UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA

INF01113 Organização de Computadores B - 2017/2

Trabalho 1

MIPS Extension: Implementação de novas instruções

Prof. Dr. Paolo Rech

Felipe Fischer Comerlato - 242264
Gabriel de Souza Abreu - 242273
Gustavo Madeira Santana - 252853
Lucas Bragante Corssac - 219820

Porto Alegre, Rio Grande do Sul

Sumário

1 Introdução

1.1 Objetivo

Este trabalho tem como objetivo estender as implementações do MIPS Monociclo, Multiciclo e Pipeline, adicionando novas instruções. As instruções a serem implementadas são: (i) jump and link, (ii) bitwise or, (iii) load upper immediate, (iv) load halfword e (v) multiply.

Para ler somente as alterações feitas nas arquiteturas MIPS para implementação das novas instruções, pule para Seção 3.

2 Organização MIPS

2.1 Instruções

Instruções são segmentos de código que definem os passos que um processador irá executar, é a unidade mais básica de uma operação que pode ser executada. O MIPS estudado é um processador de 32 bits e com um conjunto de instruções reduzido (RISC). Cada instrução tem 32 bits e existem três tipos básicos de instruções, são elas: (i) Jump, (ii) Register, e (iii) Immediate.

2.1.1 Tipo Jump

As instruções de tipo Jump possuem apenas dois campos. Os primeiros 6 bits são destinados ao *opcode*, e os outros 26 bits são destinados ao endereço.

2.1.2 Tipo Register

As instruções do tipo Register possuem seis campos. Os primeiros 6 bits são destinados ao *opcode*, os próximos 5 bits definem o primeiro registrador de leitura (R_s). Os próximos 10 bits definem os dois próximos registradores a serem utilizados (R_t, R_d), ambos de 5 bits cada. O próximo campo também tem 5 bits e é nomeado shamt ($shift\ ammount$). Por fim, os últimos 6 bits definem o parâmetro $function$, que será usado para definir operações na ULA.

2.1.3 Tipo Immediate

As instruções do tipo Immediate possuem quatro campos. Assim como as instruções do tipo Register, os primeiros três campos são *opcode*, R_t, e R_d. Os últimos 16 bits definem um valor imediato que será usado conforme operação da instrução em execução.

2.2 Datapath e Controle

Na arquitetura MIPS Monociclo, o datapath e o controle foram desenvolvidos de forma que todas instruções levem apenas um ciclo de clock para completar sua execução. Essa simplificação garante um CPI de 1, mas acaba fazendo com que todas instruções levem o mesmo tempo da maior instrução para ser executada, ou seja, o clock máximo possível é reduzido em função da instrução de maior caminho crítico (LW).

Para contornar o problema de instruções curtas levarem o mesmo tempo uma longa (e.g. jump vs lw), foi proposto uma arquitetura Multiciclo para o MIPS. Agora cada instrução levará um número de ciclos variado para completar sua execução. Assim, uma instrução de $jump$ pode levar apenas três ciclos para completar sua execução, enquanto a instrução *load word* levará cinco ciclos. Isso permite que o clock máximo do processador aumente consideravelmente. Em contrapartida, o CPI passa a ser variável, e não 1 como no Monociclo. Outra vantagem do MIPS Multiciclo é o compartilhamento de hardware, diminuindo a área total do circuito.

Buscando unir o melhor do Monociclo com o melhor do Multiciclo, foi proposto o MIPS Pipeline. Agora, toda instrução leva 5 ciclos para completar sua execução, onde cada ciclo é um estágio do pipeline. A vantagem é que uma vez que o pipeline está cheio, a cada ciclo uma instrução será completada. Considerando que o conjunto de instruções executadas é da ordem de 10^9, o CPI efetivo é 1, igual ao MIPS Monociclo. Alguns problemas podem surgir devido a dependência de dados entre instruções sendo executadas de forma paralela, ou instruções de branch onde não se sabe qual será a próxima instrução a ser executada. Para isso foi desenvolvido técnicas de branch prediction, forwarding/bypassing de dados e reordenamento de instruções, que buscam remediar esses problemas.

3 Novas Instruções

3.1 Jump and Link

3.1.1 Operação

O funcionamento da instrução Jump and Link (jal) é semelhante a instrução Jump, a diferença está no fato de que o jal salva o endereço de retorno, antes de realizar o jump, em um registrador específico ($ra; 0x1f). Isto permite que uma subrotina retorne na próxima instrução após o jal e continue a execução do programa. Assim como o Jump, o jal é uma instrução do tipo J, portanto possui apenas dois campos: opcode e valor imediato.

3.1.2 Datapath e Controle - MIPS Monociclo

Para implementar a instrução jal no MIPS Monociclo, tanto o datapath como a parte de controle foi alterada.

