ARM ASM Note.

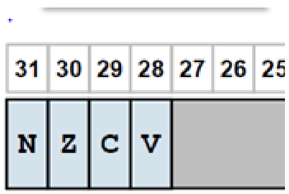
Yung-Chi

Outline

- Program Status Register(PSR)
  - Conditional Branch
- Memory Access
- Tips

Program Status Register

- 程式狀態暫存器 (PSR)
  - 要知道目前ALU運作的狀態和branch條件看他啦!
- N: Negative
  - 上一個ALU運算完後結果是否為負值
  - 通常可以用來比較2個暫存器值大小
- Z: Zero
  - …結果是否為0,
  - 通常可以用來比較2個暫存器是否相等
- C: Carry
  - 是否產生進位
- V: Overflow
  - 溢位
- 當進行完各種算術指令後皆會改變這個暫存器的值



31	30	29	28	27	26	25
N	Z	C	V			

Conditional Branch

- 在ARM組語中要進行條件跳躍時通常會需要2個指令完成
  - 比較或一般算術指令: CMP <Rs>,<Rd>
    - CMP其實是進行Rs-Rd運算但不儲存結果
    - CMP可以用ADDS, SUBS,…等算術指令取代
  - 條件跳躍指令: B<cc> Label
    - <cc>指的是跳躍條件，會根據PSR中的NZCV flags等條件判斷是否需要跳躍，例如BEQ是檢查Zero flag是否為Set
    - 藉由不同的Condition則可以完成不同的邏輯判斷
- 例子: 若R1不等於R0則跳躍至L1

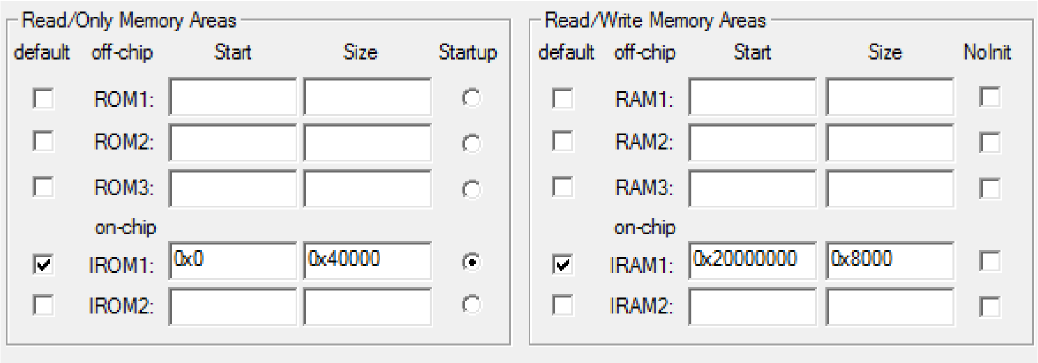
```
   SUBS R1, R0
   BNE   L1
…
L1
   BX     L10
```

Branch 指令的比較

- BX (Branch and Exchange)
  - 從某個Register讀取位址並跳躍
  - 常做為function return用
  - Ex: BX LR
- BL (Branch with link)
  - 跳躍至某個Label，並將下一個指令的位址存至LR
  - 範圍: PC+-16M bytes
- B
  - 直接跳躍，範圍PC+-2046 bytes
- B<cc>
  - 條件跳躍，範圍PC+-254 bytes
- Note: B and B<cc>通常只能用在function內的小範圍跳躍

Memory Space

- Data RAM start address: 0x20000000
- Data RAM size: 0x8000 (32kB)
- Code start address: 0x0
- Code size: 0x40000 (256kB)



Read/Only Memory Areas

default	off-chip	Start	Size	Startup
□	ROM1:			○
□	ROM2:			○
□	ROM3:			○
	on-chip			
☑	IROM1:	0x0	0x40000	●
□	IROM2:			○

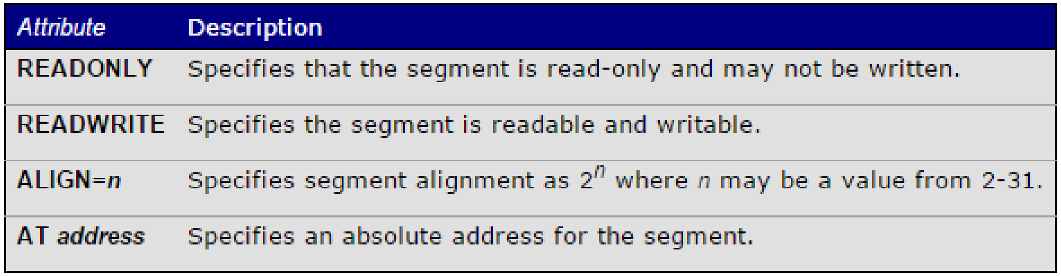
Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
□	RAM1:			□
□	RAM2:			□
□	RAM3:			□
	on-chip			
☑	IRAM1:	0x20000000	0x8000	□
□	IRAM2:			□

