Organização e Arquitetura de Computadores - Laboratório 3

João Victor de Souza Calassio[1] João Pedro Assunção Coutinho[2] Vinicius da Silva Rocha[3]
Jaqueline Gutierri Coelho[4] Luis Filipe Siqueira Ribeiro[5]

[1]Universidade de Brasília, Dep. Ciencia da Computação, Brasil

1 Requerimentos físicos e temporais das CPUs Uniciclo RV32I, RV32IM, RV32IMF

RV32I			
Requerimentos Físicos		**Requerimentos Temporais**	
Número de ALMs	4044	Maior Atraso TPD	24.839
Número de Registradores	3974	Maiores Tempos TH / TCO / TSU	0.762 / 14.268 / 2.380
Quantidade de bits de memória	2823168	Máxima Frequência de Clock Utilizável	67.3MHz
Número de DSP	12	Slack(hold) / Slack(setup)	0.453 / 9.237

Table 1: Requerimentos físicos e temporais do RV32I

RV32IM			
Requerimentos Físicos		**Requerimentos Temporais**	
Número de ALMs	6800	Maior Atraso TPD	27.357
Número de Registradores	3980	Maiores Tempos TH / TCO / TSU	0.762 / 13.498 / 3.510
Quantidade de bits de memória	2823168	Máxima Frequência de Clock Utilizável	75.6MHz
Número de DSP	24	Slack(hold) / Slack(setup)	0.426 / 10.053

Table 2: Requerimentos físicos e temporais do RV32IM

RV32IMF			
Requerimentos Físicos		**Requerimentos Temporais**	
Número de ALMs	9691	Maior Atraso TPD	29.361
Número de Registradores	6322	Maiores Tempos TH / TCO / TSU	0.762 / 15.500 / 2.215
Quantidade de bits de memória	2870784	Máxima Frequência de Clock Utilizável	75.59MHz
Número de DSP	30	Slack(hold) / Slack(setup)	0.542 / 10.052

Table 3: Requerimentos físicos e temporais do RV32IMF

2 Banco de Registradores, datapath e tratamento de exceções

2.1 Banco de Registradores

O primeiro passo foi implementar o banco de registradores de controle e status (CSR). Este se encontra no módulo CSRegisters e foi feito com base no banco de registradores inteiros.

O banco de registradores possui 70 registradores de 32 bits cada, com suporte para todos os 4.096 registradores definidos na ISA RISC-V, bastando alterar a quantidade (o grupo definiu a quantidade como 70 por não achar necessário definir todos os 4.096). Todos os registradores necessários para a correta detecção de exceções estão presentes.

Em contraste com o banco de registradores inteiros, o banco de registradores CS foi definido para sempre ler e escrever no mesmo registrador CSR (pois as instruções com os CSR sempre lêem e escrevem no mesmo), e possui pinos específicos para os registradores uepc(65) e ucause(66) para a escrita simultânea em múltiplos registradores (no caso das exceções de erro) em um mesmo período de clock no processador UNICICLO.

2.2 Exceções 2 e 8

As exceções 2 (instrução inválida) e 8 (ecall) foram implementadas com modificações no bloco de controle, de forma que este detecte e avise quando alguma delas ocorrer. O bloco de controle recebeu novas saídas, que indicam infomações sobre exceções e manuseio dos CSR:

(A explicação de cada dado está presente na descrição do datapath)

```
output          oCSRegWrite, // enable de escrita no CSR definido na instrucao
output          oUCAUSEWrite, // enable de escrita no UCAUSE
output          oUEPCWrite, // enable de escrita no UEPc
output [ 1:0] oCSType, // define qual CSR vai ser lido e escrito
output [31:0] oUCAUSEData, // dado do UCAUSE (sai do controle por causa das excecoes tipo 2 e 8)
output [ 2:0] oCSRWSource, // define qual a fonte do dado a ser escrito no CSR
```

O bloco de controle já tinha um comportamento específico para instruções inválidas (os ”default” dos multiplexadores), onde todas as saídas eram definidas como 0. Então para a exceção 2, os default foram mantidos e adicionados os fios para o correto tratamento da exceção 2.

