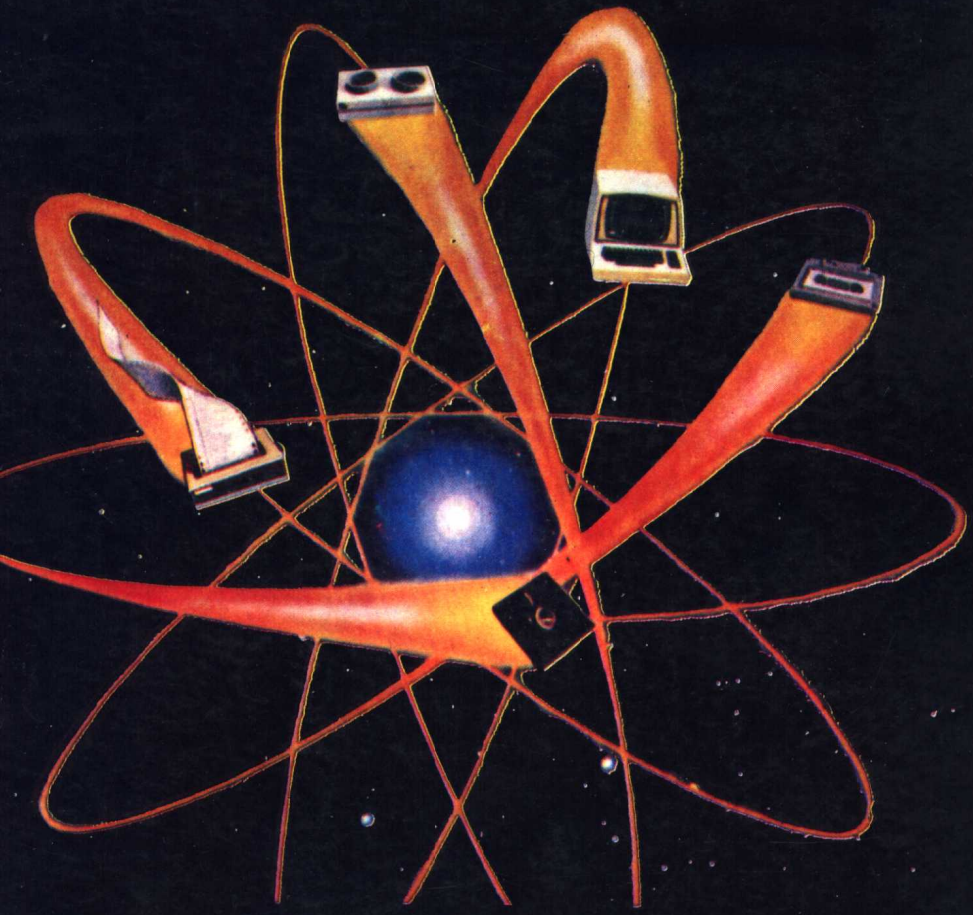
微電腦作業系統

林昇耀 譯



微電腦作業系統

林昇耀 譯

協群科技出版社

微電腦作業系統

編譯者：林昇耀

出版：協群科技出版社

發行：協群科技出版社

香港中環皇利街684號二樓

印刷者：廣源印務局

青山道875號工敏大廈

定價：H.K.$ 25.00

目　錄

第一章

導　言

「作業系統」是什麼呢?!大體上說，它是一群程式的集合，其用途是作爲使用者和電腦硬體之間的一個橋樑，舉個例子來說，在一個小型的微電腦系統中，假設它擁有 2K（ 2048 ）位元組的「唯讀記憶體」（PROM），以及少許的「隨機存取記憶體」（ RAM ），那麼它的作業系統也許包括了一些小程式以及一些提供給使用者使用的「命令」（ Command ），利用這些命令，使用者可以去命令微電腦系統去檢查記憶體中的內容、更改記憶內容、或是測試應用程式的正確性等等。

比較複雜的作業系統可能會包括「磁碟控制程式」（ Disk-Controller ）以及較高階層的記憶體管理程式。本書中所提及的最高級的作業系統包括許多功能，例如「多元程式系統」（ 同時執行幾個不同的程式）、「記憶體管理系統」、「磁碟控制系統」、以及「隨機磁軌定位系統」。「多元處理機系統」在本書中也稍有提及。

爲什麼要有作業系統?!假如電腦有一個很完整的面板，那麼使用者可以由面板輸入，或更動記憶體中的內容，理論上來說可以不用加入作業系統。但是有一點必須注意的是這種方式不但浪費時間也容易出現錯誤，經過多年的實驗，使用者發現我們必須要有一個較省時方便的方式來操作電腦。於是作業系統便應運而生了。

