PC-8201A Technical Manual

First edition

Mar 15th 1984

NEC Corporation
Personal Computer development division

IMPORTANT NOTICE

(2) All efforts have been make to ensure that the contents of this manual are correct; however, should any errors be detected, NEC would greatly appreceate being informed.

(3) NEC can assume no responsibility for errors in this manual or their consequences. The entire risk as to the results of and performance of this manual is assumed by you.

The numeric notation and rules in illustration

In this manual, all numbers are expressed in Decimal unless preceded by special radix prefix ^X, ^O ,^B and ^D. ^X, ^O and ^B represent hexadecimal, Octal and Binary numbers respectively. For instance, ^X1000 is hexadecimal number 1000, which is 4096 in Decimal. Similarly ^O400 is octal number 400, which is 256 in Decimal. ^B10000000 is Binary number 10000000, 128 in Decimal. ^D is used to explicitly tell that is the decimal number.

In the illustration, the upper side of the memory map is near ^XFFFF. So the part drawn below another part is allocated at lower address area.

```
----------------- ^XFFFF
|               |
|               |  ^
================  | Upper
|               |  |
|               |
-----------------
|               |
| File A        |
|               |
-----------------
|               |
| File B        |
|               |
----------------- |
|               |  | Lower
|               |  V
================
|               |
|               |
----------------- ^X0000
```

Fig 0.1 The 'File A' is located upper the 'File B'.
('=' means 'skipped'.)

And the address and the pointers written in the illustration points the just above the line.

```
APOINT ---> ^X8000

 ¦                 ¦
 ¦                 ¦
 ¦ (^XD000)        ¦ ^XD000
 -----------------
 ¦ (^XEFFF)        ¦
 ¦                 ¦
 ¦                 ¦
 ================= (Skipped)
 ¦                 ¦
 ¦                 ¦
 ¦ (^X8000)        ¦
 ----------------- <-- APOINT
 ¦ (^X7FFF)        ¦
 ¦                 ¦
```

Fig 0.2 Pointer and its contents

Authors:

Chapter 2 -- 4

Mr.Youshiro Hayashi

Software Engineer

Application Technology Department
Personal Computer Sales Promotion Division

Chapter 1,5 -- 8

Mr.Hiroaki Yokoyama

Software Engineer

Development department
Personal Computer Development Division

Chapter 9 -- 14

Mr.Akio Takagi

Software Engineer

Development Department
Personal Computer Development Division

Chapter 15

Mr.Moriharu Seki

Hardware Engineer

Development Department
Personal Computer Development Division

Rewritten and edited by Mr.Hiroaki Yokoyama

CHAPTER 1

INTRODUCTION

The portable personal computer, PC-8201A is a unique and practical computer. It has many special capabilities in it. For example, it uses large LCD (Liquied Crystal Display), CMOS (Complementary Metal Oxide Semiconductor) technology and special built-in Software.

The built-in software features are very powerful and useful. But for using PC-8201A fully in particular purpose, new Software written in Machine language might be requested. One of the built-in software, N82-BASIC is very useful to make a small utility, but it's not enough to make a large size utility, for instance, Spread Sheet or new Word Processor.

In order to support the programmers who want to make such a large programs, and to support the programmers who want to manage the hardware features directly, this document describes not only the detail hardware features of PC-8201A, but also the know-how to use these features without any trouble.

The most important thing is "compatibility". The built-in Software features keep the promise in using the memory, I/O interface and interrupt functions. The built-in Software checks many critical points at Power-on automatically as far as you don't remove the ROM #0. So if you break the promise, PC-8201A begins "Cold start" to initialize the all contents in RAM. In this case, the important files and data which you stored are flushed.

The promise for using PC-8201A's features is described in each chapter. Before making a your special program, please

refer to the corresponding chapters. The previous INDEX will help you.

The built-in Software uses a small part of the PC-8201A's special features. With this manual, may you make a super programs for your own purpose!!

CHAPTER 2

MEMORY MAP

2.1 OVERVIEW

The PC-8201A has the following memory capacity. The value specified with 'Max' means the maximum capacity that is greatly expanded by adding RAM/ROM chips or RAM cartridge.

ROM 32K bytes

(Max 64K bytes)

RAM 16K bytes

(Max 96 K : 32k bytes x 3 bank)
2 banks are equipped on Main board of PC-8201A and 1 bank is provided with RAM cartridge.

And PC-8201A has three useful programs in the standard ROM, ROM #0. These programs are (N82-)BASIC , TEXT and TELCOM.

N82-BASIC: Microsoft BASIC, specialized for PC-8201A.

TEXT: Simple and powerful word processor

TELCOM: Communication program with other digital computers via RS-232C.

The simple memory map of PC-8201A is illustrated in

the next figure. This illustration is a one of the standard pattern. Refer to Chapter 15 to understand the hardware expansibility, the detail configuration of memory and how to change the memory configuration.

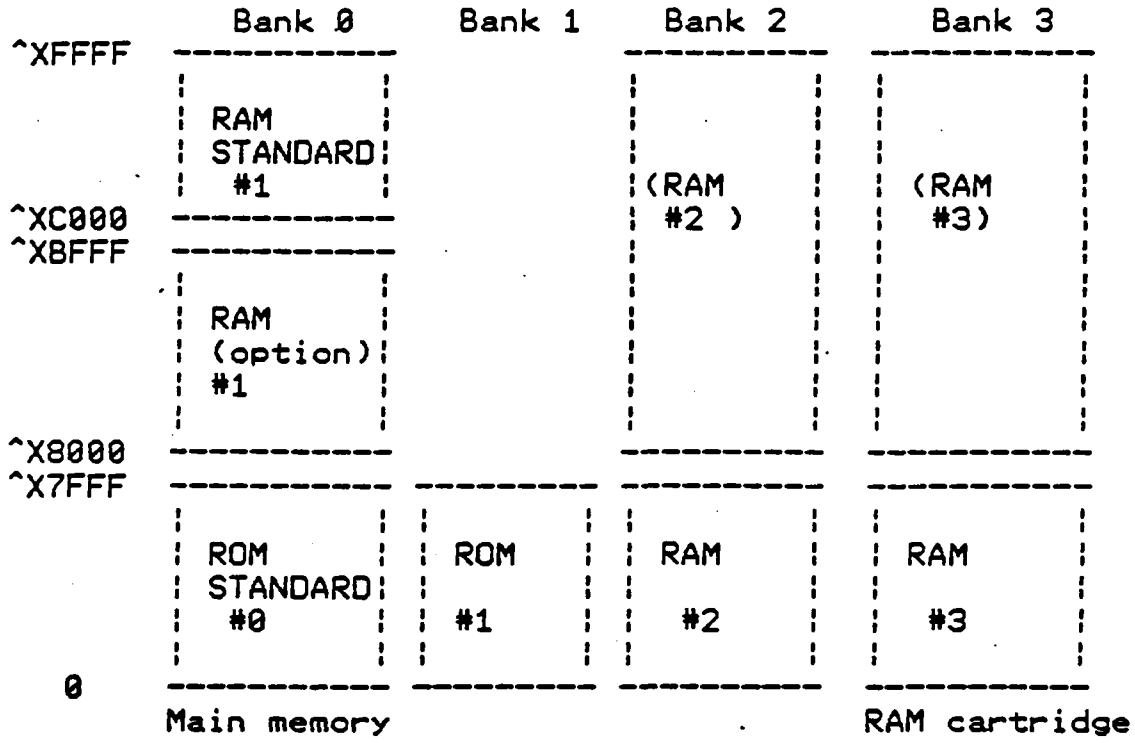


Fig 2.1 P C - 8 2 0 1 A M E M O R Y M A P

The RAM #2 and RAM #3 can be located both low address, from 0 to ^X7FFF, and high address, from ^X8000 to ^XFFFF. This selection can be done by PORT access. Refer to chapter 2.3.

2.2 BANK SWITCHING ARCHITECTURE

The heat of PC-8201A is the Intel 80C85, which is 8 bit processor and whose address bus is 16. Thus, the 80C85 can access 64K of memory at a time. In PC-8201A, however, special memory access function called memory-bank switching is supported. So the 64K barrier in 8-bit microprocessor can be tricked in PC-8201A.

The RAM in the PC-8201A is divided into units referred to as 'BANKS'. One bank can contain a maximum of 32K bytes of memory, while the RAM can be expanded to hold a maximum of three banks. (RAM #1, RAM #2, RAM #3)

The RAM #2 and RAM #3 can be located in two different positions, lower position is from ^X0000 to ^X7FFF and higher position is from ^X8000 to ^XFFFF) And RAM #3 is detachable, because it is provided in RAM cartridge. The bank-switching is executed every 32K bytes. For the sake of this limitation it is impossible to access the half part of RAM #1 and half part of RAM #2 at a time. In other words, you cannot set up the this kind of memory allocation, lower half of RAM #2, from ^X8000 to ^XBFFF, and higher part of RAM #1, from ^XC000 to ^XFFFF as 32K of memory. The variety of memory allocation is illustrated and explained kindly in Chapter 15. The explanation about the software specification in bank-switching is shown in the next section.

The RAM #2 and RAM #3 can be protected by a 'PROTECT SWITCH'. The 'PROTECT SWITCH' for RAM #2 is equipped at the real panel. Refer to the page 1-3 in PC-8201A User's guide. The RAM #3 has it at the side of the cartridge. But unfortunately, RAM #1 has no such a protect function. When you use this protect switch, you cannot use that RAM bank in usual way, for instance, BASIC. Because, PC-8201A uses the highest RAM area, from ^XF380 to ^XFFFF to save the current status of PC-8201A every time.

All RAM chips consists of CMOS and are back-uped by battery. All data and program files stored in RAM will be kept, even if the power switch is turned off. If you make a special utility for 2nd ROM or special RAM configuration, you have to consider about this Power-down sequence. Refer to chapter 3 to understand the Power-off trap in ROM #0.

2.2.1 Bank Switching Hardware

The 'bank-switching' is performed by OUT instruction. The OUT instruction outputs 8 bit data to the I/O port. The port address and that bit assign of the 8 bit data is shown below.

PORT ADDRESS ^XA1 (OUT)
Bank control

MSB	7	6	5	4	3	2	1	0

Bit 7 --- not used

Bit 6 --- not used

Bit 5 --- not used

Bit 4 --- not used

Bit 3 --- High address
(^X8000 - ^XFFFF) selection #2

Bit 2 --- High address
(^X8000 - ^XFFFF) selection #1

Bit 1 --- Low address
(^X0000 - ^X7FFF) selection #2

Bit 0 --- Low address
(^X0000 - ^X7FFF) selection #1

High address #2	High address #1	
0	0	Bank #0 (RAM #1)
0	1	not used
1	0	Bank #2 (RAM #2)
1	1	Bank #3 (RAM #3)

Low address #2	Low address #1	
0	0	Bank #0 (ROM #0)
0	1	Bank #1 (ROM #1)
1	0	Bank #2 (RAM #2)
1	1	Bank #3 (RAM #3)

The current status of the memory, the status of bank-switching, can be examined by IN instruction. The IN instruction reads a 8 bit data from the specified I/O port. See next figure about the Port address and bit assignment of the data.

PORT ADDRESS ^XA0H (IN)
Bank status

MSB	7	6	5	4	3	2	1	0

Bit 7	---	Serial interface status #2
Bit 6	---	Serial interface status #1
Bit 5	---	Not used
Bit 4	---	Not used
Bit 3	---	High address (^X8000 - ^XFFFF) status #2
Bit 2	---	High address (^X8000 - ^XFFFF) status #1
Bit 1	---	Low address (^X0000 - ^X7FFF) status #2
Bit 0	---	Low address (^X0000 - ^X7FFF) status #1

Serial I/F #2	Serial I/F #1	
0	0	Not used
0	1	SIO port
1	0	Floppy disk port
1	1	RS-232C port

High address #2	High address #1	
0	0	Bank #0 (RAM #1)
0	1	Not used
1	0	Bank #2 (RAM #2)
1	1	Bank #3 (RAM #3)

Low address #2	Low address #1	
0	0	Bank #0 (ROM #0)
0	1	Bank #1 (ROM #1)
1	0	Bank #2 (RAM #2)
1	1	Bank #3 (RAM #3)

Refer to Chapter 12 about Serial Interface.

2.2.2 Bank Switching Software

The bank-switching capability is used in Menu mode. The 'BANK' command, arranged in Function key 10 (Shift + F.5) uses this function. This function falls into the Bank handler routine, CHGBNK, ^X7EAB. The CHGBNK checks the current bank status, tests whether the bank really exists, save the new bank # in BANK (^XF3DB), changes the bank status and jumps to the address 0. Jumping to address 0 causes 'COLD START' if the bank has not ever used or the flag named FSIDSV has a wrong value. (Refer to the section 3.2 Bookkeeping area.) Otherwise, Jumping 0 does 'WARM START'.

In order to test the existence of the another bank, CHGBNK reads the contents of the address, ^XE000, in that destination bank, modifies that value, restores it, and re-reads it. If that bank were really in exist, the value read first and the value re-read last are not identical.

The reason why CHGBNK jumps into the address 0 is, you might already notice, to set up the bookkeeping area. As described in Chapter 7, all standard programs and operating system uses this area every time to keep the current status. This area contains very important pointers, flags and interrupt routines. So without setting up this area, that bank cannot be handled with ROM #0 correctly.

If you use a bank only with your special application program, which does not use the pointers on interrupt routines in the bookkeeping area, you might think that you need not care about the bookkeeping area. But please do not forget that 'SHIFT+F.5' in menu level can change the bank any time. I recommend that you will keep the current rules about Bank-switching in ROM #0, and set up the bookkeeping area.

Refer chapter 4 'HOW TO USE 2ND/3RD RAM' to get more detail documents.

2.3 GENERAL MEMORY MAPPING OF INTERNAL SOFTWARE USE

You know that the ROM #0 addressed from 0 to ^X7FFF is used for standard programs and operating systems. (Sometimes, 'standard programs' represents BASIC, TEXT and TELCOM especially. 'Operating system' also represents 'Menu'. But there is no explicit border line between the 'standard programs' and 'operating system'. But I do sometimes use these words to explain the concept of the PC-8201A's built-in software.) Also, Some parts of the RAM memory area are reserved and used by that standard programs and operating system. The memory map about the RAM area is figured at next page. The each part of the reserved area is pointed by pointers in the 'book-keeping area', located at the highest part of the RAM memory, from ^XF380 to ^XFFFF. And the following 2 items are included in the book-keeping area, too.

Interrupt routine
System work area

Fig 2.2 PC - 8201A RAM AREA MEMORY MAP

```
^XFFFF  -----------------
        | Bookkeeping    |
        |  area          |
^XF380  -----------------
        | User machine   |
        |  stored area   |<- [HIMEM]      ^XF384
        -----------------
        | File control   |
        | block area     |<- [FILTAB]     ^XFB63
        -----------------
        | 2 Bytes space  |
        -----------------
        | String area    |<- [MEMSIZ]     ^XFA9A
        |  (used)        |
        -----------------
        | String area    |<- [FRETOP]     ^XFABF
        |  (free)        |
        -----------------
        | Stack area     |<- [STKTOP]     ^XF459
        |                |<- Stack Pointer
        -----------------
        | Free area      |
        |                |<- [STREND]     ^XFAE9
        -----------------
        | Array stored   |
        |  area          |<- [ARYTAB]     ^XFAE7
        -----------------
        | Simple         |
        | variable area  |<- [VARTAB]     ^XFAE5
        -----------------
        | .CO files      |
        |  area          |<- [BINTAB]     ^XFAE3
        -----------------
        | EDIT area      |
        | for BASIC      |<- [EDTDIR]+1   ^F886+1
        -----------------
        | Paste buffer   |
        | for TEXT       |<- [SCRDIR]+1   ^XF87B
        -----------------
        | .DO files      |
        |  area          |<- [ASCTAB]     ^XFAE1
        -----------------
        | non-registered|
        | BASIC file     |<-[NULDIR]+1    ^XF870+1
        -----------------
        |                |
```

```
|                |
----------------
| .BA files      |
|   area         |<-[TXTEND]      ^XFA88
----------------
| Current BA     |
|  file          |<-[TXTTAB]      ^XFA5D
----------------
| .BA files      |
|  area          |<-[BOTTOM]+1    ^XF980+1
----------------
|                |<-[BOTTOM]      ^XF980
----------------
```

Brief explanation about pointers which appear at the previous page.

[BOTTOM] Bottom address of RAM
[TXTTAB] Beginning of the current BASIC program
[TXTEND] End of the current BASIC program
[NULDIR] Non-registered BASIC program
[ASCTAB] Lowest address of ASCII files
[SCRDIR] SCRAP file
[BINTAB] Lowest address of binary files
[VARTAB] Simple variable space
[ARYTAB] Start of array table
[STREND] End of Array table
[STKTOP] Top of stack space
[FRETOP] Top of string free space
[MEMSIZ] Highest location in memory
[HIMEM] Highest memory available to BASIC
(The same as CLEAR's 2nd parameter)

rf. Chapter 5 'UNDERSTANDING THE RAM FILE CONCEPT', 'DIRECTORY STRUCTURE' and 'RAM ORGANIZATION'. In those chapters, the concept of the files and detail explanation about the pointers are described.

2.4 SAMPLE

```
        TITLE   Bank switching program

;
;       This sample will only change the bank of
;       RAM addressed from ^X8000 to ^XFFFF.
;       You had better check that the bank which
;       you want to switch really exists. And you
;       should save the next bank # at the
;       bookkeeping area, BANK.
;
;       Entry   None
;       Exit    None
;               Bank will be changed
;
;       Bank rotation #1 -> #2 -> #3 -> #1 -> ...
;

; <<< SYSTEM labels >>>
SYSTEM  EQU     ^X0000          ; Reset address
CONTRL  EQU     ^X0A1           ; Bank control port
STATUS  EQU     ^X0A0           ; Bank status port

; <<< Bank switching program >>>

        ORG     ^X0100          ; This program must be
                                ; stored between ^X0000
                                ; and ^X7FFF

CHECK:  DI                      ; Disable interrupt
        IN      STATUS          ; Read current bank status
        MOV     B,A             ; Save current bank status
        ANI     ^B00001100      ; Pick up high bank status
                                ;  only
NEXTB:
        ADI     ^B00000100      ; Set next bank data
        CPI     ^B00000100      ; This pattern was not used!
        JZ      NEXTB           ; Set up next bank data
                                ; for lap around

        MOV     C,A             ; Save new bank data
        MOV     A,B             ; Remember old bank status
        ANI     ^B11110011      ; Do not change bit data
                                ;  without RAM bank data
        ORA     C               ; Set new RAM bank
        OUT     CONTRL          ; Select bank
```

```
EI                          ; Enable interrupt
JMP     SYSTEM              ; We must update book
                            ;  keeping area.
                            ; Jump ^X0000 is the
                            ;  best way.
END
```

CHAPTER 3

HOW TO USE 2ND ROM

When you want to make some programs stored in 2nd ROM, there are a lot of matters should be attended and stored in the 2nd ROM. The matters are interrupt jump tables and power on/power off sequences. You have to implement these tables and sequences in order to process the ROM bank switching smoothly. Otherwise, PC-8201A will run away on switching the ROM bank. First half sections describe the interrupt functions and power sequence.

And you have to know the rules to handle the files and data in RAM, too. If you will use the routines in ROM #0 to handle the RAM, you need not to care about the detail rules. (You can get the information about the RAM file handling routines in ROM #0 at the Chapter 8 and another technical manual that has already been available by NEC HE in Chicago. Please request it if you have not gotten it yet.) The last half of this chapter describes how to use the routines in ROM #0 from 2nd ROM, ROM #1. (Hereafter ROM #1 sometimes represents 2nd ROM.)

If you want to make I/O control routines and store them in 2nd ROM, you have to understand Chapter 9 to 14. If you utilize the ROM #0's I/O routines, the last half of this chapter and another manual will help you.

3.1 CONSIDERATION OF INTERRUPT

Basically, PC-8201A has some interrupt service routines in that system. The main purpose of interrupts are smooth processing in Power off trap, reading data from Bar-code reader, communicating through UART(RS-232C) and using Interval timer.

The interrupt table is located in the zero page area.

POWER OFF TRAP	NMI	^X0024
BARCODE READER	RST 5.5	^X002C
UART	RST 6.5	^X0034
INTERVAL TIMER	RST 7.5	^X003C

The Interval timer interrupt has the highest priority, and UART has the second one. The lowest interrupt is used for Barcode reader. The reason why the internal timer has the highest priority is to scan the key and to count the auto-power off counter for saving the battery power. PC-8201A has the 'Auto-Power Off' function'. Usually, this function is executed after 10 minutes has past since last key stroke was detected. (This interval can be set by the 'POWER' command in BASIC. Refer 'PC-8201A Reference Manual'.) The interval time is used to count this period.

The interrupt hook table is located from ^XF386 to ^XF394. And that table is constructed in the following fig.

Interrupt hook table in RAM area

^XF386	POWER ON SEQUENCE
^XF389	BARCODE READER INPUT SEQUENCE
^XF38C	UART INPUT SEQUENCE
^XF38F	TIMER SEQUENCE and KEY SCANNING SEQUENCE
^XF392	POWER FAILURE SEQUENCE

3.1.1 Power Off Trap (ADDRESS ^X4CFA)

This interrupt is Non maskable. When power switch is turned off, this interrupt occurs. The following sequence is the algorism of this interrupt.

1: Disable the interrupt
2: Call hook table
3: Reset Key wait counter
4: Cancel Time counter
5: Out a data to the Auto power off port
6: HLT

The detail bit assignment of the auto power off port is following.

PORT ADDRESS ^XBA (OUT)
81C55 port B

MSB	7	6	5	4	3	2	1	0

Bit 7 --- RTS output
Bit 6 --- DTR output
Bit 5 --- BELL
0:Ring bell
1:Stop bell

Bit 4 --- Auto power off
0:Off
1:On

Bit 3 --- DCD/RD select
Bit 2 --- Melody control
0:On
1:Off

Bit 1 --- LCD chip select #1
Bit 0 --- LCD chip select #0

rf.Chapter 9 to 15 about more detail information of this port.

3.1.2 Barcode Reader

(ADDRESS ^XF389 with Disable interrupt)

This interrupt is using RST 5.5. If you do not use barcode reader program, this interrupt should do 'RETURN' soon.

3.1.3 UART

(ADDRESS ^X6E00 with Disable interrupt)

This interrupt is using RST 6.5. This interrupt is caused by UART. (Serial communication device 6402) This interrupt occurs when the data in 6402 receive buffer is available.

The algorism of this interrupt is shown below.

1: Disable the interrupt
2: Call hook table
3: Read data from 6402
4: Read error status from 6402
5: Xon/Xoff control check
6: SI/SO control check
7: Return to previous process

PORT ADDRESS ^XD8 (OUT)
UART control port

	7	6	5	4	3	2	1	0
MSB								

Bit 7 --- Not used
Bit 6 --- Not used
Bit 5 --- Not used
Bit 4 --- Character length select #2
Bit 3 --- Character length select #1
Bit 2 --- Parity inhibit
0:Parity generation Check
1:Parity generation check, Inhibit

Bit 1 --- Even parity enable
0:Odd parity
1:Even parity

Bit 0 --- Stop bit select
0:Stop bit 1 bit
1:Stop bit 1.5 bit in case of DATA Length is 5
1:Stop bit 2 bit in case of DATA Length is not 5

PORT ADDRESS ^XC8 (OUT)

UART data I/O port

	7	6	5	4	3	2	1	0
MSB								

Bit 7	---	Data #7
Bit 6	---	Data #6
Bit 5	---	Data #5
Bit 4	---	Data #4
Bit 3	---	Data #3
Bit 2	---	Data #2
Bit 1	---	Data #1
Bit 0	---	Data #0

rf. Chapter 12 and 15 about more detail information about UART.

3.1.4 Interval Timer (ADDRESS ^X1EBE With Disable Interrupt)

This interrupt is using RST 7.5. This is the interrupt from interval timer. (Timer device 1990) This interrupt is also used for the key scanning.

In the system's initialization, the interval timer which is controlled by 1990, is set up as 4m second mode. The port for 1990 is illustrated below.

PORT ADDRESS ^XB9 (OUT)

Calendar clock (1990) control port

	7	6	5	4	3	2	1	0
MSB								

Bit 7	---	Not used
Bit 6	---	Not used
Bit 5	---	Not used
Bit 4	---	Data output
Bit 3	---	Shift clock
Bit 2	---	Command output #2
Bit 1	---	Command output #1
Bit 0	---	Command output #0

Command #2	Command #1	Command #0	
1	0	0	timing 64Hz
1	0	1	Timing 256Hz
1	1	0	Timing 2048Hz
1	1	1	TEST mode

In the initialization routine, the command is set up as ^X05. It means 4m second interval.

rf. Chapter 15 for more information about 1990

The following step is the algorism about interval timer sequence.

1: Disable the interrupt
2: Call hook table
3: Mask RST 7.5,RST 5.5
4: Reverse cursor character for cursor blink
5: Key matrix scanning
6: Return to the interrupted process

3.2 ROM SWAPPING METHOD

When you would like to use 2nd ROM, you must write the following information into the 2nd ROM's special reserved area. The special reserved area is located from ^X0000 to ^X0047. These area will be used for 2nd ROM starting jump instruction and ID code, and the file name of 2nd ROM. This name is displayed like a one of the RAM files on Menu screen by 1st ROM, ROM #0. The following figure is the explanation about 2nd ROM special reserved area.

```
ADDRESS          CODE
^X0000 JMP       START    ; 2nd ROM start address
^X0003 ....
^X0024           RET      ; Non maskable interrupt
^X002C           RET      ; Barcode reader interrupt
^X0034           RET      ; UART interrupt
^X003C           RET      ; Interval timer interrupt
^X003F                    ; Reserved for RST interrupt
^X0040           DB       'A'
^X0041           DB       'B'      ; ID code for 2nd ROM
^X0042           DB       '2NDROM'; File name which displayed in
                                   ; the menu

^X0048 START:                      ; 2nd ROM code
```

S P E C I A L R E S A R V E D A D D R E S S

If these data are implemented correctly, the name will appears on the 1st ROM's menu screen. So it's easy to switch the ROM and execute the program in it. When you want to start the programs in 2nd ROM from the Menu mode of ROM #0, move the cursor to 2nd ROM's file name on the screen. Then please press return key. The system will fall into the 2nd ROM program.

3.3 THE METHOD TO USE 1ST ROM ENTRY FROM 2ND ROM

If you want to use the routines in 1st ROM from 2nd ROM, at the first, you have to create a special routine in the higher memory location of RAM (^X8000-^XFFFF) and use it. That routine switches the ROM bank with using bank switching method, and calls the routine in 1st ROM. It is very important for you that the interrupts must be disabled before you change the ROM banks. And in addition, as the following sections will tell you, you have to change the hook table for Power down interrupt that was changed by 2nd ROM to restart the current process in 2nd ROM program at next power-on. With this hook table for 2nd ROM, the power down in ROM #0 will cause the fatal error. Power-off interrupt can not be prohibited. And you have to consider about the contents of the routine which you will call. The reason is that some routines in the 1st ROM routine may enable the interrupts in some parts of their code even if you disable the interrupts just before switching the ROM banks to call 1st ROM entry. Therefore you had better change the all hook tables in the current book keeping area. I suggest that all hook table should be replaced with previout contents which were stored by 1st ROM, just before calling ROM bank-switching routine ,and restored just after coming back from 1st ROM.

The following program is the sample which uses 1st ROM entry points from 2nd ROM.

3.3.1 Sample

```
;       TITLE   Using 1st ROM entry from 2nd ROM
;
;
;       This sample will enable to use 1st ROM entry from
;       2nd ROM.
;       Some routines in 1st ROM might enable interrupts,
;       so all interrupt
;       hook table should be replaced with RET code.
;       And restore them after done the 1st ROM calling.
;
;       Entry   [ENTRY]:1st ROM entry address
;       Exit    for return condition of 1st ROM
;

; <<< SYSTEM define label >>>
BNKCRL  EQU     ^X0A1           ; Bank control port
STATUS  EQU     ^X0A0           ; Bank status port

; <<< Main routine >>>
        ORG     ^X8000          ; This routine must be stay
                                ;  ^X8000-^XFFFF

ROM1ST: SHLD    WORKH           ; Save register HL
        LXI     H,RET2ND        ; Return address from 1st ROM
        PUSH    H               ; Push stack top
        LHLD    ENTRY           ; Pick up 1st ROM entry
                                ; address
        PUSH    H               ; Push stack top
        LHLD    WORKH           ; Restore HL
        PUSH    PSW             ; Save all register
        DI                      ; Disable interrupt
        IN      STATUS          ; Get current bank status
        ANI     ^B11111110      ; Switch 1st ROM data set up
        OUT     BNKCRL          ; Bank select
                                ; Now ^X0000-^X7FFF are
                                ; 1st ROM
        EI                      ; Enable interrupt
        POP     PSW             ;
        RET                     ; Jump 1st ROM entry

; <<< Return from 1st ROM >>>
RET2ND: PUSH    PSW             ; Save all register
        IN      STATUS          ; Get current bank status
        ORI     ^B00000001      ; Switch 2nd ROM data set up
        OUT     BNKCRL          ; Bank select
                                ; Now ^X0000-^X7FFF are
                                ; 2nd ROM
        POP     PSW             ; Pick up all register
```

```
        RET                         ;

; <<< SYSTEM WORK AREA >>>
ENTRY:  DW      ^X0000              ; 1st ROM entry address
WORKH:  DW      ^X0000              ; HL register saving area

        END
```

3.4 SEQUENCES IN THE 2ND ROM

1. INITIALIZE

This sequence sets up [SP](Stack Pointer), power-on trap and other interrupt routines. Then it copies the book-keeping area and system area. finally some peripherals will be initialized by this routine.

2. RETURN TO MENU

At the first, this sequence selects the standard RAM, RAM #0 and resets the power-off trap. Then it jumps to 1st ROM's menu mode.

3. POWER DOWN

When power is turned off, the control is transferred to this sequence. In this sequence, you must save all registers and circumstances which should be saved in the stack. So the stack-pointer is most important to resume the current processing on the next power-on.

The RAM bank number is always stored in RAM #0. On turning on, the 1st ROM and RAM #0 is selected automatically. And the bank-switching procedure will be called in Power on sequence if the number of the RAM bank was not identical to the RAM #0 in the power down sequence. After changing the RAM bank, all registers will be restored and pending procedure will be resumed. Therefore in the stack, the address of the process which was abandoned by Power down trap should be stored.

In addition, in order to resume the abandoned process with 2nd ROM, you have to do special power on/power off sequence. In power off trap, you should the set the start routine of the special power-on sequence which switches the ROM bank. I recommend to use the hook, ^XF38F. Usually, 'JUMP to POWER FAIL SEQUENCE' command is stored here. In 2nd ROM, however, you have to rewrite this hook table and call the special power down routine here. In

it, the address of special power-on routine on the stack. In this case, the following information should be stacked before 'HLT' command is executed.

```
|                        |
|------------------------|
| resuming  address      |
|                        |
|------------------------|
| starting address of    |
|  the ROM switching     |
|   routine              |
|------------------------|
| Contents of Pointers   |
|                        |<-- [STAKSV]
|------------------------|
```

[STAKSV] keeps the SP's value at 'HLT'.

