UNIVERSIDADE DE BRASÍLIA
DEPARTAMENTO DE CIÊNCIAS DA COMPUTAÇÃO
ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES - 1º/2016

Laboratório 3 - CPU MIPS Uniciclo

Alunos:

André Abreu R. de Almeida - 12/0007100
Arthur de Matos Beggs - 12/0111098
Bruno Takashi Tengan - 12/0167263
Gabriel Pires Iduarte - 13/0142166
Guilherme Caetano - 13/0112925
João Pedro Franch - 12/0060795
Rafael Lima - 10/0131093

Parte A - Apresentação do ambiente de desenvolvimento e Interface com o Processador

6) Analise o processador MIPS PUMv.4.2 UNICICLO desenhando o diagrama de blocos do Caminho de Dados usando a estrutura base vista em aula e a tabela verdade do Bloco Controlador.

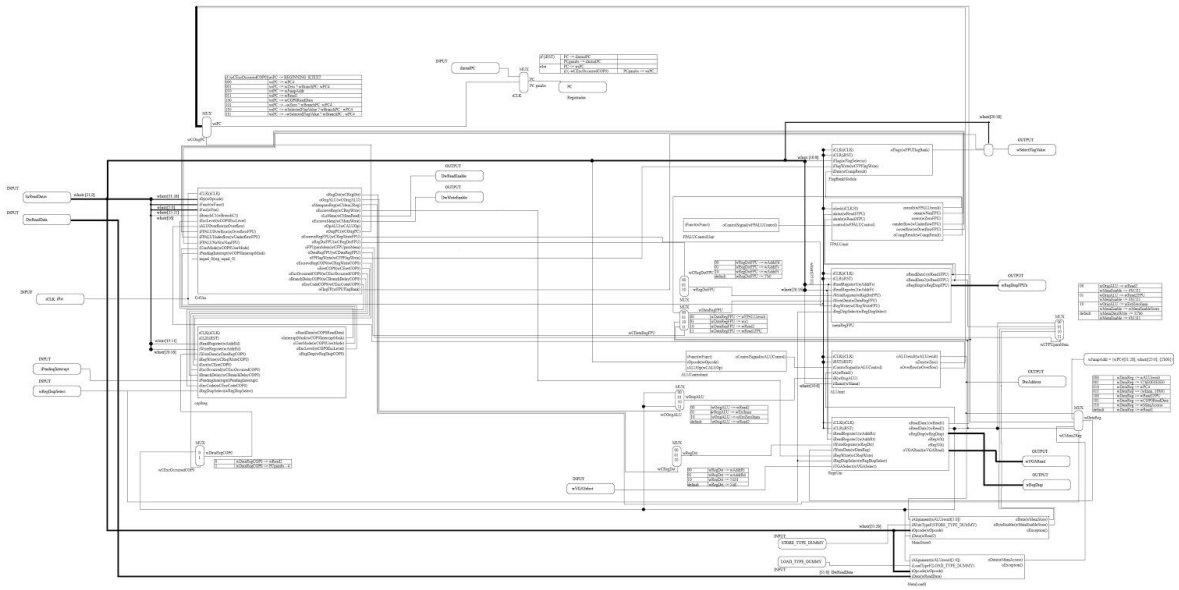

Figura 1 - Caminho de dados

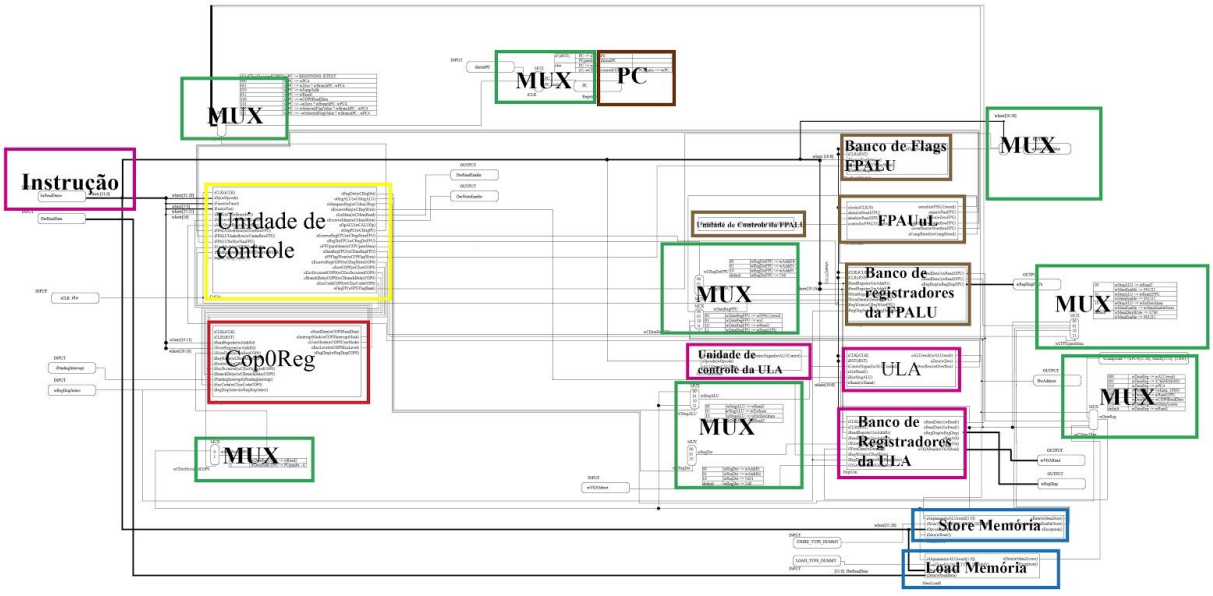

Figura 2 - Caminho de dados usando a estrutura base vista em aula

7) Crie um programa teste.s que verifique o correto funcionamento de TODAS as instruções da ISA implementada, teste usando simulação por forma de onda e pela implementação na DE2.

O programa se encontra no diretório “Lab3 > teste.s”.

8) Encontre a frequência máxima de clock do processador na qual a ISA ainda é corretamente executada.

