INF01113 – Organização de Computadores B – Trabalho Prático: MIPS

Grupo 1

Haroldo Rojas de Souza Silva <haroldorojas.silva@inf.ufrgs.br>
Ingrid Murielem <murielem.ingrid@gmail.com>
Vitória Lentz <vitoria.lentz@hotmail.com>
Wellington Espindula <wmespindula@inf.ufrgs.br>.

1. Modificações comuns

Em todas as três implementações, utilizou-se a mesma lógica para controle da ALU. Dessa forma, seguiu-se a tabela verdade abaixo e criou-se o seguinte circuito para sua realização, visualizável na figura 1.

Tabela 1. Tabela verdade do *ALU_Control*. Fonte: Os Autores (2021).

Op-Code	ALUOp	Op	Funct	Ação na ALU	ALUControl
lw	00	load word	XXXXXX	add	010
sw	00	store word	XXXXXX	add	010
beq	01	branch equal	XXXXXX	subtract	110
R	10	add	100000	add	010
		subtract	100010	subtract	110
		and	100100	and	000
		or	100101	or	001
		set-on-less-than	101010	set-on-less-than	111
		JALR	**001001**	**bypass**	**011**
		SLLV	**000100**	**shift-left-logical-variable**	**100**

A partir desta tabela verdade, chegou-se às seguintes fórmulas para os bits de *ALUControl* em função de *Funct*.

$$bit\ 0 \ = \ Funct[3]\ or\ Funct[0]$$
$$bit\ 1 \ = \ !\,Funct[2]$$
$$bit\ 2 \ = \ (!\,Funct[5]\ and\ !\,Funct[3])\ or\ Funct[1]$$

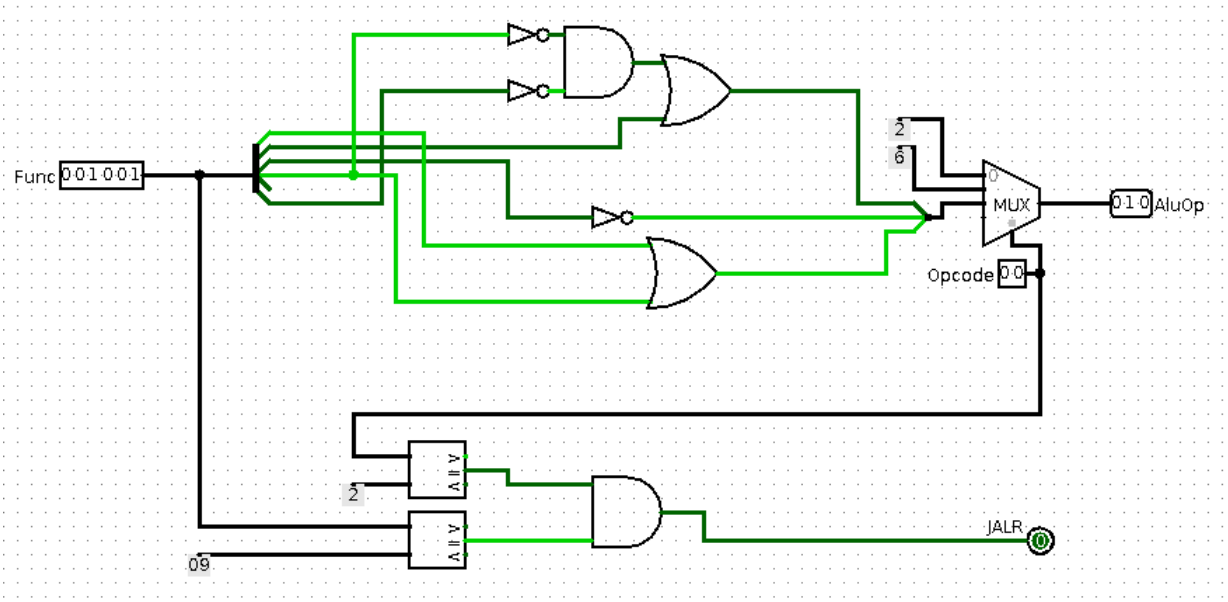

Figura 1. Esquemático do *ALU_Control*. Fonte: Os Autores (2021).

2. MIPS Monociclo

- **JALR (Jump And Link Register)**

Através da adição do sinal de JALR no *ALU_Control*, adicionou-se um multiplexador controlado por JALR a fim de selecionar se o dado de entrada para a banco de registradores iria ser da saída de dados ou se seria do PC+4. Ademais, esse sinal (JALR) também foi utilizado, a partir da criação de um novo *MUX*, para multiplexar se o conteúdo do PC iria ser da saída do multiplexador controlado por sinal de Jump (“Jump_MUX”) ou se seria da saída do resultado da *ALU*. Estas modificações podem ser vistas destacadas na figura 2.

