Universidade de São Paulo

Escola Politécnica

Processador MIPS

Grupo 4:

Carlos Eduardo de Castro Gonzales Rodrigues - 11805443

Paulo Jose Cardoso Racy Ferreira - 11916666

Thaylor Hugo Araujo da Silva - 13684425

Thiago Almeida dos santos - 12726670

Disciplina: PCS3115 - Sistemas Digitais I

Projeto: https://github.com/Thaylor-Hugo/MIPS-PCS.git

São Paulo - SP

Novembro, 2022

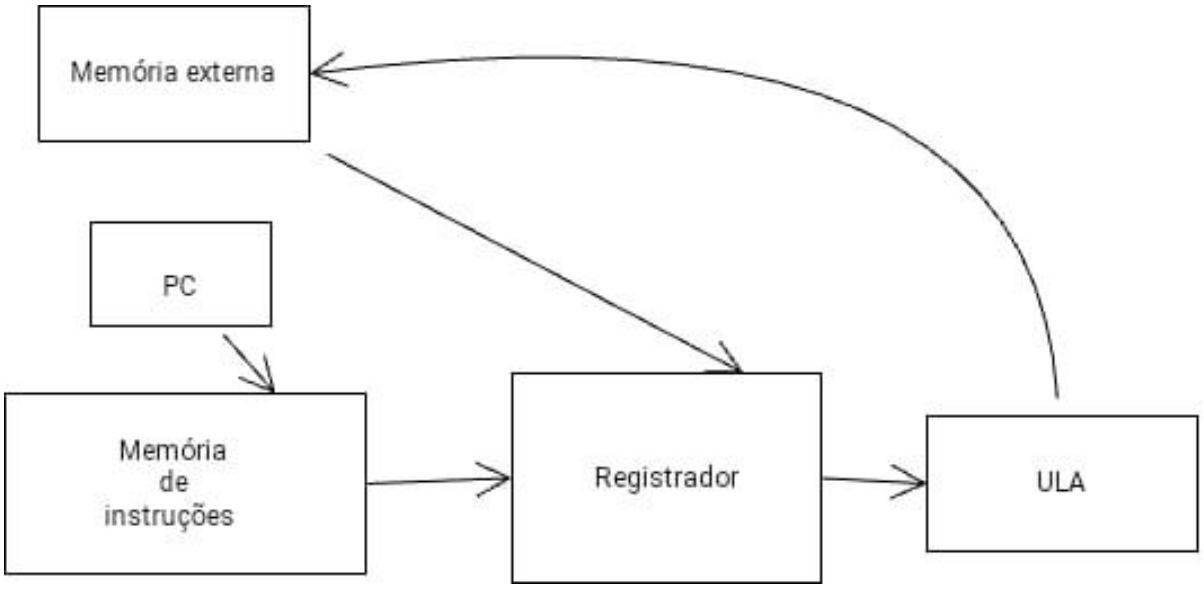
Proposta

Processador pequeno não especializado para aplicações diversas de baixa necessidade de processamento, como um substituto para sistemas mais complexos em processos que não requerem muito poder computacional.

É proposto a implementação do processador MIPS monociclo de 16 bits, inicialmente com 13 instruções das seguintes categorias:

a) Aritméticas:
    i) Adição
    ii) Subtração
    iii) Adição imediata
b) Lógicas:
    i) And
    ii) Or
    iii) Menor que
    iv) Menor que imediato
c) Desvio condicional:
    i) Branch on equal
d) Salto condicional:
    i) Jump
    ii) Jump register
    iii) Jump and link
e) Transferência de dados:
    i) Carregar palavra
    ii) Armazenar palavra

Diagrama de Blocos Abstrato



De modo simplificado, a entrada do circuito será um programa computacional. Pelas informações do programa, o circuito definirá o valor do contador de programa (PC) com o endereço da instrução desejada.

O endereço armazenado no PC é acessado na memória de instruções que, por sua vez, inicia a instrução desejada, definindo os registradores e realizando os devidos cálculos na unidade lógica aritmética (ULA).

Os dados do resultado da instrução são armazenados na memória que, eventualmente, podem retornar aos registradores para serem utilizados por novas instruções.

