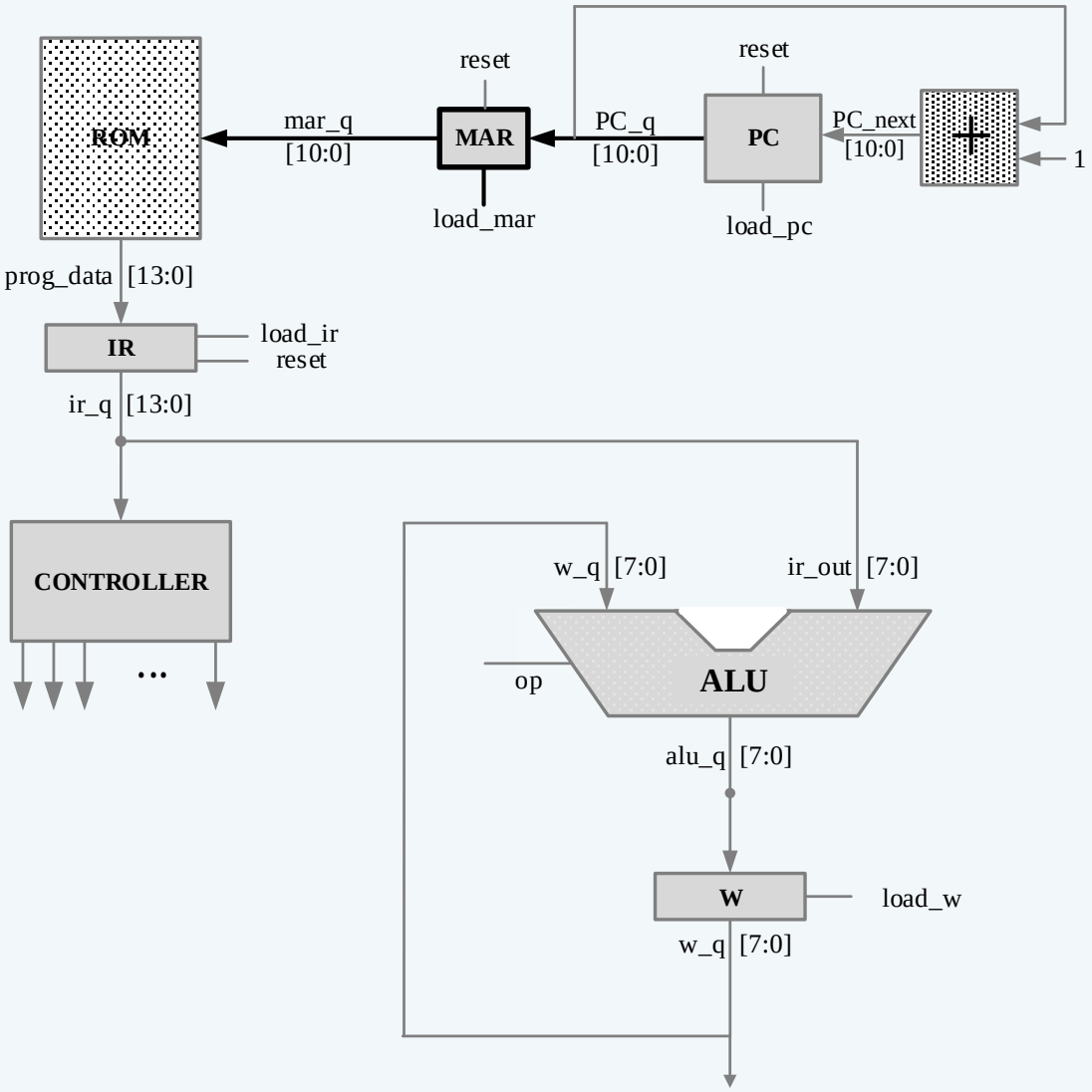
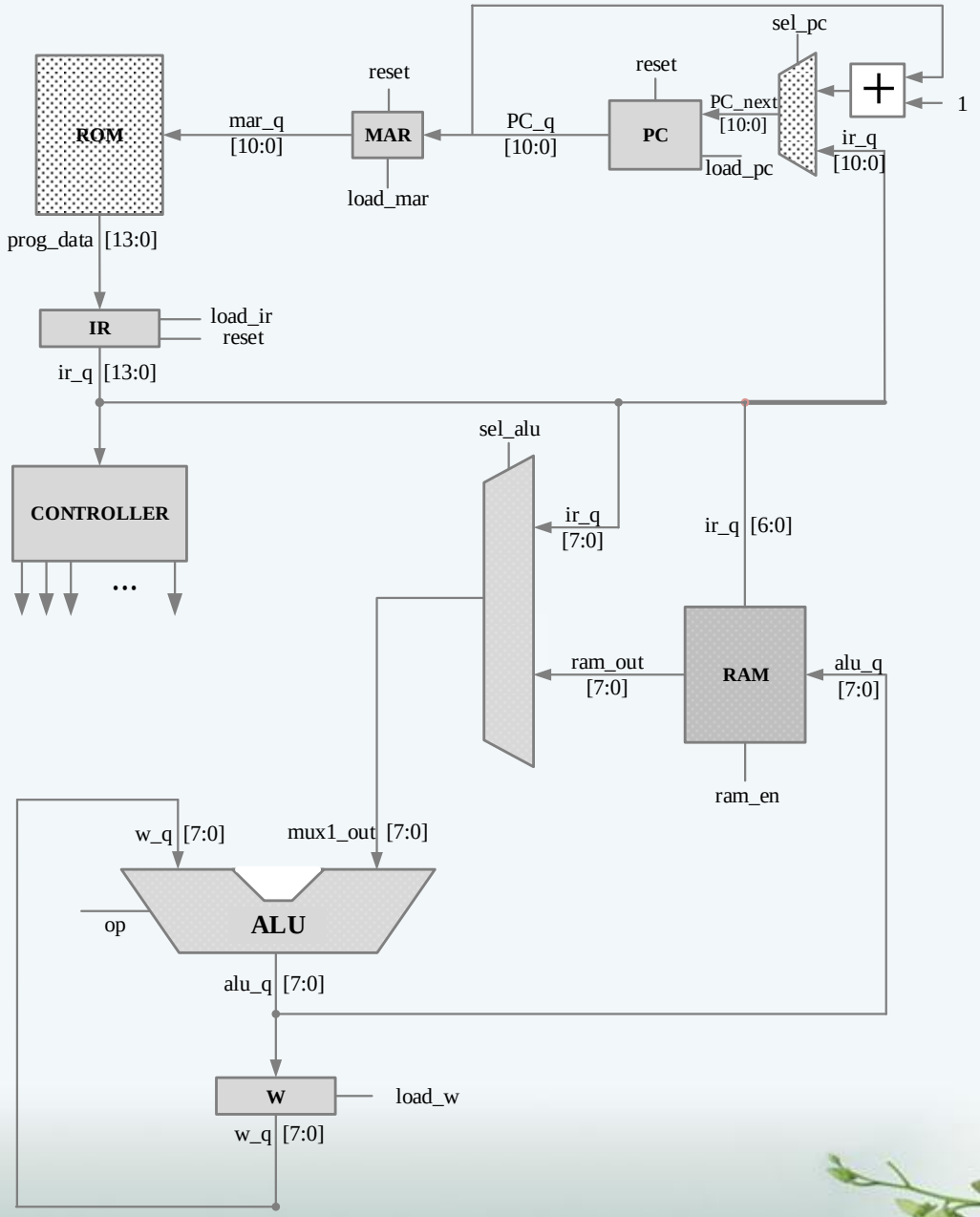
暫存器定址

