

## 任務

**任務：**驗證這些32位元（0x01de0e33）是否對應於RISC-V架構中的指令add t3,t3,t4。

0x01de0e33 → 0000000 11101 11100 000 11100 0110011

funct7 = 0000000

rs2 = 11101 = x29 (t4)

rs1 = 11100 = x28 (t3)

funct3 = 000

rd = 11100 = x28 (t3)

op = 0110011


來自DDCARV的附錄B：

31:25	24:20	19:15	14:12	11:7	6:0	
funct7	rs2	rs1	funct3	rd	op	R-Type

op	funct3	funct7	Type	Instruction	Description	Operation
0110011 (51)	000	0000000	R	add rd, rs1, rs2	add	rd = rs1 + rs2

Name	Register Number	Use
<b>zero</b>	<b>x0</b>	Constant value 0
<b>ra</b>	<b>x1</b>	Return address
<b>sp</b>	<b>x2</b>	Stack pointer
<b>gp</b>	<b>x3</b>	Global pointer
<b>tp</b>	<b>x4</b>	Thread pointer
<b>t0-2</b>	<b>x5-7</b>	Temporary variables
<b>s0/fp</b>	<b>x8</b>	Saved variable / Frame pointer
<b>s1</b>	<b>x9</b>	Saved variable
<b>a0-1</b>	<b>x10-11</b>	Function arguments / Return values
<b>a2-7</b>	<b>x12-17</b>	Function arguments
<b>s2-11</b>	<b>x18-27</b>	Saved variables
<b>t3-6</b>	<b>x28-31</b>	Temporary variables

**任務：**在自己的電腦上重複圖3中的模擬過程。為此，請按照以下步驟操作（在GSG的第7部分中詳述）：

- 必要時產生模擬二進位檔案（*Vrvfpgasim*）。
- 在PlatformIO中，開啟在以下位置提供的專案：*[RVfpgaPath]/RVfpga/Labs/Lab12/ADD\_Instruction*。
- 在檔案*platformio.ini*中建立到RVfpga模擬二進位檔案（*Vrvfpgasim*）的正確路徑。
- 使用Verilator產生模擬軌跡（產生軌跡）。
- 在GTKWave上開啟軌跡。
- 使用檔案*test\_1.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab12/ADD\_Instruction/*中提供）開啟與圖3所示訊號相同的訊號。為此，在GTKWave上，按一下「*File – Read Tcl Script File*」（檔案 – 讀取Tcl指令碼檔案）並選擇*test\_1.tcl*檔案。
- 按幾次「*Zoom In*」（放大）（）移動至15000 ps。

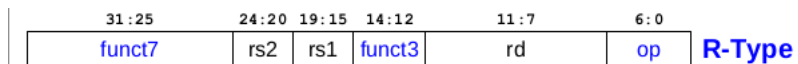
解答請參見實驗12的主文件。

**任務：**在SweRV EH1處理器的Verilog檔案中找到圖6中的主要結構和訊號。

- 模組 **dec\_decode\_ctl** 中的控制單元
- 暫存器檔案：
  - 模組 **dec** 第525行中的實例化。
  - 模組 **dec\_gpr\_ctl** 中的實作。
- 解碼階段的3:1多工器：模組 **exu** 的第279行。
- 控制訊號的管線暫存器：分布在多個模組中。
- 暫存器 **aff** 和 **bff**：模組 **exu\_alu\_ctl** 的第90行和第92行。
- EX1階段的I0 ALU：
  - 模組 **exu** 第401行中的實例化。
  - 模組 **exu\_alu\_ctl** 中的實作。
- 包含運算結果的管線暫存器（**i0e2resultff**、**i0e3resultff**、**i0e4resultff** 和 **i0wbresultff**）：模組 **dec\_decode\_ctl** 的第2260-2283行。
- EX3階段的3:1多工器：模組 **dec\_decode\_ctl** 的第2268行。
- EX4階段的3:1多工器：模組 **dec\_decode\_ctl** 的第2277行。
- 寫回階段的2:1多工器：模組 **dec\_decode\_ctl** 的第2286行。

**任務：**在Verilog程式碼（模組 **dec\_decode\_ctl**）中尋找如何使用 **i0r** 控制訊號讀取暫存器檔案。

- 暫存器識別碼從通路0中的32位元指令取得：訊號 **i0[31:0] = dec\_i0\_instr\_d[31:0]**。  
在R型指令中，它們位於以下欄位中：



在模組 **dec\_decode\_ctl** 中：

```
1121 assign i0r.rs1[4:0] = i0[19:15];
1122 assign i0r.rs2[4:0] = i0[24:20];
1123 assign i0r.rd[4:0] = i0[11:7];
```

- 暫存器識別碼和讀取啟用訊號分配給 `dec_i0_rs1_d/dec_i0_rs2_d` 和 `dec_i0_rs1_en_d/dec_i0_rs2_en_d`。這些訊號從模組 **dec** 傳送到模組 **dec\_decode\_ctl**。在模組 **dec\_decode\_ctl** 中：

```
1130 assign dec_i0_rs1_en_d = i0_dp.rs1 & (i0r.rs1[4:0] != 5'd0);
1131 assign dec_i0_rs2_en_d = i0_dp.rs2 & (i0r.rs2[4:0] != 5'd0);
1132 assign i0_rd_en_d = i0_dp.rd & (i0r.rd[4:0] != 5'd0);
1133
1134 assign dec_i0_rs1_d[4:0] = i0r.rs1[4:0];
1135 assign dec_i0_rs2_d[4:0] = i0r.rs2[4:0];
1136 assign i0_rd_d[4:0] = i0r.rd[4:0];
```

- 暫存器識別碼和讀取啟用訊號提供給暫存器檔案，該檔案在模組 **dec** 中實例化。在模組 **dec** 中：

```
525 dec_gpr_ctl #(.GPR_BANKS(GPR_BANKS),
526               .GPR_BANKS_LOG2(GPR_BANKS_LOG2)) arf (.,
527               // inputs
528               .raddr0(dec_i0_rs1_d[4:0]), .rden0(dec_i0_rs1_en_d),
529               .raddr1(dec_i0_rs2_d[4:0]), .rden1(dec_i0_rs2_en_d),
530               .raddr2(dec_i1_rs1_d[4:0]), .rden2(dec_i1_rs1_en_d),
531               .raddr3(dec_i1_rs2_d[4:0]), .rden3(dec_i1_rs2_en_d),
532
533               .waddr0(dec_i0_waddr_wb[4:0]), .wen0(dec_i0_wen_wb), .wd0(dec_i0_wdata_wb[31:0]),
534               .waddr1(dec_i1_waddr_wb[4:0]), .wen1(dec_i1_wen_wb), .wd1(dec_i1_wdata_wb[31:0]),
535               .waddr2(dec_nonblock_load_waddr[4:0]), .wen2(dec_nonblock_load_wen), .wd2(lsu_nonblock_load_data[31:0]),
536
537               // outputs
538               .rd0(gpr_i0_rs1_d[31:0]), .rd1(gpr_i0_rs2_d[31:0]),
539               .rd2(gpr_i1_rs1_d[31:0]), .rd3(gpr_i1_rs2_d[31:0])
540               );
```

**任務：**在 Verilog 程式碼（模組 **exu**）中尋找 `i0_ap` 和 `dd` 控制訊號如何從解碼階段傳播到執行階段。此外，還需尋找 `dd` 控制訊號周遊解碼到寫回的所有階段之後，如何在寫回階段被暫存器檔案使用。

訊號*i0\_ap*在模組**dec\_decode\_ctl**中取得。它提供給模組**exu**，並從中傳播到EX1、EX2、EX3和提交（EX4）階段。在模組**exu**中：

```
454   rvdffe #($bits(alu_pkt_t)) i0_ap_e1_ff (.*, .en(i0_e1_ctl_en), .din(i0_ap), .dout(i0_ap_e1) );
455   rvdffe #($bits(alu_pkt_t)) i0_ap_e2_ff (.*, .en(i0_e2_ctl_en), .din(i0_ap_e1), .dout(i0_ap_e2) );
456   rvdffe #($bits(alu_pkt_t)) i0_ap_e3_ff (.*, .en(i0_e3_ctl_en), .din(i0_ap_e2), .dout(i0_ap_e3) );
457   rvdffe #($bits(alu_pkt_t)) i0_ap_e4_ff (.*, .en(i0_e4_ctl_en), .din(i0_ap_e3), .dout(i0_ap_e4) );
```

