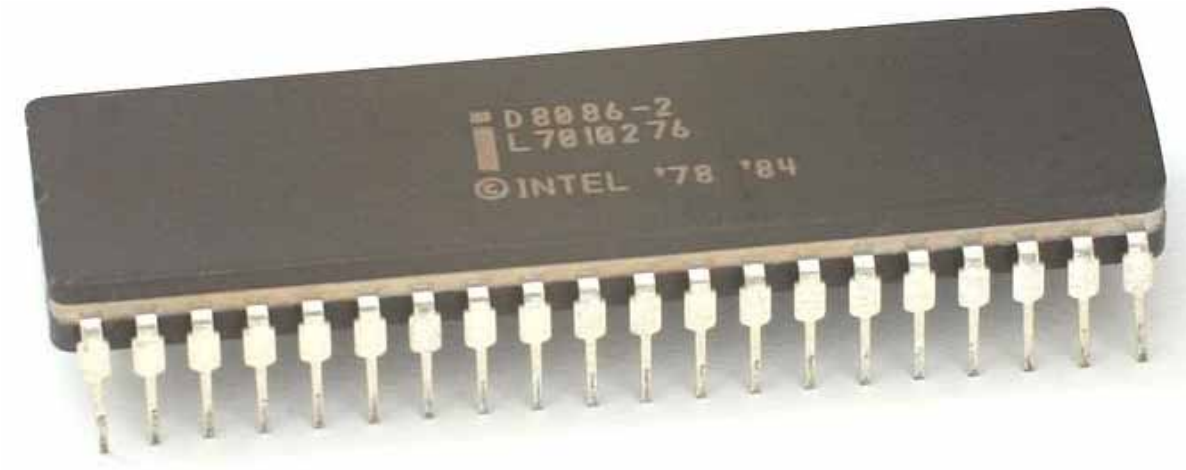
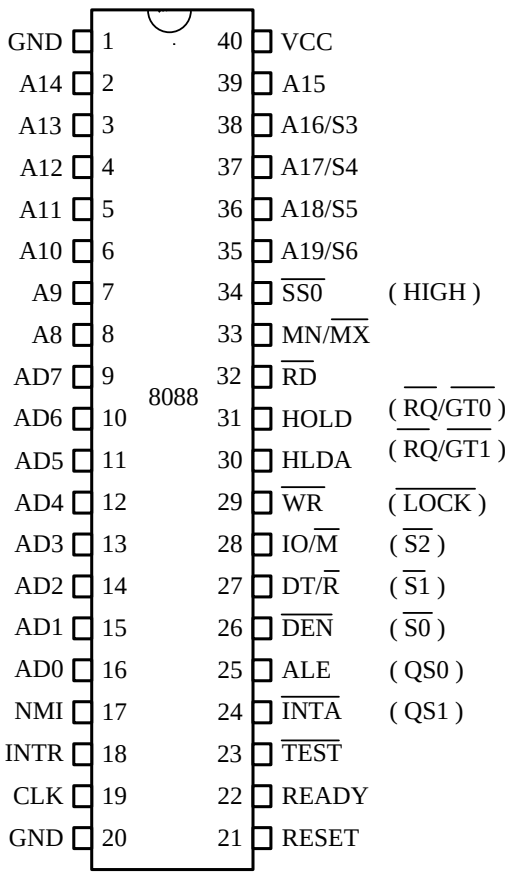
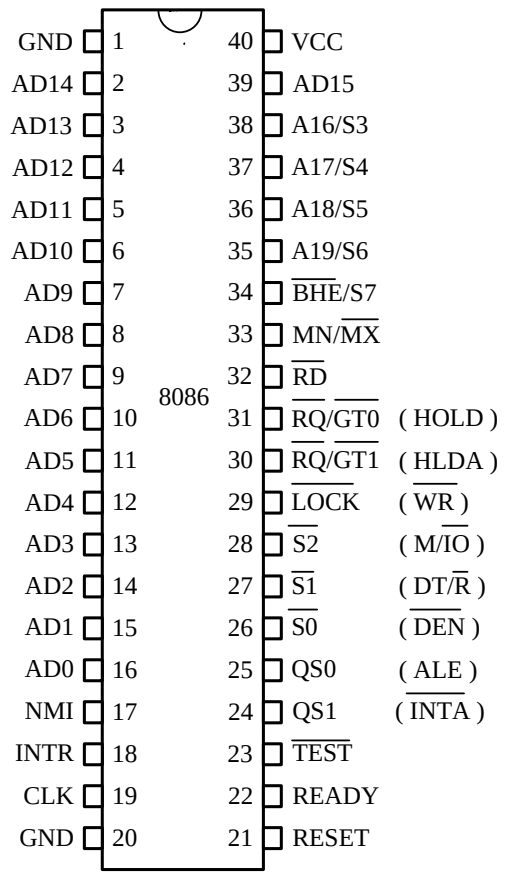
Intel 8088/86 CPU 外觀及接腳圖







組合語言直譯器-----DEBUG

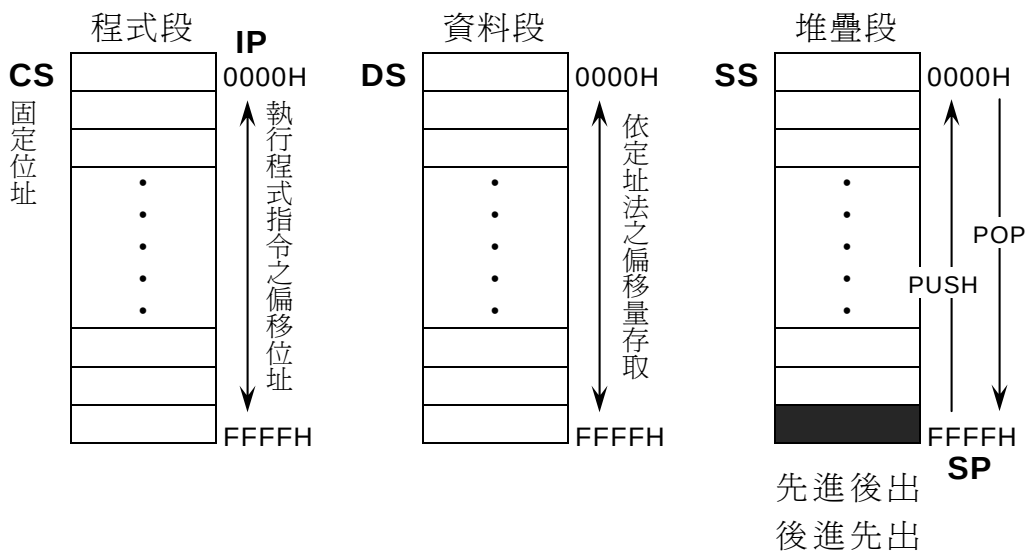
一、Debug 下的旗號狀態表示法

(記錄相關指令執行後的狀態)

旗號名稱	代碼表示法		狀態說明	
	1 狀態	0 狀態		
Overflow	OV	NV	溢位	未溢位
Direction	DN	UP	遞減	遞增
Interrupt	EI	DI	致能	抑制
Sign	NG	PL	負數	正數
Zero	ZR	NZ	零值	非零值
Auxiliary	AC	NA	有半進位	無半進位
Parity	PE	PO	偶數個 1	奇數個 1
Carry	CY	NC	有進位	無進位

