



IMAGINATION大學計劃

RVfpga實驗17

超標量執行

在第2部分，我們將分析兩個簡單的程式，比較使用SweRV EH1單指令和雙指令組態時的行為。然後，在第3部分，我們會提供幾個與超標量執行相關的練習。

2. 單指令與雙指令

在本部分中，我們使用兩個簡單的程式：第一個程式（第2.A部分）包含一個由四條AL指令組成的迴圈，第二個程式（第2.B節）包含一個由兩條乘法指令和兩條AL指令交錯而成的迴圈。

A. 四條獨立的AL指令

在本部分中，我們將對比圖2中的程式在單指令和雙指令SweRV EH1核心上執行時的效能。回想一下，在實驗11的附錄B中，我們介紹了如何配置不同的核心功能（管線執行、分支預測、超標量等）。

本程式包含一個執行1,000,000（0xF4240）次迭代的迴圈；迴圈主體包含四條獨立的AL指令（add、sub、or和xor），並由nop指令包住，以便使用者能夠單獨查看每次迭代。資料夾[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions中提供PlatformIO專案，可根據需要對其進行分析、模擬和修改。

```
.globl Test_Assembly

.text

Test_Assembly:

li t2, 0x400          # Disable Dual-Issue Execution
csrrs t1, 0x7F9, t2

li t0, 0x0
li t1, 0x1
li t2, 0x1
li t3, 0x3
li t4, 0x4
li t5, 0x5
li t6, 0x6

lui t2, 0xF4
add t2, t2, 0x240

REPEAT:
    add t0, t0, 1
    INSERT_NOPS_10
    INSERT_NOPS_4
    add t3, t3, t1
    sub t4, t4, t1
    or  t5, t5, t1
    xor t6, t6, t1
    INSERT_NOPS_10
    INSERT_NOPS_3
    bne t0, t2, REPEAT # Repeat the loop

.end
```

圖2. 包含四條AL指令的程式

在第2.A.i部分，我們將根據Verilator中的模擬結果以及Nexys A7開發板上的執行結果，分析圖2中的程式在單指令SweRV EH1處理器中的執行情況。為此，我們在程式開始時使用以下兩條指令來停用雙指令功能：

```
li t2, 0x400
csrrs t1, 0x7F9, t2
```

在第2.A.ii部分，我們將根據Verilator中的模擬結果以及Nexys A7開發板上的執行結果，分析圖2中的程式在雙指令SweRV EH1處理器中的執行情況。為此，只需註解掉先前使用的兩條指令。

i. 單指令SweRV EH1處理器中的執行情況

在單指令組態中，SweRV EH1會直接忽略第二條通路，在每個週期只執行一條指令。圖3所示為圖2中的程式在此種SweRV EH1組態下的模擬結果。我們隨機選取了REPEAT迴圈中的一次迭代（非第一次迭代，假定每次迭代均相同）。

