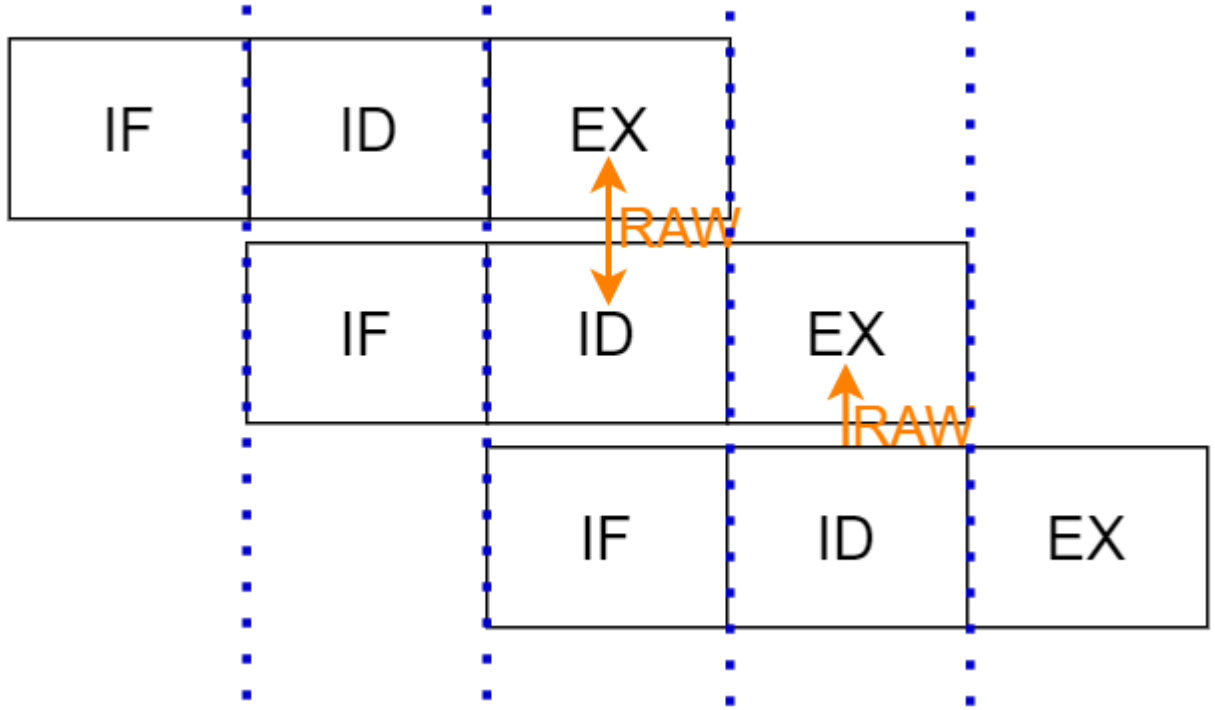
計算機系統設計

危障處理-前饋單元

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

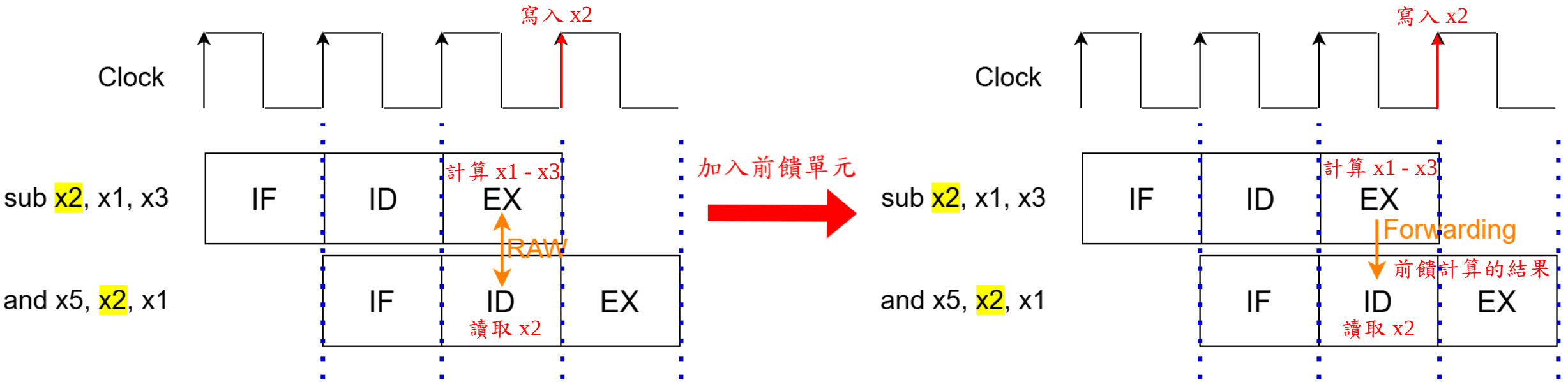
危障 (Hazard)

- **結構危障 (Structural Hazards)**
  - Hardware resource conflict. For eg: two instructions require access to memory in the same clock cycle.
    - 解法：採用哈佛架構 (Harvard architecture)
- **資料危障 (Data Hazards)**
  - ~~WAW (Write-After-Write)~~
  - RAW (Read-After-Write)
    - 解法：加入前饋單元 (Forwarding Unit)
  - ~~WAR (Write-After-Read)~~
- **控制危障 (Control Hazards)**
  - Caused by Jump/Branch instructions
    - 解法：清洗 (Flush)



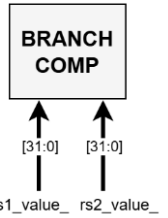
資料危障 (Data Hazards)

- **RAW (Read-After-Write)**
  - 在管線化的情況下，當第一個指令 (sub x2, x1, x3) 執行到 EX 階段時，第二個指令 (and x5, x2, x1) 正在讀取暫存器 x2 的值，因此第二個指令所讀到的暫存器 x2 的值並不是第一個指令的結果。