二、真實模式下的 RAM 記憶體配置(1MB)

(各區段的大小為 64KB)



三、Debug 使用原則

1. 程式起始位址必(即 IP 值)須爲 0100H。
2. 輸入的數值均被視爲十六進數(即數字後不必加 H)。
3. 不要更改區段及指標暫存器的內容(IP 除外)。
4. 檔案附名爲「.COM」。
5. 最後一指令爲「INT 3」，用來結束程式執行。
6. 最後一指令爲「INT 20」，用來返回 DOS。

四、Debug 常用命令

1. 顯示暫存器內容 ---------- **R**

```
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AE0  ES=0AE0  SS=0AE0  CS=0AE0  IP=0100   NV UP EI PL NZ NA PO NC
```

區段暫存器的內容在 Debug 下均相同，其值由系統安排，各電腦之間或本機每次啓動後會有所不同。

2. 更改暫存器(如 AX)內容------- **R** AX

```
-R AX
AX 0000      (AX 原內容)
:1234        (輸入新值)
-
```

3. 更改旗號狀態 ---------- **R** F

```
-R F
NV UP EI PL NZ NA PO NC   -OV ZR  (輸入更改的旗號代碼)
-
```

4. 更改記憶體內容 ---------- **E** Addr

```
-E 101                    (原內容)
0B24:0101  16.20          (新內容)
-
```

5. 查看記憶體內容 ---------- **D** Addr

6. 儲存程式 ---------- 假設程式的起迄位址爲 0B24:0100～0B24:0123，操作步驟如下

(1) 設定暫存器 BX 內容爲 0000，CX 內容爲 24。 (二者之值即爲程式大小)

(2) **N** C:\ASM\PROG1.COM (爲檔案命名及儲存路徑)

(3) **W** (存檔)

7. 離開 DEBUG ---------- **Q**

8. 載入程式(Debug 下) ---------- (1) **N** C:\ASM\PROG1.COM　(檔案位置路徑及名稱)
(2) **L**　(載入)
進入 Debug 並載入程式 ------- Debug C:\ASM\PROG1.COM

9. 查看程式(反組譯) ---------- **U** 100 12F　(顯示偏移位址 0100～012F 間之指令)
U 100　(未指定結束位址，顯示範圍爲 0100～011F)

10. 程式輸入 ---------- **A** 100　(起始位址必須爲 0100)

```
-A 100
0AE0:0100  MOV  AL,25
0AE0:0102  MOV  BL,37
0AE0:0104  ADD  AL,BL
0AE0:0106【Enter】
-
```

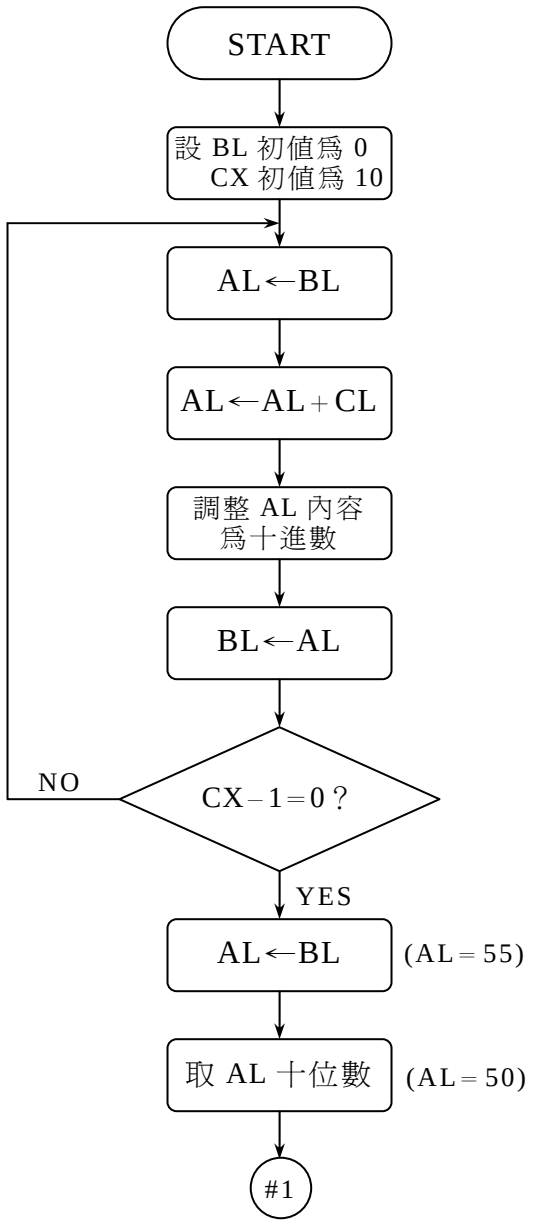
【以單步執行指令 T 執行並觀察暫存器狀態】

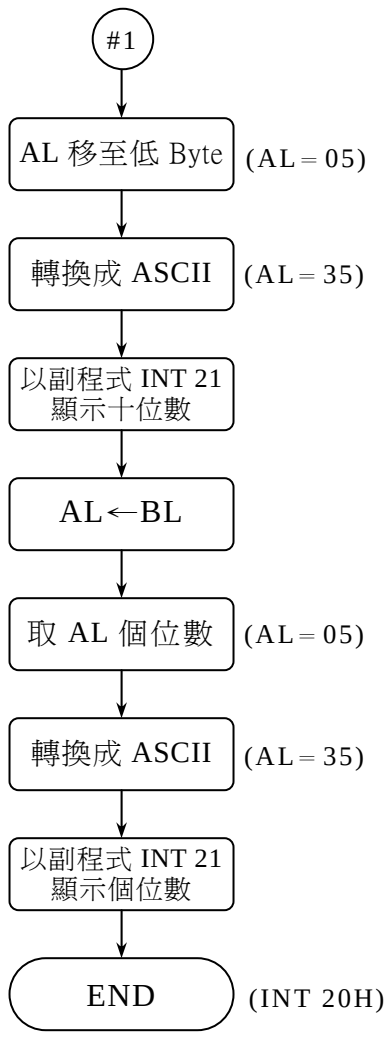
11. 單步執行 ---------- **T**　(Trace，執行 CS:IP 所在的一個指令後停止)
P　(Process，碰到系統副程式 INT xx 時，不要以 **T** 進入，以免出不來，應以 P 命令，將副程式當成一個指令執行之)

12. 執行程式 ---------- **G**　(自 CS:IP 所指的位址開始執行)
G 106　(自 CS:IP 所指的位址開始執行，至 CS:0106 處停止)
G=0AED:0100　(自指定的位址開始執行，假設 CS=0AED)

五、程式練習(計算並顯示 1 至 10 之總和)

```
0AE0:0100  MOV   BL,0
0AE0:0102  MOV   CX,0A
0AE0:0105  MOV   AL,BL
0AE0:0107  ADD   AL,CL
0AE0:0109  DAA
0AE0:010A  MOV   BL,AL
0AE0:010C  LOOP  105
0AE0:010E  MOV   AL,BL
0AE0:0110  AND   AL,F0
0AE0:0112  MOV   CL,4
0AE0:0114  SHR   AL,CL
0AE0:0116  ADD   AL,30
0AE0:0118  MOV   DL,AL
0AE0:011A  MOV   AH,2
0AE0:011C  INT   21
0AE0:011E  MOV   AL,BL
0AE0:0120  AND   AL,0F
0AE0:0122  ADD   AL,30
0AE0:0124  MOV   DL,AL
0AE0:0126  MOV   AH,2
0AE0:0128  INT   21
0AE0:012A  INT   20
```





