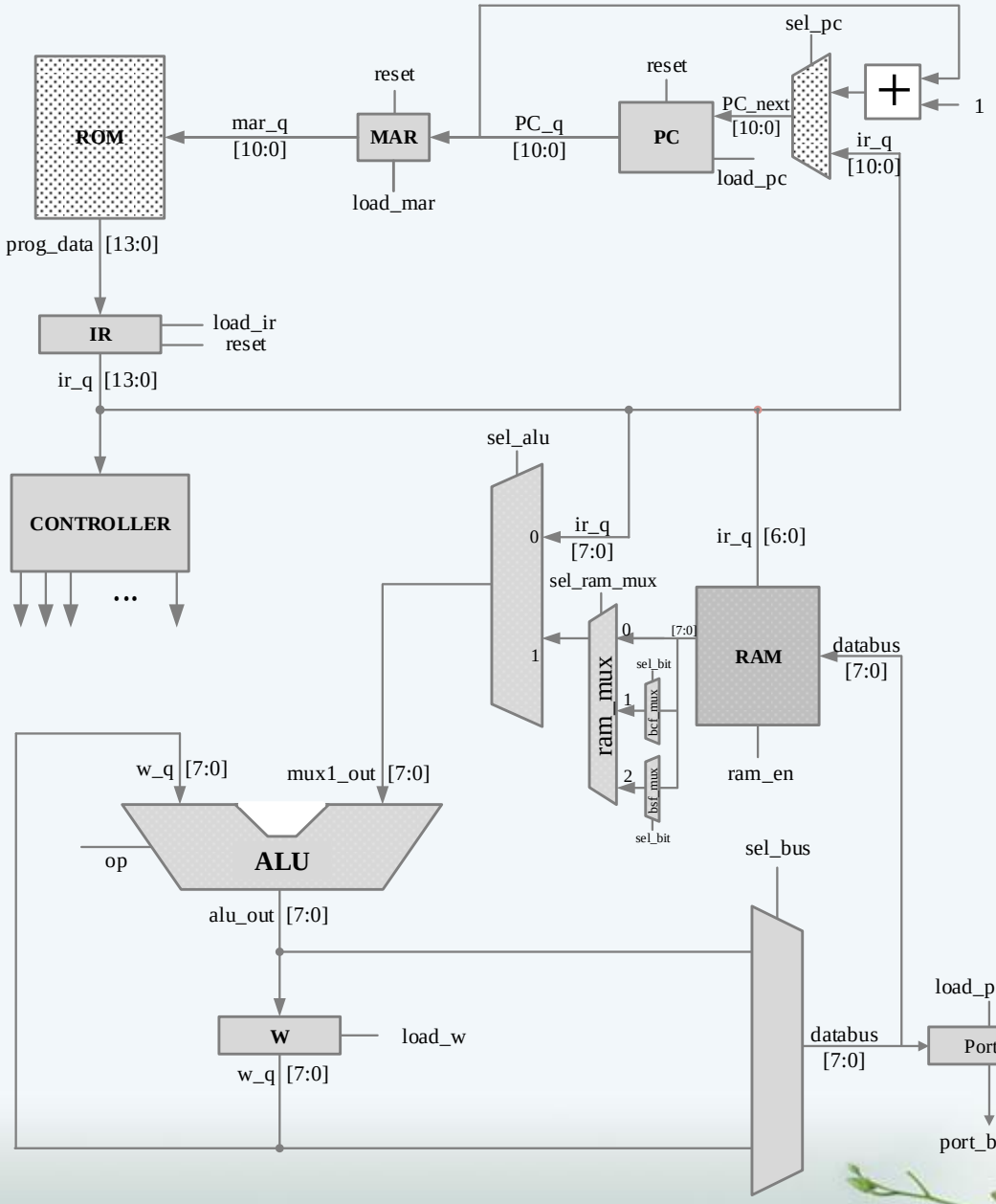