Ao fim do ciclo de instrução, o PC é alterado para dar continuidade às tarefas até que todas as instruções do programa sejam executadas.

Instruções em Nível de Registradores

As instruções seguirão o seguinte formato:

Nome	**Campos**					**Descrição**
Tamanho campo	3 bits	3 bits	3 bits	3 bits	4 bits	16 bits total para cada instrução
Formato - R	opcode	rs	rt	rd	função	Instrução aritmética
Formato - I	opcode	rs	rt	Endereço\Constante		Instrução de transferência, ramificação, imediata
Formato - J	opcode	Endereço alvo				Instrução de pulo

Legenda:

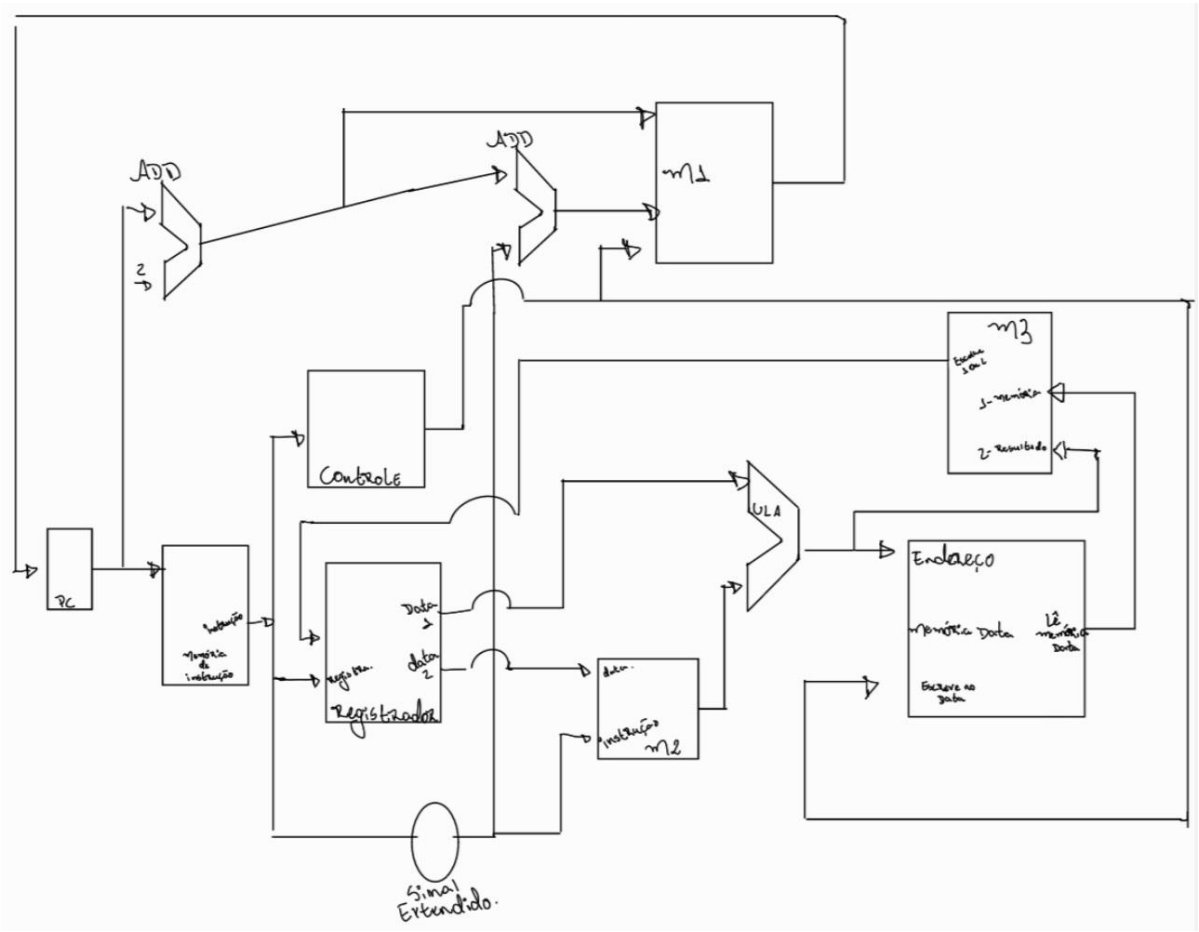
- opcode: referência da instrução
- rd: registrador de destino do resultado
- rs: registrador fonte, apenas de leitura
- rt: registrador fonte ou de destino, depende da referência da instrução

