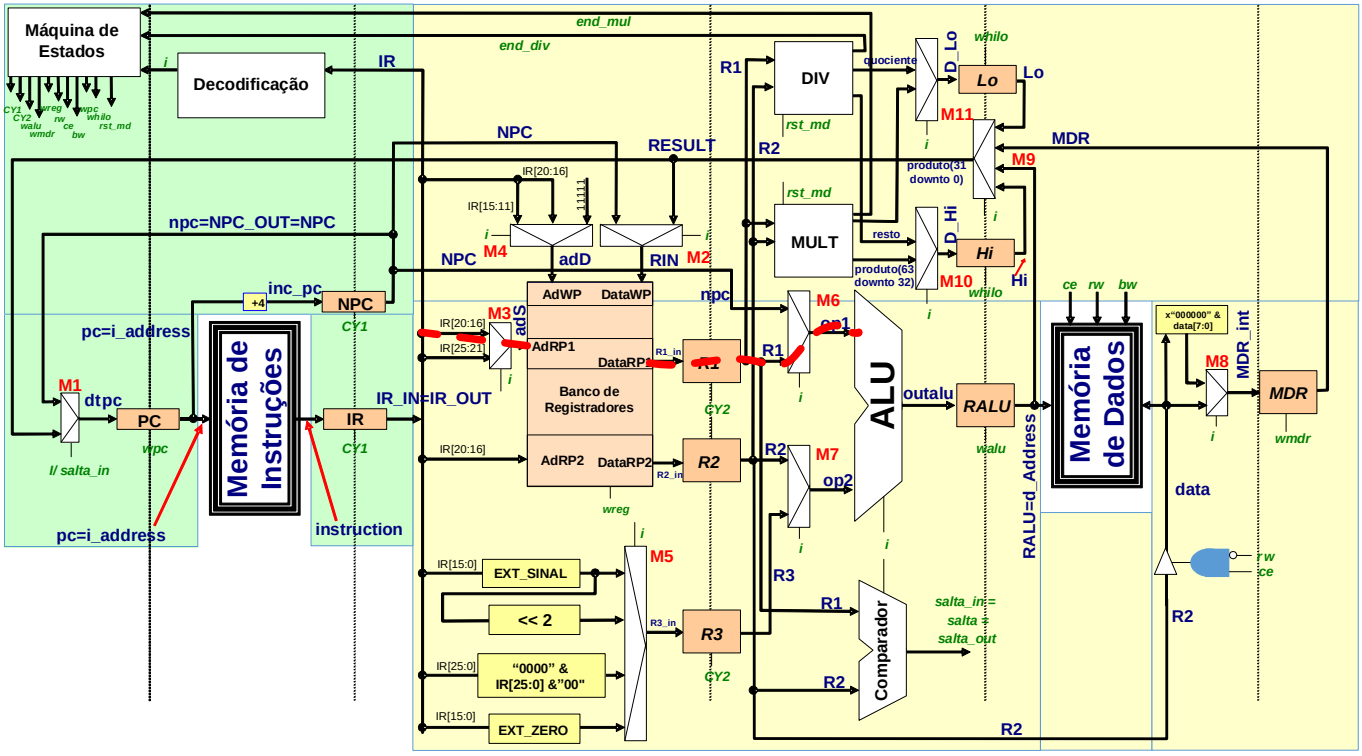
1. [3,0 pontos] Na organização MIPS monociclo mostrada acima, existe no bloco de dados (**`datapath`**) um módulo de hardware que implementa o sinal **`instR`**, um detector de quando uma instrução sendo executada é do tipo R. O código VHDL que corresponde à descrição deste módulo é dado abaixo.

```
instR <= '1' when uins.i=ADDU or uins.i=SUBU or uins.i=AAND or
   uins.i=OOR or uins.i=XXOR or uins.i=NNOR else
         '0';
```

   Pede-se que o hardware descrito nesta linha seja projetado como um conjunto de portas lógicas, partindo dos pressupostos dados a seguir. Abaixo, mostra-se o código que define o tipo `inst_type`, que identifica simbolicamente cada uma das 9 instruções, e que define ainda um símbolo de instrução inválida (`invalid_instruction`). Assuma que o sinal `uins.i` é codificado com a quantidade mínima possível **n** de bits (determine o valor de **n**), iniciando com o valor 0 (representado em **n** bits) associado à instrução `ADDU`, o valor 1 associado à instrução `SUBU` e assim por diante para todas as instruções e para `invalid_instruction`. Assumindo esta codificação, defina o circuito que gera o sinal `instR` e desenhe ele como um diagrama de portas lógicas. Otimize o circuito o máximo que puder.

