

ARM組合語言程式設計

ARM微處理器指令優點

■ 小晶片的面積低功耗

- ARM架構還採用了一些特別的技術，在性能的前提下儘量縮小晶片的面積，並降低功耗。

■ 條件式指令

- 所有的指令都可根據前面的執行結果決定是否被執行，從而提高指令的執行效率。

■ 連續存取資料

- 可用載入/存儲指令連續傳輸資料，以提高資料的傳輸效率。

■ 同時完成邏輯處理和移位元

- 可在一條資料處理指令中同時完成邏輯處理和移位處理。

■ 自動增減迴圈

- 在迴圈處理中使用位址的自動增減來提高執行效率。

ARM微處理器的指令

■ ARM兩種指令集

□ ARM指令

- 為32位元的長度，

□ Thumb指令

- 為16位元長度。
- Thumb指令集為ARM指令集的功能子集，但與等價的ARM程式碼相比較，可節省30% 40%以上的存儲空間。

ARM處理器模式

- ARM架構支援7種處理器模式。
- 模式改變
 - 在軟體控制下可以改變模式
 - 外部中斷或例外處理也可以引起模式發生改變。
- 大多數應用程式在使用者模式下執行。
 - 當處理器工作在使用者模式時，正在執行的程式不能存取某些被保護的系統資源，也不能改變模式，除非例外(exception)發生。

ARM支援的7種工作模式

處理器模式	說明
User (usr)	正常程式執行模式
FIQ (fiq)	支援高速資料傳送或通道處理
IRQ (irq)	用於通用中斷處理
Supervisor (svc)	作業系統保護模式
Abort mode (abt)	實現虛擬記憶體和/或記憶體保護
Undefined (und)	支援硬體輔助運算器的軟體仿真
System (sys)	執行特權作業系統工作

ARM微處理器的暫存器結構

- **ARM處理器共有37個暫存器，被分為若干個組(BANK)，這些暫存器包括：**
 - **31個通用暫存器。**
 - **包括程式計數器（PC指標）。**
 - **6個程式狀態暫存器**
 - 用以標識CPU的工作狀態及程式的執行狀態，均為32位元，目前只使用了其中的一部分。
- **ARM處理器有7種不同的處理器模式**
 - **在每一種處理器模式下均有一組相應的暫存器與之對應。**

ARM暫存器

■ 1) 不分組暫存器R0~R7

- R0~R7是不分組暫存器。這意味著在所有處理器模式下，它們每一個都存取一樣的32位元暫存器。它們是真正的通用暫存器，沒有架構所隱含的特殊用途。

■ 2) 分組暫存器R8~R14
















- R8~R14是分組暫存器。它們每一個存取的實體暫存器取決於當前的處理器模式。若要存取特定的實體暫存器而不依賴當前的處理器模式，則要使用規定的各字。
- 暫存器R8~R12各有兩組實體暫存器：一組為FIQ模式，另一組為除了FIQ以外的所有模式。暫存器R8~R12沒有任何指定的特殊用途。
- 只是使用R8~R14來簡單地處理中斷。暫存器R13,R14各有6個分組的實體暫存器。1個用於使用者模式和系統模式，其他5個分別用於5種例外模式。
- 暫存器R13通常用做堆疊指標，稱為SP(Stack Pointer)，每種例外模式都有自己的R13。
- 暫存器R14用作副程式鏈結暫存器，也稱為LR。

■ 3) 程式計數器R15

- 暫存器R15用做程式計數器 (PC)。

ARM暫存器

ARM State General Registers and Program Counter

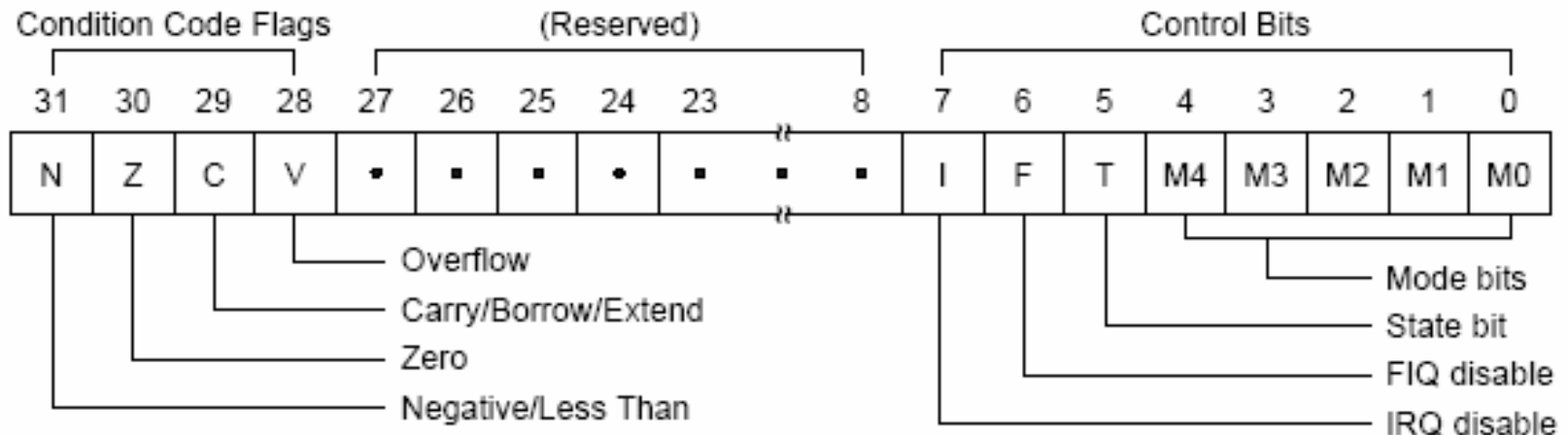
System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	 R8_fiq	R8	R8	R8	R8
R9	 R9_fiq	R9	R9	R9	R9
R10	 R10_fiq	R10	R10	R10	R10
R11	 R11_fiq	R11	R11	R11	R11
R12	 R12_fiq	R12	R12	R12	R12
R13	 R13_fiq	 R13_svc	 R13_abt	 R13_irq	 R13_und
R14	 R14_fiq	 R14_svc	 R14_abt	 R14_irq	 R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

ARM State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	 SPSR_fiq	 SPSR_svc	 SPSR_abt	 SPSR_irq	 SPSR_und

程式狀態暫存器(Program Status Register)

- 有一個**Current Program Status Register (CPSR)**
 - 在所有處理器模式下都可以存取當前的程式狀態暫存器CPSR。
 - CPSR包含條件碼旗標，中斷禁止位元，當前處理器模式以及其他狀態和控制資訊。
- 有五個**Saved Program Status Registers (SPSRs)**
 - 每種例外模式都有一個程式狀態保存暫存器SPSR。
 - SPSR用於保留CPSR的狀態。



程式狀態暫存器

- **條件碼旗標：**
 - **N, Z, C, V** 大多數指令可以測試這些條件碼旗標以決定程式指令如何執行
- **控制位元：**
 - 最低8位元 **I, F, T**和**M**位元用做控制位元。當例外出現時改變控制位元。當處理器在特權模式下也可以由軟體改變。
- **中斷禁止位元：****I** 置1則禁止**IRQ**中斷。**F**置1則禁止**FIQ**中斷。
- **T位元：**
 - **T=0** 指示**ARM**執行。**T=1**指示**Thumb**執行。在這些架構系統中，可自由地使用能在**ARM**和**Thumb**狀態之間切換的指令。
- **模式位元：**
 - **M0, M1, M2, M3**和**M4 (M[4:0])** 是模式位元.這些位元決定處理器的工作模式。

PSR Mode位元與可使用暫存器

M[4:0]	Mode	Visible THUMB state registers	Visible ARM state registers
10000	User	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR
10001	FIQ	R7..R0, LR_fiq, SP_fiq PC, CPSR, SPSR_fiq	R7..R0, R14_fiq..R8_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	R7..R0, LR_irq, SP_irq PC, CPSR, SPSR_irq	R12..R0, R14_irq, R13_irq, PC, CPSR, SPSR_irq
10011	Supervisor	R7..R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc	R12..R0, R14_svc, R13_svc, PC, CPSR, SPSR_svc
10111	Abort	R7..R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt	R12..R0, R14_abt, R13_abt, PC, CPSR, SPSR_abt
11011	Undefined	R7..R0 LR_und, SP_und, PC, CPSR, SPSR_und	R12..R0, R14_und, R13_und, PC, CPSR
11111	System	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR

ARM和Thumb之間狀態的切換

- ARM處理器可在兩種工作狀態之間切換。
- 在Thumb狀態下，程式計數器PC使用位元1選擇另一個半字。
- ARM和Thumb之間狀態的切換不影響處理器的模式或暫存器的內容。
- 進入Thumb狀態。當運算元暫存器的狀態位元0為1時，執行BX指令進入Thumb狀態。
- 如果處理器在Thumb狀態進入例外，則當例外處理（IRQ，FIQ，Undef，Abort和SWI）返回時，自動切換到Thumb狀態。
- 當運算元暫存器的狀態位元0為0時，執行BX指令進入ARM狀態。
- 處理器進行例外處理（IRQ，FIQ，Undef，Abort和SWI），從例外向量位址開始執行也可以進入ARM狀態。

Thumb指令及應用

- Thumb指令集是ARM指令集的一個子集，允許指令編碼為16位元的長度。
- Thumb指令集在保留32程式碼優勢的同時，大大的節省了系統的存儲空間。
- 大多數的Thumb指令是無條件執行的。
- 大多數的Thumb資料處理指令的目的暫存器與其中一個來原暫存器相同。
- Thumb指令與ARM指令的時間效率和空間效率關係為：
 - Thumb程式碼所需的存儲空間約為ARM程式碼的60% 70%
 - Thumb程式碼使用的指令數比ARM程式碼多約30% 40%
 - 若使用32位的記憶體，ARM程式碼比Thumb程式碼快約40%
 - 若使用16位的記憶體，Thumb程式碼比ARM程式碼快約40% 50%
 - 與ARM程式碼相比較，使用Thumb程式碼，記憶體的功耗會降低約30%
- 若對系統的性能有較高要求，應使用32位元的存儲系統和ARM指令集。
- 若對系統的成本及功耗有較高要求，則應使用16位元的存儲系統和Thumb指令集。

Thumb與ARM和暫存器的比較

Thumb 狀態

R0
R1
R2
R3
R4
R5
R6
R7
SP
LR
PC
CPSR
SPSR

ARM 狀態

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
SP (R13)
LR(R14)
PC(R15)
CPSR
SPSR

低暫存器











高暫存器

Thumb狀態的暫存器集

- Thumb狀態下的暫存器集是ARM狀態下暫存器集的子集。程式師可以直接存取8個通用的暫存器(R0~R7), PC, SP, LR和CPSR。每一種特權模式都有一組SP, LR和SPSR。
 - Thumb狀態的R0~R7與ARM狀態的R0~R7是一致的。
 - Thumb狀態的CPSR和SPSR與ARM狀態的CPSR和SPSR是一致的。
 - Thumb狀態的SP映射到ARM狀態的R13。
 - Thumb狀態的LR映射到ARM狀態的R14。
 - Thumb狀態的PC映射到ARM狀態的PC (R15)。

Thumb暫存器

THUMB State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
SP	 SP_fiq	 SP_svc	 SP_abt	 SP_und	 SP_fiq
LR	 LR_fiq	 LR_svc	 LR_abt	 LR_und	 LR_fiq
PC	PC	PC	PC	PC	PC

THUMB State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	 SPSR_fiq	 SPSR_svc	 SPSR_abt	 SPSR_irq	 SPSR_und

ARM記憶體格式

■ BIG-ENDIAN

Higher Address



Lower Address

Word Address							
31	24	23	16	15	8	7	0
8		9		10		11	8
4		5		6		7	4
0		1		2		3	0

- Most significant byte is at lowest address.
- Word is addressed by byte address of most significant byte.