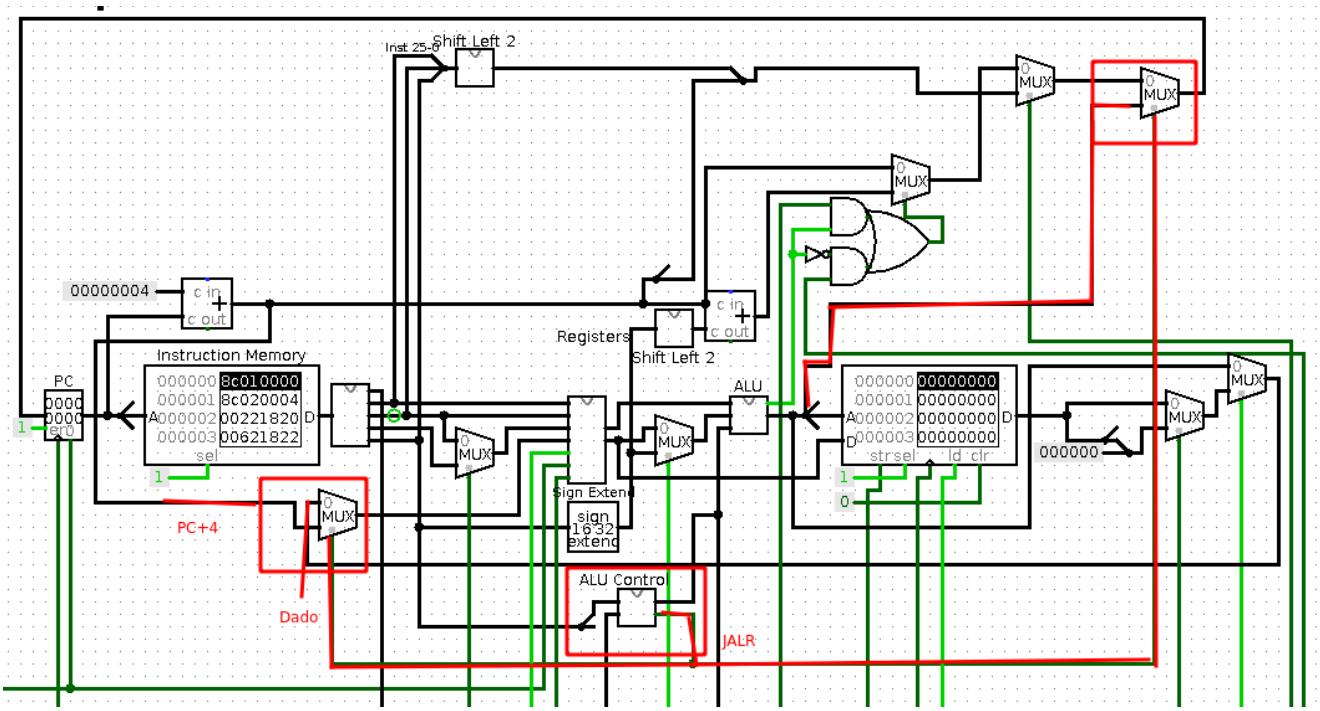

Figura 2. Esquemático do MIPS Monociclo com destaque às modificações. Fonte: Os Autores (2021).

Ademais, a ALU foi modificada de tal forma que ao receber “ALU Control = 011” irá retornar o sinal da primeira entrada (*bypass*).

Por ser uma instrução de tipo R, então os bits de controle seguirão os mesmos já pré-estabelecidos para o tipo R.

ERRATA: A MUX de Dado adicionada está com as entradas trocadas na imagem.

- **BNE (Branch on Not Equal)**

Essa instrução deve fazer um salto caso os dois operandos sejam diferentes

Monociclo, Multiciclo e Pipeline: Podemos adicionar um bit de controle BNE quando a instrução é chamada podemos ligar esse bit seguido de um "OR" com o branchEqual.

Pode-se utilizar toda a base do branchEqual somente adicionando esse bit de controle comentado acima e fazendo o or.

- **LBU (Load Byte Unsigned)**

Pode ser implementado selecionando o byte mais significativo de uma word lida e preenchendo seus bits mais significativos com a constante 0 (24 bits), completando novamente uma word (32 bits) com o byte selecionado nos 8 bits menos significativos. Essa lógica se repete no Multiciclo e no Pipeline, e foi necessária devido a limitação do Logisim de acesso à memória apenas por words.

