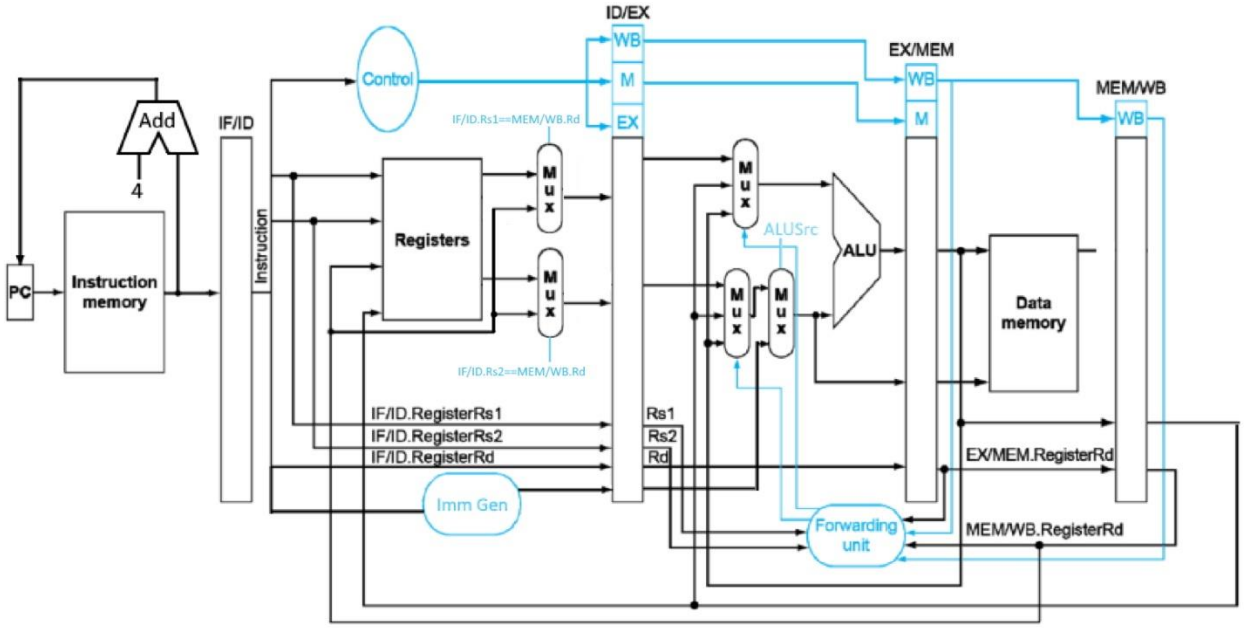
Computer Organization

0716214 江岳勳

0716222 黃偉傑

Architecture diagram:



Detailed description of the implementation:

由於這次 Lab 承接 Lab4，因此我們將只會提及有所更動的實作內容。我們在確認這次的 Lab 的需求後，決定將許多與 load, store, branch 等指令相關的物件拔除，使我們的架構簡化。這次 Lab 相較於 Lab4，新增的 module 有：Pipeline Register (IF_ID, ID_EX, EX_MEM, MEM_WB), FowardingUnit。以下我們將從三個角度來說明我們的實作，分別是 Pipeline Register、FowardingUnit、以及 Pipeline_CPU(Top Module)。

- Pipeline Register

我們是將每個 Pipeline Register 各自組成一個 module，並且控制其在 clock 正緣觸發時寫入。在省略系統必要訊號與後，剩下的各個 Port 如下（各有其輸出 / 入）：

IF_ID	Instruction
ID_EX	RegWrite, ALU_Src, ALU_Ctrl, RS_Data, RT_Data, Imm, RS_no, RT_no, RD
EX_MEM	RegWrite, ALU_Result, RD
MEM_WB	RegWrite, ALU_Result, RD

其中與 Slides 不同之處，除了將用不到的 Port 拔除以外，由於 Instruction 被我們從 ID_EX 與其後面的 Pipeline Register 中拔除，因此我們也同時將 ALU_Ctrl module 從 EX stage 往前挪至 ID stage，連帶所影響的，是原先 ID_EX 中應該要連接的 ALUOp 改成了 ALU_Ctrl。

