Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

DCC819 - Arquitetura de Computadores

Relatório V - Pipeline

Iuri Silva Castro
João Mateus de Freitas Veneroso
Ricardo Pagoto Marinho

Belo Horizonte - MG
3 de dezembro de 2017

1 Introdução

Este documento descreve a implementação do trabalho prático V da disciplina Organização de Computadores II. O trabalho visou incorporar uma *Pipeline* de três estágios sobre o caminho de dados implementado nos trabalhos anteriores. O processador de 16 bits finalizado também faz encaminhamento de dados.

2 Descrição

O *Pipeline* é uma técnica de *hardware* para promover paralelismo à nível de instrução dentro de um único processador. O objetivo da técnica é manter todas os módulos do processador ocupados com alguma instrução pelo máximo de tempo possível. Esse objetivo é realizado por meio da divisão das instruções em múltiplas etapas sequenciais, de forma que diferentes etapas de diferentes instruções possam ser executadas em paralelo como em uma linha de montagem. Essa técnica permite aumentar consideravelmente a velocidade do processador em comparação à execução puramente sequencial, pois várias tarefas podem ser executadas em um mesmo ciclo de *clock*.

No entanto, a técnica de *Pipelining* complexifica o controle do processador, uma vez que a execução paralela introduz *Hazards* no caminho de dados que não existiriam no caso da execução sequencial:

- *Hazards Estruturais* impõem restrições no número de instruções que podem utilizar um módulo do processador ao mesmo tempo. No caso do nosso processador, o caminho de dados possui uma única via, portanto existe apenas um módulo para executar cada etapa da *Pipeline*.
- *Hazards de Dados* forçam que uma instrução dependente de dados de instruções anteriores espere até que os resultados estejam disponíveis antes de ser executada. Caso não haja encaminhamento de dados, o processador é forçado a paralizar a execução de novas instruções por meio de *stalls* até que o dado esteja disponível. Nosso processador implementa o encaminhamento de dados da saída da unidade multiplicadora e da Unidade Lógica Aritmética para evitar o *stall*.
- *Hazards de Controle* acontecem quando existe um desvio de fluxo que altera o *Program Counter* e torna a próxima execução indefinida até que o processador avalie o novo valor do *Program Counter*. Nosso processador introduz um *stall* nos *branches* com o intuito de terminar a avaliação do *Program Counter* antes de prosseguir com a execução da próxima instrução.

O processador desenvolvido até o trabalho prático IV executava instruções de maneira sequencial. Com a introdução do *Pipeline* neste trabalho, obtivemos ganhos significativos na velocidade de execução como será mostrado na seção de experimentos.

3 Implementação

O processador finalizado conta com quatro módulos principais, além de uma série de módulos de controle secundários e multiplexadores. Os módulos principais são:

- *Decoder*: recebe a instrução de 16 bits e decodifica o *Opcode*, identificando os operandos e preparando os registradores que sinalizam se a instrução é uma multiplicação, se é um *Jump*, se haverá *Stall*, se haverá *Write Back*, qual registrador da multiplicação será armazenado se for o caso e se o segundo operando é um imediato.
- *Register Bank*: o banco de registradores conta com 16 registradores de 16 bits, duas portas de leituras para os operandos A e B e uma porta de escrita.

- *Unidade Lógica Aritmética*: a unidade lógica aritmética executa as instruções: ADD, SUB, SLT, AND, OR, XOR e BEZ.
- *Unidade multiplicadora*: a unidade multiplicadora recebe dois operandos de 16-bits e executa uma multiplicação produzindo um resultado de 32-bits que é armazenado em dois registradores: HI, que armazena os 16-bits mais significativos e LO, que armazena os 16-bits menos significativos. O resultado da operação armazenado nos registradores pode ser acessado por meio das instruções GHI e GLO, que escrevem o conteúdo dos registradores HI e LO, respectivamente, em um registrador do banco de registradores.

Instrução	Opcode	Bits 11-8	Bits 7-4	Bits 3-0	Descrição
ADD	0000	C	A	B	Reg(C) = Reg(A) + Reg(B)
SUB	0001	C	A	B	Reg(C) = Reg(A) - Reg(B)
SLTI	0010	C	A	Imm	Reg(C) = Reg(A) > Imm
AND	0011	C	A	B	Reg(C) = Reg(A) AND Reg(B)
OR	0100	C	A	B	Reg(C) = Reg(A) OR Reg(B)
XOR	0101	C	A	B	Reg(C) = Reg(A) XOR Reg(B)
ANDI	0110	C	A	Imm	Reg(C) = Reg(A) + Imm
ORI	0111	C	A	Imm	Reg(C) = Reg(A) OR Imm
XORI	1000	C	A	Imm	Reg(C) = Reg(A) XOR Imm
ADDI	1001	C	A	Imm	Reg(C) = Reg(A) + Imm
SUBI	1010	C	A	Imm	Reg(C) = Reg(A) - Imm
J	1011	Imm			PC = Imm
BEZ	1100	-	A	B	If (Reg(A) = 0) PC = Reg(B)
MUL	1101	C	A	B	Reg(C) = Reg(A) * Reg(B)
GHI	1110	C	-	-	Reg(C) = HI
GLO	1111	C	-	-	Reg(C) = LO

Tabela 1: Instruções

O conjunto de instruções final do processador está descrito na tabela 1. Todas as instruções aritméticas e lógicas recebem o operando A do banco de registradores e o operando B pode ser um imediato de 4-bits ou um registrador dependendo da instrução. A instrução BEZ não utiliza os bits 11-8 e as instruções GHI e GLO não utilizam os bits de 7-0. A instrução J altera o PC para um valor imediato de 12-bits que comporta qualquer endereço da memória de 4096 posições.

4 Integração

Mostrar divisão de estágios. Mostrar buffers entre estágios. Mostrar sistema inteiro Adicionar diagrama da pipeline. Adicionar diagrama da estrutura do processador.

5 Simulação e Testes

Descrever processo de simulação e programas de teste. Comparar programas rodando com pipeline e sem (trabalho IV anterior).

6 Discussões

Comentar dos ganhos de desempenho, custos dos *Stalls*, dificuldades de mudança para pipeline.