Universidade Federal Rural de Pernambuco - UFRPE

Arquitetura e Organização de Computadores - 2025.1
Prof. Vítor A Coutinho

Projeto 02 - Implementação de MIPS (subconjunto da ISA) em Verilog

ORIENTAÇÕES

1. **Esta atividade compões nota de Projetos 2 (PE2) da 2a VA;**
2. A atividade deve ser feita em grupos de no máximo 4;
3. **Data de entrega e apresentação: 24/07/2025 (horário da aula)**
4. A entrega será feita via Google classroom. Sendo em grupo, basta um dos membros anexar a entrega, escrevendo no mural quem são os demais membros; os demais membros devem dar “Turn in/Retornar” na atividade (mesmo sem anexar nada) e escrever no mural de mensagem privada da atividade quem foi o membro que enviou. Também é permitido (e recomendável) desenvolver e entregar a atividade pelo Github (OBS: mesmo assim, o link do repositório precisa ser enviado no Classroom e cada membro precisa dar turn in para colaborar na organização da correção).
5. Os códigos devem estar comentados, explicando cada bloco de código; cada arquivo deve apresentar um cabeçalho (utilizando comentários) contendo todas as informações (nome do grupo, atividade 2VA, disciplina, semestre letivo, questão referente ao arquivo, breve descrição do conteúdo do arquivo, etc.);
6. Seu projeto deve ser o mais organizado possível. Utilize múltiplos arquivos no diretório do projeto;
7. **Não serão toleradas cópias da internet ou de outros colegas;**

Quem não seguir as orientações à risca pode ter a nota anulada!!

Projeto - Visão Geral

Este projeto consiste em implementar uma versão simplificada do núcleo do MIPS monociclo em hardware utilizando uma HDL (hardware description language). O projeto deve ser implementado em verilog. Sugere-se utilizar alguma das seguintes ferramentas:

- Quartus + Modelsim (Intel/altera) **(Priorizar essa opção! Disponível versão grátis lite)**
- EDA Playground (ferramenta totalmente online);
- Icarus verilog (Ferramenta opensource, mas mais limitada).

O projeto deve ser implementado de forma organizada e hierárquica. Utilize diferentes arquivos para a implementação dos diferentes módulos. Recomenda-se fortemente o uso do github para desenvolvimento do projeto. Uma descrição detalhada do projeto é apresentada nas seções seguintes.

Instruções

Todas as instruções presentes na implementação devem ser codificadas exatamente como na forma completa do MIPS, incluindo os opcodes. Existem 3 formatos de instrução principais no MIPS. Os campos em cada tipo são dispostos de forma que os mesmos campos estejam sempre no mesmo lugar para cada tipo. Vide referência. (Obs: aparentemente o domínio desta referência está fechado. Vejam essas outras: referência 2, referência 3)

Fields: Type	31-26	25-21	20-16	15-11	10-06	05-00
R-Type	opcode	$rs	$rt	$rd	shamt	funct
I-Type	opcode	$rs	$rt	immediate		
J-Type	opcode	address				

O hardware projetado deve ser capaz de executar as instruções listadas a seguir.

Instruções tipo R

Essas instruções são identificadas por um opcode de 0 e são diferenciadas por seus valores funcionais. Com exceção das 3 primeiras instruções de deslocamento, essas operações usam apenas registradores. Observe que, além das operações aritméticas, essas instruções também incluem saltos e a instrução de chamada do sistema.

Ref	Instrução	Operação	Notas
R.1	`add $rd, $rs, $rt`	`R[$rd] = R[$rs] + R[$rt]`	
R.2	`sub $rd, $rs, $rt`	`R[$rd] = R[$rs] - R[$rt]`	
R.3	`and $rd, $rs, $rt`	`R[$rd] = R[$rs] & R[$rt]`	
R.4	`or $rd, $rs, $rt`	`R[$rd] = R[$rs] | R[$rt]`	
R.5	`xor $rd, $rs, $rt`	`R[$rd] = R[$rs] ^ R[$rt]`	
R.6	`nor $rd, $rs, $rt`	`R[$rd] = !(R[$rs] | R[$rt])`	
R.7	`slt $rd, $rs, $rt`	`R[$rd] = R[$rs] < R[$rt]`	Signed comparison

