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

Laboratório 4 - CPU MIPS Multiciclo

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

2) Analise o processador Multiciclo desenhando o diagrama de blocos do Caminho de Dados usando a estrutura base vista em aula e a máquina de estados do Bloco Controlador:

O diagrama do DataPath do processador multiciclo do MIPS-PUM-v4.2 se encontra em anexo no arquivo “Lab4 > DataPathMulticiclo.pdf” e os diagramas da máquina de estados se encontram em anexo nos arquivos “Lab4 > maquinas_estados_1.jpg“ e “Lab4 > maquinas_estados21.jpg“, sendo eles continuações uns dos outros.

3) Usando seu programa teste.s, verifique o correto funcionamento de TODAS as instruções da ISA implementada, e verifique todas as chamadas syscall listadas na documentação do processador:

Os teste foram divididos por tipo de instrução. Inicialmente foram testadas as instruções do tipo J, depois o acesso a memória, seguido dos branch. Uma vez que todas estas haviam sido testadas foram implementados os testes de operações aritméticas e o resultado de cada operação foi comparado com uma constante. Por fim foram testadas as chamadas de syscall para gera números aleatórios, imprimir números inteiros e contar o tempo.

Foi também definido uma rotina auxiliar para identificar error, na ocorrência que qualquer problema, a instrução que gerou o problema e a linha de código eram armazenadas nos registradores e mostradas na tela como valores inteiros.

Para cada teste foi impresso um código na tela identificado por uma letra. As instruções do tipo J são representadas pelo código J 1 2 3 no canto da tela. Os teste dos branches estão representados pela parte B 1 2 , que significa que o teste obteve sucesso para os branch beq e bne. Logo abaixo temos os teste da das operações aritiméticas. E por fim algumas operações com o syscall.

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

Para determinar o valor de frequência máxima a ser utilizado no processador existem duas maneiras possíveis: utilizar o divisor de frequência que foi habilitado para ser utilizado manualmente através do conjunto de chaves SW[8..0] na placa FPGA DE2-70, ou implementar um divisor de frequência dentro do código que será testado na FPGA DE2-70 antes de passá-lo para a placa, e alterar o valor de frequência dentro do código e mandá-lo novamente para a placa. Para o experimento em questão, adotou-se a primeira opção, e adotando como clock de referência o clock de 50MHz disponível na FPGA, executamos nosso arquivo teste de verificação da ISA e aumentamos sua frequência até que ele parasse de responder como o esperado.

Como resultado deste procedimento, verificou-se que o programa funciona corretamente até a configuração das chaves SW[8..0] = 01100100, que em decimal representa o valor 100. Portanto, para o caso de verificação da frequência máxima de processamento utilizando nosso arquivo teste de verificação da ISA temos que F = 50000000/100 = 500000 Hz = 500kHz. Isso significa que a frequência ideal de clock se encontra na faixa de frequência que compreende o valor de 500kHz e o valor imediatamente acima (divisor = 99 -> configuração SW[8..0] = 01100011) que indica frequência de 505050.5 Hz, e portanto uma faixa de frequência de aproximadamente 5000Hz.

5) 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	-	5'b00000	-
BGTZ	6'b000111	-	5'b00000	-

Como pode perceber, todas as instruções são instruções do tipo I não usuais no qual o seu campo "$rt" é usado como se fosse o "funct" das instruções tipo R.

Para a implementação das instruções foi necessário realizar alterações em 4 partes do processador (caminho de dados, bloco de controle, parâmetros, bloco de controle da ULA). As alterações são especificadas a seguir:

Parâmetros.v :

Foram adicionados os parâmetros para a máquina de estados do multiciclo. No total 6 parâmetros para as 6 instruções.

```
288     //Adicionados em 1/2016
289     BGEZ            = 6'd55,
290     BGEZAL          = 6'd56,
291     BLTZ            = 6'd57,
292     BLTZAL          = 6'd58,
293     BGTZ            = 6'd59,
294     BLEZ            = 6'd60;
```

Fig. 1 - Os novos parâmetros adicionados.

ALUControl.v :

Foram adicionadas novas condições para o controle das instruções blez e bgtz que estavam faltando para o seu correto funcionamento.

```
OPCBLEZ,                        //2016/1
OPCBGTZ:
    case (iRt)
        RTZERO:         //Garante que $rt seja zero/instruções válidas
            oControlSignal  = OPSGT;
        default:        //instr. inválida
            oControlSignal  = 5'b00000;
    endcase
OPCBGE_LTZ:                     //2016/1
```

Fig. 2 - modificações do bloco de controle da Ula.