80x86 組合語言定址法範例【假設資料區段暫存器 DS 為 1234H】

【使用 DEBUG 練習各種定址法，以 INT 3 結束程式執行】

一、立即定址法：不用配合 DS，資料存在該指令的操作碼後面(即資料在程式段內)。

例 1：MOV BL，12　　例 2：MOV AX，1234H　　例 3：INT 12H

例 4：Counter　EQU 20H
　　MOV　　AL，Counter

二、直接定址法： DS：[nnnn]

例 1：MOV AX，[1438H]

實際位址＝12340H＋1438H
AL←[13778H]
AH←[13779H]

例 2：Val　DB　20H，30H
MOV AX，Val

設變數 Val 的位址爲 5678H
實際位址＝12340H＋5678H
AL←[179B8H]←20H
AH←[179B9H]←30H

三、暫存器定址法：速度最快，不用配合 DS，沒有匯流排存取週期。

例 1：MOV BX，AX　　例 2：INC SI

四、暫存器間接定址法： DS：[暫存器]　【可使用的暫存器為 BX、SI、DI】

例 1：MOV CX，[SI]
設 SI＝1A00H，實際位址＝12340H＋1A00H＝13D40H，結果　CL←[13D40H]
CH←[13D41H]

例 2：MOV CX，[DI] ------有效位址計算法同上
例 3：MOV AX，[BX] -----有效位址計算法同上

五、索引(指標)定址法： DS：SI　DS：DI 須另加偏移位址，否則屬暫存器間接定址

例：MOV DX，20H[DI] 或 MOV DX，[DI] 20H 或 MOV DX，[DI]＋20H 或MOV DX，[DI＋20H]
設 DI＝6789H，實際位址＝12340H＋6789H＋20H＝18AE9H，結果　DL←[18AE9H]
DH←[18AEAH]

六、基底相對定址法： DS：BX　須另加偏移位址，否則屬暫存器間接定址

例：MOV AX，1438H[BX] 或 MOV AX，[BX]1438H 或 MOV AX，[BX]＋1438H 或
MOV AX，[BX＋1438H]
設 BX＝6789H，實際位址＝12340H＋6789H＋1438H＝1AF01H，結果　AL←[1AF01H]
AH←[1AF02H]

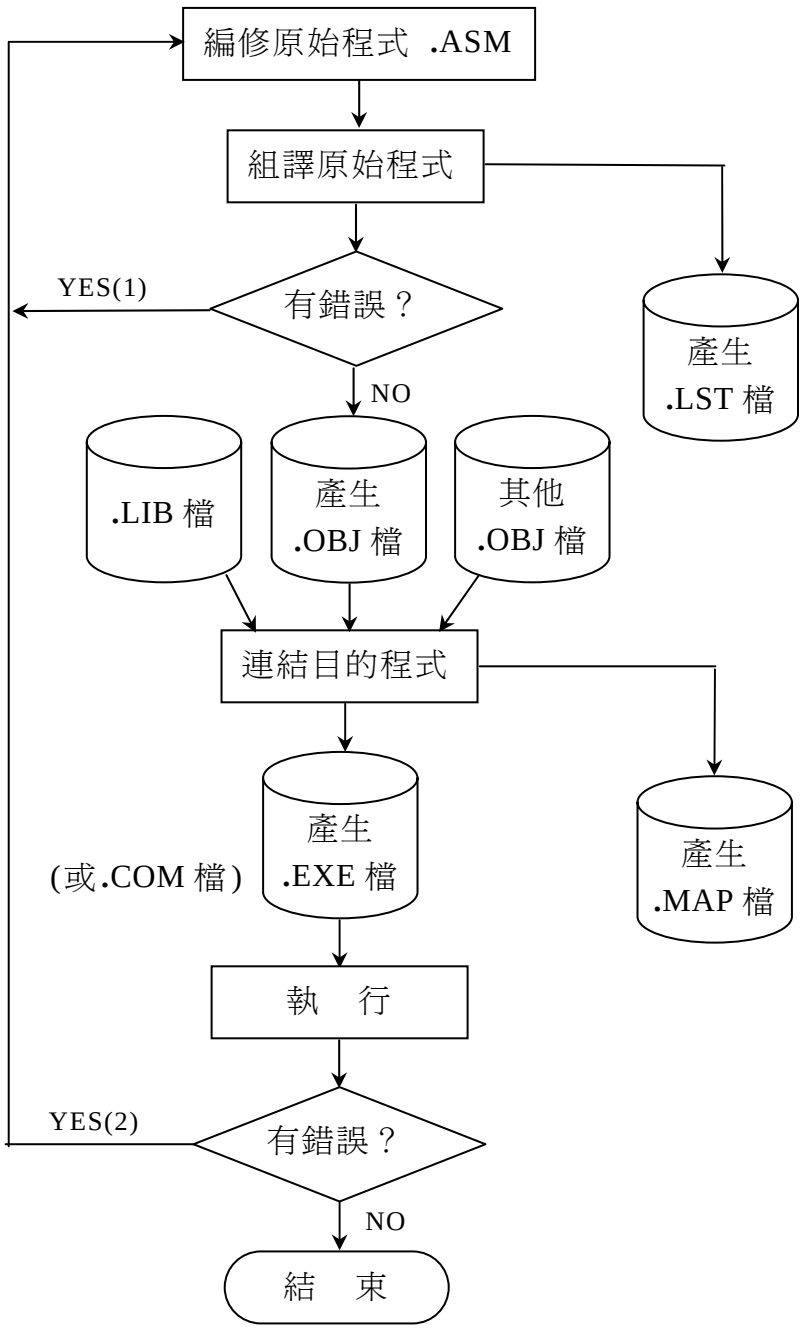
七、基底索引定址法： DS：[BX＋SI＋OFFSET]　DS：[BX＋DI＋OFFSET]

例 1：MOV CX，[BX＋SI] 或 MOV CX，[BX][SI]　(OFFSET可有可無)
設 BX＝5C86H、SI＝1200H，實際位址＝12340H＋5C86H＋1200H＝191C6H
結果 CL←[191C6H]　CH←[191C7H]

例 2：MOV AX，[BX＋DI]20H　實際位址＝DS＋BX＋DI＋20H

八、隱含定址法：指令無運算元，如 DAA、DAS、STC、CLI、NOP

組譯與連結(MASM 6.11 版)



(1) Syntax error
(2) Logical error ＆ Run-Time error

組譯/連結程式的說明
在最後一頁

◆ 自編批次檔 ASM.BAT 內容：

```
ML /FL /FM %1.ASM
DEL %1.OBJ
DEL %1.LST
DEL %1.MAP
```

◆ 使用法：ASM prog

組合語言結構與虛指令

一、指令結構---------包含四個欄位

```
標記：    運算碼     運算元        ；註解
Label：   Operating  Operand       ; Comment
          xor        ax，ax        ；Clear ax
          mov        cx，10
Again：
          add        ax，cx
          dec        cx            ；cx－1
          nop
          jnz        Again
```