As seguintes instruções serão implementadas:

Formato	**Nome**	**Sintaxe**	**Descrição**
R	Add	add $1, $2, $3	$1 = $2 + $3
R	Subtract	sub $1, $2, $3	$1 = $2 - $3
I	Add immediate	addi $1, $2, CONST	$1 = $2 + CONST
R	And	and $1, $2, $3	$1 = $2 & $3
R	Or	or $1, $2, $3	$1 = $2 | $3
R	Set on less than	slt $1, $2, $3	$1 = ($2 < $3)
I	Set on less than immediate	slti $1, $2, CONST	$1 = ($2 < CONST)
R	Branch on equal	beq $1, $2, CONST	if ($1 == $2) PC = PC + 2 + CONST
J	Jump	j CONST	PC = CONST

R	Jump register	jr $1	PC = $1
J	Jump and link	jal CONST	$7 = PC + 2; PC = CONST
I	Load word	lw $1, CONST ($2)	$1 = Memory[$2 + CONST]
I	Store word	sw $1, CONST ($2)	Memory[$2 + CONST] = $1

Diagrama de Blocos Detalhado



Para evitar uma imagem muito poluída, algumas das linhas de controle foram ocultadas.

O endereço de memória armazenado no PC ativa uma determinada instrução na memória de instrução. Diferentes tipos de instruções guiam os dados para diferentes áreas do circuito. A instrução define os registradores que serão utilizados.

Instruções aritméticas definem três registradores, dois para os dados que serão utilizados na ULA e um para armazenar o resultado da operação. Instruções que utilizam constantes definem apenas um registrador de operando.

A operação de Load recupera dados da memória e os guarda em um registrador, enquanto sua operação inversa Store armazena os dados do registrador na memória.

Operações do tipo Jump ou de ramificação alteram o valor do PC e definem o endereço da próxima instrução. Nos demais tipos de instruções, o valor do PC é incrementado de modo que este adquira o endereço da instrução seguinte.

A ULA é responsável não somente pelo cálculo das instruções aritméticas, mas também por calcular os endereços que serão utilizados nas operações de Load, Store ou Jump.

Os diferentes multiplexadores controlam quais das entradas devem ser usadas, a depender da instrução que está sendo executada. O módulo de Controle regula os caminhos que os dados seguem, de modo a realizar o caminho esperado pela instrução.

Instruções e Códigos de Controle

Como já citado anteriormente, as instruções são de 16 bits compostas por alguns campos. Seu padrão segue o seguinte formato:

Nome	**Formato**	**opcode**	**rs**	**rt**	**rd**	**funcao**	**Comentário**
add	R	000	xxx	xxx	xxx	0000	add $1, $2, $3
sub	R	000	xxx	xxx	xxx	0001	sub $1, $2, $3
and	R	000	xxx	xxx	xxx	0010	and $1, $2, $3
or	R	000	xxx	xxx	xxx	0011	or $1, $2, $3
slt	R	000	xxx	xxx	xxx	0100	slt $1, $2, $3
xor	R	000	xxx	xxx	xxx	0101	xor $1, $2, $3
sll	R	000	xxx	xxx	xxx	0110	sll $1, $2, $3
srl	R	000	xxx	xxx	xxx	0111	srl $1, $2, $3
jr	R	000	111	000	000	1000	jr $7
addi	I	001	xxx	xxx	CONST		addi $1, $2, CONST
slti	I	010	xxx	xxx	CONST		slti $1, $2, CONST
lw	I	011	xxx	xxx	CONST		lw $1, CONST ($2)
sw	I	100	xxx	xxx	CONST		sw $1, CONST ($2)
beq	I	101	xxx	xxx	CONST		beq $1, $2, CONST
j	J	110	CONST				j CONST

jal	J	111	CONST	jal CONST

Os valores do opcode e da função são arbitrários, entretanto precisam seguir o mesmo padrão em todos os módulos que são necessários. Os valores em xxx (3 bits) representam os endereços dos registradores e variam em cada instrução. Já os valores CONST representam a constante que será estendida e usada, seu tamanho e valor variam a depender da instrução.