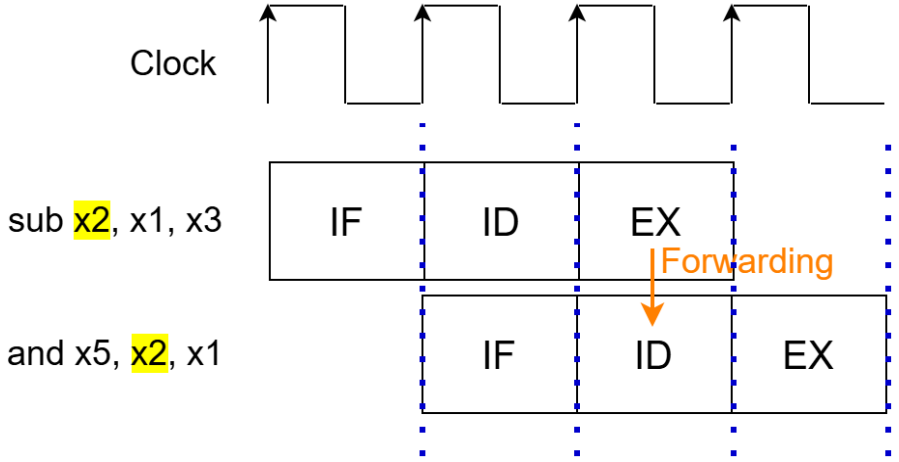
前饋單元

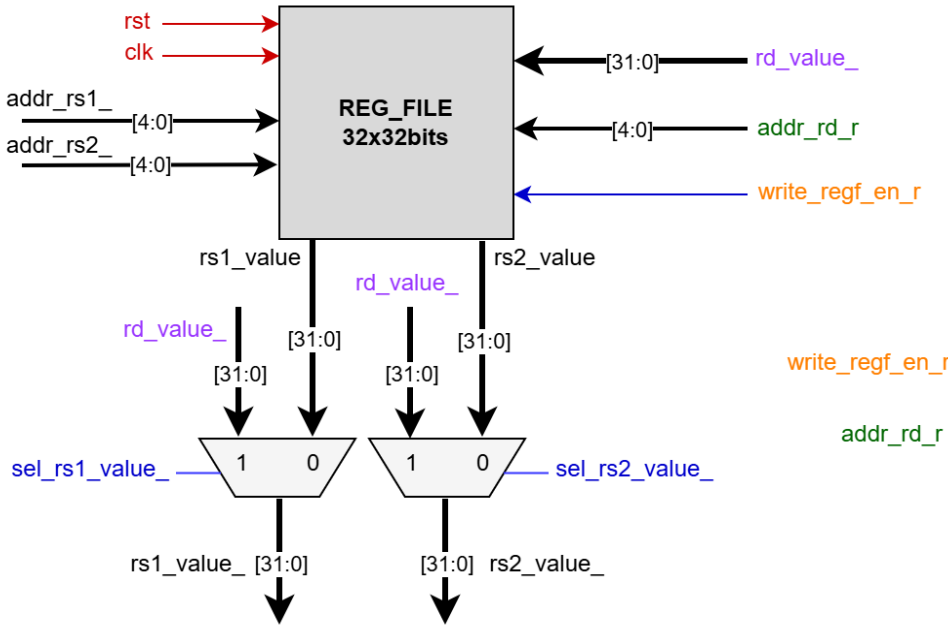
※注意 Branch Comp 的 rs1_value 和 rs2_value 皆須改成 rs1_value_ 和 rs2_value_

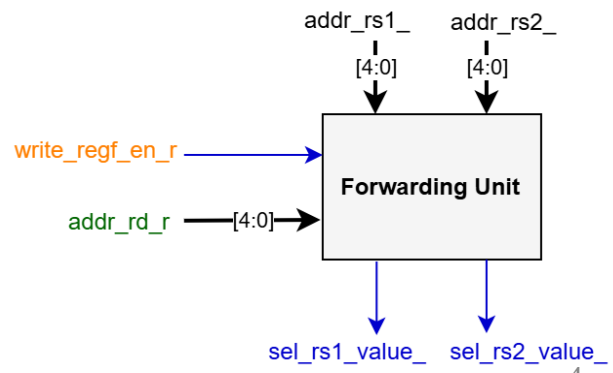


- **前饋單元 (Forwarding Unit)**
  - 前饋單元在兩指令發生 RAW 危障時，會將第一筆指令運算之結果提早回傳至指令解碼階段。
  - 前饋條件有以下兩點：
    1. 第一筆指令有寫入暫存器的動作 (write_regf_en_r == 1)。
    2. 第一筆指令的目標暫存器 (rd) 和第二個指令的來源暫存器 (rs1 或 rs2) 相同。

rs1_value 是否前饋 = (write_regf_en_r == 1) & (addr_rd_r == addr_rs1_)