Uso de 2 spliters e um MUX na saída da memória de dados. Criação de um bit de controle (SelectByteU) e atualização da ROM de controle para ativação dos bits necessários (bits de LW + SelectByteU).

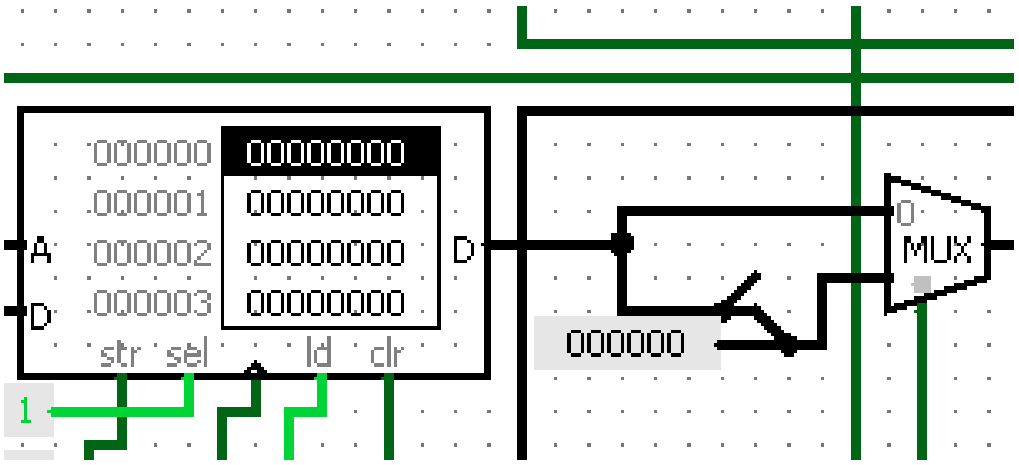

Figura 3. Modificações do LBU no MIPS Monociclo. Fonte: Os Autores (2021).

- **SLLV (Shift Left Logical Variable)**

Essa instrução pode ser considerada como uma operação da ALU usando os valores de registradores. Nesse caso, é incrementada dentro do circuito da ALU e alterado o controle de ALU para identificar essa operação.

Foi utilizado, portanto, um circuito shift left, disposto no logisim, dentro da ALU e endereçado na porta com código “100” do multiplexador (figura 4). Quando o controle de ALU recebe o código de operação da memória de operações e 2 bits do bloco de controle, envia esse número para o controle de operação da ALU.

Monociclo, multiciclo e pipeline: Uso de um circuito shift left e reorganização da lógica de controle da ALU.

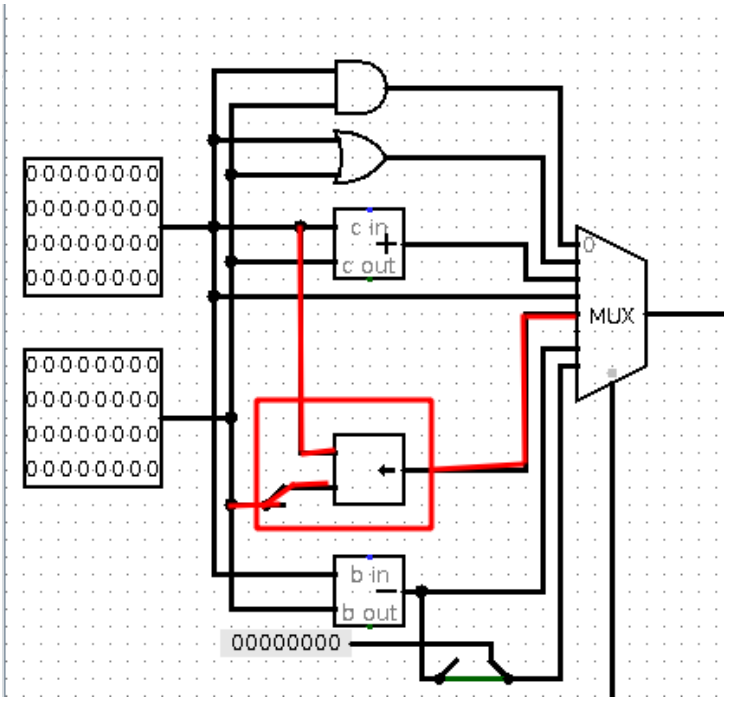

Figura 4. Modificações do SSLV na ALU do MIPS Multiciclo. Fonte: Os Autores (2021).

