計算機系統設計

乘除法指令

Mao-Hsu Yen
yenmh@mail.ntou.edu.tw

RISC-V RV32IM INSTRUCTION SET

Multiplication and Division Operations

Mnemonic, Operands	Description	Implementation
MUL rd, rs1, rs2	Performs an **32-bit × 32-bit** multiplication of **signed** *rs1* by **signed** *rs2* and places the **lower 32 bits** in the destination register.	$x[rd] = (x[rs1]_s \times_s x[rs2])[31:0]$
MULH rd, rs1, rs2	Performs an **32-bit × 32-bit** multiplication of **signed** *rs1* by **signed** *rs2* and places the **upper 32 bits** in the destination register.	$x[rd] = (x[rs1]_s \times_s x[rs2])[63:32]$
MULHSU rd, rs1, rs2	Performs an **32-bit × 32-bit** multiplication of **signed** *rs1* by **unsigned** *rs2* and places the **upper 32 bits** in the destination register.	$x[rd] = (x[rs1]_s \times_u x[rs2])[63:32]$
MULHU rd, rs1, rs2	Performs an **32-bit × 32-bit** multiplication of **unsigned** *rs1* by **unsigned** *rs2* and places the **upper 32 bits** in the destination register.	$x[rd] = (x[rs1]_u \times_u x[rs2])[63:32]$
DIV rd, rs1, rs2	perform an **32 bits by 32bits signed integer division** of *rs1* by *rs2*, rounding towards zero.	$x[rd] = x[rs1] /_s x[rs2]$
DIVU rd, rs1, rs2	perform an **32 bits by 32 bits unsigned integer division** of *rs1* by *rs2*, rounding towards zero.	$x[rd] = x[rs1] /_u x[rs2]$
REM rd, rs1, rs2	perform an **32 bits by 32 bits signed integer reminder** of *rs1* by *rs2*.	$x[rd] = x[rs1] \%_s x[rs2]$
REMU rd, rs1, rs2	perform an **32 bits by 32 bits unsigned integer reminder** of *rs1* by *rs2*.	$x[rd] = x[rs1] \%_u x[rs2]$

※ m bits 乘以 n bits 最大為 m + n 個 bits

RISC-V RV32IM INSTRUCTION SET

Multiplication and Division Instructions

Funct7	Source registers 2	Source registers 1	Funct3	Destination registers	Opcode	Instruction
0000001	rs2	rs1	000	rd	0110011	MUL
0000001	rs2	rs1	001	rd	0110011	MULH
0000001	rs2	rs1	010	rd	0110011	MULHSU
0000001	rs2	rs1	011	rd	0110011	MULHU
0000001	rs2	rs1	100	rd	0110011	DIV
0000001	rs2	rs1	101	rd	0110011	DIVU
0000001	rs2	rs1	110	rd	0110011	REM
0000001	rs2	rs1	111	rd	0110011	REMU

31 25 24 20 19 15 14 12 11 7 6 0

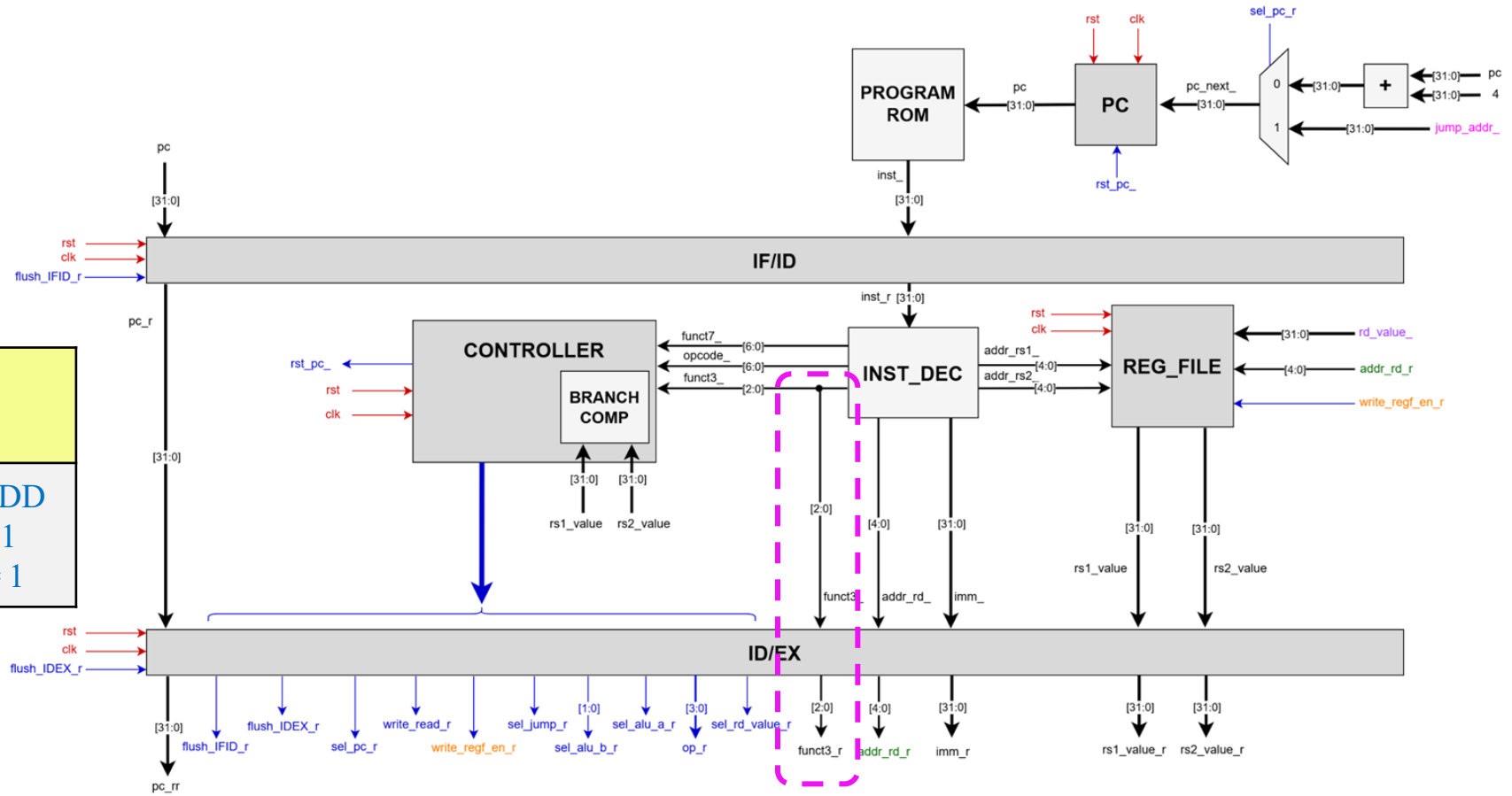
※ **暫存器定址指令**和**乘除法指令**的 **opcode 相同**，須以 **funct7 判別**。乘除法指令之 funct7 為 7‘b000_0001，而暫存器定址指令之 funct7 為 7‘b000_0000 or 7’b010_0000。

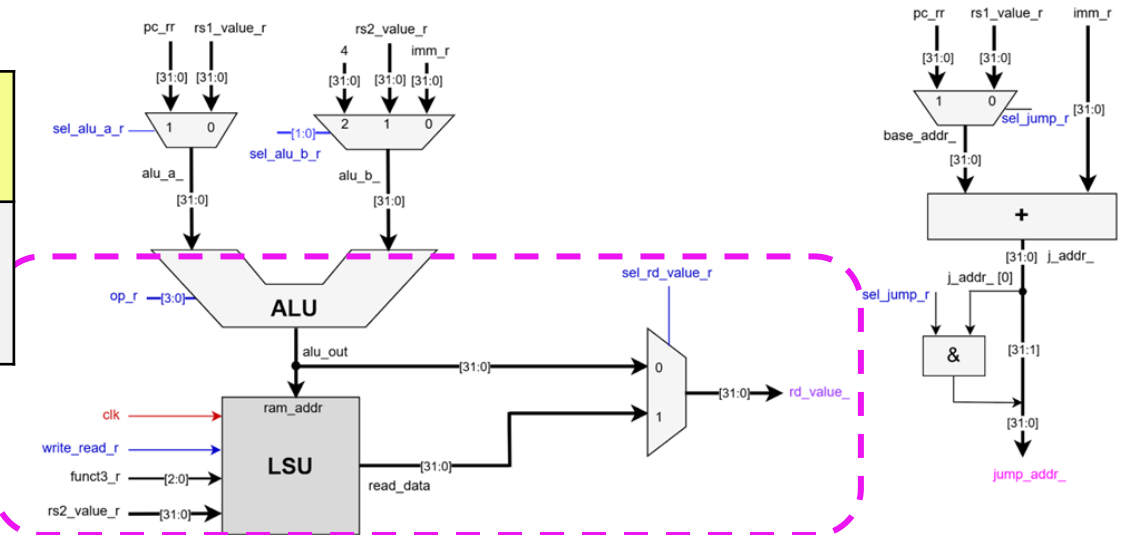
Funct7	Source registers 2	Source registers 1	Funct3	Destination registers	Opcode	Instruction
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB

上周完成之架構

動作	控制訊號
if ((opcode_ == `Opcode_LB) && (funct3_ == `F_LB)) 發出 LB 的控制訊號	op_ = `ALUOP_ADD sel_rd_value_ = 1 write_regf_en_ = 1

動作	控制訊號
※ addr = rs1_value_r + imm_r rd_value_ ← sext(M[addr])	op_r = `ALUOP_ADD sel_rd_value_r = 1 write_regf_en_r = 1

