INSTITUTO SUPERIOR TÉCNICO

MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

ARQUITECTURAS AVANÇADAS DE COMPUTADORES

Simulação de um processador μRisc

Maria Margarida Dias dos Reis	n.º 73099
Nuno Miguel Rodrigues Machado	n.º 74236

Lisboa, 29 de Março 2014

Índice

1 Introdução

Com este trabalho laboratorial pretende-se projectar um processador μRisc, de 16 *bits* com arquitectura RISC. O processador possui 8 registos de uso geral e 42 instruções. O projecto do processador é feito com recurso a uma linguagem de descrição de *hardware* - VHDL.

2 Características do Processador

O processador elaborado foi simulado para uma placa Artix 7 e tem as seguintes características:

- 16 *bits*;
- 8 registos de uso geral de 16 *bits* de largura (R0, . . . , R7);
- 42 instruções;
- instruções de 3 operandos;
- organização de dados na memória do tipo *big endian*;
- uma memória ROM de 8 KBytes (4k endereços × 2 *bytes*) endereçada com palavras de 12 *bits* utilizada para as instruções/programa e uma memória RAM de 8 KBytes (4k endereços × 2 *bytes*) endereçada com palavras de 12 *bits* para os dados.

3 Estrutura do Processador

O processador μRisc que foi projectado encontra-se dividido em quatro andares - num primeiro andar é feito o *instruction fetch* (IF), no segundo andar é feito o *instruction decode* (ID) e o *operand fetch* (OF), no terceiro andar são executadas operações da ALU (EX) e de acesso à memória de dados (MEM) e, por fim, no quarto e último andar é feita a escrita no banco de registos, o *write back* (WB).

3.1 Primeiro Andar - IF

No primeiro andar obtem-se a instrução a ser executada a cada ciclo. Como todas as instruções do programa são armazenadas na memória ROM, o *instruction fetch* (IF) tem a função de endereçar a ROM com o *program counter* e ler a instrução desse endereço.

FIGURA 1

O *instruction fetch* (IF) é simplesmente um somador, que em cada ciclo soma 1 ao endereço actual, e armazena o resultado no registo *PC*, como se pode ver na figura 1. O endereço actual além de ser um operando do somador também endereça a memória ROM.

Existe duas situações que altera o funcionamento sequencial do *instruction fetch* (IF), a primeira ocorre quando há uma transfêrencia de controlo do tipo condicional ou incondicional, selecionando o *Destino_cond* no *MUX_1* e o resultado do somador no *MUX_2*, ou seja, $S\_cond = 1$ e $S\_jump = 0$. A ultima situação o corre quando existe uma transferência de controlo do tipo *Jump and Link* ou *Jump Registor*, selecionando o *Destino_jump* no *MUX_2*, ou seja, $S\_jump = 1$.

3.2 Segundo Andar - ID e OF

3.3 Terceiro Andar - EX e MEM

Neste andar trata-se de executar operações da ALU bem com operações da memória, sendo que, ao contrário do MIPS, em que é possível utilizar a ALU e a memória na mesma instrução, no processador μRisc projectado tal não é possível.

ALU

Relativamente às operações de memória é necessário tratar de *loads* e *stores*.

3.4 Quarto Andar - WB

No último andar os diversos resultados possíveis são escritos no banco de registos - pode ser o resultado de uma operação da ALU, o resultado de uma operação sobre a memória (*load*), o carregamento de uma constante ou guardar em R7 o valor do próximo *program counter*. Como se pode ver na figura seguinte, a seleção de qual os resultados deve ser escrito é feita com recurso a um MUX 4:1.

FIGURA

Uma vez seleccionado o resultado a escrever é preciso escolher qual o registo onde se pretende escrever esse mesmo resultado (registo WC).

Para instruções da ALU a escolha do registo onde se quer escrever o resultado final é feita com recurso aos *bits* 11 a 13 da instrução, assim como para operações de carregamento de constantes. Porém, para operações de transferência de controlo esses mesmos *bits* representam a operação a realizar, pelo que não basta utilizar os 3 *bits* referidos anteriormente para controlar um *decoder* que colocasse a *high* um dos 7 *enables* (que estão armazenados nos 7 *bits* de um vector), tal como pensado originalmente e como pode ser visto na figura abaixo.

FIGURA

No entanto, a solução acima tem um problema - suponha-se o caso da instrução 1401 (HEX) que corresponde a um *jump if true* mediante a condição do resultado da ALU ser negativo. Os *bits* 11 a 13 da instrução são 010 e, como tal, o *enable* do registo R2 ficaria activo. Porém, não se pretende escrever nesse registo. O mesmo decorre para uma operação de *store* na RAM e NOP.

Assim, para resolver o problema é necessário criar um sinal que faça *override* ao *enable* que o *decoder* colocou a *high*, permitindo o sinal de *override* colocar o *enable* a *low*, tal como pretendido, para que não se escreva em nenhum registo.

O sinal de *override* é obtido com recurso à seguinte lógica.

FIGURA

Como se pode ver, para o caso de operações da ALU, operações de *load*, carregamento de constantes e o caso de *jump and link*, o sinal de *override* fica a *high*. Para o caso de *store* na memória, transferências de controlo que não *jump and link* e NOP, o sinal de *override* fica a *low*, tal como pretendido. De notar também que a escrita nos registos é feita no flanco positivo do relógio.

Na figura abaixo encontra-se o esquema completo do andar de *write back*.

Todo list