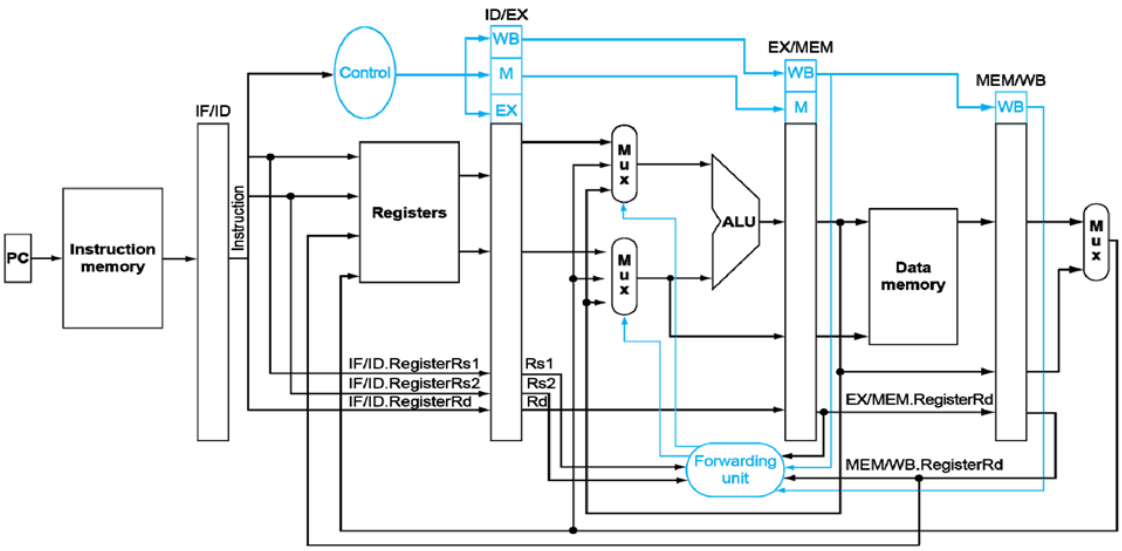
# Lab5

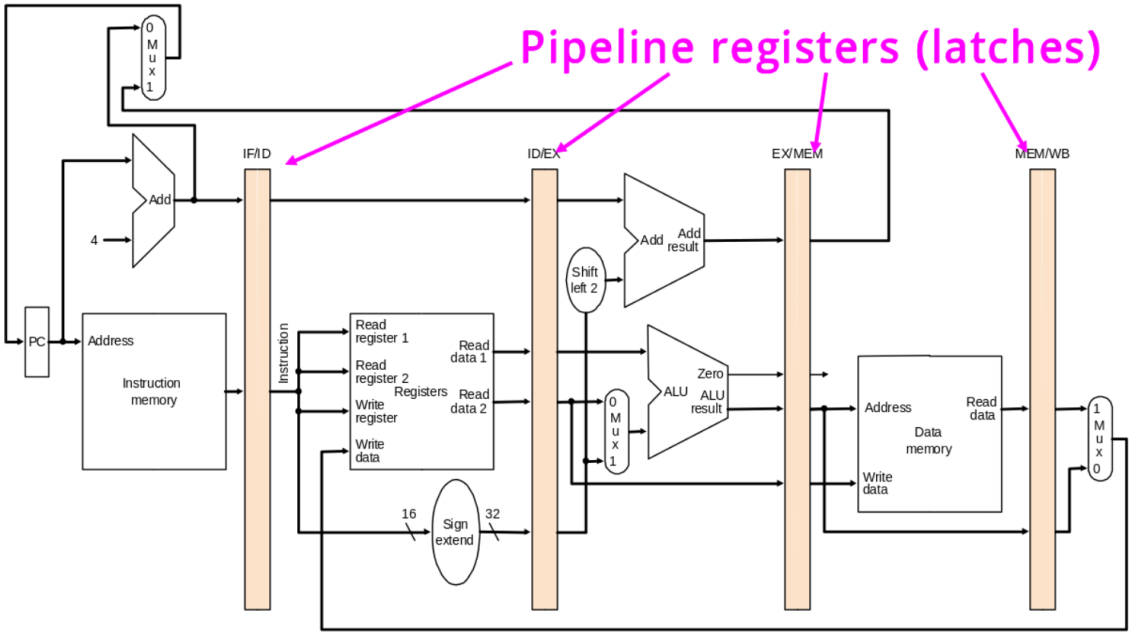
tags: CO


0712245 高魁駿
0712238 林彥彤


## Architecture diagram



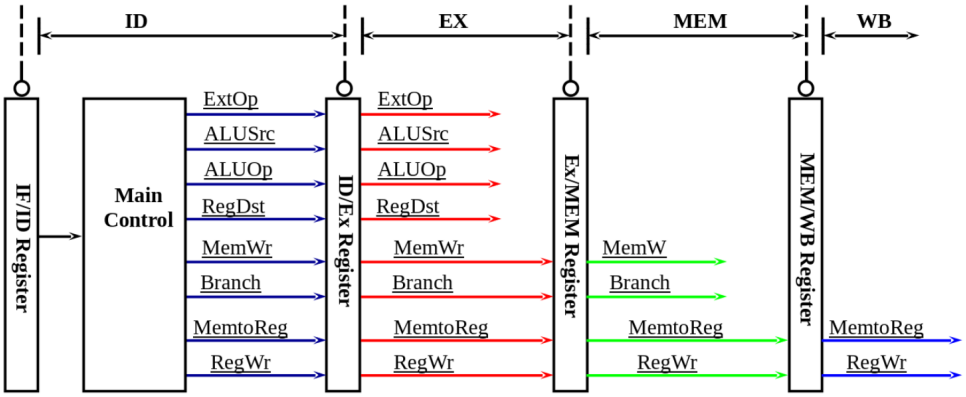

## Hardware module analysis




- 這次的主要目的是實作上面的pipeline register和加了 Forwarding Unit 來解決data hazard (R-type data hazard)
- Forwarding Unit 主要進行data的forwarding

