十速科技股份有限公司
tenx technology inc.

Advance Information

4BIT series MCU

指令說明

Tenx reserves the right to change or discontinue this product without notice.

十速科技股份有限公司
TEL: 886-2-82571700 FAX: 886-2-82571701
886-3-5737682 886-3-5737632
網址: www.tenx.com.tw

CONTENTS

第一章　指令快速索引表

OPCODE	指令類型	指令長度	指令週期 (machine cycle)	記憶體傳輸位元數 (bits)
ADC	算數/邏輯運算	1 word	4	4(/16*1)
ADCH	算數/邏輯運算	1 word	4	16
ADCI	算數/邏輯運算	1 word	4	4
ADCM	算數/邏輯運算	1 word	4	4
ADCMH	算數/邏輯運算	1 word	4	16
ADD	算數/邏輯運算	1 word	4	4(/16*1)
ADDH	算數/邏輯運算	1 word	4	16
ADDI	算數/邏輯運算	1 word	4	4
ADDM	算數/邏輯運算	1 word	4	4
ADDMH	算數/邏輯運算	1 word	4	16
ADJ	系統功能	1 word	4	
ADN	算數/邏輯運算	1 word	4	4(/16*1)
ADNH	算數/邏輯運算	1 word	4	16
ADNI	算數/邏輯運算	1 word	4	4
ADNM	算數/邏輯運算	1 word	4	4
ADNMH	算數/邏輯運算	1 word	4	16
ALM	週邊功能	1 word	4	
AND	算數/邏輯運算	1 word	4	4(/16*1)
ANDH	算數/邏輯運算	1 word	4	16
ANDI	算數/邏輯運算	1 word	4	4
ANDM	算數/邏輯運算	1 word	4	4
ANDMH	算數/邏輯運算	1 word	4	16
CAC	跳躍/呼叫	multi-words	4 or 8	
CALL	跳躍/呼叫	1 word	4	
CLPG	RAM page 設定	1 word	4	
CPHL	跳躍/呼叫	1 word	4	
CPHLH	跳躍/呼叫	2 words	8	
CPZR	跳躍/呼叫	1 word	4	
CPZRH	跳躍/呼叫	2 words	8	
DAA	算數/邏輯運算	1 word	4	4(/16*1)
DAAH	算數/邏輯運算	1 word	4	16
DAS	算數/邏輯運算	1 word	4	4(/16*1)
DASH	算數/邏輯運算	1 word	4	16
DEC*	算數/邏輯運算	1 word	4	4(/16*1)
DECH*	算數/邏輯運算	1 word	4	16

OPCODE	指令類型	指令長度	指令週期 (machine cycle)	記憶體傳輸位元數 (bits)
DISTM	Timer	1 word	4	
ELC	週邊功能	1 word	4	
ELZ	RAM page 設定	1 word	4	
EOR	算數/邏輯運算	1 word	4	4(/16*1)
EORH	算數/邏輯運算	1 word	4	16
EORI	算數/邏輯運算	1 word	4	4
EORM	算數/邏輯運算	1 word	4	4
EORMH	算數/邏輯運算	1 word	4	16
ERX	RAM page 設定	1 word	4	
ERY	RAM page 設定	1 word	4	
FAST	系統功能	1 word	4	
FRQ	系統功能	1 word	4	4
FRQX				8
HALT	系統功能	1 word	4	
IDC	索引暫存器存取	1 word	4	
IDC8	索引暫存器存取	1 word	4	
IDCH	索引暫存器存取	1 word	4	
INC*	算數/邏輯運算	1 word	4	4(/16*1)
INCH*	算數/邏輯運算	1 word	4	16
IPA	I/O Port	1 word	4	4(/16*2)
IPB	I/O Port	1 word	4	4
IPC	I/O Port	1 word	4	4
IPD	I/O Port	1 word	4	4
IPE	I/O Port	1 word	4	4
JAC	跳躍/呼叫	multi-words	4 or 8	
JB0	跳躍/呼叫	1 word	4	
JB1	跳躍/呼叫	1 word	4	
JB2	跳躍/呼叫	1 word	4	
JB3	跳躍/呼叫	1 word	4	
JC	跳躍/呼叫	1 word	4	
JMP	跳躍/呼叫	1 word	4	
JNC	跳躍/呼叫	1 word	4	
JNZ	跳躍/呼叫	1 word	4	
JZ	跳躍/呼叫	1 word	4	
LCB	LCD	1 word	4	Read: 4
				Write: 8
LCD	LCD	1 word	4	8
LCDH	Table ROM	1 word	4	16
LCE	LCD	1 word	4	8
LCP	LCD	1 word	4	Read: 4

OPCODE	指令類型	指令長度	指令週期 (machine cycle)	記憶體傳輸位元數 (bits)
				Write: 8
LCT	LCD	1 word	4	Read: 4
				Write: 8
LDA	RAM 資料傳送	1 word	4	4(/16*1)
LDAH	RAM 資料傳送	1 word	4	16
LDH	Table ROM	1 word	4	4(/16*3)
LDL	Table ROM	1 word	4	4
LDS	RAM 資料傳送	1 word	4	4
LDS8	RAM 資料傳送	1 word	4	8
LDSH	RAM 資料傳送	2 words	8	16
LID	RAM 資料傳送	1 word	4	4
LID8	RAM 資料傳送	1 word	4	8
LIDH	RAM 資料傳送	1 word	4	16
LSP	暫存器存取	1 word	4	4
MAF	暫存器存取	1 word	4	4
MCX	暫存器存取	1 word	4	4
MDX	暫存器存取	1 word	4	4
MHL	索引暫存器存取	1 word	4	16
MKI	暫存器存取	1 word	4	4
MMH	暫存器存取	1 word	4	4(/16*1)
MMHH	暫存器存取	1 word	4	16
MMW	暫存器存取	1 word	4	4
MRA	暫存器存取	1 word	4	1
MRF1	RFC 功能	1 word	4	4(/16*4)
MRF2	RFC 功能	1 word	4	4
MRF3	RFC 功能	1 word	4	4
MRF4	RFC 功能	1 word	4	4
MRH	索引暫存器存取	1 word	4	4
MRI	暫存器存取	1 word	4	4
MRIH	暫存器存取	1 word	4	16
MRL	索引暫存器存取	1 word	4	4
MRU	索引暫存器存取	1 word	4	4
MRV	索引暫存器存取	1 word	4	4(/16*5)
MRW	RAM 資料傳送	1 word	4	4(/16*1)
MSB	暫存器存取	1 word	4	4
MSC	暫存器存取	1 word	4	4
MSD	暫存器存取	1 word	4	4
MULD	算數/邏輯運算	1 word	4	4(/16*1)
MULDH	算數/邏輯運算	1 word	4	16

OPCODE	指令類型	指令長度	指令週期 (machine cycle)	記憶體傳輸位元數 (bits)
MULH	算數/邏輯運算	1 word	4	4(/16*1)
MULHH	算數/邏輯運算	1 word	4	16
MVL	索引暫存器存取	1 word	4	4
MVH	索引暫存器存取	1 word	4	4
MVU	索引暫存器存取	1 word	4	4
MVV	索引暫存器存取	1 word	4	4(/16*5)
MWM	暫存器存取	1 word	4	4
MWR	RAM 資料傳送	1 word	4	4(/16*1)
MZR	索引暫存器存取	1 word	4	13
NOP	系統功能	1 word	4	
OPA	I/O Port	1 word	4	4(/16*2)
OPAS	I/O Port	1 word	4	2
OPB	I/O Port	1 word	4	4
OPC	I/O Port	1 word	4	4
OPD	I/O Port	1 word	4	4
OPE	I/O Port	1 word	4	4
OR	算數/邏輯運算	1 word	4	4
ORI	算數/邏輯運算	1 word	4	4
PLC	系統功能	1 word	4	
PTR	Table ROM	1 word	4	
RF	系統功能	1 word	4	
RF2	系統功能	1 word	4	
RHL	索引暫存器存取	1 word	4	16
RLC	算數/邏輯運算	1 word	4	4
RTS	跳躍/呼叫	1 word	4	
RRC	算數/邏輯運算	1 word	4	4(/16*1)
RRCH	算數/邏輯運算	1 word	4	16
RRL	索引暫存器存取	1 word	4	4
RRH	索引暫存器存取	1 word	4	4
RRU	索引暫存器存取	1 word	4	4
RRV	索引暫存器存取	1 word	4	4(/16*5)
RTM2L RTM3L	Timer	1 word	4	4(/16*6)
RTM21 RTM1H RTM31	Timer	1 word	4	4
RVL	索引暫存器存取	1 word	4	4
RVH	索引暫存器存取	1 word	4	4
RVU	索引暫存器存取	1 word	4	4
RVV	索引暫存器存取	1 word	4	4(/16*5)

OPCODE	指令類型	指令長度	指令週期 (machine cycle)	記憶體傳輸位元數 (bits)
RZR	索引暫存器存取	1 word	4	16
SBC	算數/邏輯運算	1 word	4	4(/16*1)
SBCH	算數/邏輯運算	1 word	4	16
SBCI	算數/邏輯運算	1 word	4	4
SBCM	算數/邏輯運算	1 word	4	4
SBCMH	算數/邏輯運算	1 word	4	16
SBZ	週邊功能	1 word	4	
SCA	系統功能	1 word	4	
SCC	系統功能	1 word	4	
SCNT	RFC 功能	1 word	4	
SCX	系統功能	1 word	4	
SF	系統功能	1 word	4	
SF2	系統功能	1 word	4	
SHE	系統功能	1 word	4	
SHLX	索引暫存器存取	2 words	8	
SIAP	系統功能	1 word	4	
SIE*	系統功能	1 word	4	
SL0, SL1	算數/邏輯運算	1 word	4	4(/16*1)
SLOW	系統功能	1 word	4	
SLZ	RAM page 設定 (compiler 專用)	1 word	4	
SMUI	暫存器存取	1 word	4	4(/16*1)
SMUIH	暫存器存取	1 word	4	16
SPA	I/O Port	1 word	4	
SPARH	I/O Port	2 words	8	16
SPATH		1 word	4	16
SPAXH		2 words	8	
SPB	I/O Port	1 word	4	
SPBK	ROM bank 設定 (compiler 專用)	1 word	4	
SPC	I/O Port	1 word	4	
SPD	I/O Port	1 word	4	
SPE	I/O Port	1 word	4	
SPK SPKX	週邊功能	1 word	4	4/8
SPKRH	週邊功能	2 words	8	16
SPKTH		1 word	4	16
SPKXH		2 words	8	
SR0, SR1	算數/邏輯運算	1 word	4	4
SRE	系統功能	1 word	4	

OPCODE	指令類型	指令長度	指令週期 (machine cycle)	記憶體傳輸位元數 (bits)
SRF	RFC 功能	1 word	4	
SRP	Timer	1 word	4	
SRX	RAM page 設定 (compiler 專用)	1 word	4	
SRY	RAM page 設定 (compiler 專用)	1 word	4	
ST3OV	系統功能	1 word	4	
STA	RAM 資料傳送	1 word	4	4(/16*1)
STAH	RAM 資料傳送	1 word	4	16
STE	Timer	1 word	4	
STM	Timer	1 word	4	
STOP	系統功能	1 word	4	
SUB	算數/邏輯運算	1 word	4	4(/16*1)
SUBH	算數/邏輯運算	1 word	4	16
SUBI	算數/邏輯運算	1 word	4	4
SUBM	算數/邏輯運算	1 word	4	4
SUBMH	算數/邏輯運算	1 word	4	16
SXCLK	系統功能	1 word	4	
SWPWR	系統功能	1 word	4	
SZRX	索引暫存器存取	2 words	8	
T1RH	Timer	2 words	8	16
T1TH		1 word	4	16
T1XH		2 words	8	
T2M3X	Timer	2 words	8	
T2RH	Timer	2 words	8	16
T2TH		1 word	4	16
T2XH		2 words	8	
T3RH	Timer	2 words	8	16
T3TH		1 word	4	16
T3XH		2 words	8	
TM2	Timer	1 word	4	4/8
TM2X				
TM3	Timer	1 word	4	4/8
TM3X				
TMS	Timer	1 word	4	4/8
TMSX				

*1 : change to 16bits if RXHM=1 @ Rx mode
*2 : change to 16bits if HMIOA=1
*3 : change to 16bits if HMLDH=1
*4 : change to 16bits if HMMRF=1
*5 : change to 16bits if HMIDV=1
*6 : change to 16bits if HMRTM=1

第二章　指令詳細說明

2-1. 索引暫存器存取指令(Index Register Access Instructions)

IDC

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
將 HL 或是 ZR 的內容分別或是同時加 1。

註記：
在執行指令時不會影響 AC 的內容值。

指令語法：

OP code	運算元	指令動作
IDC&		HL ← HL+1
IDC%		ZR ← ZR+1
IDC$		HL ← HL+1 ZR ← ZR+1

IDC8

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
將 HL 或是 ZR 的內容分別或是同時加 2。

註記：
在執行指令時不會影響 AC 的內容值。

指令語法：

OP code	運算元	指令動作
IDC8&		HL ← HL+2
IDC8%		ZR ← ZR+2
IDC8$		HL ← HL+2 ZR ← ZR+2

IDCH

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：
將 HL 或是 ZR 的內容分別或是同時加 4。

註記：
在執行指令時不會影響 AC 的內容值。

指令語法：

OP code	運算元	指令動作
IDCH&		HL ← HL+4
IDCH%		ZR ← ZR+4
IDCH$		HL ← HL+4 ZR ← ZR+4

MVL

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的內容值儲存到 HL-L。

其位元資料的對應關係如下：

Source RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Content of HL-L	IDBF3	IDBF2	IDBF1	IDBF0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MVL	Rx	IDBF3 ~ IDBF0 ← (Rx)

MVH

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的內容值儲存到 HL-H。

架構 A，其位元資料的對應關係如下：

Source RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

Content of HL-H	IDBF7	IDBF6	IDBF5	IDBF4

架構 B，部份 MCU 並未提供 MVU 指令，而是由執行 MVH 指令時將 AC 值儲存至 HL-U(詳見文件:4BIT Series Difference List)。其位元資料的對應關係如下：

Source AC register	AC3	AC2	AC1	AC0
Content of HL-U	IDBF11	IDBF10	IDBF9	IDBF8

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：
架構 A

OP code	運算元	指令動作
MVH	Rx	IDBF7 ~ IDBF4 ← (Rx)

架構 B

OP code	運算元	指令動作
MVH	Rx	IDBF11 ~ IDBF4 ← AC,(Rx)

MVU(當 MVH 為架構 A)

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的內容值儲存到 HL-U。

其位元資料的對應關係如下：

Source RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Content of HL-U	IDBF11	IDBF10	IDBF9	IDBF8

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MVU	Rx	IDBF11 ~ IDBF8 ← (Rx)

MVV

指令特性：
單一字元長度指令，四個 machine cycle, 4/16 bits data transferred if HMIDV=0/1。

指令說明：
將 Rx 所指定的 4/16bits mode data RAM 的內容值儲存到 HL-V/V~L if HMIDV=0/1。

其位元資料的對應關係如下：

Source RAM data	(Rx)3/15	(Rx)2/14	(Rx)1/13	(Rx)0/12
Content of HL-V	IDBF15	IDBF14	IDBF13	IDBF12
Source RAM data	(Rx)-/11	(Rx)-/10	(Rx)-/9	(Rx)-/8
Content of HL-U	IDBF11	IDBF10	IDBF9	IDBF9
Source RAM data	(Rx)-/7	(Rx)-/6	(Rx)-/5	(Rx)-/4
Content of HL-H	IDBF7	IDBF6	IDBF5	IDBF4
Source RAM data	(Rx)-/3	(Rx)-/2	(Rx)-/1	(Rx)-/0
Content of HL-L	IDBF3	IDBF2	IDBF1	IDBF0

註記：
Rx 在語法上必須是以絕對位址來描述。HMIDV 目前在 EV chip (TM8999)不支援。

指令語法：

OP code	運算元	指令動作
MVV	Rx	IDBF15 ~ IDBF12/0 ← (Rx)3/15~0 if HMIDV=0/1

RVL

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 HL-L 的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of HL-L	IDBF3	IDBF2	IDBF1	IDBF0
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
RVL	Rx	(Rx) ← IDBF3 ~ IDBF0, AC ← IDBF3 ~ IDBF0

RVH

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 HL-H 的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of HL-H	IDBF7	IDBF6	IDBF5	IDBF4
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
RVH	Rx	(Rx) ← IDBF7 ~ IDBF4, AC ← IDBF7 ~ IDBF4

RVU

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 HL-U register 的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of HL-U	IDBF11	IDBF10	IDBF9	IDBF8
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
RVU	Rx	(Rx) ← IDBF11 ~ IDBF8, AC ← IDBF11 ~ IDBF8

RVV

指令特性：
單一字元長度指令，四個 machine cycle, 4/16 bits data transferred if HMIDV=0/1。

指令說明：
將 HL-V 的內容值儲存到 Rx 所指定的 4/16bits mode data RAM 以及 AC if HMIDV=0/1。

其位元資料的對應關係如下：

Content of HL-V	IDBF15	IDBF14	IDBF13	IDBF12
Destination RAM data	(Rx)3/15	(Rx)2/14	(Rx)1/13	(Rx)0/12
Content of HL-U	IDBF11	IDBF10	IDBF9	IDBF9
Destination RAM data	(Rx)-/11	(Rx)-/10	(Rx)-/9	(Rx)-/8
Content of HL-H	IDBF7	IDBF6	IDBF5	IDBF4
Destination RAM data	(Rx)-/7	(Rx)-/6	(Rx)-/5	(Rx)-/4
Content of HL-L	IDBF3	IDBF2	IDBF1	IDBF0

Destination RAM data	(Rx)-/3	(Rx)-/2	(Rx)-/1	(Rx)-/0

註記：
Rx 在語法上必須是以絕對位址來描述。HMIDV 目前在 EV chip (TM8999)不支援。

指令語法：

OP code	運算元	指令動作
RVV	Rx	(Rx)3/15~0 ← IDBF15 ~ IDBF12/0 if HMIDV=0/1, AC3/15~0 ← IDBF15 ~ IDBF12/0 if HMIDV=0/1

SHLX

指令特性：
兩個字元長度指令，八個 machine cycle。

指令說明：
將 16-bit 的 Immediate data(D)儲存到 HL。

此一指令需使用 2 words 長度的指令且需要八個 machine cycle 才能完成，指令中的第一個 word 是 Opcode，而 16 bits 的 immediate data 則位於指令的第二個 word 中。

其位元資料的對應關係如下：

Imm. Data D	D7	D6	D5	D4	D3	D2	D1	D0
HL register	IDBF7	IDBF6	IDBF5	IDBF4	IDBF3	IDBF2	IDBF1	IDBF0
Imm. Data D	D15	D14	D13	D12	D11	D10	D9	D8
HL register	IDBF15	IDBF14	IDBF13	IDBF12	IDBF11	IDBF10	IDBF9	IDBF8

註記：
(1). D = 0h ~ FFFFh (最大值會依各個 MCU 中 HL register 位元數的多寡而有所不同)
(2). MCU 會在這個指令的八個 machine cycle 週期中暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
SHLX SETDAT	D	IDBF15~0 ← D

MRL

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的內容值儲存到@ZR-L。

其位元資料的對應關係如下：

Source RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

Content of ZR-L	ZRBF3	ZRBF2	ZRBF1	ZRBF0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MRL	Rx	ZRBF3 ~ ZRBF0 ← (Rx)

MRH

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的內容值儲存到 ZR-H。

其位元資料的對應關係如下：

Source RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Content of ZR-H	ZRBF7	ZRBF6	ZRBF5	ZRBF4

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MRH	Rx	ZRBF7 ~ ZRBF4 ← (Rx)

MRU

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的內容值儲存到 ZR-U。

其位元資料的對應關係如下：

Source RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Content of ZR-U	ZRBF11	ZRBF10	ZRBF9	ZRBF8

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MRU	Rx	ZRBF11 ~ ZRBF8 ← (Rx)

MRV

指令特性：

單一字元長度指令，四個 machine cycle, 4/16 bits data transferred if HMIDV=0/1。

指令說明：

將 Rx 所指定的 4/16 data RAM 的內容值儲存到 ZR-V/V~L if HMIDV=0/1。

其位元資料的對應關係如下：

Source RAM data	(Rx)3/15	(Rx)2/14	(Rx)1/13	(Rx)0/12
Content of ZR-V	0	0	0	ZRBF12
Source RAM data	(Rx)-/11	(Rx)-/10	(Rx)-/9	(Rx)-/8
Content of ZR-U	ZRBF11	ZRBF10	ZRBF9	ZRBF8
Source RAM data	(Rx)-/7	(Rx)-/6	(Rx)-/5	(Rx)-/4
Content of ZR-H	ZRBF7	ZRBF6	ZRBF5	ZRBF4
Source RAM data	(Rx)-/3	(Rx)-/2	(Rx)-/1	(Rx)-/0
Content of ZR-L	ZRBF3	ZRBF2	ZRBF1	ZRBF0

註記：

Rx 在語法上必須是以絕對位址來描述。HMIDV 目前在 EV chip (TM8999)不支援。

指令語法：

OP code	運算元	指令動作
MRV	Rx	ZRBF12/12~0 ← (Rx)0/12~0 if HMIDV=0/1

RRL

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 ZR-L 的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of ZR-L	ZRBF3	ZRBF2	ZRBF1	ZRBF0
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：

Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
RRL	Rx	(Rx) ← ZRBF3 ~ ZRBF0, AC ← ZRBF3 ~ ZRBF0

RRH

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 ZR-H 的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of ZR-H	ZRBF7	ZRBF6	ZRBF5	ZRBF4
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
RRH	Rx	(Rx) ← ZRBF7 ~ ZRBF4, AC ← ZRBF7 ~ ZRBF4

RRU

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
此一指令將 ZR-U 的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of ZR-U	ZRBF11	ZRBF10	ZRBF9	ZRBF8
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
RRU	Rx	(Rx) ← ZRBF11 ~ ZRBF8, AC ← ZRBF11 ~ ZRBF8

RRV

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred if HMIDV=0/1。

指令說明：
此一指令將 ZR-V 的內容值儲存到 Rx 所指定的 data RAM 以及 AC if HMIDV=0/1。

其位元資料的對應關係如下：

Content of ZR-V	0	0	0	ZRBF0/12
Destination RAM data	(Rx)3/15	(Rx)2/14	(Rx)1/13	(Rx)0/12
Content of ZR-U	ZRBF11	ZRBF10	ZRBF9	ZRBF9
Destination RAM data	(Rx)-/11	(Rx)-/10	(Rx)-/9	(Rx)-/8
Content of ZR-H	ZRBF7	ZRBF6	ZRBF5	ZRBF4
Destination RAM data	(Rx)-/7	(Rx)-/6	(Rx)-/5	(Rx)-/4
Content of ZR-L	ZRBF3	ZRBF2	ZRBF1	ZRBF0
Destination RAM data	(Rx)-/3	(Rx)-/2	(Rx)-/1	(Rx)-/0

註記：
Rx 在語法上必須是以絕對位址來描述。HMIDV 目前在 EV chip (TM8999)不支援。

指令語法：

OP code	運算元	指令動作
RRV	Rx	(Rx)3~0/15~12 ← 0,0,0,ZRBF12 if HMIDV=0/1, AC3~0/15~12 ← 0,0,0,ZRBF12 if HMIDV=0/1 (Rx)11~0 ← ZRBF11~0 if HMIDV=1, AC11~0 ← ZRBF11~0 if HMIDV=1

SZRX

指令特性：
兩個字元長度指令，八個 machine cycle。

指令說明：
將 16-bit 的 Immediate data(D)儲存到 ZR。

此一指令需使用 2 words 長度的指令且需要八個 machine cycle 才能完成，指令中的第一個 word 是 Opcode，而 13 bits 的 immediate data 則位於指令的第二個 word 中。

其位元資料的對應關係如下：

Imm. Data D	D7	D6	D5	D4	D3	D2	D1	D0
ZR register	ZRBF7	ZRBF6	ZRBF5	ZRBF4	ZRBF3	ZRBF2	ZRBF1	ZRBF0
Imm. Data D				D12	D11	D10	D9	D8
ZR register				ZRBF12	ZRBF11	ZRBF10	ZRBF9	ZRBF8

註記：
(1). D = 0h ~ 1FFFh (最大值會依各個 MCU 中 ZR register 位元數的多寡而有所不同)
(2). MCU 會在這個指令的八個 machine cycle 週期中暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
SZRX SETDAT	D	ZRBF12~0 ← D

MHL

指令特性：

單一字元長度指令，四個 machine cycle，最大 16 bits data transferred。

指令說明：

將運算元 X 所指定的 16-bit 資料備份區的內容值寫入 HL index register。

每個 X 值代表 4 個連續位址的 data RAM，對應表如下：

X	Addr+3	Addr+2	Addr+1	Addr
0	$0083	$0082	$0081	$0080
1	$0087	$0086	$0085	$0084
2	$008B	$008A	$0089	$0088
3	$008F	$008E	$008D	$008C
4	$0093	$0092	$0091	$0090
5	$0097	$0096	$0095	$0094
6	$009B	$009A	$0099	$0098
7	$009F	$009E	$009D	$009C
8	$00A3	$00A2	$00A1	$00A0
9	$00A7	$00A6	$00A5	$00A4
A	$00AB	$00AA	$00A9	$00A8
B	$00AF	$00AE	$00AD	$00AC
C	$00B3	$00B2	$00B1	$00B0
D	$00B7	$00B6	$00B5	$00B4
E	$00BB	$00BA	$00B9	$00B8
F	$00BF	$00BE	$00BD	$00BC

其位元資料的對應關係如下：

Content of HL	IDBF7	IDBF6	IDBF5	IDBF4	IDBF3	IDBF2	IDBF1	IDBF0
RAM data	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
RAM addr.	Addr+1				Addr			
Content of HL	IDBF15	IDBF14	IDBF13	IDBF12	IDBF11	IDBF10	IDBF9	IDBF8
RAM data	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
RAM addr.	Addr+3				Addr+2			

註記：

X = 0 ~ Fh

指令語法：

OP code	運算元	指令動作
MHL	X	IDBF3~0 ← (Addr), IDBF7~4 ← (Addr +1), IDBF11~8 ← (Addr +2), IDBF15~12 ← (Addr +3)

RHL

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

將 HL index register 16-bit 的內容值儲存到運算元 X 所指定的 16-bit 資料備份區。

每個 X 值代表 4 個連續位址的 data RAM，對應表如下：

X	Addr+3	Addr+2	Addr+1	Addr
0	$0083	$0082	$0081	$0080
1	$0087	$0086	$0085	$0084
2	$008B	$008A	$0089	$0088
3	$008F	$008E	$008D	$008C
4	$0093	$0092	$0091	$0090
5	$0097	$0096	$0095	$0094
6	$009B	$009A	$0099	$0098
7	$009F	$009E	$009D	$009C
8	$00A3	$00A2	$00A1	$00A0
9	$00A7	$00A6	$00A5	$00A4
A	$00AB	$00AA	$00A9	$00A8
B	$00AF	$00AE	$00AD	$00AC
C	$00B3	$00B2	$00B1	$00B0
D	$00B7	$00B6	$00B5	$00B4
E	$00BB	$00BA	$00B9	$00B8
F	$00BF	$00BE	$00BD	$00BC

其位元資料的對應關係如下：

Content of HL	IDBF7	IDBF6	IDBF5	IDBF4	IDBF3	IDBF2	IDBF1	IDBF0
RAM data	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
RAM addr.	Addr+1				Addr			
Content of HL	IDBF15	IDBF14	IDBF13	IDBF12	IDBF11	IDBF10	IDBF9	IDBF8
RAM data	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
RAM addr.	Addr+3				Addr+2			

註記：

X = 0 ~ Fh

指令語法：

OP code	運算元	指令動作
RHL	X	(Addr) ← IDBF3~0, (Addr+1) ← IDBF7~4, (Addr+2) ← IDBF11~8, (Addr+3) ← IDBF15~12

MZR

指令特性：

單一字元長度指令，四個 machine cycle，最大 13 bits data transferred。

指令說明：

將運算元 X 所指定的 16-bit 資料備份區的內容值寫入 ZR index register。

每個 X 值代表 4 個連續位址的 data RAM，對應表如下：

X	Addr+3	Addr+2	Addr+1	Addr
0	$00C3	$00C2	$00C1	$00C0
1	$00C7	$00C6	$00C5	$00C4
2	$00CB	$00CA	$00C9	$00C8
3	$00CF	$00CE	$00CD	$00CC
4	$00D3	$00D2	$00D1	$00D0
5	$00D7	$00D6	$00D5	$00D4
6	$00DB	$00DA	$00D9	$00D8
7	$00DF	$00DE	$00DD	$00DC
8	$00E3	$00E2	$00E1	$00E0
9	$00E7	$00E6	$00E5	$00E4
A	$00EB	$00EA	$00E9	$00E8
B	$00EF	$00EE	$00ED	$00EC
C	$00F3	$00F2	$00F1	$00F0
D	$00F7	$00F6	$00F5	$00F4
E	$00FB	$00FA	$00F9	$00F8
F	$00FF	$00FE	$00FD	$00FC

其位元資料的對應關係如下：

Content of ZR	ZRBF7	ZRBF6	ZRBF5	ZRBF4	ZRBF3	ZRBF2	ZRBF1	ZRBF0
RAM data	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
RAM addr.	Addr+1				Addr			
Content of ZR	NA	NA	NA	ZRBF12	ZRBF11	ZRBF10	ZRBF9	ZRBF8
RAM data	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
RAM addr.	Addr+3				Addr+2			

註記：

X = 0 ~ Fh

指令語法：

OP code	運算元	指令動作
MZR	X	ZRBF3~0 ← (Addr), ZRBF7~4 ← (Addr+1), ZRBF11~8 ← (Addr+2), ZRBF12 ← (Addr+3)

RZR

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

將 ZR index register 13-bit 的內容值儲存到運算元 X 所指定的 16-bit 資料備份區。

每個 X 值代表 4 個連續位址的 data RAM，對應表如下：

X	Addr+3	Addr+2	Addr+1	Addr
0	$00C3	$00C2	$00C1	$00C0
1	$00C7	$00C6	$00C5	$00C4
2	$00CB	$00CA	$00C9	$00C8
3	$00CF	$00CE	$00CD	$00CC
4	$00D3	$00D2	$00D1	$00D0
5	$00D7	$00D6	$00D5	$00D4
6	$00DB	$00DA	$00D9	$00D8
7	$00DF	$00DE	$00DD	$00DC
8	$00E3	$00E2	$00E1	$00E0
9	$00E7	$00E6	$00E5	$00E4
A	$00EB	$00EA	$00E9	$00E8
B	$00EF	$00EE	$00ED	$00EC
C	$00F3	$00F2	$00F1	$00F0
D	$00F7	$00F6	$00F5	$00F4
E	$00FB	$00FA	$00F9	$00F8
F	$00FF	$00FE	$00FD	$00FC

其位元資料的對應關係如下：

Content of ZR	ZRBF7	ZRBF6	ZRBF5	ZRBF4	ZRBF3	ZRBF2	ZRBF1	ZRBF0
RAM data	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
RAM addr.	Addr+1				Addr			
Content of ZR	NA	NA	NA	ZRBF12	ZRBF11	ZRBF10	ZRBF9	ZRBF8
RAM data	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
RAM addr.	Addr+3				Addr+2			

註記：

X = 0 ~ Fh

指令語法：

OP code	運算元	指令動作
RZR	X	(Addr) ← ZRBF3~0, (Addr+1) ← ZRBF7~4, (Addr+2) ← ZRBF11~8, (Addr+3) ← ZRBF12

2-2. RAM資料傳送指令(RAM Data Transferred Instructions)

LID

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將@HL 或是@ZR 所指向的 data RAM 的內容值儲存到@ZR 或是@HL 所指向的 data RAM 的位址上。

註記：
(1). 在執行指令之前必須先將 HL 以及 ZR 的內容設定完成。
(2). 指令結束之後，部分語法會自動將 HL 或是 ZR 的內容值加 1。
(3). 若 MCU 有提供藉由 Rm 設定 LIDAC=1(目前在 EV chip (TM8999)不支援)則可以將 LID 指令切換為亦會將@HL/ZR 所指向的 data RAM 內容值儲存到 AC。

指令語法：

OP code	運算元	指令動作
LID	@ZR, @HL	(@ZR) ← (@HL)
LID&	@ZR, @HL	(@ZR) ← (@HL) HL ← HL+1
LID%	@ZR, @HL	(@ZR) ← (@HL) ZR ← ZR+1
LID$	@ZR, @HL	(@ZR) ← (@HL) HL ← HL+1 ZR ← ZR+1
LID	@HL, @ZR	(@HL) ← (@ZR)
LID&	@HL, @ZR	(@HL) ← (@ZR) HL ← HL+1
LID%	@HL, @ZR	(@HL) ← (@ZR) ZR ← ZR+1
LID$	@HL, @ZR	(@HL) ← (@ZR) HL ← HL+1, ZR ← ZR+1

LID8

指令特性：
單一字元長度指令，四個 machine cycle, 8 bits data transferred。

指令說明：
將@HL 或是@ZR 所指向的 data RAM 的連續兩個位址的 8 bit 內容值儲存到@ZR 或是@HL 所指向的 data RAM 的連續兩個位址上。

無論@HL 或是@ZR 的 LSB 的值為何，在指令執行時都會忽略實際 LSB 值，而存取的連續位址將是@HL/ZR(RAM 位址 bit0=0)以及@HL/ZR(RAM 位址 bit0=1)。

其位元資料的對應關係如下：

Source RAM addr. & data mapping	RAM addr. = Ns*+1				RAM addr. = Ns*			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Source 8bits data	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Source RAM addr	@HL/ZR(RAM 位址 bit0=1)				@HL/ZR(RAM 位址 bit0=0)			
Destination RAM addr. & data mapping	RAM addr. = Nd*+1				RAM addr. = Nd*			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Destination 8bits data	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Destination RAM addr.	@ZR/HL(RAM 位址 bit0=1)				@ZR/HL(RAM 位址 bit0=0)			

* ：Ns, Nd 代表偶數的 data RAM 位址。

註記：

(1). 在執行指令之前必須先將 HL 以及 ZR 的內容設定完成。

(2). 指令結束之後，部分語法會自動將 HL 或是 ZR 的內容值加 2。

指令語法：

OP code	運算元	指令動作
LID8	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← (@HL(RAM 位址 bit0=0)) (@ZR(RAM 位址 bit0=1)) ← (@HL(RAM 位址 bit0=1))
LID8&	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← (@HL(RAM 位址 bit0=0)), (@ZR(RAM 位址 bit0=1)) ← (@HL(RAM 位址 bit0=1)) HL ← HL+2
LID8%	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← (@HL(RAM 位址 bit0=0)) (@ZR(RAM 位址 bit0=1)) ← (@HL(RAM 位址 bit0=1)) ZR ← ZR+2
LID8$	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← (@HL(RAM 位址 bit0=0)) (@ZR(RAM 位址 bit0=1)) ← (@HL(RAM 位址 bit0=1)) HL ← HL+2 ZR ← ZR+2
LID8	@HL, @ZR	(@HL(RAM 位址 bit0=0)) ← (@ZR(RAM 位址 bit0=0)) (@HL(RAM 位址 bit0=1)) ← (@ZR(RAM 位址 bit0=1))
LID8&	@HL, @ZR	(@HL(RAM 位址 bit0=0)) ← (@ZR(RAM 位址 bit0=0)) (@HL(RAM 位址 bit0=1)) ← (@ZR(RAM 位址 bit0=1)) HL ← HL+2
LID8%	@HL, @ZR	(@HL(RAM 位址 bit0=0)) ← (@ZR(RAM 位址 bit0=0)) (@HL(RAM 位址 bit0=1)) ← (@ZR(RAM 位址 bit0=1)) ZR ← ZR+2
LID8$	@HL, @ZR	(@HL(RAM 位址 bit0=0)) ← (@ZR(RAM 位址 bit0=0)) (@HL(RAM 位址 bit0=1)) ← (@ZR(RAM 位址 bit0=1)) HL ← HL+2, ZR ← ZR+2

LIDH

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

將@HL 或是@ZR 所指向的 data RAM 的連續四個位址的 16 bit 內容值儲存到@ZR 或是@HL 所指向的 data RAM 的連續四個位址上。

無論@HL 或是@ZR 的最低的兩個位元的實際值為何，在指令執行時都會被忽略掉，而連續存取的四個位址是@HL/ZR(RAM 位址 bit1,0=00)，@HL/ZR(RAM 位址 bit1,0=01)，@HL/ZR(RAM 位址 bit1,0=10)，@HL/ZR(RAM 位址 bit1,0=11)。

其位元資料的對應關係如下：

Source RAM addr. & data mapping	RAM addr. = Ns*+1				RAM addr. = Ns*			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Source 16bits data	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Source RAM addr	@HL/ZR(RAM 位址 bit1,0=01)				@HL/ZR(RAM 位址 bit1,0=00)			
Destination RAM addr. & data mapping	RAM addr. = Nd*+1				RAM addr. = Nd*			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Destination 16bits data	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Destination RAM addr.	@ZR/HL(RAM 位址 bit1,0=01)				@ZR/HL(RAM 位址 bit1,0=00)			

Source RAM addr. & data mapping	RAM addr. = Ns*+3				RAM addr. = Ns*+2			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Source 16bits data	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
Destination RAM addr	@HL/ZR(RAM 位址 bit1,0=11)				@HL/ZR(RAM 位址 bit1,0=10)			
Destination RAM addr. & data mapping	RAM addr. = Nd*+3				RAM addr. = Nd*+2			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Destination 16bits data	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
Destination RAM addr.	@ZR/HL(RAM 位址 bit1,0=11)				@ZR/HL(RAM 位址 bit1,0=10)			

*：Ns, Nd 代表四的倍數的 data RAM 位址。

註記：

(1). 在執行指令之前必須先將 HL 以及 ZR 的內容設定完成。

(2). 指令結束之後，部分語法會自動將 HL 或是 ZR 的內容值加 4。

(3). 若 MCU 有提供藉由 Rm 設定 LIDAC=1(目前在 EV chip (TM8999)不支援)則可以將 LIDH 指令切換為亦會將@HL/ZR 所指向的 16bits data RAM 內容值儲存到 16bits mode AC(ACH)。

指令語法：

OP code	運算元	指令動作
LIDH	@ZR, @HL	(@ZR(RAM 位址 bit1,0=00)) ← (@HL(RAM 位址 bit1,0=00)) (@ZR(RAM 位址 bit1,0=01)) ← (@HL(RAM 位址 bit1,0=01)) (@ZR(RAM 位址 bit1,0=10)) ← (@HL(RAM 位址 bit1,0=10)) (@ZR(RAM 位址 bit1,0=11)) ← (@HL(RAM 位址 bit1,0=11))

OP code	運算元	指令動作
LIDH&	@ZR, @HL	(@ZR(RAM 位址 bit1,0=00)) ← (@HL(RAM 位址 bit1,0=00)) (@ZR(RAM 位址 bit1,0=01)) ← (@HL(RAM 位址 bit1,0=01)) (@ZR(RAM 位址 bit1,0=10)) ← (@HL(RAM 位址 bit1,0=10)) (@ZR(RAM 位址 bit1,0=11)) ← (@HL(RAM 位址 bit1,0=11)) HL ← HL+4
LIDH%	@ZR, @HL	(@ZR(RAM 位址 bit1,0=00)) ← (@HL(RAM 位址 bit1,0=00)) (@ZR(RAM 位址 bit1,0=01)) ← (@HL(RAM 位址 bit1,0=01)) (@ZR(RAM 位址 bit1,0=10)) ← (@HL(RAM 位址 bit1,0=10)) (@ZR(RAM 位址 bit1,0=11)) ← (@HL(RAM 位址 bit1,0=11)) ZR ← ZR+4
LIDH$	@ZR, @HL	(@ZR(RAM 位址 bit1,0=00)) ← (@HL(RAM 位址 bit1,0=00)) (@ZR(RAM 位址 bit1,0=01)) ← (@HL(RAM 位址 bit1,0=01)) (@ZR(RAM 位址 bit1,0=10)) ← (@HL(RAM 位址 bit1,0=10)) (@ZR(RAM 位址 bit1,0=11)) ← (@HL(RAM 位址 bit1,0=11)) HL ← HL+4 ZR ← ZR+4
LIDH	@HL, @ZR	(@HL(RAM 位址 bit1,0=00)) ← (@ZR(RAM 位址 bit1,0=00)) (@HL(RAM 位址 bit1,0=01)) ← (@ZR(RAM 位址 bit1,0=01)) (@HL(RAM 位址 bit1,0=10)) ← (@ZR(RAM 位址 bit1,0=10)) (@HL(RAM 位址 bit1,0=11)) ← (@ZR(RAM 位址 bit1,0=11))
LIDH&	@HL, @ZR	(@HL(RAM 位址 bit1,0=00)) ← (@ZR(RAM 位址 bit1,0=00)) (@HL(RAM 位址 bit1,0=01)) ← (@ZR(RAM 位址 bit1,0=01)) (@HL(RAM 位址 bit1,0=10)) ← (@ZR(RAM 位址 bit1,0=10)) (@HL(RAM 位址 bit1,0=11)) ← (@ZR(RAM 位址 bit1,0=11)) HL ← HL+4
LIDH%	@HL, @ZR	(@HL(RAM 位址 bit1,0=00)) ← (@ZR(RAM 位址 bit1,0=00)) (@HL(RAM 位址 bit1,0=01)) ← (@ZR(RAM 位址 bit1,0=01)) (@HL(RAM 位址 bit1,0=10)) ← (@ZR(RAM 位址 bit1,0=10)) (@HL(RAM 位址 bit1,0=11)) ← (@ZR(RAM 位址 bit1,0=11)) ZR ← ZR+4
LIDH$	@HL, @ZR	(@HL(RAM 位址 bit1,0=00)) ← (@ZR(RAM 位址 bit1,0=00)) (@HL(RAM 位址 bit1,0=01)) ← (@ZR(RAM 位址 bit1,0=01)) (@HL(RAM 位址 bit1,0=10)) ← (@ZR(RAM 位址 bit1,0=10)) (@HL(RAM 位址 bit1,0=11)) ← (@ZR(RAM 位址 bit1,0=11)) HL ← HL+4, ZR ← ZR+4

