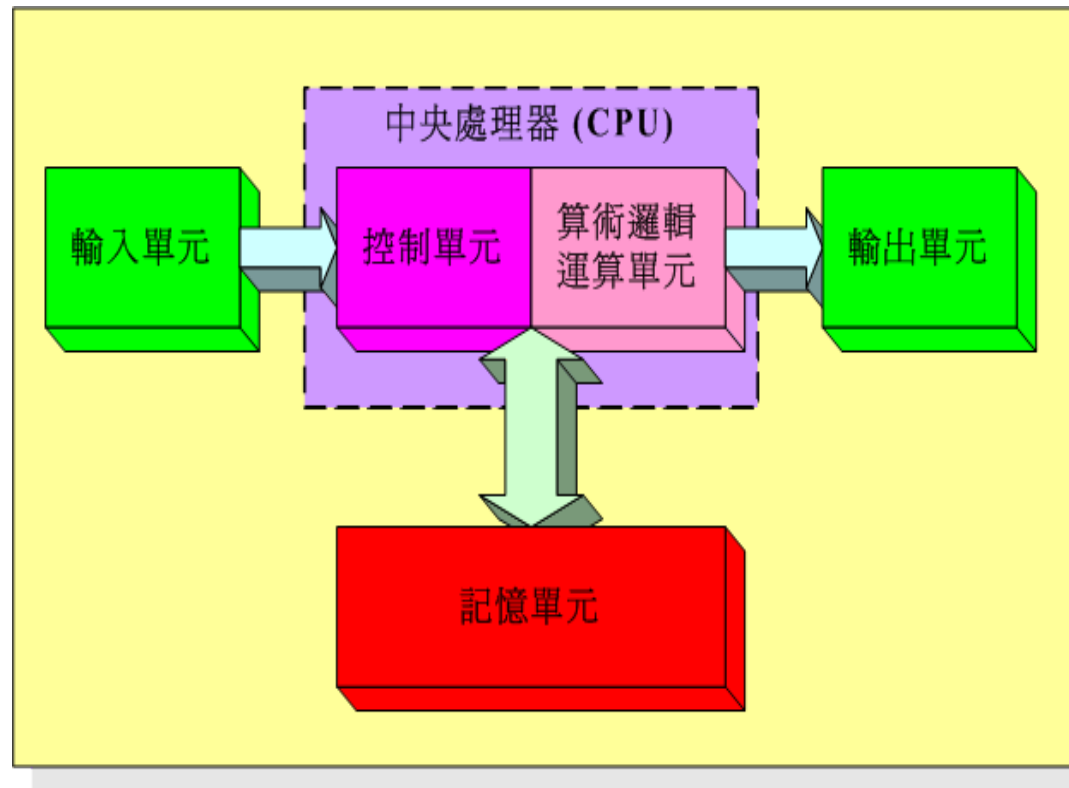


# Microprocessor 記憶體與組合語言



- 16進位的算法, 16進位A1→1010 0001, 8051組語中前綴 0x 或後綴H都表示16進位 ex: 0x30 等於 30H 等於 48
- RAM (random access memory) 斷電數據死 多存資料
- ROM (read only memory) 斷電數據在 多存程式
- 組合語言 → 記憶體(有記憶體, 就要知道如何存取存在裡面的資料) → 定址法

