NEC PC-8300

Technical Reference Manual

PC-8300A SYSTEM WORK AREA

F380-F381	Flag for 1st reset or not 4D8A Not 1st power on
F382-F383	ID area for the power down when POWER OFF, RESUME
F384-F385	Highest memory location
F386-F388	Power on hook
F389-F38B	Barcode reader hook
F38C-F38E	UART hook
F38F-F391	Interval timer hook
F392-F394	LOW BATTERY interrupt vector
F395-F3B8	2nd ROM check routine
F3B9-F3BE	2nd ROM EXECUTE routine
F3BF	2nd ROM flag
F3C0-F3C1	TELCOM function key data top address
F3C2-F3C3	TERM function key data end+1 address
F3C4-F3C5	Pointer for statement process address table
F3C6-F3C7	Pointer for function process address table
F3C8-F3CA	Value check routine hook
F3CB-F3CD	1 character input from CASSETTE hook
F3CE-F3D0	1 character output to CASSETTE hook
F3D1-F3D3	CASSETTE write on hook
F3D4-F3D6	CASSETTE read on hook
F3D7-F3D9	Statement hook
F3DA	Insert mode flag
F3DB	The bank number when power off 00: Bank 1 08: Bank 2 0C: Bank 3

F3DC	Cursor pattern in insert mode
F3DD-F3DF	Error routine hook
F3E0-F3E1	Function key data pointer
F3E2-F3E3	PAST key data pointer
F3E4	Console flag 00: LCD xx: CRT
F3E5	Cursor position Y
F3E6	Cursor position X
F3E7	Max line number on LCD
F3E8	Max character length on LCD
F3E9	Function key display flag
F3EA	Screen lock flag (No scroll)
F3EB	Cursor Flag 00: Off xx: On
F3EC	Cursor position Y on LCD
F3ED	Cursor position X on LCD
F3EE	Cursor position Y on CRT
F3EF	Cursor position X on CRT
F3F0	Max line number on CRT
F3F1	Max character length on CRT
F3F2	Escape sequence counter
F3F3	Area for F3F2
F3F4	Reverse mode flag 00: Normal xx: Reverse
F3F5	Max character length on printer
F3F6-F3F9	The string for printer line number

F3FA-F3FB	Graphics cursor (Y,X)
F3FC	Current mode Bit 7: TEXT Bit 6: TELCOM
F3FD	Flag for EDIT mode
F3FE=F3FF	Error entry
F400	Not used
F401	Key wait flag 00: Not wait xx: Wait
F402	The timer for auto power off
F403	TERM full, half flag
F404	TERM Echo flag
F405	Flag for CR,LF mode
F406-F40B	RS-232C control string
F40C-F40E	When .CO file was executed, this entry will be used for the start entry
F40F-F412	Not used
F413-F415	Subroutine for OUT
F416-F423	High speed SUB routine
F424-F426	Work area random number
F427-F446	Data for random number
F447-F44A	Last random number
F44B-F44D	Subroutine for INP function
F44E-F452	Dummy end marker for direct statement
F453	Error number save area
F454	Not used
F455	The value in LPOS(0)

F456	Output select flag 00: Console xx: Printer
F457	The column position inside of print statement
F458	DEL, BS flag
F459-F45A	Stack top address
F45B-F45C	Current line number
F45D-F45E	BASIC TEXT pointer
F45F-F461	Pointer for VAL function
F462-F59F	Buffer for pre-compile
F5A0	Terminator for INPUT
F5A1-F6A2	Buffer for LINE INPUT
F6A3	Stopper for LINE INPUT full
F6A4	Cursor position for CONSOLE (X)
F6A5-F744	Function key data area
F745	Directory flag
F746-F7E5	Function key data area for BASIC
F7E6-F826	When RESET, the IPL file execution
F827-F82E	Line terminate flag
F82F-F830	1st cursor position in SCREEN EDIT
F831	Cursor flag 00: Normal xx: Insert mode
F832-F83D	Buffer for timer IC
F83E-F83F	Interval timer counter
F840	Auto power off timer (x6 second = Power off)
F841	Flag for time out
F842-F84B	Not used

F84C-F84E	The mode of COM interrupt and entry
F84F-F86F	ROM file directory
F870-F87A	Non-registered file directory
F87B-F885	PAST file directory
F886-F890	EDIT file directory
F891-F977	User file directory
F978	End of directory (FF)
F979-F97A	Directory pointer
F97B	Back up character for CASSETTE input
F97C	Back up character for RS-232C input
F97E-F98D	Back up character for RAM file input
F98E	Back up character for PAST file
F98F-F990	File number
F991-F998	2nd ROM directory
F999	IPL flag
F99A	Cursor flag buffer for screen edit
F99B-F9A6	FA8E-FA99 buffer for EXEC
F9A7	A register buffer for EXEC
F9A8-F9A9	HL register buffer for EXEC
F9AA-F9AB	Start line address for LIST
F9AC-F9AD	Last line+1 address for LIST
F9AE-F9AF	Stack pointer save area for auto power off
F9B0-F9B1	RAM start address
F9B2	TERM Down Load flag
F9B3	TERM Up Load flag
F9B4-F9B5	Pointer for Down Load

F9B6	Before character for TERM
F9B7	RS-232C Output SI,SO flag
F9B8	RS-232C Input SI,SO flag
F9B9	When console is not CRT, the value is not zero
F9BA	Menu mode flag
F9BB	Menu mode sub-command flag
F9BC	Hook number save area
F9BD	Cursor X position in TEXT
F9BE	F3EB save area
F9BF	F3E9 save area
F9C0-F9C1	Start address for .CO file
F9C2-F9C3	Length of .CO file
F9C4-F9C5	Execute address of .CO file
F9C6-F9C9	Extension
F9CA-F9CB	Data number from LOAD start
F9CC-FA5F	Hook address table
FA60-FA87	LCD reverse attribute table
FA88-FA89	End of file in access file
FA8A	Array flag for valuable
FA8B	The type of floating accumulator
FA8C	Flag for pre-compile
FA8D	Flag for line number in pre-compile
FA8E-FA8F	Text pointer save area
FA90	Character length when TEXT read
FA91	Value type when TEXT read
FA92-FA99	Value when TEXT read

FA9A-FA9B	End of string space
FA9C-FA9D	Stack pointer for string
FA9E-FAB8	Stack area for string
FABC-FABE	Descriptor for string
FABF-FAC0	Max address of free area of string space
FAC1-FAC4	Work area for string process
FAC5-FAC6	TEXT pointer save area for FOR
FAC7-FAC8	ERROR LINE NUMBER when DATA
FAC9	Valuable type in next
FACA	READ, INPUT flag
FACB-FACC	Buffer for pre-compile code
FACD	Line number, Line address flag 00: Line number xx: Line address
FACE-FACF	TEXT address before statement
FAD0-FAD1	STACK pointer for before statement
FAD2-FAD3	Line number in ERROR
FAD4-FAD5	Current line number
FAD6-FAD7	TEXT address in ERROR
FAD8-FAD9	ON ERROR GOTO jump address
FADA	Flag for error trap
FADB-FADC	Work area for calculation
FADD-FADE	Line number when STOP and END
FADF-FAE0	TEXT address for CONT
FAE1-FAE2	ASCII file start address
FAE3-FAE4	CO file start address
FAE5-FAE6	Variable area top address

FAE7-FAE8	Array area top address
FAE9-FAEA	Free area top address
FAEB-FAEC	DATA pointer when READ
FAED-FB06	Variable table
FB07-FB0A	Work for garbage collection
FB0B	Not used
FB0C-FB0D	Work for garbage collection
FB0E-FB11	Not used
FB12-FB13	Work for garbage collection
FB14	Not used
FB15-FB16	Work for garbage collection
FB17-FB18	Not used
FB19	Work for VAL function
FB1A	Work for output
FB1B-FB22	Save area for FAC
FB23	Not used
FB24-FB2B	FAC
FB2C	Flag for FAC
FB2D	Not used
FB2E-FB35	Argument for double calculation
FB36-FB4D	Buffer for number string conversion
FB4E-FB50	Not used
FB51-FB58	Work for double calculation (DIV)
FB59-FB5D	Not used
FB5E-FB60	Work for single calculation (MUL)
FB61	MAX drive number

FB62	MAX files number
FB63-FB64	Top address of file pointer
FB65-FB66	Top address of drive data
FB67-FB68	File buffer address of number zero
FB69	Current Drive number
FB6A-FB6B	Current drive data address
FB6C-FB6D	Current FCB address
FB6E-FB6F	Directory pointer for directory search
FB70-FB71	Directory position in search
FB72	Auto run flag after LOAD
FB73-FB77	Work for disk
FB78-FB80	File name area
FB81-FB89	Old file name in NAME
FB8A	Track number in DSK1$, DSK0$
FB8B	Sector number in DSK1$,DSK0$
FB8C	Flag for program access
FB8D	Flag for program SAVE
FB8E	Disk BUSY flag
FB8F-FB90	Disk error counter
FB91	Read after write flag
FB92	EBCDIC conversion flag
FB93	EBCDIC conversion buffer
FB94	Condition of DISK adapter
FB95	Condition of disk
FB96-FBBF	Stack for initialization
FBC0-FCFF	Screen buffer for before page

FD00-FE3F	Screen buffer
FE40	Flag for XON,XOFF
FE41	Not used
FE42	Flag for X parameter
FE43	Flag for USART
FE44	90H out copy
FE45	Character counter for RS-232C buffer
FE46	Character get position for RS-232C buffer
FE47	Character put pointer for RS-232C buffer
FE48	Character position when error was detected
FE49	Flag for XON,XOFF sequence
FE4A-FE4B	Baud rate table
FE4C	RS-232C receive data mask pattern
FE4D	Key scan interval timer
FE4E	Work for key scan
FE4F-FE5F	Flag for key input
FE60	Shift mode flag for input
FE61	Work for encode of key code
FE62	Repeat counter for key input
FE63-FE67	Work for key input
FE68	Data counter for key input
FE69-FEA8	Buffer for key input
FEA9	Input condition of break character
FEAA	Status of function key display
FEA8-FEB0	Work for cursor display
FEB1-FEB6	Save area for character pattern

FEB7	Status of cursor Bit 7: Cursor display mode ON,OFF Bit 0: Cursor ON,OFF
FEB8	Cursor blink timer
FEB9	Cursor position (Y)
FEBA	Cursor position (X)
FEBB-FEBC	Work for cursor blink
FEBD-FEBE	Work for LCD display
FEBF-FEC0	Data pointer for user define character
FEC1	Disk BASIC or not flag
FEC2	Disk SIO time out timer
FEC3	Disk BASIC boot flag
FEC4-FFC1	RS-232C receive buffer

CHAPTER 1 Notation and Floating Accumulator 2
1.1 Notation of Floating Point Numbers 2
1.2 Floating Point Accumulators and Related Variables.....4

CHAPTER 2 FLOATING POINT INPUT AND OUTPUT.....6
2.1 Input.....6
2.2 Output.....7

CHAPTER 3 FLOATING POINT MOVEMENT.....9
3.1 Single Precision Numbers.....9
3.1.1 Move number from memory [HL] to FAC.....9
3.1.2 Move registers (B,C,D,E) to FAC.....10
3.1.3 Move FAC to registers (B,C,D,E).....11
3.1.4 Get number in registers (B,C,D,E) from memory [HL].....12
3.1.5 Move number from FAC to memory [HL].....13
3.1.6 Move number from [DE] to [HL].....14
3.2 Any Type Number.....15
3.2.2 Move any type value from memory [HL] to FAC.....16
3.2.3 Move any type value from FAC to memory [HL].....17

CHAPTER 4 COMPARISON.....18
4.2 Single Precision.....19
4.3 Double Precision.....20

CHAPTER 5 CONVERSION.....21
5.1 Integer to Single.....21
5.2 Single to Integer.....22
5.3 Single to Double.....23
5.4 Double to Single.....24

CHAPTER 6 Basic Operation.....25
6.1 Integer Arithmetic.....25
6.1.1 Integer addition.....25
6.1.2 Integer Subtraction.....26
6.1.3 Integer Multiplication.....27
6.1.4 Integer Division.....28
6.2 Single Precision Arithmetic.....29
6.2.1 Single Precision Addition.....30
6.2.2 Single Precision Subtraction.....31
6.2.3 Single Precision Multiplication.....32
6.2.4 Single Precision Division.....33
6.3 Double Precision Arithmetic.....34
6.3.1 Double Precision Addition.....35
6.3.2 Double Precision Subtraction.....36
6.3.3 Double Precision Multiplication.....37
6.3.4 Double Precision Division.....38

CHAPTER 7 Mathematical Functions..... 39
7.1 Power..... 9
7.2 EXP.....40
7.3 LOG.....41
7.4 SQR.....42
7.5 SIN.....43
7.6 COS.....44
7.7 TAN.....45
7.8 ATN.....46
7.9 RND.....47

Accumulator,2,4,5,34
Accumulators,4
Added,2,25,30,35
Addition,25,30,35
ADDRESS,6,7,9,10,11,12,13,
14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,30,
31,32,33,35,36,37,38,39,
40,41,42,43,44,45,46,47
Angle,43,44,45,46
Arctangent,46
Area,5,13,14,15,17
ARG,5,19,20,22,34,35,36,37,38
Argument,5,18,19,20,22,25,
26,27,28,30,31,32,33,34,
35,36,37,38,39,40,41,42,
43,44,46
Arithmetic,25,29,34
ATN,46

Base,39,40,41
Basic,25
Bias,2,29
Binary,3
Byte,29
Bytes,2

CALL,22,39,41,42
Can,22
CONDS,23,24
CONIS,22
CONSD,24
CONSI,21
COS,44
COSINE,44

DADD,35
DADH,40
DAH,24
DCOMP,20
DDIV,38
DFAC,4,6,7,20,23,24,34,35,
36,37,38
Division,28,33,38
DMULT,37
DSUB,36

Encoding,4
Equals,18
EXP,40
Exponent,2,3,4,5,8,11,12,
29,32,33,34,39

FAC,4,5,6,7,9,10,11,13,16,
17,18,19,21,22,23,24,25,
26,27,28,29,30,31,32,33,
36,37,38,39,40,41,42,43,
44,45,46,47
FADD,30
FBUFFR,5,7,8
FCOMP,19
FDIV,33
Field,8
FIN,6
Floating,2,3,4,5,6,8,9,29,34
FMULT,32
FOUT,5,7
FPWR,39
FSUB,31

Get,12,22

IADD,25
ICOMP,18
IDIV,28
IMULT,27
Integer,4,5,6,7,8,15,16,
17,18,21,22,25,26,27,28,39
ISUB,26

LOG,41
Logarithm's,40,41
Long,2
LXI,22

Mantissa,2,4,5,11,12,29,
32,33,34,39
MOVFM,9
MOVFR,10
MOVMF,13
MOVRF,11
MOVRM,12
MOVVFM,15
Multiplication,27,32,37

Order,2,3,4,5,11,12,25,
26,27,29,32,33,34,39
Overflow,6,25,26,27,28,
30,31,32,33,35,36,37,38,
39,45

POP,22
Power,39
Precision,2,3,4,5,6,7,9,
10,11,12,13,14,19,20,21,22,
23,24,25,26,27,28,29,30,
31,32,33,34,35,36,37,38,
42

CHAPTER 1 Notation and Floating Accumulator

1.1 Notation of Floating Point Numbers

Floating point numbers (single precision and double precision) are represented in the PC-8300A ROM as follows. The sign of the number is indicated by the first bit of the mantissa, 0 means positive and 1 means negative. The mantissa is 24 bits long for single precision number, or 56 bits long for double precision number. The decimal point is assumed to be to the left Most Significant Bit of the mantissa. The mantissa ranges from 0.5 inclusive to 1 exclusive.