■ LITTLE-ENDIAN

Higher Address



Lower Address

								Word Address
31	24	23	16	15	8	7	0	
11		10		9		8		8
7		6		5		4		4
3		2		1		0		0

- Least significant byte is at lowest address.
- Word is addressed by byte address of least significant byte.

ARM 指令格式(Instruction Format)及種類

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Cond	0	0	I	Opcode				S	Rn				Rd				Operand2								Data/Processing/ PSR Transfer								
Cond	0	0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm				Multiply			
Cond	0	0	0	0	0	1	U	A	S	RdHi				RdLo				Rn				1	0	0	1	Rm				Multiply Long			
Cond	0	0	0	1	0	B	0	0		Rn				Rd				0	0	0	0	1	0	0	1	Rm				Single Data Swap			
Cond	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn				Branch and Exchange				
Cond	0	0	0	P	U	0	W	L		Rn				Rd				0	0	0	0	1	S	H	1	Rm				Halfword Data Transfer: register offset			
Cond	0	0	0	P	U	1	W	L		Rn				Rd				Offset				1	S	H	1	Offset				Halfword Data Transfer: immediat offset			
Cond	0	1	I	P	U	B	W	L		Rn				Rd				Offset												Single Data Transfer			
Cond	0	1	I																					1					Undefined				
Cond	1	0	0	P	U	B	W	L		Rn				Register List															Block Data Transfer				
Cond	1	0	1	L	Offset																							Branch					
Cond	1	1	0	P	U	B	W	L		Rn				CRd				CP#				Offset								Coprocesor Data Transfer			
Cond	1	1	1	0	CP Opc					CRn				CRd				CP#				CP		0	CRm				Coprocesor Data Operation				
Cond	1	1	1	0	CP Opc				L	CRn				Rd				CP#				CP		1	CRm				Coprocesor Register Transfer				
Cond	1	1	1	1	Ignored by processor																									Software Interrupt			

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

ARM指令及功能描述

助記符	指令功能描述
ADC	帶進位元加法指令
ADD	加法指令
AND	邏輯與指令
B	跳移指令
BIC	位元清零指令
BL	帶返回的跳移指令
BLX	帶返回和狀態切換的跳移指令
BX	帶狀態切換的跳移指令
CDP	輔助運算器資料操作指令
CMN	比較反值指令
CMP	比較指令
EOR	互斥指令
LDC	記憶體到輔助運算器的資料傳輸指令
LDM	載入多個暫存器指令
LDR	記憶體到暫存器的資料傳輸指令
MCR	從ARM暫存器到輔助運算器暫存器的資料傳輸指令
MLA	乘加運算指令
MOV	資料傳送指令

ARM指令及功能描述

助記符	指令功能描述
MRC	從輔助運算器暫存器到ARM暫存器的資料傳輸指令
MRS	傳送CPSR或SPSR的內容到通用暫存器指令
MSR	傳送通用暫存器到CPSR或SPSR的指令
MUL	32位元乘法指令
MLA	32位元乘加指令
MVN	資料取反傳送指令
ORR	邏輯或指令
RSB	逆向減法指令
RSC	帶借位的逆向減法指令
SBC	帶借位減法指令
STC	輔助運算器暫存器寫入記憶體指令
STM	連續記憶體字寫入指令
STR	暫存器到記憶體的資料傳輸指令
SUB	減法指令
SWI	軟體中斷指令
SWP	交換指令
TEQ	相等測試指令
TST	位元測試指令

指令的條件碼

條件碼	助記符尾碼	旗標	含義
0000	EQ	Z置位	相等
0001	NE	Z清零	不相等
0010	CS	C置位	無符號數大於或等於
0011	CC	C清零	無符號數小於
0100	MI	N置位	負數
0101	PL	N清零	正數或零
0110	VS	V置位	溢出
0111	VC	V清零	未溢出
1000	HI	C置位Z清零	無符號數大於
1001	LS	C清零Z置位	無符號數小於或等於
1010	GE	N等於V	帶符號數大於或等於
1011	LT	N不等於V	帶符號數小於
1100	GT	Z清零且 (N等於V)	帶符號數大於
1101	LE	Z置位或 (N不等於V)	帶符號數小於或等於
1110	AL	忽略	無條件執行

ARM指令的定址方式

- 立即定址(Immediate Addressing)
- 暫存器定址(Register Addressing)
- 暫存器間接定址(Register Indirect Addressing)
- 基址變址定址(Index Addressing)
- 多暫存器定址(Multiple Register Addressing)
- 堆疊定址(Stack Addressing)

ARM指令的定址方式-立即定址

- 立即定址也叫立即數定址，這是一種特殊的定址方式，運算元本身就在指令中給出，只要取出指令也就取到了運算元。這個運算元被稱為立即數，對應的定址方式也就叫做立即定址。例如以下指令：

ADD	R0, R0, # 1	; R0 R0 + 1
ADD	R0, R0, # 0x3f	; R0 R0 + 0x3f

- 在以上兩條指令中，第二個源運算元即為立即數，要求以“#”為首碼，對於以十六進位表示的立即數，還要求在“#”後加上“0x”或“&”。

ARM指令的定址方式-暫存器定址

- 暫存器定址就是利用暫存器中的數值作為運算元，這種定址方式是各類微處理器經常採用的一種方式，也是一種執行效率較高的定址方式。以下指令：

ADD R0 , R1 , R2	; R0 = R1 + R2
------------------	----------------

- 該指令的執行效果是將暫存器R1和R2的內容相加，其結果存放在暫存器R0中。

ARM指令的定址方式-暫存器間接定址

- 暫存器間接定址就是以暫存器中的值作為運算元的位址，而運算元本身存放在記憶體中。例如以下指令：

LDR	R0 , [R1]	; R0 ← [R1]
STR	R0 , [R1]	; [R1] ← R0

- 第一條指令將以R1的值為位址的記憶體中的資料傳送到R0中。
- 第二條指令將R0的值傳送到以R1的值為位址的記憶體中。

ARM指令的定址方式-基址變址定址

■ 基址變址定址就是將暫存器（該暫存器一般稱作基址暫存器）的內容與指令中給出的位址偏移量相加，從而得到一個運算元的有效位址。變址定址方式常用於存取某基底位址附近的位址單元。採用變址定址方式的指令常見有以下幾種形式，如下所示：

LDR R0 , [R1 , # 4]	; R0 [R1 + 4]
LDR R0 , [R1 , # 4] !	; R0 [R1 + 4]、 R1 R1 + 4
LDR R0 , [R1] , # 4	; R0 [R1]、 R1 R1 + 4
LDR R0 , [R1 , R2]	; R0 [R1 + R2]

■ 在第一條指令中，將暫存器R1的內容加上4形成運算元的有效位址，從而取得運算元存入暫存器R0中。

■ 在第二條指令中，將暫存器R1的內容加上4形成運算元的有效位址，從而取得運算元存入暫存器R0中，然後，R1的內容自增4個位元組。

■ 在第三條指令中，以暫存器R1的內容作為運算元的有效位址，從而取得運算元存入暫存器R0中，然後，R1的內容自增4個位元組。

■ 在第四條指令中，將暫存器R1的內容加上暫存器R2的內容形成運算元的有效位址，從而取得運算元存入暫存器R0中。

ARM指令的定址方式-多暫存器定址

- 採用多暫存器定址方式，一條指令可以完成多個暫存器值的傳送。這種定址方式可以用一條指令完成傳送最多16個通用暫存器的值。以下指令：

LDMIA R0 , {R1 , R2 , R3 , R4}	; R1 [R0]
	; R2 [R0 + 4]
	; R3 [R0 + 8]
	; R4 [R0 + 12]

- 該指令的尾碼IA表示在每次執行完載入/存儲操作後，R0按字長度增加，因此，指令可將連續存儲單元的值傳送到R1 R4。

ARM指令的定址方式-相對定址

- 與基址變址定址方式相類似，相對定址以程式計數器PC的當前值為基底位址，指令中的位址標號作為偏移量，將兩者相加之後得到運算元的有效位址。以下程式段完成副程式的使用和返回，跳移指令BL採用了相對定址方式：

BL	NEXT	；跳移到副程式NEXT處執行
.....		
NEXT		
.....		
MOV	PC , LR	；從副程式返回

ARM指令的定址方式-堆疊定址

- 堆疊是一種資料結構，按先進後出（First In Last Out，FILO）的方式工作，使用一個稱作堆疊指標的專用暫存器指示當前的操作位置，堆疊指標總(SP, Stack Pointer)是指向堆疊頂。
- 當堆疊指標指向最後壓入堆疊的資料時，稱為滿堆疊（Full Stack），而當堆疊指標指向下一個將要放入資料的空位置時，稱為空堆疊（Empty Stack）。
- 同時，根據堆疊的生成方式，又可以分為遞增堆疊（Ascending Stack）和遞減堆疊（Descending Stack），當堆疊由低位址向高位址生成時，稱為遞增堆疊，當堆疊由高位址向低位址生成時，稱為遞減堆疊。這樣就有四種類型的堆疊工作方式，ARM微處理器支援這四種類型的堆疊工作方式，即：
 - 滿遞增堆疊：堆疊指標指向最後壓入的資料，且由低位址向高位址生成。
 - 滿遞減堆疊：堆疊指標指向最後壓入的資料，且由高位址向低位址生成。
 - 空遞增堆疊：堆疊指標指向下一個將要放入資料的空位置，且由低位址向高地址生成。
 - 空遞減堆疊：堆疊指標指向下一個將要放入資料的空位置，且由高位址向低地址生成。

ARM指令集-跳移指令

- 跳移指令用於實現程式流程的跳移，在ARM程式中有兩種方法可以實現程式流程的跳移：
 - 使用專門的跳移指令。
 - 直接向程式計數器PC寫入跳移位址值。
- 通過向程式計數器PC寫入跳移位址值，可以實現在4GB的位址空間中的任意跳移，在跳移之前結合使用。
 - MOV LR, PC
- 等類似指令，可以保存將來的返回位址值，從而實現在4GB連續的線性位址空間的副程式使用。
- ARM指令集中的跳移指令可以完成從當前指令向前或向後的32MB的位址空間的跳移，包括以下4條指令：
 - B 跳移指令。
 - BL 帶返回的跳移指令。
 - BLX 帶返回和狀態切換的跳移指令。
 - BX 帶狀態切換的跳移指令。

ARM指令集-跳移指令-B指令

- B指令的語法為：
 - B{條件} 目標位址
- B指令是最簡單的跳移指令。一旦遇到一個 B 指令，ARM 處理器將立即跳移到給定的目標位址，從那裏繼續執行。注意存儲在跳移指令中的實際值是相對當前PC值的一個偏移量，而不是一個絕對位址，它的值由彙編器來計算（參考定址方式中的相對定址）。
- 它是 24 位元有符號數，左移兩位元後有符號擴充為 32 位，表示的有效偏移為 26 位(前後32MB的位址空間)。以下指令：

B	Label	；程式無條件跳移到標號Label處執行
CMP	R1, #0	；當CPSR暫存器中的Z條件碼置位元時，程式跳移到標號Label處執行
BEQ	Label	

ARM指令集-跳移指令-BL指令

- **BL指令的語法為：**
 - **BL{條件} 目標位址**
- **BL 是另一個跳移指令，但跳移之前，會在暫存器R14中保存PC的當前內容，因此，可以通過將R14的內容重新載入到PC中，來返回到跳移指令之後的那個指令處執行。**
- **該指令是實現副程式使用的一個基本但常用的手段。以下指令：**

