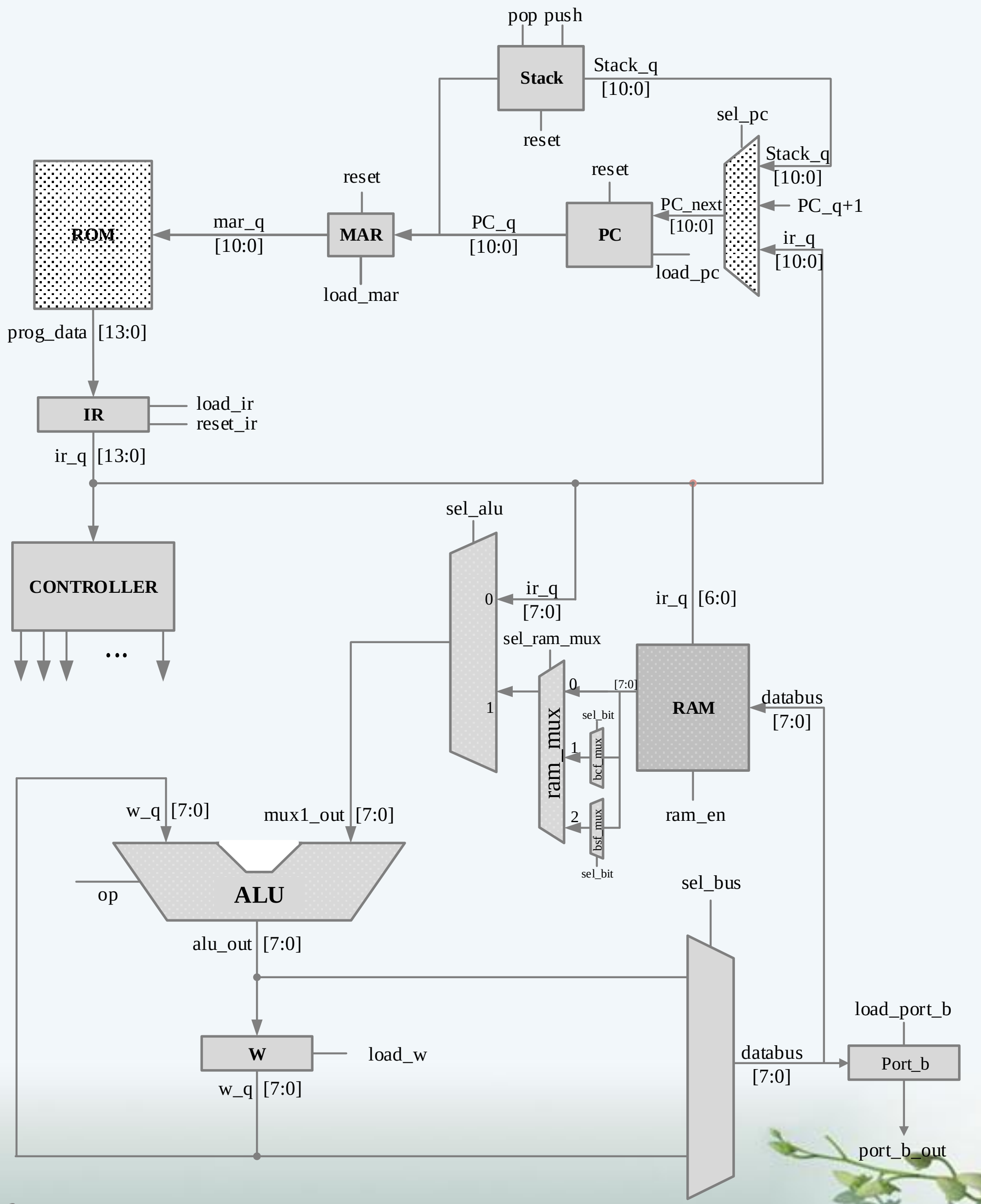
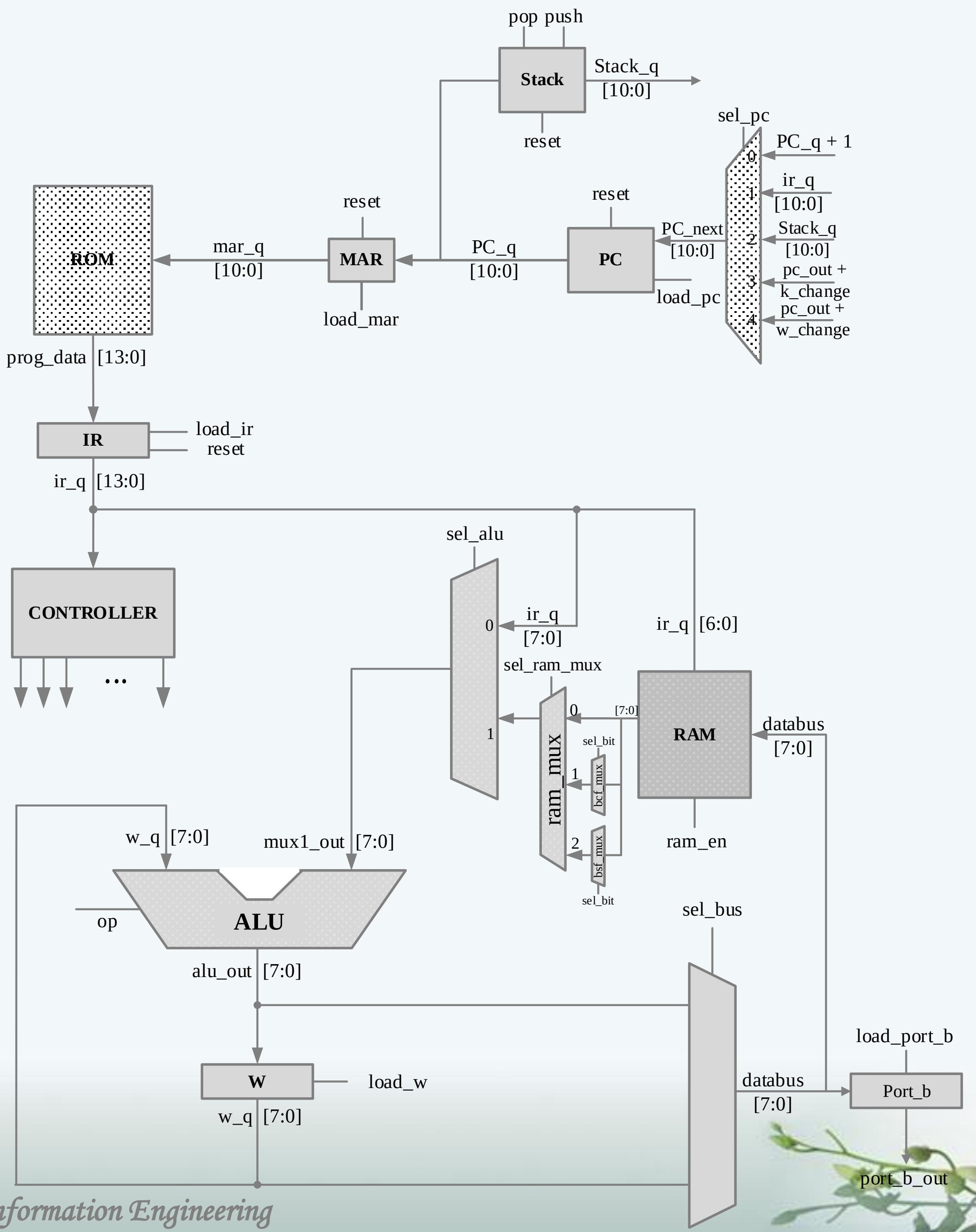
跳躍指令

