Relatório do Processador Subconjunto MIPS Monociclo

Juliana Zambon - GRR20224168
Projetos Digitais e Microprocessadores - CI1210
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
julianazambon@ufpr.br

***Resumo*—Este relatório apresenta a implementação de um processador subconjunto MIPS Monociclo, realizado como parte de um projeto no curso de Ciência da Computação. O objetivo foi desenvolver um processador simplificado capaz de executar instruções em um único ciclo. O relatório aborda os aspectos fundamentais do projeto, incluindo a organização do processador, a codificação e execução das instruções MIPS, além da análise de desempenho. A implementação do processador subconjunto MIPS Monociclo proporcionou uma compreensão aprofundada dos princípios de arquitetura de computadores e microprocessadores, contribuindo para o desenvolvimento de habilidades práticas e conhecimento teórico na área.**

***Index Terms*—Monociclo, processador, instruções, circuitos, programa teste.**

I. Introdução

O presente relatório é o resultado de um trabalho realizado no curso de Ciência da Computação da Universidade Federal do Paraná (UFPR), na disciplina de Projetos Digitais e Microprocessadores (CI1210), no segundo período. O objetivo principal deste relatório é apresentar a abordagem utilizada para a implementação de um processador subconjunto MIPS Monociclo.

O relatório apresentará a organização do processador, a codificação e execução das instruções MIPS, o registro de memórias relevantes para a execução, além da apresentação de um programa de teste escrito em linguagem *assembly* e convertido para o formato binário.

Com esse relatório, busca-se fornecer um detalhamento do desenvolvimento do trabalho, bem como oferecer uma visão abrangente do funcionamento e desempenho do processador subconjunto MIPS Monociclo implementado.

II. Implementação de Subcircuitos e Caminho de Dados

Para a implementação do MIPS Monociclo, são necessários a criação e integração de diversos subcircuitos essenciais. Entre eles, destacam-se o Bloco de Registradores, responsável pelo armazenamento dos dados e pela execução das operações de leitura e escrita nos registradores. Além disso, a Unidade Lógica Aritmética (ULA) é indispensável, pois realiza operações aritméticas e lógicas binárias, como adição, subtração, *and* e *or*. A correta implementação desses subcircuitos é fundamental para o funcionamento adequado do processador MIPS Monociclo.

A. Bloco de Registradores

Na arquitetura MIPS, existem 32 registradores, cada um com capacidade para armazenar um número binário de 32 bits. O Registrador de Instrução é um registrador de 32 bits que armazena uma cópia da última instrução acessada.

É importante que o sinal de escrita (*write enable*) seja ativado para permitir a escrita dos dados no registrador. Quando o sinal de escrita está ativo e o sinal *write data* está conectado à entrada de dados correta, o valor do *write data* será armazenado no registrador correspondente na próxima borda de subida do sinal de *clock*.

Assim, o bloco de registradores será composto por um demultiplexador de 5 bits, em que a entrada será o sinal de *write enable* (habilitação de escrita) e o seletor será o sinal de *write register* (dados a serem escritos). Esse demultiplexador permitirá selecionar o registrador específico no qual os dados serão escritos com base no valor do seletor. Dessa forma, o bloco de registradores garantirá o armazenamento adequado dos dados fornecidos para escrita nos registradores selecionados.

Além disso, será composto por dois multiplexadores, também de 5 bits, adicionais que fornecerão os dados lidos (*read data 1* e *read data 2*). Esses multiplexadores serão responsáveis por selecionar os dados corretos dos registradores com base nos endereços de leitura fornecidos. Assim, eles garantirão que os dados corretos sejam transmitidos para as etapas subsequentes do processador. Os multiplexadores serão controlados por sinais de seleção adequados para direcionar os dados corretos dos registradores para as saídas correspondentes.