◆ 標記不一定有，若程式中有跳躍或副程式，則必須以參考標記替代實際目的地位址。
◆ 運算碼、操作碼、又稱爲助憶碼(Mnemonic)。爲組合語言中的必要元素。
◆ 少數指令沒有運算元(隱含定址法)。
◆ 註解欄可有可無。

1. 註解欄係增加程式可讀性，組譯時略過，亦可置於兩指令間，說明下段程式功能。
2. 標記必須置於第一格，運算碼前必須有空格，否則組譯時會被視爲標記而產生錯誤，其後各欄位間至少空一格，。
3. 標記最好自成一列，以便插入指令。名稱須以英文字母開頭，大小寫無不同，不可使用「保留字」。一般係供跳躍指令當目的指標，其命名宜具有意義。

二、虛擬指令、假指令 -------- 非給 CPU 執行的指令，專給「組譯器」看，沒有對應的機械碼

1. EQU：Equal 的意思，定義一數值常數，例如「Number equ 30h」，不佔記憶空間。若程式中某常數須使用多次，異動時不須一一更改。既然爲常數，不可在程式中被改變。

2. DB(Define Byte)：定義一以位元組爲單位的變數，該變數名稱代表記憶體的某個起始位址，可存取資料

   Val_1　db　30h ---------- 變數內容初始值爲 30h。

   Val_2　db　? ---------- 僅保留 1Byte 記憶空間給變數，未設初始值

3. DW(Define Word)：定義一個以字組(2Byte)爲單位的變數，用法如 DB。

注意：使用 EQU 定義來取資料的為「立即定址法」，以 DB 方式來存取資料的為「直接定址法」

6. ORG(Original)：設定程式碼存放的起始位址。

7. OFFSET：如果我們要取出已定義好的字串變數資料，須先指向其起始位址，寫法爲「mov dx,offset mess」或「lea dx,mess」，則存放該字串的起始位址被置於 dx 暫存器中。
   LEA 的意思爲 Load Effective Address。

三、微程式

CPU 所有的指令未必全用硬體線路製造出來，原先可能只造出一些基本指令，工程師再利用這些基本指令寫出複雜指令的處理程式，這些處理複雜指令的程式即稱爲微程式(Microprogram)，該程式放在 CPU 裡面的 ROM 中。

80x86 常用指令【使用 DEBUG 練習指令功能，以 INT 3 結束程式執行】

一、MOV：將來源運算元的資料搬移(複製)至目的運算元。【不影響任何旗號】

1. 目的運算元與來源運算元的長度必須相同，例如：

```
MOV  AX,1234H (AH＝12H、AL＝34H)
MOV  AH,12H
MOV  AX,12H (AH＝00、AL＝12H)
MOV  AH,1234H (不可以)
```

2. 記憶位址中的資料不能直接相互搬移，例如 MOV [1000H],[2000H]要改成 MOV ax,[2000H]
   MOV [1000H],ax

3. 資料搬入區段暫存器須透過一般暫存器，例如 MOV DS,100H 要改成 MOV BX,1000H
   MOV DS,BX

   MOV CS,DS 要改成 MOV AX,DS
   MOV CS,AX

二、ADD：將目的與來源運算元的值相加，結果存回目的運算元。

【旗號影響 O、S、Z、A、P、C】

1. 目的運算元的長度必須大於或等於來源運算元，例如：

```
ADD  AX,1234H (AH←AX＋1234H)
ADD  AX,12H (AX←AX＋0012H)
ADD  AH,1234H (不可以)
ADD  AL,1234H (不可以)
ADD  AL,BX (不可以)
ADD  BL,12H (BL←BL＋12H)
ADD  AX,CL (AX←AX＋CL)
```

2. 不合法的個格式有：記憶體加至記憶體，如 ADD [1000H],[2000H]、ADD Sum,Money
   以立即值爲目的運算元，如 ADD 1000H,[2000H]
   ADD KWH,20H (其中 KWH 爲已由 EQU 定義之常數)

三、ADC：與 ADD 指令類似，但會將「進位旗號 CF」一起加進來。

【旗號影響 O、S、Z、A、P、C】

四、INC：Increment，僅有目的運算元，將其內容加 1。 【旗號影響 O、S、Z、A、P、C】

五、DEC：Decrement，僅有目的運算元，將其內容減 1。 【旗號影響 O、S、Z、A、P、C】

六、SUB：Subtract，將目的與來源運算元的值相減，結果存回目的運算元。

【旗號影響 O、S、Z、A、P、C】

七、SBB：Subtract with Borrow，與 SUB 指令類似，但會將「進位旗號 CF」一起減。

八、AAA：僅有操作碼，將 AL 暫存器的內容調整為「非聚集十進位」。

1. 該指令必須置於加法指令(ADD 或 ADC)之後。
2. AAA 乃"Ascii Adjust after Addition"之簡稱，所謂「Unpacked BCD」就是以一個 Byte 來存放十進位數的一個位數(置於低 Nibble)。

   例如：AL=8、BL=5，執行 ADD AL,BL 後，AL=0DH
   再執行 AAA 後，AH=01、AL=03

   優點是容易轉換成ASCII，如 01H OR 30H即爲 1 的ASCII (31H)，03H OR 30H即爲 3 的ASCII (33H)
   缺點是會額外佔用到 AH 暫存器。

九、DAA：僅有操作碼，將 AL 暫存器的內容調整為「聚集十進位」。

1. 該指令必須置於加法指令(ADD 或 ADC)之後。
2. DAA 乃"Decimal Adjust after Addition"之簡稱，所謂「Packed BCD」就是以一個 Byte 來存放十進位數的兩個位數，十位數置於 AL 的高 Nibble，個位數置於 AL 的低 Nibble。

   例如：AL=8、BL=5，執行 ADD AL,BL 後，AL=0DH
   再執行 DAA 後，AL=13H

   缺點是轉換成 ASCII (31H)及(33H)較爲麻煩。

十、AND：將目的與來源運算元做 AND 邏輯運算，結果存回目的運算元。

在程式中常以遮罩的方式取出暫存器的某些位元，例如 AL 的內容原爲 34H，
執行 AND AL,0FH 後，AL=04H
執行 AND AL,0F0H 後，AL=30H　　【以 1 保留選擇的位元而清除其他位元】

十一、OR：將目的與來源運算元做 OR 邏輯運算，結果存回目的運算元。

設定暫存器的某些位元爲 1 而保留其他位元，例如 AL 的內容原爲 34H，
執行 OR AL,0FH 後，AL=3FH
執行 OR AL,0F0H 後，AL=F4H

十二、XOR：將目的與來源運算元做 XOR 邏輯運算，結果存回目的運算元。

功能 1：清除暫存器的內容爲 0，如 XOR AL,AL (相當於 MOV AL,0)
功能 2：判斷兩暫存器中有那些位元不同，如 AL=10011011B、BL=00110010B，經 XOR AL,BL，AL=10101001，反應兩暫存器的 Bit0、Bit3、Bit5、Bit7 不同。
功能 3：將暫存器中某些位元反相，如 XOR AL,00001111B，則 AL 的低四位元反相，其餘不變。