Já para o ecall, foi adicionado o OPCode da instrução na verificação dos OPCodes, assim como o Funct3 e Rs2 (para facilitar na implementação das próximas instruções com os CSR, que possuem mesmo OPCode, além das instruções uret e ebreak que possuem o mesmo Funct3).

2.3 Exceções 4, 5, 6 e 7

Para as exceções de leitura e escrita na memória, foram necessárias modificações nos blocos MemStore e MemLoad, que já possuiam um sinal (chamado oException) de 1 bit indicando se havia acontecido um erro de alinhamento ou não.

Esse fio foi utilizado, mas modificado para ser de 2 bits e indicar se havia alguma exceção e o tipo (00 = nenhuma exceção, 01 = endereço de load/store desalinhado, 10 = fora do segmento data ou MMIO).

Além disso, o fio de verificação de alinhamento foi removido do input, para dar lugar ao fio do endereço de leitura completo (e assim facilitar a verificação das exceções). A verificação de alinhamento agora é feita internamente, com o fio iAlignment interno.

Também foi adicionada uma entrada do OPCode da instrução atual, pois a verificação de alinhamento só levava em consideração o Funct3 da instrução, e isso gerava exceções de desalinhamento no load/store em outras instruções com Funct3 igual.

Para ambos (load e store), há apenas uma verificação combinacional que verifica o OPCode e se o endereço de load/store está desalinhado. Se estiver alinhado, verifica se está dentro dos segmentos data ou MMIO, e definem o valor correto para a saída oException. Essa saída é tratada no datapath.

```
always @(*) begin
  if(iOPCode == OPC_LOAD) begin
    if(wCheckAlignment)
        oException  <= 2'b01; // endereco desalinhado
    else begin
      if((iReadAddr >= BEGINNING_DATA && iReadAddr <= END_DATA) || iReadAddr >= BEGINNING_IODEVICES)
        oException  <= 2'b00; // endereco dentro do .data e MMIO, valido
      else
        oException  <= 2'b10; // endereco fora dos segmentos
    end
  end else
        oException  <= 2'b00; // nao eh load, valido
end
```

2.4 Exceções 0 e 1

Essas exceções são verificadas no próprio datapath, juntamente com o sinal recebido do oException dos blocos MemLoad e MemStore.

Inicialmente, é verificada a exceção de endereço de instrução desalinhado, e se esse não ocorrer, é verificado o endereço fora do segmento text, e assim sucessivamente para as exceções 0, 1, 4, 5, 6, 7. Se nenhuma delas ocorrer, são verificados os fios recebidos do controle, para definir a leitura/escrita nos CSR e a origem do PC.

2.5 Testes detecção e tratamento de exceções

Os programas que testam a correta detecção e tratamento de exceções estão na pasta ”ExceptionTests”. Não há o teste para o tratamento da exceção 8 pois já existe o testeECALL. O teste da exceção 2 está em um arquivo .mif (por não ser possível criar um arquivo com instrução inválida pelo RARS).

3 Instruções de controle e status (CS)

As instruções CS manuseiam os registradores inteiros e os CSR. Com exceção do ecall, ebreak e uret, todas salvam o valor de um CSR específico em um registrador inteiro definido. Em seguida, escrevem um novo valor no mesmo registrador CSR.

As instruções são do tipo I, conforma a ISA RISC-V e têm o seguinte formato:

Imm	Rs1	Funct3	Rd	OPCode
12 bits	5 bits	3 bits	5 bits	1110011

Table 4: Control and Status Instructions

Onde o registrador CSR a ser lido e escrito está no imediato de 12 bits (possibilitando os 4.096 CSR definidos na ISA RISC-V).

As instruções ecall, ebreak e uret possuem o mesmo Funct3 (000), mas consideram o Rs2 e Funct7 para identificação.