Para obter-se a frequência máxima de operação do processador, deve-se descobrir qual a função que demora o maior tempo para ser executada. Para isso, utilizou-se a função Time Quest Analyser do software Quartus II 13.0 , e com isso pôde-se perceber que a instrução mais demorada era a DIV. A partir disso, para descobrir qual seria a frequência máxima de operação, basta testar repetidamente o código na FPGA com valores variáveis de frequência e descobrir o maior valor para o qual o código funciona ou o menor valor para o qual ele para de funcionar corretamente. Escolheu-se como parâmetros as frequências de 200MHz e 300MHz e os resultados para estas frequências e com os seus divisores na FPGA estão mostrados na tabela abaixo.

Frequência da FPGA	Máxima frequência operante com base no divisor	Mínima frequência inoperante com base no divisor
200 MHz	100 MHz	200 MHz
300 MHz	75 MHz	150 MHz

Com os dados obtidos da tabela, pode-se concluir que utilizando-se da frequência de 200 MHz na FPGA o maior valor obtido de frequência de clock foi de 100 MHz, e seu próximo divisor maior já não funcionava corretamente, logo se sabe que a frequência máxima de funcionamento está dentro desta faixa. Aplicando-se o mesmo pensamento para a frequência de 300 MHz temos que a região em que a frequência máxima de operação se encontra está contida na região hachurada da figura abaixo:

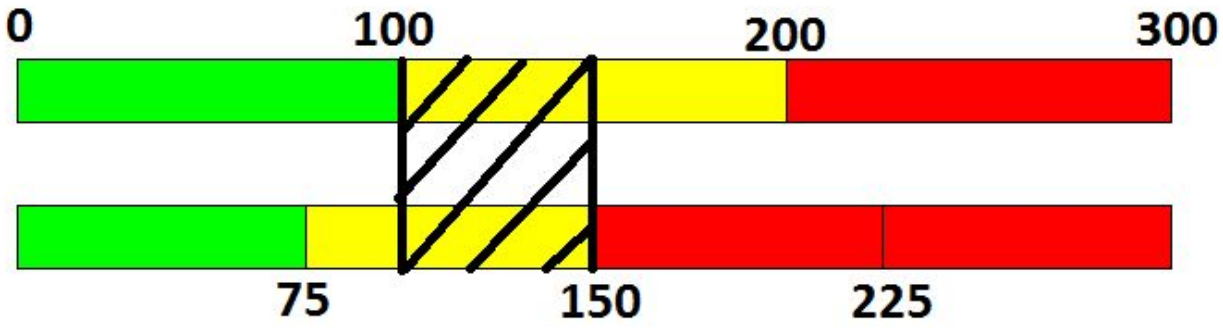

Fig. 3

Logo, a frequência máxima de operação da FPGA para o correto funcionamento da ISA implementada está entre 100MHz e 150MHz.

9) Implemente as instruções abaixo em conformidade com a ISA MIPS (livro See MIPS Run e Manual do MIPS):

bgez $t0,LABEL # $t0>=0 ? PC=LABEL : PC=PC+4

bgezal $t0,LABEL # $t0>=0 ? PC=LABEL e $ra=PC+4 : PC=PC+4

bgtz $t0,LABEL # $t0>0 ? PC=LABEL : PC=PC+4

blez $t0,LABEL # $t0<=0 ? PC=LABEL : PC=PC+4

bltz $t0,LABEL # $t0<0 ? PC=LABEL : PC=PC+4

bltzal $t0,LABEL # $t0<0 ? PC=LABEL e $ra=PC+4 : PC=PC+4

a e b) Indique as modificações necessárias no caminho de dados. / Indique as modificações necessárias no bloco de controle.

Os parametros para o funcionamento das instruções:

INSTRUÇÃO	OPCODE	$RS	$RT	IMM
BGEZ	6'b000001	-	5'b00001	-
BGEZAL	6'b000001	-	5'b10001	-
BLTZ	6'b000001	-	5'b00000	-

BLTZAL	6’b000001	-	5’b10000	-
BLEZ	6’b000110	-	-	-
BGTZ	6’b000111	-	-	-

Como pode perceber, as instruções bgez, bgezal, bltz e bltzal são instruções do tipo I unusuais no qual usa o seu campo do parametro “$rt” para funcionar como se fosse o “funct” das instruções tipo R.

Para a implementação das instruções foi necessário realizar alterações em 5 partes do processador (caminho de dados, bloco de controle, parametros, ULA, bloco de controle da ULA). As alterações são especificadas a seguir:

Parametros.v :

Foi adicionado os parametros para as 6 instruções. Foram 3 parametros de opcode, 4 parametros para $rt e 1 parametro para operação da ULA.

```
30          OPSGT = 5'b10111, //23     //2016/1

117         OPCBGE_LTZ  = 6'h01, // 1/2016 // Para as instruções bgez, bgezal, bgltz, bltzal
118         OPCBGTZ     = 6'h07, // 1/2016
119         OPCBLEZ     = 6'h06, // 1/2016
120
121   /* Campo $rt */              // 1/2016
122         RTBGEZ      = 5'b00001,
123         RTBGEZAL    = 5'b10001,
124         RTBLTZ      = 5'b00000,
125         RTBLTZAL    = 5'b10000,
```

Fig. 4 - Os novos parametros adicionados.

ALU.v :

Foi adicionado a operação sgt.

```
53        OPSGT://2016/1 - implementada para as operacoes bgtz e blez
54           oALUresult <= iA > iB;
```

Fig. 5 - Operação sgt adicionado na ULA.

ALUControl.v :

Foi adicionado um fio de entrada “iRt”, que serve para pegar o campo “$rt” da instrução, e colocamos os controles para quando forem os casos das novas instruções.

Pode perceber que adicionamos um controle para o opcode de jal, isso porque para a implementação das instruções bgezal e bltzal foi necessária uma alteração em um multiplexador do caminho de dados, que veremos logo adiante, e por este motivo mudamos a lógica da instrução jal no processador.

```
20   input wire [5:0] iFunct, iOpcode, iRt;       // 1/2016. adicionado iRt.
```

```
104              OPCJAL:                              //2016/1
105                 oControlSignal <=    OPAND;
106              OPCBLEZ,                             //2016/1
107              OPCBGTZ:
108                 oControlSignal <= OPSGT;
109              OPCBGE_LTZ:                          //2016/1
110              begin
111                 case (iRt)
112                    RTBGEZ,
113                    RTBGEZAL,
114                    RTBLTZ,
115                    RTBLTZAL:
116                       oControlSignal <= OPSLT;
117                    default:
118                       oControlSignal <= 5'b00000;
119                 endcase
120              end
```

Fig. 6 - modificações do bloco de controle da Ula.

