計算機組織 Lab2

9617145 許晏峻

程式碼：(灰色部分代表 lab1 以實作部份)

✧ **Test pattern：**

```
`timescale 1ns / 1ps

module Decoder_test;

    reg Clk;                                    // clock signal
    reg Reset;                                  // 初始化
    reg [31:0] Instruction;                     // 存指令

    Decoder uut (                               // 連接 decoder module
        .Clk(Clk),
        .Reset(Reset),
        .Instruction(Instruction)
    );

    initial begin
        Clk = 0;
        Reset = 0;
        Instruction = 0;

        #10 Reset=1;
        #10 Reset=0;
        #100 Instruction=32'b00100000_00000000_00000001_00000011; // (ADD R0 R1 R3)
        #100 Instruction=32'b00100000_00000010_00000000_00000001; // (ADD R2 R0 R1)
        #100 Instruction=32'b00010000_00000001_00000011_00000010; // (SUB R1 R3 R2)
        #100 Instruction=32'b00010000_00000011_00000010_00000000; // (SUB R3 R2 R0)
        #100 Instruction=32'b00001000_00000000_00000010_00000010; // (SL R0 2 R2)
        #100 Instruction=32'b00000100_00000001_00000011_00000000; // (SR R1 3 R0)

    end

    always
        #10 Clk = ~Clk;

endmodule
```