十三、NOT：將目的運算元的內容全部反相(1 的補數)，如NOT AX、NOT BL。

十四、NEG：將目的運算元的內容變成 2 的補數，如NEG AX、NEG BL。

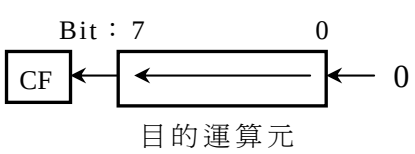
十五、TEST：以來源運算元去測試目的運算元，運算法如同 AND，但不會改變目的運算元的值。

1. 該指令後通常跟著「JZ」、「JNZ」等有條件相對跳躍指令。
2. 例如要測試 AL 暫存器的 Bit6 是否爲 1，寫法爲 TEST　AL,40H
   　　JNZ　Label
   若二者 AND 的運算結果不爲 0，則 ZF=0，表示 AL 的 Bit6=1，跳至 Label 處執行。

十六、CMP：將目的運算元與來源運算元的內容做比較。

與 SUB 相當類似，但不會影響目的運算元，只會影響旗號，做爲條件跳躍的測試指令。
例如要判斷 AL 暫存器的內容若等於 12H，則產生跳躍，寫法爲 CMP　AL,12H
　　JZ　Label

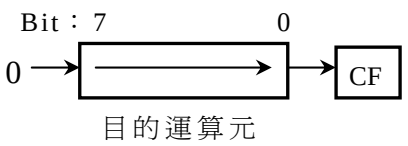
十七、SHL：將將目的運算元內容左移，依來源運算元的值左移一位或多位。



◆ 來源運算元屬移位計數值，若計數值爲 1，可採立即定址法，2 次以上則必須使用 CL 暫存器，如 SHL AL,1　　MOV CL,5
　　SHL AL,CL

◆ 以該指令做某些特別乘法時(2^n倍)，其速度會比乘法指令MUL快。

十八、SHR：將將目的運算元內容右移，依來源運算元的值左移一位或多位。

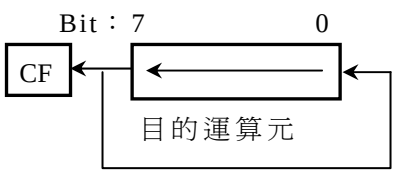


◆ 來源運算元屬移位計數值，若計數值爲 1，可採立即定址法，2 次以上則必須使用 CL 暫存器，如 SHR AL,1

```
MOV CL,5
SHR AL,CL
```

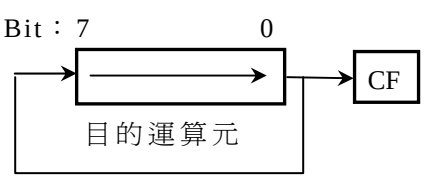
◆ 以該指令做某些特別除法時(2^n倍)，其速度會比除法指令DIV快。

十九、ROL：將將目的運算元內容向左旋轉，依來源運算元的值左移一位或多位。



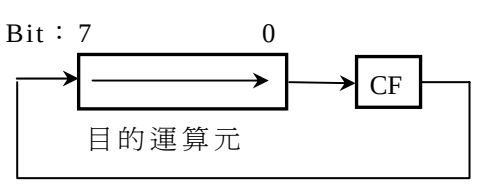
【來源運算元屬移位計數值，可為立即值或 CL 暫存器】

二十、ROR：將將目的運算元內容向右旋轉，依來源運算元的值左移一位或多位。



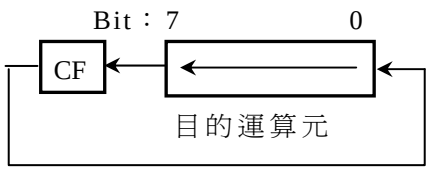
【來源運算元屬移位計數值，可為立即值或 CL 暫存器】

二十一、 RCR：將將目的運算元內容連同 CF 向右旋轉，依來源運算元的值左移一位或多位。



【來源運算元屬移位計數值，可為立即值或 CL 暫存器】

二十二、 RCL：將將目的運算元內容連同 CF 向左旋轉，依來源運算元的值左移一位或多位。



【來源運算元屬移位計數值，可為立即值或 CL 暫存器】

二十三、JMP：無條件跳躍，如同高階語言的「GOTO」。

1. 相對位址：跳躍距離－128～＋127 時，其運算元以一個 Byte 代表位移值。
   若距離超出上述值，但仍在同一區段內時，其運算元以兩個 Byte 代表位移值，跳躍距離爲－32768～＋32767
2. 絕對位址：若跳躍目的地在另一區段，運算元爲四個 Byte，兩個代表另一區段的 CS 值，兩個代表一區段的 IP 值。

【跳躍方式與距離係由組譯程式自動判斷計算】

二十四、Jxx：有條件的相對位址跳躍。

1. 利於程式載入時的重新定址(在程式載入至記憶體時，採跳躍距離方式不必重新安排位址)
2. 跳躍範圍限制：－128～＋127，超出時可以 JMP 指令輔助

指令格式	跳躍條件	指令格式	跳躍條件	指令格式	跳躍條件
JZ Lable	結果等於零	JC Lable	有進位(借位)	JG Lable	大於
JNZ Lable	結果不等於零	JNC Lable	無進位(借位)	JNG Lable	不大於(JLE)
JE Lable	等於	JNE Lable	不等於	JL Lable	小於

二十五、XCHG：將目的運算元與來源運算元的內容互換。

1. 兩個運算元的長度必須一致，不可都是記憶體。　　2. 不可有區段暫存器。

二十六、PUSH：將運算元的內容推入堆疊器(Stack)中存放。

暫存器必須爲 16 位元，不可用 PUSH AH、PUSH BL...等。

二十七、POP：將先前存放於堆疊器中的資料取回。

1. 不能 POP 到 CS 或 IP 暫存器，否則程式必將錯亂而當機。
2. PUSH 與 POP 使用時必須成雙成對(邏輯上)否則程式必將錯亂甚致當機。

二十八、LOOP：迴圈循環指令，須配合 CX 暫存器。

LOOP　Label------ 先將 CX 內容減 1，若不爲 0 則至 Label 處執行，　否則執行下一指令。

二十九、IN、OUT：

1. IN AL,240：將 I/O Port 240 位址中的資料載入 AL 暫存器中。

2. IN AX,DX：須先將 I/O Port 位址存放於 DX 暫存器中(因位址大於 255)，執行本指令時將低位址中的資料載入 AL 暫存器中，高位址中的資料載入 AH 暫存器中。

3. OUT 240,AL：將 AL 暫存器中的資料輸出到 I/O Port 240 位址中。

4. OUT DX,AL：須先將 I/O Port 位址存放於 DX 暫存器中(因位址大於 255)，執行本指令時將 AL 暫存器中的資料輸出至 DX 指定的位址中。

三十、　MUL：乘法運算(Multiply)

1. 位元組相乘範例：
```
MOV   BL,4
MOV   AL,5
MUL   BL ----- (AX=0014)
```
```
MOV   BL,12H
MOV   AL,34H
MUL   BL -----(AX=03A8)
```