Os seguintes sinais de controle e sinais de controle da ULA foram utilizados. Seus valores também são arbitrários, mas devem ser utilizados no mesmo padrão em todos os módulos.

Sinais de controle										
Instrução	ula_opcode	reg_dest	ula_src	mem_to_reg	mem_escrita	mem_leitura	reg_escrita	branch	jump	sign_zero
Tipo R	0	0	0	0	0	0	1	0	0	1
addi	1	1	1	0	0	0	1	0	0	1
slti	10	1	1	0	0	0	1	0	0	0
lw	1	1	1	1	0	1	1	0	0	1
sw	1	1	1	0	1	0	0	0	0	1
beq	11	1	0	0	0	0	0	1	0	1
j	0	0	0	0	0	0	0	0	1	1
jal	0	10	0	10	0	0	1	0	1	1

ULA CONTROL			
ULA opcode	**Função**	**ULA Operação**	**Instrução**
00	0000	adição	add
00	0001	subtração	sub
00	0010	and	and
00	0011	or	or
00	0100	menor que	slt
00	0101	xor	xor
00	0110	shift left logical	sll
00	0111	shift right logical	srl
01	zzzz	adição	addi, lw, sw
10	zzzz	menor que	slti
11	zzzz	subtração	beq

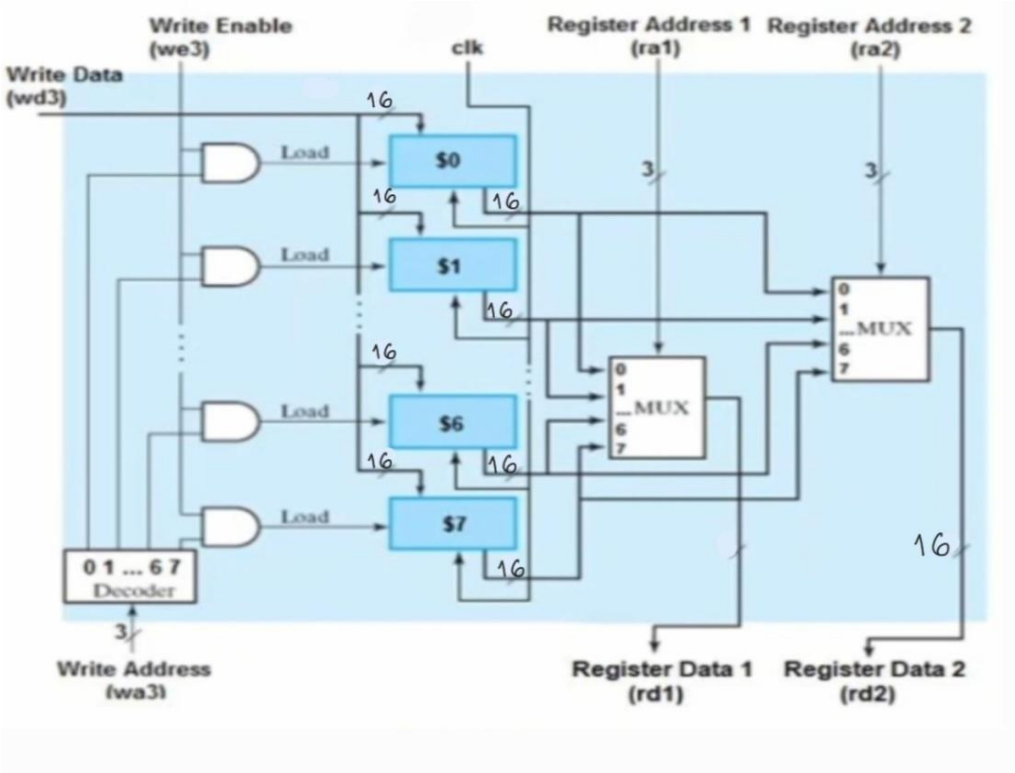
Nota: As instruções com ULA opcode diferente de zero não possuem “funcao” nos seus campos de instrução, logo os 4 bits equivalentes não são determinados ou utilizados.