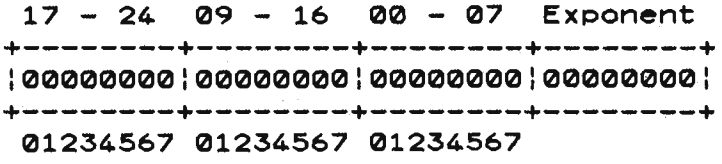
NUMBER=MANTISSA * 2 ^ Exponent

The mantissa is positive, with a one assumed to be where the sign bit is. The sign of the exponent is the first bit of the exponent. The exponent is stored in excess 80H (ie: with a bias of 80H). So, the exponent is a signed 8 bit number with 80H added it. An exponent of zero means the number is zero, the other bytes are ignored.

The above notation is the same notation use by the PC-8000 and PC-8800.

The memory representation of a single precision floating point number is as follows:

Bits 17-24 Low order of the mantissa.
Bits 09-16 Middle order of the mantissa.
Bits 00-07 High order of the mantissa.



Note: Bit 00 of the mantissa is always 1. (Implied)
This bit is also the sign bit.

For example the number 4095 is represented in single precision floating point format as follows:

```
 17 - 24   09 - 16   00 - 07   Exponent
+--------+--------+--------+--------+
|00000000|11110000|01111111|10001100|
+--------+--------+--------+--------+
 01234567 01234567 01234567

    00       F0       7F       8C
```

With the bits laid out in a logical left to right order the binary number would be displayed as follows:

.1111 1111 1111 0000 0000 0000 * 2^0Ch

or

1111 1111 1111. 0000 0000 0000

1.2 Floating Point Accumulators and Related Variables

The PC-8300 ROM maintains a few variables that are used throughout the math package.

1. VALTYP FA8BH (64139D)

VALTYP indicates the type of number stored in the Floating Point Accumulator (FAC or DFAC). The encoding is shown below.

```
[VALTYP]  =2 (Integer Number)
          =4 (Single Precision Number)
          =8 (Double Precision Number)
```

2. DFAC FB24H (64292D)

Double Precision Floating Point Accumulator.

Double precision storage format:

```
[DFAC]    =Lowest order of the mantissa
[DFAC+1]  =        .
[DFAC+2]  =        .
[DFAC+3]  =        .
[DFAC+4]  =        .
[DFAC+5]  =        .
[DFAC+6]  =Highest order of mantissa
[DFAC+7]  =Exponent
```

3. FAC FB28H (64296D)

Floating Point Accumulator.

The floating point accumulator is used for both single precision and integer math routines.

Integer storage format:

```
[FAC]    =Low order of the integer
[FAC+1]  =High order of the integer
```

Single Precision storage format:

```
[FAC]    =Low order of the mantissa
[FAC+1]  =Middle order of the mantissa
[FAC+2]  =High order of the mantissa
[FAC+3]  =Exponent
```

4. ARG FB2EH (64302D)

Second argument storage area for double precision math functions.

```
[ARG]    =Lowest order of the mantissa
[ARG+1]  =        .
[ARG+2]  =        .
[ARG+3]  =        .
[ARG+4]  =        .
[ARG+5]  =        .
[ARG+6]  =        .
[ARG+7]  =Highest order of the mantissa
```

5. FBUFFR FB37H (64311D)

FBUFFR is used by the function FOUT (Convert a floating point number to a printable string) to store the result string. The string is terminated by a null code.

```
[FBUFFR]  =First character of the string
[FBUFFR+1]=Second character of the string
[FBUFFR+2]=        .
[FBUFFR+3]=        .
[FBUFFR+4]=        .
[FBUFFR+5]=        .
[FBUFFR+6]=        .
[FBUFFR+n]=Null code
```

CHAPTER 2 FLOATING POINT INPUT AND OUTPUT

2.1 Input

DESCRIPTION

The FIN routine converts the string representation of a number in [HL] into an internal representation of the number, and leaves it in FAC. The value type is set in VALTYP. Syntax is not checked in this routine. So syntax must be checked in the user's routine, so that at exit, HL points at the last character of the string. The string must be terminated by a null. (hex 00)

NAME	FIN	
ADDRESS	3726H (14118D)	
ENTRY	[HL]	Pointer to top of string
EXIT	[FAC]	Result at this address if Integer or single precision.
	or	
	[DFAC]	Result at this address if Double precision number.
	[VALTYP] 2	(Integer)
	[VALTYP] 4	(Single)
	[VALTYP] 8	(Double)
REGISTERS ALTERED	All	

If an overflow occurs,the control is transferred to the error handling.

EXAMPLE:

Input String	At Exit
"1.5"	HL points to "5"
"1.5E+10"	HL points to "0"
"ABC"	HL points to "C"

Note: All strings must be null terminated.

2.2 Output

DESCRIPTION

The FOUT routine converts the number in FAC or DFAC to a string representation, and leaves it in FBUFFR. The format is specified in registers A,B and C.

If the format is fixed and the length of digits in the number exceeds the number of places specified by register C, "%" is output before the number and the original contents of the FAC is lost. The terminator of output string is 00H.

NAME	FOUT	
ADDRESS	38A8H (14504D)	
ENTRY		
	[FAC]	If Integer or Single Precision.
	or	
	[DFAC]	If Double Precision.
	[VALTYP] 2	(Integer)
	[VALTYP] 4	(Single)
	[VALTYP] 8	(Double)

[A] bit 7=0 Output free format

7=1 Output fixed format

bit 6=1 Group the digits in the integer part of the number into groups of three and separate the groups by commas.

bit 5=1 Fill the leading spaces in the field with asterisks.

bit 4=1 Output with a floating dollar sign.

bit 3=1 Output sign of a positive number.

bit 2=1 Output the sign of the number after the number.

bit 1 Unused

bit 0=1 Output in floating point notation.

=0 Output in fixed point notation.

[B] The number of places in the field to the left of the decimal point.

[C] The number of places in the field to the right of the decimal point (includes the decimal point, but does not include the 4 positions for the exponent).

EXIT [FBUFFR] Character

REGISTERS ALTERED All

CHAPTER 3 FLOATING POINT MOVEMENT

3.1 Single Precision Numbers

3.1.1 Move number from memory [HL] to FAC

NAME	MOVFM	
ADDRESS	31D0H (12752D)	
ENTRY	[HL]	Pointer to single precision number
EXIT	[FAC]	Single Precision Number.
REGISTERS ALTERED	[B],[C],[D],[E],[H],[L]	

3.1.2 Move registers (B,C,D,E) to FAC

NAME	MOVFR	
ADDRESS	31D3H (12755D)	
ENTRY	None	
EXIT	[FAC]	Single precision number
REGISTERS ALTERED	[D],[E]	

3.1.3 Move FAC to registers (B,C,D,E)

NAME	MOVRF	
ADDRESS	31DEH (12766D)	
ENTRY	[FAC]	Single Precision Number.
Exit	[B]	Exponent
	[C]	High order of Mantissa
	[D]	Middle order of Mantissa
	[E]	Low order of Mantissa
REGISTERS ALTERED	[B],[C],[D],[E],[H],[L]	

3.1.4 Get number in registers (B,C,D,E) from memory [HL]

NAME	MOVRM	
ADDRESS	31E1H (12769D)	
ENTRY	[HL]	Pointer to single precision number
Exit	[B]	Exponent
	[C]	High order of Mantissa
	[D]	Middle order of Mantissa
	[E]	Low order of Mantissa
REGISTERS ALTERED	[B],[C],[D],[E],[H],[L]	

3.1.5 Move number from FAC to memory [HL]

NAME	MOVMF	
ADDRESS	31EAH (12778D)	
ENTRY	[HL]	Pointer to memory area for single precision number
EXIT	[HL]	Single precision number
REGISTERS ALTERED	[A],[B],[D],[E],[H],[L] [HL]=[HL]+4	

3.1.6 Move number from [DE] to [HL]

NAME	MOVE	
ADDRESS	31EDH (12781D)	
ENTRY	[DE]	Pointer to single precision number
	[HL]	Pointer to memory area for single precision number
EXIT	[HL]	Single precision number
REGISTERS ALTERED	[A],[B],[D],[E],[H],[L] [HL]=[HL]+4	

3.2 Any Type Number

3.2.1 Move any type value from [DE] to [HL]

NAME	MOVVFM	
ADDRESS	31F2H (12786D)	
ENTRY	[DE]	Pointer to memory area for value
	[VALTYP]	2 (Integer) 4 (Single) 8 (Double)
EXIT	[HL]	Value
REGISTERS ALTERED	[A],[B],[D],[E],[H],[L]	

3.2.2 Move any type value from memory [HL] to FAC

NAME	VMOVAF	
ADDRESS	3218H (12834D)	
ENTRY	[HL]	Pointer to value
	[VALTYP]	2 (Integer)
		4 (Single)
		8 (Double)
EXIT	[FAC]	Value
REGISTERS ALTERED	[A],[B],[D].[E],[H],[L]	

3.2.3 Move any type value from FAC to memory [HL]

NAME	VMOVMF	
ADDRESS	3221H (12833D)	
ENTRY	[HL]	Pointer to memory area for value
	[VALTYP]	2 (Integer) 4 (Single) 8 (Double)
EXIT	[FAC]	Value
REGISTERS ALTERED	[A],[B],[D],[E],[H],[L]	

CHAPTER 4 COMPARISON

4.1 Integer

DESCRIPTION

The ICOMP routine compares two integer numbers in HL and FAC, and leave the result in A.

NAME	ICOMP	
ADDRESS	3259H (12889D)	
ENTRY		
	[DE]	First argument
	[HL]	Second argument
EXIT		
	A= 1	If the first argument is less than Second argument.
	A= 0	If the first argument equals Second argument.
	A=-1	If the first argument is greater than Second argument.
REGISTERS ALTERED	[A]	

4.2 Single Precision

DESCRIPTION

The FCOMP routine compares two single precision numbers in registers and FAC, and leave the result in A.

NAME	FCOMP	
ADDRESS	322EH (12846D)	
ENTRY		
	[B],[C],[D],[E]	First argument
	[FAC]	Second argument
EXIT		
	A=1	if first arg.<Second arg.
	A=0	if first arg.=Second arg.
	A=-1	if first arg.>Second arg.
REGISTERS ALTERED	A,H,L	

4.3 Double Precision

DESCRIPTION

The DCOMP routine compares two double precision numbers in ARG and DFAC and leaves the result in A.

NAME	DCOMP	
ADDRESS	326FH (12911D)	
ENTRY		
	[ARG]	First argument
	[DFAC]	Second argument
EXIT	A=1	if first arg.<Second arg.
	A=0	if first arg.=Second arg.
	A=-1	if first arg.>Second arg.
REGISTERS ALTERED	All	

CHAPTER 5 CONVERSION

5.1 Integer to Single

DESCRIPTION

The CONSI routine converts the integer in FAC to a single precision number, and leaves it in FAC.

NAME	CONSI	
ADDRESS	32EDH (13037D)	
ENTRY	[FAC]	Integer
EXIT	[FAC] [VALTYP]	Single precision number 4
REGISTERS ALTERED	All	

5.2 Single to Integer

DESCRIPTION

The CONIS routine truncates the single precision number in FAC and converts it to an integer, and leaves it in FAC. If the argument is too big (arg. < -32768 or arg. > 32767), it can not be converted to an integer. So the control is transferred to the error handler.

NAME	CONIS	
ADDRESS	32ADH	
ENTRY	[FAC]	Single precision number
EXIT	[FAC]	Integer
	[VALTYP]	2
REGISTERS ALTERED	All	

Sample program

```
          .
          .
          .
          LXI  H,NORMAL       ; Get return address
          PUSH H              ; Push it on stack
          CALL CONIS          ; Call CONIS
ERROR:
          POP  H              ; Error handling routine
                              ; Reset stack pointer
          .
          .
          .
NORMAL:
          .                   ; Normal return
          .
          .
```

5.3 Single to Double

DESCRIPTION

The CONDS routine converts the single precision number in FAC to a double precision number and leaves it in DFAC.

NAME	CONDS	
ADDRESS	3304H (13060D)	
ENTRY	[FAC]	Single precision number
EXIT	[DFAC] [VALTYP]	Double precision number 8
REGISTERS ALTERED	[A],[H],[L]	

5.4 Double to Single

DESCRIPTION

The CONDS routine converts the double precision number in DFAC to a single precision number and leaves it in FAC.

NAME	CONSD	
ADDRESS	32DAH (13018D)	
ENTRY	[DFAC]	Double precision number
EXIT	[FAC]	Single precision number
	[VALTYP]	4
REGISTERS ALTERED	All	

CHAPTER 6 Basic Operation

6.1 Integer Arithmetic

Integer arithmetic is performed with 16 bit signed integers. The values range from +32767(7FFFH) to -32768(FFFFH).

6.1.1 Integer addition

DESCRIPTION

First argument is added to the second argument. Result is in the "FAC". When an overflow occurs, the result is converted to a single precision number and placed in the FAC, and the VALTYP is set to 4.

