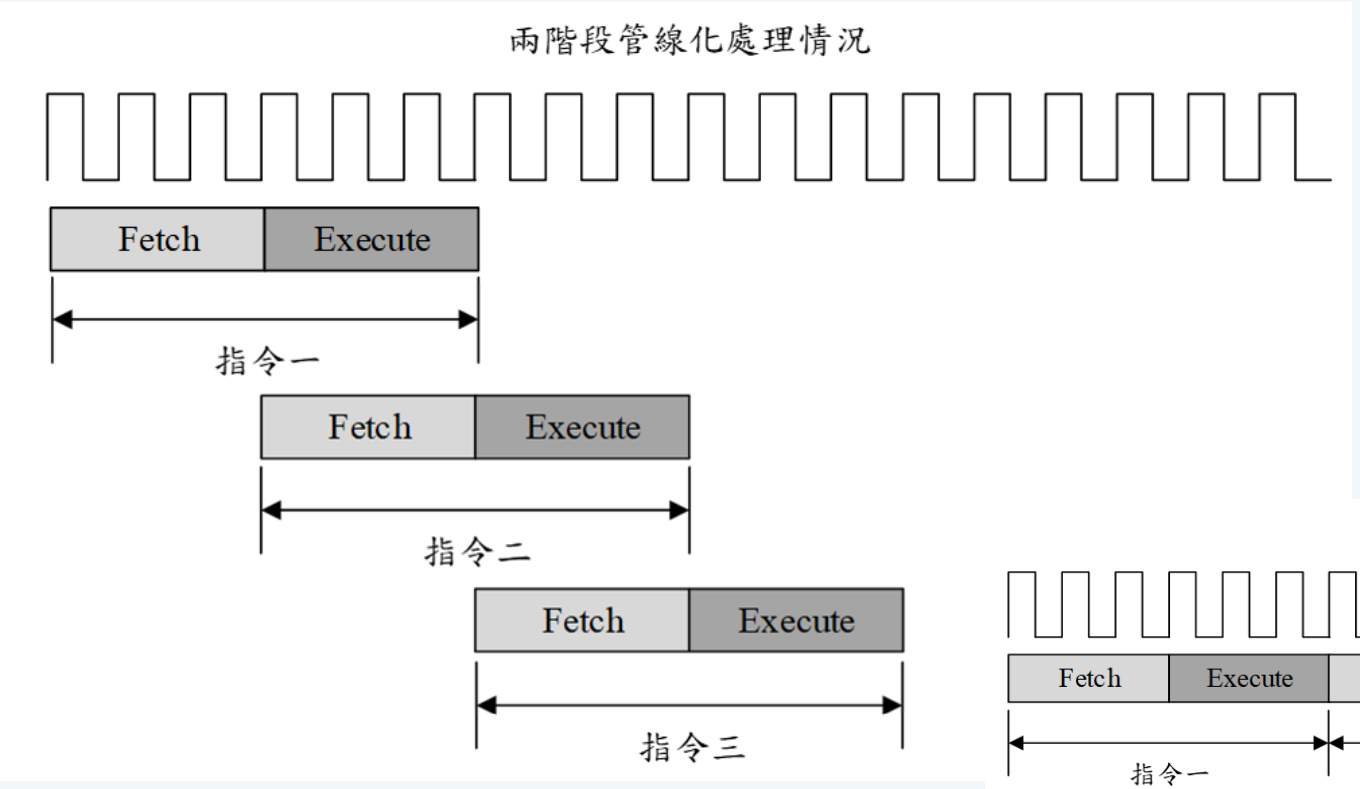
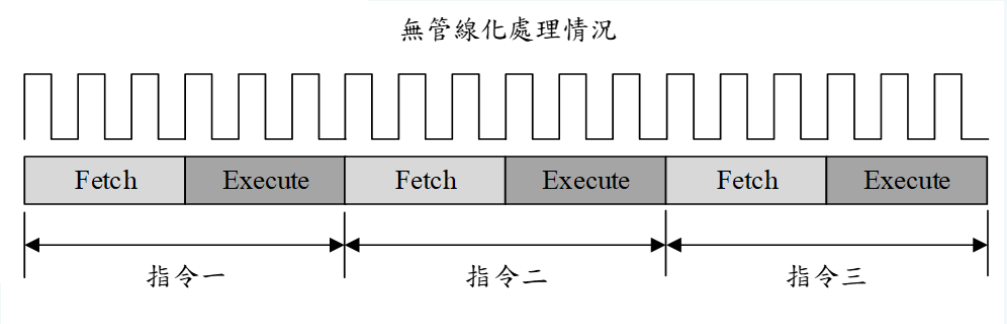
PIPELINE
管線化