舊架構圖


ROM
mar_q
[10:0]
reset
MAR
load_mar
PC_q
[10:0]
reset
PC
load_pc
PC_next
[10:0]
+
1
prog_data [13:0]
IR
load_ir
reset
ir_q [13:0]
CONTROLLER
...
w_q [7:0]
ir_out [7:0]
op
ALU
alu_q [7:0]
W
load_w
w_q [7:0]

新架構圖


sel_pc
reset
reset
ROM
mar_q
[10:0]
MAR
PC_q
[10:0]
PC
PC_next
[10:0]
1
load_pc
ir_q
[10:0]
load_mar
prog_data [13:0]
IR
load_ir
reset
ir_q [13:0]
sel_alu
CONTROLLER
ir_q
[7:0]
ir_q [6:0]
...
ram_out
[7:0]
RAM
alu_q
[7:0]
ram_en
w_q [7:0]
mux1_out [7:0]
op
ALU
alu_q [7:0]
W
load_w
w_q [7:0]

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。

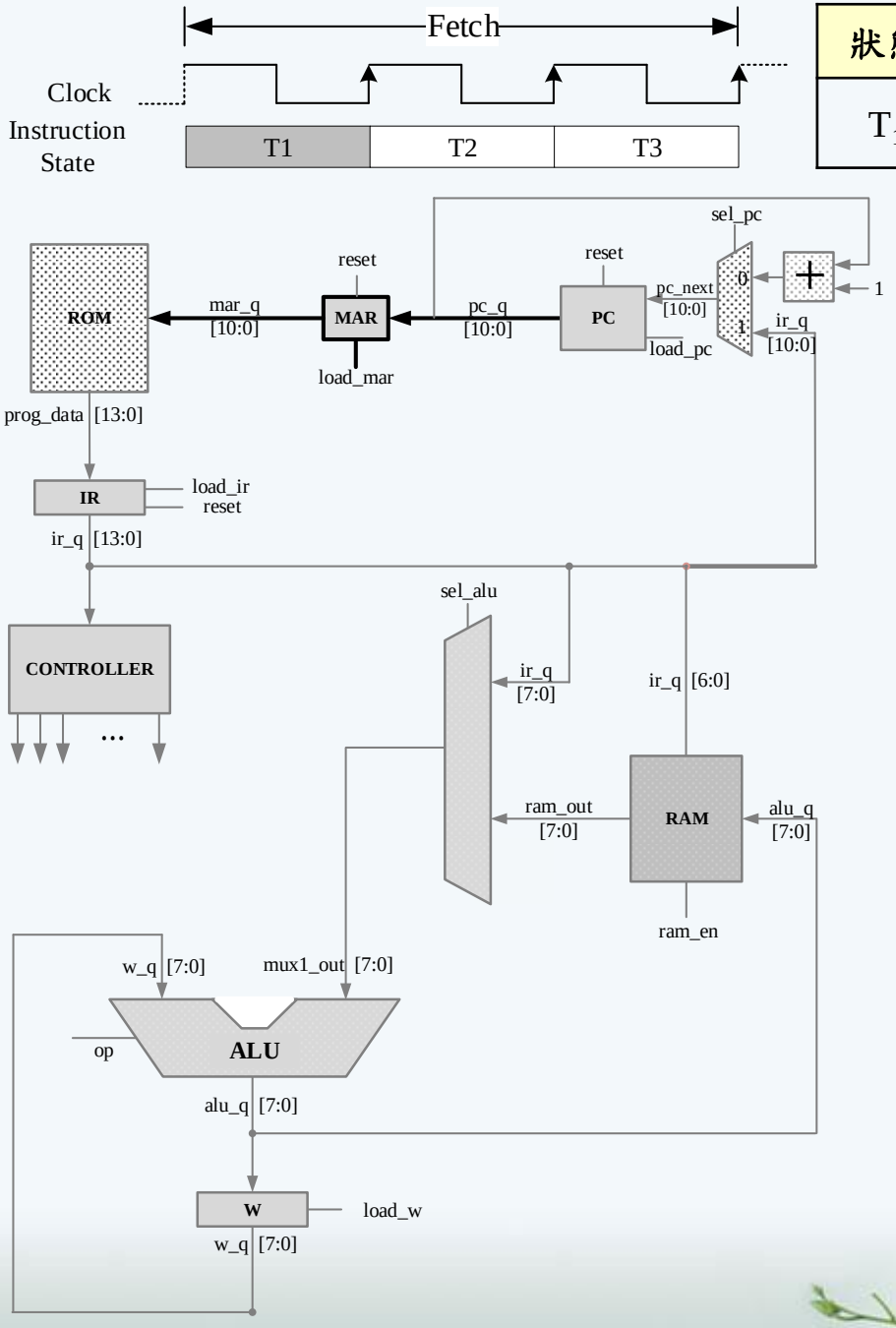
- 兩個時間週期：
  Byte-oriented File Register Operations、Bit-oriented File Register Operations、
  Bit-oriented Skip Operations。

- 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。

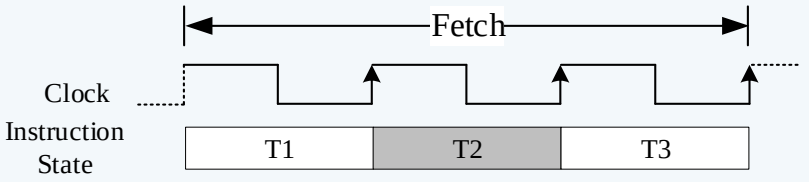
狀態	動作	控制訊號
T_1	MAR←PC	load_mar
T_2	PC←PC+1	sel_pc; load_pc
T_3	IR←ROM[MAR]	load_ir