Options for target…

AREA Assembler Statement

- 用來告知Assembler接下來的區段擺在哪
- **AREA** *segment-name*, *class-name* <[>, *attributes* <[>,...<]><]>
  - segment-name：區段名稱，可以隨意定
  - class-name：類型，可以是CODE, DATA, HEAP, STACK等
  - Attributes：描述這個區段的位址與對齊方式等



Attribute	Description
READONLY	Specifies that the segment is read-only and may not be written.
READWRITE	Specifies the segment is readable and writable.
ALIGN=*n*	Specifies segment alignment as 2^n where *n* may be a value from 2-31.
AT *address*	Specifies an absolute address for the segment.

- Note:在GNU的系統中通常使用.section的方式描述區段，並用link script來描述這些區段如何被合併

變數宣告

- Read only data
  - 相當於C中的constant variable
  - 例子：宣告一個Hello字串並擺放在CODE section

```
AREA   |.text|, CODE, READONLY
…
Hello DCB “Hello!”
```

- R/W data
  - 例子：在DATA section宣告一個為初始化且大小為100的陣列X，並宣告一個4byte初始值為0x100的Y

```
AREA   |.data|, DATA, READWRITE
X SPACE 100
Y DCW 0x100
```

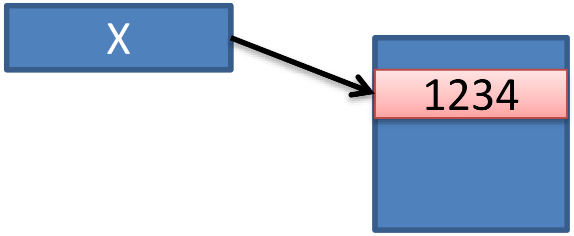
→

```
char X[100];
int    Y;
```

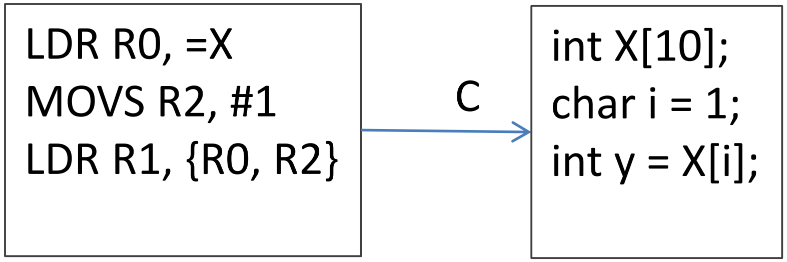
C語言版

變數存取

- 直接定址: 變數值存在Register裡
  - MOVS R1, R0
- 間接定址: 變數值放在某個Memory裡, 類似C的指標
  - LDR R0, =X
  - LDR R1, {R0}



- 相對定址: 或稱Indexed Addressing，拿一個Register當index，類似存取陣列



```
LDR R0, =X
MOVS R2, #1
LDR R1, {R0, R2}
```

```
int X[10];
char i = 1;
int y = X[i];
```

Memory存取的長度

- 1Byte
  - LDRB, STRB
- 2Bytes
  - LDRH, STRH
- 4Bytes
  - LDR, STR

Function call (1/2)