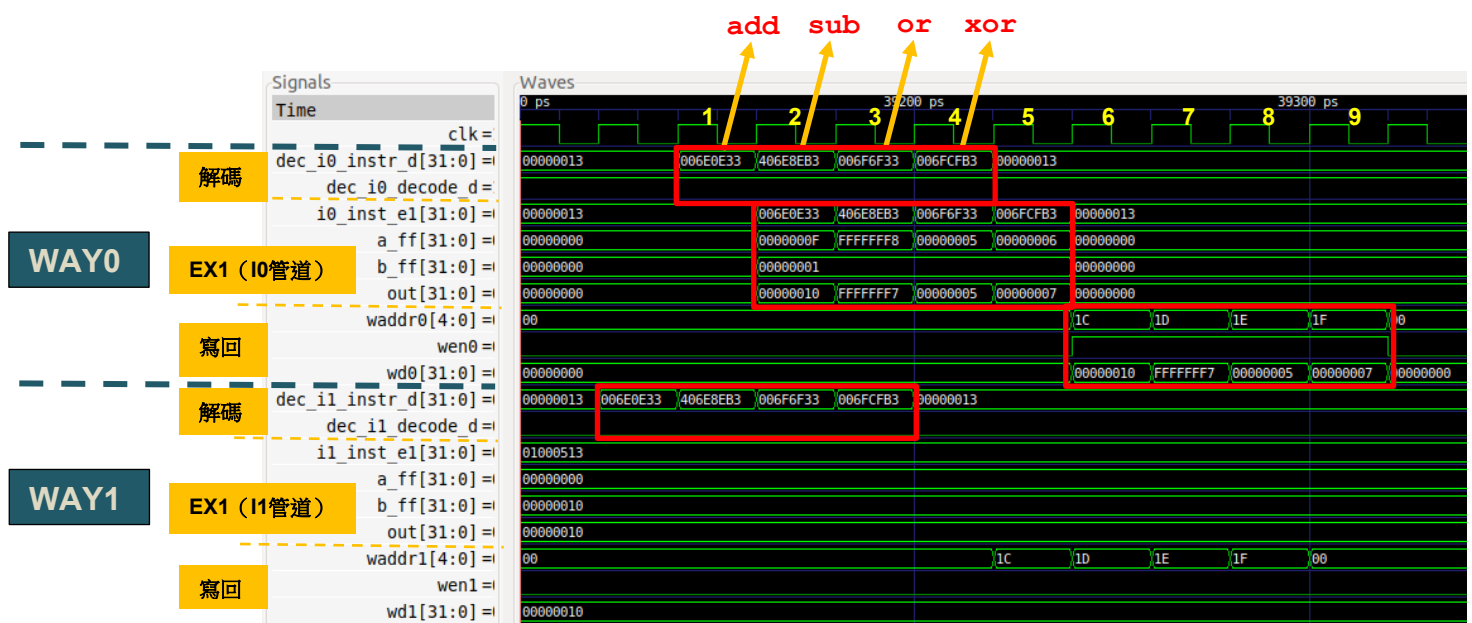


圖3. 圖2中的程式在單指令SweRV EH1處理器上的模擬結果

解碼時，兩條通路都會接收指令（參見訊號dec_i0_instr_d[31:0]和dec_i1_instr_d[31:0]），但僅通路0會將指令傳送至執行階段，因為通路1的傳送功能已停用。訊號dec_i0_decode_d和dec_i1_decode_d用於確定指令是否會從解碼階段傳播到EX1階段，訊號i0_inst_e1[31:0]和i1_inst_e1[31:0]分別包含E1階段的通路0和通路1中的指令。

- 通路0：
 - o 在我們的範例中，訊號dec_i0_decode_d始終為1；具體地說，對於我們所分析的四條AL指令，其值均為1。

- 解碼階段的指令 (`dec_i0_instr_d[31:0]`) 會傳播到I0管道 (`i0_inst_e1[31:0]`)
- 通路1：
 - 在我們的範例中，訊號`dec_i1_decode_d`始終為0；具體地說，對於我們所分析的四條AL指令，其值均為0。
 - 解碼階段的指令 (`dec_i1_instr_d[31:0]`) 不會傳播到 (`i1_inst_e1[31:0]`) 執行階段。

因此，系統僅使用來自I0管道的ALU（參見兩條通路中的訊號`aff`、`bff`和`out`），並且僅使用暫存器檔案的寫入連接埠0（參見兩條通路中的訊號`waddr`、`wen`和`wd`）。

圖4所示為I0管道中從解碼階段到寫回（EX5）階段四條AL指令流圖中展示了圖3中指定的9個週期（1至9）。留空的方格對應於四條AL指令週圍的nop指令，為簡單起見，圖中未顯示這些指令。