3. MIPS Multiciclo

- **JALR (Jump And Link Register)**

Para implementar esta operação no MIPS Multiciclo, como anteriormente, foi adicionado um pino de *JALR* de saída do *ALU_Control*. Este JALR foi enviado para a unidade de controle para ser utilizado para troca de estados. Esta troca de estados é a transição 6 ao 10 que ocorre em função de JALR. O estado 10 foi criado com o intuito específico de lidar com essa nova instrução. Este estado, por sua vez, é muito similar ao estado 7, que conclui as instruções do tipo R, porém, acresceu-se uma saída de sinal chamada PcToReg, cujo objetivo é sinalizar se o PC ou os dados irão ser armazenados em registrador. Portanto, ele é utilizado em uma multiplexação. Estas modificações estão presentes na figura 5.

ERRATA: A MUX de Dado adicionada está com as entradas trocadas na imagem.

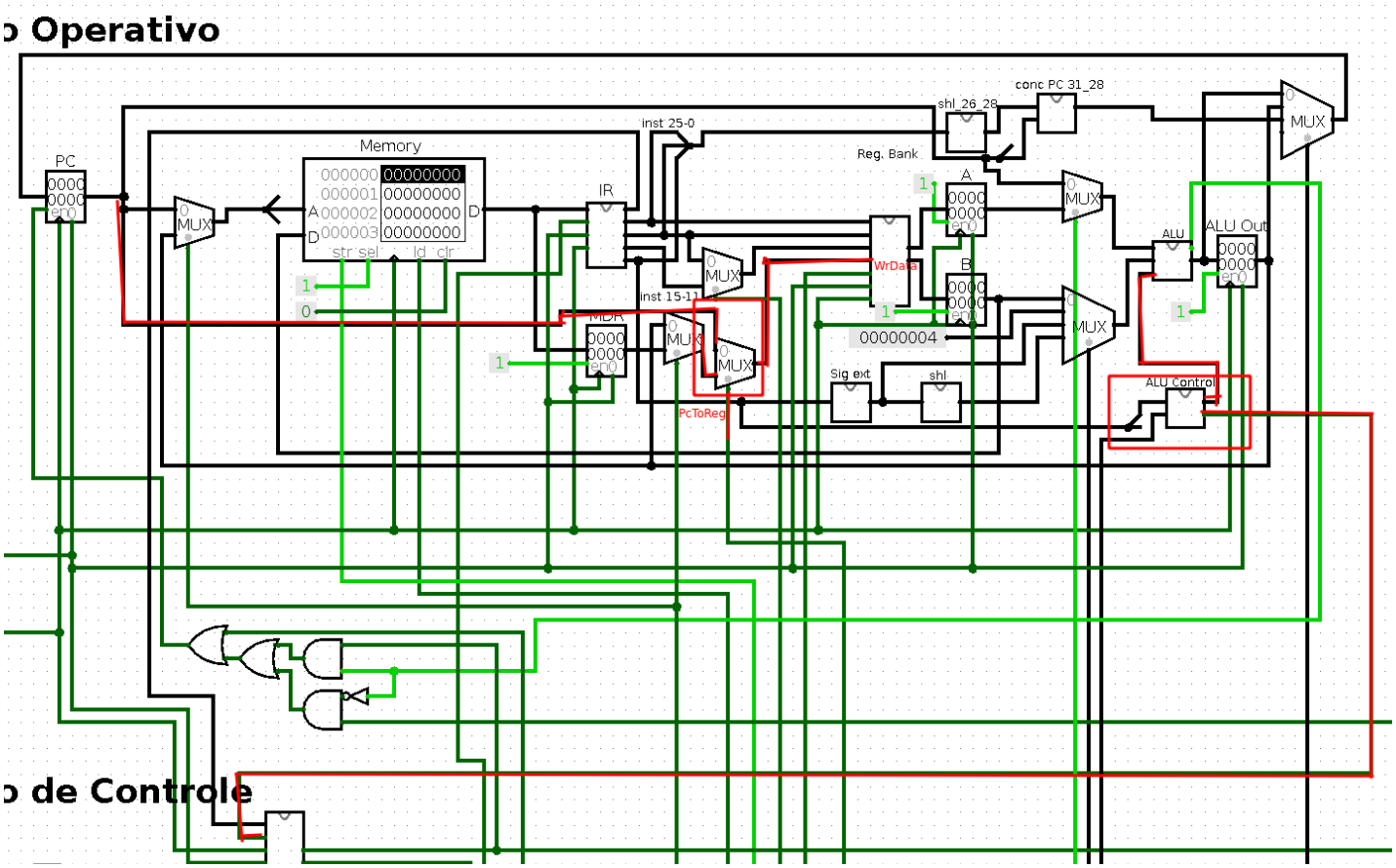

Figura 5. Esquemático do MIPS Multiciclo com destaque às modificações. Fonte: Os Autores (2021).

