Arquitecturas Avançadas de Computadores
Lab I - Desenvolvimento de um processador RISC simples

João Lages, Guilherme Pires
75286 75432

Este é o relatório do 1º Projecto da U.C. de Arquitecturas Avançadas de Computadores, no qual se pretendia que os alunos modificassem um processador multi-ciclo de 5 estágios para que este passasse a funcionar como um processador pipeline.

***Index Terms*—IST, AAC, RISC, Pipeline, Xilinx ISE, Resolução de conflitos**

I. Introdução

O objectivo deste trabalho era o desenvolvimento de um processador que suportasse o conjunto de Instruções da Arquitectura Microblaze. Para este efeito, foi-nos dado o código VHDL de um processador 5-stage multi-cycle que suportava todas as operações deste conjunto de instruções, menos as operações relativas a Branch Delay (que viriam a ser posteriormente implementadas por nós), uma vez que este tipo de operações não faria sentido numa arquitectura multi-cycle.

Ao objectivo descrito, acrescia o de modificar o processador dado, de forma a que este funcionasse como um pipeline de 5 estágios. Para tal, houve a necessidade de identificar e resolver os conflitos de dados e de controlo, decorrentes dessa arquitectura, bem como algumas outras modificações necessárias ao funcionamento pretendido. Após a análise do código VHDL fornecido, foi possível representar graficamente o processador original, o que permitiu um melhor entendimento do seu funcionamento e uma mais fácil previsão e planeamento das modificações a implementar.

II. Descrição do processador original

Passamos a descrever brevemente o funcionamento do processador original.

A. Andar ID

Como pode observar-se no Apêndice I, o registo IF/ID é responsável pela transição de apenas um sinal (PC) do estágio IF para o estágio ID. Este sinal endereça a memória de instruções no andar IF, levando a que a saída desta memória seja a instrução a ser descodificada pelo decoder no andar ID. Uma vez que a memória é de leitura síncrona, o sinal que contém a instrução não passa através do registo, já que demora um ciclo a ficar disponível.

Neste andar, o decoder descodifica a instrução e a gera todos os sinais de controlo necessários à execução da instrução pelos andares seguintes. Alguns sinais são enviados pelo decoder à unidade de Branch Control, responsável por calcular o PC seguinte (Next_PC). Este sinal é armazenado no registo MEM/WB para no último ciclo de cada instrução ser armazenado no registo IF/ID, permitindo que o sinal I e PC estejam "sincronizados" no andar de ID.

Em todo o caso, a passagem deste sinal pelos dois registos referidos é desnecessária e inviável na arquitectura pipeline, pelo que mais tarde foi retirada/modificada.

Este andar endereça e recebe (leitura assíncrona) do Register File os operandos necessários à instrução a executar (uma das origens de conflitos de dados). O operando dado pelo registo RA é ainda utilizado pela unidade de Branch Control como operando sobre o qual são testadas algumas condições de salto (maior que 0, menor que 0, igual a 0).

A flag de carry da ALU é também uma das entradas do Decoder. Esta está armazenada no MSR e provém da ALU. Mais tarde, verificou-se que o cálculo e armazenamento da flag C também podiam originar conflitos de dados, que tiveram então de ser abordados.

B. Andar EX

Os sinais de controlo e os operandos gerados pelo decoder são transmitidos para o andar EX, através do registo ID/EX. Nestes sinais incluem-se os sinais responsáveis por instruir a ALU sobre qual a operação a realizar.

O resultado é passado para o andar MEM, juntamente com os sinais de controlo da Memória de dados, cujo funcionamento será descrito adiante. A Flag de carry é armazenada no Registo de Estado do processador.

C. Andar MEM

No andar MEM são realizados os acessos de leitura da Memória de Dados. A arquitectura a implementar não continha instruções que necessitassem de leituras e escritas na mesma instrução.

Este andar recebe, no sinal que contém o resultado do andar anterior, o endereço da posição de Memória indicada pela instrução e faz um acesso de leitura a essa posição.

Tal como a Memória de Instruções, a Memória de Dados é de leitura Síncrona, pelo que o sinal resultante da sua saída de dados não atravessa o registo de transição entre o andar MEM e WB.