- No function input and output parameters

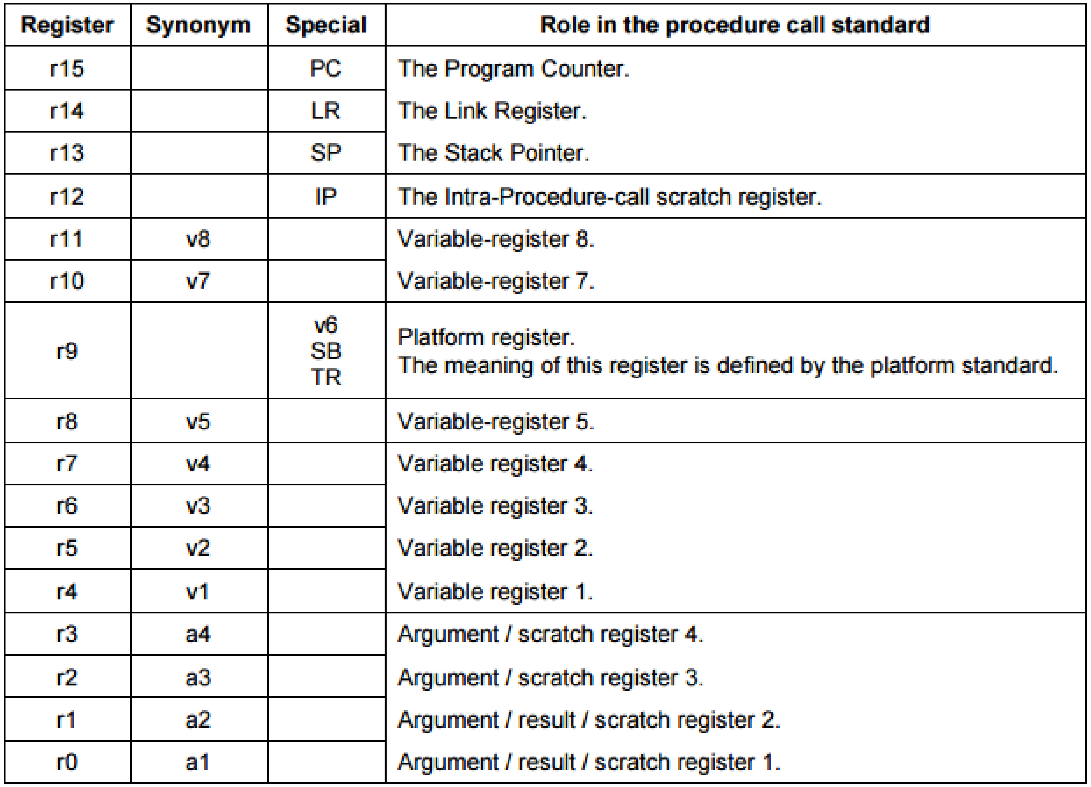
C language	ARM assembly language

void main()
{
Again:
foo();
goto Again;
}
 	
void foo()
{
char X, Y;
X = Y + 5;
}
	…
	main
	Again:
	BL foo
	B Again
	BX LR
	
	Foo
	LDR R5, =Y
	LDR R3, [R5]
	ADDS R3, #5
	LDR R4, =X
	STRB R3, [R4]
	BX LR

- Call instruction will **save next instruction address** into LR register then jump to the label.
- Do X=Y+5
- BX LR instruction will **restore the instruction address form LR** to PC

Procedure Call Standard for the ARM Architecture (AAPCS)

- ARM規範了這些registers的用途，例如那些可當參數，那些當區域變數



Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3.
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

Table 2, Core registers and AAPCS usage

Function call (2/2)

- Function call with input and output parameters

C language	ARM assembly language
char X;	…
	main
void main()	MOVS a1, #10
{	MOVS a2, #5
X = foo(10, 5);	LDR v1, =X
}	BL foo
	STRB a1, [v1]
char foo(char x, char y)	BX LR
{	Foo
return x + y;	ADDS a1, a1, a2
}	BX LR

- Setup function parameters
- Return variable

Function define

- 可以利用FUNCTION定義某個label為function，在編譯時assembler會自動在ENDFUNC前加上BX LR指令

```
Foo FUNCTION
    …
    ENDFUNC
```

