Computer Architecture 2018 Fall – Final Project 1

Team：地表計結最強的那組喔！懂？

1. Members and team work

Member			Team work	
資工三	B05902019	蔡青邑	(停修)	
資工三	B05902023	李澤諺	50%	1. 修改 Homework 4 中的 ALU.v、Sign_Extend.v、Control.v、ALU_Control.v，並修改 Homework 4 中的 single cycle CPU，使其可執行 lw、sw、beq 等指令。 2. 寫 Forward.v、Hazard.v、MUX3.v、MUX_Control.v，並接 pipelined CPU。 3. 修改 testbench.v 以及 Debug。
資工三	B05902121	黃冠博	50%	1. 寫 And.v、Data_Memory.v，並修改 Homework 4 中的 single cycle CPU，使其可執行 lw、sw、beq 等指令。 2. 寫 IF_ID.v、ID_EX.v、EX_MEM.v、MEM_WB.v，修改 PC.v，並接 pipelined CPU。 3. 修改 testbench.v 以及 Debug。

2. Implementation

(1) Modules

Module	Implementation
PC.v	沿用 Homework 4 提供的 PC.v，但在 input port 中增加 pc_write_i。 在 always 敘述中，若 start_i 和 pc_write_i 的值皆為 1’b1，則更新 pc_o 的值。 若 pc_write_i 的值為 1b’0，代表 CPU 需要 stall，則不更新 pc_o 的值。
Registers.v	沿用 Homework 4 提供的 Registers.v。
Instruction_Memory.v	沿用 Homework 4 提供的 Instruction_Memory.v。
Data_Memory.v	Input port： (1 bit) clk_i

	(1 bit) MemRead_i (1 bit) MemWrite_i (32 bits) addr_i (32 bits) data_i Output port： (32 bits) data_i 宣告共有 8 個 entry，每個 entry 皆為 8 bits 的 reg，稱為 memory，用來記錄 data memory 中所存放的值。 在 posedge clk_i 時，若 MemWrite_i 的值為 1'b1，則將 data_i 拆為 4 個 8 bits，依序寫入 addr_i 開始的 4 個 memory entry 中。 若 MemRead_i 的值為 1'b1，則將 addr_i 開始的 4 個 memory entry 中所記錄的值取出，組合起來 assign 給 data_o(不須考慮 clk_i 為 posedge 或 negedge)。
And.v	Input port： (1 bit) data1_i (1 bit) data2_i Output port： (1 bit) data_o 宣告 1 bit 的 reg，稱為 data_reg。 用 always 敘述將 data1_i 和 data2_i 的值 and 給 data_reg 後，再將 data_reg 的值 assign 給 data_o。
Adder.v	Input port： (32 bits) data1_i (32 bits) data2_i Output port： (32 bits) data_o 將 data1_i 和 data2_i 的值相加後 assign 給 data_o。
ALU.v	Input port： (32 bits) data1_i (32 bits) data2_i (4 bits) ALUCtrl_i

<table>
<tr><td></td><td>Output port：
(32 bits) data_o
(1 bit) Zero_o
(其實將 beq 的判斷提前至 ID stage 做後，就不需要 Zero_o 了。)

宣告 32 bits 的 reg，稱為 data_reg。
利用 case 敘述，根據 ALUCtrl_i 的值，決定 ALU 要進行何種運算，規則如下：
<table><tr><th>ALUCtrl_i</th><th>Operation</th></tr><tr><td>4’b0010</td><td>data1_i + data2_i</td></tr><tr><td>4’b0110</td><td>data1_i - data2_i</td></tr><tr><td>4’b0111</td><td>data1_i * data2_i</td></tr><tr><td>4’b0000</td><td>data1_i & data2_i</td></tr><tr><td>4’b0001</td><td>data1_i | data2_i</td></tr></table>將 data1_i 和 data2_i 運算的結果給 data_reg 後，再將 data_reg 的值 assign 給 data_o。</td></tr>
<tr><td>MUX32.v</td><td>Input port：
(32 bits) data1_i
(32 bits) data2_i
(1 bit) select_i