•BL	Label	；當程式無條件跳移到標號Label處執行時，同時將當前的PC值保存到R14中
-----	-------	--

ARM指令集-跳移指令-BLX指令

- **BLX指令的語法為：**
 - **BLX 目標位址**
- **BLX指令從ARM指令集跳移到指令中所指定的目標位址，並將處理器的工作狀態有ARM狀態切換到Thumb狀態，該指令同時將PC的當前內容保存到暫存器R14中。**
- **因此，當副程式使用Thumb指令集，而使用者使用ARM指令集時，可以通過BLX指令實現副程式的使用和處理器工作狀態的切換。同時，副程式的返回可以通過將暫存器R14值複製到PC中來完成。**

ARM指令集-跳移指令-BX指令

- **BX指令的語法為：**
 - **BX{條件} 目標位址**
- **BX指令跳移到指令中所指定的目標位址，目標位址處的指令既可以是ARM指令，也可以是Thumb指令。**

ARM指令集-跳移指令-資料處理指令

■ 資料處理指令可分為

□ 資料傳送指令

- 資料傳送指令用於在暫存器和記憶體之間進行資料的雙向傳輸。

□ 算術邏輯運算指令

- 算術邏輯運算指令完成常用的算術與邏輯的運算，該類指令不但將運算結果保存在目的暫存器中，同時更新CPSR中的相應條件旗標位元。

□ 比較指令

- 比較指令不保存運算結果，只更新CPSR中相應的條件旗標位元。

資料處理指令總表

□	MOV	資料傳送指令
□	MVN	資料取反傳送指令
□	CMP	比較指令
□	CMN	反值比較指令
□	TST	位元測試指令
□	TEQ	相等測試指令
□	ADD	加法指令
□	ADC	帶進位元加法指令
□	SUB	減法指令
□	SBC	帶借位減法指令
□	RSB	逆向減法指令
□	RSC	帶借位的逆向減法指令
□	AND	邏輯與指令
□	ORR	邏輯或指令
□	EOR	邏輯互斥指令
□	BIC	位元清除指令

ARM指令集-資料處理指令-MOV指令

- **MOV指令的語法為：**
 - **MOV{條件}{S}** 目的暫存器，來源運算元
- **MOV指令可完成從另一個暫存器、被移位的暫存器或將一個立即數載入到目的暫存器。其中S選項決定指令的操作是否影響CPSR中條件旗標位元的值，當沒有S時指令不更新CPSR中條件旗標位元的值。**
- **指令範例：**

MOV	R1 , R0	；將暫存器R0的值傳送到暫存器R1
MOV	PC , R14	；將暫存器R14的值傳送到PC，常用於副程式返回
MOV	R1 , R0 , LSL # 3	；將暫存器R0的值左移3位後傳送到R1

ARM指令集-資料處理指令-MVN指令

- **MVN指令的語法為：**
 - **MVN{條件}{S}** 目的暫存器，來源運算元
- **MVN指令可完成從另一個暫存器、被移位的暫存器、或將一個立即數載入到目的暫存器。與MOV指令不同之處是在傳送之前按位被取反了，即把一個被取反的值傳送到目的暫存器中。**
- **其中S決定指令的操作是否影響CPSR中條件旗標位元的值，當沒有S時指令不更新CPSR中條件旗標位元的值。**
- **指令範例：**

•MVN R0 , # 0 ; 將立即數0取反傳送到暫存器R0中，完成後R0=-1

ARM指令集-資料處理指令-CMP指令

- **CMP指令的語法為：**
 - **CMP{條件} 運算元1，運算元2**
- **CMP指令用於把一個暫存器的內容和另一個暫存器的內容或立即數進行比較，同時更新CPSR中條件旗標位元的值。該指令進行一次減法運算，但不存儲結果，只更改條件旗標位元。**
- **旗標位元表示的是運算元1與運算元2的關係(大、小、相等)，例如，當運算元1大於操作運算元2，則此後的有GT 尾碼的指令將可以執行。**
- **指令範例：**

CMP R1 , R0	；將暫存器R1的值與暫存器R0的值相減，並根據結果設定CPSR的旗標位元
CMP R1 , # 100	；將暫存器R1的值與立即數100相減，並根據結果設定CPSR的旗標位元

ARM指令集-資料處理指令-CMN指令

- CMN指令的語法為：
 - CMN{條件} 運算元1，運算元2
- CMN指令用於把一個暫存器的內容和另一個暫存器的內容或立即數取反後進行比較，同時更新CPSR中條件旗標位元的值。
- 該指令實際完成運算元1和運算元2相加，並根據結果更改條件旗標位元。
- 指令範例：

CMN	R1, R0	；將暫存器R1的值與暫存器R0的值相加，並根據結果設定CPSR的旗標位元
CMN	R1, # 100	；將暫存器R1的值與立即數100相加，並根據結果設定CPSR的旗標位元

ARM指令集-資料處理指令-TST指令

- **TST指令的語法為：**
 - **TST{條件} 運算元1，運算元2**
- **TST指令用於把一個暫存器的內容和另一個暫存器的內容或立即數進行按位的與運算，並根據運算結果更新CPSR中條件旗標位元的值。**
- **運算元1是要測試的資料，而運算元2是一個位遮罩，該指令一般用來測試是否設定了特定的位。**
- **指令範例：**

TST	R1, # %1	；用於測試在暫存器R1中是否設定了最低位（%表示二進位數字）
TST	R1, # 0xffe	；將暫存器R1的值與立即數0xffe按位元與，並根據結果設定CPSR的旗標位元

ARM指令集-資料處理指令-TEQ指令

- TEQ指令的語法為：
 - TEQ {條件} 運算元1 , 運算元2
- TEQ指令用於把一個暫存器的內容和另一個暫存器的內容或立即數進行按位的互斥運算，並根據運算結果更新CPSR中條件旗標位元的值。
- 該指令通常用於比較運算元1和運算元2是否相等。
- 指令範例：

•TEQ	R1 , R2	；將暫存器R1的值與暫存器R2的值按位元互斥，並根據結果設定CPSR的旗標位元
------	---------	---

ARM指令集-資料處理指令-ADD指令

- **ADD指令的語法為：**
 - **ADD {條件}{S} 目的暫存器，運算元1，運算元2**
- **ADD指令用於把兩個運算元相加，並將結果存放到目的暫存器中。運算元1應是一個暫存器，運算元2可以是一個暫存器，被移位的暫存器，或一個立即數。**
- **指令範例：**

ADD	R0 , R1 , R2	; R0 = R1 + R2
ADD	R0 , R1 , #256	; R0 = R1 + 256
ADD	R0 , R2 , R3 , LSL#1	; R0 = R2 + (R3 << 1)

ARM指令集-資料處理指令-ADC指令

- **ADC指令的語法為：**
 - **ADC {條件}{S} 目的暫存器，運算元1，運算元2**
- **ADC指令用於把兩個運算元相加，再加上CPSR中的C條件旗標位元的值，並將結果存放到目的暫存器中。它使用一個進位元旗標位元，這樣就可以做比32位大的數的加法，注意不要忘記設定S尾碼來更改進位元旗標。運算元1應是一個暫存器，運算元2可以是一個暫存器，被移位的暫存器，或一個立即數。**
- **以下指令序列完成兩個128位數的加法，第一個數由高到低存放在暫存器R7 R4，第二個數由高到低存放在暫存器R11 R8，運算結果由高到低存放在暫存器R3 R0：**

ADDS	R0 , R4 , R8	； 加低端的字
ADCS	R1 , R5 , R9	； 加第二個字，帶進位
ADCS	R2 , R6 , R10	； 加第三個字，帶進位
ADC	R3 , R7 , R11	； 加第四個字，帶進位

ARM指令集-資料處理指令-SUB指令

- **SUB指令的語法為：**
 - **SUB {條件}{S} 目的暫存器，運算元1，運算元2**
- **SUB指令用於把運算元1減去運算元2，並將結果存放到目的暫存器中。運算元1應是一個暫存器，運算元2可以是一個暫存器，被移位的暫存器，或一個立即數。該指令可用於有符號數或無符號數的減法運算。**
- **指令範例：**

SUB	R0 , R1 , R2	; R0 = R1 - R2
SUB	R0 , R1 , #256	; R0 = R1 - 256
SUB	R0 , R2 , R3 , LSL#1	; R0 = R2 - (R3 << 1)

ARM指令集-資料處理指令-SBC指令

- **SBC指令的語法為：**
 - **SBC {條件}{S} 目的暫存器，運算元1，運算元2**
- **SBC指令用於把運算元1減去運算元2，再減去CPSR中的C條件旗標位元的反碼，並將結果存放到目的暫存器中。運算元1應是一個暫存器，運算元2可以是一個暫存器，被移位的暫存器，或一個立即數。該指令使用進位元旗標來表示借位，這樣就可以做大於32位的減法，注意不要忘記設定S尾碼來更改進位元旗標。該指令可用於有符號數或無符號數的減法運算。**
- **指令範例：**

•SUBS R0, R1, R2 ; R0 = R1 - R2 - !C，並根據結果設定CPSR的進位元旗標位元

ARM指令集-資料處理指令- RSB指令

- **RSB指令的語法為：**
 - **RSB{條件}{S} 目的暫存器，運算元1，運算元2**
- **RSB指令稱為逆向減法指令，用於把運算元2減去運算元1，並將結果存放到目的暫存器中。運算元1應是一個暫存器，運算元2可以是一個暫存器，被移位的暫存器，或一個立即數。**
- **該指令可用於有符號數或無符號數的減法運算。**
- **指令範例：**

RSB	R0 , R1 , R2	; R0 = R2 – R1
RSB	R0 , R1 , #256	; R0 = 256 – R1
RSB	R0 , R2 , R3 , LSL#1	; R0 = (R3 << 1) - R2

ARM指令集-資料處理指令-RSC指令

- RSC指令的語法為：
 - RSC {條件}{S} 目的暫存器，運算元1，運算元2
- RSC指令用於把運算元2減去運算元1，再減去CPSR中的C條件旗標位元的反碼，並將結果存放到目的暫存器中。
- 運算元1應是一個暫存器，運算元2可以是一個暫存器，被移位的暫存器，或一個立即數。該指令使用進位元旗標來表示借位，這樣就可以做大於32位的減法，注意不要忘記設定S尾碼來更改進位元旗標。
- 該指令可用於有符號數或無符號數的減法運算。
- 指令範例：

RSC	R0 , R1 , R2	; R0 = R2 – R1 - ! C
-----	--------------	----------------------

ARM指令集-資料處理指令-AND指令

- **AND指令的語法為：**
 - **AND {條件}{S} 目的暫存器，運算元1，運算元2**
- **AND指令用於在兩個運算元上進行邏輯與運算，並把結果放置到目的暫存器中。運算元1應是一個暫存器，運算元2可以是一個暫存器，被移位的暫存器，或一個立即數。該指令常用於遮罩運算元1的某些位。**
- **指令範例：**

•AND R0 , R0 , # 3	； 該指令保持R0的0、1位，其餘位清零。
-----------------------	-----------------------

ARM指令集-資料處理指令- ORR指令

- ORR指令的語法為：
 - ORR {條件}{S} 目的暫存器，運算元1，運算元2
- ORR指令用於在兩個運算元上進行邏輯或運算，並把結果放置到目的暫存器中。運算元1應是一個暫存器，運算元2可以是一個暫存器，被移位的暫存器，或一個立即數。該指令常用於設定運算元1的某些位。
- 指令範例：

ORR	R0, R0, #3	; 該指令設定R0的0、1位，其餘位保持不變。
-----	------------	-------------------------

ARM指令集-資料處理指令- EOR指令

- EOR指令的語法為：
 - EOR{條件}{S} 目的暫存器，運算元1，運算元2
- EOR指令用於在兩個運算元上進行邏輯互斥運算，並把結果放置到目的暫存器中。運算元1應是一個暫存器，運算元2可以是一個暫存器，被移位的暫存器，或一個立即數。該指令常用於反轉運算元1的某些位。
- 指令範例：