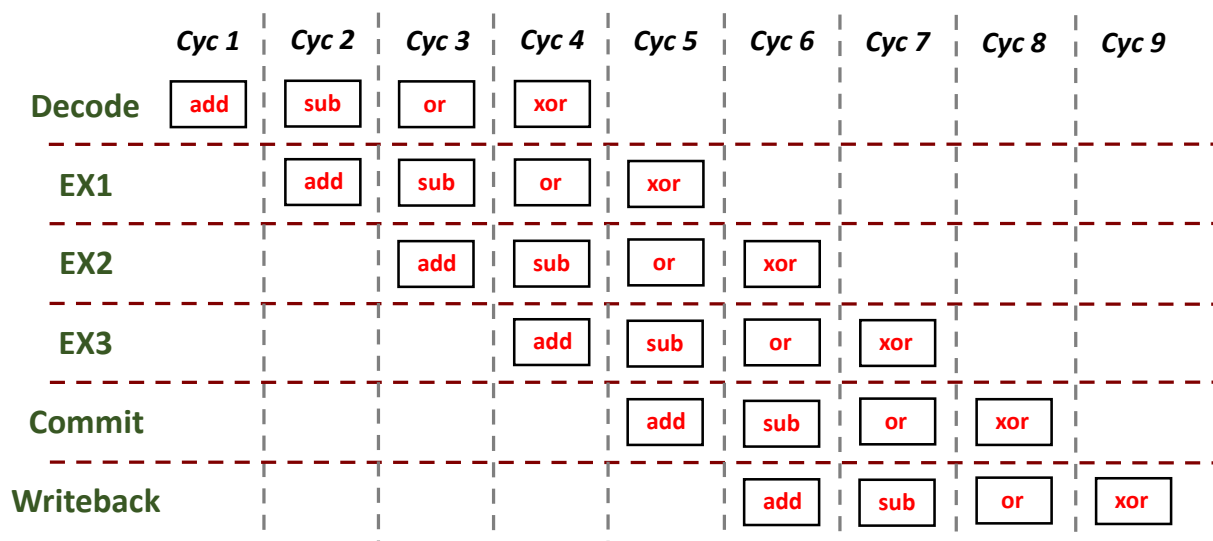


圖4. I0管道中的4條AL指令流

任務：在圖3的模擬中加入追蹤訊號，並仿照圖4的形式強調顯示管線中從解碼階段到寫回階段的指令流。可以使用以下位置提供的.tcl檔案：

`[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions/test_task1.tcl`。

任務：刪除圖2所示的迴圈主體中的所有nop指令。

重複圖3中的模擬。該程式的預期IPC是多少？

在開發板上執行程式，驗證獲得的IPC是否符合您的預期。

ii. 雙路超標量SweRV EH1中的執行情況

註解掉圖2中的兩條指令，以啟用SweRV EH1的雙指令功能。現在，只要符合條件，SweRV EH1將在每個週期向執行階段傳送兩條指令。圖5所示為該程式的模擬結果。與之前一樣，我們隨機選取了 *REPEAT* 迴圈中的一次迭代（非第一次迭代，假定每次迭代均相同）。