Output port：
(32 bits) data_o

若 select_i 的值為 1’b0，則將 data1_i 的值 assign 給 data_o。
而若 select_i 的值為 1’b1，則將 data2_i 的值 assign 給 data_o。</td></tr>
<tr><td>MUX3.v</td><td>Input port：
(32 bits) data1_i
(32 bits) data2_i
(32 bits) data3_i
(2 bits) select_i

Output port：
(32 bits) data_o</td></tr>
</table>

<table>
<tr><td></td><td>若 select_i 的值為 2’b00，則將 data1_i 的值 assign 給 data_o。
而若 select_i 的值為 2’b01，則將 data2_i 的值 assign 給 data_o。
而若 select_i 的值為 2’b10，則將 data3_i 的值 assign 給 data_o。</td></tr>
<tr><td>MUX_Control.v</td><td>將 Control 的 control signal 全部傳入 MUX_Control 的 input port 中，並在 input port 中加上 1 bit 的 select_i。
若 select_i 的值為 1’b0，則將 control signal 原封不動從 output port 輸出。
而若 select_i 的值為 1’b1，代表 CPU 需要 stall，則將 control signal 全部設為 0 再輸出。</td></tr>
<tr><td>Sign_Extend.v</td><td>Input port：
(32 bits) data_i

Output port：
(32 bits) data_o

宣告 12 bits 的 reg，稱為 data_reg。
將整個 instruction 作為 input 傳入 Sign_Extend 後，利用 always 敘述，根據 instruction 為何種 format，來計算 immediate 的值，規則如下：
<table><tr><th>Name (Field Size)</th><th colspan="6">Field</th><th>Comments</th></tr><tr><th></th><th>7 bits</th><th>5 bits</th><th>5 bits</th><th>3 bits</th><th>5 bits</th><th>7 bits</th><th></th></tr><tr><td>R-type</td><td>funct7</td><td>rs2</td><td>rs1</td><td>funct3</td><td>rd</td><td>opcode</td><td>Arithmetic instruction format</td></tr><tr><td>I-type</td><td colspan="2">immediate[11:0]</td><td>rs1</td><td>funct3</td><td>rd</td><td>opcode</td><td>Loads & immediate arithmetic</td></tr><tr><td>S-type</td><td>immed[11:5]</td><td>rs2</td><td>rs1</td><td>funct3</td><td>immed[4:0]</td><td>opcode</td><td>Stores</td></tr><tr><td>SB-type</td><td>immed[12,10:5]</td><td>rs2</td><td>rs1</td><td>funct3</td><td>immed[4:1,11]</td><td>opcode</td><td>Conditional branch format</td></tr><tr><td>UJ-type</td><td colspan="3">immediate[20,10:1,11,19:12]</td><td></td><td>rd</td><td>opcode</td><td>Unconditional jump format</td></tr><tr><td>U-type</td><td colspan="3">immediate[31:12]</td><td></td><td>rd</td><td>opcode</td><td>Upper immediate format</td></tr></table>
接著，將 data_reg 的值 assign 給 data_o[11 : 0]，再用 data_reg[11]的值(即 sign bit)將 data_o 剩下的 20 bits 填滿。</td></tr>
<tr><td>Control.v</td><td>Input port：
(6 bits) Op_i

Output port：
(2 bits) ALUOp_o
(1 bit) ALUSrc_o
(1 bit) Branch_o
(1 bit) MemWrite_o
(1 bit) MemRead_o</td></tr>
</table>

<table>
<tr><td></td><td>
(1 bit) RegWrite_o
(1 bit) MemtoReg_o

根據 Op_i 的值，設定 control signal，規則如下：
<table>
<tr><th>Instruction</th><th>R type</th><th>addi</th><th>lw</th><th>sw</th><th>beq</th></tr>
<tr><td>ALUOp</td><td>10</td><td>11</td><td>00</td><td>00</td><td>01</td></tr>
<tr><td>ALUSrc</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>Branch</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>MemRead</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>MemWrite</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>RegWrite</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>MemtoReg</td><td>0</td><td>0</td><td>1</td><td>X</td><td>X</td></tr>
</table>
※當 CPU 要 flush 時，instruction 會被設為 32’b0，此時 Op_i 為 7’b0，Control 會將所有 control signal 設為 0 輸出。
</td></tr>
<tr><td>ALU_Control.v</td><td>
Input port：
(32 bits) funct_i
(2 bits) ALUOp_i

