微處理機系統
(Microprocessor System)

第三章
ARM 微處理器的指令集

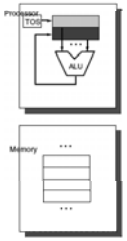
Ping-Liang Lai (賴秉樑)

章節大綱

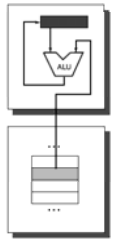
3.1 ARM 微處理器的指令集概述

- ARM 指令集為載入/存回（Load/Store）架構（Ex: C=A+B for four classes）
  - 也即指令集僅能處理暫存器中的資料，而且處理結果都要再放回暫存器中。

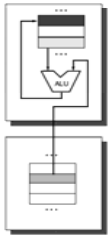
- 堆疊定址
  ```
  Push  A
  Push  B
  Add
  ```
  - Pop the top-2 values of the stack (A, B) and push the result value into the stack
  ```
  Pop   C
  ```



- 累加器定址
  ```
  Load  A
  Add   B
  ```
  - Add AC (A) with B and store the result into AC
  ```
  Store C
  ```



- 暫存器定址 (register-memory)
  ```
  Load    R1, A
  Add     R3, R1, B
  Store   R3, C
  ```



- 暫存器定址 (load-store)
  ```
  Load    R1, A
  Load    R2, B
  Add     R3, R1, R2
  Store   R3, C
  ```

指令集概述

- ARM 微處理器的指令集可分為跳躍指令、資料處理指令、程式狀態暫存器 (PSR) 處理指令、載入/存回指令、協同處理器指令和例外事件產生指令等，共六大類。

指令符號	指令功能描述
ADC	Add with Carry，包含旗標位元的算術加法指令
ADD	Add，算術加法指令
AND	Logical AND，邏輯AND運算指令
B	Branch，跳躍指令
BIC	Bit Clear，位元清除指令
BL	Branch with Link，包含返回的跳躍指令，也即是呼叫指令
BLX	Branch and Exchange Instruction Set，包含返回和狀態切換的跳躍指令
BX	Branch with Link and Exchange Instruction Set，包含狀態切換的跳躍指令
CDP	Coprocessor data processing，協同處理器資料操作指令
CMN	Compare negative，測試算術加法運算結果指令
CMP	Compare，比較指令，測試算術減法運算結果指令
EOR	Logical Exclusive OR，邏輯XOR運算指令

LDC	Load coprocessor from memory，記憶體到協同處理器的資料傳輸指令
LDM	Load multiple registers，連續載入多個暫存器資料指令
LDR	Load register from memory，記憶體到暫存器的資料傳輸指令
MCR	Move CPU register to coprocessor register，從ARM暫存器到協同處理器暫存器的資料傳輸指令
MLA	Multiply accumulate，乘加運算指令
MOV	Move，資料傳送指令
MRC	Move from coprocessor register to CPU register，從協同處理器暫存器到ARM暫存器的資料傳輸指令
MRS	Move PSR status/flags to register，傳送CPSR或SPSR的狀態旗標到通用暫存器指令
MUL	Multiply，32位元乘法指令
MLA	Move negative register，32位元乘加指令
MVN	Move Not，資料取反相後傳送指令
ORR	OR，邏輯OR運算指令
RSB	Reverse Subtract，被減數與減數角色互換的算術減法運算指令
RSC	Reverse subtract with carry，包含借位的逆向減法指令
SBC	Subtract with Carry，包含進位的算術減法指令
STC	Store coprocessor register to memory，協同處理器暫存器的寫入記憶體指令
STM	Store multiple，連續存回多筆暫存器資料指令
STR	Store register to memory，暫存器到記憶體的資料傳輸指令
SUB	Subtract，減法指令
SWI	Software Interrupt，軟體中斷指令
SWP	Swap register with memory，交換指令
TEQ	Test bit-wise equality，相等測試指令
TST	Test Bit，位元測試指令

章節大綱

- 3.1 ARM 微處理器的指令集概述
- 3.2 ARM 指令的定址模式
- 3.3 ARM 指令集
  - 程式狀態暫存器
  - 例外事件
  - 跳躍指令與資料處理指令
  - 乘法指令與乘加指令
  - 程式狀態暫存器 (PSR) 存取指令

3.2 ARM 指令的定址模式

- 所謂的定址方式就是微處理器根據指令中所給予的位址訊息來尋找出實體位址的方式。
- ARM 指令的定址模式
  - 立即定址
  - 暫存器定址
  - 暫存器間接定址
  - 基底定址
  - 相對定址
  - 多暫存器定址
  - 堆疊定址

3.2 定址模式 – 立即定址

- 立即定址: 運算元本身就在指令中直接加以設定，只要取出指令也就取到的運算元。

```
ADD   R3, R3, #1          ;R3 ← R3+1
ADD   R8, R7, #&ff        ;R8 ← R7[7:0]
```