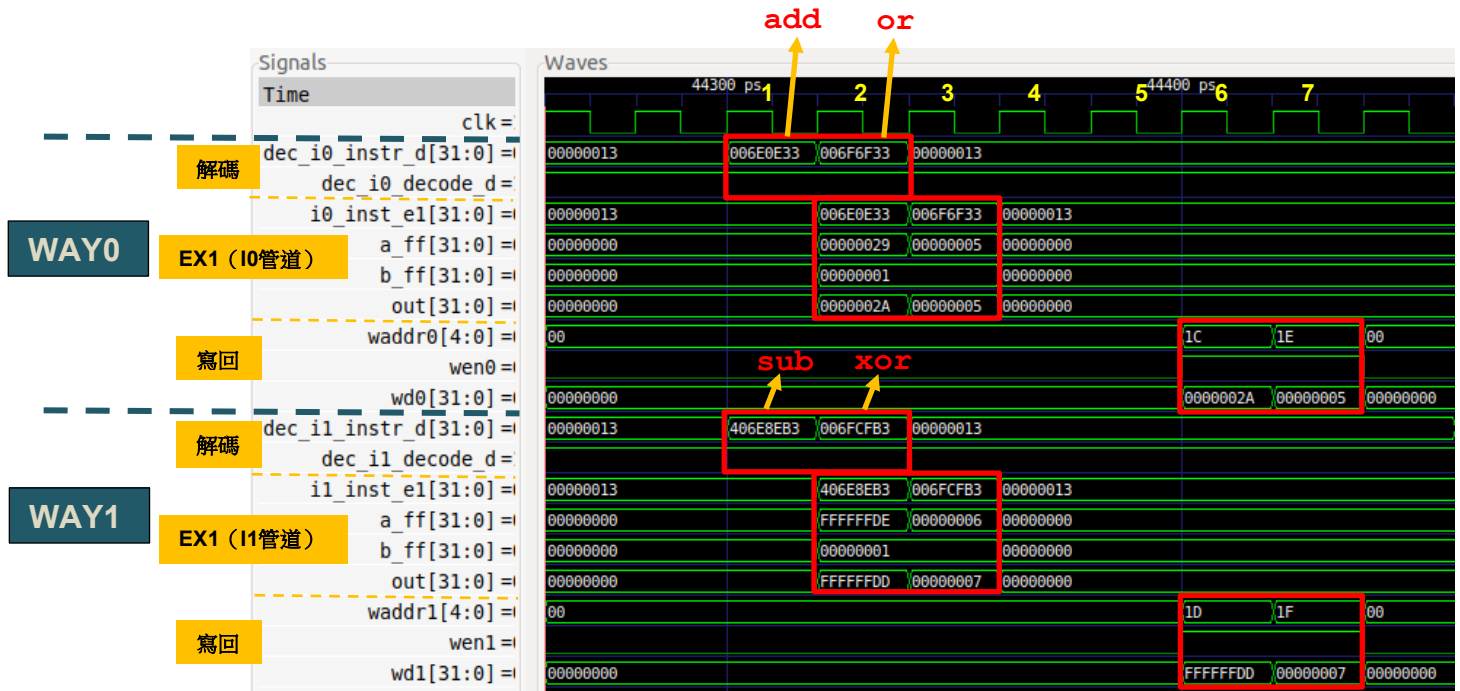


圖5. 圖2中的程式在雙指令SweRV EH1處理器上的模擬結果

解碼時，每條通路各接收一條指令（參見訊號dec_i0_instr_d[31:0]和dec_i1_instr_d[31:0]），並且在每個週期，I0管道和I1管道各會向執行階段傳送一條指令。

- 通路0：
 - 對於範例中的四條AL指令中的兩條，訊號dec_i0_decode_d始終為1（另外兩條在通路1中進行解碼）。
 - 解碼階段的指令（dec_i0_instr_d[31:0]）會傳播到I0管道（i0_inst_e1[31:0]）。
- 通路1：
 - 對於範例中的四條AL指令中的兩條，訊號dec_i1_decode_d始終為1（另外兩條在通路0中進行解碼）。
 - 解碼階段的指令（dec_i1_instr_d[31:0]）會傳播到I1管道（i1_inst_e1[31:0]）。

因此，系統會使用兩條管道（I0和I1）中的ALU（參見兩條通路中的訊號a_ff、b_ff和out），並且會使用兩個暫存器檔案寫入連接埠（參見兩條通路中的訊號waddr、wen和wd）。

圖6所示為I0和I1管道中從解碼階段到EX5階段四條AL指令流。圖中展示了圖5中指定的7個週期（1至7）。留空的方格對應於四條AL指令週圍的nop指令，為簡單起見，圖中未顯示這些指令。