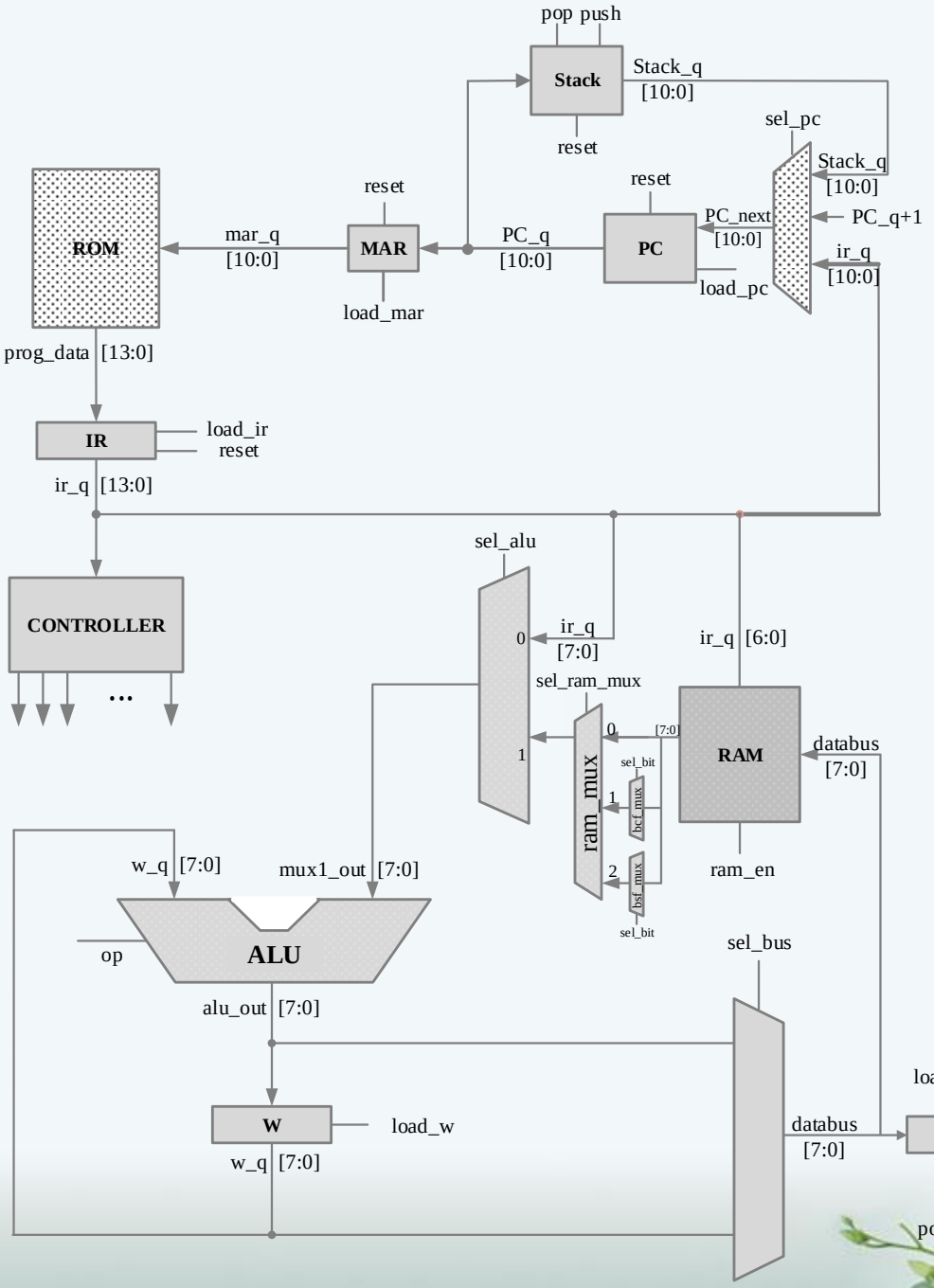
跳躍指令