- EOR R0, R0, #3 ; 該指令反轉R0的0、1位，其餘位保持不變。

ARM指令集-資料處理指令-BIC指令

- **BIC指令的語法為：**
 - **BIC {條件}{S} 目的暫存器，運算元1，運算元2**
- **BIC指令用於清除運算元1的某些位，並把結果放置到目的暫存器中。運算元1應是一個暫存器，運算元2可以是一個暫存器，被移位的暫存器，或一個立即數。運算元2為32位的遮罩，如果在遮罩中設定了某一位，則清除這一位。未設定的遮罩位保持不變。**
- **指令範例：**

BIC R0, R0, # %1011

； 該指令清除 R0 中的位 0、1、和 3，其餘的位保持不變。

ARM指令集-乘法指令與乘加指令

- ARM微處理器支援的乘法指令與乘加指令共有6條，可分為運算結果為32位和運算結果為64位兩類，與前面的資料處理指令不同，指令中的所有運算元、
- 目的暫存器必須為通用暫存器，不能對運算元使用立即數或被移位的暫存器，同時，目的暫存器和運算元1必須是不同的暫存器。
- 乘法指令與乘加指令共有以下6條：
 - **MUL** 32位元乘法指令
 - **MLA** 32位元乘加指令
 - **SMULL** 64位元有符號數乘法指令
 - **SMLAL** 64位元有符號數乘加指令
 - **UMULL** 64位元無符號數乘法指令
 - **UMLAL** 64位元無符號數乘加指令

ARM指令集-乘法指令與乘加指令- MUL指令

- **MUL指令的語法為：**
 - **MUL{條件}{S}** 目的暫存器，運算元1，運算元2
- **MUL指令完成將運算元1與運算元2的乘法運算，並把結果放置到目的暫存器中，同時可以根據運算結果設定CPSR中相應的條件旗標位元。其中，運算元1和運算元2均為32位元的有符號數或無符號數。**
- **指令範例：**

MUL	R0 , R1 , R2 ; R0 = R1 × R2
MULS	R0 , R1 , R2 ; R0 = R1 × R2 , 同時設定CPSR中的相關條件旗標位元

ARM指令集-乘法指令與乘加指令- MLA指令

- **MLA指令的語法為：**

- **MLA{條件}{S}** 目的暫存器，運算元1，運算元2，
運算元3

- **MLA指令完成將運算元1與運算元2的乘法運算，再將乘積加上運算元3，並把結果放置到目的暫存器中，同時可以根據運算結果設定CPSR中相應的條件旗標位元。其中，運算元1和運算元2均為32位元的有符號數或無符號數。**

- **指令範例：**

MLA	R0, R1, R2, R3 ; $R0 = R1 \times R2 + R3$
MLAS	R0, R1, R2, R3 ; $R0 = R1 \times R2 + R3$ ，同時設定CPSR中的相關條件旗標位元

ARM指令集-乘法指令與乘加指令- SMULL指令

- **SMULL指令的語法為：**
 - **SMULL{條件}{S}** 目的暫存器Low，目的暫存器High，運算元1，運算元2
- **SMULL指令完成將運算元1與運算元2的乘法運算，並把結果的低32位元放置到目的暫存器Low中，結果的高32位元放置到目的暫存器High中，同時可以根據運算結果設定CPSR中相應的條件旗標位元。其中，運算元1和運算元2均為32位元的有符號數。**
- **指令範例：**

```
SMULL R0 , R1 , R2 , R3 ; R0 = ( R2 x R3 ) 的低32位  
                        ; R1 = ( R2 x R3 ) 的高32位
```


ARM指令集-乘法指令與乘加指令-SMLAL指令

- SMLAL指令的語法為：

- $\text{SMLAL}\{\text{條件}\}\{\text{S}\}$ 目的暫存器Low，目的暫存器High，
運算元1，運算元2

- SMLAL指令完成將運算元1與運算元2的乘法運算，並把結果的低32位元同目的暫存器Low中的值相加後又放置到目的暫存器Low中，結果的高32位元同目的暫存器High中的值相加後又放置到目的暫存器High中，同時可以根據運算結果設定CPSR中相應的條件旗標位元。其中，運算元1和運算元2均為32位元的有符號數。

- 對於目的暫存器Low，在指令執行前存放64位元加數的低32位元，指令執行後存放結果的低32位。

- 對於目的暫存器High，在指令執行前存放64位元加數的高32位元，指令執行後存放結果的高32位。

- 指令範例：

<pre>SMLAL R0 , R1 , R2 , R3 ; R0 = (R2 x R3) 的低32位 + R0 ; R1 = (R2 x R3) 的高32位 + R1</pre>
--

ARM指令集-乘法指令與乘加指令- UMULL指令

- **UMULL指令的語法為：**
 - **UMULL{條件}{S}** 目的暫存器Low，目的暫存器High，運算元1，運算元2
- **UMULL指令完成將運算元1與運算元2的乘法運算，並把結果的低32位元放置到目的暫存器Low中，結果的高32位元放置到目的暫存器High中，同時可以根據運算結果設定CPSR中相應的條件旗標位元。其中，運算元1和運算元2均為32位元的無符號數。**
- **指令範例：**

```
UMULL R0 , R1 , R2 , R3 ; R0 = ( R2 x R3 ) 的低32位  
                        ; R1 = ( R2 x R3 ) 的高32位
```

ARM指令集-乘法指令與乘加指令- UMLAL指令

- **UMLAL指令的語法為：**
 - **UMLAL{條件}{S}** 目的暫存器Low，目的暫存器High，
運算元1，運算元2
- **UMLAL指令完成將運算元1與運算元2的乘法運算，並把結果的低32位元同目的暫存器Low中的值相加後又放置到目的暫存器Low中，結果的高32位元同目的暫存器High中的值相加後又放置到目的暫存器High中，同時可以根據運算結果設定CPSR中相應的條件旗標位元。其中，運算元1和運算元2均為32位元的無符號數。**
- **對於目的暫存器Low，在指令執行前存放64位元加數的低32位元，指令執行後存放結果的低32位。**
- **對於目的暫存器High，在指令執行前存放64位元加數的高32位元，指令執行後存放結果的高32位。**
- **指令範例：**

UMLAL R0 , R1 , R2 , R3 ; R0 = (R2 x R3) 的低32位 + R0
; R1 = (R2 x R3) 的高32位 + R1

ARM指令集-程式狀態暫存器存取指令

- **ARM微處理器支援程式狀態暫存器存取指令，用於在程式狀態暫存器和通用暫存器之間傳送資料，程式狀態暫存器存取指令包括以下兩條：**
 - **MRS** 程式狀態暫存器到通用暫存器的資料傳送指令。
 - **MSR** 通用暫存器到程式狀態暫存器的資料傳送指令。

ARM指令集-程式狀態暫存器存取指令- MRS指令

- **MRS指令的語法為：**
 - **MRS{條件} 通用暫存器，程式狀態暫存器（CPSR或SPSR）**
- **MRS指令用於將程式狀態暫存器的內容傳送到通用暫存器中。該指令一般用在以下幾種情況：**
- **當需要改變程式狀態暫存器的內容時，可用MRS將程式狀態暫存器的內容讀入通用暫存器，修改後再寫回程式狀態暫存器。**
- **當在例外處理或進程切換時，需要保存程式狀態暫存器的值，可先用該指令讀出程式狀態暫存器的值，然後保存。**
- **指令範例：**

MRS	R0, CPSR	; 傳送CPSR的內容到R0
MRS	R0, SPSR	; 傳送SPSR的內容到R0

ARM指令集-程式狀態暫存器存取指令-MSR指令

- **MSR指令的語法為：**
 - **MSR{條件} 程式狀態暫存器（CPSR或SPSR）_<域>，運算元**
- **MSR指令用於將運算元的內容傳送到程式狀態暫存器的特定域中。其中，運算元可以為通用暫存器或立即數。<域>用於設定程式狀態暫存器中需要操作的位元，32位元的程式狀態暫存器可分為4個域：**
 - **位[31：24]為條件旗標位元域，用f表示；**
 - **位[23：16]為狀態位元域，用s表示；**
 - **位[15：8]為擴充位域，用x表示；**
 - **位[7：0]為控制位域，用c表示；**
- **該指令通常用於恢復或改變程式狀態暫存器的內容，在使用時，一般要在MSR指令中指明將要操作的域。**
- **指令範例：**

MSR	CPSR , R0	；傳送R0的內容到CPSR
MSR	SPSR , R0	；傳送R0的內容到SPSR
MSR	CPSR_c , R0	；傳送R0的內容到SPSR，但僅僅修改CPSR中的控制位域

ARM指令集-載入/存儲指令

- **ARM微處理器支援載入/存儲指令用於在暫存器和記憶體之間傳送資料，載入指令用於將記憶體中的資料傳送到暫存器，存儲指令則完成相反的操作。常用的載入存儲指令如下：**
 - ❑ **LDR** 字資料載入指令
 - ❑ **LDRB** 位元組資料載入指令
 - ❑ **LDRH** 半字資料載入指令
 - ❑ **STR** 字資料存儲指令
 - ❑ **STRB** 位元組資料存儲指令
 - ❑ **STRH** 半字資料存儲指令

ARM指令集-載入/存儲指令- LDR指令

- LDR指令的語法為：

- LDR{條件} 目的暫存器，<記憶體位址>

- LDR指令用於從記憶體中將一個32位元的字資料傳送到目的暫存器中。該指令通常用於從記憶體中讀取32位元的字資料到通用暫存器，然後對資料進行處理。當程式計數器PC作為目的暫存器時，指令從記憶體中讀取的字資料被當作目的地址，從而可以實現程式流程的跳移。該指令在程式設計中比較常用，且定址方式靈活多樣，請讀者認真掌握。

- 指令範例：

LDR R0, [R1]	; 將記憶體位址為R1的字資料讀入暫存器R0。
LDR R0, [R1, R2]	; 將記憶體位址為R1+R2的字資料讀入暫存器R0。
LDR R0, [R1, # 8]	; 將記憶體位址為R1+8的字資料讀入暫存器R0。
LDR R0, [R1, R2] !	; 將記憶體位址為R1+R2的字資料讀入暫存器R0，並將新位址R1 + R2寫入R1。
LDR R0, [R1, # 8] !	; 將記憶體位址為R1+8的字資料讀入暫存器R0，並將新位址R1 + 8寫入R1。
LDR R0, [R1], R2	; 將記憶體位址為R1的字資料讀入暫存器R0，並將新位址R1 + R2寫入R1。
LDR R0, [R1, R2, LSL # 2] !	; 將記憶體位址為R1 + R2x4的字資料讀入暫存器R0，並將新位址R1 + R2x4寫入R1。
LDR R0, [R1], R2, LSL # 2	; 將記憶體位址為R1的字資料讀入暫存器R0，並將新位址R1 + R2x4寫入R1。

ARM指令集-載入/存儲指令-LDRB指令

- **LDRB指令的語法為：**

- **LDR{條件}B 目的暫存器，<記憶體位址>**

- **LDRB指令用於從記憶體中將一個8位元的位元組資料傳送到目的暫存器中，同時將暫存器的高24位清零。該指令通常用於從記憶體中讀取8位元的位元組資料到通用暫存器，然後對資料進行處理。當程式計數器PC作為目的暫存器時，指令從記憶體中讀取的字資料被當作目的地址，從而可以實現程式流程的跳移。**

- **指令範例：**

LDRB	R0, [R1]	; 將記憶體位址為R1的位元組資料讀入暫存器R0，並將R0的高24位清零。
LDRB	R0, [R1, # 8]	; 將記憶體位址為R1 + 8的位元組資料讀入暫存器R0，並將R0的高24位清零。