Fetch T1

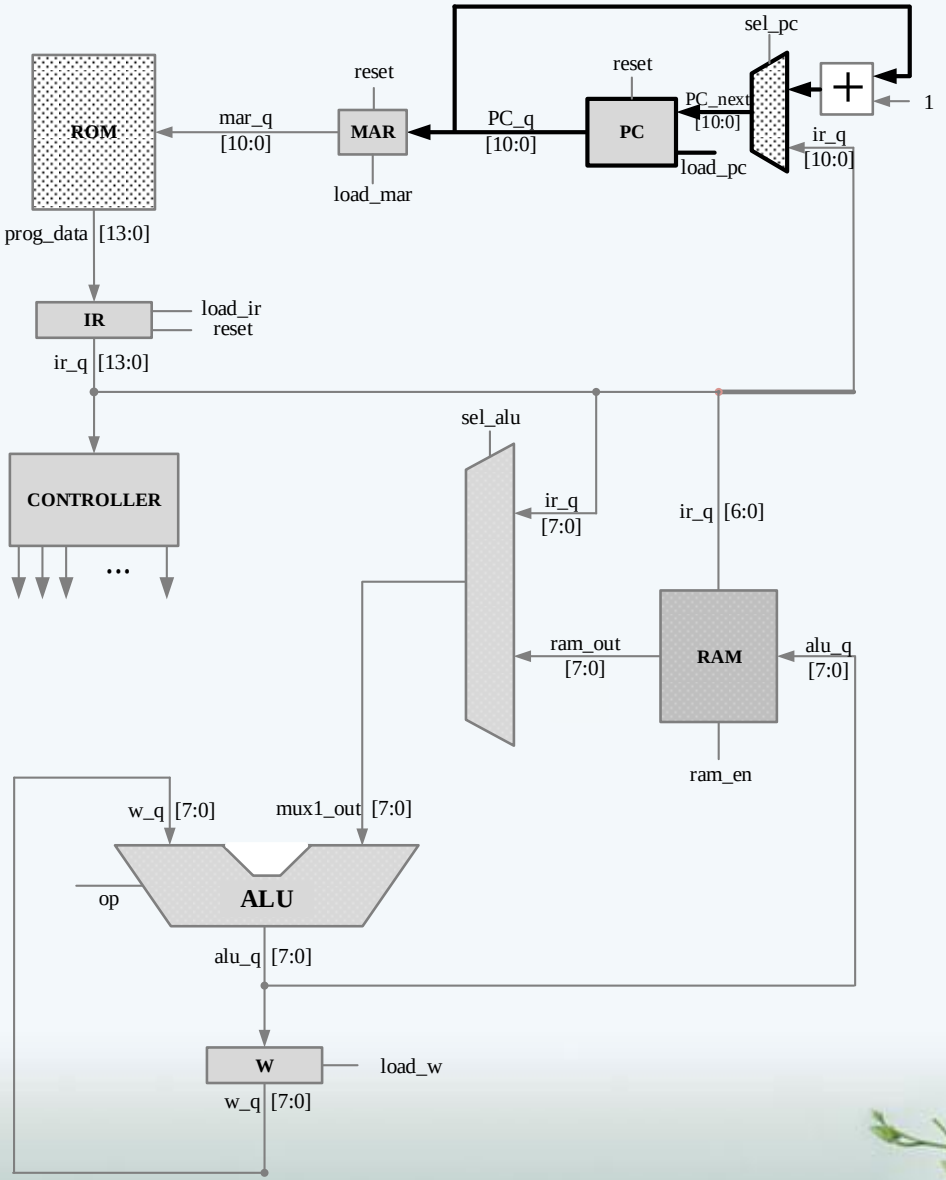


狀態	動作	控制訊號
T_1	MAR←PC	load_mar

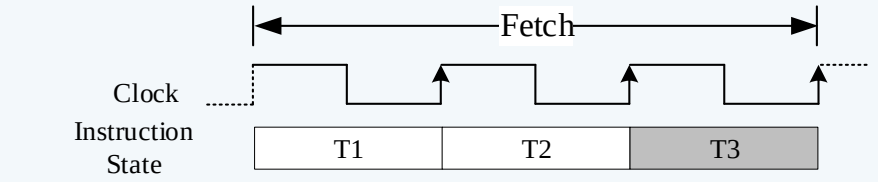
Fetch T2



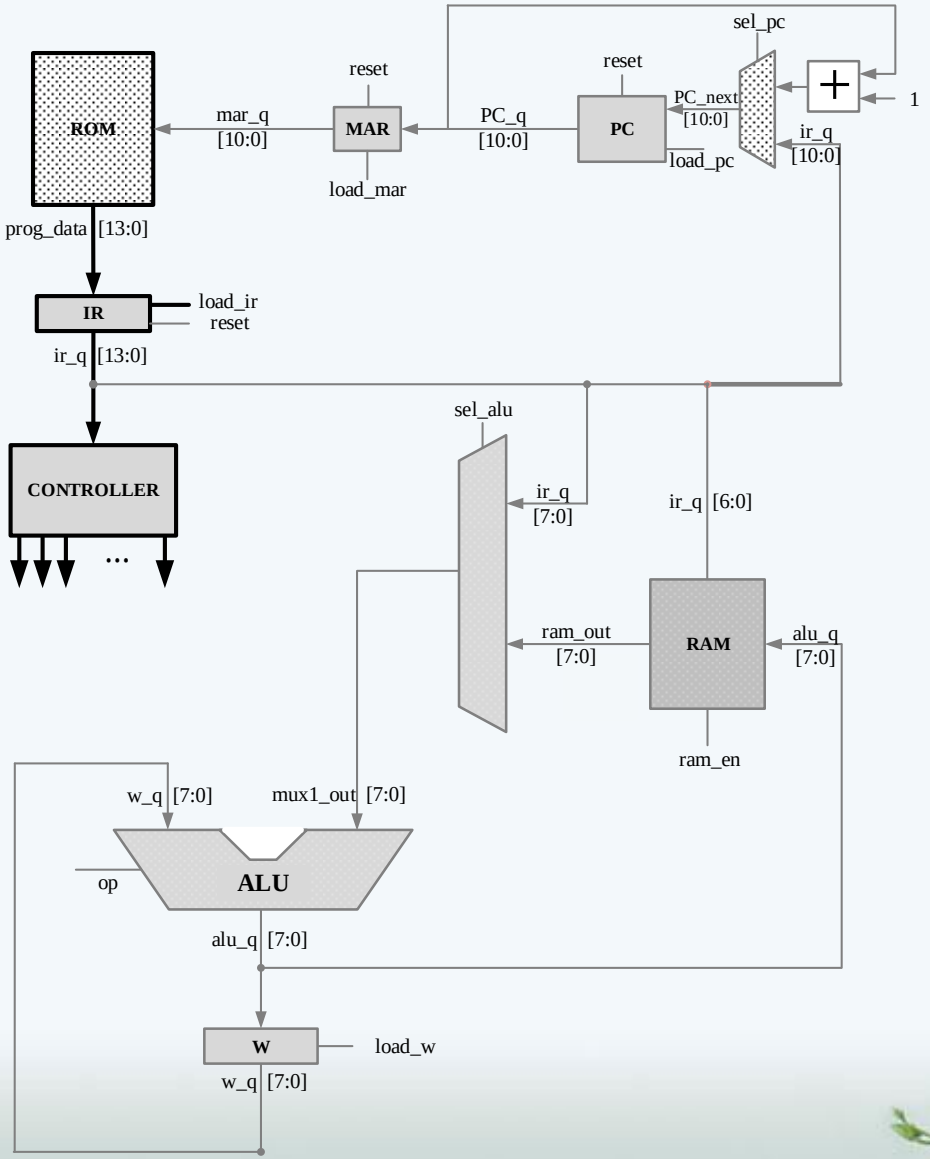
狀態	動作	控制訊號
T_2	PC←PC+1	sel_pc; load_pc



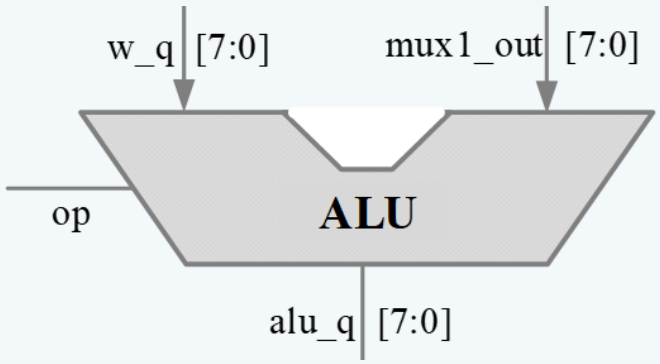
Fetch T3



狀態	動作	控制訊號
T_3	IR←ROM[MAR]	load_ir

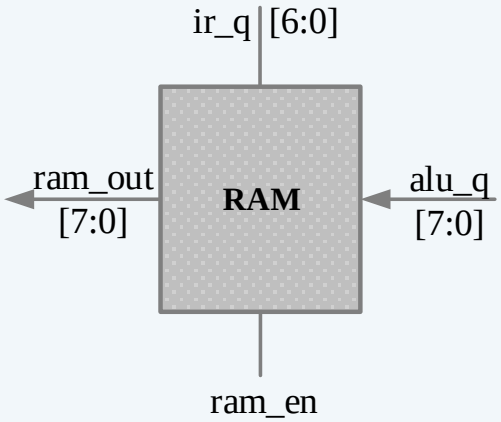


ALU




```
//ALU
always_comb
begin
    case(op)
        0: alu_q = mux1_out + w_q;
        1: alu_q = mux1_out - w_q;
        2: alu_q = mux1_out & w_q;
        3: alu_q = mux1_out | w_q;
        4: alu_q = mux1_out ^ w_q;