Fig 3.1

4. POWER ON

At the first, the initializing routine in ROM #0 checks the RAM bank number in BANK (^XF3DB) when power-off was executed. When power-off was done in non-standard RAM bank, RAM bank-switching routine is called and switched. Then, the registers' contents will be restored. If the address of the process which should be resumed was stacked, the address will be picked up and executed. When the power-down was detected in ROM #1, the address of the special ROM switching routine ought to be stacked above the address of the process should be resumed. Therefore, after switching the ROM, the abandoned process will be resumed.

The following figure are the general 2nd ROM routine control sequence.

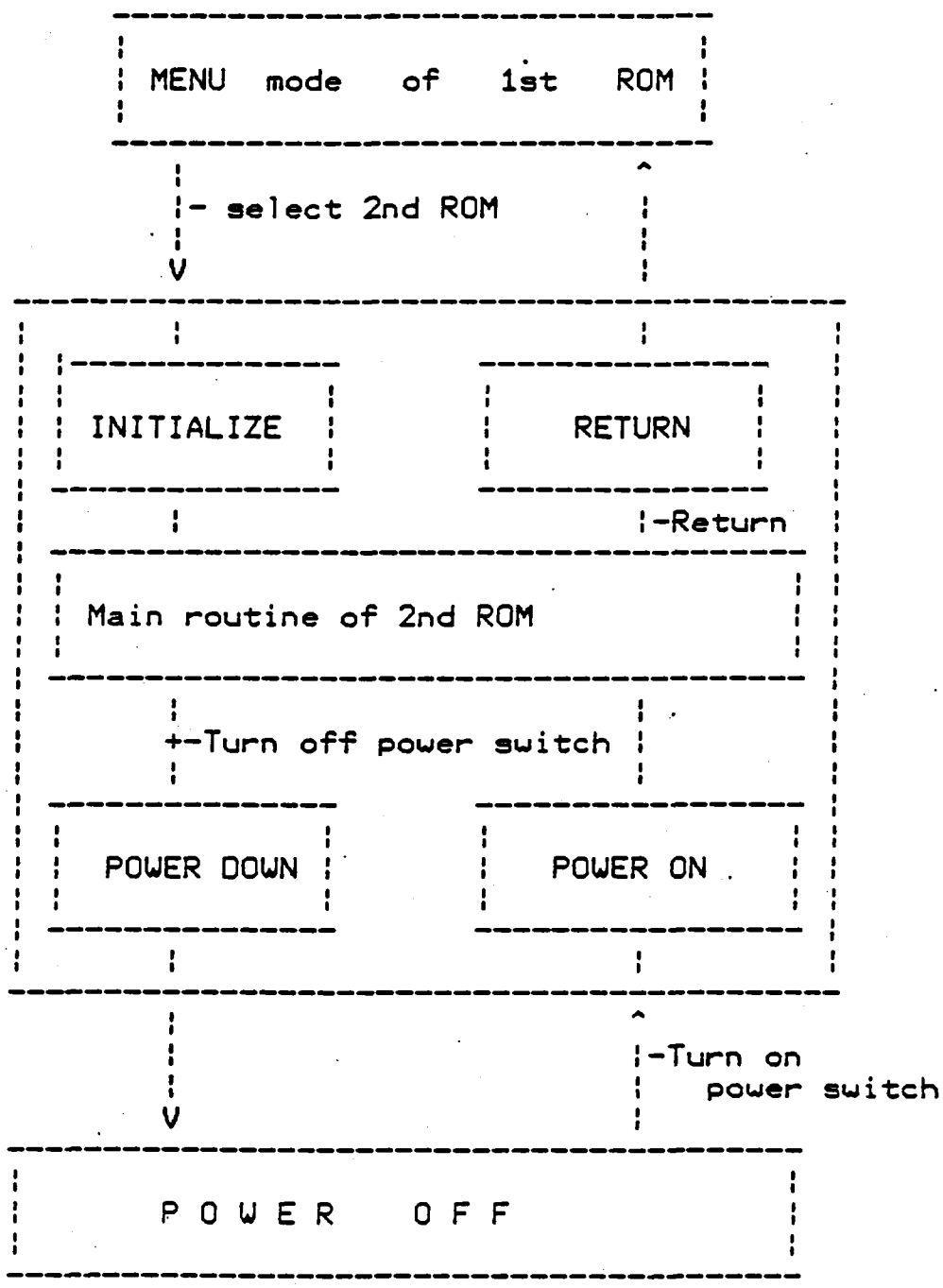


Fig 3.2

3.5 SUMMARY -- IMPORTANT NOTICE

If you want to make 2nd ROM program, you should take care of the following manner.

1. Interrupt vector

   If you do not want to use interrupt, all interrupt table should be set with only 'RET' code. But I suggest you that you had better use interval timer interrupt, because of saving the battery power by using auto power off function. The counter for this auto power off function is counted by this interval timer interrupt. If you do not use this function, the battery consumption may be more larger than now.

2. Bank of RAM

   Do not switch the ROM bank when PC, Program Counter, points a routine in that ROM. You can guess the reason and it's not so hard to imagine these bank switching will cause the fatal problem for system. At the worst case, the all files which you stored will be lost. And also you should be careful in stack area, too.

3. PC-8201A book keeping area

   The book keeping area are very important for this system, so you never change that area without careful consideration. Please read Chapter 7 'BOOK KEEPING AREA'.

4. Power on/off sequence

   Please use power off interrupt to detect the power down. I suggest that you had better use the real time interrupt service to poll the power down signal.

If you want to use 1st ROM entry from 2nd ROM, please take care of the following point. The all routines rewrite some work area sometimes. So, if you use 1st ROM entry from 2nd ROM without understanding that routine's internal specification, the system might be crashed. In addition, interrupts and stack area are other important points. Refer to 2.3 'The method to use 1st ROM Entry from 2nd ROM' and its sample program.

3.6 SAMPLE

```
        TITLE   2nd ROM sample header and useful routine

; <<< SYSTEM define label >>>
BANK    EQU     ^XF3DB          ; Bank save area
ATIDSV  EQU     ^XF382          ;
PWHOK   EQU     ^XF386          ; Power on hook table
RST55   EQU     ^XF389          ; Rst 5.5 hook table
STAKSV  EQU     ^XF9AE          ;
AUTOID  EQU     ^X9C0B          ;
SAVSTK  EQU     ^XFAD0          ;
STATUS  EQU     ^XA0            ; Bank status
BNKCRL  EQU     ^XA1            ; Bank control
PWPORT  EQU     ^XB8            ; 81C55 chip select
PORTB   EQU     ^XBA            ; 81C55 port B

FREE    EQU     ^X????          ; You must set your ram
                                ; free portion address
; <<< Main routine >>>

        ORG     ^X0000
START:
        JMP     INIT            ; 2nd ROM start address

        ORG     ^X0024          ; Non maskable interrupt
                                ; table
        JMP     POWER           ; Power down trap

        ORG     ^X002C          ; RST 5.5
        JMP     BARCOD          ; Barcode reader interrupt
                                ; table

        ORG     ^X0034          ; RST 6.5
        JMP     UART            ; UART interrupt table

        ORG     ^X003C          ; RST 7.5
        JMP     TIMER           ; Timer interrupt table

        ORG     ^X0040          ; ID code for 2nd ROM
        DB      'AB'            ; AB is ID code for 2nd ROM

        DB      '2NDROM'        ; File name which
                                ; displayed in the MENU

; <<< Initialization of 2nd ROM program >>>
INIT:   LHLD    SAVSTK          ; Set stack pointer
```

```
        SPHL                        ;
        CALL    SETTRP              ; Set hook for resume
                                    ; 2nd-ROM's program,
                                    ; and other routine into RAM.
        CALL    HINIT               ; Hardware initialization
        JMP     MAIN                ; Goto main routine

;
; <<< Hardware initialize routine >>>
HINIT:  RET                         ;

;
;
; <<< MAIN ROUTINE OF 2ND ROM >>>
;
;
MAIN:                               ; Main routine

; <<< Set up hook >>>
; Set up hook table for 2nd ROM
SETTRP: MVI     A,^B00000001        ; Select standard RAM
        OUT     BNKCRL              ; Select!
        LXI     H,DTBL              ; Set some codes into RAM
        LXI     D,PWHOK             ; for power on sequence
        MVI     B,TBLEND-DTBL       ;
        CALL    COPY                ;
        LXI     H,TBLHOK            ; Return code table
        LXI     D,FREE              ; Free area of RAM portion
        LXI     B,HOKE-TBLHOK       ; Set length
        CALL    COPY                ;
        RET                         ;

; [DE] <- [HL]
COPY:   MOV     A,M                 ; Read [HL]
        STAX    D                   ; Save [DE]
        INX     H                   ;
        INX     D                   ; Next address set
        DCR     B                   ; Decrement counter
        JNZ     COPY                ; Loop until done
        RET                         ;

;
; The following code will be copied in RAM
; portion for re-power on sequences
; these part are interrupt hook table.
;
DTBL    EQU     $
        MVI     A,^B00000001        ; These code will be
                                    ; copied into RAM
        OUT     BNKCRL              ; Bank select!
```

```
        JMP     PWON            ; Jump power on trap
BANKI:  DS      1               ;
TBLEND  EQU     $

;
; The following code will be copied
;  in RAM portion for return 1st ROM
;
TBLHOK  EQU     $
RETSB:  XRA     A               ; Clear A
        OUT     BNKCRL          ; Select 1st ROM and
                                ; standard RAM
        JMP     ^X0000          ; Return!
HOKE    EQU     $

; <<< RETURN >>>
RETURN: MVI     A,^B00000001    ; Select standard RAM
        OUT     BNKCRL          ;
        MVI     A,^B00000000    ;
        STA     BANK            ;
        LXI     H,^X0000        ; Reset
        SHLD    ATIDSV          ;
        LXI     H,RTBL          ; Rewrite code table
        LXI     D,PWHOK         ; Interrupt hook table set
        LXI     B,RTBLE-RTBL    ; Set length
        CALL    COPY            ;
        JMP     RETSB           ; Return to 1st
                                ; ROM's menu mode

;
; The following code will be copy
;  in standard ram portion.
;
RTBL    EQU     $
        RET                     ; Power on hook
        NOP
        NOP
        EI                      ; RST 5.5 hook
        RET
        NOP
RTBLE   EQU     $

; <<< Power on >>>
PWON:   CALL    HINIT           ;
        LDA     BANKI-DTBL      ; Select old RAM bank
        OUT     BNKCRL          ;
        LHLD    STAKSV          ; Restore stack pointer
        SPHL                    ;
        POP     PSW             ;
        POP     B               ;
```

```
        POP     D                   ;
        POP     H                   ;
        RET                         ; Resume old program

; <<< POWER DOWN TRAP >>>
POWER:  PUSH    PSW                 ;
        IN      PWPORT              ; Read power down port
        ANA     A                   ; Check
        JM      NTPWFL              ; No power down
        POP     PSW                 ;
        DI                          ; Disable interrupt
        PUSH    H                           ; Save HL
        PUSH    D                   ; Save DE
        PUSH    B                   ; Save BC
        PUSH    PSW                 ; Save AF
        LXI     H,^X0000            ;
        DAD     SP                  ; Now I know stack address
        SHLD    STAKSV              ; Save stack
        MVI     A,0FFH              ; Reset interval timer
                                    ; counter
        STA     PWRINT              ; Set up for next power on
        IN      STATUS              ; Save current RAM bank status
                                    ; When power on resume,
remember
                                    ; this and select RAM bank.
        MOV     B,A                 ; Save it
        MVI     A,^B00000001        ; Select standard RAM
        OUT     BNKCRL              ; Select!
        MOV     A,B                 ; Resave old status
        STA     BANKI-DTBL          ;
        MVI     A,^B00000001        ; Select RAM bank 1
        OUT     BNKCRL              ;
        MVI     A,0                 ; Set up to come back
                                    ; to 2nd ROM
        STA     BANK                ;
        LXI     H,AUTOID            ;
        SHLD    ATIDSV              ;
        IN      PORTB               ;
        ORI     ^B00010000          ;
        OUT     PORTB               ;
        HLT                         ; Never go on
;
NTPWFL: POP     PSW                 ;
        RET                         ;


; <<< BARCODEREADER interrupt >>>
BARCOD: RET                         ; Return soon

; <<< UART interrupt >>>
```

```
UART:   RET                         ; Return soon

; <<< Interval Timer interrupt >>>
TIMER:  LDA     PWRINT              ; Pick up timer value
        DCR     A                   ; Decrement!!
        STA     PWRINT              ; Save it
        RET                         ;

; <<< System work area >>>
PWRINT: DB      ^X0FF               ; Timer counter n * 1/256Hz

        END
```

CHAPTER 4

HOW TO USE 2ND/3RD RAM

When you want to change the bank of RAM, the most simple method is to do OUT instruction and to jump ^X0000 for warm start. Because book keeping area management is too difficult to do by yourselves, I think. But if you would not like to do warm start, you must manage the book keeping and system parameter by yourself and use the special RAM bank handling routine. You can easily guess that when the bank of RAM is changed, PC, the program counter must stay lower than ^X7FFF. Because bank switch is completely change the code of RAM which address ^X8000 to ^XFFFF. But the area from ^X0 to ^X7FFF is used for ROM. The only one way is to make a special RAM bank switch routine in all RAM banks with same address. The following illustration will help you to understand this curious method.

```
|                  |  |                  |
| POP   HL         |  | POP   HL         | ;Pick up return address
| MOV   A,NEXT     |  | MOV   A,NEXT     | ;set next bank status
| OUT   ^XA1       |  | OUT   ^XA1       | ;change bank
| PUSH  H          |  | PUSH H           | ; set return address
| RET              |  |                  | ;return to specified addres
|                  |  |                  |
     RAM #0            RAM #1
```

Fig 4.1

Same routine is stored in the same position of 2 RAM banks. Refer to next
section to write a program at another bank.

In addition you must take care of the STACK POINTER ,too.

4.1 READ AND WRITE TO ANOTHER RAM BANK

These are two methods to read /write another bank of RAM. The first is more simple than second one. But the first method is some limitation of that performance, because this method uses ROM #1. And the second method is more complex, but this is more powerful. The size of the second method is longer than the first one.

4.1.1 Method 1 [USING 1st ROM]

These are very useful routines in the 1st ROM. These are GETBNK and PUTBNK.

4.1.1.1 GETBNK [^X7EEC]

This routine reads one byte from other banks of RAM. The GETBNK routine temporarily changes the specified RAM bank, reads a byte pointed by [HL], and returns to the original bank. Interrupt should be disabled before calling the GETBNK routine.

```
Entry    [B] = Bank number
                 ^X00:Main bank
                 ^X08:Bank #2
                 ^X0C:Bank #3
         [HL] = Address which byte to read

Exit     [D] = Byte data which read

Altered registers
         [A],[C],[D],[F]
```

4.1.1.2 PUTBNK [^X7EEB]

The PUTBNK routine writes one byte at the specified address pointed by [HL] in the specified RAM bank. Similar to the GETBNK routine, original bank will be selected after writing that data. Before using the PUTBNK routine, interrupt should be disabled.

```
Entry    [B] = Bank number
                 ^X00:Main bank
                 ^X08:Bank #2
                 ^X0C:Bank #3
         [HL] = Location where the byte is stored
         [D] = Byte data to be stored

Exit     None

Altered registers
         [A],[C],[F]
```

4.1.2 Method 2 [USING YOUR ORIGINAL CODE]

When your code is located in upper address (^X8000-^XFFFF), and you want to read/write a lot of data in another bank of RAM, you had better change the target RAM bank at the lower position of the memory.

(1) Your code is in RAM #1. And data you want to read or write is in RAM #2.

```
^XFFFF  -------- --------
        |      | |      |
        | Your | | RAM  |
        | code | | #2   |
        |      | |      |
^X8000  -------- --------

^X7FFF  --------
        |      |
        | ROM  |
        |      |
        |      |
^X0000  --------
```

Fig 4.2

(2) Change the Bank.

```
^XFFFF  ---------
        |       |
        | Your  |
        |  code |
        |       |<--
        |       |   |
^X8000  ---------   |Handle
^X7FFF  ---------   | some
        |       |   |  data
        | RAM #2|   |
        |       |   |
        |       |<--
^0000   ---------
```

Fig 4.3

(3) Then change again into previous Bank configuration.

In this case, you have to disable to all interrupts before changing the BANK.

When your code is located lower addres (^X0000-^X7FFF), for instance, running a program in 2nd ROM, please use next method to handle the data in other RAM banks.

(1) The program in 2nd ROM is running with RAM #1.

```
^XFFFF  ---------- ---------
        |standard| |       |
        |        | |  RAM  |
        |  RAM   | |  #2   |
        |        | |       |
^X8000  ---------- ---------

^X7FFF  ----------
        |        |
        |  2nd   |
        |  ROM   |
        |        |
        |        |
        |        |
^X0000  ----------
```

Fig 4.4

(2) Read or Write RAM #2 by bank switching during all interrupts prohibited.

```
^XFFFF  ---------- -----------
        |        | |         |
        |  RAM   | |  RAM    |
        |  #2    | |standard |
        |        | |         |
        |        | |         |
^X8000  ---------- -----------
^X7FFF  ----------
        |        |
        |  2nd   |
        |  ROM   |
        |        |
        |        |
^X0000  ----------
```

Fig 4.5

(3) Switch again, and resume the previous processing.

TITLE Read Write routine for another BANK of RAM

This sample will access another bank of RAM.
There are two routines in this source program.
One is having access in byte by byte by using
.
The another one is to access in block of data to use
special bank switching.
In the architecture of bank, bank 1 (Standard RAM)
is not able to be switch low address
(^X0000H-^X7FFF).

```
Entry   HL:Address to be accessed
        C :Bank number
Exit    B :Data which be read

Entry   HL:Address to be accessed
        C :Bank number
Exit    B :Data to be written

Entry   HL:Start address to be changed
        A :Bank number
        DE:Start address in current bank
        BC:Byte length to be read
Exit    None

Entry   HL:Start address to be written
        A :Bank number
        DE:Start address in current bank
        BC:Byte length to be written
Exit    None

Bank number
        Bank #1 (Standard RAM)  :^X00
        Bank #2 (RAM #2)        :^X08
        Bank #3 (RAM #3)        :^X0C
```

```
(STEM label define >>>
EQU     ^X0A1           ; Bank control port
EQU     ^X0A0           ; Bank status port

ORG     ^X0000          ; This program can be located
                        ;  any place
                        ; This switch should be change
                        ; according to the situation
EQU     -1              ; High address (^X8000-^XFFFF)
```

```
SLOW    EQU     0                       ; Low address  (^X0000-^X7FFF)

; <<< Byte access routine >>>
BYTER:  DI                              ; Disable interrupt
        IN      STATUS                  ; Read current bank status
        PUSH    PSW                     ; Save current bank status

        IF      SHIGH
        ANI     ^B11110011              ; Clear high address of bank
                                        ; switch
        ORA     C                       ; Set new data of bank
        ELSE
        PUSH    PSW                     ; Save current bank
        MOV     A,C                     ; Pick up new bank data
        RAR                             ;
        RAR                             ; Shift 2 bit
        MOV     C,A                     ; Restore bank data
        POP     PSW                     ; Pick up current bank
        ANI     ^B11111100              ; Clear low address of bank
                                        ; switch
        ORA     C                       ; Set new data of bank
        ENDIF

        OUT     BNKCRL                  ; Select new bank!
        MOV     B,M                     ; Read data from some bank
        POP     PSW                     ; Pick up before bank
        OUT     BNKCRL                  ; Select before bank
        EI                              ; Enable interrupt
        RET                             ;

BYTEW:  DI                              ; Disable interrupt
        IN      STATUS                  ; Read current bank status
        PUSH    PSW                     ; Save current bank status

        IF      SHIGH
        ANI     ^B11110011              ; Clear high address of bank
                                        ; switch
        ORA     C                       ; Set new data of bank
        ELSE
        PUSH    PSW                     ; Save current bank
        MOV     A,C                     ; Pick up new bank data
        RAR                             ;
        RAR                             ; Shift 2 bit
        MOV     C,A                     ; Pick up current bank
        ANI     ^B11111100              ; Clear low address of bank
                                        ; switch
        ORA     C                       ; Set new data of bank
        ENDIF

        OUT     BNKCRL                  ; Bank switch!
```

```
        MOV     M,B             ; Write data
        POP     PSW             ; Pick up before bank
        OUT     BNKCRL          ; Select before bank
        EI                      ; Enable interrupt
        RET                     ;

; <<< Block access routine >>>
BLOCKR: DI                      ; Disable interrupt
        PUSH    B               ; Save length
        MOV     C,A             ; Set up bank number
        IN      STATUS          ; Read current bank status
        STA     CURBNK          ; Save current bank

        IF      SHIGH
        ANI     ^B11110011      ; Clear high address of bank
                                ; switch
        ORA     C               ; Set new data of bank
        ELSE
        PUSH    PSW             ; SAve current bank
        MOV     A,C             ; Pick up new bank data
        RAR                     ;
        RAR                     ; Shift 2 bit
        MOV     C,A             ; Restore bank data
        POP     PSW             ; Pick up current bank
        ANI     ^B11111100      ; Clear low address of bank
switch
        ORA     C               ; Set new bank data
        ENDIF
        POP     B               ; Pick up length
NEXTR:
        LDAX    D               ; Read data
        MOV     M,A             ; Write data
        INX     D               ;
        INX     H               ; Next position of data
        DCX     B               ; Decrement counter
        JNZ     NEXTR           ; Loop until done

        LDA     CURBNK          ; Set previous bank
        OUT     BNKCRL          ; Select previous bank
        EI                      ; Enable interrupt
        RET                     ;

BLOCKW: DI                      ; Disable interrupt
        PUSH    B               ; Save length
        MOV     C,A             ; Set up bank number
        IN      STATUS          ; read current bank status
        STA     CURBNK          ; Save current bank

        IF      SHIGH
        ANI     ^B11110011      ; Clear high address of bank
```

```
swit
        IRA     C                   ; Set new data of bank
        LSE
        USH     PSW                 ; Save current bank
        OV      A,C                 ; Pick up new bank data
        AR                          ;
        AR                          ; Shift 2 bit
        OV      C,A                 ; Restore bank data
        OP      PSW                 ; Pick up current bank
        NI      ^B11111100          ; Clear low address of bank
switch
        JRA     C                   ; Set new bank data
        ENDIF
        POP     B                   ; Pick up length
NEXTW:
        MOV     A,M                 ; Pick up data
        STAX    D                   ; Write data
        INX     H                   ;
        INX     D                   ; Next position of data
        DCX     B                   ; Decrement counter
        JNZ     NEXTW               ; Loop until done

        LDA     CURBNK              ; Restore previous bank #
        OUT     BNKCRL              ; Select previous bank
        RET                         ;

; <<< System work area >>>
CURBNK: DB      ^X00                ; Current bank data

        END
```

CHAPTER 5

UNDERSTANDING THE RAM FILE CONCEPT

5.1 SUMMARY

Usually, the RAM files are controlled by the ROM #0, settled in ROM socket #0 at the shipment. There are many rules to use the RAM file. Unless you replace this ROM #0 with your own ROM, ROM #0 checks the RAM file organization and pointers in the bookkeeping area sometimes, even if you don't use BASIC, TEXT or TELCOM. (For instance, at Power on and 'Bank' command in menu.) If you ignore the standard rules for RAM file handling, ROM #0 will flush not only the files which were made by your own application program ,but also the files which were made by BASIC and TEXT in ROM #0. In order to save your files from such kinds of accidents, please read following chapters about the RAM file handling and understand the standard rules in PC-8201A.

The two situations were considered for this section. Someone wants to handle RAM files with the machine language subroutine in the BASIC mode. In this case, opening the file will be done by a BASIC command, OPEN. And the file will be closed and deleted by CLOSE and KILL command in BASIC. So the machine language subroutine will make up the lacking facilities in BASIC commands. For instance, Insert a data at the middle of the opened file. In this case, you had better care about a few pointers only. You needn't know the directory structure.

But another person might try to make a his (or her) original application program without using BASIC. He (or She) might open a file, save data, append data, insert data, delete data and erase a file with his (or her) own application. In this case, the name of the data file should be registered by that application program. So that programmer need to know the Directory configuration and many parts of the pointers playing.

This section is written for supporting both of them. The programmer who wants to make a original application without BASIC, needs much more information than a user who uses BASIC. But too much data sometimes confuses a novice programmer who wants to make a subroutine for BASIC main program. After long consideration, I decided to obey the famous common saying, 'The greater serves for the lesser'. Therefore I serve everything what I know. Please find what you want to know in the following section.

In these chapters, I tried to describe each section independently. You, however, might meet unknown words sometimes. Please refer to another section or another chapter at that time. I hope you will make many good application programs with this document.

5.2 WHAT IS RAM FILE?

In PC-8201A, you can have many files in RAM area at a time ,like files on the floppy disk. The files are classified into three suffixes:.DO(cument) ,BA(sic) and .CO(mmand). Hereafter .DO(cument) file is abbreviated DO file, BA(sic) file is BA file, and .CO(mmand) file is CO file. And sometimes the word 'ASCII file' is used in place of 'DO file'.

5.2.1 DO File (ASCII File)

The DO file is created by BASIC, TEXT and TELCOM. Of course, you can load a DO file from I/O in menu mode. In BASIC, the 'OPEN' command handles the DO file. The OPEN command with'FOR OUTPUT' option makes a new DO file. OPEN with 'FOR APPEND' opens the DO file in order to add the data after the last data that has already been entered. When there is no file whose name is same as the specified in the 'OPEN' with 'FOR APPEND', that OPEN command works as the OPEN with FOR OUTPUT. The OPEN with FOR INPUT opens the specified file to be ready for sequential reading.

The 'SAVE' command with ',A' option or 'SAVE' command with the file descriptor followed the suffix, '.DO' stores a BASIC program as a DO file. This DO file is, sometimes, called as ASCII (saved program) file. (Note: A SAVE command without ',A' option creates a BA file.) In this case, the BASIC program in the BASIC files area is saved into the DO files area in the ASCII format. So you can read it in TEXT mode.('SAVE' command without ',A' or without the suffix, '.DO' only registers the file name with the suffix, '.BA' and changes some pointers. It does not make a new file. Please refer to next section about BA files. And I think almost BASIC interpreter have this 'ASCII save function' for the disk files. Refer to BASIC reference manual if you have another disk top personal computer's manual.)

```
                    |                  |
                    |                  |  ^
                    ------------------ |  Upper
                    |                  |  |
                    | DO files         |  :
                    |                  |
                    ------------------ |
                    |                  |  |  Lower
                    | non-registered   |  V
                    |  BASIC program   |
                    |     or           |
                    | saved BASIC      |
                    |  program         |
                    |  'A.BA'          |
                    |                  |
                    ------------------ <-- TXTTAB
                    |                  |
```

Type BASIC program in BASIC mode.
Do 'SAVE' command.

```
SAVE 'TEST',A
   or
SAVE 'TEST.DO'
```

```
                    |                  |
                    |                  |
                    ------------------ ^
                    |                  |  |Upper
                    | DO files         |  |
                    |                  |
                    ------------------
                    |                  |
                    | TEST.DO          |<-- New DO file
                    |                  |     is inserted
                    ------------------  |
                    |                  |  |Lower
                    | non-registered   |  V
                    |  BASIC program   |
                    |     or           |
                    | saved BASIC      |
                    |  program         |
                    |  'A.BA'          |
                    |                  |
                    ------------------ <-- TXTTAB
                    |                  |
```

Fig 5.1 SAVE with '.DO' or ',A' option

There are 2 type of hidden DO files in PC-8201A. One is the 'SCRAP' file used in TEXT, and another is the 'EDIT' file used in BASIC. The screen oriented text editor in PC-8201, named TEXT, has wonderful functions called 'CCP'. The CCP functions mean SELECT, CUT, COPY and PASTE. (The detail information about these functions are explained in the PC-8201A user's guide.) The CUT command or COPY command after SELECT command makes a temporary DO file. This DO files can be invoked by PASTE key many times. Though this file cannot be found in menu level, this file will be kept until next SELECT-COPY or SELECT-CUT will be executed and is not broken by the PASTE key.

And more good feature is in this DO file. Since the contents of this DO file is treated as the data from keyboard, this file can be used in BASIC. After saving a part of a file in SCRAP with SELECT-COPY function, return to Menu, and invoke BASIC. The contents of this 'SCRAP' file will appear by 'PAST' key. (In the PC-8201A user's guide, this temporary DO file is called 'PASTE buffer'.)

Another one, 'EDIT' file, is created by EDIT command in BASIC. The EDIT command in BASIC falls into the TEXT editor with the BASIC file. At that time, the BASIC program is translated in to ASCII format file, 'EDIT', and original BASIC file is killed. This file is erased when the EDIT mode is finished by double ESC or F.5, converted into BASIC file and saved. So no one can find this file at the menu level.

The DO file usually consist of the 'ASCII' characters. And you cannot use the 3 Control Characters, NULL (0), Control-Z (26) and Back Space (127). (The 'Control-Z' is sometimes abbreviated as '^Z'.) The Control-Z is used as the End of DO file. So if you store it as a one of the data in the middle of the DO file, the standard programs, BASIC, TEXT and TELCOM ,will regard that Control-Z as the End of that DO file. The data after that Control-Z will be lost. Otherwise the NULL is used to fill the hole dug by MAKHOL. After copying or inserting the data in to the hole, some routines tries to find the end of the data by finding the NULL. Then a routine squeezes the NULLs. Therefore the NULL in the middle of the DO file might cause the serious problems. similarly, the Back Space has special meaning in DO file. Please don't use there three Control characters in the DO file. BASIC's PRINT # command cannot save these control characters in to the DO file.

NOTE: MAKHOL and MASDEL are name of the routine

stored in ROM #0. Refer to 'Useful Routines for RAM file handling in ROM #0'.

ex. When DO file is made in PC-8201A

1. TEXT always creates and modifies DO files.

2. SAVE command with ',A' creates a DO file in BASIC.

3. UPLOAD and DOWN LOAD sends or receives a DO file through RS-232C in TELCOM.

4. DO file can be saved or loaded from CASSETTE and RS-232C in MENU.

5. OPEN with 'FOR OUTPUT' registers the file name and insert only End of file character as the DO file in BASIC.

5.2.2 BA File

The BA file is made in BASIC mode or made by LOAD function in Menu mode. There are two types of BA file in PC-8201A. One is a 'saved' BASIC program, and another is 'non-registered' BASIC program. Sometimes the 'non-registered' is called the 'un-saved' BASIC program, because 'un-saved' will make sense more than 'non-registered' for a person who knows BASIC very well. The BASIC program typed just after selecting BASIC mode in menu level, is called 'non-registered' BASIC file, since the name of the program has not been registered in the directory area yet. But after executing 'SAVE' command in BASIC mode, that 'non-registered' BASIC program becomes a 'saved' BASIC program. (In the point of view, 'SAVE' command in BASIC, the word 'un-saved' and 'saved' are suitable, I think.) The 'SAVE' command in BASIC 'register's the file name and the starting address in the directory area. Then the file name can be seen on the display screen of the MENU or by 'Files' command in the BASIC mode.