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

2. [3,0 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 `SLL`. Ressalte também todos os sinais de controle efetivamente usados para executar a instrução. A título de dica, um dos caminhos de dados usados pela instrução já está marcado no desenho, com linhas tracejadas. Use, além do diagrama de blocos dado, a especificação da instrução `SLL` que consta no Apêndice A do livro texto, a descrição VHDL da MIPS multiciclo, bem como a especificação textual da MIPS multiciclo.



3. [4 pontos] Assuma uma frequência de relógio de 500 MHz para a organização **MIPS multiciclo** estudada em aula e diga:

(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** e a pseudo-instrução **li** (equivalente a uma instrução **addiu**) levam 4 ciclos de relógio para executar cada).

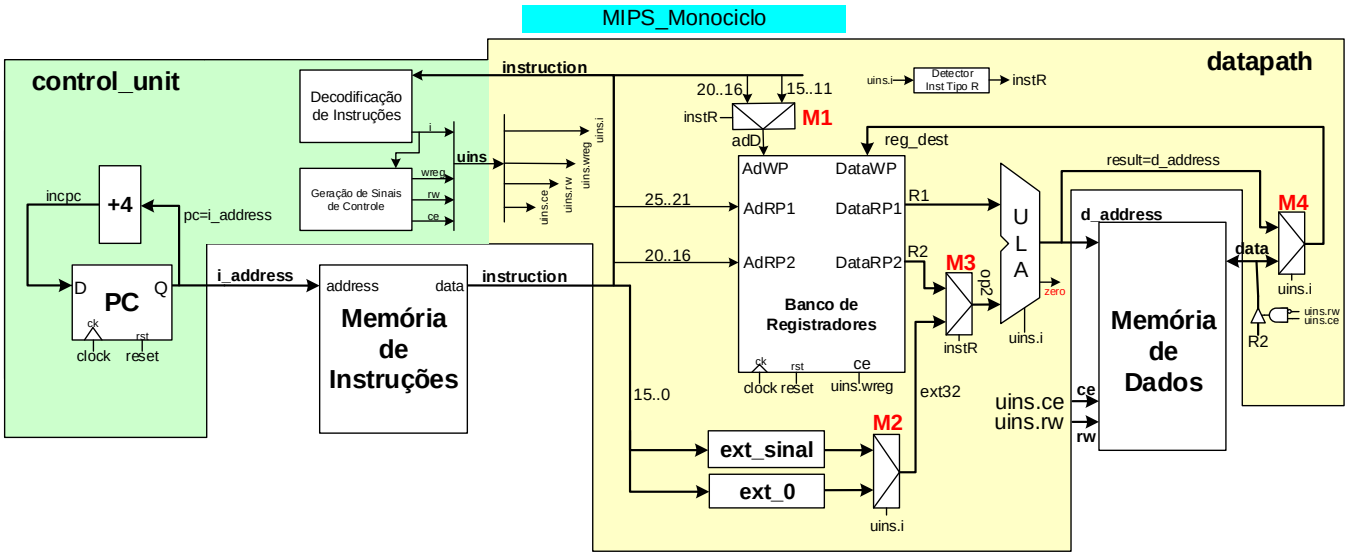
(b) Qual o tempo de execução do programa, em microssegundos?

(c) O que faz este programa.