舊架構圖


pop push
Stack
reset
Stack_q [10:0]
sel_pc
Stack_q [10:0]
PC_q+1
ir_q [10:0]
reset
PC
PC_next [10:0]
load_pc
PC_q [10:0]
reset
MAR
load_mar
mar_q [10:0]
ROM
prog_data [13:0]
IR
load_ir
reset_ir
ir_q [13:0]
CONTROLLER
…
sel_alu
0
1
ir_q [7:0]
sel_ram_mux
ram_mux
0
1
2
[7:0]
sel_bit
bcf_mux
bsf_mux
sel_bit
ir_q [6:0]
RAM
databus [7:0]
ram_en
w_q [7:0]
mux1_out [7:0]
op
ALU
alu_out [7:0]
W
load_w
w_q [7:0]
sel_bus
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
PC_q + 1
ir_q [10:0]
Stack_q [10:0]
pc_out + k_change
pc_out + w_change
0
1
2
3
4
reset
PC
PC_next [10:0]
load_pc
PC_q [10:0]
reset
MAR
load_mar
mar_q [10:0]
ROM
prog_data [13:0]
IR
load_ir
reset
ir_q [13:0]
CONTROLLER
…
sel_alu
ir_q [7:0]
0
1
sel_ram_mux
ram_mux
0
1
2
[7:0]
sel_bit
bcf_mux
bsf_mux
sel_bit
ir_q [6:0]
RAM
databus [7:0]
ram_en
w_q [7:0]
mux1_out [7:0]
op
ALU
alu_out [7:0]
W
load_w
w_q [7:0]
sel_bus
databus [7:0]
load_port_b
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。