ARM指令集-載入/存儲指令-LDRH指令

■ LDRH指令的語法為：

■ LDR{條件}H 目的暫存器，<記憶體位址>

■ LDRH指令用於從記憶體中將一個16位元的半字資料傳送到目的暫存器中，同時將暫存器的高16位清零。

■ 該指令通常用於從記憶體中讀取16位元的半字資料到通用暫存器，然後對資料進行處理。當程式計數器PC作為目的暫存器時，指令從記憶體中讀取的字資料被當作目的地址，從而可以實現程式流程的跳移。

■ 指令範例：

LDRH	R0, [R1]	; 將記憶體位址為R1的半字資料讀入暫存器R0，並將R0的高16位清零。
LDRH	R0, [R1, #8]	; 將記憶體位址為R1 + 8的半字資料讀入暫存器R0，並將R0的高16位清零。
LDRH	R0, [R1, R2]	; 將記憶體位址為R1 + R2的半字資料讀入暫存器R0，並將R0的高16位清零。

ARM指令集-載入/存儲指令-STR指令

- **STR指令的語法為：**
 - **STR{條件} 來源暫存器，<記憶體位址>**
- **STR指令用於從來源暫存器中將一個32位元的字資料傳送到記憶體中。該指令在程式設計中比較常用，且定址方式靈活多樣，使用方式可參考指令LDR。**
- **指令範例：**

STR	R0, [R1], #8	; 將R0中的字資料寫入以R1為位址的記憶體中，並將新位址R1 + 8寫入R1。
STR	R0, [R1, #8]	; 將R0中的字資料寫入以R1 + 8為位址的記憶體中。

ARM指令集-載入/存儲指令-STRB指令

- STRB指令的語法為：
 - STR{條件}B 來源暫存器，<記憶體位址>
- STRB指令用於從來源暫存器中將一個8位元的位元組資料傳送到記憶體中。該位元組資料為來源暫存器中的低8位。
- 指令範例：

STRB	R0 , [R1]	；將暫存器R0中的位元組資料寫入以R1為位址的記憶體中。
STRB	R0 , [R1 , # 8]	；將暫存器R0中的位元組資料寫入以R1 + 8為位址的記憶體中。

ARM指令集-載入/存儲指令-STRH指令

- STRH指令的語法為：
 - STR{條件}H 來源暫存器，<記憶體位址>
- STRH指令用於從來源暫存器中將一個16位元的半字資料傳送到記憶體中。該半字資料為來源暫存器中的低16位。
- 指令範例：

STRH	R0 , [R1]	; 將暫存器R0中的半字資料寫入以R1為位址的記憶體中。
STRH	R0 , [R1 , # 8]	; 將暫存器R0中的半字資料寫入以R1 + 8為位址的記憶體中。

ARM指令集-連續資料載入/存儲指令

- ARM微處理器所支援連續資料載入/存儲指令可以一次在一片連續的記憶體單元和多個暫存器之間傳送資料，連續載入指令用於將一片連續的記憶體中的資料傳送到多個暫存器，連續資料存儲指令則完成相反的操作。常用的載入存儲指令如下：

- LDM 連續資料載入指令
- STM 連續資料存儲指令

ARM指令集-連續資料載入/存儲指令

- **LDM (或STM) 指令的語法為：**
 - **LDM (或STM) {條件}{類型} 基址暫存器{!}, 暫存器列表{ }**
- **LDM (或STM) 指令用於從由基址暫存器所指示的一片連續記憶體到暫存器列表所指示的多個暫存器之間傳送資料，該指令的常見用途是將多個暫存器的內容入堆疊或出堆疊。其中，{類型}為以下幾種情況：**
- **非堆疊型定址**
 - **IA (Increment After) :**
使得基底暫存器在存取後才增加
 - **IB (Increment Before) :**
使得基底暫存器在存取前即增加
 - **DA (decrement After)**
使得基底暫存器在存取後才減少
 - **DB (decrement Before)**
使得基底暫存器在存取前即減少
- **堆疊型定址 (應用於程式堆疊資料結構上)**
 - **FD(Full Descending), 滿遞減**
 - **FA(Full Ascending), 滿遞增**
 - **ED(Empty Descending), 空遞減**
 - **EA(Empty Ascending), 空遞減**

ARM指令集-連續資料載入/存儲指令

■ 堆疊型定址（應用於程式堆疊資料結構上）

- **FULL**：指標指向在堆疊中已填入資料的最後位置
- **EMPTY**：指標指向在堆疊中下一筆資料將要填入的位置
- **Descending**：指標從較高位址開始，當堆疊內的紀錄的資料增加時，指標往低位址的地方減少
- **Ascending**：指標從較低位址開始，當堆疊內的紀錄的資料增加時，指標往高位址的地方增加

■ 尾碼

- 基址暫存器不允許為R15，暫存器列表可以為R0 R14的任意組合。
- **{!}**為可選尾碼，
 - 若選用該尾碼，則當資料傳送完畢之後，將最後的位址寫入基址暫存器
 - 否則基址暫存器的內容不改變。
- **{ }**為可選尾碼，
 - 當指令為LDM且暫存器列表中包含R15，選用該尾碼時表示：除了正常的資料傳送之外，還將SPSR複製到CPSR。
 - 同時，該尾碼還表示傳入或傳出的是用戶模式下的暫存器，而不是當前模式下的暫存器。

ARM指令集-連續資料載入/存儲指令

■ 指令範例：

STMFD R13!, {R0, R4-R12, LR} ; 將暫存器列表中的暫存器 (R0, R4到R12, LR) 存入堆疊。
LDMFD R13!, {R0, R4-R12, PC} ; 將堆疊內容恢復到暫存器 (R0, R4到R12, LR)。

ARM指令集-連續資料載入/存儲指令

Increment After

```
ldmia r12, {r2, r3, r5, r10}
```

```
ldmia r12!, {r2, r3, r5, r10} = ldmfd r12!, {r2, r3, r5, r10}
```

IA

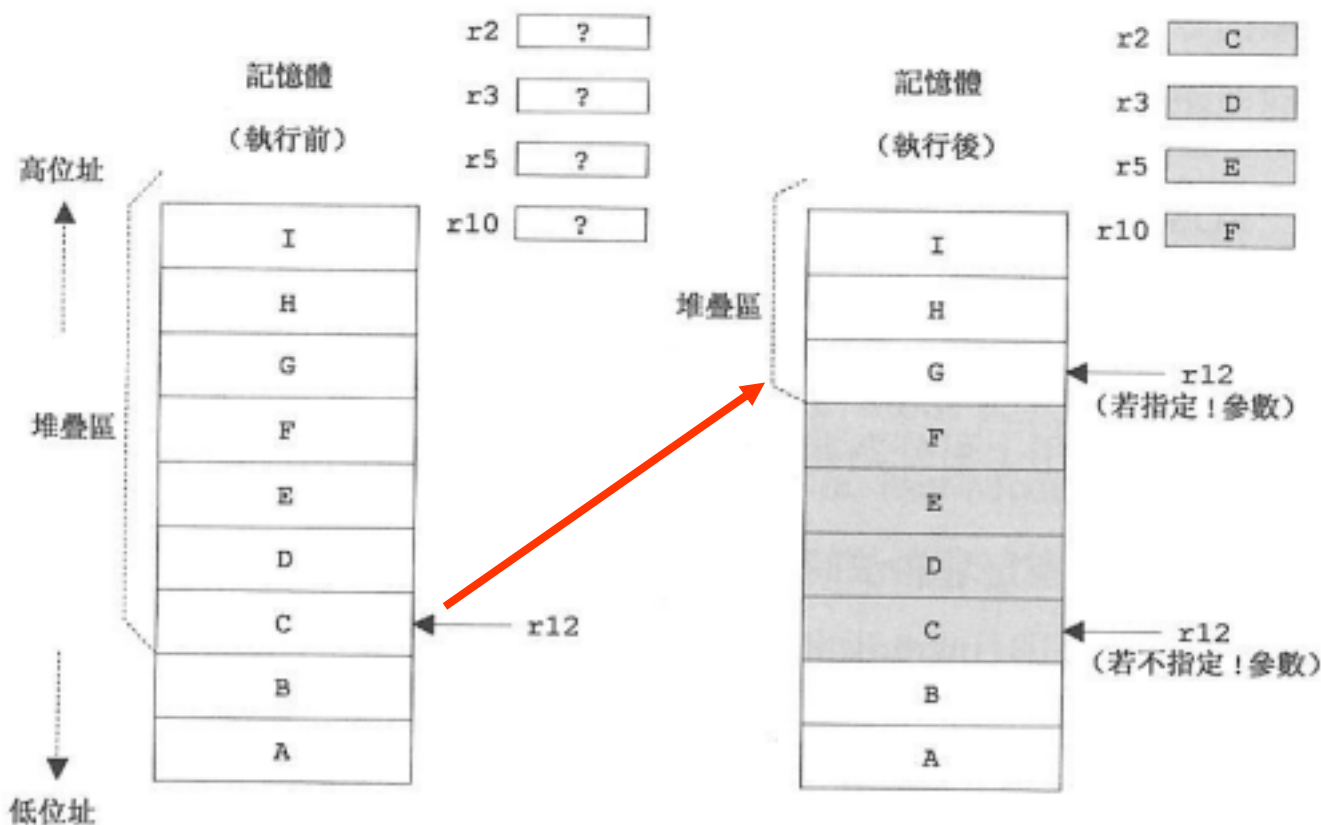


圖 4.2.5 多重載入的定址-IA

ARM指令集-連續資料載入/存儲指令

Increment After

```
stmia r12, {r2, r3, r5, r10}
stmia r12!, {r2, r3, r5, r10} = stmea r12!, {r2, r3, r5, r10}
```

