Arquitecturas Avançadas de Computadores
Lab II - Desenvolvimento de um processador RISC pipelined

João Lages, 75286
Guilherme Pires, 75432

Este é o relatório do 2º Projecto da U.C. de Arquitecturas Avançadas de Computadores, no qual se pretendia que os alunos implementassem estratégias para melhorar a performance do processador desenvolvido no primeiro trabalho de laboratório.

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

I. Introdução

O objectivo deste trabalho consistia em analisar o trabalho realizado no Mini-Projecto anterior, identificando os pontos fracos dessa implementação, para posteriormente planear e realizar melhorias sobre os mesmos, com o intuito final de obter um Speed-Up apreciável.

II. Descrição do processador original e Identificação de pontos fracos

Passamos a descrever brevemente o funcionamento do processador original.

A. Andar IF

Neste andar a Memória de Instruções é endereçada e a sua saída, que contém a instrução a executar no ciclo seguinte, é guardada no registo IF/ID. O principal ponto fraco da arquitectura original residia no facto de a memória ser directamente endereçada pela Unidade de Controlo de Saltos (situada no andar ID), o que conduzia a um caminho crítico excessivamente pesado. Nesse sentido a primeira modificação implementada foi a introdução de um registo PC, entre a Unidade de Controlo de Saltos e a porta de endereços da Memória de Instruções e, numa fase inicial, não foi implementada qualquer predição de salto. Deste modo, tiveram de ser resolvidos vários conflitos de controlo, que nunca ocorriam na versão original do processador. Contudo, este "exercício"tornou mais fácil a transição para uma implementação com predição de saltos.

B. Andar ID

Este é, porventura, o andar mais complexo da arquitectura. Aqui são descodificadas as Instruções e gerados os sinais de controlo necessários para a execução dos andares seguintes. Os conflitos de dados são resolvidos neste andar, através de forwarding. Os conflitos estruturais são resolvidos através da introdução de um Stall no pipeline. Na versão original do processador não existiam conflitos de controlo, pelo que não existia lógica relativa à predição de saltos aqui, algo que foi adicionado na nova versão.

C. Andar EX

Os sinais de controlo e os operandos gerados pelo decoder são transmitidos para o andar EX, através do registo ID/EX. Nestes 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.

Esta unidade constitui o segundo grande ponto fraco da arquitectura original, já que as operações de multiplicação e barrel-shifting são pesadas, em termos de lógica. Isto levou-nos à segunda principal melhoria implementada: apatação do multiplicador e barrel shifter em pipeline.

D. 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.

E. 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 Melhorada: Modificações feitas

A. Multiplicador e Barrel Shifter em Pipeline

Tal como descrito, as operações realizadas pelo Multiplicador e pelo Barrel Shifter são pesadas em termos de processamento. Nesse sentido, uma vez que há um andar "inutilizado"quando estas instruções estão no pipeline - o andar MEM - é possível "distribuir"a sua execução pelos andares disponíveis, permitindo uma diminuição considerável no caminho crítico (assumindo que este é estipulado por estas unidades, o que só passou a ser verdade após a implementação da predição de saltos).

Em boa verdade, a concretização desta melhoria foi extremamente simples, devido à funcionalidade de Register Balancing fornecida pelo ISE. Assim, os passos necessários à implementação foram:

- Divisão da ALU original em 3 unidades: Multiplicador, Barrel Shifter e restantes operações;
- Implementação de um selector para o resultado do andar EX;
- Resolução de conflitos oriundos de forwarding de resultados do Multiplicador e Barrel Shifter (o resultado destas instruções passa a estar disponível apenas no andar de WB);
- Activação da funcionalidade de Register Balancing.

B. Dynamic Branch Prediction

Para a realização de predição dinâmica de saltos, optou-se por implementar um Branch-Target-Buffer com 2 bits de predição. Inicialmente, implementou-se uma unidade responsável por gerir o BTB, com as portas de input e output adequadas, de forma a que pudesse ser inserida na arquitectura com facilidade. Descrevem-se adiante estes dois passos para a implementação da predição dinâmica de saltos.

1) Branch-Target-Buffer

Esta unidade contém 16 registos de 32 bits. Cada registo contem uma Tag, um endereço de salto e 2 bits de predição. Os registos são endereçados através dos 4 primeiros Bits de um endereço de instrução. Descreve-se adiante o conteúdo dos registos:

- TAG: identifica o PC que escreveu no registo pela última vez
- JUMP_ADDR: identifica o endereço de salto da instrução a que o PC se refere
- PREDICTION BITS: bits de predição

O BTB é responsável por permitir a escrita e a leitura, na altura certa, dos registos que contém. Estes registos permitem à arquitectura prever qual a instrução seguinte mais provável. Sempre que o salto é efectivamente resolvido, a entrada correspondente na tabela é actualizada concordantemente, ou seja, os bits de predição são somados ou decrementados consoante o salto tiver sido Taken ou Not Taken.