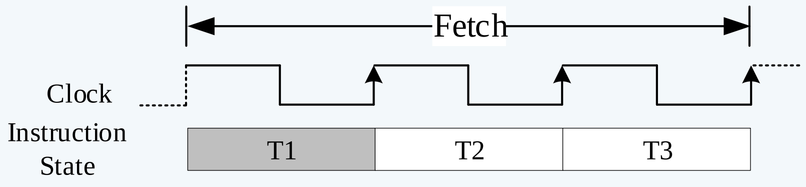
各指令執行所需時間不盡相同，大致上可由類別區分：

- 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。

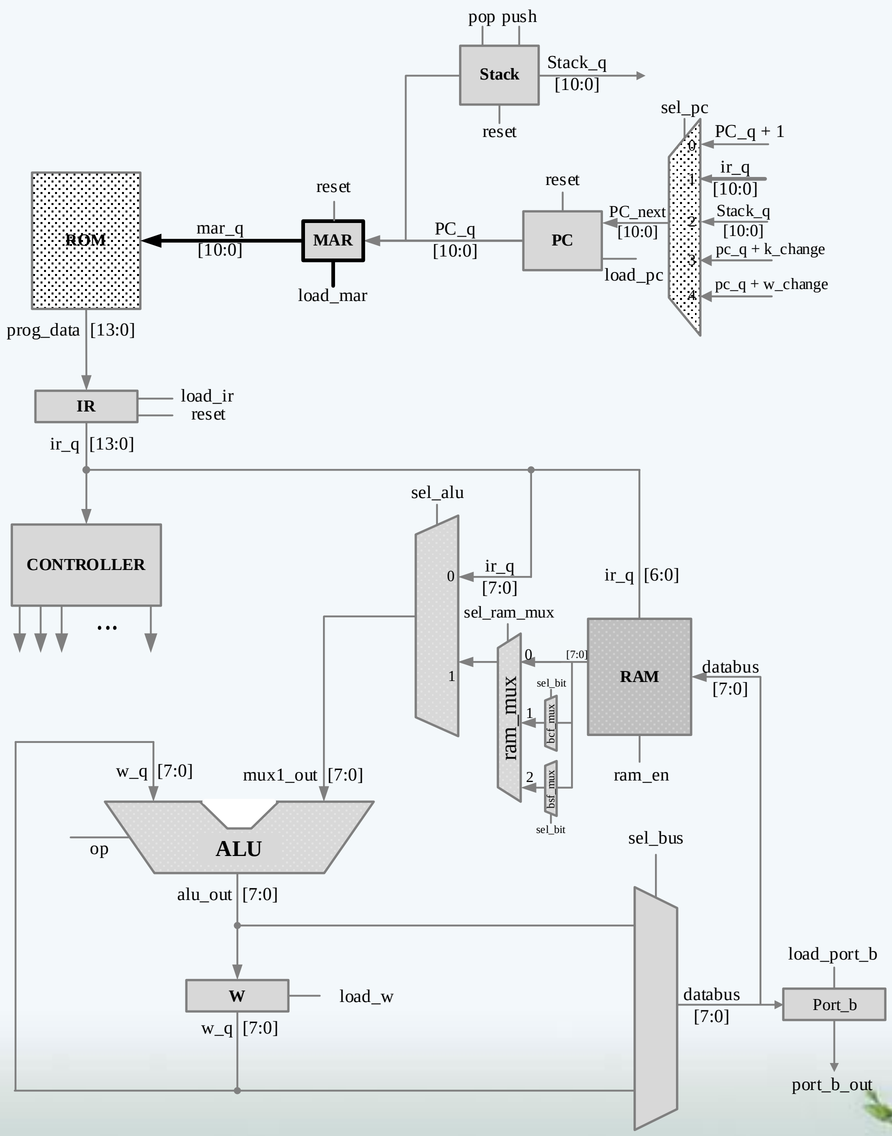
T_1、T_2及T_3擷取階段，控制訊號均相同，如下表所示。

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

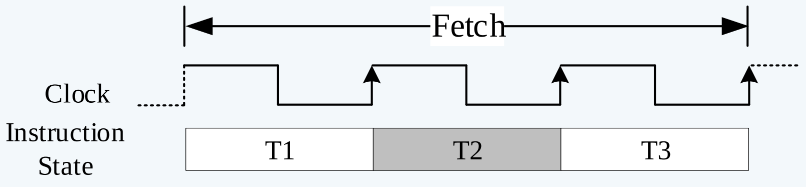
Fetch T1



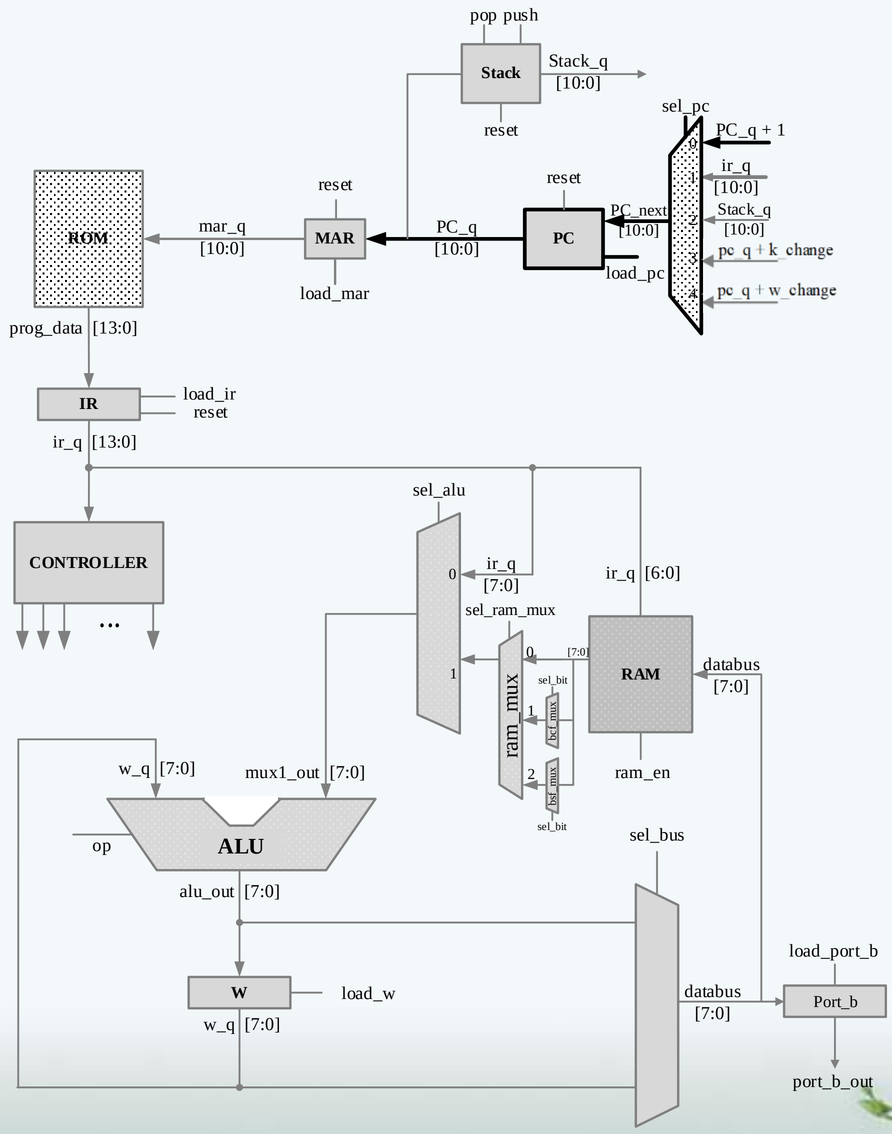
狀態	動作	控制訊號
T_1	MAR←PC	load_mar



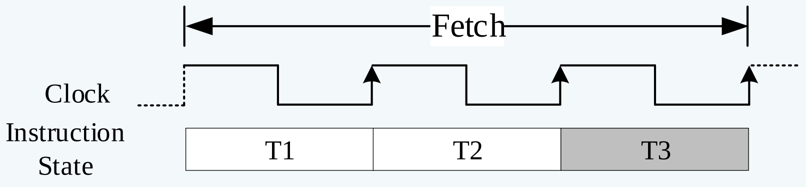
Fetch T2



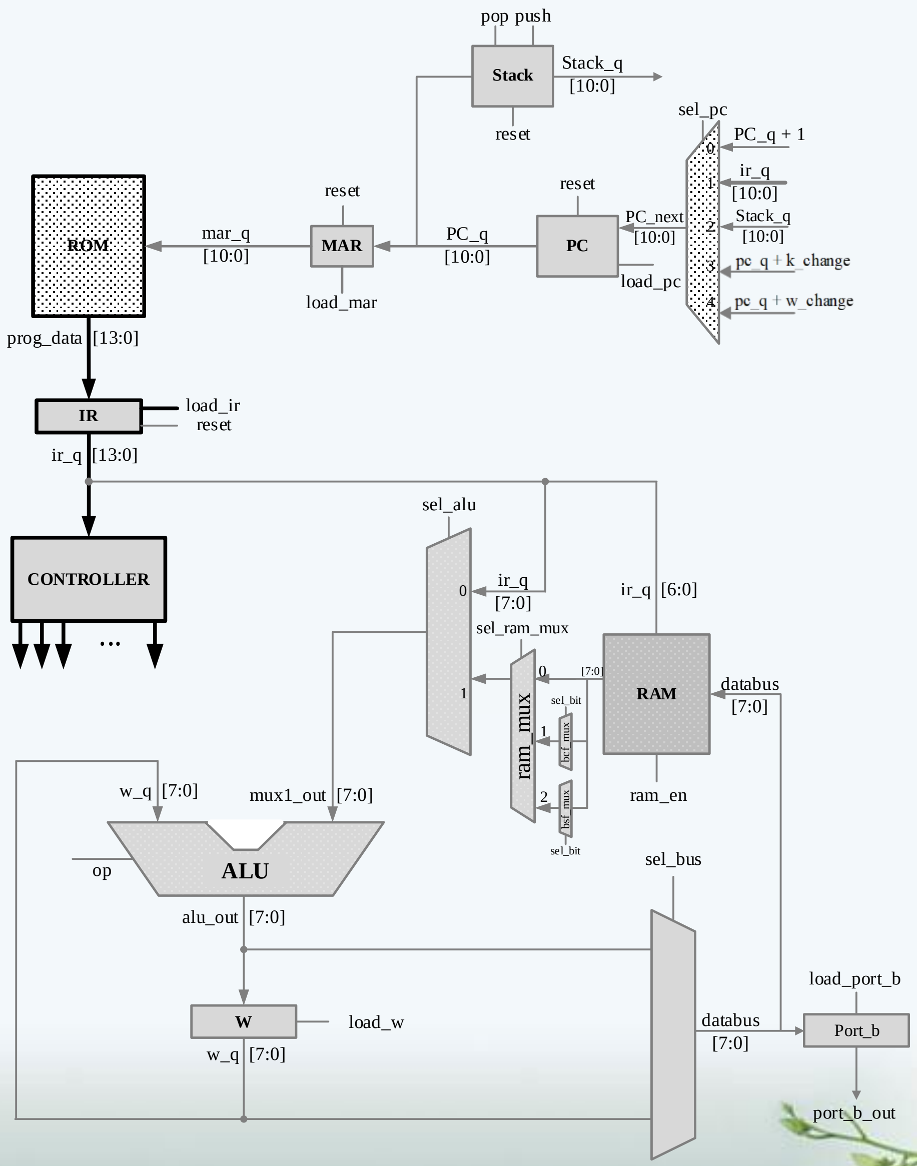
狀態	動作	控制訊號
T_2	PC←PC+1	sel_pc; load_pc



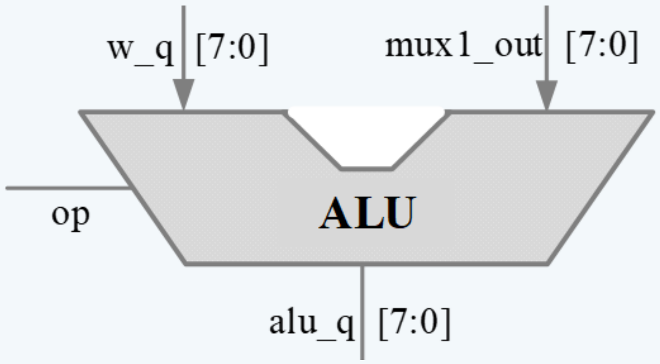
Fetch T3

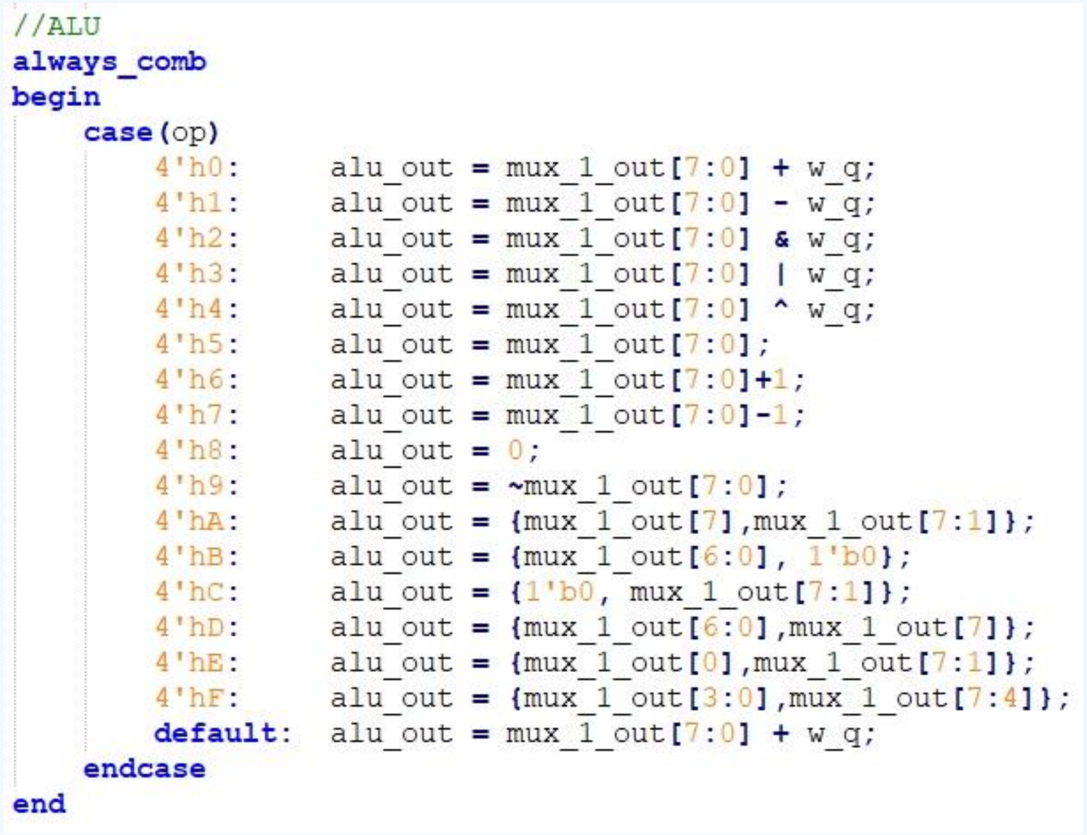


狀態	動作	控制訊號
T_3	IR←ROM[MAR]	load_ir



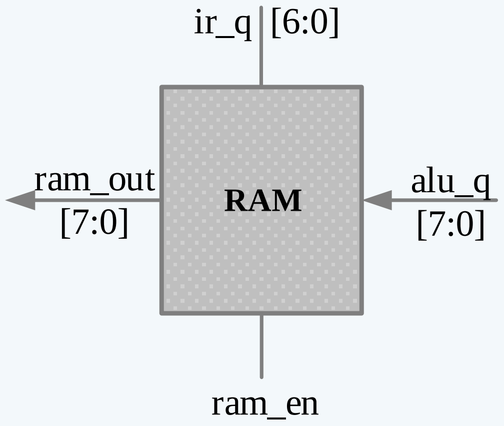
ALU





```
//ALU
always_comb
begin
    case(op)
        4'h0:     alu_out = mux_1_out[7:0] + w_q;
        4'h1:     alu_out = mux_1_out[7:0] - w_q;
        4'h2:     alu_out = mux_1_out[7:0] & w_q;
        4'h3:     alu_out = mux_1_out[7:0] | w_q;
        4'h4:     alu_out = mux_1_out[7:0] ^ w_q;
        4'h5:     alu_out = mux_1_out[7:0];
        4'h6:     alu_out = mux_1_out[7:0]+1;
        4'h7:     alu_out = mux_1_out[7:0]-1;