  - 第二個來源運算元即為立即數值，並要求以 “#” 為首碼;
  - 對於十六進制表示的立即數值，在 “#” 後加上 “0x” 或 “&”;
  - 指令1: 完成 R3 暫存器的內容加1，將結果放回 R3 中;
  - 指令2: 將32-bit的 R7 取其低8-bit的數值，也即是作 AND 邏輯運算，然後將結果傳至 R8 中。

3.2 定址模式 – 暫存器定址

- 暫存器定址: 利用暫存器中的數值作為運算元，這種定址方式是各類處理器經常採用的方式，也是執行效率較高的定址方式。

```
ADD   R0, R1, R2     ;R0 ← R1+R2
```

  - 指令中的定址碼即為暫存器編號，暫存器內容為運算元;
  - 第一個是目的 (結果) 暫存器，第二個是來源 (運算元) 暫存器，第三個是來源 (運算元) 暫存器;
  - 該指令將暫存器 R1 和 R2 的內容相加，其結果存放在暫存器 R0 中。

3.2 定址模式 – 暫存器間接定址

- 暫存器間接定址: 以暫存器中的數值作為運算元的位址，而運算元本身是存放在記憶體中。

```
ADD   R0, R1, [R2]     ; R0 ← R1+[R2]
LDR   R0, [R1]         ; R0 ← [R1]
STR   R0, [R1]         ; [R1] ← R0
```

  - 指令中的定址碼設定一通用的暫存器編號，被指定的暫存器中存放運算元的有效位址;
    - » 運算元則存放在記憶體單元中，暫存器即為位址指標;
    - » 暫存器間接定址使用一個暫存器（基底暫存器）的數值作為記憶體的位址。
  - 指令1: 以暫存器 R2 的數值作為運算元的位址，在記憶體中取得一個運算元後與 R1 相加，結果存入暫存器 R0 中;
  - 指令2: 將 R1 所指向的記憶體單元的內容載入至 R0;
  - 指令3: 將 R0 取回至 R1 所指向的記憶體單元中。

3.2 定址模式 – 基底定址 (1/3)

- 基底定址: 將暫存器 (該暫存器稱之為基底暫存器) 的內容與指令中所給予的位址偏移量加以相加，並進而得到一個運算元的有效位址。

```
LDR   R0, [R1, #4]   ;R0 ← [R1+4]
```

  - 基底定址是用來處理基底附近的記憶體，其中包含 2 種定址，
    基底加偏移量： 前索引與後索引定址 (基底加索引定址)
  - 指令1: 這是一種前索引的定址
  - 暫存器間接定址則是偏移量為0的基底加上偏移定址的方式;
  - 但基底加偏移定址中的基底暫存器包含的並不是確定的位址，基底需加（減）最大 4KB 的偏移來計算出所要處理的位址，也就是前索引與後索引定址的計算。

3.2 定址模式 – 基底定址 (2/3)

- 此外，除了找到基底定址所指向的記憶體資料外，還可順便改變這基底暫存器。

```
LDR   R0, [R1, #4] ! ;R0 ← [R1+4]，R1 ← R1+4
```

  - 改變基底暫存器來指向下一個所傳送的位址，這對於多筆資料傳送很有用的;
  - 其中，"!"表示指令在完成資料傳送後，同時更新基底暫存器;
  - ARM 微處理器對這種自動索引的方式，不消耗額外的週期。

3.2 定址模式 – 基底定址 (3/3)

- 後索引定址: 這是一種基底不包含偏移量來做為傳送的位址，且再傳送後，自動加上索引的方式。

```
LDR   R0, [R1], #4   ;R0 ← [R1]，R1 ← R1+4
```

  - 沒有使用 "!"，只用了立即數值的偏移量來作為基底暫存器的變化量。
- 此外，基底加上索引定址的方式是在指令中指定一個暫存器為基底，然後再指定另一個暫存器當索引。

```
LDR   R0, [R1, R2]   ;R0 ← [R1+R2]
```


3.2 定址模式 – 相對定址

- 相對定址: 以程式計數器 PC 的目前數值作為基底位址，指令中的位址標號作為偏移量，再將兩者相加之後得到有效位址。

```
BL      NEXT          ;跳躍到副程式 NEXT 處執行
…..
NEXT
…..
MOV   PC, LR          ;從副程式返回
```

  - 上述程式完成了副程式的跳躍與返回;
  - 其中，跳躍指令 BL 採用了相對定址的方式。

3.2 定址模式 – 多暫存器定址

- 多暫存器定址: 一個指令可以完成多個暫存器值的傳送。

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

  - 最多傳送 16 個暫存器;
  - 該指令的字尾 IA 表示在每次執行完載入/存回 (Load/Store) 操作後，R0 按字元組長度增加，因此，這個指令可將連續記憶體單元的數值傳送到 R1 ~ R4。