訊號*dd*在模組**dec\_decode\_ctl**中取得，並傳播到EX1、EX2、EX3、提交（EX4）和WB（EX5）階段。在模組**dec\_decode\_ctl**中：

```
2139   rvdffe #($bits(dest_pkt_t) ) e1ff (.*, .en(i0_e1_ctl_en), .din(dd), .dout(e1d));
2155   rvdffe #($bits(dest_pkt_t) ) e2ff (.*, .en(i0_e2_ctl_en), .din(e1d_in), .dout(e2d));
2168   rvdffe #($bits(dest_pkt_t) ) e3ff (.*, .en(i0_e3_ctl_en), .din(e2d_in), .dout(e3d));
2193   rvdffe #($bits(dest_pkt_t) ) e4ff (.*, .en(i0_e4_ctl_en), .din(e3d_in), .dout(e4d));
2219   rvdffe #($bits(dest_pkt_t) ) wbff (.*, .en(i0_wb_ctl_en | exu_div_finish | div_wen_wb), .din(e4d_in), .dout(wbd));
```

請注意，在進入下一個暫存器之前，每個暫存器的輸出都會稍作修改（並因此重新命名）。如果要查看詳細資訊，可以查看Verilog程式碼。

輸出運算元的暫存器識別碼在解碼階段分配：

```
2070   assign dd.i0rd[4:0] = i0r.rd[4:0];
```

訊號*dd*從解碼階段傳播到寫回階段（如上所示）：*dd* → *e1d* → *e2d* → *e3d* → *e4d* → *wbd*。隨後，目標暫存器在寫回階段提供給暫存器檔案：

```
2221   assign dec_i0_waddr_wb[4:0] = wbd.i0rd[4:0];
```

```

525     dec_gpr_ctl #(.GPR_BANKS(GPR_BANKS),
526                 .GPR_BANKS_LOG2(GPR_BANKS_LOG2)) arf (.*,
527                 // inputs
528                 .raddr0(dec_i0_rs1_d[4:0]), .rden0(dec_i0_rs1_en_d),
529                 .raddr1(dec_i0_rs2_d[4:0]), .rden1(dec_i0_rs2_en_d),
530                 .raddr2(dec_i1_rs1_d[4:0]), .rden2(dec_i1_rs1_en_d),
531                 .raddr3(dec_i1_rs2_d[4:0]), .rden3(dec_i1_rs2_en_d),
532
533                 .waddr0(dec_i0_waddr_wb[4:0]), .wen0(dec_i0_wen_wb), .wd0(dec_i0_wdata_wb[31:0]),
534                 .waddr1(dec_i1_waddr_wb[4:0]), .wen1(dec_i1_wen_wb), .wd1(dec_i1_wdata_wb[31:0]),
535                 .waddr2(dec_nonblock_load_waddr[4:0]), .wen2(dec_nonblock_load_wen), .wd2(lsu_nonblock_load_data[31:0]),
536
537                 // outputs
538                 .rd0(gpr_i0_rs1_d[31:0]), .rd1(gpr_i0_rs2_d[31:0]),
539                 .rd2(gpr_i1_rs1_d[31:0]), .rd3(gpr_i1_rs2_d[31:0])
540             );

```

**任務：**這兩個訊號（`i0_e1_ctl_en`和`dec_i0_alu_decode_d`）的產生過程相當複雜，這裡不做詳細說明，但您可自行在模組 **dec\_decode\_ctl**和**exu**中進一步分析。

不提供解答。

**任務：**在Verilog程式碼（模組**exu**）中尋找底部的3:1多工器（第二個輸入運算元）並嘗試找到其輸入的來源（圖6中僅顯示來自暫存器檔案的輸入）。不需要太仔細地查看輸入，因為它們將在第3部分和後續實驗提供的練習中進行分析。

```

286     assign i0_rs2_d[31:0] = ({32{~dec_i0_rs2_bypass_en_d}} & gpr_i0_rs2_d[31:0]) |
287                             ({32{~dec_i0_rs2_bypass_en_d}} & dec_i0_immed_d[31:0]) |
288                             ({32{ dec_i0_rs2_bypass_en_d}} & i0_rs2_bypass_data_d[31:0]);

```

這些3:1多工器接收3個輸入：

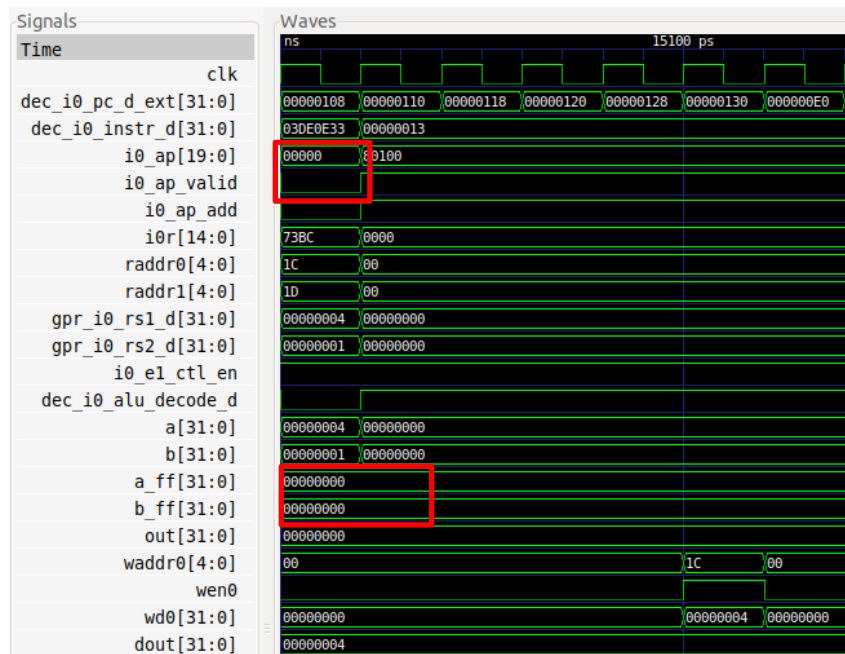
- 一個來自暫存器檔案（`gpr_i0_rs2_d`）
- 一個來自32位元指令暫存器，它構成立即數（`dec_i0_immed_d`）
- 一個來自旁路邏輯，我們將在實驗15中分析（`i0_rs2_bypass_data_d`）

**任務：**在自己的電腦上重複圖7中的模擬過程。可以使用以下位置提供的.tcl指令碼：  
[RVfpgaPath]/RVfpga/Labs/Lab12/ADD\_Instruction/test\_2.tcl。請注意，該.tcl檔案中為一些控制位元使用了別名。

解答請參見實驗12的主文件。

**任務：**在圖2的範例中，將add指令替換為非A-L指令（例如mul指令）。驗證i0\_ap訊號的所有欄位是否均等於0，等於0時I0 ALU不起作用（對於該指令，EX1階段I0管道的訊號a\_ff和b\_ff將保持不變）。可以使用與圖7中範例所用test\_2.tcl檔案相同的檔案。