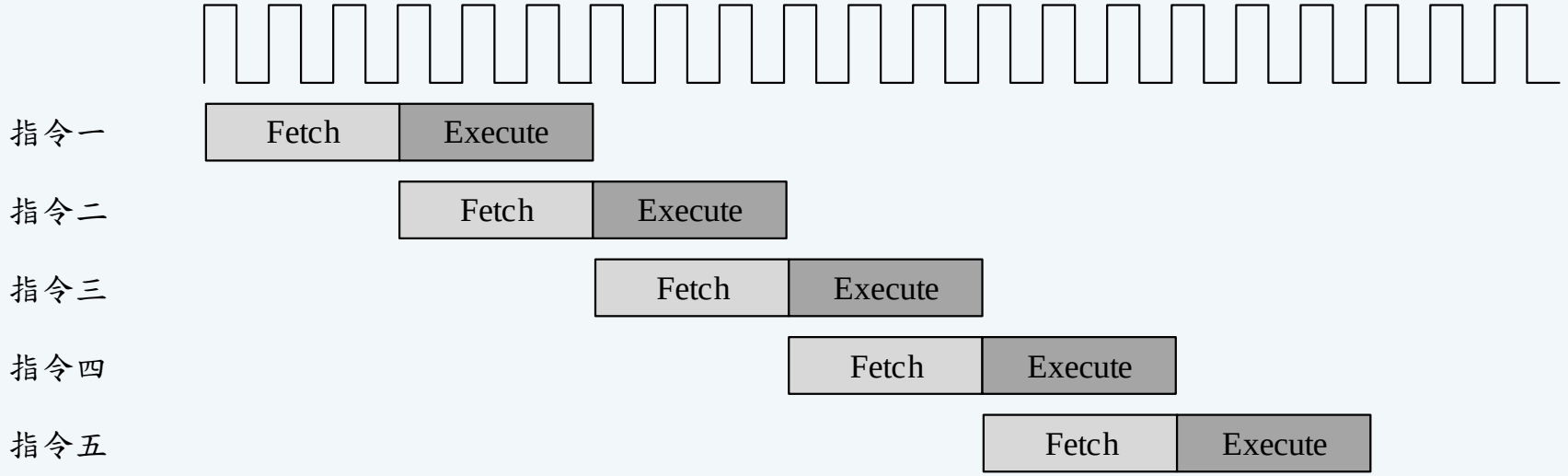
National Taiwan Ocean University
Department of Computer Science & Information Engineering



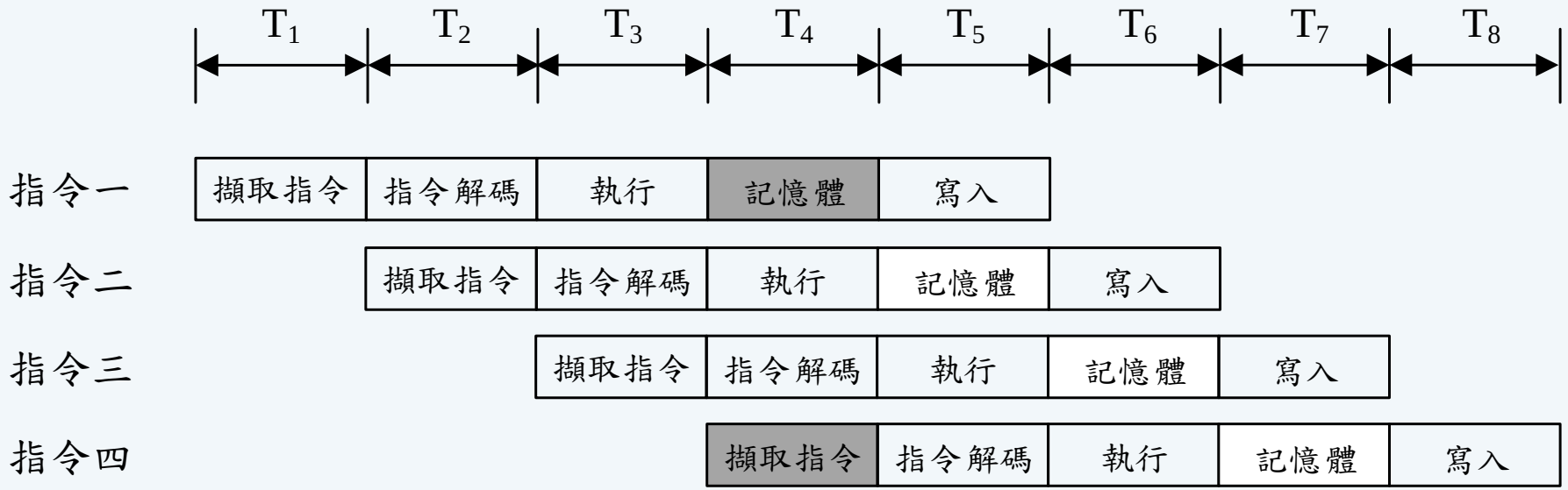


PIC16LF1826 採用程式匯流排與資料匯流排分離的哈佛結構
使得擷取階段及執行階段可以重疊進行運作
在前一個指令執行階段時，便可以進行下一個指令的擷取階段，
這種方法稱為指令管線，不需要等待前一個指令完整執行完，就可以處理下一個指令，減少硬體閒置的狀態，可以大幅提升微控制器的執行效率

PIC16LF1826使用兩階段管線化，將一個指令分成兩個部分，
分別為擷取階段(Fetch Cycle)及執行階段(Execution Cycle)，
執行程式時的管線化運作範例如下圖所示。