O *clock* será conectado para sincronizar as operações dos registradores. E, cada registrador terá uma entrada de dados (D) que permitirá a escrita de novos valores no registrador. Essa entrada receberá os dados a serem armazenados no registrador durante a operação de escrita.

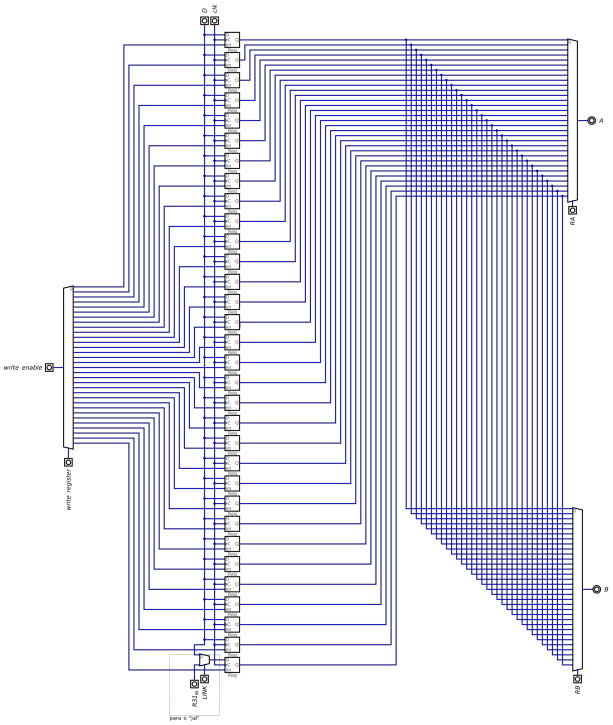

Figura 1. Bloco de registradores em um MIPS monociclo.

O registrador 31 é usado para armazenar o endereço de retorno de uma instrução de salto ou chamada de sub-rotina. Quando uma instrução de jump é executada, o endereço de destino é armazenado no registrador 31 para que, posteriormente, o programa possa retornar à instrução seguinte à chamada de salto.

B. Unidade Lógica Aritmética

As principais operações a serem implementadas na ULA são:

-Adição (*add*): Realiza a adição de dois valores.

-Subtração (*sub*): Realiza a subtração de dois valores.

-AND lógico (*and*): Realiza a operação lógica *and* bit a bit entre dois valores.

-OR lógico (*or*): Realiza a operação lógica *or* bit a bit entre dois valores.

-Comparação de igualdade (*equal*): Compara se dois valores são iguais.

-Comparação de desigualdade (*not equal*): Compara se dois valores são diferentes.

-Comparação menor que (*less than*): Compara se o primeiro valor é menor que o segundo valor.

-Deslocamento à esquerda (*shift left*): Realiza um deslocamento de bits à esquerda em um valor.

-Deslocamento à direita (*shift right*): Realiza um deslocamento de bits à direita em um valor.

Dependendo dos requisitos, outras operações podem ser adicionadas ou algumas dessas operações podem ser modificadas ou combinadas para atender às necessidades específicas.

Para realizar a operação "beq"(*Branch On Equal*) no MIPS, é necessário subtrair os valores dos dois registradores que estão sendo comparados e verificar se o resultado é igual a zero.

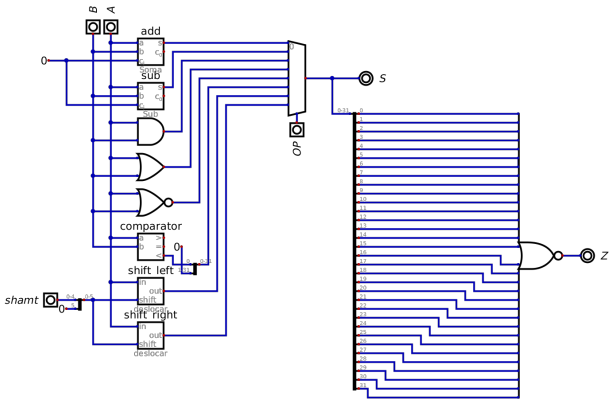

Figura 2. Unidade Lógica Artimética.