LDS

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 4-bit 的 Immediate data(D)儲存到 Rx，@HL 或是@ZR 所指向的 data RAM 以及 AC。

其位元資料的對應關係如下：

Immediate data	D3	D2	D1	D0
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Destination AC register	AC3	AC2	AC1	AC0

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). D = 0 ~ Fh。

指令語法：

OP code	運算元	指令動作
LDS	Rx, D	(Rx) ← D, AC ← D
LDS	@HL, D	(@HL) ← D, AC ← D
LDS#	@HL, D	(@HL) ← D, AC ← D HL ← HL+1
LDS	@ZR, D	(@ZR) ← D, AC ← D
LDS#	@ZR, D	(@ZR) ← D, AC ← D ZR ← ZR+1

LDS8

指令特性：
單一字元長度指令，四個 machine cycle, 8 bits data transferred。

指令說明：
將 8-bit 的 Immediate data(D)儲存到@HL 或是@ZR 所指向的兩個連續 data RAM 位址上。

無論 HL 或是 ZR 的 LSB 的實際值為何，在指令執行時，都將會被忽略掉，而連續存取的兩個位址將是@HL/ZR(RAM 位址 bit0=0), @HL/ZR(RAM 位址 bit0=1)。

其位元資料的對應關係如下：

Imm. data D	D7	D6	D5	D4	D3	D2	D1	D0
Destination RAM data	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
Destination RAM addr.	@HL/ZR(RAM 位址 bit0=1)				@HL/ZR(RAM 位址 bit0=0)			

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 2。
(2). D = 0 ~ FFh

指令語法：

OP code	運算元	指令動作
LDS8	@HL, D	(@HL(RAM 位址 bit0=0)) ← D3~D0, (@HL(RAM 位址 bit0=1)) ← D7~D4,
LDS8#	@HL, D	(@HL(RAM 位址 bit0=0)) ← D3~D0, (@HL(RAM 位址 bit0=1)) ← D7~D4, HL ← HL+2
LDS8	@ZR, D	(@ZR(RAM 位址 bit0=0)) ← D3~D0, (@ZR(RAM 位址 bit0=1)) ← D7~D4,

OP code	運算元	指令動作
LDS8#	@ZR, D	(@ZR(RAM 位址 bit0=0)) ← D3~D0, (@ZR(RAM 位址 bit0=1)) ← D7~D4, ZR ← ZR+2

LDSH

指令特性：

兩個字元長度指令，八個 machine cycle, 16 bits data transferred。

指令說明：

將 16-bit 的 Immediate data (D)儲存到@HL 或是@ZR 所指向的 data RAM 的連續 4 個位址上。

此一指令需使用 2 words 的長度，且需要八個 machine cycle 才能完成，指令中的第一個 word 是 Opcode 以及第一個運算元，而 16 bits 的 immediate data 則位於指令的第二個 word 中。

無論 HL 或是 ZR 的最低兩個位元的實際值為何，在指令執行時都將會被忽略掉，而連續存取的四個位址將是@HL/ZR(RAM 位址 bit1,0=00), @HL/ZR(RAM 位址 bit1,0=01), @HL/ZR(RAM 位址 bit1,0=10), @HL/ZR(RAM 位址 bit1,0=11)。

其位元資料的對應關係如下：

Imm. Data D	D7	D6	D5	D4	D3	D2	D1	D0
Destination RAM data	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
Destination RAM addr.	@HL/ZR(RAM 位址 bit1,0=01)				@HL/ZR(RAM 位址 bit1,0=00)			
Imm. Data D	D15	D14	D13	D12	D11	D10	D9	D8
Destination RAM data	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
Destination RAM addr.	@HL/ZR(RAM 位址 bit1,0=11)				@HL/ZR(RAM 位址 bit1,0=10)			

註記：

(1). D = 0h ~ FFFFh。

(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

(3). MCU 會在指令週期中的八個 machine cycle 期間暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
LDSH SETDAT	@HL, D	(@HL(RAM 位址 bit1,0=00)) ← D3~D0, (@HL(RAM 位址 bit1,0=01)) ← D7~D4, (@HL(RAM 位址 bit1,0=10)) ← D11~D8, (@HL(RAM 位址 bit1,0=11)) ← D15~D12
LDSH# SETDAT	@HL, D	(@HL(RAM 位址 bit1,0=00)) ← D3~D0, (@HL(RAM 位址 bit1,0=01)) ← D7~D4, (@HL(RAM 位址 bit1,0=10)) ← D11~D8, (@HL(RAM 位址 bit1,0=11)) ← D15~D12

OP code	運算元	指令動作
		HL ← HL+4
LDSH SETDAT	@ZR, D	(@ZR(RAM 位址 bit1,0=00)) ← D3~D0, (@ZR(RAM 位址 bit1,0=01)) ← D7~D4, (@ZR(RAM 位址 bit1,0=10)) ← D11~D8, (@ZR(RAM 位址 bit1,0=11)) ← D15~D12
LDSH# SETDAT	@ZR, D	(@ZR(RAM 位址 bit1,0=00)) ← D3~D0, (@ZR(RAM 位址 bit1,0=01)) ← D7~D4, (@ZR(RAM 位址 bit1,0=10)) ← D11~D8, (@ZR(RAM 位址 bit1,0=11)) ← D15~D12 ZR ← ZR+4

若 MCU 有提供設定 LDSHAC=1(目前在 EV chip (TM8999)不支援)則亦會將 16-bit 的 Immediate data (D)傳送至 AC15~0。

STA

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 AC 的內容值儲存到 Rx，@HL 或是@ZR 所指向的 data RAM。

其位元資料的對應關係如下：

AC	AC3	AC2	AC1	AC0
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 STA 指令的 Rx 模式切換成 16bits mode，將 ACH(16bits mode AC)儲存到 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)

指令語法：

OP code	運算元	指令動作
STA	Rx	(Rx)(H) ← AC(H)
STA	@HL	(@HL) ← AC
STA#	@HL	(@HL) ← AC HL ← HL+1
STA	@ZR	(@ZR) ← AC
STA#	@ZR	(@ZR) ← AC ZR ← ZR+1

STAH(目前在 EV chip (TM8999)不支援)

指令特性：
單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：
將 ACH 的內容值儲存到@HL 或是@ZR 所指向的 16bits mode data RAM((Rx)H)。

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
STAH	@HL	(@HL)H ← ACH
STAH#	@HL	(@HL)H ← ACH HL ← HL+1
STAH	@ZR	(@ZR)H ← ACH
STAH#	@ZR	(@ZR)H ← ACH ZR ← ZR+4

MRW

指令特性：
單一字元長度指令，四個 machine cycle, 4/16 bits data transferred if RXHM=0/1。

指令說明：
將 Rx 所指向的 data RAM 的內容值儲存到@HL 或是@ZR 或是 Ry 所指向的 data RAM 以及 AC。

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1/4 if RXHM=0/1。
(2). Rx, Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MRW	Ry, Rx	(Ry) ← (Rx), AC ← (Rx)
MRW	@HL, Rx	(@HL) ← (Rx), AC ← (Rx)
MRW#	@HL, Rx	(@HL) ← (Rx), AC ← (Rx) HL ← HL+1/4 if RXHM=0/1
MRW	@ZR, Rx	(@ZR) ← (Rx), AC ← (Rx)
MRW#	@ZR, Rx	(@ZR) ← (Rx), AC ← (Rx) ZR ← ZR+1/4 if RXHM=0/1

MWR

指令特性：
單一字元長度指令，四個 machine cycle, 4/16 bits data transferred if RXHM=0/1。

指令說明：
將@HL 或是@ZR 或是 Ry 所指向的 data RAM 的內容值儲存到 Rx 所指向的 data RAM 以及 AC。

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1/4 if RXHM=0/1。
(2). Rx, Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MWR	Rx, Ry	(Rx) ← (Ry), AC ← (Ry)
MWR	Rx, @HL	(Rx) ← (@HL), AC ← (@HL)
MWR#	Rx, @HL	(Rx) ← (@HL), AC ← (@HL) HL ← HL+1/4 if RXHM=0/1
MWR	Rx, @ZR	(Rx) ← (@ZR), AC ← (@ZR)
MWR#	Rx, @ZR	(Rx) ← (@ZR), AC ← (@ZR) ZR ← ZR+1/4 if RXHM=0/1

LDA

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx, @HL 或是@ZR 所指向的 data RAM 的內容值儲存到 AC。

其位元資料的對應關係如下：

Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 LDA 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)儲存到 ACH(16bits mode AC)

指令語法：

OP code	運算元	指令動作
LDA	Rx	AC(H) ← (Rx)(H)
LDA	@HL	AC ← (@HL)
LDA#	@HL	AC ← (@HL), HL ← HL + 1
LDA	@ZR	AC ← (@ZR)
LDA#	@ZR	AC ← (@ZR), ZR ← ZR + 1

LDAH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

將@HL 或是@ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值儲存到 ACH(16bits mode AC)。

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
LDAH	@HL	ACH ← (@HL)H
LDAH#	@HL	ACH ← (@HL)H, HL ← HL + 4
LDAH	@ZR	ACH ← (@ZR)H
LDAH#	@ZR	ACH ← (@ZR)H, ZR ← ZR + 4

2-3. LCD指令(LCD Instructions)

LCT

指令特性：

單一字元長度指令，四個 machine cycle，8 bits (write) 和 4 bits (read) data transferred。

指令說明：

將 Ry 或是@HL 所指向的 data RAM 的內容值送到 7-segment 解碼器產生 8 bits data (DBUSH~DBUSA)，之後再將這 8 bits data 儲存到@ZR 所指向的 data RAM 的連續兩個位址上或是 Lz 所指向的 LCD display memory 的位址上。

無論@ZR 的 LSB 的實際值為何，在指令執行時都將會被忽略掉，而連續存取的兩個位址將是@ZR (RAM 位址 bit0=0)以及@ZR(RAM 位址 bit0=1)。

其位元資料的對應關係如下：

Destination RAM addr. & data mapping	RAM addr. = N*+1				RAM addr. = N*			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
7-seg decoder output data	DBUSH	DBUSG	DBUSF	DBUSE	DBUSD	DBUSC	DBUSB	DBUSA
@ZR addr.	@ZR(RAM 位址 bit0=1)				@ZR(RAM 位址 bit0=0)			
LZ addr.	0100H + Lz x 2 + 1				0100H + Lz x 2 + 0			

*：N 代表偶數的 data RAM 位址。

“LZ addr” : 若 LCD display memory 與 data RAM 共用

註記：

(1). 在執行指令之前必須先將 HL、ZR 的內容設定完成。

(2). Ry 以及 Lz 在語法上必須是以絕對位址來描述。

(3). 指令結束之後，部分語法會自動將 ZR 的內容值加 2 或是將 HL 的內容值加 1。

指令語法：

OP code	運算元	指令動作
LCT	@ZR, Ry	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (Ry) (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (Ry)
LCT#	@ZR, Ry	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (Ry) (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (Ry) ZR ← ZR+2
LCT	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (@HL), (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (@HL)
LCT&	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (@HL), (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (@HL) HL ← HL+1
LCT%	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (@HL), (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (@HL) ZR ← ZR+2
LCT$	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (@HL), (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (@HL) HL ← HL+1 ZR ← ZR+2
LCT	Lz, Ry	(Lz) ← 7-segment decoder ← (Ry)

OP code	運算元	指令動作
LCT	Lz, @HL	(Lz) ← 7-segment decoder ← (@HL)
LCT#	Lz, @HL	*Lz) ← 7-segment decoder ← (@HL) HL ← HL+1

LCB

指令特性：

單一字元長度指令，四個 machine cycle，8 bits (write) 和 4 bits (read) data transferred。

指令說明：

將 Ry 或是@HL 所指向的 data RAM 的內容值送到 7-segment 解碼器產生 8 bits data，之後再將這 8 bits data 儲存到@ZR 所指向的 data RAM 的兩個連續位址上或是 Lz 所指向的 LCD display memory 的位址上。

如果 Ry 或是@HL 所指向的 data RAM 的內容值為 0h，則 7-segment 解碼器所產生的 8 bits data (DBUSH~A) 將為 00h。無論@ZR 的 LSB 的實際值為何，在指令執行時 都將會被忽略掉，而連續存取的兩個位址將是@ZR(RAM 位址 bit0=0)以及@ZR(RAM 位址 bit0=1)。

其位元資料的對應關係如下：

Destination RAM addr. & data mapping	RAM addr. = N*+1				RAM addr. = N*			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
7-seg decoder output data	DBUSH	DBUSG	DBUSF	DBUSE	DBUSD	DBUSC	DBUSB	DBUSA
@ZR addr.	@ZR(RAM 位址 bit0=1)				@ZR(RAM 位址 bit0=0)			
LZ addr.	0100H + Lz x 2 + 1				0100H + Lz x 2 + 0			

*：N 代表偶數的 data RAM 位址。

“LZ addr”：若 LCD display memory 與 data RAM 共用

註記：

(1). 在執行指令之前必須先將 ZR 或是 HL 的內容設定完成。

(2). Ry 以及 Lz 在語法上必須是以絕對位址來描述。

(3). 指令結束之後部分語法會自動將 ZR 的內容值加 2 或是將 HL 的內容值加 1。

指令語法：

OP code	運算元	指令動作
LCB	@ZR, Ry	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (Ry) (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (Ry) 00 ← 7-segment decoder ← 0
LCB#	@ZR, Ry	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (Ry) (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (Ry) ZR ← ZR+2 00 ← 7-segment decoder ← 0
LCB	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (@HL), (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (@HL) 00 ← 7-segment decoder ← 0
LCB&	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (@HL), (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (@HL)

OP code	運算元	指令動作
		HL ← HL+1 00 ← 7-segment decoder ← 0
LCB%	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (@HL), (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (@HL) ZR ← ZR+2 00 ← 7-segment decoder ← 0
LCB$	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← 7-segment decoder(DBUSD~A) ← (@HL), (@ZR(RAM 位址 bit0=1)) ← 7-segment decoder(DBUSH~E) ← (@HL) HL ← HL+1 ZR ← ZR+2 00 ← 7-segment decoder ← 0
LCB	Lz, Ry	(Lz) ← 7-segment decoder ← (Ry) 00 ← 7-segment decoder ← 0
LCB	Lz, @HL	(Lz) ← 7-segment decoder ← (@HL) 00 ← 7-segment decoder ← 0
LCB#	Lz, @HL	(Lz) ← 7-segment decoder ← (@HL) HL ← HL+1 00 ← 7-segment decoder ← 0

LCP

指令特性：

單一字元長度指令，四個 machine cycle，8 bits (write) 和 4 bits (read) data transferred。

指令說明：

將 Ry 或是@HL 所指向的 data RAM 的內容值與 AC 的內容值儲存到@ZR 所指向的兩個連續的 data RAM 位址上或是 Lz 所指向的 LCD display memory 位址上。

無論@ZR 的 LSB 的實際值為何，在指令執行時都將會被忽略掉，而連續存取的兩個位址將是 @ZR(RAM 位址 bit0=0) 以及 @ZR(RAM 位址 bit0=1)。

其位元資料的對應關係如下：

Destination RAM addr. & data mapping	RAM addr. = N*+1				RAM addr. = N*			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Output 8bits Data	AC3	AC2	AC2	AC0	(Ry)3/ (@HL)3	(Ry2)/ (@HL)2	(Ry)1/ (@HL)1	(Ry)0/ (@HL)0
@ZR addr.	@ZR(RAM 位址 bit0=1)				@ZR(RAM 位址 bit0=0)			
LZ addr.	0100H + Lz x 2 + 1				0100H + Lz x 2 + 0			

*：N 代表偶數的 data RAM 位址。

“LZ addr”：若 LCD display memory 與 data RAM 共用

註記：

(1). 在執行指令之前必須先將 ZR 或是 HL 的內容設定完成。

(2). Ry 以及 Lz 在語法上必須是以絕對位址來描述。

(3). 指令結束之後部分語法會自動將 ZR 的內容值加 2 或是將 HL 的內容值加 1。

指令語法：

OP code	運算元	指令動作

LCP	@ZR, Ry	(@ZR(RAM 位址 bit0=0)) ← (Ry) (@ZR(RAM 位址 bit0=1)) ← AC
LCP#	@ZR, Ry	(@ZR(RAM 位址 bit0=0)) ← (Ry) (@ZR(RAM 位址 bit0=1)) ← AC ZR ← ZR+2
LCP	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← (@HL) (@ZR(RAM 位址 bit0=1)) ← AC
LCP&	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← (@HL) (@ZR(RAM 位址 bit0=1)) ← AC HL ← HL+1
LCP%	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← (@HL) (@ZR(RAM 位址 bit0=1)) ← AC ZR ← ZR+2
LCP$	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← (@HL) (@ZR(RAM 位址 bit0=1)) ← AC HL ← HL+1, ZR ← ZR+2
LCP	Lz, Ry	$(Lz)_{low\ nibble}$ ← (Ry), $(Lz)_{high\ nibble}$ ← AC
LCP	Lz, @HL	$(Lz)_{low\ nibble}$ ← (@HL), $(Lz)_{high\ nibble}$ ← AC
LCP#	Lz, @HL	$(Lz)_{low\ nibble}$ ← (@HL), $(Lz)_{high\ nibble}$ ← AC HL ← HL+1

LCD

指令特性：

單一字元長度指令，四個 machine cycle，8 bits data transferred。

指令說明：

此一指令將@HL 所指向的 table ROM 的 8 bits 內容值儲存到@ZR 所指向的兩個連續 data RAM 位址上或是 Lz 所指向的 LCD display memory 位址上。

無論@ZR 的 LSB 的實際值為何，在指令執行時都將會被忽略掉，而連續存取的兩個位址將是 @ZR (RAM 位址 bit0=0) 以及 @ZR (RAM 位址 bit0=1) 。

其位元資料的對應關係如下：

Destination RAM addr. & data mapping	RAM addr. = N*+1				RAM addr. = N*			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Table ROM output 8bits data	T(@HL)7	T(@HL)6	T(@HL)5	T(@HL)4	T(@HL)3	T(@HL)2	T(@HL)1	T(@HL)0
@ZR addr.	@ZR(RAM 位址 bit0=1)				@ZR(RAM 位址 bit0=0)			
LZ addr.	0100H + Lz x 2 + 1				0100H + Lz x 2 + 0			

*：N 代表偶數的 data RAM 位址。

“LZ addr” : 若 LCD display memory 與 data RAM 共用

註記：

(1). 在執行指令之前必須先將 ZR 或是 HL 的內容設定完成。

(2). Lz 在語法上必須是以絕對位址來描述。
(3). 指令結束之後部分語法會自動將 ZR 的內容值加 2 或是將 H 的內容值 L 加 1。

指令語法：

OP code	運算元	指令動作
LCD	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← $T(@HL)_{low\ nibble}$, (@ZR(RAM 位址 bit0=1)) ← $T(@HL)_{high\ nibble}$
LCD&	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← $T(@HL)_{low\ nibble}$, (@ZR(RAM 位址 bit0=1)) ← $T(@HL)_{high\ nibble}$ HL ← HL+1
LCD%	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← $T(@HL)_{low\ nibble}$, (@ZR(RAM 位址 bit0=1)) ← $T(@HL)_{high\ nibble}$ ZR ← ZR+2
LCD$	@ZR, @HL	(@ZR(RAM 位址 bit0=0)) ← $T(@HL)_{low\ nibble}$, (@ZR(RAM 位址 bit0=1)) ← $T(@HL)_{high\ nibble}$ HL ← HL+1 ZR ← ZR+2
LCD	Lz, @HL	(Lz) ← T(@HL)
LCD#	Lz, @HL	(Lz) ← T(@HL) HL ← HL+1

LCE

指令特性：
單一字元長度指令，四個 machine cycle，8 bits data transferred。

指令說明：
將@HL 或是@ZR 所指向的 data RAM 的連續兩個位址的 8 bits 內容值儲存到 Lz 所指向的 LCD display memory 的位址上。

無論@ZR 或是@HL 的 LSB 的實際值為何，在指令執行時都將會被忽略掉，而連續存取的兩個位址將是@ZR(RAM 位址 bit0=0),@ZR(RAM 位址 bit0=1) 或是@HL(RAM 位址 bit0=0),@HL(RAM 位址 bit0=1)。

其位元資料的對應關係如下：

Source RAM addr. & data mapping	RAM addr. = Ns*+1				RAM addr. = Ns*			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Source 8bits data	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Source RAM addr	@HL/ZR(RAM 位址 bit0=1)				@HL/ZR(RAM 位址 bit0=0)			
Destination RAM addr. & data mapping	RAM addr. = Nd*+1				RAM addr. = Nd*			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Destination 8bits data	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Destination RAM addr.	0100H + Lz x 2 + 1				0100H + Lz x 2 + 0			

*：Ns, Nd 代表偶數的 data RAM 位址。
“LZ addr” : 若 LCD display memory 與 data RAM 共用

註記：
(1). 在執行指令之前必須先將 ZR 或是 HL 的內容設定完成。

(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 2。
(3). Lz 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
LCE	Lz, @HL	$(Lz)_{low\ nibble}$ ← (@HL(RAM 位址 bit0=0)) $(Lz)_{high\ nibble}$ ← (@HL(RAM 位址 bit0=1))
LCE#	Lz, @HL	$(Lz)_{low\ nibble}$ ← (@HL(RAM 位址 bit0=0)) $(Lz)_{high\ nibble}$ ← (@HL(RAM 位址 bit0=1)) HL ← HL+2
LCE	Lz, @ZR	$(Lz)_{low\ nibble}$ ← (@ZR(RAM 位址 bit0=0)) $(Lz)_{high\ nibble}$ ← (@ZR(RAM 位址 bit0=1))
LCE#	Lz, @ZR	$(Lz)_{low\ nibble}$ ← (@ZR(RAM 位址 bit0=0)) $(Lz)_{high\ nibble}$ ← (@ZR(RAM 位址 bit0=1)) ZR ← ZR+2

2-4. 暫存器存取指令(Register Access Instructions)

SMUI

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx，@HL 或是@ZR 所指向的 data RAM 的內容值儲存到 MUI 中，作為乘法運算中的乘數之用。

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 SMUI 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)儲存到 MUIH(16bits mode MUI)

指令語法：

OP code	運算元	指令動作
SMUI	Rx	MUI(H) ← (Rx)(H)
SMUI	@HL	MUI ← (@HL)
SMUI#	@HL	MUI ← (@HL) HL ← HL+1
SMUI	@ZR	MUI ← (@ZR)
SMUI#	@ZR	MUI ← (@ZR) ZR ← ZR+1

SMUIH(目前在 EV chip (TM8999)不支援)

指令特性：
單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：
將@HL 或是@ZR 所指向的 16bits mode data RAM 的內容值((@HL/ZR)H)儲存到 MUIH(16bits mode MUI)中，作為 16bits mode 乘法運算中的乘數之用。

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
SMUIH	@HL	MUIH ← (@HL)H
SMUIH#	@HL	MUIH ← (@HL)H

		HL ← HL+1
SMUIH	@ZR	MUIH ← (@ZR)H
SMUIH#	@ZR	MUIH ← (@ZR)H ZR ← ZR+1

MMH

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 MU(乘法運算結果的 4 個高位元資料)的內容值儲存到 Rx，@HL 或是@ZR 所指向的 data RAM 以及 AC 中。

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). 若 MCU 有提供藉由 Rm 設定 ALUMUI=1(目前在 EV chip (TM8999)不支援)則可將讀取的 MU 切換至 MUI。
(4). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 MHH 指令的 Rx 模式切換成 16bits mode，將 MU(I)H(16bits mode MU(I)) 儲存到 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)

指令語法：

OP code	運算元	指令動作
MMH	Rx	(Rx)(H) ← MU(I)(H), AC(H) ← MU(I) (H)
MMH	@HL	(@HL) ← MU(I), AC ← MU(I)
MMH#	@HL	(@HL) ← MU(I), AC ← MU(I) HL ← HL+1
MMH	@ZR	(@ZR) ← MU(I), AC ← MU(I)
MMH#	@ZR	(@ZR) ← MU(I), AC ← MU(I) ZR ← ZR+1

MMHH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

將 MUH(16bits 乘法運算結果的 16 個高位元資料)的內容值儲存到@HL 或是@ZR 所指向的 16bits mode data RAM 以及 ACH(16bits mode AC)中。

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。
(2). 若 MCU 有提供藉由 Rm 設定 ALUMUI=1(目前在 EV chip (TM8999)不支援)則可將讀取的 MUH 切換至 MUIH。

指令語法：

OP code	運算元	指令動作
MMHH	@HL	(@HL)H ← MU(I)H, ACH ← MU(I)H
MMHH#	@HL	(@HL)H ← MU(I)H, ACH ← MU(I)H HL ← HL+4
MMHH	@ZR	(@ZR)H ← MU(I)H, ACH ← MU(I)H
MMHH#	@ZR	(@ZR)H ← MU(I)H, ACH ← MU(I)H ZR ← ZR+4

MWM

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Ry 所指向的 data RAM (working register) 的內容值傳送到 Rm 所指向的 register (在 EV chip (TM8999) 是一個擴充用的 IO port) 中。

其位元資料的對應關係如下：

Source RAM data	(Ry)3	(Ry)2	(Ry)1	(Ry)0
Register Content of Rm	(Rm)3	(Rm)2	(Rm)1	(Rm)0

註記：
(1). Rm = 0 ~ F
(2). Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MWM	Rm, Ry	(Rm) ← (Ry)

MMW

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 Rm 所指向的 register (在 EV chip (TM8999) 是一個擴充用的 IO port) 的內容值傳送到 Ry 所指向的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of Rm register	(Rm)3	(Rm)2	(Rm)1	(Rm)0
Destination RAM data	(Ry)3	(Ry)2	(Ry)1	(Ry)0
Destination AC register	AC3	AC2	AC1	AC0

註記：
(1). Rm = 0 ~ F
(2). Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MMW	Ry, Rm	(Ry) ← (Rm) AC ← (Rm)

LSP

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Stack Pointer 目前的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Stack Pointer	SP3	SP2	SP1	SP0
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Destination AC register	AC3	AC2	AC1	AC0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
LSP	Rx	(Rx) ← STACK pointer, AC ← STACK pointer

MAF

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 STS1 register 目前的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of STS1	CF	Zero	SCF12(CX2)	SCF11(CX)
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

註記：
(1). Rx 在語法上必須是以絕對位址來描述。
(2). SCF11&12 僅在 RFC 架構 B 下提供。

指令語法：

OP code	運算元	指令動作
MAF	Rx	(Rx) ← STS1, AC ← STS1

MRA

指令特性：
單一字元長度指令，四個 machine cycle, 1 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的內容值的 bit3 儲存到 CF。

其位元資料的對應關係如下：

Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Content of CF	CF	-	-	-

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MRA	Rx	CF ← (Rx)3

MSB

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 STS2 register 目前的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of STS2	SCF3(IOD)	SCF2(HRx)	SCF1(IOC)	BCF(PSF)
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MSB	Rx	(Rx) ← STS2, AC ← STS2

MSC

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 STS3 register 目前的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content STS3	SCF7(Pre-divider)	PH15	SCF5(TMR1)	SCF4(INT)
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MSC	Rx	(Rx) ← STS3, AC ← STS3

MCX

指令特性：
單一字元長度指令，四個 machine cycle, 3 bits data transferred。

指令說明：
將 STS3X register 目前的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of STS3X	*	SCF0(IOA)	SCF6(TMR2)	SCF8(SKI)
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

* : RFC 架構 A => SCF(RFC)
RFC 架構 B => don't care

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MCX	Rx	(Rx) ← STS3X, AC ← STS3X

MSD

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 STS4 register 目前的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of STS4	PGMF	RFOVF	WDF	CSF
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

註記：

(1). Rx 在語法上必須是以絕對位址來描述。

(2). PGMF 由 TM87ML28 支援，目前在 EV chip (TM8999)不支援。

指令語法：

OP code	運算元	指令動作
MSD	Rx	(Rx) ← STS4, AC ← STS4

MDX

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 STS4X register 目前的內容值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of STS4X	SCF10 (TMR3)	INT	CX2	CX
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

註記：

Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作

MDX	Rx	(Rx) ← STS4X, AC ← STS4X

MKI

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 STS5 register 目前的內容值 (KI1~KI4) 儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Content of STS5	KI4	KI3	KI2	KI1
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

註記：
(1). Rx 在語法上必須是以絕對位址來描述。
(2). (若 Key-Scan 與 IOC Pins share 大部分 MCU 只用 IPC instruction 讀取 KI)

指令語法：

OP code	運算元	指令動作
MKI	Rx	(Rx) ← STS5, AC ← STS5

MRI(目前在 EV chip (TM8999)不支援)

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將@HL 所指向的 data RAM 的內容值(Rx)3~0 依照 X1,0=0/1/2/3 儲存到 16 bits 的 RILH 暫存器的 bit3~0/7~4/11~8/15~12 中，作為執行 PTR 指令時燒錄到由@HL 所指向 Table ROM word address 的資料之用。

註記：
指令結束之後部分語法會自動將 HL 的內容值加 1。

指令語法：

OP code	運算元	指令動作
MRI	X	RILH3~0 ← (@HL) if X=0 RILH7~4 ← (@HL) if X=1 RILH11~8 ← (@HL) if X=2 RILH15~12 ← (@HL) if X=3
MRI#	X	RILH3~0 ← (@HL) if X=0 RILH7~4 ← (@HL) if X=1

		RILH11~8 ← (@HL) if X=2 RILH15~12 ← (@HL) if X=3 HL ← HL+1

MRIH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

將@HL 或者@ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值的 bit15~0 儲存到 16 bits 的 RILH 暫存器的 bit15~0 中，作為執行 PTR 指令時燒錄到由@HL 所指向 Table ROM word address 的資料之用。

註記：

指令結束之後部分語法會自動將 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
MRIH	@HL	RILH15~0 ← (@HL)H
MRIH#	@HL	RILH15~0 ← (@HL)H HL ← HL+4
MRIH	@ZR	RILH15~0 ← (@ZR)H
MRIH#	@ZR	RILH15~0 ← (@ZR)H ZR ← ZR+4

2-5. 算數/邏輯運算指令(Arithmatic/Logic Operation Instructions)

MULH

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 進位的乘法運算指令，將 Rx, @HL 或是@ZR 所指向的 data RAM 的 4 bits 內容值設定為乘法運算中的被乘數，然後與 MUI 的內容值(乘數)作乘法運算。其運算結果為 8 bit data，分別儲存到 MU 以及 AC 中。

乘法運算結果與 MU, AC 位元資料的對應關係如下：

Result	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Destination register	MU3	MU2	MU1	MU0	AC3	AC2	AC1	AC0

註記：

(1). 在執行指令之前必須先確認被乘數以及乘數都必須是 2 進位的格式，其運算結果也是 2 進位的格式。

(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

(3). Rx 在語法上必須是以絕對位址來描述。

(4). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 MULH 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)設定為乘法運算中的被乘數，然後與 MUIH 的內容值(乘數)作乘法運算。其運算結果為 32 bit data，分別儲存到 MUH 以及 ACH 中。

指令語法：

OP code	運算元	指令動作
MULH	Rx	MU(H) ← High Nibble of (Rx)(H) * MUI(H) AC(H) ← Low Nibble of (Rx)(H) * MUI(H)
MULH	@HL	MU ← High Nibble of (@HL) * MUI AC ← Low Nibble of (@HL) * MUI
MULH#	@HL	MU ← High Nibble of (@HL) * MUI AC ← Low Nibble of (@HL) * MUI HL ← HL+1
MULH	@ZR	MU ← High Nibble of (@ZR) * MUI AC ← Low Nibble of (@ZR) * MUI
MULH#	@ZR	MU ← High Nibble of (@ZR) * MUI AC ← Low Nibble of (@ZR) * MUI ZR ← ZR+1

MULHH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

2 進位的乘法運算指令，將@HL 或是@ZR 所指向的 data RAM 的 16 bits 內容值設定為乘法運算中的被乘數，然後與 MUIH 的內容值(乘數)作乘法運算。其運算結果為 32 bit data，分別將 bit:31~16 儲存到 MUH 以及 bit:15~0 儲存到 ACH 中。

註記：

(1). 在執行指令之前必須先確認被乘數以及乘數都必須是 2 進位的格式，其運算結果也是 2 進位的格式。

(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
MULHH	@HL	MUH ← High Word of (@HL)H * MUIH ACH ← Low Word of (@HL)H * MUIH
MULHH#	@HL	MUH ← High Word of (@HL)H * MUIH ACH ← Low Word of (@HL)H * MUIH HL ← HL+4
MULHH	@ZR	MUH ← High Word of (@ZR)H * MUIH ACH ← Low Word of (@ZR)H * MUIH
MULHH#	@ZR	MUH ← High Word of (@ZR)H * MUIH ACH ← Low Word of (@ZR)H * MUIH ZR ← ZR+4

MULD

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

10 進位的乘法運算指令，將 Rx, @HL 或是@ZR 所指向的 data RAM 的 4 bits 內容值設定為乘法運算中的被乘數，然後與 MUI 的內容值(乘數)作乘法運算。其運算結果為 8 bit data，分別儲存到 MU 以及 AC 中。

乘法運算結果與 MU, AC register 位元資料的對應關係如下：

Result	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Destination register	MU3	MU2	MU1	MU0	AC3	AC2	AC1	AC0

註記：

(1). 在執行指令之前必須先確認被乘數以及乘數都必須是 10 進位的格式，其運算結果也是 10 進位的格式。

(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

(3). Rx 在語法上必須是以絕對位址來描述。

(4). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 MULH 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)設定為乘法運算中的被乘數，然後與 MUIH 的內容值(乘數)作乘法運算。其運算結果為 32 bit data，分別儲存到 MUH 以及 ACH 中。

指令語法：

OP code	運算元	指令動作
MULD	Rx	$MU(H)_{10}$ ← High Nibble of $(Rx)(H)_{10}$ * $MUI(H)_{10}$ $AC(H)_{10}$ ← Low Nibble of $(Rx)(H)_{10}$ * $MUI(H)_{10}$
MULD	@HL	MU_{10} ← High Nibble $(@HL)_{10}$ * MUI_{10} AC_{10} ← Low Nibble $(@HL)_{10}$ * MUI_{10}
MULD#	@HL	MU_{10} ← High Nibble $(@HL)_{10}$ * MUI_{10} AC_{10} ← Low Nibble $(@HL)_{10}$ * MUI_{10} HL ← HL+1
MULD	@ZR	MU_{10} ← High Nibble $(@ZR)_{10}$ * MUI_{10} AC_{10} ← Low Nibble $(@ZR)_{10}$ * MUI_{10}
MULD#	@ZR	MU_{10} ← High Nibble $(@ZR)_{10}$ * MUI_{10} AC_{10} ← Low Nibble $(@ZR)_{10}$ * MUI_{10} ZR ← ZR+1

MULDH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

10 進位的乘法運算指令，將@HL 或是@ZR 所指向的 data RAM 的 16 bits 內容值設定為乘法運算中的被乘數，然後與 MUIH 的內容值(乘數)作乘法運算。其運算結果為 32 bit data，分別將 bit:31~16 儲存到 MUH 以及 bit:15~0 儲存到 ACH 中。

註記：

(1). 在執行指令之前必須先確認被乘數以及乘數都必須是 10 進位的格式，其運算結果也是 10 進位的格式。

(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
MULDH	@HL	MUH_{10} ← High Word $(@HL)H_{10}$ * $MUIH_{10}$ ACH_{10} ← Low Word $(@HL)H_{10}$ * $MUIH_{10}$
MULDH#	@HL	MUH_{10} ← High Word $(@HL)H_{10}$ * $MUIH_{10}$ ACH_{10} ← Low Word $(@HL)H_{10}$ * $MUIH_{10}$ HL ← HL+4
MULDH	@ZR	MUH_{10} ← High Word $(@ZR)H_{10}$ * $MUIH_{10}$ ACH_{10} ← Low Word $(@ZR)H_{10}$ * $MUIH_{10}$
MULDH#	@ZR	MUH_{10} ← High Word $(@ZR)H_{10}$ * $MUIH_{10}$ ACH_{10} ← Low Word $(@ZR)H_{10}$ * $MUIH_{10}$ ZR ← ZR+4

ADC

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 或是 10 進位的加法運算指令，將 Rx 或是@HL/ZR 所指向的 data RAM 的內容值與 AC 的內容值或是@ZR/HL 所指向的 data RAM 的內容值(目前在 EV chip (TM8999)不支援)以及 CF 作加法運算。其運算結果會儲存到 AC，亦可選擇儲存到 Rx，@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

2 或是 10 進位加法運算的判別方式是以 OP code 中是否有"^"或者運算元中是否有"DA"作為區分，若使用"^"或者"DA"則代表 10 進位加法運算，若無則代表 2 進位加法運算。

註記：

(1). 必須確認被加數以及加數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此加法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(4). Rx 在語法上必須是以絕對位址來描述。
(5). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 ADC 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)，16bits mode AC(ACH)的內容值以及 CF 作加法運算。其運算結果會分別儲存到 ACH 或是 Rx 所指向的 16bits mode data RAM。
(6). 若 MCU 有提供藉由 Rm 設定 RXDA=1(目前在 EV chip (TM8999)不支援)則可以將"ADC/SBC/ADD/SUB Rx"指令切換成 10 進位運算。

指令語法：

OP code	運算元	指令動作
ADC	Rx	$AC(H)_{(10)} \leftarrow (Rx)(H)_{(10)} + AC(H)_{(10)} + CF$
ADC*	Rx	$AC(H)_{(10)}, (Rx)(H)_{(10)} \leftarrow (Rx)(H)_{(10)} + AC(H)_{(10)} + CF$
ADC	@HL	$AC \leftarrow (@HL) + AC + CF$
ADC#	@HL	$AC \leftarrow (@HL) + AC + CF$ $HL \leftarrow HL+1$
ADC*	@HL	$AC, (@HL) \leftarrow (@HL) + AC + CF$
ADC*#	@HL	$AC, (@HL) \leftarrow (@HL) + AC + CF$ $HL \leftarrow HL+1$
ADC	@HL, DA	$AC_{10} \leftarrow (@HL)_{10} + AC_{10} + CF$
ADC#	@HL, DA	$AC_{10} \leftarrow (@HL)_{10} + AC_{10} + CF$ $HL \leftarrow HL+1$
ADC*	@HL, DA	$AC_{10}, (@HL)_{10} \leftarrow (@HL)_{10} + AC_{10} + CF$
ADC*#	@HL, DA	$AC_{10}, (@HL)_{10} \leftarrow (@HL)_{10} + AC_{10} + CF$ $HL \leftarrow HL+1$
ADC	@HL,@ZR	$AC \leftarrow (@HL) + (@ZR) + CF$ if ALUHLZR=0 $AC, (@HL) \leftarrow (@ZR) + AC + CF$ if ALUHLZR=1
ADC&	@HL,@ZR	$AC \leftarrow (@HL) + (@ZR) + CF$ if ALUHLZR=0 $AC, (@HL) \leftarrow (@ZR) + AC + CF$ if ALUHLZR=1 $HL \leftarrow HL+1$
ADC%	@HL,@ZR	$AC \leftarrow (@HL) + (@ZR) + CF$ if ALUHLZR=0 $AC, (@HL) \leftarrow (@ZR) + AC + CF$ if ALUHLZR=1 $ZR \leftarrow ZR+1$
ADC$	@HL,@ZR	$AC \leftarrow (@HL) + (@ZR) + CF$ if ALUHLZR=0 $AC, (@HL) \leftarrow (@ZR) + AC + CF$ if ALUHLZR=1 $HL \leftarrow HL+1$ $ZR \leftarrow ZR+1$
ADC*	@HL,@ZR	$AC, (@HL) \leftarrow (@HL) + (@ZR) + CF$ if ALUHLZR=0 $AC, (@HL) \leftarrow (@ZR) + AC + CF$ if ALUHLZR=1