NAME	IADD
ADDRESS	3403H (13315D)
ENTRY	[DE] First argument [HL] Second argument
EXIT	
REGISTERS ALTERED	All

Condition: No overflow

[FAC]=[HL]=Result in integer.

Integer storage format:

[FAC] =Low order of the integer
[FAC+1] =High order of the integer

[VALTYP] =2 Integer

Condition: Overflow

[VALTYP] =4 Single Precision

[FAC] =Result in single precision.

6.1.2 Integer Subtraction

DESCRIPTION

The second argument is subtracted from the first. The result is left in the "FAC". When an overflow occurs, the result is converted to a single precision number and left in the FAC , and the VALTYP is set to 4.

NAME	ISUB
ADDRESS	33F7H (13303D)
ENTRY	[DE] First argument [HL] Second argument
EXIT	
REGISTERS ALTERED	All

Condition: No overflow

[FAC]=[HL]=Result in integer.

Integer storage format:

[FAC] =Low order of the integer
[FAC+1] =High order of the integer

[VALTYP] =2 Integer

Condition: Overflow

[VALTYP] =4 Single Precision

[FAC] =Result in single precision.

6.1.3 Integer Multiplication

DESCRIPTION

The first argument is multiplied by the second. The result is left in the "FAC". When overflow occurs, the result is converted to a single precision number and left in the FAC, and the VALTYP is set to 4.

Name	IMULT
Address	3423H (13347D)
ENTRY	[DE] First argument
	[HL] Second argument
EXIT	

REGISTERS ALTERED All

Condition: No overflow

[FAC]=[HL]=Result in integer.

Integer storage format:

[FAC]	=Low order of the integer
[FAC+1]	=High order of the integer

[VALTYP] =2 Integer

Condition: Divide by zero.

Control is transferred to the error handler.

6.1.4 Integer Division

DESCRIPTION

The first argument is divided by the second. The quotient is left in [DE] and the remainder is left in [HL]. If the divisor is 0, a "Division by zero" error occurs, and control is transferred to the error handler.

NAME IDIV
ADDRESS 347AH (13434D)

ENTRY [DE] First argument (Dividend)
[HL] Second argument (Divisor)

EXIT
[DE] Remainder
[HL] Quotient

REGISTERS ALTERED All

Condition: Overflow

[VALTYP] =4 Single Precision

[FAC] =Result in single precision.

*******This function does not calculate the remainder correctly.

6.2 Single Precision Arithmetic

Single precision arithmetic is performed on four byte numbers. The numbers have a 3 byte mantissa and a 1 byte exponent with a bias of 80h.

Single Precision storage format:

```
[FAC]     =Low order of the mantissa
[FAC+1]   =Middle order of the mantissa
[FAC+2]   =High order of the mantissa
[FAC+3]   =Exponent
```

The memory representation of a single precision floating point number is as follows:

```
Bits 17-24      Low order of the mantissa.
Bits 09-16      Middle order of the mantissa.
Bits 00-07      High order of the mantissa.
```

```
 17 - 24  09 - 16  00 - 07  Exponent
+--------+--------+--------+--------+
|00000000|00000000|00000000|00000000|
+--------+--------+--------+--------+
 01234567 01234567 01234567
```

6.2.1 Single Precision Addition

DESCRIPTION

The second argument is added to the first argument. The result is stored in FAC.

NAME	FADD
ADDRESS	2EBBH (11963D)
ENTRY	[HL] Address of the first argument.
	[FAC] Second argument.
EXIT	

REGISTERS ALTERED All

Condition: No Overflow

[FAC] Result

Condition: Overflow

Control transferred to the error handler.

6.2.2 Single Precision Subtraction

DESCRIPTION

Subtracts the second argument from the first argument, and leaves the result in FAC.

NAME FSUB
ADDRESS 2EC1H (11969D)

ENTRY

[HL] Address to first argument.
[FAC] Second argument.

EXIT

REGISTERS ALTERED All

Condition: No Overflow

[FAC] Result

Condition: Overflow

Control transferred to the error handler.

6.2.3 Single Precision Multiplication

DESCRIPTION

Multiplies first argument by second argument, and leaves the result in FAC.

NAME FMULT
ADDRESS 3040H (12352D)
ENTRY

First argument storage format.

[B]	Exponent
[C]	High order of Mantissa
[D]	Middle order of Mantissa
[E]	Low order of Mantissa

Second argument storage format.

[FAC]	Low order of the mantissa
[FAC+1]	Middle order of the mantissa
[FAC+2]	High order of the mantissa
[FAC+3]	Exponent

EXIT

REGISTERS ALTERED All

Condition: No Overflow

[FAC] Result

Condition: Overflow

Control transferred to the error handler.

6.2.4 Single Precision Division

DESCRIPTION

Divides the first argument by second argument,and leaves the result in FAC.

NAME FDIV
ADDRESS 30A5H (12453D)
ENTRY

First argument storage format. (Dividend)

[B]	Exponent
[C]	High order of Mantissa
[D]	Middle order of Mantissa
[E]	Low order of Mantissa

Second argument storage format. (Divisor)

[FAC]	Low order of the mantissa
[FAC+1]	Middle order of the mantissa
[FAC+2]	High order of the mantissa
[FAC+3]	Exponent

EXIT

REGISTERS ALTERED All

Condition: No Overflow

[FAC] Result

Condition: Overflow or divide by zero.

Control transferred to the error handler.

6.3 Double Precision Arithmetic

When performing double precision arithmetic the arguments must be in the following format. The result will be stored in the Double Precision Floating Point Accumulator. (DFAC)

The first argument is stored in the Double Precision Floating Point Accumulator. (DFAC) FB24h (64292d)

```
[DFAC]    =Lowest order of the mantissa
[DFAC+1]  =          .
[DFAC+2]  =          .
[DFAC+3]  =          .
[DFAC+4]  =          .
[DFAC+5]  =          .
[DFAC+6]  =Highest order of mantissa
[DFAC+7]  =Exponent
```

The second argument is stored in the Double Precision argument. (ARG) FB2Eh (64302d)

```
[ARG]     =Lowest order of the mantissa
[ARG+1]   =          .
[ARG+2]   =          .
[ARG+3]   =          .
[ARG+4]   =          .
[ARG+5]   =          .
[ARG+6]   =Highest order of mantissa
[ARG+7]   =Exponent
```

6.3.1 Double Precision Addition

DESCRIPTION

The first argument is added to the second argument.

NAME DADD
ADDRESS 34F8H (13560D)

ENTRY

[DFAC]	First argument
[ARG]	Second argument

EXIT

REGISTERS ALTERED All

Condition: No Overflow

[DFAC] Result

Condition: Overflow

Control transferred to the error handler.

6.3.2 Double Precision Subtraction

DESCRIPTION

Subtracts the second argument from the first argument, and leaves the result in FAC.

NAME DSUB
ADDRESS 34F1H (13553D)

ENTRY

[DFAC]	First argument
[ARG]	Second argument

EXIT

REGISTERS ALTERED All

Condition: No Overflow

[DFAC] Result

Condition: Overflow

Control transferred to the error handler.

6.3.3 Double Precision Multiplication

DESCRIPTION

Multiplies the first argument by second argument, and leaves the result in FAC.

NAME	DMULT	
ADDRESS	3639H (13881D)	
ENTRY		
	[DFAC]	First argument
	[ARG]	Second argument

EXIT

REGISTERS ALTERED All

Condition: No Overflow

[DFAC] Result

Condition: Overflow

Control transferred to the error handler.

6.3.4 Double Precision Division

DESCRIPTION

Divides the first argument by second argument, and leaves the result in FAC.

NAME	DDIV	
ADDRESS	3691H (13969D)	
ENTRY		
	[DFAC]	First argument (Dividend)
	[ARG]	Second argument (Divisor)

EXIT

REGISTERS ALTERED All

Condition: No Overflow

[DFAC] Result

Condition: Overflow or Divide by zero.

Control transferred to the error handler.

CHAPTER 7 Mathematical Functions

7.1 Power

DESCRIPTION

Calculates the formula Y^x where Y is the base and x is the exponent.

NAME FPWR

ADDRESS 3D5EH (15710D)

ENTRY

First argument.

[B]	Exponent
[C]	High order of Mantissa
[D]	Middle order of Mantissa
[E]	Low order of Mantissa

Second argument.

[FAC] Base

EXIT

Condition: No error

[FAC] Result

Condition: Error

Control is transferred to the Error Handler.

1 - Illegal Function Call:

Base is negative and exponent is not an integer.

2 - Divide By Zero:

Base is Zero and exponent is negative.

3 - Overflow:

Number is out of range.

7.2 EXP

DESCRIPTION

Compute the Natural Logarithm's base value.

NAME	EXP
ADDRESS	3DADH (15789D)
ENTRY	[FAC] Argument
EXIT	
REGISTERS ALTERED	All

Condition: No error

[FAC] Result

Condition: Error

The value in [FAC] exceeded 87.33655. Control is transferred to the error handler.

7.3 LOG

DESCRIPTION

Compute the Natural Logarithm's base value.

NAME LOG

ADDRESS 2FFCH (122840D)

ENTRY [FAC] Argument

EXIT

REGISTERS ALTERED All

Condition: No error

[FAC] Result

Condition: Error

Illegal Function Call:

The argument is negative or Zero. Control is transferred to the error handler.

7.4 SQR

DESCRIPTION

Compute the square root of a single precision number.

NAME SQR

ADDRESS 3D4DH (15693D)

ENTRY [FAC] Argument

EXIT

REGISTERS ALTERED All

Condition: No error

[FAC] Result

Condition: Error

Illegal Function Call:

The argument is negative. Control is transferred to the error handler.

7.5 SIN

DESCRIPTION

Computes the SIN of an angle. The angle must be given in radians.

NAME	SIN	
ADDRESS	3EC3H (16067D)	
ENTRY	[FAC]	Argument
EXIT	[FAC]	Result
REGISTERS ALTERED	All	

7.6 COS

DESCRIPTION

Computes the COSINE of an angle. The angle must be given in radians.

NAME	COS	
ADDRESS	3EBDH (16061D)	
ENTRY	[FAC]	Argument
EXIT	[FAC]	Result
REGISTERS ALTERED	All	

7.7 TAN

DESCRIPTION

Computes the tangent of an angle. The angle must be given in radians.

NAME TAN

ADDRESS 3F5EH (16222D)

EXIT

REGISTERS ALTERED All

Condition: No error

[FAC] Result

Condition: Overflow

Control transferred to the error handler.

7.8 ATN

DESCRIPTION

Computes arctangent of an angle. The result is given in radians between -pi/2 and pi/2.

NAME	ATN	
ADDRESS	3F73H (16243D)	
ENTRY	[FAC]	Argument
EXIT	[FAC]	Result
REGISTERS ALTERED	All	

7.9 RND

DESCRIPTION

Generates a random number between 0 and 1.

NAME	RND	
ADDRESS	3E4AH (15946D)	
ENTRY	[FAC]<0	A new sequence of random number is started
	[FAC]=0	The last random number generated is returned
	[FAC]>0	Next random number generated is returned
EXIT	[FAC]	Result
REGISTERS ALTERED	All	

DESCRIPTION OF BASIC PROGRAM FILE HANDLING ROUTINES

CHAPTER 1. Overview

CHAPTER 2. How to know the information of BASIC file
2.1 Procedure to know the information of BASIC file
2.2 Sample program

CHAPTER 3. Saving a BASIC program in RAM as ".BA" file
3.1 Procedure to save BASIC program
3.2 Sample program

CHAPTER 4. Subroutine for loading and saving
4.1 SRCBAS
4.2 SCNEMP
4.3 LDIRSB
4.4 SETNAM
4.5 LINKER
4.6 LINKER1
4.7 MAKBAS
4.8 RUNC
4.9 CHKREG
4.10 REST00
4.11 MAKBAT
4.12 Variables

CHAPTER 1 OVERVIEW

This document has been prepared to provide information about the BASIC binary program file (".BA" files) handling. The documentation is divided into three parts, method to obtain information about a certain BASIC program file (such as address of the file in memory, length of the program, and so on), saving a portion of memory as a BASIC program file and miscellaneous routines used for BASIC program file handling.

Similar to other ROM routines, it is the programmer's responsibility to make special error handling routines to prevent control from going back to the ROM error handler. For detailed information of the error handling, refer to Chapter XX, "Description of the ROM routines in the PC-8300"

CHAPTER 2 BASIC FILE SET-UP

2.1 Procedure to Know The Information of BASIC File

1. Set Up File Name

```
[FILNAM]      <--- 1st character in the file name
   .
   .
   .
[FILNAM+5]    <--- 6th character in the file name
[FILNAM+6]    <--- "B"
[FILNAM+7]    <--- "A"
[FILNAM+8]    <--- " "
```

2. Search For The File

   Search for the file we want to know the information of and get the starting address of the text. If the file does not exists,transfer control to the error handler.

```
CALL SRCBAS          ; Search for the file
JZ  FILNF          ;If it does not exists,return
                     ; "File Not Found" error.
```

3. Get Ending Address

```
CALL LINKER
```

4. Compute Length

2.2 Sample Program

```
; Load the BASIC program file whose name is "SAMPLE.BA"
; If RUN flag is on, start execution
; At exit, [DE]      start address
;          [HL]      Length
LOAD:
          LXI   H,MYFILE        ; Set up file name to FILNAM
          LXI   D,FILNAM        ;
          LXI   B,0006D         ;
          CALL  LDIRSB          ;
          LXI   H,FILNAM+6      ;
          MVI   M,"B"           ; And extension
          INX   H               ;
          MVI   M,"A"
          INX   H
          MVI   M," "           ;
          CALL  SRCBAS          ; Search the file
                                ; Start address is hold in DE
          JZ    FILNF           ; If do not exists, "file not
                                  found" error
          PUSH  D               ; Save start address of BASIC file
          CALL  LINKER1         ; Get end address of BASIC file
                                  into HL
          POP   D               ; Get start address
          MOV   A,L             ; Compute length
          SUB   E               ; (HL:=HL-DE+1)
          MOV   L,A             ;
          MOV   A,H             ;
          SBB   D               ;
          MOV   H,A             ;
          INX   H               ;
          RET                   ; All done
FILNF:
           .                    ; File not found error
           .                    ; Put your error handling routine
                                  here
MYFILE:
          DB    "SAMPLE"        ; Name of the file to be load
```

CHAPTER 3 SAVING A BASIC PROGRAM IN RAM AS A ".BA" FILE

In this chapter, the method to save a BASIC program that resides in memory into RAM file as a ".BA" file is described. The program should be written in intermediate language. The routines described later in this chapter save the contents of a specified portion of memory exactly as they are, just appending some control information. The control information consists of two words, load address and length.