例如，mul t3, t3, t4的模擬（0x03de0e33）提供以下結果：

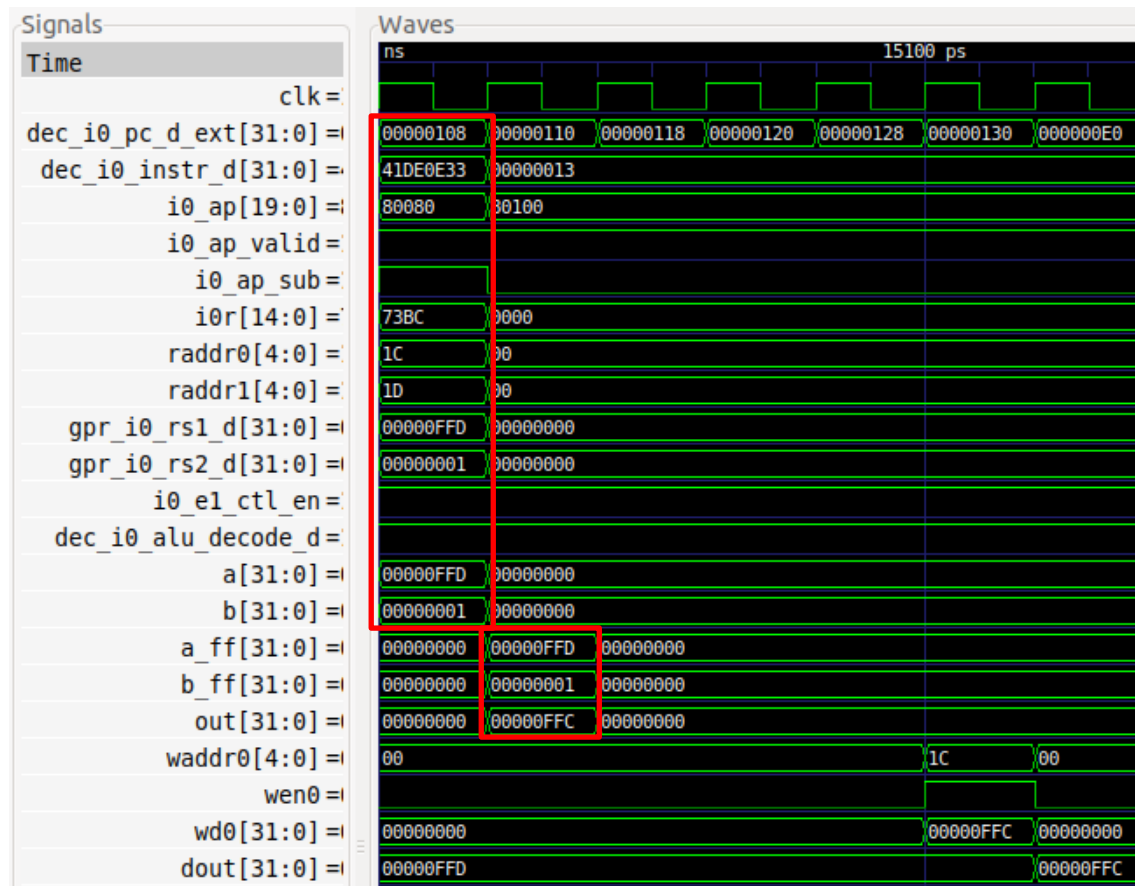


**任務：**將本部分中分析的新訊號包含在圖7的模擬中。

不提供解答。

**任務：**對sub指令執行與圖7中的模擬類似的模擬。請記住，可以透過.tcl檔案將新訊號包含在模擬中。

例如，sub t3, t3, t4的模擬（0x41de0e33）提供以下結果：





**任務：**分析模組 `exu_alu_ctl` 中實作的加法器/減法器的Verilog實作。圖8透過顯示與加法和減法運算直接相關的邏輯來提供一些協助。

```
90      rvdffe #(32) aff (.*, .en(enable & valid), .din(a[31:0]), .dout(a_ff[31:0]));
91
92      rvdffe #(32) bff (.*, .en(enable & valid), .din(b[31:0]), .dout(b_ff[31:0]));
```

輸入運算元從解碼階段（a和b）傳播到執行階段（a\_ff和b\_ff）。

```
135      assign bm[31:0] = ( ap.sub ) ? ~b_ff[31:0] : b_ff[31:0];
136
137
138      assign {cout, aout[31:0]} = {1'b0, a_ff[31:0]} + {1'b0, bm[31:0]} + {32'b0, ap.sub};
139
```

這是加法器/減法器。

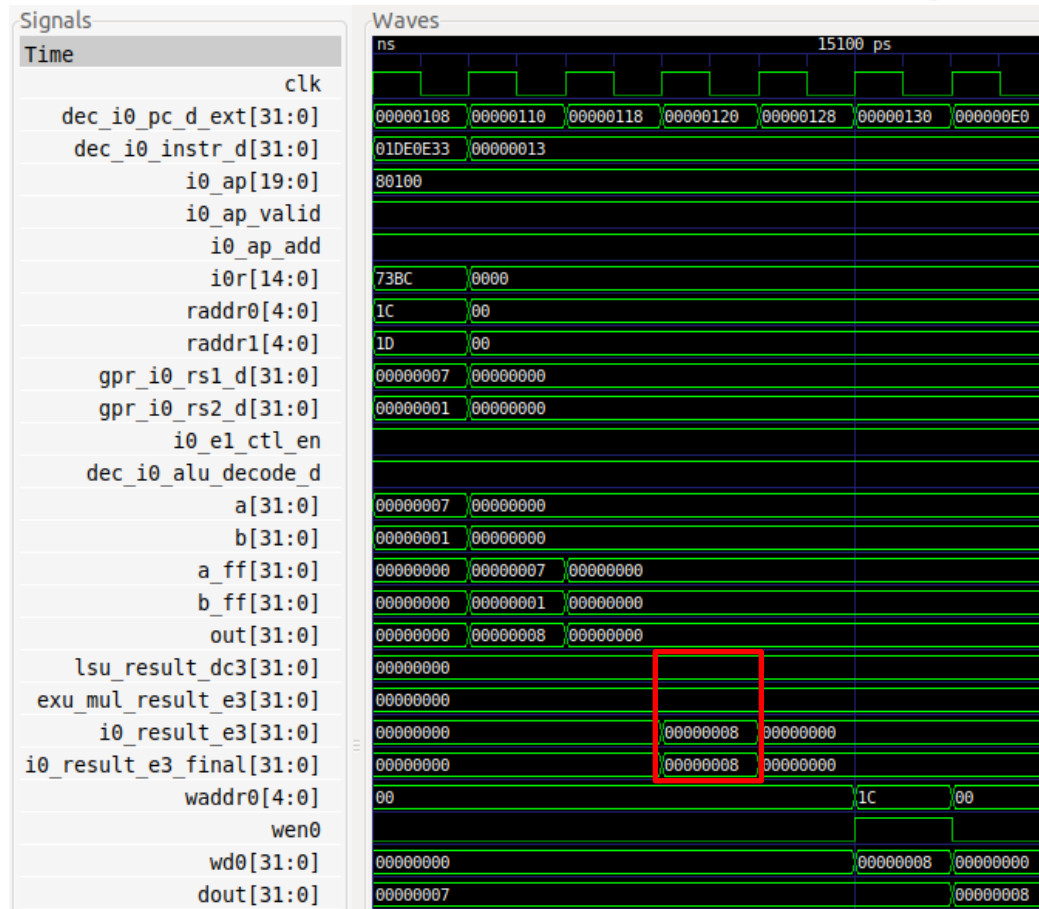
- 如果指令是加法，則 `aout = a_ff + b_ff`
- 如果指令是減法，則先計算 `b_ff` 的二進位補碼，然後計算 `a_out`。

```
172      assign sel_adder = (ap.add | ap.sub) & ~ap.slt;

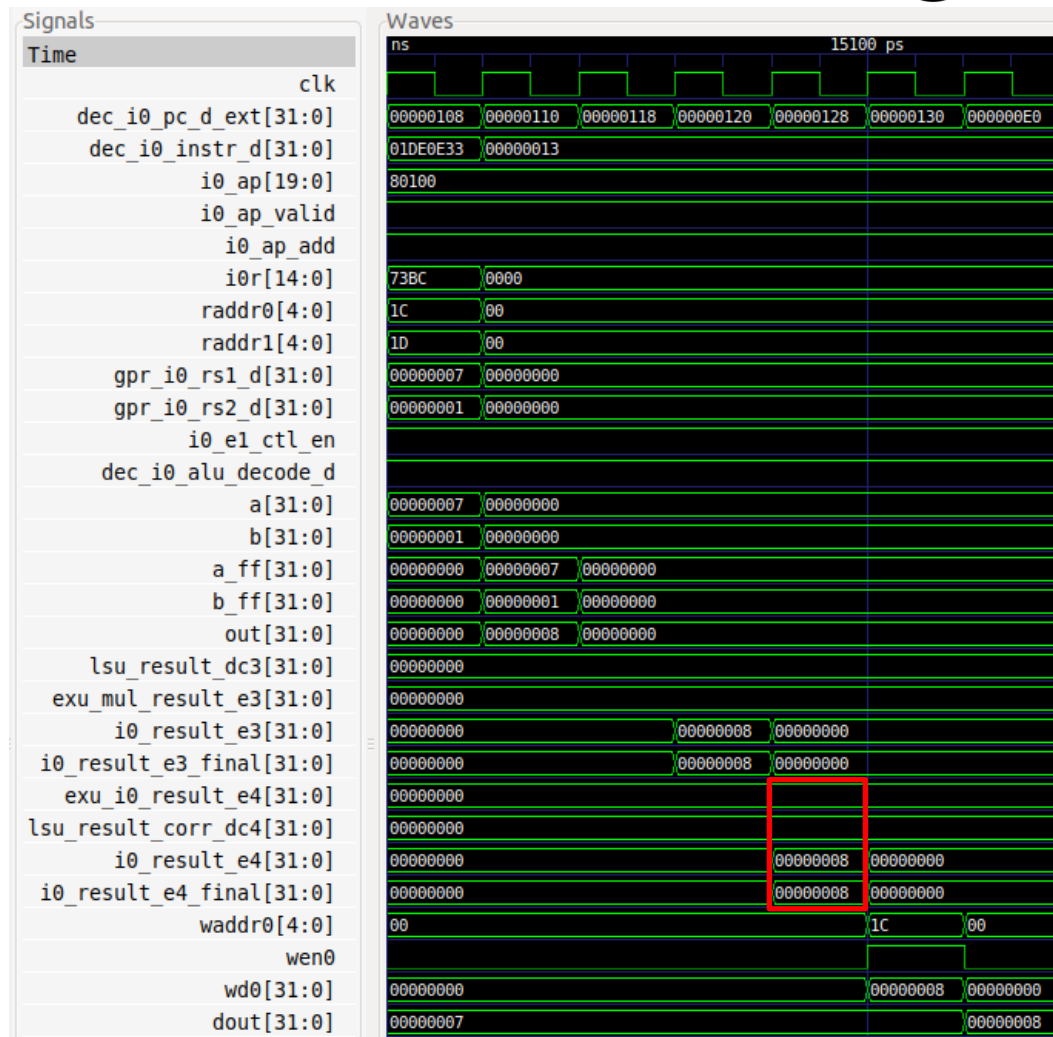
185      assign out[31:0] = ({32{sel_logic}} & lout[31:0]) |
186                        ({32{sel_shift}} & sout[31:0]) |
187                        ({32{sel_adder}} & aout[31:0]) |
188                        ({32{ap.jal | pp_ff.pcall | pp_ff.pja | pp_ff.pret}} & {pcout[31:1], 1'b0}) |
189                        ({32{ap.csr_write}} & ((ap.csr_imm) ? b_ff[31:0] : a_ff[31:0])) |
190                        ({31'b0, slt_one});
191
```