- FowardingUnit

這個 module 的邏輯與 Slides 完全相符，實作上我們則利用了三元條件運算子，如圖。

```
// 0 : original signal, 1 : EX/MEM, 2 : MEM/WB

assign SRC1 = (ID_EX__RS1==EX_MEM__RD && EX_MEM__REG_WRITE) ? 1 : (
              (ID_EX__RS1==MEM_WB__RD && MEM_WB__REG_WRITE) ? 2 :
                                                              0   );

assign SRC2 = (ID_EX__RS2==EX_MEM__RD && EX_MEM__REG_WRITE) ? 1 : (
              (ID_EX__RS2==MEM_WB__RD && MEM_WB__REG_WRITE) ? 2 :
                                                              0   );
```

- Pipeline_CPU (Top Module)

1. 我們將上述的 Pipeline Register 與 Forwarding Unit 加入 Module 之後，便按照我們所學過的 5-stage 順序將所有 submodule 按照順序排列，並且宣告了許多 wire 來滿足新的 module 的 port。

2. 在 Forwarding Unit 附近，我們模仿了 Slides 上在 ALU 前面接了兩個 3-to-1 MUX 的作法，而原先用來選擇 RT_Data 與 Imm 的 2-to-1 MUX 被放在了第二個 3-to-1 MUX 的後方。

3. 在 RegFile 與 ID/EX 中間，我們接上了兩個 2-to-1 MUX，用來 forward 從 WB stage 寫回來的資料，其判斷條件是 WB stage 的 RegWrite 為 1 且其 RD 與 ID stage 的 RS 或 RT 相等，如圖。

```
MUX_2to1 RS_ICCTRFSIPTH(
        .data0_i(RSdata_o),
        .data1_i(ALUresult_WB),
        .select_i((RegWrite_WB == 1'b1 && RD_WB == INSTR_ID[19:15])),
        .data_o(RSdata_o_ICCTRFSIPTH)
);

MUX_2to1 RT_ICCTRFSIPTH(
        .data0_i(RTdata_o),
        .data1_i(ALUresult_WB),
        .select_i((RegWrite_WB == 1'b1 && RD_WB == INSTR_ID[24:20])),
        .data_o(RTdata_o_ICCTRFSIPTH)
);
```

Implementation results:

CO_Data_1.txt

```
# PC =       136
# Data Memory =         0,         0,         0,         0,         0,         0,         0,         0
# Data Memory =         0,         0,         0,         0,         0,         0,         0,         0
# Data Memory =         0,         0,         0,         0,         0,         0,         0,         0
# Data Memory =         0,         0,         0,         0,         0,         0,         0,         0
# Registers
# R0 =          0, R1 =         23, R2 =         13, R3 =         16, R4 =         29, R5 =         10, R6 =         33, R7 =         26
# R8 =          8, R9 =         41, R10 =         0, R11 =         0, R12 =         0, R13 =         0, R14 =         0, R15 =         0
# R16 =         0, R17 =         0, R18 =         0, R19 =         0, R20 =         0, R21 =         0, R22 =         0, R23 =         0
# R24 =         0, R25 =         0, R26 =         0, R27 =         0, R28 =         0, R29 =         0, R30 =         0, R31 =         0
#
# ** Note: $finish    : D:/Materials/108-2-ComputerOrganization/CompOrgTeam/Lab5/testbench.v(30)
#    Time: 1800 ns  Iteration: 0  Instance: /testbench
```

CO_Data_2.txt

```
# PC =       136
# Data Memory =         0,         0,         0,         0,         0,         0,         0,         0
# Data Memory =         0,         0,         0,         0,         0,         0,         0,         0
# Data Memory =         0,         0,         0,         0,         0,         0,         0,         0
# Data Memory =         0,         0,         0,         0,         0,         0,         0,         0
# Registers
# R0 =          0, R1 =         50, R2 =         18, R3 =         32, R4 =         82, R5 =        114, R6 =         18, R7 =          0
# R8 =          0, R9 =          0, R10 =         0, R11 =         0, R12 =         0, R13 =         0, R14 =         0, R15 =         0
# R16 =         0, R17 =         0, R18 =         0, R19 =         0, R20 =         0, R21 =         0, R22 =         0, R23 =         0
# R24 =         0, R25 =         0, R26 =         0, R27 =         0, R28 =         0, R29 =         0, R30 =         0, R31 =         0
#
# ** Note: $finish    : D:/Materials/108-2-ComputerOrganization/CompOrgTeam/Lab5/testbench.v(30)
#    Time: 1800 ns  Iteration: 0  Instance: /testbench
```

Problems encountered and solutions:

1. 我們在學習 pipelining 時所學習的 Architecture，若是直接 apply 在這次 Lab 上，將會發生一個 Hazard：在同一個 Cycle 內寫入並讀取 RegFile，會發生讀取到舊值的情形