Note current non-registered program is lost.

3.1 Procedure To Save A BASIC Program

To save a BASIC program in memory as a ".BA" file, follow the steps below. The address of the individual routines used in this procedure will be described later in this chapter.

Step 1

Load a BASIC program in RAM as a non-registered program

1. Reset all of the variables, and update some pointers
2. Make the largest hole possible before the ASCII files
3. Update some pointers
4. Transfer the program
5. Delete the excess space, and update some pointers

Step 2

Save the program as a ".BA" file

1. Set up the file name

   Set up the file name in FILNAM, this is the same step used to open an ASCII file except the extension should be "BA" in the case of BASIC program save. Body of the file name (ie: string that precedes the extension) should be less than or equal to 6 characters long. If it is less than 6 characters long, the rest should be padded with spaces (20H).

```
[FILNAM]      <--- 1st character of the file name
   .
   .
   .
[FILNAM+5]    <--- 6th character of the file name
[FILNAM+6]    <--- "B"
[FILNAM+7]    <--- "A"
[FILNAM+8]    <--- " "
```

2. Check if the current program is registered or not

```
CALL CHKREG          ; Is current program registered?
JNZ  FCERR           ; Yes error
```

3. Search for a file with the same name

Search the directory for a file with the same name as the one that we want to save. If it exists, delete it. Obviously you can abort saving the new one, instead of deleting old one.

```
CALL SRCBAS          ; Search directory for the file
                     ; whose name is in FILNAM
CNZ  KILBAS          ; Kill old one if exist
```

4. Fix up directory structure

Refer to the "Description of ROM routines in PC-8300A" for detail of LNKFIL routine.

```
CALL LNKFIL          ; Fix up directory structure
```

5. Search directory for empty (free) slot

To register the file, make sure there is free slot in the directory and remember the location of the slot. If no free slot is available, abort saving.

```
CALL SCNEMP          ; search for empty slot
Save address of directory
```

6. Put file name into directory

Set file name and attributes into the directory slot gained by the SCNEMP call. New Basic files are stored just below the lowest "DO" file, at the top of the Basic file storage area.

```
[HL] <--- Address of directory slot gained by SCNEMP
          call
[DE] <--- Top address of BASIC text (ie: current
          TXTTAB)
[A]  <--- 80H
          This is attribute for all ".BA" files
CALL SETNAM          ; Put name into directory
```

7. Make the Basic File

```
CALL MAKBAS
```

Make a hole at ASCTAB (the end of the ".BA" files), for the null-basic-file, and pad the hole with spaces (20H).

3.2 Sample Program

```
; This sample program saves a BASIC program as a BASIC file
; "SAMPLE.BA"and  loads BASIC program in RAM as a
; non-registered program. The information of the text is in
; SOURCE and LENGTH
LOAD:
          CALL SCCPTR          ; Get rid of pointers
          CALL LNKFIL          ; Link all files
          LHLD NULDIR+1        ;
          SHLD TXTTAB          ;
          LXI  H,NULDIR        ;
          SHLD DIRPNT          ;
          LHLD VALTAB          ; Reset variables
          SHLD ARYTAB          ;
          SHLD STREND          ;
          CALL MAKBAT          ; Make a possible largest hole at
                               ; current ; ASCTAB-1.
          LHLD ASCTAB          ; Update ASCTAB
          DAD  B               ;
          SHLD ASCTAB          ;
          LHLD ASCTAB          ; Address of upper limit
          XCHG                 ;
          LHLD LENGTH          ; Length of text
          SHLD TEMP            ;
          LHLD SOURCE          ; Get top address of text that is
                               ; to be loaded
          MOV  B,H             ;
          MOV  C,L             ;
          LHLD TXTTAB          ; Load from TXTTAB
LOAD1:
          LDAX B               ; Get a byte from source
          INX  B               ; and increment its pointer
          MOV  M,A             ;
          INX  H               ;
          CALL COMPARE         ; Compare HL and DE
          JNC  OMERR           ;
          PUSH H               ; Is end of TEXT?
          LHLD TEMP            ;
          DCX  H               ;
          MOV  A,H             ;
          ORA  L               ;
          JZ   LOAD2           ; Yes, end of text
          SHLD TEMP            ; No, transfer next byte
          POP  H               ;
          JMP  LOAD1           ;
LOAD2:
          POP  H               ;
          CALL LINKER          ; Search an end of loaded program,
                               ; and get it in HL
          INX  H               ;
          XCHG                 ;
          LHLD ASCTAB          ;
          XCHG                 ;
          CALL REST00          ; Delete from HL to DE-1
```

```
; Save BASIC program in memory into RAM file.
; The file is named as "SAMPLE.BA"
SAVE:
          LXI   H,0000H          ; Remember current  stack  pointer
                                 ; value preparing to error
          DAD   SP               ; In case of an error that
                                   transfers
                                 ; control to BASIC's error handling
                                 ; routine,we need use  special
                                 ; error trapping to gain
                                   controlback
                                 ; from it, and reset stack pointer
          SHLD  MYSTACK          ;
          LXI   H,FLERR          ; Force control to come back to me
                                 ; in case of an error
          SHLD  ERRJMP           ; Activate error trapping
          LXI   H,MYFILE         ; Set up file name to FILNAM
          LXI   D,FILNAM         ;
          LXI   B,00006D         ;
          CALL  LDIRSB           ; Copy 6 bytes from MYFILE into
                                   FILNAM
          LXI   H,FILNAM+6       ; Set up extension
          MVI   M,"B"            ; It should always be "BA"
          INX   H                ;
          MVI   M,"A"            ;
          INX   H                ;
          MVI   M," "            ; Extension field is 3 bytes long,
                                 ; so pad with a space
          CALL  CHKREG           ; Is current program registered?
          JNZ   FCERR            ; Yes error
          CALL  SRCBAS           ; Search directory for a file of
                                 ; same name as we want to create
                                   now
          CNZ   KILBAS           ; If exist, delete old one
          CALL  LNKFIL           ; Fix up directory structure
          CALL  SCNEMP           ; Scan empty slot in directory
          SHLD  DIRPNT           ;
          MVI   A,10000000B      ; Is file attribute
          XCHG                   ;
          LHLD  TXTTAB           ; TXTTAB holds loc where BASIC prog
          XCHG                   ; program is saved
          CALL  SETNAM           ; Set up directory
          CALL  MAKBAS           ; Make a room for non reg program
RETURN:
          LXI   H,0000H          ; All done
          SHLD  ERRJMP           ; Reset error trapping
          RET                    ;
FLERR:
          LHLD  MYSTACK          ; Error trap code. Control comes
                                 ; here when directory is full at
                                 ; SCNEMP call.
          SPHL                   ; Reset stack pointer
            .                    ; Put your own error handling code
                                   here
            .
```

```
            .
            JMP   RETURN          ;
FCERR:
              .                   ; Illegal function call error
              .
              .                   ; Put your error handler here
            JMP   RETURN          ;
OMERR:
              .                   ; Out of memory error
              .                   ; Put your error handler here
              .                   ;
            JMP   RETURN          ;
COMPARE:
            MOV   A,H             ;
            SUB   D               ;
            RNZ                   ;
            MOV   A,L             ;
            SUB   E               ;
            RET                   ;
MYFILE:
            DB    "SAMPLE"        ; File name
MYSTACK:
            DS    02D             ; Stack pointer value is stored
                                  ; here to reset it in case of error
SOURCE:
            DW    0C000H          ; Top address of text
LENGTH:
            DW    0100H           ; Length of text
TEMP:
            DS    02D             ; Temporary storage
```

CHAPTER 4 SUBROUTINE FOR LOADING AND SAVING

4.1 SRCBAS

NAME SRCBAS

ADDRESS 2292H (8850D)

ENTRY File name should be set up in FILNAM

EXIT Same as SRCNAM (See "Description of the ROM routines in PC-8300")

REGISTERS ALTERED All

DESCRIPTION

SRCBAS searches the directory for a ".BA"file. The name of the file to be searched for should be setup in FILNAM.

4.2 SCNEMP

NAME	SCNEMP	
ADDRESS	22D3H (8915D)	
ENTRY	None	
EXIT	[HL]	Address of empty directory slot
REGISTERS ALTERED	[A],[B],[C],[H],[L]	

DESCRIPTION

SCNEMP is used to search the directory for an empty slot. If an empty slot does not exist, the control is transferred to BASIC's ROM routine. The user program must be take care of this.

4.3 LDIRSB

NAME	LDIRSB	
ADDRESS	6C78H (27768D)	
ENTRY	[HL]	Source address
	[DE]	Destination address
	[BC]	Length
EXIT	None	
REGISTERS ALTERED	All	

DESCRIPTION

Refer to Chapter XX, "Description of the ROM routines in the PC-8300A" for detailed of the LDIRSB routine.

4.4 SETNAM

NAME	SETNAM	
ADDRESS	2435H (9269D)	
ENTRY	[FILNAM]	File name
	[A]	Directory flag
	[DE]	Top address of file
	[HL]	Address of directory
EXIT	None	
REGISTERS ALTERED	[A],[B],[H],[L]	

DESCRIPTION

SETNAMis used to put the filename and file attributes (directory flag) into a slot in directory. The slot is searched for by the SCNEMP call.

4.5 LINKER

NAME LINKER
ADDRESS 0714H (1812D)
ENTRY The beginning of the BASIC text should be set up in TXTTAB
EXIT [HL] The end address of BASIC text
REGISTERS ALTERED All
DESCRIPTION

The LINKER goes through the BASIC program storage, and fixes all of the link pointers.

4.6 LINKER1

NAME LINKER1
ADDRESS 0718H (1816D)
ENTRY [DE] The beginning of BASIC text
EXIT [HL] The end address of BASIC text
REGISTERS ALTERED All
DESCRIPTION

Same as LINKER.

4.7 MAKBAS

NAME MAKBAS
ADDRESS 23D0H (9168D)
ENTRY None
EXIT None
REGISTERS ALTERED All
DESCRIPTION

MAKBAS makes a hole at ASCTAB (end of".BA" files) for null-basic-file,and pad 20H. (means end of BASIC text) at the hole.

4.8 RUNC

NAME RUNC
ADDRESS 3FF5H (16373D)
ENTRY None
EXIT None
REGISTERS ALTERED All including Stack Pointer
DESCRIPTION

This routine initializes the variable and array space by reseting ARYTAB (The end of simple variable space) and STREND (The end of array storage) It then initializes the stackpointer and resets some flags (the data on stack is lost except the return address ([SP],[SP+1]).

4.9 CHKREG

NAME CHKREG
ADDRESS 226AH (8810D)

ENTRY None
EXIT Zero is clear, if the current program is registered

4.10 REST00

NAME REST00
ADDRESS 28CBH (10443D)
ENTRY [HL] Start address
[DE]-1 End address
EXIT None
DESCRIPTION

The REST00 fixes up files to delete the useless area made by MAKBAT.

4.11 MAKBAT

NAME MAKBAT
ADDRESS 6518H
Purpose Make a possible largest hole
ENTRY Start address of the hole should be set up in ASCTAB
EXIT [BC] holds its length
DESCRIPTION

The MAKBAT make a possible largest hole at [ASCTAB]. The size of the hole is calculate by following formula. SIZE=[SP]-([STREND]+200D)

4.12 Variables

1. DIRPNT F979H (63865D)
   Points to directory of current BASIC program

2. FILNAM FB78H (64376D)
   The filename is stored here.It is 9 bytes long, the first 6 bytes are used to store the body of the file name, and the rest for the extension.If the length of the body and/or extension is less than the maximum, the space should be filled with 20H.

3. TXTTAB F45DH (62557D)
   Pointer to the beginning of the text.

4. NLONLY FB8CH (64396D)
   Flag to show if the program is loading or not. Non-zero if loading program.

5. ASCTAB FAE1H (64225D)

Pointer to the start of the ASCII files

6. VALTAB FAE5H (64229D)
   Pointer to the start of the simple variable space

7. ARYTAB FAE7H (64231D)
   Pointer to the beginning of the array table

8. STREND FAE9H (64233D)
   End of storage in use

9. NULDIR F870H (63600D)
   Directory for non-registered program
   NULDIR+1 holds its address

DESCRIPTION OF MACHINE CODE FILE HANDLING ROUTINES IN PC-8300A

CHAPTER 1. Overview

CHAPTER 2. Saving machine code program into RAM file
2.1 Procedure to save machine code program
2.2 Sample program

CHAPTER 3. Loading a machine code file
3.1 Machine language program load routine
3.2 Procedure to load a machine code program

3.3 Sample program
CHAPTER 4. Subroutine for loading and saving
4.1 SRCCOM
4.2 SCNEMP
4.3 LDIRSB
4.4 SETNAM
4.5 MAKHOL
4.6 VARIABLES

CHAPTER 1 OVERVIEW

This document has been prepared to provide information about machine language files (".CO" files) handling. The documentation is divided into three parts: loading a machine code file into memory, saving a machine code in memory into a RAM file and miscellaneous routines used for machine code file handling.

Similar to other ROM routines, it is the programmer's responsibility to make special error handling to prevent control from going back to the ROM error handler. For detailed information of the error handling, refer to Chapter XX, "Description of the ROM routines in the PC-8300A".

CHAPTER 2 SAVING A MACHINE CODE PROGRAM INTO A RAM FILE

In this chapter, the method to save a machine code program that resides in memory into a RAM file as a ".CO" file is described. The object to be saved does not need to be a machine code program, it can be a binary data file. The routines described later in this chapter save the contents of a specified portion of memory exactly as they are, just appending some control information.

The control information consists of three words; load address, length and execution address. The contents of the file is always loaded back to the location where the contents (machine code program) were located when saved. The load address in the file contains the location. Note that this is not the address of the file itself.

2.1 Procedure to Save Machine Code Program

To save a machine code program in memory as a ".CO" file, use the steps below. Address of individual routines used in this chapter will be described later in this document.

