Trabalho Prático 2 - Organização de Computadores 1

Guilherme de Abreu Lima Buitrago Miranda - 2018054788

Novembro - 2019

1 Problem 1: 5-Stage Pipeline

A tabela abaixo demonstra como se comporta o pipeline com as instruções do loop.

	1	2	3	4	5	6	7	8	9	10	11
lw	IF	ID	EX	MEM	WB						
addi		IF	ID	-	EX	MEM	WB				
addi			IF	-	ID	EX	MEM	WB			
addi					IF	ID	EX	MEM	WB		
blt						IF	ID	EX	MEM	WB	
lw							-	-	IF	ID	EX

Para encontrar quantas instruções são feitas durante o loop, devemos observar o comportamento do pipeline entre a 1ª e a última instrução lw. Vê-se que existem 4 instruções e que, se este fosse um pipeline perfeito, seriam gastos apenas cinco ciclos entre o início das duas instruções de load. Contudo, como enfrentamos dois problemas de stall, precisamos incrementar a quantidade de ciclos necessários, sendo um a mais pelo primeiro lw e outros dois pelo blt. Assim, temos que a quantidade de ciclos gastos no loop é de $(5 + 1 + 2) = 8$, conforme claramente observado na tabela, pois o último lw começa no 9º ciclo e o primeiro lw, no 1º (9 - 1 = 8).

2 Problem 2 - Study Question

What are the five stages of a typical RISC CPU pipeline? What are pipeline hazards, and how do they happen? Give examples of methods to resolve hazards.

Os cinco estágios de uma CPU t1ípica de pipelines RISC são:

- Instruction Fetch Nesse estágio a CPU lê as instruções da memória de acordo com o endereço presente no PC (Program Counter)
- Instruction Decode Nesse estágio a instrução lida no estágio anterior é decodificada e o banco de registradores é acessado para buscar os valores a serem usados na instrução
- Instruction Execute Estágio em que as operações na ALU são feitas
- Memory Access Nesse estágio as operações de memória são feitas, como leitura e escrita, a variar de acordo com a instrução
- Write Back Nesse estágio o valor calculado ou buscado é escrito no registrador de destino da instrução.

Hazards (perigos) são situações em que não se pode começar a próxima instrução no próximo ciclo. Para que isso aconteça, tem-se um dos três cenários descritos abaixo:

- Hazard Estrutural, que acontece quando um recurso está ocupado. Um exemplo é quando duas instruções tentam acessar a memória ao mesmo tempo, uma para ler a instrução e outra para fazer o load ou store de dados. Para resolver tal problema, pipelined datapaths separam as memórias de dados e de instrução.
- Hazard de Dado, que ocorre quando uma instrução depende da finalização do dado de uma instrução anterior. Um exemplo disso é quando uma instrução vai colocar uma soma no registrador x19 mas a instrução imediatamente posterior coloca em x2 a subtração de x19 e x3. Para resolver tal problema, pode-se usar o encaminhamento, que disponibiliza o dado para a próxima instrução tão logo ele seja calculado, assim, o mesmo pode ser usado antes mesmo de ser escrito no registrador. Essa estratégia requer novas conexões no datapath, mas possibilita que

os dados sejam utilizados após o terceiro estágio para instruções do tipo R ou após o quarto para loads. Para o cenário dos loads, um stall é enfrentado, mas as vezes é possível reorganizar as instruções para evita-los, afim de agilizar as operações a serem feitas.

- Hazard de Controle, que acontece quando a decisão de ação de controle depende de instrução anterior, como o exemplo dos branches, em que vai para diferentes instruções caso seja ou não tomado. Para resolver o problema, pode-se ser usado o método de previsão, em que tenta-se prever qual será o resultado do branch. Se a previsão for acertada, não ocorro o stall no pipeline e a execução segue como esperado. Caso contrário, temos a presença de stalls no pipeline.

3 Problem 3: Exercise 4.22 reproduced from the textbook

3.1

Com o pipeline tendo apenas uma memória, teriamos:

	1	2	3	4	5	6	7	8	9	10	11	12
sd	IF	ID	EX	MEM	WB							
ld		IF	ID	EX	MEM	WB						
sub			IF	ID	EX	MEM	WB					
beqz				-	-	IF	ID	EX	MEM	WB		
add							IF	ID	EX	MEM	WB	
sub								IF	ID	EX	MEM	WB

3.2

De maneira geral, a alteração não seria efetiva. Nesse cenário, reorganizar as instruções apenas causaria uma permutação entre quais pares enfrentariam um conflito, não sendo relevante para o contexto geral do pipeline.

Já no que se refere às instruções de memória (load e store), como não existe dependência para com as demais, colocá-las no fim do código seria benéfico, pois evitaria stalls causadas pelas mesmas. Assim, teriamos:

	1	2	3	4	5	6	7	8	9	10
sub	IF	ID	EX	MEM	WB					
beqz		IF	ID	EX	MEM	WB				
add			IF	ID	EX	MEM	WB			
sub				IF	ID	EX	MEM	WB		
sd					IF	ID	EX	MEM	WB	
ld						IF	ID	EX	MEM	WB

3.3

De maneira geral, não. Isso acontece pois até mesmo NOPs precisam ser lidos da memória, o que causaria o mesmo problema de Hazard estrutural das outras instruções.

4 Referências

1. Omar P. V. Neto. Aula 8 – O processador – Pipeline. `https://virtual.ufmg.br/20192/pluginfile.php/424507/mod_resource/content/1/Aula08_nova.pdf`, 2019

2. Saurabh Sharma – Computer Organization and Architecture — Pipelining. `https://www.geeksforgeeks.org/computer-organization-and-architecture-pipelining-set-1-execution-stages-and-throughput/`, 2017