舊架構圖


sel_pc
reset
reset
ROM
mar_q
[10:0]
MAR
PC_q
[10:0]
PC
PC_next
[10:0]
1
ir_q
[10:0]
load_pc
load_mar
prog_data [13:0]
IR
load_ir
reset
ir_q [13:0]
CONTROLLER
...
sel_alu
0
1
ir_q
[7:0]
ir_q [6:0]
sel_ram_mux
ram_mux
0
1
2
[7:0]
sel_bit
bsl_mux
bsl_mux
sel_bit
RAM
databus
[7:0]
ram_en
w_q [7:0]
mux1_out [7:0]
op
ALU
alu_out [7:0]
sel_bus
W
load_w
w_q [7:0]
databus
[7:0]
load_port_b
Port_b
port_b_out

新架構圖


pop push
Stack
Stack_q
[10:0]
reset
sel_pc
Stack_q
[10:0]
reset
ROM
mar_q
[10:0]
reset
MAR
load_mar
PC_q
[10:0]
PC
PC_next
[10:0]
PC_q+1
ir_q
[10:0]
load_pc
prog_data [13:0]
IR
load_ir
reset
ir_q [13:0]
CONTROLLER
...
sel_alu
0
1
ir_q
[7:0]
ir_q [6:0]
sel_ram_mux
ram_mux
0
1
2
[7:0]
sel_bit
bcl_mux
bsl_mux
sel_bit
RAM
databus
[7:0]
ram_en
w_q [7:0]
mux1_out [7:0]
op
ALU
alu_out [7:0]
sel_bus
W
load_w
w_q [7:0]
load_port_b
databus
[7:0]
Port_b
port_b_out

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。

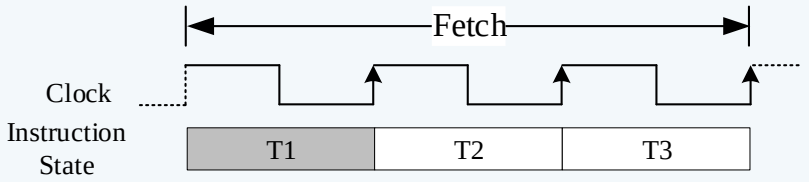
- 兩個時間週期：
  Byte-oriented File Register Operations、Bit-oriented File Register Operations、
  Bit-oriented Skip Operations。