Datapath_UNI.v :

Construimos o caminho de dado entre o campo “$rt” das instruções para o bloco de controle e o bloco de controle da ula e adicionamos um novo caso ao multiplexador que seleciona qual será a segunda entrada da ULA.

```
222   .iRt(waddrRt),                // 1/2016, Implementar intruções bgez, bgezal, bgltz, bltzal.
```

```
311      // 1/2016, Implementar intruções bgez, bgezal, bgltz, bltzal.
312      .iRt(wAddrRt)
```

```
359   /*Decide o que entrara na segunda entrada da ULA*/
360   always @(*)
361      case(wCOrigALU)
362         2'b00: wOrigALU <= wRead2;
363         2'b01: wOrigALU <= wExtImm;
364         2'b10: wOrigALU <= wExtZeroImm;
365         2'b11: wOrigALU <= 5'b00000;     // 1/2016
366      endcase
```

Fig. 7 - Caminho de dados criado e multiplexador da entrada da ULA modificada.

Também foi alterado o multiplexador que decide em qual registrador o dado será escrito. Essa mudança foi importante para a implementação do bgezal e do bltzal e resultou na modificação do jal.

Anteriormente este multiplexador possuia um caso vago e o caso 2'b10 como o registrador "$ra", para implementar o link somente quando o branch ocorresse (bgezal e bltzal) era preciso impor uma condição para que o registrador $ra fosse o registrador de destino. Como a condição para o branch ja era usado como a saída da ULA sendo zero ou não essa mesma condição foi imposta para a seleção do registrador $ra para dar o link.

Com esse novo condicional para dar o link, a instrução jal ficou afetada, na hora da implementação tinhamos duas opções, ou adicionar 1 bit a "wCRegDst" e realizar as alterações necessárias pelo processador todo, ou mudarmos apenas o funcionamento da instrução jal, e por uma decisão de projeto escolhemos seguir a segunda alternativa. Basicamente fizemos com que jal realize uma operação que seu resultado sempre fosse zero, no caso um and com zero (no jal original o resultado da ULA não interferia em nada se ele realizava o link ou não).

```
348   /*Decide em qual registrador o dado sera escrito*/
349   always @(*)
350   ⊟   case(wCRegDst)
351         2'b00: wRegDst <= wAddrRt;
352         2'b01: wRegDst <= wAddrRd;
353         2'b10: wRegDst <= wZero ? 5'd31: 5'd0; //$ra ou $zero    1/2016
354         2'b11: wRegDst <= ~wZero ? 5'd31: 5'd0; //$ra ou $zero   1/2016
355         default: wRegDst <= 5'd0;
356      endcase
```

Fig. 8 - Multiplexador do registrador de destino do dado de escrita modificado.

Control_UNI.v :

Foi adicionado um fio de entrada "iRt", que serve para pegar o campo "$rt" da instrução, mudamos o controle da instrução jal e adicionamos o controle das 6 novas instruções.

O controle das 6 novas instruções podem ser vistas a partir da linha 798 até a linha 960 do arquivo.

```
8     input wire [5:0] iOp, iFunct, iRt,        // 1/2016. adicionado iRt.

322       OPCJAL:                          //alterado em 1/2016 para implementar bgezal e bltzal.
323       begin
324          oRegDst <= 2'b10;
325          oOrigALU <= 2'b11;            //alterado 1/2016 2'b00 => 2'b11.
326          oMemparaReg <= 3'b010;
327          oEscreveReg <= 1'b1;
328          oLeMem <= 1'b0;
329          oEscreveMem <= 1'b0;
330          oOrigPC <= 3'b010;
331          oOpALU <= 2'b11;              //alterado 1/2016 2'b00 => 2'b11.
332          oEscreveRegFPU <= 1'b0;
333          oRegDstFPU <= 2'b00;
334          oFPUparaMem <= 2'b00;
335          oDataRegFPU <= 2'b00;
336          oFPFlagWrite <= 1'b0;
337          oEscreveRegCOP0 <= 1'b0;
338          oEretCOP0 <= 1'b0;
339          oExcOccurredCOP0 <= wIntException;
340          oBranchDelayCOP0 <= 1'b1;
341          oExcCodeCOP0 <= EXCODEINT;
342       end
```