如果指令是加法或減法，則 `out = aout`。

**任務：**對於圖2中的範例，在模擬中驗證該多工器是否從add指令的預期管線中選擇結果。



**任務：**對於圖2中範例的add指令，在模擬中驗證該多工器是否從正確的輸入來源選擇結果（i0\_result\_e4）。



**任務：**在Verilog程式碼中，分析訊號wen0和waddr0如何在解碼階段產生並傳播到寫回階段。

```

525 dec_gpr_ctl #(.GPR_BANKS(GPR_BANKS),
526             .GPR_BANKS_LOG2(GPR_BANKS_LOG2)) arf (.*,
527             // inputs
528             .raddr0(dec_i0_rs1_d[4:0]), .rden0(dec_i0_rs1_en_d),
529             .raddr1(dec_i0_rs2_d[4:0]), .rden1(dec_i0_rs2_en_d),
530             .raddr2(dec_i1_rs1_d[4:0]), .rden2(dec_i1_rs1_en_d),
531             .raddr3(dec_i1_rs2_d[4:0]), .rden3(dec_i1_rs2_en_d),
532
533             .waddr0(dec_i0_waddr_wb[4:0]), .wen0(dec_i0_wen_wb), .wd0(dec_i0_wdata_wb[31:0]),
534             .waddr1(dec_i1_waddr_wb[4:0]), .wen1(dec_i1_wen_wb), .wd1(dec_i1_wdata_wb[31:0]),
535             .waddr2(dec_nonblock_load_waddr[4:0]), .wen2(dec_nonblock_load_wen), .wd2(lsu_nonblock_load_data[31:0]),
536
537             // outputs
538             .rd0(gpr_i0_rs1_d[31:0]), .rd1(gpr_i0_rs2_d[31:0]),
539             .rd2(gpr_i1_rs1_d[31:0]), .rd3(gpr_i1_rs2_d[31:0])
540         );
541

```

```

2221 assign dec_i0_waddr_wb[4:0] = wbd.i0rd[4:0];

```

```

2224 assign i0_wen_wb = wbd.i0v & ~(~dec_tlu_i1_kill_writeb_wb & ~i1_load_kill_wen & wbd.i0v & wbd.i1v & (wbd.i0rd[4:0] == wbd.i1rd[4:0])) & ~dec_tlu_i0_kill_writeb_wb;
2225 assign dec_i0_wen_wb = i0_wen_wb & ~i0_load_kill_wen; // don't write a nonblock load 1st time down the pipe
2226

```

```

2070 assign dd.i0rd[4:0] = i0r.rd[4:0];
2071 assign dd.i0v = i0_rd_en_d & i0_legal_decode_d;

```

## 練習

- 1) 對邏輯指令（and、or和xor）執行與本實驗中提供的分析類似的分析。

以下範例（在[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs\_Solutions/Lab12/AND\_Instruction中提供）說明了無限迴圈中包含的and指令的執行情況。與add指令的範例中一樣，and指令（以紅色強調顯示）前後有幾條nop指令。迴圈末尾包含兩條指令，用於修改儲存在t3和t4中的值。

```
#define INSERT_NOPS_1      nop;
#define INSERT_NOPS_2      nop; INSERT_NOPS_1
#define INSERT_NOPS_3      nop; INSERT_NOPS_2
#define INSERT_NOPS_4      nop; INSERT_NOPS_3
#define INSERT_NOPS_5      nop; INSERT_NOPS_4
#define INSERT_NOPS_6      nop; INSERT_NOPS_5
#define INSERT_NOPS_7      nop; INSERT_NOPS_6
#define INSERT_NOPS_8      nop; INSERT_NOPS_7
#define INSERT_NOPS_9      nop; INSERT_NOPS_8
#define INSERT_NOPS_10     nop; INSERT_NOPS_9

.globl main
main:

li t3, 0xFC                # t3 = 0xFC
li t4, 0x7                  # t4 = 0x7

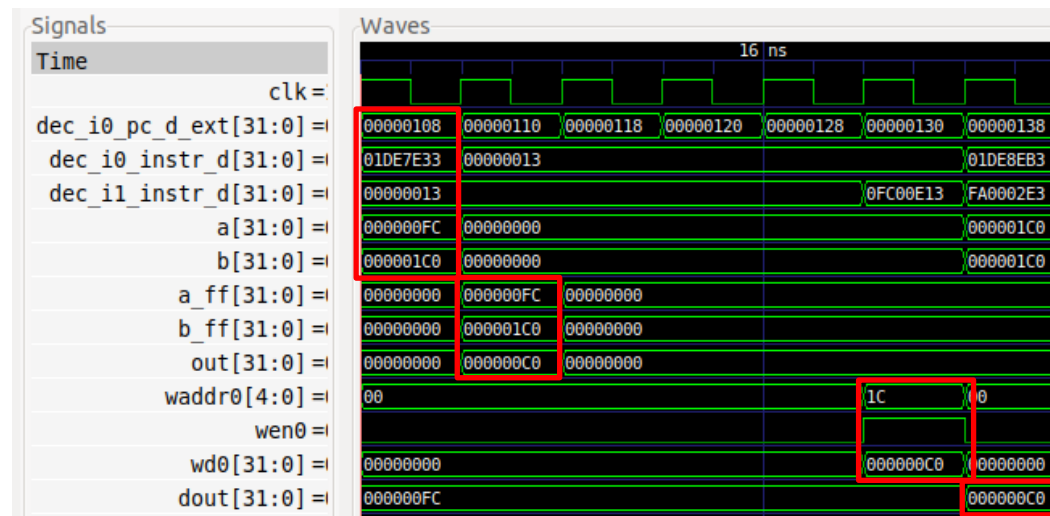
REPEAT:
    INSERT_NOPS_10
    and t3, t3, t4          # t3 = t3 & t4
    INSERT_NOPS_10
    li t3, 0xFC             # t3 = 0xFC
    add t4, t4, t4
    beq zero, zero, REPEAT # Repeat the loop

.end
```

