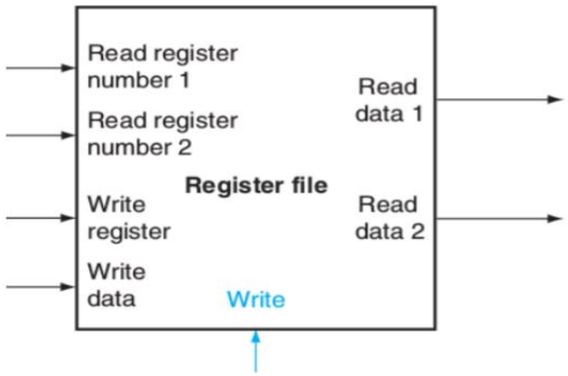
Register File:

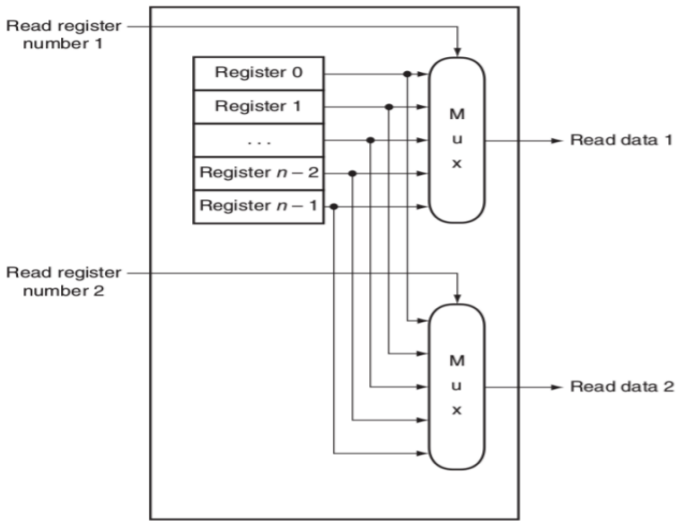
1. 32 Registers.
2. Read Register No 1 (Address of the register to read data from).
3. Read Register No 2 (Address of the register to read data from).
4. Read Data 1 (Data from the register pointed to by read register no 1).
5. Read Data 2 (Data from the register pointed to by read register no 2).
6. Write signal (Corresponds to writing data or not).
7. Write Register (Address of the data to which data has to be written)
8. Write data (Data to be written in the register pointed by write register).



Verilog Code:

```
// Register file thread instantated
RegisterFile inst ( write, reg_data, register_no, readReg1, readReg2, readData1, readData2);
```

Reading from Register File:



Verilog Code:

