

Computer Architecture 2020

Project 1 Building Pipelined CPU

team: RISK-V

Modules Explanation

Adder

- `input` :
 - `data0_in`
 - `data1_in`
- `output` :
 - `data_o`

把 `data0_in` 和 `data1_in` 加起來之後，`assign` 到 `data_o` 上面。

MUX32

- `input` :
 - `data0_i`
 - `data1_i`
 - `data2_i`
 - `data3_i`
 - `select_i` (2 bits)
- `output` :
 - `data_o`

根據 `select_i`，`assign data_o` 為 `data0_i`、`data1_i`、`data2_i` 或 `data3_i`。本次專案中，我們把所有的 `MUX` 都使用 4-way `MUX`，沒有用到的 `port` 全部都會設成 `32'b0`。

Imm_Gen

- `input` :
 - `instruc_i` (32 bits)
- `output` :
 - `imm_o`

輸入整個 instruction (`instruc_i`)，如果 `instr[5] == 0` (在用的到 `imm` 的前提下，代表的是 `addi`、`srai`、`lw`)，就輸出 `instr[31:20]` 前面補 20 位 `instr[31]`，接著如果 `instr[6] == 0` 的話 (代表是 `sw`)，輸出 (`instr[31:25]` concatenate `instr[11:7]`) 前面補 20 位 `instr[31]`，如果不是以上的 case (代表是 `beq`)，就輸出 (`instr[31]` concatenate `instr[7]` concatenate `instr[30:25]` concatenate `instr[11:8]`) 前面補 20 位 `instr[31]`。

ALU

- `input` :
 - `data0_i`
 - `data1_i`

- ALUctr1_i
- output :
 - data_o

根據 ALU_Control 給予的訊號 (ALUctr1_i) 進行各種運算。設定當任意 input (data0_i 、 data1_i 或 ALUctr1_i) 有更動時進行運算。注意若是 srar instruction，由於 immediate 只有最後 5 個 bit 有意義，故僅使用 data1_i[4:0]。

Zero

- input :
 - data0_i (32 bits)
 - data1_i (32 bits)
- output :
 - zero_o (1 bit)

如果 data0_i == data1_i，會把 zero_o assign 1'b1；否則為 1'b0。此元件是為了給 branch 在 ID 階段就可以判斷結果是否為 0 所使用。

Control

- input :
 - Op_i
 - NoOP
- output :
 - RegWrite_o
 - MemtoReg_o
 - MemRead_o
 - MemWrite_o
 - ALUOp_o
 - ALUSrc_o
 - Branch_o

輸出主要的 control signal。

- RegWrite_o：控制資料是否寫入 Register。0 表示不寫入，1 表示寫入。R-type、I-type 或 load instruction 時，輸出 1；save 或 branch instruction 時，輸出 0。
- MemtoReg_o：控制寫入 Register RD 的值。0 表示使用 ALU，1 表示使用從 Data Memory 讀出的值。R-type、I-type instruction 時，輸出 0；load instruction 時，輸出 1；save 與 branch instruction 由於不寫入 RD，設為 don't care(X)。
- MemRead_o：控制 Data Memory 是否進行讀取。0 表示不讀，1 表示進行讀取。除了 load instruction 需要讀取而設為 1，其餘 instruction 時都設為 0。
- MemWrite_o：控制 Data Memory 是否進行寫入。0 表示不寫，1 表示進行寫入。除了 save instruction 需要寫入而設為 1，其餘 instruction 時都設為 0。
- ALUOp_o：控制傳給 ALU_Control 的值，幫助 ALU_Control 進行正確的計算。我們對於 R-type instruction 輸出 2'b10；I-type instruction 輸出 2'b11；load 和 save instruction 輸出 2'b00；branch instruction 輸出 2'b01。
- ALUSrc_o：控制 ALU 的 data1 是來自 Register RS2 還是 immediate。0 表示使用 RS2，1 表示使用 immediate。R-type 和 branch instruction 時使用 RS2，輸出 0；I-type、load 和 save instruction 時使用 immediate，輸出 1。
- Branch_o：與其他 module 一同控制 PC，通知是否進行 branch。當遇到 branch instruction 時，輸出 1；其餘輸出 0。