# 程式記憶體

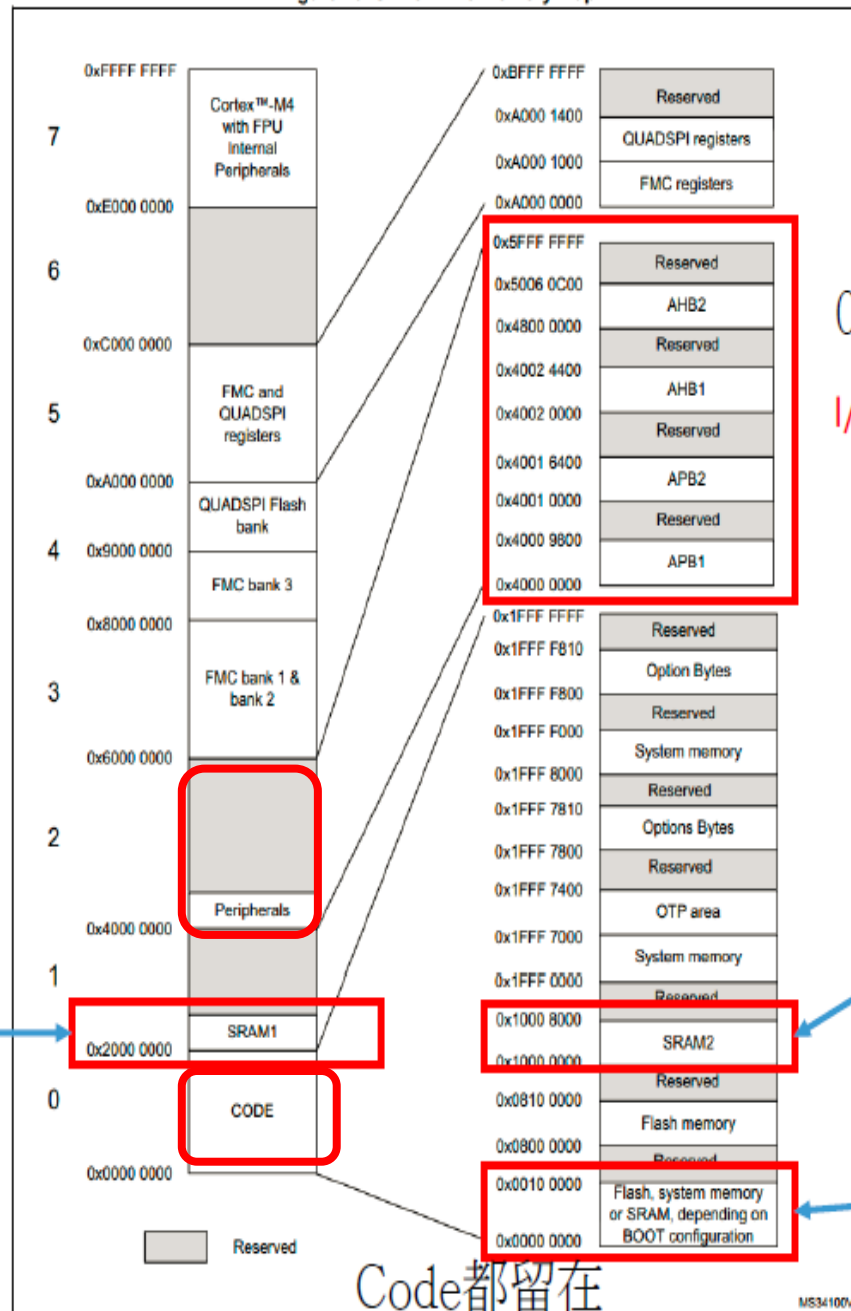
記憶體分為：

1. 程式記憶體.
2. 資料記憶體.
3. 擴充資料記憶體
4. 非揮發性資料記憶體
5. 非揮發性暫存器

Figure 10. STM32L476 memory map

# Memory Map

## ARM M4



Internal 92KB SRAM1

0x48000000 是GPIOA

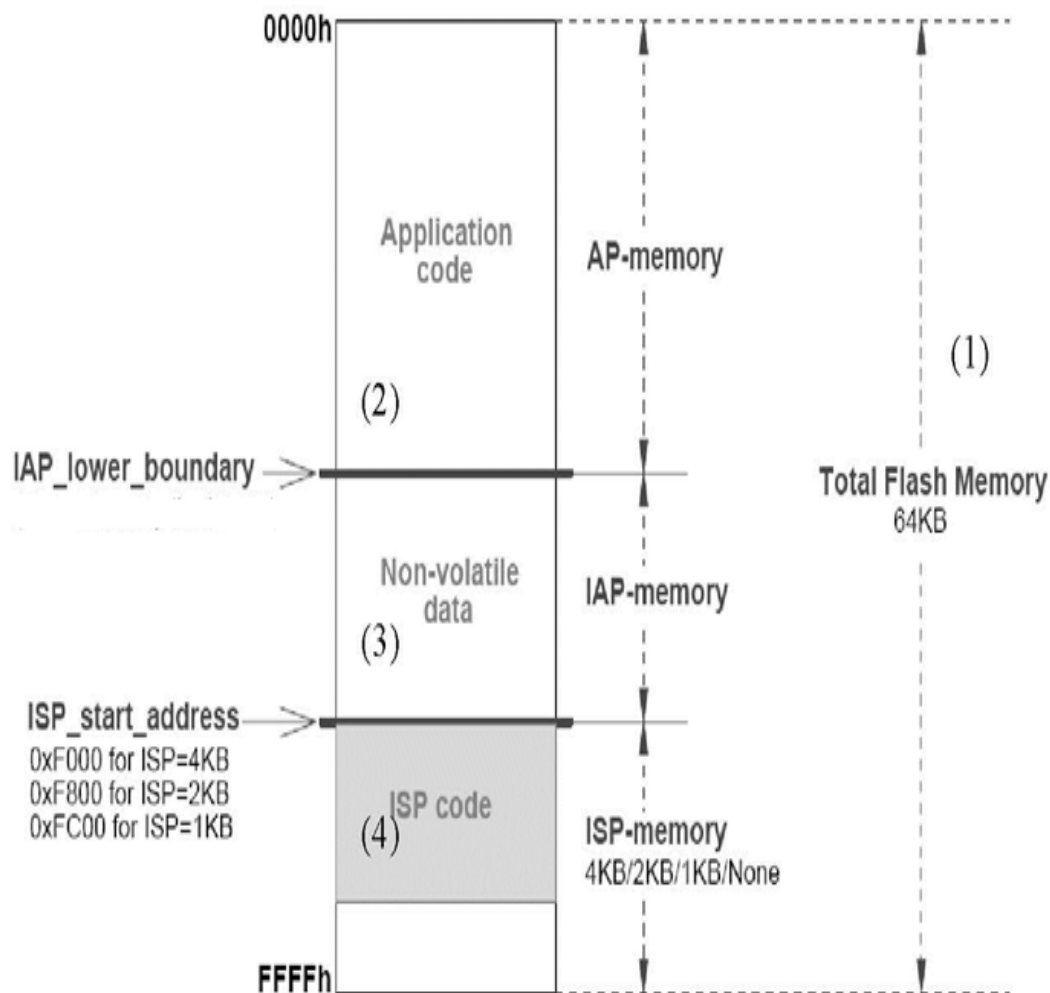
I/O memory-mapping area

Code都留在

Flash ROM內執行

MS34100V3

# 程式記憶體 (8051)



1. 全部總共64K-byte的Flash ROM (0x0000 – 0xFFFF)
2. AP(application code) memory: 程式操作碼
3. 可設定IAP memory(in application programming): 臨時的非揮發性資料(non-volatile data)
4. ISP memory(0k, 1k, 2k, 4k): 線上燒錄程式
5. 重置時, 程式計數器(program counter), 從0x0000開始, 執行指令則PC遞加. (像重開機)
6. 中斷向量位址, 前六個: INT(中斷)x2, Timer(計時)x3, UARTx1