Meanwhile the 'LOAD' function in MENU can create a 'saved' BA file directly. The 'LOAD' function can read a BASIC program from the cassette, and can 'register' its name in the directory area. So after 'LOAD'ing in Menu, the program name appears on the Menu screen.

ex. The flow diagram of making BASIC program

1. Select BASIC in menu level

2. Type BASIC program

```
10 PRINT 'HELLO'
20 END
```

3. In this point, this BASIC program is called 'non-registered' program.

4. If you return to menu level now, this program is reserved. You cannot find this program in Menu mode in this time. Next time you select BASIC in menu mode, LIST command shows you this program again. This program will be kept unless you do NEW command, LOAD ASCII saved file in RAM or LOAD a file through I/O, cassette and RS-232C.

5. Do 'SAVE' command.

```
SAVE 'TEST'
   or
```

SAVE 'TEST.BA'

(SAVE 'TEST.DO'
or
SAVE 'TEST',A has another meaning.)

6. Then 'non-registered' program becomes a 'registered' program. This program is called 'BA' file simply. And there is nothing in the non-registered program area.

Just after doing SAVE, you can list the program with LIST command. So you might be confused. But don't worry about it. The following illustration will help you to understand not only why LIST command just after SAVE command, can list the 'saved' program, but also why PC-8201A can have many BASIC programs at a time, I hope.

1. You are in MENU mode

```
-----------------
|               |  ^XFFFF
|               |
=================
|               |
-----------------
|               |
| DO files      |
|               |
-----------------  ^XUUUU
|               |
| saved BA file |
|               |
-----------------  <- BOTTOM
                      (^XF9B0)
```

Fig 5.2

2. Select BASIC in MENU and TYPE a BASIC program. LIST shows you the non-registered BASIC program.

```
-----------------
|               |  ^XFFFF
|               |
=================
|               |
-----------------
|               |
| DO files      |
|               |
-----------------  ^XYYYY
|               |
| non-registered|
| BASIC program |
|               |
-----------------  <-- TXTTAB
|               |      (^XUUUU)
| saved BA      |
|   file        |
|               |
-----------------  <-- BOTTOM
```

Fig 5.3

3. Return to menu by MENU command

```
-----------------
|               |  ^XFFFF
|               |
=================
|               |
-----------------
|               |
|  DO files     |
|               |
-----------------  ^XYYYY
|               |
| non-registered|
| BASIC  program|
|               |
-----------------  ^XUUUU
|               |
| Saved BA file |
|               |
-----------------  <-- BOTTOM
```

Fig 5.4

4. Select BASIC again. LIST command lists the non-registered BASIC program which you typed in (2).

```
-----------------
|               |  ^XFFFF
|               |
=================
|               |
-----------------
|               |
| DO files      |
|               |
-----------------  ^XYYYY
|               |
| non-registered|
|  program      |
|               |
-----------------  <- TXTTAB
|               |     (^XUUUU)
| Saved BA files|
|               |
-----------------  <-- BOTTOM
```

Fig 5.5

5. SAVE 'TEST'. TXTTAB still points the program typed in (2). So the same list appears on the screen by LIST.

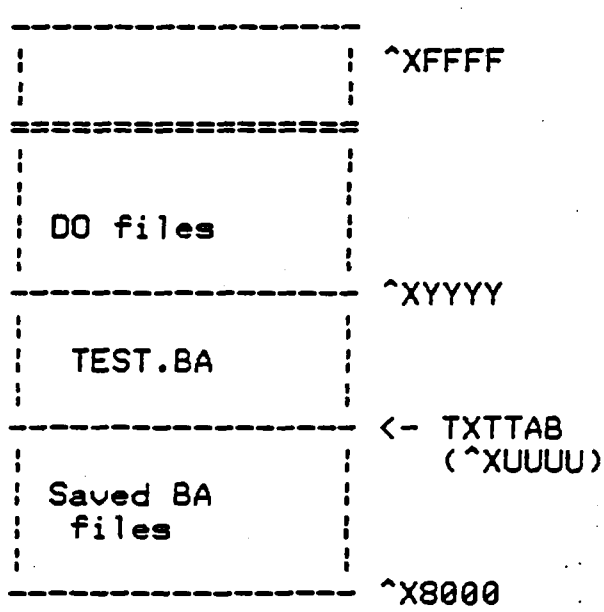


Fig 5.6

6. MENU and Select BASIC again or execute NEW command in BASIC. Now, LIST command lists nothing. Type new BASIC program, again. LIST lists the program that you typed just now.

```
-----------------
|               |  ^XFFFF
|               |
=================
|               |
-----------------
|               |
| DO files      |
|               |
-----------------  ^XZZZZ
|               |
| non-registered|
|  program area |
|               |
-----------------  <- TXTTAB
                      (^XYYYY)
|               |
| TEST.BA       |
|               |
-----------------  ^XUUUU
|               |
| Saved BA files|
|               |
-----------------  <-- BOTTOM
```

Fig 5.7

7.

LOAD 'TEST.BA' in this case, or select 'TEST.BA' directly in MENU. LIST shows you the program, TEST.BA.

```
-----------------
|               |  ^XFFFF
|               |
=================
|               |
-----------------
|               |
| DO files      |
|               |
----------------- ^XZZZZ
|               |
| non-registered|
|   program     |
|               |
----------------- ^XYYYY
|               |
| TEST.BA       |
|               |
----------------- <- TXTTAB
|               |    (^XUUUU)
| saved BA files|
|               |
----------------- <-- BOTTOM
```

Fig 5.8

BASIC interpreter regardes that the current TXTTAB indicates the current BA file. So LIST command lists the program which was saved just now because of specified by TXTTAB.

The BA file can be created in BASIC mode and can be LOADed in BASIC mode and MENU mode. Refer to the PC-8201A user's guide and reference manual. And BA file is executed with BASIC interpreter at the menu level by selecting the BA file directly, as you know. In other words, when you select the BA file name appeared on the MENU, PC-8201A invokes the BASIC interpreter, LOAD that BA file and RUN it automatically.

5.2.3 CO File

The CO file is made in BASIC with BSAVE command or can be loaded and saved from the cassette tape in MENU mode. The CO file is, sometimes, called 'machine language' file. It can be executed directly like a command in menu level,when 'Execute' address was specified in BSAVE and the start address is higher than the second parameter in the latest 'CLEAR' command in BASIC. The default value is ^XF380. So no CO file can be executed directly from the menu level without CLEAR command. The CO files are located above the DO files.

5.2.4 The Order Of The Files In RAM

The order of these files in PC-8201A is fixed.

```
----------------- ^XFFFF
|               |
=================
|  CO files     |
-----------------
|  DO files     |
-----------------
| non-registered|
| BASIC program |
-----------------
| BA files      |
----------------- <- BOTTOM
```

Fig 5.9 the order of the files in RAM

Of course, the size of each file is dynamic.

CHAPTER 6

DIRECTORY STRUCTURE

6.1 DIRECTORY CONFIGURATION PER ENTRY

The directory area is allocated in the middle of the bookkeeping area. The top of the address is F84F in hexadecimal. The directory configuration is shown below.

```
DIRTBL: BASIC    <--------- ^XF84F
        FILER
        TELCOM
NULDIR: (Directory for non-registered program)
SCRDER: (Directory for SCRAP)
EDTDIR: (Directory for EDIT command)
USRDIR: (Directory for user-defined files)
                :
        (( End-of-directory)) ^XFF
```

rf. The non-registered program means non-saved BASIC program. Refer to 'BA file' in the previous section. 'Directory for SCRAP' and 'Directory for EDIT command' are explained in 'DO file'.

Each slot in the directory consists of 11 bytes, 1 byte flag, 2 bytes address and 8 bytes file name. The first 6 slots in directory area are initialized by INIT routine at the COLD START.

Dir slot's configuration per entry

Dir flag	(1 byte)
Adc field	(2 bytes)
File	(8 bytes)

Total 11 bytes.

Bi ignment of Directory flag

Bi Master bit	(1 when directory valid)
Bi ASCII bit	(1 when ASCII-text file)
Bi Binary bit	(1 when Machine-language file)
Bi File-in-ROM	(1 when file is in ROM)
Bi Hidden file	(1 when file is hidden)
Bi	
Bi RAM file open flag	
Bi for internal use (always set to 0 normally)	

V of address-field

B e - Address which TXTTAB must be set to
D e - Beginning address of file
C e - ditto

TXTTAB in BASIC shows the lowest byte of the file, the first link pointer in the BASIC program file. Please refer to her manual to understand what 'link pointer' is, if you wo handle the BASIC programs.

Initialized values for first 6 slots in Directory are shown below. The first 3 files are stored in ROM and displaye the menu screen. (These 3 files are called the 'standarrograms'.) Next 3 files are used for hidden files created AM area. These hidden files will not appear on the Menueen. Refer to previous section, 'DO file' and 'BA file'. characteristics of these hidden files are describeere.

```
rf. First 6 slots in Directory  (Initialized data
stored in ^X6C8E)

DB      ^B1011000
DW      Start address of BASIC
DB      'BASIC  '
DB      0

DB      ^B10110000
DW      Start address of TEXT
DB      'TEXT   '
DB      0

DB      ^B10110000
DW      Start address of TELCOM
DB      'TELCOM '
DB      0

;for non-registered program

DB      ^B10001000
DW      0
DB      0
DB      'XXXXXXX'

;for SCRAP file

DB      ^B11001000
DW      0
DB      0
DB      'YYYYYYY'

;for EDIT command of BASIC

DB      ^B01001000
DW      0
DB      0
DC      'ZZZZZZZ'
```

APTER 7

RGANIZATION

M FILES

apter 2 to understand the whole of the

l are stored with the fixed order. It
les, the BASIC programs which has the
l at the bottom of the RAM area, near
CII files, the suffix is '.DO') are
files. And CO files, the Machine
CO' are saved above the DO files, near
llustration will help you understand

1. There are 5 files in RAM.

```
^XFFFF ----------------
       |Bookkeeping     | ^
       |  Area          | |
       ================ |
       |Free area &     |
       |Data area       |Upper
       |                |
       ----------------
       |                |
       |  MACHIN.CO     |
       |                |
       ----------------
       |                |
       |  DIARY.DO      |
       |                |
       ----------------
       |                |
       |  MEMO.DO       |Lower
       |                |
       ----------------
       |                | |
       |  GRAPH.BA      | |
       |                | V
       ----------------
       |                |
       |  GAME.BA       |
       |                |<- BOTTOM
       ----------------
```

Fig 7.1

2. Add new BASIC file, GOLF.

```
                 |                |
                 ----------------- <---
                 |  MACHIN.CO     |     |
        ^        |                |  Moved
        |        ----------------- up
Upper   |        |  DIARY.DO      |     |
                 |                |     |
                 ----------------- |     |
                 |  MEMO.DO       |     |
                 |                |     |
                 ----------------- <---
                 |  GOLF.BA       |<-- Added here
                 |                |
                 ----------------- <---
Lower            |  GRAPH.BA      |     |
        |        |                |  Not
        |        ----------------- Changed
        V        |  GAME.BA       |     |
                 |                |     |
                 ----------------- <---
```

Fig 7.2

3. Add new ASCII file, ADDRES.

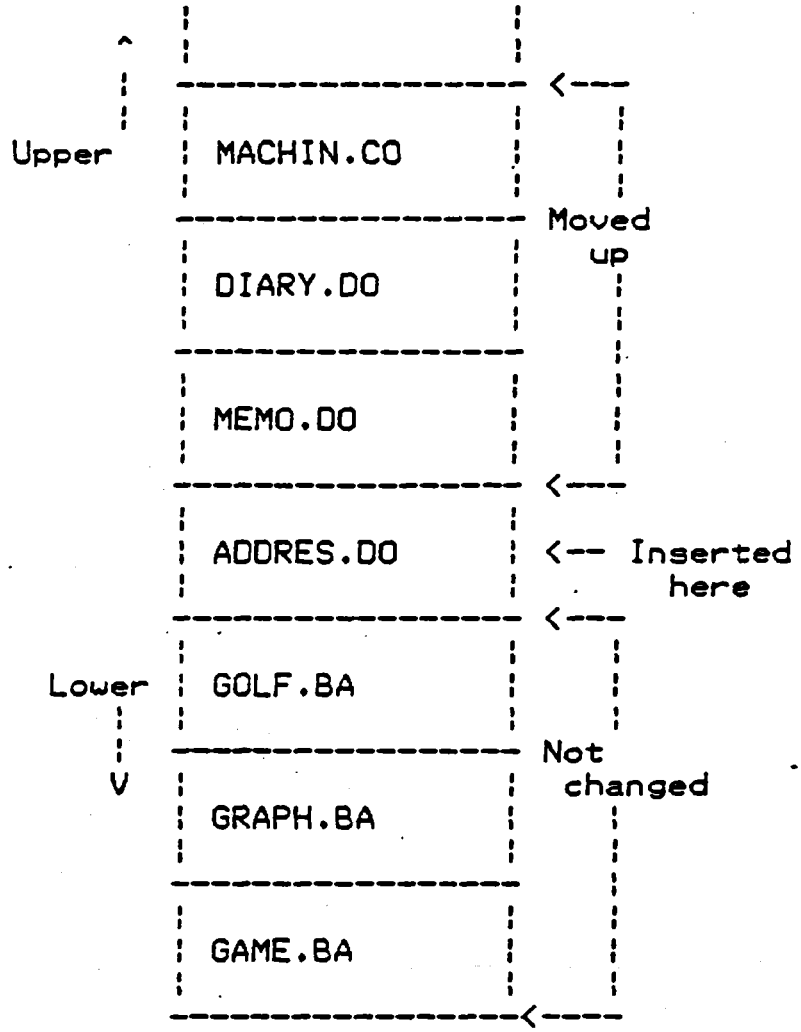


Fig 7.3

Add new CO file, CHAR.CO

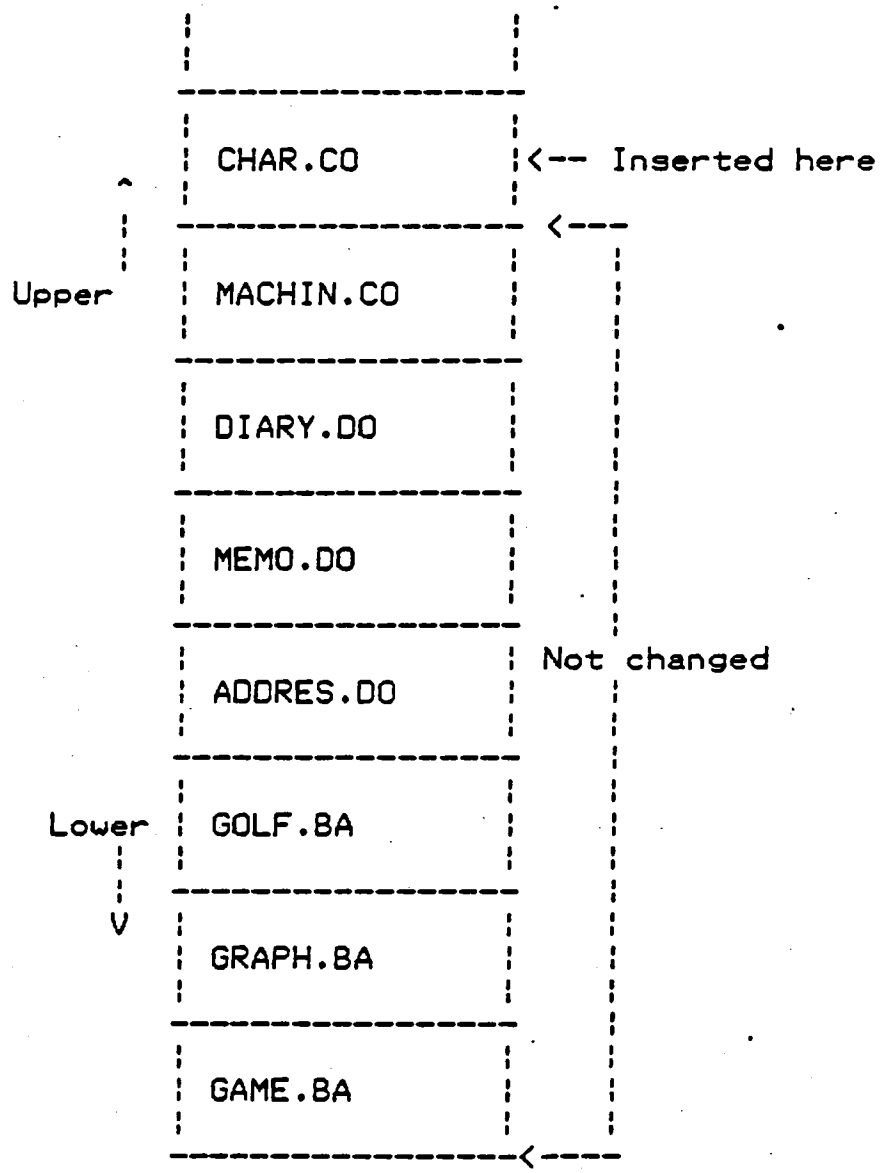


Fig 7.4

A new BA file is created above the old BA files. Otherwise a new DO file is stored below the lowest DO file, just above the BA files. A new CO file is made just ABOVE the CO files. (Just below the address which is pointed by VARTAB. Refer to 'Bookkeeping area'.)

And you know that the non-registered BA file is created between the BA files and DO files, as described in 'BA file' of 'What is RAM files'.

ex.

Non-registered program is created just under the ASCII file.

```
 ¦                ¦
 ----------------
 ¦                ¦ ^
 ¦ ASCII1.DO      ¦ ¦
 ¦                ¦ ¦ Upper
 ----------------
 ¦                ¦
 ¦ non-registered ¦
 ¦ program        ¦
 ¦                ¦
 ---------------- ¦ Lower
 ¦                ¦ ¦
 ¦ BASIC2.BA      ¦ V
 ¦                ¦
 ----------------
 ¦                ¦
```

Fig 7.5 Position of non-registered program

The detail information about the directory configuration is described in 'Directory structure'. The bookkeeping area and the directory area are situated at the top of RAM area.

```
^XFFFF  ----------------- <---
        |               |     |
        |               |     |
^XF977  -----------------     |
        |               |     |
        | Directory area| bookkeeping
        |               |      area
^XF84F  -----------------     |
        |               |     |
        |               |     |
        |               |     |
        |               |     |
^XF380  ----------------- <---
        |               |
```

Fig 7.6 Directory position

7.2 BOOKKEEPING AREA

The book-keeping area is located at the top of the RAM area. The area is divided into 3 parts. The first part, lowest part from ^XF380 to ^XFBBF, includes the pointers and flags for RAM file handling. And many BASIC interpreter's flags, pointers and temporary data area are here. As you know, the directory area is included in this part.

The second part, ^XFBC0 to ^XFE3F, is used for the line buffer of LCD display. BASIC uses this area in the Screen Editor function, also. But the concept of this line buffer is different from the VRAM in the traditional disk top personal computer. Only the character codes are stored in this buffer. There is no attribute data. The attribute data is stored in another table. Refer to the chapter 9, explanation about the LCD driver.

The third part, ^XFE40 to ^XFFFF, is reserved by BIOS. The switches and data storage for RS-232C, Key Board and other I/O drivers are stored here.

```
^XFFFF  -----------------
        |               |
        | Part  III     | BIOS's data
        |               |
^XFE40  -----------------
        |               |
        | Part  II      | LCD buffer
        |               |
^XFBC0  -----------------
        |               | BASIC's data
        | Part I        | File handling data
        |               | Directory
^XF380  -----------------
```

Fig 7.7 Bookkeeping area

7.2.1 Part I (For RAM File Handling And BASIC)

NOTE:

In this section, the articles about the pointers and flags for BASIC are omitted, because this document is written for the programmer who wants to understand the many good features in PC-8201A, in order to utilize this machine with 2nd ROM or user's machine language program. Not written for the people who wants to understand the internal specification of PC-8201A's BASIC interpreter. So I think this document is unfriendly for such kind of people. Please refer to another manuals and textbook if you need understand the BASIC interpreter.

There are many important pointers are stored in this area for RAM file handling. When some of them are mis-handled in your routine, all RAM files might be deleted at next operation of the standard ROM,ROM #0, for instance, power-on or next SAVE command in BASIC. Because the standard programs (BASIC, TEXT and TELCOM) and operating system (represented by Menu), believe that these pointers point the right address. So if a pointer which should point the lowest address of the DO files, points one byte smaller than it should point correctly, TEXT might not invoke any DO files in it. Please understand the purpose of each pointer and make sure that each pointer has a right value any time.

The important pointers for RAM files are listed below.

ADDRESS (Hex)	NAME	SIZE (Decimal)
F380	FSIDSV	2
F384	HIMEM	2
F459	STKTOP	2
F45D	TXTTAB	2
F84F	DIRTBL	33
F870	NULDIR	11
F87B	SCRDIR	11
F886	EDTDIR	11
F891	USRDIR	231
F9B0	BOTTOM	2
FA9A	MEMSIZ	2
FABF	FRETOP	2
FAE1	ASCTAB	2

FAE3	BINTAB	2
FAE5	VARTAB	2
FAE7	ARYTAB	2
FAE9	STREND	2
FB63	FILTAB	2
FB67	NULBUF	2

7.2.1.1 FSIDSV

ADDRESS ^XF380
SIZE 2 bytes

Purpose First power on or not

If this FSIDSV is not identical with FRSTID (^X8A4D), the initialization routine falls into the 'COLD START' routine. In this case, the all data and files in PC-8201A are cleared. The 'COLD START' routine sets FRSTID here after done the initialization. And no one may not change this ID value.

7.2.1.2 HIMEM

ADDRESS ^XF384
SIZE 2 Byte

PURPOSE Highest memory available memory

This pointer keeps the highest memory address available for BASIC. The area between the address in this pointer and ^XF380 is reserved for the machine language file or another user's special working area. No standard program will break the data in this area except POKE statement in BASIC. (The 'POKE' statement can write on anywhere in the RAM which is selected now. So be careful with the address in POKE statement

when you use it for storing your machine language program or character data into RAM area.) The 'HIMEM' can be changed by the second parameter of 'CLEAR' statement in BASIC. Refer to the PC-8201A BASIC reference manual.

7.2.1.3 TXTTAB

ADDRESS ^XF45D
SIZE 2 bytes

PURPOSE Pointer to beginning of current BA file

This pointer is valid in BASIC mode. In another mode, TEXT or TELCOM mode, this pointer keeps the latest value used in BASIC. In BASIC mode, the address of the first link pointer is stored here. And this value won't be changed in BASIC mode unless 'LOAD' command is executed to load another BASIC program, or 'NEW' command. Almost internal routine for BASIC interpreter refers to this pointer to know the top of the current program. And this pointer is very important when a BA file is deleted, too. You cannot kill a BA file in BASIC mode when this TXTTAB points the BA file. Refer to 'How to delete a BA file'.

7.2.1.4 STKTOP

ADDRESS ^XF459
SIZE 2 bytes

PURPOSE Top location to use for the stack

Initially set up by INIT routine in ROM #0 according to memory size to allow for 256 bytes of string space. This value will be changed by a CLEAR command with the first argument. The difference between MEMSIZ and STKTOP means total string space.

The 2 byte space between MEMSIZ and FILTAB is kept for 'VAL' function in BASIC. The 'VAL' function sets '0' at the end of the strings on evaluating the strings. So this 2 bytes area prevent to over-write the FCB area above the FILTAB.

7.2.1.5 DIRTBL

ADDRESS ^XF84F
SIZE 33 bytes

PURPOSE directory for program in ROM

The names and pointers for the programs in ROM are stored here. They are BASIC, TEXT and TELCOM. If you don't want to use these standard programs, you can use this area for your programs. This area will be kept until 'COLD START' is invoked. Refer to 'Directory construction.'

7.2.1.6 NULDIR

ADDRESS ^XF870
SIZE 11 bytes

PURPOSE Directory for non-registered program

This area is kept for internal use. The 'non-registered program' that means the BASIC program, just typed after selecting BASIC, uses this area for pointing the starting address. There is a detail explanation about the 'non-registered' program in the previous section, 'BA file'. And also, refer to 'Directory Construction'.

7.2.1.7 SCRDIR

ADDRESS ^XF87B
SIZE 11 bytes

PURPOSE Directory for SCRAP

The TEXT editor can do 'SELECT', 'CUT', 'COPY' and 'PAST'. This directory is used for this 'temporary file', SCRAP, in TEXT. This file is created when some characters are 'SELECT'ed and 'COPY'ed or 'CUT'. (Refer to PC-8201A user's guide 'SELECT', 'CUT', 'COPY' and 'PAST'.) This file is kept even if you exit from TEXT. And you can use it in another programs, BASIC, TELCOM and so on. If you CUT or COPY without SELECT, the starting address points Control-Z. It means that the SCRAP files is empty. Refer to 'DO file' and 'Directory Construction'.

7.2.1.8 EDTDIR

ADDRESS ^XF886
SIZE 11 bytes

PURPOSE Directory for EDIT in BASIC

The EDIT command in BASIC makes a temporary DO file. This slot is used for this file. Refer to 'DO file' and 'Directory Construction'.

7.2.1.9 USRDIR

ADDRESS ^XF891
SIZE 231 bytes

PURPOSE Directory for user's files (21 slots)

This area is used for BA files, DO files and CO files which user makes. 21 files can be registered

here at most. The end of directory area is indicated by "^XFF", "Directory search stopper". Refer to "Directory Construction".

7.2.1.10 BOTTOM

ADDRESS ^XF9B0
SIZE 2 bytes

PURPOSE Bottom address of RAM

The lowest available RAM address is saved here. You can know how many RAM chips are installed in this RAM bank easily by checking this pointer.

7.2.1.11 MEMSIZ

ADDRESS ^XFA9A
SIZE 2 bytes

PURPOSE Highest location in Memory

This pointer points the top of the string space. The area between the MEMSIZ and FRETOP+1 is called "Used string space", and the area between the FRETOP and STKTOP +1 is "Free string space".

7.2.1.12 FRETOP

ADDRESS ^XFABF
SIZE 2 bytes

PURPOSE Top of the string free space

The highest address (closer to ^XFFFF) of the

string free area is kept in this pointer. The lowest address is kept by STKTOP + 1.

7.2.1.13 ASCTAB

ADDRESS ^XFAE1
SIZE 2 bytes

PURPOSE Pointer to start of ASCII files

This pointer points the first byte of the first DO (ASCII) file.

7.2.1.14 BINTAB

ADDRESS ^XFAE3
SIZE 2 bytes

PURPOSE Pointer to start of COMMAND file

The lowest address of the first CO file is kept here.

7.2.1.15 VARTAB

ADDRESS ^XFAE5
SIZE 2 bytes

PURPOSE Pointer to simple variable space.

This pointer keeps the start address of VARIABLE TABLE area just above the CO files.

7.2.1.16 ARYTAB

ADDRESS ^XFAE7
SIZE 2 bytes

PURPOSE Pointer to beginning of array table

The ARRAY TABLE is allocated just above the VARIABLE TABLE. This points the beginning address of this ARRAY TABLE.

7.2.1.17 STREND

ADDRESS ^XFAE9
SIZE 2 bytes

PURPOSE End of storage in use

This pointer keeps just above the address of ARRAY TABLE. The area between this pointer and the stack pointer can be used as the FREE area.

Note:

When you will use this FREE area, you have to consider about the stack area. As the stack pointer points the current bottom of the stack area, you had better about 120 bytes for the feature stack operation.

7.2.1.18 FILTAB

ADDRESS ^XFB63
SIZE 2 bytes

PURPOSE Point to address of file data

This points to the starting address of the

file data area. The file data area consists of the FCB address. If 'MAXFILES' command in BASIC was not executed after 'COLD START', this table has 4 bytes. The first 2 bytes points the NULL files buffer. (NULBUF points the same address.) The second 2 bytes points the #1 file's FCB address. Refer to the following section about FCB.

7.2.1.19 NULBUF

ADDRESS	^XFB67
SIZE	2 bytes
PURPOSE	Points to address of file #0 buffer

The buffer for file #0 , sometimes called NULBUF, is allocated just above the file data table, pointed by FILTAB.

```
^XFFFF  ----------------
        |              |
        | Bookkeeping  |
        |              |<--- ^XF380
        ----------------
        |              |
        | User's machine|
        |  area        |
        |     or       |
        | Device code  |
        |              |<-- HIMEM
        ----------------
        |              |
        |              |
        | FCB          |
        |  (#1 -- #n)  |
        |              | <-- Address is
        ----------------      stored in FILTAB
        |              |
        | Nul buffer   |
        | (File #0)    |
        |              |<-- NULBUF
        ----------------
        |              |
        | FCB address  |
        |              |<-- FILTAB
        ----------------
        | ( 2 Bytes )  |
        ----------------
        | Used         |<-- MEMSIZ
        | String area  |
        |              |
        ----------------
        | Free         |<-- FRETOP
        |  String area |
        |              |
        ----------------
        |              |<-- STKTOP
        | Stack area   |
        |              |<- Stack Pointer
        ----------------
        |              |
        |  Free area   |
        |              |<-- STREND
        ----------------
        |              |
        | Array data   |
        |              |<-- ARYTAB
        ----------------
        |              |
```

```
|                |
----------------
| Simple         |
|  Variables     |
|                |<-- VARTAB
----------------
|                |
| CO files       |
|                |<-- BINTAB
----------------
|                |
| DO files       |
|                |<-- ASCTAB
----------------
|                |
| BA files       |
|                |<-- TXTTAB
|                |
|                |<-- BOTTOM
----------------
```

'ig 7.8 Pointers and ROM configuration

7.2.2 Part II (VRAM Area For LCD)

ADDRESS	^XFBC0
SIZE	640 bytes
PURPOSE	VRAM

This area is used for the VRAM of LCD (liquid Quristal Display). In this area, the data is stored as the character code. (ANSI character code. Refer to 'APPENDIX A4' in PC-8201A Reference Manual.) The LCD driver, installed just below the LCD panel, gets this character code and displays it on the LCD. The 320 characters (40 by 8) can be shown on the LCD panel at a time. So only second 320 bytes, from ^XFD00 to ^XFE3F, are used for VRAM. The first 320 bytes , from ^XFBD0 to ^XFCFF, are used only when TERM mode is selected in TELCOM. (You can find 'PREV' at the bottom of the screen in TERM mode. The 'PREV' shows you the previous screen in TERM mode. Refer to 'Chapter 8 TELCOM' in PC-8201 User's Guide. The 'PREVIOUS' is the first TERM SUBCOMMANDS.)

The data in VRAM appears when LCD driver is turned on. Refer to Chapter 9 about the control sequence for LCD management.