Funct7	Rs2	Rs1	Funct3	Rd	OPCode
7 bits	5 bits	00000	000	00000	1110011

Table 5: ecall, uret, ebreak format

3.1 Descrição das Alterações Feitas

3.1.1 Instruções CSR

Para as instruções CSR, o bloco de controle foi modificado de forma a detectar os seus OPCodes e Funct3 corretamente. Todas elas definem as mesmas saídas para os fios de controle e status, com excessão do "oCSRWSource", que define qual vai ser o dado a ser escrito no CSR, e é tratado no datapath.

csrrw	I	CSR Read & Write	R[rd]=CSR; CSR=R[rs1]	csrrw rd,rs1
csrrs	I	CSR Read & Set	R[rd]=CSR; CSR=CSR|R[rs1]	csrrs rd,rs1
csrrc	I	CSR Read & Clear	R[rd]=CSR; CSR=CSR&!R[rs1]	csrrc rd,rs1
csrrwi	I	CSR Read & Write Imm	R[rd]=CSR; CSR=imm	csrrwi rd,imm
csrrsi	I	CSR Read & Set Imm	R[rd]=CSR; CSR=CSR|imm	csrrsi rd,imm
csrrci	I	CSR Read & Clear Imm	R[rd]=CSR; CSR=CSR&!imm	csrrci rd,imm

Figure 1: Instruções CSR

```
FUNCT3_CSRXXX:
    begin
      oUCAUSEWrite   <= OFF; // nao escreve ucause
      oUCAUSEData    <= ZERO;
      oUEPCWrite     <= OFF; // nao escreve UEPC
      oOrigPC        <= 3'b000; // PC vem de PC+4
      oMem2Reg       <= 3'b100; // dado do rd vem do csr
      oRegWrite      <= ON; // escreve nos registradores inteiros
      oCSRegWrite    <= ON; // escreve em csr
      oCSRWSource    <= 3'bXXX; // fonte do dado vinda de uma operacao de CSR com registrador ou imediato
      oCSType        <= 2'b11; // tipo instrucao csr
    end
```

3.1.2 ecall

Para o ecall, são definidas as escritas apenas nos registradores UCAUSE e UEPC, com o UCAUSE = 8, e UEPC = PC atual. É definido que o registrador CS a ser lido é o UTVEC (através do fio CSType), e que a nova origem do PC será o dado lido do CSR.

3.1.3 uret

Para a instrução uret, não há escrita nos CSR, e é definida a leitura do CSR UEPC (através do fio CSType), e que a nova origem do PC será o dado lido do CSR.

3.1.4 ebreak

Para o ebreak, foi criado um novo fio de saída do controle, chamado oBreak. Essa é a unica instrução que define esse fio como 1, e esse fio se estende até o bloco TopDE, onde foi criado um novo bloco chamado Break_Instruction_Control, que recebe o sinal vindo do controle e, apenas na borda de descida do clock atual (ou caso haja um reset), define um sinal que é enviado para o CLOCK_INTERFACE através da entrada iBreak deste.

O bloco Break_Instruction_Control foi criado por um problema no tempo de atraso dos fios, que podia por um pequeno instante detectar os fios da instrução com OPCode, Funct3 e Rs2 iguais aos da instrução ebreak, ativando o sinal oBreak por um pequeno instante, e travando o clock. Na borda de descida do clock, todos os sinais já foram estabilizados e isso não acontece.

3.1.5 Caminho de dados

No caminho de dados, foram adicionados alguns fios de entrada para receber os dados do controle (os mesmos descritos na seção 2.2). Também foi adicionado o bloco dos CSRegisters, com o fio de saída do dado lido (chamado wCSRead).

Foi adicionado o multiplexador principal que seleciona quais são os dados definidos para entrada nos CSRegisters (conforme explicado no item 2.4).

Além deste, também há um multiplexador para definir a fonte do dado a ser escrito no CSR, escolhido à partir do dado selecionado no multiplexador CSR principal, que verifica as exceções (fio wSCSRWSource). O dado escolhido é salvo no wCSWriteData, que é a entrada para gravação no banco de registradores CS.

Há um multiplexador para definir qual CSR vai ser lido e escrito, escolhido através do dado definido no fio wSCSType. São considerados os seguintes "tipos" de leitura e escrita:

- 01 = ecall ou qualquer exceção (erros), com leitura do UTVEC e escrita do UTVAL
- 10 = uret, com leitura do UEPC e sem escrita

- 11 = instrução, que lê e escreve no mesmo CSR contido no imediato de 12 bits.

definindo assim, os fios wCSReadRegister (CSR a ser lido) e wCSWriteRegister (CSR a ser escrito).