        4'h8:     alu_out = 0;
        4'h9:     alu_out = ~mux_1_out[7:0];
        4'hA:     alu_out = {mux_1_out[7],mux_1_out[7:1]};
        4'hB:     alu_out = {mux_1_out[6:0], 1'b0};
        4'hC:     alu_out = {1'b0, mux_1_out[7:1]};
        4'hD:     alu_out = {mux_1_out[6:0],mux_1_out[7]};
        4'hE:     alu_out = {mux_1_out[0],mux_1_out[7:1]};
        4'hF:     alu_out = {mux_1_out[3:0],mux_1_out[7:4]};
        default:  alu_out = mux_1_out[7:0] + w_q;
    endcase
end
```

SRAM

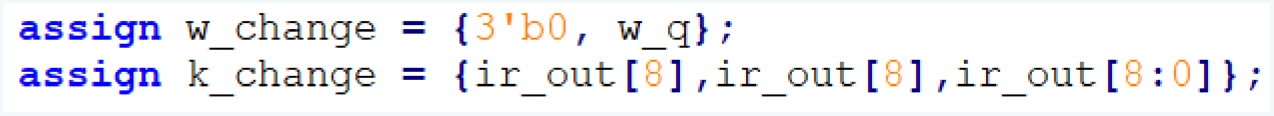


```
module single_port_ram_128x8(
    input [7:0]data,
    input [6:0]addr,
    input ram_en,
    input clk,
    output logic [7:0] q
);
    // Declare the RAM variable
    //reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
    logic [7:0] ram[127:0];

    always_ff @(posedge clk)
    begin
        // Write
        if (ram_en)
            ram[addr] <= data;
    end

    // Continuous assignment implies read returns NEW data.
    // This is the natural behavior of the TriMatrix memory
    // blocks in Single Port mode.

    assign q = ram[addr];
endmodule
```

