計算機組織 Lab1

9617145 許晏峻

程式碼：

✧ **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;
        #60 Instruction=32'b10000000_00000000_00000011_00000000;
        #60 Instruction=32'b10000000_00000001_00000010_00000000;
        #60 Instruction=32'b10000000_00000010_00000001_00000000;
        #60 Instruction=32'b10000000_00000011_00000000_00000000;

        #60 Instruction=32'b01000000_00000111_00000001_00000000;
        #60 Instruction=32'b01000000_00000110_00000000_00000000;
        #60 Instruction=32'b01000000_00000101_00000010_00000000;
        #60 Instruction=32'b01000000_00000100_00000011_00000000;

    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)
);

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
end

always@(posedge Clk)
begin
        if(Reset==0)                    //initial
        begin
                Addr[3:0] <= Instruction[19:16];
        end
        else if(Reset==0 && Instruction[31:24]==8'b1000_0000)
        begin                           //set memory address
                WriteReg <= Instruction[9:8];
                WriteRegData <= ReadData;       //load memory
        end
        else if(Reset==0 && Instruction[31:24]==8'b0100_0000)
        begin                           //set register address
                ReadReg1 <= Instruction[9:8];
```

```
            ReadReg2 <= Instruction[1:0];
            WriteMemData <= ReadData1;      //store memory
        end
    end

endmodule
```

✧ **RegisterFile：**

```
`timescale 1ns / 1ps
module RegisterFile(Clk, Reset, Instruction, RegWrite, ReadReg1, ReadReg2, WriteReg, WriteData,
ReadData1, ReadData2);
    input wire Clk; I
    input wire Reset;
    input wire [31:0] Instruction;
    input wire RegWrite;
    input wire [1:0] ReadReg1;
    input wire [1:0] ReadReg2;
    input wire [1:0] WriteReg;
    input wire [31:0] WriteData;
    output reg [31:0] ReadData1;
    output reg [31:0] ReadData2;
    reg [31:0] R0,R1,R2,R3;                         //register storage

    always@(posedge Clk)
    begin
        if(Reset==1)
        begin                                       //initial
            R0 <= 0;
            R1 <= 0;
            R2 <= 0;
            R3 <= 0;
        end
        else if(Reset==0 && RegWrite==1)
        begin                                       //Write in register
            case({WriteReg})
                2'b00:R0[31:0] <= WriteData[31:0];
                2'b01:R1[31:0] <= WriteData[31:0];
                2'b10:R2[31:0] <= WriteData[31:0];
                2'b11:R3[31:0] <= WriteData[31:0];
            endcase
        end
        else if(Reset==0 && RegWrite==0)
```

```
        begin                                   //read register1
            case({ReadReg1})
                2'b00:ReadData1[31:0] <= R0[31:0];
                2'b01:ReadData1[31:0] <= R1[31:0];
                2'b10:ReadData1[31:0] <= R2[31:0];
                2'b11:ReadData1[31:0] <= R3[31:0];
            endcase
            case({ReadReg2})                    //read register2
                2'b00:ReadData2[31:0] <= R0[31:0];
                2'b01:ReadData2[31:0] <= R1[31:0];
                2'b10:ReadData2[31:0] <= R2[31:0];
                2'b11:ReadData2[31:0] <= R3[31:0];
            endcase
        end
    end

endmodule
```

✧ **DataMemory：**

```
`timescale 1ns / 1ps
module DataMemory(Clk, Reset, Instruction, MemWrite, MemRead, Addr, WriteData, ReadData);
    input wire Clk;
    input wire Reset;
    input wire [31:0] Instruction;
    input wire MemWrite;
    input wire MemRead;
    input wire [3:0] Addr;
    input wire [31:0] WriteData;
    output reg [31:0] ReadData;
    reg [31:0] M0,M1,M2,M3,M4,M5,M6,M7;          //Memory Storage

    always@(posedge Clk)
    begin
        if(Reset==1)                            //initial
        begin
            M0 <= 0;
            M1 <= 1;
            M2 <= 2;
            M3 <= 3;
            M4 <= 4;
            M5 <= 5;
            M6 <= 6;
```