- **BNE (Branch on Not Equal)**

Como explicado no Monociclo foi adicionado um or. Pode-se utilizar toda a base do branchEqual, somente foi adicionando esse bit de controle comentado acima e fazendo o or.

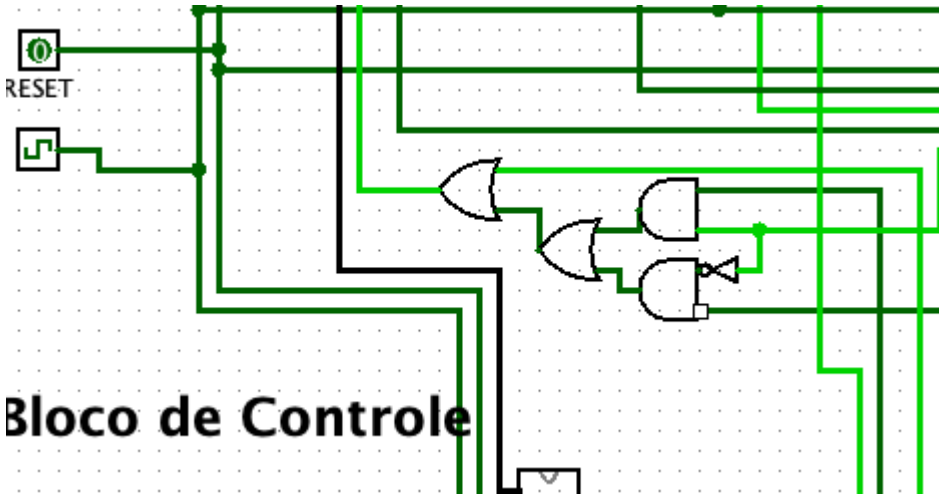

Figura 6. Modificações do BNE no MIPS Multiciclo. Fonte: Os Autores (2021).

- **LBU (Load Byte Unsigned)**

Uso de 2 spliters e um MUX na saída da ALU out. Criação de um bit de controle (SelectByteU) e adicionado um estado (estado 11) no mesmo caminho da instrução LW, mas substituindo o estado 4.

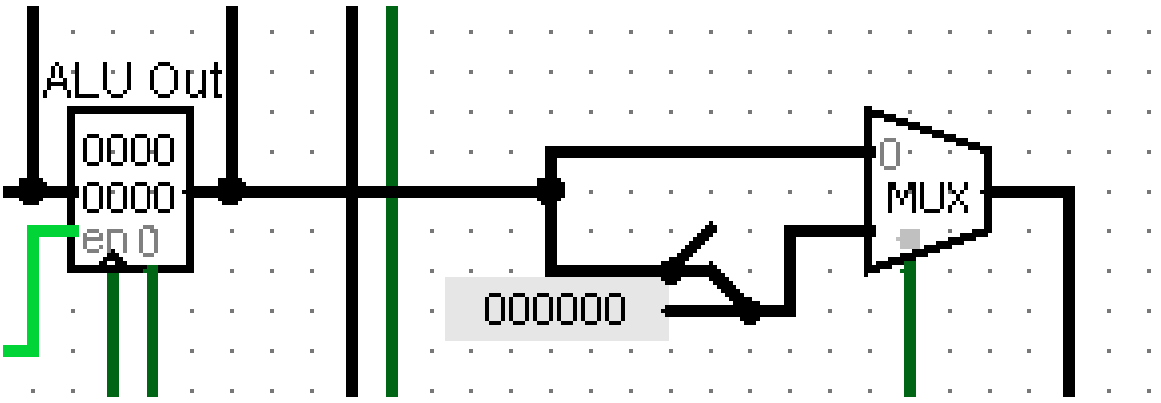

Figura 7. Modificações do LBU no MIPS Multiciclo. Fonte: Os Autores (2021).

- **SLLV (Shift Left Logical Variable)**

As modificações desta instrução seguem iguais às previamente implementadas no MIPS Monociclo.

Diagrama de Estados do Multiciclo

Na figura 8, conforme supracitado na seção, visualiza-se o diagrama de estados utilizado para a implementação do bloco de controle do MIPS Multiciclo.