✧ **Decoder：**

```
`timescale 1ns / 1ps
module Decoder(Clk, Reset, Instruction);
    input wire Clk;
    input wire Reset;
    input wire [31:0] Instruction;

    reg MemWrite;                   //enable memory write
    reg MemRead;                    //enable memory
    reg [3:0] Addr;                 //memory address
    reg [31:0] WriteMemData;        //data will store in memory
    wire [31:0] ReadData;           //data read from memory

    DataMemory uut2(
        .Clk(Clk),
        .Reset(Reset),
        .Instruction(Instruction),
        .MemWrite(MemWrite),
        .MemRead(MemRead),
        .Addr(Addr),
        .WriteData(WriteMemData),
        .ReadData(ReadData)
    );

    reg RegWrite;                   //enable register write
    reg [1:0] WriteReg;             //write register address
    reg [1:0] ReadReg1;             //read register address1
    reg [1:0] ReadReg2;             //read register address2
    reg [31:0] WriteRegData;        //data will store load in register
    wire [31:0] ReadData1;          //data read from register1
    wire [31:0] ReadData2;          //data read from register2

    RegisterFile uut3(
        .Clk(Clk),
        .Reset(Reset),
        .Instruction(Instruction),
        .RegWrite(RegWrite),
        .WriteReg(WriteReg),
        .ReadReg1(ReadReg1),
        .ReadReg2(ReadReg2),
        .WriteData(WriteRegData),
        .ReadData1(ReadData1),
```

```
        .ReadData2(ReadData2)
);

reg [31:0] ALUOp;
reg [31:0] A,B;
wire [31:0] ALUResult;

ALU uut4(
        .Clk(Clk),
        .Reset(Reset),
        .ALUOp(ALUOp),
        .A(A),
        .B(B),
        .ALUResult(ALUResult)
);

always@(posedge Clk)                    //to set enable flag
begin
        if(Reset==1)                    //initial
        begin
                MemWrite <= 0;
                MemRead <= 0;
                RegWrite <= 0;
        end
        else if(Instruction[31:24]==8'b1000_0000)
        begin
                MemWrite <= 0;
                MemRead <= 1;
                RegWrite <= 1;
        end
        else if(Instruction[31:24]==8'b0100_0000)
        begin
                MemWrite <= 1;
                MemRead <= 0;
                RegWrite <= 0;
        end
        else if(Instruction[31:24]==8'b0010_0000)
        begin                           //Add just can write to register
                MemWrite <= 0;
                MemRead <= 0;
                RegWrite <= 1;
        end
```

```
        else if(Instruction[31:24]==8'b0001_0000)
        begin                                   //Sub just can write to register
            MemWrite <= 0;
            MemRead <= 0;
            RegWrite <= 1;
        end
        else if(Instruction[31:24]==8'b0000_1000)
        begin                                   //Shl just can write to register
            MemWrite <= 0;
            MemRead <= 0;
            RegWrite <= 1;
        end
        else if(Instruction[31:24]==8'b0000_0100)
        begin                                   //Shr just can write to register
            MemWrite <= 0;
            MemRead <= 0;
            RegWrite <= 1;
        end
    end

    always@(posedge Clk)
    begin
        if(Reset==0 && Instruction[31:24]==8'b1000_0000)
        begin
            Addr[3:0] <= Instruction[19:16];
            WriteReg <= Instruction[9:8];
            WriteRegData <= ReadData;
        end
        else if(Reset==0 && Instruction[31:24]==8'b0100_0000)
        begin
            Addr[3:0] <= Instruction[19:16];
            ReadReg1 <= Instruction[9:8];
            ReadReg2 <= Instruction[1:0];
            WriteMemData <= ReadData1;
        end
        else if(Reset==0 && Instruction[31:24]==8'b0010_0000)
        begin                                       //Prepare for Add
            WriteReg <= Instruction[1:0];           //rd number
            ReadReg1 <= Instruction[17:16];         //rs number
            ReadReg2 <= Instruction[9:8];           //rt number
            A <= ReadData1;                         //value in rs
            B <= ReadData2;                         //value in rt
```

```
            WriteRegData <= ALUResult;              //Add result
            ALUOp <= 2'b00;                         //select operation
        end
        else if(Reset==0 && Instruction[31:24]==8'b0001_0000)
        begin                                       //Prepare for Sub
            WriteReg <= Instruction[1:0];           //rd number
            ReadReg1 <= Instruction[17:16];         //rs number
            ReadReg2 <= Instruction[9:8];           //rt number
            A <= ReadData1;                         //value in rs
            B <= ReadData2;                         //value in rt
            WriteRegData <= ALUResult;              //Sub result
            ALUOp <= 2'b01;                         //select operation
        end
        else if(Reset==0 && Instruction[31:24]==8'b0000_1000)
        begin                                       //Prepare for Shl
            WriteReg <= Instruction[1:0];           //rd number
            ReadReg1 <= Instruction[17:16];         //rs number
            A <= ReadData1;                         //value in rs
            B <= {8'b0000_0000,Instruction[15:8]}; //number of shift left
            WriteRegData <= ALUResult;              //Shl result
            ALUOp <= 2'b10;                         //select operation
        end
        else if(Reset==0 && Instruction[31:24]==8'b0000_0100)
        begin                                       //Prepare for Shr
            WriteReg <= Instruction[1:0];           //rd number
            ReadReg1 <= Instruction[17:16];         //rs number
            A <= ReadData1;                         //value in rs
            B <= {8'b0000_0000,Instruction[15:8]}; //number of shift right
            WriteRegData <= ALUResult;              //Shr result
            ALUOp <= 2'b11;                         //select operation
        end
    end

endmodule
```

✧ **ALU：**

```
`timescale 1ns / 1ps
module ALU(Clk, Reset, ALUOp, A, B, ALUResult,
);
input wire Clk;                             // clock signal
input wire Reset;
input wire [1:0]ALUOp;                      // control signal
```

```
    input wire [31:0]A,B;                   //operands
    output reg [31:0]ALUResult;             //result

    always@(posedge Clk)
    begin
        if(Reset==1)
        begin
            ALUResult <= 0;                 //initial
        end
        else
        begin
            if(ALUOp == 2'b00)
            begin
                ALUResult <= (A+B);         //Add
            end
            else if(ALUOp == 2'b01)
            begin
                ALUResult <= (A-B);         //Sub
            end
            else if(ALUOp == 2'b10)
            begin
                ALUResult <= (A << B);      //Shift left
            end
            else if(ALUOp == 2'b11)
            begin
                ALUResult <= (A >> B);      //Shift right
            end
        end
    end

endmodule
```