# 資料記憶體

記憶體分為：

1. 程式記憶體.
2. 資料記憶體.
3. 擴充資料記憶體
4. 非揮發性資料記憶體
5. 非揮發性暫存器

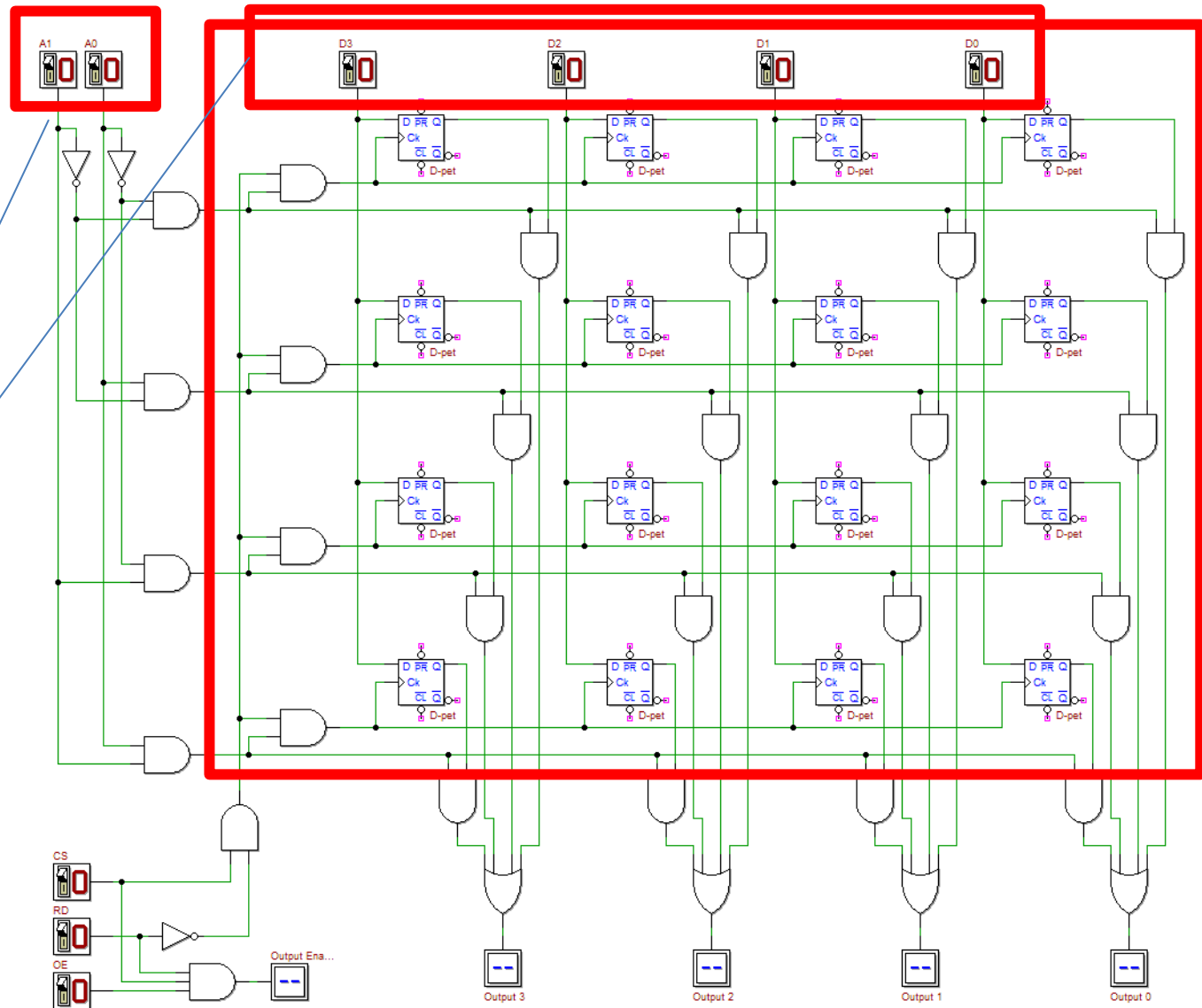
# Memory

Data (值)

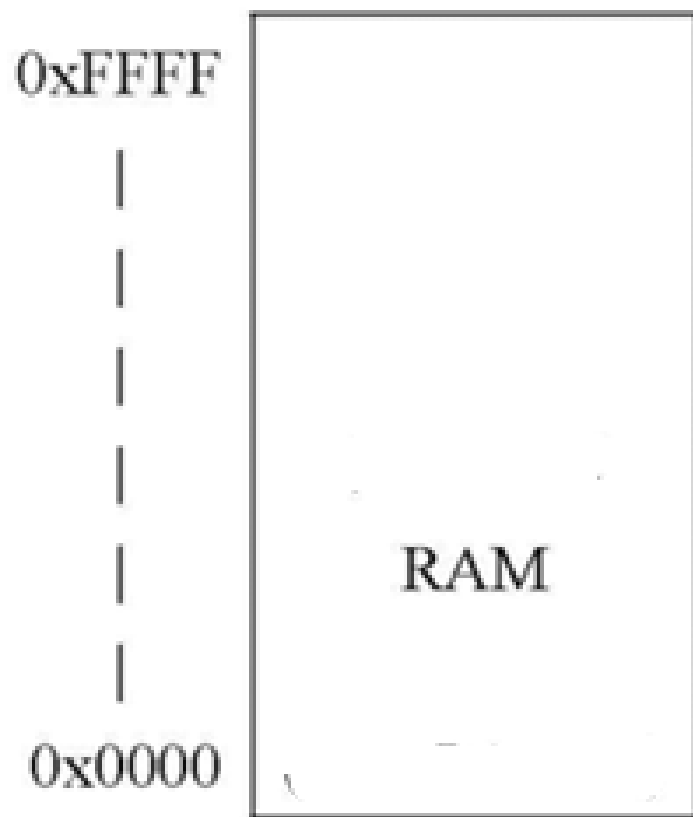
- Register
- Catch
- RAM

Address(位址)