Datapath_MULT.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.

```
384     //adicionado em 1/2016
385     .iRt (wRT)
```

```
458     //adicionado em 1/2016
459     .iRt (wRT)
```

```
473   // Mux ALU input 'B'
474   always @(*)
475       case (ALUSrcB)
476           3'd0: wALUMuxB <= B;
477           3'd1: wALUMuxB <= 32'd4;
478           3'd2: wALUMuxB <= wImmediate;
479           3'd3: wALUMuxB <= wLabelAddress;
480           3'd4: wALUMuxB <= wuImmediate;
481           3'd5: wALUMuxB <= 32'd0;                //adicionado em 1/2016.
482           default: wALUMuxB <= 32'd0;
483       endcase
```

Fig. 3 - 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.

Os motivos e as soluções criadas para a implementação foram as mesmas do relatório passado mudando apenas a codificação. Foi necessário adicionar um novo caso ao multiplexador que decide o que é escrito no registrador.

```
407   // Mux WriteReg
408   always @(*)
409      case (Store)
410         3'd0: wWriteRegister <= wRtorRd;  //Normal mode
411         3'd1: wWriteRegister <= wALUZero ? 5'd31: 5'd0;     //  $ra ou $zero    1/2016
412         3'd2: wWriteRegister <= 5'd04;    //$a0 Store timer LO
413         3'd3: wWriteRegister <= 5'd05;    //$a0 Store timer HI
414         3'd4: wWriteRegister <= 5'd04;    //$a0 Store Random
415         3'd5: wWriteRegister <= wRT;      //mfc1
416         3'd6: wWriteRegister <= wRT;      //mfc0 - feito no semestre 2013/1 para implementar a
417         3'd7: wWriteRegister <= ~wALUZero ? 5'd31: 5'd0;     //  $ra ou $zero    1/2016
418      endcase
```

Fig. 4 - Multiplexador do registrador de destino do dado de escrita modificado.

```
422   // Mux WriteData
423   always @(*)
424      case (Store)
425         3'd0: wRegWriteData <= wMemorALU;   //Nor
426         3'd1: wRegWriteData <= PC;        // $RA J
427         3'd2: wRegWriteData <= RegTimerLO;  //Sto
428         3'd3: wRegWriteData <= RegTimerHI;  //Sto
429         3'd4: wRegWriteData <= RandInt;     //Sto
430         3'd5: wRegWriteData <= FP_A;     //mfc1
431         3'd6: wRegWriteData <= COP0_A;      //mfc
432         3'd7: wRegWriteData <= PC;     //1/2016
433
434      endcase
```

Fig. 5 - Multiplexador do dado de escrita modificado.

Control_MULT.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.

```
16       //adicionado no 1/2016
17       iRt

23    input wire [4:0] iFmt, iRt;      // 1/2016. Adicionado iRt.
```

```
177            //operações implementadas em 1/2016 - bgtz, blez, bgez, bgezal, bgltz, bltzal.
178            OPCBGTZ:
179               nx_state <= wCOP0PendingInterrupt ? COP0EXC : BGTZ;
180            OPCBLEZ:
181               nx_state <= wCOP0PendingInterrupt ? COP0EXC : BLEZ;
182            OPCBGE_LTZ:
183               case (iRt)
184                  RTBGEZ:
185                     nx_state <= wCOP0PendingInterrupt ? COP0EXC : BGEZ;
186                  RTBGEZAL:
187                     nx_state <= wCOP0PendingInterrupt ? COP0EXC : BGEZAL;
188                  RTBLTZ:
189                     nx_state <= wCOP0PendingInterrupt ? COP0EXC : BLTZ;
190                  RTBLTZAL:
191                     nx_state <= wCOP0PendingInterrupt ? COP0EXC : BLTZAL;
192               endcase
```

```
512         JAL:
513         begin
514            //FPRegDst[2], FPDataReg[2], FPRegWrite, FPPCWriteBc1t, FPPCW
515            word  <= 40'b0000000000000000000011000000000010111010010;
516            nx_state <= FETCH;
517         end

520         //adicionado em 1/2016, bgez, bgezal, bltz, bltzal.
521         BGEZ:
522         begin
523            word  <= 40'b0000000000000000000000010000000011111010100;
524            nx_state <= FETCH;
525         end
526
527         BGEZAL:
528         begin
529            word  <= 40'b0000000000000000000010010000000011111010110;
530            nx_state <= FETCH;
531         end

533         BLTZ:
534         begin
535            word  <= 40'b0000000000000000000000010000000011111010100;
536            nx_state <= FETCH;
537         end
538
539         BLTZAL:
540         begin
541            word  <= 40'b0000000000000000000111010000000011111010110;
542            nx_state <= FETCH;
543         end

551         BGTZ: //1/2016
552         begin
553            //FPRegDst[2], FPDataReg[2], FPRegWrite, FPPCWriteBc1t, FP
554            word  <= 40'b0000000000000000000000010000000011111010100;
555            nx_state <= FETCH;
556         end
557         BLEZ://1/2016
558         begin
559            //FPRegDst[2], FPDataReg[2], FPRegWrite, FPPCWriteBc1t, FP
560            word  <= 40'b0000000000000000000000010000000011111010100;
561            nx_state <= FETCH;
562         end
```

Fig. 6 - Entrada iRt adicionado, modificações do controle do jal e controle das instruções novas.