1. Set Up The File Name

Set up the file name in FILNAM. This is the same step used to open an ASCII file except the extension should be "CO" in the case of a machine code program save. Body the of file name (ie:string that precedes the extension) should be less than or equal to 6 characters long. If it is less than 6 characters long, the rest should be padded with spaces (20H).

```
FILNAM    <--- 1st character of file name
  .
  .
  .
FILNAM+5  <--- 6th character of file name
FILNAM+6  <--- "C"
FILNAM+7  <--- "O"
FILNAM+8  <--- " "
```

2. Set Up Parameter

Set the address of the machine code program, its length and execution address.

```
BINADR    <--- Start address of binary data (2 bytes)
BINLEN    <--- Length of binary data (2 bytes)
BINEXE    <--- Execution address (2 bytes)
               0 if this file needs not be executable
               at IPL.
```

3. Fix up directory structure

Refer to Chapter XX, "Description of the ROM Routine in the PC-8300A" for detail of LIKFIL routine.

```
        CALL LNKFIL             ; Fix up directory structure
```

4. Search For The File of the Same Name.

Search the directory for a file that has the same name as the one that we want to save the machine code program. If it exists, delete it. Obviously here you can abort saving the new one, instead of deleting the old one.

```
        CALL SRCCOM             ; Search directory for the file
                                ; whose name is FILNAM
        CNZ  KILCOM             ; Kill old one if exist
```

5. Search Directory for Empty (Free) Slot

To register the file, make sure there is a free slot in the directory and remember the location of the slot. If no free slot is available, abort saving.

```
        CALL SCNEMP             ; Search for empty slot
                                ; Save address of directory
```

6. Allocate Room in RAM File

Allocate room in the RAM for the machine code program and control information (location, length and execution address of the machine code program). The length of the control information is 6 bytes long (ie: 3 words). MAKHOL is the routine to allocate room in [BC] (length) at [HL]. Here [VARTAB] tells the location where the room is to be allocated. The "CO" file is usually saved just under the address pointed to by VARTAB, so the starting address of other files does not need to be changed. However it is a good idea to call LNKFIL after saving a new CO file. When using MAKHOL be sure to adjust the pointer, BINTAB, because MAKHOL changes BINTAB!

```
        [HL] <--- [VARTAB]
        [BC] <--- [BINLEN]+6
        CALL MAKHOL             ; Allocate room
        JC   OMERR              ; Error if out of memory
```

7. Copy Control Information

Copy control information to the top of the room allocated in above step.

8. Copy Machine Code Program

Copy the machine code program into the directory slot obtained by SCNEMP the call.

```
        [HL] <--- [BINADR]      ; Address of machine code file to
```

be saved

```
        [DE] <--- (Top address of the room)+6
                              ;Location in RAM file where the
                              ; program is saved
        [BC] <--- [BINLEN]    ; Length of the program
        CALL LDIRSB           ; Do block transfer
```

9. Reset BINTAB

For BASIC bookkeeping

10. Put File Name into the Directory

Set file name and attributes into the directory slot obtained by SCNEMP call.

```
        [HL] <--- Address of directory slot gained by SCNEMP
call
        [DE] <--- Address of room in RAM file. One used at MAK-
HOL call
        [A]  <--- A0H
                  This is attribute for all ".CO" files
        CALL SETNAM           ; Put name into directory
```

11. Fix up directory structure

```
        CALL LNKFIL
```

2.2 Sample Program to Make a 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        6C0AH          ; Make room
LNKFIL   EQU        233AH          ; make up directory address
                                     field
HEADLN   EQU        6              ; header length of co file
BINTAB   EQU        FAE3           ; lowest address of existing
                                   ; CO file
VARTAB   EQU        FAE5           ; lowest address of variable
                                     table
MAKECO:
         MVI      A,10100000B      ; 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
                                   ; existing 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 isn't
                                   ; enough room
                  XCHG
                                   ;save the top address of the ;
                                   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 the directory is omitted
;
        XCHG                            ; [DE] top of 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,H
        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                  ; update start addresses 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
```

2.21 Sample Program

This sample program saves 0800H bytes in memory from A000H (40960D) as a machine code file "SAMPLE.CO"

```
; Save machine code program in memory into RAM file
; The program is assumed to start at A000H and 0800H bytes long.
; The file is named as "SAMPLE.CO"

        LXI   H,0000H       ; Remember current stack point
                              value
                            ; preparing to error
        DAD   SP            ; In case of an error that
                              transfers
                            ; control to BASIC's error handling
                            ; routine, we need use special
                              error
                            ; trapping to gain control back
; from it, and reset stack pointer
        SHLD  MYSTACK       ;
        LXI   H,ERROR       ; Force control to come back to me
                            ; in case of an error
        SHLD  ERRJMP        ; Activate error trapping
        LXI   H,MYFILE      ; Set up file name to FILNAM
        LXI   D,FILNAM      ;
        LXI   B,0006D       ;
        CALL  LDIRSB        ; Copy 6 bytes from MYFILE into
                              FILNAM
        LXI   H,FILNAM+6    ; Set up extension
        MVI   M,"C"         ; It should always be "CO"
        INX   H             ;
        MVI   M,"O"         ;
        INX   H             ;
        MVI   M," "         ; Extension field is 3 bytes long,
                            ; so pad with a space
        LXI   H,MYBIPA      ; Set up address of machine code
                            ; program, length and execution
                            ; address into BINADR, BINLEN, and
                            ; BINEXE
        LXI   D,BINADR      ; Since BINADR,BINLEN and BINEXE
                            ; stay together
        LXI   B,0006D       ;
        CALL  LDIRSB        ;
        CALL  LNKFIL        ; Fix up directory structure
        CALL  SRCCOM        ; Search directory for a file of
                            ; same name as we want to create
                              now.
        CNZ   KILCOM        ; If exists, delete old one
        CALL  SCNEMP        ; Scan empty slot in directory
                              where
                            ; our file name is out
        PUSH  H             ; Save the address
        LHLD  BINTAB        ; Save current BINTAB
        PUSH  H             ;
        LXI   B,0006D       ;
        LHLD  BINLEN        ; Get length of machine code
                              program
        PUSH  H             ; Save it for future use
        DAD   B             ; 6 bytes for control information
```

```
            MOV   B,H             ;
            MOV   C,L             ;
            LHLD  VARTAB          ; Where file is created
            SHLD  TEMP            ; Save it for future use
            CNC   MAKHOL          ; Make room for control information
                                  ; and machine code program
            JC    OMERR           ; If out of memory was detected
            XCHG                  ;
            LXI   H,BINADR        ; Copy control information into RAM
                                  ; file. Note that control
                                    information
                                  ; must be stay together
            LXI   B,00060         ;
            CALL  LDIRSB          ; Do copy them
            LHLD  BINADR          ; Where machine code program
                                    resides
            POP   B               ; Pick up length of the program
            CALL  LDIRSB          ; Copy the program into RAM file
            POP   H               ;
            SHLD  BINTAB          ; Reset BINTAB
            POP   H               ; Recall address of directory slot
            MVI   A,10100000B     ; This is file attribute code for
                                  ; machine code file
            XCHG                  ;
            LHLD  TEMP            ; temp hold location in RAM file
                                  ; where machine code program is
                                  ; saved (copied)
            XCHG                  ;
            CALL  SETNAM          ; Set up directory
            CALL  LNKFIL          ; Fix up directory structure
RETURN:
            LXI   H,0000H         ; All done
            SHLD  ERRJMP          ; Reset error trapping
            RET                   ;
ERROR:
            LHLD  MYSTACK         ; Error trapping code. Control
                                  ; comes here when directory is full
                                  ; at SCNEMP call.
            SPHL                  ; Reset stack pointer
             .                    ; Put your own error handling code
                                    here
            JMP   RETURN          ;

; Out of memory error
OMERR:
             .                    ; Put your own error handling code
                                    here
             .                    ;
             .                    ;
            JMP   RETURN          ;
MYFILE:
            DB    "SAMPLE"        ; File name
MYBIPA:
            DW    0A000H          ; Address of machine code program
                                  ; to be saved
```

```
            DW    0800H             ; Its length
            DW    0A000H            ; Execute address
MYSTACK:
            DS    02D               ; Stack pointer value is stored
                                      here
                                    ; to reset it
                                    ; in case of error
TEMP:
            DS    02D               ; Temporary storage
```

CHAPTER 3 Loading a Machine Code File

This chapter describes the procedure used to load a machine code program file into memory. The machine code program cannot be executed in the internal RAM. It should be loaded into memory at exactly same location as the program was saved as the RAM.

3.1 Machine Language Program Load Routine

NAME	RLOADM
ADDRESS	28BBH (10427D)
ENTRY	File name should be set up in FILNAM
EXIT	None
REGISTERS ALTERED	[A][B][C][D][E]

DESCRIPTION

RLOADM is used to load a machine code program file (RAM file) into memory at the location where the program was located when saved. If the specified file was not found, or an out of memory error occurred, the control is transferred to BASIC's ROM Error routine.

3.2 Procedure to Load a Machine Code Program

1. Set up File Name

```
[FILNAM]      <--- 1st character in file name
   .
   .
   .
[FILNAM+5]    <--- 6th character in file name
[FILNAM+6]    <--- "C"
[FILNAM+7]    <--- "O"
[FILNAM+8]    <--- " "
```

2. Load Machine Language Program File

Once the file name is set up, you can use the RLOADM routine to load the machine code program file.

```
CALL RLOADM
```

3. Execute The Program (optional)

If the execute address is non zero, the program is executable. Execute the program when necessary.

3.3 Sample Program

```
; Load the machine language file whose name is "SAMPLE.CO".
```

```
; If execution start address is set, start execution.
          LXI   H,0000H          ; get current stack pointer value
                                 ; and save it preparing in case of
                                   error
          DAD   SP               ;
          SHLD  MYSTACK          ;
          LXI   H,ERROR          ; Enable error trapping
          SHLD  ERRJMP           ; Force control to come my error
                                   handler
          LXI   H,MYFILE         ; Set up file name to FILNAM
          LXI   D,FILNAM         ;
          LXI   B,0006D          ;
          CALL  LDIRSB           ;
          LXI   H,FILNAM+6       ;
          MVI   M,"C"            ; And extension
          INX   H                ;
          MVI   M,"O"            ;
          INX   H                ;
          MVI   M," "            ;
          CALL  RLOADM           ; Do load machine the machine code
                                 ; program file into it's real
                                   address
          LHLD  BINEXE           ; Get execution address of the file
                                   (program)
          MOV   A,H              ; See if it is zero
          ORA   L                ; Zero indicates the program is not
                                   executable
          SHLD  JMPT+1           ; Assume executable
          CNZ   JMPT             ; If execution address is given
                                 ; (nonzero),  start execution
RETURN:
          LXI   H,0000H          ; All done
          SHLD  ERRJMP           ; Disable error trapping
          RET                    ;
ERROR:
          LHLD  MYSTACK          ; Error trapping code
          SPHL                   ; reset stack pointer
          MOV   A,E              ; Check error code
          CPI   07D              ; Is this "Out of memory" error
          JZ    OMERR            ; Yes
FILNF:
                                   ; Otherwise should be "File not
found" error
                .                ; Put your own error handler here
                .
          JMP   RETURN           ;
MYFILE:
          DB    "SAMPLE"         ; Name of the file to be load
JMPT:
          DB    0C3H             ; Jump instruction
          DW    0000H            ; Where to jump. This is filled
                                   with
                                 ; execution address of the loaded
                                 ; program.
MYSTACK:
```