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

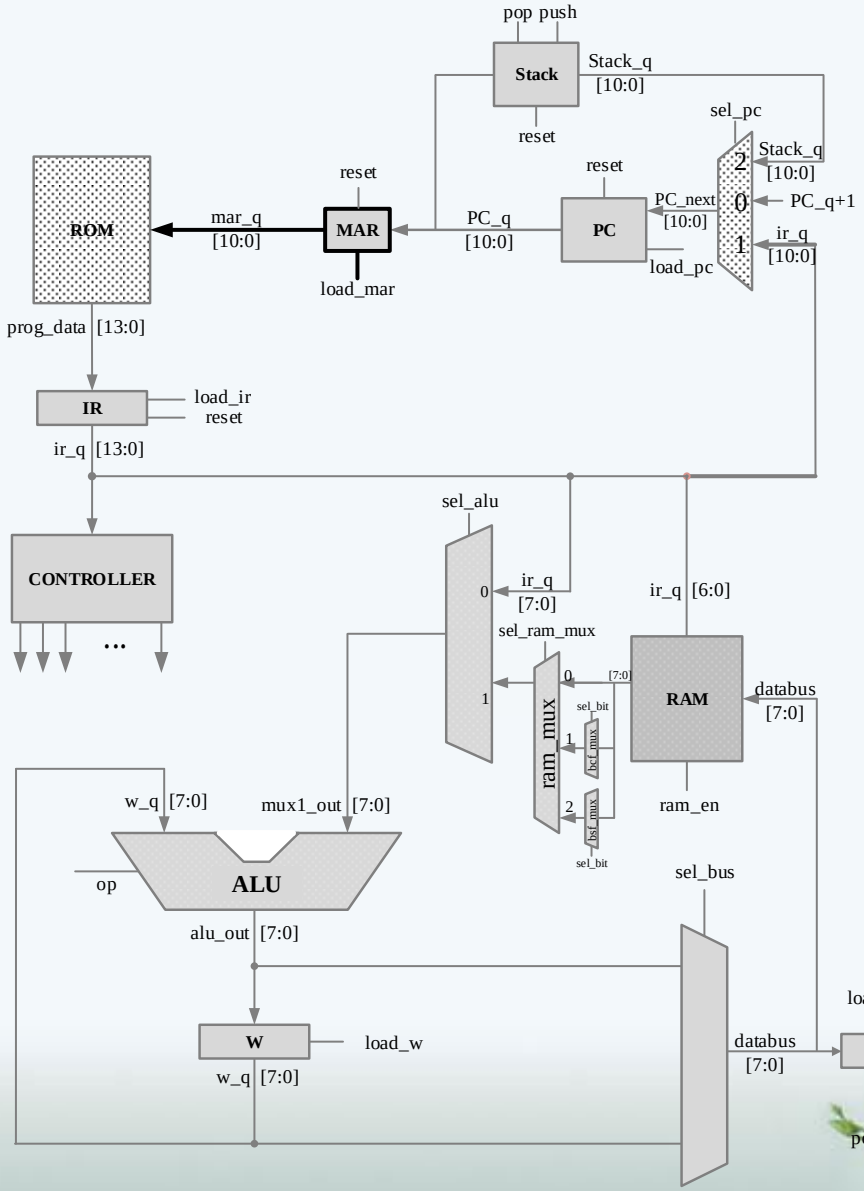
T_1、T_2及T_3擷取階段，控制訊號均相同，如下表所示。

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

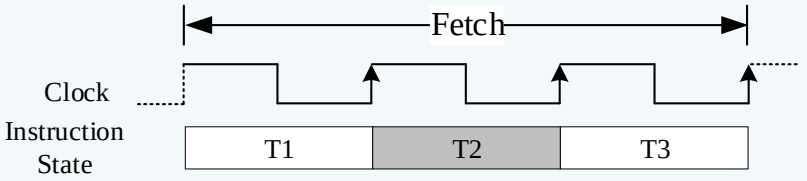
Fetch T1



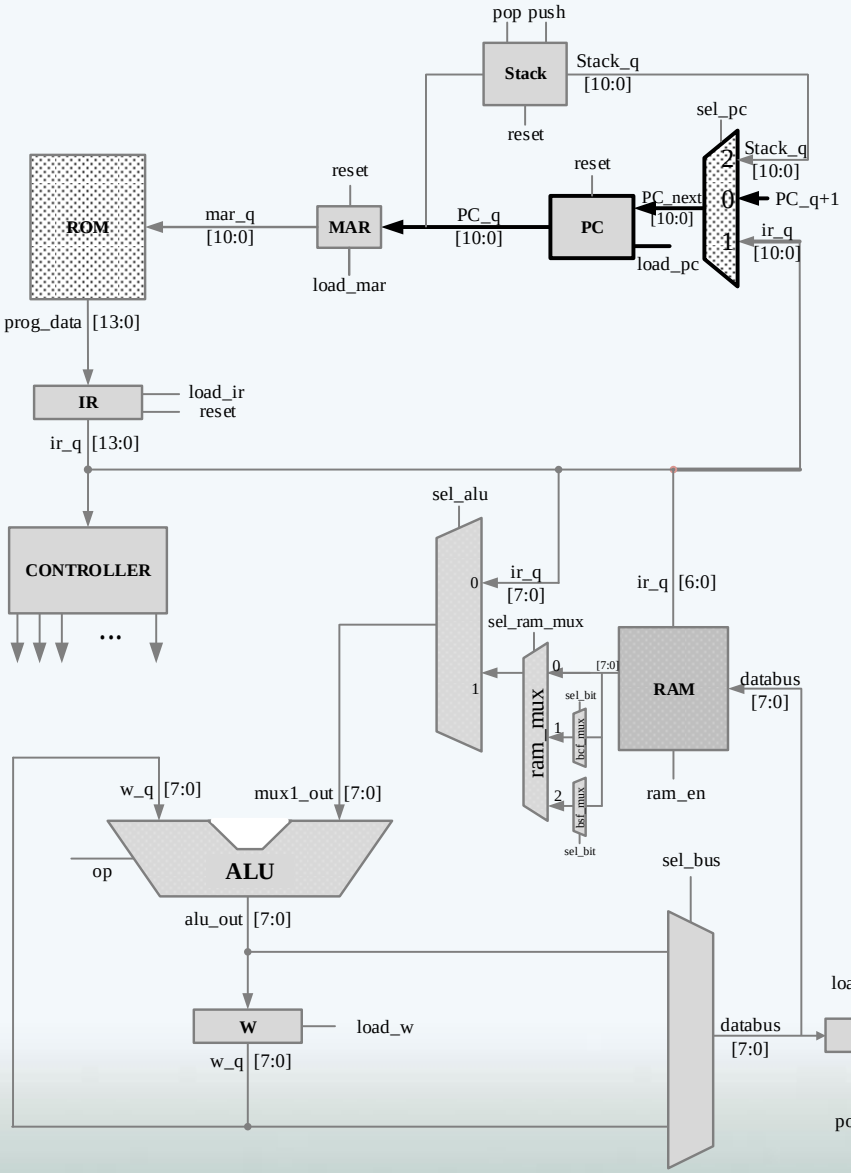
狀態	動作	控制訊號
T_1	MAR←PC	load_mar



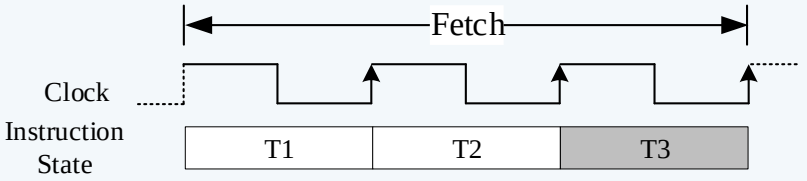
Fetch T2