OP code	運算元	指令動作
ADC*&	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR) +CF if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC +CF if ALUHLZR=1 HL ← HL+1
ADC*%	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR) +CF if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC +CF if ALUHLZR=1 ZR ← ZR+1
ADC*$	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR) +CF if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADC^	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} +CF if ALUHLZR=1
ADC^&	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} +CF if ALUHLZR=1 HL ← HL+1
ADC^%	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} +CF if ALUHLZR=1 ZR ← ZR+1
ADC^$	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADC^*	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} +CF if ALUHLZR=1
ADC^*&	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} +CF if ALUHLZR=1 HL ← HL+1
ADC^*%	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} +CF if ALUHLZR=1 ZR ← ZR+1
ADC^*$	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADC	@ZR	AC ← (@ZR) + AC +CF
ADC#	@ZR	AC ← (@ZR) + AC +CF ZR ← ZR+1
ADC*	@ZR	AC , (@ZR) ← (@ZR) + AC +CF
ADC*#	@ZR	AC , (@ZR) ← (@ZR) + AC +CF ZR ← ZR+1
ADC	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + AC_{10} +CF
ADC#	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + AC_{10} +CF ZR ← ZR+1
ADC*	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + AC_{10} +CF
ADC*#	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + AC_{10} +CF ZR ← ZR+1
ADC	@ZR,@HL	AC ← (@ZR) + (@HL) +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC +CF if ALUHLZR=1
ADC&	@ZR,@HL	AC ← (@ZR) + (@HL) +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC +CF if ALUHLZR=1 HL ← HL+1
ADC%	@ZR,@HL	AC ← (@ZR) + (@HL) +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC +CF if ALUHLZR=1 ZR ← ZR+1
ADC$	@ZR,@HL	AC ← (@ZR) + (@HL) +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC +CF if ALUHLZR=1 HL ← HL+1

OP code	運算元	指令動作
		ZR ← ZR+1
ADC*	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC +CF if ALUHLZR=1
ADC*&	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC +CF if ALUHLZR=1 HL ← HL+1
ADC*%	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC +CF if ALUHLZR=1 ZR ← ZR+1
ADC*$	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADC^	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} +CF if ALUHLZR=1
ADC^&	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} +CF if ALUHLZR=1 HL ← HL+1
ADC^%	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} +CF if ALUHLZR=1 ZR ← ZR+1
ADC^$	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADC^*	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} +CF if ALUHLZR=1
ADC^*&	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} +CF if ALUHLZR=1 HL ← HL+1
ADC^*%	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} +CF if ALUHLZR=1 ZR ← ZR+1
ADC^*$	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1

ADCH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

2 或是 10 進位的加法運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode AC(ACH)的內容值或是@ZR/HL 所指向的 16bits mode data RAM 的內容值以及 CF 作加法運算。其運算結果會儲存到 ACH，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

2 或是 10 進位加法運算的判別方式是以 OP code 中是否有“^”或者運算元中是否有“DA”作為區分，若使用”^”或者”DA”則代表 10 進位加法運算，若無則代表 2 進位加法運算。

註記：

(1). 必須確認被加數以及加數以及運算指令都是相同的 2 或是 10 進位的格式。

(2). 此加法運算結果的進位會改變 CF 內容值。

(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
ADCH	@HL	ACH ← (@HL)H + ACH +CF
ADCH#	@HL	ACH ← (@HL)H + ACH +CF HL ← HL+4
ADCH*	@HL	ACH , (@HL)H ← (@HL)H + ACH +CF
ADCH*#	@HL	ACH , (@HL)H ← (@HL)H + ACH +CF HL ← HL+4
ADCH	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + ACH_{10} +CF
ADCH#	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + ACH_{10} +CF HL ← HL+4
ADCH*	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} +CF
ADCH*#	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} +CF HL ← HL+4
ADCH	@HL,@ZR	ACH ← (@HL)H + (@ZR)H +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH +CF if ALUHLZR=1
ADCH&	@HL,@ZR	ACH ← (@HL)H + (@ZR)H +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH +CF if ALUHLZR=1 HL ← HL+4
ADCH%	@HL,@ZR	ACH ← (@HL)H + (@ZR)H +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH +CF if ALUHLZR=1 ZR ← ZR+4
ADCH$	@HL,@ZR	ACH ← (@HL)H + (@ZR)H +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADCH*	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH +CF if ALUHLZR=1
ADCH*&	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH +CF if ALUHLZR=1 HL ← HL+4
ADCH*%	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH +CF if ALUHLZR=1 ZR ← ZR+4
ADCH*$	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADCH^	@HL,@ZR	ACH_{10} ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} +CF if ALUHLZR=1
ADCH^&	@HL,@ZR	ACH_{10} ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} +CF if ALUHLZR=1 HL ← HL+4
ADCH^%	@HL,@ZR	ACH_{10} ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} +CF if ALUHLZR=1 ZR ← ZR+4
ADCH^$	@HL,@ZR	ACH_{10} ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADCH^*	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ +CF if ALUHLZR=0

OP code	運算元	指令動作
		ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} +CF if ALUHLZR=1
ADCH^*&	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} +CF if ALUHLZR=1 HL ← HL+4
ADCH^*%	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} +CF if ALUHLZR=1 ZR ← ZR+4
ADCH^*$	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADCH	@ZR	ACH ← (@ZR)H + ACH +CF
ADCH#	@ZR	ACH ← (@ZR)H + ACH +CF ZR ← ZR+4
ADCH*	@ZR	ACH , (@ZR)H ← (@ZR)H + ACH +CF
ADCH*#	@ZR	ACH , (@ZR)H ← (@ZR)H + ACH +CF ZR ← ZR+4
ADCH	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + ACH_{10} +CF
ADCH#	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + ACH_{10} +CF ZR ← ZR+4
ADCH*	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} +CF
ADCH*#	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} +CF ZR ← ZR+4
ADCH	@ZR,@HL	ACH ← (@ZR)H + (@HL)H +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH +CF if ALUHLZR=1
ADCH&	@ZR,@HL	ACH ← (@ZR)H + (@HL)H +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH +CF if ALUHLZR=1 HL ← HL+4
ADCH%	@ZR,@HL	ACH ← (@ZR)H + (@HL)H +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH +CF if ALUHLZR=1 ZR ← ZR+4
ADCH$	@ZR,@HL	ACH ← (@ZR)H + (@HL)H +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADCH*	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH +CF if ALUHLZR=1
ADCH*&	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH +CF if ALUHLZR=1 HL ← HL+4
ADCH*%	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH +CF if ALUHLZR=1 ZR ← ZR+4
ADCH*$	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADCH^	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} +CF if ALUHLZR=1
ADCH^&	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} +CF if ALUHLZR=1 HL ← HL+4
ADCH^%	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} +CF if ALUHLZR=1 ZR ← ZR+4
ADCH^$	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} +CF if ALUHLZR=1

OP code	運算元	指令動作
		HL ← HL+4 ZR ← ZR+4
ADCH^*	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} +CF if ALUHLZR=1
ADCH^*&	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} +CF if ALUHLZR=1 HL ← HL+4
ADCH^*%	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} +CF if ALUHLZR=1 ZR ← ZR+4
ADCH^*$	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4

ADCM

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 或是 10 進位的加法運算指令，將@HL/ZR 所指向的 data RAM 的內容值與 MU(I)的內容值以及 CF 作加法運算。其運算結果會儲存到 AC，亦可選擇儲存到@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

2 或是 10 進位加法運算的判別方式是以 OP code 中是否有“^”或者運算元中是否有“DA”作為區分，若使用”^”或者”DA”則代表 10 進位加法運算，若無則代表 2 進位加法運算。

註記：

(1). 必須確認被加數以及加數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此加法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(4). 若 MCU 有提供藉由 Rm 設定 ALUMUI=1(目前在 EV chip (TM8999)不支援)則可將讀取的 MU 切換至 MUI。

指令語法：

OP code	運算元	指令動作
ADCM	@HL	AC ← (@HL) + MU(I) +CF
ADCM#	@HL	AC ← (@HL) + MU(I) +CF HL ← HL+1
ADCM*	@HL	AC , (@HL) ← (@HL) + MU(I) +CF
ADCM*#	@HL	AC , (@HL) ← (@HL) + MU(I) +CF HL ← HL+1
ADCM	@HL, DA	AC_{10} ← $(@HL)_{10}$ + $MU(I)_{10}$ +CF
ADCM#	@HL, DA	AC_{10} ← $(@HL)_{10}$ + $MU(I)_{10}$ +CF HL ← HL+1
ADCM*	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$ +CF
ADCM*#	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$ +CF HL ← HL+1

ADCM	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I) +CF
ADCM&	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I) +CF HL ← HL+1
ADCM%	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I) +CF ZR ← ZR+1
ADCM$	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I) +CF HL ← HL+1 ZR ← ZR+1
ADCM^	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$ +CF
ADCM^&	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$ +CF HL ← HL+1
ADCM^%	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$ +CF ZR ← ZR+1
ADCM^$	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$ +CF HL ← HL+1 ZR ← ZR+1
ADCM	@ZR	AC ← (@ZR) + MU(I) +CF
ADCM#	@ZR	AC ← (@ZR) + MU(I) +CF ZR ← ZR+1
ADCM*	@ZR	AC , (@ZR) ← (@ZR) + MU(I) +CF
ADCM*#	@ZR	AC , (@ZR) ← (@ZR) + MU(I) +CF ZR ← ZR+1
ADCM	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + $MU(I)_{10}$ +CF
ADCM#	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + $MU(I)_{10}$ +CF ZR ← ZR+1
ADCM*	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$ +CF
ADCM*#	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$ +CF ZR ← ZR+1
ADCM	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I) +CF
ADCM&	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I) +CF HL ← HL+1
ADCM%	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I) +CF ZR ← ZR+1
ADCM$	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I) +CF HL ← HL+1 ZR ← ZR+1
ADCM^	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$ +CF
ADCM^&	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$ +CF HL ← HL+1
ADCM^%	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$ +CF ZR ← ZR+1
ADCM^$	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$ +CF HL ← HL+1 ZR ← ZR+1

ADCMH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

2 或是 10 進位的加法運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode MU(I)(MU(I)H)的內容值以及 CF 作加法運算。其運算結果會儲存

到 16bits mode AC(ACH)，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

2 或是 10 進位加法運算的判別方式是以 OP code 中是否有"^"或者運算元中是否有"DA"作為區分，若使用"^"或者"DA"則代表 10 進位加法運算，若無則代表 2 進位加法運算。

註記：

(1). 必須確認被加數以及加數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此加法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。
(4). 若 MCU 有提供藉由 Rm 設定 ALUMUI=1(目前在 EV chip (TM8999)不支援)則可將讀取的 MUH 切換至 MUIH。

指令語法：

OP code	運算元	指令動作
ADCMH	@HL	ACH ← (@HL)H + MU(I)H +CF
ADCMH#	@HL	ACH ← (@HL)H + MU(I)H +CF HL ← HL+4
ADCMH*	@HL	ACH , (@HL)H ← (@HL)H + MU(I)H +CF
ADCMH*#	@HL	ACH , (@HL)H ← (@HL)H + MU(I)H +CF HL ← HL+4
ADCMH	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $MU(I)H_{10}$ +CF
ADCMH#	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $MU(I)H_{10}$ +CF HL ← HL+4
ADCMH*	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$ +CF
ADCMH*#	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$ +CF HL ← HL+4
ADCMH	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)H +CF
ADCMH&	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)H +CF HL ← HL+4
ADCMH%	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)H +CF ZR ← ZR+4
ADCMH$	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)H +CF HL ← HL+4 ZR ← ZR+4
ADCMH^	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ +CF
ADCMH^&	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ +CF HL ← HL+4
ADCMH^%	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ +CF ZR ← ZR+4
ADCMH^$	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ +CF HL ← HL+4 ZR ← ZR+4
ADCMH	@ZR	ACH ← (@ZR)H + MU(I)H +CF
ADCMH#	@ZR	ACH ← (@ZR)H + MU(I)H +CF ZR ← ZR+4
ADCMH*	@ZR	ACH , (@ZR)H ← (@ZR)H + MU(I)H +CF
ADCMH*#	@ZR	ACH , (@ZR)H ← (@ZR)H + MU(I)H +CF ZR ← ZR+4
ADCMH	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ +CF
ADCMH#	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ +CF ZR ← ZR+4
ADCMH*	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ +CF
ADCMH*#	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ +CF

		ZR ← ZR+4
ADCMH	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)H +CF
ADCMH&	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)H +CF HL ← HL+4
ADCMH%	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)H +CF ZR ← ZR+4
ADCMH$	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)H +CF HL ← HL+4 ZR ← ZR+4
ADCMH^	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$ +CF
ADCMH^&	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$ +CF HL ← HL+4
ADCMH^%	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$ +CF ZR ← ZR+4
ADCMH^$	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$ +CF HL ← HL+4 ZR ← ZR+4

SBC

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 或是 10 進位的減法運算指令，將 Rx 或是@HL/ZR 所指向的 data RAM 的內容值(被減數)與 AC 的內容值或是@ZR/HL 所指向的 data RAM 的內容值(減數)以及 CF(借位)作減法運算。其運算結果會儲存到 AC，亦可選擇儲存到 Rx，@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

2 或是 10 進位減法運算的判別方式是以 OP code 中是否有"^"或者運算元中是否有"DA"作為區分，若使用"^"或者"DA"則代表 10 進位減法運算，若無則代表 2 進位減法運算。

註記：

(1). 必須確認被減數以及減數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此減法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(4). Rx 在語法上必須是以絕對位址來描述。
(5). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 SBC 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)，ACH 的內容值以及 CF 作減法運算。其運算結果會分別儲存到 ACH 或是 Rx 所指向的 16bits mode data RAM。
(6). 若 MCU 有提供藉由 Rm 設定 RXDA=1(目前在 EV chip (TM8999)不支援)則可以將"ADC/SBC/ADD/SUB Rx"指令切換成 10 進位運算。

指令語法：

OP code	運算元	指令動作
SBC	Rx	$AC(H)_{(10)}$ ← $(Rx)(H)_{(10)}$ + $ACB(H)_{(10)}$ +CF
SBC*	Rx	$AC(H)_{(10)}$, $(Rx)(H)_{(10)}$ ← $(Rx)(H)_{(10)}$ + $ACB(H)_{(10)}$ +CF

OP code	運算元	指令動作
SBC	@HL	AC ← (@HL) + ACB +CF
SBC#	@HL	AC ← (@HL) + ACB +CF HL ← HL+1
SBC*	@HL	AC , (@HL) ← (@HL) + ACB +CF
SBC*#	@HL	AC , (@HL) ← (@HL) + ACB +CF HL ← HL+1
SBC	@HL, DA	AC_{10} ← $(@HL)_{10}$ + ACB_{10} +CF
SBC#	@HL, DA	AC_{10} ← $(@HL)_{10}$ + ACB_{10} +CF HL ← HL+1
SBC*	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +CF
SBC*#	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +CF HL ← HL+1
SBC	@HL,@ZR	AC ← (@HL) + (@ZR)B +CF if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +CF if ALUHLZR=1
SBC&	@HL,@ZR	AC ← (@HL) + (@ZR)B +CF if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +CF if ALUHLZR=1 HL ← HL+1
SBC%	@HL,@ZR	AC ← (@HL) + (@ZR)B +CF if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +CF if ALUHLZR=1 ZR ← ZR+1
SBC$	@HL,@ZR	AC ← (@HL) + (@ZR)B +CF if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SBC*	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR)B +CF if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +CF if ALUHLZR=1
SBC*&	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR)B +CF if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +CF if ALUHLZR=1 HL ← HL+1
SBC*%	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR)B +CF if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +CF if ALUHLZR=1 ZR ← ZR+1
SBC*$	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR)B +CF if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SBC^	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +CF if ALUHLZR=1
SBC^&	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +CF if ALUHLZR=1 HL ← HL+1
SBC^%	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +CF if ALUHLZR=1 ZR ← ZR+1
SBC^$	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SBC^*	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +CF if ALUHLZR=1
SBC^*&	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +CF if ALUHLZR=1 HL ← HL+1
SBC^*%	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +CF if ALUHLZR=1 ZR ← ZR+1
SBC^*$	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +CF if ALUHLZR=0

OP code	運算元	指令動作
		AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SBC	@ZR	AC ← (@ZR) + ACB +CF
SBC#	@ZR	AC ← (@ZR) + ACB +CF ZR ← ZR+1
SBC*	@ZR	AC , (@ZR) ← (@ZR) + ACB +CF
SBC*#	@ZR	AC , (@ZR) ← (@ZR) + ACB +CF ZR ← ZR+1
SBC	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + ACB_{10} +CF
SBC#	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + ACB_{10} +CF ZR ← ZR+1
SBC*	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +CF
SBC*#	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +CF ZR ← ZR+1
SBC	@ZR,@HL	AC ← (@ZR) + (@HL)B +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +CF if ALUHLZR=1
SBC&	@ZR,@HL	AC ← (@ZR) + (@HL)B +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +CF if ALUHLZR=1 HL ← HL+1
SBC%	@ZR,@HL	AC ← (@ZR) + (@HL)B +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +CF if ALUHLZR=1 ZR ← ZR+1
SBC$	@ZR,@HL	AC ← (@ZR) + (@HL)B +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SBC*	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL)B +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +CF if ALUHLZR=1
SBC*&	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL)B +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +CF if ALUHLZR=1 HL ← HL+1
SBC*%	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL)B +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +CF if ALUHLZR=1 ZR ← ZR+1
SBC*$	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL)B +CF if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SBC^	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +CF if ALUHLZR=1
SBC^&	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +CF if ALUHLZR=1 HL ← HL+1
SBC^%	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +CF if ALUHLZR=1 ZR ← ZR+1
SBC^$	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SBC^*	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +CF if ALUHLZR=1
SBC^*&	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +CF if ALUHLZR=1 HL ← HL+1
SBC^*%	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +CF if ALUHLZR=0

OP code	運算元	指令動作
		AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +CF if ALUHLZR=1 ZR ← ZR+1
SBC^*$	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +CF if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +CF if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1

SBCH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

2 或是 10 進位的減法運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值(被減數)與 16bits mode AC(ACH)的內容值或是@ZR/HL 所指向的 16bits mode data RAM 的內容值(減數)以及 CF(借位)作減法運算。其運算結果會儲存到 ACH，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

2 或是 10 進位減法運算的判別方式是以 OP code 中是否有"^"或者運算元中是否有"DA"作為區分，若使用"^"或者"DA"則代表 10 進位減法運算，若無則代表 2 進位減法運算。

註記：

(1). 必須確認被減數以及減數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此減法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
SBCH	@HL	ACH ← (@HL)H + ACHB +CF
SBCH#	@HL	ACH ← (@HL)H + ACHB +CF HL ← HL+4
SBCH*	@HL	ACH , (@HL)H ← (@HL)H + ACHB +CF
SBCH*#	@HL	ACH , (@HL)H ← (@HL)H + ACHB +CF HL ← HL+4
SBCH	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF
SBCH#	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF HL ← HL+4
SBCH*	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF
SBCH*#	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF HL ← HL+4
SBCH	@HL,@ZR	ACH ← (@HL)H + (@ZR)HB +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +CF if ALUHLZR=1
SBCH&	@HL,@ZR	ACH ← (@HL)H + (@ZR)HB +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +CF if ALUHLZR=1 HL ← HL+4
SBCH%	@HL,@ZR	ACH ← (@HL)H + (@ZR)HB +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +CF if ALUHLZR=1 ZR ← ZR+4
SBCH$	@HL,@ZR	ACH ← (@HL)H + (@ZR)HB +CF if ALUHLZR=0

OP code	運算元	指令動作
		ACH , (@HL)H ← (@ZR)H + ACHB +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SBCH*	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)HB +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +CF if ALUHLZR=1
SBCH*&	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)HB +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +CF if ALUHLZR=1 HL ← HL+4
SBCH*%	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)HB +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +CF if ALUHLZR=1 ZR ← ZR+4
SBCH*$	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)HB +CF if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SBCH^	@HL,@ZR	ACH_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +CF if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF if ALUHLZR=1
SBCH^&	@HL,@ZR	ACH_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +CF if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF if ALUHLZR=1 HL ← HL+4
SBCH^%	@HL,@ZR	ACH_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +CF if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF if ALUHLZR=1 ZR ← ZR+4
SBCH^$	@HL,@ZR	ACH_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +CF if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SBCH^*	@HL,@ZR	ACH_{10} , (@HL)H_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +CF if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF if ALUHLZR=1
SBCH^*&	@HL,@ZR	ACH_{10} , (@HL)H_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +CF if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF if ALUHLZR=1 HL ← HL+4
SBCH^*%	@HL,@ZR	ACH_{10} , (@HL)H_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +CF if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF if ALUHLZR=1 ZR ← ZR+4
SBCH^*$	@HL,@ZR	ACH_{10} , (@HL)H_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +CF if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SBCH	@ZR	ACH ← (@ZR)H + ACHB +CF
SBCH#	@ZR	ACH ← (@ZR)H + ACHB +CF ZR ← ZR+4
SBCH*	@ZR	ACH , (@ZR)H ← (@ZR)H + ACHB +CF
SBCH*#	@ZR	ACH , (@ZR)H ← (@ZR)H + ACHB +CF ZR ← ZR+4
SBCH	@ZR, DA	ACH_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF
SBCH#	@ZR, DA	ACH_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF ZR ← ZR+4
SBCH*	@ZR, DA	ACH_{10} , (@ZR)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF
SBCH*#	@ZR, DA	ACH_{10} , (@ZR)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +CF ZR ← ZR+4
SBCH	@ZR,@HL	ACH ← (@ZR)H + (@HL)HB +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +CF if ALUHLZR=1
SBCH&	@ZR,@HL	ACH ← (@ZR)H + (@HL)HB +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +CF if ALUHLZR=1 HL ← HL+4
SBCH%	@ZR,@HL	ACH ← (@ZR)H + (@HL)HB +CF if ALUHLZR=0

OP code	運算元	指令動作
		ACH , (@ZR)H ← (@HL)H + ACHB +CF if ALUHLZR=1 ZR ← ZR+4
SBCH$	@ZR,@HL	ACH ← (@ZR)H + (@HL)HB +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SBCH*	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)HB +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +CF if ALUHLZR=1
SBCH*&	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)HB +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +CF if ALUHLZR=1 HL ← HL+4
SBCH*%	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)HB +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +CF if ALUHLZR=1 ZR ← ZR+4
SBCH*$	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)HB +CF if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SBCH^	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF if ALUHLZR=1
SBCH^&	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF if ALUHLZR=1 HL ← HL+4
SBCH^%	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF if ALUHLZR=1 ZR ← ZR+4
SBCH^$	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SBCH^*	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF if ALUHLZR=1
SBCH^*&	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF if ALUHLZR=1 HL ← HL+4
SBCH^*%	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF if ALUHLZR=1 ZR ← ZR+4
SBCH^*$	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +CF if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +CF if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4

SBCM

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 或是 10 進位的減法運算指令，將 Rx 或是@HL/ZR 所指向的 RAM 的內容值(被減數)與 MU(I)的內容值(減數)以及 CF(借位)作減法運算。其運算結果會儲存到 AC，亦可選擇儲存到 Rx，@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

2 或是 10 進位減法運算的判別方式是以 OP code 中是否有“^”或者運算元中是否有“DA”作為區分，若使用”^”或者”DA”則代表 10 進位減法運算，若無則代表 2 進位減法運算。

註記：

(1). 必須確認被減數以及減數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此減法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

指令語法：

OP code	運算元	指令動作
SBCM	@HL	AC ← (@HL) + MU(I)B +CF
SBCM#	@HL	AC ← (@HL) + MU(I)B +CF HL ← HL+1
SBCM*	@HL	AC , (@HL) ← (@HL) + MU(I)B +CF
SBCM*#	@HL	AC , (@HL) ← (@HL) + MU(I)B +CF HL ← HL+1
SBCM	@HL, DA	AC_{10} ← $(@HL)_{10}$ + $MU(I)B_{10}$ +CF
SBCM#	@HL, DA	AC_{10} ← $(@HL)_{10}$ + $MU(I)B_{10}$ +CF HL ← HL+1
SBCM*	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +CF
SBCM*#	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +CF HL ← HL+1
SBCM	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I)B +CF
SBCM&	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I)B +CF HL ← HL+1
SBCM%	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I)B +CF ZR ← ZR+1
SBCM$	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I)B +CF HL ← HL+1 ZR ← ZR+1
SBCM^	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +CF
SBCM^&	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +CF HL ← HL+1
SBCM^%	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +CF ZR ← ZR+1
SBCM^$	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +CF HL ← HL+1 ZR ← ZR+1
SBCM	@ZR	AC ← (@ZR) + MU(I)B +CF
SBCM#	@ZR	AC ← (@ZR) + MU(I)B +CF ZR ← ZR+1
SBCM*	@ZR	AC , (@ZR) ← (@ZR) + MU(I)B +CF
SBCM*#	@ZR	AC , (@ZR) ← (@ZR) + MU(I)B +CF ZR ← ZR+1
SBCM	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +CF
SBCM#	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +CF ZR ← ZR+1
SBCM*	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +CF
SBCM*#	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +CF ZR ← ZR+1
SBCM	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I)B +CF
SBCM&	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I)B +CF HL ← HL+1
SBCM%	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I)B +CF ZR ← ZR+1

SBCM$	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I)B +CF HL ← HL+1 ZR ← ZR+1
SBCM^	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +CF
SBCM^&	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +CF HL ← HL+1
SBCM^%	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +CF ZR ← ZR+1
SBCM^$	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +CF HL ← HL+1 ZR ← ZR+1

SBCMH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

2 或是 10 進位的減法運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode MU(I)(MU(I)H)的內容值以及 CF 作減法運算。其運算結果會儲存到 16bits mode AC(ACH)，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

2 或是 10 進位減法運算的判別方式是以 OP code 中是否有“^”或者運算元中是否有“DA”作為區分，若使用”^”或者”DA”則代表 10 進位減法運算，若無則代表 2 進位減法運算。

註記：

(1). 必須確認被減數以及減數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此減法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
SBCMH	@HL	ACH ← (@HL)H + MU(I)HB +CF
SBCMH#	@HL	ACH ← (@HL)H + MU(I)HB +CF HL ← HL+4
SBCMH*	@HL	ACH , (@HL)H ← (@HL)H + MU(I)HB +CF
SBCMH*#	@HL	ACH , (@HL)H ← (@HL)H + MU(I)HB +CF HL ← HL+4
SBCMH	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +CF
SBCMH#	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +CF HL ← HL+4
SBCMH*	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +CF
SBCMH*#	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +CF HL ← HL+4
SBCMH	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)HB +CF
SBCMH&	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)HB +CF HL ← HL+4
SBCMH%	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)HB +CF ZR ← ZR+4

SBCMH$	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)HB +CF HL ← HL+4 ZR ← ZR+4
SBCMH^	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +CF
SBCMH^&	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +CF HL ← HL+4
SBCMH^%	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +CF ZR ← ZR+4
SBCMH^$	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +CF HL ← HL+4 ZR ← ZR+4
SBCMH	@ZR	ACH ← (@ZR)H + MU(I)HB +CF
SBCMH#	@ZR	ACH ← (@ZR)H + MU(I)HB +CF ZR ← ZR+4
SBCMH*	@ZR	ACH , (@ZR)H ← (@ZR)H + MU(I)HB +CF
SBCMH*#	@ZR	ACH , (@ZR)H ← (@ZR)H + MU(I)HB +CF ZR ← ZR+4
SBCMH	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +CF
SBCMH#	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +CF ZR ← ZR+4
SBCMH*	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +CF
SBCMH*#	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +CF ZR ← ZR+4
SBCMH	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)HB +CF
SBCMH&	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)HB +CF HL ← HL+4
SBCMH%	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)HB +CF ZR ← ZR+4
SBCMH$	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)HB +CF HL ← HL+4 ZR ← ZR+4
SBCMH^	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +CF
SBCMH^&	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +CF HL ← HL+4
SBCMH^%	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +CF ZR ← ZR+4
SBCMH^$	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +CF HL ← HL+4 ZR ← ZR+4

ADD

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 或是 10 進位的加法運算指令，將 Rx 或是@HL/ZR 所指向的 data RAM 的內容值與 AC 的內容值或是@ZR/HL 所指向的 data RAM 的內容值(目前在 EV chip (TM8999)不支援)作加法運算。其運算結果會儲存到 AC，亦可選擇儲存到 Rx，@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

2 或是 10 進位加法運算的判別方式是以 OP code 中是否有“^”或者運算元中是否有“DA”作為區分，若使用”^”或者”DA”則代表 10 進位加法運算，若無則代表 2 進位加法運算。

註記：

(1). 必須確認被加數以及加數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此加法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(4). Rx 在語法上必須是以絕對位址來描述。
(5). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 ADD 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)，16bits mode AC(ACH)的內容值以及 CF 作加法運算。其運算結果會分別儲存到 ACH 或是 Rx 所指向的 16bits mode data RAM。
(6). 若 MCU 有提供藉由 Rm 設定 RXDA=1(目前在 EV chip (TM8999)不支援)則可以將"ADC/SBC/ADD/SUB Rx"指令切換成 10 進位運算。

指令語法：

OP code	運算元	指令動作
ADD	Rx	$AC(H)_{(10)} \leftarrow (Rx)(H)_{(10)} + AC(H)_{(10)}$
ADD*	Rx	$AC(H)_{(10)}, (Rx)(H)_{(10)} \leftarrow (Rx)(H)_{(10)} + AC(H)_{(10)}$
ADD	@HL	AC ← (@HL) + AC
ADD#	@HL	AC ← (@HL) + AC HL ← HL+1
ADD*	@HL	AC , (@HL) ← (@HL) + AC
ADD*#	@HL	AC , (@HL) ← (@HL) + AC HL ← HL+1
ADD	@HL, DA	$AC_{10} \leftarrow (@HL)_{10} + AC_{10}$
ADD#	@HL, DA	$AC_{10} \leftarrow (@HL)_{10} + AC_{10}$ HL ← HL+1
ADD*	@HL, DA	$AC_{10}, (@HL)_{10} \leftarrow (@HL)_{10} + AC_{10}$
ADD*#	@HL, DA	$AC_{10}, (@HL)_{10} \leftarrow (@HL)_{10} + AC_{10}$ HL ← HL+1
ADD	@HL,@ZR	AC ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1
ADD&	@HL,@ZR	AC ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 HL ← HL+1
ADD%	@HL,@ZR	AC ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 ZR ← ZR+1
ADD$	@HL,@ZR	AC ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADD*	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1
ADD*&	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 HL ← HL+1
ADD*%	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 ZR ← ZR+1
ADD*$	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADD^	@HL,@ZR	$AC_{10} \leftarrow (@HL)_{10} + (@ZR)_{10}$ if ALUHLZR=0

OP code	運算元	指令動作
		AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} if ALUHLZR=1
ADD^&	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)_{10}$ if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} if ALUHLZR=1 HL ← HL+1
ADD^%	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)_{10}$ if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} if ALUHLZR=1 ZR ← ZR+1
ADD^$	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)_{10}$ if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADD^*	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)_{10}$ if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} if ALUHLZR=1
ADD^*&	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)_{10}$ if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} if ALUHLZR=1 HL ← HL+1
ADD^*%	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)_{10}$ if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} if ALUHLZR=1 ZR ← ZR+1
ADD^*$	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)_{10}$ if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + AC_{10} if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADD	@ZR	AC ← (@ZR) + AC
ADD#	@ZR	AC ← (@ZR) + AC ZR ← ZR+1
ADD*	@ZR	AC , (@ZR) ← (@ZR) + AC
ADD*#	@ZR	AC , (@ZR) ← (@ZR) + AC ZR ← ZR+1
ADD	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + AC_{10}
ADD#	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + AC_{10} ZR ← ZR+1
ADD*	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + AC_{10}
ADD*#	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + AC_{10} ZR ← ZR+1
ADD	@ZR,@HL	AC ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1
ADD&	@ZR,@HL	AC ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1 HL ← HL+1
ADD%	@ZR,@HL	AC ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1 ZR ← ZR+1
ADD$	@ZR,@HL	AC ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADD*	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1
ADD*&	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1 HL ← HL+1
ADD*%	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1 ZR ← ZR+1
ADD*$	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1

OP code	運算元	指令動作
		HL ← HL+1 ZR ← ZR+1
ADD^	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)_{10}$ if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} if ALUHLZR=1
ADD^&	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)_{10}$ if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} if ALUHLZR=1 HL ← HL+1
ADD^%	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)_{10}$ if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} if ALUHLZR=1 ZR ← ZR+1
ADD^$	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)_{10}$ if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADD^*	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)_{10}$ if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} if ALUHLZR=1
ADD^*&	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)_{10}$ if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} if ALUHLZR=1 HL ← HL+1
ADD^*%	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)_{10}$ if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} if ALUHLZR=1 ZR ← ZR+1
ADD^*$	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)_{10}$ if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + AC_{10} if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1

ADDH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

2 或是 10 進位的加法運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode AC(ACH)的內容值或是@ZR/HL 所指向的 16bits mode data RAM 的內容值作加法運算。其運算結果會儲存到 ACH，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

2 或是 10 進位加法運算的判別方式是以 OP code 中是否有“^”或者運算元中是否有“DA”作為區分，若使用”^”或者”DA”則代表 10 進位加法運算，若無則代表 2 進位加法運算。

註記：

(1). 必須確認被加數以及加數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此加法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
ADDH	@HL	ACH ← (@HL)H + ACH

OP code	運算元	指令動作
ADDH#	@HL	ACH ← (@HL)H + ACH HL ← HL+4
ADDH*	@HL	ACH , (@HL)H ← (@HL)H + ACH
ADDH*#	@HL	ACH , (@HL)H ← (@HL)H + ACH HL ← HL+4
ADDH	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + ACH_{10}
ADDH#	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + ACH_{10} HL ← HL+4
ADDH*	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10}
ADDH*#	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} HL ← HL+4
ADDH	@HL,@ZR	ACH ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1
ADDH&	@HL,@ZR	ACH ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1 HL ← HL+4
ADDH%	@HL,@ZR	ACH ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1 ZR ← ZR+4
ADDH$	@HL,@ZR	ACH ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADDH*	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1
ADDH*&	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1 HL ← HL+4
ADDH*%	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1 ZR ← ZR+4
ADDH*$	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADDH^	@HL,@ZR	ACH_{10} ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} if ALUHLZR=1
ADDH^&	@HL,@ZR	ACH_{10} ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} if ALUHLZR=1 HL ← HL+4
ADDH^%	@HL,@ZR	ACH_{10} ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} if ALUHLZR=1 ZR ← ZR+4
ADDH^$	@HL,@ZR	ACH_{10} ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADDH^*	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} if ALUHLZR=1
ADDH^*&	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} if ALUHLZR=1 HL ← HL+4
ADDH^*%	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} if ALUHLZR=1 ZR ← ZR+4
ADDH^*$	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $(@ZR)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} if ALUHLZR=1

OP code	運算元	指令動作
		HL ← HL+4 ZR ← ZR+4
ADDH	@ZR	ACH ← (@ZR)H + ACH
ADDH#	@ZR	ACH ← (@ZR)H + ACH ZR ← ZR+4
ADDH*	@ZR	ACH , (@ZR)H ← (@ZR)H + ACH
ADDH*#	@ZR	ACH , (@ZR)H ← (@ZR)H + ACH ZR ← ZR+4
ADDH	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + ACH_{10}
ADDH#	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + ACH_{10} ZR ← ZR+4
ADDH*	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10}
ADDH*#	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + ACH_{10} ZR ← ZR+4
ADDH	@ZR,@HL	ACH ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1
ADDH&	@ZR,@HL	ACH ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 HL ← HL+4
ADDH%	@ZR,@HL	ACH ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 ZR ← ZR+4
ADDH$	@ZR,@HL	ACH ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADDH*	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1
ADDH*&	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 HL ← HL+4
ADDH*%	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 ZR ← ZR+4
ADDH*$	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADDH^	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} if ALUHLZR=1
ADDH^&	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} if ALUHLZR=1 HL ← HL+4
ADDH^%	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} if ALUHLZR=1 ZR ← ZR+4
ADDH^$	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADDH^*	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} if ALUHLZR=1
ADDH^*&	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} if ALUHLZR=1 HL ← HL+4
ADDH^*%	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} if ALUHLZR=1

OP code	運算元	指令動作
		ZR ← ZR+4
ADDH^*$	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)H_{10}$ if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + ACH_{10} if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4

ADDM

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 或是 10 進位的加法運算指令，將@HL/ZR 所指向的 data RAM 的內容值與 MU(I)的內容值作加法運算。其運算結果會儲存到 AC，亦可選擇儲存到@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

2 或是 10 進位加法運算的判別方式是以 OP code 中是否有"^"或者運算元中是否有"DA"作為區分，若使用"^"或者"DA"則代表 10 進位加法運算，若無則代表 2 進位加法運算。

註記：

(1). 必須確認被加數以及加數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此加法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(4). 若 MCU 有提供藉由 Rm 設定 ALUMUI=1(目前在 EV chip (TM8999)不支援)則可將讀取的 MU 切換至 MUI。

指令語法：

OP code	運算元	指令動作
ADDM	@HL	AC ← (@HL) + MU(I)
ADDM#	@HL	AC ← (@HL) + MU(I) HL ← HL+1
ADDM*	@HL	AC , (@HL) ← (@HL) + MU(I)
ADDM*#	@HL	AC , (@HL) ← (@HL) + MU(I) HL ← HL+1
ADDM	@HL, DA	AC_{10} ← $(@HL)_{10}$ + $MU(I)_{10}$
ADDM#	@HL, DA	AC_{10} ← $(@HL)_{10}$ + $MU(I)_{10}$ HL ← HL+1
ADDM*	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$
ADDM*#	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$ HL ← HL+1
ADDM	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I)
ADDM&	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I) HL ← HL+1
ADDM%	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I) ZR ← ZR+1
ADDM$	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I) HL ← HL+1 ZR ← ZR+1
ADDM^	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$

ADDM^&	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$ HL ← HL+1
ADDM^%	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$ ZR ← ZR+1
ADDM^$	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$ HL ← HL+1 ZR ← ZR+1
ADDM	@ZR	AC ← (@ZR) + MU(I)
ADDM#	@ZR	AC ← (@ZR) + MU(I) ZR ← ZR+1
ADDM*	@ZR	AC , (@ZR) ← (@ZR) + MU(I)
ADDM*#	@ZR	AC , (@ZR) ← (@ZR) + MU(I) ZR ← ZR+1
ADDM	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + $MU(I)_{10}$
ADDM#	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + $MU(I)_{10}$ ZR ← ZR+1
ADDM*	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$
ADDM*#	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $MU(I)_{10}$ ZR ← ZR+1
ADDM	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I)
ADDM&	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I) HL ← HL+1
ADDM%	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I) ZR ← ZR+1
ADDM$	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I) HL ← HL+1 ZR ← ZR+1
ADDM^	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$
ADDM^&	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$ HL ← HL+1
ADDM^%	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$ ZR ← ZR+1
ADDM^$	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)_{10}$ HL ← HL+1 ZR ← ZR+1

ADDMH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

2 或是 10 進位的加法運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode MU(I)(MU(I)H)的內容值作加法運算。其運算結果會儲存到 16bits mode AC(ACH)，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

2 或是 10 進位加法運算的判別方式是以 OP code 中是否有“^”或者運算元中是否有“DA”作為區分，若使用”^”或者”DA”則代表 10 進位加法運算，若無則代表 2 進位加法運算。

註記：

(1). 必須確認被加數以及加數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此加法運算結果的進位會改變 CF 內容值。

(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。
(4). 若 MCU 有提供藉由 Rm 設定 ALUMUI=1(目前在 EV chip (TM8999)不支援)則可將讀取的 MUH 切換至 MUIH。

指令語法：

OP code	運算元	指令動作
ADDMH	@HL	ACH ← (@HL)H + MU(I)H
ADDMH#	@HL	ACH ← (@HL)H + MU(I)H HL ← HL+4
ADDMH*	@HL	ACH , (@HL)H ← (@HL)H + MU(I)H
ADDMH*#	@HL	ACH , (@HL)H ← (@HL)H + MU(I)H HL ← HL+4
ADDMH	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $MU(I)H_{10}$
ADDMH#	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $MU(I)H_{10}$ HL ← HL+4
ADDMH*	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$
ADDMH*#	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$ HL ← HL+4
ADDMH	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)H
ADDMH&	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)H HL ← HL+4
ADDMH%	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)H ZR ← ZR+4
ADDMH$	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)H HL ← HL+4 ZR ← ZR+4
ADDMH^	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$
ADDMH^&	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ HL ← HL+4
ADDMH^%	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ ZR ← ZR+4
ADDMH^$	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ HL ← HL+4 ZR ← ZR+4
ADDMH	@ZR	ACH ← (@ZR)H + MU(I)H
ADDMH#	@ZR	ACH ← (@ZR)H + MU(I)H ZR ← ZR+4
ADDMH*	@ZR	ACH , (@ZR)H ← (@ZR)H + MU(I)H
ADDMH*#	@ZR	ACH , (@ZR)H ← (@ZR)H + MU(I)H ZR ← ZR+4
ADDMH	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + $MU(I)H_{10}$
ADDMH#	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ ZR ← ZR+4
ADDMH*	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$
ADDMH*#	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)H_{10}$ ZR ← ZR+4
ADDMH	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)H
ADDMH&	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)H HL ← HL+4
ADDMH%	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)H ZR ← ZR+4
ADDMH$	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)H HL ← HL+4 ZR ← ZR+4
ADDMH^	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$
ADDMH^&	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$