7.2.3 Part III (Bookkeeping Area For BIOS)

ADDRESS ^XFE40 --- ^XFFFF

This area includes the data area for RS-232C driver, the buffers relevant to Key Board driver and working area for LCD driver. Refer to Chapter 9 - 15 to know how to use the peripheral drivers and the data in this area.

7.2.4 FC Control Block)

Y the FILTAB points the lowest address of the file condata area. It does NOT mean FCB. The FILTAB points th of the starting address of the FCBs, FCB Offset, ifile is opened.

eB and FCB

F(^XFB63) ------> ^XF16A

Dnory (in hexadecimal)

F6E F1 77 F2 .. ..

Tht 2 bytes (^XF16E) points the starting address of FCB of #0 file (NULL buffer). The second 2 b XF277) is the top address of the FCB for the fi. These starting addresses are called FCBOFF (Fset address).

Th area for NUL and file #1 are allocated by the INITIALIZEine in ROM #0. The 2nd and more FCB area will be allocatthe BASIC language, MAXFILES command. Refer to PC-8201rence manual.

The FCB co of 9 bytes parameter area and 256 bytes buffer arcept for NULBUF. NULBUF consists of only 256 bytes buffa. The purpose and the size of the parameters are listew. Since this FCB can support the Floppy Disk file, you ind some meaningless parameters for RAM files. Of cause, n use them for own your purpose if you wish.

(1) FL.MOD

Address: FCBOFF+0
Size: 1 byte

The file mode of the FCB. If this byte is not se is FCB is not used in BASIC. If you obey the BA rule, you have to set non zero value here when yo that file.

1 INPUT only
2 OUTPUT only
8 APPEND only

(2) FL.FCA

ADDRESS: FCBOFF + 1
SIZE: 1 byte

The first cluster allocated to file. In RAM file handling, this parameter has no meaning.

(3) FL.LCA

ADDRESS: FCBOFF + 2
SIZE: 1 byte

The last cluster accessed. For RAM file open, this and next byte is used for the storage of the Directory address of that RAM file.

(4) FL.LSA

ADDRESS FCBOFF + 3
SIZE 1 byte

The last sector accessed. For RAM file open, this and previous byte is used for the storage of the Directory address of that RAM file.

(5) FL.DSK

ADDRESS: FCBOFF + 4
SIZE 1 byte

Disk # of the file or Device ID. The table listed below is the Device ID table in PC-8201A.

Device name	ID number
LCD	^XFF
(CRT	^XFE)
CAS	^XFD
COM	^XFC
(WAND	^XFB)
LPT	^XFA
RAM	^XF9

CRT and WAND is option I/O.

(6) FL.SLB

ADDRESS: FCBOFF + 5
SIZE: 1 byte

Size of last buffer read.

(7) FL.BPS

ADDRESS: FCBOFF + 6
SIZE: 1 byte

The position in buffer for both PRINT and INPUT with the file #. One of the most important parameter in FCB.

(8) FL.FLG

ADDRESS: FCBOFF + 7
SIZE

This byte and next byte are used for the offset address of the RAM file which is opened now. For example, in the 'INPUT' mode file, this offset address is advanced by 256 bytes when the block-read command reads 256 bytes from the file into the buffer in FCB. So in reading or writing to the RAM file (DO file), the starting address and this offset show the next byte should be read or written.

(9) FL.OPS

ADDRESS: FCBOFF + 8
SIZE: 1 byte

High byte of the offset address for RAM file. Refer to FL.FLG.

(10)FL.BUF

ADDRESS: FCBOFF + 9
SIZE: 256 bytes

Buffer for the file.

CHAPTER 8

RAM FILE HANDLING

In this chapter, the technic to manage the RAM file is described. The main purpose is to create or delete a RAM file for the applications stored RAM area or 2nd ROM. As described before, if there is some violation in standard rules of RAM file handling, the file you made (or sometimes all files in the RAM) will be lost by the standard manipulation. (The 'standard manipulation' means the file handling or operation with Menu, BASIC, TEXT or TELCOM in the ROM #0.)

There are many useful routines to make up these violation in standard rules in ROM #0. But using ROM #0 from ROM #1 will reduce the speed of the application. If you want to handle the RAM file without ROM #0, please make sure 'What you should do' in this chapter. And refer to 'Bookkeeping' and 'Directory structure'.

NOTE: The another technical manual for PC-8201A has been available already. There are many information about the RAM file handling routines in ROM #0 in it. For example, 'OPEN RAM FILES', 'KILL ASCII FILE', 'READ A CHARACTER FROM A RAM FILE' and 'CLOSE ALL FILES'. If you will use your application or subroutine with ROM #0, you had better refer to that manual.

8.1 WHAT SHOULD WE DO IN RAM FILE HANDLING

In the 'Directory structure' and 'Bookkeeping area', many rules about the RAM file handling are described. I do explain again about the important rules.

1. Make sure that there is enough free area

When a new file is opened , or new data is appended and inserted, please investigate whether there is enough free bytes in the current RAM bank. Especially, the free area requested in OPEN is sometimes ignored. At least, one byte is necessary for OPEN a DO file. 3 bytes for CO file. Refer to 'What is RAM file' and following sections.

You can find where the free space is in the figure in 'Bookkeeping are'. The difference between the pointer 'STREND' and the value in the stack pointer indicates the free size. But don't forget that some area will be used for the stack operation in that free area. For instance, the make-room routine used in BASIC and TEXT recognizes that the current free space is less 120 bytes than that difference. In other words, 120 bytes is always maintained for the 60 stack area when new data is stored. Refer to 'MAKHOL' in 'Useful Routine For RAM File Handling In ROM #0'.

2. Register file name correctly

The contents of the directory is described in 'Directory construction'. No one forgets to register the file name in it. But someone forgets to set up the directory flag byte and the starting address of the file. If you don't set the directory flag, the file might be deleted by Menu or another operation. If you write a bad starting address in the address field, the link of the directory and the files will be lost. By the result, you cannot select a file properly in the Menu mode or PC-8201A is hung up. Any way, the directory flag and address field have very important meaning. Please refer to the 'Directory construction' and following sections.

3. Maintain the order of the files

In order to maintain the order of the file, we have to do a special trick in setting the starting address of the

new file. For a new DO file, we have to set ASCTAB -1 as the starting address of that new file at the directory area. And for a new BA file, you have to register the ASCTAB -1 in the 'non-registered' file's directory area and insert double NULL code there. That new BA file will be created at ASCTAB -1 and will have the starting address, ASCTAB - 2. In making both of a new DO file and a new BA file, LNKFIL should be executed before end of its process. Refer to 'Useful Routines for RAM file handling in ROM #0' to understand what is LNKFIL.

4. Make and shrink a hole safely

The calculation of the free space is very important. And you have to maintain the stack area when you make a your room. And one more important thing is the management of the pointers. The reason why many programs, Menu, BASIC, TEXT and so on, can use the same RAM area safely is that they adjust the pointers for RAM every time when they change the RAM configuration. For example, BASIC deletes a BASIC program file, he changes many pointers, STREND, ARYTAB, VARTAB, BINTAB and ASCTAB. And he turns off the directory flag in order to indicate that the slot in the directory is not used now. Refer to MAKNOL and MASDEL in 'Useful Routines for RAM file handling in ROM #0.'

5. Insert the promissory byte in the file

When you open a DO file, you have to enter one byte data at least. The data is Control-Z (^X1A), it shows the end of file in RAM. Sometimes this promissory byte is forgotten. So the routine which makes up the starting address in the directory area is confused. Simultaneously BASIC needs 2 NULL bytes at the end of the file. Otherwise CO file requires the 6 bytes file header at the top of the file. Refer to 'What is RAM file'.

6. Make up the starting address in the directory

When you changes the RAM configuration, you have to care not only the pointers but also the starting address in the directory area. It is easy to image that the starting address in the address field of all the DO files should be

changed when you make a new BASIC file. (BASIC file is created under the lowest DO file. Refer to 'Memory Map about RAM files') And when some data are inserted in 'A.DO', a DO file, the starting address of the DO file and CO file located above 'A.DO' should be changed. Refer to 'LNKFIL' in the 'Useful Routines For RAM file Handling in ROM #0'. You can get the know-how to make up the starting address in the directory area.

7. Bad data in DO file

You cannot store the data which include the character whose code is 0, ^X8 and ^X1A. The '0' is used 'NULL' to indicate the hole which is not used. Or double NULL means the end of the BA file. The '^X8' is used 'Back space'. The '^1A' is regarded as the end of the DO file, as you know. Refer to 'DO file'.

8.2 HOW TO MAKE NEW FILE

8.2.1 How To Register The New File Name

At the first, the new file name should be registered in the user's directory area when you create a new file. The user's directory area is started from USRDIR. And the next byte of the user's directory area, the end of the directory area, has ^XFF (255 in decimal). This byte is called 'Directory Stopper'. The used slot starts with the number larger than ^X80 as the directory flag. Therefore it is easy to find the free slot. Refer to the sample program shown later.

You had better compare the new file name with the file name which is existed already. Two files which have same file name sometimes occur a serious problem. So during searching the free slot, the existed file name should be checked. And if there is a same file name, you had better delete it before making new file or abandon to make a new file.

If you succeed to find a free slot in the user's directory area, you have to register the directory flag, the address of the file and the file name. In this time, you have already known the file name. And you can set the directory flag now. (You can get the detail information about the Directory flag in the section, DIRECTORY STRUCTURE.) The address of the file will be fixed later. Because the way to get the address for the new file is depend on the file type, DO file, BA file and CO file. Any way, don't forget to set up the directory flag when you register the new file name. Otherwise someone, Menu, BASIC or TEXT and so on, will destroy your new file without any caution.

Refer to 'Directory construction'.

8.2.2 How To Make DO File

If you have already registered the file name and directory flag at the slot in the directory area, now the only one information lacking in the new directory area is the address of the new DO file. If you didn't read 'How to Register The New File Name' and you have not set the file name

and directory flag yet, please read that section and make up them first.

Usually the new DO file is created just above the ASCTAB, the lowest address of the existed DO files. Refer to the figure in the "What is RAM file" to make sure your image. If you go with the standard rule which Menu, BASIC and others in ROM #0 is used, you can copy the contents of the ASCTAB-1 as the starting address of the new files. Then the registration of the new DO file is done completely. The reason why we have to use ASCTAB-1 instead of ASCTAB is to maintain the order of the files. The LNKFIL, to make up starting address in directory area, searches the file name from top to end and links the starting address of each file. For LNKFIL searches the directory from younger address to older address and older file has younger address, the order of the DO file will be swapped if you use ASCTAB instead of ASCTAB-1. Refer to "LNKFIL" in "Useful Routine for RAM file handling in ROM #0".

But you have to do two more steps for that new DO file. One is to insert the end of file flag at the bottom of that new DO file. Another one is, as you know, to make up the starting address of other files in the directory area.

There is no DO file whose size is zero, because the final character of the DO file should be ^Z (^X1A, 26 in Decimal). In other words, the ^Z indicates the End of File of the DO file. So the DO file will spend one byte at least. If you only want to open the new DO file without any data, you have to insert a ^Z at the starting address. If you want to save some data now, you have to append a ^Z at the end of the data. Never forget to insert a ^Z at the end of the file. Otherwise, next RAM file operation might destroy the all RAM files.

In order to make a room for the new file, a convenient routine is in the ROM #0. Its name is MAKHOL, MAKe HOLe. This routine makes a hole from the specified point and whose size can be decided by the contents in [BC] register. Refer to "MAKHOL" in "Useful Routine For RAM file handling in ROM #0". The concept of the MAKHOL is shown briefly in that section.

If there is no free area in RAM, and you cannot insert a ^Z, you cannot continue to enter data to the file. And, of course, you have to clear the directory flag for next user.

To make up the starting address in the directory area, the routine named LNKFIL is ready in ROM #0. The flow diagram

of that routine is shown in the 'Useful Routine For RAM file handling in ROM #0'. You can get information to make your own LNKFIL routine in it, too.

If you succeed to insert a ^Z and to make up the starting address field in the directory, the opening a new DO file has been done successfully. You can save the data to the new file with using MAKHOL and LNKFIL. Refer to another section to know how to Append, Insert, and Delete data. The sample program in the following section will show you how to make a new file and save data.

Cf. How to make a new DO file

1. Find a free slot in the user's directory. If you cannot find a free slot in the directory area, you have to give up to make a new DO file. Or if you find the same name in the directory, delete that file or abandon to continue.

2. Register the file name and directory flag at the free slot.

3. Get the ASCTAB-1 and save it in the address field of the slot.

4. Try to make a one byte hole at the address where ASCTAB pointed.

5. If you fail to make a hole, clear the directory flag which you registered at (2).

6. If you succeed to make a hole, insert a ^Z at that point.

7. Make up the pointers and starting address in the directory area.

8. That's all. The new DO file has been created without fail.

NOTE: If you make a hole by your own routine, please make sure that the your own routine refines the pointers. Refer to the explanation about the MAKHOL. And refer to 'LNKFIL' to know how to make up the address in Directory.

8.2.3 How To Make A BA File

There is few difference between how to make DO file and How to make BASIC file. There is no difference in the registration of the file name and the directory flag. The first difference is that you have to end the BASIC file with double NULLs (0) instead of ^Z in DO files. In order to understand what double NULLs means, you have to familiar with the function of the LINK POINTER in the Microsoft BASIC. The inner specification of the Microsoft BASIC file is too difficult to described here briefly. You can get some good texts to learn the information about the BASIC programs and their data constructions at the book store or the computer shop. But the basic concept about RAM file handling is exactly same as DO file. (Register the file name and another information at the directory and make a room for the program.)

The second difference is the new BA file is created just above the BA files which has already stored. In other words, the new BA file is inserted just below the lowest DO file. Refer to the section, 'WHAT IS RAM FILE?'.

I believe that the person who wants to handle the BA files, is an expert about the BASIC program and BASIC interpreter. If you are a novice class programmer about the BASIC interpreter, you had better not try to handle the BA file yourself. Please use BASIC mode in ROM #0.

ex. How to create a new BA file in PC-8201A

1. Search a free slot in the user's directory area. If you find a same name in the directory area, delete the file or abandon to continue.

2. Set up the directory flag and copy the file name into the directory.

3. Copy ASCTAB -1 into NULDIR, non-register program's directory area. And make 2 bytes hole and store the double NULL for non-register program.

4. Make a hole as large as possible at the ASCTAB-1.

5. The size of that hole is too small for the new BA file, clear that directory flag written in (2).

6. If you succeed to make a big hole for your BA file, copy the BASIC program into the hole. Don´t forget to insert the double NULLs at the end of the program.

7. Register the starting address at the starting address area in the directory area. Usually, the address that is one byte less than the starting address of the non-registered program is used.

8. Squeeze the hole, when you made a too large hole.

9. Adjust the pointers, ASCTAB, BINTAB, VARTAB, ARYTAB and STREND. Make up the starting address of other files in the directory area. All DO files´ and CO files´ starting address in the directory field should be changed. Refer to LNKFIL.

10. End

8.2.4 How To Make A CO File

The CO file is the another type of the file which you want to make yourself beside the DO file. The difference between DO file and CO file is the heading instruction of the file. The CO file needs the heading data instead of the End of File character, ^Z. So you have to make sure that there are more than 6 bytes besides the size of your machine language program in the free area. And if there is no enough free area, you cannot continue to make a new CO file. If you have already set up the directory flag and file name, clear them soon. Don't leave the illegal flag and file name in the directory.

Heading of CO file

START ADDRESS	2 bytes
LENGTH	2 bytes
EXECUTION ADDRESS	2 bytes

So the file length of CO file can be calculated by LENGTH + 6. In making CO file, don't forget to renew the pointers, VARTAB ,ARYTAB and STREND.

The CO file is usually made just under the address pointed by VARTAB. So the starting address of the other files need not be changed after saving new CO file. But I recommend to do LNKFIL after saving new CO file for safety.

ex. BSAVE "MAC",50000,10,50000 in BASIC mode

Dump the data in CO file is;

^X50 ^XC3 ^X0A ^X00 ^X50 ^XC3 .. .. ..

^XC350	(50000)	Starting address
^X000A	(10)	Length
^XC350	(50000)	Execution address

Cf. The flow of making a new CO file

1. Search the free slot in the directory area. If there is the same file name in the directory, delete that file or abandon to continue.

2. Check the free area. Estimate the free size is greater than your CO file's length + 6 bytes.

3. If there is no room, stop making a new CO file.

4. Make a hole just under address pointed by VARTAB and store the data (or machine language program). Make sure that all pointers are proper. In this time, if you use MAKHOL to make a room, you have to adjust the pointer, BINTAB. Because MAKHOL changes BINTAB always.

5. Register the file name, directory flag and start address at the directory.

6. Adjust VARTAB, ARYTAB and STREND. Make up the starting address of all other files in the directory for safety. If you use LNKFIL for adjustment of the all start addresses in directory, you have to care about the BINTAB as you do in MAKHOL.

7. That's all.

8.3 HOW TO DELETE A FILE

You can guess how to delete a file from the RAM file system in PC-8021 easily. The things that you have to do are to clear the directory flag and to remove the data of the file.

To delete a directory entry, you only turn off the directory flag. If the directory flag is less than ^X80, other programs regardes that slot is not used now.

And when you squeeze the body of the file, you have to check the pointers and the start address of other files in the directory. When you are using the subroutines in ROM #0, these pointers are adjusted automatically. But if you do it by your own routine, you have to care about the pointers. You can find the good clues in 'How to make new file', and 'MAKHOL' in 'Useful Routines for RAM file handling in ROM #0'.

Whether you treat the pointers by your own routine or utilize the MASDEL in ROM #0, you have to make up the starting addresses of the another files. The LNKFIL will do it well. Refer to the following section to know the ENTRY information about the LNKFIL. That section will give you a clue what LNKFIL should do when you will make a LNKFIL by yourself.

8.3.1 How To Delete A DO File

At the first, search the file name which you want to delete in the file. If you don't remember the directory construction, please refer to the chapter 'DIRECTORY CONSTRUCTION', and make sure it. When you find the file name in the directory, check the directory flag of the file. The file which is opened in BASIC, cannot be deleted. If you do it by force, the RAM file system might be crushed or the system might be hung up.

Cf. The flow of deleting a DO file (Calling Machine language program by USR function in BASIC.)

1. Search the file name in the directory

2. Check the directory flag and if the file is opened by BASIC , you cannot delete it.

3. Get the starting address of the file

4. Search ^Z (End of File)

5. Count the size of the file

6. Remove the data of the file and shrink. The ROM routine MASDEL will do it automatically. MASDEL changes the pointers, BINTAB, VARTAB, ARYTAB and STREND automatically.

7. Refine the starting address of other files. LNKFIL will help you.

8. Clear the directory flag of the file which you deleted.

9. That's all

8.3.2 How To Delete A BA File

When you are not in BASIC program, there is few differences between killing DO file and killing BA file. The differences are in searching the end of file. In DO file, ^Z (26 in decimal) indicates the End of file. But in BA file, there is no such a good terminater. The only one way to get the end of the BA file is tracing the 'link pointer' from the beginning of the BA file to end. If you can utilize the ROM #0, you may use the useful routines, CHEAD. The CHEAD searches the end of the BA file. And MASDEL removes the data and refines the pointers. You have to care about the TXTTAB position. If you delete a BA file which is located under the file pointed by TXTTAB, you have to adjust the TXTTAB. This case is occurred when TXTTAB points the second BA file and you delete the first BA file. Finally, you have to do make up the all starting address (link pointers) in directory area. LNKFIL will do it.

NOTE: MASDEL does not change the ASCTAB. When a BA file is killed, ASCTAB should be changed. So after

calling MASDEL, you have to adjust the ASCTAB. Refer to the sample program in the following section. Also "How to make a BA file" will give you a clue.

Another difference is that there is a limitation in deleting a BA file when you are executing that BASIC program. The following caution is available when you make a machine language subroutine for a program written in BASIC. If you won't make a machine language subroutine which handles the BASIC file, you may skip to read this caution.

NOTE: You cannot kill the BA file when you are in it. In other words, when you are running a machine language subroutine with a BASIC program, you may not delete that BASIC program in the subroutine. I'm afraid that this explanation will not make sense for you. So I will show you the short sample.

In the BASIC mode, you can know where you are in by "FILES" command. The file name with "*" is the current file which you are treating. You don't kill it.

1. Select BASIC mode in the menu

2. Type a BASIC program.
```
10 PRINT "HELLO"
```

3. Save it.
```
SAVE "TEST"
```

4. Load it again.
```
LOAD "TEST"
```

5. Try to kill it
```
KILL "TEST.BA" (Return)
?FC Error
Ok
```

6. This result show you what I want to say. BASIC's KILL command checks the current TXTTAB and avoid to kill himself. Your machine language routine should do same check before killing a BASIC file.

NOTE: The comparison between TXTTAB and the starting address of the BA file is available only when you are executing the BASIC program or executing the machine language subroutine in BASIC mode. It is meaningless to care about the TXTTAB and starting address when you are not in BASIC mode.

Refer to 'What is RAM file' and 'Bookkeeping area' to understand the position of the BA files and TXTTAB.

Cf. The flow of the deleting the BA file

1. Search the file name in the directory

2. Check the directory flag and if the file is not BA file, of course, you cannot delete it.

3. Get the starting address of the file in the directory

4. Compare that starting address to TXTTAB. If they are identical, you cannot delete it. If not, you have to remember which is larger, the starting address or TXTTAB.

5. Search End of the File
   CHEAD will help you to find the end of file. Refer to 'Useful Routines for RAM file Handling in ROM #0'.

6. Count the size of the file

7. Remove the data of the file and shrink.
   The ROM routine MASDEL will do it automatically. MASDEL changes the pointers, BINTAB, VARTAB, ARYTAB and STREND. Refer to 'What is RAM file' and 'Bookkeeping area'. And MASDEL returns the negative length in BC register. You can use it to adjust the ASCTAB.

8. Adjust ASCTAB

9. Refine the starting address of other files.
   LNKFIL will help you. Refer to 'Useful Routine For RAM file handling in ROM #0'.

Restore the result of the comparison between the starting address of the file and TXTTAB. If TXTTAB is greater than the starting addres, adjust it.

Clear the directory flag of the file which you deleted.

That's all

To DELETE A CO File

don't have to care about where you are in now like
file or killing DO file. You may delete any CO
nt to delete, even if you are executing that CO
CO file is loaded at the specified area when the
oked in menu mode or in BASIC mode. So the 'CO'
delete the 'CO' file itself, and can save the free

ex. Delete a CO file itself

1. Load a CO file in BASIC or MENU .

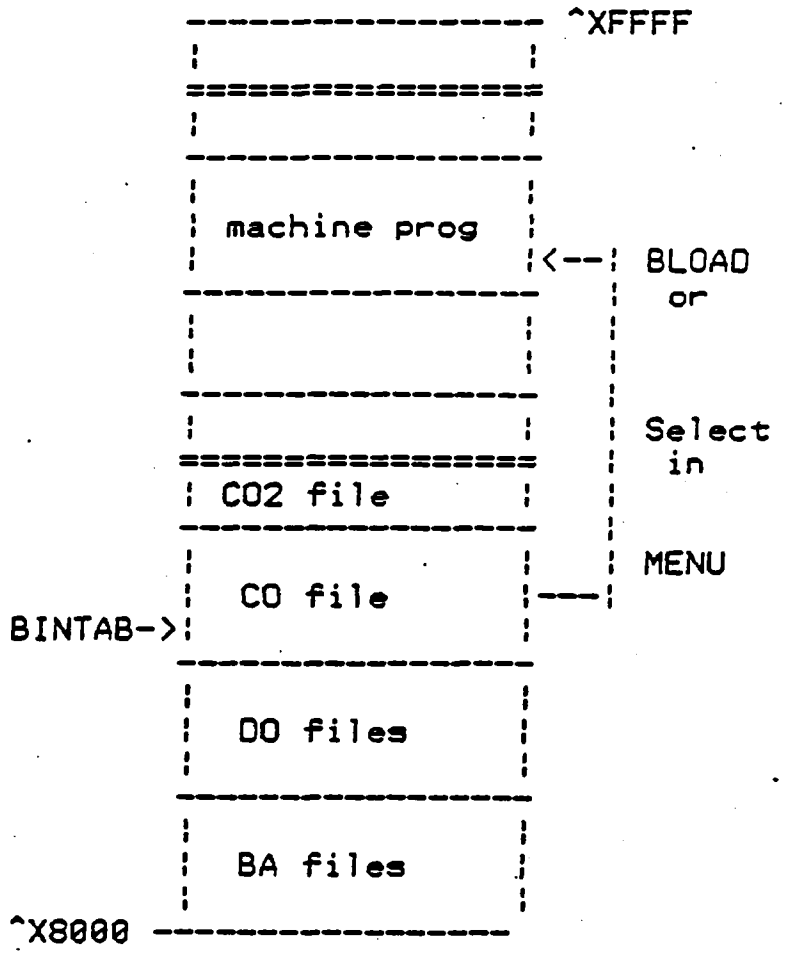


Fig 8.1

:hine prog

```
                 ^XFFFF
     ================
     ================
     ----------------
     ----------------
     | machine prog |  <-- PC
     ----------------
     ----------------
     |              |  <- STREND
     ----------------
     ----------------
     ================
     | CO2 file     |  <- ^XAAAA
     ----------------
     ----------------
     | CO file      |  <- BINTAB
     ----------------
     ----------------
     | DO files     |
     ----------------
     ----------------
     | BA files     |
000  ----------------
```

Fig 8.2

3. Delete the CO file and move the data between the STREND and ^XAAAA.

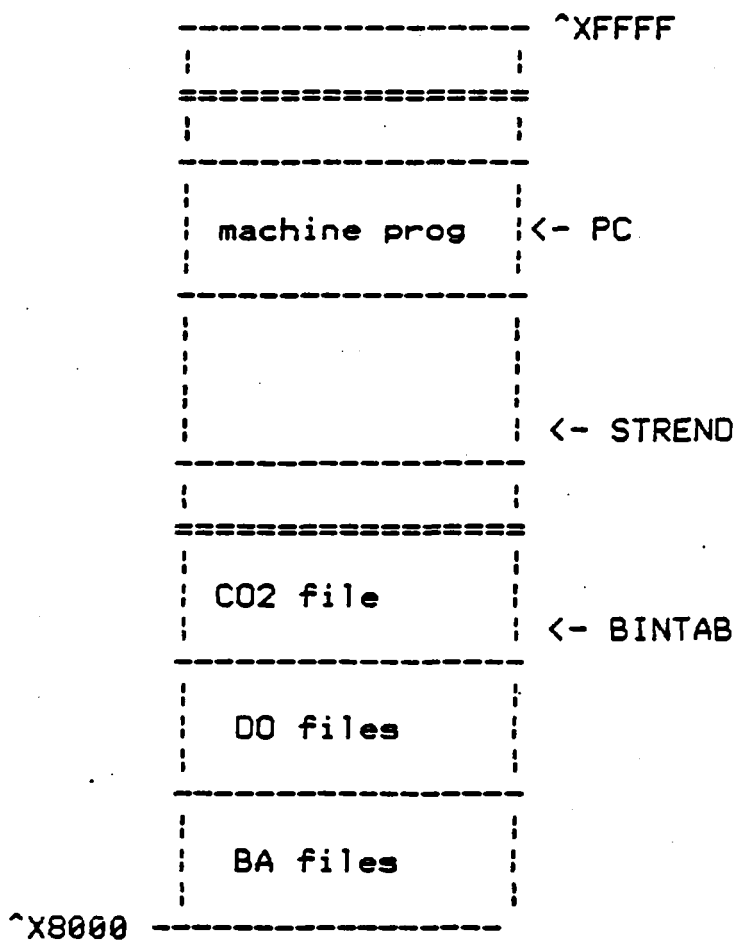


Fig 8.3

NOTE: PC means Program Counter

Unfortunately, you cannot use MASDEL simply for shrinking the hole which is made by killing the CO file, like in deleting a BA file and a DO file. Because MASDEL changes the pointer, BINTAB. (You can understand why BINTAB should not be changed by reviewing the section, 'What is RAM files' and 'Bookkeeping area'.) So if you want to use MASDEL, I do recommend that, you have to save the BINTAB before calling MASDEL and restore it after calling MASDEL.

Cf. The flow of deleting CO file.

1. Search a file name which you want to delete
2. Save the starting address in the directory
3. Calculate the size of that file. The 2nd and 3rd byte in that file show the data length. So the total size of the file is made by adding 6 bytes to the data length. (The 6 bytes includes the starting address, data length and the execution address. Refer to 'What is the RAM file.')
4. Set the starting address and the length for MASDEL
5. Save BINTAB
6. Call MASDEL
7. Recover BINTAB
8. Clear the directory flag of the file
9. That's all

8.4 HOW TO APPEND DATA TO DO FILE

The way to append data to the DO file is very easy. At the first, get the starting address of the DO file in the directory and search the end of file, ^Z. Then, make a room for data you want to store at that point. The routine, MAKHOL, is a best routine to make a room. Refer to 'Useful Routine For RAM file handling in ROM #0'. And don't forget to refine the starting address of other files in the directory area. LNKFIL will help you. Refer to previous chapter, 'How to make a DO file' also.

Cf. APPEND data to the DO file

1. Search the file name in the directory

2. Make sure the file type and status by checking the directory flag.

3. Get the starting address in the directory

4. Search the end of file, ^Z (26 in Decimal)

5. Make a hole just before the ^Z.
   I recommend to use MAKHOL.

6. Store data in the hole

7. Shrink the hole, when the hole you made is too large for the data

   MASDEL in ROM #0 is useful.

8. Refine the starting address in the Directory area.
   LNKFIL will help you.

9. End

There is a sample program of how to APPEND data to DO file in the following section.

8.5 HOW TO INSERT DATA TO DO FILE

When you want to insert some data to the DO file, you can use the know-how which you use to APPEND data to the DO file. The difference is that you have to search the address where you want to insert the data instead of searching the end of file.

Cf. Insert data to DO file

1. Search the file name in the directory

2. Make sure the file type and status by checking the directory flag

3. Get the starting address in the directory

4. Get the address where you want to insert the data

5. Make a hole for the data at the point

   Usually, MAKHOL in ROM #0 is used. MAKHOL changes the pointers, BINTAB, VARTAB, ARYTAB and STREND.

6. Copy data in the hole

7. Shrink the hole, when the hole is too large for the data

   MASDEL in ROM #0 is useful. MASDEL adjusts the pointers, BINTAB, VARTAB, ARYTAB and STREND.

8. Adjust the starting address in the RAM.

   LNKFIL in ROM #0 is useful. Refer to 'Useful Routines for RAM file Handling in ROM #0'.

9. End

8.6 HOW TO DELETE DATA FROM DO FILE

To DELETE data from the DO file is easier than to INSERT data to the DO file. If you will use the ROM #0, the routine named MASDEL delete the data. The MASDEL refines the pointers and LNKFIL adjusts the starting addresses of other file's. You can find the detail information about MASDEL and LNKFIL in 'Useful Routine for RAM file in ROM #0. If you cannot use the ROM #0, you have to renew the pointers, BINTAB, VARTAB, ARYTAB and STREND by YOURSELF. And you must modify the starting addresses in the directory YOURSELF. Refer to the chapter 'Directory construction' and 'Bookkeeping' to under stand the directory structure and pointers. 'MAKHOL' and 'LNKFIL' in 'Useful Routine for RAM file handling in ROM #0' show you how to do it.