Ref	Instrução	Operação	Notas
R.8	`sltu $rd, $rs, $rt`	`R[$rd] = R[$rs] < R[$rt]`	Unsigned comparison
R.9	`sll $rd, $rt, shamt`	`R[$rd] = R[$rt] << shamt`	
R.10	`srl $rd, $rt, shamt`	`R[$rd] = R[$rt] >> shamt`	Unsigned right shift
R.11	`sra $rd, $rt, shamt`	`R[$rd] = R[$rt] >> shamt`	Signed right shift
R.12	`sllv $rd, $rt, $rs`	`R[$rd] = R[$rt] << R[$rs]`	
R.13	`srlv $rd, $rt, $rs`	`R[$rd] = R[$rt] >> R[$rs]`	Unsigned right shift
R.14	`srav $rd, $rt, $rs`	`R[$rd] = R[$rt] >> R[$rs]`	Signed right shift
R.15	`jr $rs`	`PC = R[$rs]`	

OBS: **`R[$rd]`** ***refere-se ao registrador especificado por*** **`$rd`** ***no banco de registrador (regfile) denominado aqui por*** **`R`**.

Instruções tipo I

Essas instruções são identificadas e diferenciadas por seus números de opcode (qualquer número maior que 3). Todas essas instruções apresentam um imediato de 16 bits, que é estendido por sinal para um valor de 32 bits em todas as instruções (exceto as instruções and, or e xor que estendem zero e a instrução lui na qual não importa). As instruções de desvio também multiplicam efetivamente o imediato por 4, para obter um deslocamento de byte.

Ref	Instrução	Operação	Notas
I.1	`addi $rt, $rs, imm`	`$rt = $rs + SignExt(imm)`	
I.2	`andi $rt, $rs, imm`	`$rt = $rs & {0x0000, imm}`	
I.3	`ori $rt, $rs, imm`	`$rt = $rs | {0x0000, imm}`	
I.4	`xori $rt, $rs, imm`	`$rt = $rs xor {0x0000, imm}`	
I.5	`beq $rs, $rt, imm`	`if($rs == $rt)` `  PC = PC + 4 + SignExt(imm) << 2`	Mutiplica por 4 no hardware
I.6	`bne $rs, $rt, imm`	`if($rs != $rt)` `  PC = PC + 4 + SignExt(imm) << 2`	Mutiplica por 4 no hardware
I.7	`slti $rt, $rs, imm`	`$rt = $rs < SignExt(imm)`	Signed comparison
I.8	`sltiu $rt, $rs, imm`	`$rt = $rs < SignExt(imm)`	Unsigned comparison
I.9	`lui $rt, imm`	`$rt = {imm, 0x0000}`	
I.10	`lw $rt, imm($rs)`	`$rt = D_mem[$rs + SignExt(imm)]`	

Ref	Instrução	Operação	Notas
I.11	`sw $rt, imm($rs)`	`D_mem[$rs + SignExt(imm)] = $rt`	

Instruções tipo J

Essas instruções são identificadas e diferenciadas por seus números de opcode (2 e 3). As instruções de salto usam endereçamento pseudo-absoluto, no qual os 4 bits superiores do endereço computado são obtidos em relação ao contador do programa.

Ref	Instrução	Operação
J.1	`j address`	`PC = {(PC + 4)[31:28], address, 0b00};`
J.2	`jal address`	`R[31] = PC + 4;` `PC = {(PC + 4)[31:28], address, 0b00};`