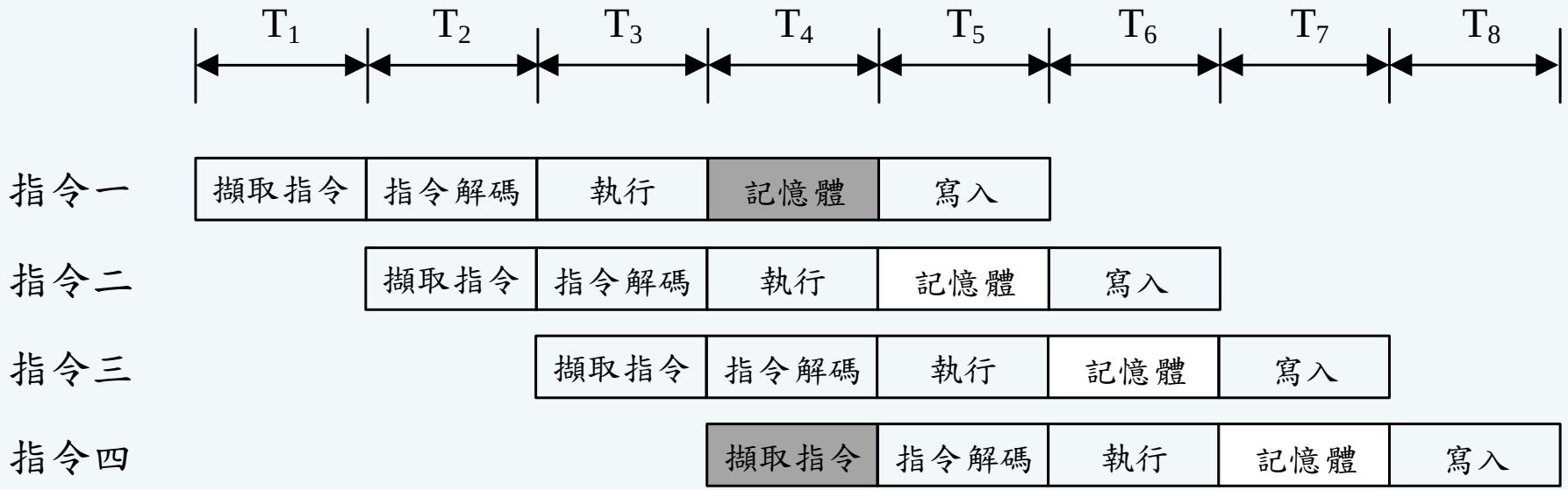


結構危害（Structure Hazard）



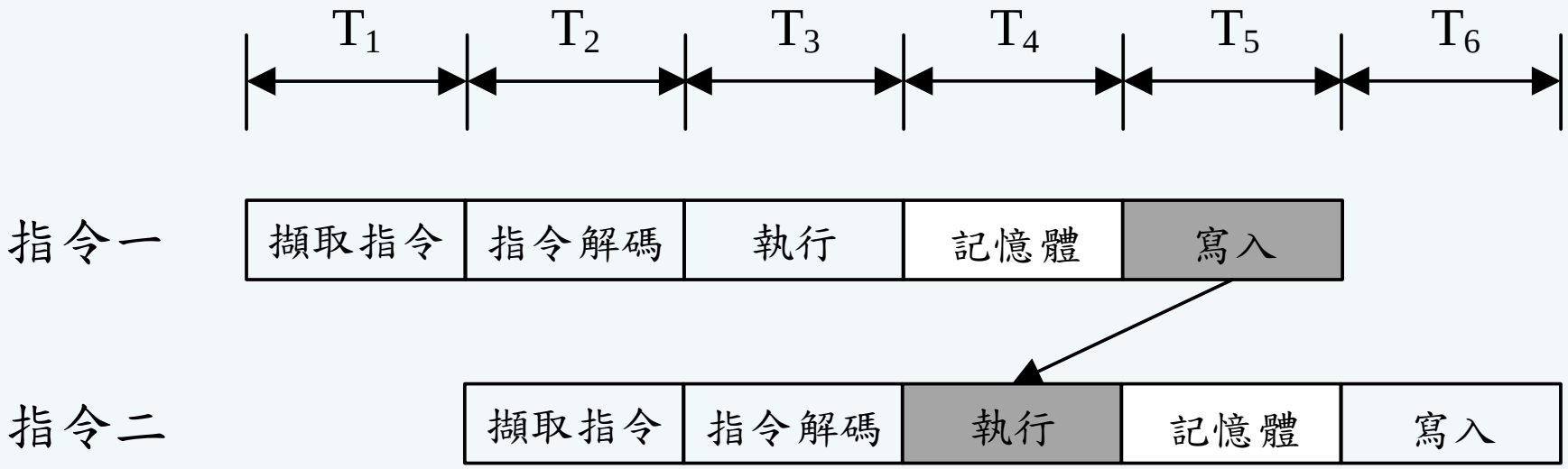
當硬體資源不足，而管線內不同的指令需要同時使用同一個硬體資源，導致使用資源衝突，這種現象稱之為結構危害。
例如：微控制器只有一個記憶體，在T_4時，指令一的第4個階段需要存取記憶體運算元，與指令四的第1個階段從記憶體讀取指令，因為同時間使用記憶體，而發生資料衝突導致結果錯誤，如上圖所示