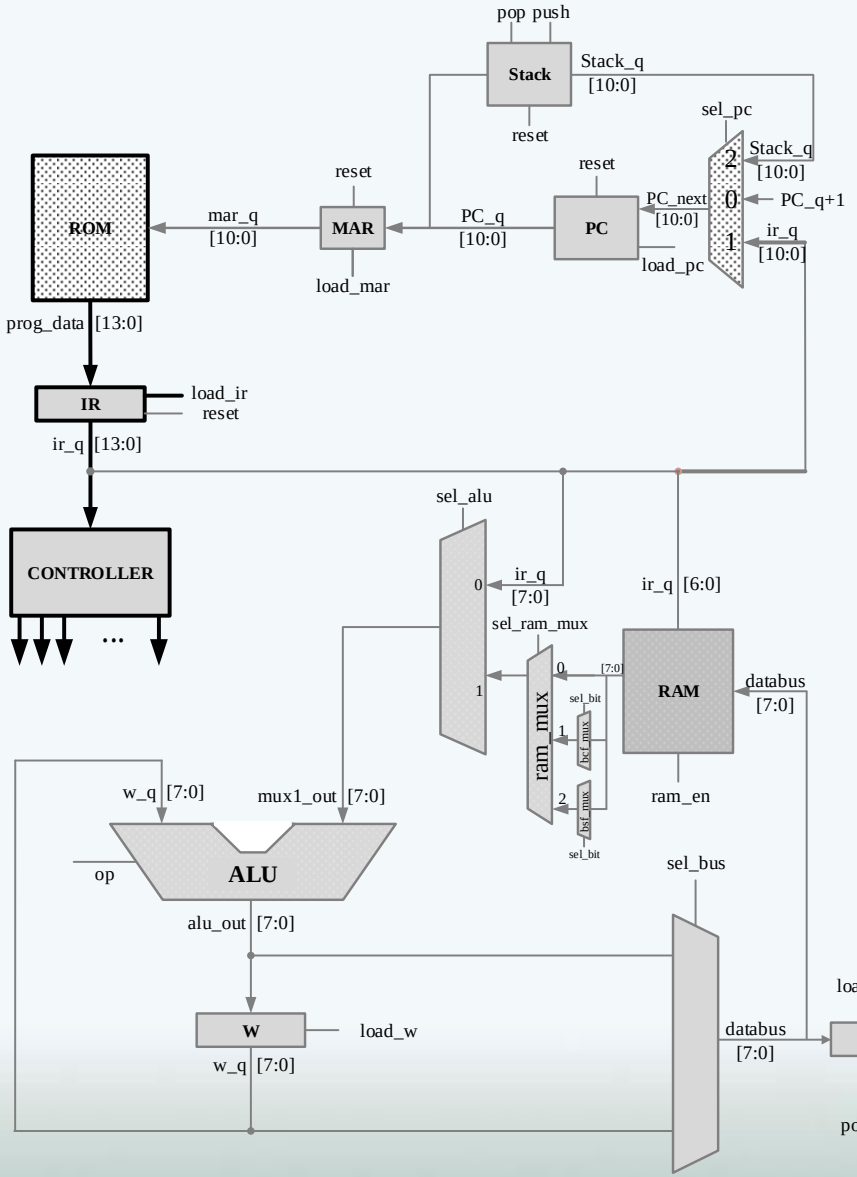
狀態	動作	控制訊號
T_2	PC←PC+1	sel_pc = 0; load_pc



Fetch T3



狀態	動作	控制訊號
T_3	IR←ROM[MAR]	load_ir



CONTROL OPERATIONS

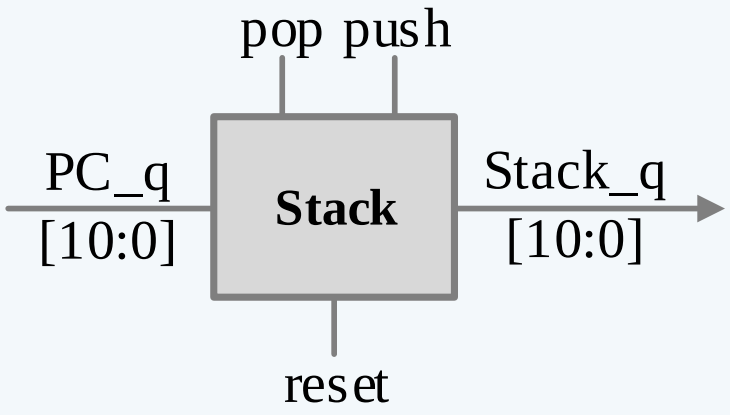
PIC16F1826 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected	Notes
				MSB			LSB		
BIT-ORIENTED SKIP OPERATIONS									
CALL	k	Call Subroutine	2	10	0kkk	kkkk	kkkk		
RETURN	–	Return from Subroutine	2	00	0000	0000	1000		

CALL	**Call Subroutine**
Syntax:	[*label*] CALL k
Operands:	0 ≤ k ≤ 2047
Operation:	(PC)+ 1→ TOS, k → PC<10:0>, (PCLATH<6:3>) → PC<14:11>
Status Affected:	None
Description:	Call Subroutine. First, return address (PC + 1) is pushed onto the stack. The eleven-bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH. CALL is a two-cycle instruction.