D. Andar WB

Neste andar, os resultados da Instrução executada podem ser armazenados na Memória de Dados ou em algum dos registos do Register File. Os sinais WE gerados pelo decoder no andar ID vão indicar aos respectivos destinos se estes devem permitir a sua escrita neste ciclo de relógio ou não.

No caso de uma escrita num registo, a origem dos dados pode ser o resultado de uma operação na ALU ou o resultado de uma leitura da memória.

No caso de uma escrita na Memória, a origem dos dados é o resultado de uma leitura de um registo. O endereço da posição de memória que irá ser escrita é dado pelo resultado da ALU.

Note-se que a memória não pode ser acedida para escrita e leitura em simultâneo, nem em ciclos de relógio consecutivos, já que a leitura é síncrona e por isso "demora" um ciclo de relógio. Este facto está na origem de mais um conflito que foi necessário resolver.

III. Arquitectura Pipeline: Alterações e conflitos

Em suma, para que o processador funcionasse em pipeline, era necessário permitir a "convivência" de 5 instruções distintas - uma em cada andar - em cada ciclo de relógio. Esse comportamento leva ao aparecimento de alguns conflitos:

- Uma instrução necessita do resultado de outra, que ainda não foi escrito, para realizar uma operação lógica ou aritmética (seja ela para endereçar a Memória ou armazenar um valor num registo). → Conflito de Dados
- A arquitectura necessita do resultado da unidade de Branch Control e este ainda não está disponível. → Conflito de controlo
- Mais do que uma instrução necessita de usar um recurso que só pode ser utilizado por uma instrução de cada vez.→ Conflito estrutural

A. Alterações gerais

Para um funcionamento em pipeline, houve algumas alterações simples que foram desde logo feitas:

- Os WE dos registos de transição passaram a estar sempre a 1, excepto na fase de inicialização.
- O sinal NextPC deixou de ser armazenado no registo MEM/WB

As restantes alterações prenderam-se sobretudo com a resolução dos conflitos acima descritos. De seguida passamos a descrever como identificámos e resolvemos esses conflitos.

B. Conflitos de dados

Para identificar os conflitos de dados, relembrámos que um registo que seja escrito por uma instrução, só tem o seu valor actualizado após o andar WB. Desta maneira, um conflito de dados pode ocorrer quando a instrução que escreve no registo estiver nos andares de EX, MEM e WB. Os conflitos de dados podem ser resolvidos através da introdução de Stalls no pipeline, que permitem à instrução que está no andar de ID esperar que o registo de que precisa seja disponibilizado. Contudo, a introdução de Stalls significa congelar os andares de IF e ID, atrasando a execução do pipeline. Nesse sentido, procurámos implementar outros métodos de resolução de conflitos de dados.

No caso de a instrução de escrita estar no andar de WB, a resolução dos conflitos de dados é, de facto, bastante simples: cinge-se a passar a efectuar as escritas nos registos nos flancos descendentes do relógio. Assim, a escrita ocorre meio ciclo antes, permitindo ao decoder ter ainda meio ciclo para ler o registo correctamente.

No caso de a instrução de escrita estar nos andares de EX ou MEM, a estratégia que escolhemos implementar denomina-se por Data Forwarding e consiste em ligar os resultados dos andares de EX e MEM e as saídas do RF a um multiplexer, cuja saída será ligada a uma das entradas de operandos do decoder (cada entrada terá o seu multiplexer). A entrada de selecção do multiplexer é dada pela lógica que selecciona o input correcto quando detecta a ocorrência de um conflito de dados.

Para implementar este Multiplexer, tivemos de identificar as instruções que podiam dar origem a conflitos de dados: no andar de ID, instruções que utilizassem os registos como operandos; no andar de EX e MEM, instruções que escrevessem em registos. A partir daí, implementámos em VHDL a comparação dos Opcodes das instruções destes estágios com os das Instruções identificadas, e a comparação dos operandos do andar de ID com os registos de destino dos andares de EX e MEM.

Para que isto fosse possível, tivemos ainda de propagar o sinal da Instrução para os andares de EX e MEM, já que no processador original este sinal só existia no andar de ID.