如果在PlatformIO中開啟、編譯項目，然後開啟反組譯檔案（位於  
[RVfpgaPath]/RVfpga/Labs/RVfpgaLabsSolutions/Programs\_Solutions/Lab12/AND\_Instruction/.pio/build/swervolf\_nexys/firmware.dis中），可  
以看到and指令位於位址0x00000108處，還可以看到指令的機器程式碼（0x01de7e33）：

```
0x00000108:      01de7e33      and    t3,t3,t4
```

接下來，我們在Verilator中模擬程式，然後在GTKWave上開啟模擬器產生的追蹤檔案。移至迴圈的任何一次迭代（第一次除外）。



分析波形（以紅色強調顯示的值對應於and指令）。在本實驗中，我們將跳過擷取和對齊階段，這兩個階段將在後面的實驗中說明。

- **解碼**階段：訊號dec\_i0\_pc\_d\_ext包含指令的位址（在教材中，該訊號通常稱為程式計數器），and的位址為0x00000108，訊號dec\_i0\_instr\_d包含32位元機器指令0x01DE7E33（在教材中，該訊號通常稱為指令暫存器）。

在RISC-V中，and指令的操作碼如下（參見[Harris&Harris]的附錄B）：

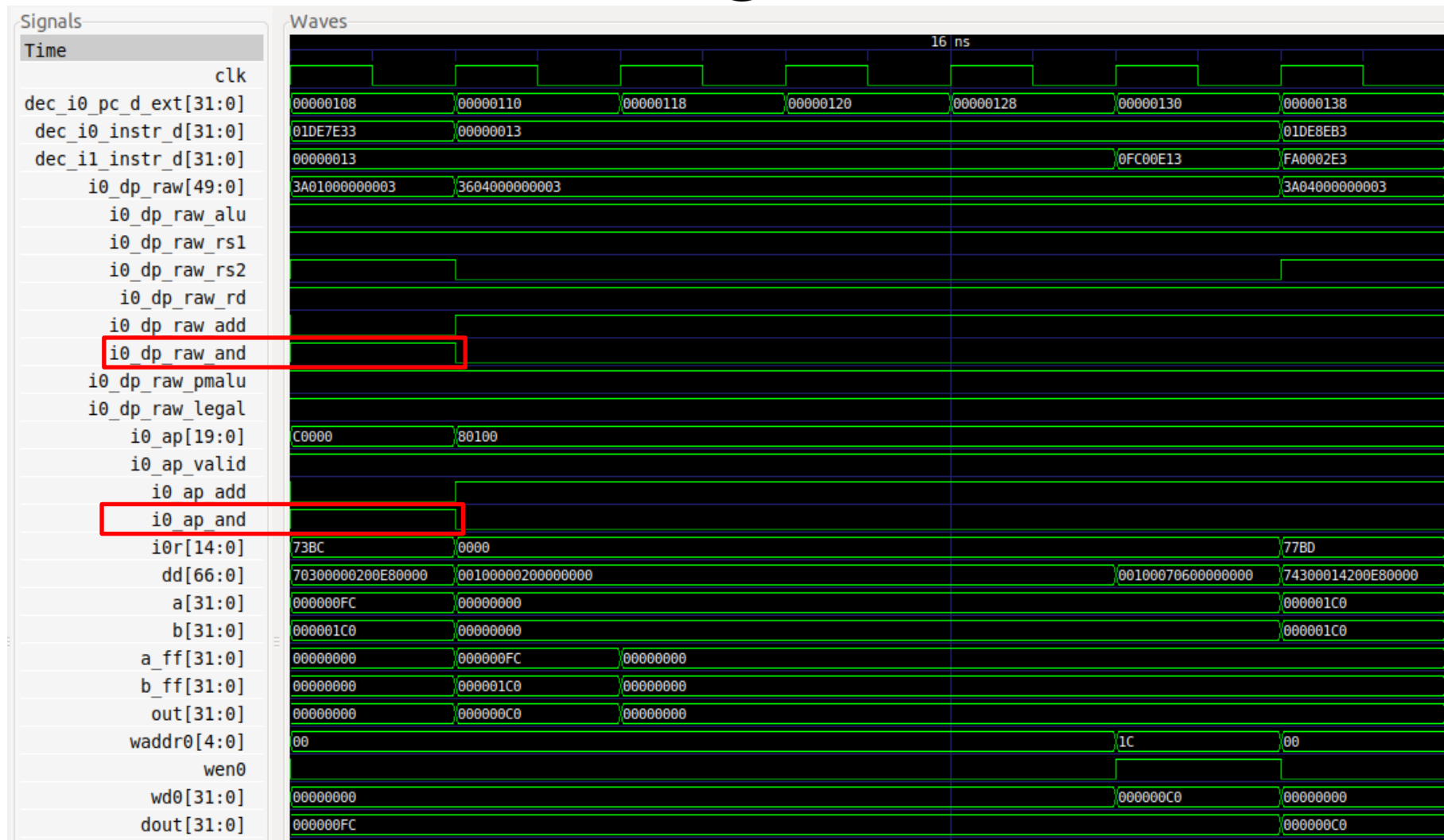
```
00000000 | rs2 | rs1 | 111 | rd | 0110011
```

因此可以輕鬆驗證0x01DEFE33是否對應於：and t3, t3, t4（請記住，t3=x28且t4=x29）。

在此階段將產生管線控制訊號（我們將在下一部分中詳細介紹）。此外，在此階段還將讀取暫存器檔案。訊號a和b包含ALU的輸入，本例中與從暫存器檔案讀取的值一致（對於後續實驗中將分析的其他範例，情況並非如此）。

- **EX1**階段：在下一週期中，將執行and指令。訊號a\_ff和b\_ff包含ALU的輸入（分別為0xFC和0x1C0），而out包含加法的結果（0xC0）。
- **EX5**階段（也稱為寫回）：最後，在4個週期後，加法結果透過訊號wd0=0xC0寫回到暫存器檔案中，其中包含要寫入的資料。鑒於wen0=1（寫入啟用），邏輯與運算結果在相應週期結束時寫入暫存器x28（暫存器索引，waddr0=0x1C）。可以發現，在接下來的週期（圖中最後一個週期）中，暫存器x28包含新值（dout=0xC0）。

接下來，我們將控制訊號新增到之前的模擬中：



可以看到，在第一個週期中，and指令的控制位元為1。



以下Verilog片段顯示了SweRV EH1的邏輯單元。

```

90      rvdffe #(32) aff (.*, .en(enable & valid), .din(a[31:0]), .dout(a_ff[31:0]));
91
92      rvdffe #(32) bff (.*, .en(enable & valid), .din(b[31:0]), .dout(b_ff[31:0]));
93
149     assign logic_sel[3] = ap.land | ap.lor;
150     assign logic_sel[2] = ap.lor | ap.lxor;
151     assign logic_sel[1] = ap.lor | ap.lxor;
152
153
154
155     assign lout[31:0] = ( a_ff[31:0] & b_ff[31:0] & {32{logic_sel[3]}} ) |
156                        ( a_ff[31:0] & ~b_ff[31:0] & {32{logic_sel[2]}} ) |
157                        ( ~a_ff[31:0] & b_ff[31:0] & {32{logic_sel[1]}} );
158
168     assign sel_logic = {ap.land,ap.lor,ap.lxor};
169
185     assign out[31:0] = ({32{sel_logic}} & lout[31:0]) |
186                       ({32{sel_shift}} & sout[31:0]) |
187                       ({32{sel_adder}} & aout[31:0]) |
188                       ({32{ap.jal | pp_ff.pcall | pp_ff.pja | pp_ff.pret}} & {pcout[31:1],1'b0}) |
189                       ({32{ap.csr_write}} & ((ap.csr_imm) ? b_ff[31:0] : a_ff[31:0])) |
190                       ({31'b0, slt_one});
191