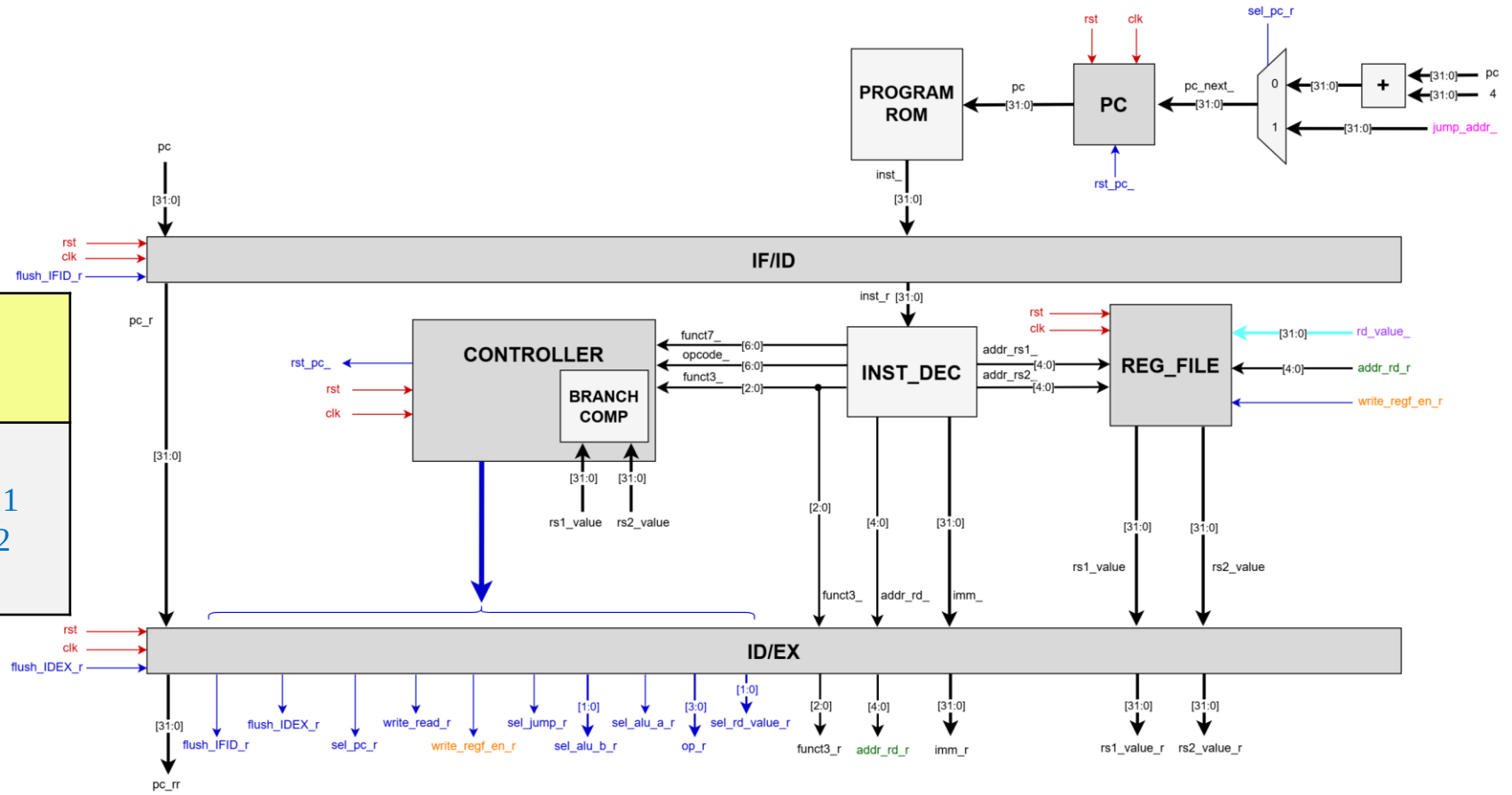




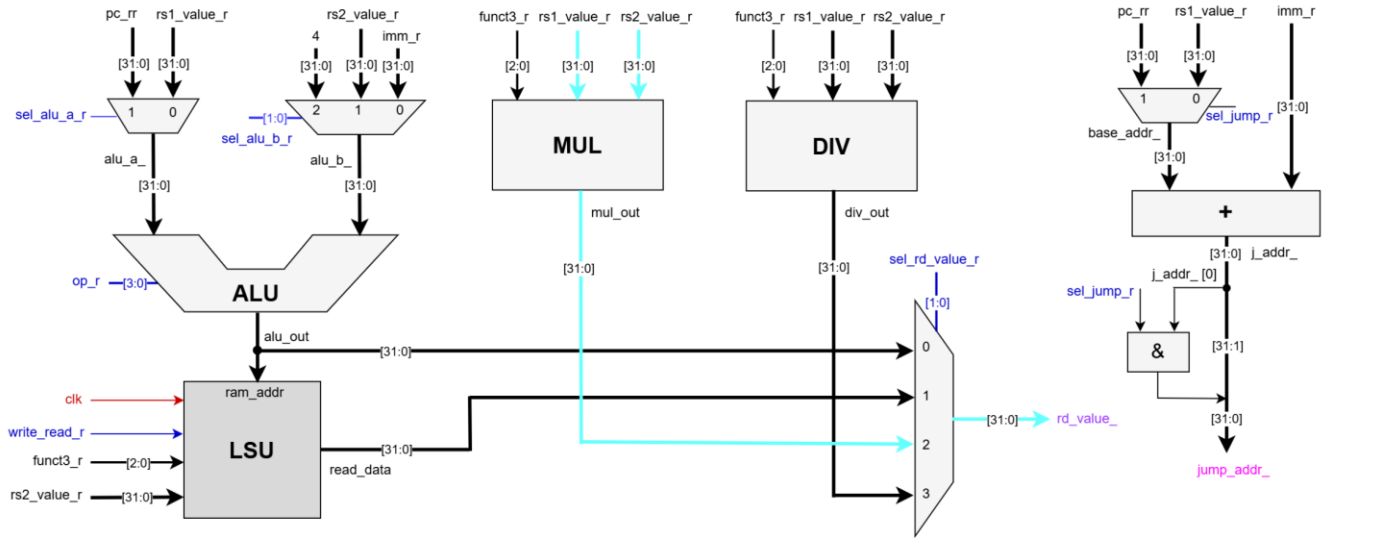

```
//兩個條件來判斷是否前饋
assign sel_rs1_value_ = write_regf_en_r & (addr_rd_r == addr_rs1_);
assign sel_rs2_value_ = write_regf_en_r & (addr_rd_r == addr_rs2_);
assign rs1_value_ = sel_rs1_value_ ? rd_value_ : rs1_value;
assign rs2_value_ = sel_rs2_value_ ? rd_value_ : rs2_value;
```

上周完成之架構

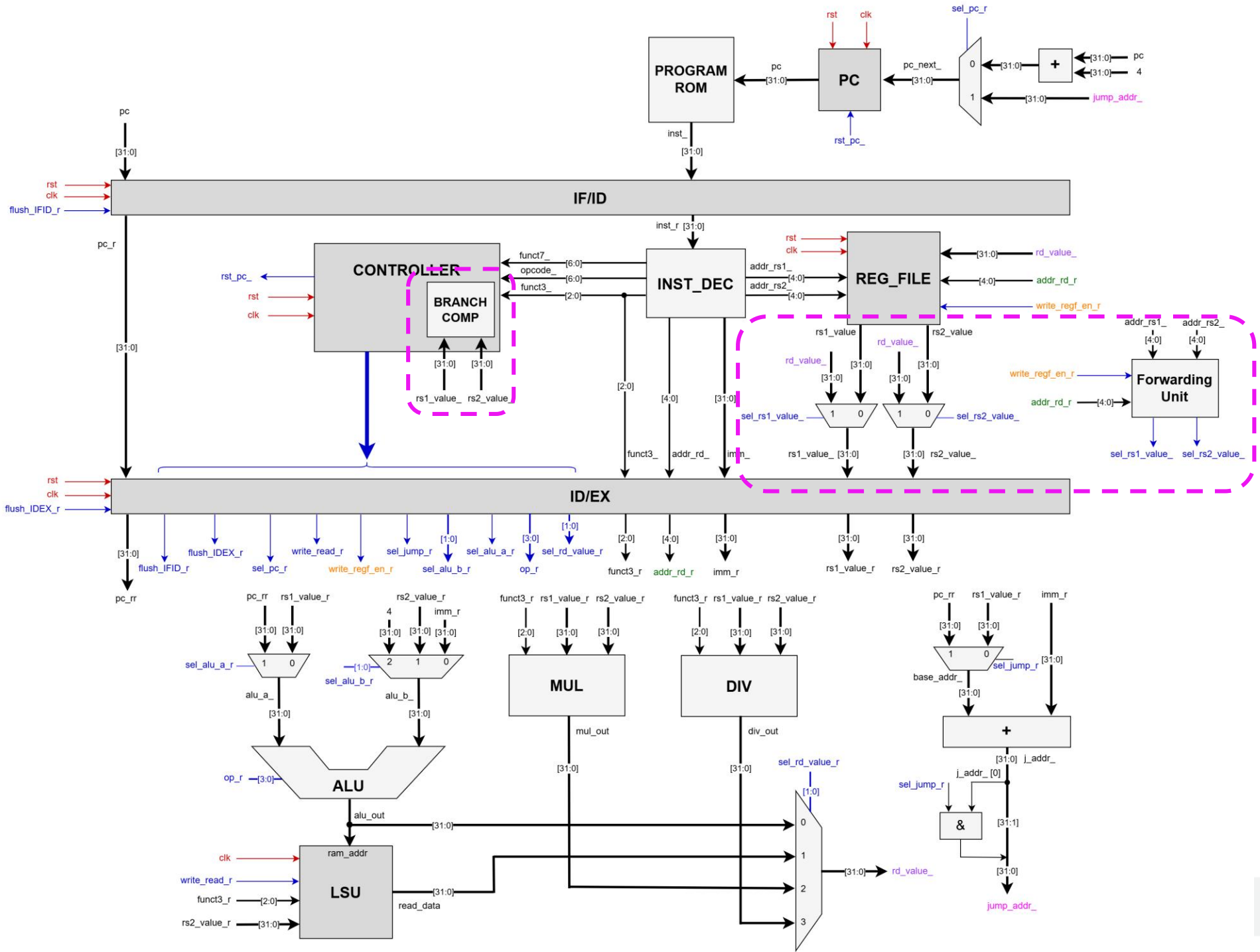
動作	控制訊號
if ((opcode_ == `Opcode_R_M) && (funct3_ == `F_MUL) && (funct7_ == 7'b000_0001)) 發出 LB 的控制訊號	write_regf_en_ = 1 sel_rd_value_ = 2