實作說明：

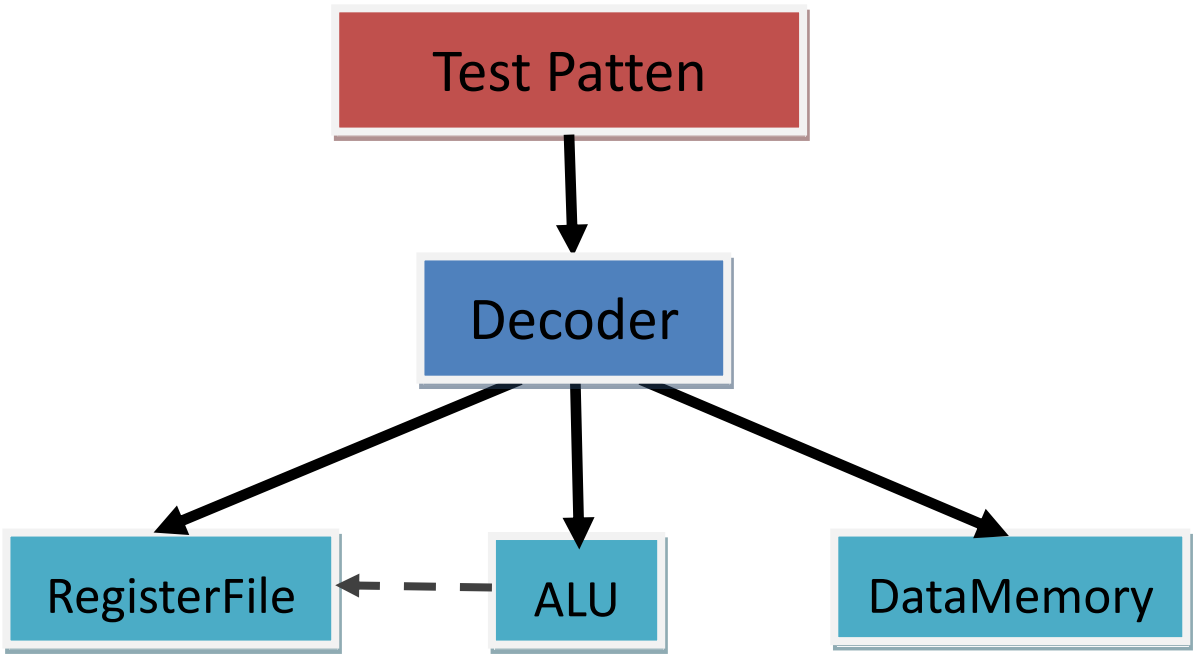
decoder_test.v 檔為 test pattern 負責做 clk,reset 和下指令到 instruction 中。

decoder.v 檔則負責在接到指令後做 set enable flag 的動作，且把需要的相關資訊，例如 memory address、register address 放到暫存器中傳至連結的 RegisterFile、DataMemory 和 ALU 中。

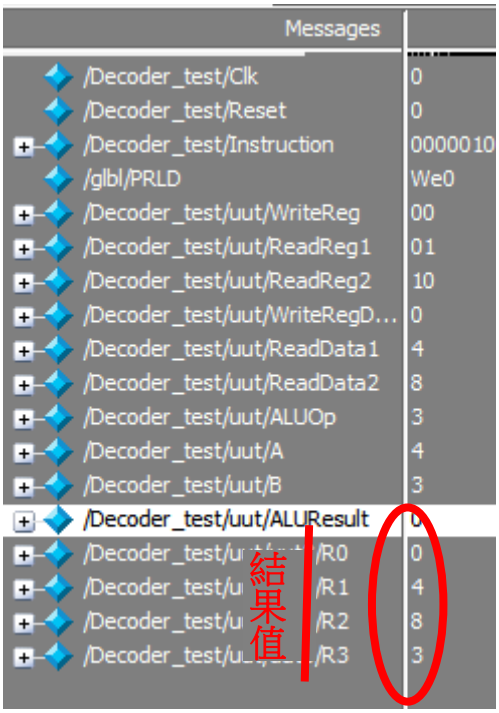
RegisterFile.v 接收從 Decoder 來的值做 input，看 enable flag 的值做 read or write。