```
DS    02D             ; Stack pointer value is saved here
```

CHAPTER 4 SUBROUTINE FOR LOADING AND SAVING

4.1 SRCCOM

NAME	SRCCOM
ADDRESS	2272H (8818D)
ENTRY	File name should be set up in FILNAM
EXIT	Same as SCRNAM
REGISTERS ALTERED	All

DESCRIPTION

The SRCCOM searches the directory for a".CO" file. The name of the file to be searched for should be already setup in FILNAM.

4.2 SCNEMP

NAME	SCNEMP
ADDRESS	22D3H (8915D)
ENTRY	None
EXIT	[HL] Address of empty directory
REGISTERS ALTERED	[A],[B],[C],[H],[L]

DESCRIPTION

SCNEMP is used to search the directory for an empty slot. If an empty directory slot does not exist,control is transferred to BASIC's ROM Error routine. Therefore the user's program must be take care of this error.

4.3 LDIRSB

NAME	LDIRSB
ADDRESS	6C78H (27768D)
ENTRY	[HL] Source address
	[DE] Destination address
	[BC] Length
EXIT	None
REGISTERS ALTERED	All

DESCRIPTION

Refer to the chapter "Description of the ROM routines in the PC-8300A" for details on the LDIRSB routine.

4.4 SETNAM

NAME	SETNAM
ADDRESS	2435H (9269D)

ENTRY	[FILNAM]	File name
	[A]	Directory flag
	[DE]	Top address of file
	[HL]	Address of directory
EXIT	None	
REGISTERS ALTERED	[A][B][H][L]	

DESCRIPTION

SETNAM is used to put a filename and attributes (directory flag) into a slot in directory. The slot is searched for by the SCNEMP call.

4.5 MAKHOL

NAME	MAKHOL	
ADDRESS	6C0AH (27658D)	
ENTRY	[HL]	Start address
	[BC]	Length
EXIT	Carry	1 if out of memory
REGISTERS ALTERED	[A],[D],[E]	

DESCRIPTION

The MAKHOL routine is used to open a "hole" in the RAM file and change a few pointers related to RAM files. The purpose of the MAKHOL is to allocate room in the RAM file to store the machine code program.

4.6 Variables

1. BINADD F9C0H (63936D)
   Location of the machine language program to be saved, or location where the machine code program is loaded.

2. BINLEN F9C2H (63938D)
   Length of the machine language program to be saved or loaded, set by RLOADM after the program was loaded.

3. BINEXE F9C4H (63940D)
   Execution address of the machine program, also set by RLOADM.

4. FILNAM FB78H (64376D)
   The filename is stored here.FILNAM is 9 bytes long, the first 6 bytes are used to store the body of the filename, and the rest are for extension.If the length of the body and/or extension is less than the maximum, the space character (20H) should be used.

5. VARTAB FA8BH (64139D)
   Pointer to the start of the simple variable space. The location pointed to by VARTAB becomes the location where the RAM file for machine code program is created.

6. BINTAB FAE3H (64227D)
   Pointer to the lowest address of the machine code program file area.

PC-8300A MISCELLANEOUS INFORMATION

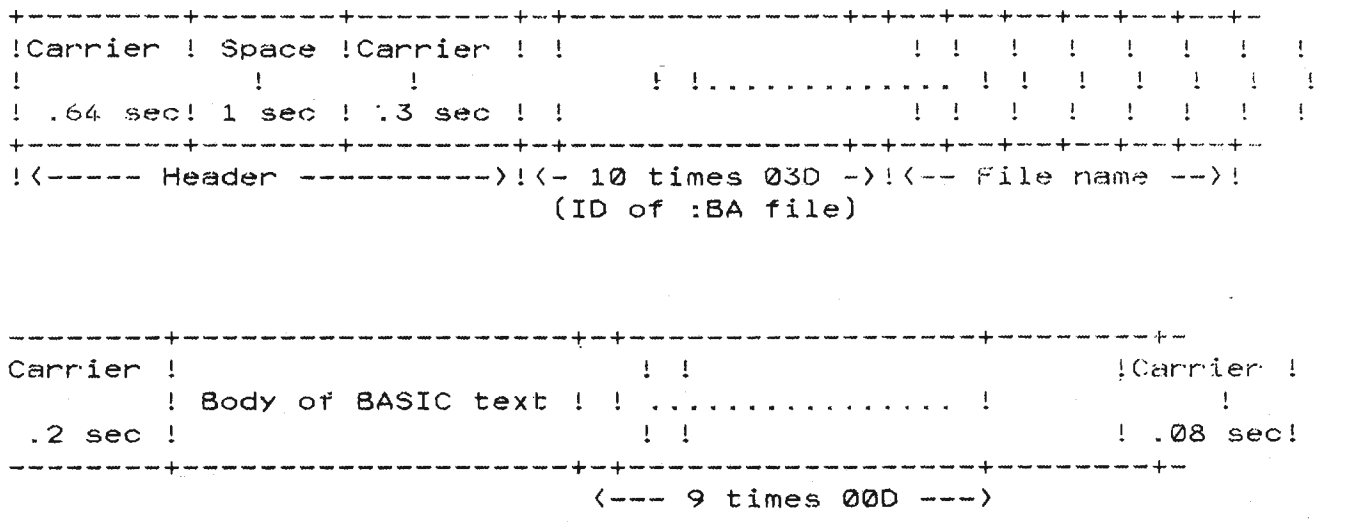
CHAPTER 1. Tape Formats
1.1 ".BA" file
1.2 ".CO" file
1.3 ".DO" file
1.4 Notes

CHAPTER 2. 2nd ROM Information
2.1 ID of the 2ND ROM
2.21 Procedure for using the 2nd ROM
2.2 Method to use 1st ROM entry from 2nd ROM
2.3 Assignment of the Interrupts
2.4 Routine for Using the 2ND ROM
2.5 Sequences in the 2nd ROM
2.6 Sample Code
2.7 Variables used in Sample Routine

CHAPTER 3 Summary

CHAPTER 1 TAPE FORMAT

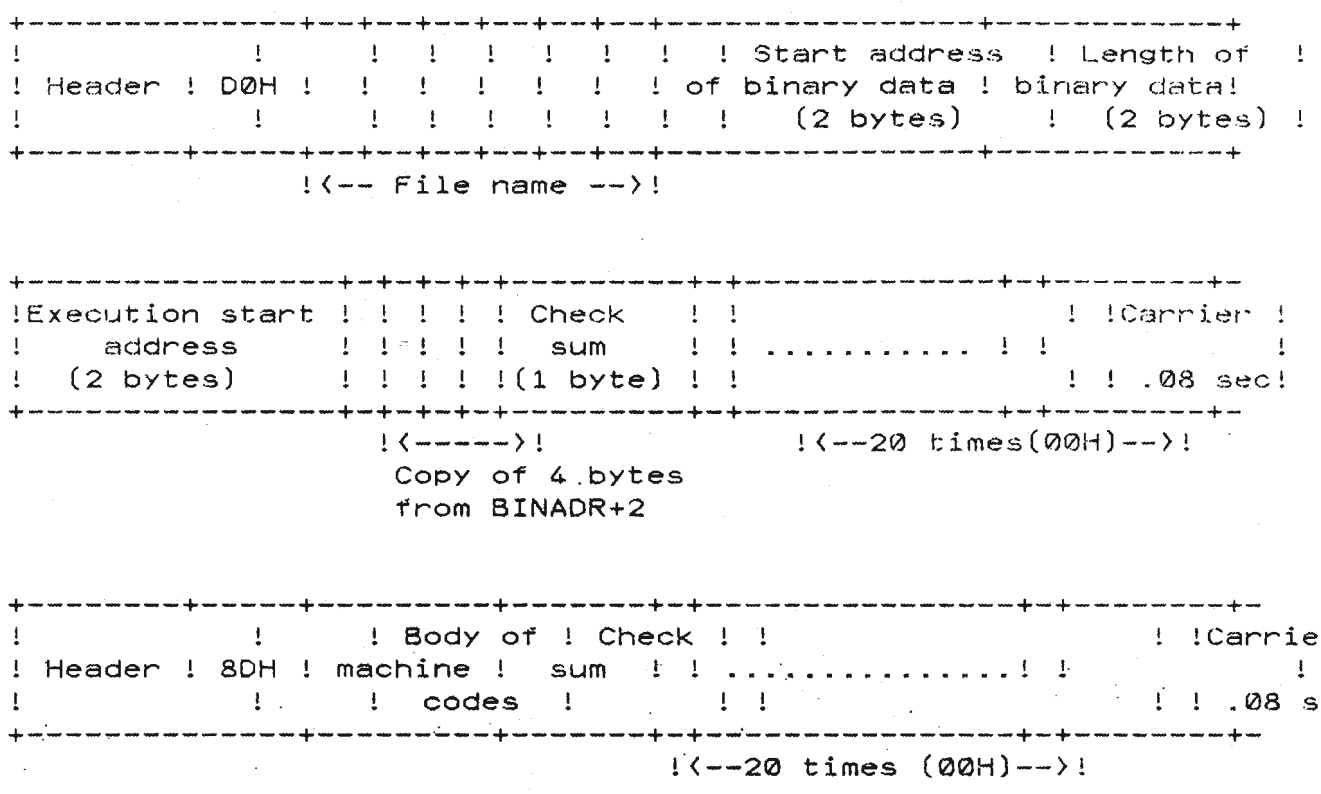
1.1 ".BA" file



The first carrier in the header causes the motor to wait that the cassette will rotate smoothly. The following space and carrier are used for synchronizing the code.

The body of the filename should be less than or equal to 6 characters long. If it is less than 6 characters long, the excess should be packed with nulls (00D). The body of the BASIC text contains the intermediate codes from the top, pointed to by TXTTAB or the address field in the directory to triple 00D. (ie: end mark of BASIC text).

1.2 ".CO" file

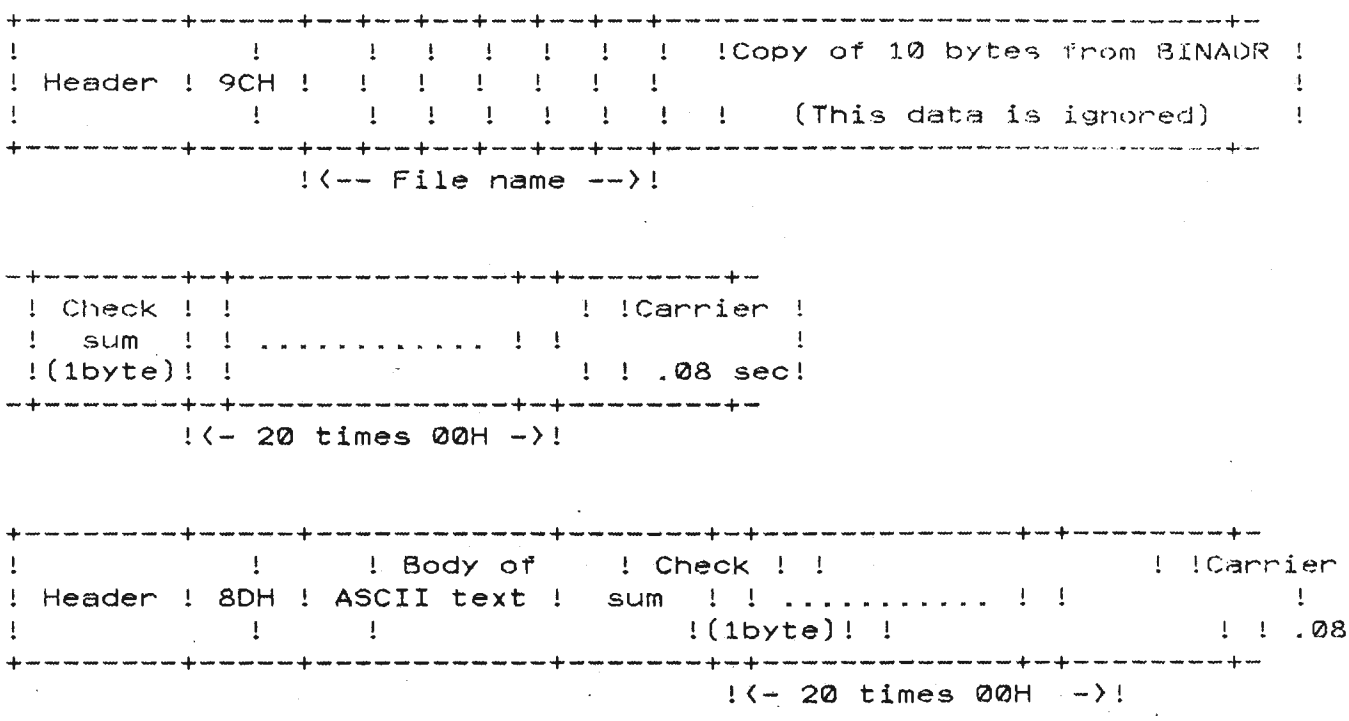


The header is the same as for the ".BA" The value D0H after the header is the ID for a ".CO" file, and 8DH is the ID of the data block.

The body of the file name should be less than or equal to 6 characters long. If it is less than 6 characters long, the rest should be padded with spaces, 20H.

The checksum after the execution start address is a check sum of the file name, start address of binary data, length of binary data, execution start address and next the 4bytes. The next checksum is for the body of the machine codes. This checksum is the sum's complement of 2.

1.3 ".DO" file



The header is the same as the ".BA" file. The value 9CH after the header is the ID of a ".DO" file, and 8DH is the ID of the data block.

The body of filename should be less than or equal to 6 characters long. If it is less than 6 characters long, the rest should be padded with spaces, 20H. The body of the ASCII text is ASCII data from the top, pointed to by the address field in the directory to 1AH (ie: terminator of ASCII file).

The first check sum is a check sum of the filename and the next 10 bytes. The next check sum is of the body of the ASCII data. The checksum is the sum's complement of 2.

1.4 Notes

The carrier is a high frequency sound, and the space is a low frequency sound.

The Basic Input/Output Routines for cassette are written in Chapter XX, "Description of the ROM routines in the PC-8300". For further information please refer to Chapter XX, "Description of machine code file handling routines in PC-8300", and Chapter XX, "Internal structure of PC-8300 file system".

CHAPTER 2 INFORMATION ON THE 2ND ROM

Before storing programs in the second ROM, there are a lot of matters which should be attended to and stored in the second ROM, such as the interrupt jump tables and the power on/power off sequences. One has to implement these tables smoothly, otherwise the PC-8300A can run away with the ROM bank switching. The following chapter contains the information needed to utilize the second ROM bank.

2.1 ID of the 2nd ROM

Before using the second ROM, one must write the following information into the second ROM's reserved memory area. The reserved area is located from 0000H to 47H (71D). This area is used for the second ROM's starting jump instruction and ID code, and the file name for the second ROM. The file name is displayed like one of the RAM files on the MENU screen by the first ROM, ROM #0. The following illustration explains the special reserved area in the second ROM.

```
Address
0000H     JMP  START              ; The execution starts here
                                 ; when 2nd ROM is selected
0001H                              ; start address low
0002H                              ; start address high
  .
  .
  .
0024H     RET                      ; Non-maskable interrupt
002CH     RET                      ; Barcode reader interrupt
0034H     RET                      ; UART interrupt
003CH     RET                      ; Interval timer interrupt
003FH     RET                      ; Reserved for RST interrupt
0040H     DB          "A"
0041H                 "B"          ; ID code for the 2nd ROM
0042H     DB          "2NDROM"     ; Name of 2nd ROM
0043H     ...                      ; Indicates name in menu
0044H     ...                      ; display of RAM bank 1
  .
  .
0047H     ...                      ;
0048H     START:                   ; 2nd ROM's code
```

If this data is implemented correctly, the name, "2NDROM", will appear on the 1st ROM's menu screen. Therefore it is easy to switch to the 2nd ROM and execute the programs. To start the programs in the 2nd ROM from the menu mode of ROM #0, move the cursor to the 2nd ROM's file name on the screen and press return. The system will then fall into the 2nd ROM programs.

2.2 Method to use 1st ROM entry from 2nd ROM

If one wants to use the routines in the 1st ROM from the 2nd

ROM, one first has to create a special routine in a higher memory location of RAM (8000H-FFFFH) and implement it. That routine will switch the ROM bank using the bank switching method, and call the routine in the 1st ROM. It is very important that the interrupts be disabled before the ROM bank is changed. The following sections explain that one has to change the hook table for the Power down interrupt that was changed by the 2nd ROM to restart the current process in the 2nd ROM program at the next power on. With the hook table for the 2nd ROM, the power down in ROM #0 will cause a fatal error. The Power-off interrupt can not b prohibited. The contents of the routine which will be called must also be considered, because some routines in the first ROM may enable the interrupts in some part of their code, even if you disable the interrupts just before switching ROM banks. Therefore all of the values in the hook table should be changed just before calling the ROM bank switching routine.

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

2.21 Sample Program

```
; This program will enable one to use the 1st ROM entry from
; the second ROM. Some routines in the 1st ROM might enable
; interrupts, so all of the interrupts in the hook table
; should be replaced with a RET code. And they should be
; restored after one is done calling the 1st ROM
; Entry   1st ROM entry address
; Exit    for return condition of the 1st ROM

; <<< SYSTEM define label >>>
BNKCRL    EQU       0A1H              ; Bank control port
STATUS    EQU       0A0H              ; Bank status port