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



新架構

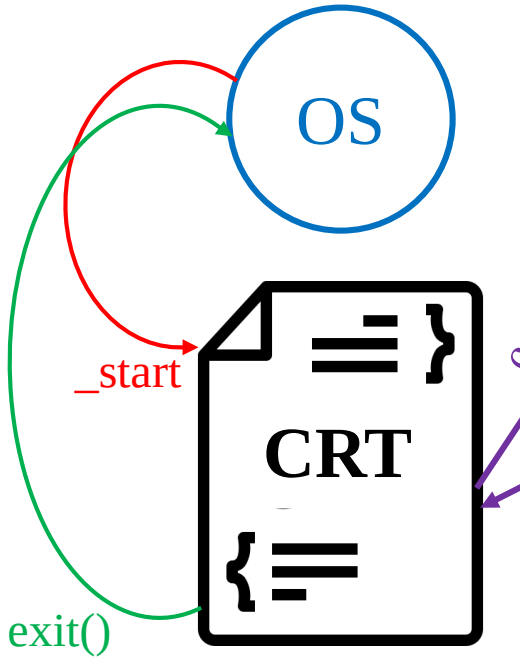

PROGRAM ROM
PC
pc_next_
sel_pc_r
jump_addr_
rst_pc_
inst_
IF/ID
flush_IFID_r
pc_r
CONTROLLER
BRANCH COMP
rst_pc_
funct7_
opcode_
funct3_
INST_DEC
addr_rs1_
addr_rs2_
REG_FILE
rd_value_
addr_rd_r
write_regf_en_r
rs1_value
rs2_value
rs1_value_
rs2_value_
Forwarding Unit
sel_rs1_value_
sel_rs2_value_
ID/EX
flush_IDEX_r
pc_rr
write_read_r
sel_jump_r
sel_alu_a_r
sel_alu_b_r
op_r
sel_rd_value_r
funct3_r
addr_rd_r
imm_r
rs1_value_r
rs2_value_r
alu_a_
alu_b_
ALU
alu_out
MUL
mul_out
DIV
div_out
base_addr_
j_addr_
LSU
ram_addr
read_data

main() 函數的開始與結束 (1/2)

- 執行 C 或 C++ 程式時，作業系統讀入可執行檔 (如 .elf) 並跳到 entry point (通常是 C runtime 的 _start，也就是說 Program Counter 指到 _start)。 C runtime 負責初始化程式環境 (如 Stack、全域變數、命令列參數等)，然後呼叫使用者的 main()。
- main() 結束後， C runtime 透過 exit() 將控制權和結束狀態交回作業系統，由 OS 負責清理資源。

※ C runtime (CRT) 介紹

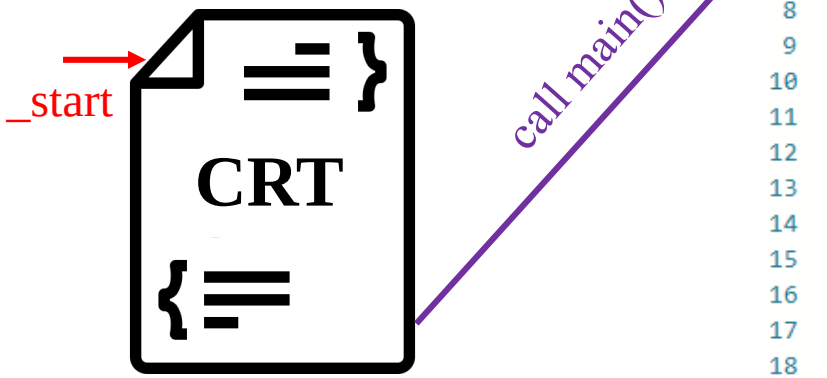
1. 由編譯器工具鏈提供的函式庫與啟動程式碼 (startup code) 集合。
2. 在 main() 前後負責初始化與收尾。
3. 不同平台提供不同 CRT，如 GCC 的 glibc/newlib、Microsoft Visual C++ (MSVC) 的 MSVCRT，嵌入式平台可能使用輕量級 CRT。




```c
// 費式數列函數 (遞迴方式)
int fibonacci_recursive(int); // 函式宣告

int main() {
    // 使用 register 關鍵字並指定 asm("x31")
    // 將變數強制綁定到 x31 暫存器
    register int ans asm("x31");

    ans = fibonacci_recursive(10);

    return 0;
}

int fibonacci_recursive(int n) { // 函式定義
    if (n <= 1) {
        return n;
    }
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2);
}
```

main() 函數的開始與結束 (2/2)

- 在沒有作業系統的 RISC-V 裸機環境中，當 Reset 發生後，Program Counter (PC) 會指向 C Runtime (CRT) 的 _start。由於系統中沒有作業系統介入，main() 的執行結束無法回傳到任何管理程式，因此通常不應依賴返回值，以避免 PC 跳往未定義的記憶體區域或執行錯誤指令。
- 在嵌入式產品中，廠商通常會提供對應的 CRT，開發者只需撰寫應用程式即可；而在自製裸機環境中，CRT 需由開發者自行實作，例如在後續範例中使用 li sp, 0x1fc 來設定 Stack Pointer 的初始位址。




```c
// 費式數列函數 (遞迴方式)
int fibonacci_recursive(int); // 函式宣告

int main() {
    // 使用 register 關鍵字並指定 asm("x31")
    // 強制將變數綁定到 RISC-V 的 x31 暫存器
    register int ans asm("x31");

    ans = fibonacci_recursive(10);

    while(1);
}

int fibonacci_recursive(int n) { // 函式定義
    if (n <= 1) {
        return n;
    }
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2);
}
```

C 語言範例 1：費式數列

```c
// 費式數列函數 (遞迴方式)
int fibonacci_recursive(int); // 函式宣告

int main() {
    // 使用 register 關鍵字並指定 asm("x31")
    // 強制將變數綁定到 RISC-V 的 x31 暫存器
    register int ans asm("x31");

    ans = fibonacci_recursive(10);

    while(1);
}

int fibonacci_recursive(int n) { // 函式定義
    if (n <= 1) {
        return n;
    }
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2);
}
```

※ 請將函式的宣告與定義分開撰寫

編譯

```
main:
        addi    sp,sp,-16
        sw      ra,12(sp)
        sw      s0,8(sp)
        addi    s0,sp,16
        li      a0,10
        call    fibonacci_recursive(int)
        mv      a5,a0
        mv      t6,a5
.L2:
        j       .L2
fibonacci_recursive(int):
        addi    sp,sp,-32
        sw      ra,28(sp)
        sw      s0,24(sp)
        sw      s1,20(sp)
        addi    s0,sp,32
        sw      a0,-20(s0)
        lw      a4,-20(s0)
        li      a5,1
        bgt     a4,a5,.L4
        lw      a5,-20(s0)
        j       .L5
.L4:
        lw      a5,-20(s0)
        addi    a5,a5,-1
        mv      a0,a5
        call    fibonacci_recursive(int)
        mv      s1,a0
        lw      a5,-20(s0)
        addi    a5,a5,-2
        mv      a0,a5
        call    fibonacci_recursive(int)
        mv      a5,a0
        add     a5,s1,a5
.L5:
        mv      a0,a5
        lw      ra,28(sp)
        lw      s0,24(sp)
        lw      s1,20(sp)
        addi    sp,sp,32
        jr      ra
```

暫存器	ABI	註解
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary/alternate link register
x6 ~ x7	t1 ~ t3	Temporaries
x8 ~ x9	s0 ~ s1	Saved registers
x10 ~ x11	a0 ~ a1	Function arguments/return values