8.7 USEFUL ROUTINES FOR RAM FILE HANDLING IN ROM #0

There are several useful routines in ROM #0 for RAM file handling. Indeed that you have to do 'bank-switching' to use these RAM file handling routines from ROM #1. (Refer to Chapter 3.3) But you don't have to worry about the pointers, if you use them. And also, you can save the time to make your own subroutines. I do recommend you to use these RAM file handling routines in ROM #0 for saving time and making applications smoothly.

The presented useful routine in ROM #0.

MAKHOL: Make a room for data entry with changing the pointers

LNKFIL: Make sure the start address in the directory area

MASDEL: Shrink the room made by MAKHOL. This file help you when you made a too large hole.

CHEAD: Search the end of file in BA file.

8.7.1 MAKHOL

Make a hole

ADDRESS ^X6C0A (^O66012, 27658)

ENTRY [HL] points where you want to make a hole

[BC] size of the hole

EXIT [HL] and [BC] are preserved
Carry is set if out of memory

In order to know the free area's size, STREND is the best pointer. The amount of the STREND and your file's size, in this case, should be less than [SP] - 120. (The 'SP' means Stack Pointer, as you know.) The 120 bytes are reserved for Stack's operation. If there is a enough room, MAKHOL shifts the all data between the specified address and STREND. If not, MAKHOL returns with carry set. The flow of MAKHOL is listed at next page.

ex. The flow of MAKHOL. (How to make a room safely.)

```
/------------¥
¦  MAKHOL     ¦
¥------------/
        ¦
-----------------------------------------
¦ STREND + Required bytes                ¦
¦        < SP - Stack area ( 120 bytes ) ¦
-----------------------------------------
        ¦
        ¦   No
        ¦---------> Out of Memory
        ¦
-------------------------------------
¦ Move the data between STREND and   ¦
¦    specified address               ¦
-------------------------------------
        ¦
------------------------------------- **
¦ Change the pointers                ¦
¦  ASCTAB, BINTAB, VARTAB, ARYTAB    ¦
¦    and STREND                      ¦
-------------------------------------
        ¦
  /-----------¥
  ¦  RETURN   ¦
  ¥-----------/
```

Fig 8.4

It is unnecessary to care about the pointers unless you make your own MAKHOL routine. The MAKHOL in ROM #0 manages the pointers automatically. But it does not change the starting address in the directory field. Refer to LNKFIL.

** When you make a hole just above the ASCTAB to create a new DO file, you have to change the pointers, BINTAB, VARTAB and ARYTAB. The ASCTAB is modified only when you make a hole under ASCTAB to register a new BA file.

It is easy to guess that calling MAKHOL too many times will reduce the processing speed. So you had better call the MAKHOL with a good large number in BC register. It makes a good hole which is large enough to save the data you want to keep. The only one thing you have to care of is that you have to shrink the hole when you made a too big hole. The DO file cannot include NUL (0) and ^Z (26) in the file. (The ^Z means the End of File, as you know.) There is a convenient routine to shrink the hole and it refines the pointers, also. Its name is MASDEL and you can get the information about it in the following section.

8.7.2 LNKFIL

Fix up directory structure

ADDRESS ^X233A (^021472, 9018)

ENTRY : NONE

EXIT : NONE
All registers might be altered

This routine fixes up all possible incomplete 'links' between files and their directories. There are many chances in that the link pointers (same as starting address) in the directory fields are not maintained properly. For instance, Making a new DO file will change the starting address of other DO files and CO file. I agree that these link pointers should be modified every time when the RAM organization is modified. But it is also true that such a operation will make a big overhead in RAM file handling. Since you had better make sure when LNKFIL should be called. For instance, when a file is deleted during further file I/O, all link pointers should be fixed up.

Internal flow of LNKFIL

```
/---------------¥
¦  LNKFIL        ¦
¥---------------/
        ¦
-------------------------------
¦ Mark the all valid directory¦
¦ flag (turn 0 bit of all     ¦
¦ valid directory flag )      ¦
-------------------------------
        ¦
-------------------------------
¦ Get the lowest file address ¦
-------------------------------
        ¦
-------------------------------
¦ Get the lowest link pointer ¦
¦  in the valid file´s        ¦
¦   directory                 ¦
-------------------------------
        ¦
--------------------------
¦ Save this link pointer ¦
--------------------------
        ¦<------------------------------¦
---------------------------------       ¦
¦ Search the lowest link pointer¦       ¦
¦ in the marked files in        ¦       ¦
¦ directory area                ¦       ¦
---------------------------------       ¦
        ¦                               ¦
---------------------------------       ¦
¦ Save the saved link pointer   ¦       ¦
¦ at this marked files link     ¦       ¦
¦ pointer field                 ¦       ¦
---------------------------------       ¦
        ¦                               ¦
---------------------------------       ¦
¦ Demark the directory flag of  ¦       ¦
¦ that file. (turn off the bit 0¦       ¦
¦ of that file)                 ¦       ¦
---------------------------------       ¦
        ¦                               ¦
       (A)                             (B)
```

```
          (A)                                  (B)
           ¦                                    ¦
 ---------------------------------              ¦
 ¦ Get next lowest file address ¦               ¦
 ¦ from the bottom of RAM       ¦               ¦
 ---------------------------------              ¦
           ¦                                    ¦
 ---------------------------------              ¦
 ¦ All marked directory flag    ¦               ¦
 ¦ has demaked?                 ¦               ¦
 ---------------------------------              ¦
           ¦       Not End of directory         ¦
           ----------------------------------------
           ¦
           ¦ End of directory
           ¦
 /---------------¥
 ¦     END       ¦
 ¥---------------/
```

Fig 8.5

When the top address of the next file is searched, the pointers, ASCTAB and BINTAB are useful to know what kind of file is searched now.

8.7.3 MASDEL

Delete [BC] bytes from [HL]

ADDRESS : ^X6C3C (^O66074, 27708)

ENTRY: [HL] pointer of the hole should be squeezed
[BC] size of the hole

EXIT: [HL] preserved
[BC] negated

This routine do exactly reverse operation of MAKHOL. The data above the [HL]+[BC] is moved up. And the pointers, BINTAB, VARTAB, ARYTAB are modified. If you use this routine for shrinking a hole of BA file, you can adjust the ASCTAB with the negated [BC] after exit this routine. And also you can adjust the TXTTAB by using this negated BC register if necessary. You have to adjust the TXTTAB when you remove a BA file which is located under the address where is pointed by TXTTAB.

If you want utilize this routine for CO file, you need save BINTAB and recover it after exit. The BINTAB is not modified by killing CO file.

8.7.4 CHEAD

Search for the end of this BASIC program

ADDRESS ^X718 (34300, 1816)

ENTRY : [HL] Top address of that BASIC file

EXIT : [HL] The last address of that BASIC file
All registers and flags are modified possibly

The main purpose of CHEAD is fix links of the BASIC program. In other words, CHEAD goes through program storage and fixes up all the links. The end of each line is found by searching for the zero at the end. The double zero link is used to detect the end of the program. So EXIT [HL] and one will show you the top address of the next file.

8.8 SAMPLE PROGRAM

The sample programs listed here are the exactly "SAMPLE". So some processes are omitted to make explanation clearly. For instance, searching directory to find the good slot for file handling is not described except "How to make a DO file". You know that you have to survey the all directory for checking the same file name and free slot, when you make a new file.

And also, these programs, stored this section are written in plane program technic. You will find another good algorism to handle the RAM files safely and quickly.

8.8.1 Make A New DO File (ASCii File)

```
; Register new DO file in the Directory area
;     OPEN DO file
;
USRDIR          EQU     ^XF891  ;Top address of user's
                        ;directory
EDTDIR          EQU     USRDIR - Directory length
DIRLEN          EQU     11      ;Length of the directory per
file
NAMLEN          EQU     6       ;Length of the file name
ASCTAB          EQU     ^XFAE1  ;Points the lowest address of
                        ;DO files
LNKFIL          EQU     ^X233A ;Make up the address in
                        ;Directory
MAKHOL          EQU     ^X6C0A ;Make a room for file
EOFFIL          EQU     ^1AH    ;End of DO file

OPENDO:
        XRA     A       ;Clear HL
        MOV     H,A     ;
        MOV     L,A     ;
        SHLD    SLTADR  ;Clear slot address
;
        LXI     H,EDTDIR ;Set [top of user directory]
                        ;   - Directory length
SEANAM:
        LXI     B,DIRLEN;Set Directory length
        DAD     B       ;Get next slot
        MOV     A,M     ;Get directory flag
        CPI     ^X80    ;Valid?
        JC      NONVAL  ;Jump if not valid slot
        INR     A       ;End of directory area?
        JZ      ENDSEA  ;Jump if end of test
;
; Is the file DO file?
;
        DCR     A       ;Adjust directory flag
        MOV     D,A     ;copy flag for later use
        ANI     ^B01000000
                        ;Pick up ASCII flag
        ORA     A       ;DO file?
        JZ      SEANAM  ;Jump if not DO file
;
; Compare the name
;
        PUSH    H       ;Save the slot address
        INX     H
        INX     H       ;Advance to name field in
```

```
                                ; directory
        XCHG                    ;[DE] name address
        LXI     H,NAME          ;name of the file which
                                ; we want to make
        MVI     B,NAMLEN;Set name length
CMPNAM:
        LDAX    D               ;Get directory's name
        CPI     M               ;Compare with our file
        JNZ     NOTSAM          ;Jump if not same
        INX     H               ;Advance the pointers
        INX     D
        DCR     B
        JNZ     CMPNAM          ;compare next
;
; Same file name is found
;
        POP     H               ;Top of the slot address
        MOV     A,M             ;Get directory flag
        ANI     ^B00000010
                                ;Pick up OPEN BIT
        ORA     A               ;File already opened?
        JNZ     FILAOP          ;Jump if file already opened
;
; Find same name and not opened file
;
        SHLD    SLTADR          ;Save it
        CALL    DELFIL          ;Delete this file
        JMP     FINDNM          ;go to Registration
;
; Find free slot
;
        XCHG                    ;[DE] free slot address
        LHLD    SLTADR          ;Get free slot address
                                ; that has been found
        MOV     A,H             ;
        ORA     L               ;Never found?
        JNZ     EVERFN          ;jump if already found
        XCHG                    ;This is the first time
        SHLD    SLTADR
        JMP     SEANAM          ;Check next slot
;
EVERFN:
        XCHG                    ;Don't renew the address
        JMP     SEANAM          ;
;
; To search the directory is done
;
        LHLD    SLTADR          ;Is there good free slot?
        MOV     A,H
        ORA     L               ;
```

```
        JZ      DIRFULL ;Jump if directory full
;
        PUSH    H       ;Save the top of the slot
        MVI     M,^B11000000
                        ;Set directory flag as DO
                        ; file
        INX     H       ;Advance to name field
        INX     H
        LXI     D,NAME  ;Top of our file name
        MVI     B,NAMLEN;Name length
CPYNAM:
        LDAX    D       ;get our file name
        MOV     M,A     ;copy it in directory
        INX     H       ;
        INX     D
        DCR     B       ;Continue to end of name
        JNZ     CPYNAM
;
        LHLD    ASCTAB  ;Get lowest address for DO
                        ; files
        LXI     B,1     ;Make one byte hole
        CALL    MAKHOL  ;Dig
        JC      MEMFUL  ;Jump if out of memory
        MVI     M,EOFFIL;Set end of file marker
        DCX     H       ;Lowest address - 1
                        ; for maintain the file order
        POP     D       ;Recover Top of that slot
        INX     D       ;Advance to address field
        MOV     A,L     ;set start address
        STAX    D
        INX     D
        MOV     A,H
        STAX    D
;
; Make up starting address of other files in
;    directory area
;
        CALL    LNKFIL

        RET
;
; External routines
;
DELFIL:
        ; Delete the specified file

FILAOP:
        ; Error handling --- File already opened

MEMFUL:
```

```
        ; Error handling --- Memory full

DIRFUL:
        ; Error handling --- Directory full
                                    :
;
; DATA AREA
;
NAME: DB        'TEST  DO'

        END
```

8.8.2 Save Data Into DO File

```
;
; Save data into DO file
;
; ENTRY: [HL] points directory of the file
;        [DE] address of source data
;        [BC] length of data
;
MAKHOL  EQU     ^X6C0A  ;Make a room for data
LNKFIL  EQU     ^X233A  ;Make up starting address

ENDFIL  EQU     ^X1A    ;End of DO file

;
;
SAVDAT:

;
; Check the directory flag of the file
;
        MOV     A,M     ;Get directory flag
        PUSH    B       ;Save data length
        MOV     B,A     ;Save directory flag
        ANI     ^B11000000
                        ;Pick up mode bits
        CPI     ^B11000000
                        ;DO file?
        JNZ     BADFIL  ;Jump if not DO file
        MOV     A,B     ;Get flag again
        ANI     ^B00000010
                        ;Pick up OPEN bit
        ORA     A       ;File already opened?
        JNZ     FILAOP  ;Jump if file already opened
        MOV     A,B     ;Get directory flag
        ORI     00000010B
                        ;Say this file is opened
        MOV     M,A
;
;Search end of file
;
        POP     B       ;Recover DATA length
        PUSH    H       ;Save Top of directory address
        PUSH    B       ;Save DATA length
        INX     H       ;Advance to Address field
        MOV     A,M     ;get address in [HL]
        INX     H
```

```
        MOV     H,M
        MOV     L,A     ;Set top of the file
;
SEALOP:
        MOV     A,M     ;Get Data
        CPI     ENDFIL  ;End of file?
        JZ      FNDEOF  ;Jump if end of file
        INX     H
        JMP     SEALOP  ;Search next
;
;MAKE A ROOM FOR DATA
;
        POP     B       ;Recover data lengtH
        PUSH    D       ;Save source address
        CALL    MAKHOL  ;Dig a hole for data
        JC      MEMFUL  ;jump if error detected
        POP     D       ;Recover source address
;
;copy data in to the hole
;
COPYLP:
        LDAX    D       ;Get source data
        MOV     M,A     ;save it into file
        INX     H
        INX     D
        DCX     B       ;Decrement DATA length
        MOV     A,B     ;
        ORA     C       ;End of data?
        JNZ     COPYLP  ;Continue till end of data

;
;Make up starting address of other files in
;  directory area
;
        CALL    LNKFIL
;
;Turn off the opened bit in directory flag
;
        POP     H       ;Recover directory address
        MOV     A,M     ;Get directory flag
        ANI     ^B11111101
                        ;Turn off the flag
        MOV     M,A     ;Renew the flag
        RET
;
;External routines
;
BADFIL:
        ; Bad file mode
```

```
FILAOP:
        ; File already opened
MEMFUL:
        ; Memory full error.
        END
```

8.8.3 DELETE SOME DATA FROM DO FILE

```
;
; Delete some data from DO file
;
; ENTRY : [HL] Top of the directory address
;         [DE] Offset address of Top data
;                should be deleted
;         [BC] Length of data should be deleted
;
MASDEL  EQU     ^X6C3C  ;Remove some data
LNKFIL  EQU     ^X233A  ;Make up starting address
;

DELDAT:

;
;Check directory flag
;
        MOV     A,M     ;Get directory flag of
                        ; the file
        ANI     ^B11000000
                        ;Pick up VALID bit and ASCII
                        ; bit
        CPI     ^B11000000
                        ;Valid DO file?
        JNZ     BADFIL  ;Jump if bad file
        MOV     A,M     ;Get directory flag again
        ANI     ^B00000010
                        ;Pick up OPEN bit
        ORA     A       ;Already opened?
        JNZ     FILAOP  ;jump if the file already opened
        MOV     A,M     ;Set opened bit
        ORI     ^B00000010
        MOV     M,A     ;Say, the file is opened
;
        PUSH    H       ;Save directory address
        INX     H       ;Get start address of the file
        MOV     A,M     ;
        INX     H
        MOV     H,M     ;
        MOV     L,A     ;[HL] start address of the file
;
        DAD     D       ;Absolute address of the data
                        ; which should be removed
; Delete data
;  [HL] TOP of the data, [BC] data length
;
```

```
        CALL    MASDEL  ;Remove the data from file
;
; Turn off the OPENED bit
;
        POP     H       ;Restore the directory address
        MOV     A,M     ;Get directory flag
        ANI     ^B11111101
                        ;Turn off
        MOV     M,A
;
;Adjust the directory

        CALL    LNKFIL  ;Make up all start address in the
                        ; directory flag
;
        RET

;
; External routine
;
BADFIL:
        ;Bad file mode -- Error

FILAOP:
        ;File already opened -- Error
```

8.8.4 DELETE DO FILE

```
;
; Delete DO file
;
; ENTRY: [HL] points the directory of the file
;
MASDEL  EQU     ^X6C3C  ;remove data
LNKFIL  EQU     ^X233A  ;adjust address field in
                        ; directory area

DELDO:
        MOV     A,M     ;Get directory flag
        ANI     ^B11000000
                        ;Pick up VALID and ASCII bit
        CPI     ^B11000000
                        ;Valid do file
        JNZ     BADFIL  ;jump if bad file mode
        MOV     A,M     ;get directory flag
        ANI     ^B00000010
                        ;pick up opened bit
        ORA     A       ;Already opened?
        JNZ     FILAOP  ;jump if already opened

;
; Calculate the size of the file
;
        PUSH    H       ;save directory address
        INX     H       ;get start address
        MOV     A,M     ;
        INX     H
        MOV     H,M     ;[HL] start address
        MOV     L,A
;
        PUSH    H       ;Save start address
SEALOP:
        MOV     A,M     ;end of file?
        CPI     EOFFIL  ;
        INX     H       ;next field
        JNZ     SEALOP  ;continue till EOF
;
        POP     D       ;Restore start address
        MOV     A,L     ;[HL]-[DE]= length
        SUB     E
        MOV     C,A
        MOV     A,H     ;
        SBB     D
        MOV     B,A     ;Set length in [BC]
```

```
        XCHG                ;[HL] start address
;
        CALL    MASDEL      ;Remove the data
;
        POP     H           ;recover directory address
        XRA     A
        MOV     M,A         ;clear it
;
;Make up all start address in directory
;
        CALL    LNKFIL
;
        RET
;
;External routine
;
FILAOP:
        ;File already opened error

BADFIL:
        ;Bad file mode error
;

        END
```

8.8.5 DELETE BA FILE

```
;
;Delete BASIC file
;
; Assume that this subroutine is used with BASIC
;   main program
;
; ENTRY: [HL] directory address of the file
;

MASDEL  EQU     ^X6C3C  ; remove data from file
LNKFIL  EQU     ^X233A  ; make up starting address

CHEAD   EQU     ^X0718 ; search end of BASIC file

TXTTAB  EQU     ^XF45D  ; lowest address of current
                        ; BASIC program
ASCTAB  EQU     ^XFAE1  ; Lowest address of DO files

;
;
DELBAS:
        MOV     A,M     ;Get directory flag
        CPI     ^B10000000
                        ;BASIC file?
        JNZ     BADFIL  ;Jump if not BASIC
                        ; file
        XCHG            ;[DE] directory address
        LHLD    TXTTAB  ;get lowest address of the
                        ;current BASIC program
                        ; (We are executing the
                        ; BASIC program with this
                        ; machine subroutine.)
        XCHG            ;[DE] TXTTAB [HL] Directory
                        ;                 address
        PUSH    H       ;save directory address
        INX     H       ;advance to address field
        MOV     A,M     ;get start address of BA file
                        ; which we want to delete
        INX     H
        MOV     H,M
        MOV     L,A     ;[HL] start address
        MOV     A,H     ;compare to TXTTAB
        SUB     D
        JNZ     NOSAM   ;jump if not same
        MOV     A,L     ;compare lower address
        SUB     E       ;
        JNZ     NOSAM   ;jump if not same
```

```
        JMP     FCERR   ;you cannot kill your mother
                        ;BASIC
NOSAM:
        XCHG            ;save start address
        POP     H       ;recover directory address
        PUSH    PSW     ;save result of comparison
        XRA     A       ;[A]=0
        MOV     M,A     ;clear directory flag
        PUSH    D       ;save start address
;
;[DE] start address of the BA file
;
        CALL    CHEAD   ;search the end of BA file
        INX     H       ;adjust for calculation the length
        POP     D       ;recover start address
        PUSH    D       ;Save start address again
        MOV     A,L     ;Calculate the length
        SUB     E       ;
        MOV     C,A     ;Set length in [BC]
        MOV     A,H
        SBB     D
        POP     H       ;recover start address
;
;Remove body of the file
;
        CALL    MASDEL  ;return negative length in [BC]
;
        LHLD    ASCTAB  ;adjust ASCTAB because MASDEL
                        ;doesn't change it
        DAD     B
        SHLD    ASCTAB
;
        PUSH    B       ;save this value for later use
;
;Adjust starting address in directory
;
        CALL    LNKFIL  ;
;

        POP     B       ;Restore adjustment value
        POP     PSW     ;recall result of comparison
                        ; TXTTAB and start address
        RNC             ;Return if TXTTAB is smaller
                        ; than start address
        LHLD    TXTTAB  ;Adjust TXTTAB because we
                        ; delete BA file under TXTTAB
        DAD     B
        SHLD    TXTTAB
        RET
;
```

```
; EXTERNAL ROUTINE
;
FCERR:
        ; Illegal function call error
BADFIL:
        ; Bad file mode error
        END
```

8.8.6 MAKE NEW CO FILE

```
;
; MAKE NEW CO FILE
;
; ENTRY: [STRADR] start address of CO file data
;        [LENGTH] length of data
;        [EXECAD] execution address
;
;        [HL] directory address for this CO file
;
MAKHOL  EQU     ^X6C0A  ;make a room·
LNKFIL  EQU     ^X233A  ;make up directory address field

HEADLN  EQU     6       ;Header length of CO file

BINTAB  EQU     ^XFAE3  ;lowest address of existed CO
                        ; files
VARTAB  EQU     ^XFAE5  ;lowest address of Variable
                        ; table


MAKECO:
;
; Refer HOW TO MAKE NEW DO FILE to know how to find
; the directory address for new files.
;

        MVI     A,^B10100000
                        ;Set directory flag as CO file
        MOV     M,A     ;register it
        PUSH    H       ;save directory address
        LHLD    LENGTH  ;get file length of new CO
        LXI     B,HEADLN;Set header length
        DAD     B       ;Get total length of new CO file
        MOV     B,H     ;Set length in [BC]
        MOV     C,L     ;
        LHLD    BINTAB  ;[HL] lowest address of existed
                        ;CO files
        PUSH    H       ;Save current BINTAB
        LHLD    VARTAB  ;[HL] just above highest CO file
        CALL    MAKHOL  ;Try to make a hole
        JC      MEMFUL  ;jump if there is no enough room
        XCHG            ;Save the top address of hole
        POP     H       ;recover BINTAB
        SHLD    BINTAB  ;Adjust BINTAB
        XCHG            ;restore TOP of hole
```

```
        POP     D           ;[DE] directory address
        INX     D           ;advance to address field
        MOV     A,L         ;Set start address
        STAX    D
        INX     D
        MOV     A,H         ;
        STAX    D
;
; To register the file name in directory is omitted.
;
        XCHG                ;[DE] top of the vacant room
        MVI     B,HEADLN;Set header length
        LXI     H,STARAD;offset of header data
COPYHD:
        MOV     A,M         ;Get header data
        STAX    D           ;store it in file
        INX     D
        INX     H
        DCR     B           ;end of header data?
        JNZ     COPYHD      ;copy 3 address as header
;
        LHLD    LENGTH      ;Get data length
        MOV     B,H         ;set length in [BC]
        MOV     C,L
        LHLD    STARAD      ;[DE] destination address
                            ;[HL] source address
COPYLP:
        MOV     A,M         ;copy contents of file
        STAX    D
        INX     D
        INX     H
        DCX     B           ;count down
        MOV     A,B         ;end of data?
        ORA     C
        JNZ     COPYLP      ;continue till end of data
;
        CALL    LNKFIL      ;make up all start address of
                            ;other files in directory area
        RET
;
; ERROR HANDLING ROUTINE
;
MEMFUL:
;       memory full error

;
; DATA AREA
;
STARAD: DS      2
LENGTH: DS      2
```

```
EXECAD: DS      2

        END
```

8.8.7 DELETE A CO FILE

```
;
; DELETE A CO FILE
;
; ENTRY : [HL] addres of its directory
;
MASDEL  EQU     ^X6C3C  ;remove data
LNKFIL  EQU     ^X233A  ;make up starting address
                        ;in the directory

BINTAB  EQU     ^XFAE3  ;lowest address of CO files
HEADLN  EQU     6       ;length of the header in CO
                        ; file

DELCO:
        MOV     A,M     ;Get DIRECTORY flag
        CPI     ^B10100000
                        ;CO file?
        JNZ     BADFIL  ;Jump if BAD file mode
        XRA     A       ;
        MOV     M,A     ;Clear directory flag
        INX     H       ;Advance to address field
        MOV     A,M     ;Get start address of the CO
                        ;file
        INX     H
        MOV     H,M     ;[HL] start address
        MOV     L,A
        PUSH    H       ;save start address
        INX     H       ;Get file length in the
                        ; header
        INX     H
        MOV     C,M     ;get length in [BC]
        INX     H       ;
        MOV     B,M
        LXI     H,HEADLN;add header length
        DAD     B       ;
        MOV     B,H     ;Set total length in [BC]
        MOV     C,L
        POP     H       ;recover start address
        XCHG            ;save it at once
        LHLD    BINTAB  ;get lowest address of existed
                        ;CO files
        PUSH    H       ;save it for after adjustment
        XCHG            ;[HL] start address
                        ;[BC] file length
        CALL    MASDEL  ;remove the body of the file
        POP     H       ;recover BINTAB
        SHLD    BINTAB  ;adjust BINTAB
```

```
;
        CALL    LNKFIL  ;make up starting address in
                        ;the directory area
        RET

; EXTERNAL ERROR ROUTINE
;
BADFIL:
        ;Bad file mode

        END
```

CHAPTER 9

LCD INTERFACE

This chapter describes how to control LCD (Liquid Crystal Display) of PC-8201A.

9.1 OVER VIEW

The LCD (LR-202C), full bit map screen which consists of 240 * 64 dots, displays 40 characters per line and 8 lines per screen. A character on the LCD consists of 6 by 8 pixels. The LCD is driven by 10 Segment Drivers (HD44102B) with 200 bytes Display RAM and 2 Common Drivers (HD44023b). Segment Drivers are selected by Port A/B of PPI (81C55).

9.2 CONSTRUCTION OF LCD

The LCD is divided into the following IC blocks. Each block has its own Segment Driver with 200 bytes Display RAM. And each IC block can display 50 * 32 dots. However, B5 and B10 displays only 40 * 32 dots. Of course, you can write dots on the remaining area of Display RAM of B5 and B10 with no error, but they will never appear on the screen.

```
        |<------- 240 dots ------------------->|
     -----+-----------------------------------+
     |    |    |    |    |    |    |    |    |
     |    | B1 | B2 | B3 | B4 | B5 |    |
     |    |    |    |    |    |    |    |
64 dots +-----------------------------------+
     |    |    |    |    |    |    |    |    |
     |    | B6 | B7 | B8 | B9 | B10|    |
     |    |    |    |    |    |    |    |
     -----+-----------------------------------+
```

Fig 9.1

The Display RAM may be regarded as the VRAM in the traditional desk top type personal computer. Setting a Bit On/Off in the Display RAM means setting/resetting a dot on the LCD.

Refer to following sections how to control each Segment Driver.

9.3 I/O PORT RELATED TO LCD

9.3.1 BLOCK SELECT --- PPI 81C55 PORT A/B

```
msb 7   6   5   4   3   2   1   0 lsb
  +---+---+---+---+---+---+---+---+
  |PA7|PA6|PA5|PA4|PA3|PA2|PA1|PA1|        OUT ^XB9
  +-------------------------------+
  | X | X | X | X | X | X |PB1|PB0|        OUT ^XBA
  +-------------------------------+
```

PA0 to PB7 is associated to BLOCK1 to BLOCK8
PB0,PB1 to BLOCK9,10 respectively.

0 = Not Select / 1 = Select

Description:

Selecting a LCD Block (same meaning as selecting a Segment Driver IC) which you want to access. You cannot select two blocks at a time.

9.3.2 LCD COMMAND SET

There are 5 commands to control the Segment Driver IC. These commands are executed via Port ^XFE.

9.3.2.1 Display ON/OFF.

```
msb 7   6   5   4   3   2   1   0   lsb
  +---+---+---+---+---+---+---+-------+
  | 0 | 0 | 1 | 1 | 1 | 0 | 0 | DISP |     OUT ^XFE
  +---+---+---+---+---+---+---+-------+
```

DISP: Display ON/OFF
0 = Display Off
1 = Display On

Description:

DISP decides whether the data in Display RAM is displayed on the screen.
This port doesn't effect the contents of Display RAM.

9.3.2.2 Set Address Counter

```
msb 7   6   5   4   3   2   1   0 lsb
  +---+---+---+---+---+---+---+---+
  |PG1|PG0|OF5|OF4|OF3|OF2|OF1|OF0|        OUT ^XFE
  +---+---+---+---+---+---+---+---+
```

Select PAGE

PG1	PG0	
0	1	-- PAGE0
0	1	-- PAGE1
1	0	-- PAGE2
1	1	-- PAGE3

OFn means 'OFfset counter' in each PAGE.
It must be from 0 to 49.

The Display RAM is divided into 4(0 to 3) pages and each page contains 50 bytes (0 to 49) as shown at next page. Segment driver has PAGE counter and OFFSET Counter. These counter is set by this command. The OFFSET counter works as the loop counter, it's value from 0 to 49. The OFFSET counter is automatically Incremented/Decremented after read/write operation. The counter mode is described blow. Page counter is not changed by read/write operation.

```
                    OFFSET counter
PAGE      0<-------------------->49(39 if B5/B10)
counter  +---+--------------------+
         |lsb|                    |
^B00     |   |    PAGE 0          |
         |msb|                    |
         +---+--------------------+
         |lsb                     |
^B01     |        PAGE 1          |
         |msb                     |
         +------------------------+
         |lsb                     |
^B10     |        PAGE 2          |
         |msb                     |
         +------------------------+
         |msb                     |
^B11     |        PAGE 3          |
         |lsb                     |
         +------------------------+
```

Fig 9.2

9.3.2.3 Set Starting Page.

```
msb 7    6    5   4   3   2   1   0 lsb
  +----+----+---+---+---+---+---+---+
  |SPG1|SPG2| 1 | 1 | 1 | 1 | 1 | 1 |      OUT ^XFE
  +----+----+---+---+---+---+---+---+
```

SPG1/0: Specify the Starting Page to be display on LCD.