C. Conflitos de Controlo

Nesta arquitectura, uma vez que a unidade de Branch Control está no andar ID, a resolução dos conflitos de controlo acabou por ser relativamente simples: o endereço dado pelo

resultado do Branch está disponível no mesmo ciclo em que é que calculado e foi directamente ligado à entrada de endereço da Memória de Instruções, sem passar pelos registo de transição que atravessava no processador original.

Assim a unidade de Branch Control continha sempre o PC adequado, excepto nas Instruções de Branch Delayed. Neste caso, o endereço de salto (se o branch for taken) só deve ser tomado depois da instrução que se segue à instrução de Branch entrar no pipeline. Isto é, deixar a instrução a seguir ao BR delayed ser executada e só depois efectuar o salto. Nessas situações a saída da unidade de Branch Control contém o endereço que pretendemos 2 ciclos depois, o que significa que necessitamos de armazenar esse valor para poder efectuar o salto no ciclo devido.

Para resolver implementar esta funcionalidade, o sinal ID_NextPC foi propagado para o andar EX (como EX_NextPC), de modo a guardarmos o endereço de salto e foi adicionada uma lógica de selecção no andar IF, para que PC correcto da próxima instrução seja correctamente seleccionado.

D. Conflitos Estruturais

O único caso que despoleta conflitos estruturais nesta arquitectura é a ocorrência de dois Stores consecutivos, já que a memória é de escrita síncrona. Neste caso, a única forma de resolver este tipo de conflitos consiste na introdução de um Stall no pipeline. Para tal desactivamos o WE do registo de transição IF/ID durante um ciclo, para que a primeira operação de Store prossiga a sua execução, mas fazendo com que a segunda instrução "espere" durante dois ciclos no andar de IF.

IV. Comparação de performance

Sendo o processador original um processador multi-cycle, sabemos que necessitava de 5 ciclos de relógio para executar 1 instrução, tendo um período de 9.904ns (T_{mc}). O caminho crítico desta arquitectura situava-se no andar de EX, passando pelo multiplicador.

Após a implementação do funcionamento em pipeline, o periodo de relógio aumentou para 37.448ns (T_{pl}, o que corresponde a uma frequência de aproximadamente 27MHz). No entanto, há que ter em conta que o processador original demora 5 ciclos a executar uma instrução (5*9.904ns), ao passo que o processador modificado necessita de menos ciclos para computar uma instrução.

Para fazer uma análise mais concreta do speed-up alcançado, calculámos o CPI (Cycles per Instruction) do novo processador, usando para isso um dos programas disponibilizados na memória.

Uma vez que esta arquitectura não suporta a execução de uma operação de Store seguida imediatamente de uma operação de Load (situação em que é introduzido um Stall no pipeline), a frequência com que esta sequência de operações aparece no programa executado tem influência no número de Stalls que ocorrem, e por conseguinte no CPI médio da arquitectura, na execução daquele programa. Esta é a explicação para o facto de o número de instruções ser diferente (e inferior) ao número de ciclos, lembrando ainda que na fase inicial da execução, o pipeline está "vazio" e nessa altura não há uma instrução a terminar em cada ciclo, o que também faz com que o CPI médio não seja exactamente 1. Introduzindo um contador na simulação, foi possível calcular o CPI na execução deste programa (1.000678), que utilizámos para calcular o speed-up.