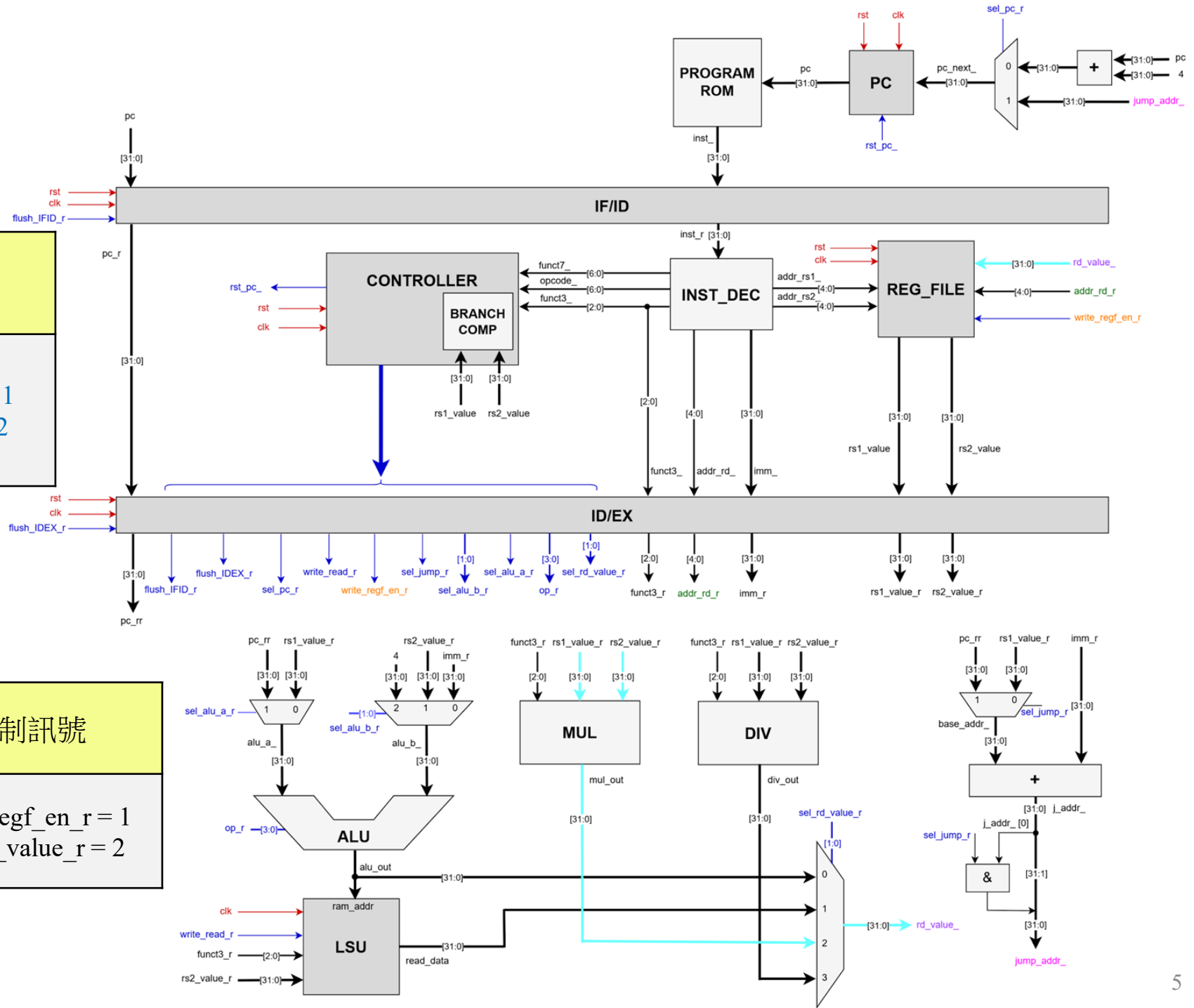




新架構

動作	控制訊號
if ((opcode_ == `Opcode_R_M) && (funct3_ == `F_MUL) && (funct7_ == `F7_R_M)) 發出 LB 的控制訊號	write_regf_en_ = 1 sel_rd_value_ = 2

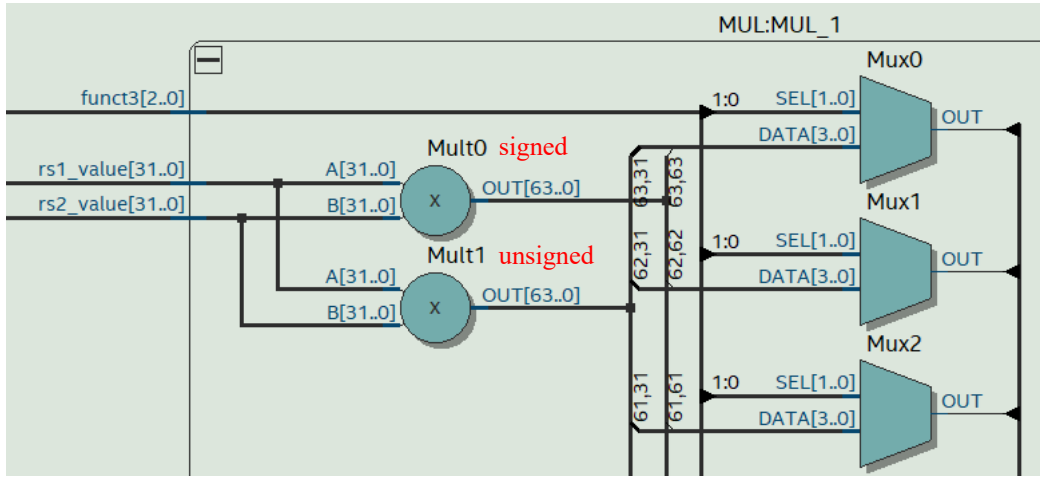
動作	控制訊號
rd_value_ ← (rs1_value_r $_s\times_s$ rs2_value_r) [31:0]	write_regf_en_r = 1 sel_rd_value_r = 2



乘法器

- 原理介紹
  - RISC-V 的 M 指令集(乘除法)支援**有/無號數乘法**以及**有號數乘以無號數**計算。
    - 有號數乘法：若乘數或被乘數為負數，則需先取二補數後再進行運算。
    - 有號數乘以無號數：若被乘數為負數，則需先取二補數後再進行運算。
    - 無號數乘法：直接進行運算即可。
  - 可以使用以下方式撰寫，是一種撰寫相對簡單的寫法，但在合成時會合成兩個硬體元件 (signed and unsigned)。

```
module MUL(
    input   logic [2:0] funct3,
    input   logic [31:0] rs1_value,
    input   logic [31:0] rs2_value,

    output  logic [31:0] mul_out
);

    logic [63:0] product;

    always_comb begin
        unique case (funct3)
            `F_MUL:     product = $signed(rs1_value) * $signed(rs2_value);
            `F_MULH:    product = $signed(rs1_value) * $signed(rs2_value);
            `F_MULHSU:  product = $signed($signed(rs1_value) * rs2_value);
            `F_MULHU:   product = rs1_value * rs2_value;
        endcase
    end

    assign mul_out = (funct3 == `F_MUL) ? product[31:0] : product[63:32];

endmodule
```



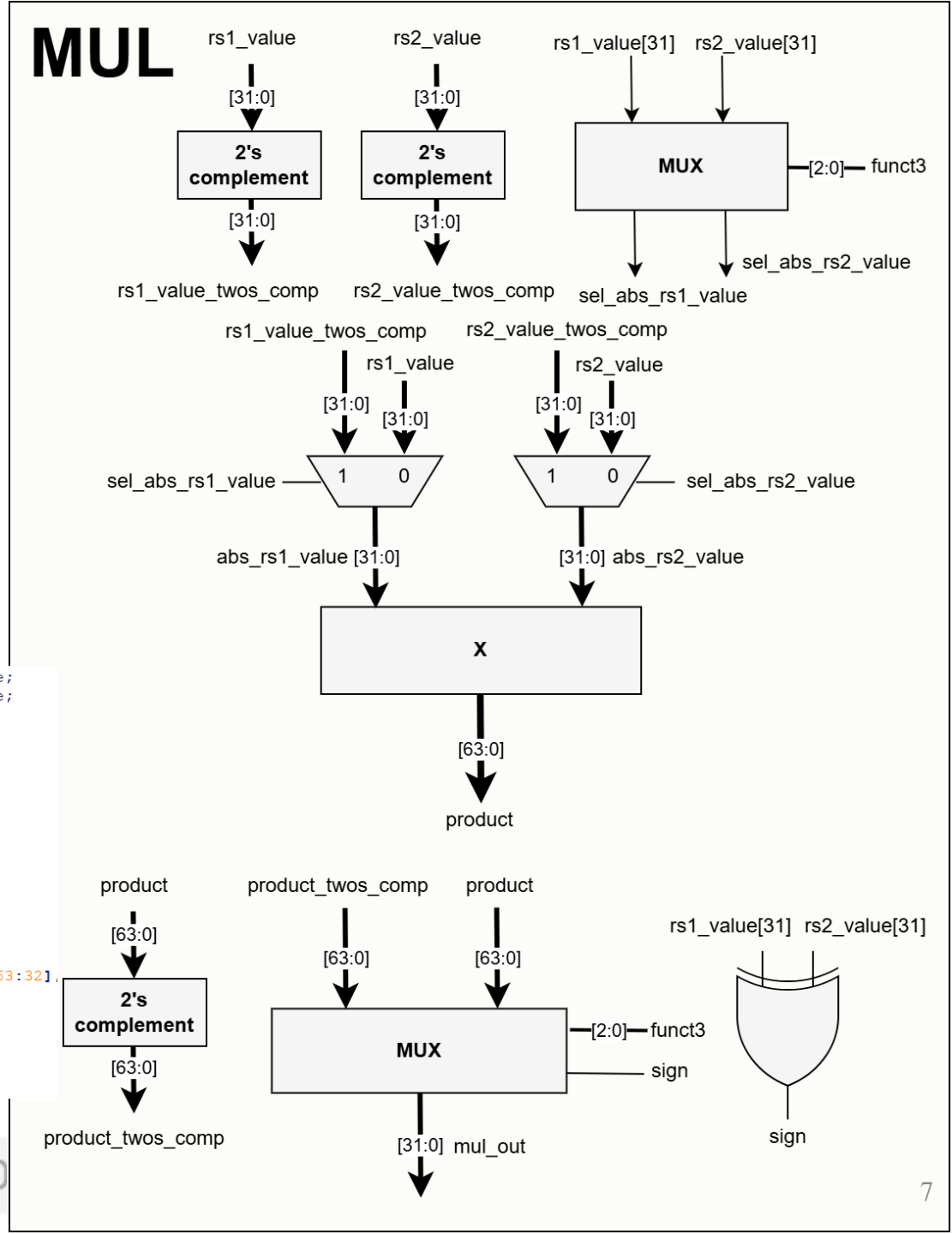
乘法器

- 進階寫法
  - 若輸入 rs1_value 或 rs2_value 為負數則透過多工器選擇二補數作為乘法器的輸入。
  - MUX 根據 rs1_value 或 rs2_value 的 sign bit 做 XOR 運算決定輸出要不要取二補數。

```
logic [31:0] rs1_value_twos_comp;
logic [31:0] rs2_value_twos_comp;
logic [31:0] abs_rs1_value;
logic [31:0] abs_rs2_value;
logic [63:0] product;
logic [63:0] product_twos_comp;
logic sel_abs_rs1_value;
logic sel_abs_rs2_value;
logic sign;

assign sign = rs1_value[31] ^ rs2_value[31];
assign rs1_value_twos_comp = ~rs1_value + 1;
assign rs2_value_twos_comp = ~rs2_value + 1;

always_comb begin
    unique case(funct3)
        `F_MUL: begin
            sel_abs_rs1_value = rs1_value[31];
            sel_abs_rs2_value = rs2_value[31];
        end
        `F_MULH: begin
            sel_abs_rs1_value = rs1_value[31];
            sel_abs_rs2_value = rs2_value[31];
        end
        `F_MULHSU: begin
            sel_abs_rs1_value = rs1_value[31];
            sel_abs_rs2_value = 0;
        end
        `F_MULHU: begin
            sel_abs_rs1_value = 0;
            sel_abs_rs2_value = 0;
        end
    endcase
end
```