3.2 定址模式 – 堆疊定址

- 堆疊是按照特定順序進行存取的記憶體區塊;
- 透過後進先出（LIFO）或是先進後出（FILO）的順序;
- 堆疊定址是隱含的，透過一個堆疊指標器來指向一塊堆疊區域之堆疊的頂端。記憶體堆疊分為兩種:

  向上生長: 向高位址方向生長， 遞增堆疊
  向下生長: 向低位址方向生長， 遞減堆疊

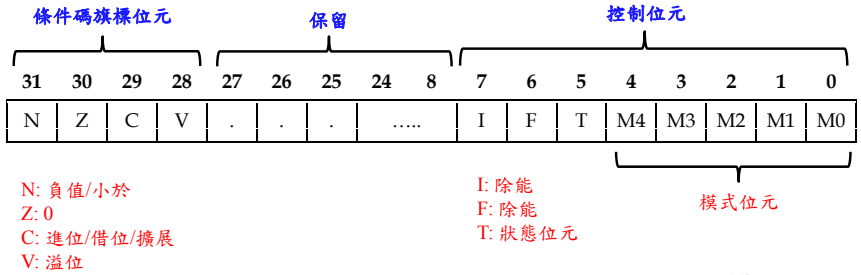
- 堆疊指標指向最後進入的有效數據，稱為滿堆疊;
- 堆疊指標指向下一個數據放入的空位置，稱為空堆疊;
- 可分為滿遞增(FA)、空遞增(EA)、滿遞減(FD)、空遞減(ED);
- 透過 push 寫入資料與 pop 讀取資料。

章節大綱

- 3.1 ARM 微處理器的指令集概述
- 3.2 ARM 指令的定址模式
- 3.3 ARM 指令集
  - 程式狀態暫存器
  - 例外事件
  - 跳躍指令與資料處理指令
  - 乘法指令與乘加指令
  - 程式狀態暫存器 (PSR) 存取指令

程式狀態暫存器

- ARM 系統結構中，包含一個目前程式狀態暫存器 (CPSR) 和5個儲存程式狀態暫存器 (SPSR)。而儲存程式狀態暫存器用來進行例外事件處理，其功能包括:
  - 保存 ALU 中目前操作資訊;
  - 控制允許和禁止中斷;
  - 設定處理器的執行模式。
- 程式狀態暫存器 (CPSR) 的每一位元如下



條件碼旗標欄位 (1/2)

- 條件碼旗標欄位 (Condition Code Flags)
  - N、Z、C 與 V 均為條件碼旗標位元。它們的內容可被算術或邏輯運算的結果而有所改變，並且可以決定某條指令是否被執行。
    - 在 ARM 狀態下，絕大多數的指令都是有條件執行的;
    - 在 Thumb 狀態下，僅有跳躍指令都是有條件執行的;

條件碼旗標欄位 (2/2)

旗標位元	意義
N	負值旗標，當用2的補數所表示的有號數進行運算時，N=1，表示運算的結果為負數; N=0，表示運算的結果為正數或零。
Z	零值旗標，Z=1表示運算的結果為零; Z=0表示運算的結果為非零。
C	進位旗標，可以有4種方法來設定C的值: 1. 加法運算 (包括比較指令CMN): 當運算結果產生了進位時 (無號數溢出)，C=1，否則C=0。 2. 減法運算 (包括比較指令CMP): 當運算時產生了借位 (無號數溢出)，C=0，否則C=1。 3. 對於包含移位操作的非加/減運算指令，C為移出值的最後一位元。 4. 對於其他的非加/減運算指令，C值通常不改變。
V	溢位旗標，可以有2種方法來設置V的值: 1. 對於加/減運算指令，當運算元和運算結果為2補數表示的有號數時，V=1表示有號數溢位，反之則V=0。 2. 對於其他的非加/減運算指令，V值通常不改變。
Q	在ARM v5及以上版本的E系列處理器中，用Q旗標位元來指示增強的DSP運算指令是否發生了溢位。在其他版本的處理器中，Q旗標位元無定義。

控制欄位 (1/2)

- PSR 的低 8 位元 (包括 I、F、T和 M[4:0]) 稱之為控制位元。當發生例外事件時，這些位元可以被改變。如果處理器要執行特權模式時，這些位元也可以由程式來修改。
  - 中斷禁止位元 I 與 F
    - I =1，禁止 IRQ 中斷;
    - F = 1，禁止 FIQ 中斷。
  - T 旗標位元: 該位元反映處理器的執行狀態
    - T = 0，ARM 狀態;
    - T = 1，Thumb 狀態。
  - 執行模式位元 M[4:0]: M0、M1、M2、M3 與 M4 是模式位元。這些位元決定了處理器的執行模式。

