Final_project _MIPS_CPU

DCS121_109511314_鄭旭恩

1. MIPS_CPU_Introduction

MIPS（Microprocessor without Interlocked Pipeline Stages）架構是一種經典的精簡指令集（RISC）架構，具有高效、簡潔和易於實現的特點。以下是對 MIPS 架構 CPU 的詳細介紹：

MIPS 指令集結構

MIPS 架構最初由於 1981 年在加州大學柏克萊分校開發。它的設計遵循精簡指令集計算機（RISC）的設計理念，目標是將指令的執行時間降低到最小，通過精簡指令集和高度優化的硬件實現來實現高性能和高效能。MIPS 指令集被設計成固定大小的 32 位元長度，大多數指令在執行時具有相同的執行時間。主要的指令類型包括：

- **算術運算和邏輯操作**：包括加法、減法、乘法、除法等基本算術操作。
- **數據轉移**：用於將數據從存儲器讀取到寄存器或將數據從寄存器寫入存儲器。
- **控制流**：支持條件分支、無條件分支和跳轉指令，例如分支相等（beq）、無條件跳轉（j）、加載字（lw）和存儲字（sw）等。

MIPS 架構特點

1. **固定長度指令**：所有指令均為 32 位元，簡化了硬件設計和指令解碼。
2. **單一週期執行**：大多數指令在單個時脈週期內完成執行，有助於實現高速執行。
3. **五級流水線**：典型的 MIPS 處理器具有五級流水線，包括指令提取(IF)、指令解碼(ID)、執行(EX)、存儲器訪問(MEM)和寫回(WB)階段。
4. **通用寄存器**：MIPS 架構有 32 個通用寄存器（$0 - $31），所有算術運算和數據操作都是在這些寄存器之間進行的。

Hazard:

1. **Structural hazard:** 這是指某個指令由於硬體資源無法同時提供所需的組合指令，導致該指令無法在預定的時鐘周期內執行。這種風險通常可以在設計階段提前避免，以確保硬體資源的合理分配。
2. **Data Hazard**：當一個指令需要的數據在執行時尚未準備好，導致該指令無法在正確的時鐘周期內執行。這種風險是動態產生的，需要使用 Data forwarding 等方式來解決。
3. **Control hazard**：也稱為 branch hazard。當 pipeline 中的一個指令的執行依賴於前面的分支指令結果，而該分支指令的執行方向未知時，可能導致流水線停滯等待分支結果。這是一種動態產生的風險，可以通過 branch prediction 來減少其對性能的影響。

Solving Method:

4. **Pipeline stall**：也稱為“Bubble”。這是一種為了解決”Hazard”而啟動的停滯機制。通過暫停流水線中的指令執行，以等待所需資源或數據的到來，從而解決指令間的衝突。
5. **Forwarding:** 這是一種通過從內部資料送到下一個指令進而解決 Data hazard 的方法，而不是等待數據從可見的寄存器或內存中到達。這可以在不引入停滯的情況下，直接將上一個指令計算出的結果轉發給需要該數據的後續指令，提高流水線的效率。
6. **Branch prediction**：這是一種通過假設分支的一個給定結果來解決 Branch hazard 的方法，並基於這個假設繼續執行，而不是等待確定實際的分支結果。通過預測分支的方向，減少流水線因等待分支結果而引起的停滯，從而提高指令執行效率。

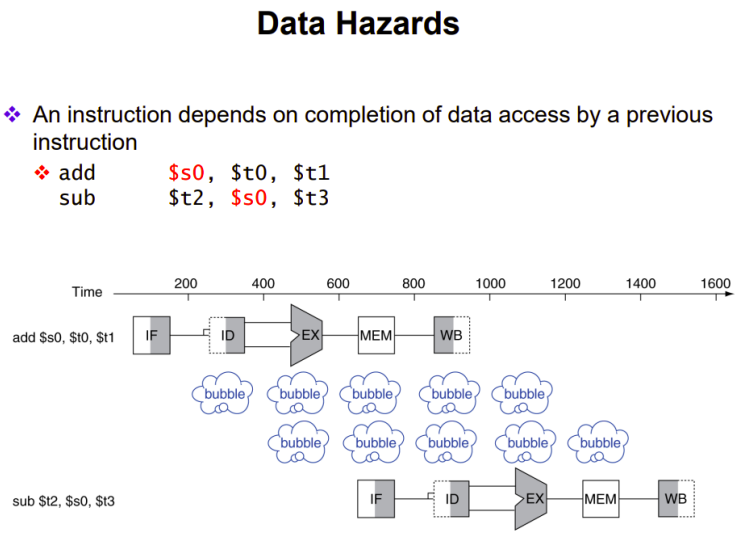
2. Problem finding:

以下為再實作 pipeline MIP 中遇到的幾個問題，包括衝突(Hazard)以及解決方式。以下舉例原先的第四個指令: 4. `0011402A`，經過解析後為: slt $8, $0, $17`。然而接著的下一個指令為: 5. `1100000C`，解析後為:`beq $8, $0, $12`。

這類指令在這次大量出現，因為本身是做 bubble sort 的功能。這些指令會同時遇到兩種 hazard，分別是 Data hazard 以及 Branch hazard。在這次專題中，使用不同方法去解 hazard 並且加以比較。

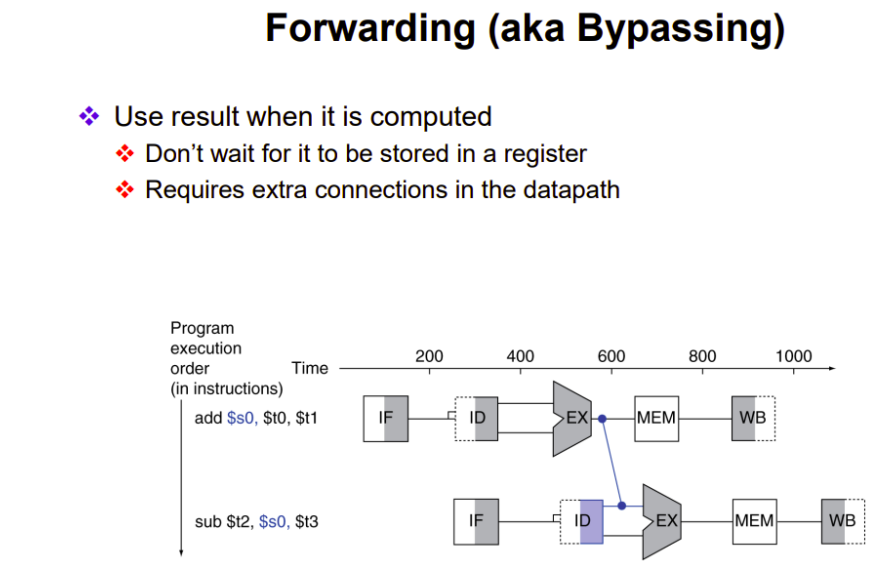
Insert bubble:

首先的做法是在各個會出現 hazard 的地方後方加入 bubble。這可以很簡單的避免 pipeline 的延遲特性。使下一個指令要存取寄存器的值的時候，上一個指令已經將正確的值寫回寄存器了。這個做法實現了 pipeline CPU。然而，這個也衍生一個很大的問題，也就是大量 instruction 數量爆增，使得 cycle 也跟著暴增。如下圖解法。



Data forwarding:

Data hazard 的部分是在 slt 判斷寄存器 rt[$0]的大小是否小於寄存器 rd[$17]的值，接著再將 1 或 0 寫入寄存器 rs[$8]的時候，下一個指令 beq 需要存取寄存器 rs[$8]的值。在指令還未寫回寄存器的時候，下一個指令已經讀取到尚未改變的值，這個就是所謂的 Data hazard。這會造成所有後續的 bubble sorting 都出錯。因此在這裡我們使用 data forwarding ，在 EXE 階段計算完的當下就把值送到到下一個指令 beq 使其可以讀取到正確的值。如下圖所示。




```
always @(*) begin
    if(IF_ID_instr[31:26]=='b000100)begin //現在beq
        if ((ID_EX_instr[31:26]=='b000000)&&(ID_EX_instr[5:0]=='b101010)) begin // 前一個slt
            forwardbeq = 1;
            forwardrs=(reg_file[ID_EX_rs] < reg_file[ID_EX_rt]) ? 1 : 0;
```

Data forwarding 解決 slt 及 beq 之間的 data hazard 問題。

Branch prediction:

接著為 control hazard 也叫做 branch hazard。在前面我們為了讓所有 instruction pipeline 的程度皆為五個 cycle。因此我們也將 beq 原先只有三個 cycle 就可做完的指令在後面塞入兩個 nop 延長到五個 cycle。但這也就意味著每個指令跳轉之前會多兩個錯誤的指令，導致一步錯步步錯。因此首先的解決方法為在 beq 後面無論如何都先加入兩個空的 bubble。但這也導致 instruction 的數量大增加。導致 cycle time 大量增加。接著我將部分改成在第 IF_ID 這個 pipeline 的時候就就做 branch prediction，接著直接跳轉，等於只需要在後方插入一 bubble 即可。大量減少 cycle。如下段的比較所示，降低了 3000~4000 個 cycle。這也是我認為最需要先處理的 hazard 部分。因為在指令中大量的在這幾個指令間跳動。因此解決這個便可以減低很多的 cycle。這部分的解決方法如下圖。