Output port：
(4 bits) ALUCtrl_o

將整個 instruction 傳給 funct_i，再加上 ALUOp_i 的值，設定 ALUC_trl_o，規則如下：
<table>
<tr><th>Instruction</th><th>ALUOp</th><th>funct 3</th><th>funct 7</th><th>ALUCtrl</th></tr>
<tr><td>add</td><td rowspan="5">10</td><td rowspan="3">000</td><td>0000000</td><td>0010</td></tr>
<tr><td>sub</td><td>0100000</td><td>0110</td></tr>
<tr><td>mul</td><td>0000001</td><td>0111</td></tr>
<tr><td>and</td><td>111</td><td>(不用判斷)</td><td>0000</td></tr>
<tr><td>or</td><td>110</td><td>(不用判斷)</td><td>0001</td></tr>
<tr><td>addi</td><td>11</td><td colspan="2" rowspan="3">(不用判斷)</td><td rowspan="3">0010</td></tr>
<tr><td>lw</td><td rowspan="2">00</td></tr>
<tr><td>sw</td></tr>
<tr><td>beq</td><td>01</td><td>0110</td></tr>
</table>
根據 ALUCtrl_o 的值，可決定 ALU 要做何種運算。
</td></tr>
<tr><td>IF_ID.v</td><td>將要傳入 ID stage 的資料，在 posedge clk_i 時寫入對應的 reg 中，並在 negedge clk_i 時，將資料讀給對應的 output reg，傳入 ID stage 中。</td></tr>
</table>

	而在 input port 中，加入 IF_ID_write_i 與 IF_ID_flush_i。 若 IF_ID_write_i 為 1’b0，代表 CPU 需要 stall，此時不更新 IF_ID 中 reg 已經存的值。 若 IF_ID_flush_i 為 1’b1，代表 CPU 需要 flush，此時將存 instruction 的 reg 的值設為 32’b0。
ID_EX.v	將要傳入 EX stage 的資料，在 posedge clk_i 時寫入對應的 reg 中，並在 negedge clk_i 時，將資料讀給對應的 output reg，傳入 EX stage 中。
EX_MEM.v	將要傳入 MEM stage 的資料，在 posedge clk_i 時寫入對應的 reg 中，並在 negedge clk_i 時，將資料讀給對應的 output reg，傳入 MEM stage 中。
MEM_WB.v	將要傳入 WB stage 的資料，在 posedge clk_i 時寫入對應的 reg 中，並在 negedge clk_i 時，將資料讀給對應的 output reg，傳入 WB stage 中。
Forward.v	若 ID_EX 中的 RDaddr 和 EX_MEM 中的 RSaddr 或 RTaddr 相同，代表出現 EX hazard，則將 select signal 設為 2’b10 傳給 MUX3。 若 ID_EX 中的 RDaddr 和 MEM_WB 中的 RSaddr 或 RTaddr 相同，代表出現 MEM hazard，則將 select signal 設為 2’b01 傳給 MUX3。 若沒有出現 hazard，則將 select signal 設為 2’b00 傳給 MUX3。
Hazard.v	若 ID_EX 中的 MemRead signal 為 1’b1，且 ID_EX 中的 RDaddr 和 IF_ID 中的 RSaddr 或 RTaddr 相同，代表出現 load hazard，則將 PC_write_o 和 IF_ID_write_o 皆設為 1’b1 分別傳給 PC 和 IF_ID，以 stall CPU。

(2) Pipelined CPU

a. IF stage 中，宣告 32 bits 的 wire，稱為 inst_addr，PC 輸出的值傳給 inst_addr 後，inst_addr 的值會傳給兩個 Adder：Add_PC 和 Add_PC_branch，Add_PC 會將 inst_addr 的值加 4，並將結果傳給 MUX_PC 的 data1_i，而 inst_addr 的值也會透過 IF_ID 傳給 Add_PC_branch，其將要 branch 的 address 算出後，會將結果傳給 MUX_PC 的 data2_i，接著由 MUX_PC 的 select signal，來決定下一個 cycle 中 PC 的值為何。

b. ID stage 中，會將 instruction 自 IF_ID 中讀出，並利用 instruction 自 Registers 中讀出 RSdata 與 RTdata，將其傳給 ID_EX。而 instruction 也會傳給 Sign_Extend，以計算 immediate 的值。接著，immediate 會傳到兩個地方：