SPG1	SPG0		Order of Display Page
0	0	----	0 -> 1 -> 2 -> 3
0	1	----	1 -> 2 -> 3 -> 0
1	0	----	2 -> 3 -> 0 -> 1
1	1	----	3 -> 0 -> 1 -> 2

Description:

Assume that each LCD block is divided into 4 pages corresponding with the Display RAM. The combination with the Page of LCD Block and Display RAM page can changed. The 'SET STARTING PAGE' defines the mapping between the Page in Display RAM and the Page of LCD Block.

Ex.

Assume that Starting page is set to 2. Then mapping between Display RAM and LCD PAGE becomes as shown as follows.

```
LCD BLOCK

Upper   +-----------------------+
        |                       |
        | PAGE2 in Display RAM  |
        |  is displayed here    |
        |                       |
        +-----------------------+
        |                       |
        | PAGE3 in Display RAM  |
        |  is displayed here    |
        |                       |
        +-----------------------+
        |                       |
        | PAGE0 in Display RAM  |
        |  is displayed here    |
        |                       |
        +-----------------------+
        |                       |
        | PAGE1 in Display RAM  |
        |  is displayed here    |
        |                       |
Lower   +-----------------------+
```

Fig 9.3

9.3.2.4 Select Address Counter Mode

```
msb 7   6   5   4   3   2   1   0
  +---+---+---+---+---+---+---+---+
  | 0 | 0 | 1 | 1 | 0 | 0 | 1 |U/D|   OUT ^XFE
  +---+---+---+---+---+---+---+---+

      U/D(Up/Down count) --- 0 Up Count
                             1 Down count
```

Description:

Set OFFSET Counter Mode.

9.3.3 Read Status --- Read The Status Of Segment Driver.

```
msb 7      6       5       4    3 - 0  lsb
  +----+-------+------+-----+-----+
  |BUSY|UP/DOWN|ON/OFF|RESET|XXXX |  IN ^XFE
  +----+-------+------+-----+-----+

  RESET    -----    Status of the RST pin
     0              Normal
     1              RST is low level
                    (BUSY must be 1)

  ON/OFF   -----    Display ON/OFF
     0              Display OFF
     1              Display ON

  UP/DOWN  -----    Mode of Address counter
     0              Down counter
     1              Up   counter

  BUSY     -----
     0              Normal
     1              Operating Command or
                    Writing/Reading a data.
```

9.3.4 Write/Read Display Data

```
+--+--+--+--+--+--+--+--+
|D7|D6|D5|D4|D3|D2|D1|D0|    IN/OUT ^XFF
+--+--+--+--+--+--+--+--+
```

Description:

Read the data from the Display RAM that is pointed by PAGE and OFFSET counter. If you want read some portion of the Display RAM, use this command after Setting the PAGE counter and OFFSET counter by 'Set Address Counter' command and 'Set Page Counter' command described before. Note that one dummy read must be done before using this command in order to get a correct data.

9.4 SOFTWARE FOR LCD

This section describes not only how to handle the LCD without reading the routines stored in ROM #0 about LCD, but also how to maintain the book-keeping area for LCD in the RAM.

9.4.1 How To Initialize The LCD.

What should be done in initialization is following.

1) Set up Address counter. Usually Page 0, Offset 0.
2) Set up Offset Counter Mode.
3) Set up Starting Page.
4) Select Display ON/OFF.

The tiny program shown blow initializes LCD's all Segment Drivers as below.

PAGE COUNTER = 0
OFFSET COUNTER = 0
UP COUNTER MODE
STARTION PAGE = 0
DISPLAY ON

Note:

Whenever the power is turned on, LCD is initialized by the reset pulse of the hard wear. At that time, Display is turned OFF, Offset Counter is set to count up mode. Another status is not determined.

The ROM #0 always reinitializes LCD as Display ON, Starting Page = 0 and Offset counter count up mode when a character is displayed.

9.4.1.1 Sample Program For LCD Initialization.

```
;
; Initialize Segment driver.
;


;--- Equaters ---

PORTA   EQU     ^X0B9
PORTB   EQU     ^X0BA
LCDCOM  EQU     ^X0FE
LCDSTAT EQU     ^X0FE


LCDINIT:
        DI                      ; Inhibit disturbance for Port
A/B
        CALL    SELALL          ; Select all Segment Driver.

        CALL    LCDBUSY         ; Wait until LCD become Ready.
        XRA     A
        OUT     LCDCOM          ; Reset Address Counter.

        CALL    LCDBUSY
        MVI     A,^X3B          ; Offset counter Up mode.
        OUT     LCDCOM          ;

        CALL    LCDBUSY
        MVI     A,^X3E          ; Set starting PAGE=0
        OUT     LCDCOM

        CALL    LCDBUSY
        MVI     A,^X39          ; Display ON.
        OUT     LCDCOM          ;


LCDBUSY:
; Wait until LCD become Ready.
        IN      LCDSTAT         ; Get LCD status.
        RLC                     ; Move MSB to CF.
        JC      LCDBUSY         ; Wait if LCD is busy.
        RET


SELALL:
; Select all Segment Drivers
```

```
MVI     A,^XFF          ;
OUT     PORTA           ; B9h
IN      PORTB           ; Get current status.
ORI     03              ; Select block 9,10.
OUT     PORTB
RET

END
```

9.4.2 How To Write A Character.

Writing a character on the LCD is performed by writing some Bit patterns in the Display RAM of Segment Driver.

Basic sequence of writing a character on the LCD is as follows.

1. Select LCD Block(Segment Driver) which you want to PUT a character.
2. Set the Offset counter mode.(Usually Up mode)
3. Set the Address where 1st byte should be written.
4. Write the Bit pattern.
5. Set Starting PAGE counter
6. Insure Display ON.

rf. Next sample program.

9.4.2.1 Sample Program Of Writing A Character On The LCD.

This Sample program shows how to write a character on the LCD. This routine updates the pointers which is used by System ROM, ROM #0, to maintain the system circumstance.

```
; Sample program to write a character on LCD.
; This program performs same function as the following BASIC
; program.
;
; 10 LOCATE 0,0
; 20 PRINT "A"
; 30 END

CSRY     EQU     ^XF3E5          ; Cursor Y position
                                 ;     (1 to 8)
CSRX     EQU     ^XF3E6          ; Cursor X position
                                 ;     (1 to 40)
LCTEY    EQU     ^XFEB9          ; Character Y Position
                                 ;     (0 to 7)
LCTEX    EQU     ^XFEBA          ; Character X Position
                                 ;     (0 to 39)
PORTA    EQU     ^XB9            ; Segment Driver Select
                                 ; Port.
PORTB    EQU     ^XBA            ; ditto
LCDCOM   EQU     ^XFE            ; LCD command Port.
LCDSTAT  EQU     ^XFE            ; LCD Status Port.
LCDIO    EQU     ^XFF            ; LCD data I/O Port.
         ORG     ^XF000          ; 61440D

LOCATE:
; LOCATE 0,0
         LXI     H,^X0101        ; To set cursor position
                                 ;  (0,0)
         SHLD    CSRY            ;
         LXI     H,^X0000
         SHLD    LCTEY

PREP:
;-- Select Block 1 to write (1,1)
         DI                      ; Inhibit disturbance for
                                 ; Port A/B of 81C55.
                                 ; You need not do DI as
                                 ; far as no one
```

```
                                        ; changes the data port of
                                        ; 81C55. You have to consider
                                        ; other INT routines.

        MVI     A,^X01                  ; Select Block 1
        OUT     PORTA                   ;
        IN      PORTB                   ; Get current status.
        ANI     ^B11111100              ; Deselect Block 9/10.
        OUT     PORTB                   ;

        CALL    LCDBUSY                 ; Wait until LCD become ready.
        MVI     A,0                     ; Page 0,offset 0.
        OUT     LCDCOM                  ;

        CALL    LCDBUSY                 ;
        MVI     A,^B00110010            ; Offset counter Up mode.
        OUT     LCDCOM

CHROUT:
        LXI     H,FONTA                 ; Get start Address of Font A.
        MVI     C,^X06                  ; Set Font size.

WRITE:
;
; Write data to Display RAM of LCD
;
; ENTRY:  [HL] = Font start address.
;         [C]  = Length of Font.

        CALL    LCDBUSY                 ; Wait until LCD become Ready.
        MOV     A,M                     ; Get font Pattern to send.
        OUT     LCDIO                   ; Write to Display RAM of LCD.
        INX     H                       ; Up date PTR.
        DCR     C                       ; Bump Counter.
        JNZ     WRITE                   ; To send next pattern.
                                        ; Offset counter is Auto
                                        ; increment Mode, so we don't
                                        ; care about OFFSET counter.
        LXI     H,CSRX                  ; Up date Cursor PTR.
        INR     M                       ; No check for end of line in
                                        ; this program.
        LXI     H,LCTEY
        INR     M

;---- Set starting page --------

        MVI     A,^X0FF ; Select all Block.
        OUT     PORTA
        IN      PORTB
```

```
        ORI     ^B00000011
        OUT     PORTB

        CALL    LCDBUSY         ; Wait until LCD become Ready.
        MVI     A,^X3F          ; Starting page 0.
        OUT     LCDCOM          ;

        MVI     A,^X00111001    ; Insure display ON.
        OUT     LCDCOM
        EI
        RET

LCDBUSY:
        IN      LCDSTAT         ; Get LCD status.
        RLC                     ; Move msb to CF.
        JC      LCDBUSY
        RET

FONTA:  DB      ^X3C,^X12,^X11  ; Font data for 'A'
        DB      ^X12,^X3C,^X00

        END
```

9.4.3 How To Set/reset A Dot On The LCD.

The Sample program shown blow explains how to set/reset a dot on the LCD. It does same function as the following BASIC program.

```
100 CLS
110 FOR Y=9 TO 22
120   FOR X=60 TO 80
130    PSET(X,Y)
140   NEXT X
150 NEXT Y
160 '
170 FOR Y=14 TO 18
180   FOR X=64 TO 76
190     PRESET(X,Y)
200   NEXT X
210 NEXT Y
220 END
```

9.4.3.1 Sample Program For SET/RESET Dot.

```
;
; Sample program for SET/RESET a Dot.
;

PORTA   EQU     ^XB9            ; LCD block select.
PORTB   EQU     ^XBA            ; //
LCDCOM  EQU     ^XFE            ; LCD command.
LCDSTAT EQU     LCDCOM          ; LCD status.
LCDIO   EQU     ^XFF            ; LCD data I/O.


PSET:
        DI                      ; Disable all interrupt
                                ; to keep correct block
select.
        XRA     A               ; To set SET flag.
        STA     SR              ; Set/Reset Flag.
```

```
        LXI     B,^X140E        ; [B]=20 X count,[C]=14 Y
count.
        LXI     H,^X0A09        ; [H]=X Position,[L]= Y
Position.
PSET1:
        PUSH    H               ; Save (X,Y) Position.
        PUSH    B               ; Save X,Y count.
        CALL    MAIN
        POP     B               ; Restore X,Y count.
        POP     H               ; Restore X,Y position
        INR     L               ; Advance Y position.
        DCR     C               ; Bump Y counter.
        JNZ     PSET1

PRESET:
        MVI     A,^XFF          ; To set SR Flag.
        STA     SR              ; Set Unplot Flag.
        LXI     B,^X0C06        ; [B]=12,[C]=06
        LXI     H,^X0E0D        ; ([H],[L])=(14,13)
PRESET1:
        PUSH    H               ; Save X,Y Position.
        PUSH    B               ; Save X,Y counter.
        CALL    MAIN
        POP     B               ; Restore X,Y counter.
        POP     H               ; Restore X,Y position.
        INR     L               ; Advance Y position.
        DCR     C               ; Bump Y counter.
        JNZ     PRESET1
        RET

MAIN:
;
;       [H] = X position
;       [L] = Y Position
;       [B] = X count
;       [C] = Y count
        PUSH    H               ; Save X.Y Position.
        CALL    DOT             ; Plot/Unplot a dot at (X,Y)
        POP     H               ; Retrieve Position.
        INR     H               ; Advance X POSITION.
        DCR     B               ; Bump X counter.
        JNZ     MAIN            ;
        RET

DOT:
        CALL    LMAIN
        LDA     SR              ; Get SR flag.
        ORA     A               ; See if set/reset?
        JNZ     RESET           ; Branch if Reset.
        MOV     A,E             ; Get MASK pattern.
```

```
        ORA     D               ; [A] = data to write.
        JMP     DISP

RESET:
        MOV     A,E             ; Get Mask Pattern.
        XRI     ^XFF            ; Reverse MSK pattern.
        ANA     D               ; [A] = data to write.

DISP:
        MOV     D,A
        CALL    WRITE
        DI
        MVI     A,^XFF          ; Select all Block.
        OUT     PORTA
        IN      PORTB
        ORI     ^B00000011
        OUT     PORTB
        CALL    LCDBUSY         ; See if Lcd Busy.
        MVI     A,^B00111111    ; Starting Page 0
        OUT     LCDCOM
        CALL    LCDBUSY
        MVI     A,^B00111001    ; Display ON.
        OUT     LCDCOM
        EI
        RET


LMAIN:
; ENTRY: [H] = X position in Block-1
;        [L] = Y Position in Block-1
; Reg:

        PUSH    H               ; Save X,Y position.
        PUSH    H
        CALL    SEL2            ; Select Block-2.
        CALL    SETADR          ; Set Address of Display RAM.
        CALL    READ            ; Read the LCD.
        POP     H               ; Retrieve X,Y position.
        CALL    GETMSK          ; Get Mask Pattern.
        POP     H               ; Retrieve (X,Y) Position
        CALL    SETADR
        RET

WRITE:
; Func: Output [ODAT] to LCD.
;
; Reg:  A and Flags.

        CALL    LCDBUSY
        MOV     A,D             ; Get Data to write.
```

```
        OUT     LCDIO
        NOP                         ; Must be EI at final.
        RET

READ:
;
; Entry: Non
;
; Exit:  [D] = Current Data in Display RAM.
;
; Reg:  A,D and Flags.

        CALL    LCDBUSY             ; Wait until LCD become Ready.
        IN      LCDIO               ; Dummy Read.You must do this
to
                                    ; get correct data.
        CALL    LCDBUSY             ;
        IN      LCDIO               ; Get Valid Data.
        MOV     D,A                 ; Save it.
        RET

GETMSK:
; Entry: [L] = Y Position
;
; Exit:  [E] = Mask Pattern.
;
; Reg:  A,L,E and Flags.

        MOV     A,L                 ; Get Y position.
        ANI     ^B00000111          ;
        MOV     L,A                 ; Set counter.
        MVI     A,^B80
MSK1:
        RLC
        DCR     L                   ; Bump counter.
        JP      MSK1                ; Branch if not finished.
        MOV     E,A                 ; Save Mask pattern.
        RET

SETADR:
;
; ENTRY: [H] = X Position on Block-2
;        [L] = Y Position on Block-2
;
; FUNC: Set Address
;
; Register:
;       A,H,L and Flags.

        MOV     A,L                 ; Get Y position.
```

```
        RAL                     ; Move Bit4/3 to Bit7/6.
        RAL
        RAL
        ANI     ^B11000000      ; Get page.
        ORA     H               ; [A] = Page and OFFSET.
        MOV     L,A             ; Save it.
        CALL    LCDBUSY         ; Wait until LCD become Ready.
        MOV     A,L             ; Retrieve Address.
        OUT     LCDCOM
        RET

LCDBUSY:
; Entry: Non
;
; Func: Wait until LCD become Ready.
;
; Exit: Non
;
; Reg:  A and Flags.
;

        IN      LCDSTAT         ; Get LCD status.
        RLC                     ; Set Busy FLG to CF.
        JC      LCDBUSY         ; Wait if LCD is BUSY.
        RET

SEL2:
; Select Block-2
;
; Reg:  A and Flags.
;

        DI
        MVI     A,^B00000010    ; Select Block-2
        OUT     PORTA
        IN      PORTB
        ANI     ^B11111100
        OUT     PORTB
        RET


SR:      DB     00              ; Set/Reset flag.
                                ; 0=set/FF=reset.

        END
```

9.4.4 How To Define A Character

This section describes how to define the User Definable characters in PC-8201A. And how to store them in a portion of RAM where ROM #0 can use this your new Fonts. In this section, BASIC command will be used to do some operation.

9.4.4.1 Structure Of Character And How To Define It.

One character consists of 6 * 8 dots. Vertical 8 dots is handled by a byte. So in order to define a character, you must define Sequential 6 bytes of data. The data ^X3C, ^X12, ^X11, ^X12, ^X3C, ^X00 define 'A' as follows.

```
        DB      <^X3C,^X12,^X11,^X12,^X3C,^X3C,^X00>
                ; CG pattern for 'A'
```

DATA Pattern

	0	1	2	3	4	5
lsb 0	0	0	1	0	0	0
1	0	1	0	1	0	0
2	1	0	0	0	1	0
3	1	0	0	0	1	0
4	1	1	1	1	1	
5	1	0	0	0	1	0
6	1	0	0	0	1	0
7 msb	0	0	0	0	0	0

Font pattern

	0	1	2	3	4	5
			*			
		*		*		
	*				*	
	*				*	
	*	*	*	*	*	
	*				*	
	*				*	

^X3C ^X12 ^X11 ^X12 ^X3C ^X00

Fig 9.4

9.4.5 How To Store The Your Own CG

This section explains how to store USER CG in to RAM which also can be used by ROM #0.

Assume that you have to define Fonts as described in the previous section. Each Font consists of 6 bytes. Font Data has been BSAVEed in the RAM file named 'FONT.CO', whose start address is ^XYYZZ.

You can make 'FONT.CO' in the following sequence.

1. Reserve area for 'FONT.CO' by CLEAR command in BASIC.

```
CLEAR <length>, <startaddress>
```

2. Load 'FONT.CO' into RAM

```
BLOAD 'FONT'
```

3. Register the top address of the CG.

```
POKE ^D65216,<Start Address (High byte)>
POKE ^D65215,<Start Address (Low byte)>
```

After this sequence, ROM #0, for instance, BASIC, can use the new Defined CG.

9.5 AVAILABLE SYSTEM WORK AREA

This section explains how to use the system Character Generator and how to use the available System work area.

9.5.1 How To Use The CG In System ROM.

You might want to use the CG of ROM #0 instead of making new CG by yourself. In such a case, this Section will help you.

The Character Generator of characters whose code is from ^X20 to ^X7E , are stored in the highest portion of the ROM #0, from ^X78B7 to ^X7B37. Each Character consists of 5 bytes. The sample program shown blow explains how to get the character pattern and how to expand it into the standard shape, 6 * 8 pixels. Assume that this program is written to be stored as the CO File in the RAM files and will be executed with ROM #0.

```
;
; ENTRY [A] = character Code (^X20 to ^X7E)
;
EXPAND:
        SUI     A,^X20          ;
        MOV     C,A             ;
        ADD     C               ; *2
        ADD     A               ; *4
        ADD     C
        MOV     C,A             ; [C] offset from base of CG.
        MVI     B,^X00          ;
        LXI     H,CGADR
        DAD     B
        LXI     B,TEMP
        MVI     D,^X5           ; Set font data length.
NEXT:
        MOV     A,M             ; Get Font data.
        STAX    B
        INX     H
        INX     B
        DCR     D
```

```
        JNZ     NEXT
        ORA     A
        STA     TEMP+5
        RET
```

9.5.2 VRAM AREA IN SYSTEM Work Area

The area from XFBC0 to XFE3F in the RAM, is reserved for VRAM area of the LCD. It is divided into 2 portions. Each portion can be hold the character codes displayed on the LCD at a time. So the each portion has 320 bytes. The attribute data is not saved in this area. Only the character code is stored.

```
1st     ^XFBC0-^XFCFF    ; Keep previous Page
                         ; in TELCOM.
2nd     ^XFD00-^XFE3F    ; Current Displayed
                         ; character is Saved.
```

The character code of the character displayed at the location (1,1) on the LCD display is stored at ^XFD00, and the code of the character at (2,1) is stored at ^XFBC1 , and so on. So the code of the left-lowest character, (40,8) is stored at ^XFC3F. This rule is used in the standard program in ROM #0. For instance, BASIC, TEXT and TELCOM use that area like a VRAM in the traditional disk top personal computer. The menu screen also utilize that area. But You can use this area as you like. The data in this area does not effect the information on the LCD display, as far as you use your own display routine.

9.5.3 Reverse The Attribute Of The Specified Area.

ROM #0 has the Reverse Attribute Table in Work Area.

The attribute data is kept in the area from ^XFA60 to ^XFA87. Each bit represents the each character Box on LCD. (Therefore only 40 bytes can be handle the attribute of whole LCD screen.) When the bit is off (0), it shows that the character Box is displayed in normal mode. And the bit is turned on, 1, that character Box is displayed in Reverse mode. The relation between the Attribute bit and Character Box is shown blow. The relation of the reverse attribute bit and each character box is as follows.

```
+-----------------------------------------------+
|( 1,1)|( 2,1)|( 3,1)|          |(39,1)|(40,1)|
+-----------------------------------------------+
|( 1,2)|( 2,2)|( 3,2)|          |(39,1)|(40,1)|
+-----------------------------------------------+
|                                               |
|                                               |
|                                               |
|                                               |
+-----------------------------------------------+
|( 1,8)|( 2,8)|( 3,8)|          |(39,8)|(40,8)|
+-----------------------------------------------+
```

```
^XFA60  Bit0    -- (01,1)
        Bit1    -- (02,1)
        Bit2    -- (03,1)
        Bit3    -- (04,1)
        Bit4    -- (05,1)
        Bit5    -- (06,1)
        Bit6    -- (07,1)
        Bit7    -- (08,1)
^XFA61  Bit0    -- (09,1)
        Bit1    -- (10,1)
         |          |
^XFA87  Bit0    -- (33,8)
        Bit1    -- (34,8)
        Bit2    -- (35,8)
        Bit3    -- (36,8)
        Bit4    -- (37,8)
        Bit5    -- (38,8)
        Bit6    -- (39,8)
        Bit7    -- (40,8)
```

CHAPTER 10

KEYBOARD INTERFACE

10.1 THE KEYBOARD MATRIX

The Keyboard matrix of PC-8201A is as follows.

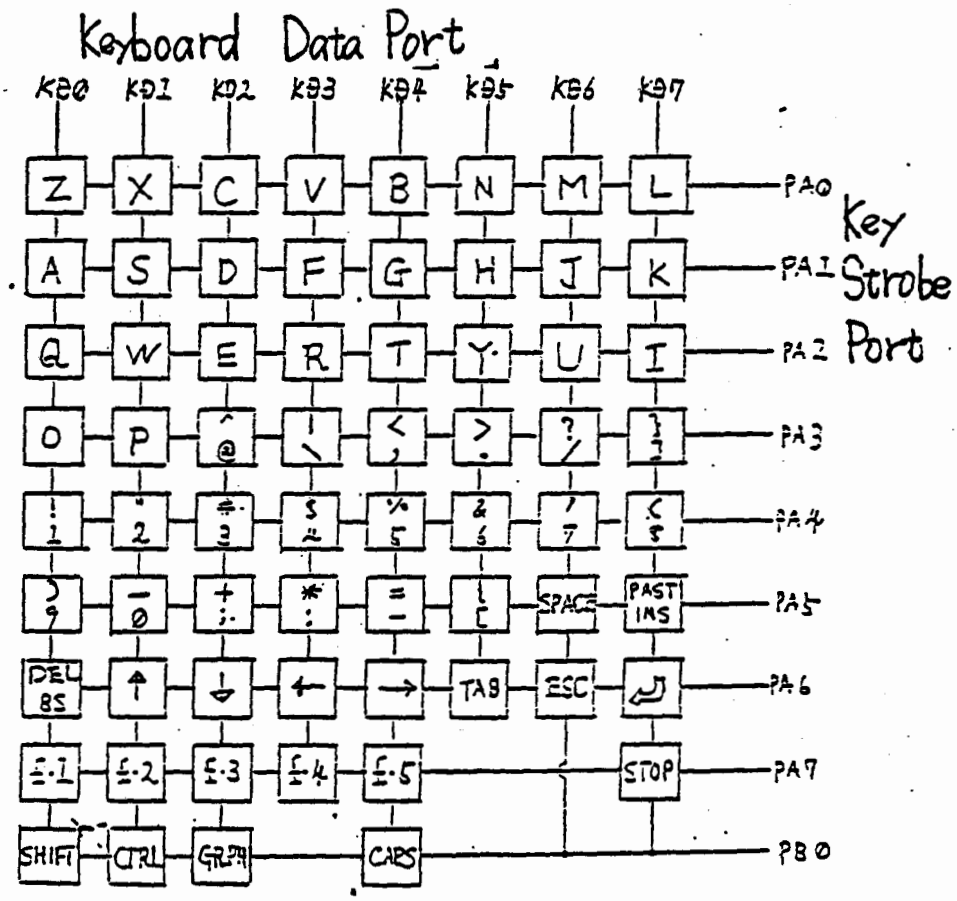


Fig 10.1

The abbreviation PAn (PA7, PA6, ..., PA0) and PBn means the bit of PORT A and B of 81C55. Please refer to the following sections about I/O ports. And also, KDn (KD7, KD6 .., KD0) represents the bit of the KEYIN, Input port for the Keyboard.

Note: '/' means (SHIFTED CODE) / (UNSHIFTED CODE)

10.1.1 I/O Port For Keyboard

10.1.2 KEYBOARD STROBE ----- PART A/B Of 81C55

```
msb 7   6   5   4   3   2   1   0 lsb
  +---+---+---+---+---+---+---+---+
  |KS7|KS6|KS5|KS4|KS3|KS2|KS1|KS0|   OUT ^XB9
  +-------------------------------+
  | X | X | X | X | X | X | X |KS8|   OUT ^XBA
  +-------------------------------+
```

KS8 ... KS0 KEYBOARD Strobe
0 = Strobe OFF
1 = Strobe ON

10.1.3 KEYIN ----- Read Keyboard Data

```
msb  7    6    5    4    3    2    1    0 lsb
     +----+----+----+----+----+----+----+----+
     |KD7|KD6|KD5|KD4|KD3|KD2|KD1|KD0|   IN ^XEB
     +----+----+----+----+----+----+----+----+

     KD7 ... KD0 ---- Keyboard data
             0 = Depressed
             1 = Not depressed
```

Read the strobed column of the keyboard. Please refer to KEY MATRIX shown before to understand the relation between KDn and Key on the key board.

10.1.4 Keyboard Scanning

Key scan must be performed by software. It can be done by the interrupt, RST 7.5. The RST 7.5 Pin of 80C55 is connected to the TP Pin(No.10) of calendar clock (uPD1990). So that interrupt occurs every 4 msec in the standard system.

10.2 SOFT WARE FOR KEYBOARD OPERATION.

10.2.1 How To Read The Keyboard

Basic Keyboard read sequence is as follows.

1. Turn on the strobe pulse to the desired column you want to read.
2. Read the column from KYIN port.
3. Strobe off

The following Sample program shows how to read the Keyboard in detail.

10.2.1.1 Sample Program Reading Keyboard.

Following Sample program read the every column and save the data into the KYBUF(Keyboard Buffer).

```
;
; Read CURRENT KEY BOARD STATUS.
;
; Note: Make sure Keyboard strobe is
;       not disturbed while reading the key board.
;       You have to care of the other interrupts.
;
; Equator

PORTA  EQU      ^XB9              ; Keyboard Strobe Port
PORTB  EQU      ^XBA              ;   ditto
KEYIN  EQU      ^XE8              ; Keyboard data Port.


        ORG     ^XF000
READKEY:
        LXI     B,KYDATA          ; Get PTR for buffer.
        MVI     A,^XFF            ; Disable normal key strobe
        OUT     PORTA             ;
        IN      PORTB             ; Get PortB Status.
        ANI     ^XFE              ; SET B0=0ff.
        OUT     PORTB             ; Activate Strobe for
                                  ;  Special key.
        IN      KEYIN             ; Read keyboard.
        STAX    B                 ; Save Data.
        IN      PORTB             ; Get Status of Port B.
        ORI     ^X01              ; Set B0=On.
        OUT     PORTB             ; Strobe off.
        MVI     A,^B11111110

NOMAL:
        INX     B                 ; Prepare PTR for key Buffer
                                  ; for next data.
        OUT     PORTA             ; Strobe On
        MOV     D,A
        IN      KEYIN             ; Get data.
        STAX    B                 ; Store it.
        MVI     A,^XFF
        OUT     PORTA             ; Strobe off.
        MOV     A,D               ; Retrieve strobe data.
        RLC                       ; Strobe for next column.
```

```
        JC      NOMAL
        RET                     ; All done return to caller.


        DS      1               ; PB0 column
        DS      1               ; PA0  ditto
        DS      1               ; PA1  ditto
        DS      1               ; PA2  ditto
        DS      1               ; PA3  ditto
        DS      1               ; PA4  ditto
        DS      1               ; PA5  ditto
        DS      1               ; PA6  ditto
        DS      1               ; PA7  ditto
                                ; Be careful that
                                ; Bit OFF means key
                                ; is depressed.

        END
```

CHAPTER 11

CMT INTERFACE

The physical interface of the CMT is described in this chapter. You can find how to control the Motor of the CMT, how to write a data to the CMT and how to read a data from CMT.

There is no description about file record format of PC-8201A. If you want the information about it, please refer to another technical manual about PC-8201A, which has already been released by NEC HE in Chicago.

11.1 HARDWARE FOR CMT

PC-8201A has the CMT interface for reading/writing data with Audio Cassette.

Reading/writing data with CMT is done via SID Pin ,SOD Pin of CPU(80C85). And Motor is controlled by SCP (System Control Port,^X90). The on-bit, Logical High, is represented by 2400Hz wave (called MARK) and the off-bit, Logical Low, is 1200Hz wave (called SPACE). So the Baud Rate of the CMT can be up to 1200 bps, bit per second. (System ROM, ROM #0 Uses 600 bps to maintain the compatibility with PC-8001A.)

11.1.1 Writing Operation.

While SOD is high, MARK is put out to MIC and TxC. Otherwise, SPACE is put out. Refer the next illustration.

```
        high    +-------+       +-------+       +--------
SIO             |       |       |       |       |
        low     |       +-------+       +-------+

                |       |       |       |       |
MIC/TxC         |< MARK>|<SPACE>|<MARK >|<SPACE>|MARK
```

Fig 11.1

11.1.2 Reading Operation.

Input wave from EAR Pin is reformed to Square wave and sent to SID Pin of 80C85 as shown blow. The input wave is inverted on the way to SID Pin from EAR Pin. In reading operation, the electric high/low level has no meaning. The pulse frequency indicate whether high or low of the data. The frequency, 2400Hz means logical high ,and the frequency, 1200Hz means low.

```
EAR -----          |<-- MARK --->|<--- SPACE ----->|
                   |             |                 |
                  -+ +-+ +-+ +-+ +--+  +--+  +--+  +---
SDI -----          | | | | | | | |  |  |  |  |  |  |
                   +-+ +-+ +-+ +-+  +--+  +--+  +--+
                      2400Hz         1200Hz
```

Fig 11.2

11.1.3 Baud Rate Generation.

Baud rate is Generated by software timing routine. In writing operation, the bit data for SOD Pin is set and it is held during the proper duration by the software wait-routine. On reading, a bit data is read in proper interval which is controlled by software. Refer to the following section about the software.