RETURN	**Return from Subroutine**
Syntax:	[*label*] RETURN
Operands:	None
Operation:	TOS → PC
Status Affected:	None
Description:	Return from subroutine. The stack is POPed and the top of the stack (TOS) is loaded into the program counter. This is a two-cycle instruction.

Stack.v




```
module Stack(
    output logic [10:0] stack_out,
    input [10:0] stack_in;
    input push,
    input pop,
    input reset,
    input clk
);
//---------
    logic [3:0] stk_ptr;
    logic [10:0] stack [15:0];
    logic [10:0] stack_out;
    logic [3:0] stk_index;

//---------
    assign stk_index = stk_ptr + 1;
    assign stack_out = stack[stk_ptr];

//---------
    always_ff @(posedge clk)
    begin
        if (reset)
            stk_ptr <= 4'b1111;

        else if (push)
        begin
            stack[stk_index] <= stack_in;
            stk_ptr <= stk_ptr + 1;
        end

        else if (pop)
            stk_ptr <= stk_ptr - 1;
    end
endmodule
```

Stack

stk_ptr: stack頂部的位址
stk_index: 下一筆資料要放在stack的哪一個位址(也就是stk_ptr + 1)

if stack 為空

stack_ptr→	[15]	(empty)
	. . .	. . .
	[2]	(empty)
	[1]	(empty)
stack_index→	[0]	(empty)

if stack不為空

	[15]	(empty)
	. . .	. . .
	[2]	(empty)
stack_index→	[1]	(empty)
stack_ptr→	[0]	(data)

if push

	[15]	(empty)
	. . .	. . .
stack_index→	[2]	(empty)
stack_ptr→	[1]	(data)
	[0]	(data)

if pop

	[15]	(empty)
	. . .	. . .
	[2]	(empty)
stack_ptr→	[1]	(empty)
stack_index→	[0]	(data)