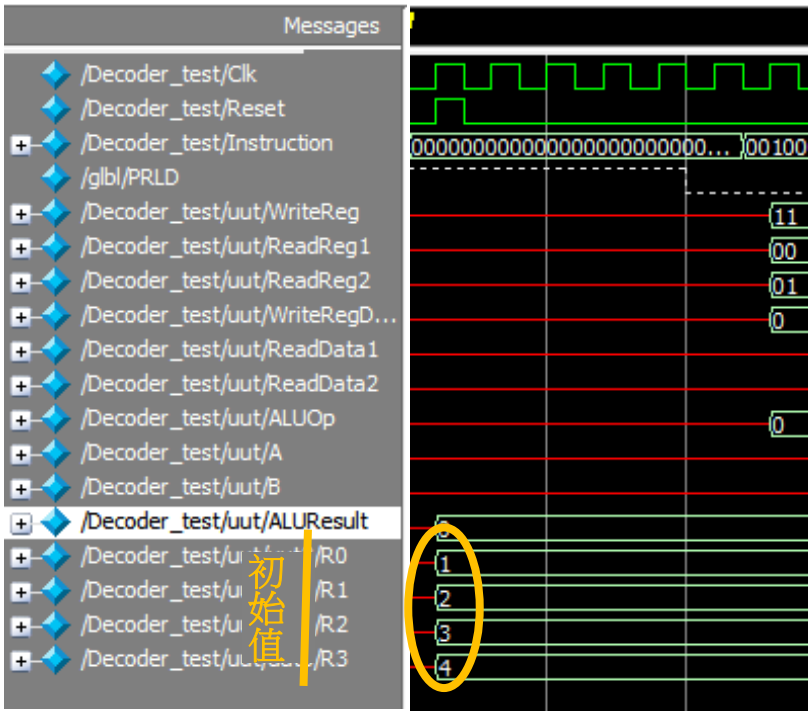
DataMemory.v 接收從 Decoder 來的值做 input，看 enable flag 的值做 read or write。
(DataMemory 在這次 lab2 沒用到)