```

當and控制位為1時，選擇邏輯與運算的結果：

$\text{logic\_sel}[3]=1$  且  $\text{logic\_sel}[2]=\text{logic\_sel}[1]=0 \rightarrow \text{lout} = \text{a\_ff} \& \text{b\_ff}$

2) (以下練習基於《電腦組織結構和設計》(RISC-V版本，作者Patterson & Hennessy ([HePa])) 中的練習4.1。)

請看下面的指令：`and rd, rs1, rs2`

- a. SweRV EH1為該指令產生的控制訊號的值是多少？
- b. 哪些資源（區塊）對該指令執行有用的功能？
- c. 哪些資源（區塊）不為該指令產生輸出？哪些資源產生不使用的輸出？

不提供解答。

### 3) 在Verilator模擬中以及直接在Verilog程式碼中分析RV32I基本整數指令集中提供的*shift left/right*指令：srl、sra和sll。

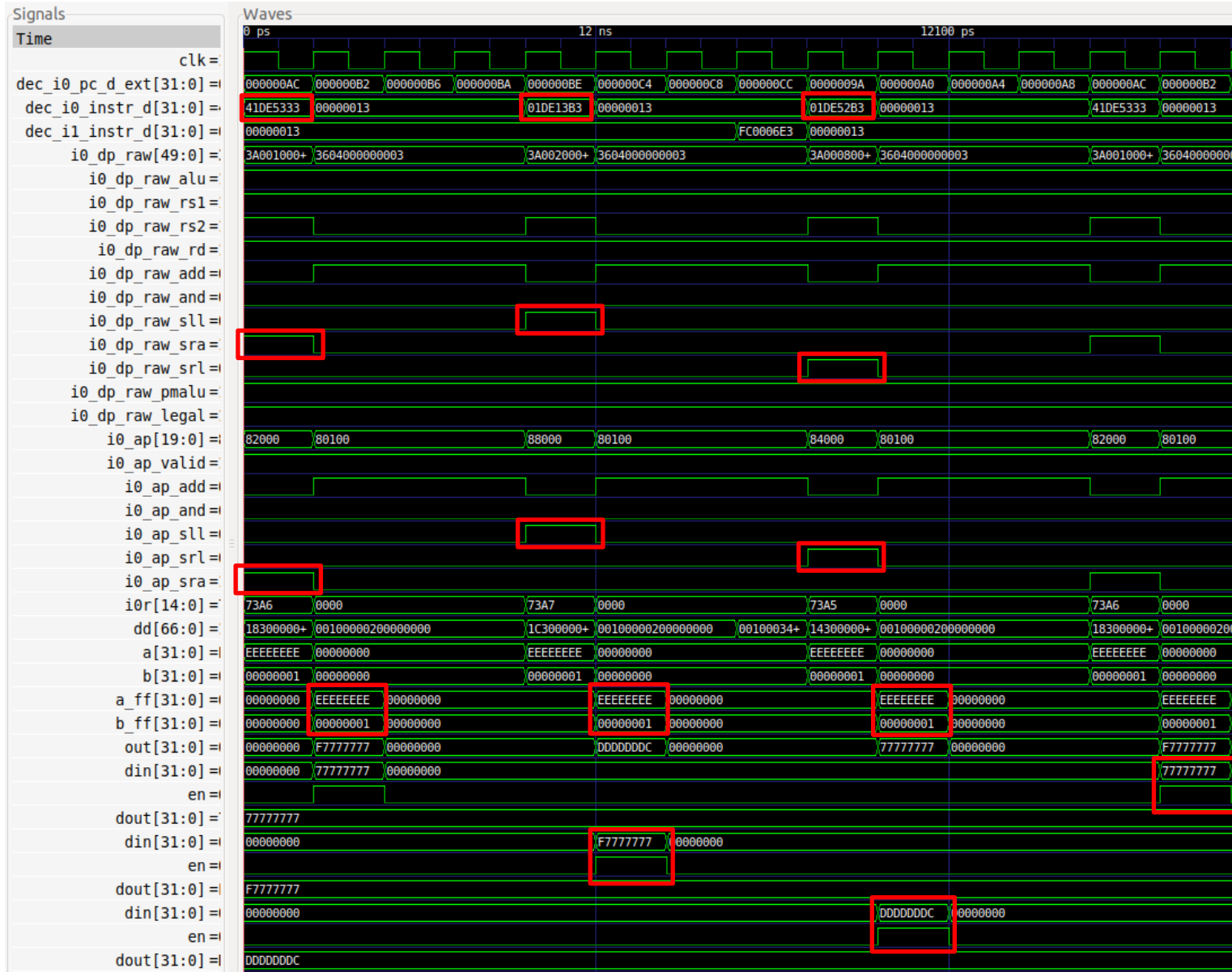
```
#define INSERT_NOPS_0
#define INSERT_NOPS_1      nop; INSERT_NOPS_0
#define INSERT_NOPS_2      nop; INSERT_NOPS_1
#define INSERT_NOPS_3      nop; INSERT_NOPS_2
#define INSERT_NOPS_4      nop; INSERT_NOPS_3
#define INSERT_NOPS_5      nop; INSERT_NOPS_4
#define INSERT_NOPS_6      nop; INSERT_NOPS_5
#define INSERT_NOPS_7      nop; INSERT_NOPS_6
#define INSERT_NOPS_8      nop; INSERT_NOPS_7
#define INSERT_NOPS_9      nop; INSERT_NOPS_8
#define INSERT_NOPS_10     nop; INSERT_NOPS_9

.globl main
main:

li t3, 0xEEEEEEEEE
li t4, 0x1

REPEAT:
    srl t0, t3, t4
    INSERT_NOPS_7
    sra t1, t3, t4
    INSERT_NOPS_7
    sll t2, t3, t4
    INSERT_NOPS_6
    beq zero, zero, REPEAT # Repeat the loop

.end
```



以下Verilog片段顯示了SweRV EH1的移位單元。

```
161      assign ashift[31:0] = a_ff >>> b_ff[4:0];
```

```
163      assign sout[31:0] = ( {32{ap.sll}} & (a_ff[31:0] << b_ff[4:0]) ) |
164      ( {32{ap.srl}} & (a_ff[31:0] >> b_ff[4:0]) ) |
165      ( {32{ap.sra}} & ashift[31:0] );
```

```
170      assign sel_shift = |{ap.sll,ap.srl,ap.sra};
```

```
185      assign out[31:0] = ({32{sel_logic}} & lout[31:0]) |
186      ({32{sel_shift}} & sout[31:0]) |
187      ({32{sel_adder}} & aout[31:0]) |
188      ({32{ap.jal | pp_ff.pcall | pp_ff.pja | pp_ff.pret}} & {pcout[31:1],1'b0}) |
189      ({32{ap.csr_write}} & ((ap.csr_imm) ? b_ff[31:0] : a_ff[31:0])) |
190      ({31'b0, slt_one});
```