2) Integração na arquitectura

Uma vez implementado o BTB em si, passou-se à integração do mesmo na arquitectura. O funcionamento do BTB é o que em seguida se descreve:

- No andar IF, o sinal ID_PC é lido, extraindo-se deste os bits correspondentes ao Index do BTB. Através do Index, averigua-se se o ID_PC actual já existe na tabela. Caso exista, o novo IF_PC passa a ser o JumpAddress correspondente, lido da respectiva entrada da tabela. Em caso contrário, o PC é incrementado e passado para o andar ID;
- No andar ID, o sinal ID_PC é comparado com o sinal Ex_NextPC. Caso estes sinais sejam iguais, conclui-se que a predição foi bem tomada, já que o sinal Ex_NextPC contém o resultado (correcto) do salto da instrução anterior. Quando uma instrução de controlo/branch chega ao andar EX, o BTB é actualizado da seguinte forma:
  - Se o index já existir na tabela, os prediction bits correspondentes são incrementados ou decrementados, consoante o salto tenha sido tomado ou não, respectivamente;
  - Se o index não existir, a entrada indexada da tabela é escrita com a TAG+Jump Address+Prediction Bits;

IV. Comparação de Performance e Conclusões

Para levarmos a cabo uma comparação de performance justa, implementámos ambas as arquitecturas na placa Spartan6 XC6SLX75, com um Design Goal de *Timing Performance* e uma Strategy de *Performance without IOB Packing*. Aquando da simulação, utilizou-se um contador para verificar

o número de ciclos de relógio que cada arquitectura levou a completar os programas de teste (*fibonnaci*, *randomize* e *sort-up*). Pode ver-se, na tabela adiante, os resultados de performance obtidos:

Arquitectura	**Período mínimo (ns)**	**Nº total de ciclos para executar os programas**
Original (lab1)	31.06	72531
Melhorada (lab2)	19.021	87369

Tal como esperado, conseguiu-se um descréscimo significativo no período mínimo da arquitectura (e, como tal, um aumento da frequência máxima). Contudo, devido à implementação da predição de saltos, os programas de teste acabam por levar mais ciclos de relógio a executar, uma vez que passam a haver instruções executadas indevidamente, que são posteriormente descartadas, o que não acontecia na arquitectura original. Ainda assim, a arquitectura modificada é efectivamente mais rápida que a arquitectura original, como pode observar-se pelo cálculo do speed-up, utilizando os resultados disponibilizados na tabela anterior. (Identificam-se com os subscritos *new* e *old* as variáveis relativas à arquitectura modificada e original, respectivamente).

$$Speed_{up} = \frac{\#Ciclos_{new} * T_{min_{new}}}{\#Ciclos_{old} * T_{min_{old}}} = 1.35 \rightarrow 35\%$$

V. APÊNDICE I - *Device Utilization Summary*

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                 1,566 out of  93,296    1%
    Number used as Flip Flops:               1,566
    Number used as Latches:                      0
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                      2,047 out of  46,648    4%
    Number used as logic:                    2,039 out of  46,648    4%
      Number using O6 output only:           1,950
      Number using O5 output only:               0
      Number using O5 and O6:                   89
      Number used as ROM:                        0
    Number used as Memory:                       0 out of  11,072    0%
    Number used exclusively as route-thrus:      8
      Number with same-slice register load:      8
      Number with same-slice carry load:         0
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                 1,084 out of  11,662    9%
  Number of MUXCYs used:                        76 out of  23,324    1%
  Number of LUT Flip Flop pairs used:        2,866
    Number with an unused Flip Flop:         1,309 out of   2,866   45%
    Number with an unused LUT:                 819 out of   2,866   28%
    Number of fully used LUT-FF pairs:         738 out of   2,866   25%
    Number of slice register sites lost
      to control set restrictions:               0 out of  93,296    0%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
  one Flip Flop within a slice.  A control set is a unique combination of
  clock, reset, set, and enable signals for a registered element.
  The Slice Logic Distribution report is not meaningful if the design is
  over-mapped for a non-slice resource or if Placement fails.

IO Utilization:
  Number of bonded IOBs:                        67 out of     328   20%

Specific Feature Utilization:
  Number of RAMB16BWERs:                        32 out of     172   18%
  Number of RAMB8BWERs:                          0 out of     344    0%
  Number of BUFIO2/BUFIO2_2CLKs:                 0 out of      32    0%
  Number of BUFIO2FB/BUFIO2FB_2CLKs:             0 out of      32    0%
  Number of BUFG/BUFGMUXs:                       1 out of      16    6%
    Number used as BUFGs:                        1
    Number used as BUFGMUX:                      0
  Number of DCM/DCM_CLKGENs:                     0 out of      12    0%
  Number of ILOGIC2/ISERDES2s:                   0 out of     442    0%
  Number of IODELAY2/IODRP2/IODRP2_MCBs:         0 out of     442    0%
  Number of OLOGIC2/OSERDES2s:                   0 out of     442    0%
  Number of BSCANs:                              0 out of       4    0%
  Number of BUFHs:                               0 out of     384    0%
  Number of BUFPLLs:                             0 out of       8    0%
  Number of BUFPLL_MCBs:                         0 out of       4    0%
  Number of DSP48A1s:                            4 out of     132    3%
```

Number of ICAPs:	0	out of	1	0%
Number of MCBs:	0	out of	4	0%
Number of PCILOGICSEs:	0	out of	2	0%
Number of PLL_ADVs:	0	out of	6	0%
Number of PMVs:	0	out of	1	0%
Number of STARTUPs:	0	out of	1	0%
Number of SUSPEND_SYNCs:	0	out of	1	0%