控制欄位 (2/2)

- 執行模式位元 M[4:0] 的具體定義表

M[4:0]	處理器模式	可存取的暫存器
0b10000	使用者模式	PC，CPSR，R0~R14
0b10001	FIQ模式	PC，CPSR，SPSR_FIQ，R14_FIQ~R8_FIQ，R7~R0
0b10010	IRQ模式	PC，CPSR，SPSR_IRQ，R14_IRQ，R13_IRQ，R12~R0
0b10011	管理者模式	PC，CPSR，SPSR_SVC，R14_SVC，R13_SVC，R12~R0
0b10111	終止模式	PC，CPSR，SPSR_ABT，R14_ABT，R13_ABT，R12~R0
0b11011	未定義模式	PC，CPSR，SPSR_UND，R14_UND，R13_UND，R12~R0
0b11111	系統模式	PC，CPSR (ARM v4 及以上版本，R14~R0

保留位元

- PSR 中的其餘位元為保留位元，當改變 PSR 中的條件碼旗標位元或控制位元時，保留位元不能被改變，在程式中也不要用保留位元來儲存資料。這些保留位元將應用於未來 ARM 版本的擴充上。

例外事件 (1/2)

- 例外事件: 當正常的程式執行流程發生暫時的停止時。
  - ARM 的例外事件可分為3類:
    - 指令執行引起的直接例外。軟體中斷，未定義指令 (包括所要求的協同處理器不存在時的協同處理器指令) 和預取指令中止 (因為取指令過程中的記憶體故障導致的無效指令) 屬於這一類。
    - 指令執行引起的間接例外。資料中止 (在 Load 和 Store 資料存取時的記憶體故障) 屬於這一類。
    - 外部產生的與指令流無關的例外。重置、 IRQ 和 FIQ屬於這一類。。
  - 例如: 處理一個外部的中斷請求。在處理例外事件之前，目前處理器的狀態必須被加以保留，這樣當例外事件完成後，目前的程式才可以繼續執行。
  - 允許多個例外事件同時發生，但有固定的優先順序;
  - ARM 的例外事件與 8 位元/16 位元微處理器系列結構的中斷很相似，但概念不完全相同。

例外事件 (2/2)

- ARM 系列結構所支援的例外事件類型

例外事件類型	具體含義
重置 (1)	當處理器的重置電位(nRESET) 有效時，產生重置例外事件，程式會跳到重置例外事件處理程式處開始執行。
未定義指令 (7)	當 ARM 或輔助運算器遇到不能處理的指令時，產生未定義指令例外事件。但我們可以使用這種例外事件的機制來進行軟體的模擬與除錯的目的。
軟體中斷 (6)	該例外事件由執行 SWI 指令產生，可用於使用者模式下的程式來引用特權操作指令。但我們可以使用這種例外事件的機制來實現系統功能的引用。
指令預取終止 (5)	若處理器預取指令的位址是不存在的，或是該位址不允許目前指令來存取，記憶體會向處理器發出終止信號，但當預取的指令被執行時，才會產生指令預取終止的例外事件。
資料終止 (2)	若處理器資料存取指令的位址不在，或是該位址不允許目前指令來存取，產生資料終止的例外事件。
IRQ (外部中斷請求) (4)	當處理器的外部中斷請求接腳 (nIRQ) 有效，且 CPSR 中的 I 位元為 0 時，就會產生 IRQ 例外事件。系統的外部設備可通過該例外事件來請求中斷服務。
FIQ (快速中斷請求) (3)	當處理器的快速中斷請求接腳 (nFIQ) 有效，且 CPSR 中的 F 位元為 0 時，就會產生 FIQ 例外事件。

章節大綱

3.3 ARM 指令集 – 跳躍指令

- 跳躍指令用於實現程式流程的跳躍，有兩種方法可以實現:
  - 使用專門的跳躍指令;
  - 直接向程式計數器 PC 寫入跳躍位址值，可以實現在 4GB 的位址空間中任意跳躍。
- ARM 指令集中的跳躍指令可以完成從目前指令向前、或向後的 32MB 的位址空間的跳躍。其中，包括以下 4 個指令:
  - B: Branch (分岐)，跳躍指令
  - BL: Branch with Link (分岐連結)，包含返回的跳躍指令，也即是呼叫副程式
  - BLX: Branch and Exchange Instruction Set (分岐交換)，包括返回和狀態切換的跳躍指令
  - BX: Branch with Link and Exchange Instruction Set (分岐連結交換)，包含狀態切換的跳躍指令