```
            M7 <= 7;
        end
        else if(Reset==0 && MemWrite==1 && MemRead==0)
        begin                                           //write in memory
            case({Addr[3:0]})
                4'b0000:M0[31:0] <= WriteData[31:0];
                4'b0001:M1[31:0] <= WriteData[31:0];
                4'b0010:M2[31:0] <= WriteData[31:0];
                4'b0011:M3[31:0] <= WriteData[31:0];
                4'b0100:M4[31:0] <= WriteData[31:0];
                4'b0101:M5[31:0] <= WriteData[31:0];
                4'b0110:M6[31:0] <= WriteData[31:0];
                4'b0111:M7[31:0] <= WriteData[31:0];
            endcase
        end
        else if(Reset==0 && MemRead==1 && MemWrite==0)
        begin                                           //load memory
            case({Addr[3:0]})
                4'b0000:ReadData[31:0] <= M0[31:0];
                4'b0001:ReadData[31:0] <= M1[31:0];
                4'b0010:ReadData[31:0] <= M2[31:0];
                4'b0011:ReadData[31:0] <= M3[31:0];
                4'b0100:ReadData[31:0] <= M4[31:0];
                4'b0101:ReadData[31:0] <= M5[31:0];
                4'b0110:ReadData[31:0] <= M6[31:0];
                4'b0111:ReadData[31:0] <= M7[31:0];
            endcase
        end
    end