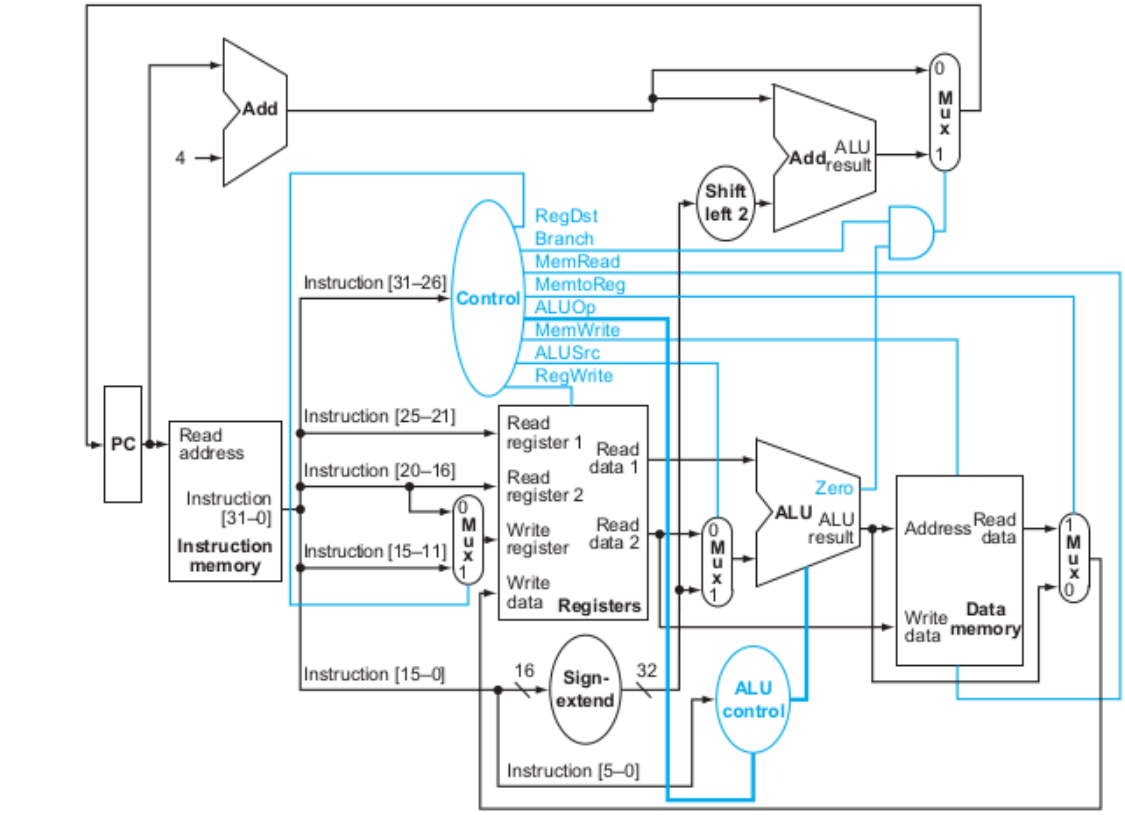
Módulos

O núcleo MIPS deve ser composto pelos seguintes módulos (utilize o nome especificado entre parênteses para cada)

- Contador de Programa (PC);
- Memória de instrução (i_mem);
- Memória de dados (d_mem);
- Banco de registradores (regfile);
- Unidade lógica e aritmética (ula);
- Unidade de controle (ctrl).
- Unidade de controle da ULA (ula_ctrl)

Além disso, você deve empregar multiplexadores, portas lógicas, somadores, comparadores, dentre outros elementos básicos para realizar a integração de modo a poder executar as instruções especificadas. A figura a seguir esboça um diagrama de blocos para o MIPS

monociclo **(OBS: não todos os detalhes estão inclusos na figura. O estudante deve ser capaz de identificar e adicionar novos elementos básicos quando necessário. Ex: MUXes, sinais de controle, novas entradas nos MUXes que já existem, etc**).



Contador de Programa (PC)

Trata-se de um registrador que armazena o endereço de instrução atual. A saída do PC fornece o endereço (entrada) para a memória de instrução (I_mem). Este módulo deve ser síncrono (apesar de se tratar de um monociclo, a cada novo ciclo, um novo PC deve ser atualizado, significando que uma nova instrução deve ser lida da memória). Dessa forma, o PC deverá ter os seguintes sinais:

Nome	Descriçao	Tamanho (bits)
clock	Atualiza o PC na borda de subida	1 bits
nextPC (entrada)	Valor a ser escrito no PC	32 bits
PC (saída)	Valor armazenado no PC (usado para endereçar a memória).	32 bits

Memória de instrução (i_mem)

A memória de instrução é responsável por armazenar o programa a ser executado, em ordem linear (sequencial). Implemente a memória de instrução como uma memória ROM (read-only memory) assíncrona com tamanho parametrizável. Os dados (instruções) a serem armazenados na memória devem ser fornecidos através de um arquivo de texto externo denominado “instruction.list”, em que as instruções são dispostas em código binário, separadas por quebra de linha. A memória deve apresentar os seguintes sinais:

Nome	**Descriçao**	**Tamanho (bits)**
address (entrada)	Endereço a ser lido. Valor fornecido pelo PC	32 bits
i_out (saída)	Dado armazenado (instrução) na posição especificada pelo address	32 bits

Banco de registradores (regfile)

O regfile deve conter 32 registradores de 32 bits cada, numerados de 0 a 31. O registrador $0 deve sempre constante e igual a 0x00000000. Isto significa que não é possível escrever outro valor diferente de 0 em $0. Os demais registradores devem ser capazes de armazenar valores de 32 bits. O regfile deve ser capaz de realizar leituras em dois registradores simultaneamente de forma assíncrona (sem clock) e deve ser capaz de escrever em um registrador de forma síncrona (com clock). O módulo deve apresentar os seguintes sinais de entrada/saída:

Nome	**Descriçao**	**Tamanho (bits)**
ReadAddr1 (entrada)	Número (endereço) de um dos registradores a ser lido (Read data 1). Valor fornecido pela instrução.	5 bits
ReadAddr2 (entrada)	Número (endereço) do outro registrador a ser lido (Read data 2).	5 bits