第二章中我們將討論微電腦系統中的作業系統該具有那些功能，第三章中我們討論小型電腦的作業系統，第四章討論中型系統以及資料管

理、磁碟位置安排等等。第五章中涉及一個「多使用者」和「多元程式」系統的設計所須考慮的許多問題。第六章介紹「多元處理機」的觀念。第七章涵蓋了記憶體的管理系統，包括「綫性選擇位址指示法」，以及「虛擬記憶」（Virtual Memory）。第八章我們討論電腦的周邊環境。第九章則是一些系統提供的功能。第十章介紹使用者和系統的介面。第十一章我們將各部份總合做個結論。

附錄一、二則是使用者的參考手册及UNIX作業系統簡介。

微電腦硬體及一些術語

微電腦的結構和一般中型及大型的電腦比較起來要簡單得多了，本書中所提及的大多針對8位元（8 - bit）的微電腦而言，事實上許多觀念也可以應用在較新的16位元微電腦上，此外有些16位元的功能在本書中也有述及。

中央處理機（Central Processor）之結構

典型的8位元微處理機的結構如圖1－1所示，其中算數邏輯單元（ALU）執行所有的算術和邏輯的資料處理，它通常有一些內在的暫存器，至少有一個「累進器」（Accumulator），這些是用來保存運算的運算元以及中間值。此外「索引暫存器」（Index Registor）是用來安排記憶體位址或作爲「指標」（Pointer）之用。「指令暫存器」（Instruction Register）用來存放目前正在執行的機器指令（通常指令暫存器的內容是無法讓使用者探知的），「程式計數器」（Program Counter）是用來存放下一個指令在記憶體中所存放的位置。

內部滙流帶（Internal Bus）和系統的外部滙流帶互相介面。通

常外在滙流帶包括「位址滙流帶」（Address Bus）（通常是16位元）、資料滙流帶（Data Bus）（通常8位元），另外還有一個控制滙流帶，例如 READ 、WRITE 、MEMORY-REQ 、 I/O REQ 、 WAIT 等等控制訊號。

控制滙流帶上的訊號是提供給內在記憶體以及輸入/輸出埠使用的，它們可以利用這些訊號來確定 CPU 是否正在與它們交換資料，一般的微處理機至少都具有一個「中斷」的輸入訊號。所謂「中斷」乃是一種特殊功能，它能要求中央處理機（ CPU ）暫時放下目前的工作而去