É importante destacar o uso de extensores de bits para garantir que a segunda entrada dos *shifters* receba apenas 6 bits. Esses extensores são responsáveis por replicar o bit de sinal do valor a ser deslocado, a fim de manter a consistência do resultado durante o deslocamento. Dessa forma, ao aplicar o deslocamento à esquerda ou à direita em um valor, somente os 6 bits necessários são considerados.

Para garantir que a saída do comparador tenha 32 bits, onde os bits de 1 a 31 sejam definidos como zero, é necessário utilizar um distribuidor de bits. O distribuidor de bits irá propagar o valor constante zero nos bits de 1 a 31 da entrada do comparador, enquanto mantém o bit de 0 como saída inalterado. Dessa forma, a saída do comparador terá 32 bits, onde os bits de 1 a 31 serão definidos como zero e o bit de 0 refletirá o resultado da comparação.

Foi implementado também uma operação *nor* com 32 entradas, conhecida como *nor* generalizado. Essa operação *nor* generalizado realiza a operação lógica *nor* entre 32 bits de entrada.

III. MIPS Monociclo

O MIPS monociclo é um tipo de processador que executa instruções em um único ciclo de *clock*. Ele possui uma arquitetura simplificada, onde cada instrução é buscada, decodificada, executada e armazenada na memória em apenas um ciclo de *clock*. Nesse tipo de processador, todas as instruções têm a mesma duração de execução, independentemente de sua complexidade. Embora seja uma abordagem simples, o MIPS monociclo é limitado em termos de desempenho, já que apenas uma instrução é executada por ciclo de *clock*, mesmo que algumas instruções não exijam todos os recursos disponíveis.

O *Program Counter* (PC) armazena o endereço da próxima instrução a ser buscada na memória de instruções. O PC é incrementado a cada ciclo de *clock* para apontar para a próxima instrução. A unidade de incremento é responsável por adicionar 4 ao valor atual do PC, uma vez que as instruções no

MIPS são armazenadas em palavras de 32 bits e cada palavra possui 4 *bytes*.

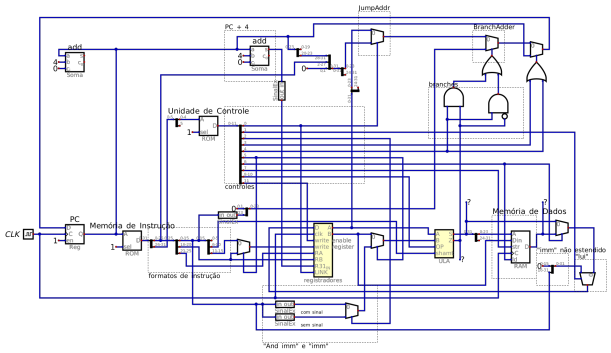

Figura 3. Implementação de um Circuito Monociclo MIPS.

Para implementar a operação and imediato (*andi*) foi utilizado dois extensores: um com sinal (*sign-extend*) e outro sem sinal (*zero-extend*). Ambos os valores estendidos, com sinal e sem sinal, são conectados às entradas do multiplexador.

Para a instrução *Load Upper Immediate* (*lui*), onde um valor imediato de 16 bits é estendido para 32 bits e carregado em um registrador, será necessário utilizar um distribuidor. O distribuidor terá duas entradas de 16 bits cada, e uma saída de 32 bits. Na entrada de 0 a 15, conecta uma constante zero de 16 bits. Na outra entrada,conecta o valor imediato (*imm*) de 16 bits (não estendido). A saída do distribuidor será conectada à entrada 1 de um multiplexador, enquanto a saída do mux que está conectada à memória de dados será conectada à entrada 0 do mux. Dessa forma, quando a instrução lui for executada, o *imm* será estendido para 32 bits pelo distribuidor, e esse valor estendido será selecionado como a entrada do mux. Caso contrário, quando uma instrução que não seja *lui* for executada, a saída do mux conectada à memória de dados será selecionada como a entrada do mux. Essa configuração permitirá que o *imm* seja estendido adequadamente para 32 bits antes de ser carregado no registrador durante a instrução lui.