位址 vs 值  
Address vs value



# 位址和值



位址

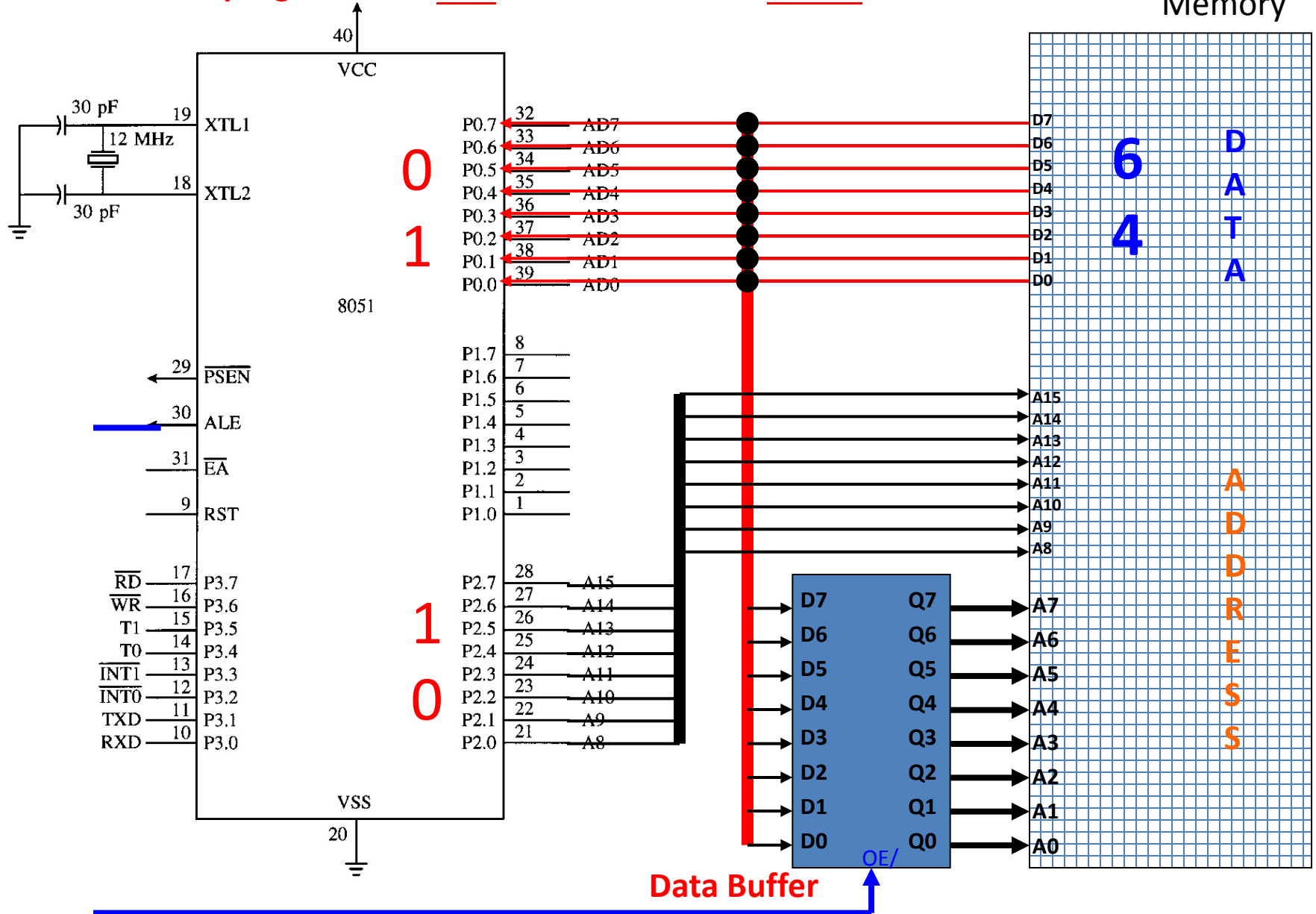
儲存值



## 8051 CPU 存取外部儲存器RAM

**CPU fetches the program code 99H stored at address 1234H**

# Program Memory

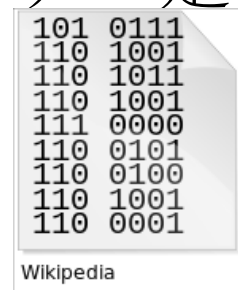


# 電腦語言

- 機器語言(machine code)
- 組合語言(assembly language)
- 硬體描述語言(hardware description language)  
ex: VHDL, verilog
- 程式語言 ex: C, C++, Matlab, python

# 電腦語言

- 機器語言(machine code):是一種指令集的體系。這種指令集稱為機器代碼（machine code），是電腦的CPU可直接解讀的資料。
- 組合語言(assembly language):
  - 是一種用於電子電腦、微處理器、微控制器，或其他可程式設計硬體的低階語言。
  - 在不同的設備中，組合語言對應著不同的機器語言指令集。一種組合語言專用於某種電腦系統結構，而不像許多高階語言，可以在不同系統平臺之間移植。
  - 使用組合語言編寫的原始程式碼，然後通過相應的組合語言程式將它們轉換成可執行的機器代碼。



# 電腦語言

- 硬體描述語言(hardware description language)是用來描述電子電路（特別是數位電路）功能、行為的語言，可以在暫存器傳輸級、行為級、邏輯閘級等對數位電路系統進行描述。

ex: VHDL, verilog

```
entity <实体名称> is
  port(
    a : IN STD_LOGIC;
    b : OUT STD_LOGIC
  );
end [实体名称];
```

- 程式語言 ex: C, C++, Matlab, python

# 電腦語言

- 人類易讀性:

????語言 < ????語言 < ????語言 < ????語言

- 執行速度(機器易讀性):

????語言 > ????語言 > ????語言 > ????語言

# 電腦語言

- 人類易讀性:

機器語言 < 組合語言 < 硬體描述語言 < 程式語言

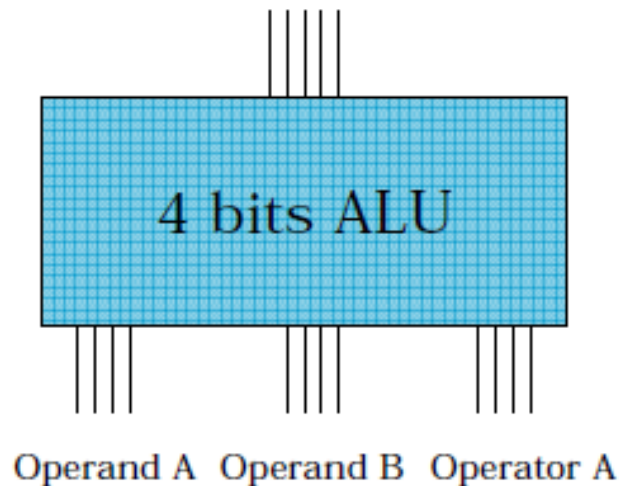
- 執行速度(機器易讀性):

