Computer Architecture Project 1

Members & Team Work

B04902003 董書博：接線、debug、module(Data_Memory.v)、pipeline、report(34%)
B04902017 李立譽： module(Forward.v,Hazard_detection.v)、pipeline、report (33%)
B04902043 謝宏祺：接線、debug、module(Jump.v, Beq 實作)、pipeline、report(33%)

Implementation of Pipelined CPU

- IF_ID.v

.

```
module IF_ID(
    clk_i,
    pc_i,
    inst_i,
    flush_i,
    write_i,
    pc_o,
    inst_o
);
```

. 1.將一些 data(ex, inst_i)儲存並傳到下一個 stage。

. 2.檢查 flush_i 以及 write_i 的值來決定是否需要 stall 或 flush。

.

- ID_EX.v

```
module ID_EX(
    clk_i,
    regA_i,//data in register
    regB_i,//data in register
    PC_i,
    RegDst_i,//control signal
    ALUOp_i,
    ALUSrc_i,
    RegWrite_i,
    MemtoReg_i,
    MemWrite_i,
    MemRead_i,
    RegistersRS_i,  //5bits instruction
    RegistersRT_i,  //5bits instruction
    RegistersRD_i, //5bits instruction
    immidiate_i, //immidiate value
    regA_o,
    regB_o,
    PC_o,
    RegDst_o,
    ALUOp_o,
    ALUSrc_o,
    RegWrite_o,
    MemtoReg_o,
    MemWrite_o,
    MemRead_o,
    RegistersRS_o,
    RegistersRT_o,
    RegistersRD_o,
    immidiate_o
);
```

. 1.將 input 轉傳到各自的 module。

.

- EX_MEM.v

```
module EX_MEM(
    clk_i,
    ALUout_i, //32bits
    Zero_i, //1 bit
    PC_i, //32bits
    regB_i, //32bits
    JumpAddr_i, //32bits
    RegistersRD_i, //32bits
    RegWrite_i,//control signal
    RegRead_i,
    MemtoReg_i,
    MemWrite_i,
    MemRead_i,

    ALUout_o,
    Zero_o,
    PC_o,
    regB_o,
    JumpAddr_o,
    RegistersRD_o,
    RegWrite_o,
    RegRead_o,
    MemtoReg_o,
    MemWrite_o,
    MemRead_o
);
```

. 1.**將 input 轉傳到各自的 module。**

.

- MEM_WB.v

```
module MEM_WB(
    clk_i,
    ALUout_i,
    MemoryDataRead_i,//read data from memory
    RegistersRD_i,
    RegWrite_i,
    MemtoReg_i,

    ALUout_o,
    MemoryDataRead_o,
    RegistersRD_o, //to forward unit
    RegWrite_o,
    MemtoReg_o
);
```

. 1.將 input 轉傳到各自的 module。

.

- Hazard_Detection.v

```
module Hazard_Detection(
    IDEXMemRead_i,
    IDEXRegRT_i,
    IFIDRegRT_i,
    IFIDRegRS_i,

    IFID_o,
    PCWrite_o,
    Hazard_o              //flush control signal
);
```

. 1.透過 IDEXMemRead_i 等 input signal 判斷是否要 stall 一個 cycle

.

```
if ( IDEXMemRead_i && ((IDEXRegRT_i == IFIDRegRT_i) || (IDEXRegRT_i == IFIDRegRS_i)) )
  begin
    IFID_o <= 1;    //flush IFID
    PCWrite_o <= 0; //don't update PC
    Hazard_o <= 1; //flush control signal
  end
else // as usual
  begin
    IFID_o <= 0;
    PCWrite_o <= 1;
    Hazard_o <= 0;
  end
```

- Forward.v

```
module Forward(
  EXMEMRegRD_i,//destination from EX_MEM
  EXMEMRegWrite_i,//control signal from EX_MEM
  EXMEMMemtoReg_i,//control signal from EX_MEM
  MEMWBRegRD_i,//destination from MEM_WB
  MEMWBRegWrite_i,//control signal from MEM_WB
  IDEXRegRT_i,//form ID_EX
  IDEXRegRS_i,//from ID_EX

  ForwardA_o,//control signal to MUX6
  ForwardB_o //control signal to MUX7
);
```

. 1.透過 control signal 判斷 ALU 的 input 要取哪一個值，藉此達到消除 Data Hazzard 的目的

- Flush(寫在 IF_ID 內)

. 當 Control Hazzard 發生時，透過 OR Gate 比對 Jump 跟 Branch，在將結果傳入 IF_ID.v 來覺是否要 flush data。

```
or or1(IF_Flush,Jump,branch);
```

Implementation of each module

- lw/sw
  - Data_Memory.v

```
module Data_Memory
(
    clk_i,
    addr_i,
    data_i,
    MemWrite_i,
    data_o
);
```

. 只有當 MemWrite_i 為真的時候才將 data_i 寫入 addr_i 在 memory 中的位置。

. 其他時候直接將 addr_i 在 memory 中的資料讀出放到 data_o。

  - Control.v / ALU_Control.v

. 新增 lw 和 sw 對應的 signal。

- beq
  - Shift32.v

. 將 Sign_Extend.v 輸出的值左移 2 bit。

  - Control.v / ALU_Control.v

. 新增 beq 對應的 signal。

  - CPU.v

. 以一個 AND gate 和一個 MUX 來決定下一次 PC 是否要選擇 beq。

- j
  - Shift26.v

. 將 inst[25:0]左移 2 bit。

  - Jump.v

. 將原本的 PC 最左邊 4 bit 和 Shift26.v 的輸出合成 32 bit。

  - Control.v

. 新增 j 對應的 signal。

  - CPU.v

. 以一個 AND gate 來決定下一次 PC 是否要選擇 j。

Problems and solution

- 在最剛開始的時候，有些 control signal 會是 'X' 的 case。
  - Solution : initialize 為 0

- 在同一個 cycle 同時要 write register 跟 read register 時，無法保證讀出來的 data 會是 write 之後的 data。
    - Solution : 將原本 assign RS, RT data 的 code 改為以下，並且 RDaddr 的位置使用的是 ' === '的判斷式，因為 ' == '只能判斷 0 or 1，' === '可以判斷 0, 1, x, z 的 case。

```
assign RSdata_o = (RegWrite_i==1 && !(^RDaddr_i === 1'bx) && (RSaddr_i==RDaddr_i)) ? RDdata_i : register[RSaddr_i];
assign RTdata_o = (RegWrite_i==1 && !(^RDaddr_i === 1'bx) && (RTaddr_i==RDaddr_i)) ? RDdata_i : register[RTaddr_i];
```

- 剛開始 Debug 時，只能慢慢用 fdisplay 來檢查各個 input、output。
    - Solution : 使用 GTKWave 直接檢查各個時間點各個參數的值。
- MUX5 與 MUX1 的 source 跟其他 MUX 剛好相反。
    - Solution : Module 不變，將 MUX5, MUX1 的兩個 input wire 對調。