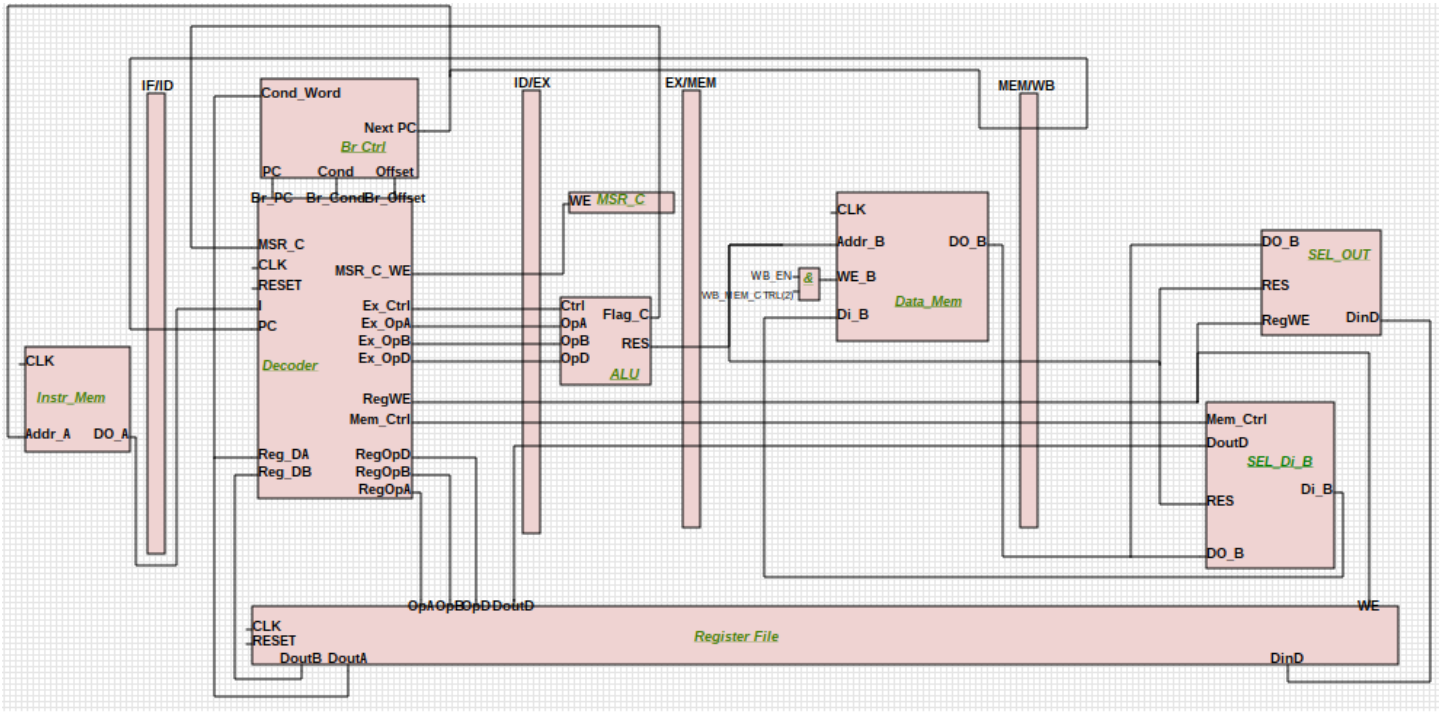
$$Speed_{up} = \frac{CPI_{mc} * T_{mc}}{CPI_{pl} * T_{pl}} = \frac{5 * 9.904ns}{1 * 37.448} \approx 1.322 \qquad (1)$$

Como se pode constatar, o novo processador é apenas aproximadamente 32.2% mais rápido que o multicycle. O objectivo deste trabalho não visava uma grande preocupação com a performance, não se pretendendo, contudo, descurá-la por completo. Nesse sentido, é importante explicar a razão pela qual o speed-up não é tão elevado como podia.

A principal razão é o facto de a unidade de Branch Control estar no andar de ID e de a memória ser endereçada directamente pela saída desta unidade. Isto conduz a um caminho crítico pesado, que explica o período de relógio tão alto. Em termos de implementação, foi muito mais simples deixar que o PC fosse directamente ditado pela unidade de Branch Control. Contudo, isso leva a que, num único ciclo de relógio, tenha que se esperar que o valor do sinal NextPC seja calculado para o ciclo seguinte. Introduzindo um registo intermédio entre a unidade de Branch Control e a porta de endereço da Memória de Instruções, reduziríamos bastante o caminho crítico, mas teríamos de implementar lógica adicional para a resolução de conflitos de controlo, que no caso actual não existem.

Posto isto, no próximo trabalho de laboratório, a nossa primeira prioridade será mudar optimizar o funcionamento da unidade de Branch Control, nomeadamente escolher o andar mais adequado para o seu posicionamento e implementar um funcionamento de Static Branch prediction, o que irá indubitavelmente melhorar o speed-up face ao obtido agora.

V. Apêndice I - Representação gráfica da arquitectura original (multicycle)



VI. Apêndice II - Representação gráfica da arquitectura final (pipeline)