Verilog 的乘法為無號數的乘法

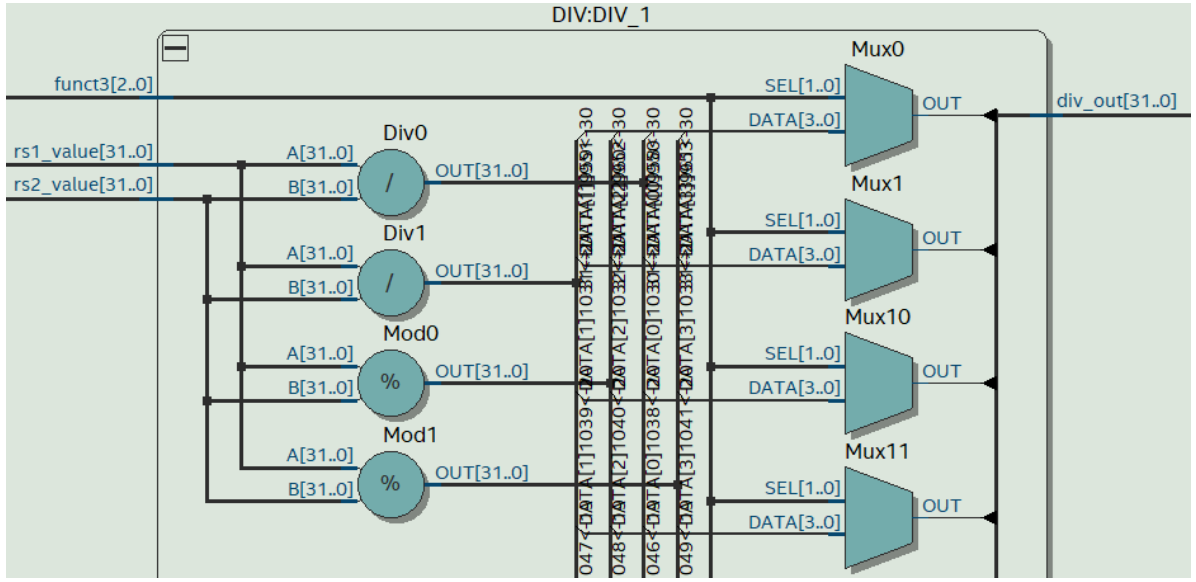
```
assign abs_rs1_value = sel_abs_rs1_value ? rs1_value_twos_comp : rs1_value;
assign abs_rs2_value = sel_abs_rs2_value ? rs2_value_twos_comp : rs2_value;

assign product = abs_rs1_value * abs_rs2_value;

assign product_twos_comp = ~product + 1;

always_comb begin
    unique case(funct3)
        `F_MUL: begin
            mul_out = sign ? product_twos_comp[31:0] : product[31:0];
        end
        `F_MULH: begin
            mul_out = sign ? product_twos_comp[63:32] : product[63:32];
        end
        `F_MULHSU: begin
            mul_out = rs1_value[31] ? product_twos_comp[63:32] : product[63:32];
        end
        `F_MULHU: begin
            mul_out = product[63:32];
        end
    endcase
end
```



除法器

- 原理介紹
  - RISC-V 的 M 指令集(乘除法)支援**有/無號數除法**和**有/無號數餘數**計算。
    - 有號數除法/餘數計算：若除數或被除數為負數，則需先取二補數後再進行運算。
    - 無號數除法/餘數計算：直接進行運算即可。
  - 可以使用以下方式撰寫，是一種撰寫相對簡單的寫法，但在合成時會合成四個硬體元件。

```
module DIV(
    input   logic [2:0] funct3,
    input   logic [31:0] rs1_value,
    input   logic [31:0] rs2_value,

    output  logic [31:0] div_out
);
    always_comb begin
        unique case (funct3)
            `F_DIV:     div_out = $signed(rs1_value) / $signed(rs2_value);
            `F_DIVU:    div_out = rs1_value / rs2_value;
            `F_REM:     div_out = $signed(rs1_value) % $signed(rs2_value);
            `F_REMU:    div_out = rs1_value % rs2_value;
        endcase
    end

endmodule
```

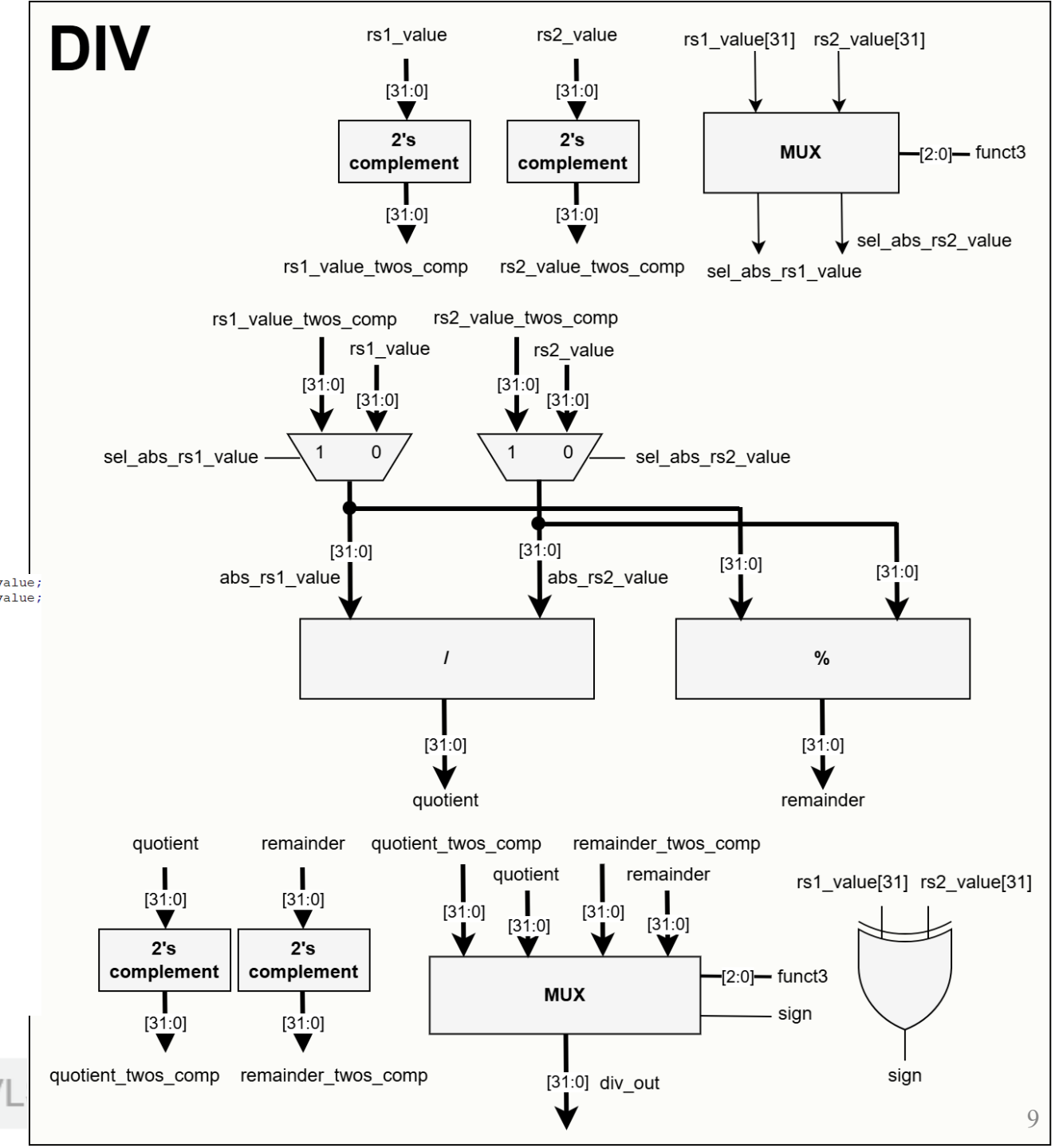


除法器

- 進階寫法
  - 若輸入 rs1_value 或 rs2_value 為負數則透過多工器選擇二補數作為除法器 (/ or %) 的輸入。
  - MUX 根據 rs1_value 或 rs2_value 的 sign bit 做 XOR 運算決定輸出要不要取二補數。

```
logic [31:0] rs1_value_twos_comp;
logic [31:0] rs2_value_twos_comp;
logic [31:0] abs_rs1_value;
logic [31:0] abs_rs2_value;
logic [31:0] quotient;
logic [31:0] remainder;
logic [31:0] quotient_twos_comp;
logic [31:0] remainder_twos_comp;
logic sel_abs_rs1_value;
logic sel_abs_rs2_value;
logic sign;

//二補數
assign rs1_value_twos_comp = ~rs1_value + 1;
assign rs2_value_twos_comp = ~rs2_value + 1;

//根據 funct3 和 sign bit 辨別
//x[rs1] 和 x[rs2] 是否進行二補數轉換
always_comb begin
    case (funct3)
        `F_DIV: begin
            sel_abs_rs1_value = rs1_value[31];
            sel_abs_rs2_value = rs2_value[31];
        end
        `F_REM: begin
            sel_abs_rs1_value = rs1_value[31];
            sel_abs_rs2_value = rs2_value[31];
        end
        default: begin
            sel_abs_rs1_value = 0;
            sel_abs_rs2_value = 0;
        end
    endcase
end
```

```
assign abs_rs1_value = sel_abs_rs1_value ? rs1_value_twos_comp : rs1_value;
assign abs_rs2_value = sel_abs_rs2_value ? rs2_value_twos_comp : rs2_value;

assign quotient = abs_rs1_value / abs_rs2_value;
assign remainder = abs_rs1_value % abs_rs2_value;

//二補數
assign quotient_twos_comp = ~quotient + 1;
assign remainder_twos_comp = ~remainder + 1;

//根據 sign bit 辨別結果是否需要轉換
assign sign = rs1_value[31] ^ rs2_value[31];

always_comb begin
    unique case (funct3)
        `F_DIV: begin
            div_out = sign ? quotient_twos_comp : quotient;
        end
        `F_DIVU: begin
            div_out = quotient;
        end
        `F_REM: begin
            div_out = sign ? remainder_twos_comp : remainder;
        end
        `F_REMU: begin
            div_out = remainder;
        end
    endcase
end
```