		HL ← HL+4
ADDMH^%	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$ ZR ← ZR+4
ADDMH^$	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)H_{10}$ HL ← HL+4 ZR ← ZR+4

SUB

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 或是 10 進位的減法運算指令，將 Rx 或是@HL/ZR 所指向的 data RAM 的內容值(被減數)與 AC 的內容值或是@ZR/HL 所指向的 data RAM 的內容值(減數)作沒有借位的減法運算。在此運算中，借位的值會固定為 1(沒有借位發生)。其運算結果會儲存到 AC，亦可選擇儲存到 Rx，@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

2 或是 10 進位減法運算的判別方式是以 OP code 中是否有"^"或者運算元中是否有"DA"作為區分，若使用"^"或者"DA"則代表 10 進位減法運算，若無則代表 2 進位減法運算。

註記：

(1). 必須確認被減數以及減數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此減法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(4). Rx 在語法上必須是以絕對位址來描述。
(5). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 SUB 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)，ACH 的內容值以及 CF 作減法運算。其運算結果會分別儲存到 ACH 或是 Rx 所指向的 16bits mode data RAM。
(6). 若 MCU 有提供藉由 Rm 設定 RXDA=1(目前在 EV chip (TM8999)不支援)則可以將"ADC/SBC/ADD/SUB Rx"指令切換成 10 進位運算。

指令語法：

OP code	運算元	指令動作
SUB	Rx	$AC(H)_{(10)}$ ← $(Rx)(H)_{(10)}$ + $ACB(H)_{(10)}$ +1
SUB*	Rx	$AC(H)_{(10)}$, $(Rx)(H)_{(10)}$ ← $(Rx)(H)_{(10)}$ + $ACB(H)_{(10)}$ +1
SUB	@HL	AC ← (@HL) + ACB +1
SUB#	@HL	AC ← (@HL) + ACB +1 HL ← HL+1
SUB*	@HL	AC , (@HL) ← (@HL) + ACB +1
SUB*#	@HL	AC , (@HL) ← (@HL) + ACB +1 HL ← HL+1
SUB	@HL, DA	AC_{10} ← $(@HL)_{10}$ + ACB_{10} +1
SUB#	@HL, DA	AC_{10} ← $(@HL)_{10}$ + ACB_{10} +1 HL ← HL+1
SUB*	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +1
SUB*#	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +1

OP code	運算元	指令動作
		HL ← HL+1
SUB	@HL,@ZR	AC ← (@HL) + (@ZR)B +1 if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +1 if ALUHLZR=1
SUB&	@HL,@ZR	AC ← (@HL) + (@ZR)B +1 if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +1 if ALUHLZR=1 HL ← HL+1
SUB%	@HL,@ZR	AC ← (@HL) + (@ZR)B +1 if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +1 if ALUHLZR=1 ZR ← ZR+1
SUB$	@HL,@ZR	AC ← (@HL) + (@ZR)B +1 if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +1 if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SUB*	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR)B +1 if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +1 if ALUHLZR=1
SUB*&	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR)B +1 if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +1 if ALUHLZR=1 HL ← HL+1
SUB*%	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR)B +1 if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +1 if ALUHLZR=1 ZR ← ZR+1
SUB*$	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR)B +1 if ALUHLZR=0 AC , (@HL) ← (@ZR) + ACB +1 if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SUB^	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +1 if ALUHLZR=1
SUB^&	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +1 if ALUHLZR=1 HL ← HL+1
SUB^%	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +1 if ALUHLZR=1 ZR ← ZR+1
SUB^$	@HL,@ZR	AC_{10} ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +1 if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SUB^*	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +1 if ALUHLZR=1
SUB^*&	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +1 if ALUHLZR=1 HL ← HL+1
SUB^*%	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +1 if ALUHLZR=1 ZR ← ZR+1
SUB^*$	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $(@ZR)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +1 if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SUB	@ZR	AC ← (@ZR) + ACB +1
SUB#	@ZR	AC ← (@ZR) + ACB +1 ZR ← ZR+1
SUB*	@ZR	AC , (@ZR) ← (@ZR) + ACB +1
SUB*#	@ZR	AC , (@ZR) ← (@ZR) + ACB +1 ZR ← ZR+1
SUB	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + ACB_{10} +1
SUB#	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + ACB_{10} +1

OP code	運算元	指令動作
		ZR ← ZR+1
SUB*	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +1
SUB*#	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + ACB_{10} +1 ZR ← ZR+1
SUB	@ZR,@HL	AC ← (@ZR) + (@HL)B +1 if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +1 if ALUHLZR=1
SUB&	@ZR,@HL	AC ← (@ZR) + (@HL)B +1 if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +1 if ALUHLZR=1 HL ← HL+1
SUB%	@ZR,@HL	AC ← (@ZR) + (@HL)B +1 if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +1 if ALUHLZR=1 ZR ← ZR+1
SUB$	@ZR,@HL	AC ← (@ZR) + (@HL)B +1 if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +1 if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SUB*	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL)B +1 if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +1 if ALUHLZR=1
SUB*&	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL)B +1 if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +1 if ALUHLZR=1 HL ← HL+1
SUB*%	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL)B +1 if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +1 if ALUHLZR=1 ZR ← ZR+1
SUB*$	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL)B +1 if ALUHLZR=0 AC , (@ZR) ← (@HL) + ACB +1 if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SUB^	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +1 if ALUHLZR=1
SUB^&	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +1 if ALUHLZR=1 HL ← HL+1
SUB^%	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +1 if ALUHLZR=1 ZR ← ZR+1
SUB^$	@ZR,@HL	AC_{10} ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +1 if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
SUB^*	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +1 if ALUHLZR=1
SUB^*&	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +1 if ALUHLZR=1 HL ← HL+1
SUB^*%	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +1 if ALUHLZR=1 ZR ← ZR+1
SUB^*$	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $(@HL)B_{10}$ +1 if ALUHLZR=0 AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + ACB_{10} +1 if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1

SUBH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

2 或是 10 進位的減法運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值(被減數)與 16bits mode AC(ACH)的內容值或是@ZR/HL 所指向的 16bits mode data RAM 的內容值(減數) 作沒有借位的減法運算。在此運算中，借位的值會固定為 1(沒有借位發生)。其運算結果會儲存到 ACH，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

2 或是 10 進位減法運算的判別方式是以 OP code 中是否有"^"或者運算元中是否有"DA"作為區分，若使用"^"或者"DA"則代表 10 進位減法運算，若無則代表 2 進位減法運算。

註記：

(1). 必須確認被減數以及減數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此減法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
SUBH	@HL	ACH ← (@HL)H + ACHB +1
SUBH#	@HL	ACH ← (@HL)H + ACHB +1 HL ← HL+4
SUBH*	@HL	ACH , (@HL)H ← (@HL)H + ACHB +1
SUBH*#	@HL	ACH , (@HL)H ← (@HL)H + ACHB +1 HL ← HL+4
SUBH	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $ACHB_{10}$ +1
SUBH#	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $ACHB_{10}$ +1 HL ← HL+4
SUBH*	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +1
SUBH*#	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +1 HL ← HL+4
SUBH	@HL,@ZR	ACH ← (@HL)H + (@ZR)HB +1 if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +1 if ALUHLZR=1
SUBH&	@HL,@ZR	ACH ← (@HL)H + (@ZR)HB +1 if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +1 if ALUHLZR=1 HL ← HL+4
SUBH%	@HL,@ZR	ACH ← (@HL)H + (@ZR)HB +1 if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +1 if ALUHLZR=1 ZR ← ZR+4
SUBH$	@HL,@ZR	ACH ← (@HL)H + (@ZR)HB +1 if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +1 if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SUBH*	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)HB +1 if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +1 if ALUHLZR=1
SUBH*&	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)HB +1 if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +1 if ALUHLZR=1 HL ← HL+4
SUBH*%	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)HB +1 if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACHB +1 if ALUHLZR=1 ZR ← ZR+4
SUBH*$	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)HB +1 if ALUHLZR=0

OP code	運算元	指令動作
		ACH , (@HL)H ← (@ZR)H + ACHB +1 if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SUBH^	@HL,@ZR	ACH_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +1 if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1 if ALUHLZR=1
SUBH^&	@HL,@ZR	ACH_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +1 if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1 if ALUHLZR=1 HL ← HL+4
SUBH^%	@HL,@ZR	ACH_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +1 if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1 if ALUHLZR=1 ZR ← ZR+4
SUBH^$	@HL,@ZR	ACH_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +1 if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1 if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SUBH^*	@HL,@ZR	ACH_{10} , (@HL)H_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +1 if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1 if ALUHLZR=1
SUBH^*&	@HL,@ZR	ACH_{10} , (@HL)H_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +1 if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1 if ALUHLZR=1 HL ← HL+4
SUBH^*%	@HL,@ZR	ACH_{10} , (@HL)H_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +1 if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1 if ALUHLZR=1 ZR ← ZR+4
SUBH^*$	@HL,@ZR	ACH_{10} , (@HL)H_{10} ← (@HL)H_{10} + (@ZR)HB_{10} +1 if ALUHLZR=0 ACH_{10} , (@HL)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1 if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SUBH	@ZR	ACH ← (@ZR)H + ACHB +1
SUBH#	@ZR	ACH ← (@ZR)H + ACHB +1 ZR ← ZR+4
SUBH*	@ZR	ACH , (@ZR)H ← (@ZR)H + ACHB +1
SUBH*#	@ZR	ACH , (@ZR)H ← (@ZR)H + ACHB +1 ZR ← ZR+4
SUBH	@ZR, DA	ACH_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1
SUBH#	@ZR, DA	ACH_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1 ZR ← ZR+4
SUBH*	@ZR, DA	ACH_{10} , (@ZR)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1
SUBH*#	@ZR, DA	ACH_{10} , (@ZR)H_{10} ← (@ZR)H_{10} + $ACHB_{10}$ +1 ZR ← ZR+4
SUBH	@ZR,@HL	ACH ← (@ZR)H + (@HL)HB +1 if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +1 if ALUHLZR=1
SUBH&	@ZR,@HL	ACH ← (@ZR)H + (@HL)HB +1 if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +1 if ALUHLZR=1 HL ← HL+4
SUBH%	@ZR,@HL	ACH ← (@ZR)H + (@HL)HB +1 if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +1 if ALUHLZR=1 ZR ← ZR+4
SUBH$	@ZR,@HL	ACH ← (@ZR)H + (@HL)HB +1 if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +1 if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SUBH*	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)HB +1 if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +1 if ALUHLZR=1
SUBH*&	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)HB +1 if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +1 if ALUHLZR=1 HL ← HL+4
SUBH*%	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)HB +1 if ALUHLZR=0

OP code	運算元	指令動作
		ACH , (@ZR)H ← (@HL)H + ACHB +1 if ALUHLZR=1 ZR ← ZR+4
SUBH*$	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)HB +1 if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACHB +1 if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SUBH^	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +1 if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +1 if ALUHLZR=1
SUBH^&	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +1 if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +1 if ALUHLZR=1 HL ← HL+4
SUBH^%	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +1 if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +1 if ALUHLZR=1 ZR ← ZR+4
SUBH^$	@ZR,@HL	ACH_{10} ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +1 if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +1 if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
SUBH^*	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +1 if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +1 if ALUHLZR=1
SUBH^*&	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +1 if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +1 if ALUHLZR=1 HL ← HL+4
SUBH^*%	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +1 if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +1 if ALUHLZR=1 ZR ← ZR+4
SUBH^*$	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $(@HL)HB_{10}$ +1 if ALUHLZR=0 ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $ACHB_{10}$ +1 if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4

SUBM

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 或是 10 進位的減法運算指令，將 Rx 或是@HL/ZR 所指向的 RAM 的內容值(被減數)與 MU(I)的內容值(減數) 作沒有借位的減法運算。在此運算中，借位的值會固定為 1(沒有借位發生)。其運算結果會儲存到 AC，亦可選擇儲存到 Rx，@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

2 或是 10 進位減法運算的判別方式是以 OP code 中是否有“^”或者運算元中是否有“DA”作為區分，若使用”^”或者”DA”則代表 10 進位減法運算，若無則代表 2 進位減法運算。

註記：

(1). 必須確認被減數以及減數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此減法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

指令語法：

OP code	運算元	指令動作
SUBM	@HL	AC ← (@HL) + MU(I)B +1
SUBM#	@HL	AC ← (@HL) + MU(I)B +1 HL ← HL+1
SUBM*	@HL	AC , (@HL) ← (@HL) + MU(I)B +1
SUBM*#	@HL	AC , (@HL) ← (@HL) + MU(I)B +1 HL ← HL+1
SUBM	@HL, DA	AC_{10} ← $(@HL)_{10}$ + $MU(I)B_{10}$ +1
SUBM#	@HL, DA	AC_{10} ← $(@HL)_{10}$ + $MU(I)B_{10}$ +1 HL ← HL+1
SUBM*	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +1
SUBM*#	@HL, DA	AC_{10} , $(@HL)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +1 HL ← HL+1
SUBM	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I)B +1
SUBM&	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I)B +1 HL ← HL+1
SUBM%	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I)B +1 ZR ← ZR+1
SUBM$	@HL,@ZR	AC , (@HL) ← (@ZR) + MU(I)B +1 HL ← HL+1 ZR ← ZR+1
SUBM^	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +1
SUBM^&	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +1 HL ← HL+1
SUBM^%	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +1 ZR ← ZR+1
SUBM^$	@HL,@ZR	AC_{10} , $(@HL)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +1 HL ← HL+1 ZR ← ZR+1
SUBM	@ZR	AC ← (@ZR) + MU(I)B +1
SUBM#	@ZR	AC ← (@ZR) + MU(I)B +1 ZR ← ZR+1
SUBM*	@ZR	AC , (@ZR) ← (@ZR) + MU(I)B +1
SUBM*#	@ZR	AC , (@ZR) ← (@ZR) + MU(I)B +1 ZR ← ZR+1
SUBM	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +1
SUBM#	@ZR, DA	AC_{10} ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +1 ZR ← ZR+1
SUBM*	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +1
SUBM*#	@ZR, DA	AC_{10} , $(@ZR)_{10}$ ← $(@ZR)_{10}$ + $MU(I)B_{10}$ +1 ZR ← ZR+1
SUBM	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I)B +1
SUBM&	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I)B +1 HL ← HL+1
SUBM%	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I)B +1 ZR ← ZR+1
SUBM$	@ZR,@HL	AC , (@ZR) ← (@HL) + MU(I)B +1 HL ← HL+1 ZR ← ZR+1
SUBM^	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +1
SUBM^&	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +1 HL ← HL+1
SUBM^%	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +1 ZR ← ZR+1
SUBM^$	@ZR,@HL	AC_{10} , $(@ZR)_{10}$ ← $(@HL)_{10}$ + $MU(I)B_{10}$ +1 HL ← HL+1

		ZR ← ZR+1

SUBMH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

2 或是 10 進位的減法運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode MU(I)(MU(I)H)的內容值作沒有借位的減法運算。在此運算中，借位的值會固定為 1(沒有借位發生)。其運算結果會儲存到 16bits mode AC(ACH)，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

2 或是 10 進位減法運算的判別方式是以 OP code 中是否有"^"或者運算元中是否有"DA"作為區分，若使用"^"或者"DA"則代表 10 進位減法運算，若無則代表 2 進位減法運算。

註記：

(1). 必須確認被減數以及減數以及運算指令都是相同的 2 或是 10 進位的格式。
(2). 此減法運算結果的進位會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
SUBMH	@HL	ACH ← (@HL)H + MU(I)HB +1
SUBMH#	@HL	ACH ← (@HL)H + MU(I)HB +1 HL ← HL+4
SUBMH*	@HL	ACH , (@HL)H ← (@HL)H + MU(I)HB +1
SUBMH*#	@HL	ACH , (@HL)H ← (@HL)H + MU(I)HB +1 HL ← HL+4
SUBMH	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +1
SUBMH#	@HL, DA	ACH_{10} ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +1 HL ← HL+4
SUBMH*	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +1
SUBMH*#	@HL, DA	ACH_{10} , $(@HL)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +1 HL ← HL+4
SUBMH	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)HB +1
SUBMH&	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)HB +1 HL ← HL+4
SUBMH%	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)HB +1 ZR ← ZR+4
SUBMH$	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MU(I)HB +1 HL ← HL+4 ZR ← ZR+4
SUBMH^	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +1
SUBMH^&	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +1 HL ← HL+4
SUBMH^%	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +1 ZR ← ZR+4
SUBMH^$	@HL,@ZR	ACH_{10} , $(@HL)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +1 HL ← HL+4

		ZR ← ZR+4
SUBMH	@ZR	ACH ← (@ZR)H + MU(I)HB +1
SUBMH#	@ZR	ACH ← (@ZR)H + MU(I)HB +1 ZR ← ZR+4
SUBMH*	@ZR	ACH , (@ZR)H ← (@ZR)H + MU(I)HB +1
SUBMH*#	@ZR	ACH , (@ZR)H ← (@ZR)H + MU(I)HB +1 ZR ← ZR+4
SUBMH	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +1
SUBMH#	@ZR, DA	ACH_{10} ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +1 ZR ← ZR+4
SUBMH*	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +1
SUBMH*#	@ZR, DA	ACH_{10} , $(@ZR)H_{10}$ ← $(@ZR)H_{10}$ + $MU(I)HB_{10}$ +1 ZR ← ZR+4
SUBMH	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)HB +1
SUBMH&	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)HB +1 HL ← HL+4
SUBMH%	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)HB +1 ZR ← ZR+4
SUBMH$	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MU(I)HB +1 HL ← HL+4 ZR ← ZR+4
SUBMH^	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +1
SUBMH^&	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +1 HL ← HL+4
SUBMH^%	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +1 ZR ← ZR+4
SUBMH^$	@ZR,@HL	ACH_{10} , $(@ZR)H_{10}$ ← $(@HL)H_{10}$ + $MU(I)HB_{10}$ +1 HL ← HL+4 ZR ← ZR+4

ADN

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 進位的加法運算指令，將 Rx 或是@HL/ZR 所指向的 data RAM 的內容值與 AC 的內容值或是@ZR/HL 所指向的 data RAM 的內容值(目前在 EV chip (TM8999)不支援)作加法運算。其運算結果會儲存到 AC，亦可選擇儲存到 Rx，@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

註記：

(1). 在執行指令之前必須先確認被加數以及加數都必須是相同的進位的格式。
(2). 此加法運算結果的進位不會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(4). Rx 在語法上必須是以絕對位址來描述。
(5). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 ADN 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內

容值((Rx)H)，16bits mode AC(ACH)的內容值以及 CF 作加法運算。其運算結果會分別儲存到 ACH 或是 Rx 所指向的 16bits mode data RAM。
若 MCU 有提供藉由 Rm 設定 ADNIDP=1(目前在 EV chip (TM8999)不支援)則可以將 16bits mode “AND Rx”指令切換為每個 4bits 獨立進行加法運算。

指令語法：

OP code	運算元	指令動作
ADN	Rx	AC(H) ← (Rx)(H) + AC(H)
ADN*	Rx	AC(H) , (Rx)(H) ← (Rx)(H) + AC(H)
ADN	@HL	AC ← (@HL) + AC
ADN#	@HL	AC ← (@HL) + AC HL ← HL+1
ADN*	@HL	AC , (@HL) ← (@HL) + AC
ADN*#	@HL	AC , (@HL) ← (@HL) + AC HL ← HL+1
ADN	@HL,@ZR	AC ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1
ADN&	@HL,@ZR	AC ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 HL ← HL+1
ADN%	@HL,@ZR	AC ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 ZR ← ZR+1
ADN$	@HL,@ZR	AC ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADN*	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1
ADN*&	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 HL ← HL+1
ADN*%	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 ZR ← ZR+1
ADN*$	@HL,@ZR	AC , (@HL) ← (@HL) + (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) + AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
ADN	@ZR	AC ← (@ZR) + AC
ADN#	@ZR	AC ← (@ZR) + AC ZR ← ZR+1
ADN*	@ZR	AC , (@ZR) ← (@ZR) + AC
ADN*#	@ZR	AC , (@ZR) ← (@ZR) + AC ZR ← ZR+1
ADN	@ZR,@HL	AC ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1
ADN&	@ZR,@HL	AC ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1 HL ← HL+1
ADN%	@ZR,@HL	AC ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1 ZR ← ZR+1
ADN$	@ZR,@HL	AC ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1 HL ← HL+1

OP code	運算元	指令動作
		ZR ← ZR+1
ADN*	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1
ADN*&	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1 HL ← HL+1
ADN*%	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1 ZR ← ZR+1
ADN*$	@ZR,@HL	AC , (@ZR) ← (@ZR) + (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) + AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1

ADNH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

2 進位的加法運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode AC(ACH)的內容值或是@ZR/HL 所指向的 16bits mode data RAM 的內容值作加法運算。其運算結果會儲存到 ACH，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

註記：

(1). 在執行指令之前必須先確認被加數以及加數都必須是相同的進位的格式。
(2). 此加法運算結果的進位不會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。
(4). 若 MCU 有提供藉由 Rm 設定 ADNIDP=1(目前在 EV chip (TM8999)不支援)則可以將 ADNH 指令切換為每個 4bits 獨立進行加法運算。

指令語法：

OP code	運算元	指令動作
ADNH	@HL	ACH ← (@HL)H + ACH
ADNH#	@HL	ACH ← (@HL)H + ACH HL ← HL+4
ADNH*	@HL	ACH , (@HL)H ← (@HL)H + ACH
ADNH*#	@HL	ACH , (@HL)H ← (@HL)H + ACH HL ← HL+4
ADNH	@HL,@ZR	ACH ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1
ADNH&	@HL,@ZR	ACH ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1 HL ← HL+4
ADNH%	@HL,@ZR	ACH ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1

OP code	運算元	指令動作
		ZR ← ZR+4
ADNH$	@HL,@ZR	ACH ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADNH*	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1
ADNH*&	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1 HL ← HL+4
ADNH*%	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1 ZR ← ZR+4
ADNH*$	@HL,@ZR	ACH , (@HL)H ← (@HL)H + (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H + ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADNH	@ZR	ACH ← (@ZR)H + ACH
ADNH#	@ZR	ACH ← (@ZR)H + ACH ZR ← ZR+4
ADNH*	@ZR	ACH , (@ZR)H ← (@ZR)H + ACH
ADNH*#	@ZR	ACH , (@ZR)H ← (@ZR)H + ACH ZR ← ZR+4
ADNH	@ZR,@HL	ACH ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1
ADNH&	@ZR,@HL	ACH ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 HL ← HL+4
ADNH%	@ZR,@HL	ACH ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 ZR ← ZR+4
ADNH$	@ZR,@HL	ACH ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ADNH*	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1
ADNH*&	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 HL ← HL+4
ADNH*%	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 ZR ← ZR+4
ADNH*$	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H + (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H + ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4

ADNM(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 進位的加法運算指令，將@HL/ZR 所指向的 data RAM 的內容值與 MUI 的內容值作加法運算。其運算結果會儲存到 AC，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 data RAM。

註記：

(1). 在執行指令之前必須先確認被加數以及加數都必須是相同的進位的格式。
(2). 此加法運算結果的進位不會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

指令語法：

OP code	運算元	指令動作
ADNM	@HL	AC ← (@HL) + MUI
ADNM#	@HL	AC ← (@HL) + MUI HL ← HL+1
ADNM*	@HL	AC , (@HL) ← (@HL) + MUI
ADNM*#	@HL	AC , (@HL) ← (@HL) + MUI HL ← HL+1
ADNM	@HL,@ZR	AC , (@HL) ← (@ZR) + MUI
ADNM&	@HL,@ZR	AC , (@HL) ← (@ZR) + MUI HL ← HL+1
ADNM%	@HL,@ZR	AC , (@HL) ← (@ZR) + MUI ZR ← ZR+1
ADNM$	@HL,@ZR	AC , (@HL) ← (@ZR) + MUI HL ← HL+1 ZR ← ZR+1
ADNM	@ZR	AC ← (@ZR) + MUI
ADNM#	@ZR	AC ← (@ZR) + MUI ZR ← ZR+1
ADNM*	@ZR	AC , (@ZR) ← (@ZR) + MUI
ADNM*#	@ZR	AC , (@ZR) ← (@ZR) + MUI ZR ← ZR+1
ADNM	@ZR,@HL	AC , (@ZR) ← (@HL) + MUI
ADNM&	@ZR,@HL	AC , (@ZR) ← (@HL) + MUI HL ← HL+1
ADNM%	@ZR,@HL	AC , (@ZR) ← (@HL) + MUI ZR ← ZR+1
ADNM$	@ZR,@HL	AC , (@ZR) ← (@HL) + MUI HL ← HL+1 ZR ← ZR+1

ADNMH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

2 進位的加法運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode MUI(MUIH)的內容值作加法運算。其運算結果會儲存到 16bits mode AC(ACH)，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

註記：
(1). 在執行指令之前必須先確認被加數以及加數都必須是相同的進位的格式。
(2). 此加法運算結果的進位不會改變 CF 內容值。
(3). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
ADNMH	@HL	ACH ← (@HL)H + MUIH
ADNMH#	@HL	ACH ← (@HL)H + MUIH HL ← HL+4
ADNMH*	@HL	ACH , (@HL)H ← (@HL)H + MUIH
ADNMH*#	@HL	ACH , (@HL)H ← (@HL)H + MUIH HL ← HL+4
ADNMH	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MUIH
ADNMH&	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MUIH HL ← HL+4
ADNMH%	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MUIH ZR ← ZR+4
ADNMH$	@HL,@ZR	ACH , (@HL)H ← (@ZR)H + MUIH HL ← HL+4 ZR ← ZR+4
ADNMH	@ZR	ACH ← (@ZR)H + MUIH
ADNMH#	@ZR	ACH ← (@ZR)H + MUIH ZR ← ZR+4
ADNMH*	@ZR	ACH , (@ZR)H ← (@ZR)H + MUIH
ADNMH*#	@ZR	ACH , (@ZR)H ← (@ZR)H + MUIH ZR ← ZR+4
ADNMH	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MUIH
ADNMH&	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MUIH HL ← HL+4
ADNMH%	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MUIH ZR ← ZR+4
ADNMH$	@ZR,@HL	ACH , (@ZR)H ← (@HL)H + MUIH HL ← HL+4 ZR ← ZR+4

AND

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
AND 邏輯運算指令，將 Rx 或是@HL/ZR 所指向的 data RAM 的內容值與 AC 的內容值或是@ZR/HL 所指向的 data RAM 的內容值(目前在 EV chip (TM8999)不支援)作 AND 邏輯運算。其運算結果會儲存到 AC，亦可選擇儲存到 Rx，@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

Truth Table:

A	B	A & B
0	0	0

0	1	0
1	1	1
1	0	0

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

(2). Rx 在語法上必須是以絕對位址來描述。

(3). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 AND 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)，16bits mode AC(ACH)的內容值作 AND 邏輯運算。其運算結果會分別儲存到 ACH 或是 Rx 所指向的 16bits mode data RAM。

指令語法：

OP code	運算元	指令動作
AND	Rx	AC(H) ← (Rx)(H) & AC(H)
AND*	Rx	AC(H) , (Rx)(H) ← (Rx)(H) & AC(H)
AND	@HL	AC ← (@HL) & AC
AND#	@HL	AC ← (@HL) & AC HL ← HL+1
AND*	@HL	AC , (@HL) ← (@HL) & AC
AND*#	@HL	AC , (@HL) ← (@HL) & AC HL ← HL+1
AND	@HL,@ZR	AC ← (@HL) & (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) & AC if ALUHLZR=1
AND&	@HL,@ZR	AC ← (@HL) & (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) & AC if ALUHLZR=1 HL ← HL+1
AND%	@HL,@ZR	AC ← (@HL) & (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) & AC if ALUHLZR=1 ZR ← ZR+1
AND$	@HL,@ZR	AC ← (@HL) & (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) & AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
AND*	@HL,@ZR	AC , (@HL) ← (@HL) & (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) & AC if ALUHLZR=1
AND*&	@HL,@ZR	AC , (@HL) ← (@HL) & (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) & AC if ALUHLZR=1 HL ← HL+1
AND*%	@HL,@ZR	AC , (@HL) ← (@HL) & (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) & AC if ALUHLZR=1 ZR ← ZR+1
AND*$	@HL,@ZR	AC , (@HL) ← (@HL) & (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) & AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
AND	@ZR	AC ← (@ZR) & AC
AND#	@ZR	AC ← (@ZR) & AC ZR ← ZR+1
AND*	@ZR	AC , (@ZR) ← (@ZR) & AC
AND*#	@ZR	AC , (@ZR) ← (@ZR) & AC ZR ← ZR+1
AND	@ZR,@HL	AC ← (@ZR) & (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) & AC if ALUHLZR=1
AND&	@ZR,@HL	AC ← (@ZR) & (@HL) if ALUHLZR=0

OP code	運算元	指令動作
		AC , (@ZR) ← (@HL) & AC if ALUHLZR=1 HL ← HL+1
AND%	@ZR,@HL	AC ← (@ZR) & (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) & AC if ALUHLZR=1 ZR ← ZR+1
AND$	@ZR,@HL	AC ← (@ZR) & (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) & AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
AND*	@ZR,@HL	AC , (@ZR) ← (@ZR) & (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) & AC if ALUHLZR=1
AND*&	@ZR,@HL	AC , (@ZR) ← (@ZR) & (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) & AC if ALUHLZR=1 HL ← HL+1
AND*%	@ZR,@HL	AC , (@ZR) ← (@ZR) & (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) & AC if ALUHLZR=1 ZR ← ZR+1
AND*$	@ZR,@HL	AC , (@ZR) ← (@ZR) & (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) & AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1

ANDH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

AND 邏輯運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode AC(ACH)的內容值或是@ZR/HL 所指向的 16bits mode data RAM 的內容值作 AND 邏輯運算。其運算結果會儲存到 ACH，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
ANDH	@HL	ACH ← (@HL)H & ACH
ANDH#	@HL	ACH ← (@HL)H & ACH HL ← HL+4
ANDH*	@HL	ACH , (@HL)H ← (@HL)H & ACH
ANDH*#	@HL	ACH , (@HL)H ← (@HL)H & ACH HL ← HL+4
ANDH	@HL,@ZR	ACH ← (@HL)H & (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H & ACH if ALUHLZR=1
ANDH&	@HL,@ZR	ACH ← (@HL)H & (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H & ACH if ALUHLZR=1 HL ← HL+4
ANDH%	@HL,@ZR	ACH ← (@HL)H & (@ZR)H if ALUHLZR=0

OP code	運算元	指令動作
		ACH , (@HL)H ← (@ZR)H & ACH if ALUHLZR=1 ZR ← ZR+4
ANDH$	@HL,@ZR	ACH ← (@HL)H & (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H & ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ANDH*	@HL,@ZR	ACH , (@HL)H ← (@HL)H & (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H & ACH if ALUHLZR=1
ANDH*&	@HL,@ZR	ACH , (@HL)H ← (@HL)H & (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H & ACH if ALUHLZR=1 HL ← HL+4
ANDH*%	@HL,@ZR	ACH , (@HL)H ← (@HL)H & (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H & ACH if ALUHLZR=1 ZR ← ZR+4
ANDH*$	@HL,@ZR	ACH , (@HL)H ← (@HL)H & (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H & ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ANDH	@ZR	ACH ← (@ZR)H & ACH
ANDH#	@ZR	ACH ← (@ZR)H & ACH ZR ← ZR+4
ANDH*	@ZR	ACH , (@ZR)H ← (@ZR)H & ACH
ANDH*#	@ZR	ACH , (@ZR)H ← (@ZR)H & ACH ZR ← ZR+4
ANDH	@ZR,@HL	ACH ← (@ZR)H & (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H & ACH if ALUHLZR=1
ANDH&	@ZR,@HL	ACH ← (@ZR)H & (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H & ACH if ALUHLZR=1 HL ← HL+4
ANDH%	@ZR,@HL	ACH ← (@ZR)H & (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H & ACH if ALUHLZR=1 ZR ← ZR+4
ANDH$	@ZR,@HL	ACH ← (@ZR)H & (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H & ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ANDH*	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H & (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H & ACH if ALUHLZR=1
ANDH*&	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H & (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H & ACH if ALUHLZR=1 HL ← HL+4
ANDH*%	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H & (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H & ACH if ALUHLZR=1 ZR ← ZR+4
ANDH*$	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H & (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H & ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4

ANDM(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

AND 邏輯的加法運算指令，將@HL/ZR 所指向的 data RAM 的內容值與 MUI 的內容值作 AND 邏輯運算。其運算結果會儲存到 AC，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 data RAM。

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

指令語法：

OP code	運算元	指令動作
ANDM	@HL	AC ← (@HL) & MUI
ANDM#	@HL	AC ← (@HL) & MUI HL ← HL+1
ANDM*	@HL	AC , (@HL) ← (@HL) & MUI
ANDM*#	@HL	AC , (@HL) ← (@HL) & MUI HL ← HL+1
ANDM	@HL,@ZR	AC , (@HL) ← (@ZR) & MUI
ANDM&	@HL,@ZR	AC , (@HL) ← (@ZR) & MUI HL ← HL+1
ANDM%	@HL,@ZR	AC , (@HL) ← (@ZR) & MUI ZR ← ZR+1
ANDM$	@HL,@ZR	AC , (@HL) ← (@ZR) & MUI HL ← HL+1 ZR ← ZR+1
ANDM	@ZR	AC ← (@ZR) & MUI
ANDM#	@ZR	AC ← (@ZR) & MUI ZR ← ZR+1
ANDM*	@ZR	AC , (@ZR) ← (@ZR) & MUI
ANDM*#	@ZR	AC , (@ZR) ← (@ZR) & MUI ZR ← ZR+1
ANDM	@ZR,@HL	AC , (@ZR) ← (@HL) & MUI
ANDM&	@ZR,@HL	AC , (@ZR) ← (@HL) & MUI HL ← HL+1
ANDM%	@ZR,@HL	AC , (@ZR) ← (@HL) & MUI ZR ← ZR+1
ANDM$	@ZR,@HL	AC , (@ZR) ← (@HL) & MUI HL ← HL+1 ZR ← ZR+1

ANDMH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

AND 邏輯運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode MUI(MUIH)的內容值作 AND 邏輯運算。其運算結果會儲存到 16bits mode AC(ACH)，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
ANDMH	@HL	ACH ← (@HL)H & MUIH
ANDMH#	@HL	ACH ← (@HL)H & MUIH HL ← HL+4
ANDMH*	@HL	ACH , (@HL)H ← (@HL)H & MUIH
ANDMH*#	@HL	ACH , (@HL)H ← (@HL)H & MUIH HL ← HL+4
ANDMH	@HL,@ZR	ACH , (@HL)H ← (@ZR)H & MUIH
ANDMH&	@HL,@ZR	ACH , (@HL)H ← (@ZR)H & MUIH HL ← HL+4
ANDMH%	@HL,@ZR	ACH , (@HL)H ← (@ZR)H & MUIH ZR ← ZR+4
ANDMH$	@HL,@ZR	ACH , (@HL)H ← (@ZR)H & MUIH HL ← HL+4 ZR ← ZR+4
ANDMH	@ZR	ACH ← (@ZR)H & MUIH
ANDMH#	@ZR	ACH ← (@ZR)H & MUIH ZR ← ZR+4
ANDMH*	@ZR	ACH , (@ZR)H ← (@ZR)H & MUIH
ANDMH*#	@ZR	ACH , (@ZR)H ← (@ZR)H & MUIH ZR ← ZR+4
ANDMH	@ZR,@HL	ACH , (@ZR)H ← (@HL)H & MUIH
ANDMH&	@ZR,@HL	ACH , (@ZR)H ← (@HL)H & MUIH HL ← HL+4
ANDMH%	@ZR,@HL	ACH , (@ZR)H ← (@HL)H & MUIH ZR ← ZR+4
ANDMH$	@ZR,@HL	ACH , (@ZR)H ← (@HL)H & MUIH HL ← HL+4 ZR ← ZR+4

EOR

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
EOR 邏輯運算指令，將 Rx 或是@HL/ZR 所指向的 data RAM 的內容值與 AC 的內容值或是@ZR/HL 所指向的 data RAM 的內容值(目前在 EV chip (TM8999)不支援)作 EOR 邏輯運算。其運算結果會儲存到 AC，亦可選擇儲存到 Rx，@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

Truth Table:

A	B	A xor B
0	0	0
0	1	1
1	1	0
1	0	1

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

(2). Rx 在語法上必須是以絕對位址來描述。

(3). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 EOR 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)，16bits mode AC(ACH)的內容值作 EOR 邏輯運算。其運算結果會分別儲存到 ACH 或是 Rx 所指向的 16bits mode data RAM。

指令語法：

OP code	運算元	指令動作
EOR	Rx	AC(H) ← (Rx)(H) xor AC(H)
EOR*	Rx	AC(H) , (Rx)(H) ← (Rx)(H) xor AC(H)
EOR	@HL	AC ← (@HL) xor AC
EOR#	@HL	AC ← (@HL) xor AC HL ← HL+1
EOR*	@HL	AC , (@HL) ← (@HL) xor AC
EOR*#	@HL	AC , (@HL) ← (@HL) xor AC HL ← HL+1
EOR	@HL,@ZR	AC ← (@HL) xor (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) xor AC if ALUHLZR=1
EOR&	@HL,@ZR	AC ← (@HL) xor (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) xor AC if ALUHLZR=1 HL ← HL+1
EOR%	@HL,@ZR	AC ← (@HL) xor (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) xor AC if ALUHLZR=1 ZR ← ZR+1
EOR$	@HL,@ZR	AC ← (@HL) xor (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) xor AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
EOR*	@HL,@ZR	AC , (@HL) ← (@HL) xor (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) xor AC if ALUHLZR=1
EOR*&	@HL,@ZR	AC , (@HL) ← (@HL) xor (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) xor AC if ALUHLZR=1 HL ← HL+1
EOR*%	@HL,@ZR	AC , (@HL) ← (@HL) xor (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) xor AC if ALUHLZR=1 ZR ← ZR+1
EOR*$	@HL,@ZR	AC , (@HL) ← (@HL) xor (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) xor AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
EOR	@ZR	AC ← (@ZR) xor AC
EOR#	@ZR	AC ← (@ZR) xor AC ZR ← ZR+1
EOR*	@ZR	AC , (@ZR) ← (@ZR) xor AC
EOR*#	@ZR	AC , (@ZR) ← (@ZR) xor AC ZR ← ZR+1
EOR	@ZR,@HL	AC ← (@ZR) xor (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) xor AC if ALUHLZR=1
EOR&	@ZR,@HL	AC ← (@ZR) xor (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) xor AC if ALUHLZR=1 HL ← HL+1
EOR%	@ZR,@HL	AC ← (@ZR) xor (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) xor AC if ALUHLZR=1

OP code	運算元	指令動作
		ZR ← ZR+1
EOR$	@ZR,@HL	AC ← (@ZR) xor (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) xor AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
EOR*	@ZR,@HL	AC , (@ZR) ← (@ZR) xor (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) xor AC if ALUHLZR=1
EOR*&	@ZR,@HL	AC , (@ZR) ← (@ZR) xor (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) xor AC if ALUHLZR=1 HL ← HL+1
EOR*%	@ZR,@HL	AC , (@ZR) ← (@ZR) xor (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) xor AC if ALUHLZR=1 ZR ← ZR+1
EOR*$	@ZR,@HL	AC , (@ZR) ← (@ZR) xor (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) xor AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1

EORH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

EOR 邏輯運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode AC(ACH)的內容值或是@ZR/HL 所指向的 16bits mode data RAM 的內容值作 EOR 邏輯運算。其運算結果會儲存到 ACH，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
EORH	@HL	ACH ← (@HL)H xor ACH
EORH#	@HL	ACH ← (@HL)H xor ACH HL ← HL+4
EORH*	@HL	ACH , (@HL)H ← (@HL)H xor ACH
EORH*#	@HL	ACH , (@HL)H ← (@HL)H xor ACH HL ← HL+4
EORH	@HL,@ZR	ACH ← (@HL)H xor (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H xor ACH if ALUHLZR=1
EORH&	@HL,@ZR	ACH ← (@HL)H xor (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H xor ACH if ALUHLZR=1 HL ← HL+4
EORH%	@HL,@ZR	ACH ← (@HL)H xor (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H xor ACH if ALUHLZR=1 ZR ← ZR+4
EORH$	@HL,@ZR	ACH ← (@HL)H xor (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H xor ACH if ALUHLZR=1