2. 16Bit 相乘範例：
```
MOV   BX,1234H
MOV   AX,1122H
MUL   BX----- (DX=0137、AX=DEE8)
```

三十一、DIV：除法運算(Divide)

1. 除數(來源運算元)爲 8Bit 時，被除數必爲 AX，除完後的商放在 AL，餘數放在 AH
```
MOV   AX,34H
MOV   BL,5
DIV   BL--------(AX=020A)
```

2. 除數(來源運算元)爲 16Bit時，被除數必爲DX：AX，除完後的商放在AX，餘數放在DX
```
MOV   DX,0001H
MOV   AX,0006H
MOV   BX,10H
DIV   BX -------00010006H(65542D)÷10H(16D)=4096D 餘 6------ AX=1000H(4096D)、DX=0006
```

三十二、旗號設定指令：STC、CLC、STI、CLI

三十三、NOP：什麼事也不做(可作時間延遲微調指令，或在程式中預留替代指令之空間)

三十四、CBW：Convert Byte【AL】to Word(用於有正負數運算時，將運算值變成相同長度)

1.
```
MOV   AL,72H
CBW ------------AX=0072H
              (因爲原 AL 之 MSB=0)
```

2.
```
MOV   AL,83H
CBW ------------- AX=FF83H
              (因爲原 AL 之 MSB=1)
```

三十五、CWD：Convert Word【AX】to Double word(16Bit 轉成 32Bit)

1.
```
MOV   AX,6666H
CWD -------------- DX=0000、AX=6666H
              (因爲原 AX 之 MSB=0)
```

2.
```
MOV  AL,9999H
CWD--------------- DX=FFFF、AX=9999H
              (因爲原 AX 之 MSB=1)
```

DOS 中斷服務程式 INT 21H (工作編號在暫存器 AH 設定)

【從鍵盤讀取一字元並顯示於螢幕】

```
MOV     AH，1
INT     21H
```

鍵按下後立即回應，不必按[Enter]。
按鍵字元的 ASCII 碼存放於暫存器 AL。

【從鍵盤讀取一字元但不顯示於螢幕】

```
MOV     AH，8
INT     21H
```

鍵按下後立即回應，不必按[Enter]。
按鍵字元的 ASCII 碼存放於暫存器 AL。

【顯示一字元於螢幕】

```
MOV     DL，Char
MOV     AH，2
INT     21H
```

Char 爲該字元的 ASCII 或以立即定址法，如'C'。
MOV DL,43H 或 MOV DL,'C'

【顯示一字串於螢幕】

```
LEA     DX，String
MOV     AH，9
INT     21H
```

String 爲該字串的起始位址，必須先予以定義。
也可寫成：MOV DX，OFFSET String
字串結尾符號爲'$'

【自鍵盤讀取一字串&顯示於螢幕】

```
LEA     DX，Buffer
MOV     AH，0AH
INT     21H
```

字元輸入直到按下[Enter]鍵爲止。
自訂變數 Buffer 爲存放鍵盤輸入的字串起始位址，必須先予以定義。
例如：Buffer DB 4,5 DUP(?)

Buffer	Buffer+1	Buffer+2	Buffer+3	Buffer+4	Buffer+5
4	?	?	?	?	?

Buffer： 內容設定 4，表示最多接受並存放 4 個按鍵，3 個爲字元鍵，1 個爲[Enter]鍵的 ASCII 碼 0DH 存放於 Buffer+5。

Buffer+1：用來記錄 User 按了幾個鍵，當按鍵超出指定數時均被丟棄且發出"嗶"聲提醒。

Buffer+2～Buffer+4：存放 User 按鍵的 ASCII 碼。

DUP 的意思爲 Duplicate(複製的)，此例表示 Buffer+1～Buffer+5 等 5 個位址內容均相同且未設初值。

程式習作：顯示 ASCII 00H～FFH 的對應字元於螢幕 (使用 MASM6.11 版組譯/連結程式)

```
; ASC.ASM：Display ASCII characters (00...FF)
;=============================================================
            .MODEL      TINY                ；.COM Type，若改成 SMALL 則爲.EXE
;-------------------------------------------------------------------------------------------
            .DATA                           ；資料區段的宣告
LFCR        DB          10,13,'$'
ASC         DB          00,'  $'
Chr_Count   DB          00
;-------------------------------------------------------------------------------------------
            .CODE                           ;程式區段的宣告
            ORG         100H                ;.COM Type，若爲.EXE 檔則省略該指令
            .STARTUP                        ;主程式開始
Next_Row:
            MOV         CX,16               ;每列顯示 16 個字元
Next_Chr:
            LEA         DX,ASC              ;顯示一個字元
            MOV         AH,9
            INT         21H
            INC         ASC                 ;下一字元
            DEC         Chr_Count           ;若已顯示 256 個字元則 Return
            JZ          Stop
            LOOP        Next_Chr            ;若顯示 16 個字元則換列
            LEA         DX,LFCR
            MOV         AH,9
            INT         21H
            JMP         Next_Row
Stop:
            .EXIT                           ;主程式結束，返回 DOS
;=============================================================
            .STACK      100H                ;設定堆疊區的大小，若 STACK 後無數值則內定爲 1024Bytes
            END
```

巨集程式與副程式

一、巨集程式：分為巨集定義與巨集呼叫

1. 巨集定義：巨集名稱
   巨集開始
   參數
   巨集指令集(MACRO BODY)
   巨集結束

2. 巨集呼叫：以巨集名稱及參數即可使用已定義好的巨集程式，主程式在組譯時，自動將巨集程式本體取代呼叫指令，此動作稱之為「巨集展開」。

二、副程式與巨集程式相同點：

1. 將主程式中常用的功能另行編寫一程式處理之。
2. 可縮短主程式之編寫、程式發展分工、增加程式可讀性、利於程式修改除錯。

三、副程式與巨集程式不同點：

1. 須使用 CALL 指令呼叫副程式，副程式中必須有 RET 指令才能回返至主程式。
2. 巨集程式則直接使用其名稱即可，亦無回返指令。

四、副程式與巨集程式優缺點：

	優　點	缺　點
副程式	較不佔儲存體及記憶空間，因組譯後維持原程式大小。	有程式轉移問題，至副程式前須先保存 IP (超過一區段須再保存 CS)至 STACK，由副程式回返前則須取出位址，整體執行速度較慢。
巨集程式	呼叫處已被巨集程式取代，無程式轉移動作，執行速度較副程式快。	如叫用多次，組譯時會被展開插入叫用處，較佔儲存體及記憶空間。