結構危害（Structure Hazard）



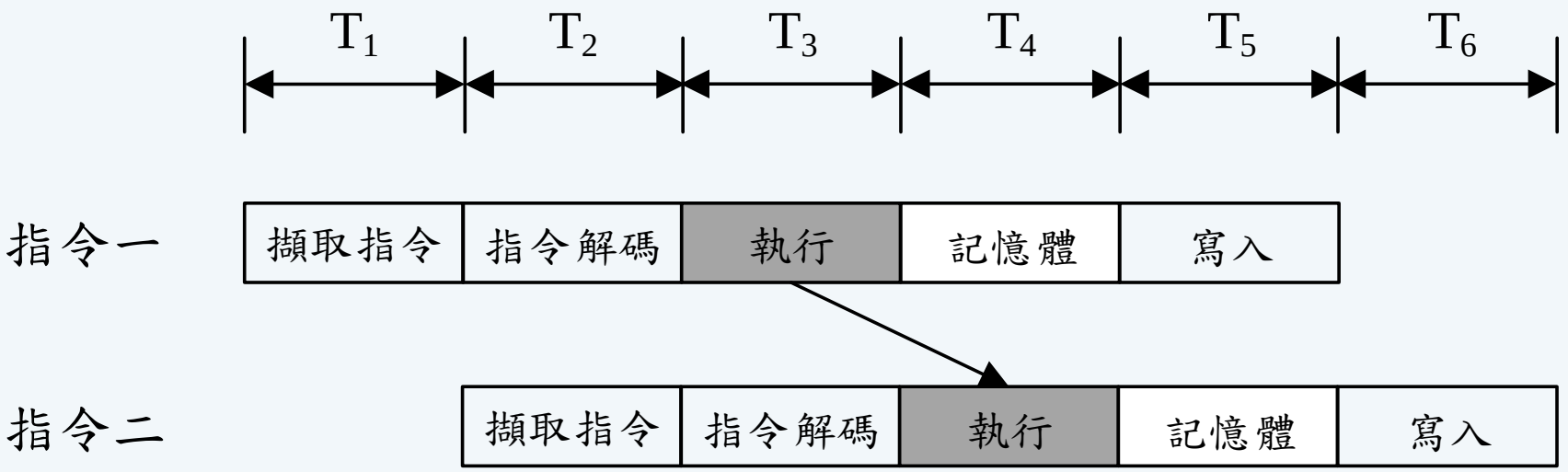
解決方式：將指令記憶體與資料記憶體分開。PIC16LF1826 微控制器採用的是哈佛架構，此架構已將指令記憶體與資料記憶體分開，並具有各自獨立的記憶體匯流排，所以可以避免這項危害。

資料危害（Data Hazard）



當管線中的某一個指令，需要使用到管線中前一個指令的運算結果，但是前一個指令的運算尚未完成，這種現象稱之為資料危害，例如：指令二需要用到指令一寫入暫存器的結果，由於指令一在T_5才會將結果寫到暫存器，但是指令二在T_4的時候必須讀出暫存器的結果進行運算，因此會造成指令執行錯誤，如上圖所示。

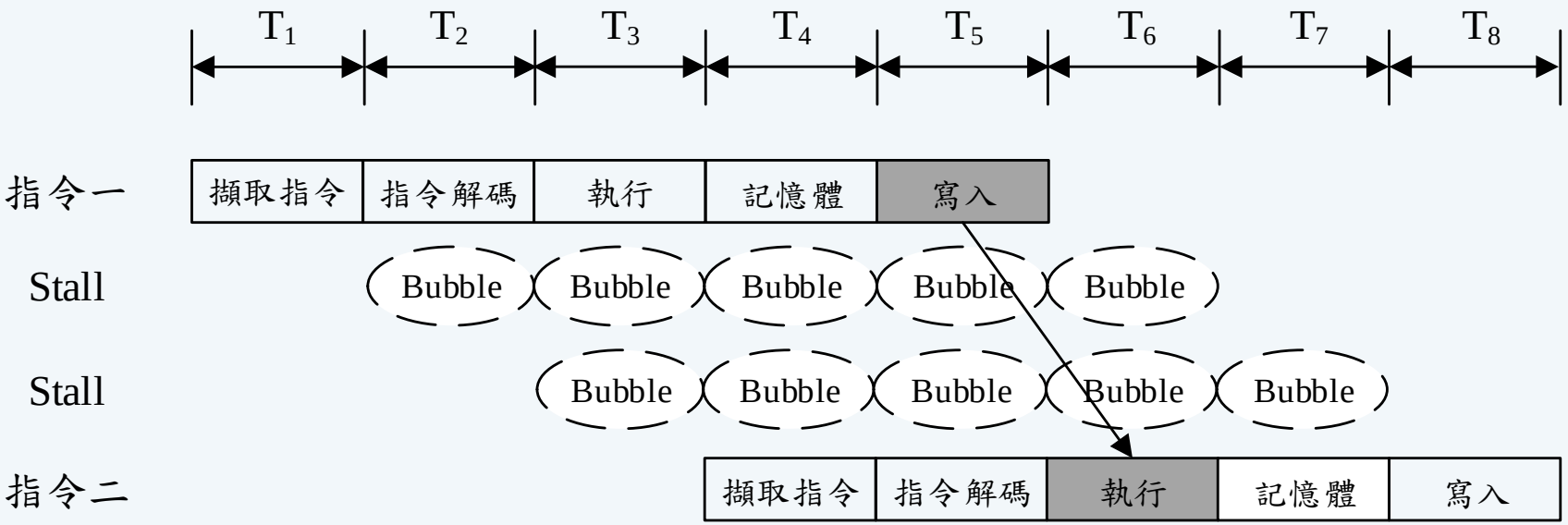
資料危害（Data Hazard）



解決方法1：前饋(Forwarding)