跳躍指令 – B

- B 指令的格式為

B {條件} 目的位址

- B 為單純的跳躍指令，程式控制權從目前的位址轉移到另一個位址。
- 若超過 32MB 範圍時，可以使用 BX，LDR 指令直接改變 PC 值。

```
B       LABEL1      ; 程式將無條件跳躍到
                    LABEL1 處執行
CMP     R1, #0      ; 若 Z 旗標欄位被設定的話，則程
BEQ     LABEL1      式跳躍到 LABEL1處執行

BCS     LABEL2      ; 若 C 進位旗標被設定的話，則跳
                    躍至 LABEL2 執行
```

跳躍指令 – BL

- BL 指令的格式為

  BL {條件} 目的位址

  - BL 除了轉移程式的控制權外，同時也將程式執行順序的下一個位址紀錄到鏈結暫存器（LR）中，如此可以作為呼叫副程式呼叫時使用。
  - 副程式距離 PC 的範圍與指令B是一樣：：+ – 32MB
  - 當副程式執行完畢後，可將鏈結暫存器（LR）的值複製回 PC 中，即可達到返回原程式的目的。

```
BL      SUB1            ; 當程式無條件呼叫副程式 SUB1 時，同
                        時將目前的 PC 值保存到 R14 中
.
SUB1                    ; 副程式進入點
.
MOV     PC, LR          ; 返回原程式
```


跳躍指令 – BX

- BX 指令的格式為

  BX {條件} 目的位址

  - BX 將暫存器 <Rm> 的數值複製至 PC 中，以達到轉移程式控制權;
  - 根據暫存器 <Rm> 的最低位元 <Rm>[0] 來變更指令集狀態，<Rm>[0] 為 1，則變更為 THUMB 指令集狀態。<Rm>[0] 為 0，則變更為 ARM 指令集狀態; 其餘 <Rm>[31:1] 移入PC。
  - 此指令可將程式控制權轉移到 4GB 絕對位址的任一字元位址。

```
CODE    32          ; 從此處起的程式以 ARM 指令集編譯
.
BX    R0            ; 若 R0[31:1] 為位址 LABEL1，R0[0]為1，
                    當跳躍至 LABEL1 處執行時，切換為
                    THUMB 指令集狀態
.
CODE    16          ; 指示從此處的程式為 THUMB 指令集狀
                    態
LABEL   1:          ; LABEL1 程式進入點
```


跳躍指令 – BLX (1/2)

- BLX 指令的格式為

  BLX 目的位址

  - BLX 指令從 ARM 指令集跳躍到指令中所指定的目的位址，並將處理器的工作狀態從 ARM 狀態切換到 THUMB 狀態，該指令同時將 PC 的目前內容保存到暫存器 R14 中。
  - BLX 有兩種用法。第一種是 BX 與 BL 指令結合在一起。此指令可將程式控制權轉移到 4GB 絕對位址的任一字元位址。

```
        CODE    32      ; 從此處起的程式以 ARM 指令集編譯
        …
        BLX   R0        ; 呼叫副程式 SUB1，R0[0]為1，切換為THUMB
                        指令集狀態
        …
        CODE    16      ; 指示從此處的程式為 THUMB 指令集狀態
SUB1 :                  ; SUB1 程式進入點
        BX      R14
```


跳躍指令 – BLX (2/2)

- 第二種是轉移程式控制權，同時也將原本下一個執行位址紀錄到 LR 暫存器中，並變更 THUMB 指令集狀態。整合 BL 與 B 指令。
- 副程式距離 PC 的範圍與指令 B 是一樣: + – 32MB

```
CODE    32          // 從此處起的程式以 ARM 指令集編譯
.
BLX   R0            // 呼叫副程式SUB1，且改變為THUMB指
                    令集狀態
.
CODE    16          // 指示從此處的程式為THUMB指令
                    集狀態
SUB1 :              // SUB1程式進入點
```

資料處理指令

- 資料處理指令可分為資料傳送指令、算數邏輯運算指令和比較指令。
  - 資料傳送指令: 用於暫存器和記憶體之間進行資料的雙向傳輸;
  - 算數邏輯運算指令: 完成常用的算術與邏輯的運算，該類指令不但是將運算結果保存在目的暫存器中，同時更新 CPSR 中相應的條件旗標位元;
  - 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。
- 資料處理指令包括算術指令（加，減），邏輯指令（AND，OR，NOT，XOR，位元清除），比較測試指令，以及複製指令。