; <<< Main Routine >>>
          ORG       8000H             ; Routine must be between
                                      ; 8000H and FFFFH
ROM1ST:   SHLD      WORKH             ; Save register HL
          LXI       H,RET2ND          ; Return address from 1st ROM
          PUSH      H                 ; Push stack top
          LHLD      ENTRY             ; 1st ROM entry address
          PUSH      H                 ; Push stack top
          LHLD      WORKH             ; Restore HL
          PUSH      PSW               ; Save all registers
          DI                          ; Disable interrupts
          IN            STATUS            ; Get current bank status
          ANI       11111110B         ; Switch 1st ROM data setup
          OUT       BNKCRL            ; Bank select
                                      ; Now 0000H-7FFFH are 1st ROM
          EI                          ; Enable interrupts
          POP       PSW               ;
          RET                         ; Jump 1st ROM entry
; <<<Return from 1st ROM >>>
RET2ND:
          PUSH      PSW               ; Save all registers
          IN          STATUS              ; Get current bank status
          ORI       00000001B         ; Switch 2nd ROM data setup
          OUT       BNKCRL            ; Bank select
                                      ; Now 0000H-7FFFH
          POP       PSW               ;
          RET                         ;
; <<< SYSTEM Work Area >>>
ENTRY:    DW            0000H             ; 1st ROM entry address
WORKH:    DW            0000H             ; HL register saving area
          END
```

Note:

At the first power on after setting the 2nd ROM, the PC-8300A must be cold started (hold down the shift key, the ctrl key, and the stop key, and press the reset button on the rear of the unit). Therefore all of the data which was stored in the memory before setting of 2nd ROM will be destroyed.

2.3 Assignment of interrupts

The main purpose of the interrupts is smooth processing in

the Power Off Trap, reading data from the Bar-Code reader, communicating through the UART (RS-232C) and using the Interval Timer. The interrupts are located at the Zero Page Area. The interrupts of the PC-8300 are assigned as follows.

The Interval timer interrupt has the highest priority, and the UART is second. The lowest priority interrupt is used for the Barcode reader. The Interval timer has the highest priority to be able to scan the keyboard and to count the auto-power off counter for saving the battery power. The PC-8300A's autopower off function is executed after 10 minutes has past since the last key stroke was detected. This interval can be set by the "POWER" command in BASIC. The interval timer is used to count this period.

The interrupt table is located in the zero page area.

POWER OFF TRAP	NMI	0024H (36D)
BARCODE READER	RST 5.5	002CH (44D)
UART	RST 6.5	0034H (52D)
INTERVAL TIMER	RST 7.5	003CH (60D)

The interrupt hook table is in the RAM area.

F386H (62342D)	Power On Sequence
F389H (62345D)	Barcode Reader Input Sequence
F38CH (62348D)	UART Input Sequence
F38FH (62351D)	Timer Sequence and Key Scanning Sequence
F392H (62354D)	Power Failure Sequence

1. TRAP (NMI) Power off trap 24H (36D)

This interrupt is non-maskable. When the power switch is turned off, this interrupt occurs. The following sequence is the algorithm of this interrupt.

1: Disable the interrupt
2: Call the hook table
3: Reset the key wait counter
4: Cancel the time counter
5: Output data to the auto power off port
6: HALT

The bit assignment for the Auto power off port is as follows.

PORT ADDRESS BAH [OUT] (186D)
81C55 port B

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 block select
Bit 0 LCD block select

2. RST 5.5 Barcode reader 2CH (44D)

This interrupt is using RST 5.5. If one does not use the barcode reader program, this interrupt should RETURN.

(ADDRESS F389H (62345D) with Disable Interrupt)

3. RST 6.5 UART 34H (52D)

(ADDRESS 6E00H (28160D) with Disable Interrupt)

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

The algorithm of this interrupt is shown below.

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

PORT ADDRESS D8H (216D) [OUT]
(rs232c Command and status Port)

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 Data Length is 5
1: Stop bit 2 bit
in case Data Length is not 5

PORT ADDRESS C8H (200D) [OUT}
UART data I/O port (6402 Data Register)

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

4. RST 7.5 Interval timer 3CH (60D)

This interrupt is using RST 7.5. The Interval Timer in-

terrupt (Timer device 1990) is also used for the key scar
ning.

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

PORT ADDRESS A 89H (185D) [OUT]
Calendar clock (1990) control port
Printer strobe, Keyboard strobe, LCD Chip select, and Clock Data

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 64
1	0	1	timing 25
1	1	0	timing 20
1	1	1	TES de

In the initialization routine, the command is set up as 05H, which means 4m second interval.

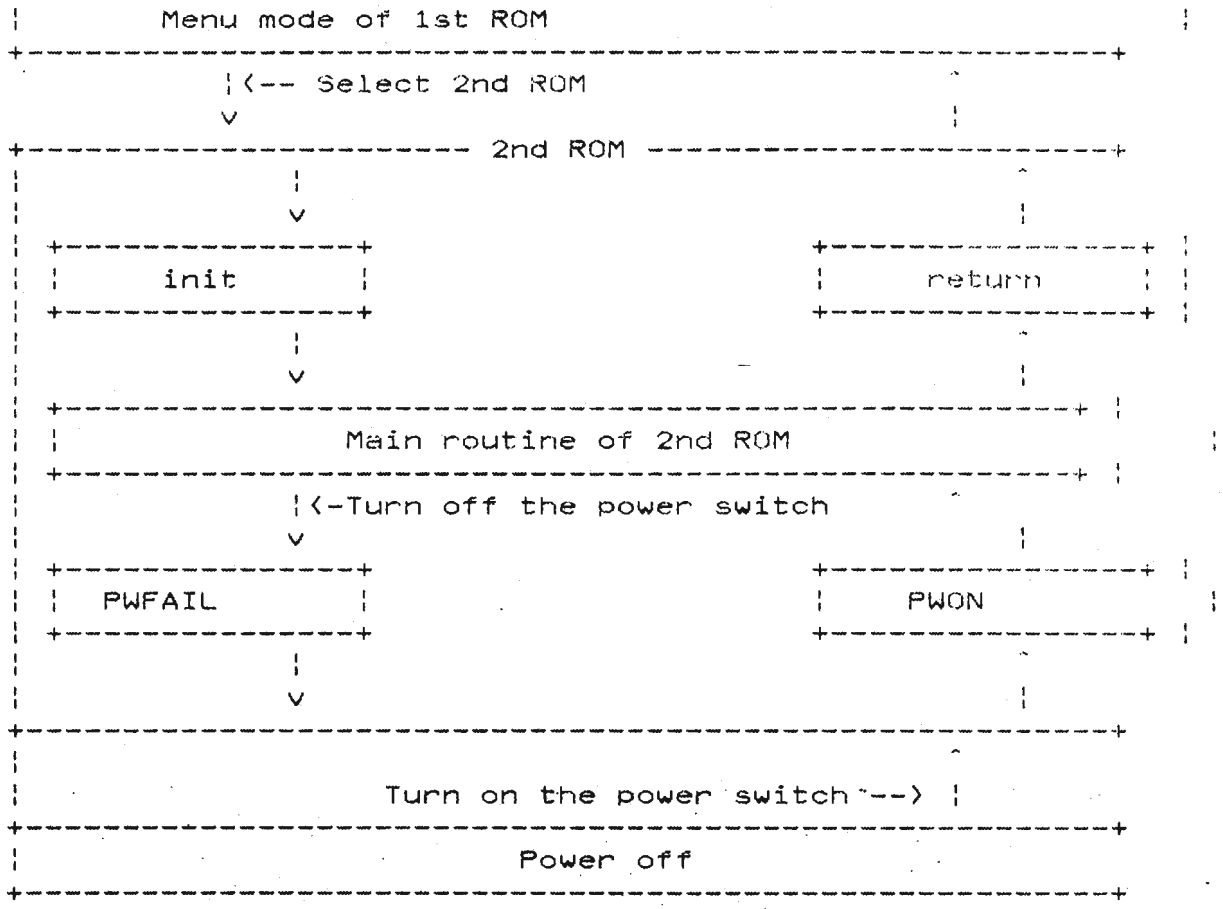
The following step is the algorithm for the interval timer sequence.

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

2.4 Some routine for using 2nd ROM

We prepare some routines in order to use 2nd ROM. When you use the following routine with the 2nd ROM, the PC-8300 perfor as follows.

+----------+


Menu mode of 1st ROM
<-- Select 2nd ROM
2nd ROM
init
return
Main routine of 2nd ROM
<-Turn off the power switch
PWFAIL
PWON
Turn on the power switch --->
Power off

2.5 SEQUENCES IN THE 2ND ROM

1. INIT

INIT sets up the SP (Stack Pointer), the power on trap and other interrupt routines. Then it copies the bookkeeping area and the system area. Also some peripherals will be initialized by this routine.

2. RETURN TO MENU

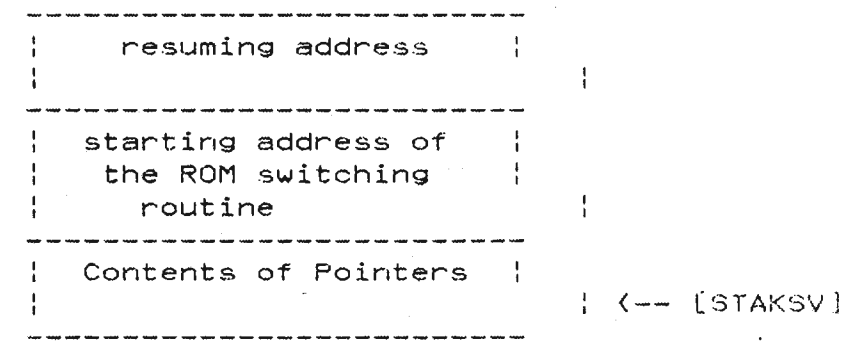
RETURN selects the standard RAM, RAM #0 and resets the power-off trap. It then jumps to the 1st ROM's menu mode.

3. PWFAIL (Power Down)

When the power is turned off,the control is transfered to the routine. One must save all registers and circumstances which should be saved in the stack. The stack pointer is the most important register to resume the current processing on the next power-on.

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

In addition, in order to resume the abandoned process with the 2nd ROM, one has to do a special power on/power off sequence. In the power off trap, one should set the start routine of the special power on sequence which switches the ROM bank. It is recommended that one uses the hook, F38FH. Usually, the "Jump To Power Fail Sequence" command is stored here. In the 2nd ROM, however, one has to rewrite this hook table and call the special power down routine here. In this routine, the address of the special power on routine on the stack. In this case, the following information should be stacked before the "HLT" command is executed.



[STAKSV] keeps the Stack Pointers value at "HLT"

4. PWON

At initial power on, the initialization routine in ROM #0 checks the RAM bank number in BANK (F3DBH) when power off was executed. When power off occurs in the non-standard RAM bank, the RAM bank-switching routine is called and switched. Then, the register 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 to be resumed. Therefore, after switching the ROM, the abandoned process will be resumed.

Note

The following routine needs an internal work area. So, one has to secure any memory area by BASIC's clear command, before 2nd ROM start. The HINIT in PWON initializes only interrupt, 8155, the interval timer and LCD. So, if one wants to use other hardware, please put the initialization code in here (PWON).

2.6 SAMPLE CODE

```
;
; Sample code to use 2nd ROM
;
BANK        EQU   F3DBH              ; Bank save area
ATIDSV      EQU   F383H              ;
PWHOK       EQU   F386H              ; Power on hook table
RST55       EQU   F389H              ; Rst 5.5 hook table
STAKSV      EQU   F9AEH              ;
AUTOID      EQU   9C0BH              ;
SAVSTK      EQU   FAD0H              ;
STATUS      EQU   A0H                ; Bank status
BNKCRL      EQU   A1H                ; Bank control
PWPORT      EQU   B8H                ; 81C55 chip select
PORTB       EQU   BAH                ; 81C55 port B
FREE        EQU   ????H              ; You must set your RAM
                                   ; free portion address
;
; <<< Main Routine >>>

                    ORG   0000H              ; Restart 0
START:
            JMP   INIT               ; Jump to initialization routine
                                   ;when 2nd ROM is called from 1st
                                   ; ROM's menu mode
            ORG   0024H              ; Non-maskable interrupt
            JMP   POWER              ; Power off trap
                                   ;
            ORG   002CH              ; RST 5.5
            JMP   BARCOD             ; Barcode interrupt
                                   ;
            ORG   0034H              ; RST 6.5
            JMP   UART               ; UART interrupt
                                   ;
            ORG   003CH              ;
            JMP   TIMER              ; Timer interrupt
                                   ;
            ORG   0040H              ; ID code for 2nd ROM
            DB    'AB'               ; 2nd ROM's ID
            DB    '2NDROM'           ; Filename displayed in menu
```

```
;
INIT:
            LHLD SAVSTK              ; Set stack pointer
            SPHL                     ;
            CALL SETTRP              ; Set hook for resume 2nd ROM's
                                     ; programmed other routine into RAM
            CALL HINIT               ; Hardware initialization
            JMP  MAIN                ; Jump your main routine
;
MAIN:
; <<< Set up hook >>>
; Set up the hook table for the 2nd ROM
SETTRP:
            MVI  A,00000001B     ; 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                 ; Copy [B] bytes
            STAX D                   ; Source:[HL]
                                     ; Destination:[DE]
            INX  H                   ;
            INX  D                   ;
            DCR  B                   ;
            JNZ  COPY                ;
            RET                      ;
;
; The following code will be copied into RAM
; portion for re-power on sequences
; these parts are interrupt hook table
;
DTBL        EQU  $
            MVI  A,00000001B     ; This code will be copied into RAM
            OUT  BNKCRL              ;
            JMP  PWON                ;
BANKI:      DS   1                   ;
TBLEND      EQU  $                   ;
```

```
;
; The following code will be copied
; into the RAM portion for return to 1st ROM
;
TBLHOK      EQU   $
RETSB:
            XRA   A                    ; Clear A
            OUT   BNKCRL               ; Select 1st ROM and
                                     ; standard RAM
            JMP   0                    ; Return!
HOKE        EQU   $
;
;
RETURN:
            MVI   A,00000001B      ; Select standard RAM
            OUT   BNKCRL               ;
            MVI   A,00000000B      ;
            STA   BANK                 ;
            LXI   H,0000H              ; Reset ATIDSV
            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
;
; The following code will be copied
; into the 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                ; Initialization of hardware
            LDA   BANKI-DTBL       ; Select old RAM bank
            OUT   BNKCRL               ;
            LHLD  STAKSV               ; Restore stack pointer
            SPHL                       ;
                                     ; If you do not want to resume, put
                                     ; that any code for