11.1.4 I/O Port For CMT

11.1.4.1 SCP ---- SYSTEM CONTROL PORT

I/O Address and Data Pattern

```
msb 7    6    5    4      3    2 - 0   lsb
  +----+----+----+----+------+--------+
  | XX | XX | XX | XX |REMOTE| XXXXXX |    OUT ^X90
  +----+----+----+----+------+--------+
```

REMOTE CMT Motor control.

0 = CMT Motor OFF
1 = CMT Motor ON

Description:

The current status of this port is saved at SYSSTAT(^XFE44), so you have to update this area when you want to change the status of this port.

11.1.4.2 PPI 81C55 Command Set

I/O Address and Data Pattern

```
msb 7   6   5   4   3   2   1   0 lsb
  +---+---+---+---+---+---+---+---+
  |TM2|TM1| 0 | 0 | ? | ? | 1 | 1 |    OUT ^XB8
  +---+---+---+---+---+---+---+---+
```

TM2/1 Timer Command for PPI

TM2	TM1	
0	0	--- NOP
0	1	--- Stop
1	0	--- Stop after Terminal Count
1	1	--- Start

11.2 SOFTWARE FOR CMT

11.2.1 CMT MOTOR CONTROL

CMT Motor on/off is simply performed by having access to the I/O PORT, SCP (System Control Port; ^X90). Output to SCP with on at the bit 3 starts the CMT Motor, and with off at bit 3 stops it.

Please make sure to update SYSSTAT(XFE44) in work Area.

```
;
; Turn on the motor.
;
CMTON:
        LDA     ^XFE44          ; Get SCP port status.
        ANI     ^B11110111      ; See if Motor ON?
        RNZ                     ;  then return.
        ORI     ^B00001000      ; Bit 4 on.
        OUT     SCP             ; Turn on Motor.
        STA     ^XFE44          ; Up-date Scp status.
        RET
;
; Turn off CMT Motor.
;
CMTOFF:
        LDA     ^XFE44          ; Get SCP Status.
        ANI     ^B11110111      ; Bit 4 OFF.
        OUT     SCP             ; Turn off Motor.
        STA     ^XFE44          ; Up-date SCP status.
        RET
```

11.2.2 Baud Rate Generation

Baud Rate must be generated by software timing routine. The CPU uses 2.4576MHz clock, so the time of 1 bit output/input should be counted with this clock. The sequence of the counting operation is shown blow.

BAUD RATE	NUMBER OF STATE for 1 Bit
75 bps	32448
150	16224
300	8112
600	4056
1200	2028

Fig 11.3

11.2.3 Write A Data To The CMT

Writing a data to the CMT is performed by controlling SOD pin. Following sample program illustrates how to write a byte to the CMT.

```
; Sample Program for writing data to the CMT
;
;
; Write a byte to the CMT,the lowest routine.
;
; Assumption:
;       CMT Motor rotating regularly and CALLED
;       Interrupt disable.
;
; INPUT:   [A] = Data to be send.
;
; OUTPUT:  Non.
;
; BAUD Rate = 600 bps
;
WRITE:
        MOV     B,A             ;   4: Save data.
        MVI     A,^X50          ;   7: Write start bit.
        SIM                     ;   4:
        CALL    HOLD            ;  18: Wait 4043 State.
        IN      PORTC           ;  10: Dummy to adjust timing.
        MOV     C,08            ;   4: Set data length in bit.
BYTEO:
        MOV     A,B             ;   4: Retrieve data.
        RLC                     ;   4: Set a bit in CF.
        MOV     B,A             ;   4: Save data.
        MVI     A,^XD0          ;   7: To send MARK.
        JC      BITO            ;10/7:  Branch if HIGH.
        MVI     A,^X50          ;   7: To send SPACE.
BITO:
        SIM                     ;   4:
        CALL    HOLD            ;  18: Wait 4018 state.
        DCR     C               ;   4: Bump counter.
        JNZ     BYTEO           ;10/7: To send next bit.
        MVI     A,^XD0          ;   4: To send stop bit.
        RET                     ;  10: It is responsible to
                                ;    : CALLER Routine for
                                ;    : making
                                ;    : an adequate
                                ;    : length of the stop
                                ;    :  bits.
;
```

```
; HOLD1 gives
;       24 * [HL] + 7 (+18)
; states delay. (+18) means 'CALL' instruction Status.
; So HOLD gives 4043 states delay including 'CALL' of Caller.
;
HOLD:
        LXI     H,167           ;  10: For 1 BIT (600Baud)
HOLD1:
        DCX     H               ;   6:
        MOV     A,L             ;   4:
        ORA     H               ;   4:
        JNZ     HOLD1           ;10/7:
        RET                     ;  10:
```

11.2.4 Reading A Data From The CMT

Following sample program shows how to read a byte form CMT.

```
;
; Sample Program for Reading a BYTE.
;       Assume Called with Interrupt disable.
READ:
        CALL    BITI            ;  10: Search for start
        JC      READ            ;10/7: Wait until Start bit
                                ;    : has come.
        LXI     H,????          ;  10:
        CALL    HOLD1
        MVI     C,8             ;   7: Read 8 BIT.
BYTEI:
        CALL    BITI            ;  18:
        MOV     A,B             ;   4:
        RLC                     ;   4: Move CF to Bit-0.
        MOV     B,A             ;   4:
        DCR     C               ;   4: Bump counter.
        JNZ     BYTEI           ;10/7: Read next BIT.
        RET                     ;  10: No check for Stop bit.

;
; Get a BIT.
;
; EXIT: CF = 1 if MARK.
;       CF = 0 if SPACE.
;
BITI:
        CALL    SYNC            ;  18:
        MOV     A,D             ;   4: Get counter.
        CPI     16              ;   7: See whether MARK
                                ;    : or SPACE.
                                ;    : If MARK then CF=1,
                                ;    : else CF=0.
        PUSH    PSW             ;  12: Save CF.
        LXI     H,???           ;  10: Assume MARK.
        JC      BITI1           ;10/7: Good assumption.
        LXI     H,???           ;  10:

BITI1:
        CALL    HOLD1           ;  18:
        POP     PSW             ;  10:
```

```
        RET                     ;  10:

;
; Calculate Pulse Duration.
;
; EXIT: [D] = loop count in this routine.
;
SYNC:
        MVI     D,36            ;   7: Reset counter.
                                ;    : Margin is about 10%.
        RIM                     ;   4:
        ANI     ^X80            ;   7: Isolate SID bit.
        MOV     E,A             ;   4:Save it.
SYNC1:
        RIM                     ;   4: Get Current status.
        ANI     ^X80            ;   7: Isolate SID bit.
        CMP     E               ;   4: Same status?
        JZ      SYNC1           ;10/7:   then wait.
SYNC2:
        RIM                     ;   4: Get current SID.
        DCR     D               ;   4: Bump counter.
        JZ      SYNC            ;10/7: Too long,Restart.
        ANI     ^X80            ;   7: Isolate SID.
        CMP     E               ;   4:
        JNZ     SYNC2           ;10/7:
        MOV     A,D             ;   4: Get result.
        CPI     11              ;   7: Too short?(392 state,
                                ;    : margin 20%)
        JNC     SYNC            ;10/7:  then restart.
        RET                     ;  10:
```

CHAPTER 12

SERIAL INTERFACE

PC-8201A has 3 channels of Serial Interface. They are used by RS-232C, SIO1, SIO2. The difference between SIO1 and SIO2 is only the shape of connector.

This chapter describes how to control the Serial Port.

12.1 HARDWARE OF SERIAL INTERFACE

UART(6402) and PPI(81C55) control the Serial Interface. Since they are shared by 3 channels, Only one channel is available at a time. Refer to the 'PC-8201A USER'S GUIDE' about capacity of the hardware.

12.1.1 I/O Port

12.1.1.1 Channel Select -- (System Control Port)

I/O Address and Data Pattern

```
msb 7    6    5 -- 0      lsb
  +----+----+------------+
  |SRI2|SRI1| XXXXXXXXXX |   OUT ^X90
  +----+----+------------+
```

SRI2/1 Serial Interface Select.

SRI2	SRI1	User
0	0	---- Not Used
0	1	---- SIO2 (Disk Driver)
1	0	---- SIO1
1	1	---- RS-232C

Note: Current status of this port is saved in SYSSTAT (^XFE44) by System ROM.

12.1.1.2 UART Mode Control

```
msb 7 - 5   4    3    2    1    0 lsb
+-------+----+----+----+---+---+
| XXXXX |CLS2|CLS1| PI |EPE|SBS|   OUT ^XD8
+-------+----+----+----+---+---+
```

SBS Stop Bit Select
- 0 = 1 bit
- 1 = 2 bits (*)

(*) When Data length is 5 bits, Stop Bits is 1.5 bit.

EPE Even Parity Enable
- 0 = Odd Parity
- 1 = Even Parity

(Meaningless if PI = 1)

PI Parity Inhibit
- 0 = Parity Enable.
- 1 = Parity Disable

CLS2/1 Character Length Select
- ^B00 = 5 bits
- ^B01 = 6 bits
- ^B10 = 7 bits
- ^B11 = 8 bits

12.1.1.3 UART Status Read

I/O Address and Data Pattern

```
msb          4     3    2    1    0  lsb
  +-------+----+----+----+----+------+
  ¦ XXXXX ¦TBRE¦ PE ¦ FE ¦ OE ¦dcd/dr¦    IN ^XDB
  +-------+----+----+----+----+------+
```

dcd/dr	DCD/DR on off (0=on/1=off)
OE	Over-run Error (1=Detected)
FE	Framing Error (1=Detected)
PE	Parity Error (1=Detected)
TBRE	Transmit Buffer Register Empty 1 = Ready to receive data to transmit.

12.1.1.4 UART Baud Rate (PPI 81C55 Timer Section)

I/O Addrend Data Definition

msb 6	5	4	3	2	1	0 lsb	
M2	T13	T12	T11	T10	T09	T08	OUT ^XBD
T06	T05	T04	T03	T02	T01	T00	OUT ^XBC

Specify timer output Mode

^B00 = Single Square Wave
^B01 = Continuous Square Wave
^B10 = Single Pulse On
^B11 = Continuous Pulse

set a Baud Rate use blow value.

ud Rate	^XBC	^XBD
75	00	48
150	6B	45
300	00	42
600	00	41
1200	80	40
2400	40	40
2400	40	40
4800	20	40
9600	10	40
19200	08	40

Fig 11.1

NOTE:

It is impossible to read the current UART status directly. ROM #0 always saves the new status in RAM when it is changed. Refer to Chapter 12.3.

12.1.1.5 UART DATA I/O Port

I/O Port and Data Pattern

```
msb                     lsb
+--+--+--+--+--+--+--+--+
|D7|D6|D5|D4|D3|D2|D1|D0| IN/OUT ^XCB
+--+--+--+--+--+--+--+--+
```

Note:
If the data length is less than 8 bits, Output data must be right justified. Input data is right justified by UART.

12.2 SOFTWARE DESCRIPTION.

12.2.1 How To Initialize Serial Port

The basic sequence to initialize Serial Port is as follows.

1. Select Channel
2. Set Baud Rate.
3. Set transfer mode.

Following Sample program shows the Initialization sequence more detailed.

The sample program listed blow explains how to initialize serial port. This sample program Initialize RS-232C Channel as 9600bps, even party,7 bit data length,1 stop bit and no control for Xon/Xoff,SI/SO. And it Updates work area for ROM#0 can be use the same mode. You may skip that portion if you want. They is no problem even if you skip the updating the data,because ROM#0 always initialize RS-232C Port when entering to Term mode or 'OPEN 'COM:'' of Basic command is issued by the Mode string.

12.2.1.1 Sample Program ... How To Initialize SERIAC Port

```
; Sample Program Initialize Serial Port.

; Data in system area which you must update.

SERMOD  EQU     ^XF406          ; 6 bytes for MODE string.
;               ^XF406          ; Baud rate Specifier.
;               ^XF407          ; Parity Mode.
;               ^XF408          ; Word Length.
;               ^XF409          ; Stop bits.
;               ^XF40A          ; XON/XOFF contorl.
;               ^XF40B          ; SI/SO control.
INHDSP
INHIBIT
COMACT  EQU     ^XFE43          ; current user ID for
                                ; serial port.
                                ;   ^B00 = Not used.
                                ;   ^B01 = SIO2
                                ;   ^B10 = SIO1
                                ;   ^B11 = RS-232C
SYSSTAT EQU     ^XFE44          ; SCP port status.
BAUDRT  EQU     ^XFE4A          ; Baud Rate Table entry
address.
INHBIT  EQU     ^XFE41          ; 0 inhibits XON/XOFF control.

; I/O Port Address.

SCP     EQU     ^X90            ; System Control Port.
PORTB   EQU     ^XBA            ; RTS/DTR set port.
TIMEL   EQU     ^XBC            ; Timer Set Low.
TIMEH   EQU     ^XBD            ;  //   //  High.

RTSDTR  EQU     ^X3F            ; RST/DTR data for RS-232C.
                                ; Use ^XFF for SIO1/2.


INITSERI:
; ENTRY: [C] = USER ID.
;        [B] = Baud rate specifier. ASCII Number (1 to 9)
;              Same Number of 'STAT' of TELCOM.

; -- See if Serial Port is available.

        LDA     COMACT          ; Get current user ID.
```

```
        ORA     A               ; No one use Serial I/O?
        JZ      SELECT          ; then branch.
        CMP     C               ; SAME USER?
        JZ      SELECT          ; Then branch.
        STC                     ; Set Error FLG.
        RET                     ; Return to caller.

SELECT:
; -- Reserve Serial Port -------
        DI                      ; Inhibit all disturbance.
        MOV     A,C             ; GET USER ID.
        STA     COMACT          ; Set User ID. Be sure reset
                                ; Use ID to 00 after all task
                                ; finished,else the serial
                                ; port
                                ; can not be shared to
                                ; another user.
        RRC                     ; Move Bit0-1 to Bit 6-7
        RRC
        MOV     C,A             ; Save it.
        LDA     SYSSTAT         ; Get current SCP status.
        ANI     ^B00111111      ; cancel channel control.
        ORA     C               ; Set new channel control
                                ; bits.
        OUT     SCP             ; Select channel.
        STA     SYSSTAT         ; Update SCP status.

;-- Set BAUD RATE --------------

SETBAUD:
        MOV     A,B             ; Get BAUD RATE ID.
        STA     SERMODE         ; Update Baud rate Specifier.
        SBI     '1'             ; Convert to Binary Number.
        RLC                     ; *2,Because table entry is
                                ; 2 bytes.
        LXI     H,TIMTBL        ;
        MOV     C,B             ; [C] = Offset
        MVI     B,0
        DAD     B
        SHLD    BAUDRT          ; Save entry point for
                                ; Music routine.
                                ; Music routine in ROM #0
                                ; destroy temporary changes
                                ; the timer count and
                                ; reinitializes it with
                                ; this entry data after
                                ; finish.
                                ; Refer Chapter 12.3

        MOV     A,M             ; Get Lower value.
```

```
        OUT     TIMEL           ;
        INX     H
        MOV     A,M             ; Get Higher Value.
        OUT     TIMEH
        MVI     A,^XC3          ; To start timer.
        OUT     ^XB8            ; Use this value to
                                ; start Timer.

; SET TRANSFER MODE.

MODE:
        IN      PORTB           ;
        ANI     RTSDTR          ; IF 232C RTSDTR=^X3F to
                                ; activate RtS/DTR,
                                ; else ^XFF to unactivate.
        OUT     PORTB
        IN      ^XC8            ; Dummy read to clear
                                ; Receive Buffer Register.
        MVI     A,^B00001110    ; 7bit,Even Parity,1 stop bit.
        OUT     0D8H            ; Set Mode.

; -- Update SERMODE

        LHLI    SERMODE+1       ; Set PTR
        MVI     M,'E'           ; Set Parity check mode.
        INX     H
        MVI     M,'7'           ; Set Word length.
        INX     H
        MVI     M,'1'           ; Set Stop bit length.
        INX     H
        MVI     M,'N'           ; Set XON/OFF control mode.
        INX     H
        MVI     M,'N'           ; Set SI/SO control Mode.
        XRA     A               ; Set CF=0
        STA     INHIBIT         ; Disable XON/XOFF control.
        EI
        RET

TIMTBL: DB      ^X00,^X48       ;    75 bps
        DB      ^X6B,^X45       ;   150
        DB      ^X00,^X42       ;   300
        DB      ^X00,^X41       ;   600
        DB      ^X80,^X40       ;  1200
        DB      ^X40,^X40       ;  2400
        DB      ^X20,^X40       ;  4800
        DB      ^X10,^X40       ;  9600
        DB      ^X08,^X40       ; 19200
```

12.2.2 SEND A Data To The Serial Port

The sample program shown blow describes how to send data to the serial port. It performs no XON/NOFF and no SI/SO control.

```
; SEND A data to the serial port
;
; ENTRY: [C] = DATA TO BE SEND
;

WRITE:
        IN      ^XD8            ; Get UART status.
        CPI     ^B00010000      ; See if transmitter buffer
                                ; register Empty?
        JZ      WRITE           ; Wait TBR become empty.
        MOV     A,C             ; Get character to send.
        OUT     ^XC8            ; Send it to the serial port.
        RET
```

12.2.3 Read A Data From Serial Port.

Sample program shown blowlexplains how to read data from serial port by RST6.5. This sample only read data form serial port with RST6.5,no XON/XOFF and no SI/SO control is performed.

```
;** Read a data from Serial Port.

;Read a data By RST6.5


        ORG     ^X3C            ; Entry point of RST6.5

RST65:  DI
        JMP     READ


        ORG     ????
READ:
        PUSH    H               ; Save registers.
        PUSH    D
        PUSH    B
        PUSH    PSW
        IN      ^XC8            ; Read the data
        MOV     L,A             ; Save it.
        IN      ^XD8            ; Get error status.
        ANI     ^B00001110      ; Strip error bit.
        MOV     H,A             ;
        SHLD    BUFFER
        POP     PSW             ; Restore Registers.
        POP     B
        POP     D
        POP     H
        EI
        RET

BUFFER  DS      1               ; Got Data.
        DS      1               ; Error status.
```

12.3 AVAILABLE SYSTEM AREA.

You may want to use the system area for your use. In this section, the available work area of ROM #0 is described. Make sure to keep the compatibility with System ROM, if you want use this area.

Serial input Buffer from ^XFE4C to ^XFFC3, is reserved by System ROM as SERIAL Input Buffer. And You can use it for your own routine.

SERMOD saves their RS-232C mode string

This area has 6 bytes data which indicates the RS-232C String Mode, specified by 'STAT' command in TELCOM or OPEN 'COM:' command in BASIC. The contents are following.

```
SERMOD  at ^XF406 DS6   ; RS232C String mode Buffer
           ^XF406       ; Baud rate specifier (1 to 9)
           ^XF407       ; PArity Mode (N/E/O/I)
           ^XF408       ; Word length specifier ( 5 to 8)
           ^XF409       ; Stop bit (1/2)
           ^XF40A       ; Xon off control (X/N)
           ^XF40B       ; SI/SO control (S/N)
```

INHIBIT (at ^XFE42)

This byte is the XON/XOFF Inhibit Flag. 0 inhibit XON/XOFF control,else enabled.

COMMACT (^XFE43 Byte)

This byte indicate who is using serial port as blow. Please reset to 0 after using the serial port, otherwise the serial port is not available for another user.

```
^X00 = No user
^X01 = SIO2
^X02 = SIO1
^X03 = RS-232C
```

CMPNT (at ^XFE46) DS1 ; Character count in Buffer.

This byte has the character count in Serial Buffer.

REDADDR (^XFE46 Byte)

This byte indicate last read character displacement.

WTADR (^XFE47 Byte)

This byte indicate last written character displacement.

BAUDRT (^XFE4A)

This points the table of the Baud rate. Refer to the Chapter 12.2.1.1 Sample Program.

CHAPTER 13

BARCODE READER

This chapter explains Electric specification and Basic theory of Operation of the Barcode Reader.

The Barcode Reader program included in the PC-8201A Personal Application Kit assumes that operation is done with the HEDS-3071(produced by HP Corp.)

13.1 ELECTRIC SPECIFICATION

Refer to the 'PC-8201A USER'S GUIDE' about the shape and Pin Connection of the BAR Code interface and electric specification.

You may connect any Bar Code Pen to this interface. But NEC recommends the products of YHP(YOKOKAWA HP) or (MECANO Kogyo) and it is better that the Pen has the Power switch, for saving the electric power of the PC-8201A.

The data line of Barcode Reader is connected to the Pin-2 of BCR. And this pin is connected to the RST5.5 of CUP(80C85) and Port C-3 of 81C55 as shown blow.

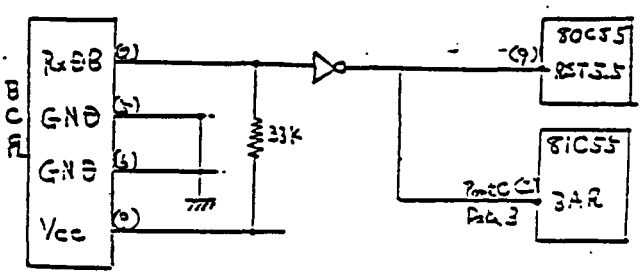


Fig 13.1

While the Barcode Reader is powered on, PIN-2 is kept as low level, and RST5.5 is High.

BLACK BAR is represented by logical Low, SPACE BAR by High respectively.

13.2 THEORY OF OPERATION

This section describes the basic sequence of the reading data from Barcode Reader.

1. If power on. RST5.5 is activated. At the first point of the RST5.5 routine which is interrupted by RST5.5 disable all interrupt.

2. Pole the Bar Code DATA port. And calculate the duration of same status and save the status and Duration.

3. If Low level continues too long assume that Power off and enable

4. Decode the got Data and transfer the data to the upper routine.

CHAPTER 14

PARALLEL INTERFACE

This chapter describes how to control the Printer Interface of the PC-8201A. It is the Centronics compatible a 8-bit parallel interface.

14.1 HARDWARE SPECIFICATION

14.1.1 Physical Interface Of PC-8201A

PC-8201A has the Centronics compatible parallel interface. It uses 26-pin connector. Refer to the PC-8201A USER'S GUIDE about the Pin connection and signal name.

14.1.2 I/O Port For PRINTER Interface.

14.1.2.1 Port A ---- Data Out Put Port For Printer.

```
lsb 7   6   5   4   3   2   1   0 lsb
  +---+---+---+---+---+---+---+---+
  |PD7|PD6|PD5|PD4|PD3|PD2|PD1|PD0| OUT ^XB9
  +---+---+---+---+---+---+---+---+
```

PD7 to PD0 -- DATA output to Printer.

NOTE: This port is used by another user.

14.1.2.2 Port C ---- BUSY,SLCT Signal Read

```
msb 7  6  5  4  3  2    1    0 lsb
  +--+--+--+--+--+----+----+----+
  |XX|XX|XX|XX|XX|BUSY|SLCT| XX | IN ^XBB
  +--+--+--+--+--+----+----+----+

            BUSY --- 0  Printer READY
                     1  Printer BUSY

            SLCT --- 0 deselect
                     1 Select
```

14.1.2.3 SPC(System Control Port) --- STROBE Output Port

```
msb 7  6  5    4  3  2  1  0 lsb
  +--+--+----+--+--+--+--+--+
  |XX|XX|PSTB|XX|XX|XX|XX|XX| OUT ^X90
  +--+--+----+--+--+--+--+--+

            PSTB --- 0 Strobe OFF
                     1 Strobe ON
```

14.1.3 Basic Theory Of Writing A Data To Centronics

The basic sequence to write data to the Centronics printer is as follows.

1. If Printer is busy, wait a while. Otherwise go ahead.
2. Output a byte to the data lines and hold it.
3. Change the strobe level to low.
4. Wait a adequate duration holding the DATA.
5. All has been done, then finish else repeat from (1).

The timing chart illustrates the sequence.

```
Parallel   __XXXXXXXXXXXXXX_________
 DATA            ->|T1|<- ->| T2 |<-
              -------+     +-------------------
DATA                ->| T3  |<-
 STROBE               +------+

                          +---------------+
BUSY                      |               |
             --------------+               +-----
```

T1,T2 >= 1.0 uSec
1.0 uSec < T3 < 600uSec

Fig 14.1

Refer to the Manual of Printer about the actual Duration of T1 to T3.

14.2 SOFTWARE SPECIFICATION

14.2.1 How To Write A Byte To The Printer.

Tiny program shown blow explains how to send a character to the Parallel port. That sample Program does same function as Basic command,

```
LPRINT "ABCDEFGHIJ"
```

```
;
;
;
 60000D
;-- Equater --

SCP    EQU     ^X90              ; System Control Port.
PORTA  EQU     ^XB9              ; Printer Data Port.
PORTC  EQU     ^XBB              ; Printer Status Port.
SYSSTAT        EQU     ^XFE44          ; SPC status.


START:
       LXI     H,BUF             ; Set PTR.
       MVI     C,10+2            ; Set data length.
PRINT:
       IN      PORTC             ; Get Printer status.
       ANI     6                 ; Strip BUSY,SLCT bits.
       XRI     2                 ; See if ready.
       JNZ     PRINT             ;  if not,then wait.

       DI                        ; Inhibit disturb for Port A
                                 ; of 81C55.
       MOV     A,M               ; Get character to Print.
       OUT     PORTA             ; Put data on the DATA line.
       LDA     SYSSTAT           ; Get SCP status.
       MOV     B,A               ; Save It.
       ORI     ^B00100000        ; Set STROBE.
       OUT     SCP               ;
       MOV     A,B               ;
       OUT     SCP

       MOV     B,^X03            ; Please set appropriate
                                 ; value for your Printer.
```

```
WAIT:
        DCR     B
        JNZ     WAIT
        EI

        INX     H               ; Point to Next
        DCR     C
        JNZ     PRINT
        RET

BUF:    DB      'ABCDEFGHIJ'
        DB      13,10

        END
```

CHAPTER 15

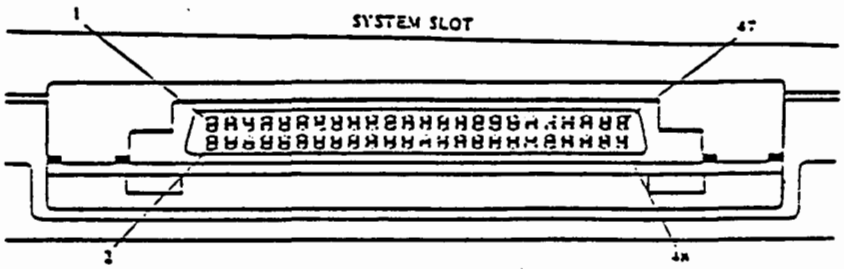
HARDWARE

er to another technical manual about the detail specion of PC-8201A's hardware. That manual has already been by NECHE, Chicago. Please contact with them. In this r, only most important data is listed up.

15.1 SYSTEM SLOT

15.1.1 Assignment Of Signal

System Slot



Pin number	Signal name	Remarks
1	VDD	+5 V
2	VDD	+5 V
3	AD0	Address/Data 0
4	AD4	Address/Data 4
5	AD1	Address/Data 1
6	AD5	Address/Data 5
7	AD2	Address/Data 2
8	AD6	Address/Data 6
9	AD3	Address/Data 3
10	AD7	Address/Data 7
11	NC	No Connection
12	NC	No Connection
13	A8	Address 8
14	A12	Address 12

Fig 15.1

Pin number	Signal name	Remarks
15	A9	Address 9
16	A13	Address 13
17	A10	Address 10
18	A14	Address 14
19	A11	Address 11
20	A15	Address 15
21	A16	No Connection
22	A18	No Connection
23	A17	No Connection
24	A19	No Connection
25	NC	No Connection
26	NC	No Connection
27	RD	Read
28	$\overline{WR}$	Write
29	IO/M	IO OR Memory
30	ALE	Address Latch Enable
31	HOLD	HOLD
32	HOLDA	HOLD Acknowledge

Fig 15.2

Pin number	Signal name	Remarks
33	INTR	INTERRUPT
34	INTA	INTER Acknowledge
35	RESET	RESET
36	READY	READY
37	$\overline{\text{ROME}}$	ROM Enable
38	E	Enable
39	$\overline{\text{BANK\#3}}$	RAM Cassette Select signal
40	NC	No Connection
41	HADRD	High Address Disable
42	LADRD	Low Address Disable
43	CLK	Clock
44	POWER	RAM Protect signal
45	GND	Ground
46	GND	Ground
47	NC	No Connection
48	NC	No Connection

Fig 15.3

15.1.2 Explanation Of Pin

15.1.2.1 Function Of Signal

1. Vdd (Out)

   If you don't use the BCD, this Pin can supply with the current of 50mA or so.

2. AD0-AD7 (In/Out)

   Lower 8 bits of the memory address (or I/O address) appear on the bus during the first clock cycle of a machine cycle. It then becomes the data bus during the other cycles.

3. A8-A15 (Out)

   The most significant 8 bits of the memory address or the I/O address. The output goes off during Hold mode,it then becomes 'H' level, because it is connected to a pull up resister (100k Ohm) inside.

4. /RD (Out/3-state)

   The read control signal, 3-state during Hold mode.

5. /WR (Out/3-state)

   The write control signal, 3-state during Hold mode.

6. IO/M (Out/3-state)

   When this signal is 'H' level and 'L' level,

respectively, the CPU have access to the I/O and the memory. 3-state during Hold mode.

7. ALE (Out/3-state)

It is used to strobe the address information (AD0-AD7). 3-state during Hold mode.

8. HOLD (In)

The CPU, upon receiving the hold request, will relinquish the use of the bus as soon as the completion of the current bus transfer. When the Hold is acknowledged, the /RD, /WR, IO/M, ALE lines are 3-stated and the AD8-AD15 lines are 'H' level.

9. HLDA (Out)

It indicates that the CPU has received the HOLD request and that it will relinquish the bus in the next clock cycle.

10. INTR (In)

The general purpose interrupt. It is sampled only during the next to the last clock cycle of an instruction and during Hold and Halt states.

11. /INTA (Out)

It is used instead of (and has the same timing as) /RD during the instruction cycle after an INTR is accepted.

12. RESETO (Out)

It indicates CPU is being reset. Can be used as a system reset.

13. READY (In)

If it is 'L', the CPU will wait an integral number of clock cycles for it to go 'H' before completing the read or write cycle.

14. /ROME (Out)

The enable signal for external ROM cartridge or general purpose. When the upper 4 bits of the I/O address is 8, it goes 'L'.