O multiplexador já existente que define qual a origem do PC foi modificado, para poder comportar a nova origem do PC (dado lido do CSR). O fio wCOrigPC também foi modificado, e agora é de 3 bits.

3.1.6 MemLoad e MemStore

As modificações nos blocos MemLoad e MemStore já foram especificadas no item 2.3

3.1.7 ImmGen

O bloco gerador de imediato teve uma pequena modificação, para reconhecer o OPCode das instruções de controle e status e gerar o imediato corretamente.

3.1.8 Parametros

O arquivo de parâmetros teve pequenas alterações, apenas para adicionar os OPCodes, Funct3, Funct7 e Rs2 referentes às instruções com os CSR.

3.2 Desenho do Caminho de Dados

O desenho do caminho de dados pode ser encontrado na pasta do projeto (em melhor definição que a Figura 2).

Lembrando que o caminho de dados não está exatamente como o caminho de dados real do processador, por não haver módulos importantes (como o clock interface, para poder representar o break) e os multiplexadores com mais de 2 entradas foram separados em multiplexadores em cascata para melhor visualização e adaptação.

Foi baseado no desenho já existente disponibilizado pelo monitor da disciplina.

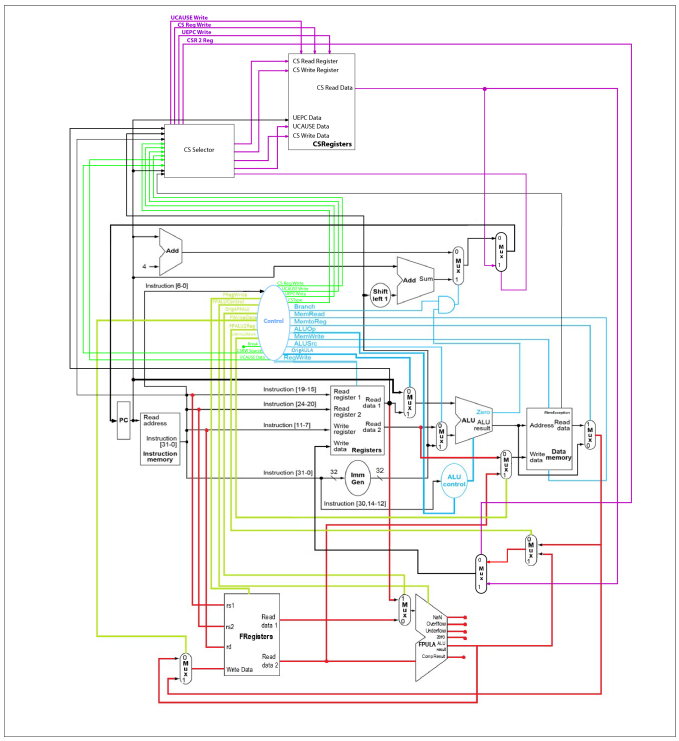

Figure 2: Caminho de dados com os CS Registers

3.3 Tabela da Verdade do Bloco de Controle

A tabela verdade do bloco de controle pode ser encontrada na pasta do projeto, juntamente com o desenho do caminho de dados, por ser muito grande.

Foi baseada na tabela verdade já existente disponibilizada pelo monitor da disciplina.

3.4 Analise e Comentários sobre as dificuldades

A maior dificuldade foi encontrar a causa do problema descrito com o ebreak. Neste, ao contrário das outras instruções que só vão precisar que os valores definidos no controle estejam corretos na borda de subida do clock, a parte do Clock_Inteface que recebe o sinal de parada não depende do clock.

Então no momento em que foi implementada a ligação direta do fio de break do controle com a interface de clock, os testbenches começavam a parar em certas instruções todas as vezes que encontravam. A conclusão do atraso dos fios foi tomada por causa da semelhança dos dados das instruções que causavam esse problema, com os dados do ebreak. Por exemplo:

- beq (OPCode 1100011, Funct3 000)
- add (OPCode 0110011, Funct3 000)

Poderiam causar, na transição dos fios, o opcode 1110011, que é o OPCode das instruções relativas aos CSR, e assim, podendo entrar na condição do ebreak (dependendo do restante da instrução).