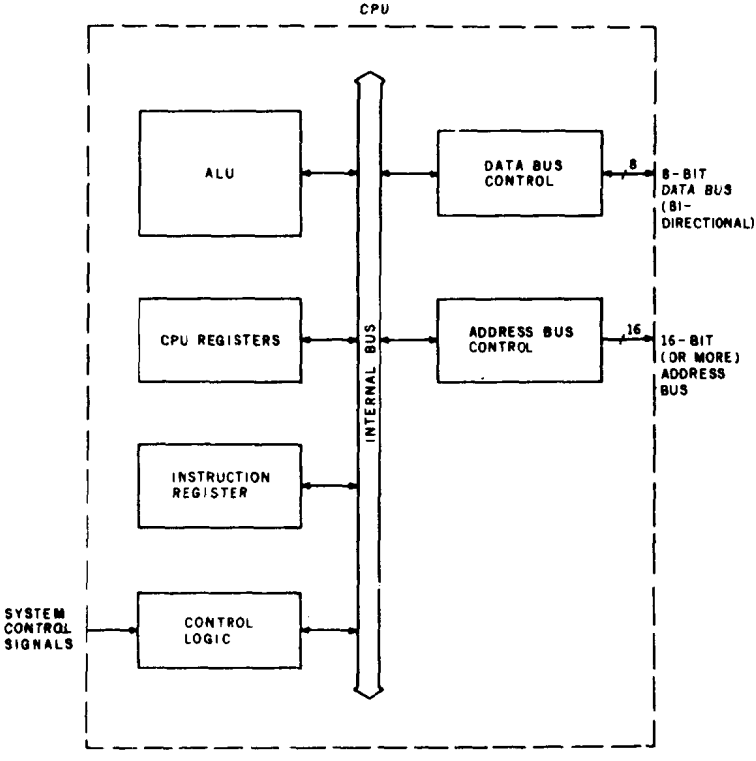


圖 1 － 1

執行特殊的程式，通常鍵盤及終端機的按鍵都會產生中斷的訊號，以避免按鍵輸入時遺失資料。

當電腦經過重新開機（Reset）之後，必須去記憶體中取出第一個指令來執行，通常我們把這個啓始的指令放在記憶體位置 0000 的地方，而且是用唯讀記憶體來存放（ROM），每當一重新開機，CPU 便到 0000 的位置上把該指令讀出並解碼、執行。如果指令長度不只一個位元組，則 CPU 會依程式計數器的內容去記憶體中讀取下一個位元組，繼續執行。除非是執行到分支指令（Branch）或呼叫（Call）指令，通常程式計數器的內容每當讀取一個位元組之後會自動加 1，通常指令會一直執行直到電源被切斷或者執行到一個 HALT 的指令。

```
0000H      MVI   B,0FFH          ;Load the number 'FF' into the
                                  ;B register.
0003H      JMP   0020H           ;Branch to location 0020.
0006H      OUT   23H             ;Output the A register to
                                  ;port 23H.
0008H      RET                   ;Return to calling program.


0020H      MVI   A,10H           ;Load the number 10H into
0022H      CALL  0006H           ;call the subroutine at 0006.
0025H      INR   A               ;Increment the A register.
0026H      DCR   B               ;Decrement the B register.
0027H      JNZ   0022H           ;Branch to location 0022.
002AH      HLT                   ;If done, Halt the CPU.
```

PC	A	B	INSTRUCTION		
0000	?	?	06 FF	MVI	B,0FFH
0003	?	FF	C3 00 00	JMP	0020H
0020	?	FF	3E 10	MVI	A,10H
0022	10	FF	CD 06 00	CALL	0006H
0006	10	FF	D3 23	OUT	23H
0008	10	FF	C9	RET	
0025	10	FF	3C	INR	A
0026	11	FF	05	DCR	B
0027	11	FE	C2 22 00	JNZ	0022H
0022	11	FE	CD 06 00	CALL	0006H
0006	11	FE	D3 23	OUT	23
—	—	—	—	—	
—	—	—	—	—	

圖 1－2　執行運算的過程

通常記憶部份和 I/O 部份所用的觀念是相同的，記憶體位址或 I/O 埠的編號先被放在位址滙流帶上，之後資料便會顯示在資料滙流帶上，再由控制訊號來決定是做什麼動作（讀或寫）。

記憶體結構

微處理機的記憶體大致上可分爲兩大類，讀/寫記憶體以及唯讀記憶體。目前的記憶 IC 晶片的密度相當高，大致上可以在一個晶片上放入 65535 個位元之記憶。而且一直在增加之中。若有一個晶片上的記憶體安排形式爲 16384 × 1 ，則我們通常將 8 個這種晶片並排成 16384 × 8 的結構，亦即 16K 個位元組，每個位元組 8 個位元。若某微電腦爲 16 位元的處理機，則我們便要把 16 個這種晶片排起來成爲 16384 × 16 的結構。圖 1 − 3 爲一個典型的記憶模板。（ $1K = 2^{10} = 1024$ ）

唯讀記憶體（ ROM ）也可分爲以下幾類：

1. 一般的 ROM
2. 可程式的 ROM （ PROM ）
3. 可程式、可清除的 ROM （ EPROM ）

EPROM的使用多半是在小量的程式方面，因爲它的造價較高，而且是利用紫外光去寫進程式或更改內部的內容。它方便的地方是隨時發現程式的錯誤時，隨時可以加以改正。 PROM 和EPROM唯一的不同是 PROM 的內容一經寫入即無法再予更改。 ROM 是使用在大量程式的貯存上，在將程式寫入 ROM 中之前必須經過許多的測試，以確定該程式正確無誤。