Para o *Jump Address*, utiliza os 26 bits menos significativos da instrução (*ISA*) e concatena com outras informações, incluindo o PC+4. Após a execução da instrução de salto (j), obtém os 26 bits menos significativos da instrução, que representam o endereço de salto. Em seguida, concatena esses 26 bits com os 4 bits mais significativos do PC+4, que é o valor atual do PC somado com 4. A concatenação dos bits do endereço de salto com os bits do PC+4 resultará em um endereço de salto completo de 30 bits. Esse endereço de salto será utilizado para direcionar o PC para a próxima instrução a ser executada no programa. O endereço de salto é relativo ao endereço atual da instrução, por isso é necessário somar o PC+4 para calcular o endereço absoluto de salto.

Para o *Branch Address*, o *imm* da instrução de *branch* (*beq* ou *bne*) será estendido para 30 bits utilizando um extensor de sinal. Esse extensor receberá uma entrada de 16 bits (o *imm*) e terá uma saída de 30 bits. A saída do extensor de sinal será conectada a um distribuidor de bits, que possui duas entradas (uma de 2 bits e outra de 30 bits) e uma saída de 32 bits. A saída do distribuidor de bits, que agora terá 32 bits, será conectada à entrada do mux correspondente ao *Branch Address*. Assim, o mux selecionará o valor correto (endereço de desvio) com base na condição de desvio e fornecerá esse valor ao PC, direcionando-o para a próxima instrução a ser executada.

IV. Memória de Instruções e Dados: Organização e Implementação

A memória de instruções em um MIPS monociclo é onde as instruções do programa são armazenadas. Essas instruções são buscadas sequencialmente e carregadas no processador para execução. A memória de instruções é uma memória de acesso aleatório somente de leitura (ROM - *Read-Only Memory*). A utilização de uma memória ROM garante que as instruções sejam armazenadas de forma permanente e não possam ser alteradas durante a execução do programa.

A memória de dados, porém, é onde os dados utilizados pelo programa são armazenados. Essa memória é usada para armazenar valores intermediários, variáveis, constantes e qualquer outro dado necessário durante a execução do programa. A memória de dados também é uma RAM que pode ser lida e escrita pelo processador.

Em um MIPS monociclo, as memórias de instruções e dados são distintas e possuem endereçamento separado. Isso permite que o processador busque as instruções na memória de instruções e acesse os dados na memória de dados conforme necessário durante a execução do programa.

A implementação da Memória de Instruções deu-se por meio de uma memória ROM, assim, a saída do PC foi utilizada como entrada para a memória ROM, para que retornasse a instrução correspondente ao endereço apontado pelo PC.

Enquanto a implementação da Memória de Dados, ocorreu por meio de uma memória RAM que é volátil, o que significa que seu conteúdo é perdido quando a energia é desligada.

V. Unidade de Controle: Funcionamento e Implementação

A unidade de controle em um MIPS monociclo é responsável por coordenar e controlar as operações do processador. Ela é responsável por gerar os sinais de controle necessários para executar corretamente as instruções do programa.

Assim, ela determina qual operação deve ser realizada, como a busca de instruções, leitura e escrita em registradores, acesso à memória, operações aritméticas e lógicas, desvios e interrupções. Ela utiliza os sinais de controle para ativar os componentes apropriados do processador em cada ciclo de *clock*, garantindo a execução adequada do programa.

Para sua implementação, foi utilizada uma memória ROM, ela contém as instruções de controle que são responsáveis por gerar os sinais de controle adequados para cada instrução executada pelo processador.

