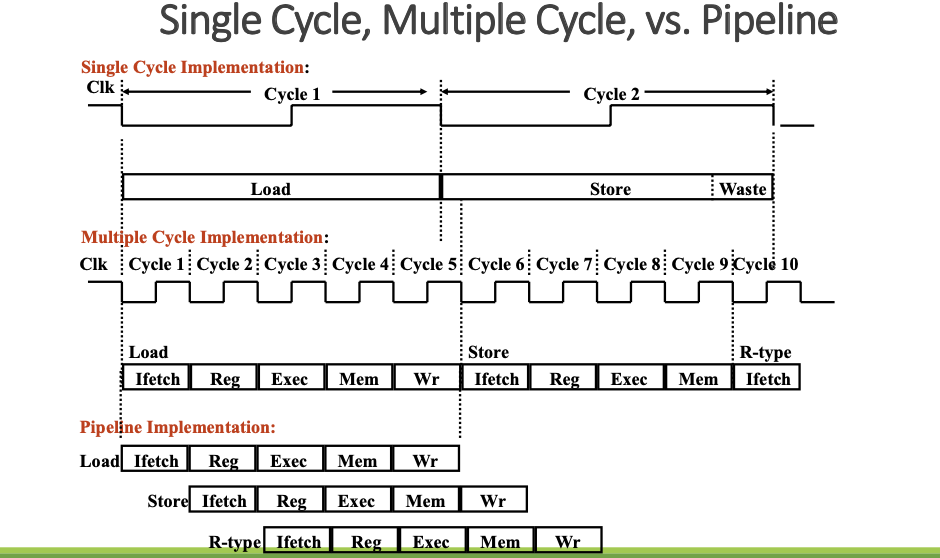
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Single Cycle | Multiple Cycle | | Pipeline |
| Cyecle time | 以instruciotn最長執行時間為cycle time | 將指令切割為五個步驟 （If,Reg,Exec,Mem,Wr）  以最長的步驟時長為Cycle time | | |
| Resource使用 | 在一個cycle內使用多個resource，ˇ但同一時間只使用一個resource | 每個cycle只使用一個resource。 | 每個cycle可能有數個resource被同時使用，但皆由不同的instruction使用。  （一個instructionu一次只能使用一個resource） | |
|  | Cycl time過長  Resource idle時間較長 | 不一定能比singlecycle省時  Resource idle時間較長 | 有效使用resource  ＃提高整個workload的throughput但對單一task的latency並沒有幫助。 | |
| 缺點 |  |  | 會有pipeline hazard | |



Pipeline 的Hazard

Hazard:pipeline中不能順利的在下一週期執行下一指令。

不論是什麼hazard都可以使用stall（暫停一個cycle）來避免

1. Structural hazard:

原因：同時使用相同的resource

解決：1.增加硬體資源 2. Stall

1. Data hazard:

原因：需要用到前一指令尚未產生的結果，也就是所需資料還沒拿到

解決：1.插入ＮＯＰ（bubble）2. 重新編排順序

3.如果不是load-use就可以使用forwarding

＃load-use：在load指令後，下個指令的來源暫存器與load

指令的暫存器相同

#forwarding:在所需資料在前面的指令產生結果時就送過來，

不需等到WB。（需增加硬體（線）來送資料）

3. Control hazard:

原因：branch指令尚未決定好是否跳躍時，後續指令就進入pipeline

解決：1.在ID時就決定是否跳，而當在ID時，下一指令已進入IF階

段，我們就可以決定是否flush掉它。

2.延遲分支：將不論是否branch都會用的指令插到branch指令

後

＃static branch prediction

一律假設branch不發生

＃dynamic branch prediction

使用branch history table(BHT)（有一位元來記錄branch最近是否發生）還有branch target buffer(BTB)（作為cache來存放目的地PC或目的地指令來消除計算分之目的位址的penalty(1 cycle)）

Advanced pipeline:

提高ILP:

1. 增加pipeline深度，讓更多指令進入pipeline（superpipeline）
2. Multiple issue:pipeline中每個stage執行多個指令

＃loop unrolling:將迴圈展開，複製迴圈部分的程式碼，可將不同的loop body加入排程。

＃動態多重分發處理器(super scalar):

指令照順序分發，並且由處理器決定每個週期分發多少指令，試著避免hazard和stall下找出指令在同一週期執行。