OP code	運算元	指令動作
		HL ← HL+4 ZR ← ZR+4
EORH*	@HL,@ZR	ACH , (@HL)H ← (@HL)H xor (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H xor ACH if ALUHLZR=1
EORH*&	@HL,@ZR	ACH , (@HL)H ← (@HL)H xor (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H xor ACH if ALUHLZR=1 HL ← HL+4
EORH*%	@HL,@ZR	ACH , (@HL)H ← (@HL)H xor (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H xor ACH if ALUHLZR=1 ZR ← ZR+4
EORH*$	@HL,@ZR	ACH , (@HL)H ← (@HL)H xor (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H xor ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
EORH	@ZR	ACH ← (@ZR)H xor ACH
EORH#	@ZR	ACH ← (@ZR)H xor ACH ZR ← ZR+4
EORH*	@ZR	ACH , (@ZR)H ← (@ZR)H xor ACH
EORH*#	@ZR	ACH , (@ZR)H ← (@ZR)H xor ACH ZR ← ZR+4
EORH	@ZR,@HL	ACH ← (@ZR)H xor (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H xor ACH if ALUHLZR=1
EORH&	@ZR,@HL	ACH ← (@ZR)H xor (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H xor ACH if ALUHLZR=1 HL ← HL+4
EORH%	@ZR,@HL	ACH ← (@ZR)H xor (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H xor ACH if ALUHLZR=1 ZR ← ZR+4
EORH$	@ZR,@HL	ACH ← (@ZR)H xor (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H xor ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
EORH*	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H xor (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H xor ACH if ALUHLZR=1
EORH*&	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H xor (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H xor ACH if ALUHLZR=1 HL ← HL+4
EORH*%	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H xor (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H xor ACH if ALUHLZR=1 ZR ← ZR+4
EORH*$	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H xor (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H xor ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4

EORM(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

EOR 邏輯的加法運算指令，將@HL/ZR 所指向的 data RAM 的內容值與 MUI 的內容值作 EOR 邏輯運算。其運算結果會儲存到 AC，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 data RAM。

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

指令語法：

OP code	運算元	指令動作
EORM	@HL	AC ← (@HL) xor MUI
EORM#	@HL	AC ← (@HL) xor MUI HL ← HL+1
EORM*	@HL	AC , (@HL) ← (@HL) xor MUI
EORM*#	@HL	AC , (@HL) ← (@HL) xor MUI HL ← HL+1
EORM	@HL,@ZR	AC , (@HL) ← (@ZR) xor MUI
EORM&	@HL,@ZR	AC , (@HL) ← (@ZR) xor MUI HL ← HL+1
EORM%	@HL,@ZR	AC , (@HL) ← (@ZR) xor MUI ZR ← ZR+1
EORM$	@HL,@ZR	AC , (@HL) ← (@ZR) xor MUI HL ← HL+1 ZR ← ZR+1
EORM	@ZR	AC ← (@ZR) xor MUI
EORM#	@ZR	AC ← (@ZR) xor MUI ZR ← ZR+1
EORM*	@ZR	AC , (@ZR) ← (@ZR) xor MUI
EORM*#	@ZR	AC , (@ZR) ← (@ZR) xor MUI ZR ← ZR+1
EORM	@ZR,@HL	AC , (@ZR) ← (@HL) xor MUI
EORM&	@ZR,@HL	AC , (@ZR) ← (@HL) xor MUI HL ← HL+1
EORM%	@ZR,@HL	AC , (@ZR) ← (@HL) xor MUI ZR ← ZR+1
EORM$	@ZR,@HL	AC , (@ZR) ← (@HL) xor MUI HL ← HL+1 ZR ← ZR+1

EORMH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

EOR 邏輯運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode MUI(MUIH)的內容值作 EOR 邏輯運算。其運算結果會儲存到 16bits mode AC(ACH)，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
EORMH	@HL	ACH ← (@HL)H xor MUIH
EORMH#	@HL	ACH ← (@HL)H xor MUIH HL ← HL+4
EORMH*	@HL	ACH , (@HL)H ← (@HL)H xor MUIH
EORMH*#	@HL	ACH , (@HL)H ← (@HL)H xor MUIH HL ← HL+4
EORMH	@HL,@ZR	ACH , (@HL)H ← (@ZR)H xor MUIH
EORMH&	@HL,@ZR	ACH , (@HL)H ← (@ZR)H xor MUIH HL ← HL+4
EORMH%	@HL,@ZR	ACH , (@HL)H ← (@ZR)H xor MUIH ZR ← ZR+4
EORMH$	@HL,@ZR	ACH , (@HL)H ← (@ZR)H xor MUIH HL ← HL+4 ZR ← ZR+4
EORMH	@ZR	ACH ← (@ZR)H xor MUIH
EORMH#	@ZR	ACH ← (@ZR)H xor MUIH ZR ← ZR+4
EORMH*	@ZR	ACH , (@ZR)H ← (@ZR)H xor MUIH
EORMH*#	@ZR	ACH , (@ZR)H ← (@ZR)H xor MUIH ZR ← ZR+4
EORMH	@ZR,@HL	ACH , (@ZR)H ← (@HL)H xor MUIH
EORMH&	@ZR,@HL	ACH , (@ZR)H ← (@HL)H xor MUIH HL ← HL+4
EORMH%	@ZR,@HL	ACH , (@ZR)H ← (@HL)H xor MUIH ZR ← ZR+4
EORMH$	@ZR,@HL	ACH , (@ZR)H ← (@HL)H xor MUIH HL ← HL+4 ZR ← ZR+4

OR

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

OR 邏輯運算指令，將 Rx 或是@HL/ZR 所指向的 data RAM 的內容值與 AC 的內容值或是@ZR/HL 所指向的 data RAM 的內容值(目前在 EV chip (TM8999)不支援)作 OR 邏輯運算。其運算結果會儲存到 AC，亦可選擇儲存到 Rx，@HL/ZR 或者@ZR/HL(目前在 EV chip (TM8999)不支援)所指向的 data RAM。

Truth Table:

A	B	A | B
0	0	0
0	1	1
1	1	1
1	0	1

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 OR 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)，16bits mode AC(ACH)的內容值作 OR 邏輯運算。其運算結果會分別儲存到 ACH 或是 Rx 所指向的 16bits mode data RAM。

指令語法：

OP code	運算元	指令動作
OR	Rx	AC(H) ← (Rx)(H) | AC(H)
OR*	Rx	AC(H) , (Rx)(H) ← (Rx)(H) | AC(H)
OR	@HL	AC ← (@HL) | AC
OR#	@HL	AC ← (@HL) | AC HL ← HL+1
OR*	@HL	AC , (@HL) ← (@HL) | AC
OR*#	@HL	AC , (@HL) ← (@HL) | AC HL ← HL+1
OR	@HL,@ZR	AC ← (@HL) | (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) | AC if ALUHLZR=1
OR&	@HL,@ZR	AC ← (@HL) | (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) | AC if ALUHLZR=1 HL ← HL+1
OR%	@HL,@ZR	AC ← (@HL) | (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) | AC if ALUHLZR=1 ZR ← ZR+1
OR$	@HL,@ZR	AC ← (@HL) | (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) | AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
OR*	@HL,@ZR	AC , (@HL) ← (@HL) | (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) | AC if ALUHLZR=1
OR*&	@HL,@ZR	AC , (@HL) ← (@HL) | (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) | AC if ALUHLZR=1 HL ← HL+1
OR*%	@HL,@ZR	AC , (@HL) ← (@HL) | (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) | AC if ALUHLZR=1 ZR ← ZR+1
OR*$	@HL,@ZR	AC , (@HL) ← (@HL) | (@ZR) if ALUHLZR=0 AC , (@HL) ← (@ZR) | AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
OR	@ZR	AC ← (@ZR) | AC
OR#	@ZR	AC ← (@ZR) | AC ZR ← ZR+1
OR*	@ZR	AC , (@ZR) ← (@ZR) | AC
OR*#	@ZR	AC , (@ZR) ← (@ZR) | AC ZR ← ZR+1
OR	@ZR,@HL	AC ← (@ZR) | (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) | AC if ALUHLZR=1
OR&	@ZR,@HL	AC ← (@ZR) | (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) | AC if ALUHLZR=1 HL ← HL+1
OR%	@ZR,@HL	AC ← (@ZR) | (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) | AC if ALUHLZR=1 ZR ← ZR+1
OR$	@ZR,@HL	AC ← (@ZR) | (@HL) if ALUHLZR=0

OP code	運算元	指令動作
		AC , (@ZR) ← (@HL) | AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1
OR*	@ZR,@HL	AC , (@ZR) ← (@ZR) | (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) | AC if ALUHLZR=1
OR*&	@ZR,@HL	AC , (@ZR) ← (@ZR) | (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) | AC if ALUHLZR=1 HL ← HL+1
OR*%	@ZR,@HL	AC , (@ZR) ← (@ZR) | (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) | AC if ALUHLZR=1 ZR ← ZR+1
OR*$	@ZR,@HL	AC , (@ZR) ← (@ZR) | (@HL) if ALUHLZR=0 AC , (@ZR) ← (@HL) | AC if ALUHLZR=1 HL ← HL+1 ZR ← ZR+1

ORH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

OR 邏輯運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode AC(ACH)的內容值或是@ZR/HL 所指向的 16bits mode data RAM 的內容值作 OR 邏輯運算。其運算結果會儲存到 ACH，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
ORH	@HL	ACH ← (@HL)H | ACH
ORH#	@HL	ACH ← (@HL)H | ACH HL ← HL+4
ORH*	@HL	ACH , (@HL)H ← (@HL)H | ACH
ORH*#	@HL	ACH , (@HL)H ← (@HL)H | ACH HL ← HL+4
ORH	@HL,@ZR	ACH ← (@HL)H | (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H | ACH if ALUHLZR=1
ORH&	@HL,@ZR	ACH ← (@HL)H | (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H | ACH if ALUHLZR=1 HL ← HL+4
ORH%	@HL,@ZR	ACH ← (@HL)H | (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H | ACH if ALUHLZR=1 ZR ← ZR+4
ORH$	@HL,@ZR	ACH ← (@HL)H | (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H | ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4

OP code	運算元	指令動作
ORH*	@HL,@ZR	ACH , (@HL)H ← (@HL)H | (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H | ACH if ALUHLZR=1
ORH*&	@HL,@ZR	ACH , (@HL)H ← (@HL)H | (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H | ACH if ALUHLZR=1 HL ← HL+4
ORH*%	@HL,@ZR	ACH , (@HL)H ← (@HL)H | (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H | ACH if ALUHLZR=1 ZR ← ZR+4
ORH*$	@HL,@ZR	ACH , (@HL)H ← (@HL)H | (@ZR)H if ALUHLZR=0 ACH , (@HL)H ← (@ZR)H | ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ORH	@ZR	ACH ← (@ZR)H | ACH
ORH#	@ZR	ACH ← (@ZR)H | ACH ZR ← ZR+4
ORH*	@ZR	ACH , (@ZR)H ← (@ZR)H | ACH
ORH*#	@ZR	ACH , (@ZR)H ← (@ZR)H | ACH ZR ← ZR+4
ORH	@ZR,@HL	ACH ← (@ZR)H | (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H | ACH if ALUHLZR=1
ORH&	@ZR,@HL	ACH ← (@ZR)H | (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H | ACH if ALUHLZR=1 HL ← HL+4
ORH%	@ZR,@HL	ACH ← (@ZR)H | (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H | ACH if ALUHLZR=1 ZR ← ZR+4
ORH$	@ZR,@HL	ACH ← (@ZR)H | (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H | ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4
ORH*	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H | (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H | ACH if ALUHLZR=1
ORH*&	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H | (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H | ACH if ALUHLZR=1 HL ← HL+4
ORH*%	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H | (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H | ACH if ALUHLZR=1 ZR ← ZR+4
ORH*$	@ZR,@HL	ACH , (@ZR)H ← (@ZR)H | (@HL)H if ALUHLZR=0 ACH , (@ZR)H ← (@HL)H | ACH if ALUHLZR=1 HL ← HL+4 ZR ← ZR+4

ORM(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

OR 邏輯的加法運算指令，將@HL/ZR 所指向的 data RAM 的內容值與 MUI 的內容值作 OR 邏輯運算。其運算結果會儲存到 AC，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 data RAM。

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

指令語法：

OP code	運算元	指令動作
ORM	@HL	AC ← (@HL) | MUI
ORM#	@HL	AC ← (@HL) | MUI HL ← HL+1
ORM*	@HL	AC , (@HL) ← (@HL) | MUI
ORM*#	@HL	AC , (@HL) ← (@HL) | MUI HL ← HL+1
ORM	@HL,@ZR	AC , (@HL) ← (@ZR) | MUI
ORM&	@HL,@ZR	AC , (@HL) ← (@ZR) | MUI HL ← HL+1
ORM%	@HL,@ZR	AC , (@HL) ← (@ZR) | MUI ZR ← ZR+1
ORM$	@HL,@ZR	AC , (@HL) ← (@ZR) | MUI HL ← HL+1 ZR ← ZR+1
ORM	@ZR	AC ← (@ZR) | MUI
ORM#	@ZR	AC ← (@ZR) | MUI ZR ← ZR+1
ORM*	@ZR	AC , (@ZR) ← (@ZR) | MUI
ORM*#	@ZR	AC , (@ZR) ← (@ZR) | MUI ZR ← ZR+1
ORM	@ZR,@HL	AC , (@ZR) ← (@HL) | MUI
ORM&	@ZR,@HL	AC , (@ZR) ← (@HL) | MUI HL ← HL+1
ORM%	@ZR,@HL	AC , (@ZR) ← (@HL) | MUI ZR ← ZR+1
ORM$	@ZR,@HL	AC , (@ZR) ← (@HL) | MUI HL ← HL+1 ZR ← ZR+1

ORMH(目前在 EV chip (TM8999)不支援)

指令特性：
單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：
OR 邏輯運算指令，將@HL/ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 16bits mode MUI(MUIH)的內容值作 OR 邏輯運算。其運算結果會儲存到 16bits mode AC(ACH)，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

註記：
(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作

ORMH	@HL	ACH ← (@HL)H | MUIH
ORMH#	@HL	ACH ← (@HL)H | MUIH HL ← HL+4
ORMH*	@HL	ACH , (@HL)H ← (@HL)H | MUIH
ORMH*#	@HL	ACH , (@HL)H ← (@HL)H | MUIH HL ← HL+4
ORMH	@HL,@ZR	ACH , (@HL)H ← (@ZR)H | MUIH
ORMH&	@HL,@ZR	ACH , (@HL)H ← (@ZR)H | MUIH HL ← HL+4
ORMH%	@HL,@ZR	ACH , (@HL)H ← (@ZR)H | MUIH ZR ← ZR+4
ORMH$	@HL,@ZR	ACH , (@HL)H ← (@ZR)H | MUIH HL ← HL+4 ZR ← ZR+4
ORMH	@ZR	ACH ← (@ZR)H | MUIH
ORMH#	@ZR	ACH ← (@ZR)H | MUIH ZR ← ZR+4
ORMH*	@ZR	ACH , (@ZR)H ← (@ZR)H | MUIH
ORMH*#	@ZR	ACH , (@ZR)H ← (@ZR)H | MUIH ZR ← ZR+4
ORMH	@ZR,@HL	ACH , (@ZR)H ← (@HL)H | MUIH
ORMH&	@ZR,@HL	ACH , (@ZR)H ← (@HL)H | MUIH HL ← HL+4
ORMH%	@ZR,@HL	ACH , (@ZR)H ← (@HL)H | MUIH ZR ← ZR+4
ORMH$	@ZR,@HL	ACH , (@ZR)H ← (@HL)H | MUIH HL ← HL+4 ZR ← ZR+4

ADCI

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

2 進位的加法運算指令，將 Ry 所指向的 data RAM 的內容值與 Immediate data(D)以及 CF 作加法運算。

其運算結果會分別儲存到 AC 或是 Ry 所指向的 data RAM。

註記：

(1). 加法運算結果的進位會改變 CF 內容值。
(2). 在執行指令之前必須先確認被加數以及加數都必須是相同的進位的格式。
(3). D = 0 ~Fh
(4). Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
ADCI	Ry, D	AC ← (Ry) + D + CF
ADCI*	Ry, D	(Ry) ← (Ry) + D + CF, AC ← (Ry) + D + CF

SBCI

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
2 進位的減法運算指令，將 Ry 所指向的 data RAM 的內容值(被減數)與 Immediate data(D)(減數)以及 CF(借位)作減法運算。

其運算結果會分別儲存到 AC 或是 Ry 所指向的 data RAM。

註記：
(1). 此一減法運算結果的進位會改變 CF 內容值。
(2). 在執行此一指令之前必須先確認被加數以及加數都必須是相同的進位的格式。
(3). D = 0 ~Fh
(4). Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
SBCI	Ry, D	AC ← (Ry) + DB + CF
SBCI*	Ry, D	(Ry) ← (Ry) + DB + CF AC ← (Ry) + DB + CF

ADDI

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
2 進位的加法運算指令，將 Ry 所指向的 data RAM 的內容值與 Immediate data(D)作加法運算。

其運算結果會分別儲存到 AC 或是 Ry 所指向的 data RAM。

註記：
(1). 加法運算結果的進位會改變 CF 內容值。
(2). 在執行指令之前必須先確認被加數以及加數都必須是相同的進位的格式。
(3). D = 0 ~Fh
(4). Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
ADDI	Ry, D	AC ← (Ry) + D
ADDI*	Ry, D	(Ry) ← (Ry) + D,

		AC ← (Ry) + D

SUBI

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
此一指令是 2 進位的減法運算指令，將 Ry 所指向的 data RAM 的內容值(被減數)與 Immediate data(D) (減數)以及借位=1 作減法運算。在此運算中，借位的值會固定為 1 (沒有借位發生)。

其運算結果會分別儲存到 AC 或是 Ry 所指向的 data RAM。

註記：
(1). 減法運算結果的進位會改變 CF 內容值。
(2). 在執行指令之前必須先確認被減數以及減數都必須是相同的進位的格式。
(3). D = 0 ~Fh
(4). Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
SUBI	Ry, D	AC ← (Ry) + DB + 1
SUBI*	Ry, D	(Ry) ← (Ry) + DB + 1, AC ← (Ry) + DB + 1

ADNI

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
2 進位的加法運算指令，將 Ry 所指向的 data RAM 的內容值與 Immediate data(D)作加法運算。

其運算結果會分別儲存到 AC 或是 Ry 所指向的 data RAM。

註記：
(1). 加法運算結果的進位不會改變 CF 內容值。
(2). 在執行指令之前必須先確認被加數以及加數都必須是相同的進位的格式。
(3). D = 0 ~Fh
(4). Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作

ADNI	Ry, D	AC ← (Ry) + D
ADNI*	Ry, D	(Ry) ← (Ry) + D, AC ← (Ry) + D

DAA

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 2 進位的加法運算後所得到的 AC 值轉換成 10 進位的格式，並將轉換的結果存回 AC 或是 Rx, @HL 或是@ZR 所指向的 data RAM 中。使用此指令時必須確認 AC 的內容值是 2 進位加法運算指令的結果。

其資料轉換的對應關係如下：

AC data before DAA execution	CF data before DAA execution	AC data after DAA execution	CF data after DAA execution
0 ≤ AC ≤ 9	CF = 0	no change	no change
A ≤ AC ≤ F	CF = 0	AC= AC+ 6	CF = 1
0 ≤ AC ≤ 3	CF = 1	AC= AC+ 6	no change

註記：

(1). 此一指令的運算結果會改變 CF 內容值。

(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

(3). Rx 在語法上必須是以絕對位址來描述。

(4). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 DAA 指令的 Rx 模式切換成 16bits mode，將 2 進位的加法運算後所得到的 16bits mode AC(H)值轉換成 10 進位的格式，並將轉換的結果存回 ACH 以及 Rx 所指向的 16bits mode data RAM((Rx)H)中。使用此指令時必須確認 ACH 的內容值是 2 進位加法運算指令的結果 & Rm 的 CFN3~0 是由執行 DAA 指令(@RXHM=1)產生的。

指令語法：

OP code	運算元	指令動作
DAA		AC_{10} ← DAA ← AC
DAA*	Rx	$AC(H)_{10}$ ← DAA ← AC(H), $(Rx)(H)_{10}$ ← DAA ← AC(H)
DAA*	@HL	AC_{10} ← DAA ← AC, $(@HL)_{10}$ ← DAA ← AC
DAA*#	@HL	AC_{10} ← DAA ← AC, $(@HL)_{10}$ ← DAA ← AC HL ← HL+1
DAA*	@ZR	AC_{10} ← DAA ← AC, $(@ZR)_{10}$ ← DAA ← AC
DAA*#	@ZR	AC_{10} ← DAA ← AC, $(@ZR)_{10}$ ← DAA ← AC ZR ← ZR+1

DAAH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

將 2 進位的加法運算後所得到的 16bits mode AC(ACH)值轉換成 10 進位的格式，並將轉換的結果存回 ACH 或是@HL 或是@ZR 所指向的 16bits mode data RAM((@HL/ZR)H)中。使用此指令時必須確認 ACH 的內容值是 2 進位加法運算指令的結果 & Rm 的 CFN3~0 是由執行 DAAH 指令產生的。

註記：

(1). 此一指令的運算結果會改變 CF 內容值。
(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
DAAH		ACH_{10} ← DAA ← ACH
DAAH*	@HL	ACH_{10} ← DAA ← ACH, $(@HL)H_{10}$ ← DAA ← ACH
DAAH*#	@HL	ACH_{10} ← DAA ← ACH, $(@HL)H_{10}$ ← DAA ← ACH HL ← HL+4
DAAH*	@ZR	ACH_{10} ← DAA ← ACH, $(@ZR)H_{10}$ ← DAA ← ACH
DAAH*#	@ZR	ACH_{10} ← DAA ← ACH, $(@ZR)H_{10}$ ← DAA ← ACH ZR ← ZR+4

DAS

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

此一指令會將 2 進位的減法運算之後所得到的 AC 值轉換成 10 進位的格式，並將轉換的結果存回 AC 或是 Rx, @HL 或是@ZR 所指向的 data RAM 中。使用此指令時必須確認 AC 的內容值是 2 進位減法運算指令的結果。

其資料轉換的對應關係如下：

AC data before DAS execution	CF data before DAS execution	AC data after DAS execution	CF data after DAS execution
0 ≤ AC ≤ 9	CF = 1	No change	no change
6 ≤ AC ≤ F	CF = 0	AC= AC+A	no change
AC data before DAS execution	CF data before DAS execution	AC data after DAS execution	CF data after DAS execution

註記：

(1). 此一指令的運算結果會改變 CF 內容值。
(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(3). Rx 在語法上必須是以絕對位址來描述。
(4). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 DAS 指令的 Rx 模式切換成 16bits mode，將 2 進位的減法運算後所得到的 16bits mode AC(H)值轉換成 10 進位的格式，並將轉換的結果存回 ACH 以及 Rx 所指向的 16bits mode data RAM((Rx)H)中。使用此指令時必須確認 ACH 的內容值是 2 進位減法運算指令的結果 & Rm 的 CFN3~0 是由執行 DAS 指令(@RXHM=1)產生的。

指令語法：

OP code	運算元	指令動作
DAS		AC_{10} ← DAS ← AC
DAS*	Rx	$AC(H)_{10}$ ← DAS ← AC(H), $Rx(H)_{10}$ ← DAS ← AC(H)
DAS*	@HL	AC_{10} ← DAS ← AC, $(@HL)_{10}$ ← DAS ← AC
DAS*#	@HL	AC_{10} ← DAS ← AC, $(@HL)_{10}$ ← DAS ← AC HL ← HL+1
DAS*	@ZR	AC_{10} ← DAS ← AC, $(@ZR)_{10}$ ← DAS ← AC
DAS*#	@ZR	AC_{10} ← DAS ← AC, $(@ZR)_{10}$ ← DAS ← AC ZR ← ZR+1

DASH(目前在 EV chip (TM8999)不支援)

指令特性：
單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：
將 2 進位的減法運算後所得到的 16bits mode AC(ACH)值轉換成 10 進位的格式，並將轉換的結果存回 ACH 或是@HL 或是@ZR 所指向的 16bits mode data RAM((@HL/ZR)H)中。使用此指令時必須確認 ACH 的內容值是 2 進位減法運算指令的結果 & Rm 的 CFN3~0 是由執行 DASH 指令產生的。

註記：
(1). 此一指令的運算結果會改變 CF 內容值。
(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
DASH		ACH_{10} ← DAS ← ACH
DASH*	@HL	ACH_{10} ← DAS ← ACH, $(@HL)H_{10}$ ← DAS ← ACH
DASH*#	@HL	ACH_{10} ← DAS ← ACH, $(@HL)H_{10}$ ← DAS ← ACH

		HL ← HL+4
DASH*	@ZR	ACH_{10} ← DAS ← ACH, $(@ZR)H_{10}$ ← DAS ← ACH
DASH*#	@ZR	ACH_{10} ← DAS ← ACH, $(@ZR)H_{10}$ ← DAS ← ACH ZR ← ZR+4

INC*

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx, @HL 或是@ZR 所指向的 data RAM 的內容值加 1，並將結果會儲存到 AC，亦可選擇儲存到 Rx, @HL 或是@ZR 所指向的 data RAM。

註記：
(1). 加法運算結果的進位會改變 CF 內容值。
(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(3). Rx 在語法上必須是以絕對位址來描述。
(4). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 INC*指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)儲存到 ACH(16bits mode AC)

指令語法：

OP code	運算元	指令動作
INC*	Rx	AC(H) , (Rx) (H) ← (Rx) (H)+1
INC*	@HL	AC , (@HL) ← (@HL)+1
INC*#	@HL	AC , (@HL) ← (@HL)+1 HL ← HL+1
INC*	@ZR	AC , (@ZR) ← (@ZR)+1
INC*#	@ZR	AC , (@ZR) ← (@ZR)+1 ZR ← ZR+1

INCH*(目前在 EV chip (TM8999)不支援)

指令特性：
單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：
將@HL 或是@ZR 所指向的 16bits mode data RAM 的內容值加 1，並將結果其運算結果會儲存到 16bits mode AC (ACH)，亦可選擇儲存到@HL 或者@ZR 所指向的 16bits mode data RAM。

註記：

(1). 加法運算結果的進位會改變 CF 內容值。
(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
INCH*	@HL	ACH , (@HL)H ← (@HL)H+1
INCH*#	@HL	ACH , (@HL)H ← (@HL)H+1 HL ← HL+4
INCH*	@ZR	ACH , (@ZR)H ← (@ZR)H+1
INCH*#	@ZR	ACH , (@ZR)H ← (@ZR)H+1 ZR ← ZR+4

DEC*

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx, @HL 或是@ZR 所指向的 data RAM 的內容值減 1，並將結果會儲存到 AC，亦可選擇儲存到 Rx, @HL 或是@ZR 所指向的 data RAM。

註記：
(1). 減法運算結果的進位會改變 CF 內容值。
(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(3). Rx 在語法上必須是以絕對位址來描述。
(4). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 DEC*指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)H)儲存到 ACH(16bits mode AC)

指令語法：

OP code	運算元	指令動作
DEC*	Rx	AC(H) , (Rx)(H) ← (Rx)(H)-1
DEC*	@HL	AC , (@HL) ← (@HL)-1
DEC*#	@HL	AC , (@HL) ← (@HL)-1 HL ← HL+1
DEC*	@ZR	AC , (@ZR) ← (@ZR)-1
DEC*#	@ZR	AC , (@ZR) ← (@ZR)-1 ZR ← ZR+1

DECH*(目前在 EV chip (TM8999)不支援)

指令特性：
單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

將@HL 或是@ZR 所指向的 16bits mode data RAM 的內容值減 1，並將結果其運算結果會儲存到 16bits mode AC (ACH)，亦可選擇儲存到@HL/ZR 或者@ZR/HL 所指向的 16bits mode data RAM。

註記：
(1). 減法運算結果的進位會改變 CF 內容值。
(2). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。

指令語法：

OP code	運算元	指令動作
DECH*	@HL	ACH , (@HL)H ← (@HL)H-1
DECH*#	@HL	ACH , (@HL)H ← (@HL)H-1 HL ← HL+4
DECH*	@ZR	ACH , (@ZR)H ← (@ZR)H-1
DECH*#	@ZR	ACH , (@ZR)H ← (@ZR)H-1 ZR ← ZR+4

ANDI

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
AND 邏輯運算指令，將 Ry 所指向的 data RAM 的內容值與 Immediate data(D)作 AND 邏輯運算。

其運算結果會分別儲存到 AC 或是 Ry 所指向的 data RAM。

Truth Table:

A	B	A & B
0	0	0
0	1	0
1	1	1
1	0	0

註記：
(1). D = 0 ~Fh
(2). Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
ANDI	Ry, D	AC ← (Ry) & D
ANDI*	Ry, D	AC ← (Ry) & D, (Ry) ← (Ry) & D

EORI

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
EOR 邏輯運算指令，將 Ry 所指向的 data RAM 的內容值與 Immediate data(D)作 EOR 邏輯運算。

其運算結果會分別儲存到 AC 或是 Ry 所指向的 data RAM。

Truth Table:

A	B	A xor B
0	0	0
0	1	1
1	1	0
1	0	1

註記：
(1). D = 0 ~Fh
(2). Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
EORI	Ry, D	AC ← (Ry) xor D
EORI*	Ry, D	AC ← (Ry) xor D, (Ry) ← (Ry) xor D

ORI

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
OR 邏輯運算指令，將 Ry 所指向的 data RAM 的內容值與 Immediate data(D)作 OR 邏輯運算。

其運算結果會分別儲存到 AC 或是 Ry 所指向的 data RAM。

Truth Table:

A	B	A | B
0	0	0
0	1	1
1	1	1
1	0	1

註記：
(1). D = 0 ~Fh
(2). Ry 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
ORI	Ry, D	AC ← (Ry) | D
ORI*	Ry, D	AC ← (Ry) | D, (Ry) ← Ry | D

SR0, SR1

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 Rx 所指定的 data RAM 的內容值朝 LSB 的方向右位移一個 bit 的位置，並將 MSB 填入 0(SR0)或是 1(SR1)。位移後的結果存回 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Original RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Shifted RAM data	0(1)	(Rx)3	(Rx)2	(Rx)1
Destination AC register	AC3	AC2	AC1	AC0

註記：

(1). Rx 在語法上必須是以絕對位址來描述。

(2). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 SR0/1 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)(H)) 位移後的結果存回 Rx 所指定的 16bits mode data RAM 以及 16bits mode AC(ACH) 。

指令語法：

OP code	運算元	指令動作
SR0	Rx	$(Rx)(H)_n$, AC_n ← $(Rx)(H)_{n+1}$, $AC(H)_{n+1}$, $(Rx)(H)_x$, $AC(H)_x$ ← 0 x=3/f if RXHM=0/1
SR1	Rx	$(Rx)(H)_n$, AC_n ← $(Rx)(H)_{n+1}$, $AC(H)_{n+1}$, $(Rx)(H)_x$, $AC(H)_x$ ← 1 x=3/f if RXHM=0/1

SL0, SL1

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

此一指令將 Rx 所指定的 data RAM 的內容值朝 MSB 的方向左位移一個 bit 的位置，並將 LSB 填入 0(SL0)或是 1(SL1)。位移後的結果存回 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Original RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

Shifted RAM data	(Rx)2	(Rx)1	(Rx)0	0(1)
Destination AC register	AC3	AC2	AC1	AC0

註記：

(1). Rx 在語法上必須是以絕對位址來描述。

(2). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 SR0/1 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)(H))位移後的結果存回 Rx 所指定的 16bits mode data RAM 以及 16bits mode AC(ACH) 。

指令語法：

OP code	運算元	指令動作
SL0	Rx	$(Rx)(H)_{n+1}$, $AC(H)_{n+1} \leftarrow (Rx)(H)_n$, $AC(H)_n$, $(Rx)(H)_0$, $AC(H)_0 \leftarrow 0$
SL1	Rx	$(Rx)(H)_{n+1}$, $AC(H)_{n+1} \leftarrow (Rx)(H)_n$, $AC(H)_n$, $(Rx)(H)_0$, $AC(H)_0 \leftarrow 1$

RRC

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 Rx, @HL 或是@ZR 所指向的 data RAM 的內容值與 CF 朝 LSB 的方向右旋轉一個 bit 的位置。旋轉後的結果存回 Rx, @HL 或是@ZR 所指向的 data RAM 以及 AC。

其位元資料的對應關係如下：

Original RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0	-
Original CF	-	-	-	-	C
Rotated RAM data	C	(Rx)3	(Rx)2	(Rx)1	-
New CF					(Rx)0
Destination AC register	AC3	AC2	AC1	AC0	

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。

(2). 旋轉運算的結果會改變 CF 內容值。

(3). Rx 在語法上必須是以絕對位址來描述。

(4). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 RRC 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值((Rx)(H)) 位移後的結果存回 Rx 所指定的 16bits mode data RAM 以及 16bits mode AC(ACH) 。

指令語法：

OP code	運算元	指令動作
RRC	Rx	Temp $\leftarrow (Rx)_0$,

OP code	運算元	指令動作
		$(Rx)(H)_n$, $AC(H)_n$ ← $(Rx)(H)_{n+1}$, $AC(H)_{n+1}$, $(Rx)(H)_x$, $AC(H)_x$ ← CF, CF ← Temp x=3/15 if RXHM=0/1
RRC	@HL	Temp ← $(@HL)_0$, $(@HL)_n$, AC_n ← $(@HL)_{n+1}$, AC_{n+1}, $(@HL)_3$, AC_3 ← CF, CF ← Temp
RRC#	@HL	Temp ← $(@HL)_0$, $(@HL)_n$, AC_n ← $(@HL)_{n+1}$, AC_{n+1}, $(@HL)_3$, AC_3 ← CF, CF ← Temp HL ← HL+1
RRC	@ZR	Temp ← $(@ZR)_0$, $(@ZR)_n$, AC_n ← $(@ZR)_{n+1}$, AC_{n+1}, $(@ZR)_3$, AC_3 ← CF, CF ← Temp
RRC#	@ZR	Temp ← $(@ZR)_0$, $(@ZR)_n$, AC_n ← $(@ZR)_{n+1}$, AC_{n+1}, $(@ZR)_3$, AC_3 ← CF, CF ← Temp ZR ← ZR+1

RRCH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

將@HL 或是@ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 CF 朝 LSB 的方向右旋轉一個 bit 的位置。旋轉後的結果存回@HL 或是@ZR 所指向的 16bits mode data RAM((@HL/ZR)H)以及 16bits mode AC(ACH)。

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。
(2). 旋轉運算的結果會改變 CF 內容值。

指令語法：

OP code	運算元	指令動作
RRCH	@HL	Temp ← $(@HL)H_0$, $(@HL)H_n$, ACH_n ← $(@HL)H_{n+1}$, ACH_{n+1}, $(@HL)H_{15}$, ACH_{15} ← CF, CF ← Temp
RRCH#	@HL	Temp ← $(@HL)H_0$, $(@HL)H_n$, ACH_n ← $(@HL)H_{n+1}$, ACH_{n+1}, $(@HL)H_{15}$, ACH_{15} ← CF, CF ← Temp

OP code	運算元	指令動作
		HL ← HL+4
RRCH	@ZR	Temp ← (@ZR)H_0 , (@ZR)H_n, ACH_n ← (@ZR)H_{n+1}, ACH_{n+1}, (@ZR)H_{15}, ACH_{15} ← CF, CF ← Temp
RRCH#	@ZR	Temp ← (@ZR)H_0 , (@ZR)H_n, ACH_n ← (@ZR)H_{n+1}, ACH_{n+1}, (@ZR)H_{15}, ACH_{15} ← CF, CF ← Temp ZR ← ZR+4

RLC

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 Rx, @HL 或是@ZR 所指向的 data RAM 的內容值與 CF 朝 MSB 的方向左旋轉一個 bit 的位置。旋轉後的結果存回 Rx, @HL 或是@ZR 所指向的 data RAM 以及 AC。

其位元資料的對應關係如下：

Original RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0	-
Original CF	-	-	-	-	C
Rotated RAM data	(Rx)2	(Rx)1	(Rx)0	C	-
New CF					(Rx)3
Destination AC register	AC3	AC2	AC1	AC0	

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 1。
(2). 旋轉指令的結果會改變 CF 內容值。
(3). Rx 在語法上必須是以絕對位址來描述。
(4). 若 MCU 有提供藉由 Rm 設定 RXHM=1(目前在 EV chip (TM8999)不支援)則可以將 RLC 指令的 Rx 模式切換成 16bits mode，將 Rx 所指向的 16bits mode data RAM 的內容值 ((Rx)(H)) 位移後的結果存回 Rx 所指定的 16bits mode data RAM 以及 16bits mode AC(ACH) 。

指令語法：

OP code	運算元	指令動作
RLC	Rx	Temp ← (Rx)$(H)_x$, (Rx) $(H)_{n+1}$, $AC(H)_{n+1}$ ← (Rx) $(H)_n$, $AC(H)_n$, (Rx) $(H)_0$, $AC(H)_0$ ← CF, CF ← Temp x=3/15 if RXHM=0/1
RLC	@HL	Temp ← $(@HL)_3$, $(@HL)_{n+1}$, AC_{n+1} ← $(@HL)_n$, AC_n, $(@HL)_0$, AC_0 ← CF,

		CF ← Temp
RLC#	@HL	Temp ← $(@HL)_3$, $(@HL)_{n+1}$, AC_{n+1} ← $(@HL)_n$, AC_n, $(@HL)_0$, AC_0 ← CF, CF ← Temp, HL ← HL+1
RLC	@ZR	Temp ← $(@ZR)_3$, $(@ZR)_{n+1}$, AC_{n+1} ← $(@ZR)_n$, AC_n, $(@ZR)_0$, AC_0 ← CF, CF ← Temp
RLC#	@ZR	Temp ← $(@ZR)_3$, $(@ZR)_{n+1}$, AC_{n+1} ← $(@ZR)_n$, AC_n, $(@ZR)_0$, AC_0 ← CF, CF ← Temp, ZR ← ZR+1

RLCH(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

將@HL 或是@ZR 所指向的 16bits mode data RAM((@HL/ZR)H)的內容值與 CF 朝 MSB 的方向左旋轉一個 bit 的位置。旋轉後的結果存回@HL 或是@ZR 所指向的 16bits mode data RAM((@HL/ZR)H)以及 16bits mode AC(ACH)。

註記：

(1). 指令結束之後部分語法會自動將 ZR 或是 HL 的內容值加 4。
(2). 旋轉指令的結果會改變 CF 內容值。

指令語法：

OP code	運算元	指令動作
RLCH	@HL	Temp ← $(@HL)H_{15}$, $(@HL)H_{n+1}$, ACH_{n+1} ← $(@HL)H_n$, ACH_n, $(@HL)H_0$, ACH_0 ← CF, CF ← Temp
RLCH#	@HL	Temp ← $(@HL)H_{15}$, $(@HL)H_{n+1}$, ACH_{n+1} ← $(@HL)H_n$, ACH_n, $(@HL)H_0$, ACH_0 ← CF, CF ← Temp, HL ← HL+4
RLCH	@ZR	Temp ← $(@ZR)H_{15}$, $(@ZR)H_{n+1}$, ACH_{n+1} ← $(@ZR)H_n$, ACH_n, $(@ZR)H_0$, ACH_0 ← CF, CF ← Temp
RLCH#	@ZR	Temp ← $(@ZR)H_{15}$, $(@ZR)H_{n+1}$, ACH_{n+1} ← $(@ZR)H_n$, ACH_n, $(@ZR)H_0$, ACH_0 ← CF,

		CF ← Temp, ZR ← ZR+4

2-6. I/O Port指令(I/O Port Instructions)

SPATH, SPAXH, SPARH(目前在 EV chip (TM8999)不支援)

指令特性：
SPATH：單一字元長度指令，四個 machine cycle，16-bit data transferred。
SPARH：兩個字元長度指令，八個 machine cycle，16-bit data transferred。
SPAXH：兩個字元長度指令，八個 machine cycle。

指令說明：
指令運算元中的內容值可用來同步設定 IOA,B,C,D port 的各項功能。

D = 1, 同時開啟 IOA,B,C,D1~4 各腳位上的 pull-low/high 元件
= 0, 同時關閉 IOA,B,C,D1~4 各腳位上的 pull-low/high 元件

下表說明不同的指令中其運算元的各個位元與 I/O 腳位之間的對應關係如下：

Instruction	IOD4	IOD3	IOD2	IOD1	IOC4	IOC3	IOC2	IOC1	IOB4	IOB3	IOB2	IOB1	IOA4	IOA3	IOA2	IOA1
SPAXH D SETDAT SD	SD15	SD14	SD13	SD12	SD11	SD10	SD9	SD8	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0
SPATH(#) D, @HL	TD15	TD14	TD13	TD12	TD11	TD10	TD9	TD8	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0
SPARH D SETDAT Rx	(RX)15	(RX)14	(RX)13	(RX)12	(RX)11	(RX)10	(RX)9	(RX)8	(RX)7	(RX)6	(RX)5	(RX)4	(RX)3	(RX)2	(RX)1	(RX)0
Rx address	Rx(RAM 位址 bit1,0=11)				Rx(RAM 位址 bit1,0=10)				Rx(RAM 位址 bit1,0=01)				Rx(RAM 位址 bit1,0=00)			