ARM資料處理指令

指令	運算碼	動作
AND	0000	Operand1 AND operand2，運算元1 AND 運算元2
EOR	0001	Operand1 EOR operand2，運算元1 EOR 運算元2
SUB	0010	Operand1 - operand2，運算元1 - 運算元2
RSB	0011	Operand2 – operand1，運算元2 - 運算元1
ADD	0100	Operand1 + operand2，運算元1 + 運算元2
ADC	0101	Operand1 + operand2 +carry，運算元1 +運算元2 +進位
SBC	0110	Operand1 + operand2 +carry -1，運算元1 +運算元2 +進位 - 1
RSC	0111	Operand2 + operand1 +carry -1，運算元2 +運算元1 +進位 - 1
TST	1000	As AND, but result is not written，與AND一樣，但結果值不寫入
TEQ	1001	As EOR, but result is not written，與EOR一樣，但結果值不寫入
CMP	1010	As SUB, but result is not written，與SUB一樣，但結果值不寫入
CMV	1011	As ADD, but result is not written，與ADD一樣，但結果值不寫入
ORR	1100	Operand1 OR operand2，運算元1 OR 運算元2
MOV	1101	Operand2 (operand1 is ignored)，運算元2 (運算元1被忽略)
BIC	1110	Operand1 AND NOT operand 2 (Bit clear)，運算元1 AND NOT運算元2 (位元清除)
MVN	1111	NOT operand2 (operand1 is ignored)，NOT 運算元2 (運算元1被忽略)

MOV 指令

- MOV 指令的格式

MOV {條件} {S} 目的暫存器，來源暫存器

- MOV 指令可以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。
- 程式範例

```
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位元後傳送到 R1
```


MVN 指令

- MVN 指令的格式

MVN {條件} {S} 目的暫存器，來源暫存器

- MVN 指令可以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。而與 MOV 指令不同之處是在傳送之前，被加以反相了。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。
- 程式範例

```
MVN   R0, #0          ; 將立即數 0 取反相值傳送到暫存器 R0 中，完成後 R0= – 1
```

CMP 指令

CMP 指令的格式

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，但不儲存結果，只更改條件旗標位元。
- 程式範例

```
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
```


CMN 指令

CMN 指令的格式

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，或立即數反相後進行比較。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與運算元2相加，並根據結果更改條件旗標位元。
- 程式範例

```
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
```


TST 指令

TST 指令的格式

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，或立即數進行每一位元的 AND 運算，並根據結果更改條件旗標位元。
- 運算元1是要測試的資料，而運算元2是一個位元遮罩，該指令一般用來檢測是否設定了特殊位元。
- 程式範例

```
TST   R1, #%1       ;用於測試在暫存器 R1 中是否設定了最低位元 (%表示二進制數值)
TST   R1, #oxffe    ;將暫存器 R1 的值與立即數 oxffe 按位元 AND 運算，並根據結果來設定 CPSR 的旗標位元
```


TEQ 指令

TEQ 指令的格式

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，或立即數進行每一位元的 XOR 運算，並根據結果來更新 CPSR 的條件旗標位元的值。
- 該指令通常用於比較運算元1和運算元2是否相等。
- 程式範例

```
TEQ   R1, R2        ;將暫存器 R1 的值與暫存器 R2 的值執行 XOR 運算，並根據結果來設定 CPSR 的旗標位元
```

ADD 指令

ADD 指令的格式

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，並將結果值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，或是一個立即值。
- 程式範例

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


ADC 指令

ADC 指令的格式

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，並將結果值存放到目的暫存器中。它使用了進位旗標，可以執行比32位元還要大的數值的加法。
- 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，或是一個立即值。
- 程式範例: 完成兩個128位元數值的加法

```
ADDS   R0, R4, R8     ; 加最低端的字元組
ADCS   R1, R5, R9     ; 加第二個字元組，包含進位
ADCS   R2, R6, R10    ;加第三個字元組，包含進位
ADC    R3, R7, R11    ;加第四個字元組，包含進位
```


SUB 指令

SUB 指令的格式

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，並將結果值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，或是一個立即值。
- 程式範例

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


SBC 指令

SBC 指令的格式

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 條件旗標位元的反相碼，並將結果值存放到目的暫存器中。它使用了進位旗標表示借位，可以執行大於32位元的減法。
- 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，或是一個立即值。
- 程式範例

```
SUBS   R0, R1, R2     ; R0 = R1 – R2 – !C，並根據結果來設定
                      CPSR 的條件旗標位元的值
```

RSB 指令

RSB 指令的格式

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，並將結果值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，或是一個立即值。
- 程式範例

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


RSC 指令

RSC 指令的格式

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 條件旗標位元的反相值，並將結果值存放到目的暫存器中。它使用了進位旗標表示借位，可以執行大於32位元的減法。
- 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，或是一個立即值。
- 程式範例:

```
RSC   R0, R1, R2          ; R0 = R2 – R1 – !C，並根據結果來設定
                          CPSR 的條件旗標位元的值
```


AND 指令

AND 指令的格式

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 AND 運算，並將結果值存放到目的暫存器中。該指令常用於遮罩運算元1的某些位元。
- 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，或是一個立即值。
- 程式範例:

```
AND   R0, R0, #3          ; 該指令保持 R0 的第0與1位，其餘位元清
                          除
```