4) 在Verilator模擬中以及直接在Verilog程式碼中分析RV32I基本整數指令集中提供的小於則置位指令：slt和sltu。

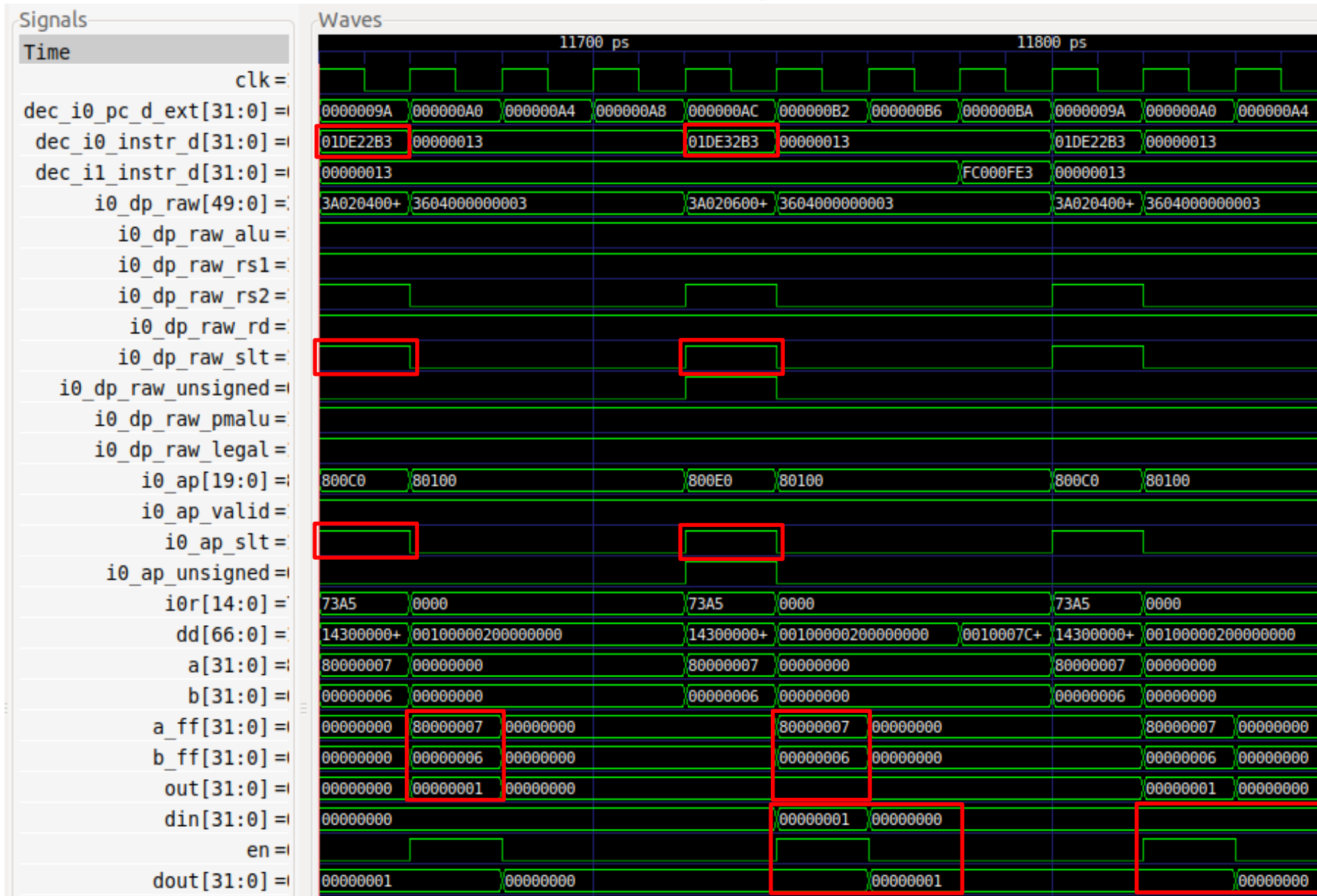
```
#define INSERT_NOPS_0
#define INSERT_NOPS_1      nop; INSERT_NOPS_0
#define INSERT_NOPS_2      nop; INSERT_NOPS_1
#define INSERT_NOPS_3      nop; INSERT_NOPS_2
#define INSERT_NOPS_4      nop; INSERT_NOPS_3
#define INSERT_NOPS_5      nop; INSERT_NOPS_4
#define INSERT_NOPS_6      nop; INSERT_NOPS_5
#define INSERT_NOPS_7      nop; INSERT_NOPS_6
#define INSERT_NOPS_8      nop; INSERT_NOPS_7
#define INSERT_NOPS_9      nop; INSERT_NOPS_8
#define INSERT_NOPS_10     nop; INSERT_NOPS_9

.globl main
main:

li t3, 0x80000007
li t4, 0x6

REPEAT:
    slt  t0, t3, t4
    INSERT_NOPS_7
    sltu t0, t3, t4
    INSERT_NOPS_6
    beq  zero, zero, REPEAT # Repeat the loop

.end
```



以下Verilog片段顯示了在SweRV EH1中執行這些運算的邏輯。

```

135   assign bm[31:0] = ( ap.sub ) ? ~b_ff[31:0] : b_ff[31:0];
136
137
138   assign {cout, aout[31:0]} = {1'b0, a_ff[31:0]} + {1'b0, bm[31:0]} + {32'b0, ap.sub};
139
140   assign ov = (~a_ff[31] & ~bm[31] & aout[31]) |
141             ( a_ff[31] & bm[31] & ~aout[31] );
142
143   assign neg = aout[31];
144

```

```

177   assign lt = (~ap.unsign & (neg ^ ov)) |
178             ( ap.unsign & ~cout);
179
180   assign ge = ~lt;
181
182
183   assign slt_one = (ap.slt & lt);
184
185   assign out[31:0] = ({32{sel_logic}} & lout[31:0]) |
186                     ({32{sel_shift}} & sout[31:0]) |
187                     ({32{sel_adder}} & aout[31:0]) |
188                     ({32{ap.jal | pp_ff.pcall | pp_ff.pja | pp_ff.pret}} & {pcout[31:1], 1'b0}) |
189                     ({32{ap.csr_write}} & ((ap.csr_imm) ? b_ff[31:0] : a_ff[31:0])) |
190                     ({31'b0, slt_one});
191

```



5) 在Verilator模擬中以及直接在Verilog程式碼中分析RV32I基本整數指令集中提供的*immediate*指令：addi、andi、ori、xori、srli、srai、slli、slti和sltui。