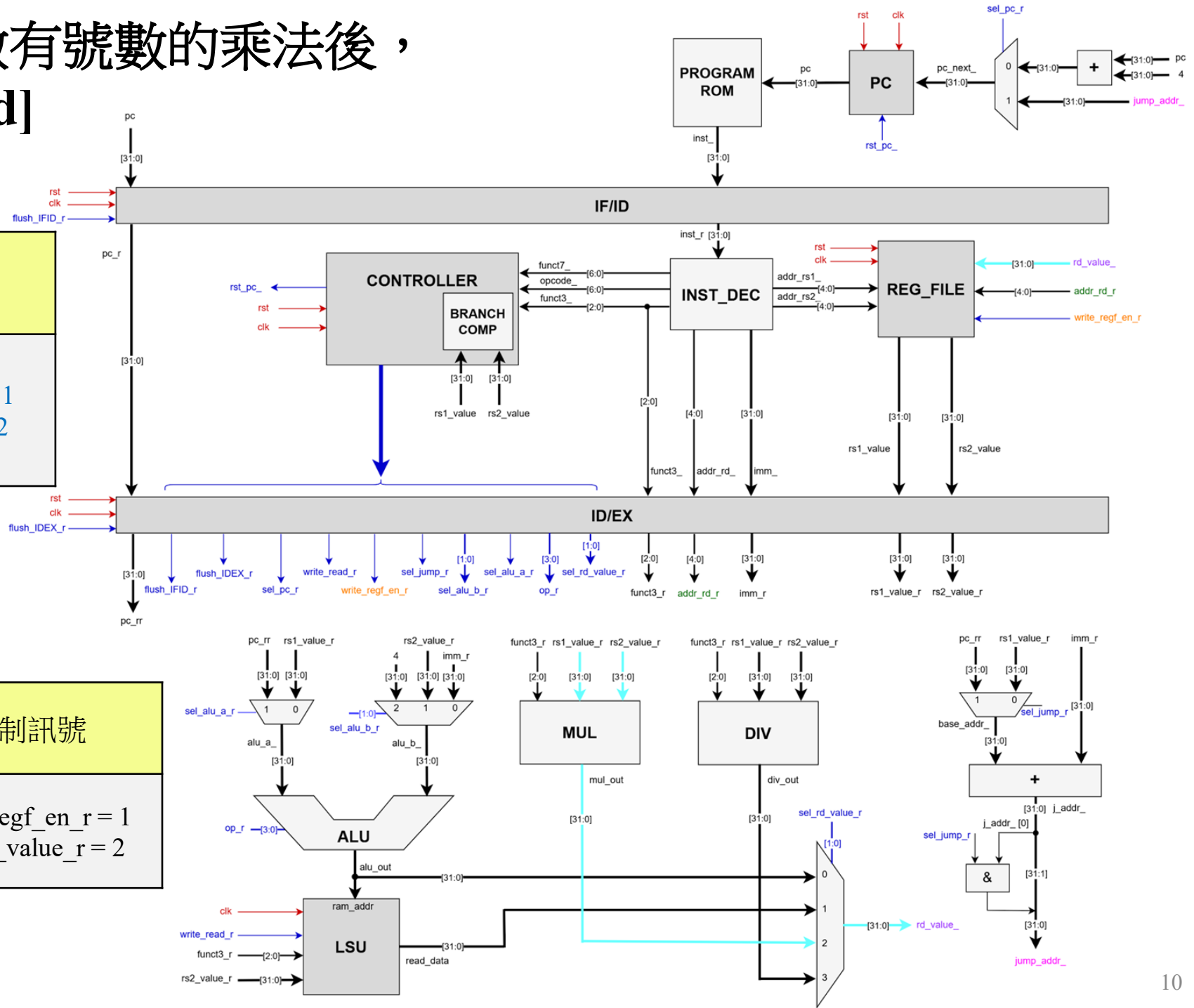
MUL：x[rs1] 和 x[rs2] 做有號數的乘法後，將較低的32位元存入 x[rd]

```
`define F7_R_M          7'b0000001
`define Opcode_R_M      7'b0110011
```

動作	控制訊號
if ((opcode_ == `Opcode_R_M) && (funct3_ == `F_MUL) && (funct7_ == `F7_R_M)) 發出 MUL 的控制訊號	write_regf_en_ = 1 sel_rd_value_ = 2

```
`define F_MUL           3'b000
`define F_MULH          3'b001
`define F_MULHSU        3'b010
`define F_MULHU         3'b011
`define F_DIV           3'b100
`define F_DIVU          3'b101
`define F_REM           3'b110
`define F_REMU          3'b111
```

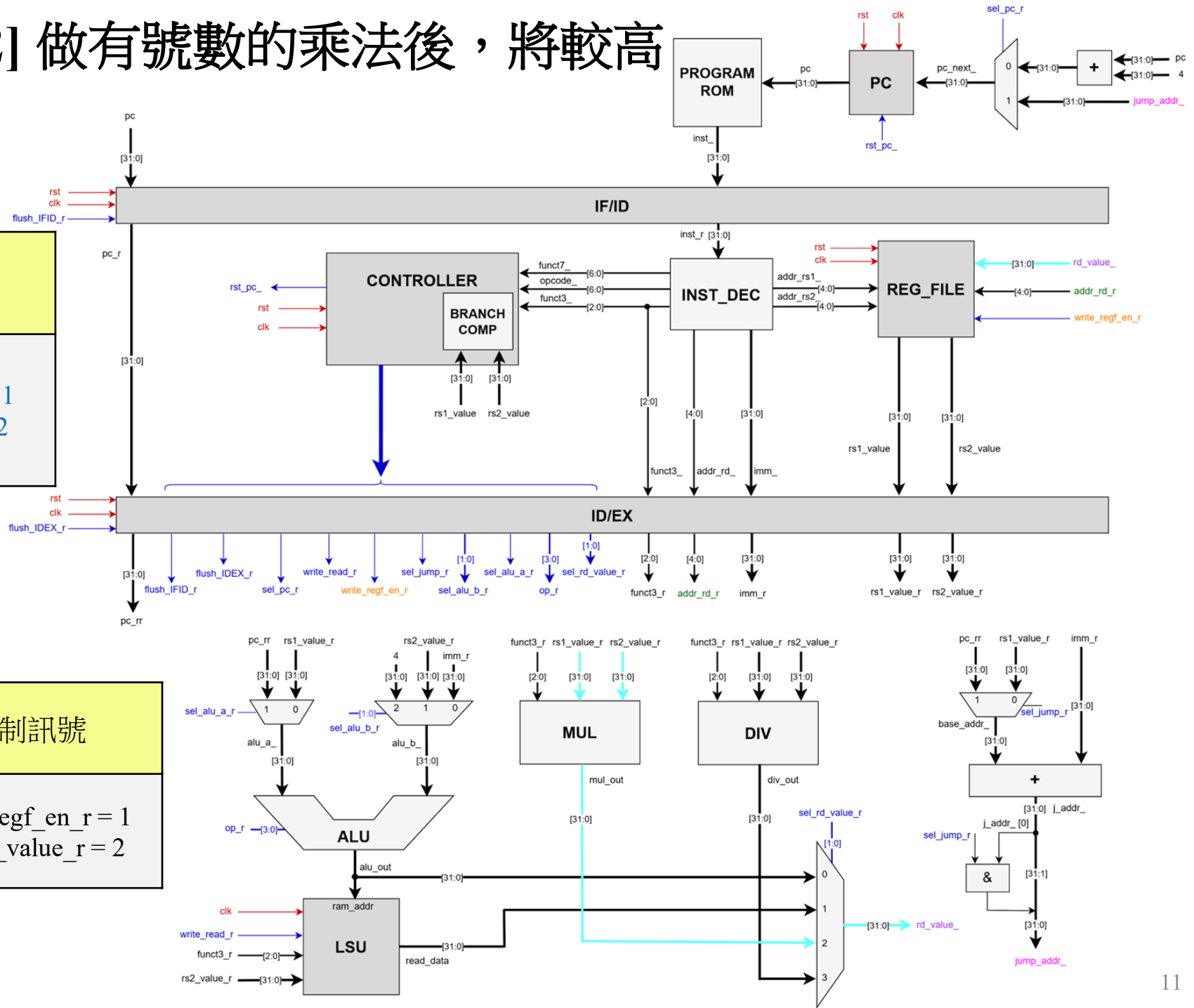
動作	控制訊號
rd_value_ ← (rs1_value_r $_s\times_s$ rs2_value_r) [31:0]	write_regf_en_r = 1 sel_rd_value_r = 2



MULHU：x[rs1] 和 x[rs2] 做有號數的乘法後，將較高的32位元存入 x[rd]

動作	控制訊號
if ((opcode_ == `Opcode_R_M) && (funct3_ == `F_MULHU) && (funct7_ == `F7_R_M)) 發出 MULHU 的控制訊號	write_regf_en_ = 1 sel_rd_value_ = 2