Control hazard

- ***Untaken branch***
  - One that falls through to the successive instruction. A taken branch is one that causes transfer to the branch target

- ***Solutions: Branch prediction***
  - A method of resolving a branch hazard that assumes a given outcome for the branch, and proceeds from that assumption rather than waiting to ascertain the actual outcome

```
else if(IF_ID_instr[31:26]==6'b00100)begin
    if(forwardbeq==1)
        pc <= (((forwardrs - reg_file[IF_ID_instr[20:16]]) == 0) ? pc + 1 -1 + {{16{IF_ID_instr[15]}},IF_ID_instr[15:0]} : pc + 1);
    else
        pc<=pc+1;
```

Branch prediction 解決 beq 跳轉延遲的 control hazard 問題

Analysis of different ways to solve hazard:

下列為不使用不同方式去解 Data hazard 所得出的數據比較。在左下側圖是使用在遇到各個不同的 hazard 的時候，在後方加入兩個 bubble 使得其可以不被 pipeline 的延遲特性影響。不過缺點便是 cycle 暴增，因為多了許多無用的 bubble 指令。接著為了解決這個問題，我先從波型圖中尋找在那些指令間出現最多的無用指令(bubble)，發現大量的 slt 指令接著 beq 指令。而這些造成許多的無用指令。因此嘗試用 data forwarding 以及 branch prediction 如上述 2.Problem finding 的敘述。成功的壓下 cycle 至 7318。而且同時面積並沒有增加太多。

```
[PASS] DCS121 03_GATE
 Account          DCS121
 Error_Message    No_Error
 01_RTL           0
 02_SYN           0
 03_GATE          0
 Cycle_Time       6
 Latency          11413.0
 Syn_Area         171817.128962
 Submiss_Date     2024/06/19
 Submiss_Time     20:08:14
```

Add bubble

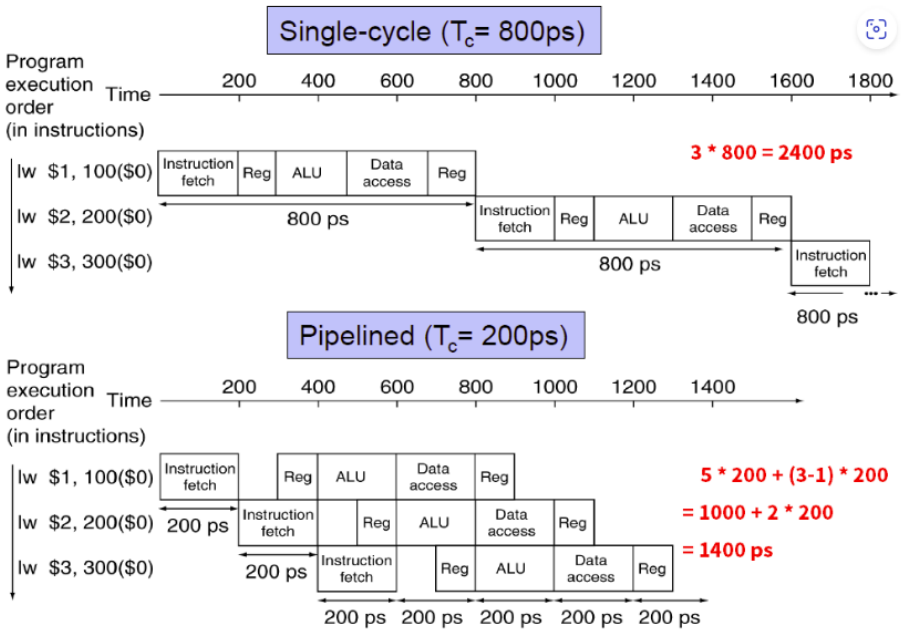
```
[PASS] DCS121 03_GATE
 Account          DCS121
 Error_Message    No_Error
 01_RTL           0
 02_SYN           0
 03_GATE          0
 Cycle_Time       4.5
 Latency          7318.0
 Syn_Area         185272.517652
 Submiss_Date     2024/06/21
 Submiss_Time     15:38:32
```

with data forwarding and branch prediction

	Add bubble	**Data forwarding & branch prediction**
Cycle time	6	4.5
Cycle	11413	7318
Area	171817	185272
Performance	11,765,684,526	6,101,192,232

3. Comparison with HW04

Performance between Single-Cycle and Pipeline



HW04 Single Cycle MIPS:

在 HW04 中是使用 single cycle 並且所以指令在單一個 cycle 中做完。也因為在同一個 cycle 中要執行得更多，導致 cycle time 的有個下限。再往下給有可能會出現 timing violation。不過相較於 pipeline MIPS 架構的好處就是面積以及 cycle 可以壓到最小，因為每個指令皆在一個 cycle 中便可以完成，也不需要而外的硬體資源來做而外的 data forwarding 等等。