使用 Header File 的巨集程式範例

類似 C 語言的 Head File，巨集程式可另存 xxx.H 檔，每個 Header File 可放一個或數個巨集程式，欲使用時在主程式開頭下達「INCLUDE xxx.H」即可，組譯程式會將使用到的巨集程式展開插入至主程式中，未使用到的其他巨集程式則不會被展開插入。

```
; MACRO.ASM
; 計算/顯示鍵盤輸入的兩數之和(小於 10)
; 使用巨集 Header File (SCREEN.H)
;=========================================
        INCLUDE     SCREEN.H
        .MODEL      SMALL
        .DATA
LFCR    DB          10,13,'$'
Mess    DB          'Key in two numerals:$'
Buffer  DB          3,4 DUP(?)
SUM     DB          ?
;-----------------------------------------------------------
        .CODE
        .STARTUP
Start:
        Disp_Str    Mess
        LEA         DX,Buffer
        MOV         AH,0AH
        INT         21H
;
        MOV         AL,Buffer+2
        ADD         AL,Buffer+3
        MOV         SUM,AL
        CMP         AL,69H
        JG          Start
;
        SUB         SUM,30H
        Disp_Str    LFCR
        Disp_Chr    Buffer+2
        Disp_Chr    '+'
        Disp_Chr    Buffer+3
        Disp_Chr    '='
        Disp_Chr    SUM
        .EXIT
;=========================================
        .STACK      100H
        END
```

```
; Header File：SCREEN.H
;=========================
; Display one character
Disp_Chr  MACRO   Chr
          MOV     DL,Chr
          MOV     AH,2
          INT     21H
          ENDM
;-----------------------------------
; Display string
Disp_Str  MACRO   String
          MOV     DX,String
          MOV     AH,9
          INT     21H
          ENDM
;-----------------------------------
; Clear screen
Clr_Scrn  MACRO
          MOV     CX,0
          MOV     DH,24
          MOV     DL,79
          MOV     AH,6
          MOV     AL,0
          MOV     BH,7
          INT     10H
          ENDM
```

中斷(Interrupt)與 I/O

一、中斷處理流程：

CPU 執行完目前指令→將旗號暫存器推入堆疊器
→禁止其他中斷(不含 NMI)
→將下一指令的 CS 及 IP 推入堆疊器
→至中斷向量表找出中斷服務程式的起始位址
→執行中斷服務程式

執行中斷回返指令 IRET→取回堆疊區中的旗號暫存器
→取回堆疊區中的 CS 及 IP，返回主程式繼續未完成的工作

執行 CALL 副程式時不會自動將旗號暫存器推入堆疊器，回返指令 RET 沒有取回旗號暫存器的動作，這就是一般副程式回返與中斷副程式回返使用不同指令的原因。

二、中斷種類及優先順位：(硬＞軟＞硬＞軟＞硬)

1. 重置(Cold RESET)，重新啓動 CPU。
2. CPU 內部中斷：如除數爲 0 的 INT 0、單步執行的 INT 1。
3. 不可抑制中斷(Non-Maskable Interrupt，NMI)：屬外部對 CPU 所發出的中斷，如 PC 的記憶體錯誤時。
4. 軟體中斷：執行程式中的 BIOS、DOS 服務程式呼叫 INT xx。
5. 可抑制中斷(INTR、IRQ)：屬外部裝置對 CPU 所發出的中斷請求。

三、中斷相關的位址配置：

區段位址 CS：IP	內　容	實體位址	使　用　時　機
0000：0000 ↓ 0000：03FF	中斷向量表 (1K)	00000 ↓ 003FF	除 RESET 中斷外，均會至此處取得中斷服務程式的起始位址。
F000：0000 ↓ F000：FFFF	ROM BIOS (64K)	F0000 ↓ FFFFF	1. Power On Self Test(POST)程式及 BIOS 服務程式存放區。 2. RESET 時 CS＝FFFF、IP＝0000，即會跳至 FFFF0 處執行一個 JMP 指令，進入 POST 程式，剩餘的空間用來存放 BIOS 的日期(可用 DEBUG 查看)。 D FFFF：0000 (看日期) U FFFF：0000 (看 JMP 指令)

四、中斷指令 INT n、中斷向量與中斷服務程式起始位址取得：

1. n 代表中斷的編號，其值爲 0～255(FFH)，共有 256 個中斷向量，每個中斷向量須包含區段位址 CS 及偏移位址 IP，共 4Bytes，故整個中斷向量表佔用了 1024Bytes 的記憶體空間。
2. 中斷服務程式起始位址的取得【n×4】：
   (例 1) INT 4：　4×4＝16＝10H　→至中斷向量表 0000：0010H 取 IP 低 Byte(如 12H)
   →至中斷向量表 0000：0011H 取 IP 高 Byte(如 34H)
   →至中斷向量表 0000：0012H 取 CS 低 Byte(如 56H)
   →至中斷向量表 0000：0013H 取 CS 高 Byte(如 78H)
   →得中斷服務程式起始位址 7856H：3412H
   (例 2) INT 16H：　22×4＝88＝58H
   →至中斷向量表 0000：0058H 至 0000：005BH 取中斷服務程式起始位址

五、在 Debug 下寫一個自己的中斷服務程式(60H～67H 係保留給 User 的中斷編號)

本實驗使用 INT 60H 中斷，由於 60H×4＝0180H，須將中斷服務程式的起始位址置於 0180H～0183H

中斷服務程式的工作爲顯示一字串 Hello! INT 60H

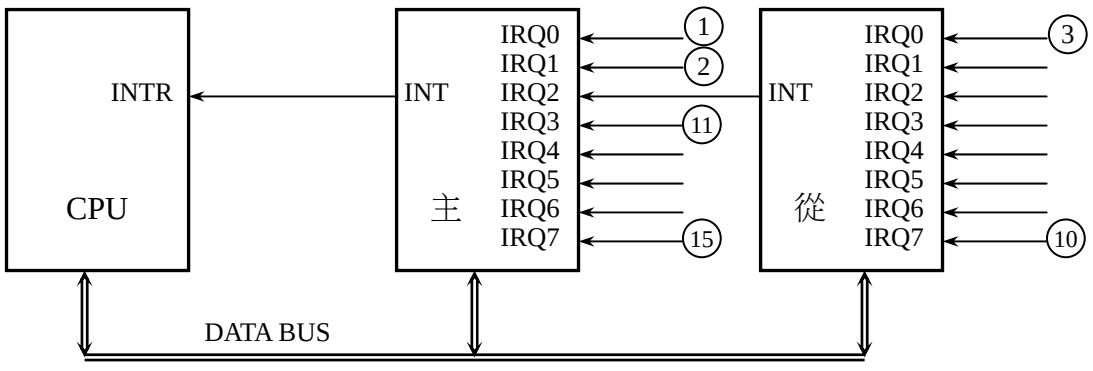
```
-a 1000:0000  ------------------------------ 在此位址開始輸入中斷服務程式
1000:0000  mov  ax,1000
1000:0003  mov  ds,ax---------------------- 依規定透過 AX 更改區段暫存器 DS 的內容
1000:0005  mov  dx,000d------------------- 配合 INT 21H，取字串的起始位址爲 DS：DX＝1000：000D
1000:0008  mov  ah,9----------------------- 指定 INT 21H 的服務功能編號爲 9
1000:000A  int  21------------------------- 執行 DOS 中斷服務程式
1000:000C  iret ---------------------------- 返回主程式
1000:000D  db 'Hello! INT 60H$' ----- 定義字串內容，$爲結束字元
1000:001C
                IPL    IPH
-f 0000:0180 L4  00  00  00  10 ---------- 將中斷服務程式起始位址 1000：0000 填入(Fill)中斷向量表
      長度 4Bytes      CSL   CSH
-a 100  ------------------------------------- 自 CS：0100 開始編寫主程式
0AED:0100  int  60------------------------- 執行自編的中斷服務程式
0AED:0102  int  3-------------------------- 程式結束
0AED:0103

-g=0AED:0100 -------------------------------- 執行主程式
Hello! INT 60H  ---------------------------- 成功！
```