4 TESTBENCH DAS INSTRUÇÕES CRIADAS E ANÁLISE

Link do testbench no RARS.
https://www.youtube.com/watch?v=SU33fhT4qXA&list=PLzC5HQPWD90mQ-djTuhrP1Kfi1cznPDGj&index=6&t=0s
Link do testbench na placa.
https://www.youtube.com/watch?v=29mDd-JMAIY&list=PLzC5HQPWD90mQ-djTuhrP1Kfi1cznPDGj&index=1

5 TESTEECALLV14.S E SYSTEMV14.S E ANÁLISE

A atualização do SYSTEMv14 e testeECALL foi simples e sem grandes problemas.

A detecção dos erros foi inserida no SYSTEMv14 (assim como a tela azul). O único pequeno problema enfrentado foi a modificação dos registradores t0 e t1 na verificação da causa de chamada do exceptionHandling, que não estavam sendo salvos na pilha. Esse problema foi rapidamente identificado e corrigido. Todos os "M_Ecall" foram substituidos por "ecall", assim como o "M_Uret" por "uret". As words "UTVEC" e "UEPC" foram removidas, pois não são mais necessárias.

Com relação ao testeECALL, apenas substituimos os "M_Ecall" por "ecall".

O teste de todos os ecalls funcionou corretamente e da forma esperada, sem nenhum problema.

Link para o vídeo no YouTube a seguir.
https://www.youtube.com/watch?v=ohJDH9QsM_s&list=PLzC5HQPWD90mQ-djTuhrP1Kfi1cznPDGj&index=2

O arquivo macros2.s também foi modificado, removendo as macros M_Ecall e M_Uret, e modificando a M_SetEcall para utilizar somente as instruções csrrw e csrrsi.

6 ENTREGADOR DE PIZZAS DE1-SOC VS RARS

Como pode ser observado no video abaixo, a execução do programa da DE1-SoC foi consideravelmente mais rápida que a execução no RARS. O cálculo de todas as rotas, para 10 casas, terminou em 3 minutos e COMPLETAR AQUI OS SEGUNDOS, em oposição aos 10m44s quando executado no RARS.

Isso acontece pois o RARS é um simulador e é executado através de um sistema operacional, com o interpretador Java.

Na DE1-SoC, há apenas um programa sendo executado. Para a gravação, o divisor de frequências foi definido como 4 (00100, nos SW[4:0]).

Link para o vídeo no YouTube a seguir.
https://www.youtube.com/watch?v=h4Y_CxFMKBs&list=PLzC5HQPWD90mQ-djTuhrP1Kfi1cznPDGj&index=3

7 CONVERSOR ANALÓGICO PARA DIGITAL

Para implementação do conversor ADC, foi utilizada a IP do Quartus "ADC Controller for DE-series Boards". No projeto, foi definido o nome ADC_Controller, e foi criada uma instância deste no arquivo "ADC_Interface" já existente no projeto RISC-V2.0 inicial.

O barramento de IO é utilizado para escrever na memória MMIO o dado recebido de cada canal correspondente do ADC (0 a 7).

Os dados do ADC só podem ser lidos (sem escrita nos endereços de memória dos canais 0 a 7).

O programa que movimenta um pixel branco na tela é utilizado como um meio de testar o conversor ADC. Sua implementação é simples; os conteúdos das coordenadas horizontais e verticais são lidos dos endereços predefinidos e de acordo com seus valores o pixel se moverá. Foram definidos 'ranges' para que o movimento fosse sensível à sensibilidade, no caso, 3 ranges para cada sentido (cima, baixo, esquerda e direita). Dessa forma, o pixel possui três velocidades, rápido(pulam-se 9 pixeis por vez), médio(pulam-se 3 pixeis por vez) e lento(pula-se 1 pixel de cada vez).

Como extra, foi implementado uma forma de desenhar utilizando o programa responsável por mover o pixel. Sempre que o botão central do joystick é pressionado, o programa 'muda de modo' deixando de pintar de preto a posição anterior à atual. A função de desenhar linhas, utilizada no programa do caixeiro viajante foi utilizada para facilitar a implementação dessa parte.

Link para o vídeo no YouTube a sguir.
https://www.youtube.com/watch?v=G-lD_fKp-hY&list=PLzC5HQPWD90mQ-djTuhrP1Kfi1cznPDGj&index=5&t=0s