Final Pipeline MIPS:

在 pipeline MIPS 架構中。將同一個指令拆成五個部分，因此每個 cycle 所需要的時間相較於 single cycle MIPS 可以往下壓。不過相對的 pipeline 要去解決不同指令之間所出現的 hazard。因此需要犧牲而外的硬體資源去做 forwarding 等等。加上會出現部分的 bubble 也會增加 cycle 數量。

Analysis:

在 HW04 中，我們採用了 Single Cycle MIPS 架構，其中所有指令在單一個時鐘週期內完成。這意味著每個指令的執行時間等於 cycle time 的長度，導致 cycle time 無法縮短。這種設計簡化了電路結構，減少了硬體資源需求，但也可能導致了高頻時脈的 timing violation。

相比之下，我們在最終設計中使用了 Pipeline MIPS 架構，將每條指令拆分為多個階段，每個階段在單獨的 cycle time 內完成。這樣，每個指令的執行時間可以分散到多個 cycle 中，從而減少了 cycle time 的長度。Pipeline 架構提高了指令吞吐量和整體性能，但需要額外的硬體資源來處理 data forwarding 和 control hazard 問題，還有部分的 bubble，影響指令的執行效率。

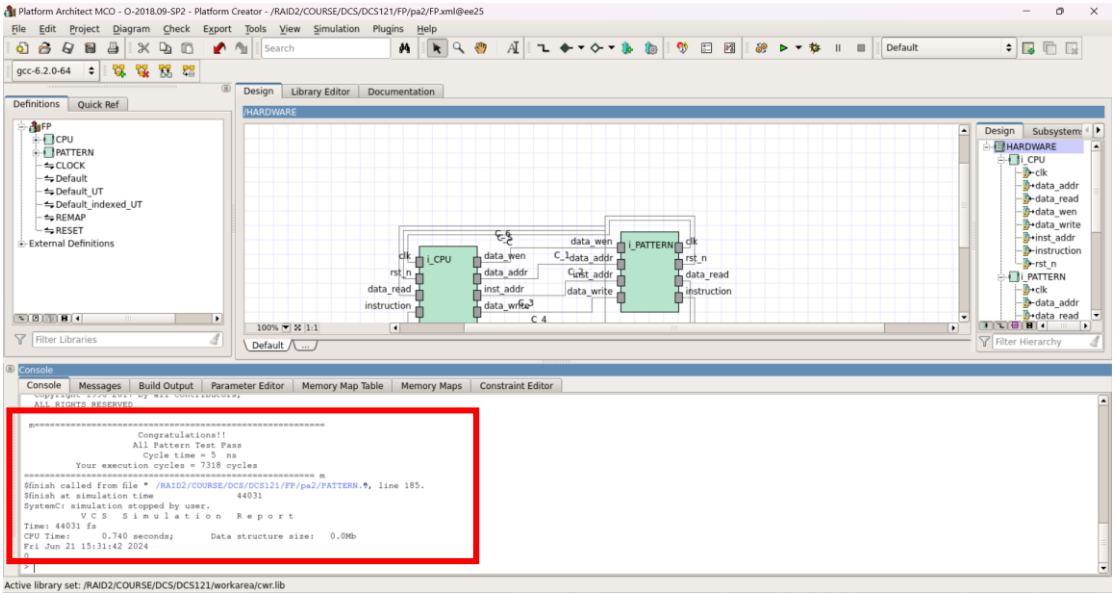
比較這兩種架構，Single Cycle MIPS 架構結構簡單，硬體資源需求少，但 cycle time 長，且各個指令所需時間不同，可能會因為少數需要較長處理時間的指令而使整體 cycle time 都需要增加，對處理時間相對短的指令反而出現許多空等待時間，或是有可能導致在高頻下的 timing violation。Pipeline MIPS 架構則有更高的指令吞吐量和性能，但設計複雜，需要處理設計處理 hazard 的架構以及額

外的硬體資源。總結來說，Single Cycle MIPS 適合簡單系統，而 Pipeline MIPS 適合高時脈需要高效能的應用。最終設計選擇了 Pipeline MIPS 架構，並針對 hazard 進行了優化，最終達到了較好的效果。

	HW4	**Data forwarding & branch prediction**
Cycle time	10	4.5
Cycle	5142	7318
Area	138438	185272
Performance	7,118,481,960	6,101,192,232

4. PA

下圖為 platform architecture 的實作。



5. REFERENCE

- Lecture10_Pipelined MIPS (2).pdf
- Pipelining – MIPS Implementation – Computer Architecture (umd.edu)
- Ch.4-3 Pipeline Processor - HackMD
- iT 邦幫忙::一起幫忙解決難題，拯救 IT 人的一天 (ithome.com.tw)