暫存器	ABI	註解
x12 ~ x17	a2 ~ a7	Function arguments
x18 ~ x27	s2 ~ s11	Saved registers
x28 ~ x31	t3 ~ t6	Temporaries

C 語言範例 1：費式數列

- 由於本實作使用較小的記憶體 (2^9 Bytes)，因此需要重新指定 SP (Stack Pointer) 的位址，**li sp, 0x1fc** 。
- 組譯的過程不需要 Label 中的參數，fibonacci_recursive(int): 。

```
1       li      sp,0x1fc
2   main:
3           addi    sp,sp,-16
4           sw      ra,12(sp)
5           sw      s0,8(sp)
6           addi    s0,sp,16
7           li      a0,10
8           call    fibonacci_recursive
9           mv      a5,a0
10          mv      t6,a5
11  .L2:
12          j       .L2
13  fibonacci_recursive:
14          addi    sp,sp,-32
15          sw      ra,28(sp)
16          sw      s0,24(sp)
17          sw      s1,20(sp)
18          addi    s0,sp,32
19          sw      a0,-20(s0)
20          lw      a4,-20(s0)
21          li      a5,1
22          bgt     a4,a5,.L4
23          lw      a5,-20(s0)
24          j       .L5
```

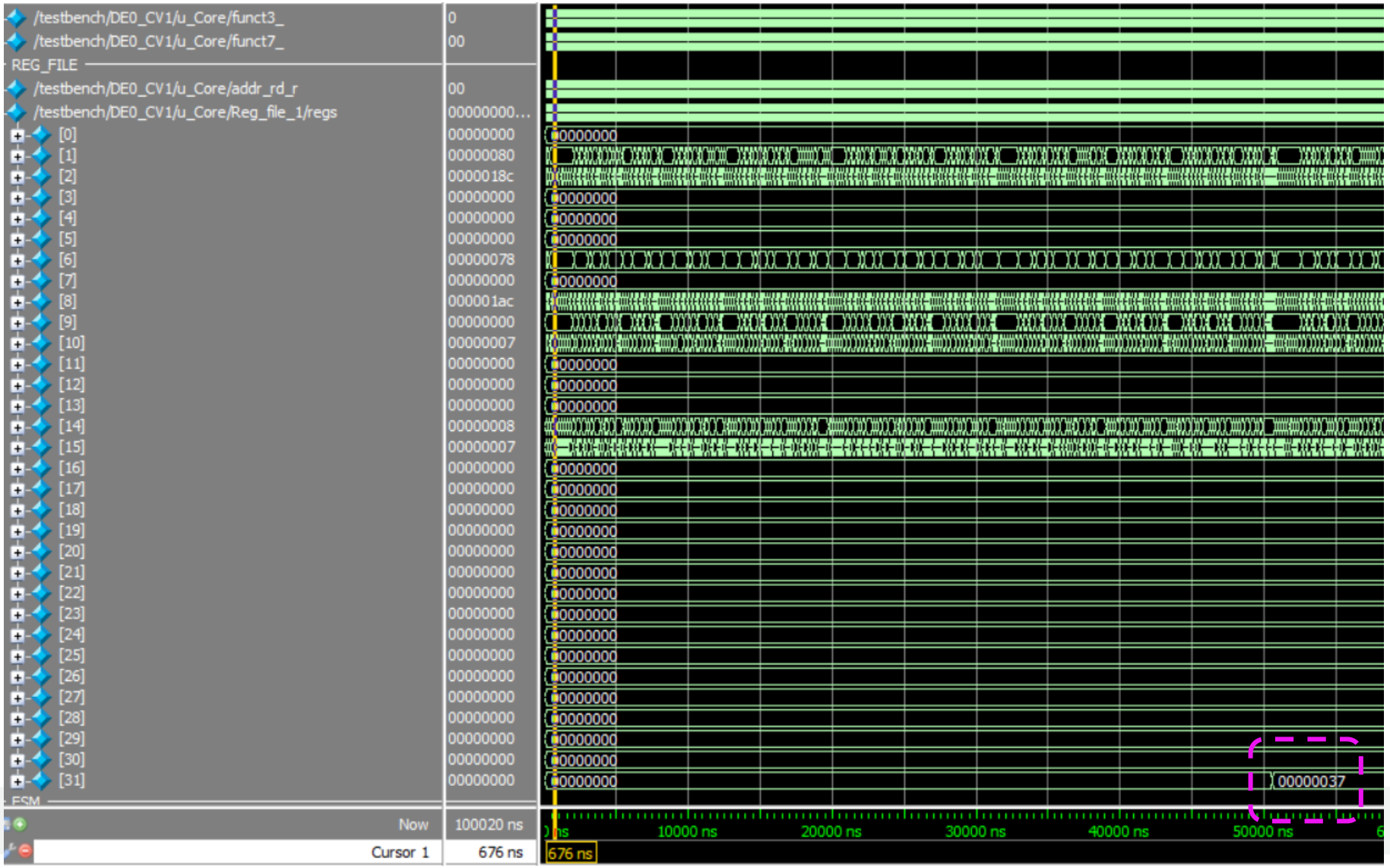
```
25  .L4:
26          lw      a5,-20(s0)
27          addi    a5,a5,-1
28          mv      a0,a5
29          call    fibonacci_recursive
30          mv      s1,a0
31          lw      a5,-20(s0)
32          addi    a5,a5,-2
33          mv      a0,a5
34          call    fibonacci_recursive
35          mv      a5,a0
36          add     a5,s1,a5
37  .L5:
38          mv      a0,a5
39          lw      ra,28(sp)
40          lw      s0,24(sp)
41          lw      s1,20(sp)
42          addi    sp,sp,32
43          jr      ra
```

組譯

```
1   0x00000000  0xFF010113  addi x2 x2 -16
2   0x00000004  0x00112623  sw x1 12(x2)
3   0x00000008  0x00812423  sw x8 8(x2)
4   0x0000000C  0x01010413  addi x8 x2 16
5   0x00000010  0x00A00513  addi x10 x0 10
6   0x00000014  0x00000317  auipc x6 0
7   0x00000018  0x014300E7  jalr x1 x6 20
8   0x0000001C  0x00050793  addi x15 x10 0
9   0x00000020  0x00078F93  addi x31 x15 0
10  0x00000024  0x0000006F  jal x0 0
11  0x00000028  0xFE010113  addi x2 x2 -32
12  0x0000002C  0x00112E23  sw x1 28(x2)
13  0x00000030  0x00812C23  sw x8 24(x2)
14  0x00000034  0x00912A23  sw x9 20(x2)
15  0x00000038  0x02010413  addi x8 x2 32
16  0x0000003C  0xFEA42623  sw x10 -20(x8)
17  0x00000040  0xFEC42703  lw x14 -20(x8)
18  0x00000044  0x00100793  addi x15 x0 1
19  0x00000048  0x00E7C663  blt x15 x14 12
20  0x0000004C  0xFEC42783  lw x15 -20(x8)
21  0x00000050  0x0380006F  jal x0 56
22  0x00000054  0xFEC42783  lw x15 -20(x8)
23  0x00000058  0xFFF78793  addi x15 x15 -1
24  0x0000005C  0x00078513  addi x10 x15 0
25  0x00000060  0x00000317  auipc x6 0
26  0x00000064  0xFC8300E7  jalr x1 x6 -56
27  0x00000068  0x00050493  addi x9 x10 0
28  0x0000006C  0xFEC42783  lw x15 -20(x8)
29  0x00000070  0xFFE78793  addi x15 x15 -2
30  0x00000074  0x00078513  addi x10 x15 0
31  0x00000078  0x00000317  auipc x6 0
32  0x0000007C  0xFB0300E7  jalr x1 x6 -80
33  0x00000080  0x00050793  addi x15 x10 0
34  0x00000084  0x00F487B3  add x15 x9 x15
35  0x00000088  0x00078513  addi x10 x15 0
36  0x0000008C  0x01C12083  lw x1 28(x2)
37  0x00000090  0x01812403  lw x8 24(x2)
38  0x00000094  0x01412483  lw x9 20(x2)
39  0x00000098  0x02010113  addi x2 x2 32
40  0x0000009C  0x00008067  jalr x0 x1 0
```