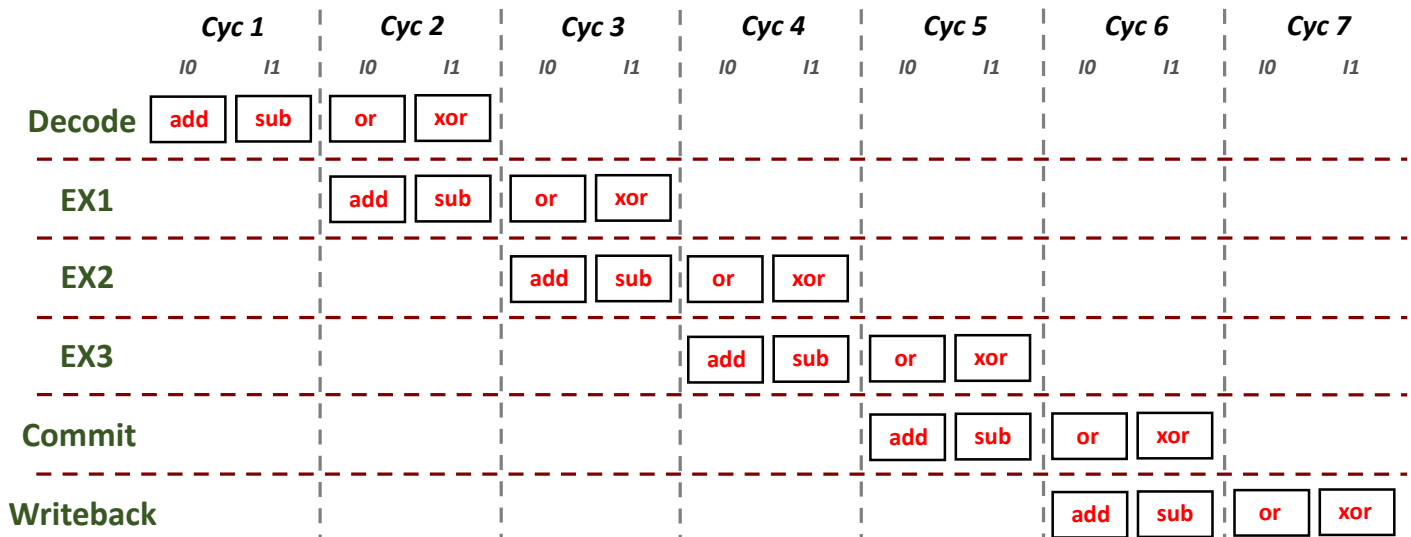


圖6. 兩條管道（I0和I1）中的4條指令流

任務：在圖5的模擬中加入追蹤訊號，並仿照圖6的形式強調顯示管線中從解碼階段到寫回階段的指令流。可以使用以下位置提供的.tcl檔案：

[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions/test_task2.tcl。

任務：刪除圖2所示的迴圈主體中的所有nop指令。

重複圖5中的模擬。該程式的預期IPC是多少？

在開發板上執行程式，驗證獲得的IPC是否符合您的預期。

B. 互相交錯的兩條mul指令和兩條AL指令

在本部分中，我們將分析圖7中的程式在雙指令SweRV EH1核心上時的執行情況。本程式包含一個執行1,000,000次迭代的迴圈；迴圈主體包含互相交錯的兩條mul指令和兩條AL指令（mul、add、mul和sub），並由nop指令包圍，以便使用者能夠單獨查看每次迭代。這四條指令互相獨立。我們將在Verilator中進行模擬分析，強調顯示主要的訊號，並對每種情況下的程式行為進行簡要解釋。資料夾[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL_TwoMUL_Instructions中提供PlatformIO專案，可根據需要對其進行分析、模擬和修改。

```
.globl Test_Assembly

.text

Test_Assembly:
```



```
# li t2, 0x400          # Disable Dual-Issue Execution
# csrrs t1, 0x7F9, t2

li t3, 0x3
li t4, 0x4
li t5, 0x5
li t6, 0x6
li t0, 0x0
lui t1, 0xF4
add t1, t1, 0x240

REPEAT:
    add t0, t0, 1
    INSERT_NOPS_10
    INSERT_NOPS_4
    mul t3, t3, t1
    add t4, t4, t1
    mul t5, t5, t1
    sub t6, t6, t1
    INSERT_NOPS_10
    INSERT_NOPS_3
    bne t0, t1, REPEAT # Repeat the loop

.end
```