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-v4.3 > Docs > testes” e “Lab4” com o nome “testeBRANCH.s”. O teste pode ser verificado pelo acompanhamento dos valores dos registradores $s7, $ra e PC.

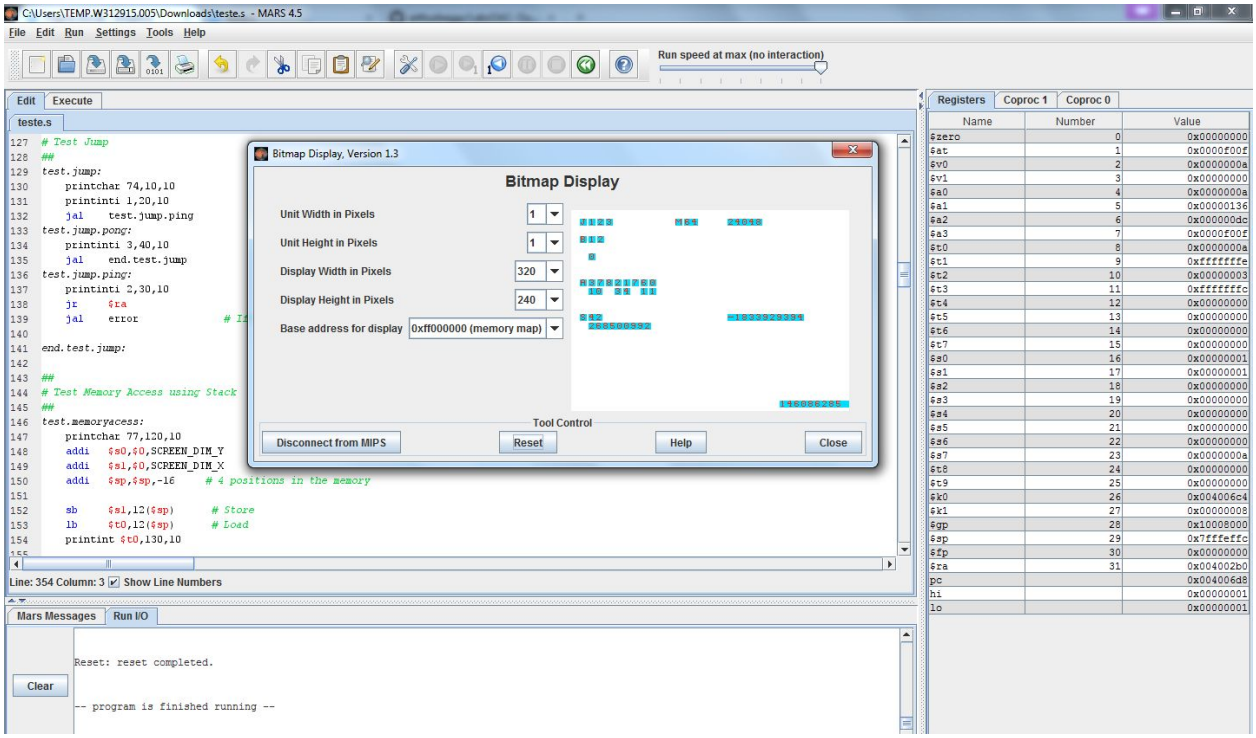

Fig. 7 - Teste rodando no Mars

6) Implemente a chamada do sistema Syscall 49, que receba como argumentos: $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 SYSTEMv52.s do processador MIPS-PUM-v4.3. Seis bytes de memória são endereçados para o controlador do cartão SD, 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. As posições de memória foram adicionadas na tabela de Mapeamento de Memória do arquivo MIPS.docx.

O syscall implementado verifica o estado do controlador, e caso seu estado seja IDLE, inicia a leitura dos dados byte a byte do cartão. Depois da primeira leitura, o registrador $a0 que armazena o endereço da memória de origem (cartão SD) e o registrador $a1 que armazena o endereço da memória de destino (alguma posição de memória na DataMemory) são incrementados em 1, e o registrador $a2 que armazena a quantia de bytes a serem lidos é decrementado em 1. Outra leitura é realizada, formando um loop que termina quando a quantia de bytes a serem lidos (registrador $a0) for igual a zero. Entre o envio do endereço desejado (por meio de uma instrução sw) e a leitura do dado lido do cartão (por meio de uma instrução lw), o syscall verifica se o valor de SD_INTERFACE_CTRL é igual a 0x00 para se assegurar de que o dado está pronto.