- 當遇到 Stall 時(NoOP = 1)，所有 signal 都設為 0，確保所有 Register 與 Data Memory 都沒有被寫入。

ALU_Control

- input :
 - funct
 - ALUOp_i
- output :
 - ALUCtrl_o

讀取 instruction 中的 funct7 與 funct3，以及 Control 的 ALUOp，決定 ALU 要進行何種運算。

- ALUOp_i = 2'b00 時，為 save instruction，ALU 要進行加法。
- ALUOp_i = 2'b01 時，為 branch instruction，ALU 要進行減法。
- ALUOp_i = 2'b11 時，為 I-type instruction。根據 funct 決定是進行 addi instruction，還是 srai instruction。
- ALUOp_i = 2'b10 時，為 R-type instruction。根據 funct 決定輸出 and、xor、sll、add、sub、mul。
- 由於 add 與 addi instruction 皆是進行加法運算，故兩種情形輸出給 ALU 的信號皆相同 (4'b0011)。

CPU

- input :
- output :

適當地配置各種元件然後正確地接線。

AND

- input :
 - data0_i (1 bit)
 - data1_i (1 bit)
- output :
 - bool_o (1 bit)

會把 bool_o assign 為 data0_i & data1_i 的結果。此元件是要結合 Control.Branch 來判斷 zero 的輸出是否有意義。bool_o 的結果會決定是否需要 flush IF/ID 以及下一個 PC 的 source。

Forwarding Unit

- input :
 - IDEX_rs1_addr_i
 - IDEX_rs2_addr_i
 - EXMEM_rd_addr_i
 - MEMWB_rd_addr_i
 - EXMEM_Regwrite_i
 - MEMWB_Regwrite_i
- output :
 - ForwardA_o
 - ForwardB_o
- 判斷是否需要進行Forwarding。

- 當 pipeline latch EX/MEM 中的 RegWrite signal 為 1，Register RD 不為 0，且 RD = pipeline latch ID/EX 中的 RS1 時，會發生 EX hazard。此時 ForwardA_o 輸出 2'b10，使 ALU 使用 EX/MEM pipeline latch 裡的 register 資料做運算。
- 當 pipeline latch MEM/WB 中的 RegWrite signal 為 1，Register RD 不為 0，且 RD = pipeline latch ID/EX 中的 RS1 時，會發生 MEM hazard。此時 ForwardA_o 輸出 2'b01，使 ALU 使用 MEM/WB pipeline latch 裡的 register 資料做運算。
- ForwardB_o 與 ForwardA_o 雷同。ForwardA_o 控制 ALU 的 data0 來源，ForwardB_o 控制 ALU 的 data1 來源。
- 當 EX hazard 與 MEM hazard 同時發生時，需使用 EX hazard 的 forwarding 結果，故優先判斷 EX hazard。

Hazard Detection Unit

- input :
 - rs1_addr_i
 - rs2_addr_i
 - IDEX_MemRead_i
 - IDEX_rd_addr_i
- output :
 - Stall_o
 - PCwrite_o
 - NOOP_o

本次專案裡面的 Hazard Detection 主要處理的是 load instruction 所造成必須要 stall 的問題。而會造成 hazard 的條件是：

1. IDEX_MemRead_i 為 1'b1
2. IDEX_rd_addr_i 等於 rs1_addr_i 或是 rs2_addr_i

當以上兩個條件都滿足時，會 assign Stall_o 為 1'b1。代表 IF/ID 需要暫停寫入，且 PC 也不需要更新，control 也需要送一個 NoOp 讓整個 pipeline stall 一個 cycle。

因此 assign PCwrite_o 為 ~Stall_o，表示 PC 不能寫入；NOOP_o 為 Stall_o，表示需要給一個 NoOp。

Pipeline latch IF/ID

- input :
 - clk_i
 - rst_i
 - IFID_Write_i
 - Flush_i
 - PC_i
 - instruc_i
- output :
 - PC_o
 - instruc_o

使用 reg 宣告所需要的暫存變數。

用 always block 判斷，當 clk_i 或 rst_i 上升時：

- 如果 IFID_Write_i 為 1'b1，會更新 (使用 non-blocking 把 input 指定給 output) PC、instruc_o。
- 如果 Flush_i 為 1'b1，會把 PC、instruc_o 更新為 32'b0。

Pipeline latch ID/EX

- `input` :
 - `clk_i`
 - `rst_i`
 - `RegWrite_i`
 - `MemtoReg_i`
 - `MemRead_i`
 - `MemWrite_i`
 - `ALUOp_i` (2 bits)
 - `ALUSrc_i`
 - `rs1_data_i`
 - `rs2_data_i`
 - `rs1_addr_i`
 - `rs2_addr_i`
 - `rd_addr_i`
 - `funct_i` (10 bits)
 - `imm_i` (32 bits)
- `output` :
 - `RegWrite_o`
 - `MemtoReg_o`
 - `MemRead_o`
 - `MemWrite_o`
 - `ALUOp_o` (2 bits)
 - `ALUSrc_o`
 - `rs1_data_o`
 - `rs2_data_o`
 - `rs1_addr_o`
 - `rs2_addr_o`
 - `rd_addr_o`
 - `funct_o` (10 bits)
 - `imm_o` (32 bits)