圖7. 包含2條mul指令和2條AL指令的程式

在此程式中，處理器每個週期將向執行階段傳送一條mul指令和一條AL指令，因此會使用乘法管道以及I0（或I1）管道。圖8所示為圖7中的程式在雙路超標量SweRV EH1處理器上的模擬結果。我們隨機選取了REPEAT迴圈中的一次迭代（非第一次迭代，假定每次迭代均相同）。

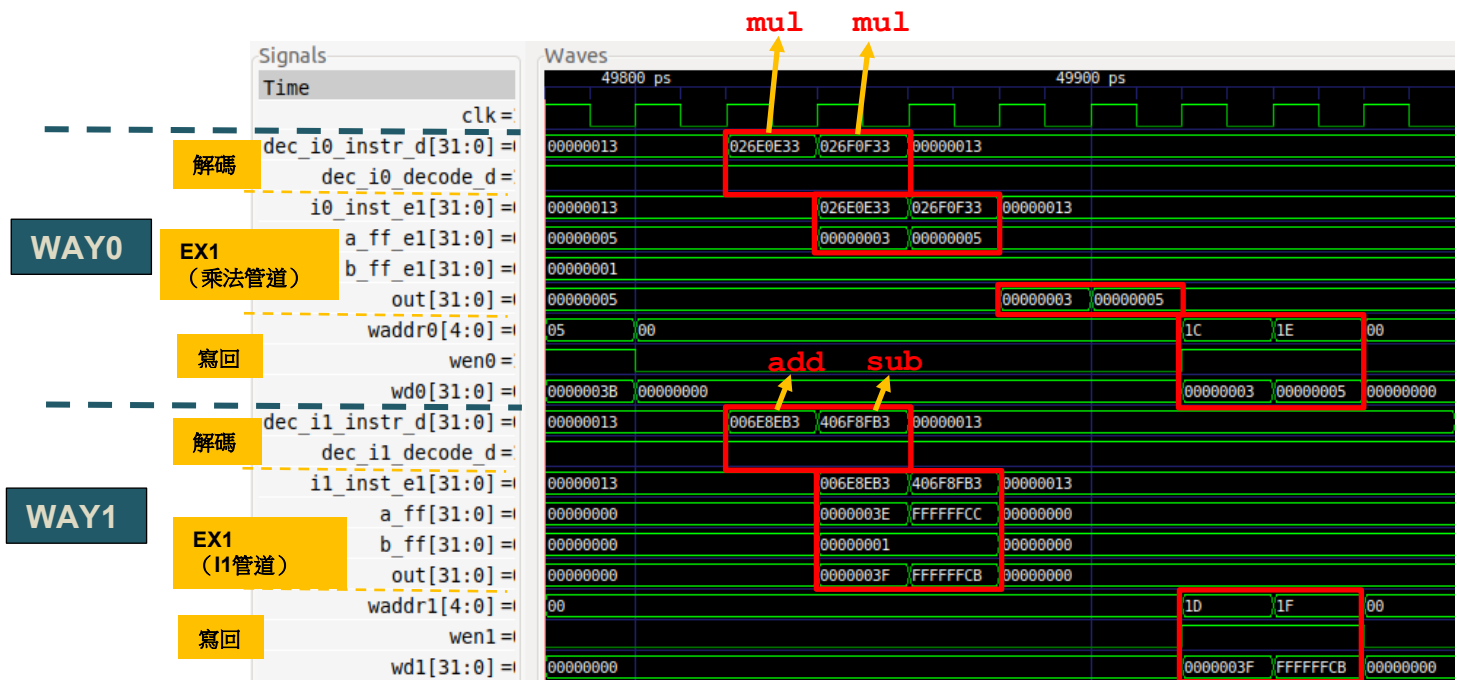


圖8. 圖7中程式的模擬結果

解碼時，兩條通路都會接收指令（參見訊號`dec_i0_instr_d[31:0]`和`dec_i1_instr_d[31:0]`）並將指令傳送至執行階段。

- **通路0：**
 - 對於範例中所分析的四條指令中的兩條，訊號`dec_i0_decode_d`始終為1（另外兩條在通路1中進行解碼）。
 - 解碼階段的指令（`dec_i0_instr_d[31:0]`）會傳播到乘法管道（`i0_inst_e1[31:0]`）
- **通路1：**
 - 對於範例中所分析的四條指令中的兩條，訊號`dec_i1_decode_d`始終為1（另外兩條在通路1中進行解碼）。
 - 解碼階段的指令（`dec_i1_instr_d[31:0]`）會傳播到I1管道（`i1_inst_e1[31:0]`）

因此，系統會使用I1管道和乘法器中的ALU（參見訊號`a_ff_e1`、`b_ff_e1`和`out`，以及訊號`a_ff`、`b_ff`和`out`），並且會使用兩個暫存器檔案寫入連接埠（參見兩條通路中的訊號`waddr`、`wen`和`wd`）。

任務：在圖8的模擬中加入追蹤訊號，並強調顯示管線中從解碼階段到寫回階段的指令流。可以使用以下位置提供的.tcl檔案：
`[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL_TwoMUL_Instructions/test_taskMuls.tcl`。

任務：刪除圖7所示的迴圈主體中的所有nop指令。
 重複圖8中的模擬。該程式的預期IPC是多少？
 在開發板上執行程式，驗證獲得的IPC是否符合您的預期。
 使用單指令組態重複實驗，並比較實驗結果。

3. 練習

- 1) 使用能夠展示雙指令執行相關新情境的指令組合，建立與圖2和圖7所示程式類似的程式。