BYTE-ORIENTED FILE REGISTER OPERATIONS

PIC16F1826 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected	Notes
				MSB			LSB		
CONTROL OPERATIONS									
BRA	k	Relative Branch	2	11	001k	kkkk	kkkk		
BRW	-	Relative Branch with W	2	00	0000	0000	1011		
INHERENT OPERATIONS									
NOP	-	No Operation	1	00	0000	0000	0000		



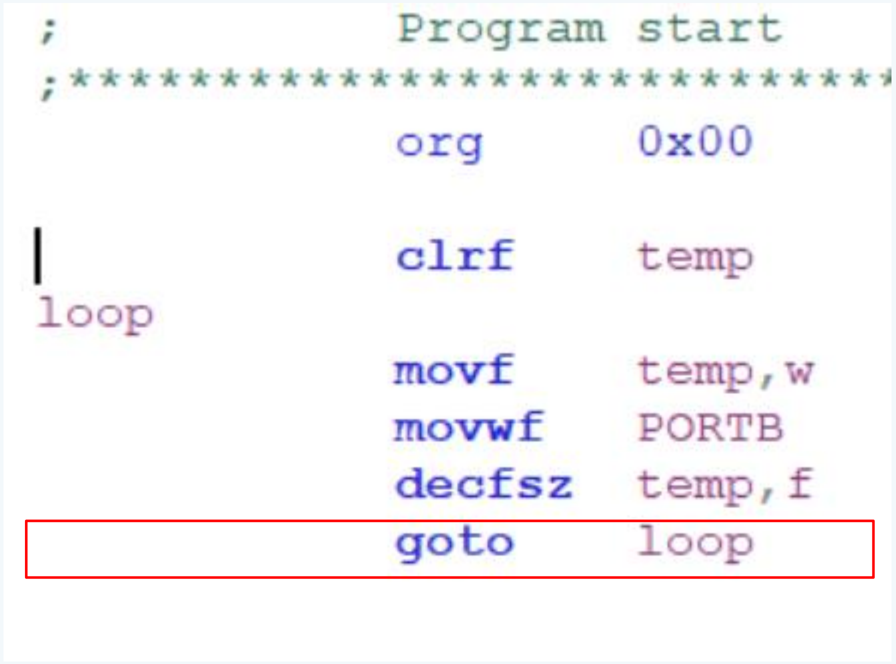
```
assign w_change = {3'b0, w_q};
assign k_change = {ir_out[8],ir_out[8],ir_out[8:0]};
```

Note

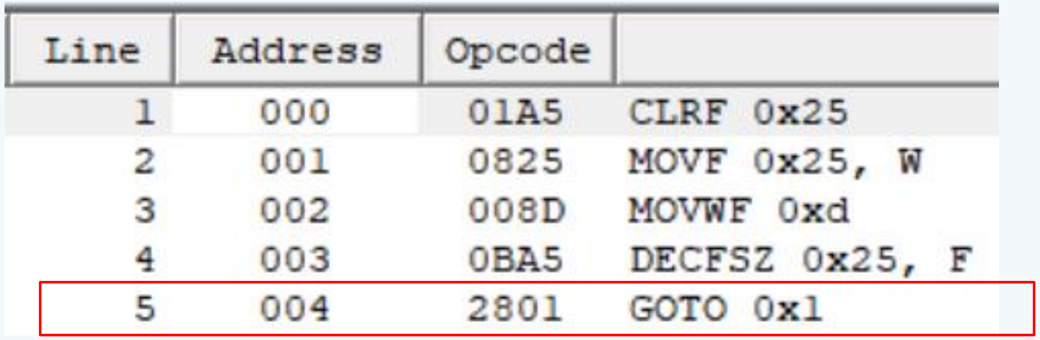
1：If the Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

2：If this instruction addresses an INDF register and the MSb of the corresponding FSR is set, this instruction will require one additional instruction cycle.

GOTO INSTRUCTION
絕對定址



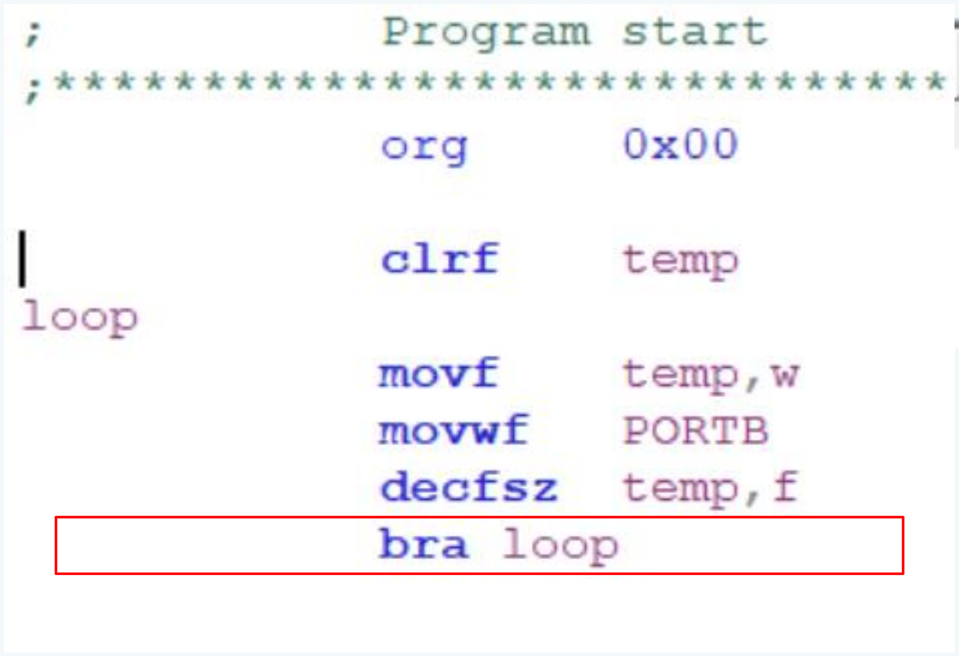
```
;                Program start
;*****************************
                 org        0x00

                 clrf       temp
loop
                 movf       temp,w
                 movwf      PORTB
                 decfsz     temp,f
                 goto       loop
```