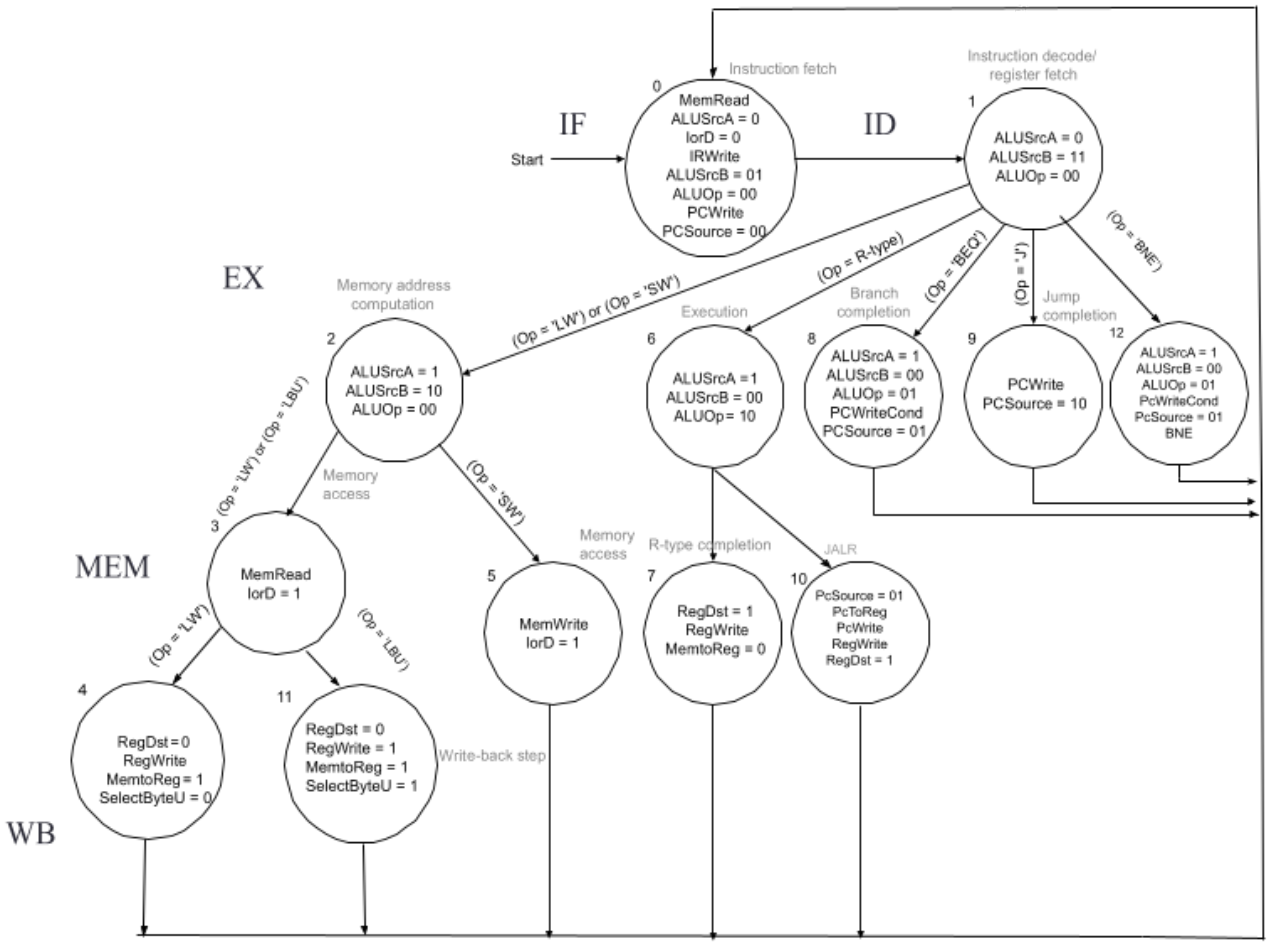

Figura 8. Diagrama de estados do MIPS Multiciclo. Fonte: Os Autores (2021).

4. MIPS Pipeline

- **JALR (Jump And Link Register)**

Para a implementação da instrução no pipeline, modificou-se da mesma forma que anteriormente no MIPS Pipeline, dessa forma, adicionando o sinal de *JALR* a partir da *ALU_Control*. Esse sinal foi passado para o próximo estado do pipeline e após foi concatenado junto *PcSource* pré-existente criando um fio PcSource de 2 bits, cujo mais significativo é o *JALR*. Por conseguinte, o multiplexador comandado por PcSource foi estendido, aceitando mais duas entradas que foram utilizadas para a entrada do dado de resultado da ALU. A ALU para a operação de JALR opera um *bypassing* da primeira entrada.