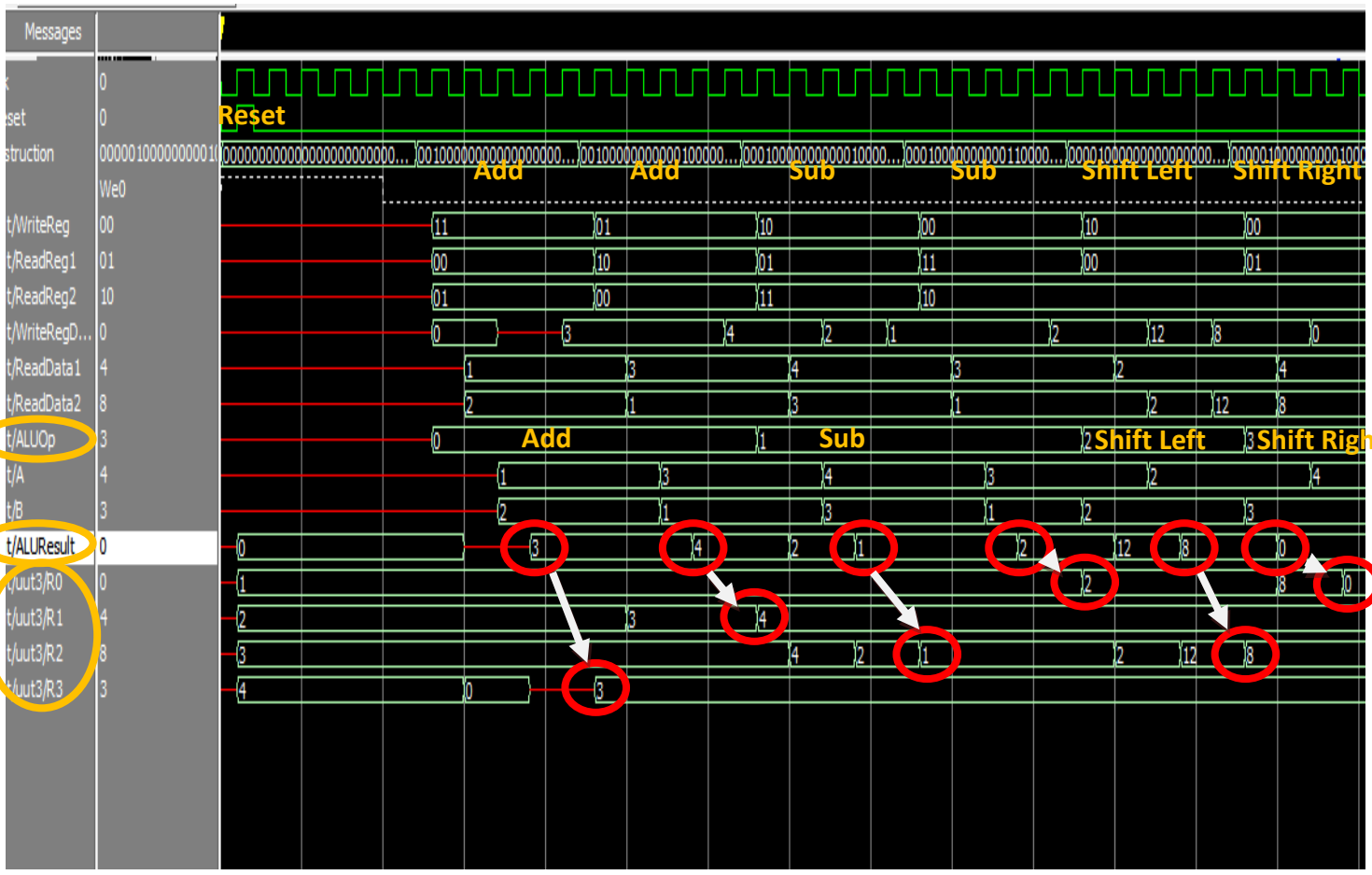
ALU.v 接收從 Decoder 來的 ALUOp 決定要將 A、B 做 Add、Sub、Shift Left 或 Shift Right 並將結果值存入 ALUResult 傳回給 Decoder 再作存值得動作(傳至 RegisterFile 中)。



波型圖：


Messages
/Decoder_test/Clk
/Decoder_test/Reset
/Decoder_test/Instruction
/glbl/PRLD
/Decoder_test/uut/WriteReg
/Decoder_test/uut/ReadReg1
/Decoder_test/uut/ReadReg2
/Decoder_test/uut/WriteRegD...
/Decoder_test/uut/ReadData1
/Decoder_test/uut/ReadData2
/Decoder_test/uut/ALUOp
/Decoder_test/uut/A
/Decoder_test/uut/B
/Decoder_test/uut/ALUResult
/R0
/R1
/R2
/R3
結果值


Messages
/Decoder_test/Clk
/Decoder_test/Reset
/Decoder_test/Instruction
/glbl/PRLD
/Decoder_test/uut/WriteReg
/Decoder_test/uut/ReadReg1
/Decoder_test/uut/ReadReg2
/Decoder_test/uut/WriteRegD...
/Decoder_test/uut/ReadData1
/Decoder_test/uut/ReadData2
/Decoder_test/uut/ALUOp
/Decoder_test/uut/A
/Decoder_test/uut/B
/Decoder_test/uut/ALUResult
/R0
/R1
/R2
/R3
初始值


Reset
Add
Add
Sub
Sub
Shift Left
Shift Right
Add
Sub
Shift Left
Shift Right
t/ALUOp
t/ALUResult