Módulos do Processador

Com os códigos de controle e o fluxo de dados é possível desenvolver os módulos do processador, relacionando suas funções, entradas e saídas.

Banco de Registradores

Temos 8 registradores de 16 bits:



Como podemos ver, o circuito de escrita é sequencial (depende do clock), já o circuito de leitura é combinacional.

Entradas:

- Clock: Ativam o funcionamento de escrita do banco;
- Dado para escrita (wd3) (16 bits) - Entrada de dados;
- Endereço da escrita (wa3) (3 bits) - Seleção do registrador que armazena o dado de os dados de escrita;

- Permissão para escrita (we3) (1 bit) - Habilita ou desabilita a gravação de dados nos registradores;
- Endereço do primeiro registrador (ra1) (3 bits) - Seleção de qual registrador será disponibilizado na saída;
- Endereço do segundo registrador (ra2) (3 bits) - Seleção de qual registrador será disponibilizado na saída;

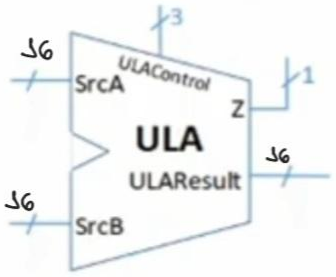
Saídas:

- Valor do primeiro registrador (rd1) (16 bits) - Dado de leitura do primeiro registrador;
- Valor do primeiro registrador (rd2) (16 bits) - Dado de leitura do segundo registrador;

O registrador $0 é somente de leitura e seu valor é fixo em 0. O registrador $7 o recomendado para a instrução jal, mas sua leitura e escrita é permitida.

ULA

A ULA é assíncrona (circuito combinacional) e possui 8 operações lógicas, são elas: adição, subtração, and, or, menor entre, xor, shift left lógico e shift right lógico.



Operação	ULAControl	ULAResult
And	000	= SrcA & SrcB
Or	001	= SrcA | SrcB
Add	010	= SrcA + SrcB
Subtract	110	= SrcA + $\overline{\text{SrcB}}$ + 1
Set less than (SLT)	111	1, se SrcA < SrcB 0, c.c.

Entradas:

- Dado 1 (SrcA) (16 bits): Entrada de dados do operando A (vem de um registrador);
- Dado 2 (SrcB) (16 bits): Entrada de dados do operando B (vem de um outro registrador ou de uma constante);
- Código de controle da ULA (ULAControl) (3 bits): Entrada de seleção da operação realizada;

Saídas:

- Resultado da ULA (ULAResult) (16 bits): Saída do resultado da operação;
- Alerta de zero (Z) (1 bit): Bit de status que indica se a saída da operação realizada é zero (resultado igual a zero: Z=1; resultado diferente de zero: Z=0);

Unidade de Controle da ULA

Um decodificador que determina o código de controle ULA, responsável por selecionar a operação da ULA desejada para a instrução.

Entradas:

- Código de operação da ULA (2 bits): Refere-se ao tipo da instrução;
- Função (4 bits): Parte do código das instruções do tipo R (aritméticas ou lógicas);

Saídas:

- Controle do JR (1 bit): Caso 1, informa que a instrução atual é do tipo jump register;
- Código de controle da ULA (3 bits): Seletor de operação da ULA.

Unidade de Controle

Este circuito é um decodificador combinacional. O circuito é responsável por determinar os valores da maioria das variáveis de controle que decidem as funções e operações realizadas, de modo a seguir a funcionalidade da instrução.

Entradas:

- Código de operação (3 bits) - Parte inicial do código de instrução, determina o tipo da instrução;

Saídas:

- Código de operação da ULA (2 bits): Usado no controle da ULA.