Além disso, também se criou um registrador para o PC+4 ser passado para o próximo estado. No estado de MEM, então, para a escrita em registrador, o dado de saída da ALU foi multiplexado com o PC+4 acionado também pelo sinal de *JALR*. As modificações estão em destaque na figura 9.

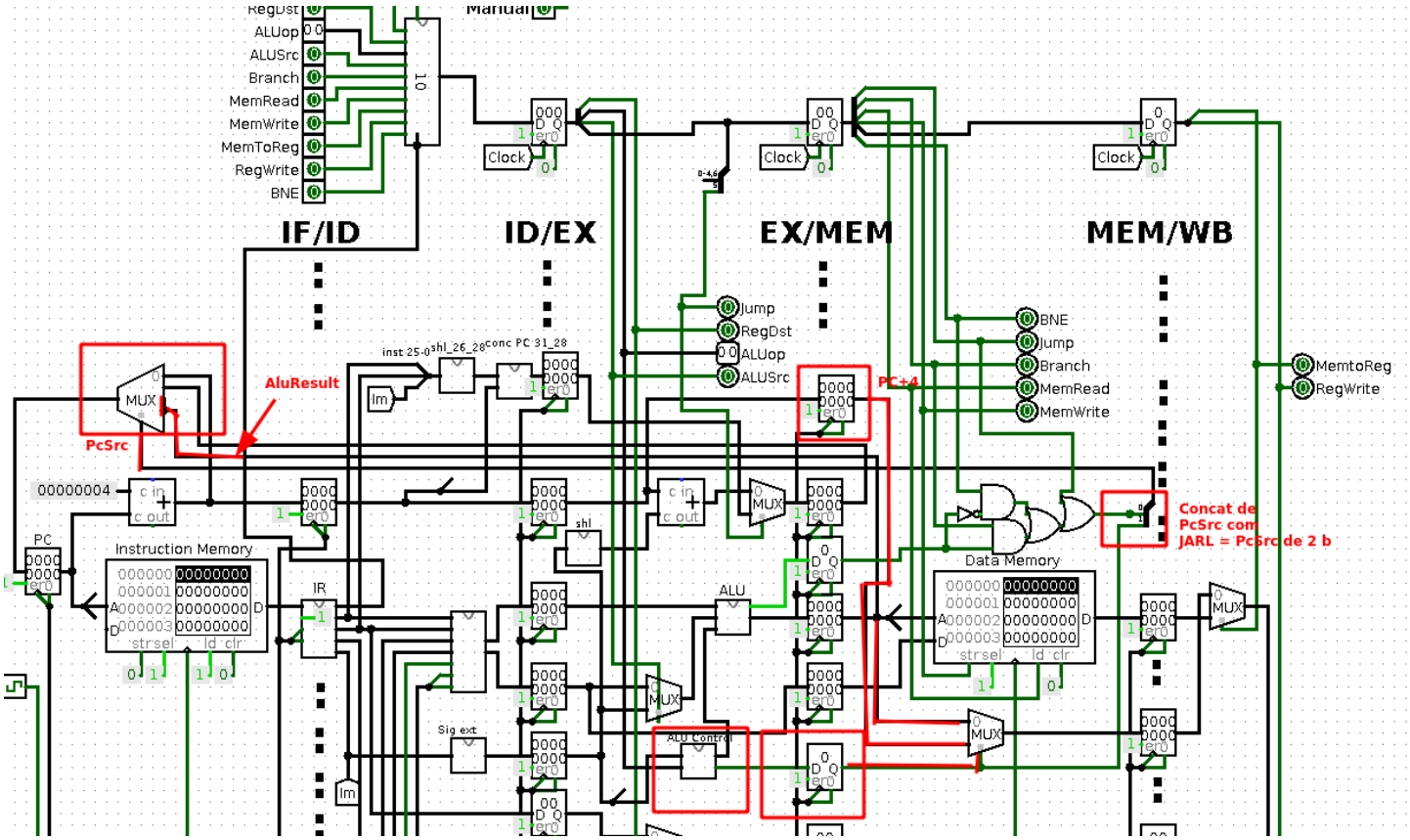

Figura 9. Modificações do JALR no MIPS Pipeline. Fonte: Os Autores (2021).