機器語言 > 組合語言 > 硬體描述語言 > 程式語言

# Machine Cycle

# Machine Cycle

- Let's review computer architecture first
  - 4 bits ALU

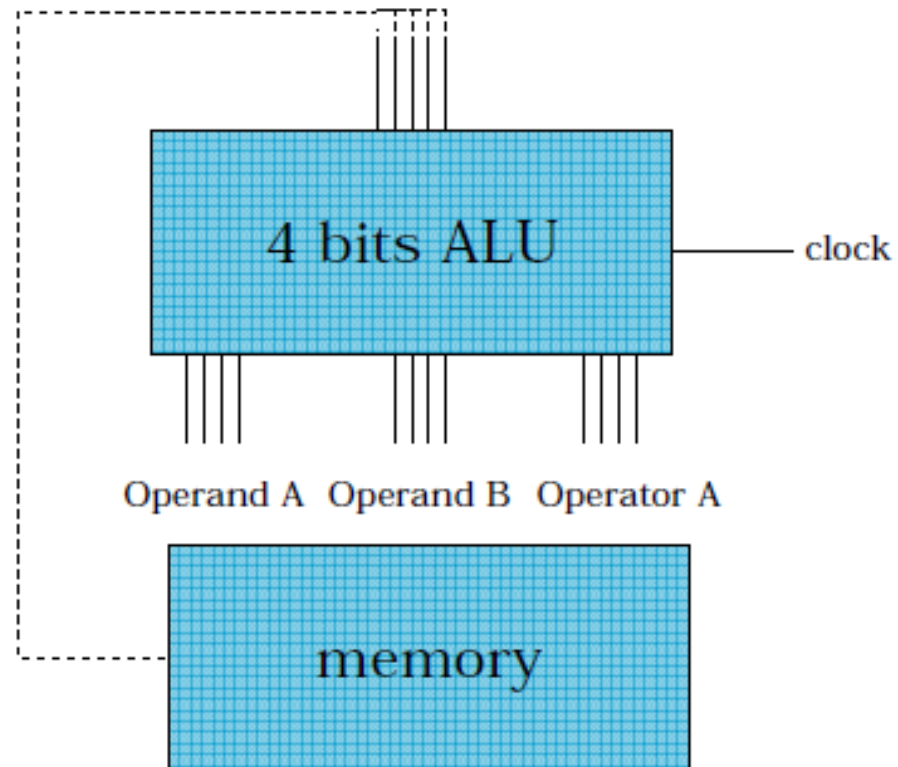


OP code	
0000	+
0001	-
0010	X
0011	/
0100	DCL
0101	MOV



# Machine Cycle

## ■ With memory



# Machine Cycle

- Clock rate
  - The fundamental rate in cycles per second at which a computer performs its most basic operations such as adding two numbers or transferring a value from one register to another
- Machine cycle
  - The four steps which the CPU carries out for each machine language instruction: **fetch, decode, execute, and store**.  
These steps are performed by the control unit, and may be fixed in the logic of the CPU or may be programmed as microcode which is itself usually fixed (in ROM) but may be (partially) modifiable (stored in RAM)

# Machine Cycle

■  $A=5+2-3$ ; (C language)

DCL A

DCL B

MOV B 5

ADD B 2

SUB B 3

MOV A B (assemble language)

4 00H

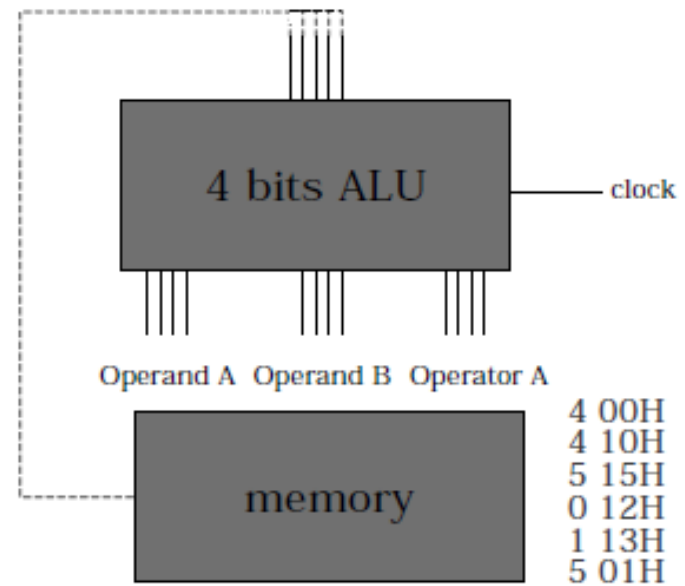
4 10H

5 15H

0 12H

1 13H

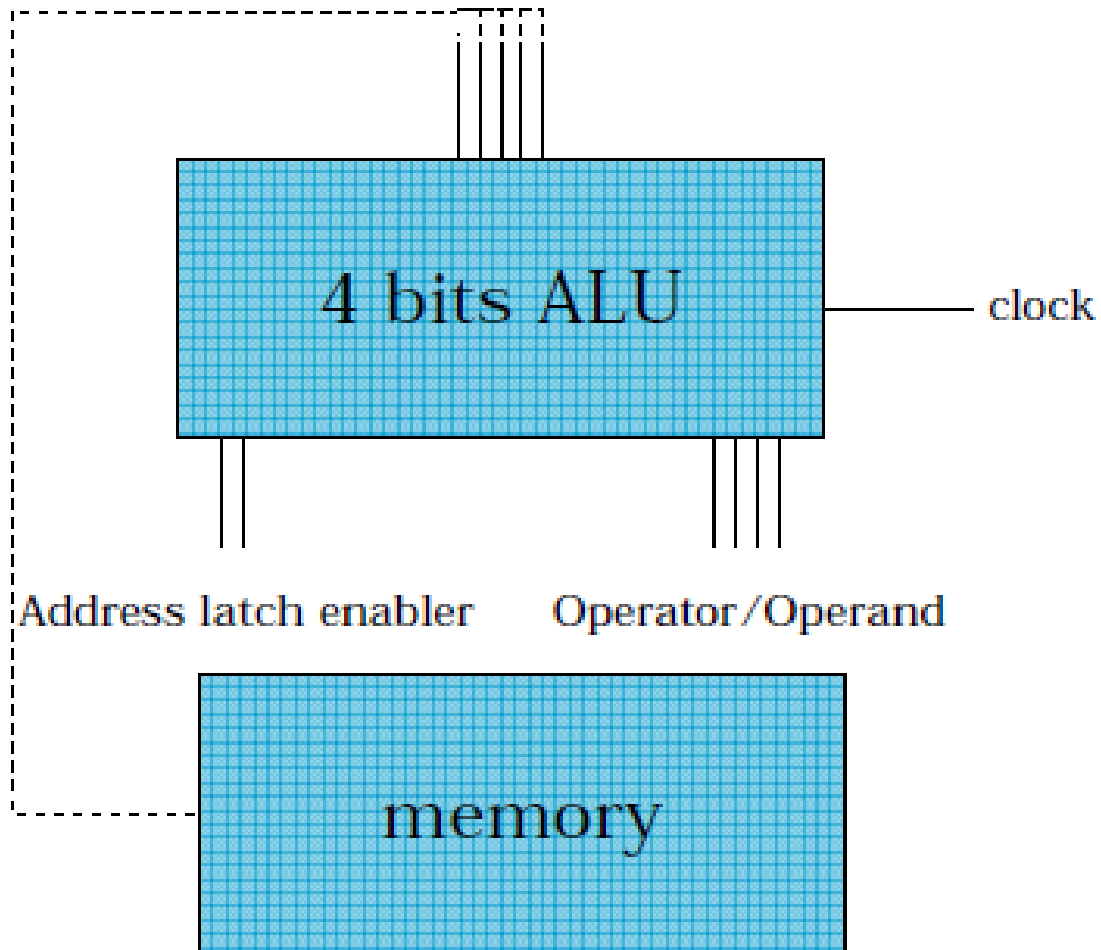
5 01H (machine code)



# Machine Cycle

- Machine cycle
  - One machine cycle
  - fetch, decode, execute, and store
  - To improve the efficiency of instruction execution (pipeline)
- Reduce the number of pins
  - Multiplexer

# Machine Cycle



Data也可以從外部讀取

# What else of CPU functions (pins) ?

- IO control pins
  - Interrupt
  - Timer
  - Counter
  - ...
- Memory control pins
  - Selector
  - ...
- Power control pins
  - ...