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

```
1.              .text
2.  main:       la          $s0, string
3.              la          $s1, letras
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.
13.             .data
14. string:     .asciiz     "teste de byte abcde"
15. letras:     .word 0x0
```

Gabarito



1. [3,0 pontos] Na organização MIPS monociclo mostrada acima, existe no bloco de dados (**`datapath`**) um módulo de hardware que implementa o sinal **`instR`**, um detector de quando uma instrução sendo executada é do tipo R. O código VHDL que corresponde à descrição deste módulo é dado abaixo.

```
instR <= '1' when uins.i=ADDU or uins.i=SUBU or uins.i=AAND or
     uins.i=OOR or uins.i=XXOR or uins.i=NNOR else
          '0';
```

Pede-se que o hardware descrito nesta linha seja projetado como um conjunto de portas lógicas, partindo dos pressupostos dados a seguir. Abaixo, mostra-se o código que define o tipo `inst_type`, que identifica simbolicamente cada uma das 9 instruções, e que define ainda um símbolo de instrução inválida (`invalid_instruction`). Assuma que o sinal `uins.i` é codificado com a quantidade mínima possível **n** de bits (determine o valor de **n**), iniciando com o valor 0 (representado em **n** bits) associado à instrução `ADDU`, o valor 1 associado à instrução `SUBU` e assim por diante para todas as instruções e para `invalid_instruction`. Assumindo esta codificação, defina o circuito que gera o sinal `instR` e desenhe ele como um diagrama de portas lógicas. Otimize o circuito o máximo que puder.

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

Solução:

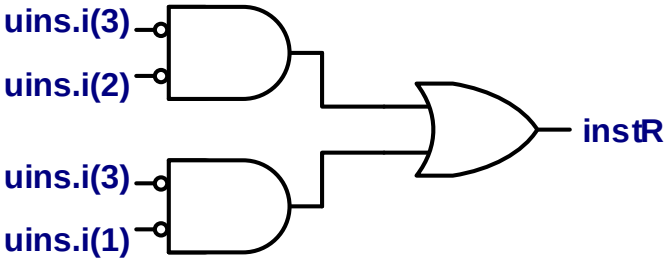
Como existem 9 instruções e o símbolo `invalid_instruction` é necessário codificar 10 informações distintas, o que implica um código com no mínimo 4 bits, donde **n=4**. Com a convenção estabelecida, `ADDU` será codificada 0000, `SUBU` com 0001, etc e `invalid_instruction` com 1001 (9 em decimal). Observando a linha VHDL que define `instR`, nota-se que, dada (i) a codificação dos valores do tipo **`inst_type`** e como (ii) **`uins.i`** é do tipo **`inst_type`** (o que se observa no VHDL do processador), é possível descrever o hardware que implementa instR como uma tabela verdade de uma função Booleana de 4 variáveis Booleanas (os 4 bits de **`uins.i`** são as entradas da função e instR é a saída). A função em questão, com *don´t cares* (X) de saída para os códigos de 4 bits não usados é:

instrução	uins.i(3)	uins.i(2)	uins.i(1)	uins.i(0)	instR
ADDU	0	0	0	0	1
SUBU	0	0	0	1	1
AAND	0	0	1	0	1
OOR	0	0	1	1	1
XXOR	0	1	0	0	1
NNOR	0	1	0	1	1
LW	0	1	1	0	0
SW	0	1	1	1	0
ORI	1	0	0	1	0
Invalid_instruction	1	0	1	0	0
Código não usado	1	0	1	1	X
Códigos não usados	1	1	X	X	X

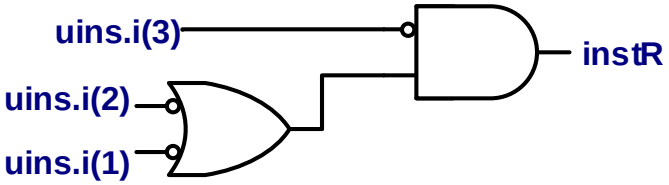
A partir desta tabela verdade é simples usar os conhecimentos advindos da disciplina de Circuitos Digitais e implementar a função usando um mapa de Karnaugh. Apenas a título de informação, uma aplicação web que permite fazer isto está disponível em http://www.32x8.com/. O mapa de Karnaugh, onde já se mostram as adjacências para cobri-lo com foco em gerar uma soma de produtos é:

uins(3) uins(2) → / uins(1) uins(0)	00	01	11	10
00	1	1	X	0
01	1	1	X	0
11	1	0	X	X
10	1	0	X	X

Este mapa, com estas adjacências, corresponde a uma expressão Booleana para instR que é instR=((not (uins(3))) and (not (uins(2)))) or ((not (uins(3))) and (not (uins(1)))), que implementado em portas lógicas corresponde ao circuito abaixo:

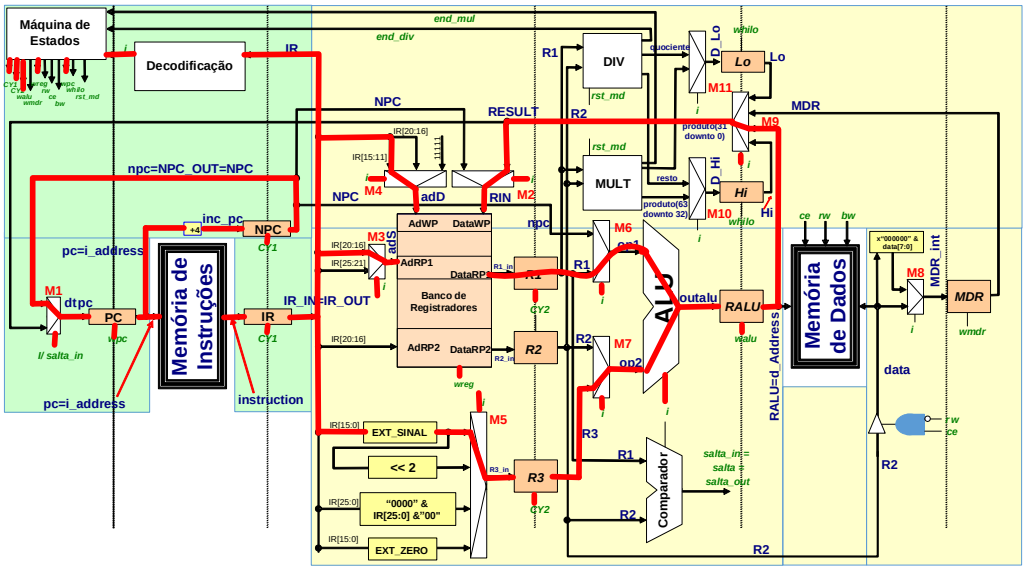


Ainda, note-se que o circuito pode ser mais simplificado usando a opção produto de somas do Karnaugh (cobrindo os 0s e empregando os *don´t cares*, Xs, ao máximo), o que fornece instR=(not (uins(3))) and ((not (uins(2))) or (not (uins(1)))), correspondendo ao circuito:



2. [3,0 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 `SLL`. Ressalte também todos os sinais de controle efetivamente usados para executar a instrução. A título de dica, um dos caminhos de dados usados pela instrução já está marcado no desenho, com linhas tracejadas. Use, além do diagrama de blocos dado, a especificação da instrução `SLL` que consta no Apêndice A do livro texto, a descrição VHDL da MIPS multiciclo, bem como a especificação textual da MIPS multiciclo.

Solução:



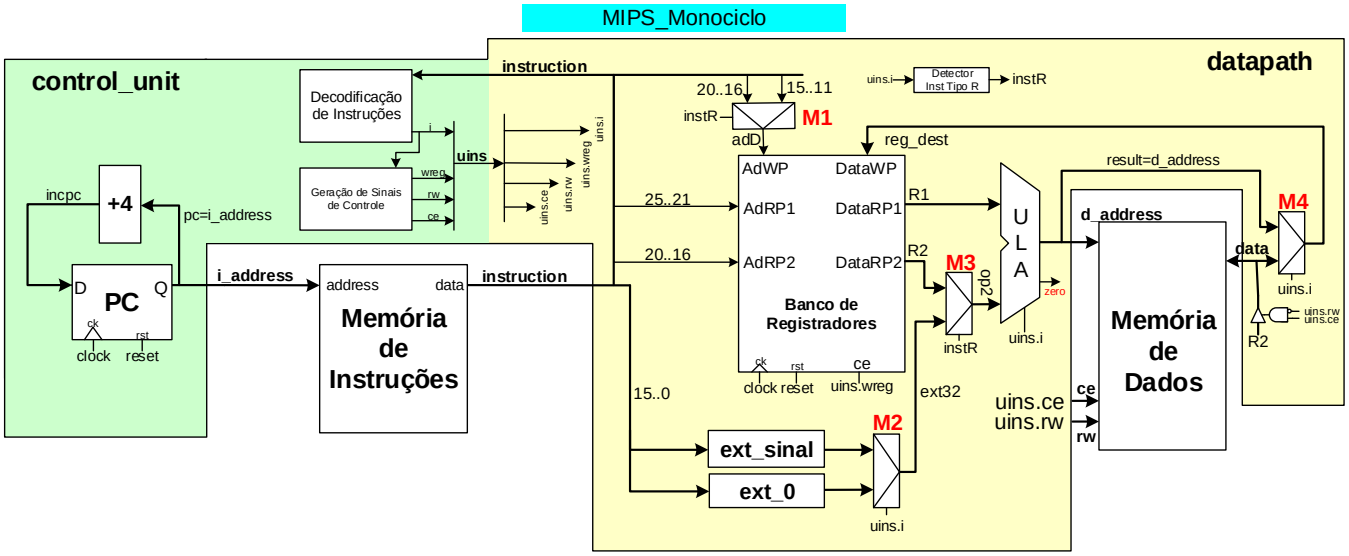
3. [4 pontos] Assuma uma frequência de relógio de 500 MHz para a organização **MIPS multiciclo** estudada em aula e diga:

