Project 1

Members: 曾民君 (R08922167), 徐浩翔(R07922163)

Division of labor:

曾民君（50%）	Plane architecture, Pipeline Registers, Shif.v,Equal.v ALU.v, ALU_Control.v, modify testbench.v, debug
徐浩翔（50%）	MUX.v、Forwarding_Unit.v、Hazard_Detection_Unit.v、 Sign_Extend.v、Control.v、modify testbench.v 、debug。

Implementation:

Modules name	Description
CPU.v	與HW3 一樣負責將所有modules接在一起
PC.v	與第三次作業一樣是由助教提供的modules，負責指出目前程式執行的instruction address
Instruction_Memory.v	存有instruction的memory，當值從PC輸出後到相對應的記憶體位址讀出instruction。
Registers.v	32個32bits 的Register file
Data_Memory.v	由助教提供的modules，data memory
Adder.v	與第HW3一樣負責將兩個input加起來並輸出答案。
MUX32.v	32 bits multiplexer 由控制信號決定輸出input 1,2，接在PC前決定PC address為PC+4 or branch
MUX5.v	5 bits multiplexer， 決定寫入的register 位置
MUX_MemToReg.v	32 bits multiplexer， 決定寫入register data為memory output 或是 alu output
MUX_AluSrc.v	32 bits multiplexer 由控制信號決定輸出input 1,2,3，在ALU前面接了兩個用來決定input data是register讀出的或是由後面forwarding的。
MUX_Stall.v	接在Control後面當hazard發生時將控制信號線設成零
ALU.v	與HW3相同根據control signal 負責執行算術運算 0000: (and)&　　0001:(or) |　　0010:(add)+ 0110:(sub) -　　0011: (mul)*　　0110:(beq) - 其中因為branch 在前面一個stage已經做完了所以0110訊號為多餘的

<table>
<tr><td></td><td></td></tr>
<tr><td>Control.v</td><td>input 7 bits opcode
用來判斷為何種 instruction 產生控制信號線
<table>
<tr><th rowspan="2">Instruction</th><th colspan="2">Execution/address calculation stage control lines</th><th colspan="3">Memory access stage control lines</th><th colspan="2">Write-back stage control lines</th></tr>
<tr><th>ALUOp</th><th>ALUSrc</th><th>Branch</th><th>Mem-Read</th><th>Mem-Write</th><th>Reg-Write</th><th>Memto-Reg</th></tr>
<tr><td>R-format</td><td>10</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>ld</td><td>00</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>sd</td><td>00</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>X</td></tr>
<tr><td>beq</td><td>01</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>X</td></tr>
</table></td></tr>
<tr><td>ALU_Control.v</td><td>根據下表把funct 與 aluop 輸出 aluctrl

funct : aluOp : aluCtrl
and -> 0000000111 : 10 : 0000
or -> 0000000110 : 10 : 0001
add -> 0000000000 : 10 : 0010
addi-> xxxxxxx000 : 01 : 0010
sub -> 0100000000 : 10 : 0110
mul -> 0000001000 : 10 : 0011
lw -> xxxxxxx010 : 00 : 0010
sw -> xxxxxxx010 : 00 : 0010
beq -> xxxxxxx000 : 01 : 0110 -> don't care</td></tr>
<tr><td>Pipeline Registers</td><td>For every input data we declared a corresponding reg output, and initialized them to 0, also take clk as one of the input. Update all output when positive edge occur.</td></tr>
<tr><td>Shift.v</td><td>Use Signed left shift (<<<) 1 bit to immExtended</td></tr>
<tr><td>Branch_Equal.v</td><td>Check if rsData equals to rtData, if yes then set equal_o to 1, else set equal_o to zero.</td></tr>
<tr><td>Forwarding_Unit.v</td><td>Basically the implementation is same as Power Point. First, check MEM stage forwarding first (10). Second, check if WB stage forwarding (01). If not necessary using forwarding then set (00).</td></tr>
<tr><td>Hazard_Detection_Unit.v</td><td>Set output register hazardDected_o as zero first. Then checked if EX_memRead is 1, EX_wbAddr not equal to 0, and EX_wbAddr is equal to ID_rsAddr or ID_rtAddr</td></tr>
</table>

Difficulties encountered and solutions of this projects:

1. Wrong Sign_Extention_Unit implementation for different type of instructions first, remodeled contents to make the unit work well.
2. Lots of x at wires and registers, to solve this problems we redefine every output wire as registers and initialized to zero.
3. Wrong PCs at the earlier version, cause we filled wrong parameter to pc_write, we use negate of hazardDetected_signal as input.
4. The multiplexer for choosing imm and rtData was used before forwarding multiplexer, but this made the execution result wrong, so we switched it after the forwarding multiplexer.
5. We did not do well on version control, so it took some time for us checking what another one changed.