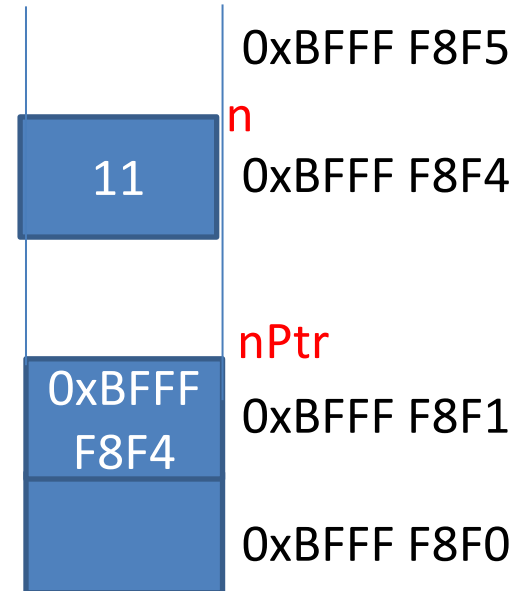
# C 語言的位址與數值 (非常重要! 務必弄懂!)

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(void)
6  {
7      int n = 11;
8      cout << n << endl;
9
10     int *nPtr = &n;
11     cout << nPtr << endl;
12
13     int t = *nPtr;
14     cout << t << endl;
15
16     return 0;
17 }
18
19 /* 《程式語言教學誌》的範例程式
20    http://pydoing.blogspot.com/
21    檔名: pointerdemo.cpp
22    功能: 示範 C++ 程式
23    作者: 張凱慶
24    時間: 西元 2012 年 10 月 */
```

n存的是數值

nPtr存的是位址

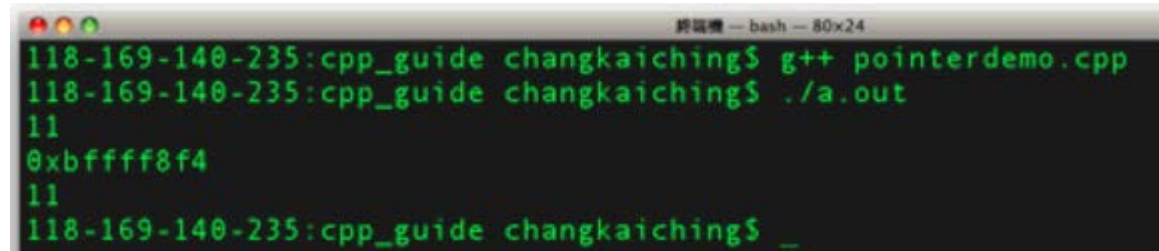
\*nPtr存的是數值



# C 語言的位址與數值

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(void)
6  {
7      int n = 11;
8      cout << n << endl;
9
10     int *nPter = &n;
11     cout << nPter << endl;
12
13     int t = *nPter;
14     cout << t << endl;
15
16     return 0;
17 }
18
19 /* 《程式語言教學誌》的範例程式
20    http://pydoing.blogspot.com/
21    檔名: pointerdemo.cpp
22    功能: 示範 C++ 程式
23    作者: 張凱慶
24    時間: 西元 2012 年 10 月 */
```

執行結果



```
118-169-140-235:cpp_guide changkaiching$ g++ pointerdemo.cpp
118-169-140-235:cpp_guide changkaiching$ ./a.out
11
0xbffff8f4
11
118-169-140-235:cpp_guide changkaiching$ _
```

n存的是數值

nPter存的數值是n的位址

\*nPter存的是數值



# 組合語言

# 暫存器(Register)

- 暫存器（ **Register** ），是中央處理器內的其中組成部份。暫存器是有限儲存容量的高速儲存元件，它們可用來暫存指令、資料和位址。
- 在中央處理器的控制部件中，包含的暫存器有指令暫存器（ **IR** ）和程式計數器。
- 在中央處理器的算術及邏輯部件中，包含的暫存器有累加器。

# Types of Registers

- 資料暫存器(data register)

用來儲存整數數字（參考以下的浮點暫存器）。在某些簡單（或舊）的CPU，特別的資料暫存器是累加器，作為數學計算之用。

- 位址暫存器(Address registers)

持有記憶體位址，以及用來存取記憶體。在某些簡單/舊的CPU裡，特別的位址暫存器是索引暫存器（可能出現一個或多個）。

- 通用目的暫存器General-purpose registers (GPRs)

可以儲存資料或位址兩者，也就是說他們是結合 資料/位址 暫存器的功用。

- 浮點暫存器Floating-point registers (FPRs):

用來儲存浮點數字。

- 常數暫存器(Constant registers)

用來持有唯讀的數值（例如0、1、圓周率等等）。

# Types of Registers

- 向量暫存器(Vector registers)

用來儲存由向量處理器執行[SIMD](#)指令所得到的資料。

- 特殊目的暫存器(Special-purpose registers (SPRs))or (Special-function registers (SFRs))

儲存CPU內部的資料，像是[程式計數器](#)（或稱為[指令指標](#)），[堆疊暫存器](#)，以及[狀態暫存器](#)（或稱微處理器狀態字組）。

- [指令暫存器](#) - 儲存現在正在被執行的指令
- [變址暫存器](#) - 是在程式執行時用來更改運算元位址之用。
- 在某些架構下，模式指示暫存器儲存和設定跟處理器自己有關的資料。由於他們的是附加到特定處理器的設計，因此他們並不被預期會成微處理器世代之間保留的標準。
- 有關從[隨機存取記憶體](#)提取資訊的暫存器與CPU（位於不同晶片的儲存暫存器集合）
  - [記憶體緩衝暫存器](#)
  - [記憶體資料暫存器](#)
  - [記憶體位址暫存器](#)
  - [記憶體型態範圍暫存器](#)