(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** e a pseudo-instrução **li** (equivalente a uma instrução **addiu**) levam 4 ciclos de relógio para executar cada).

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

(c) O que faz este programa.

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



Solução:

				ciclos
1.		.text		# 0
2.	main:	la	$s0,string	# 8 $s0 recebe ponteiro para cadeia string
3.		la	$s1,letras	# 8 $s1 recebe ponteiro para cadeia destino (letras)
4.	loop:	lbu	$t1,0($s0)	# 5 $t1 recebe próximo caracter de string
5.		blez	$t1,fim	# 4 se chegou ao fim da cadeia, sai do laço
6.		sw	$t1,0($s1)	# 4 senão armazena caracter descompactado em letras
7.		addiu	$s0,$s0,1	# 4 avança ponteiro para novo elemento de string
8.		addiu	$s1,$s1,4	# 4 avança ponteiro para novo elemento de letras
9.		j	loop	# 4 volta a tentar executar o laço
10.	fim:	li	$v0,10	# 4 chegou ao fim do programa, prepara saída
11.		syscall		# 4 e sai.
12.				
13.		.data		
14.	string:	.asciiz	"teste de byte abcde"	
15.	letras:	.word	0x0	

a) O programa é apenas um laço com duas instruções preparatórias do laço (linhas 2 e 3) e duas instruções de fechamento do programa (linhas 10 e 11). Estas linhas executam em 8+8+4+4=24 ciclos. O laço ocupa as linhas 4-10 e é executado ou totalmente (para todo caracter de string diferente de NULL) ou apenas as primeiras duas linhas (quando encontrar o caracter NULL). O laço executado totalmente gasta 5+4+4+4+4+4=25 ciclos e a última vez gasta 9 ciclos. Como a cadeia string possui 20 caracteres (19 imprimíveis e o NULL), o tempo de execução total deste programa pode ser facilmente determinado: num_ciclos= 24 + 19*25 + 9 = 508 ciclos.

b) Como a frequência de execução dada é de 500MHz, um ciclo dura (1/(500 * 10^6))s ou 2ns ou 0,002 microssegundos (μs). Logo, o tempo total de execução do programa que gasta 508 ciclos de relógio (clock) para executar é 508*(0,002)μs ou seja, *tempo total de execução do programa=1,016μs*.

c) Ele descompacta uma cadeia de caracteres string em uma cadeia idêntica em termos de conteúdos (letras), onde cada caracter da nova cadeia ocupa uma palavra (de 32 bits), ao invés de apenas 1 byte. Este é um procedimento que em algumas situações, por exemplo, ele é usado para transferir texto de um processador que possui instruções dedicadas para operar com bytes (lbu, lb, sb no MIPS) para outro processador que eventualmente não possua estas instruções dedicadas e processa mais facilmente uma cadeia de caracteres se ela estiver decompactada (cada caracter ocupando uma palavra inteira).

d) O programa não possui sub-rotinas, pois não existe nele uso de instruções jal/jalr e jr ($ra).