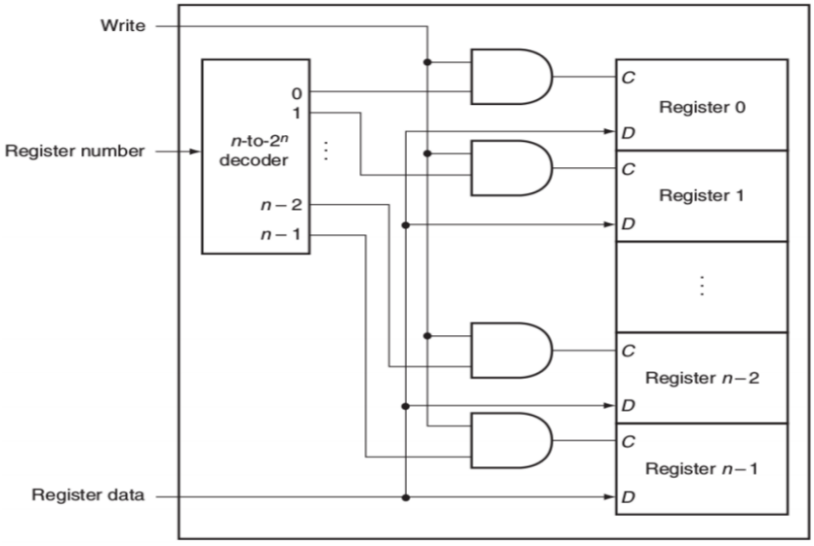
```
else begin
    // decode read Register 1 address
    case (readReg1)
        5'b00000: begin write1 <= 1'b0; readData1 <= out1; end
        5'b00001: begin write2 <= 1'b0; readData1 <= out2; end
        5'b00010: begin write3 <= 1'b0; readData1 <= out3; end
        5'b00011: begin write4 <= 1'b0; readData1 <= out4; end
        5'b00100: begin write5 <= 1'b0; readData1 <= out5; end
        5'b00101: begin write6 <= 1'b0; readData1 <= out6; end
        5'b00110: begin write7 <= 1'b0; readData1 <= out7; end
        5'b00111: begin write8 <= 1'b0; readData1 <= out8; end
        5'b01000: begin write8 <= 1'b0; readData1 <= out9; end
        5'b01001: begin write10 <= 1'b0; readData1 <= out10; end
        5'b01010: begin write11 <= 1'b0; readData1 <= out11; end
        5'b01011: begin write12 <= 1'b0; readData1 <= out12; end
        5'b01100: begin write13 <= 1'b0; readData1 <= out13 ; end
        5'b01101: begin write14 <= 1'b0; readData1 <= out14 ; end
        5'b01110: begin write15 <= 1'b0; readData1 <= out15 ; end
        5'b01111: begin write16 <= 1'b0; readData1 <= out16 ; end
        5'b10000: begin write17 <= 1'b0; readData1 <= out17 ; end
        5'b10001: begin write18 <= 1'b0; readData1 <= out18 ; end
        5'b10010: begin write19 <= 1'b0; readData1 <= out19 ; end
        5'b10011: begin write20 <= 1'b0; readData1 <= out20; end
        5'b10100: begin write21 <= 1'b0; readData1 <= out21; end
        5'b10101: begin write22 <= 1'b0; readData1 <= out22; end
        5'b10110: begin write23 <= 1'b0; readData1 <= out23; end
        5'b10111: begin write24 <= 1'b0; readData1 <= out24; end
        5'b11000: begin write25 <= 1'b0; readData1 <= out25; end
        5'b11001: begin write26 <= 1'b0; readData1 <= out26; end
        5'b11010: begin write27 <= 1'b0; readData1 <= out27; end
        5'b11011: begin write28 <= 1'b0; readData1 <= out28; end
        5'b11100: begin write29 <= 1'b0; readData1 <= out29; end
        5'b11101: begin write30 <= 1'b0; readData1 <= out30; end
        5'b11110: begin write31 <= 1'b0; readData1 <= out31; end
        5'b11111: begin write32 <= 1'b0; readData1 <= out32; end
    endcase
```

```
// decode read Register 2 address
case (readReg2)
    5'b00000: begin write1 <= 1'b0; readData2 <= out1; end
    5'b00001: begin write2 <= 1'b0; readData2 <= out2; end
    5'b00010: begin write3 <= 1'b0; readData2 <= out3; end
    5'b00011: begin write4 <= 1'b0; readData2 <= out4; end
    5'b00100: begin write5 <= 1'b0; readData2 <= out5; end
    5'b00101: begin write6 <= 1'b0; readData2 <= out6; end
    5'b00110: begin write7 <= 1'b0; readData2 <= out7; end
    5'b00111: begin write8 <= 1'b0; readData2 <= out8; end
    5'b01000: begin write8 <= 1'b0; readData2 <= out9; end
    5'b01001: begin write10 <= 1'b0; readData2 <= out10; end
    5'b01010: begin write11 <= 1'b0; readData2 <= out11; end
    5'b01011: begin write12 <= 1'b0; readData2 <= out12; end
    5'b01100: begin write13 <= 1'b0; readData2 <= out13; end
    5'b01101: begin write14 <= 1'b0; readData2 <= out14; end
    5'b01110: begin write15 <= 1'b0; readData2 <= out15; end
    5'b01111: begin write16 <= 1'b0; readData2 <= out16; end
    5'b10000: begin write17 <= 1'b0; readData2 <= out17; end
    5'b10001: begin write18 <= 1'b0; readData2 <= out18; end
    5'b10010: begin write19 <= 1'b0; readData2 <= out19; end
    5'b10011: begin write20 <= 1'b0; readData2 <= out20; end
    5'b10100: begin write21 <= 1'b0; readData2 <= out21; end
    5'b10101: begin write22 <= 1'b0; readData2 <= out22; end
    5'b10110: begin write23 <= 1'b0; readData2 <= out23; end
    5'b10111: begin write24 <= 1'b0; readData2 <= out24; end
    5'b11000: begin write25 <= 1'b0; readData2 <= out25; end
    5'b11001: begin write26 <= 1'b0; readData2 <= out26; end
    5'b11010: begin write27 <= 1'b0; readData2 <= out27; end
    5'b11011: begin write28 <= 1'b0; readData2 <= out28; end
    5'b11100: begin write29 <= 1'b0; readData2 <= out29; end
    5'b11101: begin write30 <= 1'b0; readData2 <= out30; end
    5'b11110: begin write31 <= 1'b0; readData2 <= out31; end
    5'b11111: begin write32 <= 1'b0; readData2 <= out32; end
endcase
```

Explanation

The multiplexer operation for decoding the address and 32 lines each of 32 bit wide is done using behavioural model in verilog. Decoding the address is done using switch cases in Verilog

Writing to a Register File :