        5: alu_q = mux1_out;
        6: alu_q = mux1_out + 1;
        7: alu_q = mux1_out - 1;
        8: alu_q = 0;
        9: alu_q = ~mux1_out;
        default: alu_q = mux1_out + 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		
BYTE-ORIENTED FILE REGISTER OPERATIONS (1)									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C, DC, Z	2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	–	Clear W	1	00	0001	0000	00xx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	2
跳躍指令									
GOTO	f		1	10	1ffff	ffff	ffff		

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.

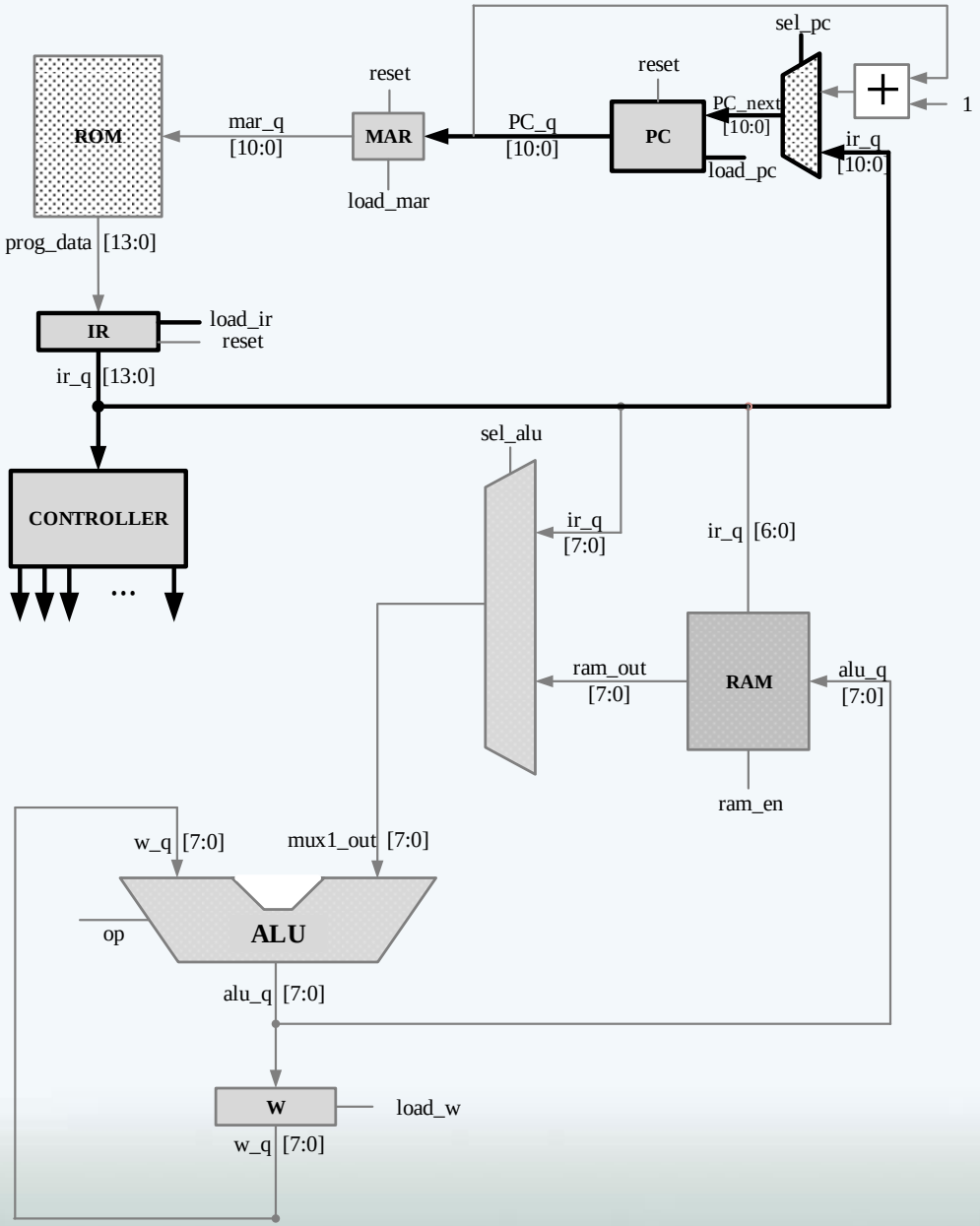
2：If this instruction addresses an INDF register and the MSb of the corresponding FSR is set, this instruction will require one additional instruction cycle.

TABLE 29-1: OPCODE FIELD DESCRIPTIONS

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1). The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1.
n	FSR or INDF number. (0-1)
mm	Pre-post increment-decrement mode selection

GOTO：跳到指定的指令

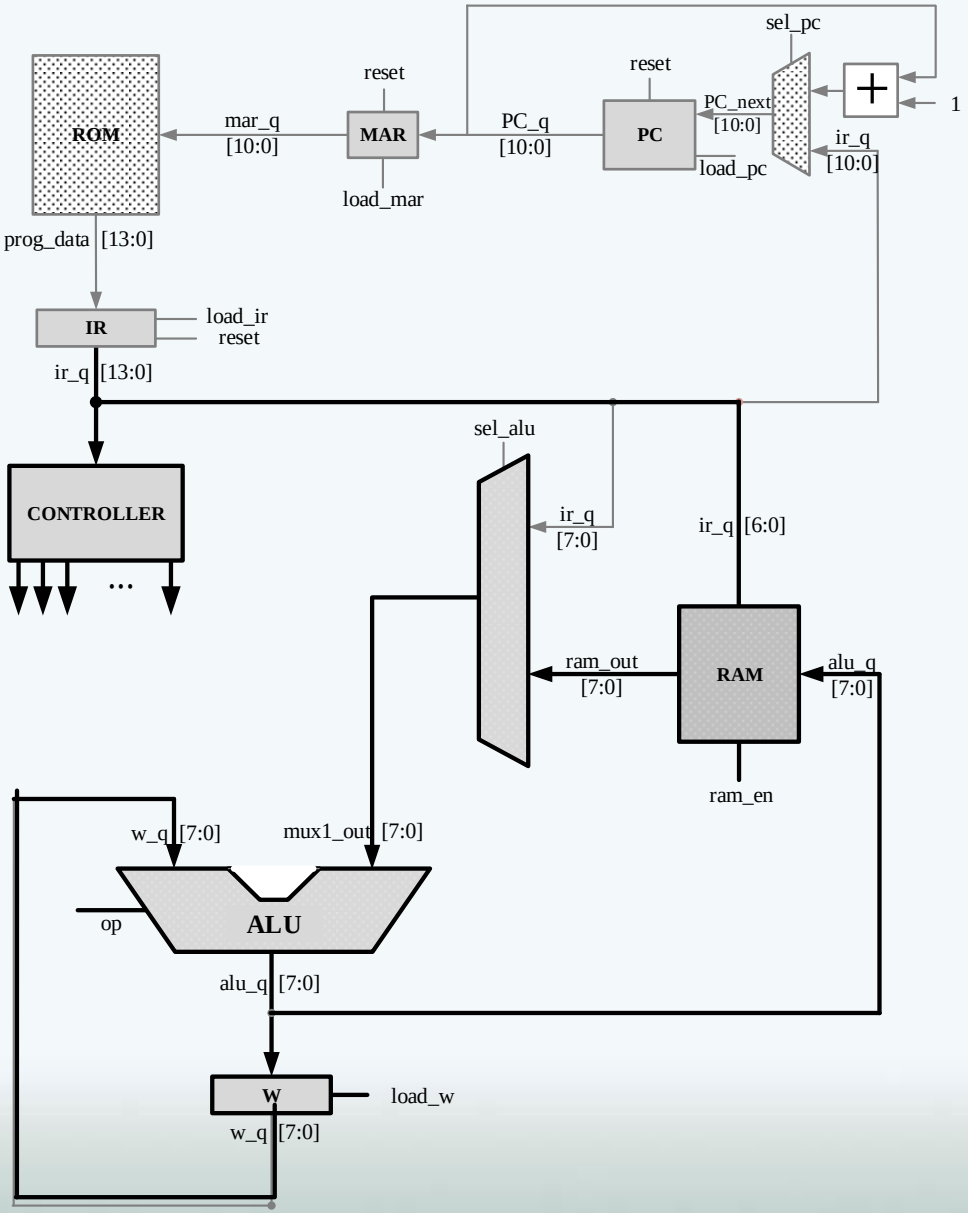
10 1fff ffff ffff



狀態	動作	控制訊號
T_4	PC_q ← ir_q[10:0]	sel_pc=1 load_pc = 1;