   - 我們在 RegFile 的讀取處加上了兩個 MUX，若是發生上述的情形時，則會將寫入值直接 Forward，詳細的實作在實作解釋的環節有說明

2. 我們在寫這次 Lab 時，從 Lab4 的檔案複製過來，但發現像 CPU 這種結構複雜的 module，很容易讓人看得眼花撩亂

   - 我們按照 Pipelining 中 Stage 的順序將 Module 與自定義的 Wire 排列好，讓我們在編寫與除錯上能夠對變數有更高的掌握，如圖

```
wire        RegWrite_EX;
wire        ALUSrc_EX;
wire [ 3:0] ALU_CTRL_EX;
wire [31:0] RSdata_o_EX;
wire [31:0] RTdata_o_EX;
wire [31:0] IMM_EX;
wire [ 4:0] RS_EX;
wire [ 4:0] RT_EX;
wire [ 4:0] RD_EX;
////////////////////////////////////////////////EX
wire [31:0] ALU_SRC2;
wire [31:0] ALUresult;
wire        ZERO;
wire        OVERFLOW;
wire        COUT;
////////////////////////////////////////////////EX
wire        RegWrite_MEM;
wire [31:0] ALUresult_MEM;
wire [ 4:0] RD_MEM;

wire [31:0] RSdata_o_FORWARDED;
wire [31:0] RTdata_o_FORWARDED;
wire [ 1:0] FORWARD_SRC1;
wire [ 1:0] FORWARD_SRC2;
////////////////////////////////////////////////MEM
wire [31:0] DM_o;
////////////////////////////////////////////////MEM
wire        RegWrite_WB;
wire [31:0] ALUresult_WB;
wire [ 4:0] RD_WB;
////////////////////////////////////////////////WB

////////////////////////////////////////////////WB
```

```
ForwardingUnit ForwardingUnit(
        .ID_EX__RS1(RS_EX),
        .ID_EX__RS2(RT_EX),
        .EX_MEM__RD(RD_MEM),
        .MEM_WB__RD(RD_WB),
        .EX_MEM__REG_WRITE(RegWrite_MEM),
        .MEM_WB__REG_WRITE(RegWrite_WB),
        .SRC1(FORWARD_SRC1),
        .SRC2(FORWARD_SRC2)
        );

//////////////////////////////////////////////////////////////////////    EX stage
EX_MEM EX_MEM(
        .clk_i(clk_i),
        .rst_i(rst_i),
        .REG_WRITE(RegWrite_EX),
        .ALU_RESULT(ALUresult),
        .RD(RD_EX),
        .REG_WRITE_O(RegWrite_MEM),
        .ALU_RESULT_O(ALUresult_MEM),
        .RD_O(RD_MEM)
        );
//////////////////////////////////////////////////////////////////////    MEM stage

Data_Memory Data_Memory(
        .clk_i(clk_i),
        .addr_i(ALUresult_MEM),
        .data_i(RTdata_o),
        .MemRead_i(1'b0),
        .MemWrite_i(1'b0),
        .data_o(DM_o)
        );

//////////////////////////////////////////////////////////////////////    MEM stage
```

Comment:

0716214 江岳勳：

這次的作業實作了上課講了非常久的 Pipeline，在上課的時候學 Pipeline 聽到都產生恐懼了，不過還好這次的作業沒有要實作 Branch, Jump 和 load/store 的指令，瞬間變得非常簡單，只要將線正確的接上，然後注意 Forward 就可以了。

比較值得注意的是因為 RegFile 是 posedge 寫入，在 WB 要寫回時如果剛好有 instruction 要讀出來，會不受 ForwardingUnit 的控制讀到寫入前的資料，所以我們加了一個自己寫的 module 進去做區分，其實就只是加上兩個 MUX 而已。

0716222 黃偉傑：

這次的 Lab 比我預想的來的輕鬆，或許是因為不需要處理 Load use 這類令人頭痛的問題，在學習了各式各樣既有 Architecture 下的問題與其解決方案後，感覺讓我們自己處理如這次 RegFile 的 Hazard 簡直是熟門熟路。很感謝助教這一連串的 Lab，讓我們由淺入深的累積 CPU 的架構知識，即使到了這堂課的尾聲我們仍僅能看到 CO 的皮毛，但實在是收穫甚多！感謝助教、教授，也感謝這幾次 Lab 的隊友！