ORR 指令

ORR 指令的格式

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 OR 運算，並將結果值存放到目的暫存器中。該指令常用於設置運算元1的某些位元。
- 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，或是一個立即值。
- 程式範例:

```
ORR   R0, R0, #3          ; 該指令設置 R 0 的0第0與1位，其餘位元
                          保持不變
```

EOR指令

- EOR 指令的格式

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 XOR 運算，並將結果值存放到目的暫存器中。該指令常用於反轉運算元1的某些位元。
- 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，或是一個立即值。
- 程式範例:

```
EOR    R0, R0, #3        ; 該指令反轉 R0 的第0與1位，其餘位元保持不變
```


BIC 指令

- BIC 指令的格式

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的某些位元，並將結果值存放到目的暫存器中。該指令常用於反轉運算元1的某些位元。
- 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，或是一個立即值。
- 程式範例:

```
BIC    R0, R0, #%1011    ; 該指令清除 R0 的第0、1與3位元，其餘位元保持不變
```


章節大綱

乘法指令與乘加指令

- 支援的乘法指令與乘加指令共有 6 個，可分為運算結果為 32 位元與 64 位元等兩種類型。
  - MUL: 32 位元乘法指令
  - MLA: 32 位元乘加指令
  - SMULL 64: 64 位元有號數乘法指令
  - SMLAL 64: 64 位元有號數乘加指令
  - UMULL 64: 64 位元無號數乘法指令
  - UMLAL 64: 64 位元無號數乘加指令

† 註1：與資料處理指令不同的是，指令中了所有運算元與目的暫存器必須為通用暫存器，且不能對運算元使用立即數值或是被移位的暫存器。

† 註2: 目的暫存器和運算元1必須是不同的暫存器。

MUL 指令

MUL 指令的格式

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，並將結果值存放到目的暫存器中，並根據結果來更新 CPSR 的條件旗標位元的值。
- 運算元1與運算元2均為 32 位元的有號數或無號數。
- 程式範例:

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


MUA 指令

MUA 指令的格式

MUA {條件} {S} 目的暫存器，運算元1，運算元2 ，運算元3

- MUA 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，並將結果值存放到目的暫存器中，並根據結果來更新 CPSR 的條件旗標位元的值。
- 運算元1與運算元2均為32位元的有號數或無號數。
- 程式範例:

```
MUA     R0, R1, R2, R3  ; R0 = R1 × R2 + R3
MUAS    R0, R1, R2, R3  ; R0 = R1 × R2 + R3，同時設定 CPSR 中相關條件旗標位元
MUL     R1, R2, R3      ; R1 = R2 × R3
MLAEQS  R1, R2, R3, R4  ; R1 = R2 × R3 + R4，同時設定 CPSR 中相關條件旗標位元
```


SMULL 指令

SMULL 指令的格式

SMULL {條件} {S} 低目的暫存器 (RdLo)，高目的暫存器 (RdHi)，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 RdLo 中，結果的高32位元放置到 RdHi 中，同時可以根據運算結果來設定 CPSR 中相應的條件旗標位元。
- 運算元1與運算元2均為32位元的有號數。
- 程式範例:

```
SMULL   R0, R1, R2, R3  ; R0 = (R2 × R3) 的低32位元
                        ; R1 = (R2 × R3) 的高32位元
```


SMLAL 指令

SMLAL 指令的格式

SMLAL {條件} {S} 低目的暫存器 (RdLo)，高目的暫存器 (RdHi)，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 RdLo 的值相加後又放置到 RdLo 中，結果的高32位元同目的暫存器 RdHi 的值相加後又放置到 RdHi 中，同時可以根據運算結果來設定 CPSR 中相應的條件旗標位元。
- 運算元1與運算元2均為32位元的有號數。
- 程式範例:

```
SMLAL   R0, R1, R2, R3  ; R0 = (R2 × R3) 的低32位元 +R0
                        ; R1 = (R2 × R3) 的高32位元 +R1
```

UMULL 指令

- UMULL 指令的格式

UMULL {條件} {S} 低目的暫存器 (RdLo)，高目的暫存器 (RdHi)，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 RdLo 中，結果的高32位元放置到 RdHi 中，同時可以根據運算結果來設定 CPSR 中相應的條件旗標位元。
- 運算元1與運算元2均為32位元的無號數。
- 程式範例:

```
SMULL    R0, R1, R2, R3  ; R0 = (R2 × R3) 的低32位元
                         ; R1 = (R2 × R3) 的高32位元
```


UMLAL 指令

- UMLAL 指令的格式

UMLAL {條件} {S} 低目的暫存器 (RdLo)，高目的暫存器 (RdHi)，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 RdLo 的值相加後又放置到 RdLo 中，結果的高32位元同目的暫存器 RdHi 的值相加後又放置到 RdHi 中，同時可以根據運算結果來設定 CPSR 中相應的條件旗標位元。
- 運算元1與運算元2均為32位元的有號數。
- 程式範例:

```
UMLAL    R0, R1, R2, R3  ; R0 = (R2 × R3) 的低32位元 +R0
                         ; R1 = (R2 × R3) 的高32位元 +R1
```