C 語言範例 1：費式數列

- 使用 “asm2sv.exe” 產生 Program_Rom.sv 後利用 ModelSim 觀察程式執行結果。
- 費式數列：0, 1, 2, 3, 5, 8, 13, 21, 34, 55(0x37)



C 語言範例 2：10 階層

```c
// 計算階層的函數 ( 遞迴方式 )
int factorial_recursive(int); // 函式宣告

int main() {
    // 使用 register 關鍵字並指定 asm("x31")
    // 強制將變數 ans 綁定到 RISC-V 的 x31 暫存器
    register int ans asm("x31");

    ans = factorial_recursive(10);

    while(1);
}

int factorial_recursive(int n) { // 函式定義
    if (n == 0 || n == 1) {
        return 1;
    }
    return n * factorial_recursive(n - 1);
}
```

※ 請將函式的宣告與定義分開撰寫

編譯

```
main:
        addi    sp,sp,-16
        sw      ra,12(sp)
        sw      s0,8(sp)
        addi    s0,sp,16
        li      a0,10
        call    factorial_recursive(int)
        mv      a5,a0
        mv      t6,a5
.L2:
        j       .L2
factorial_recursive(int):
        addi    sp,sp,-32
        sw      ra,28(sp)
        sw      s0,24(sp)
        addi    s0,sp,32
        sw      a0,-20(s0)
        lw      a5,-20(s0)
        beq     a5,zero,.L4
        lw      a4,-20(s0)
        li      a5,1
        bne     a4,a5,.L5
.L4:
        li      a5,1
        j       .L6
.L5:
        lw      a5,-20(s0)
        addi    a5,a5,-1
        mv      a0,a5
        call    factorial_recursive(int)
        mv      a4,a0
        lw      a5,-20(s0)
        mul     a5,a4,a5
.L6:
        mv      a0,a5
        lw      ra,28(sp)
        lw      s0,24(sp)
        addi    sp,sp,32
        jr      ra
```

暫存器	ABI	註解
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary/alternate link register
x6 ~ x7	t1 ~ t3	Temporaries
x8 ~ x9	s0 ~ s1	Saved registers
x10 ~ x11	a0 ~ a1	Function arguments/return values

暫存器	ABI	註解
x12 ~ x17	a2 ~ a7	Function arguments
x18 ~ x27	s2 ~ s11	Saved registers
x28 ~ x31	t3 ~ t6	Temporaries

C 語言範例 2：10 階層

- 由於本實作使用較小的記憶體 (2^9 Bytes)，因此需要重新指定 SP (Stack Pointer) 的位址，**li sp, 0x1fc** 。
- 組譯的過程不需要 Label 中的參數，fibonacci_recursive(int); 。

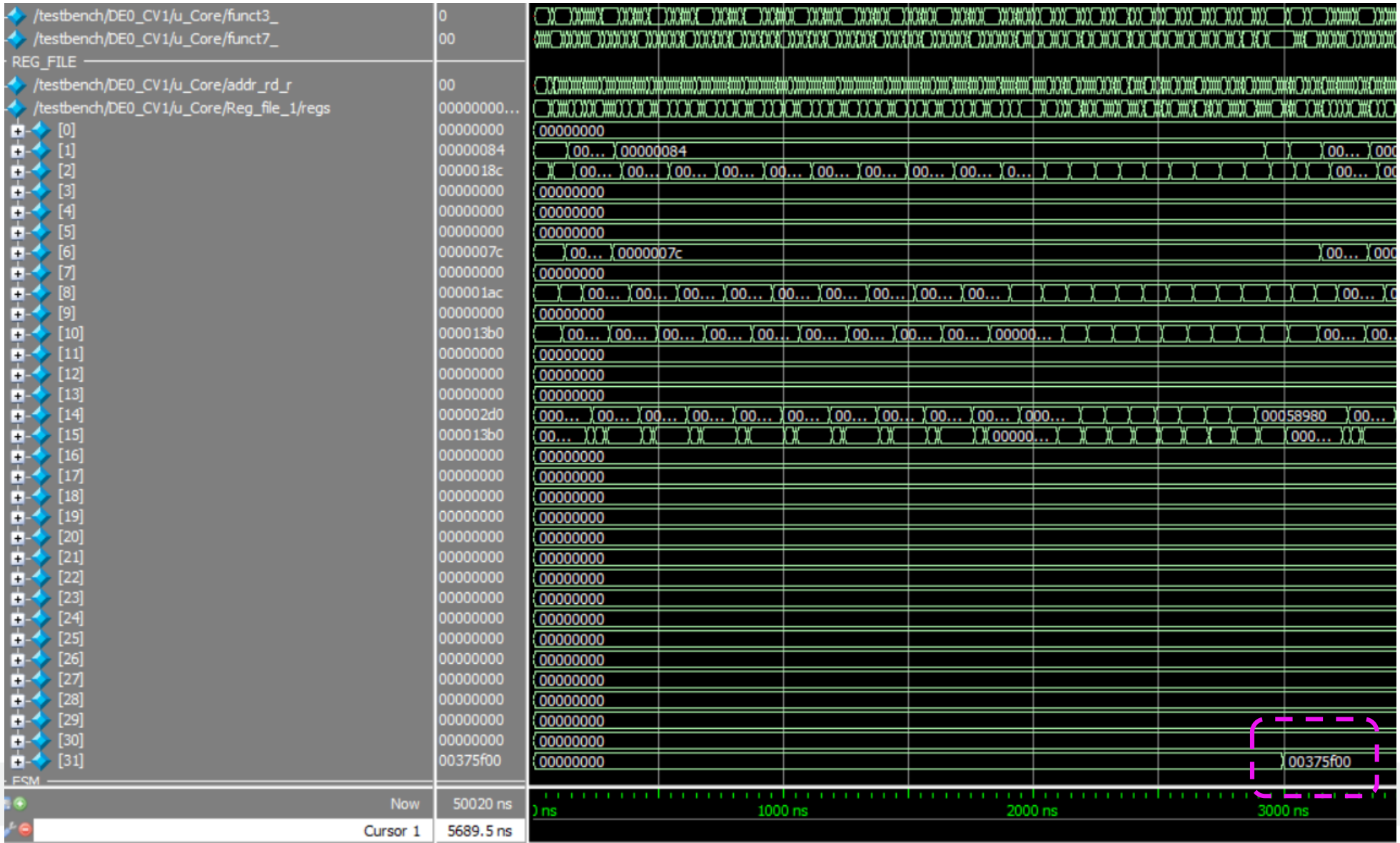
```
1         li      sp,0x1fc
2   main:
3         addi    sp,sp,-16
4         sw      ra,12(sp)
5         sw      s0,8(sp)
6         addi    s0,sp,16
7         li      a0,10
8         call    factorial_recursive
9         mv      a5,a0
10        mv      t6,a5
11  .L2:
12        j       .L2
13  factorial_recursive:
14        addi    sp,sp,-32
15        sw      ra,28(sp)
16        sw      s0,24(sp)
17        addi    s0,sp,32
18        sw      a0,-20(s0)
19        lw      a5,-20(s0)
20        beq     a5,zero,.L4
21        lw      a4,-20(s0)
22        li      a5,1
23        bne     a4,a5,.L5
24  .L4:
25        li      a5,1
26        j       .L6
27  .L5:
28        lw      a5,-20(s0)
29        addi    a5,a5,-1
30        mv      a0,a5
31        call    factorial_recursive
32        mv      a4,a0
33        lw      a5,-20(s0)
34        mul     a5,a4,a5
35  .L6:
36        mv      a0,a5
37        lw      ra,28(sp)
38        lw      s0,24(sp)
39        addi    sp,sp,32
40        jr      ra
```