IA

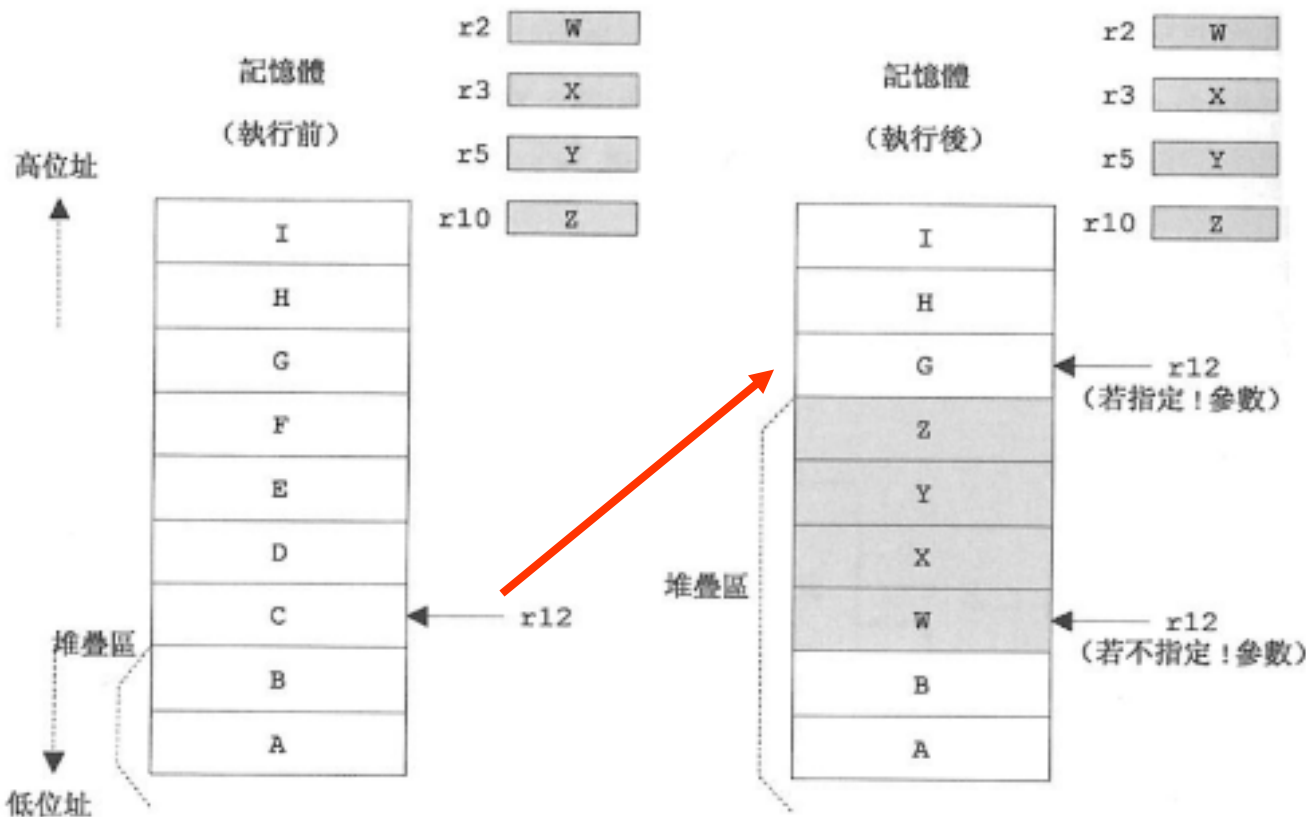
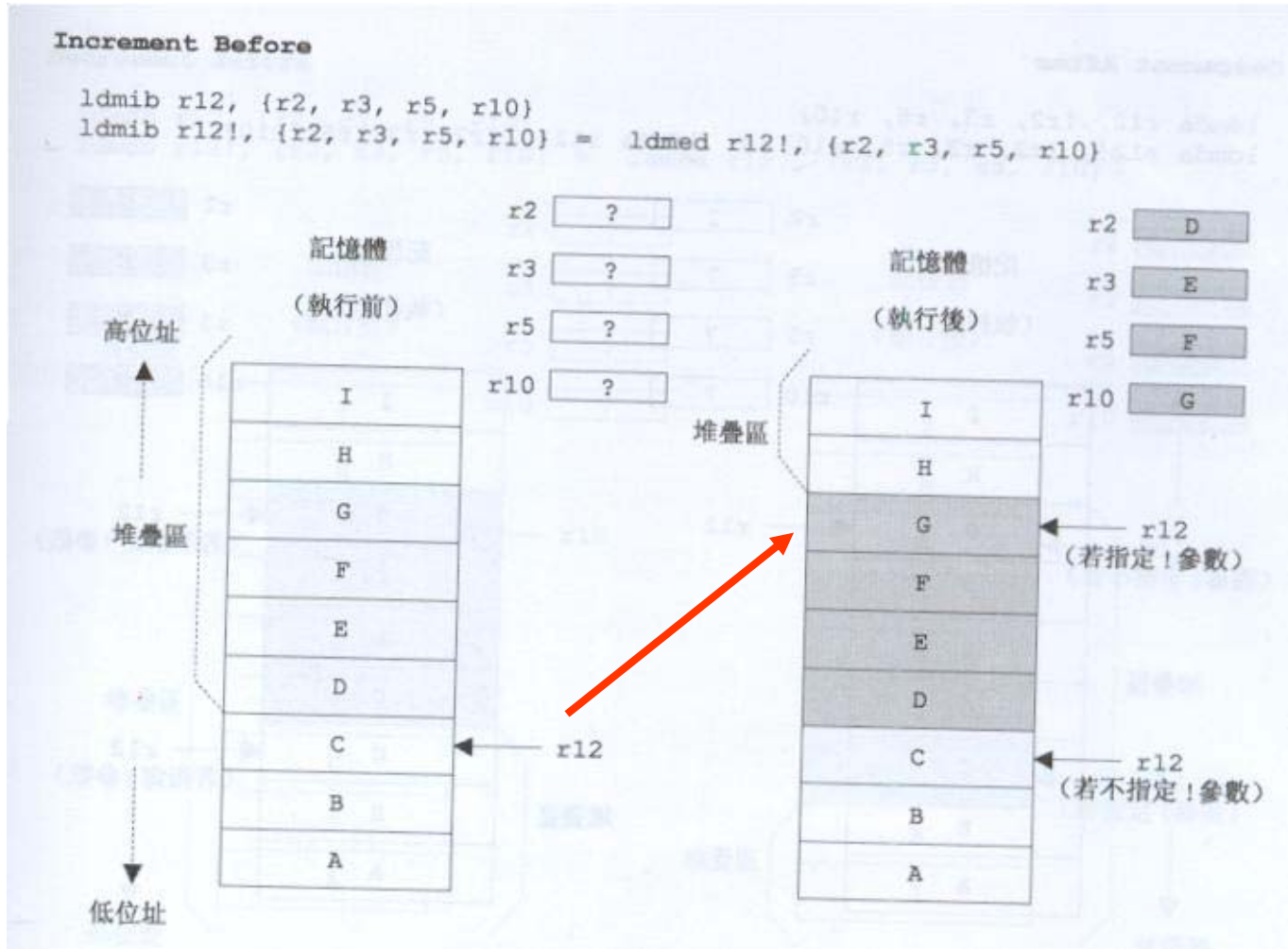


圖 4.2.6 多重存回的定址-IA

ARM指令集-連續資料載入/存儲指令

IB



ARM指令集-連續資料載入/存儲指令

IB

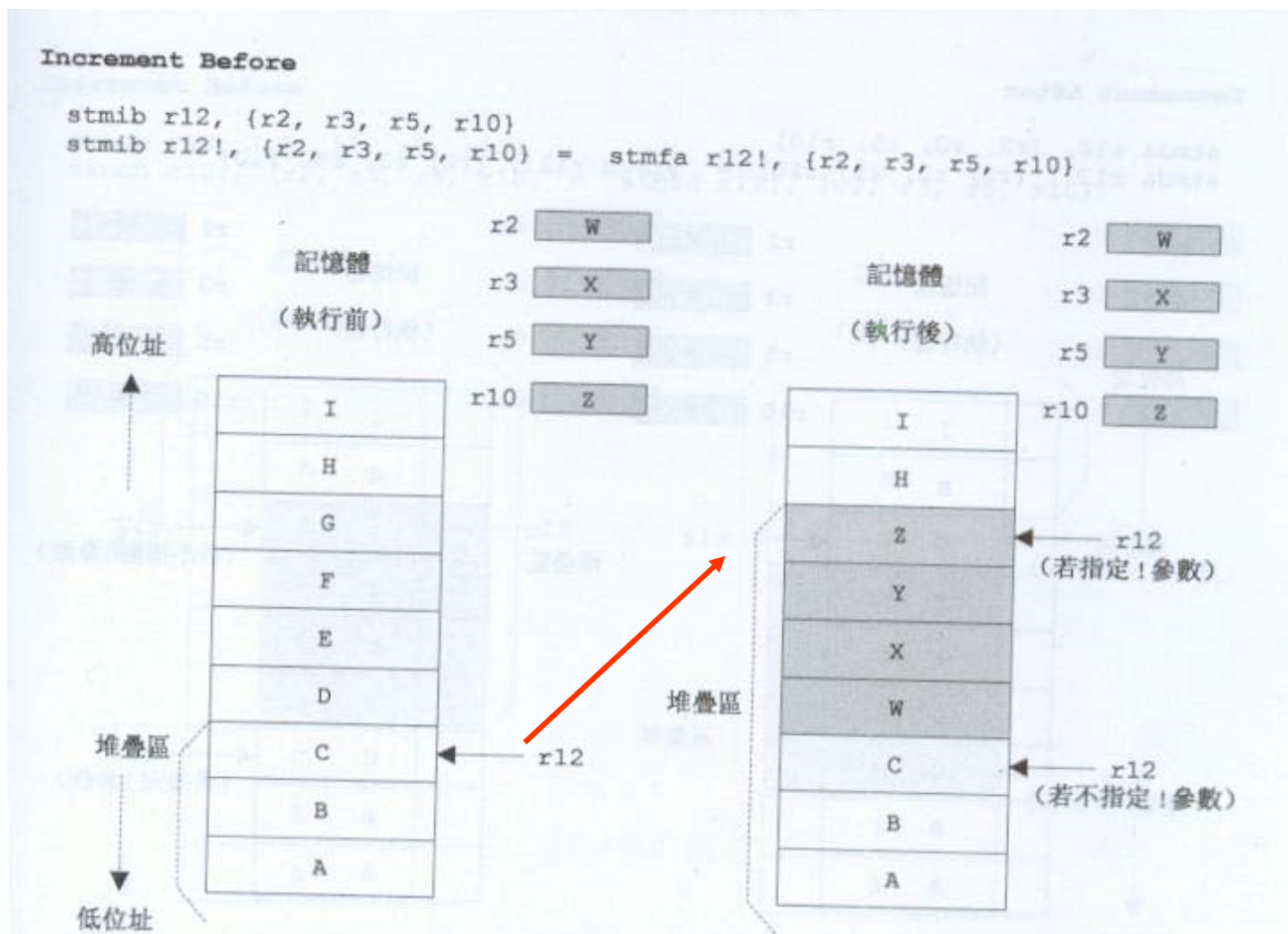


圖 4.2.8 多重存回的定址-IB

ARM指令集-連續資料載入/存儲指令

DA

Decrement After

```
ldmda r12, {r2, r3, r5, r10}
```

```
ldmda r12!, {r2, r3, r5, r10} = ldmfa r12!, {r2, r3, r5, r10}
```

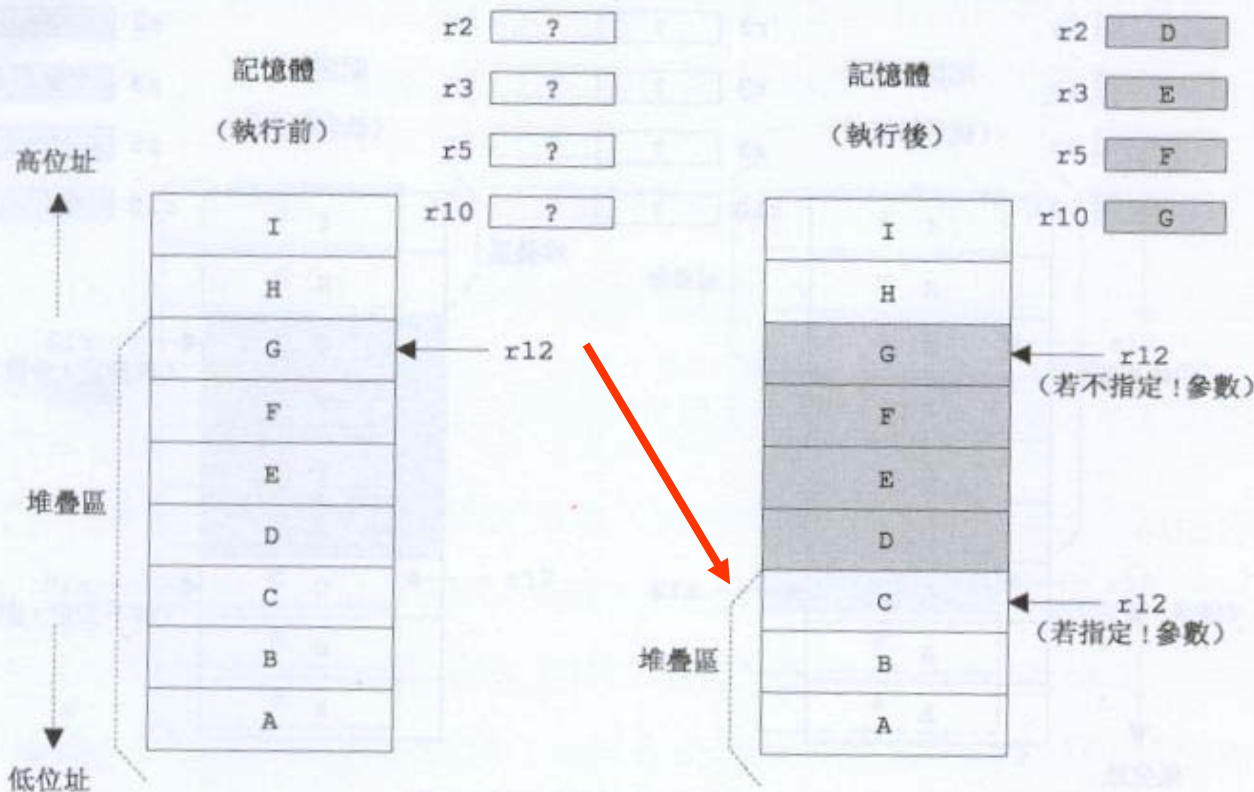


圖 4.2.9 多重載入的定址-DA

ARM指令集-連續資料載入/存儲指令

DA

Decrement After

```
stmda r12, {r2, r3, r5, r10}
stmda r12!, {r2, r3, r5, r10} = stmed r12!, {r2, r3, r5, r10}
```