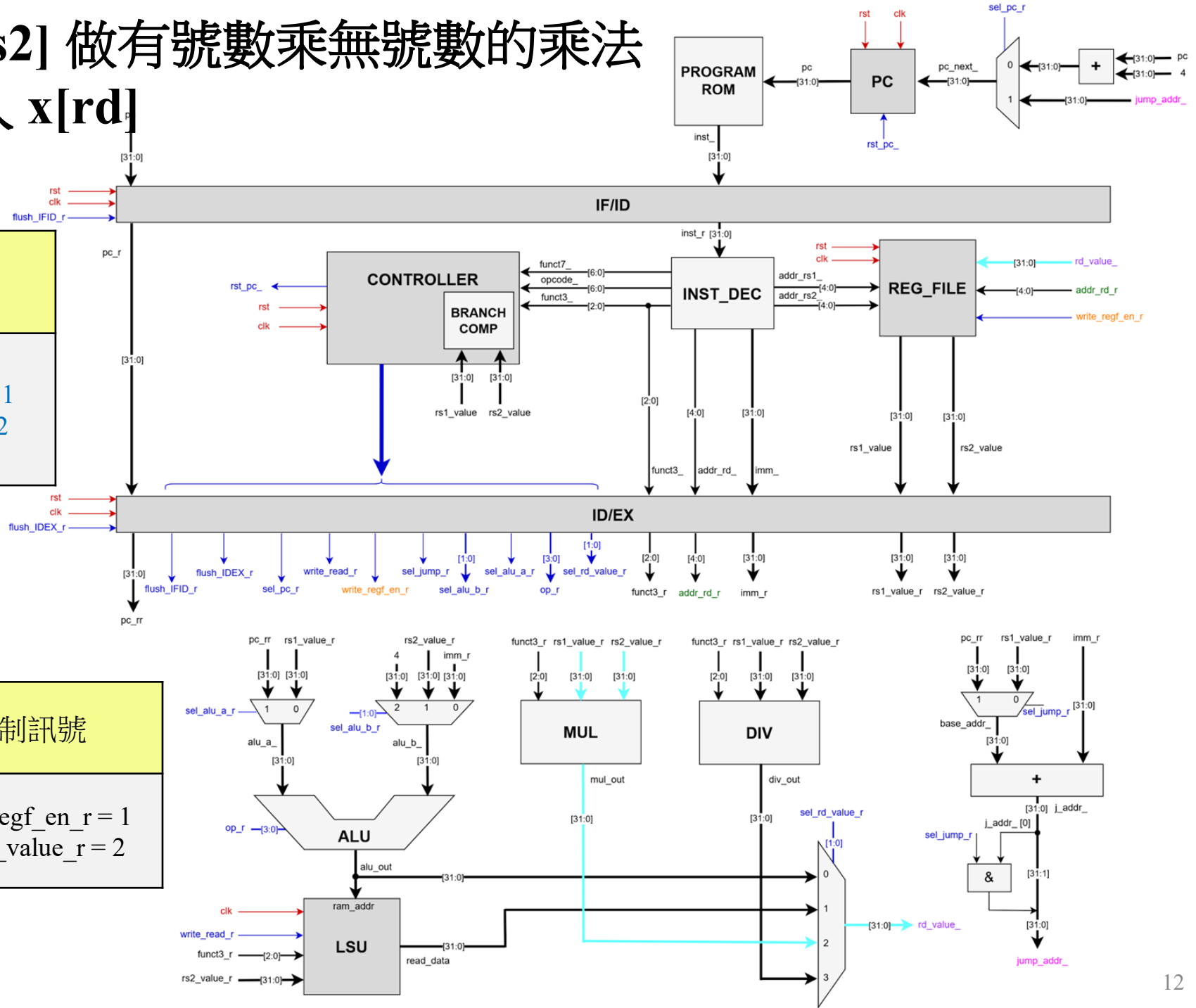
動作	控制訊號
rd_value_ ← (rs1_value_r $_s\times_s$ rs2_value_r) [63:32]	write_regf_en_r = 1 sel_rd_value_r = 2



MULHSU：x[rs1] 和 x[rs2] 做有號數乘無號數的乘法後，將較高的32位元存入 x[rd]

動作	控制訊號
if ((opcode_ == `Opcode_R_M) && (funct3_ == `F_MULHSU) && (funct7_ == `F7_R_M)) 發出 MULHSU 的控制訊號	write_regf_en_ = 1 sel_rd_value_ = 2

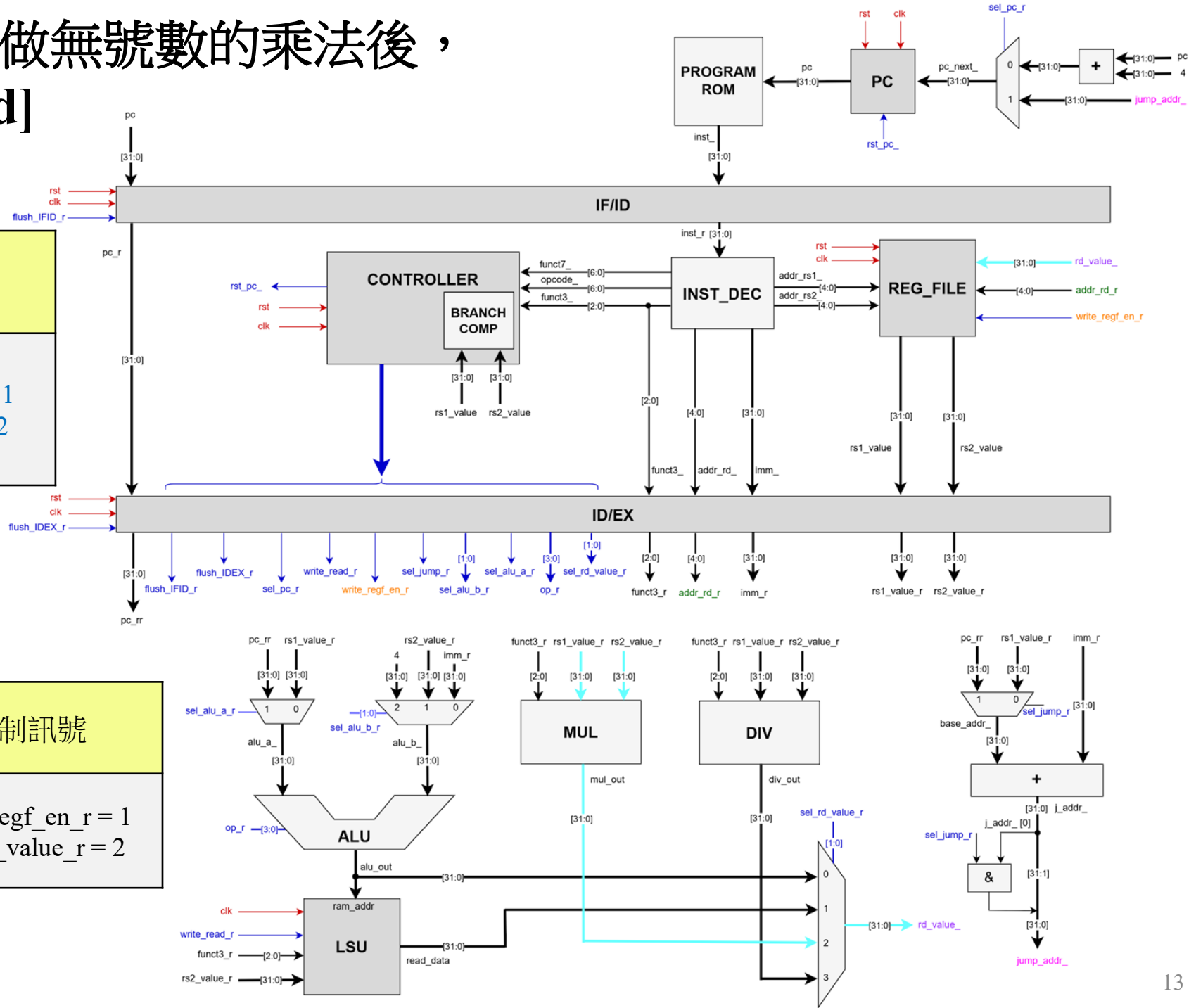
動作	控制訊號
rd_value_ ← (rs1_value_r $_s\times_u$ rs2_value_r) [63:32]	write_regf_en_r = 1 sel_rd_value_r = 2



MULH : x[rs1] 和 x[rs2] 做無號數的乘法後，將較高的32位元存入 x[rd]

動作	控制訊號
if ((opcode_ == `Opcode_R_M) && (funct3_ == `F_MULH) && (funct7_ == `F7_R_M)) 發出 MULH 的控制訊號	write_regf_en_ = 1 sel_rd_value_ = 2