組譯

```
1   0x00000000   0x1FC00113   addi x2 x0 508
2   0x00000004   0xFF010113   addi x2 x2 -16
3   0x00000008   0x00112623   sw x1 12(x2)
4   0x0000000C   0x00812423   sw x8 8(x2)
5   0x00000010   0x01010413   addi x8 x2 16
6   0x00000014   0x00A00513   addi x10 x0 10
7   0x00000018   0x00000317   auipc x6 0
8   0x0000001C   0x014300E7   jalr x1 x6 20
9   0x00000020   0x00050793   addi x15 x10 0
10  0x00000024   0x00078F93   addi x31 x15 0
11  0x00000028   0x0000006F   jal x0 0
12  0x0000002C   0xFE010113   addi x2 x2 -32
13  0x00000030   0x00112E23   sw x1 28(x2)
14  0x00000034   0x00812C23   sw x8 24(x2)
15  0x00000038   0x02010413   addi x8 x2 32
16  0x0000003C   0xFEA42623   sw x10 -20(x8)
17  0x00000040   0xFEC42783   lw x15 -20(x8)
18  0x00000044   0x00078863   beq x15 x0 16
19  0x00000048   0xFEC42703   lw x14 -20(x8)
20  0x0000004C   0x00100793   addi x15 x0 1
21  0x00000050   0x00F71663   bne x14 x15 12
22  0x00000054   0x00100793   addi x15 x0 1
23  0x00000058   0x0240006F   jal x0 36
24  0x0000005C   0xFEC42783   lw x15 -20(x8)
25  0x00000060   0xFFF78793   addi x15 x15 -1
26  0x00000064   0x00078513   addi x10 x15 0
27  0x00000068   0x00000317   auipc x6 0
28  0x0000006C   0xFC4300E7   jalr x1 x6 -60
29  0x00000070   0x00050713   addi x14 x10 0
30  0x00000074   0xFEC42783   lw x15 -20(x8)
31  0x00000078   0x02F707B3   mul x15 x14 x15
32  0x0000007C   0x00078513   addi x10 x15 0
33  0x00000080   0x01C12083   lw x1 28(x2)
34  0x00000084   0x01812403   lw x8 24(x2)
35  0x00000088   0x02010113   addi x2 x2 32
36  0x0000008C   0x00008067   jalr x0 x1 0
```

C 語言範例 2 : 10 階層

- 使用 “asm2sv.exe” 產生 Program_Rom.sv 後利用 ModelSim 觀察程式執行結果。(10! = 0x00375f00)



上課實作 1 : GCD

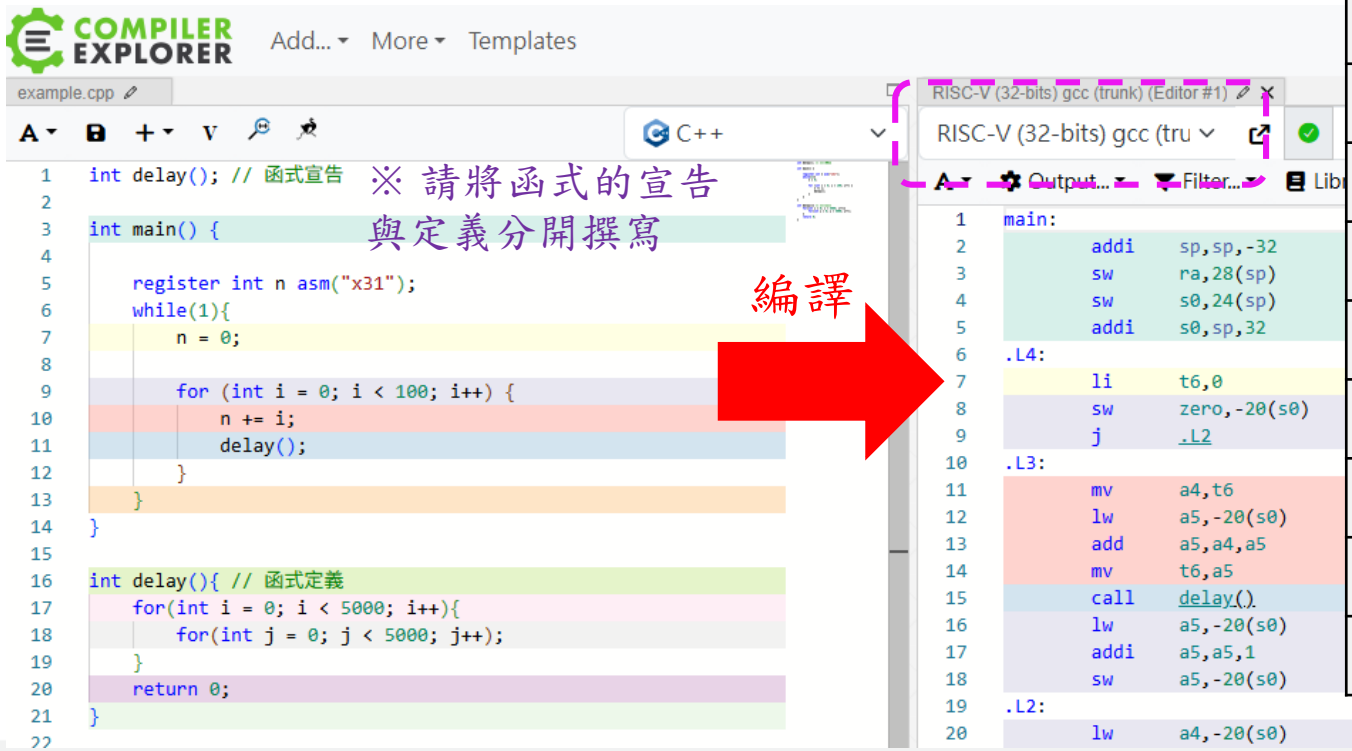
- 撰寫一個 C 程式找出 323 和 209 的最大公因數。
- 請將結果 (最大公因數 = 19) 載入到 x31(t6) 暫存器中 。
- 注意 Function 寫法。
- 使用線上編譯器將撰寫的 C 程式編譯成組合語言。
- Compiler 選擇 RISC-V (32-bits) gcc (trunk)。
- 由於本實作使用較小的記憶體 (2^9 Bytes) ，因此需要重新指定 SP (Stack Pointer) 的位址，**li sp, 0x1fc** 。
- 使用 VSCode 的 RISC-V Venus Simulator 組譯完後以 “asm2sv.exe” 產生 Program_Rom.sv。
- 使用 ModelSim 觀察程式執行結果。