Fig. 9 - Entrada iRt adicionado e modificações do controle do jal.

A numeração das linhas de código apresentadas nas imagens podem diferir das linhas dos arquivos, uma vez que o código foi reformatado. Entretanto, o conteúdo do código continua idêntico ao apresentado.

c) Crie um programa teste que comprove o correto funcionamento das novas instruções:

O arquivo pode ser encontrado nos diretórios “Processors > MIPS-PUM-v4.2 > Docs > testes” e “Lab3” com o nome “testeBRANCH.s”. O teste pode ser verificado pelo acompanhamento dos valores dos registradores $t0, $ra e PC.

10) Implemente a chamada do sistema Syscall 49, que recebe como argumento $a0 o endereço do cartão SD, $a1 endereço da memória e $a2 a quantidade de bytes a serem transferidos, e retorne $v0=0 em caso de sucesso e $v0=1 em caso de erro.

A chamada de sistema Syscall 49 foi adicionada ao arquivo SYSTEMv51.v, porém ainda não está completamente funcional por problemas ainda não resolvidos no hardware. O controlador do cartão SD utiliza 6 bytes da memória para suas operações, sendo eles 4 bytes denominados SD_INTERFACE_ADDR iniciados na posição de memória 0xFFFF0250, por onde o software passa para o hardware controlador o argumento da posição de memória que se deseja obter, 1 byte denominado

SD_INTERFACE_CTRL iniciado na posição 0xFFFF0254 que fornece ao software informações do estado do controlador (IDLE ou BUSY, além de informações para depuração) e 1 byte denominado SD_INTERFACE_DATA iniciado na posição 0xFFFF0255 que contém o byte lido do cartão SD.

O syscall implementado verifica o estado do controlador, e caso seu estado seja IDLE, inicia a leitura dos dados byte a byte do cartão. O retorno do syscall é hardcoded para sempre retornar 0 (sucesso) pela atual falta de mecanismos no hardware que indiquem a falha de comunicação.

Para que a leitura do cartão possa ser feita byte a byte, um cartão SDSC de 2 Gb é utilizado. O estado CMD16 foi adicionado à máquina de estados para enviar ao cartão um comando CMD16 para setar o tamanho do bloco de leitura para 1 byte. O estado POLL_CMD16 foi criado para verificar a resposta R1 do cartão SD e averiguar se o cartão aceitou o comando. Cartões dos tipos SDHC e SDXC não funcionarão nessa implementação, uma vez que o tamanho do bloco de leitura desses cartões é fixado no tamanho do setor, que é de 512 bytes.

O arquivo testeReadSD.s foi criado e adicionado ao diretório “Processors > MIPS-PUM-v4.2 > Docs > testes” e possui um cabeçalho que explica o uso do syscall 49. O valor de teste setado para o .eqv SD_DATA_ADDR deve ser obtido por meio de um hex editor para o cartão em que se está realizando os testes.

No estado atual do projeto, o cartão SD inicializa corretamente, como demonstra a resposta R1 de POLL_CMD e POLL_CMD16. Porém, os dados ainda não são lidos do cartão SD. Para a correta inicialização, é necessário utilizar o clock iCLK controlado pelas chaves iSW[7:0]. Durante a inicialização, o cartão precisa de um clock de 100~400KHz, e após a correta inicialização, a velocidade pode ser aumentada até 25MHz. Porém, alguns cartões SD não suportam 25MHz em modo SPI, podendo suportar somente 10MHz. Um divisor de clock está em elaboração para fazer a transição automática entre a frequência de inicialização e a de operação a partir do iCLK_50.