ROM 、 PROM 、EPROM通常是做成如下的晶片： 1?? × 8

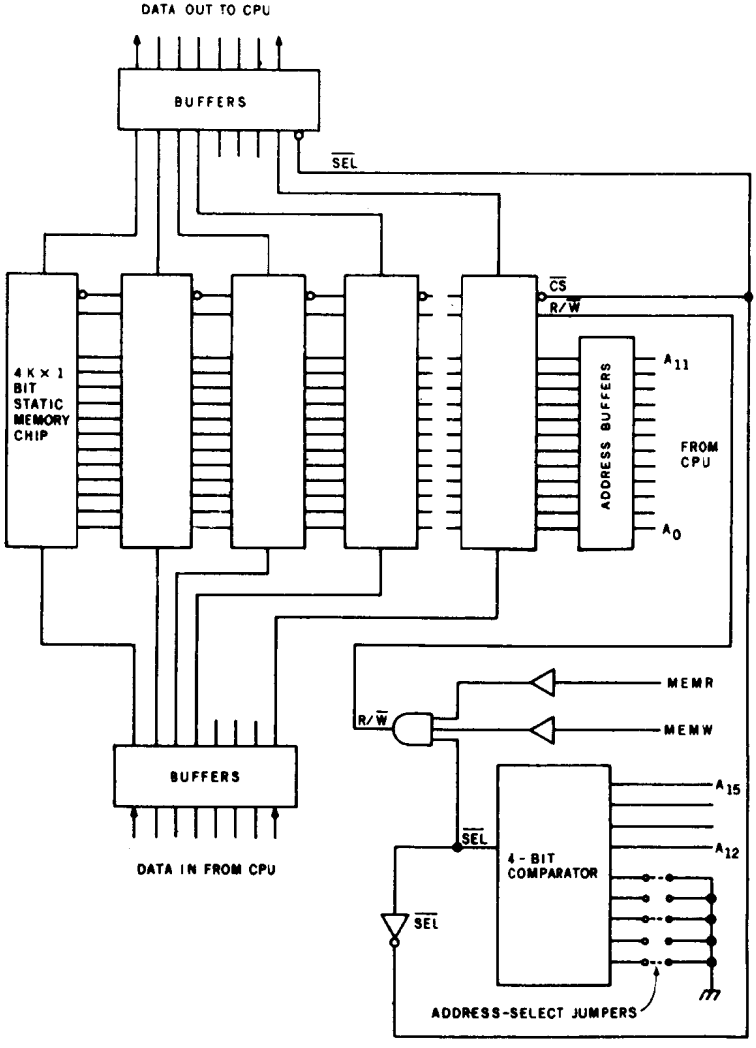

DATA OUT TO CPU
BUFFERS
SEL
4 K × 1 BIT STATIC MEMORY CHIP
CS
R/W
ADDRESS BUFFERS
A11
FROM CPU
A0
MEMR
MEMW
R/W
BUFFERS
DATA IN FROM CPU
SEL
4 - BIT COMPARATOR
A15
A12
SEL
ADDRESS-SELECT JUMPERS

圖 1 － 3

8K × 8 等等，而且密度天天在增加。

軟體概念

和微電腦溝通的最簡便方式是利用組合語言或高階語言（PASCAL、BASIC 等）來寫程式。通常高階語言比起組合語言要容易得多，但經過編譯程式轉換之後，所得到的目標程式的效率則比較低，且執行速度較慢。

資料形式

許許多多的資料形式都是由最基本的「位元組」及「指標」（Pointer）所組成。一個位元組是由 8 個位元所組成，每個位元只可能是 0 或 1，換句話說，一個位元組的內容只能是 0～255 或 0～FF（十六進位），若在某一位元組中存放了 41H（十六進位）則可視爲數字 65，亦可視爲 ASCII 碼中的「A」字母，也可視爲數字 41 等等。此外 01000001 也可視爲 8 個「旗標」（flag）的組合，1 表示該旗標已被設定（set），0 則表示未被設定。

其他還有一種結構稱爲「字串」（string），它是一串字母的集合，每個字母占一個記憶單位（位元組），字母的最後並有一個特殊符號來表示字串結束。若不加特殊符號來表示結束時也可在字串的起始標上該字串的長度即可。

```
Assembler input:
                    TEXT: DB    'THIS IS A CHARACTER STRING',0
                    MSG2: DB    'THIS STRING ENDS WITH A CR',0DH
                    MSG3: DB    'THIS STRING ENDS WITH A $'
                    MSG4: DB    49,'THIS STRING CONTAINS A '
                          DB    'HEADER BYTE FOR THE LENGTH'

Basic statement:    TEXT = "THIS IS A STRING ASSIGNMENT IN BASIC"
```

圖 1 － 4

圖 1 － 5 前幾個字串有不同的結束字母，這些字母是由程式設計師來決定，當程式需要用到字串資料時，必須經由一個「指標」告訴 CPU 「字串」存放在什麼地方，於是 CPU 可以對它加以處理。

```
Assembler input:
                        LXI   H,MSG2     ;Load address of MSG2
                                         ;into the HL registers.
                        CALL  DMSG       ;Call the "Display-
                                         ;Message" subroutine.

Basic statement:        PRINT TEXT
```

圖 1 － 5

圖 1 － 6 中， MSG 2 的位址放在暫存器 HL 中，然後呼叫「顯示資料」的程式，該程式便將字串的字母一個一個的輸出到顯示器上，直到碰到一個「返回」的位元組符號（ ODH ）。

```
Assembler input:    DMSG: MOV A,M        ;Get character from (HL)
                          CPI ODH        ;is the character a CR?
                          RZ             ;if so, return to caller
                          CALL CO        ;Output character in A
                          INX  H         ;Increment HL pointer.
                          JMP DMSG       ;Get next character.
```

圖 1 － 6

圖 1 － 6 是一個「 DMSG 」（資料顯示）程式的形式（利用 8080 的組合語言寫成），這個程式先將一個字母讀出，然後檢查是否爲「 ODH 」（返回符號），若不是則呼叫「 CO 」程式（控制台輸出程式），接著 HL 的內容加 1 ，如此重覆執行，圖 1 － 7 是「 DMSG 」相對的流程圖。