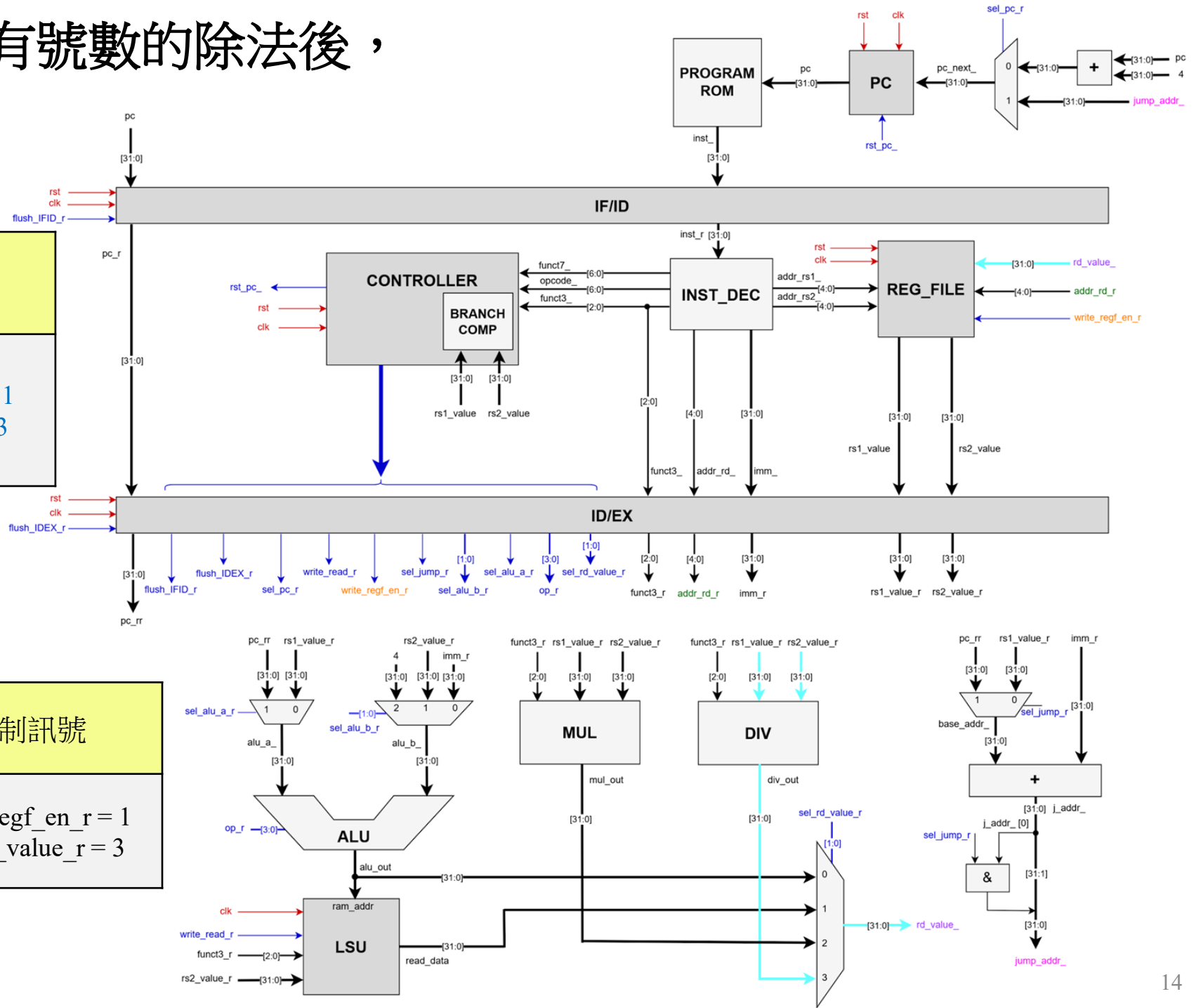
動作	控制訊號
rd_value_ ← (rs1_value_r $_u\times_u$ rs2_value_r) [63:32]	write_regf_en_r = 1 sel_rd_value_r = 2



DIV：x[rs1] 和 x[rs2] 做有號數的除法後，將商傳回 x[rd]

動作	控制訊號
if ((opcode_ == `Opcode_R_M) && (funct3_ == `F_DIV) && (funct7_ == `F7_R_M)) 發出 DIV 的控制訊號	write_regf_en_ = 1 sel_rd_value_ = 3

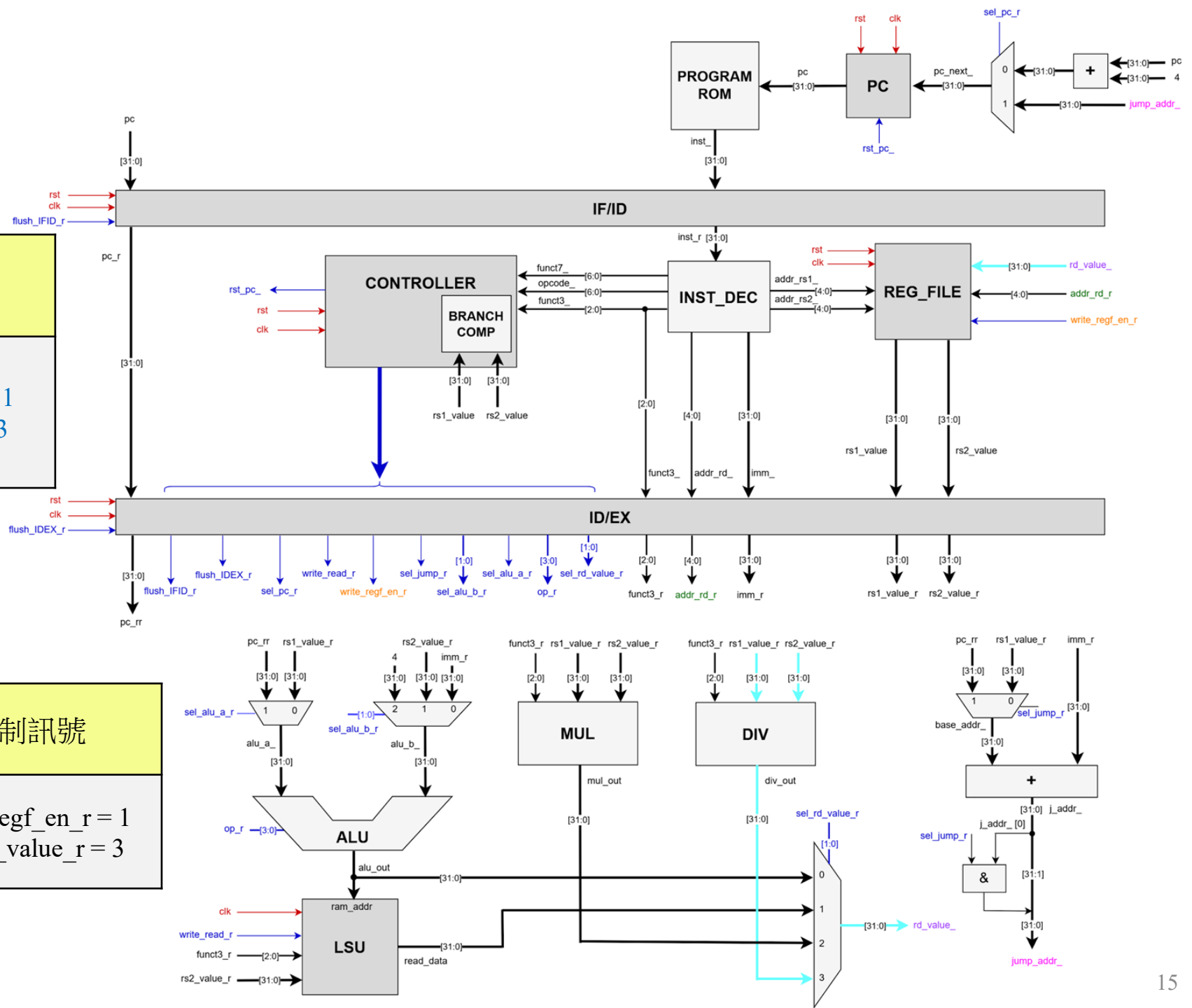
動作	控制訊號
rd_value_ ← rs1_value_r /$_s$ rs2_value_r	write_regf_en_r = 1 sel_rd_value_r = 3



DIVU：x[rs1] 和 x[rs2] 做，將商傳回 x[rd]

動作	控制訊號
if ((opcode_ == `Opcode_R_M) && (funct3_ == `F_DIVU) && (funct7_ == `F7_R_M)) 發出 DIVU 的控制訊號	write_regf_en_ = 1 sel_rd_value_ = 3

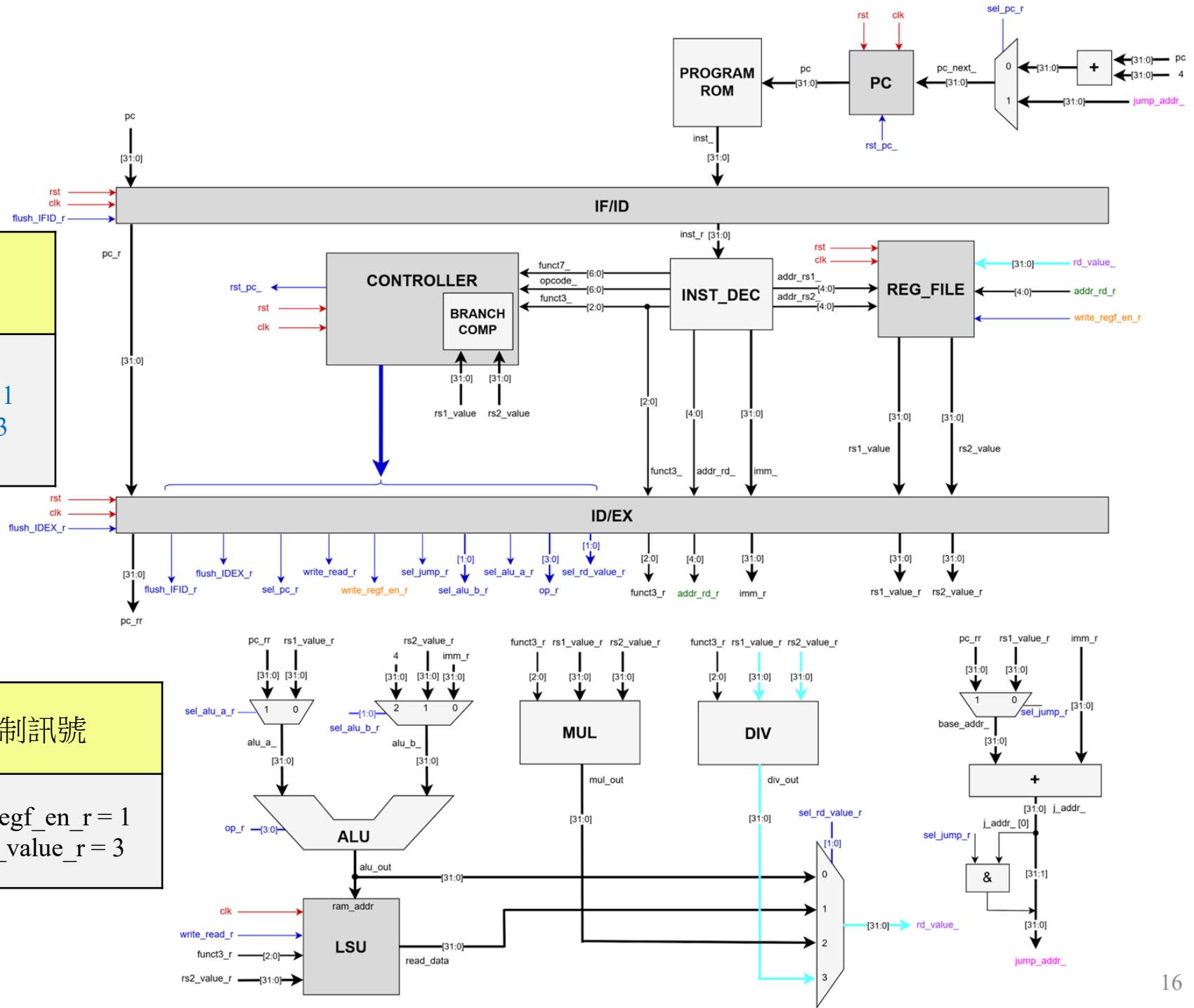
動作	控制訊號
rd_value_ ← rs1_value_r /$_u$ rs2_value_r	write_regf_en_r = 1 sel_rd_value_r = 3