O mux 2:1 que define qual será o registrador em escrita foi alterado para ter sua saída ligada em outro mux 2:1. Esse novo mux tem como seletor um novo sinal de controle denominado *jal*. A primeira entrada deste novo mux é a saída do mux que escolhe entre os registradores R_t e R_d, selecionado pelo sinal RegDst. Já a segunda entrada tem o valor constante 0x1f, endereço referente ao registrador que receberá o endereço de retorno ($ra).

Um novo mux 2:1 foi inserido para definir o dado que será escrito no registrador selecionado no banco de registradores. A entrada 0 deste novo mux receberá a saída do mux que seleciona entre a ALU e a memória, enquanto a entrada 1 do mux terá o sinal $(PC + 4)$, ou seja, o endereço da próxima instrução após o jal. O seletor deste novo mux é o sinal de controle *jal*.

A Tabela 1 mostra o comportamento dos sinais de controle envolvidos na operação da instrução jal.

Tabela 1: Comportamento do MIPS Monociclo de acordo com o sinais de controle RegDst e jal.

RegDst	jal	Registrador em escrita	Dados em escrita	PC
0	0	R_t	ALU/Memória	PC ← PC+4
1	0	R_d	ALU/Memória	PC ← PC+4
X	1	$ra	PC+4	PC ← (jump address)

3.1.3 Datapath e Controle - MIPS Multiciclo

No MIP Multiciclo, o datapath foi alterado de forma semelhante ao MIPS Monociclo, porém não foi necessário adicionar o mux que seleciona o dado que será escrito no banco de registradores pois ele já existe, e é controlado pelo sinal *MemtoReg*.

As maiores alterações ficaram na parte do controle. O estado da instrução "jump" foi alterado mas semanticamente continua tendo a mesma funcionalidade. Onde antes os sinais do estado jump eram apenas $PCWrite = 1$ e $PCSource = 10$, agora também são ativados os sinais $ALUsrcA = 0$, $ALUsrcB = 10$, $ALUop = 00$. Dessa forma, o jump continua funcionando como jump, mas agora temos o valor do endereço da próxima instrução no registrador $ULAout$. Com isso, caso a instrução seja "jal", o próximo estado escreverá o valor do registrador $ALUout$ no registrador $ra. Para isso, é ativado os sinais $jal = 1$ e $MemtoReg = 0$, assim o registrador $ALUout$ é selecionado no mux através do sinal $MemtoReg$ e o endereço $0x1f$ referente ao registrador $ra é selecionado no mux controlador pelo sinal jal, que define qual registrador é selecionado para escrita no banco de registradores.

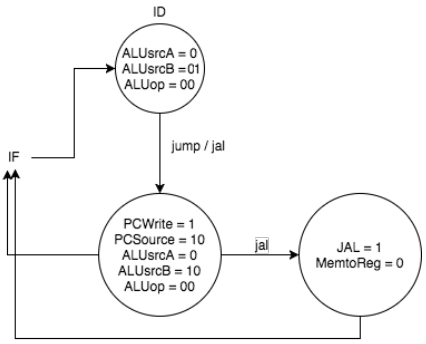

Figura 1: Novo estado jal incluído na FSM do MIPS Multiciclo.

3.1.4 Datapath e Controle - MIPS Pipeline

As alterações realizadas no MIPS Pipeline foram semelhantes ao Monociclo, incluindo um mux controlado pelo sinal jal para selecionar o registrador $ra como o registrador destino e escrevendo o valor do PC+8, considerando o branch delay slot, como endereço de retorno.

Para ter o valor do PC no estágio de Write-back, foi necessário incluir um novo registrador, uma vez que o PC+4 só era salvo até o estágio de execução. Para economizar um somador, o PC+4 foi obtido do estágio ID e não do estágio EX, assim temos de forma direta o valor do PC+8.

3.2 Bitwise OR (or) e Multiply (mult)

3.2.1 Operação

Tanto a instrução *or* como a *mult* são operações realizadas na ULA. A instrução *or* realiza a função lógica *OR* entre as duas entradas selecionadas na ULA, enquanto a instrução *mult* realiza a multiplicação das duas entradas da ULA.

3.2.2 Datapath e Controle

Ambas instruções necessitam inserção de seus blocos lógicos na ULA. Tanto o datapath como o bloco de controle permanceram sem alterações, independente da arquitetura (monociclo, multiciclo, pipeline).

Ao executar a instrução, a operação desejada é selecionada através do campo $function$ da instrução, uma vez que o sinal $ALUop$ será 10, tanto para a instrução *or* como para a instrução *mult*. Com isso, ambas instruções compartilham o mesmo estado na FSM.

