Projeto processador MIPS

Francisco Pegoraro Etcheverria
Gabriel Castelo Branco Gomes
Vinícius Daniel Spadotto
Guilherme de Sousa Cirumbolo
Erico Breyer de Freitas

December 2, 2023

Contents

1 Introdução

O presente projeto foi realizado mediante sutis modificações nos blocos de controle, criação de barreiras sequenciais (para o multiciclo e o *pipeline*) e específicas lógicas combinacionais para instruções cujos requisitos não estavam inclusos no modelo padrão do processador.

2 MIPS Monociclo

2.1 JAL

A instrução de *Jump And Link* foi implementada mediante a adição de uma nova *flag* ao bloco de controle, a qual atua como seleção para um multiplexador conectado ao *Write Address* do banco de registradores. Quando a *flag* está ativa, o multiplexador fornece a constante 31 como valor para a entrada supracitada, o que faz com que o registrador *$ra* seja escrito.
Similarmente, tal *flag* também participa ao fornecer o valor de $PC + 4$ para a *Write Data* por meio do uso de um *mux*.

2.2 BLTZAL

Similarmente à etapa anterior, a *flag* de *Link* foi utilizada para tal instrução. No entanto, foi-se utilizada uma entrada adicional na *ALU* cujo índice fora mapeado pelo *ALU Control* (o circuito de controle da *ALU* foi refatorado para permiti-lo).
Tal entrada na *ALU* analisa o *bit* mais significativo do primeiro operando e o inverte, levando, pois, a saída *Zero* a conter 1 se o *branch* deverá ser tomado.
Dessa forma, o *branch* é tomado da mesma forma que a instrução *BEQ* o faria, mudando, pois, somente a escrita no *$ra*, a qual utiliza utiliza como seletor no *MUX* supracitado o valor *AND(Zero, Link)*.

2.3 LBU

Tal instrução utilizou, na *ALU*, o mesmo bloco de divisão, a fim de obter o endereço correto de memória a ser acessado. Fez-se isso multiplexando o divisor com as *flags* (*ALUop2, ALUop1, ALUop0*) únicas designadas a tal instrução, usando, posteriormente, o *Low* (quociente) como o endereço de memória e o *High* (resto) como seletor da parte da palavra, sempre utilizando as *flags* únicas a tal instrução como meio de multiplexar adequadamente.

2.4 DIV

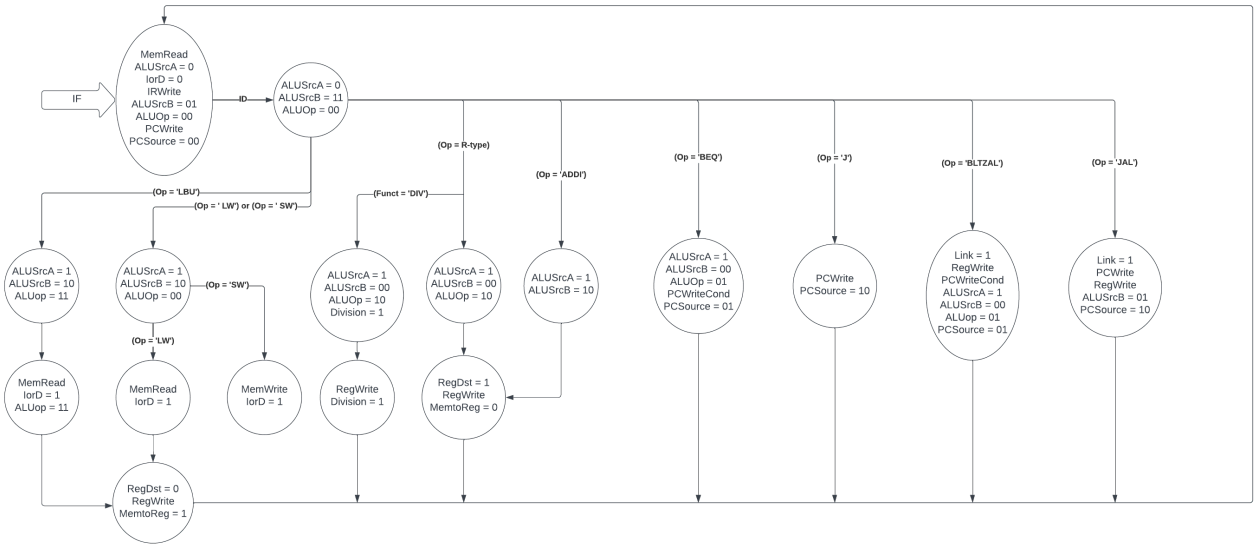
Primeiramente, criaram-se os registradores *High* e *Low* para armazenar, respectivamente, o quociente o resto da divisão. Em seguida, criou-se um mapeamento na *ALU* para a operação de divisão, utilizando o *ALU Control*. Nisso, a *ALU* passou a dar como resultado também o resto da divisão (32 *bits*).
Fica a cargo do controle da *ALU* também informar se o campo *funct* é de uma divisão, para assim propagar tal sinal e informar ao banco de registradores que se deve escrever no *High* e no *Low*.