T_5	無動作	無
T_6	無動作	無

ADDWF：W和F相加

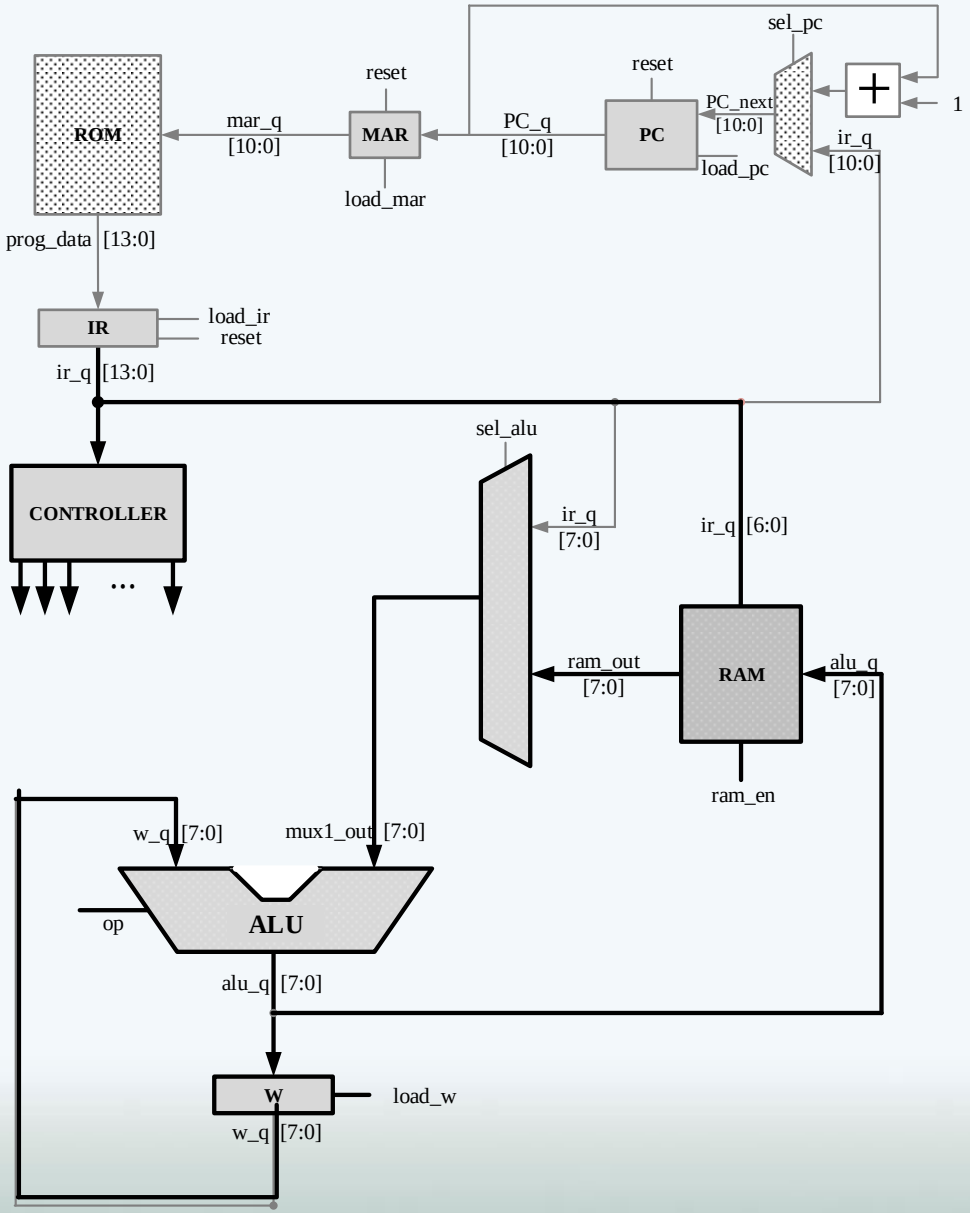
00 0111 dfff ffff



狀態	動作	控制訊號
T_4	**d = 0：** W← W + F **d = 1：** F ← W + F	op=0 sel_alu = 1 **d = 0：** load_w=1 **d = 1：** ram_en=1
T_5	無動作	無
T_6	無動作	無

ANDWF：W和F做邏輯與運算

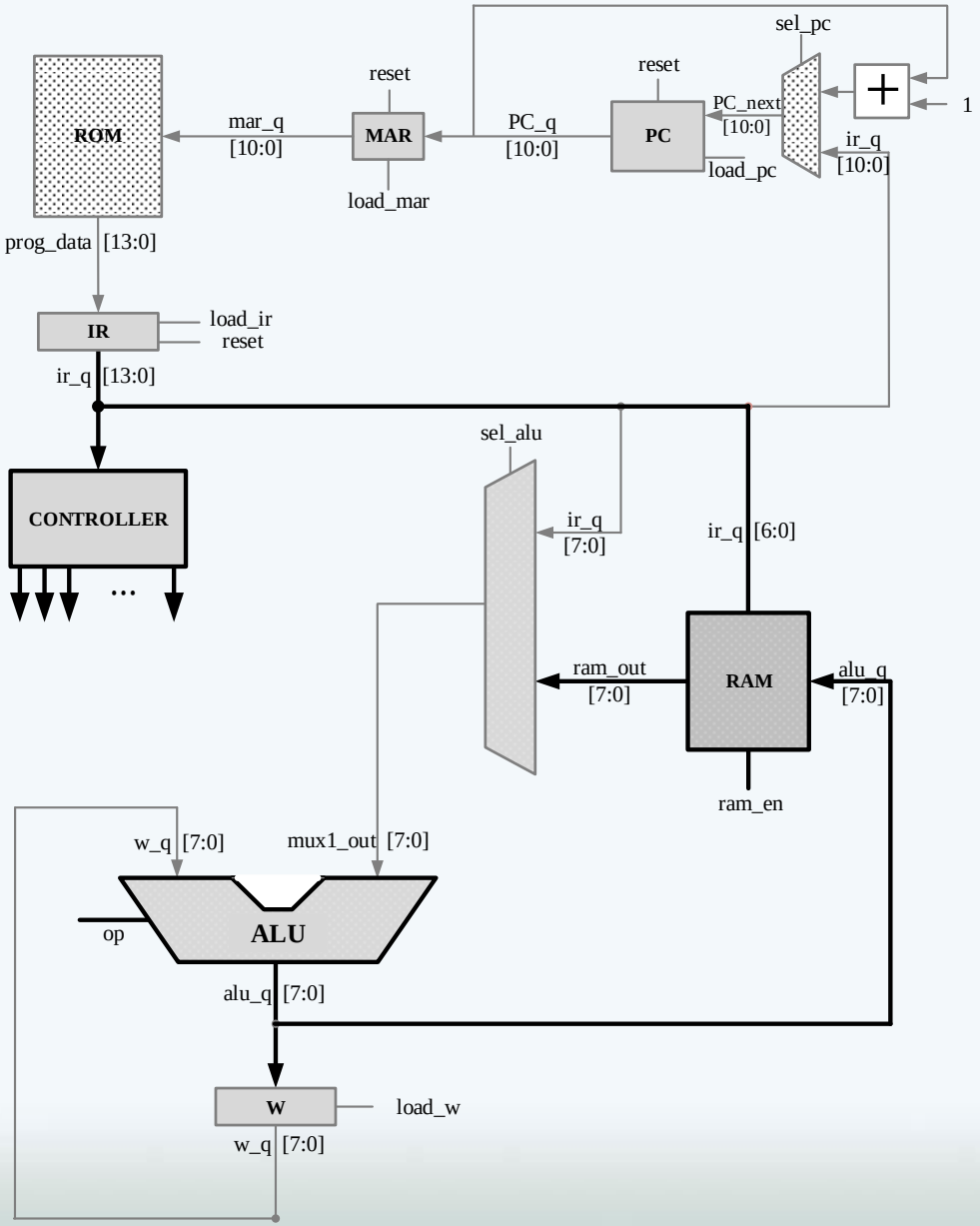
00 0101 dfff ffff



狀態	動作	控制訊號
T_4	**d = 0：** W← W & F **d = 1：** F ← W & F	op=2 sel_alu = 1 **d = 0：** load_w=1 **d = 1：** ram_en=1
T_5	無動作	無
T_6	無動作	無

CLRF：將F清零

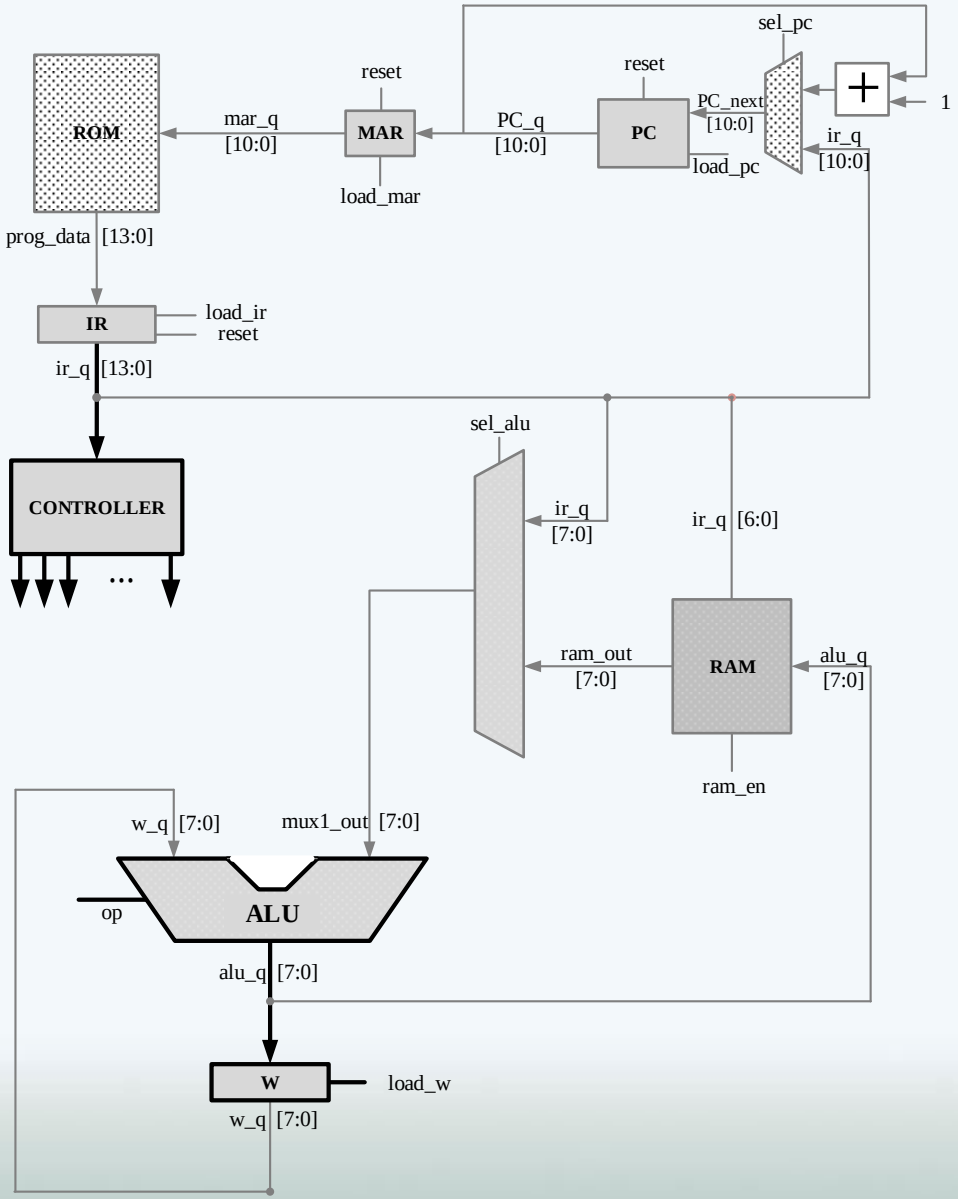
00 0001 1fff ffff



狀態	動作	控制訊號
T_4	F ← 0	op=8 ~~sel_alu = 1~~ ram_en=1