- ex: **movs r1, #20** //把數值 20 的內容存入暫存器r1  
運算碼 運算元 //註解
- (1)運算碼(OP CODE):是由助憶符號寫成，代表指令。
- (2)運算元(OPERAND):即運算碼所要處理的資料。個數: 0 - 3 個。
- (3)標記(LABEL)與註解: 給人看的, 沒有實際作用。
- (4)註解: 程式之後必須用雙斜線(//)與運算碼區分。
- (5)標記: 新的一行開頭用分號(;), 之後的內容即為註解。

# ARM-M4 的組合語言

- 如何存取資料, 像是matlab →variable, C→設定變數如int, double等
- 0x30 和 30H 和30 Hexadecimal是一樣的意思

# Register使用 #

- 若是運算元前面加上井號# (例如：#0x55, #AA)，表示他是一個“值”
- Ex1: `.equ AA, 0x55`  
`movs r2, #AA`  
Ex2: `movs r1, #85`

# Register使用 =

- 若是運算元前面加上等號=(例如：=L1)，表示取出該記憶體空間“位址”

```
LDR    R3,=MY_NUMBER    ; Get the memory location of MY_NUMBER
LDR    R4, [ R3]          ; Read the value 0x12345678 into R4
```

- 此時R3這個reg內存一個位址，  
ex: 0x800022c

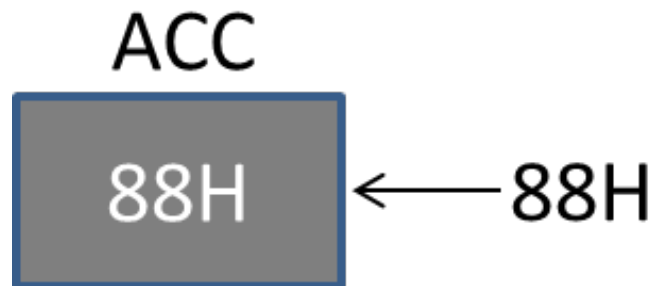


# Register使用 []

- 若是運算元加上兩個中括號(例如：[L1])，表示所使用的為該記憶體空間中儲存的值
- Ex:  
ldr r1, =str  
ldr r2, [r1]

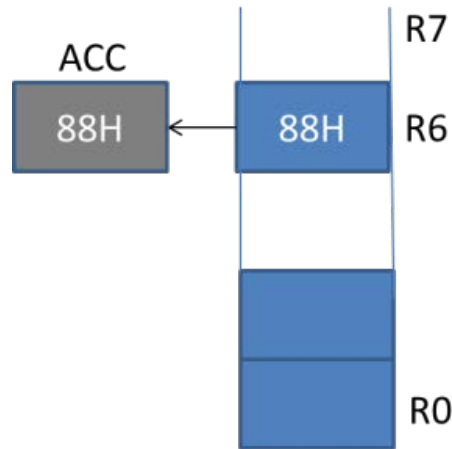
# 1. 立即定址法 ARM-M4

- 指令來源是一立即資料(常數)，而非位址或者暫存器
- C語言: `a = 123;`
- `MOV A, #0x88` ;把一個常數88H存入累加器A。“#”是常數前置資料



## 2. 暫存器定址法 ARM-M4

- 暫存器  $\leftarrow$  暫存器
- C語言: `b=123; a = b;`
- `MOV A,R6` ;把暫存器R6的內容存入暫存器A。



- 8051 的RAM的每個暫存器庫均含有8個暫存器，稱為R0-R7，ARM-M4則有R0-R12

### 3. 暫存器間接定址法 ARM-M4

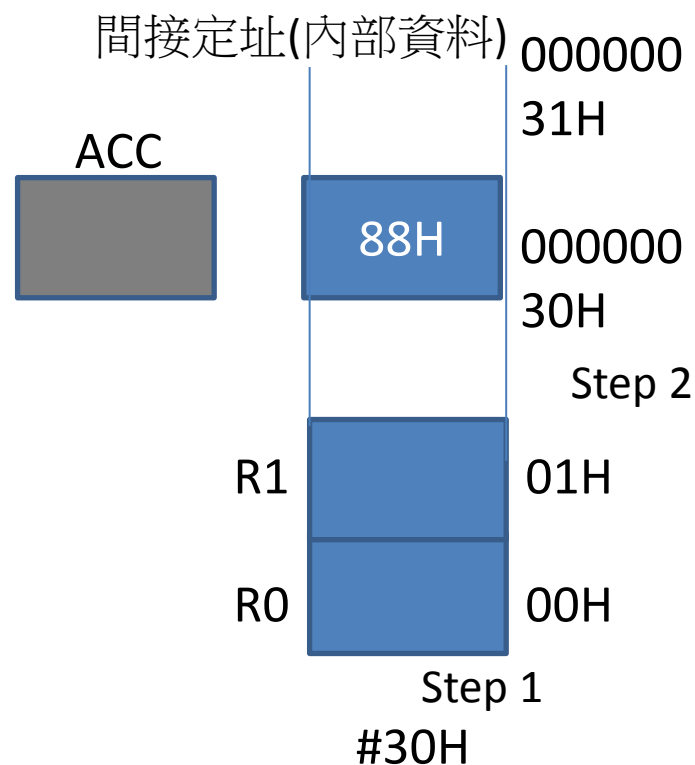
- 把運算元的位址存放在一個暫存器，當作運算元位址的指標(pointer)作為指令來源。符號：“[]”。
- 累加器  $\leftarrow$  \*位址
- 跟C++裡面的pointer很像 (C語言: \*ptrC=&nb; na= \*ptrC; )
- Ex:

MOV R0, #0x00000030

; R0儲存常數0x30亦即30H

LDR A, [R0];