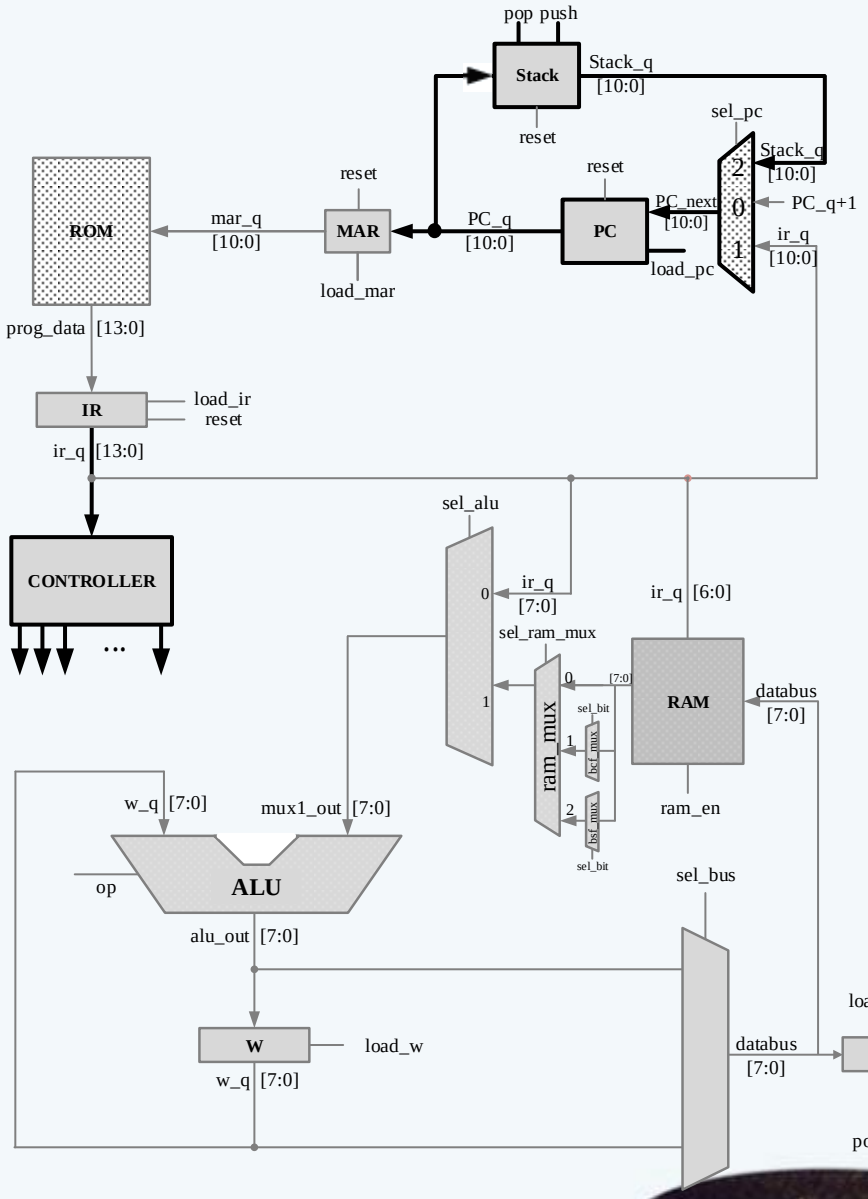
CALL T4

T4

狀態	動作	控制訊號
T_4	stack_q←PC_q	sel_pc = 1 load_pc = 1 push = 1

RETURN T4

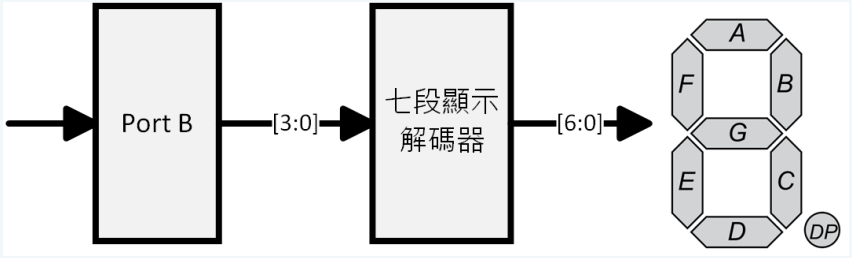
狀態	動作	控制訊號
T_4	PC_q ←stack_q	sel_pc = 2 load_pc = 1 pop = 1



上課實作

由1數到15，再由15數回1

- Demo1: 模擬波形圖(組合語言中不使用delay)
- Demo2: 燒錄DE0(需在Port B 外，接上一個「七段顯示器解碼器」，且組合語言需使用delay，如右圖)




```
#include    <p16Lf1826.inc>     ; Include file locate at defult directory
;

temp        equ 0x25
templ       equ 0x24
countl      equ h'20'
count2      equ h'21'
count3      equ h'22'
;*************************************
;           Program start              *
;*************************************
            org     0x00        ; reset vector

start       movlw   .15
            movwf   templ
            clrf    temp        ; //ram[37]<=0
            clrw                ; //w<=0
loopl       movlw   1           ; //w<=1
            addwf   temp,1      ; //ram[37]<=1
            movf    temp,0      ; //w<=ram[37], w<=1
            movwf   PORTB       ; //PORTB<=1
            call    delay
            decfsz  templ,1
            goto    loopl
            movlw   .15
            movwf   templ
loop2       movlw   1
            subwf   temp,1
            movf    temp,0
            movwf   PORTB
            call    delay
            decfsz  templ,1
            goto    loop2
            goto    start

delay       movlw   .30
            movwf   countl
delayl      clrf    count2
delay2      clrf    count3
delay3      decfsz  count3,1
            goto    delay3
            decfsz  count2,1
            goto    delay2
            decfsz  countl,1
            goto    delayl
            return

            end
```