上課實作 2：排列組合

- 撰寫一個 C 程式 C 10 取 3。
- 請將 10 存在 t4，3 存在 t5。
- 請將結果 (120) 載入到 x31(t6) 暫存器中 。
- 注意 Function 寫法。
- 使用線上編譯器將撰寫的 C 程式編譯成組合語言。
- Compiler 選擇 RISC-V (32-bits) gcc (trunk)。
- 由於本實作使用較小的記憶體 (2^9 Bytes)，因此需要重新指定 SP (Stack Pointer) 的位址，**li sp, 0x1fc** 。
- 使用 VSCode 的 RISC-V Venus Simulator 組譯完後以 “asm2sv.exe” 產生 Program_Rom.sv。
- 使用 ModelSim 觀察程式執行結果。

上課實作：FPGA 燒錄

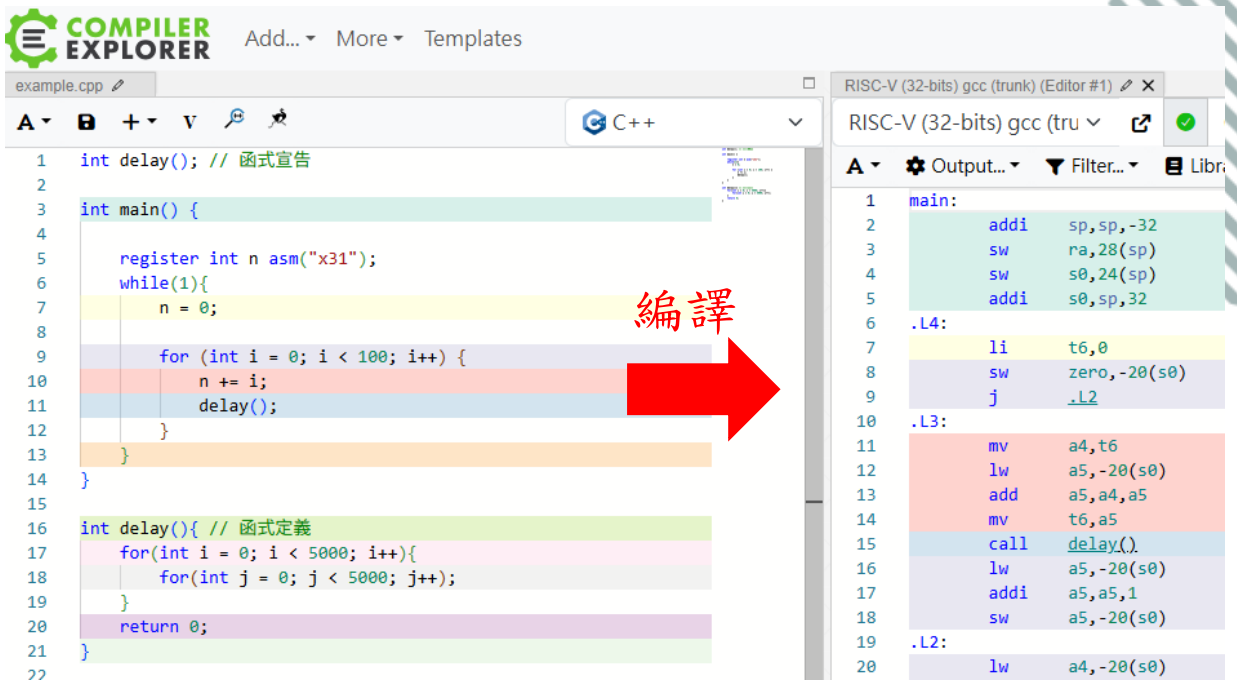
- 撰寫一個循環執行 0 累加到 99 的 C 程式。
- 使用線上編譯器將撰寫的 C 程式編譯成組合語言。
- Compiler 選擇 RISC-V (32-bits) gcc (trunk)。
- 注意 Function 寫法。



暫存器	ABI	註解
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary/alternate link register
x6 ~ x7	t1 ~ t3	Temporaries
x8 ~ x9	s0 ~ s1	Saved registers
x10 ~ x11	a0 ~ a1	Function arguments/return values
x12 ~ x17	a2 ~ a7	Function arguments
x18 ~ x27	s2 ~ s11	Saved registers
x28 ~ x31	t3 ~ t6	Temporaries

上課實作：FPGA 燒錄

- 由於本實作使用較小的記憶體 (2^9 Bytes)，因此需要重新指定 SP (Stack Pointer) 的位址，**li sp, 0x1fc**。
- 使用 VSCode 的 RISC-V Venus Simulator 組譯完後以“asm2sv.exe”產生 Program_Rom.sv
- 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。




```
module Program_Rom(
    input   logic [31:0] Rom_addr,
    output  logic [31:0] Rom_data
);

    always_comb begin
        case (Rom_addr)
            32'h0  : Rom_data = 32'h1fc00113;
            32'h4  : Rom_data = 32'hfe010113;
            32'h8  : Rom_data = 32'h00112e23;
```

asm2sv.exe

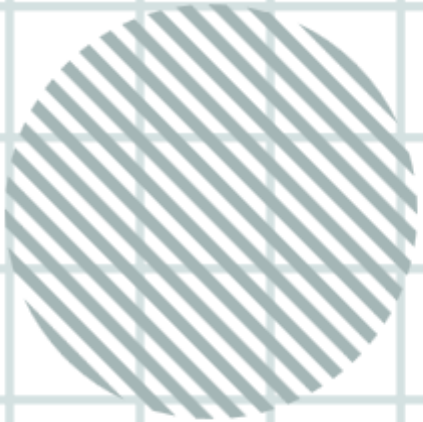
assembly.txt

```
0x00000000    0x1FC00113    addi x2 x0 508
0x00000004    0xFE010113    addi x2 x2 -32
0x00000008    0x00112E23    sw x1 28(x2)
0x0000000C    0x00812C23    sw x8 24(x2)
0x00000010    0x02010413    addi x8 x2 32
0x00000014    0x00000F93    addi x31 x0 0
0x00000018    0xFE042623    sw x0 -20(x8)
0x0000001C    0x0280006F    jal x0 40
0x00000020    0x000F8713    addi x14 x31 0
0x00000024    0xFEC42783    lw x15 -20(x8)
```

組譯

指定 SP

```
        li      sp, 0x1fc
main:
        addi    sp,sp,-32
        sw      ra,28(sp)
        sw      s0,24(sp)
        addi    s0,sp,32
.L4:
        li      t6,0
        sw      zero,-20(s0)
        j       .L2
```



THANK YOU