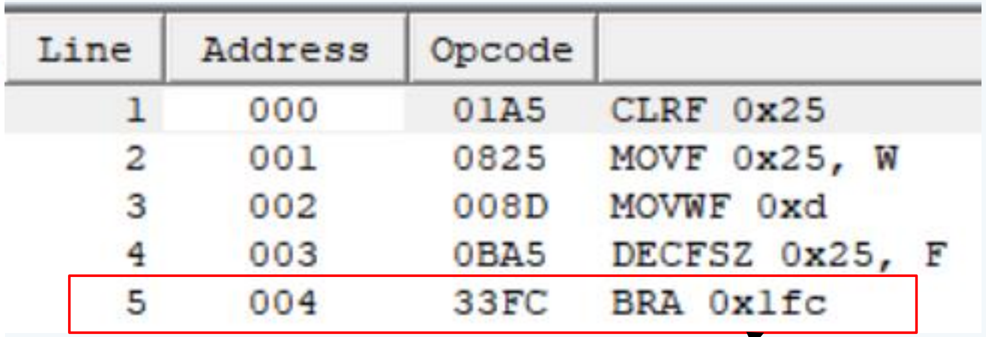
Line	Address	Opcode	
1	000	01A5	CLRF 0x25
2	001	0825	MOVF 0x25, W
3	002	008D	MOVWF 0xd
4	003	0BA5	DECFSZ 0x25, F
5	004	2801	GOTO 0x1

$$\boxed{PC <= 1}$$

BRA INSTRUCTION
相對定址



```
;               Program start
;*******************************
                org     0x00

                clrf    temp
loop
                movf    temp,w
                movwf   PORTB
                decfsz  temp,f
                bra loop
```

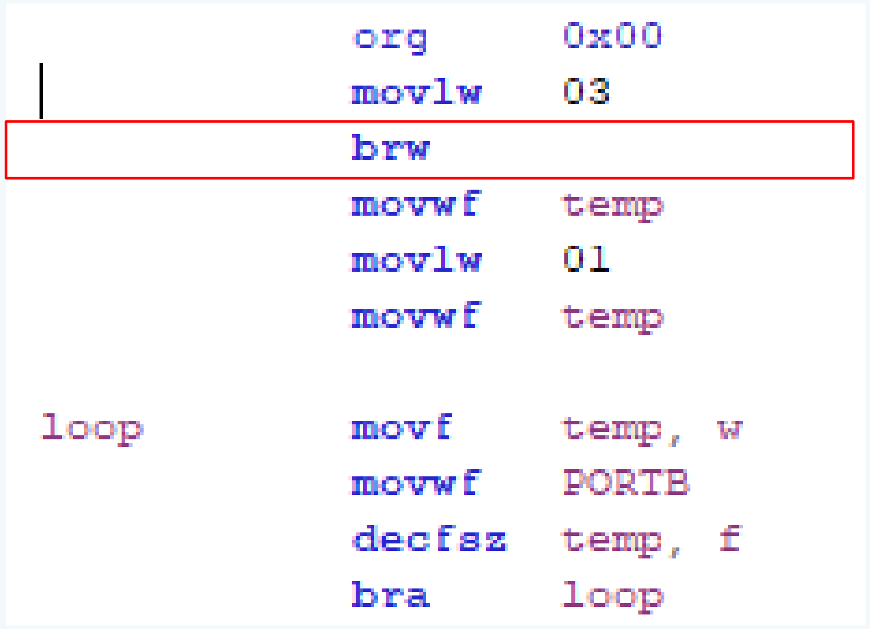


Line	Address	Opcode	
1	000	01A5	CLRF 0x25
2	001	0825	MOVF 0x25, W
3	002	008D	MOVWF 0xd
4	003	0BA5	DECFSZ 0x25, F
5	004	33FC	BRA 0x1fc

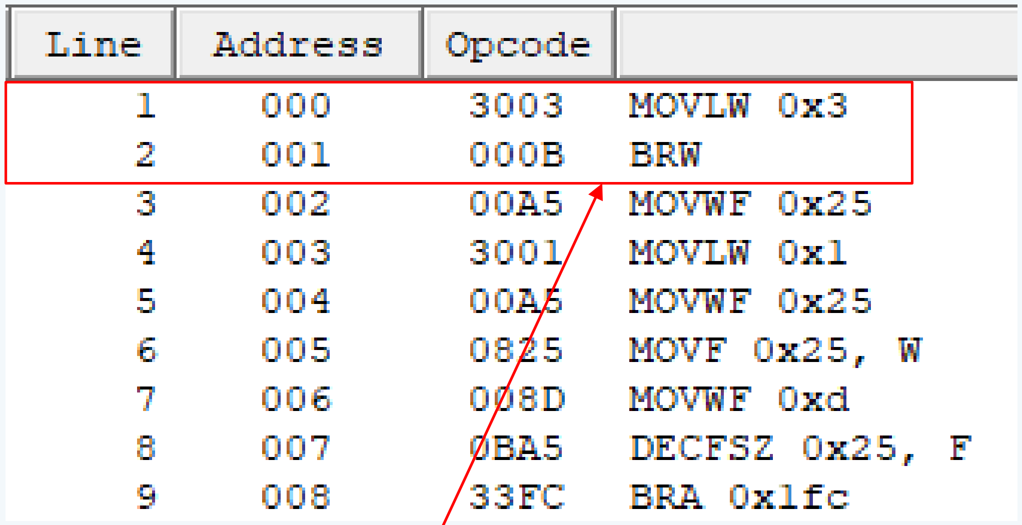
0x1fc = -4
PC = 005

PC <= 005 + 01fc
PC <= 005 – 4 = 1
PC <= 1

BRW INSTRUCTION
相對定址with w



```
                org       0x00
                movlw     03
                brw
                movwf     temp
                movlw     01
                movwf     temp

loop            movf      temp, w
                movwf     PORTB
                decfsz    temp, f
                bra       loop
```



Line	Address	Opcode	
1	000	3003	MOVLW 0x3
2	001	000B	BRW
3	002	00A5	MOVWF 0x25
4	003	3001	MOVLW 0x1
5	004	00A5	MOVWF 0x25
6	005	0825	MOVF 0x25, W
7	006	008D	MOVWF 0xd
8	007	0BA5	DECFSZ 0x25, F
9	008	33FC	BRA 0x1fc