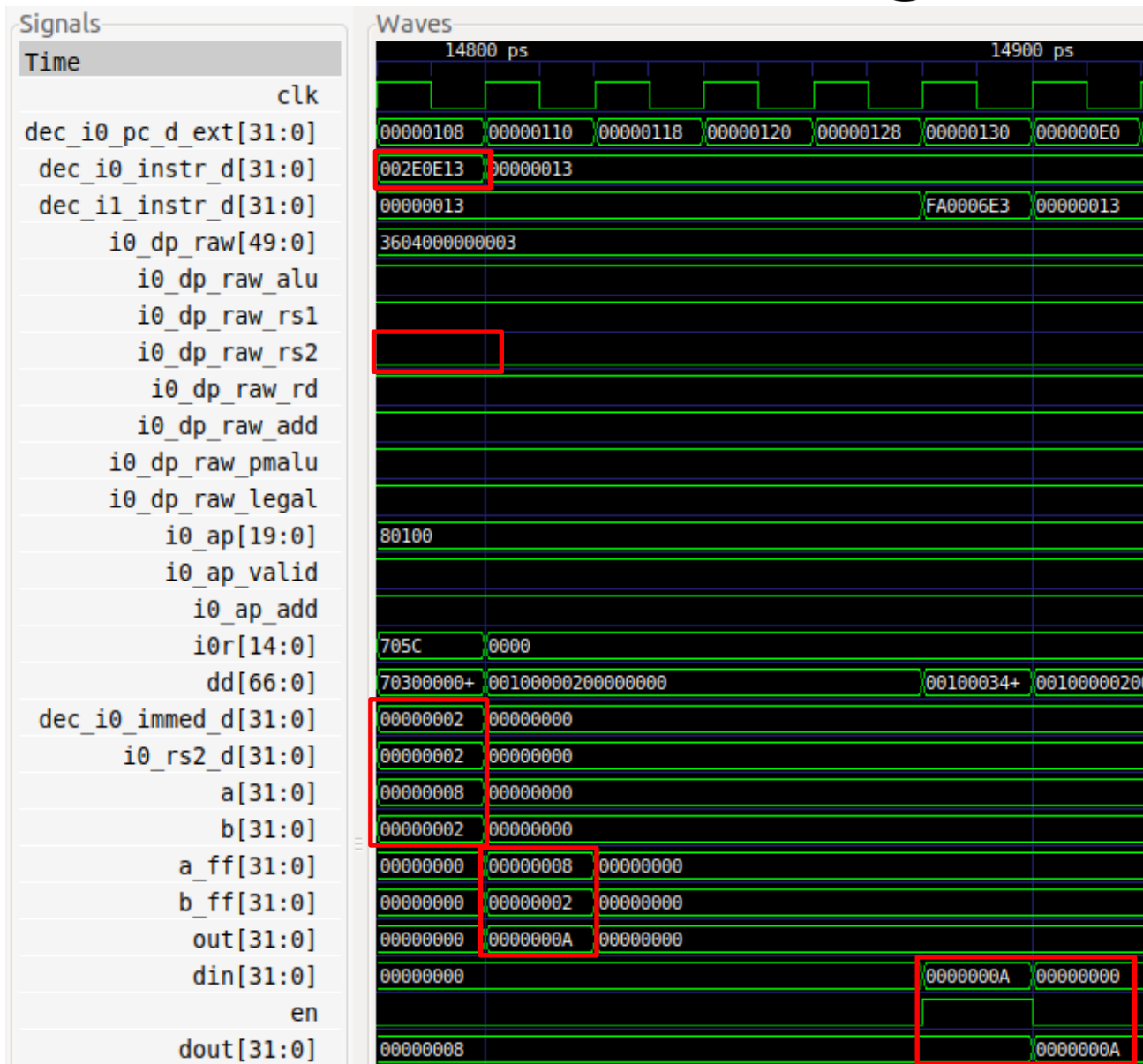
```
#define INSERT_NOPS_0
#define INSERT_NOPS_1      nop; INSERT_NOPS_0
#define INSERT_NOPS_2      nop; INSERT_NOPS_1
#define INSERT_NOPS_3      nop; INSERT_NOPS_2
#define INSERT_NOPS_4      nop; INSERT_NOPS_3
#define INSERT_NOPS_5      nop; INSERT_NOPS_4
#define INSERT_NOPS_6      nop; INSERT_NOPS_5
#define INSERT_NOPS_7      nop; INSERT_NOPS_6
#define INSERT_NOPS_8      nop; INSERT_NOPS_7
#define INSERT_NOPS_9      nop; INSERT_NOPS_8
#define INSERT_NOPS_10     nop; INSERT_NOPS_9

.globl main
main:

li t3, 0x4                # t3 = 4
INSERT_NOPS_1

REPEAT:
    INSERT_NOPS_10
    addi t3, t3, 2          # t3 = t3 + 4
    INSERT_NOPS_10
    beq zero, zero, REPEAT # Repeat the loop

.end
```



在模組**dec\_decode\_ctl**中，計算32位立即數。

```

1231 // read the csr value through rs2 immid port
1232 assign dec_i0_immed_d[31:0] = ({32{ i0_dp.csr_read}} & dec_csr_rddata_d[31:0]) |
1233                               ({32{~i0_dp.csr_read}} & i0_immed_d[31:0]);
1234
1235 // end csr stuff
1236
1237 assign i0_immed_d[31:0] = ({32{i0_dp.imm12}} & { {20{i0[31]}},i0[31:20] }) | // jalr
1238                               ({32{i0_dp.shimm5}} & {27'b0, i0[24:20]}) |
1239                               ({32{i0_jalimm20}} & { {12{i0[31]}},i0[19:12],i0[20],i0[30:21],1'b0}) |
1240                               ({32{i0_uiimm20}} & {i0[31:12],12'b0}) |
1241                               ({32{i0_csr_write_only_d & i0_dp.csr_imm}} & {27'b0,i0[19:15]}); // for csr's that only write csr, dont read csr
1242

```

在模組**exu**中，選擇正確的**rs2**源。本例中我們使用**dec\_i0\_immed\_d**。

```

286 assign i0_rs2_d[31:0] = ({32{~dec_i0_rs2_bypass_en_d}} & gpr_i0_rs2_d[31:0]) |
287                               ({32{~dec_i0_rs2_bypass_en_d}} & dec_i0_immed_d[31:0]) |
288                               ({32{ dec_i0_rs2_bypass_en_d}} & i0_rs2_bypass_data_d[31:0]);

```

在模組**dec\_gpr\_ctl**中，使能訊號**rden1**確定是否訪問暫存器檔案來獲取第二個操作數。如果指令使用立即數操作數：**i0\_dp.rs2=0 → rden1=0 → rd1[31:0]=0x00000000 → gpr\_i0\_rs2\_d[31:0]=0x00000000**。

```

90 rd1[31:0] |= ({32{rden1 & (raddr1[4:0]== 5'(j)) & (gpr_bank_id[GPR_BANKS_LOG2-1:0] == 1'(i))}} & gpr_out[i][j][31:0]);

```

## 6) (以下練習基於[HePa]的練習4.4以及S. Harris和D. Harris所編教材《數字設計和電腦體系結構：RISC-V版本》[DDCARV]第7章的練習1。)

製造矽晶片時，材料（如矽）中的缺陷和製造錯誤會導致電路有缺陷。一個非常常見的缺陷是一根訊號線「損壞」，邏輯始終為0。這通常稱為「**stuck-at-0**」（固定為0）故障。確定訊號**i0\_ap**（**alu\_pkt\_t**類型）中包含的每個控制位元傳送「固定為0」故障的影響。

結構類型在檔案**swerv\_types.sv**中定義：

```
typedef struct packed {
    logic valid;
    logic land;
    logic lor;
    logic lxor;
    logic sll;
    logic srl;
    logic sra;
    logic beq;
    logic bne;
    logic blt;
    logic bge;
    logic add;
    logic sub;
    logic slt;
    logic unsign;
    logic jal;
    logic predict_t;
    logic predict_nt;
    logic csr_write;
    logic csr_imm;
} alu_pkt_t;
```

- 訊號valid固定為0：無法執行任何A-L指令，因為任何A-L指令都將被視為無效。
- 訊號land、lor、lxor、sll、srl、sra、beq、bne、blt、bge、add、sub、slt和jal固定為0：對於上述每一位元，都無法執行相應的A-L指令；例如，如果land固定為0，將無法執行and指令。
- 訊號unsign固定為0：無法向處理器傳達運算必須為無符號運算的訊息。
- 訊號predict\_t和predict\_nt：無法向處理器傳達預測採用或不採用分支的訊息。
- 訊號csr\_write和csr\_imm：無法在CSR暫存器中寫入或使用立即數進行運算。

7) (以下練習基於[HePa]的練習4.6。)  
 圖5不討論I型指令，如addi或andi。

- 需要哪些額外的邏輯區塊（如果有）來支援SweRV EH1中I型指令的執行？將所有必要的邏輯區塊新增到圖5並說明其用途。
- 列出addi的控制單元產生的訊號的值。

解碼階段兩個3-1多工器的輸入之一來自訊號dec\_i0\_immed\_d[31:0]中的立即數。立即數是一個32位元訊號，根據執行的I型指令進行不同的計算。它是組成指令的一個子集（32位元），相應位元的選擇和符號延伸過程如下：

```

1231 // read the csr value through rs2 immed port
1232 assign dec_i0_immed_d[31:0] = ({32{i0_dp.csr_read}} & dec_csr_rddata_d[31:0]) |
1233                               ({32{~i0_dp.csr_read}} & i0_immed_d[31:0]);
1234
1235 // end csr stuff
1236
1237 assign i0_immed_d[31:0] = ({32{i0_dp.imm12}} & {20{i0[31]},i0[31:20]}) | // jalr
1238                               ({32{i0_dp.shimm5}} & {27'b0, i0[24:20]}) |
1239                               ({32{i0_jalimm20}} & {12{i0[31]},i0[19:12],i0[20],i0[30:21],1'b0}) |
1240                               ({32{i0_uimm20}} & {i0[31:12],12'b0}) |
1241                               ({32{i0_csr_write_only_d & i0_dp.csr_imm}} & {27'b0,i0[19:15]}); // for csr's that only write csr, dont read csr
1242

```

addi的控制訊號的值位於練習5的模擬中。