T_5	無動作	無
T_6	無動作	無

CLRW：將W清零

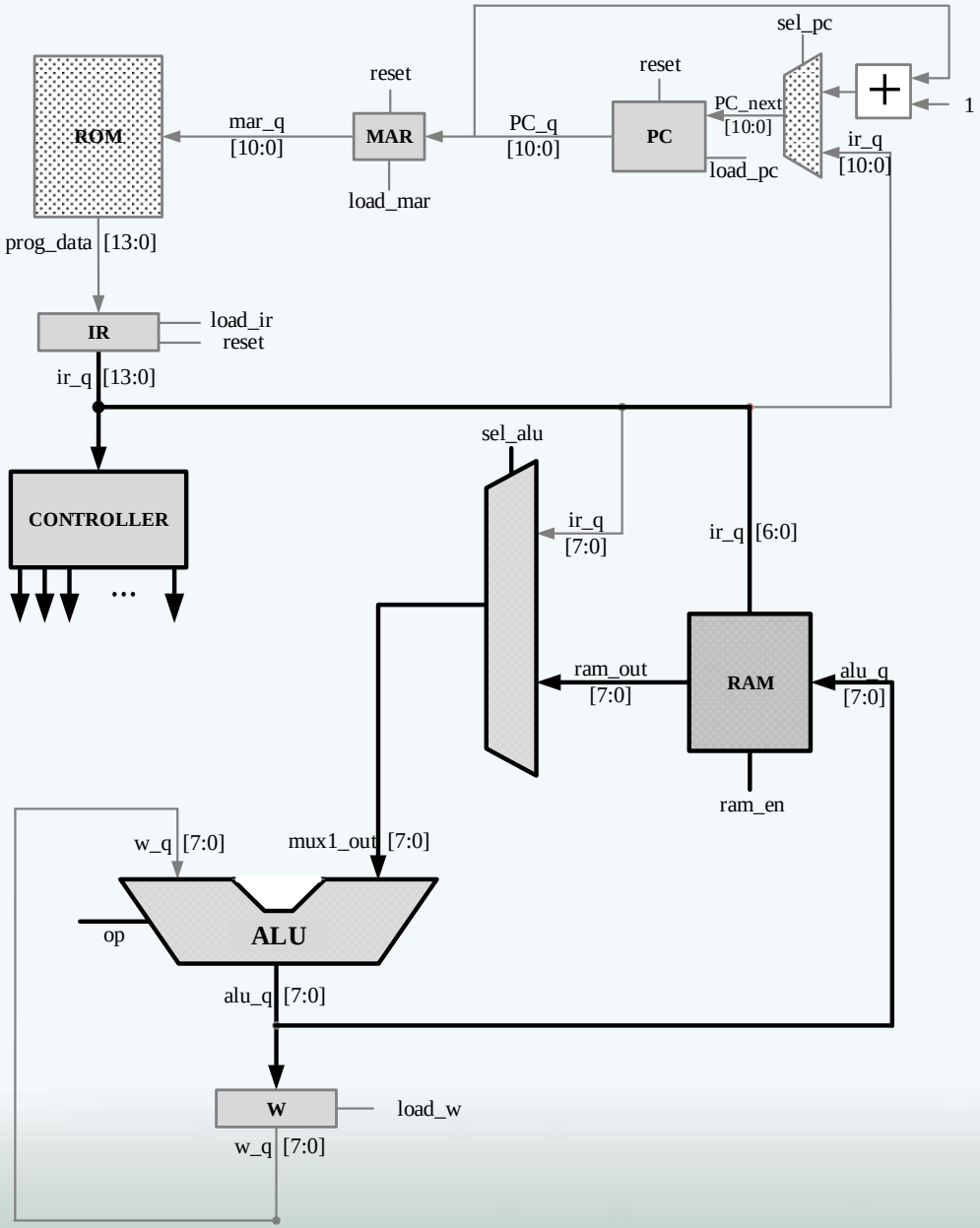
00 0001 0000 00xx



狀態	動作	控制訊號
T_4	W ← 0	op=8 ~~sel_alu = 1~~ load_w=1
T_5	無動作	無
T_6	無動作	無

COMF：將F反向

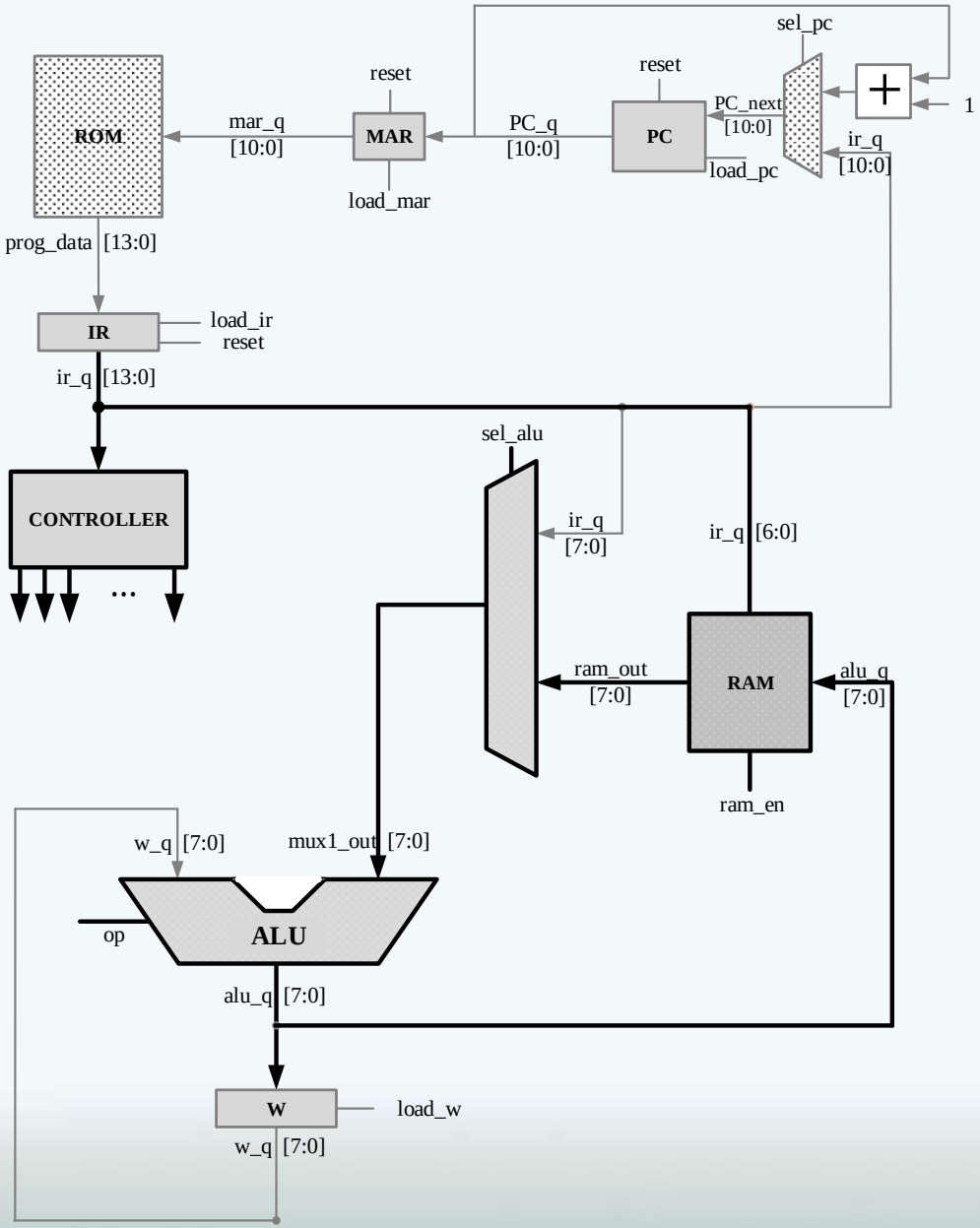
00 1001 dfff ffff



狀態	動作	控制訊號
T_4	**d = 0：** W← ~F **d = 1：** F ← ~F	op=9 sel_alu = 1 **d = 0：** load_w=1 **d = 1：** ram_en=1
T_5	無動作	無
T_6	無動作	無

DECF：F遞減1

00 0011 dfff ffff



狀態	動作	控制訊號
T_4	**d = 0：** W← F - 1 **d = 1：** F ← F - 1	op=7 sel_alu = 1 **d = 0：** load_w=1 **d = 1：** ram_en=1
T_5	無動作	無
T_6	無動作	無

上課實作

DEMO的時候要燒板子，將W接到LED上

ASM code