首先，immediate 會透過 ID_EX 傳到 EX stage，另一方面，immediate 向左 shift 1 個 bit 後也會傳給 Add_PC_branch，用以得出要 branch 的 address。此外，instruction 中的 Op code 會傳給 Control，用以得出 control signal，control signal 中，除了 Branch 之外，所有的 control signal 皆會傳入 ID_EX 中。而 Branch 會傳給 And 的 data1_i，data2_i 的值則會根據 RSdata 和 RTdata 的值是否相等來決定，若相等則為 1'b1，不相等則為 1'b0，如此一來，And 的結果即代表要不要 branch，該結果會透過稱為 taken 的 wire 傳給 MUX_PC 的 select_i 以及 IF_ID 的 IF_ID_select_i。最後，instruction、RSaddr、RTaddr、RDaddr 亦會透過 ID_EX 傳入 EX stage。

c. EX stage 中，RSdata 和 RTdata 自 ID_EX 讀出後，會先分別傳給 MUX_ALU_data1 和 MUX_ALU_data2，此外，MEM stage 和 WB stage 中的 ALU result 也皆會傳給 MUX_ALU_data1 與 MUX_ALU_data2，並透過 Forward 來決定是否要 forward，MUX_ALU_data1 的結果會傳給 ALU，而 MUX_ALU_data2 的結果會再傳給 MUX_ALUSrc 與 EX_MEM，MUX_ALUSrc 也會自 ID_EX 得到 immediate 的值，並透過 Control signal 來決定 MUX_ALUSrc 的結果，將其傳給 ALU，ALU 再透過 ALU_Control 來做出相應的運算，將結果傳給 EX_MEM。最後，ID_EX 中除了 ALUOp 和 ALUSrc 之外的 control signal，以及 RDaddr，皆會透過 EX_MEM 傳入 MEM stage。
d. MEM stage 中，EX_MEM 會將 ALU result 傳給 Data Memory 以及 MEM_WB，Data Memory 會將 ALU result 作為 memory 要讀寫的 address，而 EX_MEM 中的 RTdata 則會傳給 Data Memory 做為要寫入的值。Data Memory 讀出的值、RDaddr，以及 EX_MEM 中 MemRead、MemWrite 之外的 control signal 皆會透過 MEM_WB 傳入 WB stage。
e. WB stage 中，MEM_WB 中存的自 Data Memory 讀出的值，以及 ALU result，皆會傳給 MUX_MemtoReg，透過 MemtoReg signal，會決定出要將何者寫回 Registers，而 MEM_WB 中存的 RegWrite signal 和 RDaddr，會決定使否要寫 Registers，以及要寫回 Registers 的位置。
f. Forward 的運作原理如上，其結果會傳給 MUX_ALU_data1 和 MUX_ALU_data2 作為 select signal。
g. Hazard 的運作原理如上，若其發現 load hazard，則會利用 PC_write_i 和 IF_ID_write_i signal 分別傳給 PC 和 IF_ID，以此 stall CPU。
h. And 的結果除了會透過 taken 傳給 MUX_PC 之外，taken 的值也會傳給 IF_ID，作為 IF_ID_flush_i。若 taken 為 1'b1，代表要 branch，此時必須 flush CPU，故 taken 的值即為 IF_ID_flush_i 的值。

3. Problems and solutions in this project

在剛接完 pipelined CPU 並編譯執行後，發現 PC 會輸出 X，整個 CPU 完全動不

了，之後向 TA hour 求助，萬分感謝助教提供的 debug 方法，讓我們組順利找出了 bug，發現是因為 control signal 沒有初始化所致。而在初始化 control signal 時，用了各種方式(包括 testbench.v 中的 initial)都無法將 control signal 初始化，試了很多種方法後終於找到解決辦法：在 CPU.v 中為每一個無法被初始化的 control signal 皆宣告一個 reg，在這些 control signal 傳到所需的 module 之前，先將其傳給對應的 reg，再將 control signal 自 reg 傳給 module，而在 CPU.v 中，利用 initial 敘述，將這些 reg 的值初始化，才解決了無法初始化 control signal 的問題。將此問題解決後，我們組的 pipelined CPU 終於能正常執行，並得到正確的結果。