2.5 ADDI

A *ADDI* é uma das instruções mais triviais, dado que basta multiplexar, com o sinal de controle $ALUsrc = 1$, a segunda entrada da *ALU* com o imediato com sinal estendido e o resultado do segundo endereço lido do banco de registradores.

3 MIPS Multiciclo

3.1 Diagrama de estados



3.2 JAL

De mesmo modo, o *JAL* foi implementado mediante o uso de uma *flag* de controle denominada *Link*, a qual serve para indicar se o registrador *ra* deve ser atualizado com o endereço da próxima instrução ou não. O *JAL* é implementado em duas etapas: a primeira funciona como um *Jump* qualquer, no entanto, a segunda utiliza a flag de controle *Link* para atualizar o registrador *ra*.

3.3 BLTZAL

Analogamente, o *BLTZAL* foi implementado mediante o uso de uma *flag* de controle denominada *Link*, a qual serve para indicar se o registrador *ra* deve ser atualizado com o endereço da próxima instrução ou não. O *BLTZAL* é implementado em duas etapas: a primeira funciona como um *Branch* qualquer, no entanto, a segunda utiliza a flag de controle *Link* e o *Zero* da *ALU* para atualizar o registrador *ra*.

3.4 LBU

A instrução *LBU* foi implementada forçando a *ALU* a realizar uma divisão por 4 (dado que uma palavra possui 4 *bytes*), utilizando a *flag* de controle *ALUop* com valor 011. No demais, a instrução *LBU* é implementada como uma instrução de *Load* qualquer, somente utilizando o quociente de tal divisão como o endereço de memória, e o resto como o *offset*.
Não precisou ser utilizada nenhuma *flag* adicional para identificação da instrução, dado que o uso de *ALUop* com valor 011 já é suficiente para tal.

3.5 DIV

A instrução *DIV* foi implementada mediante um mapeamento na *ALU* específico para tal instrução, a qual é realizada pelo *ALU Control*. Dessa forma, a *ALU* passa a dar como resultado também o resto da divisão (32 *bits*), utilizando, pois, uma *flag* de divisão advinda do *ALU Control* como meio para escrever nos registradores especiais *HI* e *LO*.

3.6 ADDI

De longe, a instrução mais fácil de ser implementada, dado que basta forçar a soma na *ALU*, porém com o *ALUSrcB* em 10, para forçar o imediato a ser o segundo operando. No mais, a instrução é um clone de qualquer instrução de *R-Type*.

4 MIPS Pipeline

4.1 JAL

Similarmente aos processadores anteriores, tal instrução foi implementada mediante a adição de uma *flag* de controle denominada *Link*, a qual determina se o valor do registrador `$ra` deve ser atualizado com o endereço da próxima instrução ou não. A instrução comporta-se igualmente a um `J` normal, com a diferença de que o valor do registrador `$ra` é atualizado com o endereço da próxima instrução, sendo mutiplexado pela *flag* de controle *Link*.

4.2 BLTZAL

A instrução `BLTZAL` foi implementada de forma similar à instrução *JAL*, com a diferença de que o registrador `$ra` é atualizado apenas se a condição de *branch* for satisfeita. Ademais, a instrução `BLTZAL` é implementada com uma entrada única na *ALU*, tendo seu índice dado pelo *ALUop*. A operação é feita analisando o *bit* mais significativo e negando seu valor, de forma que o *Zero* dê o valor correto. Finalmente, o *branch* é tomado se o valor do *Zero* for igual a 1, assim como o registrador `$ra` é atualizado com o endereço da próxima instrução se o valor do *Zero* for igual a 1 e a *flag* de controle *Link* estiver ativa.

4.3 LBU

Tal instrução utilizou a lógica da instrução de divisão, com a diferença de que o valor do segundo operando é multiplexado (por meio de um circuito combinacional que ativa uma *flag* informando que a instrução é *LBU*) para ser 4. De tal forma, o endereço da memória é o quociente da divisão, e o resto é o *offset* dentro da palavra.
Os valores são carregados de etapa a etapa por registradores criados para tal fim. No mais, o funcionamento é equivalente a um *LW*.

4.4 DIV

A instrução *DIV* foi implementada mediante uma entrada específica na *ALU* para tal operação, tendo seu índice calculado pelo *ALU Control*. A *ALU* dá como resultado também o resto da divisão, o qual é armazenado em registradores intermediários até chegar na etapa de *Write Back*, onde é armazenado no registrador `$LO`, assim como o quociente é armazenado no registrador `$HI`.

4.5 ADDI

Trivialmente, a instrução comporta-se de maneira igual a qualquer instrução do tipo *R*, com a diferença de que a operação é sempre uma soma e o segundo operando é um valor imediato. De tal forma, a instrução possui as mesmas flags de controle que uma instrução do tipo *R*, mudando apenas o valor de *ALUop* para ser sempre 00, forçando uma soma, e o de *ALUSrcB* para ser 10, forçando o segundo operando a ser sempre o valor imediato.

5 Conclusão

Infere-se, destarte, que houve alta similaridade de implementação entre os processadores, sendo a maior diferença a necessidade de inclusão de registradores adicionais e o reuso de componentes específicos. Com o presente trabalho, desenvolveu-se o entendimento sobre o funcionamento básico de um processador de arquitetura *RISC*.