```
#include     <pl6Lf1826.inc>      ; Include file locate a
;

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

             clrf  temp            ; //ram[25]<=0
             clrw                  ; //w<=0
             movlw 7               ; //w<=6
             addwf temp,1          ; //ram[25]<=ram[25]+w
             addwf temp,0          ; //w<=ram[25]+w
             goto $
             end
```

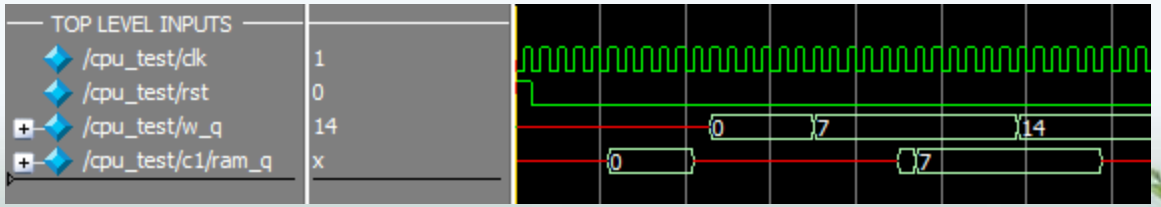
Program Rom

```
module Program_Rom(
    output logic [13:0] Rom_data_out,
    input [10:0] Rom_addr_in
);
//---------
    logic [13:0] data;

    always_comb
    begin
        case (Rom_addr_in)
            11'h0:  data = 14'h01A5;    //CLRF          ram[25]=0
            11'h1:  data = 14'h0103;    //CLRW          w=0
            11'h2:  data = 14'h3007;    //MOVLW 7       w=7
            11'h3:  data = 14'h07A5;    //ADDLW 0x25,1  ram[25]=7
            11'h4:  data = 14'h0725;    //ADDLW 0x25,0  w=14
            11'h5:  data = 14'h2805;    //GOTO  5
            //這兩行為MPLAB清除暫存器的指令，不用管
            11'h6:  data = 14'h3400;
            11'h7:  data = 14'h3400;
            default:data = 14'h0;
        endcase
    end
    assign Rom_data_out = data;

endmodule
```

模擬結果



回家作業

ASM code

```
#include     <pl6Lf1826.inc>      ; Include file locate at
;

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

             clrf   temp           ; //ram[25]<=0
             clrw                  ; //w<=0
             movlw 6               ; //w<=6
             addwf temp,1          ; //ram[25]<=ram[25]+w
             movlw 5               ; //w<=5
             addwf temp,0          ; //w<=ram[25]+w
             addlw 2               ; //w<=w + 2
             andwf temp,1          ; //ram[25]<=ram[25]&w
             decf   temp           ; //ram[25]<=ram[25]-1
             comf   temp           ; //ram[25]<=~ram[25]
             goto $
             end
```

Program Rom

```
module Program_Rom(
    output logic [13:0] Rom_data_out,
    input [10:0] Rom_addr_in
);
//---------
    logic [13:0] data;

    always_comb
    begin
        case (Rom_addr_in)
            11'h0:  data = 14'h01A5;    //CLRF          ram[25]=0
            11'h1:  data = 14'h0103;    //CLRW          w=0
            11'h2:  data = 14'h3006;    //MOVLW 6       w=6
            11'h3:  data = 14'h07A5;    //ADDLW 0x25,1  ram[25]=6
            11'h4:  data = 14'h3005;    //MOVLW 5       w=5
            11'h5:  data = 14'h0725;    //ADDWF 0x25,0  w=11
            11'h6:  data = 14'h3E02;    //ADDLW 2       w=13
            11'h7:  data = 14'h05A5;    //ANDWF 0x25,1  ram[25]=4
            11'h8:  data = 14'h03A5;    //DECF  0x25    ram[25]=3
            11'h9:  data = 14'h09A5;    //COMF  0x25    ram[25]=252
            11'ha:  data = 14'h280A;    //GOTO  8
            //這兩行為MPLAB清除暫存器的指令，不用管
            11'hb:  data = 14'h3400;
            11'hc:  data = 14'h3400;
            default:data = 14'h0;
        endcase
    end
    assign Rom_data_out = data;

endmodule
```