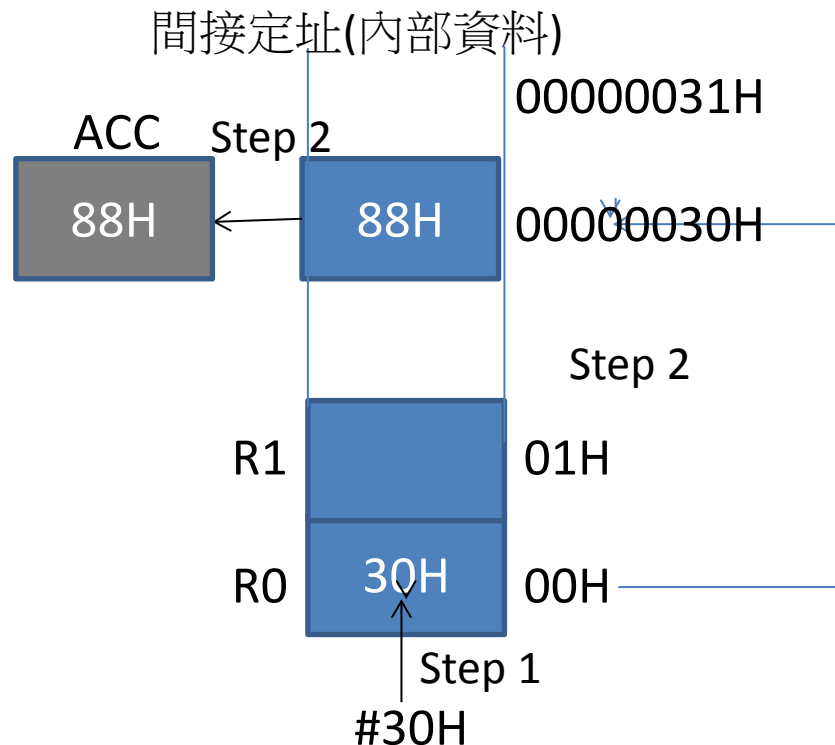
到這裡，若30H位址內存是88H  
，A裡面的值是多少？



### 3. 暫存器間接定址法 ARM-M4

MOV R0, #0x30 ; R0儲存常數30H

LDR A,[R0] ;把 RAM 位址30H 的內容存入累加器 A。所以結果A裡面的值是88H



## 4. 索引|定址法 ARM-M4

- 以一個基底暫存器的內容，再加上一個索引暫存器的內容，所得的值即是運算元所在的位址。
- ex:  
MOV A, #0x00000030 ;索引值為0x00000030  
LDR R8, [A, #0x20] ;將記憶體位址  
;0x00000050H(0x00000030+0x20)的  
;內容存入R8。在這裡他是上移0x20 bytes
- C語言: \*ptrC1=&na; nb = \*(ptrC1+ptrC2);
- 8051/8052的索引|定址法僅適用於ROM (程式記憶體)，而且read only。

## 5. 位元定址法ARM-M4

- 對RAM及特殊功能暫存器(SFR)的某個位元直接設定或清除，只能使用於可位元定址的暫存器。例如：
- BSRR ;
- BRR ;

# ARM Assembly Code



# ARM Assembly Code

- Assembly code 可以分為三個部分  
  **.data**部分，  
  **.text**部分，  
  **main**部分。
- **.data** 數據部分 數據段用來聲明初始化數據或常數。這個數據在運行時不發生變化。可以聲明各種常數值，文件名，或緩衝器大小等。

```
.data  
    str: .asciz "Hello World!"
```

- **.text**部分是用於聲明變量。

```
.text  
X: .word 100  
.global main  
.equ AA, 0x55
```

- **main:** 是程式指令

```
1      .syntax unified
2      .cpu cortex-m4
3      .thumb
4
5      .data
6
7      str: .asciz "Hello World!"
8      .text
9      X: .word 100
10     .global main
11     .equ AA, 0x55
12
13
14     main:
15
16     movs r0, #AA
17     movs r1, #20
18     adds r2, r0, r1
19
```

有些宣告是segment .text有些是 .text 有些是 section. Text

照LinkerScript.ld中的宣告，LinkerScript.ld宣告.text就是.text，宣告segment .text 就是segment .text 。

- 如何知道有沒有bug? 看register
- 如何知道程式執行成功? 看register
- LD1紅綠閃爍表示debugging，綠燈是正常連接debug結束，紅燈是尚未連接

# MOV 家族

學習看 programming manual 指令集與ARM cortex-M4架構參考文件

- MOV<sub>S</sub> R11, #0x000B ; write value of 0x000B to R11, flags get updated
- MOV R1, #0xFA05 ; write value of 0xFA05 to R1, flags not updated
- MOV<sub>S</sub> R10, R12 ; write value in R12 to R10, flags get updated
- MOV R3, #23 ; write value of 23 to R3
- MOV R8, SP ; write value of stack pointer to R8

# MOV 家族

- Flag update 會到APSR中，最好update，之後code使用方便

## 1. mov的用法

在ARM体系中，mov只能用于数据在寄存器之间的移动或者往寄存器中写入立即数。格式如下：**mov{条件}{s} 目的寄存器，源操作数**

```
1  MOV      R1,R2      ;R1=R2
```

Reg和reg之間的assign, 或者#數字

## 2. ldr的用法

LDR是将内存中的数载入到寄存器，LDR可以载入立即数。格式如下：**LDR 目的寄存器，源**

```
1  LDR      R1,=0xE0000000 ;R1=0xE0000000
2  LDR      R1,0xE0000000  ;将内存中地址为0xE0000000的内容载入到R1
3  LDR      R1,[R0]        ;将R0中的数所指定的地址的内容传输到R1
```

把内存資料，内存位址等讀入reg中

## 3. str的用法

STR是将寄存器中的数字载入内存。格式如下：**STR{条件} 源寄存器，<存储器地址>**

```
1  STR      R1,[R0]      ;将R1中的内容传输到R0中的数所指定的地址的内存中去
```

# MUL

- $MUL\{S\}\{cond\} \{Rd,\} Rn, Rm$  ; Multiply  $MLA\{cond\} Rd, Rn, Rm, Ra$  ; Multiply with accumulate  $MLS\{cond\} Rd, Rn, Rm, Ra$  ; Multiply with subtract
- Where:
- ‘*cond*’ is an optional condition code (see [Conditional execution on page 64](#)). • ‘*S*’ is an optional suffix. If *S* is specified, the condition code flags are updated on the result of the operation (see [Conditional execution on page 64](#)). • ‘*Rd*’ is the destination register. If *Rd* is omitted, the destination register is *Rn*. • ‘*Rn*’, ‘*Rm*’ are registers holding the values to be multiplied. • ‘*Ra*’ is a register holding the value to be added to or subtracted from.
- MUL只能夠register相乘，不能夠乘數字

# 本章重點

- ARM記憶體位址
- C語言和組合語言中的位址和值的搬動