Verilog Code:

```
// check write signal
if(write == 1'b1) begin
    // Decode register address
    case (register_no)
        5'b00000: begin write1 <= 1'b1; data_write1 <= reg_data; end
        5'b00001: begin write2 <= 1'b1; data_write2 <= reg_data; end
        5'b00010: begin write3 <= 1'b1; data_write3 <= reg_data; end
        5'b00011: begin write4 <= 1'b1; data_write4 <= reg_data; end
        5'b00100: begin write5 <= 1'b1; data_write5 <= reg_data; end
        5'b00101: begin write6 <= 1'b1; data_write6 <= reg_data; end
        5'b00110: begin write7 <= 1'b1; data_write7 <= reg_data; end
        5'b00111: begin write8 <= 1'b1; data_write8 <= reg_data; end
        5'b01000: begin write8 <= 1'b1; data_write9 <= reg_data; end
        5'b01001: begin write10 <= 1'b1; data_write10 <= reg_data; end
        5'b01010: begin write11 <= 1'b1; data_write11 <= reg_data; end
        5'b01011: begin write12 <= 1'b1; data_write12 <= reg_data; end
        5'b01100: begin write13 <= 1'b1; data_write13 <= reg_data; end
        5'b01101: begin write14 <= 1'b1; data_write14 <= reg_data; end
        5'b01110: begin write15 <= 1'b1; data_write15 <= reg_data; end
        5'b01111: begin write16 <= 1'b1; data_write16 <= reg_data; end
        5'b10000: begin write17 <= 1'b1; data_write17 <= reg_data; end
        5'b10001: begin write18 <= 1'b1; data_write18 <= reg_data; end
        5'b10010: begin write19 <= 1'b1; data_write19 <= reg_data; end
        5'b10011: begin write20 <= 1'b1; data_write20 <= reg_data; end
        5'b10100: begin write21 <= 1'b1; data_write21 <= reg_data; end
        5'b10101: begin write22 <= 1'b1; data_write22 <= reg_data; end
        5'b10110: begin write23 <= 1'b1; data_write23 <= reg_data; end
        5'b10111: begin write24 <= 1'b1; data_write24 <= reg_data; end
        5'b11000: begin write25 <= 1'b1; data_write25 <= reg_data; end
        5'b11001: begin write26 <= 1'b1; data_write26 <= reg_data; end
        5'b11010: begin write27 <= 1'b1; data_write27 <= reg_data; end
        5'b11011: begin write28 <= 1'b1; data_write28 <= reg_data; end
        5'b11100: begin write29 <= 1'b1; data_write29 <= reg_data; end
        5'b11101: begin write30 <= 1'b1; data_write30 <= reg_data; end
        5'b11110: begin write31 <= 1'b1; data_write31 <= reg_data; end
        5'b11111: begin write32 <= 1'b1; data_write32 <= reg_data; end
    endcase
end

else begin
```

Explanation:

The decoding of which register to write to, while taking care of the write signal is done using register is done using cases in behavioural model in Verilog.

Other modules :-

32 Bit Register:

```
// 32 bit Register
module rg32b (input [0:31] data_write , input write, output reg [0:31] data);

    // initialize data in registers
    initial begin
        data <= 32'b00000000000000000000000000000000;
    end

    always @ (write or data_write) begin
        // check write signal
        if (write == 1'b1) begin
            // write data
            data <= data_write;
        end
        else begin
            // memory state
            data <= data;
        end
    end

endmodule
```

32 Registers in RF:

```
    // 32 Registers in the Register file
    rg32b rg1 (data_write1, write1, out1);
    rg32b rg2 (data_write2, write2, out2);
    rg32b rg3 (data_write3, write3, out3);
    rg32b rg4 (data_write4, write4, out4);
    rg32b rg5 (data_write5, write5, out5);
    rg32b rg6 (data_write6, write6, out6);
    rg32b rg7 (data_write7, write7, out7);
    rg32b rg8 (data_write8, write8, out8);
    rg32b rg9 (data_write9, write9, out9);
    rg32b rg10 (data_write10, write10, out10);
    rg32b rg11 (data_write11, write11, out11);
    rg32b rg12 (data_write12, write12, out12);
    rg32b rg13 (data_write13, write13, out13);
    rg32b rg14 (data_write14, write14, out14);
    rg32b rg15 (data_write15, write15, out15);
    rg32b rg16 (data_write16, write16, out16);
    rg32b rg17 (data_write17, write17, out17);
    rg32b rg18 (data_write18, write18, out18);
    rg32b rg19 (data_write19, write19, out19);
    rg32b rg20 (data_write20, write20, out20);
    rg32b rg21 (data_write21, write21, out21);
    rg32b rg22 (data_write22, write22, out22);
    rg32b rg23 (data_write23, write23, out23);
    rg32b rg24 (data_write24, write24, out24);
    rg32b rg25 (data_write25, write25, out25);
    rg32b rg26 (data_write26, write26, out26);
    rg32b rg27 (data_write27, write27, out27);
    rg32b rg28 (data_write28, write28, out28);
    rg32b rg29 (data_write29, write29, out29);
    rg32b rg30 (data_write30, write30, out30);
    rg32b rg31 (data_write31, write31, out31);
    rg32b rg32 (data_write32, write32, out32);
```