將前一個指令的運算結果提前取出，提供給下一個指令使用

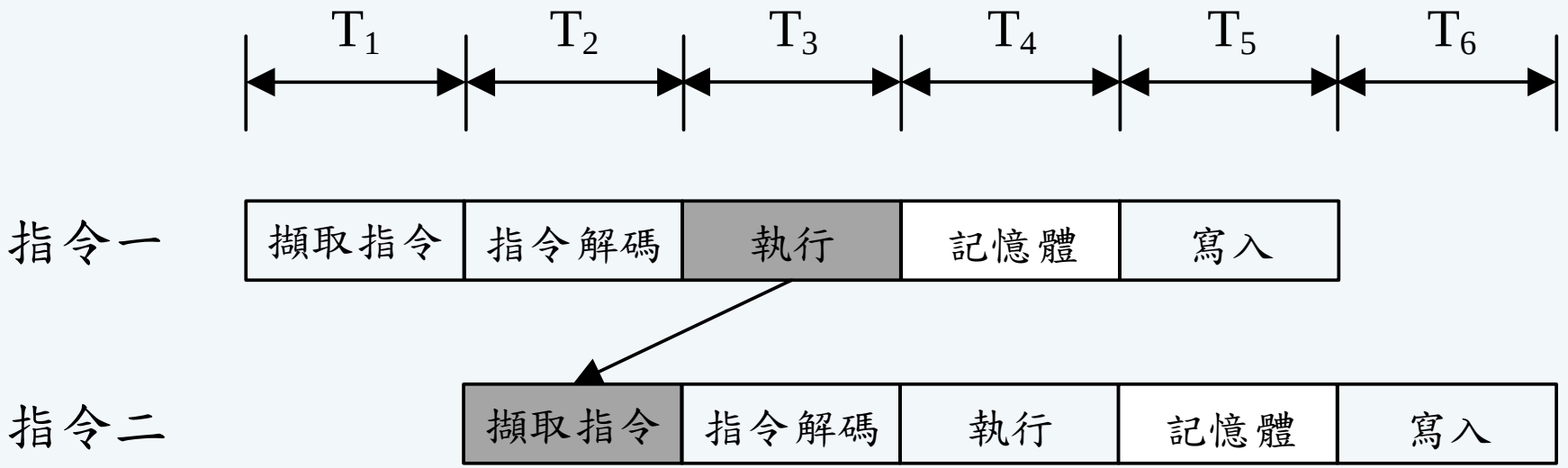
資料危害（Data Hazard）



解決方法2：延遲(Stall)

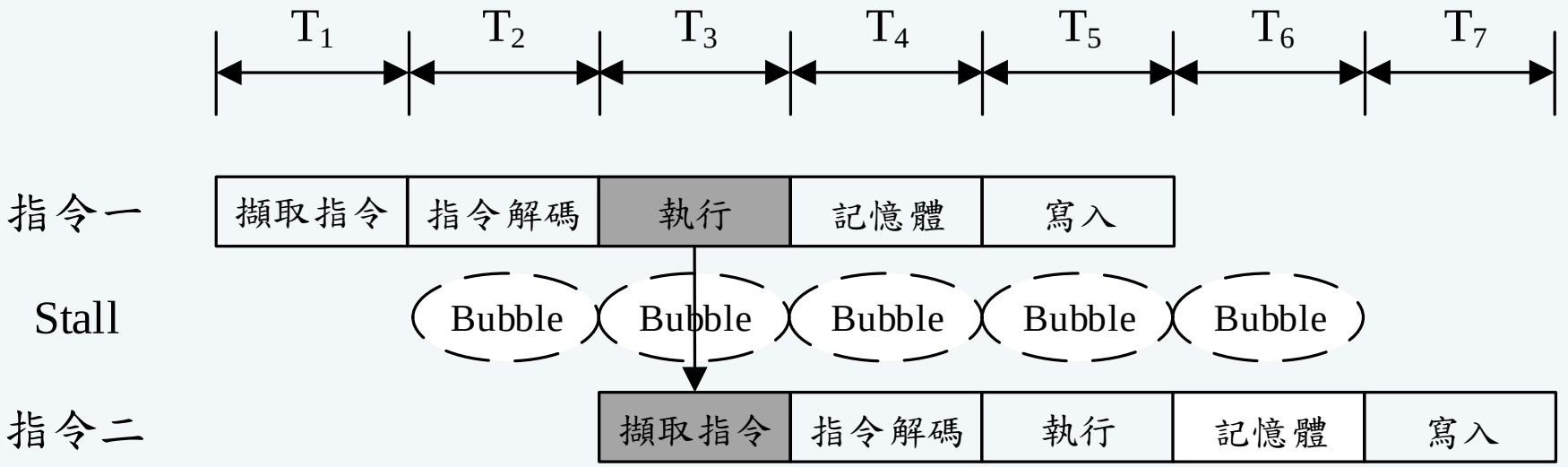
將下一個指令延遲，直到前一個指令運算完成並將資料寫回暫存器

控制危害（Control Hazard）



當執行到會改變程式計數器的指令時，必須等到新的位址後，才能進行指令擷取，這種現象稱之為控制危害。例如：指令一是條件分支指令(Branch on Equal)，在分支指令還沒完成條件判斷並將分支地址擷取出來時，指令二就已經開始擷取指令，因此會造成指令執行結果錯誤

控制危害（Control Hazard）



處理方式：
可以使用延遲方式解決，我們延遲指令二開始執行的時間，等到指令一將分支地址擷取出來後，才可以擷取下一個指令

指令資料流

49個指令分成八個類別，
從八個類別中各挑出部分指令做控制訊號及資料流向範例。

各指令執行所需時間不盡相同，大致上可由類別區分：

- 一個時間週期：
  Literal Operations、Inherent Operations。

- 兩個時間週期：
  Byte-oriented File Register Operations、Bit-oriented File Register Operations、
  Bit-oriented Skip Operations。

- 三個時間週期：
  Byte-oriented Skip Operations、Control Operations、C-Compiler Optimized。