- Registrador de destino (2 bits): Determina se será usado a parte rd da instrução (se 0), a parte rt (se 1) ou se será o registrador $7 (se 10, especifico da instrução jal);
- Fonte da dado da ULA (1 bit): Determina se o segundo dado da ULA será o registrador de endereço rt (se 0) ou se será uma constante (se 1).
- Memória para registrador (2 bits): Determina se o dado que irá para o registrador vem da ULA (se 0), da memória (se 1) ou se do contador de programa (se 10);
- Permissão de escrita na memória (1 bit): Se 1, existe permissão;
- Permissão de leitura da memória (1 bit): Se 1, existe permissão;
- Permissão de escrita no registrador (1 bit): Se 1, existe permissão;
- Alerta de função branch (1 bit): Se 1, instrução é do tipo branch;
- Alerta de função jump (1 bit): Se 1, instrução é do tipo jump;
- Extensão do sinal ou com zero (1 bit): Informa se a constante da instrução deve ser estendida com o sinal (se 1), ou com zeros (se 0, para instrução slti);

Contador de Programa

Tal componente é um registrador de 16 bits que armazena o endereço da instrução que está sendo realizada. Este módulo é responsável por atualizar o valor do pc conforme a instrução.

Entradas:

- Alerta de Branch (1 bit, vale 1 caso a instrução seja do tipo branch e o resultado da ULA seja 0) e Constante estendida (16 bits): Neste caso, o pc, além de ser incrementado por dois, é também somado com a constante extendida shiftada para a esquerda uma vez;
- Alerta de Jump (1 bit) e Alvo do Jump (13 bits): Nesse caso, o valor do pc é alterado para a concatenação dos dois bits mais significativos do valor anterior, com o valor do alvo do jump e um bit 0;
- Alerta de JR (1 bit) e Dado do Registrador (16 bits, usualmente $7): Neste caso, o pc recebe o valor armazenado no registrador;

Saída:

- Endereço do PC (16 bits): Endereço da instrução que será realizada;

Memória de instrução

Esse circuito é combinacional, é uma memória ROM, essa memória será somente de leitura, o conteúdo da memória será definido pela sequência de instruções do programa a ser executado.

Entrada:

- Endereço da instrução (PC) (16 bits): Endereço da instrução que será executada;

Saída:

- Instrução (16 bits): Instrução contida no endereço fornecido;

Memória de Dados

Essa é a memória RAM, onde ocorre o armazenamento e recuperação de dados do programa executado, está relacionada diretamente com as instruções do tipo I Carregar Palavra (lw) e Armazenar Palavra (sw).

A memória segue o padrão de Byte Aligned, assim cada palavra ocupa dois espaços da memória (cada espaço armazena 1 byte = 8 bits), logo os endereços devem ser sempre números pares. Conjuntamente, a memória foi projetada com o padrão de armazenamento Little Endian Byte Order, assim, os bits menos significativos dos dados são armazenados nos endereços menores da memória.

Entradas:

- Permissão de escrita na memória (1 bit);
- Permissão de leitura da memória (1 bit);
- Endereço da memória (16 bits): Endereço usado para busca da memória, compartilhado pela leitura e escrita;
- Dado para escrita (16 bits);

Saídas:

- Dado da leitura (16 bits);

Processador MIPS

Módulo principal que reúne os demais módulos, todas as entradas e saídas. Possibilita o funcionamento e sincronização dos módulos com o intuito de realizar as instruções da memória de instruções. Recebe como entrada um sinal de clock que comanda o funcionamento dos circuitos sequenciais e um sinal de reset que reinicia os circuitos sequenciais. Como saída foi utilizado apenas o valor do contador de programa.

Simulação

Para testar o processador foi criado uma memória de instruções buscando realizar todas as instruções disponíveis. Com o simulador Icarus Verilog, foi realizada a simulação dos testbenches dos módulos do processador.

Para o processador completo foi realizada a análise da Waveform da simulação, que possibilitou a correção de alguns erros iniciais e no momento demonstra que o processador está realizando corretamente as instruções designadas.

Conclusão

Foi possível a realização de um processador MIPS de 16 bits que realiza uma função por ciclo. O processador é capaz de realizar suas instruções, podendo ser utilizado para aplicações simples de baixo processamento.

A alteração na memória de instruções é responsável por gerar diferentes funções para o processador. Apesar de ser um processador simples, sua utilização requer conhecimento técnico específico, pois o gerenciamento de memória é feito totalmente pelo programador e suas instruções devem ser fornecidas diretamente em linguagem de máquina.

Anexos

Os códigos dos módulos do processador e seus testbenches podem ser encontrados no repositório:

https://github.com/Thaylor-Hugo/MIPS-PCS.git