- 一般的label也可以是function

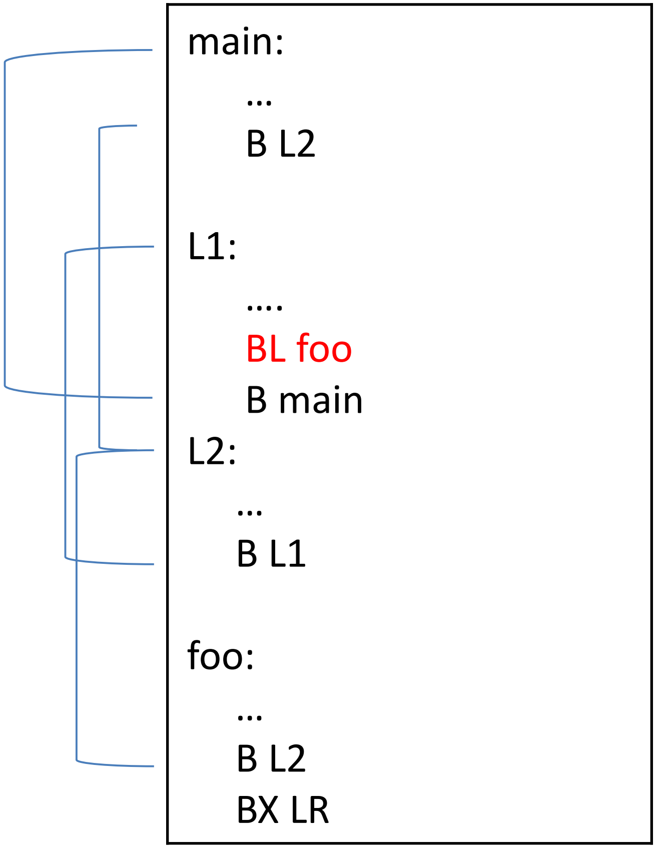
```
Foo
   …
     BX LR
```

Questions

- How express “char **ptr;” in assembly?
- If function have more then six parameters how can we pass it?
  - Ans: use push/pop stack or pass a stack pointer

Tips

- Don’t use unstructured branch in your code.

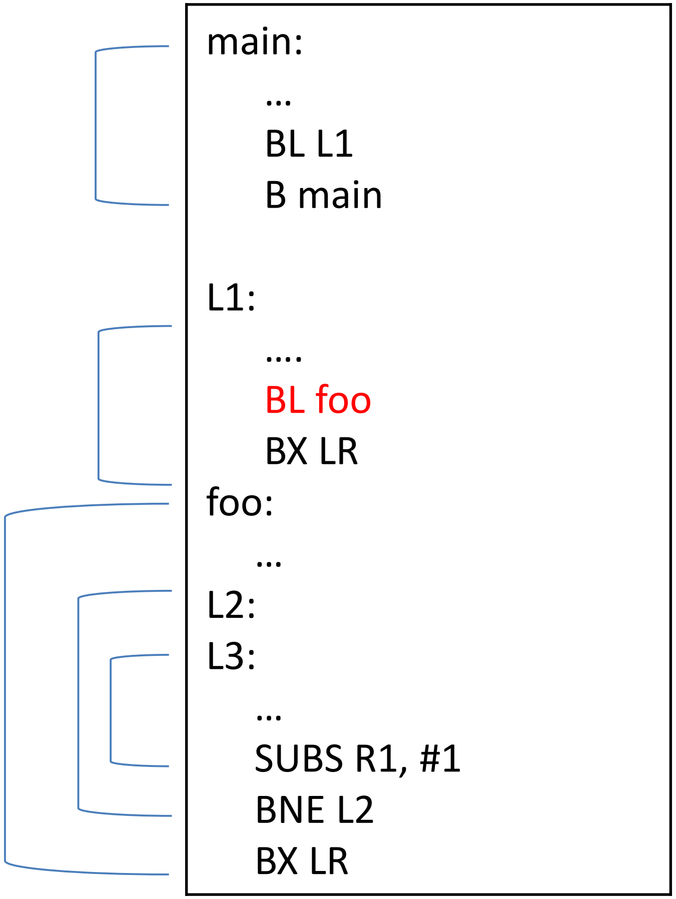


```
main:
     …
     B L2

L1:
     ….
     BL foo
     B main
L2:
    …
    B L1

foo:
    …
    B L2
    BX LR
```