Control signal

- control signal 跟 single cycle datapath 時並無不同，只是用 control signal 的時間不一樣
- 所以就根據使用的時間將 control signal 分類，用到的就可以不要了，沒用到的要繼續傳下去




## How to handle R-type hazard

> 從圖清楚的看到R-type的兩種hazard
> 第一種的條件為IF/ID.RegRt(Rs) = EX/MEMRegRd
> 第二種為條件為IF/ID.RegRt(Rs) = MEM/WB.RegRd
> 總之原因是寫入的目標與下個指令讀取的來源相衝突
> 加入forwarding後可以不用bubble解決衝突，code的條件判斷如下

- 需要Forwarding
  if(EX/MEM_WB[1] && EX/MEM_Rd != 0 && EX/MEM_Rd == ID/EX_Rs)
  Rs_forward <= 2'b10;

  if(EX/MEM_WB[1] && EX/MEM_Rd != 0 && EX/MEM_Rd == ID/EX_Rt)
  Rt_forward <= 2'b10;

  if(MEM/WB_WB[1] && MEM/WB_Rd != 0 && MEM/WB_Rd == ID/EX_Rs)
  Rs_forward <= 2'b01;

  if(MEM/WB_WB[1] && MEM/WB_Rd != 0 && MEM/WB_Rd == ID/EX_Rt)
  Rt_forward <= 2'b01;

- 不需要Forwarding

  if(MEM/WB_Rd != ID/EX_Rs && EX/MEM_Rd != IDEX_Rs)
  Rs_forward <= 2'b00;

  if(MEM/WB_Rd != ID/EX_Rt && EX/MEM_Rd != ID/EX_Rt)
  Rt_forward <= 2'b00;

- **討論Pipe_Reg (總共五個stage，四個pipe_reg)**

```
        /* WB */
        MemtoReg_o <= 1'd0;
        RegWrite_o <= 1'd0;
        /* M*/
        MemRead_o  <= 1'd0;
        MemWrite_o <= 1'd0;
        Branch_o   <= 1'd0;
        /* Data*/
        PC_add_sum_o   <= 32'd0;
        zero_o  <= 1'd0;
        alu_result_o  <= 32'd0;
        RTdata_o <= 32'd0;
        RDaddr_o   <= 5'd0;

d

gin
        /* WB */
        MemtoReg_o <= MemtoReg_i;
        RegWrite_o <= RegWrite_i;
        /* M*/
        MemRead_o  <= MemRead_i;
        MemWrite_o <= MemWrite_i;
        Branch_o   <= Branch_i;
        /* Data*/
        PC_add_sum_o  <= PC_add_sum_i;
        zero_o   <= zero_i;
        alu_result_o  <= alu_result_i;
        RTdata_o <=  RTdata_i;
        RDaddr_o   <= RDaddr_i;
```

> 這次作業中我認為最重要的module就是各種Pipe_Reg 藉由這個module我們才得以實現Pipeline CPU， 在其中可以注意到我們使用<=而並非=是因為我們需要我們的訊號能夠在register中維持一個clock cycle的時間，而這是<=才能做到的，若使用 =的話，data_i一變data_o就會馬上改變，也不會使我們希望的結果。

## Implementation results

- Test case 1

```
# 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
```

- Test case 2

```
# 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
```

## Problems encountered and solutions

1. 發現 PipeLine 沒有控管好，不小心把不同stage的線直接接再一起，導致指令亂掉
2. pipeLine reg的 rst訊號很重要，是用來清除開機時的不穩定狀況，一開始沒接結果讓訊號不太乾淨，後來修正了。
3. 更改RegFile 模組，讓同一個clock可以同時讀取跟寫入而不會有資料問題
4. 在ID/EX前面要加上兩個mux去判斷根據ID.RS跟ID.RT有沒有跟WB.RD一樣，選擇要不要用WB寫回的資料當作是register讀出的資料

```
MUX_2to1 Mux_IDEX_control1(
.data0_i(RSdata_o),
.data1_i(Mux_MemtoReg_o),
.select_i(!(id_instr_o[19:15] ^ RD_wb)),
.data_o(RSdata2_o)
);
```

```
MUX_2to1 Mux_IDEX_control2(
.data0_i(RTdata_o),
.data1_i(Mux_MemtoReg_o),
.select_i(!(id_instr_o[24:20] ^ RD_wb)),
.data_o(RTdata2_o)
);
```

## Lesson learnt (if any)

There are a lot of small details in this lab before I actually did it, but I was too persistent and utopian before I did it.
As a result, thinking too many problems at a time has no clue to solve because of too many problems. Finally, honestly, I only want to think about some small practical problems, and plan some small progress every time before slowly grouping the whole CPU. This method is obviously used in daily life, but I really used this method in writing code after this lab. In the future, you may find a way to favor this model!