- 2) 分析（雙指令）SweRV EH1處理器與S.Harris和D.Harris所著教材《數位設計和電腦體系結構》（RISC-V版本，簡稱[DDCARV]）第7.7.4節中所示的超標量處理器（為方便起見，該處理器已在圖1中提供）之間的差異。
- 3) 分析DDCARV第7.7.4節的圖7.70中的程式，該程式位於資料夾
`[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_SuperscalarExample`下的PlatformIO專案中。使用模擬器和開發板兩種方式，在SweRV EH1上執行該程式（對於後一種方式，請刪除nop指令）。解釋得到的結果。如有必要，可調整程式以獲得最理想的IPC。

接下來，根據本實驗及SweRVref.docx（第2部分）中所述，停用雙指令執行。將模擬結果和開發板上的執行結果與啟用雙指令功能時的相應結果進行比較。

- 4) 修改練習3中的程式，將指令add s9, s8, t1替換為add t2, s8, t1。解釋得到的結果。如有必要，可調整程式以獲得最理想的IPC。

然後，根據本實驗及SweRVref.docx（第2節）中所述，停用雙指令執行。將模擬結果和開發板上的執行結果與啟用雙指令功能時的相應結果進行比較。

- 5) （以下練習基於《電腦組織結構和設計》（RISC-V版本，Patterson & Hennessy ([HePa])）中的練習4.31。）

在本練習中，我們將比較單指令處理器和雙指令處理器的效能，並考慮可最佳化雙指令執行的程式修改。本練習中的問題涉及以下迴圈（用C語言編寫）：

```
for(i=0;i!=j;i+=2) b[i]=a[i]-a[i+1];
```

僅經過少量最佳化或未經最佳化的編譯器可能會產生以下RISC-V組合語言程式碼：

```
li x12, 0
li x13, 8000
li x14, 0
TOP:
    slli x5, x12, 2
    add x6, x10, x5
    lw x7, 0(x6)
    lw x29, 4(x6)
    sub x30, x7, x29
    add x31, x11, x5
    sw x30, 0(x31)
    addi x12, x12, 2
    ENT: bne x12, x13, TOP
```