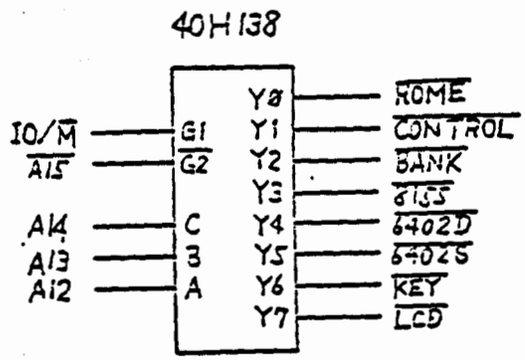


Fig 15.4

15. E (Out)

It is used as a memory enable signal of the read or write cycle. E is the logical OR (active high) of /RD and /WR.

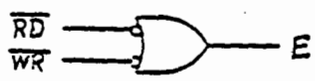


Fig 15.5

16. /BANK 3 (Out)

The memory enable signal of external RAM cartridge. (See next section)

17. HADRSD (IN)

If it is 'H',the memory of high address (^X8000 to ^XFFFF) in PC is disabled. (See next section)

18. LADRSD (IN)

If it is 'H',the memory of LOW address (^X0 to ^X7FFF) in PC is disabled. (See next section)

19. CLK (Out)

2.5MHz clock output. It is the same phase as CPU clock.

20. POWER (Out)

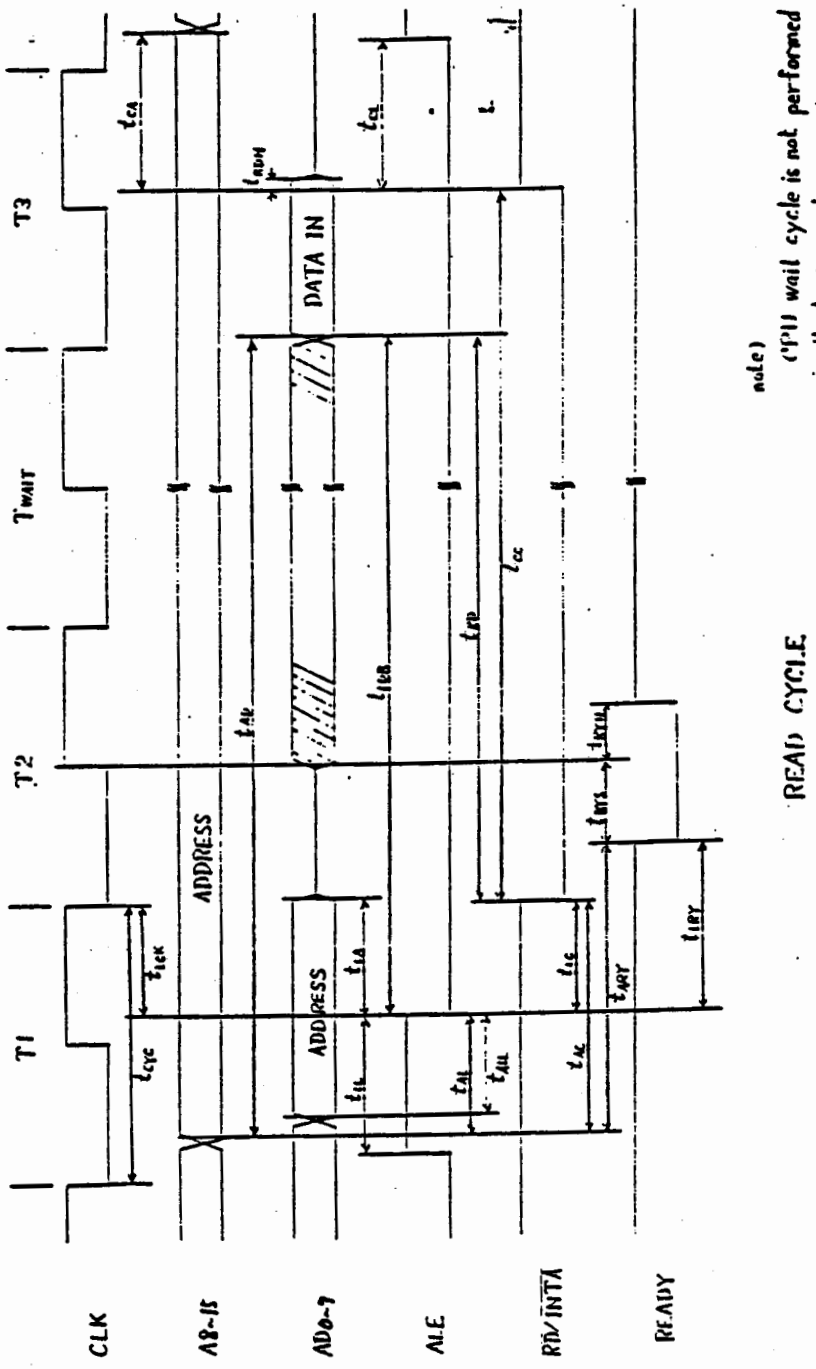
It is the signal /RESET (connected to the CPU) is reversed.

15.1.3 DC Characteristics

Symbol	Drive capacity (mA)
AD0-AD7	4.4
A8-A15	4.4
/RD,/WR,IO/M ALE,RESETO	4.4
HLDA,/INTA,CLK	2.0
E,/ROME,/BANK 3	1.1

Fig 15.6

15.1.4 AC Characteristics



READ CYCLE

Fig 15.7

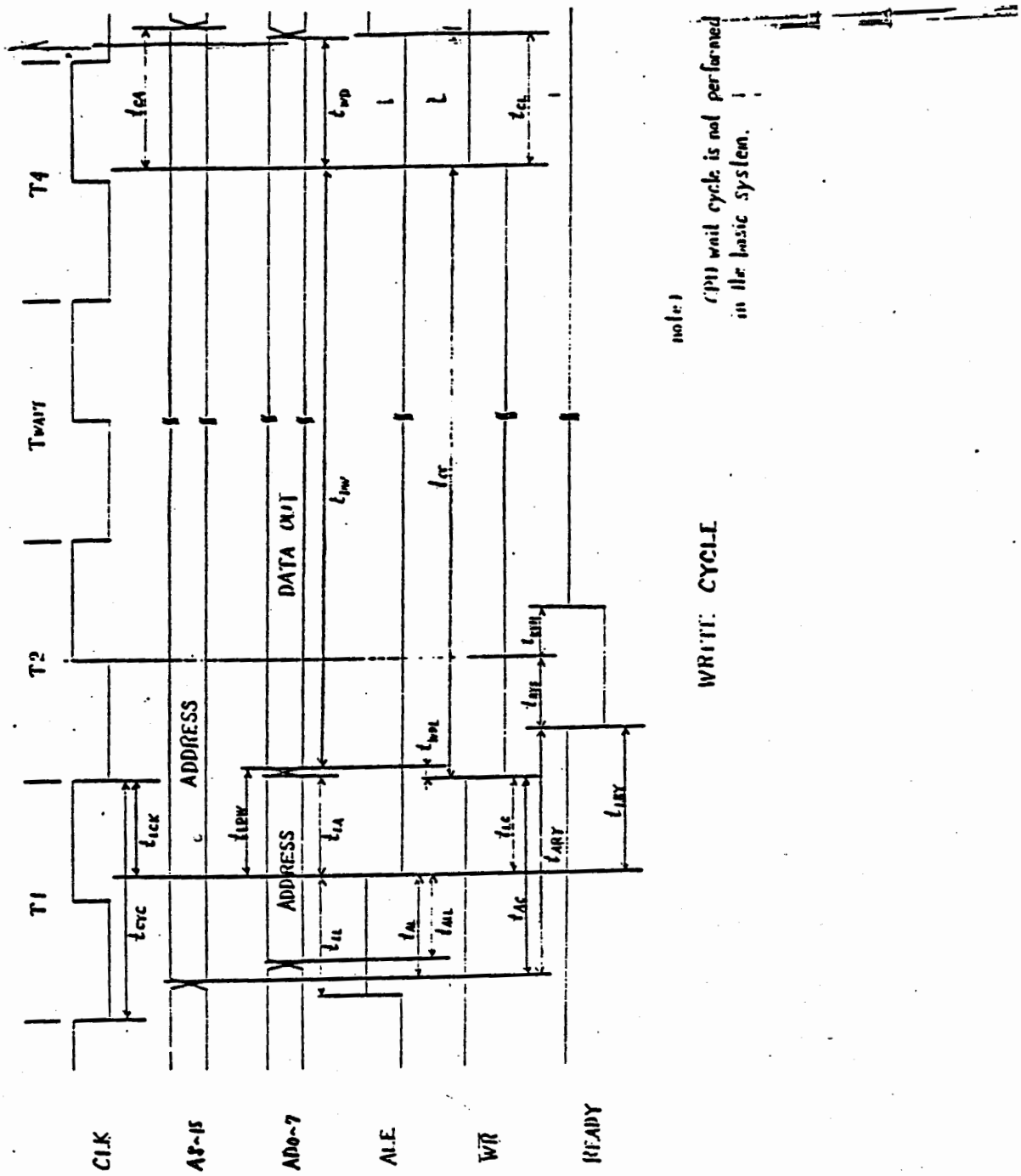

T1
T2
TWAIT
T4
CLK
A8~15
AD0~7
ALE
WR
READY
ADDRESS
ADDRESS
DATA OUT
WRITE CYCLE
CPU wait cycle is not performed in the basic system.

Fig 15.8

* disregard T_{WAIT}

	min (nS)	typ (nS)	max (nS)
t_{CYC}		407	
t_{LCX}	112		
t_{AL}	112		
t_{ALL}	71		
t_{LL}	162		
t_{LA}	142		
t_{AC}	357		
t_{LC}	173		
t_{AD}			[illegible]
t_{LDR}			563
t_{RD}			334
t_{CC}	525		
t_{CA}	163		
t_{RDH}	0		
t_{WDL}			75
t_{CL}	93		
t_{WD}	88		
t_{DW}	515		
t_{LDW}			242
t_{ARY}			309
t_{LRY}			69
t_{RYS}	110		
t_{RYH}	0		

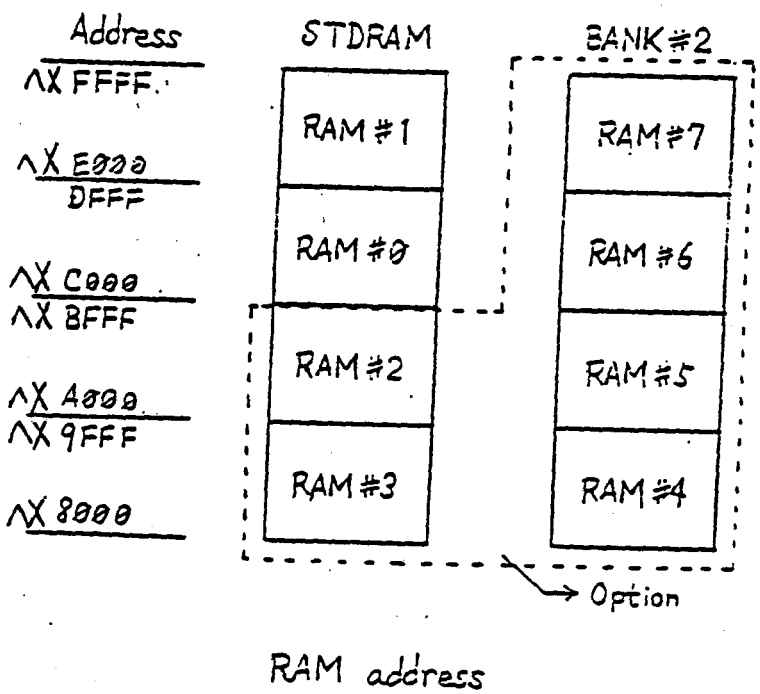
Fig 15.9

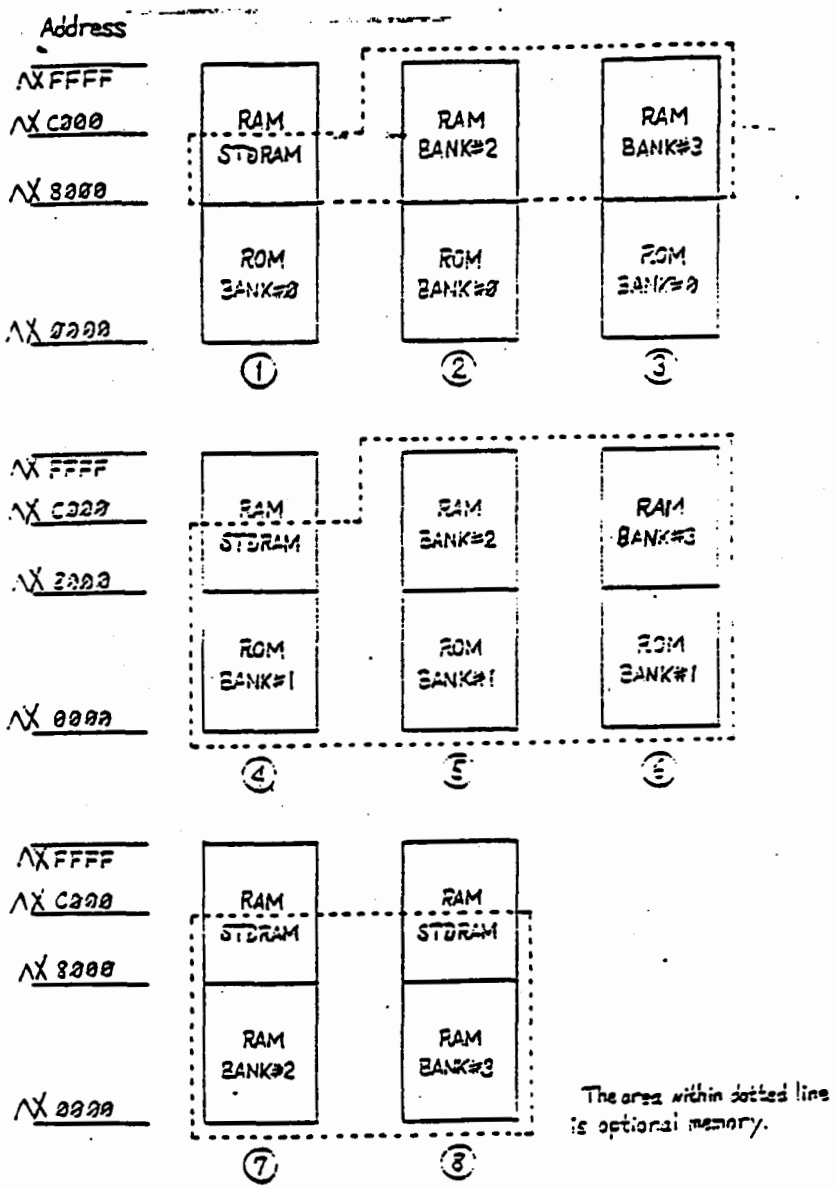
15.2 MEMORY CONTROL CIRCUIT

In this section, RAM #n means the chip number on the main board.

The memory of PC-8201A consists of RAM 16K and ROM 32K bytes,and can be expanded to 48K bytes on optional RAM socket (RAM Chip #2- #7) and to 32K bytes on user ROM socket (ROM #1) in PC.

Show the composition of memory in Fig 15.11 RAM Chip (#0- #7) and ROM (#0- #1) is connected to the same DATA bus and their outputs are controlled by /CE and /BANK signal. There are five banks of BANK #0(available ROM #0), BANK #1(user ROM #1), STDRAM(available RAM #0- #1 and optional RAM #2-#3) ,BANK #2 (optional RAM #4- #7) and BANK #3 (RAM cartridge). Show the bank control circuit in Fig 15.12 By means of this, you can assign each back to the memory address in 64K bytes area of CPU shown in Fig 15.13 and Fig 15.10.





Variety of memory composition

Fig 15.10

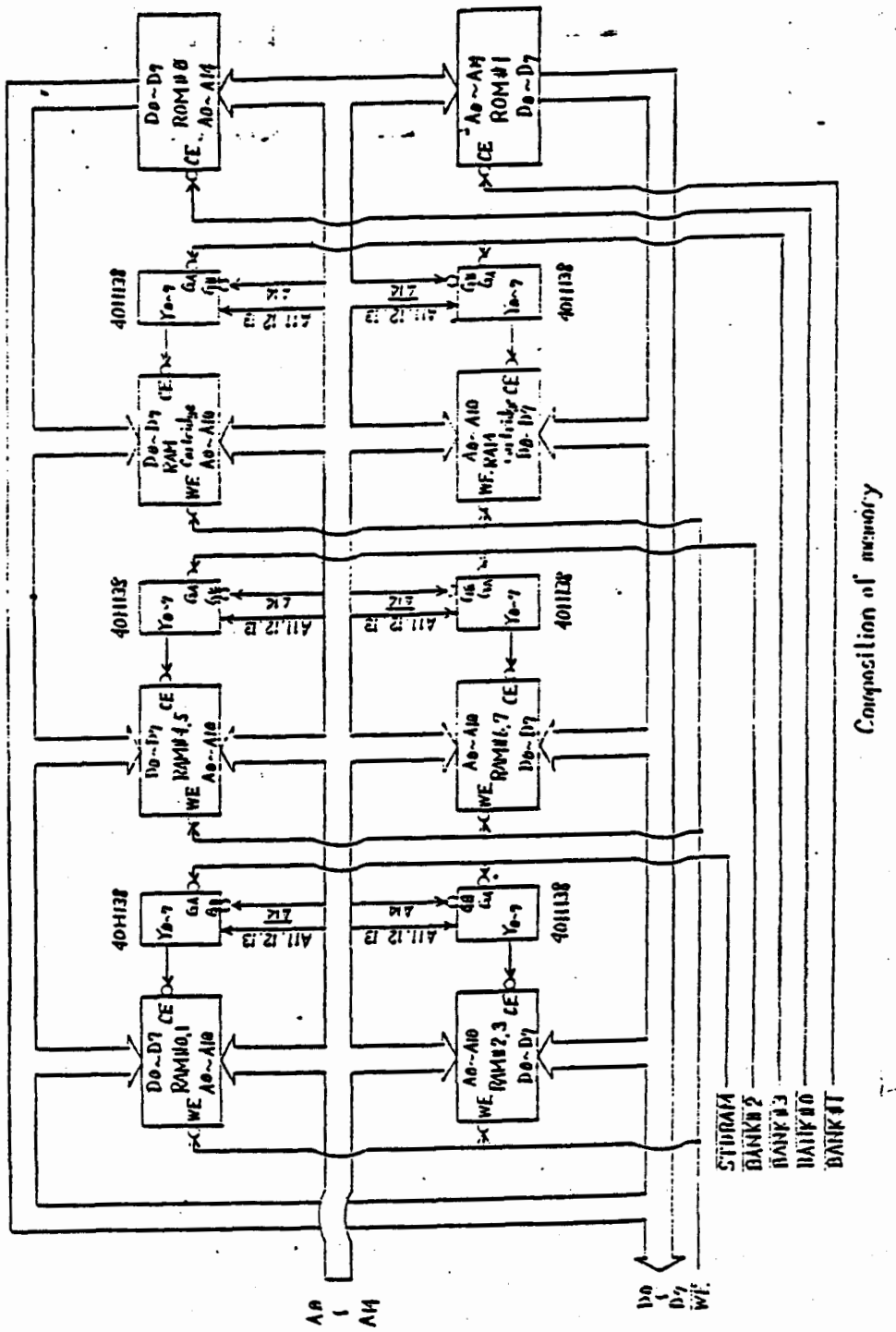
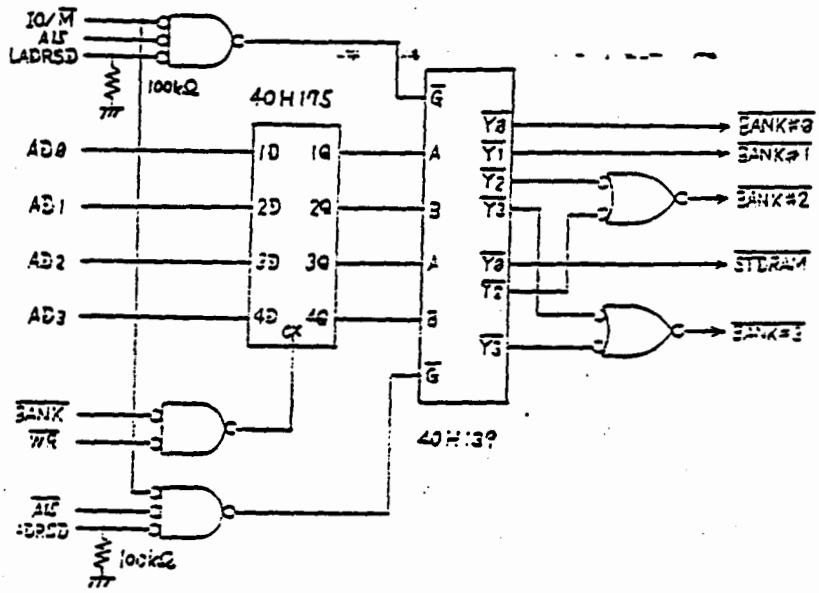

Composition of memory
4011138
4011138
4011138
4011138
ROM#0
ROM#1
RAM0,1
RAM2,3
RAM4,5
RAM6,7
RAM Controller
RAM Controller
STDRAM
BANK#2
BANK#3
BANK#0
BANK#1

Fig 15.11



Bank Control Circuit

Fig 15.12

Variety of memory composition

Bank Control Signal \ Composition	①	②	③	④	⑤	⑥	⑦	⑧
LADR1	0	0	0	0	0	0	1	1
LADR2	0	0	0	1	1	1	0	1
HADR1	0	1	1	0	1	1	0	0
HADR2	0	0	1	0	0	1	0	0

Fig 15.13

The way of bank conversion by software control illustrates in next section. When PC is reset, it becomes any mode (before reset)of the composition No.1-3. But in the case of nothing of optional RAM BANK #2- #3, it can become only No.1 mode. If optional ROM is installed, another composition No.4-6 are possible. Further, as it becomes the mode of 64K bytes full RAM by optional RAM BANK #2- #3, you can use a CP/M, etc.

15.3 I/O ADDRESS

(Address is expressed in Binary.)

I/O address	In/Out	I/O device	Operation
00000000 ↓ 01011111			user
01100000 ↓ 01111111			NEC reserve
1000XXXX	O	NEC reserve (ROM cartridge or general purpose) A decoded signal appears on /ROME pin.	
1001XXXX	O	D-FF	System Control *Cassette Motor Control *Clock Command Strobe *Printer Strobe *Serial I/F Select
1010XXXX	O	D-FF	Bank Control
1010XXXX	I	3-S -Buff	Bank Status *Bank Status *Serial I/F Select Status
1011X000	I/O	PPI 81C55	Command/Status Resister
1011X001	O		Port A Output *LCD Chip Select *Printer Data *Keyboard Scan Data *Clock Command/Data

1011X010	O		Port B Output *LCD Chip Select *Buzzer Control *RS-232C Control *Auto Power Off Control
1011X011	I		Port C Input *Clock Data *Printer Status *BCR Data *RS-232C Status
1011X100	O		Timer Resister (lower 8 bits) *Lower 8 bits of counter
1011X101	O		Timer Resister (upper 8 bits) *Upper 6 bits of counter *Mode Select
1100XXXX	I/O	UART 6402	Data Write/Data Read
1101XXXX	O		Control
1101XXXX	I	3-S-Buff	Input Port *UART Status *Low Power Signal
1110XXXX	I	3-S-Buff	Keyboard Input
1111XXX0	O	LCDC	Command Write/Status Read
1111XXX1	O		Data Write/Data Read

Fig 15.14

15.3.1 Detail Information About I/O

This following is the particulars of each function. The I/O address is shown in the number which is used really in system.

15.3.1.1 Reserve Area

As this area is reserved for NEC,don´t use it.

15.3.1.2 System Control

```
-----------------
|1 0 0 1 0 0 0 0| OUT ^X90
-----------------

  7    6    5     4    3
-----------------------------------
|SELA|SELB|PSTB|TSTB|REMOTE|       |
-----------------------------------
REMOTE  CASSETTE MOTOR CONTROL
  0      motor Off
  1      motor On

TSTB    CLOCK COMMAND STROBE
  0      Strobe Off
  1      Strobe On

PSTB    PRINTER STROBE
  0      Strobe Off
  1      Strobe On

SEL A  SEL B    SERIAL INTERFACE SELECT
  0      0       Not used
  0      1       SIO2
  1      0       SIO1
  1      1       RS-232C
```

15.3.1.3 Bank Control

```
-----------------
|1 0 1 0 0 0 0 1|  OUT ^A1
-----------------

          3     2     1     0
-----------------------------------
|       |HARD2|HARD1|LADR2|LADR1|
-----------------------------------
```

LADR 2	LADR 1	SELCT ADDRESS ^X0 To ^X7FFF
0	0	Bank #0 (ROM #0)
0	1	Bank #1 (ROM #1)
1	0	Bank #2 (RAM #4 - #7)
1	1	Bank #3 (RAM cartridge)

HADR 2	HADR 1	SELECT ADDRESS (^X8000 TO ^XFFFF)
0	0	Standard RAM (RAM #0 - #3)
0	1	Not Used
1	0	Bank #2 (RAM #4 - #7)
1	1	Bank #3 (RAM Cartridge)

15.3.1.4 Bank Status

```
-----------------
!1 0 1 0 0 0 0 0! IN ^XA0
-----------------

 7   6        3  2  1  0
---------------------------
!   !   !    !   !   !   !
---------------------------
```

BIT 1	BIT 0	STATUS OF ADDRESS (^X0 TO ^X7FFF)
0	0	Bank #0 (ROM #0)
0	1	Bank #1 (ROM #1)
1	0	Bank #2 (RAM #4 - #7)
1	1	Bank #3 (RAM cartridge)

BIT 3	BIT 2	STATUS OF ADDRESS (^X8000 TO ^XFFFF)
0	0	Standard RAM (RAM #0 - #3)
0	1	Not Used
1	0	Bank #2 (RAM #4 - #7)
1	1	Bank #3 (RAM cartridge)

BIT 7	BIT 6	STATUS OF SERIAL INTERFACE
0	0	Not used
0	1	SIO2
1	0	SIO1
1	1	RS-232C

15.3.1.5 PIO 81C55 Address

*Command / Status Resister

1 0 1 1 1 0 0 0	IN/OUT ^XB8

*Port A output

1 0 1 1 1 0 0 1	OUT ^XB9

7	6	5	4	3	2	1	0
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD1
KS7	KS6	KS5	KS4	KS3	KS2	KS1	KS0
			CCK	CD0	C2	C1	C0

PA7 to PA0	LCD Chip Select
PD7 to PD0	Printer Data Port
KS7 to KS0	Keyboard
C2 to C0	Clock command Output Port
CD0	Clock Data Output Port
CCK	Calendar Shift Clock
0	Clock Off
1	Clock On

*Port B Output

1 0 1 1 1 0 1 0	OUT ^XBA

```
  7    6    5    4    3    2    1    0
-------------------------------------
|---|---|    |    |DCD/|--|    |    |
|RTS|DTR|BELL|APO|RD  |MC|PB1|PB0|
-------------------------------------
-------------------------------------
|                              |KS8|
-------------------------------------
```

PB1 -- PB0 LCD Chip Select

--
MC MEMORY CONTROL OUTPUT
0 On
1 Off

--
DCD/RD DCD/RD SELECT OF THE RS-232C
0 Ring Detect
1 Data Carrier Detect

AP0 AUTO POWER OFF OUTPUT
0 Output Off
1 Output On

BELL BUZZER OUTPUT
0 Ring
1 Not Ring

DTR RS-232C DTR output Active Low

RTS RTS output Active Low

*Port C Input

```
        -----------------
        |1 0 1 1 1 0 1 1|  IN ^XBB
        -----------------

                 5   4   3   2    1    0
      ------------------------------------
      |        |---|---|   |    |    |   |
      |        |DSR|CTS|BCR|BUSY|SLCT|CDI|
      ------------------------------------
```

CDI	Clock Data Input Port
SLCT	PRINTER BUSY
0	Printer Ready
1	Printer Busy
BCR	Bar Code Reader Data Input Port

CTS	CTS Input Active Low

DSR	RS-232C DSR Input Active Low

*81C55 Timer Resister

```
-----------------
|1 0 1 1 1 1 0 0| OUT/IN ^XBC
-----------------

  7   6   5   4   3   2   1   0
-----------------------------------
|TL7|TL6|TL5|TL4|TL3|TL2|TL1|TL0|
-----------------------------------
```

TL7 -- TL0 Timer Counter Lower 8 bit

```
-----------------
|1 0 1 1 1 1 0 1| OUT/IN ^XBD
-----------------

 7   6  5   4   3   2   1   0
-------------------------------
|M2|M1|TH5|TH4|TH3|TH2|TH1|TH0|
-------------------------------
```

TH5 -- TH0 Timer Counter Upper 6 bit

M2	M1	
0	0	This mode transmits a single-square wave which the first half of the number of count is high and remaining is low. (Mode 0)
0	1	This mode continually transmits a Mode 0 type square wave. (Mode 1)
1	0	This mode transmits a L-pulse (single pulse) during one clock when finishing the terminal count. (Mode 2)
1	1	This mode continually transmits a Mode 2 type pulse. (Mode 3)

15.3.1.6 UART Data I/O Port

```
-----------------
!1 1 0 0 1 0 0 0! IN/OUT ^XC8
-----------------
UART DATA PORT
```

15.3.1.7 UART Control Port

*Command Write

```
-----------------
!1 1 0 1 1 0 0 0! OUT ^XD8
-----------------

-------------------------------
!        !CLS2!CLS1!PI!EPE!SBS!
-------------------------------
```

SBS STOP BIT SELECT

0 Stop bit length is 1 bit

1 Stop bit length is 1 bit.
If data length is 5 bits,
stop bit length is 1.5 bits.
In the other case, it is 2 bit.

EPE EVEN PARITY ENABLE

0 Odd Parity

1 Even Parity

P1 PARITY INHIBIT

0 Generate parity and check

1 Inhibit generating parity

and check

CLS 2	CLS 1	CALENDAR LENGTH SELECT
0	0	Data Length 5 bits
0	1	Data length 6 bits
1	0	Data length 7 bits
1	1	Data length 8 bits

*Status read

```
-----------------
|1 1 0 1 1 0 0 0| IN ^XD8
-----------------

  7         4   3  2  1    0
-----------------------------------
|LPS|      |TBRE|PE|FE|OE|--- /--|
|   |      |    |  |  |  |DCD/ RD|
-----------------------------------
```

DCD/RD	Data Carrier Detect/Ring Detect
0	On
1	Off
OE	Overrun Error
1	Detected
FE	Framing Error
1	Detected
PE	Parity Error
1	Detected
TBRE	Transmitter Buffer register Empty
1	ready to receive data to transmit
LPS	LOW POWER SIGNAL
1	low power voltage

15.3.1.8 Keyboard Input

```
-----------------
|1 1 1 0 1 0 0 0|  IN ^XE8
-----------------
```

15.3.1.9 LCDC Address

* Command Write /Status Read

```
-----------------
|1 1 1 1 1 1 1 0|  IN/OUT ^XFE
-----------------
```

* Data Write/Read

```
-----------------
|1 1 1 1 1 1 1 1|  IN/OUT ^XFF
-----------------
```