在使用 SPAXH D 指令時，以 SETDAT 所設定的 immediate data (SD) 中的每個位元=0/1 來設定相對應的 I/O 腳位關閉/啟動輸出信號。

在使用 SPATH D 指令時會將@HL 的 LSB 忽略掉，並且以@HL(table ROM 位址 bit0=0, @HL(table ROM 位址 bit0=1) 的連續位址去 table ROM 讀取 16-bit 的資料，並以此資料=0/1 來設定相對應的 I/O 腳位關閉/啟動輸出信號。

在使用 SPARH D 指令時，SETDAT 所設定的 Rx 的最低兩個位元會被忽略掉，並且以 Rx (RAM 位址 bit1,0=00), Rx (RAM 位址 bit1,0=01), Rx (RAM 位址 bit1,0=10), Rx (RAM 位址 bit1,0=11)的連續位址去 data RAM 讀取 16-bit 的資料，並以此資料=0/1 來設定相對應的 I/O 腳位關閉/啟動輸出信號。

註記：
(1). SPATH# D 的語法會在指令結束後自動將 HL 的內容值加 2。
(2). D = 0 or 1
(3). SD = 0 ~ FFFFh
(4). Rx 在語法上必須是以絕對位址來描述。
(5). MCU 會在八個 machine cycle 的指令週期期間暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
SPAXH SETDAT	D SD	開啟/關閉 IOA,B,C,D Pull-low/high 元件 ← D 設定 IOD,C,B,A4~1 的輸出/輸入模式 ← SD15 ~ SD0
SPATH	D, @HL	開啟/關閉 IOA,B,C,D Pull-low/high 元件 ← D 設定 IOD,C,B,A4~1 的輸出/輸入模式 ← T(@HL)15 ~ T(@HL)0
SPATH#	D, @HL	開啟/關閉 IOA,B,C,D Pull-low/high 元件 ← D 設定 IOD,C,B,A4~1 的輸出/輸入模式 ← T(@HL)15 ~ T(@HL)0 HL ← HL + 2
SPARH SETDAT	D Rx	開啟/關閉 IOA,B,C,D Pull-low/high 元件 ← D 設定 IOD,C,B,A4~1 的輸出/輸入模式 ← (Rx)15 ~ (Rx)0

SPA

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元中 X 的值來設定 IOA port 的各項功能

X4 = 1, 同時開啟 IOA1 ~4 各腳位上的 pull-low/high 元件
= 0, 同時關閉 IOA1 ~4 各腳位上的 pull-low/high 元件

X3 = 1, 將 IOA4 設定成 output mode
= 0, 將 IOA4 設定成 input mode

X2 = 1, 將 IOA3 設定成 output mode
= 0, 將 IOA3 設定成 input mode

X1 = 1, 將 IOA2 設定成 output mode
= 0, 將 IOA2 設定成 input mode

X0 = 1, 將 IOA1 設定成 output mode
= 0, 將 IOA1 設定成 input mode

註記：

X4 代表運算元 X 的 MSB, X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SPA	X	開啟/關閉 Pull-low/high 元件 ← X4 設定 IOA4 的輸出/輸入模式 ← X3 設定 IOA3 的輸出/輸入模式← X2 設定 IOA2 的輸出/輸入模式← X1 設定 IOA1 的輸出/輸入模式← X0

IPA

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 IOA port 上的狀態值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Port IOA	IOA4	IOA3	IOA2	IOA1
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Destination AC register	AC3	AC2	AC1	AC0

註記：
(1). Rx 在語法上必須是以絕對位址來描述。
(2). 若 MCU 有提供藉由 Rm 設定 HMIOA=1(目前在 EV chip (TM8999)不支援)則可以切換 IPA 指令將 IOA,B,C,D 16 pins port 的狀態值同步儲存到 Rx 所指定的 16bits mode data RAM((Rx)H)以及 16bits mode AC(ACH) 。

指令語法：

OP code	運算元	指令動作
IPA	Rx(H)	$AC(H)_{3\sim0}$, $(Rx)(H)_{3\sim0} \leftarrow IOA_{4\sim1}$ $ACH_{7\sim4}$, $(Rx)H_{7\sim4} \leftarrow IOB_{4\sim1}$ if HMIOA=1 $ACH_{11\sim8}$, $(Rx)H_{11\sim8} \leftarrow IOC_{4\sim1}$ if HMIOA=1 $ACH_{15\sim12}$, $(Rx)H_{15\sim12} \leftarrow IOD_{4\sim1}$ if HMIOA=1

OPA

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的內容值輸出到 IOA port 的 output register 上。當 IOA port 設定成 output 模式時，output register 的內容值就會輸出到 IOA port。

其位元資料的對應關係如下：

Port IOA	IOA4	IOA3	IOA2	IOA1
Source RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：
(1). Rx 在語法上必須是以絕對位址來描述。
(2). 若 MCU 有提供藉由 Rm 設定 HMIOA=1(目前在 EV chip (TM8999)不支援)則可以切換 OPA 指令將 Rx 所指定的 16bits mode data RAM((Rx)H)的內容值同步輸出到 IOA,B,C,D 16 pins port 的 output register 上。

指令語法：

OP code	運算元	指令動作
OPA	Rx	Output register of $IOA_{4\sim1}$ port $\leftarrow (Rx)(H)_{3\sim0}$ Output register of $IOB_{4\sim1}$ port $\leftarrow (Rx)H_{7\sim4}$ if HMIOA=1

		Output register of $IOC_{4~1}$ port ← $(Rx)H_{11~8}$ if HMIOA=1 Output register of $IOD_{4~1}$ port ← $(Rx)H_{15~12}$ if HMIOA=1

OPAS

指令特性：
單一字元長度指令，四個 machine cycle, 2 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的 bit0, bit1 內容值輸出到 IOA port 的 IOA1 以及 IOA2 pin 上。

將 D 所設定的 1-bit data(0/1)輸出到 IOA port 的 IOA3 pin 上。

IOA4 pin 會在指令的 instruction cycle 結束前輸出一個 BCLK/2 寬度的 High Pulse。

其位元資料的對應關係如下：

Port IOA	IOA4	IOA3	IOA2	IOA1
Bit data	Pulse	D	(Rx)1	(Rx)0

註記：
(1). Rx 在語法上必須是以絕對位址來描述。
(2). D = 0 or 1

指令語法：

OP code	運算元	指令動作
OPAS	Rx, D	IOA1, A2 ← (Rx)0, (Rx)1 IOA3 ← D IOA4 ← pulse

SPB

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中 X 的值來設定 IOB port 的各項功能

X4 = 1, 同時開啟 IOB1 ~4 各腳位上的 pull-low 元件
= 0, 同時關閉 IOB1 ~4 各腳位上的 pull-low 元件

X3 = 1, 將 IOB4 設定成 output mode
= 0, 將 IOB4 設定成 input mode

X2 = 1, 將 IOB3 設定成 output mode
= 0, 將 IOB3 設定成 input mode

X1 = 1, 將 IOB2 設定成 output mode
= 0, 將 IOB2 設定成 input mode

X0 = 1, 將 IOB1 設定成 output mode
= 0, 將 IOB1 設定成 input mode

註記：
X4 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SPB	X	開啟/關閉 Pull-low 元件 ← X4 設定 IOB4 的輸出/輸入模式 ← X3 設定 IOB3 的輸出/輸入模式← X2 設定 IOB2 的輸出/輸入模式← X1 設定 IOB1 的輸出/輸入模式← X0

IPB

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 IOB port 上的狀態值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Port IOB	IOB4	IOB3	IOB2	IOB1
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Destination AC register	AC3	AC2	AC1	AC0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
IPB	Rx	(Rx) ← IOB port, AC ← IOB port

OPB

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的內容值輸出到 IOB port 的 output register 上。當 IOB port 設定成 output 模式時，output register 的內容值就會輸出到 IOB port。

其位元資料的對應關係如下：

Port IOB	IOB4	IOB3	IOB2	IOB1

Source RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
OPB	Rx	Output register of IOB port ← (Rx)

SPC

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中 X 的值來設定 IOC port 的各項功能

X4 = 1, 同時開啟 IOC1 ~4 各腳位上的 pull-low/high 元件
= 0, 同時關閉 IOC1 ~4 各腳位上的 pull-low/high 元件

X3 = 1, 將 IOC4 設定成 output mode
= 0, 將 IOC4 設定成 input mode

X2 = 1, 將 IOC3 設定成 output mode
= 0, 將 IOC3 設定成 input mode

X1 = 1, 將 IOC2 設定成 output mode
= 0, 將 IOC2 設定成 input mode

X0 = 1, 將 IOC1 設定成 output mode
= 0, 將 IOC1 設定成 input mode

註記：
X4 代表運算元 X 的 MSB, X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SPC	X	開啟/關閉 Pull-low/high 元件 ← X4 設定 IOC4 的輸出/輸入模式 ← X3 設定 IOC3 的輸出/輸入模式← X2 設定 IOC2 的輸出/輸入模式← X1 設定 IOC1 的輸出/輸入模式← X0

IPC

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 IOC port 上的狀態值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Port IOC	IOC4	IOC3	IOC2	IOC1
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Destination AC register	AC3	AC2	AC1	AC0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
IPC	Rx	(Rx) ← IOC port, AC ← IOC port

OPC

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的內容值輸出到 IOC port 的 output register 上。當 IOC port 設定成 output 模式時，output register 的內容值就會輸出到 IOC port。

其位元資料的對應關係如下：

Port IOC	IOC4	IOC3	IOC2	IOC1
Source RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
OPC	Rx	Output register of IOC port ← (Rx)

SPD

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中 X 的值來設定 IOD port 的各項功能

X4 = 1, 同時開啟 IOD1 ~4 各腳位上的 pull-low 元件

= 0, 同時關閉 IOD1 ~4 各腳位上的 pull-low 元件

X3 = 1, 將 IOD4 設定成 output mode
= 0, 將 IOD4 設定成 input mode

X2 = 1, 將 IOD3 設定成 output mode
= 0, 將 IOD3 設定成 input mode

X1 = 1, 將 IOD2 設定成 output mode
= 0, 將 IOD2 設定成 input mode

X0 = 1, 將 IOD1 設定成 output mode
= 0, 將 IOD1 設定成 input mode

註記：
X4 代表運算元 X 的 MSB, X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SPD	X	開啟/關閉 Pull-low 元件 ← X4 設定 IOD4 的輸出/輸入模式 ← X3 設定 IOD3 的輸出/輸入模式← X2 設定 IOD2 的輸出/輸入模式← X1 設定 IOD1 的輸出/輸入模式← X0

IPD

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 IOD port 上的輸入值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Port IOD	IOD4	IOD3	IOD2	IOD1
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Destination AC register	AC3	AC2	AC1	AC0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
IPD	Rx	(Rx) ← IOD port, AC ← IOD port

OPD

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 Rx 所指定的 data RAM 的內容值輸出到 IOD port 的 output register 上。當 IOD port 設定成 output 模式時，output register 的內容值就會輸出到 IOD port。

其位元資料的對應關係如下：

Port IOD	IOD4	IOD3	IOD2	IOD1
Source RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：
Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
OPD	Rx	Output register of IOD port ← (Rx)

SPE

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中 X 的值來設定 IOE port 的各項功能

X4 = 1, 同時開啟 IOE1 ~4 各腳位上的 pull-low/high 元件
= 0, 同時關閉 IOE1 ~4 各腳位上的 pull-low/high 元件

X3 = 1, 將 IOE4 設定成 output mode
= 0, 將 IOE4 設定成 input mode

X2 = 1, 將 IOE3 設定成 output mode
= 0, 將 IOE3 設定成 input mode

X1 = 1, 將 IOE2 設定成 output mode
= 0, 將 IOE2 設定成 input mode

X0 = 1, 將 IOE1 設定成 output mode
= 0, 將 IOE1 設定成 input mode

註記：
X4 代表運算元 X 的 MSB, X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SPE	X	開啟/關閉 Pull-low/high 元件 ← X4 設定 IOE4 的輸出/輸入模式 ← X3

		設定 IOE3 的輸出/輸入模式← X2 設定 IOE2 的輸出/輸入模式← X1 設定 IOE1 的輸出/輸入模式← X0

IPE

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 IOE port 上的輸入值儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

Port IOE	IOE4	IOE3	IOE2	IOE1
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Destination AC register	AC3	AC2	AC1	AC0

註記：

Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
IPE	Rx	(Rx) ← IOE port, AC ← IOE port

OPE

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 Rx 所指定的 data RAM 的內容值輸出到 IOE port 的 output register 上。當 IOE port 設定成 output 模式時，output register 的內容值就會輸出到 IOE port。

其位元資料的對應關係如下：

Port IOE	IOE4	IOE3	IOE2	IOE1
Source RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0

註記：

Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
OPE	Rx	Output register of IOE port ← (Rx)

2-7. Table ROM指令(Table ROM Instructions)

LDH

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 HL 所指向的 Table ROM 內容值的高位元的 4 個 bit data 儲存到 AC 以及 Rx 所指向的 data RAM 中。

其資料儲存的對應關係如下：

Source table data	T(@HL)7	T(@HL)6	T(@HL)5	T(@HL)4
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Destination AC register	AC3	AC2	AC1	AC0

註記：
(1). 指令結束之後部分語法會自動將 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). 若 MCU 有提供藉由 Rm 設定 HMLDH=1(目前在 EV chip (TM8999)不支援)則可以切換執行 LDH 指令時會將@HL 的 LSB 忽略掉，並且以@HL(table ROM 位址 bit0=0, @HL(table ROM 位址 bit0=1) 的連續位址去 table ROM 讀取的 16-bits 資料(T(@HL)H) 儲存到 Rx 所指定的 16bits mode data RAM((Rx)H)以及 16bits mode AC(ACH) 。

指令語法：

OP code	運算元	指令動作
LDH	Rx, @HL	AC , (Rx) ← T(@HL) high nibble if HMLDH=0 ACH , (Rx)H ← T(@HL)H if HMLDH=1
LDH*	Rx, @HL	AC , (Rx) ← T(@HL) high nibble if HMLDH=0 ACH , (Rx)H ← T(@HL)H if HMLDH=1 HL ← HL+1/2 if HMLDH=0/1

LDL

指令特性：
單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：
將 HL 所指向的 Table ROM 內容值的低位元的 4 個 bit data 儲存到 AC 以及 Rx 所指向的 data RAM 中。

其資料儲存的對應關係如下：

Source table data	T(@HL)3	T(@HL)2	T(@HL)1	T(@HL)0
Destination RAM data	(Rx)3	(Rx)2	(Rx)1	(Rx)0
Destination AC register	AC3	AC2	AC1	AC0

註記：
(1). 指令結束之後部分語法會自動將 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
LDL	Rx, @HL	(Rx) ← T(@HL) low nibble, AC ← T(@HL) low nibble
LDL*	Rx, @HL	(Rx) ← T(@HL) low nibble, AC ← T(@HL) low nibble, HL ← HL+1

LCDH

指令特性：
單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：
將@HL 所指向兩個連續位址的 table ROM 的 16 bits 內容值儲存到@ZR 所指向的四個連續的 data RAM 位址上。

無論@HL 的 LSB 的實際值為何，在指令執行時都將會被忽略掉，而連續存取的兩個位址將是@HL(RAM 位址 bit0=0)以及@HL(RAM 位址 bit0=1)。

無論@ZR 的最後兩個位元的實際值為何，在指令執行時都將會被忽略掉，而連續存取的四個位址將是@ZR(RAM 位址 bit1,0=00), @ZR(RAM 位址 bit1,0=01), @ZR(RAM 位址 bit1,0=10), @ZR(RAM 位址 bit1,0=11)。

其位元資料的對應關係如下：

Destination RAM addr. & data mapping	RAM addr. = N*+1				RAM addr. = N*			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Table ROM output 16bits data	T(@HL)7	T(@HL)6	T(@HL)5	T(@HL)4	T(@HL)3	T(@HL)2	T(@HL)1	T(@HL)0
@ZR addr.	@ZR(RAM 位址 bit1,0=01)				@ZR(RAM 位址 bit1,0=00)			
Destination RAM addr. & data mapping	RAM addr. = N*+3				RAM addr. = N*+2			
	bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0
Table ROM output 16bits data	T(@HL) 15	T(@HL) 14	T(@HL) 13	T(@HL) 12	T(@HL) 11	T(@HL) 10	T(@HL) 9	T(@HL) 8
@ZR addr.	@ZR(RAM 位址 bit1,0=11)				@ZR(RAM 位址 bit1,0=10)			

*：N 代表四的倍數的 data RAM 位址。

註記：
(1). 在執行指令之前必須先將 ZR 或是 HL 的內容設定完成。
(2). 指令結束之後部分語法會自動將 ZR 的內容值加 4 或是將 HL 的內容值加 2。

指令語法：

OP code	運算元	指令動作
LCDH	@ZR, @HL	(@ZR(RAM 位址 bit1,0=00)) ← T(@HL)3~0, (@ZR(RAM 位址 bit1,0=01)) ← T(@HL)7~4, (@ZR(RAM 位址 bit1,0=10)) ← T(@HL)11~8, (@ZR(RAM 位址 bit1,0=11)) ← T(@HL)15~12
LCDH&	@ZR, @HL	(@ZR(RAM 位址 bit1,0=00)) ← T(@HL)3~0, (@ZR(RAM 位址 bit1,0=01)) ← T(@HL)7~4, (@ZR(RAM 位址 bit1,0=10)) ← T(@HL)11~8, (@ZR(RAM 位址 bit1,0=11)) ← T(@HL)15~12 HL ← HL+2
LCDH%	@ZR, @HL	(@ZR(RAM 位址 bit1,0=00)) ← T(@HL)3~0, (@ZR(RAM 位址 bit1,0=01)) ← T(@HL)7~4, (@ZR(RAM 位址 bit1,0=10)) ← T(@HL)11~8, (@ZR(RAM 位址 bit1,0=11)) ← T(@HL)15~12 ZR ← ZR+4
LCDH$	@ZR, @HL	(@ZR(RAM 位址 bit1,0=00)) ← T(@HL)3~0, (@ZR(RAM 位址 bit1,0=01)) ← T(@HL)7~4, (@ZR(RAM 位址 bit1,0=10)) ← T(@HL)11~8, (@ZR(RAM 位址 bit1,0=11)) ← T(@HL)15~12 HL ← HL+2 ZR ← ZR+4

PTR(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle, 16 bits data transferred。

指令說明：

將 16bits 的 RILH 暫存器內容值燒錄到@HL 所指向的 table ROM 的 word 位址上。

<TM87ML28>

執行 PTR 指令後系統會自動進入 HALT 模式直到燒錄動作完成後才會離開(執行前必須先暫停所有應用中斷以避免造成不可逆的系統異常，可由執行 MSD 指令讀取 PGMF 確認燒錄是否完成，若 PGMF=1 可能原因為燒錄不足需再重覆執行燒錄或是因有其他 HALT rlease 而異常離開。

<TM87P32/PL35/PL36>

OTP Type 只能執行 0->1 單向燒錄，必須先執行 SIAP 指令方可進入 IAP 模式，執行 PTR 指令後系統會自動進入 HALT 模式 & 啟動 Timer2 計數控制燒錄時間，由 Timer2 產生 HALT release 後離開(執行前必須先暫停所有應用中斷以避免因其他 HALT rlease 提早離開造成燒錄時間不足問題)。

其位元資料的對應關係如下：

Destination HL addr.	Table ROM addr. = N*							
RILH output 16bits data	RILH7	RILH6	RILH5	RILH4	RILH3	RILH2	RILH1	RILH0
Destination Table ROM	T(@HL)7	T(@HL)6	T(@HL)5	T(@HL)4	T(@HL)3	T(@HL)2	T(@HL)1	T(@HL)0

16bits data								
Destination HL addr.	Table ROM addr. = N+1*							
RILH output 16bits data	RILH15	RILH14	RILH13	RILH12	RILH11	RILH10	RILH9	RILH8
Destination Table ROM 16bits data	T(@HL)15	T(@HL)14	T(@HL)13	T(@HL)12	T(@HL)11	T(@HL)10	T(@HL)9	T(@HL)8

*：N 代表二的倍數的 Table ROM 位址。

註記：

(1). 在執行指令之前必須先將 HL 的內容設定完成。

(2). 由於 EV chip(TM8999)不支援，故 TM87ML28 執行 PTR 指令後會無法離開 HALT 模式，須啟動 INT 等與應用無關的外部 HALT Release 離開。

指令語法：

OP code	運算元	指令動作
PTR		(@HL(Table ROM 位址 bit0=0)) ← RILH7~0 (@HL(Table ROM 位址 bit0=1)) ← RILH15~8

2-8. RFC功能指令(RFC Function Instructions)

SRF

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
<架構 A>
利用指令運算元中 X 的值來設定各種操作模式，RR,RT,RH 的輸出功能，16bits RFC counter 的啟動。
X6,5,4 = 000, 設定 RFC counter 計數功能是由執行 SRF(X3=1)指令後即立即啟動，執行 SRF(X3=0)指令後才會停止。
在此模式下 RFC Counter 的 clock Source 為 CX。
X6,5,4 = 001, 為設定 RFC counter 計數功能是在 SRF(X3=1)指令執行後由 TMR2 的啟動與停止來同步控制，控制的時間長度如下：
時間長度 = (TMR2 的 clock source 週期) x (TMR2 的起始值)。
在此模式下 RFC Counter 的 clock Source 為 CX。
X6,5,4 = 010, 設定 RFC counter 計數功能會在執行 SRF(X3=1)指令後，由 CX 腳位上出現的第一個輸入信號的下降緣時啟動，並在下一個輸入信號的下降緣出現時停止。
在此模式下 RFC Counter 的 clock Source 為 FREQ。
X6,5,4 = 011, 設定 RFC counter 計數功能會在執行 SRF(X3=1)指令後，由 CX 腳位上出現的第一個輸入信號的上升緣時啟動，並在下一個輸入信號的下降緣出現時停止。
在此模式下 RFC Counter 的 clock Source 為 FREQ。
X6,5,4 = 100, 設定 RFC counter 計數功能是由執行 SRF(X3=1)指令後即立即啟動，執行 SRF(X3=0)指令後才會停止。
在此模式下 RFC Counter 的 clock Source 為 FREQ。
X6,5,4 = 101, 為設定 RFC counter 計數功能是在 SRF(X3=1)指令執行後由 TMR2 的啟動與停止來同步控制，控制的時間長度如下：
時間長度 = (TMR2 的 clock source 週期) x (TMR2 的起始值)。
在此模式下 RFC Counter 的 clock Source 為 FREQ。
X6,5,4 = 111, 設定 RFC counter 計數功能會在執行 SRF(X3=1)指令後立即啟動，並在 CX 腳位上出現的第一個輸入信號的上升緣出現時停止。
在此模式下 RFC Counter 的 clock Source 為 FREQ。
X6,5,4=011 & 1XX：目前在 EV chip (TM8999)不支援。
X3 = 1, 開啟 16bits RFC counter 的計數功能。
= 0, 關閉 16bits RFC counter 的計數功能。
X2 = 1, 開啟 RH(RFC2)腳位的輸出功能。
= 0, 關閉 RH(RFC2)腳位的輸出功能，高阻抗狀態。
X1 = 1, 開啟 RT(RFC1)腳位的輸出功能。
= 0, 關閉 RT(RFC1)腳位的輸出功能，高阻抗狀態。
X0 = 1, 開啟 RR(RFC0)腳位的輸出功能。
= 0, 關閉 RR(RFC0)腳位的輸出功能，高阻抗狀態。
<架構 B>

利用指令運算元中 X 的值來設定 RFC0~RFC5 的輸出功能。

X5 = 1, 開啟 RFC5 腳位的輸出功能。
= 0, 關閉 RFC5 腳位的輸出功能，高阻抗狀態。

X4 = 1, 開啟 RFC4 腳位的輸出功能。
= 0, 關閉 RFC4 腳位的輸出功能，高阻抗狀態。

X3 = 1, 開啟 RFC3 腳位的輸出功能。
= 0, 關閉 RFC3 腳位的輸出功能，高阻抗狀態。

X2 = 1, 開啟 RFC2 腳位的輸出功能。
= 0, 關閉 RFC2 腳位的輸出功能，高阻抗狀態。

X1 = 1, 開啟 RFC1 腳位的輸出功能。
= 0, 關閉 RFC1 腳位的輸出功能，高阻抗狀態。

X0 = 1, 開啟 RFC0 腳位的輸出功能。
= 0, 關閉 RFC0 腳位的輸出功能，高阻抗狀態。

註記：
X5 代表運算元 X 的 MSB, X0 代表運算元 X 的 LSB。

指令語法：
<架構 A>

OP code	運算元	指令動作
SRF	X	設定 RFC 的控制模式 ← X5,4 開啟 16bits RFC counter 的計數功能 ← X3 設定 RH(RFC2) 的輸出功能 ← X2 設定 RT(RFC1) 的輸出功能 ← X1 設定 RR(RFC0) 的輸出功能 ← X0

<架構 B>

OP code	運算元	指令動作
SRF	X	設定 RFC5 的輸出功能 ← X5 設定 RFC4 的輸出功能 ← X4 設定 RFC3 的輸出功能 ← X3 設定 RFC2 的輸出功能 ← X2 設定 RFC1 的輸出功能 ← X1 設定 RFC0 的輸出功能 ← X0

SCNT(RFC 架構 B)

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元 X 值來設定 CX2/CX 這兩組 RFC counter 的各種操作模式以及選擇不同類型的 RFC counter 以及所使用的 clock source。

X5 = 1, 設定使用 CX2 pin 的這組 RFC counter。
利用 X3，X2 設定操作模式，X1，X0 設定 Counter 類型以及 X4 設定 Clock Source。
= 0, 設定使用 CX pin 的這組 RFC counter。
利用 X3，X2 設定操作模式，X1，X0 設定 Counter 類型以及 X4 設定 Clock Source。

X4 = 1, 在不是以 CX2/CX pin 控制 RFC counter 的操作模式下，將頻率產生器的輸出信號(FREQ)設定為 X5 所選擇的 RFC counter 的 clock source。此設定值須在 X3=0 時才有效。(此設定值須在 X3=0 時才有效)
= 0, 在不是以 CX2/CX pin 控制 RFC counter 的操作模式下，將 CX2/CX pin 上的輸入信號設定為 X5 所選擇的 RFC counter 的 clock source。(此設定值須在 X3=0 時才有效)

X3, X2 = 00, 設定 CX2/CX RFC counter 計數功能是由執行 SF2 指令後即立即啟動，執行 RF2 指令後才會停止。
= 01, 設定 CX2/CX RFC counter 計數功能是在 SF2 指令執行後由 TMR2 的啟動與停止來同步控制，控制的時間長度如下：
時間長度 = (TMR2 的 clock source 週期) x (TMR2 的起始值)。
在此操作模式下的 RFC 功能一旦開始啟動後，另一組的 RFC 功能則不可再以同樣的 TMR2 控制方式來啟動。
= 10, 設定 CX2/CX RFC counter 計數功能會在執行 SF2 指令後，由 CX2/CX 腳位上出現的第一個輸入信號的下降緣時啟動，並在下一個輸入信號的下降緣出現時停止。
在此模式下 RFC Counter 的 clock Source 固定為 frequency generator 的輸出信號(FREQ)。
= 11, 設定 CX2/CX RFC counter 計數功能會在執行 SF2 指令後，由 CX2/CX 腳位上出現的第一個輸入信號的上昇緣時啟動，並在下一個輸入信號的下降緣出現時停止。
在此模式下 RFC Counter 的 clock Source 固定為 FREQ。

X1, X0 = 00, 設定 16-bit RFC counter 作為 CX2/CX RFC counter
這是 MCU 的原始設定狀態。
= 01, 設定 TMR1 作為 CX2/CX RFC counter
= 10, 設定 TMR2 作為 CX2/CX RFC counter
= 11, 設定 TMR3 作為 CX2/CX RFC counter

註記：

(1). 當 TMR1~3 設成 RFC Counter 並開始動作時，不可再同時執行設定 Timer 的相關指令，以免影響 RFC counter 的計數功能。
(2). X5 代表運算元 X 的 MSB, X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SCNT	X	選擇設定 CX2 or CX ← X5 設定 CX2/CX RFC 計數器在 X3=0 時的 clock source ← X4

		設定 CX2/CX RFC Counter 啟動/停止的控制方式 ← X3, X2 設定 CX2/CX RFC Counter ← X1, X0

MRF1

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 16-bit RFC counter 內容值中的 bit3 ~ bit0 儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

RFC counter	RFC3	RFC2	RFC1	RFC0
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

註記：

(1). Rx 在語法上必須是以絕對位址來描述。

(2). 若 MCU 有提供藉由 Rm 設定 HMMRF=1(目前在 EV chip (TM8999)不支援)則可以切換 MRF1 指令將 16-bit RFC counter 內容值中的 all bits 儲存到 Rx 所指定的 16bits mode data RAM((Rx)H)以及 16bits mode AC(ACH) 。

指令語法：

OP code	運算元	指令動作
MRF1	Rx	AC(H) , (Rx)(H) ← $RFC_{3\sim0}$ $ACH_{7\sim4}$, $(Rx)H_{7\sim4}$ ← $RFC_{7\sim4}$ if HMMRF=1 $ACH_{11\sim8}$, $(Rx)H_{11\sim8}$ ← $RFC_{11\sim8}$ if HMMRF=1 $ACH_{15\sim12}$, $(Rx)H_{15\sim12}$ ← $RFC_{15\sim12}$ if HMMRF=1

MRF2

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 16-bit RFC counter 內容值中的 bit7 ~ bit4 儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

RFC counter	RFC7	RFC6	RFC5	RFC4
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

註記：

Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MRF2	Rx	(Rx) ← RFC7 ~ RFC4, AC ← RFC7 ~ RFC4

MRF3

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 16-bit RFC counter 內容值中的 bit11 ~ bit8 儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

RFC counter	RFC11	RFC10	RFC9	RFC8
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

註記：

Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MRF3	Rx	(Rx) ← RFC11 ~ RFC8, AC ← RFC11 ~ RFC8

MRF4

指令特性：

單一字元長度指令，四個 machine cycle, 4 bits data transferred。

指令說明：

將 16-bit RFC counter 內容值中的 bit15 ~ bit12 data 儲存到 Rx 所指定的 data RAM 以及 AC。

其位元資料的對應關係如下：

RFC counter	RFC15	RFC14	RFC13	RFC12
Rx	(Rx)3	(Rx)2	(Rx)1	(Rx)0
AC	AC3	AC2	AC1	AC0

註記：

Rx 在語法上必須是以絕對位址來描述。

指令語法：

OP code	運算元	指令動作
MRF4	Rx	(Rx) ← RFC15 ~ RFC12, AC ← RFC15 ~ RFC12

2-9. 跳躍/呼叫指令(Jump/Call Instructions)

JB0

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
當 AC 的 bit0 的內容值為 1 時，程式在下一個指令的時候會跳到 PC = X 的位址去執行指令；

如果 AC 的 bit0 的內容值不等於 1 時，程式會執行下一個位址的指令。

註記：
(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定，若 MCU 未提供 SPBK 的 bank 指令功能，則因 JB0 只能在同一 Page(2K)內跳躍，故此時 X <= 7FFh)
(2). 若 MCU 提供 SPBK 的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動在這個指令之前插入一個 SPBK 的指令。

指令語法：

OP code	運算元	指令動作
JB0	X	If AC0 = 1, PC ← X； If AC0 != 1, PC ← PC+1

JB1

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
當 AC 的 bit1 的內容值為 1 時，程式在下一個指令的時候會跳到 PC = X 的位址去執行指令；

如果 AC 的 bit1 的內容值不等於 1 時，程式會執行下一個位址的指令。

註記：
(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定，若 MCU 未提供 SPBK 的 bank 指令功能，則因 JB1 只能在同一 Page(2K)內跳躍，故此時 X <= 7FFh)
(2). 若 MCU 提供 SPBK 的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動在這個指令之前插入一個 SPBK 的指令。

指令語法：

OP code	運算元	指令動作

JB1	X	If AC1 = 1, PC ← X； If AC1 != 1, PC ← PC+1

JB2

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
當 AC 的 bit2 的內容值為 1 時，程式在下一個指令的時候會跳到 PC = X 的位址去執行指令；

如果 AC 的 bit2 的內容值不等於 1 時，程式會執行下一個位址的指令。

註記：
(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定，若 MCU 未提供 SPBK 的 bank 指令功能，則因 JB2 只能在同一 Page(2K)內跳躍，故此時 X <= 7FFh)
(2). 若 MCU 提供 SPBK 的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動在這個指令之前插入一個 SPBK 的指令。

指令語法：

OP code	運算元	指令動作
JB2	X	If AC2 = 1, PC ← X； If AC2 != 1, PC ← PC+1

JB3

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
當 AC 的 bit3 的內容值為 1 時，程式在下一個指令的時候會跳到 PC = X 的位址去執行指令；

如果 AC 的 bit3 的內容值不等於 1 時，程式會執行下一個位址的指令。

註記：
(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定，若 MCU 未提供 SPBK 的 bank 指令功能，則因 JB3 只能在同一 Page(2K)內跳躍，故此時 X <= 7FFh)
(2). 若 MCU 提供 SPBK 的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動在這個指令之前插入一個 SPBK 的指令。

指令語法：

OP code	運算元	指令動作
JB3	X	If AC3 = 1, PC ← X； If AC3 != 1，PC ← PC+1

JNZ

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

當 AC 的內容值不等於 0 時，程式在下一個指令的時候會跳到 PC = X 的位址去執行指令；

如果 AC 的內容值等於 0 時，程式會執行下一個位址的指令。

註記：

(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定，若 MCU 未提供 SPBK 的 bank 指令功能，則因 JNZ 只能在同一 Page(2K)內跳躍，故此時 X <= 7FFh)

(2). 若 MCU 提供 SPBK 的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動在這個指令之前插入一個 SPBK 的指令。

指令語法：

OP code	運算元	指令動作
JNZ	X	If AC != 0, PC ← X； If AC = 0, PC ← PC+1

JZ

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

當 AC 的內容值等於 0 時，程式在下一個指令的時候會跳到 PC = X 的位址去執行指令；

如果 AC 的內容值不等於 0 時，程式會執行下一個位址的指令。

註記：

(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定，若 MCU 未提供 SPBK 的 bank 指令功能，則因 JZ 只能在同一 Page(2K)內跳躍，故此時 X <= 7FFh)

(2). 若 MCU 提供 SPBK 的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動在這個指令之前插入一個 SPBK 的指令。

指令語法：

OP code	運算元	指令動作
JZ	X	If AC = 0, PC ← X； If AC ! = 0, PC ← PC+1

JNC

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

當 CF 的內容值不等於 1 時，程式在下一個指令的時候會跳到 PC = X 的位址去執行指令；

如果 CF 的內容值等於 1 時，程式會執行下一個位址的指令。

註記：

(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定，若 MCU 未提供 SPBK 的 bank 指令功能，則因 JNC 只能在同一 Page(2K)內跳躍，故此時 X <= 7FFh)

(2). 若 MCU 提供 SPBK 的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動在這個指令之前插入一個 SPBK 的指令。

指令語法：

OP code	運算元	指令動作
JNC	X	If CF != 1, PC ← X； If CF = 1，PC ← PC+1

JC

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

當 CF 的內容值等於 1 時，程式在下一個指令的時候會跳到 PC = X 的位址去執行指令；

如果 CF 的內容值不等於 1 時，程式會執行下一個位址的指令。

註記：

(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定，若 MCU 未提供 SPBK 的 bank 指令功能，則因 JC 只能在同一 Page(2K)內跳躍，故此時 X <= 7FFh)

(2). 若 MCU 提供 SPBK 的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動在這個指令之前插入一個 SPBK 的指令。

指令語法：

OP code	運算元	指令動作
JC	X	If CF = 1, PC ← X； If CF != 1, PC ← PC+1

CALL

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

呼叫副程式的指令，指令執行後會先將下個指令的位址儲存到 Stack 中，程式在下一個指令週期的時候會跳到 PC = X 的位址去執行指令。

註記：

(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定)

(2). 若 MCU 提供 SPBK 的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動在這個指令之前插入一個 SPBK 的指令。

(3). 若 MCU 提供 SF2(X6=1)的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動將這個指令跳躍位址更改為同一 bank 的轉換區內位址，並在原來和目標的 bank 轉換區內分別插入 SF2(X6=1)和 JMP 共兩個指令。

指令語法：

OP code	運算元	指令動作
CALL	X	STACK ← PC+1, STACK pointer ← STACK pointer + 1, PC ← X

JMP

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

指令執行後，程式在下一個指令週期的時候會跳到 PC = X 的位址去執行指令。

註記：

(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定)

(2). 若 MCU 提供 SPBK 的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動在這個指令之前插入一個 SPBK 的指令。

(3). 若 MCU 提供 SF2(X6=1)的 bank 指令功能，則如果跳躍目的地的位址與目前的指令位址分屬於不同的 bank 時，ICE 的 compiler 程式會自動將這個指令跳躍位址更改為同一 bank 的轉換區內位址，並在原來和目標的 bank 轉換區內分別插入 SF2(X6=1)和 JMP 共兩個指令。

指令語法：

OP code	運算元	指令動作
JMP	X	PC ← X

CPHL

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
當 HL 的低位元組(IDBF7~IDBF0)的內容值不等於 X 的設定值時，程式會正常執行下一個位址的指令；

如果 HL 的低位元組(IDBF7~IDBF0)的內容值等於 X 的設定值時，程式會在下一個指令週期強制執行 NOP 指令。

註記：
(1). X = 0h ~ FFh
(2). 目前除了 TM8726 以外其他 MCU 會在指令週期中的四個 machine cycle 期間與下一個位址指令的指令週期暫停　所有的中斷請求。

指令語法：

OP code	運算元	指令動作
CPHL	X	If IDBF7~0 != X, 下一個指令正常執行； If IDBF7~0 = X, 下一個指令改成執行 NOP

CPZR

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
當 ZR 的低位元組(ZRBF7~ZRBF0)的內容值不等於 X 的設定值時，程式會正常執行下一個位址的指令；

如果 ZR 的低位元組(ZRBF7~ZRBF0)的內容值等於 X 的設定值時，程式會在下一個指令週期強制執行 NOP 指令。

註記：

(1). X = 0h ~ FFh
(2). MCU 會在指令週期中的四個 machine cycle 期間與下一個位址指令的指令週期暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
CPZR	X	If ZRBF7~0 != X, 下一個指令正常執行； If ZRBF7~0 = X, 下一個指令改成執行 NOP

CPHLH

指令特性：
兩個字元長度指令，八個 machine cycle。

指令說明：
當 HL 的 16-bit 內容值不等於 X 的設定值時，程式會正常執行下一個位址的指令；如果 HL 的 16-bit 的內容值等於 X 的設定值時，程式會在下一個指令週期強制執行 NOP 指令。

註記：
(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定)
(2). MCU 會在指令週期中的八個 machine cycle 期間與下一個位址指令的指令週期暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
CPHLH SETDAT	X	If IDBF != X, 下一個指令正常執行； If IDBF = X, 下一個指令改成執行 NOP

CPZRH

指令特性：
兩個字元長度指令，八個 machine cycle。

指令說明：
當 ZR 的 16-bit 內容值不等於 X 的設定值時，程式會正常執行下一個位址的指令；如果 ZR 的 16-bit 的內容值等於 X 的設定值時，程式會在下一個指令週期強制執行 NOP 指令。

註記：
(1). X = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定)
(2). MCU 會在指令週期中的八個 machine cycle 期間與下一個位址指令的指令週期暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
CPZRH SETDAT	X	If ZRBF != X, 下一個指令正常執行； If ZRBF = X, 下一個指令改成執行 NOP

CAC

指令特性：

多字元長度指令，四個或八個 machine cycle。

指令說明：

這是一個多重選擇的副程式呼叫指令，指令中可以設定最多 16 個不同的副程式位址，並且依據 AC 的內容值來決定被呼叫的副程式位址。

指令運算元 X 的值代表指令中可供呼叫的副程式位址的總數(X+1)。

當 AC<=X 時，MCU 會先將下一個指令的位址(PC+X+2)儲存到 STACK 中，然後將 AC 值所對應的副程式位址載入 program counter。此情況下指令週期為八個 machine cycle。

當 AC>X 時，MCU 不會呼叫任何副程式，只會將下一個指令的位址(PC+X+2)載入 program counter 中。此情況下指令週期為四個 machine cycle。

註記：

(1). X = 0 ~ Fh。
(2). Addr(X) = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定)
(3). MCU 會在指令的八個 machine cycle 期間暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
CAC SETDAT SETDAT : SETDAT	X Addr(0) Addr(1) : Addr(X)	If AC <= X, PC=Addr(AC), STACK ← PC +X + 2, STACK pointer ← STACK pointer +1; If AC > X, PC ← PC+X+2;

JAC

指令特性：

多字元長度指令，四個或八個 machine cycle。

指令說明：

這是一個多重選擇的程式跳躍指令，指令中可以設定最多 16 個不同的目的位址，並且依據 AC 的內容值來決定跳躍的目的地位址。

指令運算元 X 的值代表指令中可供跳躍的目的位址的總數(X+1)。

當 AC<=X 時，MCU 會將目前 AC 內容值所對應到的目的位址載入 program counter 中，此情況下指令週期為八個 machine cycle。