Nome	Descriçao	Tamanho (bits)
	Valor fornecido pela instrução	
ReadData1 (saída)	Dado lido assincronamente na posição especificada por ReadAddr1	32 bits
ReadData2 (saída)	Dado lido assincronamente na posição especificada por ReadAddr2	32 bits
Clock (entrada)	Sinal de clock (vai sincronizar as escritas). Sinal externo (fornecido pela interface do top-level).	1 bit
WriteAddr (entrada)	Número (endereço) do registrador a ser escrito. Valor fornecido pela instrução.	5 bits
WriteData (entrada)	Dado a ser escrito no registrador especificado por WriteAddr. Valor fornecido pela ULA ou memória de dados.	32 bits
RegWrite (entrada)	Sinal de controle que habilita a escrita nos registradores. Caso RegWrite = 1, o dado especificado por WriteData deve ser escrito no registrador especificado por WriteAddr na borda positiva do clock. Caso RegWrite = 0, nada é escrito. Valor fornecido pela unidade de controle.	1 bit
Reset (entrada)	Reseta o regfile, isto é, sobrescreve todos os 32 registradores com 0x00000000. Sinal externo (fornecido pela interface do top-level).	1 bit

Unidade lógica e aritmética (ula)

Deve ser capaz de realizar as operações aritméticas e lógicas necessárias para executar as instruções especificadas. Deve haver os seguintes sinais.

Nome	Descriçao	Tamanho (bits)
In1 (entrada)	Operando 1	32 bits
In2 (entrada)	Operando 2	32 bits
OP (entrada)	Operação a ser realizada. Fornecido pelo módulo ula_ctrl. Os valores de cada operação devem ser especificados pelos projetistas e está de acordo com a implementação da ula_ctrl.	4 bits
result (saída)	Resultado da operação	32 bits
Zero_flag (saída)	Flag que indica que result == 0	1 bit

Memória de dados (d_mem)

A memória de dados deve ser implementada na forma de uma memória RAM assíncrona, em que dados podem ser lidos ou escritos. Implemente a D-MEM com tamanho parametrizável. Os sinais de entrada e saída devem ser:

Nome	Descriçao	Tamanho (bits)
Address (entrada)	Endereço de acesso (leitura ou escrita). Deve ser fornecido pela ULA.	32 bits
WriteData (entrada)	Dado a ser escrito na memória na posição especificada por Address. Deve ser fornecido pelo regfile.	32 bits
ReadData (Saída)	Dado lido na memória na posição especificada por Address	32 bits
MemWrite (entrada)	Sinal de controle. Quando TRUE, indica que a memória deve escrever o WriteData na posição especificada por Address. Quando FALSE, nada deve ser escrito na memória. Sinal deve ser fornecido pela unidade de controle	1 bits

Nome	Descriçao	Tamanho (bits)
MemRead (entrada)	Sinal de controle. Quando TRUE, indica que a memória deve ler o conteúdo da memória na posição especificada por Address e colocar o resultado na saída ReadData . Quando FALSE, ReadData deve ficar em alta impedância. Sinal deve ser fornecido pela unidade de controle	1 bit

Unidade de controle (control)

Unidade responsável por gerar todos os sinais de controle para a execução de todas as instruções especificadas. Como entrada, deve receber o OPCODE da instrução (fornecido pela memória de instrução). A seguir, um exemplo dos sinais da unidade de controle. ***OBS: sua implementação pode precisar de MAIS SINAIS DE CONTROLE. Os projetistas devem ser capazes de identificar quando for necessário adicionar sinais de controle à unidade de controle.***

Unidade de controle da ULA

Responsável por informar a ULA a operação aritmética que deve ser realizada. Como entrada, recebe o sinal ALUOp da unidade de controle e também os 6 bits menos significativos da instrução (func). A saída deve ser um código que informa a ULA qual operação será realizada. Tais códigos devem ser especificados pelo projetista e devem estar em conformidade com a implementação da ULA.

Núcleo MIPS (Top-level)

O núcleo MIPS consiste em um módulo que incorpora, de maneira adequada, todos os módulos descritos anteriormente. **Este módulo deve ser feito em verilog**, assim como os demais. A interface do módulo toplevel deve consistir de uma entrada de clock e outra de reset (para resetar regfile). Como saída, deve apresentar o valor atual do PC, o valor atual da saída da ULA e o valor atual da saída de memória de dados.

Teste com código gerado pelo MARS

Em anexo, segue um código em assembly, bem como o binário gerado por esse código. Esse binário deve ser compilado dentro da memória ROM (memória de programa) do seu processador MIPS e deve ser possível ver as instruções do código assembly serem executadas corretamente (Observando o PC, ALU out, Mem Out, etc).