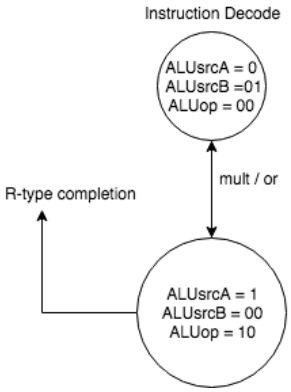

Figura 2: Estado da FSM do MIPS Multiciclo para instruções aritméticas que utilizam o campo $function$.

3.3 Load Upper Immediate (lui)

3.3.1 Operação

A instrução Load Upper Immediate concatena os 16 bits do valor imediato (15-0) com 16 bits de zeros à direita, obtendo uma palavra de 32 bits. Essa palavra é escrita no registrador R_t.

3.3.2 Datapath e Controle - MIPS Monociclo

A implementação da instrução LUI no MIPS Monociclo exigiu a alteração do datapath e do controle.

No bloco de controle, foi adicionado um bit de controle (LUI) para selecionar a entrada de um novo mux. Esse novo mux tem como entrada 0 outro mux que seleciona entre a saída da ALU ou da memória. Já a entrada 1 seleciona a palavra (32 bits) referente a concatenação dos 16 bits do valor imediato (15-0) com os 16 zeros à direita. A saída do mux é ligada no banco de registradores, onde será escrita no registrador R_t.

3.3.3 Datapath e Controle - MIPS Multiciclo

No MIPS Multiciclo, foi inserido um pequeno circuito denominado LUI, semelhante ao implementado no MIPS Monociclo, mas com um novo mux em seu interior, que seleciona entre a palavra da instrução LUI e o valor do mux controlado pelo sinal MemtoReg. A saída desse novo mux é ligada ao banco de registradores, onde será escrita no registrador R_t.

Na máquina de estados do bloco de controle, houve apenas a adição de um estado (0xd), que parte do estado 0x1 e finaliza a instrução, retornando ao estado 0x0. Esse novo estado tem apenas os sinais RegWrite e LUI ativados.

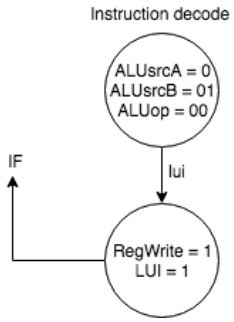

Figura 3: Novo estado LUI incluído na FSM do MIPS Multiciclo.

3.3.4 Datapath e Controle - MIPS Pipeline

As alterações realizadas no MIPS Pipeline seguem as alterações realizadas no Monociclo, garantindo que o bit de controle LUI propague até o estágio de Write-back.

3.4 Load Halfword (lh)

3.4.1 Operação

A instrução Load Halfword busca na memória o valor apontado pelo endereço que é a soma da base (R_t) com o valor imediato (bits 15-0), e estende o sinal dos 16 bits menos significativos para ter uma

palavra de 32 bits. Este novo dado é então escrito no registrador R_s.

3.4.2 Datapath e Controle - MIPS Monociclo

Para implementar a instrução lh no MIPS Monociclo, tanto o datapath como a parte de controle foi alterada.

No bloco de controle foi adicionado um bit de controle (LH), que seleciona a entrada de um novo mux. Esse novo mux inserido no datapath mux escolhe entre a saída da memória ou meia palavra estendida. A saída deste mux é uma entrada no mux controlado pelo sinal MemtoReg.

3.4.3 Datapath e Controle - MIPS Multiciclo

No MIPS Multiciclo, a saída da memória é sempre escrita no registrador MDR. Isto foi alterado para que o valor no MDR seja ou a saída da memória, ou meia palavra com o sinal estendido. Para isso, um novo bit de controle (LH) e um novo mux, controlado por este novo sinal, foi inserido.

Na máquina de estados do bloco de controle foi adicionado apenas um estado, 0xb (LH), que parte do estado 0x2 (Memory Address Computation) e vai para o estado 0x4 (Write-back). Este estado é idêntico ao estado 0x3 (LW), exceto pelo sinal LH que é ativado.

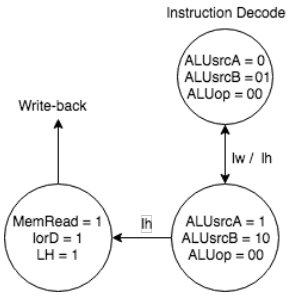

Figura 4: Novo estado LH incluído na FSM do MIPS Multiciclo.

3.4.4 Datapath e Controle - MIPS Pipeline

As alterações realizadas no MIPS Pipeline seguem as alterações realizadas no Monociclo, garantindo que o bit de controle LH propague até o estágio de Write-back.