endmodule
```

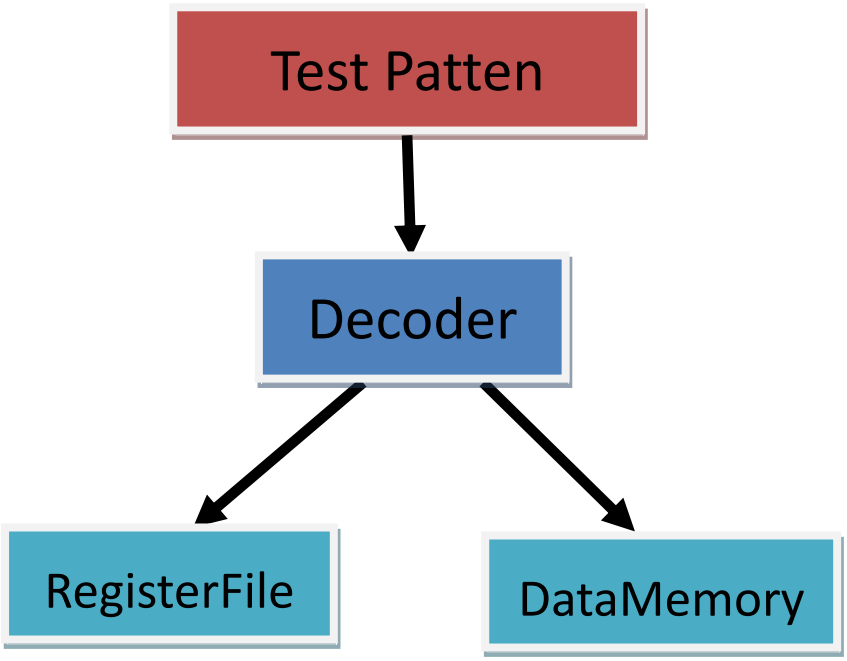
實作說明：

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

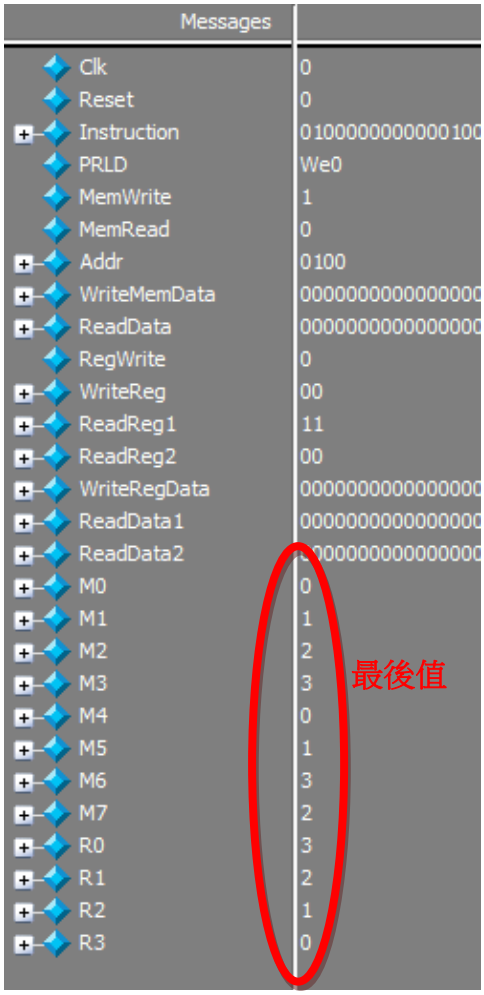
decoder.v 檔則負責在接到指令後做 set enable flag 的動作，且把需要的相關資訊，例如 memory address、register address 放到暫存器中傳至連結的 RegisterFile 和 DataMemory 中；並且做從 Register read 出來的 output 值和要存進 Memory 的暫存器做轉換(WriteRegData <= ReadData)、從 Memory load 出來的 output 值和要存進 Register 的暫存器做轉換(WriteMemData <= ReadData1)

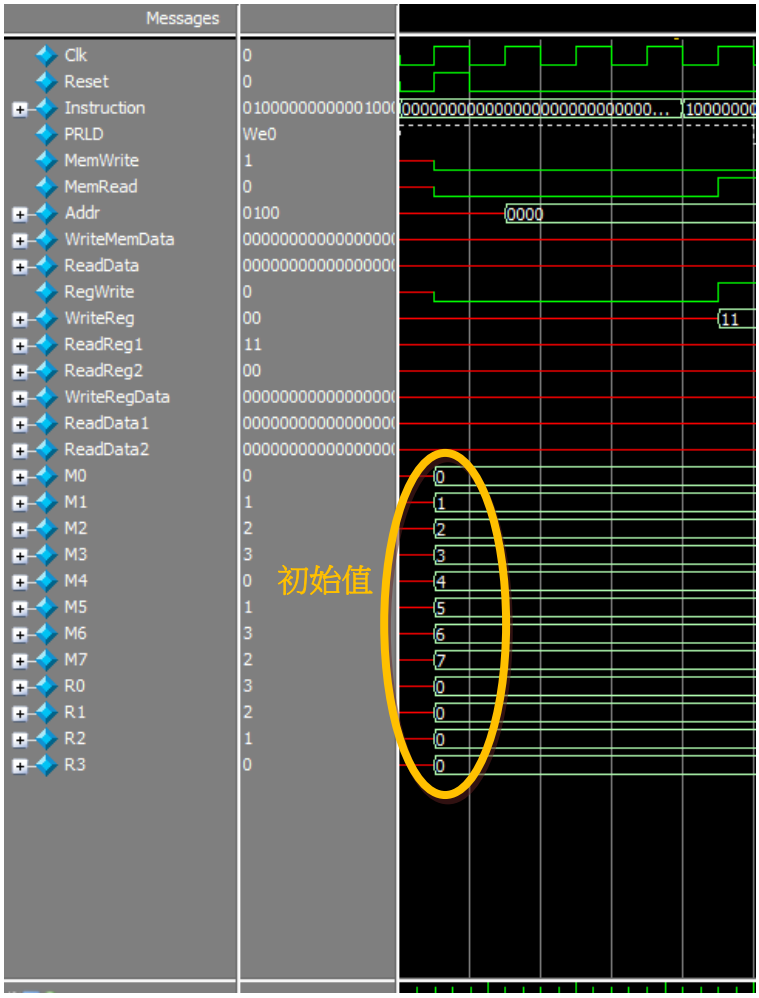
RegisterFile.v 接收從 Decoder 來的值做 input，看 enable flag 的值做 read or write

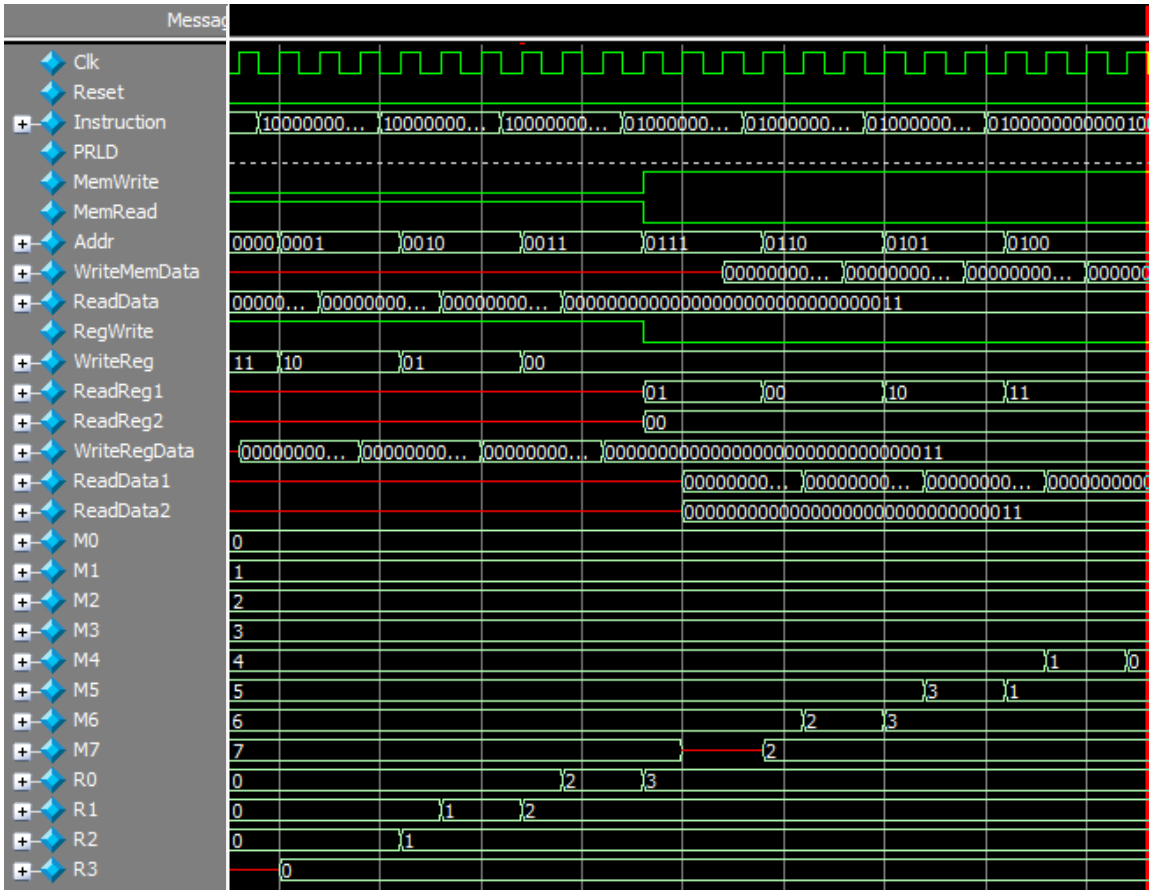
DataMemory.v 接收從 Decoder 來的值做 input，看 enable flag 的值做 read or write



波型圖：


Messages
Clk 0
Reset 0
Instruction 0100000000000100
PRLD We0
MemWrite 1
MemRead 0
Addr 0100
WriteMemData 0000000000000000
ReadData 0000000000000000
RegWrite 0
WriteReg 00
ReadReg1 11
ReadReg2 00
WriteRegData 0000000000000000
ReadData1 0000000000000000
ReadData2 0000000000000000
M0 0
M1 1
M2 2
M3 3
M4 0
M5 1
M6 3
M7 2
R0 3
R1 2
R2 1
R3 0
最後值


Messages
Clk 0
Reset 0
Instruction 0100000000000100
PRLD We0
MemWrite 1
MemRead 0
Addr 0100
WriteMemData 0000000000000000
ReadData 0000000000000000
RegWrite 0
WriteReg 00
ReadReg1 11
ReadReg2 00
WriteRegData 0000000000000000
ReadData1 0000000000000000
ReadData2 0000000000000000
M0 0
M1 1
M2 2
M3 3
M4 0
M5 1
M6 3
M7 2
R0 3
R1 2
R2 1
R3 0
初始值


Clk
Reset
Instruction
PRLD
MemWrite
MemRead
Addr
WriteMemData
ReadData
RegWrite
WriteReg
ReadReg1
ReadReg2
WriteRegData
ReadData1
ReadData2
M0
M1
M2
M3
M4
M5
M6
M7
R0
R1
R2
R3