該程式碼使用以下暫存器：

i	j	a	b	Temporary values
x12	x13	x10	x11	x5-x7, x29-x31

該程式碼位於[RVfpgaPath]/RVfpga/Labs/Lab17/PaHe_SuperscalarExample路徑下，與本書練習中提供的程式碼相比，存在一些小的修改，但不會影響程式的行為：

- 將暫存器x13初始化為8000，以使程式執行4000次迭代。
- 刪除了jal指令。
- 將ld和sd指令替換為lw和sw指令。這表示存取從4個位元組寬變為8個位元組寬。

假設有一個具備以下屬性的雙指令靜態調度處理器：

1. 一條指令必須是記憶體存取操作；另一條必須是算術/邏輯指令或分支指令。
2. 處理器的各階段之間支援所有可能的轉送路徑。
3. 處理器可實現完美分支預測。
4. 如果兩條指令相關，則可能無法同時發出。
5. 如果某一階段需要暫停指令，則必須同時暫停兩條指令。

- a) 比較該範例處理器與SweRV EH1處理器的屬性。
- b) 繪製管線圖和模擬圖，顯示上述RISC-V程式碼的任意一次迴圈迭代（第一次迭代除外）在雙指令SweRV EH1處理器上的執行情況。假設在進行4000次迭代後退出迴圈（上述程式碼即為如此）。
- c) 從單指令SweRV EH1處理器變為雙指令SweRV EH1處理器時，加速比為多少？解釋得到的結果。在開發板上測試程式並啟用/停用雙指令執行。
- d) 重新排列/重寫上述RISC-V程式碼，以在雙指令SweRV EH1處理器上實現更高的效能。（但請勿展開迴圈。）
- e) 接下來展開RISC-V程式碼，使展開後迴圈的每次迭代等於原始迴圈的兩次迭代。然後，重新排列/重寫展開的程式碼，以在雙指令SweRV EH1處理器上實現更高的效能。

6) （以下練習基於DDCARV第7章的練習7.30、7.32和7.34。）

假設SweRV EH1處理器正在執行以下程式碼片段。回想一下，SweRV EH1具有冒險單元。可以假設一個能夠在一個週期內傳回結果的記憶體系統（為此，我們使用DCCM，將程式碼片段插入一個迴圈中，並避免使用第一次迭代，以免發生IS未命中）。

```
addi s1, t0, 11    # t0 contains the base address of the DCCM
lw    s2, 25(s1)
lw    s5, 16(s2)
add   s3, s2, s5
or    s4, s3, t4
and   s2, s3, s4
```

- a) 使用Verilator和GTKWave模擬該程式。分析結果，並針對每個週期指明以下內容：
 - * 對哪些指令進行解碼？將哪些指令傳送至執行階段？提交哪些指令？
 - * 對哪些暫存器進行寫入操作和讀取操作？
 - * 發生哪些轉送和暫停情況？
- b) 對於該程式，處理器的CPI是多少？先做出理論解答，然後在開發板上執行程式，確認您的答案。
- c) 在單指令處理器上執行相同的分析，將結果與使用雙指令處理器時的結果進行比較。

PlatformIO專案的路徑如下：**[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_Exercises-30-32-34**。所分析的程式將插入一個迴圈中，以使模擬更易理解（可使用除第一次迭代外的任何迭代進行分析）並且可以使用效能計數器進行測定。

7) (以下練習基於DDCARV第7章的練習7.31、7.33和7.35。)

使用以下程式碼片段重複練習7。

```
addi s1, t0, 52
addi s0, s1, -4
lw    s3, 16(s0)
sw    s3, 20(s0)
xor   s2, s0, s3
or    s2, s2, s3
```

PlatformIO專案的路徑如下：`[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_Exercises-31-33-35`。