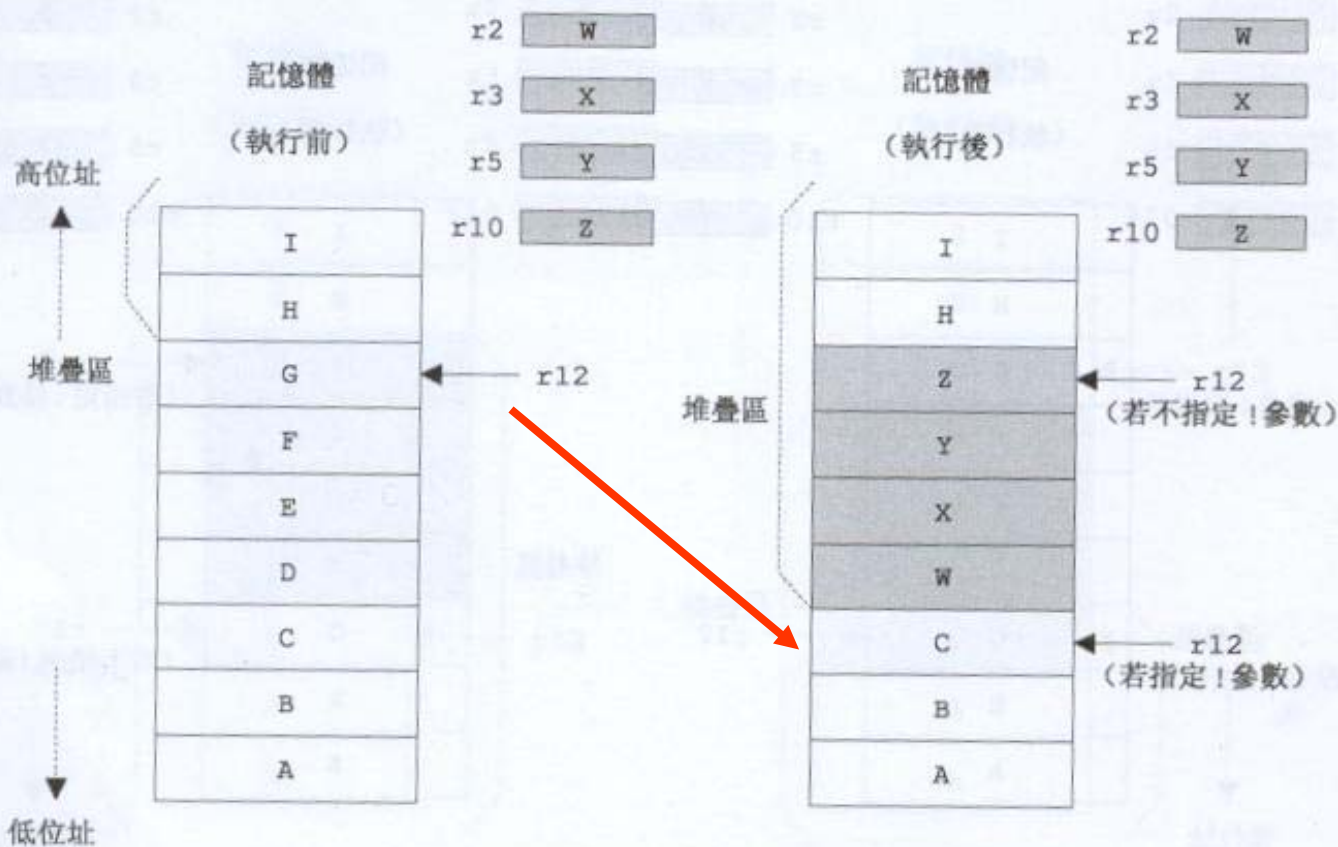


圖 4.2.10 多重存回的定址-DA

ARM指令集-連續資料載入/存儲指令

DB

Decrement Before

```
ldmdb r12, {r2, r3, r5, r10}  
ldmdb r12!, {r2, r3, r5, r10} = ldmea r12!, {r2, r3, r5, r10}
```

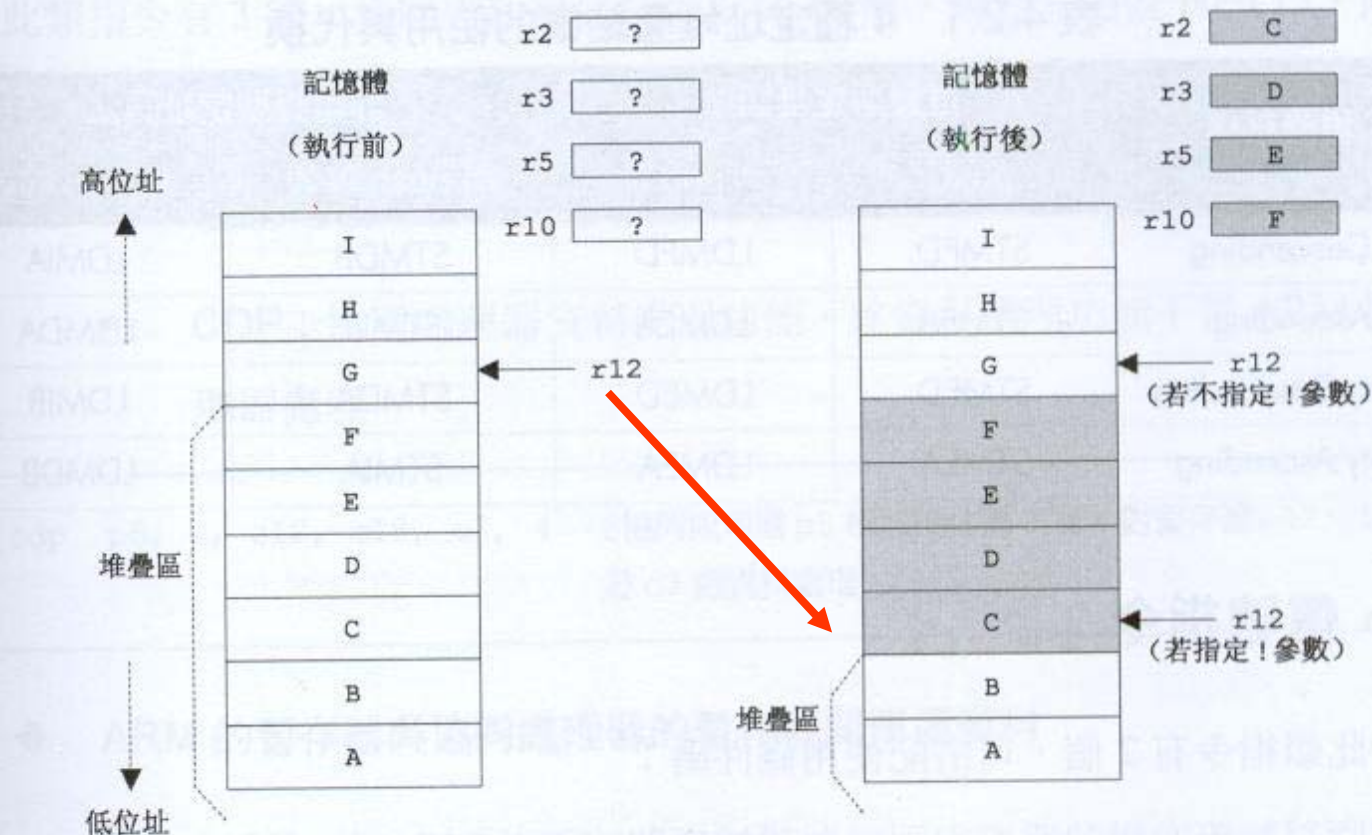


圖 4.2.11 多重載入的定址-DB

ARM指令集-連續資料載入/存儲指令

DB

Decrement Before

```
stmdb r12, {r2, r3, r5, r10}  
stmdb r12!, {r2, r3, r5, r10} = stmfd r12!, {r2, r3, r5, r10}
```