使用 `reg` 宣告所需要的暫存變數。

用 `always` block 判斷，當 `clk_i` 或 `rst_i` 上升時，如果 `rst_i` 為 `1'b0`，則把所有的變數 (除了 `clk_i` 和 `rst_i`) 都以 ID stage 來的 input 使用 `non-blocking` 指定給 EX stage 的 output。

註：因為初始化是由 `testbench` 來執行，因此當 `rst_i` 為 `1'b1` 時，代表目前的 output 已經被初始化好了，不需要進行更新，否則會被 input 蓋掉變成 `1'bx`。

Pipeline latch EX/MEM

- `input` :
 - `clk_i`
 - `rst_i`
 - `RegWrite_i`
 - `MemtoReg_i`
 - `MemRead_i`
 - `MemWrite_i`
 - `ALUout_i`
 - `rs2_data_i`

- rd_addr_i
- output :
 - RegWrite_o
 - MemtoReg_o
 - MemRead_o
 - MemWrite_o
 - ALUout_o
 - rs2_data_o
 - rd_addr_o

使用 `reg` 宣告所需要的暫存變數。

用 `always` block 判斷，當 `clk_i` 或 `rst_i` 上升時，如果 `rst_i` 為 `1'b0`，則把所有的變數（除了 `clk_i` 和 `rst_i`）都以 `Ex` stage 來的 input 使用 `non-blocking` 指定給 `MEM` stage 的 output。

註：因為初始化是由 `testbench` 來執行，因此當 `rst_i` 為 `1'b1` 時，代表目前的 output 已經被初始化好了，不需要進行更新，否則會被 input 蓋掉變成 `1'bx`。

Pipeline latch MEM/WB

- input :
 - clk_i
 - rst_i
 - RegWrite_i
 - MemtoReg_i
 - ALUout_i
 - Memout_i
 - rd_addr_i
- output :
 - RegWrite_o
 - MemtoReg_o
 - ALUout_o
 - Memout_o
 - rd_addr_o

使用 `reg` 宣告所需要的暫存變數。

用 `always` block 判斷，當 `clk_i` 或 `rst_i` 上升時，如果 `rst_i` 為 `1'b0`，則把所有的變數（除了 `clk_i` 和 `rst_i`）都以 `MEM` stage 來的 input 使用 `non-blocking` 指定給 `WB` stage 的 output。

註：因為初始化是由 `testbench` 來執行，因此當 `rst_i` 為 `1'b1` 時，代表目前的 output 已經被初始化好了，不需要進行更新，否則會被 input 蓋掉變成 `1'bx`。

Testbench

- input :
- output :

初始化的過程中，依據各個資料的 bit 數，設成適當 bit 數的 0，然後在 `always` block 裡面加上計算 stall 和 flush 的次數。

Members and Teamwork

- B07209016 鍾晨瑋：負責新增 `pipeline latch`、`testbench`、`Hazard Detection Unit`，和修改 `Immediate Generator` 和其餘小模組（`MUX`、`Adder`、`AND`、`Zero`）。第一階段 `Debug`（使 CPU 可以正常執行一般指令、處理 Forwarding、Stall）。
- B07201022 杜宗穎：主要負責 `CPU` 的接線工作。

- B07902063 陳耕宇：從 HW4 成果新增 load、save、branch instruction。更新 `Control`、`ALU_Control`、`ALU`。新增 `Forwarding Unit`。
- 共同撰寫 Report。

Difficulties Encountered and Solutions

- Q：一開始進行 pipeline 初始化的時候，過不了幾個 cycle 整個 PC 停止前進。

A：主要問題有兩個：

1. pipeline latch 在 rest 時的 clock positive edge 會把預設好的 output 被 not defined 的 input 蓋掉，造成輸出全部都是 `x`。
2. Control Unit 原本所有的預設值都是 `x`，造成 Hazard Unit 對於 `PCwrite` 也會是 `x`，讓 `PC` 無法往下。

- Q：port 太多，名稱過於混亂。

A：使用共同編輯文件的方式，事先約定好各自 module 的名稱。

Development Environment

- OS: Windows 10 Home ver 1909
- Compiler: Icarus Verilog version 11.0