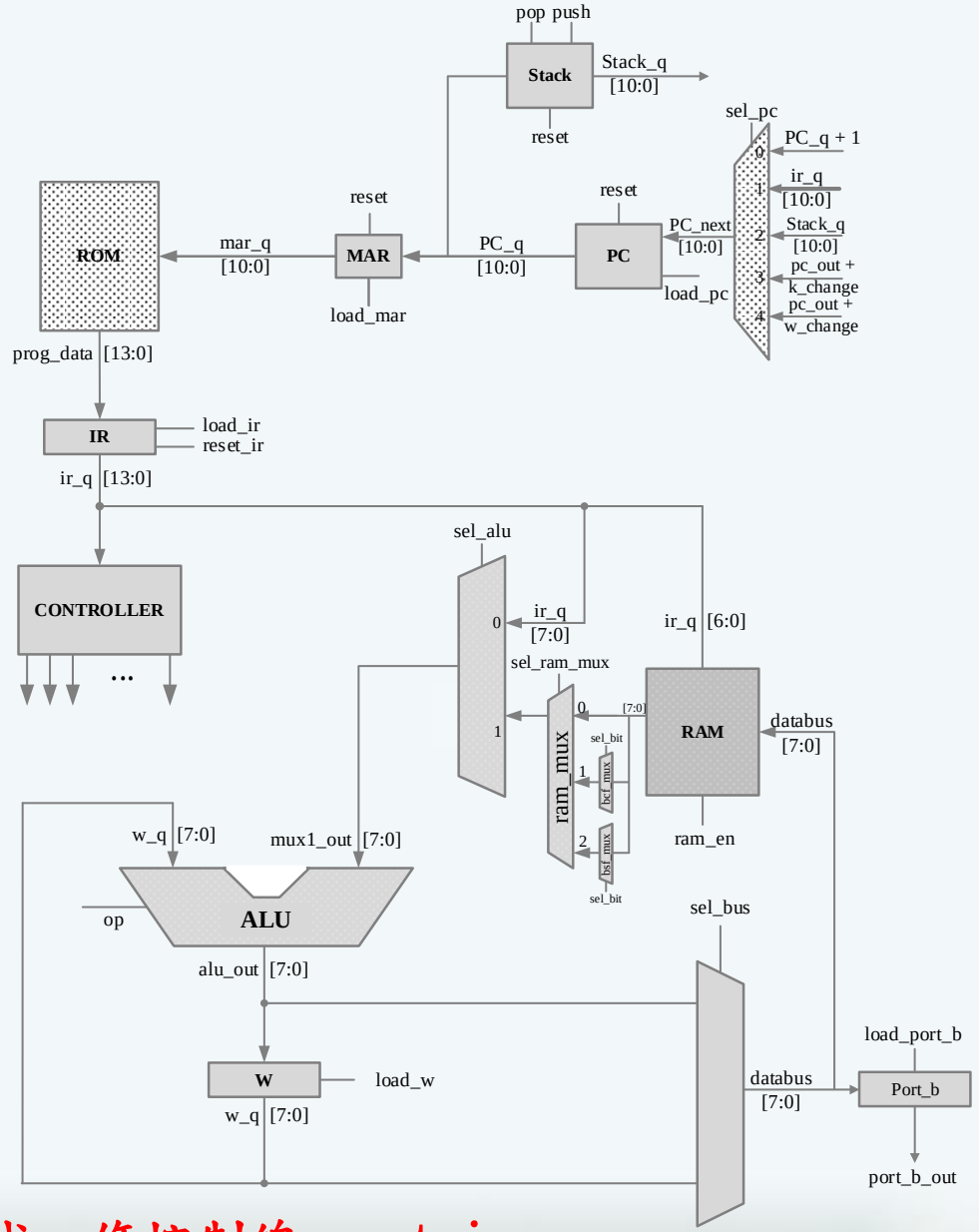
T_1、T_2及T_3擷取階段，控制訊號均相同，如下表所示。

狀態	動作	控制訊號
T_1	MAR←PC	load_mar
T_2	PC←PC+1	sel_pc; load_pc
T_3	IR←ROM[MAR]	load_ir

指令資料流

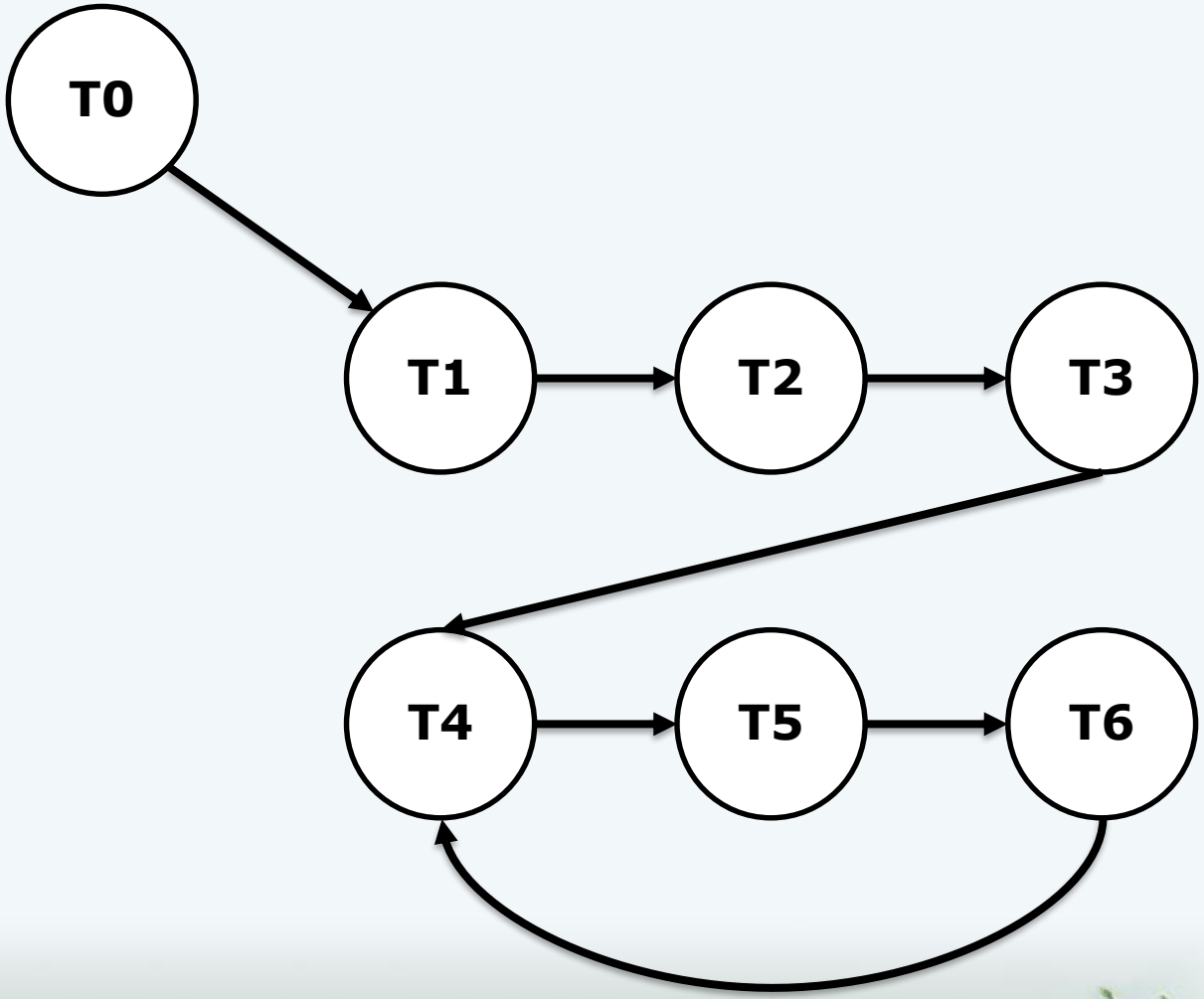
狀態	動作	控制訊號
T_1	MAR←PC PC←PC+1	load_mar sel_pc; load_pc
T_2	X	none
T_3	IR←ROM[MAR]	load_ir

