Лаб. Упражнение No: 4

Дисциплина: Компютърни Архитектури

Асистент: Иван Янчев

Тема: Зависимости между инструкциите

========================================================================

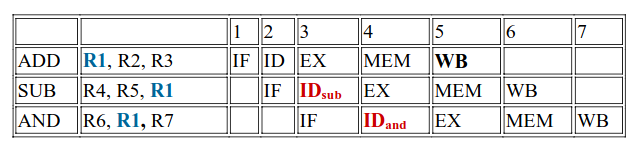
1. **Теория:**В предходното упражнение беше разгледана конвейерна реализация на процесор(pipeline). При него целта е да се изпълняват множество на брой независими инструкции едновременно. Това не винаги е възможно и зависиТ от изпълняваната програма.

**Типове зависимости:**

**1.1 Ресурсни(Структурни) зависимости:** Те са по-често проблем при суперскаларните реализации, които все още не са били разгледани. Настъпват когато две или повече инструкции искат да използват един ѝ същи хардуерен ресурс едновременно (шини, функционални у-ва, памети, регистри и т.н.)

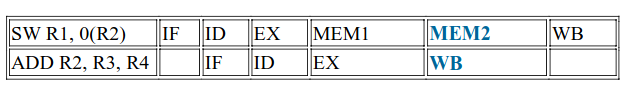
**1.2 Даннови зависимости:** Те настъпват две или повече последващи инструкции използват един и същ операнд, чиято промяна трябва да бъде отчетена. Данновите зависимости са три типа:

Read After Write(RAW**) -**



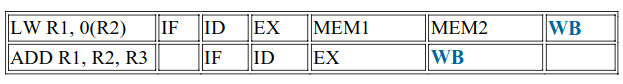
Тук инструкцията **ADD** записва в **R1** резултата от сбора на **R2** и **R3**. Инструкцията **SUB** очаква да използва този сбор като втори операнд, обаче той още не е записан. Записването става във фаза Writeback, резултата от събирането се получава във фаза Execute, а четенето на операндите – във фаза Decode. Вижда се, че за ADD инструкцията фаза Writeback е на 5-тия такт, което означава, че за коректно изпълнение на SUB инструкцията, Decode фазата ѝ трябва да е поне на 5-ия такт (Поради спецификата на регистровия файл - писане в първата половина на такта и четене във втората). Тук фазите Decode на SUB и на AND инструкциите са на трети и четвърти такт, което означава, че ще прочетат старата стойност, преди новата да се е записала. За решаване на този проблем трябва да се поместят две празни инструкции (NOP), между ADD и SUB. Тяхната задача е да осигурят двата нужни такта за коректното изпълнение.

Write After Read(WAR**) -**



Подобно на RAW, но тук първо е четенето, след това е записът. Тук записването в паметта отнема два такта и се получава проблем. Операндът е R2.Това е само пример, кога може да възникне проблем.

Write After Write(WAW**) -**



Когато имаме запис след който следва запис (Операндът е R1).

Голяма част от данновите зависимости се преодоляват с т.н. **Forwarding**(стр 313), при който изхода на ALU-то и паметта се свързват с входовете на ALU-то.

От разглежданите три даннови зависимости, WAR и WAW са “фалшиви зависимости(false dependancies)”, докато RAW е “истинска зависимост(true dependancy)”. Това е защото фалшивите зависимости могат изцяло да се отстранят с концепцията **register renaming.**

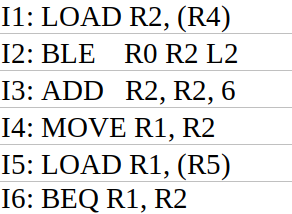
**1.3 Процедурни зависимости:** Те сa основно свързани с условните преходи, понеже не се знае дали преходът ще бъде взет или не, няма как да се извлече следващата инструкция.

Тези зависимости се предотвратяват с ефективен **branch predictor**,ще бъде взет следващия част, или с разместване на инструкциите, като след инструкцията за преход се поставят допълнително независими от скока инструкции, често взети от кода, който е преди скока (стр. 326).

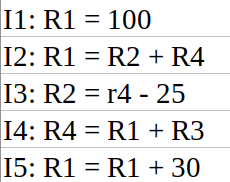
При откриване на зависимост се активира hazard detection unit-а, който променя всички контролни сигнали на 0-ли (забранява всякакво писане) и забранява писане в PC, за да не се обнови с нова инструкция. Това е методът за добавяне на празни инструкции (NOP).

1. **За протокола**Намерете зависимостите в :

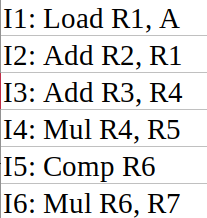
**1)**



**2)**



**3)**



Учебник стр.374 - 4.18, 4.19, 4.20, 4.22, 4.23, 4.24, 4.25, 4.27

1. **Линкове**

**Упражнения**: [https://github.com/tu-iyan/Computer-Architecture-2024](./упражнения)

**Учебник:** [http://library.lol/main/373C67B0C5E22C9B92B1D8FACDC47E68](./учебник)