REM : x[rs1] 和 x[rs2] 做，將餘數傳回 x[rd]

動作	控制訊號
if ((opcode_ == `Opcode_R_M) && (funct3_ == `F_REM) && (funct7_ == `F7_R_M)) 發出 REM 的控制訊號	write_regf_en_ = 1 sel_rd_value_ = 3

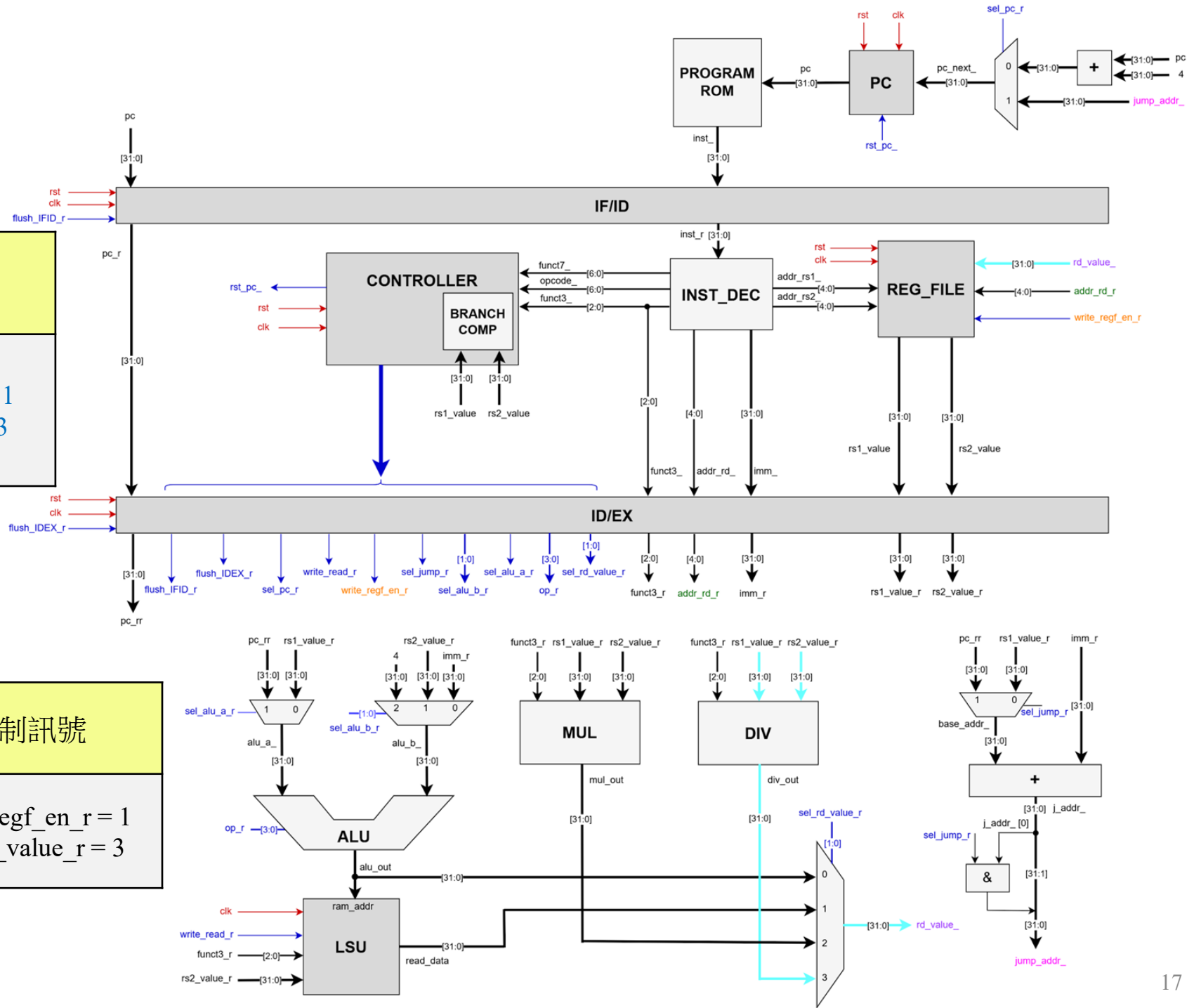
動作	控制訊號
rd_value_ ← rs1_value_r %$_s$ rs2_value_r	write_regf_en_r = 1 sel_rd_value_r = 3



REMU：x[rs1] 和 x[rs2]，將餘數傳回 x[rd]

動作	控制訊號
if ((opcode_ == `Opcode_R_M) && (funct3_ == `F_REMU) && (funct7_ == `F7_R_M)) 發出 REMU 的控制訊號	write_regf_en_ = 1 sel_rd_value_ = 3

動作	控制訊號
rd_value_ ← rs1_value_r $\%_u$ rs2_value_r	write_regf_en_r = 1 sel_rd_value_r = 3