Bad case

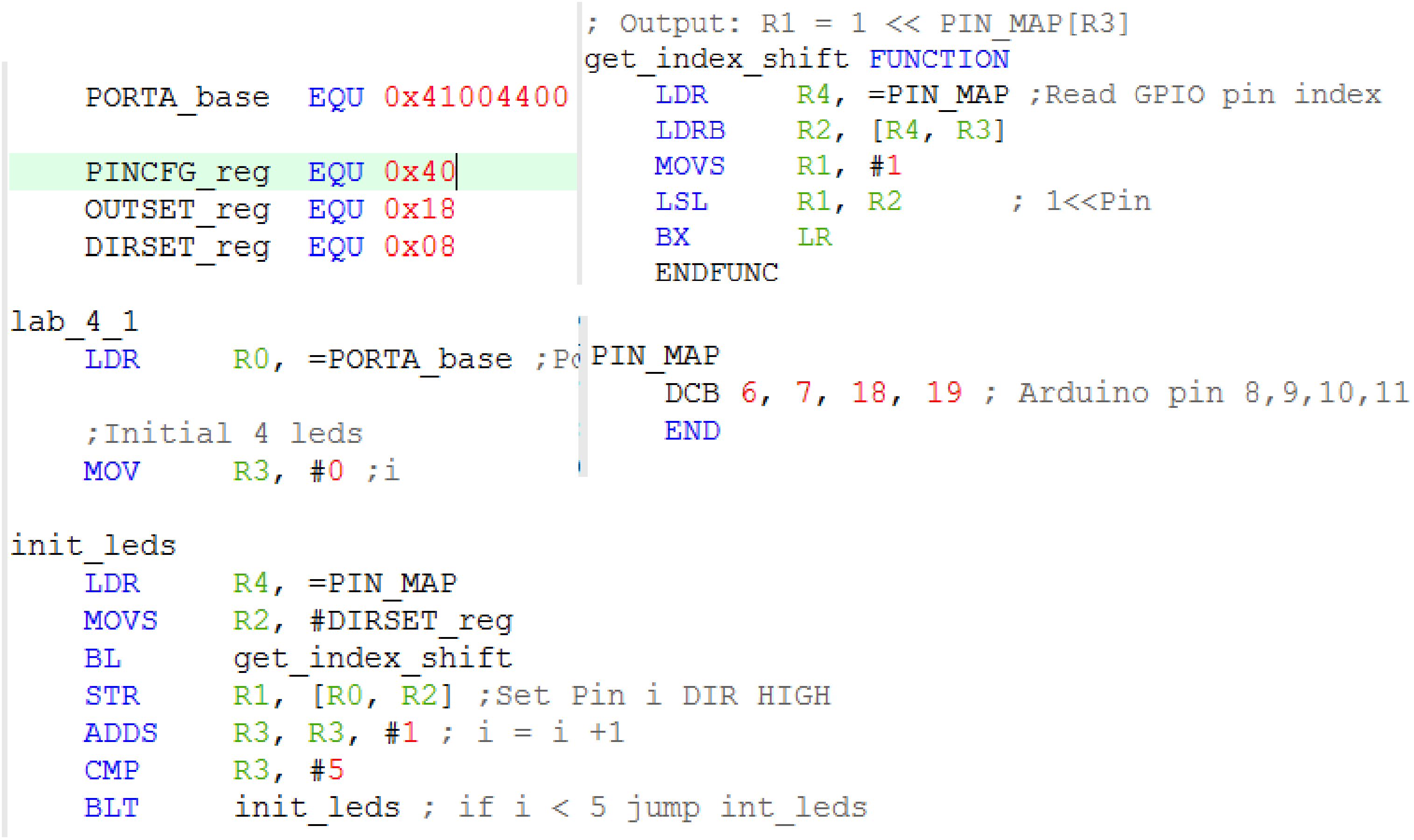


```
main:
     …
     BL L1
     B main

L1:
     ….
     BL foo
     BX LR
foo:
    …
L2:
L3:
    …
    SUBS R1, #1
    BNE L2
    BX LR
```

Mach batter

Initial GPIO Example



```
    PORTA_base   EQU 0x41004400

    PINCFG_reg   EQU 0x40
    OUTSET_reg   EQU 0x18
    DIRSET_reg   EQU 0x08

lab_4_1
    LDR      R0, =PORTA_base ;Po

    ;Initial 4 leds
    MOV      R3, #0 ;i

init_leds
    LDR      R4, =PIN_MAP
    MOVS     R2, #DIRSET_reg
    BL       get_index_shift
    STR      R1, [R0, R2] ;Set Pin i DIR HIGH
    ADDS     R3, R3, #1 ; i = i +1
    CMP      R3, #5
    BLT      init_leds ; if i < 5 jump int_leds
```

```
; Output: R1 = 1 << PIN_MAP[R3]
get_index_shift FUNCTION
    LDR      R4, =PIN_MAP ;Read GPIO pin index
    LDRB     R2, [R4, R3]
    MOVS     R1, #1
    LSL      R1, R2       ; 1<<Pin
    BX       LR
    ENDFUNC

PIN_MAP
    DCB 6, 7, 18, 19 ; Arduino pin 8,9,10,11
    END
```