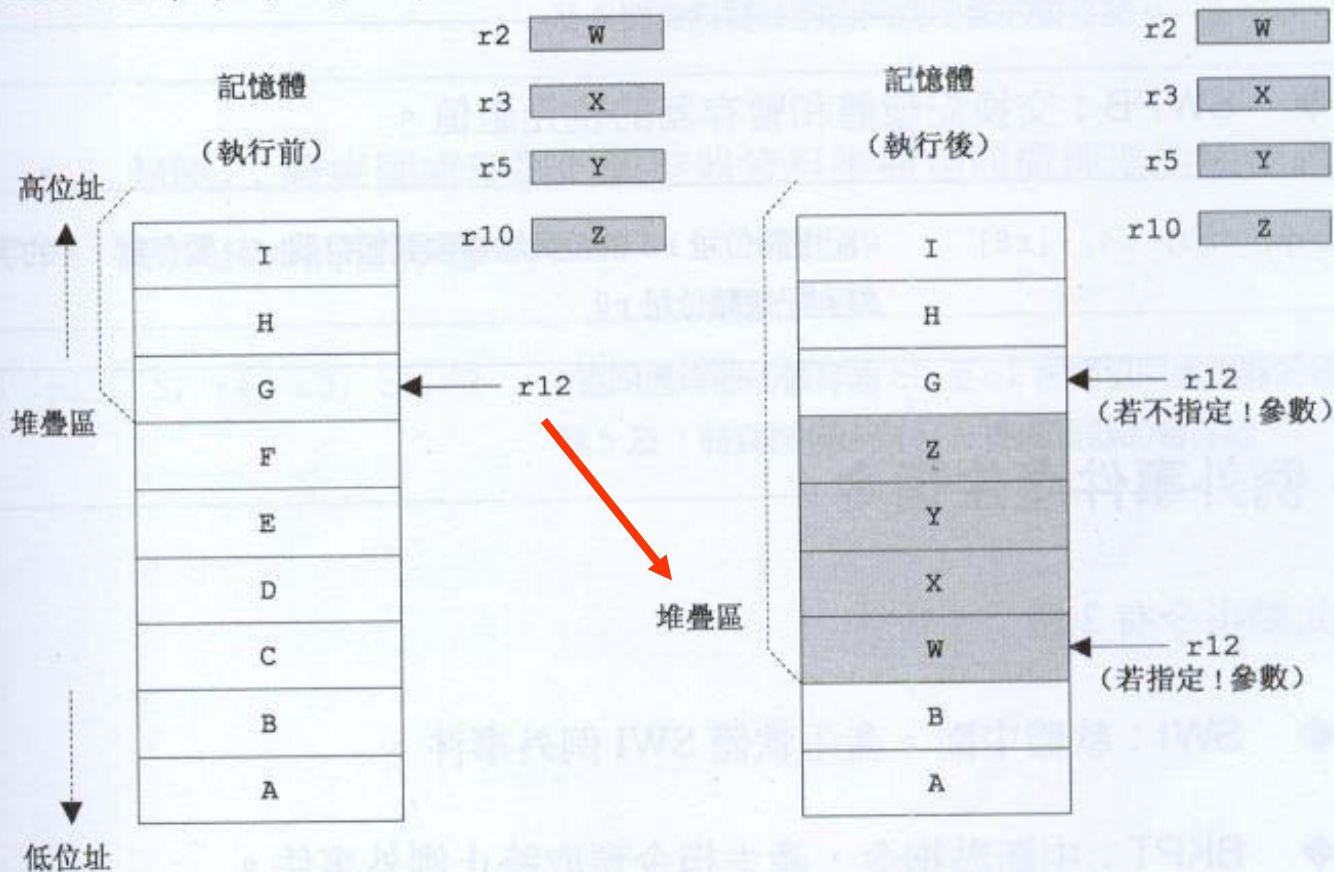


圖 4.2.12 多重存回的定址-DB

ARM指令集-資料交換指令

- ARM微處理器所支援資料交換指令能在記憶體和暫存器之間交換資料。資料交換指令有如下兩條：
 - SWP 字資料交換指令
 - SWPB 位元組資料交換指令

ARM指令集-資料交換指令-SWP指令

- **SWP指令的語法為：**
 - **SWP{條件} 目的暫存器，來源暫存器1，[來源暫存器2]**
- **SWP指令用於將來源暫存器2所指向的記憶體中的字資料傳送到目的暫存器中，同時將來源暫存器1中的字資料傳送到來源暫存器2所指向的記憶體中。顯然，當來源暫存器1和目的暫存器為同一個暫存器時，指令交換該暫存器和記憶體的內容。**
- **指令範例：**

SWP R0, R1, [R2] ; 將R2所指向的記憶體中的字資料傳送到R0，同時將R1中的字資料傳送到R2所指向的存儲單元。
SWP R0, R0, [R1] ; 該指令完成將R1所指向的記憶體中的字資料與R0中的字資料交換。

ARM指令集-資料交換指令-SWPB指令

- **SWPB指令的語法為：**
 - **SWP{條件}B** 目的暫存器，來源暫存器1，[來源暫存器2]
- **SWPB指令用於將來源暫存器2所指向的記憶體中的位元組資料傳送到目的暫存器中，目的暫存器的高24位清零，同時將來源暫存器1中的位元組資料傳送到來源暫存器2所指向的記憶體中。顯然，當來源暫存器1和目的暫存器為同一個暫存器時，指令交換該暫存器和記憶體的內容。**
- **指令範例：**

SWPB R0, R1, [R2] ; 將R2所指向的記憶體中的位元組資料傳送到R0，R0的高24位清零，同時將R1中的低8位元資料傳送到R2所指向的存儲單元。
SWPB R0, R0, [R1] ; 該指令完成將R1所指向的記憶體中的位元組資料與R0中的低8位元資料交換。

ARM指令集-移位元指令

- ARM微處理器內嵌的桶型移位器（Barrel Shifter），支援資料的各種移位元操作，移位元操作在ARM指令集中不作為單獨的指令使用，它只能作為指令格式中是一個欄位，在組合語言中表示為指令中的選項。
- 例如，資料處理指令的第二個運算元為暫存器時，就可以加入移位元操作選項對它進行各種移位操作。
- 移位操作包括如下6種類型，ASL和LSL是等價的，可以自由互換：
 - LSL 邏輯左移
 - ASL 算術左移
 - LSR 邏輯右移
 - ASR 算術右移
 - ROR 迴圈右移
 - RRX 帶擴充的迴圈右移

ARM指令集-移位元指令-LSL（或ASL）操作

- LSL（或ASL）操作的格式為：
 - 通用暫存器，LSL（或ASL）運算元
- LSL（或ASL）可完成對通用暫存器中的內容進行邏輯（或算術）的左移操作，按運算元所指定的數量向左移位，低位用零來填充。其中，運算元可以是通用暫存器，也可以是立即數（0 31）。
- 操作範例：

MOV R0, R1, LSL#2

；將R1中的內容左移兩位元後傳送到R0中。

ARM指令集-移位元指令-LSR操作

- **LSR操作的格式為：**
 - **通用暫存器，LSR 運算元**
- **LSR可完成對通用暫存器中的內容進行右移的操作，按運算元所指定的數量向右移位，左端用零來填充。其中，運算元可以是通用暫存器，也可以是立即數（0 31）。**
- **操作範例：**

MOV R0, R1, LSR#2 ; 將R1中的內容右移兩位元後傳送到R0中，左端用零來填充。

ARM指令集-移位元指令-ASR操作

- **ASR操作的格式為：**
 - 通用暫存器，ASR 運算元
- **ASR可完成對通用暫存器中的內容進行右移的操作，按運算元所指定的數量向右移位，左端用第31位的值來填充。其中，運算元可以是通用暫存器，也可以是立即數（0 31）。**
- **操作範例：**

MOV R0, R1, ASR#2 ; 將R1中的內容右移兩位元後傳送到R0中，左端用第31位的值來填充。

ARM指令集-移位元指令-ROR操作

- ROR操作的格式為：
 - 通用暫存器，ROR 運算元
- ROR可完成對通用暫存器中的內容進行迴圈右移的操作，按運算元所指定的數量向右迴圈移位，左端用右端移出的位來填充。其中，運算元可以是通用暫存器，也可以是立即數（0 31）。顯然，當進行32位的迴圈右移操作時，通用暫存器中的值不改變。
- 操作範例：

```
MOV    R0, R1, ROR#2    ; 將R1中的內容迴圈右移兩位元後傳送到R0中。
```

ARM指令集-移位元指令-RRX操作

- **RRX操作的格式為：**
 - **通用暫存器，RRX 運算元**
- **RRX可完成對通用暫存器中的內容進行帶擴充的迴圈右移的操作，按運算元所指定的數量向右迴圈移位，左端用進位元旗標位元C來填充。其中，運算元可以是通用暫存器，也可以是立即數（0 31）。**
- **操作範例：**

MOV R0, R1, RRX#2 ; 將R1中的內容進行帶擴充的迴圈右移兩位後傳送到R0中。

ARM指令集-輔助運算器指令

- ARM微處理器可支援多達16個輔助運算器，用於各種協處理操作，在程式執行的過程中，每個輔助運算器只執行針對自身的協處理指令，忽略ARM處理器和其他輔助運算器的指令。
- ARM的輔助運算器指令主要用於ARM處理器初始化ARM輔助運算器的資料處理操作，以及在ARM處理器的暫存器和輔助運算器的暫存器之間傳送資料，和在ARM輔助運算器的暫存器和記憶體之間傳送資料。ARM輔助運算器指令包括以下5條：
 - CDP 輔助運算器數操作指令
 - LDC 輔助運算器資料載入指令
 - STC 輔助運算器資料存儲指令
 - MCR ARM處理器暫存器到輔助運算器暫存器的資料傳送指令
 - MRC 輔助運算器暫存器到ARM處理器暫存器的資料傳送指令

ARM指令集-輔助運算器指令- CDP指令

- CDP指令的語法為：

- CDP{條件} 輔助運算器編碼，輔助運算器操作碼1，目的暫存器，來源暫存器1，來源暫存器2，輔助運算器操作碼2。

- CDP指令用於ARM處理器通知ARM輔助運算器執行特定的操作,若輔助運算器不能成功完成特定的操作，則產生未定義指令例外。其中輔助運算器操作碼1和輔助運算器操作碼2為輔助運算器將要執行的操作，目的暫存器和來源暫存器均為輔助運算器的暫存器，指令不涉及ARM處理器的暫存器和記憶體。

- 指令範例：

CDP P3 , 2 , C12 , C10 , C3 , 4

；該指令完成輔助運算器P3的初始化

ARM指令集-輔助運算器指令-LDC指令

- LDC指令的語法為：
 - LDC{條件}{L} 輔助運算器編碼,目的暫存器, [來源暫存器]
- LDC指令用於將來源暫存器所指向的記憶體中的字資料傳送到目的暫存器中，若輔助運算器不能成功完成傳送操作，則產生未定義指令例外。其中，{L}選項表示指令為長讀取操作，如用於雙精度資料的傳輸。
- 指令範例：

LDC P3, C4, [R0] ; 將ARM處理器的暫存器R0所指向的記憶體中的字資料傳送到輔助運算器P3的暫存器C4中。

ARM指令集-輔助運算器指令-STC指令

- STC指令的語法為：
 - STC{條件}{L} 輔助運算器編碼,來源暫存器 , [目的暫存器]
- STC指令用於將來源暫存器中的字資料傳送到目的暫存器所指向的記憶體中，若輔助運算器不能成功完成傳送操作，則產生未定義指令例外。其中，{L}選項表示指令為長讀取操作，如用於雙精度資料的傳輸。
- 指令範例：

STC P3, C4, [R0] ; 將輔助運算器P3的暫存器C4中的字資料傳送到ARM處理器的暫存器R0所指向的記憶體中。

ARM指令集-輔助運算器指令-MCR指令

- **MCR指令的語法為：**

- **MCR{條件} 輔助運算器編碼，輔助運算器操作碼1，來源暫存器，目的暫存器1，目的暫存器2，輔助運算器操作碼2。**

- **MCR指令用於將ARM處理器暫存器中的資料傳送到輔助運算器暫存器中,若輔助運算器不能成功完成操作，則產生未定義指令例外。其中輔助運算器操作碼1和輔助運算器操作碼2為輔助運算器將要執行的操作，來源暫存器為ARM處理器的暫存器，目的暫存器1和目的暫存器2均為輔助運算器的暫存器。**

- **指令範例：**

MCR P3, 3, R0, C4, C5, 6 ; 該指令將ARM處理器暫存器R0中的資料傳送到輔助運算器P3的暫存器C4和C5中。

ARM指令集-輔助運算器指令-MRC指令

- **MRC指令的語法為：**

- **MRC{條件} 輔助運算器編碼，輔助運算器操作碼1，目的暫存器，來源暫存器1，來源暫存器2，輔助運算器操作碼2。**

- **MRC指令用於將輔助運算器暫存器中的資料傳送到ARM處理器暫存器中,若輔助運算器不能成功完成操作，則產生未定義指令例外。其中輔助運算器操作碼1和輔助運算器操作碼2為輔助運算器將要執行的操作，目的暫存器為ARM處理器的暫存器，來源暫存器1和來源暫存器2均為輔助運算器的暫存器。**

- **指令範例：**

MRC P3, 3, R0, C4, C5, 6 ; 該指令將輔助運算器P3的暫存器中的資料傳送到ARM處理器暫存器中。

ARM指令集-例外產生指令

- ARM微處理器所支援的例外指令有如下兩條：
 - SWI 軟體中斷指令
 - BKPT 中斷點中斷指令

ARM指令集-例外產生指令-SWI指令

- **SWI指令的語法為：**
 - **SWI{條件} 24位的立即數**
- **SWI指令用於產生軟體中斷，以便用戶程式能使用作業系統的系統常式。作業系統在SWI的例外處理程式中提供相應的系統服務，指令中24位元的立即數指定用戶程式使用系統常式的類型，相關參數通過通用暫存器傳遞，當指令中24位元的立即數被忽略時，用戶程式使用系統常式的類型由通用暫存器R0的內容決定，同時，參數通過其他通用暫存器傳遞。**
- **指令範例：**

SWI 0x02 ; 該指令使用作業系統編號位元02的系統常式。

ARM指令集-例外產生指令-BKPT指令

- **BKPT指令的語法為：**
 - **BKPT** 16位的立即數
- **BKPT指令產生軟體中斷點中斷，可用於程式的除錯。**