w = 3
PC = 002

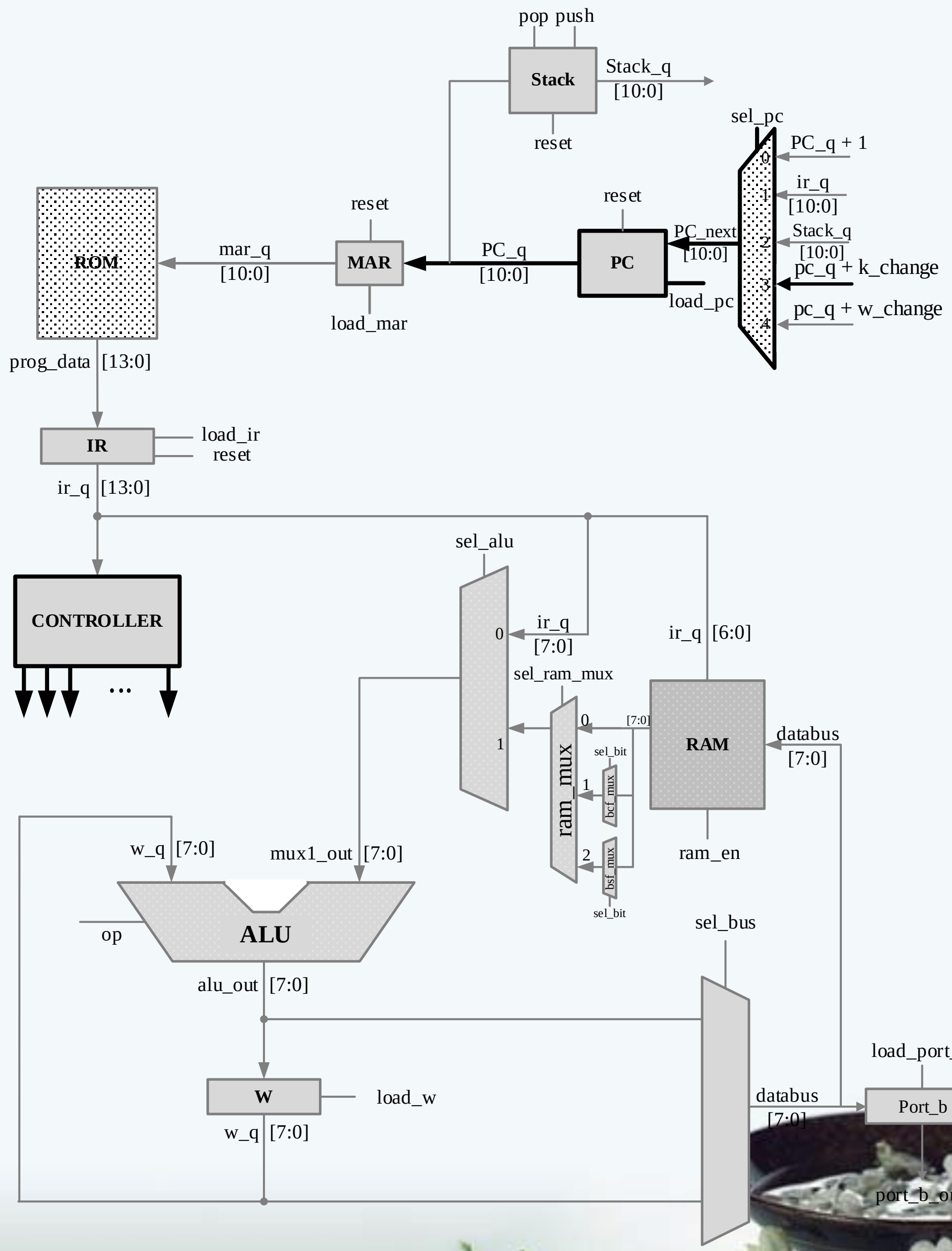
PC <= 002 + w
PC <= 002 + 3 = 5
PC <= 5

BRA

T4

狀態	動作	控制訊號

T_4	pc_q <= pc_q + k_change	load_pc = 1
		sel_pc = 3

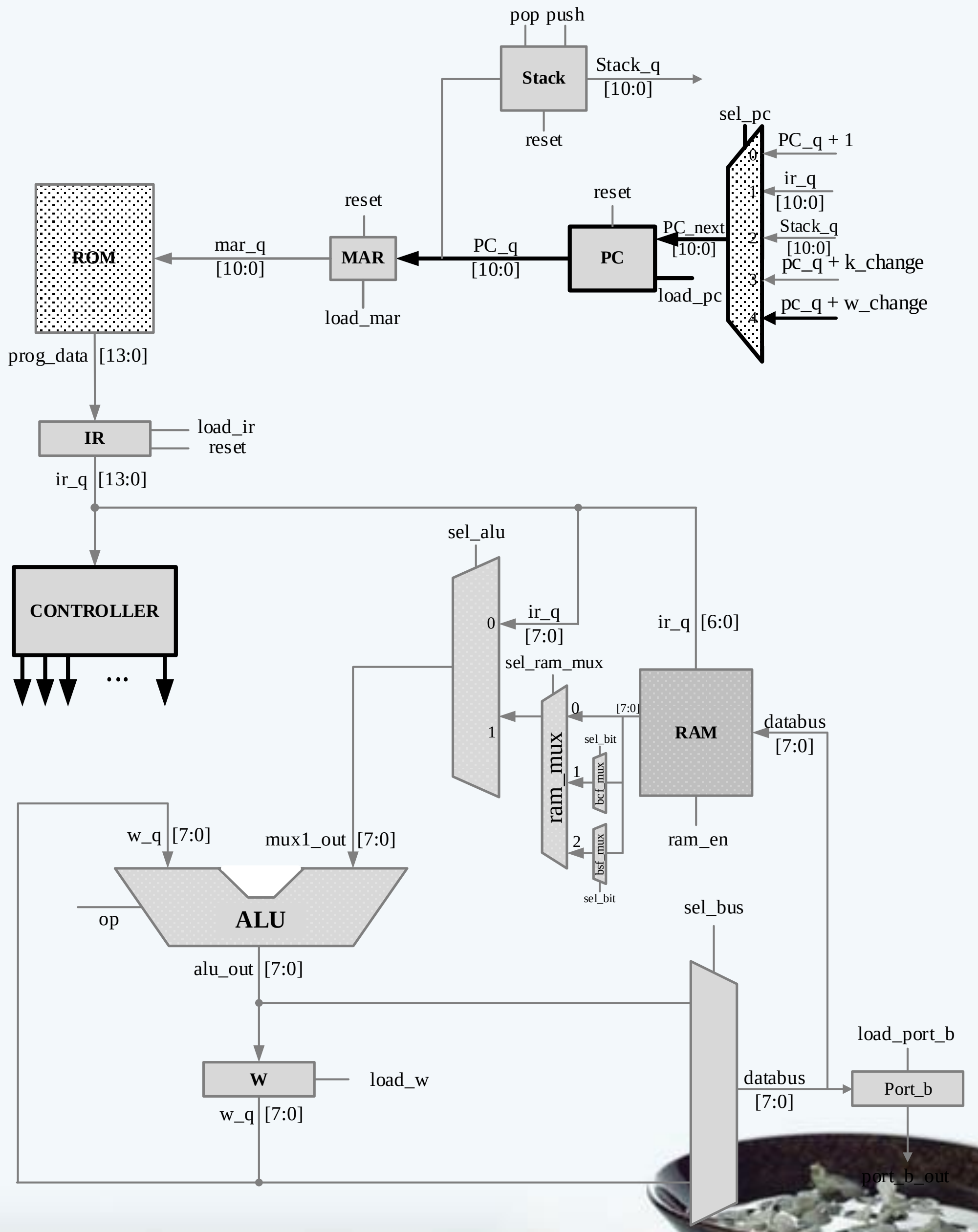


BRW

T4

狀態	動作	控制訊號

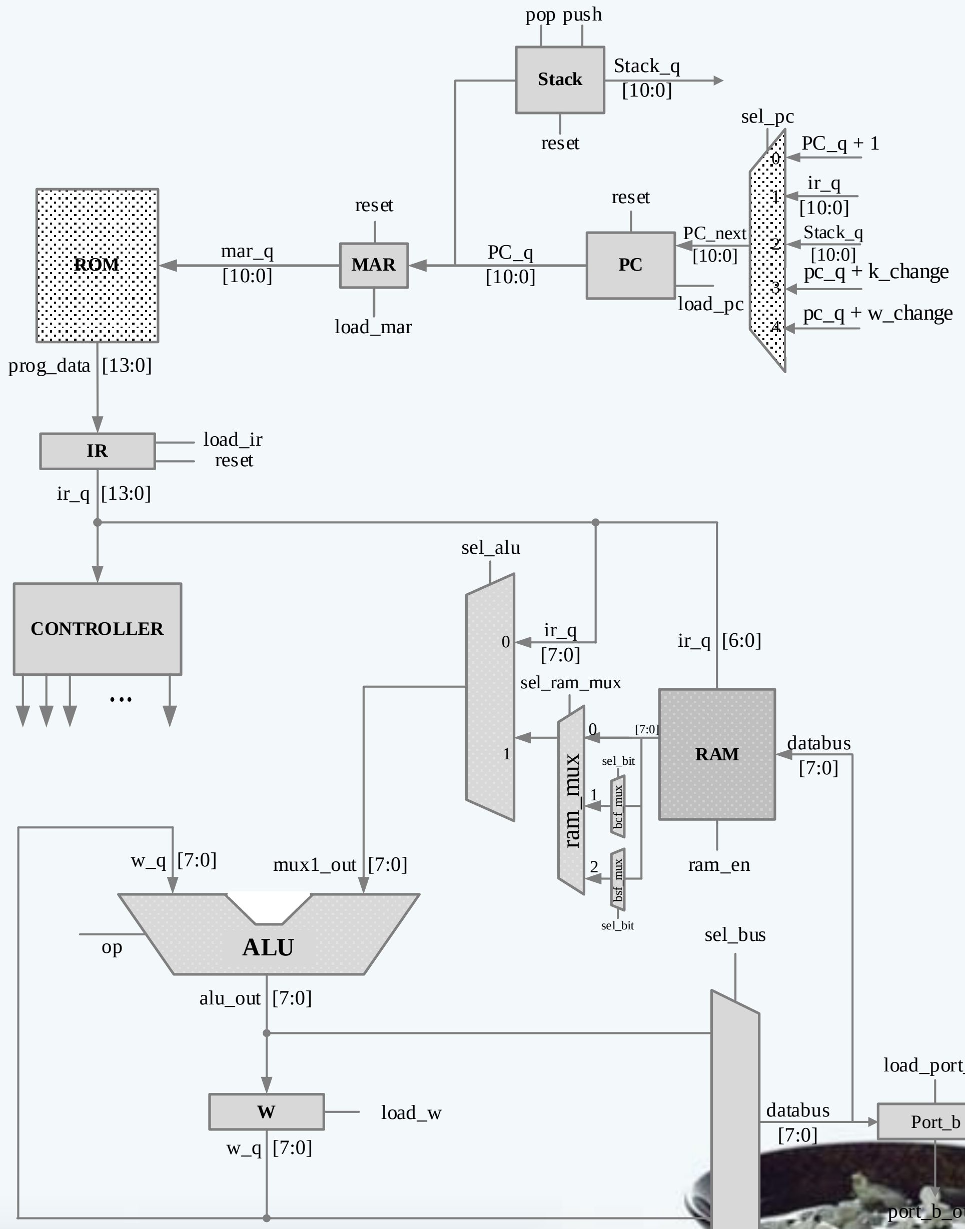
T_4	pc_q <= pc_q + w_change	load_pc = 1
		sel_pc = 4



NOP

T4

狀態	動作	控制訊號
