B06902021 吳聖福

環境

- 執行環境：CSIE 工作站。
- 編譯方式：在 `code/` 資料夾中執行 `make` 即可，會產生 `testbench.vvp` 與 `mytest.vvp`，後者請見「測試程式」一節。
- 解壓縮後資料夾為一個 git repository。

實作說明

此處（`master` branch）實作完整的 RV32IM instruction set（除去 memory fence 與系統相關的 instruction 如 `ECALL` 等，以下稱「完整版」）。在 repository 中亦有包含另一個 branch（`simple`），是只有實作作業要求的 6 個 instruction 的簡化版。

以下分別說明各檔案中之 module，其中標示為綠色者代表僅有完整版有出現。

Instruction_Memory.v 為了方便測試程式（見下節）的實作，此處**將 instruction memory 的存取改為只用最低的 10 個 bit（包含最後兩個 0）進行定址**。

Data_Memory.v 類似 instruction memory，不過需支援 unaligned access、不同的存取寬度與讀取時的 sign extension。

Register.v 此檔案原先並沒有考慮到 x0 是 hardwired zero，因此在寫入前新增了一個檢查。

Adder.v 單純的加法器，用來計算下一個 instruction 位址（PC + 4）或 branch target（immediate + PC）。

MUX32.v 單純的 32-bit multiplexer。其中有 2-way 的 `MUX32_2` 與 4-way 的 `MUX32_4`，4-way 是供完整版中有些需要 3-way MUX 之處使用。

Sign_Extend.v 為了與完整版的一致性，將該 module 稱為 `Immediate_Gen`，功能即是將 instruction 中的 immediate 取出。簡化版僅處理 I-type 與 R-type，完整版則處理全部六種 format。

ALU.v 其輸入有兩個 operand 與 `alu_op` 與 `flag`，前者為所要執行的運算（對應 `instruction[25|14:12]`，完整版共 16 種運算），後者則是分別 `ADD/SUB`、`SRA/SRL`、`SRAI/SRLI` 的 bit（對應 `instrction[30]`）。

（實際上簡化版只需要 3 個 bit 就足以表達所有運算，但為了與完整版的相似性，仍然保持 5 個 bit 的輸入。）

完整版中另有一輸入 `eq`，代表 branch 時若 ALU 結果為 0 代表 taken 或非 0 代表 taken，並有一個輸出 `taken` 代表 taken 與否。

ALU_Control.v 產生 ALU 所需之 `alu_op`、`flag` 與 `eq`。輸入 instruction 中與操作有關的 bit（`[30|25|14:12]`）與指令種類（來自 control signal）。

Control.v 產生各種 control signal：

- **`alu_control`** 分辨指令是 immediate arithmetic、register arithmetic、branch 或其他。簡化版只有前二者。
- **`alu_1_src`, `alu_2_src`** ALU 兩個 operand 的來源（0、PC、immediate 或 register）。（簡化版中第一個 operand 固定。）
- **`reg_write`** 是否寫入 register。（簡化版中每個 instruction 都會寫入 register。）
- **`reg_src`** 寫入 register 資料的來源（ALU、data memory 或 PC + 4）。
- **`is_branch`, `is_jalr`, `is_jal`** 是否是 branch、`JALR` 或 `JAL` instruction，用以決定下一個 cycle PC 的來源。
- **`mem_write`, `mem_width`, `mem_sign_extend`** Data memory 的 control signal：是否寫入、存取寬度、是否做 sign extension。

CPU.v 依照 datapath 將上述 module 組合。為了實作方便，有將程式碼整理為類似 5-stage pipeline 的格式（實際 datapath 仍不變），亦有為同一個訊號在不同 stage 中給予不同的 wire 名稱。

testbench.v 此檔案並未修改，然而其中有一個錯誤是所列印出的 register 名稱採用的命名規則並非 RISC-V，而是 MIPS。`instrctions.txt` 中亦有此錯誤。

測試程式

為了方便測試正確性，在 `code/testcases/` 目錄下有一個 Bash script `test.sh`（需切換至該目錄下執行）。其功能為產生一個隨機的 RISC-V 程式，並以 jupiter 和 Verilog module 各自執行，比對兩者執行過程中 register file 是否相同。

為了使 Verilog 的初始設定能與 jupiter 相同，有新增一個 `mytest.v` 檔案（類似 `testbench.v`），其中有以 jupiter 的設定初始化各 register 與 PC，並且能以 command line argument 指定 instruction 檔案（格式為 `vvp mytest.vvp +file=[insturction file]`）。