架構圖



＊請將ir的reset獨立成一條控制線reset_ir

PIPELINE PIC16F1826



Next instruction

NON - BRANCH INSTRUCTIONS

狀態	Fetch 動作	Fetch 控制訊號	執行指令 動作	執行指令 控制訊號
T_4	**MAR <= PC** **PC <= PC+1**	load_mar = 1 sel_pc = 2'b00 load_pc = 1	**NORMAL T4 OPERATIONS**	**NORMAL T4 OPERATIONS**
T_5	X	X	**NORMAL T5 OPERATIONS**	**NORMAL T5 OPERATIONS**
T_6	**IR <= ROM[MAR]**	load_ir = 1	**NORMAL T6 OPERATIONS**	**NORMAL T6 OPERATIONS**

Next instruction GOTO

狀態	Fetch 動作	Fetch 控制訊號	GOTO 動作	GOTO 控制訊號
T_4	**MAR <= PC** **PC <= PC+1**	load_mar = 1 sel_pc = 2'b00 load_pc = 1	**X**	**X**
T_5	X	X	**PC <= IR[10:0]**	**sel_pc = 2'b01** **load_pc = 1**
T_6	**IR <=** **ROM[MAR]**	load_ir = 1	**IR <= 0**	**reset_ir = 1** //將下個指令改為NOP

Next instruction CALL

狀態	Fetch 動作	Fetch 控制訊號	CALL 動作	CALL 控制訊號
T_4	**MAR <= PC** **PC <= PC+1**	load_mar = 1 sel_pc = 2'b00 load_pc = 1	**Stack_in <= PC**	**push = 1**
T_5	X	X	**PC <= IR[10:0]**	**sel_pc = 2'b01** **load_pc = 1**
T_6	**IR <=** **ROM[MAR]**	load_ir = 1	**IR <= 0**	**reset_ir = 1** //將下個指令改為NOP

Next instruction RETURN

狀態	Fetch 動作	Fetch 控制訊號	RETURN 動作	RETURN 控制訊號
T_4	**MAR <= PC** **PC <= PC+1**	load_mar = 1 sel_pc = 2'b00 load_pc = 1	**X**	X
T_5	X	X	**PC <= Stack_out**	**pop = 1** **sel_pc = 2'b10** **load_pc = 1**
T_6	**IR <= ROM[MAR]**	load_ir = 1	**IR <= 0**	**reset_ir = 1** //將下個指令改為NOP

Next instruction： DECFSZ

狀態	Fetch 動作	Fetch 控制訊號	DECFSZ 動作	DECFSZ 控制訊號
T_4	**MAR <= PC** **PC <= PC+1**	load_mar = 1 sel_pc = 2'b00 load_pc = 1	**X**	X
T_5	X	X	**X**	X

T_6	**IR <= ROM[MAR]**	load_ir = 1	**addr= ir[6:0]**
			
			if d==0:
			W <= RAM[addr] -1
			if d==1:
			RAM[addr] <=RAM[addr] -1
			
			if alu_zero== 1 :
			IR <= 0
				op=7;
				sel_alu=1;
				if d == 0:
				load_w=1
				if d == 1:
				sel_bus=0;
				ram_en=1;
				if aluout_zero==1:
				reset_ir = 1
				//將下個指令改為NOP

Next instruction： INCFSZ

狀態	Fetch 動作	Fetch 控制訊號	INCFSZ 動作	INCFSZ 控制訊號
T_4	**MAR <= PC** **PC <= PC+1**	load_mar = 1 sel_pc = 2'b00 load_pc = 1	**X**	X
T_5	X	X	**X**	X

T_6	**IR <= ROM[MAR]**	load_ir = 1	**addr= ir[6:0]**
			if d==0:
			W <= RAM[addr] -1
			if d==1:
			RAM[addr] <=RAM[addr] -1
			if alu_zero== 1 :
			IR <= 0
				op=6;
				sel_alu=1;
				if d == 0:
				load_w=1
				if d == 1:
				sel_bus=0;
				ram_en=1;
				if aluout_zero==1:
				reset_ir = 1
				//將下個指令改為NOP