章節大綱

程式狀態暫存器 (PSR) 存取指令

- 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間傳送資料。
- 包括兩種指令
  - MRS 存取指令: 程式狀態暫存器到通用暫存器的資料傳送指令
  - MSR 存取指令: 通用暫存器到程式狀態暫存器的資料傳送指令

MRS 指令

MRS 指令的格式

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，該指令一般用於以下幾種情況:
  - 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。
  - 當在例外事件處理或程式切換時，需要保持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，然後加以保存。
- 程式範例:

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


MSR 指令 (1/3)

MSR 指令的格式

MSR {條件} 程式狀態暫存器 (CPSR 或 SPSR) _<區域>，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。<區域>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個區域:
  - 位元 [31:24]: 為條件旗標位元區域，用 f 表示;
  - 位元 [23:16]: 為狀態位元區域，用 s 表示;
  - 位元 [15:8]為擴展位元區域，用 x 表示;
  - 位元 [0:7]為控制位元區域，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指令中指明要操作的區域。
- 程式範例:

```
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 CPSR，但僅僅修改
                        CPSR 中的控制位元區域
```


MSR 指令 (2/3)

- 此外，在使用者模式的指令動作是如下所列的程式

```
MSR     CPSR_all, Rm            ; 傳送 Rm[31:28] 的內容到
                                  CPSR[31:0]
MSR     CPSR_flg, Rm            ; 傳送 Rm[31:28] 的內容到
                                  CPSR[31:28]
MSR     CPSR_flg, #0xA0000000   ; 設定CPSR[31:28]為0xA
                                  (設定N，C; 清除Z，V)
MRS     Rd, CPSR                ; 傳送Rm[31:0]的內容到
                                  CPSR[31:0]
```


MSR 指令 (3/3)

- 在特權模式下，這些的指令動作是如下所列的程式

```
MSR     CPSR_all, Rm            ; 傳送 Rm[31:0] 的內容到
                                  CPSR[31:0]
MSR     CPSR_flg, Rm            ; 傳送 Rm[31:28] 的內容到
                                  CPSR[31:28]
MSR     CPSR_flg, #0x50000000   ; 設定CPSR[31:28]為0x5
                                  (設定Z，V; 清除N，C)
MSR     SPSR_all, Rm            ; 傳送 Rm[31:0] 的內容到
                                  SPSR[31:0]
MSR     SPSR_flg, Rm            ; 傳送 Rm[31:28] 的內容到
                                  SPSR[31:28]
MSR     SPSR_flg, #0xC0000000   ; 設定SPSR[31:28]為0xC
                                  (設定N，Z; 清除C，V)
MRS     Rd, SPSR                ; 傳送Rm[31:0]的內容到
                                  SPSR[31:0]
```

載入/存回指令

- ARM 微處理器支援載入/存回指令，用於暫存器和記憶體之間傳送資料，載入指令用於將記憶體的資料傳送到暫存器，存回指令則完成相反的動作。
- 常用的載入/存回指令
  - LDR　字元組資料載入指令
  - STR　字元組資料存回指令
  - LDRB　位元組資料載入指令
  - STRB　位元組資料存回指令
  - LDRH　半字元組資料載入指令
  - STRH　半字元組資料存回指令

LDR 指令

- LDR 指令的格式為:

  LDR　{條件}　目的暫存器，<記憶體位址>

  - 程式範例

```
LDR   R0, [R1]          ;將記憶體位址為 R1 的字元組資料讀到 R0
LDR   R0, [R1, R2]      ;將記憶體位址為 R1+R2 的字元組資料讀到 R0
LDR   R0, [R1, #8]      ;將記憶體位址為 R1+8 的字元組資料讀到 R0
LDR   R0, [R1, R2] !    ;將記憶體位址為 R1+R2 的字元組資料讀到 R0，並將新位址 R1+R2 寫入 R1
LDR   R0, [R1, #8] !    ;將記憶體位址為 R1+8 的字元組資料讀到 R0，並將新位址 R1+8 寫入 R1
LDR   R0, [R1], R2      ;將記憶體位址為 R1 的字元組資料讀到 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 的字元組資料讀到 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 的字元組資料讀到 R0，並將新位址 R1+R2x4 寫入 R1
```


STR 指令

- STR 指令的格式為:

  STR　{條件}　來源暫存器，<記憶體位址>

  - 程式範例

```
STR   R0, [R1, #8]      ;將 R0 中的字元組資料寫入到以 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 中的字元組資料寫入到以 R1+8 為位址的記憶體中，並將新位址 R1+8 寫入 R1
```