當 AC>X 時，MCU 會將下一個指令的位址載入 program counter 中，此情況下指令週期為四個 machine cycle。

註記：
(1). X = 0 ~ Fh。
(2). Addr(X) = 0h ~ FFFFh。(X 值範圍視各個 MCU 規格而定)
(3). MCU 會在指令的八個 machine cycle 期間暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
JAC SETDAT SETDAT : SETDAT	X Addr(0) Addr(1) : Addr(X)	If AC <= X, PC ← Addr(AC); If AC > X, PC ← PC+X+2;

RTS

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
讓程式離開副程式。

指令語法：

OP code	運算元	指令動作
RTS		STACK pointer ← STACK pointer – 1 PC ← STACK

2-10. RAM Page/ROM bank設定指令 (RAM Page/ROM bank Setting Instructions)

SRY

指令特性：
單一字元長度指令，四個 machine cycle, compiler 專用指令。

指令說明：
利用指令運算元中的 X 來設定 Ry memory 的 page 值。在程式中使用 Ry 的位址時必須以絕對位址($0h ~ $1FFFh)來描述，當 compiler 發現 Ry 的位址不屬於 initial page 的範圍，則會自動在該指令前插入 SRY 指令，以便程式能在正確的 Ry page 中存取資料。

因為 page 7 才是 Ry 的 initial page，如果將指令中的 Ry 設定為 0~FH 時， 則表示指令是在 Ry page 0 中存取資料，Compiler 將在該指令之前插入 “SRY 0” 指令。

註記：
(1). X = 0h ~ 6h, 8h ~ 1FFh。(X 值範圍視各個 MCU 規格而定)
(2). Ry memory 的 initial page 是在 page 7 (Ry=70~7Fh)
(3). MCU 會在指令週期中的四個 machine cycle 期間與下一個位址指令的指令週期暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
SRY	X	Ry page number ← X

ERY

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中的 X 來設定 Ry memory 的 page 值，且 Page 值將一直維持不變，除非執行另一個 ERY 指令來修改 page 值，或是執行 CLPG X 指令(設定 X1=1)來解除設定。

程式執行任何一個 ERY，ERX 或是 ELZ 指令後，MCU 就會禁止所有的中斷服務，必須等到所有 Ry，Rx 以及 Lz page 的設定解除之後才會恢復中斷服務。

註記：
(1). X = 0h ~ 6h, 8h ~ 1FFh。(X 值範圍視各個 MCU 規格而定)
(2). 程式不可在 memory page 限制區間中存取 initial page 的 data RAM 資料。

指令語法：

OP code	運算元	指令動作
ERY	X	Ry page number ← X 固定 Ry page number 禁止所有的中斷服務

SRX

指令特性：

單一字元長度指令，四個 machine cycle，compiler 專用指令。

指令說明：

利用指令運算元中的 X 來設定 Rx memory 的 page 值。在程式中使用 Rx 的位址時必須以絕對位址($0h ~ $1FFFh)來描述，當 compiler 發現 Rx 的位址不屬於 initial page 的範圍，則會自動在該指令前插入 SRX 指令，以便程式能在正確的 Rx page 中存取資料。

註記：

(1). X = 1h ~ 3Fh。(X 值範圍視各個 MCU 規格而定)
(2). Rx memory 的 initial page 是在 page 0(Rx=00~7FH)。
(3). MCU 會在指令週期中的四個 machine cycle 期間與下一個位址指令的指令週期暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
SRX	X	Rx page number ← X

ERX

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元中的 X 來設定 Rx memory 的 page 值，且 Page 值將一直維持不變，除非執行另一個 ERX 指令來修改 page 值，或是執行 CLPG X 指令(設定 X0=1)來解除設定。

程式執行任何一個 ERY，ERX 或是 ELZ 指令後，MCU 就會禁止所有的中斷服務，必須等到所有 Ry，Rx 以及 Lz page 的設定解除之後才會恢復中斷服務。

註記：

(1). X = 1h ~ 3Fh。(X 值範圍視各個 MCU 規格而定)
(2). 程式不可在 memory page 限制區間中存取 initial page 的 data RAM 資料。

指令語法：

OP code	運算元	指令動作
ERX	X	Rx page number ← X 固定 Rx page number 禁止所有的中斷服務

SLZ

指令特性：

單一字元長度指令，四個 machine cycle, compiler 專用指令。

指令說明：

利用指令運算元中的 X 來設定 Lz memory 的 page 值。在程式中使用 Lz 的位址時必須以絕對位址($00h ~ $7Fh)來描述，當 compiler 發現 Lz 的位址不屬於 initial page 的範圍，則會自動在該指令前插入 SLZ 指令，以便程式能在正確的 Lz page 中存取資料。

註記：

(1). X = 1h ~ 3h。(X 值範圍視各個 MCU 規格而定)
(2). Lz memory 的 initial page 是在 page 0
(3). MCU 會在指令週期中的四個 machine cycle 期間與下一個位址指令的指令週期暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
SLZ	X	Lz page number ← X

ELZ

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元中的 X 來設定 Lz memory 的 page 值，且 Page 值將一直維持不變，除非執行另一個 ELZ 指令來修改 page 值，或是執行 CLPG X 指令(設定 X2=1)來解除設定。

程式執行任何一個 ERY，ERX 或是 ELZ 指令後，MCU 就會禁止所有的中斷服務，必須等到所有 Ry，Rx 以及 Lz page 的設定解除之後才會恢復中斷服務。

註記：

(1). X = 1h ~ 3h。(X 值範圍視各個 MCU 規格而定)
(2). 程式不可在 memory page 限制區間中存取 initial page 的 data RAM 資料。

指令語法：

OP code	運算元	指令動作

ELZ	X	Lz page number ← X 固定 Lz page number 禁止所有的中斷服務

CLPG

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

此一指令可以將 ERX, ERY, ELZ 等指令所設定的 page number 動作解除，並且讓所有的中斷服務恢復運作。

其位元資料的對應關係如下：

運算元 X	X0=1	X1=1	X2=1
執行動作	解除 ELZ 的設定	解除 ERY 的設定	解除 ERX 的設定

註記：

X2 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
CPLG	X	解除 ELZ 的設定 ← X0=1 解除 ERY 的設定← X1=1 解除 ERX 的設定← X2=1

SPBK

指令特性：

單一字元長度指令，四個 machine cycle, compiler 專用指令。

指令說明：

利用指令運算元中的 X 來設定 Program ROM 的 bank 值。當 Compiler 在編譯程式的過程中發現 JMP, CALL 等跳躍指令的目的位址 bank 值與該指令目前的 bank 值不同時，會自動在該指令之前插入 SPBK 指令，以便讓程式能夠正確的跳躍到目的位址。

註記：

(1). X = 00h ~ 1Fh(X 值範圍視各個 MCU 規格而定)

(2). MCU 會在指令週期中的四個 machine cycle 期間與下一個位址指令的指令週期暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
SPBK	X	Program ROM bank number ← X

2-11. Timer指令(Timer Instructions)

TMS, TMSX

指令特性：
單一字元長度指令，四個 machine cycle, 4/8 bits data transferred。

指令說明：
指令運算元中的 immediate data(X)，HL 所指向的 table ROM 內容值，AC 以及 Rx 所指向的 data RAM 內容值可用來設定 6-bit TMR1 的 clock source 以及 timer 的起始值。指令結束之後 TMR1 會開始動作。

這個指令只能用來設定 6-bit 的 TMR1。

每個位元的設定值與其對應的功能選項如下所示：

OPCODE	Select clock(CS3~0)				Pre-set data of timer(CT5~0)					
	CS3	CS2	CS1	CS0	CT5	CT4	CT3	CT2	CT1	CT0
TMSX X	0	X8	X7	X6	X5	X4	X3	X2	X1	X0
TMS Rx	0	0	AC3	AC2	AC1	AC0	(Rx)3	(Rx)2	(Rx)1	(Rx)0
TMS @HL	0	0	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0

下表說明 6-bit TMR1 的 clock source 的設定方式：

Bit setting in instruction				clock source of timer 1
CS3	CS2	CS1	CS0	
0	0	0	0	~PH9
0	0	0	1	~PH3
0	0	1	0	~PH15
0	0	1	1	FREQ
0	1	0	0	~PH5
0	1	0	1	~PH7
0	1	1	0	~PH11
0	1	1	1	~PH13

在使用 TMSX 指令時會以 immediate data(X)直接設定 TMR1 的 clock source 以及預設值。

在使用 TMS Rx 指令時會以 Rx 所指向的 data RAM 內容值以及 AC 的內容值來設定 TMR1 的 clock source 以及預設值。

在使用 TMS @HL 指令時會以 HL 所指向的 table ROM 內容值用來設定 TMR1 的 clock source 以及預設值。

註記：
(1). TMS# @HL 的語法會在指令結束後自動將 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). X = 0 ~ 1FFh。

指令語法：

OP code	運算元	指令動作
TMS	Rx	Preset initial data ← (AC1, 0), ((Rx)3 ~ (Rx)0)

		Selection of clock source ← (AC3, 2)
TMS	@HL	Preset initial data ← T(@HL)5 ~ T(@HL)0 Selection of clock source ← T(@HL)7 ~ T(@HL)6
TMS#	@HL	Preset initial data ← T(@HL)5 ~ T(@HL)0 Selection of clock source ← T(@HL)7 ~ T(@HL)6 HL ← HL + 1
TMSX	X	Preset initial data ← (X5 ~ X0) Selection of clock source ← (X8 ~ X6)

T1XH, T1RH, T1TH

指令特性：

T1TH：單一字元長度指令，四個 machine cycle，16-bit data transferred。
T1RH：兩個字元長度指令，八個 machine cycle，16-bit data transferred。
T1XH：兩個字元長度指令，八個 machine cycle。

指令說明：

指令中會以 SETDAT 所設定的 immediate data(SD)，@HL 所指向的 table ROM 內容值或是 Rx 所指向的 data RAM 內容值來設定 6-bit TMR1 的 clock source 以及 timer 的預設值。指令結束之後 TM1 會開始動作。

除了 STE X1,0=01 之外這個指令只能用來設定 6-bit 的 TMR1。

每個位元的設定值與其對應的功能選項如下所示：(下表中未列出的設定位元表示該位元的設定值是可以忽略的)

	Select clock(CS3~0)				Pre-set data of timer(CT5~0)					
OPCODE	CS3	CS2	CS1	CS0	CT5	CT4	CT3	CT2	CT1	CT0
T1XH SETDAT SD	SD15	SD14	SD13	SD12	SD11	SD10	SD9	SD8	SD7	SD6
T1TH @HL	TD15	TD14	TD13	TD12	TD11	TD10	TD9	TD8	TD7	TD6
	Select clock(CS3~0)				Pre-set data of timer(CT5~0)					
OPCODE	CS3	CS2	CS1	CS0	CT5	CT4	CT3	CT2	CT1	CT0
T1RH SETDAT RX	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2
	(Rx)15	(Rx)14	(Rx)13	(Rx)12	(Rx)11	(Rx)10	(Rx)9	(Rx)8	(Rx)7	(Rx)6
Rx address	Rx(RAM 位址 bit1,0=11)				Rx(RAM 位址 bit1,0=10)				Rx(RAM 位址 bit1,0=01)	

當這些指令用來設定 6-bit 模式的 TMR1 為了可以選擇 Ctm1=CX/CX2/INT 時，SD5~SD0, TD5~TD0 以及(Rx)5~(Rx)0 的設定值不會影響 TMR1 的動作。

TMR1 本身只有在執行 STE 指令設定 X1,0=01 時方可變成 12-bit timer，此時 T1X/R/TH 指令 SD/(Rx)/TD11~0 與 CT11~0 對應關係則變成與 TMR2,3 12-bit timer 相同。每個位元的設定值與其對應的功能選項更改如下所示：

OPCODE	Select clock(CS3~0)				Pre-set data of timer(CT11~0)											
T1XH SETDAT SD	SD15	SD14	SD13	SD12	SD11	SD10	SD9	SD8	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0
T1TH @HL	TD15	TD14	TD13	TD12	TD11	TD10	TD9	TD8	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0

OPCODE	Select clock(CS3~0)				Pre-set data of timer(CT11~0)											
T1RH	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
SETDAT RX	(RX)15	(RX)14	(RX)13	(RX)12	(RX)11	(RX)10	(RX)9	(RX)8	(RX)7	(RX)6	(RX)5	(RX)4	(RX)3	(RX)2	(RX)1	(RX)0
Rx address	Rx(RAM 位址 bit1,0=11)				Rx(RAM 位址 bit1,0=10)				Rx(RAM 位址 bit1,0=01)				Rx(RAM 位址 bit1,0=00)			

下表說明 6/12-bit TMR1 的 clock source 的設定方式：

Bit setting in instruction				
CS3	CS2	CS1	CS0	clock source of timer 1
0	0	0	0	~PH9
0	0	0	1	~PH3
0	0	1	0	~PH15
0	0	1	1	FREQ
0	1	0	0	~PH5
0	1	0	1	~PH7
0	1	1	0	~PH11
0	1	1	1	~PH13
1	0	0	0	CX
1	0	0	1	CX2
1	0	1	0	INT

在使用 T1XH 指令時會以 SETDAT 所設定的 immediate data(SD 直接設定 TMR1 的 clock source 以及預設值。

在使用 T1RH 指令時， SETDAT 所設定的 Rx 位址的最低兩個位元會被忽略掉，並且以 Rx (RAM 位址 bit1,0=00), Rx (RAM 位址 bit1,0=01), Rx (RAM 位址 bit1,0=10), Rx (RAM 位址 bit1,0=11)的連續位址去 data RAM 同時讀取 16-bit 的資料來設定 TMR1 的 clock source 以及預設值。

在使用 T1TH 指令時會將@HL 的 LSB 忽略掉，並且以@HL(Table ROM 位址 bit0=0), @HL(Table ROM 位址 bit0=1)的連續位址去 table ROM 同時讀取 16-bit 的資料來設定 TMR1 的 clock source 以及預設值。

註記：

(1). T1TH# 的語法會在指令結束後自動將 HL 的內容值加 1。
(2). SD = 0 ~ FFFFh
(3). Rx 在語法上必須是以絕對位址來描述。
(4). MCU 會在八個 machine cycle 的指令週期期間暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
T1XH SETDAT	SD	Preset initial data ← (SD11 ~ SD6/0) if STE X1,0≠/=1 Selection of clock source ← (SD15 ~ SD12)
T1TH	@HL	Preset initial data ← T(@HL)11 ~ T(@HL)6/0 if STE X1,0≠/=1 Selection of clock source ← T(@HL)15 ~ T(@HL)12
T1TH#	@HL	Preset initial data ← T(@HL)11 ~ T(@HL)6/0 if STE X1,0≠/=1 Selection of clock source ← T(@HL)15 ~ T(@HL)12 HL ← HL + 2
T1RH		Preset initial data ← ((Rx)11 ~ (Rx)6/0) if STE X1,0≠/=1

SETDAT	Rx	Selection of clock source ← ((Rx)15 ~ (Rx)12)

TM2, TM2X

指令特性：

單一字元長度指令，四個 machine cycle，4/8 bits data transferred。

指令說明：

指令運算元中所設定的 immediate data(X)，@HL 所指向的 table ROM 內容值，AC 以及 Rx 所指向的 data RAM 內容值可以用來設定 6-bit TMR2 的 clock source 以及 timer 的預設值。指令結束之後 TMR2 會開始動作。

這個指令只能用來設定 6-bit 的 TMR2。

每個位元的設定值與其對應的功能選項如下所示：

OPCODE	Select clock(CS3~0)				Pre-set data of timer(CT5~0)					
	CS3	CS2	CS1	CS0	CT5	CT4	CT3	CT2	CT1	CT0
TM2X X	0	X8	X7	X6	X5	X4	X3	X2	X1	X0
TM2 Rx	0	0	AC3	AC2	AC1	AC0	(Rx)3	(Rx)2	(Rx)1	(Rx)0
TM2 @HL	0	0	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0

下表說明 6-bit TMR2 的 clock source 的設定方式：

Bit setting in instruction				clock source of timer 2
CS3	CS2	CS1	CS0	
0	0	0	0	~PH9
0	0	0	1	~PH3
0	0	1	0	~PH15
0	0	1	1	FREQ
0	1	0	0	~PH5
0	1	0	1	~PH7
0	1	1	0	~PH11
0	1	1	1	~PH13

在使用 TM2X X 指令時會以 immediate data(X)直接設定 TMR2 的 clock source 以及預設值。

在使用 TM2 Rx 指令時會以 Rx 所指向的 data RAM 內容值以及 AC 的內容值來設定 TMR2 的 clock source 以及預設值。

在使用 TM2 @HL 指令時會以@HL 所指向的 table ROM 內容值來設定 TMR2 的 clock source 以及預設值。

註記：

(1). TM2# @HL 的語法會在指令結束後自動將 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述
(3). X = 0 ~ 1FFh

指令語法：

OP code	運算元	指令動作
TM2	Rx	Preset initial data ← (AC1, 0), ((Rx)3 ~ (Rx)0) Selection of clock source ← (AC3, 2)
TM2	@HL	Preset initial data ← T(@HL)5 ~ T(@HL)0 Selection of clock source ← T(@HL)7 ~ T(@HL)6
TM2#	@HL	Preset initial data ← T(@HL)5 ~ T(@HL)0 Selection of clock source ← T(@HL)7 ~ T(@HL)6 HL ← HL + 1
TM2X	X	Preset initial data ← (X5 ~ X0) Selection of clock source ← (X8 ~ X6)

T2XH, T2RH, T2TH

指令特性：

T2TH：單一字元長度指令，四個 machine cycle，16-bit data transferred。
T2RH：兩個字元長度指令，八個 machine cycle，16-bit data transferred。
T2XH：兩個字元長度指令，八個 machine cycle。

指令說明：

指令中 SETDAT 所設定的 immediate data (SD)，@HL 所指向的 table ROM 內容值，Rx 所指向的 data RAM 內容值可以用來設定 12-bit 或是 6-bit TMR2 的 clock source 以及 timer 的預設值。指令結束之後 TMR2 會開始動作。

每個位元的設定值與其對應的功能選項如下所示：

OPCODE	Select clock(CS3~0)				Pre-set data of timer(CT11~0)											
T2XH SETDAT SD	SD15	SD14	SD13	SD12	SD11	SD10	SD9	SD8	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0
T2TH @HL	TD15	TD14	TD13	TD12	TD11	TD10	TD9	TD8	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0
OPCODE	Select clock(CS3~0)				Pre-set data of timer(CT11~0)											
T2RH	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
SETDAT RX	(RX)15	(RX)14	(RX)13	(RX)12	(RX)11	(RX)10	(RX)9	(RX)8	(RX)7	(RX)6	(RX)5	(RX)4	(RX)3	(RX)2	(RX)1	(RX)0
Rx address	Rx(RAM 位址 bit1,0=11)				Rx(RAM 位址 bit1,0=10)				Rx(RAM 位址 bit1,0=01)				Rx(RAM 位址 bit1,0=00)			

當這些指令用來設定 6-bit 模式的 TMR2 為了可以選擇 Ctm2=CX/CX2/INT/XCLK 時，SD11~SD6, TD11~TD6 以及(Rx)11~(Rx)6 的設定值不會影響 TMR2 的動作。

下表說明 12-bit 或是 6-bit TMR2 的 clock source 的設定方式：

Bit setting in instruction				
CS3	CS2	CS1	CS0	clock source of timer 2
0	0	0	0	~PH9
0	0	0	1	~PH3
0	0	1	0	~PH15
0	0	1	1	FREQ
0	1	0	0	~PH5

0	1	0	1	~PH7
0	1	1	0	~PH11
0	1	1	1	~PH13
1	0	0	0	CX
1	0	0	1	CX2
1	0	1	0	INT
1	0	1	1	~XCLK

在使用 T2XH 指令時，SETDAT 所設定的 immediate data(SD)可以直接設定 TMR2 的 clock source 以及預設值。

在使用 T2RH 指令時， SETDAT 所設定的 Rx 位址的最低兩個位元會被忽略掉，並且以 Rx (RAM 位址 bit1,0=00), Rx (RAM 位址 bit1,0=01), Rx (RAM 位址 bit1,0=10), Rx (RAM 位址 bit1,0=11)的連續位址去 data RAM 同時讀取 16-bit 的資料來設定 TMR2 的 clock source 以及預設值。

在使用 T2TH 指令時會將@HL 的 LSB 忽略掉，並且以@HL(Table ROM 位址 bit0=0), @HL(Table ROM 位址 bit0=1)的連續位址去 table ROM 讀取 16-bit 的資料來設定 TMR2 的 clock source 以及預設值。

註記：
(1). T2TH# 的語法會在指令結束後自動將 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). SD = 0 ~ FFFFh。
(4). MCU 會在八個 machine cycle 的指令週期期間暫停所有的中斷請求。
(5). Ctm2=XCLK 目前只有 TM87ML28 支援。

指令語法：

OP code	運算元	指令動作
T2XH SETDAT	SD	Preset initial data ← (SD11 ~ SD0) Selection of clock source ← (SD15 ~ SD12)
T2TH	@HL	Preset initial data ← T(@HL)11 ~ T(@HL)0 Selection of clock source ← T(@HL)15 ~ T(@HL)12
T2TH#	@HL	Preset initial data ← T(@HL)11 ~ T(@HL)0 Selection of clock source ← T(@HL)15 ~ T(@HL)12 HL ← HL + 2
T2RH SETDAT	Rx	Preset initial data ← ((Rx)11 ~ (Rx)0) Selection of clock source ← ((Rx)15 ~ (Rx)12)

TM3, TM3X

指令特性：
單一字元長度指令，四個 machine cycle，4/8 bits data transferred。

指令說明：

指令運算元中所設定的 immediate data(X)，@HL 所指向的 table ROM 內容值，AC 以及 Rx 所指向的 data RAM 內容值可以用來設定 6-bit TMR3 的 clock source 以及 timer 的預設值。指令結束之後 TMR3 會開始動作。

這個指令只能用來設定 6-bit 的 TMR3。

每個位元的設定值與其對應的功能選項如下所示：

	Select clock(CS3~0)				Pre-set data of timer(CT5~0)					
OPCODE	CS3	CS2	CS1	CS0	CT5	CT4	CT3	CT2	CT1	CT0
TM3X X	0	X8	X7	X6	X5	X4	X3	X2	X1	X0
TM3 Rx	0	0	AC3	AC2	AC1	AC0	(Rx)3	(Rx)2	(Rx)1	(Rx)0
TM3 @HL	0	0	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0

下表說明 6-bit TMR3 的 clock source 的設定方式。

Bit setting in instruction				clock source of timer 3
CS3	CS2	CS1	CS0	
0	0	0	0	~PH9
0	0	0	1	~PH3
0	0	1	0	~PH15
0	0	1	1	FREQ
0	1	0	0	~PH5
0	1	0	1	~PH7
0	1	1	0	~PH11
0	1	1	1	~PH13

在使用 TM3X X 指令時， immediate data(X)會直接設定 TMR3 的 clock source 以及預設值。

在使用 TM3 Rx 指令時會以 Rx 所指向的 data RAM 內容值以及 AC 的內容值來設定 TMR3 的 clock source 以及預設值。

在使用 TM3 @HL 指令時會以@HL 所指向的 table ROM 內容值來設定 TMR3 的 clock source 以及預設值。

註記：

(1). TM3# @HL 的語法會在指令結束後自動將 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). X = 0 ~ 1FFh

指令語法：

OP code	運算元	指令動作
TM3	Rx	Preset initial data ← (AC1, 0), ((Rx)3 ~ (Rx)0) Selection of clock source ← (AC3, 2)
TM3	@HL	Preset initial data ← T(@HL)5 ~ T(@HL)0 Selection of clock source ← T(@HL)7 ~ T(@HL)6
TM3#	@HL	Preset initial data ← T(@HL)5 ~ T(@HL)0 Selection of clock source ← T(@HL)7 ~ T(@HL)6 HL ← HL + 1
TM3X	X	Preset initial data ← (X5 ~ X0) Selection of clock source ← (X8 ~ X6)

T3XH, T3RH, T3TH

指令特性：

T3TH：單一字元長度指令，四個 machine cycle，16-bit data transferred。
T3RH：兩個字元長度指令，八個 machine cycle，16-bit data transferred。
T3XH：兩個字元長度指令，八個 machine cycle。

指令說明：

指令中 SETDAT 所設定的 immediate data(SD)，@HL 所指向的 table ROM 內容值或是 Rx 所指向的 data RAM 內容值可以用來設定 12-bit 或是 6-bit TMR3 的 clock source 以及 timer 的起始值。指令結束之後 TMR3 會開始動作。

每個位元的設定值與其對應的功能選項如下所示：

OPCODE	Select clock(CS3~0)				Pre-set data of timer(CT11~0)											
T3XH SETDATSD	SD15	SD14	SD13	SD12	SD11	SD10	SD9	SD8	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0
T3TH @HL	TD15	TD14	TD13	TD12	TD11	TD10	TD9	TD8	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0
OPCODE	Select clock(CS3~0)				Pre-set data of timer(CT11~0)											
T3RH SETDATRX	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
	(RX)15	(RX)14	(RX)13	(RX)12	(RX)11	(RX)10	(RX)9	(RX)8	(RX)7	(RX)6	(RX)5	(RX)4	(RX)3	(RX)2	(RX)1	(RX)0
Rx address	Rx(RAM 位址 bit1,0=11)				Rx(RAM 位址 bit1,0=10)				Rx(RAM 位址 bit1,0=01)				Rx(RAM 位址 bit1,0=00)			

當這些指令用來設定 6-bit 模式的 TMR3 為了可以選擇 Ctm3=CX/CX2/INT/XCLK 時，SD11~SD6, TD11~TD6 以及(Rx)11~(Rx)6 的設定值不會影響 TMR3 的動作。

下表說明 12-bit 或是 6-bit TMR3 的 clock source 的設定方式：

Bit setting in instruction				clock source of timer 3
CS3	CS2	CS1	CS0	
0	0	0	0	~PH9
0	0	0	1	~PH3
0	0	1	0	~PH15
0	0	1	1	FREQ
0	1	0	0	~PH5
0	1	0	1	~PH7
0	1	1	0	~PH11
0	1	1	1	~PH13
1	0	0	0	CX
1	0	0	1	CX2
1	0	1	0	INT

1	0	1	1	~XCLK

在使用 T3XH 指令時，SETDAT 所設定的 immediate data (SD) 可以直接設定 TMR3 的 clock source 以及起始值。

在使用 T3RH 指令時， SETDAT 所設定的 Rx 位址的最低兩個位元會被忽略掉，並且以 Rx (RAM 位址 bit1,0=00), Rx (RAM 位址 bit1,0=01), Rx (RAM 位址 bit1,0=10), Rx (RAM 位址 bit1,0=11)的連續位址去 data RAM 同時讀取 16-bit 的資料來設定 TMR3 的 clock source 以及起始值。

在使用 T3TH 指令時會將@HL 的 LSB 忽略掉，並且以@HL(Table ROM 位址 bit0=0), @HL(Table ROM 位址 bit0=1)的連續位址去 table ROM 同時讀取 16-bit 的資料來設定 TMR3 的 clock source 以及起始值。

註記：
(1). T3TH# 的語法會在指令結束後自動將 HL 的內容值加 1。
(2). SD = 0 ~ FFFFh
(3). Rx 在語法上必須是以絕對位址來描述。
(4). MCU 會在八個 machine cycle 的指令週期期間暫停所有的中斷請求。
(5). Ctm3=XCLK 目前只有 TM87ML28 支援。

指令語法：

OP code	運算元	指令動作
T3XH SETDAT	SD	Preset initial data ← (SD11 ~ SD0) Selection of clock source ← (SD15 ~ SD12)
T3TH	@HL	Preset initial data ← T(@HL)11 ~ T(@HL)0 Selection of clock source ← T(@HL)15 ~ T(@HL)12
T3TH#	@HL	Preset initial data ← T(@HL)11 ~ T(@HL)0 Selection of clock source ← T(@HL)15 ~ T(@HL)12 HL ← HL + 2
T3RH SETDAT	Rx	Preset initial data ← ((Rx)11 ~ (Rx)0) Selection of clock source ← ((Rx)15 ~ (Rx)12)

RTM2L, RTM21, RTM1H, RTM3L, RTM31

指令特性：
單一字元長度指令，四個 machine cycle，4-bit data transferred。

指令說明：
直接從 TMR1~TMR3 或是從 18-bit 的特定暫存器中讀取目前的內容值並儲存到 Rx 指定的 data RAM 位址以及 AC。

下表說明各個指令能夠讀取不同 timer 內容值的對應表：

6-bit TMR1/2/3 by STM X1,0=00	Content of 6-bit TMR1						Content of 6-bit TMR3						Content of 6-bit TMR2					
	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

12/6-bit TMR2/3 by STM X1,0=01	Content of 12-bit TMR2						Content of 6-bit TMR3						Content of 12-bit TMR2					
	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6/12-bit TMR2/3 by STM X1,0=10	Content of 12-bit TMR3						Content of 12-bit TMR3						Content of 6-bit TMR2					
	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
18-bit TMR2 by STM X1,0=11	Content of 18-bit TMR2						Content of 18-bit TMR2						Content of 18-bit TMR2					
	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit17	Bit16	Bit15	Bit14	Bit13	Bit12	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RTM2L															(Rx)3	(Rx)2	(Rx)1	(Rx)0
RTM21					(Rx)3	(Rx)2							(Rx)1	(Rx)0				
RTM1H	(Rx)3	(Rx)2	(Rx)1	(Rx)0														
RTM3L									(Rx)3	(Rx)2	(Rx)1	(Rx)0						
RTM31					(Rx)3	(Rx)2	(Rx)1	(Rx)0										

因為在 TM87ML28 可以提供 STE 指令可選擇 TMR1/2/3 另外獨立延展為 12-bit timer，故提供由 STE 指令 設定 X2=1 可切換將 RTM1H/RTM31/RTM21 指令中原本讀取 TMR1 bit5~0 內容值變成讀取此延展 12-bit timer 中 CT11~6 的內容值。而當設定 STE 指令的 X1,0=01 使 TMR1 延展成 12-bit timer 時，當由 STE 指令 設定 X2=1 時則會將 RTM2L/RTM21 指令中原本讀取 TMR2 bit5~0 的內容值切換成 TMR1 延展 12-bit timer 中 CT5~0 的內容值。
下表列出所有正確 STM&STE 設定組合&說明當執行 STE 指令設定 X2=0/1 各個指令能夠讀取不同 timer 內容值的對應表：

6-bit TMR1,2,3 by STM/STE X1,0=0/0		Content of No-Use 6-bit Counter						Content of 6-bit TMR1						Content of 6-bit TMR3						Content of 6-bit TMR2					
		C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0
12/6-bit TMR1,2,3 by STM/STE X1,0=0/1		Content of 12-bit TMR1						Content of 12-bit TMR1						Content of 6-bit TMR3						Content of 6-bit TMR2					
		C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0
12/6bit-TMR2/1,3 by STM/STE X1,0=0/2		Content of 12-bit TMR2						Content of 6-bit TMR1						Content of 6-bit TMR3						Content of 12-bit TMR2					
		C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0
12/6bit-TMR3/1,2 by STM/STE X1,0=0/3		Content of 12-bit TMR3						Content of 6-bit TMR1						Content of 12-bit TMR3						Content of 6-bit TMR2					
		C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0
12/6bit-TMR2/3 by STM/STE X1,0=1/0		Content of No-Use 6-bit Counter						Content of 12-bit TMR2						Content of 6-bit TMR3						Content of 12-bit TMR2					
		C5	C4	C3	C2	C1	C0	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0
12/12bit-TMR2/3 by STM/STE X1,0=1/3		Content of 12-bit TMR3						Content of 12-bit TMR2						Content of 12-bit TMR3						Content of 12-bit TMR2					
		C11	C10	C9	C8	C7	C6	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0
12/6bit-TMR3/2 by STM/STE X1,0=2/0		Content of No-Use 6-bit Counter						Content of 12-bit TMR3						Content of 12-bit TMR3						Content of 6-bit TMR2					
		C5	C4	C3	C2	C1	C0	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0
12/12bit-TMR3/2 by STM/STE X1,0=2/2		Content of 12-bit TMR2						Content of 12-bit TMR3						Content of 12-bit TMR3						Content of 12-bit TMR2					
		C11	C10	C9	C8	C7	C6	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0	C5	C4	C3	C2	C1	C0
18/-bit-TMR2/- by STM/STE X1,0=3/0		Content of No-Use 6-bit Counter						Content of 18-bit TMR2						Content of 18-bit TMR2						Content of 18-bit TMR2					
		C5	C4	C3	C2	C1	C0	C11	C10	C9	C8	C7	C6	C17	C16	C15	C14	C13	C12	C5	C4	C3	C2	C1	C0
RTM2L (STE X1,0 !=1)	X2=0																					B3	B2	B1	B0
	X2=1																					B3	B2	B1	B0
RTM21 (STE X1,0 !=1)	X2=0											B3	B2							B1	B0				
	X2=1					B3	B2													B1	B0				
RTM2L (STE X1,0 =1)	X2=0																					B3	B2	B1	B0
	X2=1									B3	B2	B1	B0												
RTM21 (STE X1,0 =1)	X2=0											B3	B2							B1	B0				
	X2=1					B3	B2	B1	B0																
RTM1H	X2=0							B3	B2	B1	B0														
	X2=1	B3	B2	B1	B0																				
RTM3L	-															B3	B2	B1	B0						
RTM31	X2=0											B3	B2	B1	B0										
	X2=1					B3	B2							B1	B0										

C17/11/5~0：18/12/6-bit Timer Count bit17/11/5~0.
B3~0：(Rx) bit3~0.

當 STM & STE 設定組合造成衝突時系統以 STM 設定組合優先而使 STE 設定組合失效，故當 STM X1,0=1/2/3 時若設定 STE X1,0=1~2/1,3/1~3 皆會被強迫等同 STE X1,0=0，但執行 STE 指令設定 X2=1 時切換執行 RTM21/31 指令傳送到 AC,(Rx)的 bit3,2 & RTM1H 指令傳送到 AC,(Rx)的 bit3~0 變成讀取延展 6-bit counter 的內容值動作則不受影響。

註記：

(1). Rx 在語法上必須是以絕對位址來描述。

(2). 若 MCU 有提供藉由 Rm 設定 HMRTM=1(目前在 EV chip (TM8999)不支援)則可以切換 RTM2L/RTM3L 指令變成同時執行"RTM3L$_{3\sim0}$, RTM1H$_{3\sim0}$, RTM21$_{3\sim0}$, RTM2L$_{3\sim0}$" / "RTM2L$_{3\sim0}$, RTM1H$_{3\sim0}$, RTM31$_{3\sim0}$, RTM3L$_{3\sim0}$"指令的 16 bits 讀取動作&將依 STM&STE 設定組合的 timer counter 的內容值儲存到 Rx 所指定的 16bits mode data RAM((Rx)H$_{15\sim0}$)以及 16bits mode AC(ACH$_{15\sim0}$) 。

指令語法：

OP code	運算元	指令動作
RTM2L	Rx	(Rx), AC ← TMR2(bit3 ~ bit0) if STE X2=0 | X1,0≠1 (Rx), AC ← TMR1(bit3 ~ bit0) if STE X2=1 & X1,0=1 & STM X1,0=0
RTM21	Rx	(Rx), AC ← TMR1(bit1, bit0), TMR2(bit5, bit4) if STE X2=0 (Rx), AC ← TMR2/3(bit7, bit6), TMR2(bit5, bit4) if STE X2=1 & X1,0=2/3 & STM X1,0=0,2/1 (Rx), AC ← TMR1(bit7~4) if STE X2=1 & X1,0=1 & STM X1,0=0
RTM1H	Rx	(Rx), AC ← TMR1(bit5 ~ bit2) if STE X2=0 & X1,0≠1 (Rx), AC ← TMR2/3(bit11~8) if STE X2=1 & X1,0=2/3 & STM X1,0=0,2/1 (Rx), AC ← TMR1(bit11~8) if STE X2=1 & X1,0=1 & STM X1,0=0
RTM3L	Rx	(Rx), AC ← TMR3(bit3 ~ bit0)
RTM31	Rx	(Rx), AC ← TMR1(bit1, bit0), TMR3(bit5, bit4) if STE X2=0 (Rx), AC ← TMR2/3(bit7,bit6), TMR3(bit5, bit4) if STE X2=1 & X1,0=2/3 & STM X1,0=0,2/1 (Rx), AC ← TMR1(bit7,bit6), TMR3(bit5, bit4) if STE X2=1 & X1,0=1 & STM X1,0=0

若 STE X2=1 & X1,0=0 則因延展的 6-bit counter 未連接使用，故 RTM21&RTM31 請忽略(Rx) & AC bit3,2 & RTM1H 請忽略(Rx) & AC bit3~0。

T2M3X

指令特性：

兩個字元長度指令，八個 machine cycle。

指令說明：

指令運算元 X 中的內容值以及 immediate data(SD)可以設定 18-bit TMR2 的 clock source 以及 timer 的起始值。

指令結束後 TMR2 會開始動作。

每個位元的設定值與其對應的功能選項如下所示：

OPCODE	Select clock source(CS3~0)				設定 timer 的起始值(CT17~0)					
T2M3X X SETDAT SD	SD15	SD14	SD13	SD12	CT17	CT16	CT15	CT14	CT13	CT12
					X5	X4	X3	X2	X1	X0
					CT11	CT10	CT19	CT18	CT17	CT16

					SD11	SD10	SD9	SD8	SD7	SD6
					CT5	CT4	CT3	CT2	CT1	CT0
					SD5	SD4	SD3	SD2	SD1	SD0

下表說明 18-bit TMR2 的 clock source 的設定方式：

Bit setting in instruction				clock source of TMR2
CS3	CS2	CS1	CS0	
0	0	0	0	~PH9
0	0	0	1	~PH3
0	0	1	0	~PH15
0	0	1	1	FREQ
0	1	0	0	~PH5
0	1	0	1	~PH7
0	1	1	0	~PH11
0	1	1	1	~PH13
1	0	0	0	CX
1	0	0	1	CX2
1	0	1	0	INT
1	0	1	1	~XCLK

註記：

(1). 這個指令只能用來設定 18-bit 模式的 TMR2。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). X = 0 ~ 3Fh
(4). SD = 0 ~ FFFFh
(5). MCU 會在指令週期中的八個 machine cycle 期間暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
T2M3X SETDAT	X SD	TMR2 的起始值 ← (X5 ~ X0), (SD11 ~ SD0) 選擇 clock source ← (SD15 ~ SD12) TMR2 開始動作

STM

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元 X 中的內容值來設定不同 TMR 之間的串接組合方式。

X1, X0 = 00, 將 TMR1 和 TMR2 和 TMR3 設定成 3 個獨立的 6-bit timer。

這是 MCU 的原始設定。

X1, X0 = 01, 將 TMR1 與 TMR2 串接使用，TMR2 變成 12-bit timer (TMR1 併入 TMR2 的 bit11~bit6 部份使用)，而 TMR3 仍是獨立的 6-bit timer。

X1, X0 = 10, 將 TMR1 與 TMR3 串接使用，TMR3 變成 12-bit timer (TMR1 併入,TMR3 的 bit11~bit6 部份使用)，而 TMR2 仍是獨立的 6-bit timer。

X1, X0 = 11, 將 TMR1 與 TMR3 與 TMR2 串接使用，TMR2 變成 18-bit timer。(TMR1 併入 TMR2 的 bit11~bit6 部分使用，TMR3 則併入 TMR2 的 bit17~bit12 部份使用)。

註記：

(1). 執行 STM 指令後會停止所有 Timer 的動作 (包含關閉 re-load 功能)，故執行 STM 指令前請先確認 TMR1、2、3 皆在無動作狀態。

(2). X1 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
STM	X	TMR1, 2, 3 are 6-bit ← X = 0 (default) TMR2=TMR1+TMR2 ← X = 1 TMR2 (12-bit), TMR3 (6-bit) TMR3=TMR1+TMR3 ← X = 2 TMR3 (12-bit), TMR2 (6-bit) TMR2=TMR3+TMR1+TMR2 ← X = 3 TMR2 (18-bit)

DISTM

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元 X 中內容值的設定來停止 TMR1 ~TMR3 的動作。

X2 = 1, 停止 TMR3 的動作(包含關閉 re-load 功能)

X1 = 1, 停止 TMR2 的動作(包含關閉 re-load 功能)

X0 = 1, 停止 TMR1 的動作(包含關閉 re-load 功能)

註記：

X2 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
DISTM	X	停止 TMR3 ← X2 = 1 停止 TMR2 ← X1 = 1 停止 TMR1 ← X0 = 1

STE (目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元 X 中的內容值來設定是否選擇 TMR1~3 其中一個獨立延展成 12-bit timer。