Next instruction： BTFSC BTFSS

狀態	Fetch 動作	Fetch 控制訊號	BTFSC/BTFSS 動作	BTFSC/BTFSS 控制訊號
T_4	**MAR <= PC** **PC <= PC+1**	load_mar = 1 sel_pc = 2'b00 load_pc = 1	**X**	X
T_5	X	X	**X**	X
T_6	**IR <= ROM[MAR]**	load_ir = 1	**addr= ir[6:0]** **If btfsc_btfss_skip_bit==1:** **IR<=0**	**If btfsc_btfss_skip_bit == 1:** **reset_ir = 1**

Next instruction： BRA

狀態	Fetch 動作	Fetch 控制訊號	BRA 動作	BRA 控制訊號
T_4	**MAR <= PC** **PC <= PC+1**	load_mar = 1 sel_pc = 2'b00 load_pc = 1	**X**	X
T_5	X	X	pc_q <= pc_q + k_change	**load_pc=1** **sel_pc=3**
T_6	**IR <=** **ROM[MAR]**	load_ir = 1	**IR <= 0**	**reset_ir = 1** //將下個指令改為NOP

```
assign k_change = {ir_out[8],ir_out[8],ir_out[8:0]} - 1;
```

Next instruction： BRW

狀態	Fetch 動作	Fetch 控制訊號	BRW 動作	BRW 控制訊號
T_4	**MAR <= PC** **PC <= PC+1**	load_mar = 1 sel_pc = 2'b00 load_pc = 1	**X**	X
T_5	X	X	pc_q <= pc_q + w_change	**load_pc=1** **sel_pc=4**
T_6	**IR <=** **ROM[MAR]**	load_ir = 1	**IR <= 0**	**reset_ir = 1** //將下個指令改為NOP

```
assign w_change = {3'b0, w_q} - 1;
```

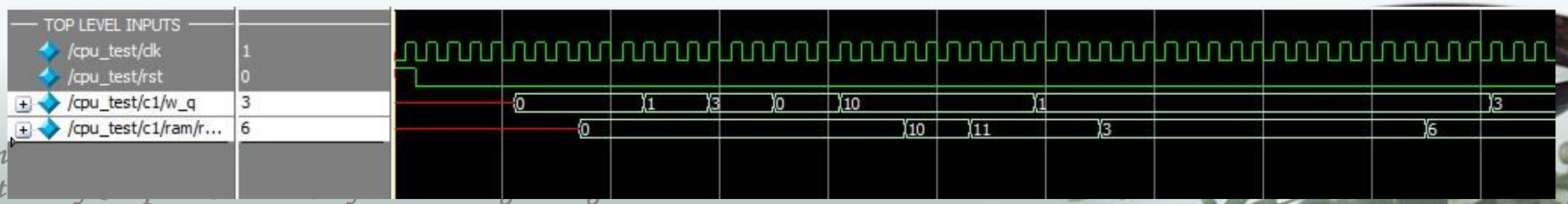
PIPELINE測試程式1

```
#include    <p16Lf1826.inc>     ; Include file l
;

temp        equ     0x25
;*************************************
;           Program start              *
;*************************************
            org     0x00        ; reset vector

loop        clrw
            clrf    temp
            movlw   .1
            addlw   .2
            sublw   .3
            movlw   .10
            movwf   temp
            incf    temp,1
            subwf   temp,0
            bcf     temp,3
            btfsc   temp,3
            btfsc   temp,1
            brw
            nop
            lslf    temp,1
            lsrf    temp,0
            goto    loop


            end
```

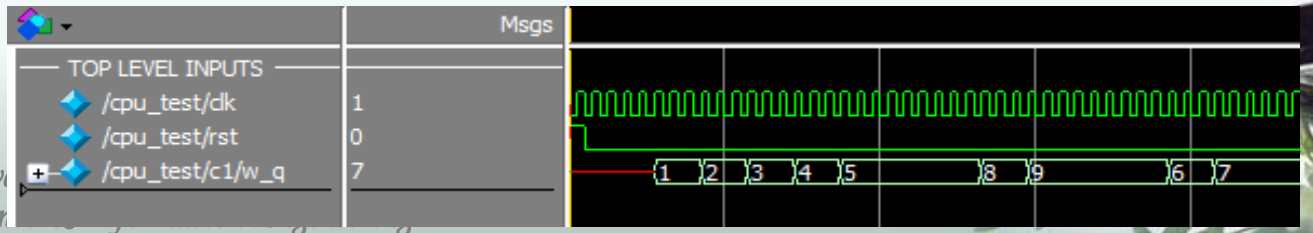


PIPELINE測試程式2

```
#include     <p16Lf1826.inc>      ; Include file lo
;

temp         equ 0x25
;*****************************************
;            Program start                *
;*****************************************
             org      0x00         ; reset vector

             movlw    .1
             movlw    .2
             movlw    .3
             movlw    .4
             movlw    .5
             call     first
             movlw    .6
             movlw    .7
             goto     $
first        movlw    .8
             movlw    .9
             return
             end
```



PIPELINE測試程式3

```
#include    <p16Lf1826.inc>     ; Include file
;

temp        equ     0x25
;*****************************************
;           Program start                *
;*****************************************
            org     0x00        ; reset vector

            clrw
            clrf    temp
            movlw   .3
            movwf   temp
            movlw   0
loop        addlw   .1
            decfsz  temp
            goto    loop
            goto    $
```