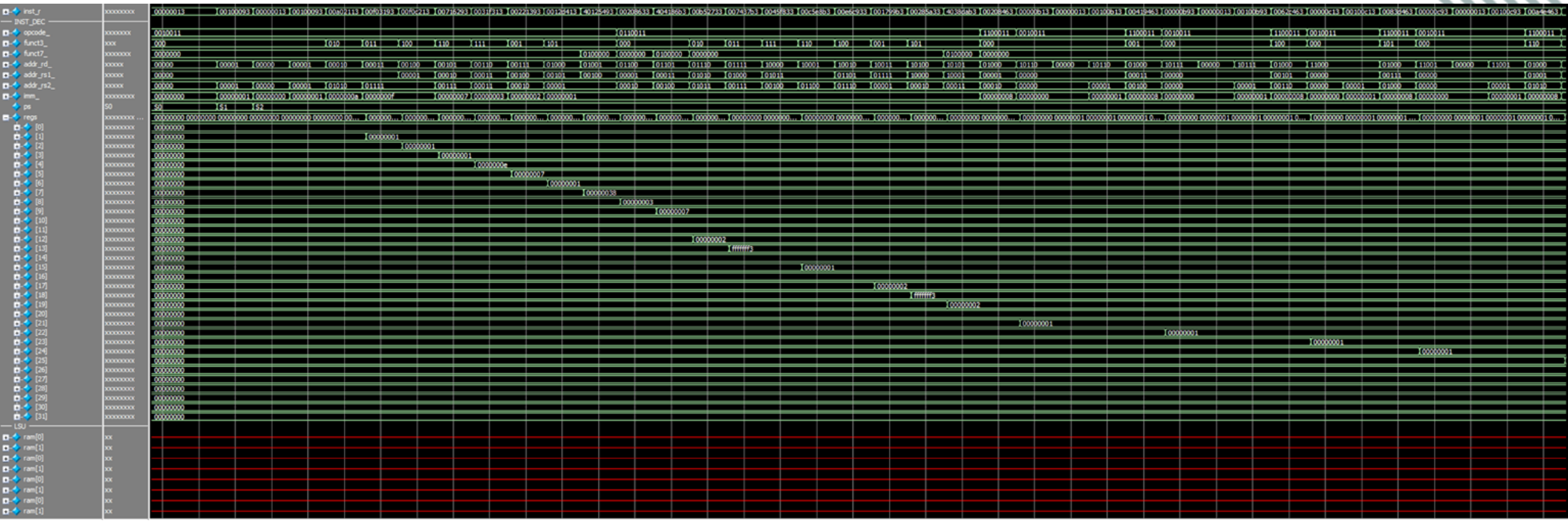


上課實作：ModelSim 模擬

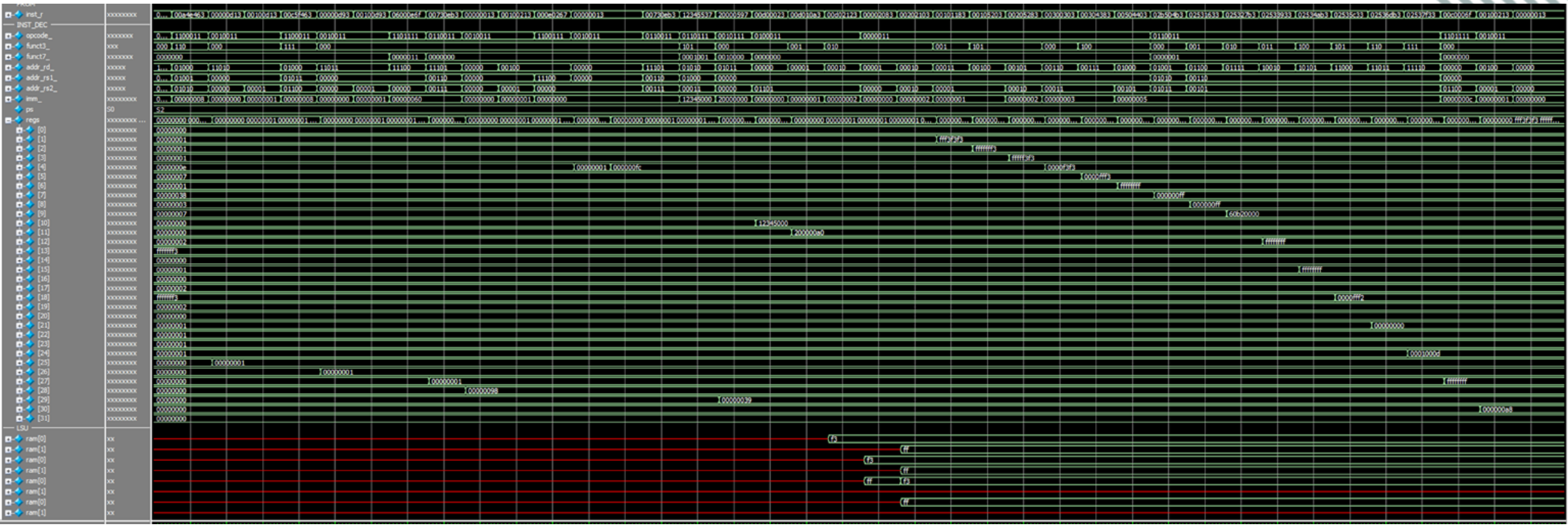
```
#立即值定址
    addi x1, x0, 1
    slti x2, x0, 10
    sltiu x3, x0, 15
    xori x4, x1, 0xF
    ori x5, x2, 0x7
    andi x6, x3, 0x3
    slli x7, x4, 2
    srli x8, x5, 1
    srai x9, x4, 1

#暫存器定址
    add x12, x1, x2
    sub x13, x3, x4
    slt x14, x10, x11
    sltu x15, x8, x7
    and x16, x11, x4
    or  x17, x11, x12
    xor x18, x13, x14
    sll x19, x15, x1
    srl x20, x16, x2
    sra x21, x17, x3

#分支預測
        beq x1, x2, test_beq
        addi x22, x0, 0
    test_beq:
        addi x22, x0, 1
        bne x3, x4, test_bne
        addi x23, x0, 0
    test_bne:
        addi x23, x0, 1
        blt x5, x6, test_blt
        addi x24, x0, 0
    test_blt:
        addi x24, x0, 1
        bge x7, x8, test_bge
        addi x25, x0, 0
    test_bge:
        addi x25, x0, 1
        bltu x9, x10, test_bltu
        addi x26, x0, 0
    test_bltu:
        addi x26, x0, 1
        bgeu x11, x12, test_bgeu
        addi x27, x0, 0
    test_bgeu:
        addi x27, x0, 1

#無條件跳躍
    jal x28, jump
    add x29, x6, x7
    lui x10, 0x12345
    auipc x11, 0x20000

#記憶體存取
    sb x13, 0(x0)
    sh x13, 1(x0)
    sw x13, 2(x0)
    lw x1, 0(x0)
    lw x2, 2(x0)
    lh x3, 1(x0)
    lhu x4, 1(x0)
    lhu x5, 2(x0)
    lb x6, 3(x0)
    lbu x7, 3(x0)
    lbu x8, 5(x0)

#乘除法
    mul x9, x10, x11
    mulh x12, x6, x5
    mulhsu x15, x6, x5
    mulhu x18, x6, x5
    div x21, x6, x5
    divu x24, x6, x5
    rem x27, x6, x5
    remu x30, x6, x5

    j end
jump:
    addi x4, x0, 1
    jalr x4, x28, 0
end:
    nop
```

```
0x00000000  0x00100093  addi x1 x0 1
0x00000004  0x00A02113  slti x2 x0 10
0x00000008  0x00F03193  sltiu x3 x0 15
0x0000000C  0x00F0C213  xori x4 x1 15
0x00000010  0x00716293  ori x5 x2 7
0x00000014  0x0031F313  andi x6 x3 3
0x00000018  0x00221393  slli x7 x4 2
0x0000001C  0x0012D413  srli x8 x5 1
0x00000020  0x40125493  srai x9 x4 1
0x00000024  0x00208633  add x12 x1 x2
0x00000028  0x404186B3  sub x13 x3 x4
0x0000002C  0x00B52733  slt x14 x10 x11
0x00000030  0x007437B3  sltu x15 x8 x7
0x00000034  0x0045F833  and x16 x11 x4
0x00000038  0x00C5E8B3  or x17 x11 x12
0x0000003C  0x00E6C933  xor x18 x13 x14
0x00000040  0x001799B3  sll x19 x15 x1
0x00000044  0x00285A33  srl x20 x16 x2
0x00000048  0x4038DAB3  sra x21 x17 x3
0x0000004C  0x00208463  beq x1 x2 8
0x00000050  0x00000B13  addi x22 x0 0
0x00000054  0x00100B13  addi x22 x0 1
0x00000058  0x00419463  bne x3 x4 8
0x0000005C  0x00000B93  addi x23 x0 0
0x00000060  0x00100B93  addi x23 x0 1
0x00000064  0x0062C463  blt x5 x6 8
0x00000068  0x00000C13  addi x24 x0 0
0x0000006C  0x00100C13  addi x24 x0 1
0x00000070  0x0083D463  bge x7 x8 8
0x00000074  0x00000C93  addi x25 x0 0
0x00000078  0x00100C93  addi x25 x0 1
0x0000007C  0x00A4E463  bltu x9 x10 8
0x00000080  0x00000D13  addi x26 x0 0
0x00000084  0x00100D13  addi x26 x0 1
0x00000088  0x00C5F463  bgeu x11 x12 8
0x0000008C  0x00000D93  addi x27 x0 0
0x00000090  0x00100D93  addi x27 x0 1
0x00000094  0x06000E6F  jal x28 96
0x00000098  0x00730EB3  add x29 x6 x7
0x0000009C  0x12345537  lui x10 74565
0x000000A0  0x20000597  auipc x11 131072
0x000000A4  0x00D00023  sb x13 0(x0)
0x000000A8  0x00D010A3  sh x13 1(x0)
0x000000AC  0x00D02123  sw x13 2(x0)
0x000000B0  0x00002083  lw x1 0(x0)
0x000000B4  0x00202103  lw x2 2(x0)
0x000000B8  0x00101183  lh x3 1(x0)
0x000000BC  0x00105203  lhu x4 1(x0)
0x000000C0  0x00205283  lhu x5 2(x0)
0x000000C4  0x00300303  lb x6 3(x0)
0x000000C8  0x00304383  lbu x7 3(x0)
0x000000CC  0x00504403  lbu x8 5(x0)
0x000000D0  0x02B504B3  mul x9 x10 x11
0x000000D4  0x02531633  mulh x12 x6 x5
0x000000D8  0x025327B3  mulhsu x15 x6 x5
0x000000DC  0x02533933  mulhu x18 x6 x5
0x000000E0  0x02534AB3  div x21 x6 x5
0x000000E4  0x02535C33  divu x24 x6 x5
0x000000E8  0x02536DB3  rem x27 x6 x5
0x000000EC  0x02537F33  remu x30 x6 x5
0x000000F0  0x00C0006F  jal x0 12
0x000000F4  0x00100213  addi x4 x0 1
0x000000F8  0x000E0267  jalr x4 x28 0
0x000000FC  0x00000013  addi x0 x0 0
```

上課實作：ModelSim 模擬



上課實作：ModelSim 模擬



上課實作：FPGA 燒錄

- 本次實作一個計數器從 3 開始計數，每次將當前的數值乘以 7 直到超過 0xffffff 才返回 3 重新計數。
- 本次使用記憶體位址 0x0 來儲存上數計數器中計數的數值，每次迴圈使用 LW 載入數值並在累加完後使用 SW 存回記憶體。
- 本次燒錄無須再使用除頻器，而是透過 Delay 副程式來達成除頻的效果。

```
init:
    li x2, 3
    li x3, 7
    lui x4 4096
    nop
    addi x4 x4 -1

start:
    # 初始化計數器
    sw x2, 0(x0)

loop:
    lw x31, 0(x0)
    call delay

    mul x31, x31, x3
    nop
    bge x31, x4, start
    sw x31, 0(x0)

    # 無窮迴圈
    j loop

# 延遲副程式
delay:
    # 外層迴圈次數
    li x10, 5000      # 外層迴圈次數 (5000)
outer_loop:
    # 內層迴圈次數
    li x11, 1000      # 內層迴圈次數 (1000)
    nop
    nop
inner_loop:
    addi x11, x11, -1 # 內層迴圈遞減
    nop
    nop
    bnez x11, inner_loop # 如果 x11 不為零，繼續內層迴圈
    addi x10, x10, -1 # 外層迴圈遞減
    nop
    nop
    bnez x10, outer_loop # 如果 x10 不為零，繼續外層迴圈

    ret               # 返回主程式
```

```
0x00000000  0x00300113  addi x2 x0 3
0x00000004  0x00700193  addi x3 x0 7
0x00000008  0x01000237  lui x4 4096
0x0000000C  0x00000013  addi x0 x0 0
0x00000010  0xFFF20213  addi x4 x4 -1
0x00000014  0x00202023  sw x2 0(x0)
0x00000018  0x00002F83  lw x31 0(x0)
0x0000001C  0x00000317  auipc x6 0
0x00000020  0x01C300E7  jalr x1 x6 28
0x00000024  0x023F8FB3  mul x31 x31 x3
0x00000028  0x00000013  addi x0 x0 0
0x0000002C  0xFE4FD4E3  bge x31 x4 -24
0x00000030  0x01F02023  sw x31 0(x0)
0x00000034  0xFE5FF06F  jal x0 -28
0x00000038  0x00001537  lui x10 1
0x0000003C  0x38850513  addi x10 x10 904
0x00000040  0x3E800593  addi x11 x0 1000
0x00000044  0x00000013  addi x0 x0 0
0x00000048  0x00000013  addi x0 x0 0
0x0000004C  0xFFF58593  addi x11 x11 -1
0x00000050  0x00000013  addi x0 x0 0
0x00000054  0x00000013  addi x0 x0 0
0x00000058  0xFE059AE3  bne x11 x0 -12
0x0000005C  0xFFF50513  addi x10 x10 -1
0x00000060  0x00000013  addi x0 x0 0
0x00000064  0x00000013  addi x0 x0 0
0x00000068  0xFC051CE3  bne x10 x0 -40
0x0000006C  0x00008067  jalr x0 x1 0
```

上課實作 ： 語撰寫練習—多項式計算

- **實驗說明：**
  - 請透過當前課堂所學的 RISC-V 組合語言，撰寫一個可計算下列多項式的程式：

$$\left(\frac{f(a)}{b}\right) \bmod c, \qquad f(a) = 3a^3 - 5a^2 + 12a - 7, \qquad a = 4, b = 3, c = 5$$

  - 請依照下列流程依序計算三項結果，並**依序存入 x31**：

    計算 $f(a)$（使用 *mul* ）→ **計算 $f(a) / b$**（使用 *div* ）→ **計算 $(f(a) / b) \bmod c$**（使用 *rem* ）

  - 編寫完成後，請使用 Tool Chain 轉換成 Program_ROM.sv，並於 ModelSim 中載入觀察 x1 ～ x31 的變化。請將所有用到的暫存器加入波形以便檢查資料 hazard 及結果正確性。

```
1   # 常數
2   addi    x1, x0, 4       # x1 = a = 4
3   addi    x2, x0, 3       # x2 = b = 3
4   addi    x3, x0, 5       # x3 = c = 5
```

THANK YOU