X1, X0 = 00, TMR1 和 TMR2 和 TMR3 皆不會被選擇延展成 12-bit timer。

這是 MCU 的原始設定。

X1, X0 = 01, 將 TMR1 延展成 12-bit timer，但必須 STM X1,0=00 否則視同 X1,X0=00。

X1, X0 = 10, 將 TMR2 延展成 12-bit timer，但必須 STM X1,0=X0 否則視同 X1,X0=00。

X1, X0 = 11, 將 TMR3 延展成 12-bit timer，但必須 STM X1,0=0X 否則視同 X1,X0=00。

X2 = 0, 正常 RTM2L/RTM21/RTM1H/RTM31 讀取模式。

X2 = 1, 切換 RTM21 & RTM31 所讀取 bit3,2 變成由 STE 所延展 12-bit timer 的 bit7,6 &切換 RTM1H 所讀取 bit3~0 變成由 STE 所延展 12-bit timer 的 bit11~8。若 STE X1,0=01 則切換 RTM2L/RTM21 所讀取 bit3~0/bit3~2 變成由 STE 所延展 12-bit TMR1 的 bit3~0/bit5~4。

註記：

(1). 執行 STE 指令後前請先確認所選擇 TMR1、2、3 在無動作狀態以避免產生錯亂風險。
(2). X2 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
STE	X	正常 RTM2L/21/31/1H 讀取模式 ← X2=0 (default) 切換 RTM2L/21/31/1H 讀取為 STE 模式 ← X2=1 TMR1, 2, 3 are 6-bit ← X1,0 = 0 (default) TMR1 延展為 12bit ← X1,0 = 1 & STM X1,0=00 TMR2 延展為 12bit ← X1,0 = 2 & STM X1,0=X0 TMR3 延展為 12bit ← X1,0 = 3 & STM X1,0=0X

SRP(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元 X 中的內容值來設定將 TMR1~3 re-load 功能切換成每次 underflow 時重新載入所設定起始值。X=0 是 MCU 的原始設定。

X2 = 0, TMR3 當設定 Reload3=1 時每次 underflow 時會依所設定 6/12-bit timer 模式從 3F/FFFH 開始重新倒數。

X2= 1, TMR3 當設定 Reload3=1 時每次 underflow 時會從所設定起始值開始重新倒數。

X1 = 0, TMR2 當設定 Reload2=1 時每次 underflow 時會依所設定 6/12/18-bit timer 模式從 3F/FFF/3FFFFH 開始重新倒數。

X1 = 1, TMR2 當設定 Reload2=1 時每次 underflow 時會從所設定起始值開始重新倒數。

X0 = 0, TMR1 當設定 Reload1=1 時每次 underflow 時會依所設定 6/12-bit timer 模式從 3F/FFFH 開始重新倒數。

X0 = 1, TMR1 當設定 Reload1=1 時每次 underflow 時會從所設定起始值開始重新倒數。

註記：
X2 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SRP	X	切換 TMR3 Reload 為重覆載入設定值模式 ← X2=1 切換 TMR2 Reload 為重覆載入設定值模式 ← X1=1 切換 TMR1 Reload 為重覆載入設定值模式 ← X0=1

ST3OV(目前在 EV chip (TM8999)不支援)

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元 X 中的內容值來設定在 TMR3 每次 underflow 時 mask FREQ/TMR2 一個 Cfq/Ctm2 週期。X=0 是 MCU 的原始設定。

X1 = 1, 啟動 TMR3 在每次 underflow 時 mask FREQ 一個 Cfq 時鐘週期。

X0 = 1, 啟動 TMR3 在每次 underflow 時 mask TMR2 一個 Ctm2 時鐘週期。

註記：
X1 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
ST3OV	X	Enable TMR3 underflow mask one Cfq cycle ← X1=1 Enable TMR3 underflow mask one Ctm2 cycle ← X0=1

2-12. 週邊功能指令

SPKTH, SPKXH, SPKRH

指令特性：

SPKTH：單一字元長度指令，四個 machine cycle，16-bit data transferred。
SPKRH：兩個字元長度指令，八個 machine cycle，16-bit data transferred。
SPKXH：兩個字元長度指令，八個 machine cycle。

指令說明：

指令運算元中的內容值可用來設定矩陣式鍵盤掃描功能中各個輸出腳位 (KO) 是否輸出掃描信號以及產生 HALT release request 的方式。

D = 0, 矩陣式鍵盤掃描功能一旦偵測到 KI1~4 的輸入腳位上有任何掃描信號出現時就會將 halt release request flag 5 (HRF5)設定為 1。

= 1, 矩陣式鍵盤掃描功能會在每次掃描週期結束時將 halt release request flag 5 (HRF5) 設定為 1。

下表說明不同的指令中其運算元的各個位元與 KO 腳位之間的對應關係如下：

Instruction	KO16	KO15	KO14	KO13	KO12	KO11	KO10	KO9	KO8	KO7	KO6	KO5	KO4	KO3	KO2	KO1
SPKXH D SETDAT SD	SD15	SD14	SD13	SD12	SD11	SD10	SD9	SD8	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0
SPKTH(#) D, @HL	TD15	TD14	TD13	TD12	TD11	TD10	TD9	TD8	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0
SPKRH D SETDAT Rx	(RX)15	(RX)14	(RX)13	(RX)12	(RX)11	(RX)10	(RX)9	(RX)8	(RX)7	(RX)6	(RX)5	(RX)4	(RX)3	(RX)2	(RX)1	(RX)0
Rx address	Rx(RAM 位址 bit1,0=11)				Rx(RAM 位址 bit1,0=10)				Rx(RAM 位址 bit1,0=01)				Rx(RAM 位址 bit1,0=00)			

在使用 SPKXH D 指令時，SETDAT 所設定的 immediate data (SD)就是代表相對應的 KO 腳位可以輸出掃描信號。

在使用 SPKTH D 指令時會將@HL 的 LSB 忽略掉，並且以@HL(table ROM 位址 bit0=0), @HL(table ROM 位址 bit0=1) 的連續位址去 table ROM 讀取 16-bit 的資料，並以此資料來設定相對應的 KO 腳位可以輸出掃描信號。

在使用 SPKRH D 指令時，SETDAT 所設定的 Rx 的最低兩個位元會被忽略掉，並且以 Rx (RAM 位址 bit1,0=00), Rx (RAM 位址 bit1,0=01), Rx (RAM 位址 bit1,0=10), Rx (RAM 位址 bit1,0=11)的連續位址去 data RAM 讀取 16-bit 的資料，並以此資料來設定相對應的 KO 腳位可以輸出掃描信號。

註記：

(1). SPKTH# D 的語法會在指令結束後自動將 HL 的內容值加 2。
(2). D = 0 or 1
(3). SD = 0 ~ FFFFh
(4). Rx 在語法上必須是以絕對位址來描述。
(5). MCU 會在八個 machine cycle 的指令週期期間暫停所有的中斷請求。

指令語法：

OP code	運算元	指令動作
SPKXH SETDAT	D SD	選擇產生 halt released 的方式← D 設定 KO 腳位的掃描信號 ← SD15 ~ SD0
SPKTH	D, @HL	選擇產生 halt released 的方式← D 設定 KO 腳位的掃描信號 ← T(@HL)15 ~ T(@HL)0
SPKTH#	D, @HL	選擇產生 halt released 的方式← D 設定 KO 腳位的掃描信號 ← T(@HL)15 ~ T(@HL)0 HL ← HL + 2
SPKRH SETDAT	D Rx	選擇產生 halt released 的方式← D 設定 KO 腳位的掃描信號 ← (Rx)15 ~ (Rx)0

SPK, SPKX

指令特性：

單一字元長度指令，四個 machine cycle，4/8 bits data transferred。

指令說明：

指令運算元中的內容值可用來設定矩陣式鍵盤掃描功能的不同自動掃描方式以及產生 HALT release request 的方式。

下表說明不同的 SPK 指令中其運算元各個位元之間的對應關係如下：

運算元的位元定義	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPKX X	X7	X6	X5	X4	X3	X2	X1	X0
SPK Rx	AC3	AC2	AC1	AC0	(Rx)3	(Rx)2	(Rx)1	(Rx)0
SPK(#) @HL	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0

X6 = 0, 矩陣式鍵盤掃描功能一旦偵測到 KI1~4 的輸入腳位上有任何掃描信號出現時就會將 halt release request flag 5 (HRF5)設定為 1。

= 1, 矩陣式鍵盤掃描功能會在與 LCD 波形同步的掃描信號出現時將 halt release request flag 5 (HRF5)設定為 1。

X7, X5, X4 = 000, 設定每次掃描週期中只有一個 KO 輸出腳位會送出掃描信號，而輸出腳位的選擇是由 X3 ~ X0 來設定。

X3 ~ X0 = 0000, 選擇 KO1 送出掃描信號
X3 ~ X0 = 0001, 選擇 KO2 送出掃描信號
……
X3 ~ X0 = 1110, 選擇 KO15 送出掃描信號
X3 ~ X0 = 1111, 選擇 KO16 送出掃描信號

X7, X5, X4 = 001, 設定每次掃描週期中所有的 KO 輸出腳位(KO1 ~ KO16)都會同時送出掃描信號，此時 X3 ~ X0 可以是任意值。

X7, X5, X4 = 010, 關閉矩陣式鍵盤掃描功能，此時 X3 ~ X0 可以是任意值。

X7, X5, X4 = 10X, 設定每次掃描週期中只有 8 個 KO 輸出腳位同時送出掃描信號，而不同輸出腳位的選擇是由 X3 來設定。
X3 = 0, 選擇 KO1 ~ KO8 同時送出掃描信號
X3 = 1, 選擇 KO9 ~ KO16 同時送出掃描信號
此時 X4 ,X2~ X0 可以是任意值。

X7, X5, X4 = 110, 設定每次掃描週期中只有 4 個 KO 輸出腳位同時送出掃描信號，而不同輸出腳位的選擇是由 X3 及 X2 來設定。
X3X2 = 00, 選擇 KO1 ~ KO4 同時送出掃描信號
X3X2 = 01, 選擇 KO5 ~ KO8 同時送出掃描信號
X3X2 = 10, 選擇 KO9 ~ KO12 同時送出掃描信號
X3X2 = 11, 選擇 KO13 ~ KO16 同時送出掃描信號
此時 X1, X0 可以是任意值。

X7, X5, X4 = 111, 設定每次掃描週期中只有 2 個 KO 輸出腳位同時送出掃描信號，而不同輸出腳位的選擇是由 X3, X2, X1 來設定。
X3X2X1 = 000, 選擇 KO1 ~ KO2 同時送出掃描信號
X3X2X1 = 001, 選擇 KO3 ~ KO4 同時送出掃描信號
……
X3X2X1 = 110, 選擇 KO13 ~ KO14 同時送出掃描信號
X3X2X1 = 111, 選擇 KO15 ~ KO16 同時送出掃描信號
此時 X0 可以是任意值。

註記：
(1). SPK# @HL 的語法會在指令結束後自動將 HL 的內容值加 1。
(2). Rx 在語法上必須是以絕對位址來描述。
(3). X7 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SPK	Rx	選擇產生 halt released 的方式 ← AC2 選擇掃描的方式 ← AC3, AC1, AC0, (Rx)3 ~(Rx)0
SPK	@HL	選擇產生 halt released 的方式← T(@HL)6 選擇掃描的方式 ← T(@HL)7, T(@HL)5 ~ T(@HL)0
SPK#	@HL	選擇產生 halt released 的方式 ← T(@HL)6 選擇掃描的方式 ← T(@HL)7, T(@HL)5 ~ T(@HL)0 HL ← HL + 1
SPKX	X	選擇產生 halt released 的方式← X6 選擇掃描的方式 ← X7, X5 ~X0

ELC

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元中 X 值的設定來選擇 EL panel driver 的驅動方式。

設定 ELP 輸出腳位的輸出頻率與 duty cycle：

X8, 7, 6 = 111, 選擇 ELP 輸出波形是來自 BCLK/8.
= 110, 選擇 ELP 輸出波形是來自 BCLK/4.
= 101, 選擇 ELP 輸出波形是來自 BCLK/2.
= 100, 選擇 ELP 輸出波形是來自 BCLK.
= 011, 選擇 ELP 輸出波形是來自 frequency generator 的輸出信號(FREQ)的反相信號。
= 000, 選擇 ELP 輸出波形是來自 PH0.

X9, 5, 4 = 101, 選擇 ELP 輸出波形的 duty cycle 是 2/3 duty.
= 100, 選擇 ELP 輸出波形的 duty cycle 是 3/4 duty.
= x11, 選擇 ELP 輸出波形的 duty cycle 是 1/1 duty.
= x10, 選擇 ELP 輸出波形的 duty cycle 是 1/2 duty.
= 001, 選擇 ELP 輸出波形的 duty cycle 是 1/3 duty.
= 000, 選擇 ELP 輸出波形的 duty cycle 是 1/4 duty.

設定 ELC 輸出腳位的輸出頻率與 duty cycle：

X3, 2 = 11, 選擇 ELC 輸出波形是來自 PH5.
= 10, 選擇 ELC 輸出波形是來自 PH6.
= 01, 選擇 ELC 輸出波形是來自 PH7.
= 00, 選擇 ELC 輸出波形是來自 PH8.

X1, 0 = 11, 選擇 ELC 輸出波形的 duty cycle 是 1/1.
= 10, 選擇 ELC 輸出波形的 duty cycle 是 1/2.
= 01, 選擇 ELC 輸出波形的 duty cycle 是 1/3.
= 00, 選擇 ELC 輸出波形的 duty cycle 是 1/4.

註記：

(1). X9 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。
(2). 在設定 ELP 輸出波形的 duty cycle 為 1/1 或是 1/2 duty 時，X9 可以是任意值。

指令語法：

OP code	運算元	指令動作
ELC	X	設定 ELP 輸出腳位的頻率← X8, 7, 6 設定 ELP 輸出腳位的 duty cycle ← X9, 5, 4 設定 ELC 輸出腳位的頻率 ← X3, 2 設定 ELC 輸出腳位的 duty cycle← X1, 0

ALM

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中 X 的設定值來合成一個調變波形。調變波形中的 envelope 以及 carrier 的信號來源都可以 X 值來設定。

X8, 7, 6 = 111, 選擇 carrier 信號是來自 frequency generator 的輸出(FREQ)
= 100, 選擇 carrier 信號是直流高電位(DC1)
= 011, 選擇 carrier 信號是來自 PH3.
= 010, 選擇 carrier 信號是來自 PH4.
= 001, 選擇 carrier 信號是來自 PH5.
= 000, 選擇 carrier 信號是直流低電位(DC0)

X5 = 1, 選擇 envelope 信號中含有 PH15 的頻率成分。

X4 = 1, 選擇 envelope 信號中含有 PH14 的頻率成分。

X3 = 1, 選擇 envelope 信號中含有 PH13 的頻率成分。

X2 = 1, 選擇 envelope 信號中含有 PH12 的頻率成分。

X1 = 1, 選擇 envelope 信號中含有 PH11 的頻率成分。

X0 = 1, 選擇 envelope 信號中含有 PH10 的頻率成分。

註記：
(1). Envelope 的波形是由 X5~X0 所選擇的頻率信號經過 AND 邏輯處理後才產生。
(2). 調變波形在 envelope 信號是 GND 電位時才會輸出 carrier 信號。
(3). X8 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
ALM	X	設定調變波形中的 carrier 信號← X8, 7, 6 設定調變波形中的 envelope 信號← X5 ~ 0

SBZ

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中 X 值的設定來選擇 BZB 以及 BZ 兩個輸出腳位上的輸出波形是來自 frequency generator 的輸出信號或是經由 ALM 指令設定之後所產生的調變信號。

X1 = 0, 將 ALM 指令所設定的調變信號經過反相之後輸出到 BZB pin。
這是 MCU 的原始設定狀態。
= 1, 將 frequency generator 的輸出信號(FREQ)經過反相之後輸出到 BZB pin。

X0 = 0, 將 ALM 指令所設定的調變信號直接輸出到 BZ pin。

這是 MCU 的原始設定狀態。

= 1, 將 frequency generator 的輸出信號(FREQ)直接輸出到 BZ pin。

註記：

X1 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SBZ	X	選擇 BZB pin 的信號來源 ← X1 選擇 BZ pin 的信號來源 ← X0

2-13. 系統功能指令(System Function Instructions)

NOP

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
MCU 在執行此一指令後並不會執行任何動作，但是會佔用 4 個 machine cycle，可作為程式中的 delay 之用。

指令語法：

OP code	運算元	指令動作
NOP		No operation

HALT

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
MCU 進入 HALT mode。
下面 4 種情況發生時可以讓 MCU 離開 HALT mode：
(1). 當任何一個中斷請求發生時。
但是在程式處理完中斷服務副程式並執行 RTS 指令之後，MCU 會重新進入 HALT mode。

(2). 當 SCA 指令所設定的條件成立時 (當 IOA, IOC, IOD 這 3 個 IO port 上的輸入信號變化被 MCU 判定是有效時)。

(3). 當 SCX 指令所設定的條件成立時 (RFC counter 在 CX2/CX pin 的控制模式下, 當 CX2/CX pin 的控制信號結束之後)。

(4). 當 SHE 指令所設定的條件成立時。

指令語法：

OP code	運算元	指令動作
HALT		MCU 進入 HALT mode.

STOP

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：

MCU 進入 STOP mode 並且停止所有震盪器的運作。
下面 3 種情況發生時可以讓 MCU 離開 STOP mode：
(1). 當 IOA, IOC, IOD 的任何一個輸入腳位上出現高電位的(低電位：若 IOA 為 pull-high 電阻)信號以及 SRE 指令所設定的條件成立時，而且必須維持高電位(低電位：若 IOA 為 pull-high 電阻)的信號直到 MCU 解除 HALT mode 為止。

若 IOA,C,D port 沒有選用"Chattering Prevention"的功能時則不需使用 SRE 指令來設定 STOP release condition，且不需維持高電位(低電位：若 IOA 為 pull-high 電阻)的信號到 MCU 解除 HALT mode 為止。

(2). 當 INT pin 的輸入信號有變化時。

(3). 當 key matrix 掃描功能在 KI1~4 腳位上偵測到低電位(高電位：若 LED 模式為"LED HIGH/LOW ACTIVE")的輸入信號時。

指令語法：

OP code	運算元	指令動作
STOP		MCU 進入 STOP mode

FRQ, FRQX

指令特性：
單一字元長度指令，四個 machine cycle，4/8 bits data transferred。

指令說明：
指令運算元中的 immediate data(D)可用來設定 frequency generator 的 output waveform 的 duty cycle。Rx 與 AC 的內容值，@HL 所指向 table ROM 的內容值以及 immediate data (X)則可以用來設定 frequency generator 中 programmable divider 的預設值。指令結束後 frequency generator 就會開始輸出所設定的波形。

下表說明指令運算元中的內容值與 frequency generator 中 programmable divider 的預設值的對應關係：

Instruction\運算元	programmable divider 的預設值的位元對應表							
	bit7	Bit6	bit 5	bit 4	bit 3	Bit 2	bit 1	bit 0
FRQ D,Rx	AC3	AC2	AC1	AC0	(Rx)3	(Rx)2	(Rx)1	(Rx)0
FRQ D,@HL	T7	T6	T5	T4	T3	T2	T1	T0
FRQX D,X	X7	X6	X5	X4	X3	X2	X1	X0

Notes: 1. T0 ~ T7 represents the data of table ROM.
2. X0 ~ X7 represents the immediate data specified in 運算元 X.

設定 frequency generator 的 output waveform 的 duty cycle 方式如下：
D = 0, 可輸出 1/4 duty 的波形。
= 1, 可輸出 1/3 duty 的波形。
= 2, 可輸出 1/2 duty 的波形。
= 3, 可輸出 1/1 duty 的波形。

註記：
(1). FRQ#指令結束後會自動將 HL 的內容值加 1。

(2). Rx 在語法上必須是以絕對位址來描述。
(3). D = 0 ~ 3h。
(4). X = 0 ~ FFh。

指令語法：

OP code	運算元	指令動作
FRQ	D, Rx	Programmable divider ← AC, (Rx) Duty cycle generator ← D
FRQ	D, @HL	Programmable divider ← T(@HL) Duty cycle generator ← D
FRQ#	D, @HL	Programmable divider ← T(@HL) Duty cycle generator ← D HL = HL + 1
FRQX	D, X	Programmable divider ← X Duty cycle generator ← D

SCC

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用 SCC 指令運算元 X 的內容值來選擇 frequency generator 的 clock source 以及選擇 IOA, IOC, IOD port 上的 chattering prevention function 的 clock source。

X6 = 1, 選擇 system clock (BCLK/XCLK)作為 frequency generator 的 clock source(Cfq)
= 0, 選擇 PH0 作為 frequency generator 的 clock source (Cfq)

X5 = 1, 選擇設定 IOA port 上 chattering prevention function 的 clock source

X4 = 1, 選擇設定 IOC port 上 chattering prevention function 的 clock source

X3 = 1, 選擇設定 IOD port 上 chattering prevention function 的 clock source

X2,X1,X0 = 001, 將 PH10 設定為 X5~X3 所選擇的 IO port 上的 chattering prevention function 的 clock source
X2,X1,X0 = 010, 將 PH8 設定為 X5~X3 所選擇的 IO port 上的 chattering prevention function 的 clock source
X2,X1,X0 = 100, 將 PH6 設定為 X5~X3 所選擇的 IO port 上的 chattering prevention function 的 clock source

註記：
(1). X6 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。
(2). 針對部分 MCU 的"X6=1"是 XCLK 則可以藉由 SXCLK 指令設定 X0=0:Cfq= BCLK & X0=1: XCLK=SLOSC/FTOSC(@CSF=0/1)使設定 Cfq =XCLK 時可以避開執行 SLOW/FAST 指令除頻造成 BCLK 頻率變動影響 Cfq 的問題。

指令語法：

OP code	運算元	指令動作
SCC	X	選擇 frequency generator 的 Clock source ← X6 IOA 的 Chattering prevention 能否設定 clock source ← X5 IOC 的 Chattering prevention 能否設定 clock source ← X4 IOD 的 Chattering prevention 能否設定 clock source ← X3 選擇 chattering prevention 的 Clock source ← X2, 1, 0

SCA

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中的 X 值來設定 SEF5, 4, 3 等 3 個 flag。

當 IOA, IOC, IOD 這 3 個 IO port 上的輸入信號有任何變化時，會向 MCU 送出 HALT release 的請求。

X5 = 1, 將 SEF5 flag 設定為 1，MCU 會在 IOA port 上的輸入信號有任何變化時產生 HALT release。

X4 = 1, 將 SEF4 flag 設定為 1，MCU 會在 IOC port 上的輸入信號有任何變化時產生 HALT release。

X3 = 1, 將 SEF3 flag 設定為 1，MCU 會在 IOD port 上的輸入信號有任何變化時產生 HALT release。

註記：
(1). X2~0 可以是任意值。
(2). X5 代表運算元 X 的 MSB；X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SCA	X	Set/clear SEF5 flag ← X5 Set/clear SEF4 flag ← X4 Set/clear SEF3 flag ← X3

SCX(RFC 架構 B)

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中的 X 值來設定 SEF1, SEF0 兩個 flag。

當 CX2/CX pin 用來控制 RFC counter 的功能啟動後，RFC 功能會在控制信號結束之後送出 HALT release 的請求。

X1 = 1, 將 SEF1 flag 設定為 1，MCU 會在 CX2 pin 的控制信號結束之後產生 HALT release。

X0 = 1, 將 SEF0 flag 設定為 1，MCU 會在 CX pin 的控制信號結束之後產生 HALT release。

註記：
X1 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SCX	X	Set/clear SEF1 flag ← X1 Set/clear SEF0 flag ← X0

SIE*

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中的 X 值來設定不同的 interrupt enable flag。

這些 flag 的設定可以讓 MCU 決定是否接受這些 interrupt 請求而去執行該項中斷服務。

X7 = 1, 將 IEF7 設定成 1，MCU 會接受因為 TMR3 發生 underflow 而產生的 interrupt request。

X6 = 1, 將 IEF6 設定成 1，MCU 會接受 RFC counter (在 CX or CX2 pin 控制模式下) 控制信號結束後所產生的 interrupt request。

X5 = 1, 將 IEF5 設定成 1，MCU 會接受因為矩陣式鍵盤掃描功能上接收到掃描信號時而產生的 interrupt request。

X4 = 1, 將 IEF4 設定成 1，MCU 會接受因為 TMR2 發生 underflow 而產生的 interrupt request。

X3 = 1, 將 IEF3 設定成 1，MCU 會接受因為 Pre-divider 的 PH15 輸出信號發生轉態而產生的 interrupt request。

X2 = 1, 將 IEF2 設定成 1，MCU 會接受因為 INT pin 上發生有效觸發而產生的 interrupt request。

X1 = 1, 將 IEF1 設定成 1，MCU 會接受因為 TMR1 發生 underflow 而產生的 interrupt request。

X0 = 1, 將 IEF0 設定成 1，MCU 會接受因為 IOA, IOC, IOD port 上輸入信號的有效變化而產生的 interrupt request。

註記：

X7 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SIE*	X	Set/clear IEF7 flag ← X7 Set/clear IEF6 flag ← X6 Set/clear IEF5 flag ← X5 Set/clear IEF4 flag ← X4 Set/clear IEF3 flag ← X3 Set/clear IEF2 flag ← X2 Set/clear IEF1 flag ← X1 Set/clear IEF0 flag ← X0

SHE

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中的 X 值來設定不同的 halt release enable flag。

X7 = 1, 將 HEF7 設定成 1，允許 MCU 在 TMR3 發生 underflow 時產生 HALT release。

X5 = 1, 將 HEF5 設定成 1，允許 MCU 在矩陣式鍵盤掃描功能上接收到掃描信號時產生 HALT release。

X4 = 1, 將 HEF4 設定成 1，允許 MCU 在 TMR2 發生 underflow 時產生 HALT release。

X3 = 1, 將 HEF3 設定成 1，允許 MCU 在 Pre-divider 的 PH15 輸出信號發生轉態時產生 HALT release。

X2 = 1, 將 HEF2 設定成 1，允許 MCU 在 INT pin 上發生有效觸發時產生 HALT release。

X1 = 1, 將 HEF1 設定成 1，允許 MCU 在 TMR1 發生 underflow 時產生 HALT release。

註記：
(1). X7 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。
(2). X6, X0 兩個位元必須固定為 0。

指令語法：

OP code	運算元	指令動作
SHE	X	Set/clear HEF7 flag ← X7 Set/clear HEF5 flag ← X5 Set/clear HEF4 flag ← X4 Set/clear HEF3 flag ← X3 Set/clear HEF2 flag ← X2 Set/clear HEF1 flag ← X1

SRE

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中的 X 值來設定 stop release enable flag(SRF6, SRF4, SRF3)。

只有在 IOA, IOC, IOD port 選用 chattering prevention 功能之後才需使用這個指令。

X6 = 1, 將 SRF6 設定成 1，允許 MCU 在 IOA port 的輸入腳位上出現高電位信號時產生 STOP release。

X4 = 1, 將 SRF4 設定成 1，允許 MCU 在 IOC port 的輸入腳位上出現高電位信號時產生 STOP release。

X3 = 1, 將 SRF3 設定成 1，允許 MCU 在 IOD port 的輸入腳位上出現高電位信號時產生 STOP release。

註記：
(1). X6 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。
(2). X5, X2, X1, X0 等位元可以是任意值。
(3). 若有 MCU SRE instruction 提供 X7(KEY_S) & X5(INT) 可忽略不設定一樣可產生 STOP release。

指令語法：

OP code	運算元	指令動作
SRE	X	Set/clear SRF6 flag ← X6 Set/clear SRF4 flag ← X4 Set/clear SRF3 flag ← X3

FAST

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
若系統 option 為 Dual Mode，指令執行後會先啟動 CFOSC 震盪器，然後再將執行指令的 system clock(BCLK)切換到由 CFOSC 震盪器產生的震盪頻率(FTOSC)經運算元所設定的除頻值除頻後(若 MCU 有提供除頻功能)所產生的高速 clock(CF clock)。但若已進入 FAST Mode 或系統 option 為 FAST ONLY Mode，則執行 FAST 指令時依設定除頻值直接切換系統頻率(若 MCU 有提供除頻功能)。
利用指令運算元中的 X 值的設定來啟動除頻的功能
X2~0 = 7, 設定 BCLK&SCLK=FTOSC/128。
X2~0 = 6, 設定 BCLK&SCLK=FTOSC/64。
X2~0 = 5, 設定 BCLK&SCLK=FTOSC/32。
X2~0 = 4, 設定 BCLK&SCLK=FTOSC/16。
X2~0 = 3, 設定 BCLK&SCLK=FTOSC/8。

X2~0 = 2, 設定 BCLK&SCLK=FTOSC/4。
X2~0 = 1, 設定 BCLK&SCLK=FTOSC/2。
X2~0 = 0 or NONE, 設定 BCLK&SCLK=FTOSC。

指令語法：

OP code	運算元	指令動作
FAST	(X)	將 system clock 切換到高速的 clock, 或在高速的 clock 除頻頻率間切換系統頻率.

SLOW

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
若系統 option 為 Dual Mode&在 FAST Mode(CSF=1)下執行 SLOW 指令則會將執行指令的 system clock(BCLK)切換到由 XTOSC 震盪器所產生的低速 clock(XT clock)，之後停止 CFOSC 震盪器的動作。
若 MCU 有提供除頻功能，則可以利用指令運算元中的 X 值的設定來啟動除頻的功能
X2~0 = 7, 設定 BCLK&SCLK=SLOSC/128。
X2~0 = 6, 設定 BCLK&SCLK=SLOSC/64。
X2~0 = 5, 設定 BCLK&SCLK=SLOSC/32。
X2~0 = 4, 設定 BCLK&SCLK=SLOSC/16。
X2~0 = 3, 設定 BCLK&SCLK=SLOSC/8。
X2~0 = 2, 設定 BCLK&SCLK=SLOSC/4。
X2~0 = 1, 設定 BCLK&SCLK=SLOSC/2。
X2~0 = 0 or NONE, 設定 BCLK&SCLK=SLOSC。

指令語法：

OP code	運算元	指令動作
SLOW	(X)	將 system clock 切換到低速的 clock, 或在低速的 clock 除頻頻率間切換系統頻率.

SF

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中的 X 值的設定來啟動對應的功能

X7 = 1, 開啟 TMR1 的 re-load function

X6 = 1, 開啟 TMR3 的 re-load function

X5 = 1, 設定 all timer 的內容值只能透過特定暫存器才能讀取，並只有在每次這個指令執行之後才會將所有 TMR1~3 內容值同時更新到此特定暫存器中

X4 = 1, 先將 watchdog timer 歸零之後啟動 watchdog timer，並將 WDF flag 設定為 1

X3 = 1, 如果將 X2 與 X3 同時設定為 1，在指令執行後會開啟 EL plant driver 的功能， 並且讓 MCU 進入 halt mode

X2 = 1, 開啟 EL plant driver 的功能

X1 = 1, MCU 會進入電力備援 (Back Up) 模式並將 BCF flag 設定為 1

X0 = 1, 將 CF 設定為 1

註記：
X7 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SF	X	開啟 TMR1 的 re-load function ← X7 開啟 TMR3 的 re-load function ← X6 設定 timer 內容值的先儲存後讀取功能← X5 將 watchdog timer 歸零後啟動，WDF flag 設定為 1← X4 開啟 EL panel driver 後進入 halt mode ← X3 開啟 EL panel driver ← X2 進入電力備援模式並將 BCF flag 設定為 1← X1 CF flag 設定為 1 ← X0

RF

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元中的 X 值的設定來取消 SF 指令所啟動的對應功能

X7 = 1, 關閉 TMR1 的 re-load function

X6 = 1, 關閉 TMR3 的 re-load function

X5 = 1, 設定 timer 的內容值可以直接讀取並儲存到 data RAM 中

X4 = 1, 停止 watchdog timer 的動作並將 WDF flag 清除為 0

X2 = 1, 關閉 EL panel driver 的功能

X1 = 1, MCU 會離開電力備援模式並將 BCF flag 清除為 0

X0 = 1, 將 CF 清除為 0

註記：
X7 代表運算元 X 的 MSB, X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
RF	X	關閉 TMR1 的 re-load function ← X7 關閉 TMR3 的 re-load function ← X6 設定 timer 的內容值是可以直接讀取 ← X5 停止 watchdog timer, WDF flag 清除為 0 ← X4 關閉 EL panel driver ← X2 離開 backup mode 並將 BCF flag 清除為 0 ← X1 CF flag 清除為 0 ← X0

SF2

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元中的 X 值的設定來啟動對應的功能。

<RFC 架構 A:X6~4>

X6 = 1, 啟動跳躍至 X5,4 所設定的 BANK 對應位址。

X5,4 = 0, 設定跳躍至 BANK0。

X5,4 = 1, 設定跳躍至 BANK1。

X5,4 = 2, 設定跳躍至 BANK2。

X5,4 = 3, 設定跳躍至 BANK3。

<RFC 架構 B:X6~4>

X6 = 1, 啟動以 CX2 pin 作為輸入信號的 RFC 功能，包括 RC 震盪器的起振以及 RFC counter 的計數動作

X5 = 1, 啟動以 CX pin 作為輸入信號的 RFC 功能，包括 RC 震盪器的起振以及 RFC counter 的計數動作

X3 = 1, 開啟 INT 輸入 pin 上內建低阻值的 pull-low 元件

X2 = 1, 強制將所有 LCD driver 的 COM,SEG 腳位只輸出 GND 的信號

X1 = 1, 將 Dis-ENX flag 設定成 1，TMR2 在 re-load 模式下不會因為發生 timer underflow 而停止 RFC counter 動作

X0 = 1, 開啟 TMR2 的 re-load function

註記：

X6 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

<RFC 架構 A>

OP code	運算元	指令動作
SF2	X	啟動跳躍至 X5,4 所設定的 BANK 對應位址← X6 設定跳躍 BANK← X5,4 開啟 INT pin 上的低阻值 pull-low 元件← X3 強制 LCD 所有 COM, SEG 腳位輸出 GND ← X2 將 Dis_ENX flag 設定為 0 ← X1 開啟 TMR2 的 re-load function ← X0

<RFC 架構 B>

OP code	運算元	指令動作
SF2	X	開啟 CX2 控制 RFC counter 的功能← X6 開啟 CX 控制 RFC counter 的功能← X5 開啟 INT pin 上的低阻值 pull-low 元件← X3 強制 LCD 所有 COM, SEG 腳位輸出 GND ← X2 將 Dis_ENX flag 設定為 0 ← X1 開啟 TMR2 的 re-load function ← X0

RF2

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元中 X 值的設定來取消 SF2 指令所啟動的對應功能

X6 = 1, 關閉以 CX2 pin 作為輸入信號的 RFC 功能(僅供 RFC 架構 B)

X5 = 1, 關閉以 CX pin 作為輸入信號的 RFC 功能(僅供 RFC 架構 B)

X3 = 1, 關閉 INT 輸入 pin 上內建低阻值的 pull-low 元件

X2 = 1, 讓所有的 LCD COM, SEG 輸出腳位恢復正常的輸出波形

X1 = 1, 將 Dis-ENX flag 清除為 0，TMR2 在 re-load 模式下發生 timer underflow 時會停止 RFC counter 動作

X0 = 1, 關閉 TMR2 的 re-load function

註記：

X6 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
RF2	X	取消 CX2 對 RFC counter 的控制 ← X6 取消 CX 對 RFC counter 的控制 ← X5 關閉 INT pin 上的低阻值 pull-low 元件 ← X3 所有 COM,SEG 腳位恢復輸出正常波形 ← X2 將 Dis_ENX flag 清除為 0← X1 關閉 TMR2 的 re-load function ← X0

PLC

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
設定指令運算元中的 X 值來清除相對應的 halt release request flags (HRFn)。

X8 = 1, 將 pre-divider 的最後 5 個 bit(PH11~15)的暫存器內容值清除為 0。為避免因 PH14 清除為 0 時而產生 halt release request(HRF3)，本設定應與 X3 同時使用。

X7 = 1, 將 TMR3 underflow 之後所設定的 halt release request flag (HRF7) 清除為 0。如果此時 TMR3 已經啟動而且操作在單次倒數計時功能下，TMR3 會立即停止動作；如果是操作在 re-load 功能下，TMR3 的動作並不會被影響。

X6 = 1, 將 RFC counter 的控制信號結束後所設定的 halt release request flag (HRF6)清除為 0。
這個設定使用於 RFC counter 的開啟或是關閉是由 CX2 或是 CX pin 上的輸入信號來控制的操作模式。

X5 = 1, 將鍵盤掃描功能接收到掃描信號之後所設定的 halt release request flag (HRF5) 清除為 0。

X4 = 1, 將 TMR2 underflow 之後所設定的 halt release request flag (HRF4) 清除為 0。如果此時 TMR2 已經啟動而且操作在單次倒數計時功能下，TMR2 會立即停止動作；如果是操作在 re-load 功能下，TMR2 的動作並不會被影響。

X3 = 1, 將 pre-divider overflow 之後所設定的 halt release request flag (HRF3) 清除為 0。

X2 = 1, 將 INT 輸入腳位上偵測到信號有效變化後所設定的 halt release request flag (HRF2) 清除為 0。

X1 = 1, 將 TMR1 underflow 之後所設定的 halt release request flag (HRF1)清除為 0。如果此時 TMR1 已經啟動而且操作在單次倒數計時功能下，TMR1 會立即停止動作；如果是操作在 re-load 功能下，TMR1 的動作並不會被影響。

X0 = 1, 將 IOA, IOC 以及 IOD 輸入腳位上偵測到有效的信號變化後所設定的 halt release request flag (HRF0)清除為 0。

註記：
X8 代表運算元 X 的 MSB, X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
PLC	X	清除 pre-divider 的最後 5 個 bit 暫存器 ← X8 清除 HRF7 flag ← X7 清除 HRF6 flag ← X6 清除 HRF5 flag ← X5 清除 HRF4 flag ← X4

		清除 HRF3 flag ← X3 清除 HRF2 flag ← X2 清除 HRF1 flag ← X1 清除 HRF0 flag ← X0

SXCLK(目前在 EV chip (TM8999)不支援)

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元 X 中的內容值來設定切換 RFC 16bits counter clock souce = XCLK & 設定 XCLK=BCLK/FTOSC。X=0 是 MCU 的原始設定。

X1 = 1, 切換 RFC 16bits counter clock souce = XCLK。

X0 = 1, 啟動在 SLOW/FAST 模式時由 XCLK=BCLK 切換至 XCLK=SLOSC/FTOSC。

註記：
X1 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SXCLK	X	切換 Crfc = XCLK ← X1=1 啟動在 FAST 模式切換 XCLK=FTOSC ← X0=1

ADJ(目前在 EV chip (TM8999)不支援)

指令特性：
單一字元長度指令，四個 machine cycle。

指令說明：
利用指令運算元 X 中的內容值來設定是否選擇 TMR1~3 其中一個校正 PDV(PH1~15)頻率。

X1, X0 = 00, 不校正 PDV。這是 MCU 的原始設定。

X1, X0 = 01, 在 TMR1 每次 underflow 時依 X2=0/1 增加/減少一個 PH0 週期來校正 PDV。

X1, X0 = 10, 在 TMR2 每次 underflow 時依 X2=0/1 增加/減少一個 PH0 週期來校正 PDV。

X1, X0 = 11, 在 TMR3 每次 underflow 時依 X2=0/1 增加/減少一個 PH0 週期來校正 PDV。

註記：
(1). Alarm, Timer,EL 等選用 PH1~15 功能須注意特性是否會明顯受到 PDV 校正影響 。
(2). X2 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
ADJ	X	設定以降低/升高頻率模式來校正 PDV ← X2=0/1 不校正 PDV ← X1,0 = 00 (default) 由 TMR1 校正 PDV ← X1,0 = 01 由 TMR2 校正 PDV ← X1,0 = 10 由 TMR3 校正 PDV ← X1,0 = 11

SWPWR(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元 X 中的內容值來切換 1.5/3.0V 電源模式。

X1, X0 = 0X, Mask Option 所設定原始電源模式。

X1, X0 = 10, 切換至 3V(BCF=0 : BAK < VBAT)電源模式。

X1, X0 = 11, 切換至 1.5V 電源模式。

註記：

(1). 強烈建議 BAK=VREG(BCF=0) & LCD 使用 VDL (VL3 = 3 x VDL for 1/3Bias)穩壓模式使執行 SWPWR 指令切換電源模式前後仍可維持 BAK & VL1~3 電位不受影響，以達到單純針對目前 VBAT 電壓範圍調整 I/O pins Pull-Low/High 阻值的目的。

(2). X1 代表運算元 X 的 MSB，X0 代表運算元 X 的 LSB。

指令語法：

OP code	運算元	指令動作
SWPWR	X	原始 1.5/3.0V 電源模式 ← X1,0 = 00 (default) 3.0V(BCF=0 : BAK < VBAT) 電源模式 ← X1,0 = 10 1.5V 電源模式 ← X1,0 = 11

SIAP(目前在 EV chip (TM8999)不支援)

指令特性：

單一字元長度指令，四個 machine cycle。

指令說明：

利用指令運算元 X 中的內容值來啟動 IAP 模式 & 設定 Byte/Word Program Mode。

X1 = 1, Word Program 模式。

X1 = 0, Byte Program 模式。