O retorno do syscall é hardcoded para sempre retornar 0 (sucesso) pela atual falta de mecanismos no hardware que indiquem a falha de comunicação. Um novo estado pode ser adicionado à FSM do sd_controller para que se faça a verificação da resposta R1 dos comandos enviados, e caso a resposta seja diferente de 0x00, o syscall retorne 1.

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 FSM para enviar ao cartão um comando CMD16 para setar o tamanho do bloco de leitura para 1 byte. O estado POLL_CMD foi alterado para verificar se a resposta R1 do cartão SD é 0x00, indicando que todos os comandos enviados estão OK e que o cartão inicializou corretamente. 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.

Durante o processo de inicialização do cartão, deve ser usada uma frequência de 400KHz no sclk, e após a correta inicialização, a frequência pode ser aumentada para 10~25MHz. O controlador possui um divisor de frequência que alterna automaticamente a frequência do sclk de acordo com o estado da FSM. Para testes, as frequências foram diminuídas para 195.25KHz para inicialização e 12.5MHz para transferência de dados.

O arquivo testeReadSD.s foi criado e adicionado ao diretório “Processors > MIPS-PUM-v4.3 > 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 mostra a resposta R1 obtida após o comando ACMD41, e as respostas subsequentes dos demais comandos (obtidas com o uso do SignalTap). Ao enviar um comando de READ_BLOCK, o cartão responde conforme o esperado, isto é, resposta R1 igual a 0x00 e bloco de dado com 1 byte de início igual a 0xFE, 1 byte de dado e 2 bytes de CRC16, conforme a figura abaixo.

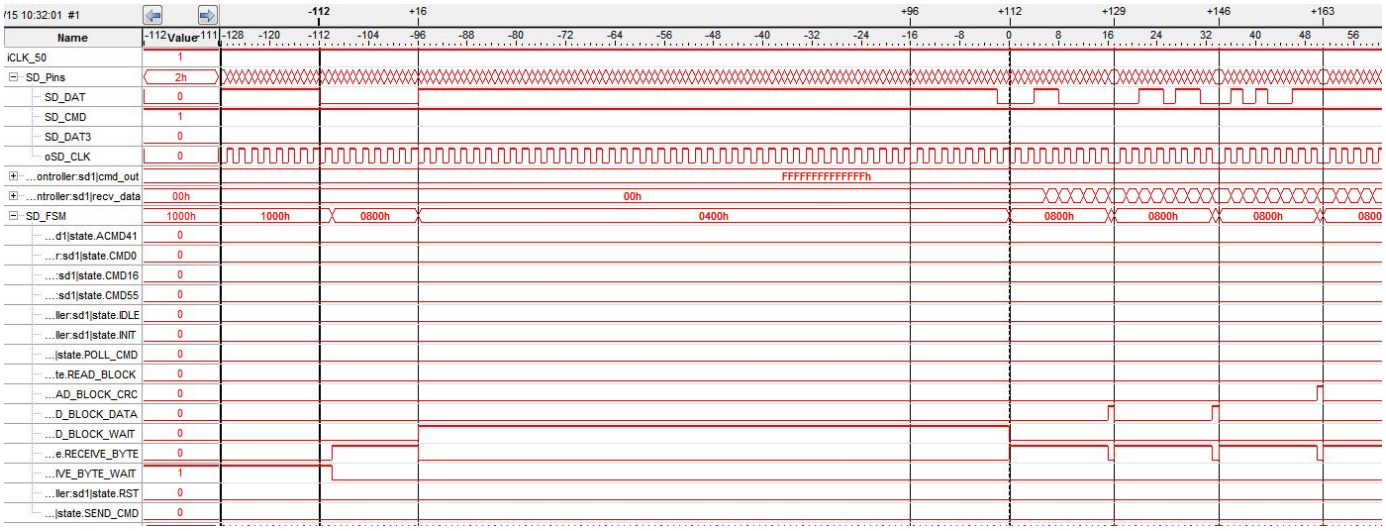

Fig. 8 - Leitura dos sinais do controlador do cartão SD

Porém, o byte lido não coincide com o esperado ao analisar o cartão SD com um hex editor. Ao ler de endereços diferentes do cartão, dados diferentes são recebidos, mas nunca o que era esperado. A origem do problema ainda não foi identificada.