T_4	NONE	NONE

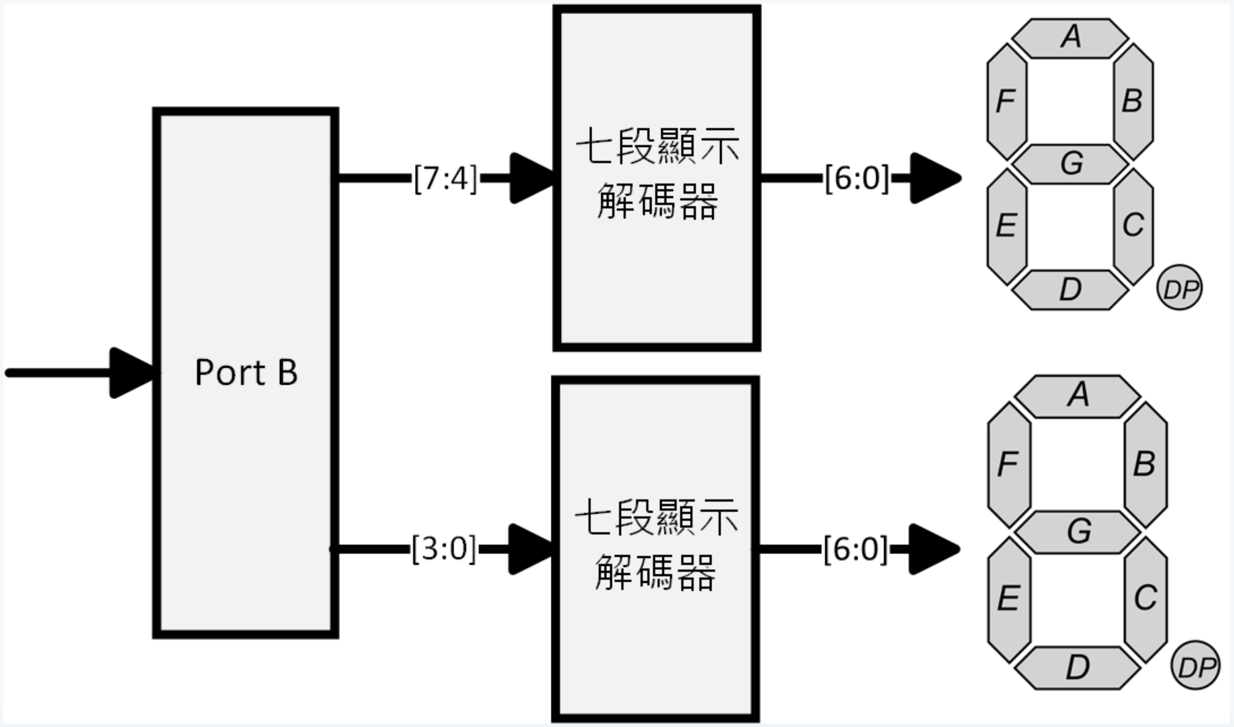


回家作業一

設計組合語言，用來顯示時鐘的「秒」，運行於PIC MCU上。由00數到59再歸00，並且不斷重複，將結果輸出於port_ B上並以16進位顯示(10進位不算分)

請用BRA或BRW指令代替GOTO指令

加分：在課堂實作或補強時將此架構燒錄到DE0上的顯示結果給助教檢查，即可加分。兩個七段顯示器分別顯示時鐘的秒之高低位數。接法如下



回家作業二

用MPLAB設計一個Rom，使0x21和0x22兩個位址的16進制(用10進位顯示不算分)分別表示時鐘的分及秒，即0x22(秒)的16進制會由1數到59後歸零，每當0x22(秒)歸零0x21(分)就會加1