六、多重及優序性中斷控制 IC8259

◆主要接腳及功能

1. IRQ0～IRQ7 ：可接受 8 個周邊裝置的中斷請求。
2. INT 輸出　　：接至 CPU 的 INTR 腳，對 CPU 發出中斷信號。

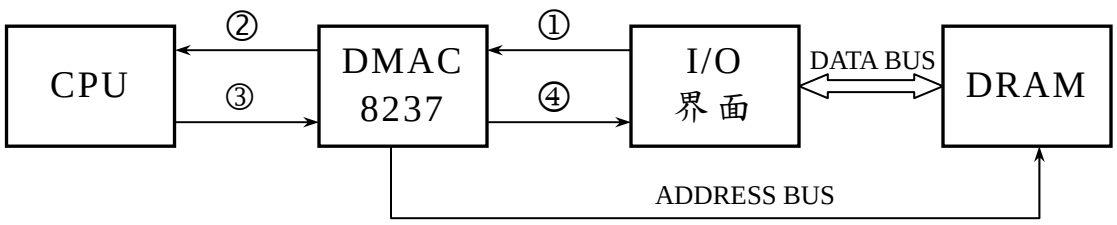
◆主從式串接：以「主 8259」的 IRQ2 腳接至「從 8259」可接受 15 個中斷源



PC 的外部中斷源中，最高順位①用於系統計時器 8253/54，
其次②用於鍵盤

目前主機板上，如 8259、8237、8253(8254)、RTC 等，均已 VLSI 化，製造在一晶片上，俗稱「南橋」，靠近 CPU 的那一顆是「北橋」。

七、直接記憶體存取控制器(Direct Memory Access Controller，DMAC)8237



①：I/O 元件向 DMA 控制器提出 DMA 傳輸請求。

②：DMA 控制器對 CPU 發出 HOLD(持住)請求。

③：CPU 接受後將 DATA 及 ADDRESS BUS 浮接後，送出 HLDA(Hold Acknowedge 持住認可)信號給 DMAC8237，開始進入 DMA 週期。

④：DMAC 收到 HLDA 後，送出 DACK(DMA Acknowedge)信號給 I/O 界面，進行 DMA 傳輸。

八、計時/計數 IC8253/8254

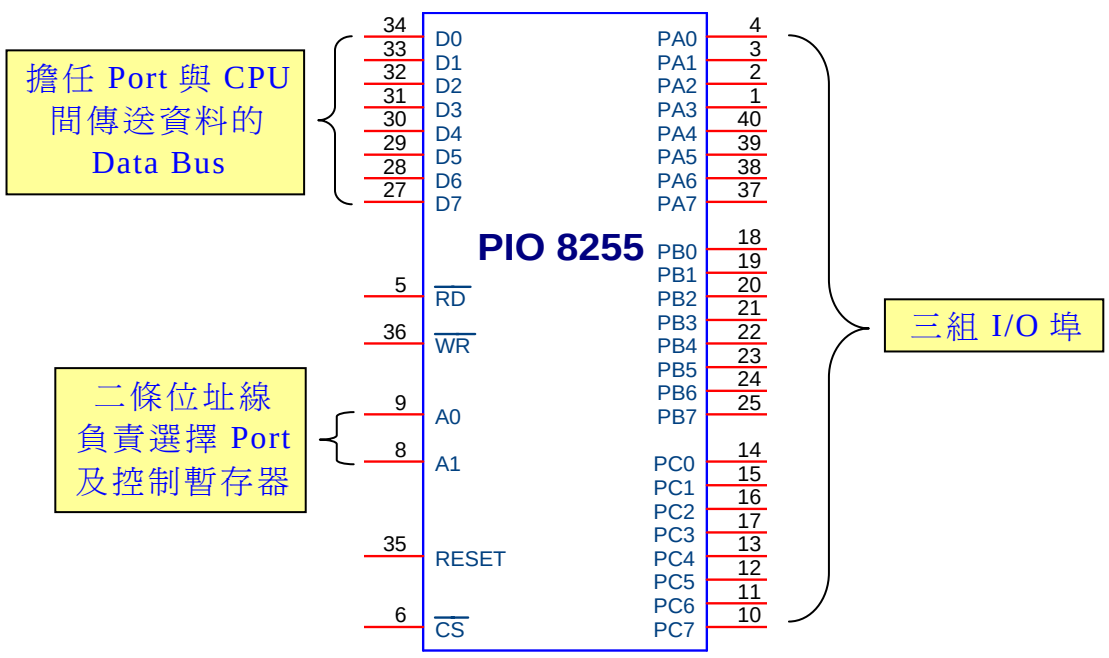
1. 具有三個獨立的 16 位元計時計數器，一個控制暫存器。
2. 計數器與計時器事實上均爲同一電路，只是工作脈波來源不同，計數器的工作脈波來自外部，而脈波間隔不定。計時器的工作脈波取自系統本身，屬週期性脈波。
3. 在 PC 中的功用爲：

   計時器 0----- 定時向 8259 晶片發出訊號，使其產生 INT 8 中斷。

   計時器 1----- 定時更新(Refresh)主記憶體 DRAM。

   計時器 2----- 推動喇叭(或蜂鳴器)。

九、並式輸入輸出 PIO： 8255 PPI(Programmable Peripheral Interface)



十、串列傳輸介面 IC： 8250/8251

可將 CPU 的並式資料轉換爲串列資料傳輸，亦可將外界傳送來的串列資料轉換爲的並式資料給 CPU。

組譯&連結程式用法

1. masm prog ---------- 僅產生.OBJ 檔。(注意空格，prog 代表組合語言原始程式檔.ASM)
2. masm prog /L ---------- 產生.OBJ 檔及.LST 檔。
3. Link prog ---------- (逐次回答以下詢問)
   Run File [prog.exe]：
   List File [nul.map]：
   Libraries [.lib]：
   Definitios File [nul.def]：
4. Link prog; ---------- 直接產生.EXE 執行檔。
5. Link prog,,prog.map; ---------- 產生.MAP 檔及.EXE 執行檔。
6. Link prog1+prog2+prog3; ---------- 連結三個程式，且以第一個檔名爲主，產生 prog1.EXE 執行檔。
7. ML prog.asm ---------- 組譯並連結，依程式中之宣告，產生.EXE 或.COM 執行檔。
8. ML /c prog.asm ---------- 僅組譯成.OBJ 檔，不進行連結。
9. ML /c /FL prog.asm ---------- 組譯產生.OBJ 檔及.LST 檔，不進行連結。
10. ML prog1.asm+prog2.asm+prog3.asm ------- 組譯並連結三個程式，產生 prog1 執行檔。
11. ML /FL /FM prog.asm ---------- 組譯並連結，產生 prog 執行檔、.OBJ 檔、.LST 檔及.MAP 檔。

◆ 自編批次檔 ASM.BAT 內容：

```
ML /FL /FM %1.ASM
DEL %1.OBJ          (刪除 OBJ 檔，若改成 REM DEL %1.OBJ 則不執行，
DEL %1.LST           由於目前磁碟的容量大，可移除此三個指令)
DEL %1.MAP
```

◆ 使用法：ASM prog