Para a elaboração da tabela de controle, a fim de fornecer seu binário e correta conversão para hexadecimal, foram

listadas as entradas relevantes para cada instrução suportada pelo processador. Cada coluna é representada por um número adequado de bits, de modo a acomodar todos os valores possíveis. Assim, a tabela de controle fornece os sinais de controle necessários para cada instrução.

A tabela de controle contém os seguintes campos: *Write Enable* (WE), que indica se um registrador deve ser atualizado; ULA, que especifica o código de controle para a unidade lógica e aritmética; STR e LD, que indicam se a instrução é de armazenamento ou carregamento de dados; IMM, que indica se a instrução possui um operando imediato; *Branch* e *Jump*, que identificam se a instrução é um desvio condicional ou um salto incondicional. Outros sinais como Sinal, LUI e JAL têm funções específicas em instruções selecionadas.

Tabela I
TABELA DE CONTROLE DE INSTRUÇÕES: SINAIS DE CONTROLE E OPERAÇÕES CORRESPONDENTES

OP	we	ula	str	ld	im	b	j	s	lui	jal
00000	1	000	0	0	0	0	0	0	0	0
00001	1	000	0	0	1	0	0	0	0	0
00010	1	010	0	0	0	0	0	0	0	0
00011	1	010	0	0	1	0	0	1	0	0
00100	0	001	0	0	0	1	0	0	0	0
00101	0	001	0	0	0	1	0	0	0	0
00110	0	000	0	0	0	0	1	0	0	0
00111	1	000	0	0	0	0	1	0	0	1
01000	0	000	0	0	0	0	1	0	0	0
01001	1	000	0	1	0	0	0	0	1	0
01010	1	000	0	1	1	0	0	0	0	0
01011	1	100	0	0	0	0	0	0	0	0
01100	1	011	0	0	0	0	0	0	0	0
01101	1	011	0	0	1	0	0	1	0	0
01110	1	101	0	0	0	0	0	0	0	0
01111	1	101	0	0	1	0	0	0	0	0
10001	1	110	0	0	0	0	0	0	0	0
10010	1	111	0	0	0	0	0	0	0	0
10011	0	000	1	0	1	0	0	0	0	0
10100	1	001	0	0	0	0	0	0	0	0

Assim, a tabela de controle de instruções foi desenvolvida em conformidade com o conjunto de instruções descrito no *MIPS Green Card*, seguindo as especificações detalhadas neste relatório. Para a alocação adequada dos bits na unidade de controle para cada instrução específica, foi empregado um distribuidor de bits, responsável por direcionar e distribuir corretamente os sinais de controle para cada uma das instruções.

VI. DESENVOLVIMENTO DO PROGRAMA DE TESTE

No processo de desenvolvimento do programa de teste, foram implementadas e destacadas as seguintes instruções, conforme descritas no *MIPS Green Card*:

Tabela II
CONJUNTO DE INSTRUÇÕES IMPLEMENTADAS

Nome	Mnemônico
Add	add
Add Immediate	addi
And	and
And Immediate	andi
Branch On Equal	beq
Branch On Not Equal	bne
Jump	j
Jump And Link	jal
Jump Register	jr
Load Upper Immediate	lui
Load Word	lw
Nor	nor
Or	or
Or Immediate	ori
Set Less Than	slt
Set Less Than Immediate	slti
Shift Left Logical	sll
Shift Right Logical	srl
Store Word	sw
Subtract	sub

Abaixo estão as instruções do programa e o resultado esperado para cada uma delas:

Tabela III
INSTRUÇÕES EM ASSEMBLY

Instrução	Resultado
addi R1, R0, 5	R1 = 5
addi R0, R1, 2	R0 = 7
add R3, R1, R0	R3 = R1 + R0 = 12
and R4, R2, R1	R4 = R2 AND R1
andi R1, R4, 3	R1 = R4 AND 3
bne R1, R4, 9	Desvia para o rótulo ”9”se R1 for diferente de R4
add R1, R1, R0	R1 = R1 + R0 = 7
addi R31, R4, 5	R31 = R4 + 5
jr R31	Salto para o endereço contido em R31
nor R5, R6, R7	R5 = NOT (R6 OR R7)
or R6, R7, R5	R6 = R7 OR R5
ori R6, R5, 2	R6 = R5 OR 2
slt R8, R5, R6	R8 = 1 se R5 <R6, senão R8 = 0
slti R9, R6, 10	R9 = 1 se R6 <10, senão R9 = 0
sub R7, R7, R8	R7 = R7 - R8
sw R8, 10(R6)	Armazena o valor de R8 no endereço de memória 10 + valor de R6
lw R10, 10(R9)	Carrega o valor do endereço de memória 10 + valor de R9 para R10
lui R9, 10	R9 = 10 <<16 (preenchendo os bits mais significativos)
sll R7, R5, 3	R7 = R5 <<3 (deslocamento à esquerda)
srl R11, R31, 2	R11 = R31 >>2 (deslocamento à direita)
jal 23	Salto para o rótulo ”23”, armazenando o endereço de retorno em R31
addi R0, R0, 0	Instrução sem efeito prático
add R14, R15, R16	R14 = R15 + R16
addi R13, R11, 4	R13 = R11 + 4
add R15, R13, R14	R15 = R13 + R14
sub R3, R3, R15	R3 = R3 - R15
beq R1, R3, 29	Desvia para o rótulo ”29”se R1 for igual a R3
addi R6, R6, 5	R6 = R6 + 5
j 26	Salto para o rótulo ”26”

Desse modo, os bits foram concatenados para formar a representação binária completa, e em seguida, foi realizada a conversão dessa representação para o formato hexadecimal.

Tabela IV
TABELA DE INSTRUÇÕES

Conversão	
Binário	**Hexadecimal**
00000100000000010000000000000101	4010
00000100001000000000000000000010	4200
00000000001000000011000000000000	201
00001000001000100010000000000000	8222
00001100100000010000000000000011	c810
00010100100000010000000000001001	14810
00000000001000000000100000000000	200
00000100100111110000000000000101	49f0
00100000000111110000000000000000	201f0
00101100110001110010100000000000	b31c
00110000111001010011000000000000	c394
00110100111001010000000000000010	34e50
00111000101001100100000000000000	38a64
00111100110010010000000000001010	3cc90
01001100111010000011100000000000	4ce80
01001000110010100000000000001010	48ca0
00101001001010100000000000001010	292a0
00100100000010010000000000001010	24090
01000000101000110011100000000000	40a30
01000100101100010111110000000000	44b14
00011100000000000000000000010111	1c000
00000100000000000000000000000000	4000
00000001111100000111000000000000	1f07
00000101101010110000000000000100	5ab0
00000001101011100111100000000000	1ae7
00010000011000010000000000000001	10610
00000100001000010000000000000101	4210
00011000000000000000000000000000	18000

VII. CONCLUSÃO

O processo de implementação permitiu uma compreensão aprofundada dos princípios da arquitetura de computadores e microprocessadores.

O processador subconjunto MIPS Monociclo foi projetado para executar instruções em um único ciclo, o que resultou em um design simplificado, porém funcional. Durante os testes, o processador demonstrou ser capaz de executar com sucesso tarefas básicas e programas de teste escritos em linguagem *assembly*.

REFERÊNCIAS

Relatório elaborado em LaTeX utilizando a plataforma *Overleaf* .

PATTERSON, David A.; HENNESSY, John L. Computer Organization and Design: The Hardware/Software Interface. 4th Edition. Burlington: Morgan Kaufmann Publishers, 2008.

CENTODUCATTE, Paulo C. Conjunto de Instruções MIPS.

LEIRIA, Ana Isabel; MOURA, Margarida Moreira; ROSADO, António. Arquitectura de Computadores.