- **BNE (Branch on Not Equal)**

As alterações do BNE no MIPS Pipeline seguiram as mesmas diretivas utilizadas no monociclo, com modificação local na lógica de jump e adição de um pino de controle.

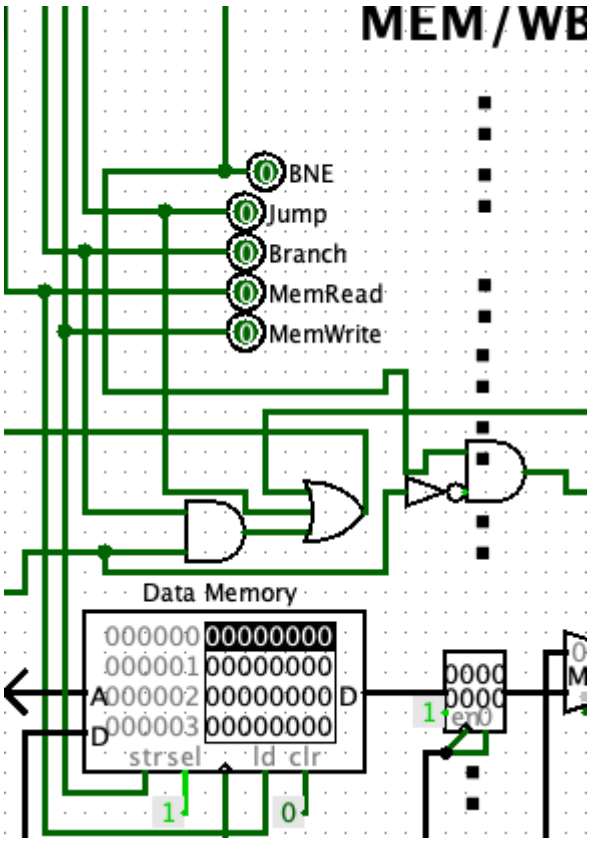

Figura 10. Modificações do BNE no MIPS Pipeline. Fonte: Os Autores (2021).

- **LBU (Load Byte Unsigned)**

Uso de 2 spliters e um MUX na saída da memória de dados (região WB). Criação de um bit de controle (SelectByteU). e o propagando até a área WB. Os bits de controle utilizados são os mesmos da instrução LW em conjunto com SelectByteU.

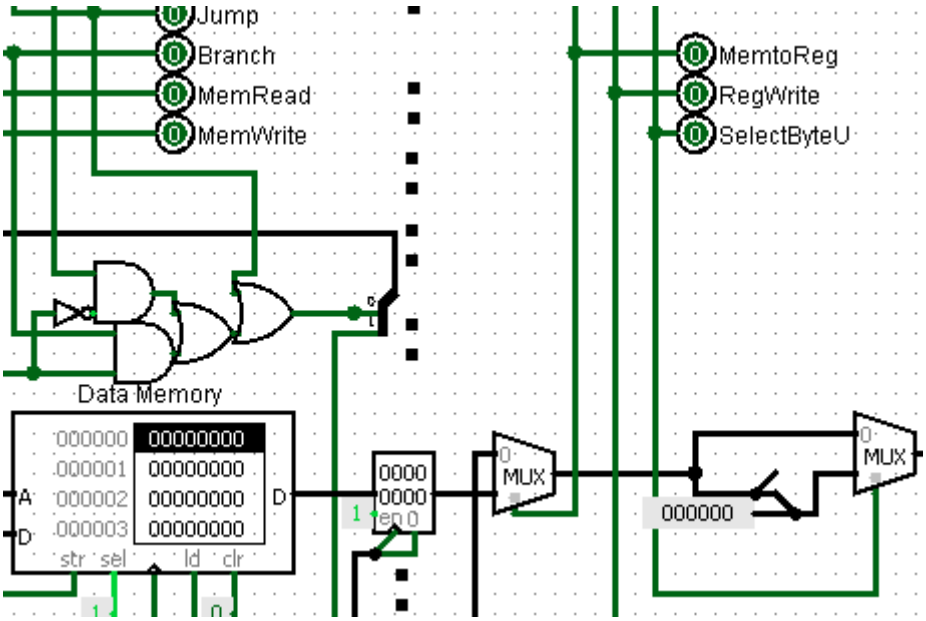

Figura 11. Modificações do LBU no MIPS Pipeline. Fonte: Os Autores (2021).

- **SLLV (Shift Left Logical Variable)**

As modificações desta instrução seguem iguais às previamente implementadas no MIPS Monociclo.