            POP   PSW                  ; Restore all register
            POP   B                    ;
            POP   D                    ;
            POP   H                    ;
            RET                        ; Resume old program
```

```
;
PWFAIL:
            PUSH  PSW                   ;
            IN    PWPORT                ; Read power down port
            ANA   A                     ; Check
            JM    NTPWFL                ; No power down
            POP   PSW                   ;
            DI                          ; Disable interrupt
            PUSH  H                     ; Save all registers
            PUSH  D                     ;
            PUSH  B                     ;
            PUSH  PSW                   ;
            LXI   H,0000H               ;
            DAD   SP                    ;
            SHLD  STAKSV                ; Save stack
            MVI   A,0FFH                ; Reset interval timer
            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,00000001B          ; Select standard RAM
            OUT   BNKCRL                   ; Select!
            MOV   A,B                      ; Resave old status
            STA   BANKI-DTBL           ;
            MVI   A,00000001B          ; Select RAM bank 1
            OUT   BNKCRL                   ;
            MVI   A,00H                    ; Set up to come back to 2nd ROM
                                       ; when next power on
            STA   BANK                     ;
            LXI   H,AUTOID             ;
            SHLD  ATIDSV                   ;
            IN    PORTB                    ; Power off
            ORI   00010000B            ;
            OUT   PORTB                    ;
            HLT                            ;
NTPWFL:
            POP   PSW                   ;
            RET                         ;
; <<< BARCODE READER 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    0FFH                  ; Timer counter n * 1/256Hz

            END
```

2.7 Variables Used in this Sample Routine

1. ATIDSV F383H (62338D)
   To resume operation or not
   (Address to jump to Resume)

2. BANK F3DBH (62427D)
   The Bank status is saved here.
   (Current Bank selected)

3. PWRHOK F386H (62342D)
   Power on hook
   (RST 00 Hook)

4. ROMSEL FE44H (65092D)
   Holds the value output to IOCNT (System Control Port)
   (Copy of Port 090 IOCNT)

5. SAVSTK FAD0H (64208D)
   User can use this area (above [SAVSTK]) in the 2nd ROM as the stack area (Save for SP used by Resume)

6. STAKSV F9AEH (63918D)
   Stack pointer save area during auto power off state

3.0 SUMMARY

To make a 2nd ROM program, one should take care of the following manner.

A. INTERRUPT VECTOR

If one does not want to use interrupts, the entire interrupt table should be set with "RET" code. It is suggested that one use the interval timer interrupt, because of saving the battery power by using the auto power off function. The counter for this auto power off function is counted by this interval timer interrupt. If not used, battery consumption may be higher than normal.

B. BANK of RAM

Do not switch the ROM bank when the PC, Program Counter, points to a routine in ROM. This is because this bank switching will cause a fatal problem for the system, the worst case being, all of the files stored in RAM will be lost. Care should also be taken in the stack area.

C. PC-8300A BOOKKEEPING AREA

The bookkeeping area is very important for this system, so never change that area without careful consideration. For more information see Chapter XX, "Bookkeeping Area".

D. POWER ON/OFF SEQUENCE

It is recommended that one use the power off interrupt to detect the power down, using the real time interrupt service to poll the power down signal.

If one is using the 1st ROM entry from the 2nd ROM, please take care of the following points. All routines rewrite work area sometimes, so if using the 1st ROM entry from the 2nd ROM without understanding the routine's internal specification, the system might crash. In addition, pay special attention to the interrupts and stack area.

INTERNAL STRUCTURE OF THE PC-8300A RAM FILE SYSTEM

CHAPTER 1 Directory
1.1 Directory configuration
1.2 Directory configuration per entry
1.3 Bit assignment of directory flag
1.4 Value of address field

CHAPTER 2 File structure
2.1 File structure of machine language file
2.2 File structure of ASCII file
2.3 File Storage

CHAPTER 3 Bookkeeping Area
3.1 Part I of the Bookkeeping Area
3.10 Variables For The RAM File Handling and Basic
3.11 Memory Map
3.12 Description of the Variables

CHAPTER 4. Part II of the Bookkeeping Area

CHAPTER 5 Part III of the Bookkeeping Area

CHAPTER 6 The File Control Block (FCB)

CHAPTER 7 Notes

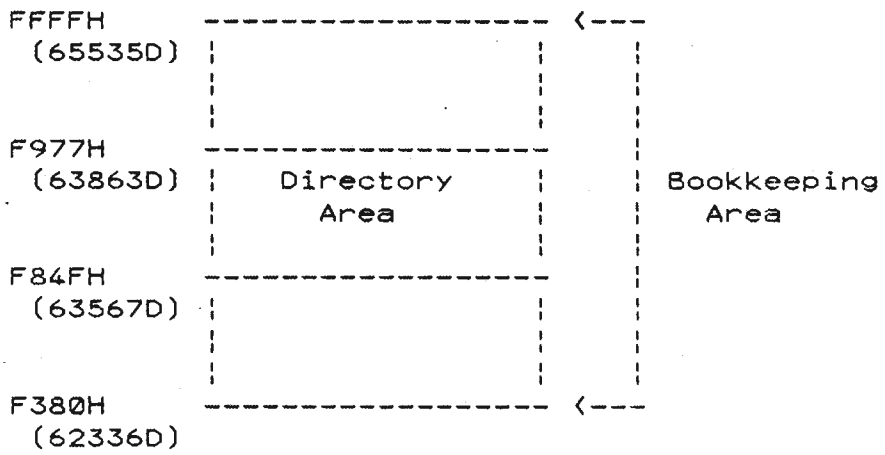
CHAPTER 8 Sample Program

CHAPTER 1 DIRECTORY

1.1 Directory Configuration

The directory area is allocated in the middle of the bookkeeping area. The top address is F84FH. A non-registered Basic program is a program which has not been saved.

F84FH (63567D)	BASIC	Directory for program in ROM
F85AH (63578D)	TEXT	Directory for program in ROM
F865H (63589D)	TELCOM	Directory for program in ROM
F870H(63600D)	NULDIR	Directory for non-registered BASIC program
F878H (63611D)	SCRDIR	Directory for SCRAP (used by EDIT and TEXT)
F886H (63622D)	EDTDIR	Directory for temporary used by EDIT
F891H (63633D)	USRDIR	Directory for user-defined files
.		
.	(21 directories total)	
.		
F978H (63864D)	FFH	Directory search stopper



1.2 Directory configuration per entry

The first six slots in the directory area are initialized by the INIT routine at Cold Start.

Directory flag	(1 byte)
Address field	(2 bytes)
File name	(8 bytes)

The initialized values for the first six slots in the directory are shown below. The first three files are stored in ROM and displayed on the menu screen. The next three files are used for

hidden files created in the RAM area. These hidden files will not appear on the menu screen. The characteristics of these hidden files are described below.

(Initialized data is stored in 6C8EH)

```
DB        10110000B
DW        Start address of BASIC
DB        'BASIC   '
DB        0

DB        10110000B
DW        Start address of TEXT
DB        'TEXT    '
DB        0

DB        10110000B
DW        Start address of TELCOM
DB        'TELCOM  '
DB        0

; for non-registered program

DB        10001000B
DW        0
DB        0
DB        'XXXXXXX'

; for Scrap file

DB        11001000B
DW        0
DB        0
DB        'YYYYYYY'

; for EDIT command of BASIC

DB        01001000
DW        0
DB        0
DB        'ZZZZZZZ'
```

1.3 Bit assignment of directory flag

Bit	7	Master bit	(1 means directory valid)
Bit	6	ASCII bit	(1 means ASCII-TEXT file)
Bit	5	Binary bit	(1 means Machine language file)
Bit	4	File in ROM	(1 means file is in ROM)

Bit	3	Hidden file	(1 means file is hidden)
Bit	2	IPL	(1 means IPL set)
Bit	1	RAM file open flag	
Bit	0	Internal use	(Always set to 0 normally)

1.4 Value of address field

BASIC text	Address which TXTTAB must be set
ASCII text	Beginning address of file
Machine language	Beginning address of file
Machine language in ROM	Entry address

The TXTTAB in Basic shows the lowest byte of the file, the first link pointer in the Basic program file.

CHAPTER 2 File Structure

2.1 File Structure of Machine Language File

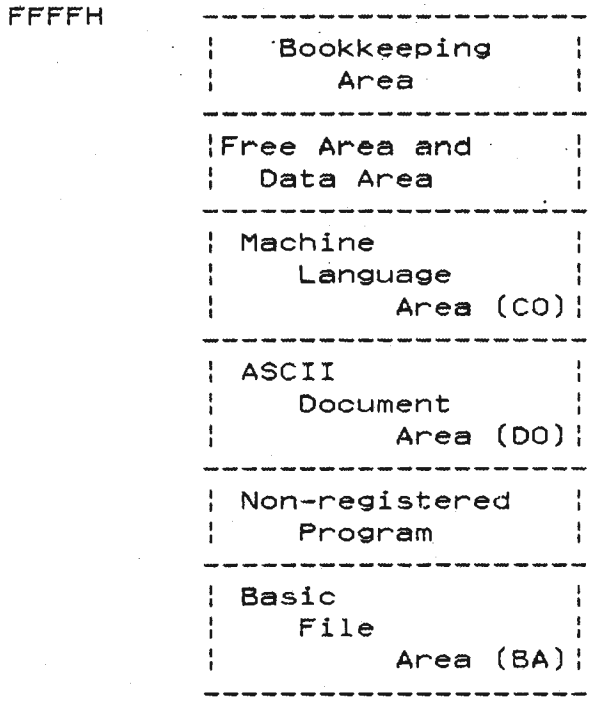
Start address	2 bytes
Length	2 bytes
Execution address	2 bytes
Machine code program	

2.2 File Structure of ASCII File

ASCII text	
1AH (End of TEXT)	1 byte

2.3 File Storage

The files in the internal RAM are stored in a fixed order. The Basic files ("BA") are stored at the bottom of the RAM area, near 8000H. When a new Basic program is stored is will be placed at the top location of the Basic files allocated space (at the next lowest address after the Document files). The "DO" files (ASCII files with the suffix ".DO") are allocated above the BA files. Machine Language files ("CO" files) are saved above the DO files, near FFFFH. The following illustration shows the order in which files are saved.



A new BA file is created above the old BA files. 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 to by VARTAB. The non-registered BA file is created between the BA files and the DO files.

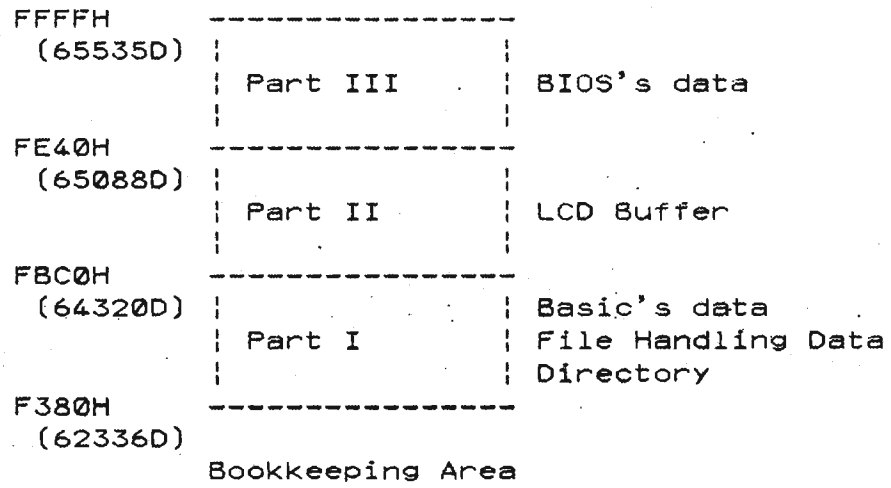
CHAPTER 3 BOOKKEEPING AREA

The bookkeeping area is located at the top of the RAM area.

The area is divided into 3 parts. The first part, the lowest part from F380H to FBBFH, includes the pointers and flags for RAM file handling. Also many of the Basic interpreter's flags, pointers and temporary data is stored here. The Directory Area is included here.

The second part, FBC0H to FE3FH, is used for the line buffer of the LCD display. Basic also uses this area in the Screen Editor function.The concept of this line buffer is different from the VRAM in traditional desk top personal computers. Only the character codes are stored in this buffer. There is no attribute data. The attribute data is stored in another table (see the explanation of the LCD driver).

The third part, FE40H to FFFFH, is reserved by the BIOS. The switches and data storage for the RS-232C, Key Board and other I/O drivers are stored here.



3.1 PART I of the BOOKKEEPING AREA

3.10 VARIABLES For the RAM File Handling and Basic

Many important pointers are stored in this area for RAM file handling. When some of the pointers are mishandled in your routines, all RAM files might be deleted at the next operation of the standard ROM (ROM #0). The built-in programs assume that the pointers point to the correct addresses. So if a pointer which should point to the lowest address of the DO files, points one byte smaller than it should, text might not invoke any DO files. Make sure that the pointers contain the correct values at all times.

1. ARYTAB FAE7H (64231D)
   Pointer to the beginning of the array table

2. ASCTAB FAE1H (64225D)
   Pointer to the start of the ASCII files

3. BINTAB FAE3H (64227D)
   Pointer to the start of the binary files

4. BOTTOM F9B0H (63920D)
   Bottom address of RAM

5. DIRPNT F979H (63865D)
   Pointer to the directory of the current Basic program

6. DIRTBL F84FH (63567D)
   Points to the beginning of the Basic ROM program

7. EDTDIR F886H (63622D)
   Directory entry for EDIT in Basic

8. FILTAB FB63H (64355D)
   Pointer to the address of the file data

9. FRETOP FABFH (64191D)
   Top of the string free space

10. FSIDSV F380H (62336D)
    First power on or not, to determine a Cold Start situation (ie, address to jump to on first power up)

11. HIMEM F384H (62340D)
    Highest memory available to BASIC (ie:same as CLEAR command's 2nd parameter)

12. MEMSIZ FA9AH (64154D)
    Highest location in memory

13. NULBUF FB67H (64359D)
Pointer to the address of file buffer #00

14. NULDIR F870H (63600D)
Directory entry for a non-registered Basic program

15. SCRDIR F87BH (63611D)
Directory entry for scrap file

16. STKTOP F459H (62553D)
Top location to use for the stack

17. STREND FAE9H (64233D)
End of storage in use

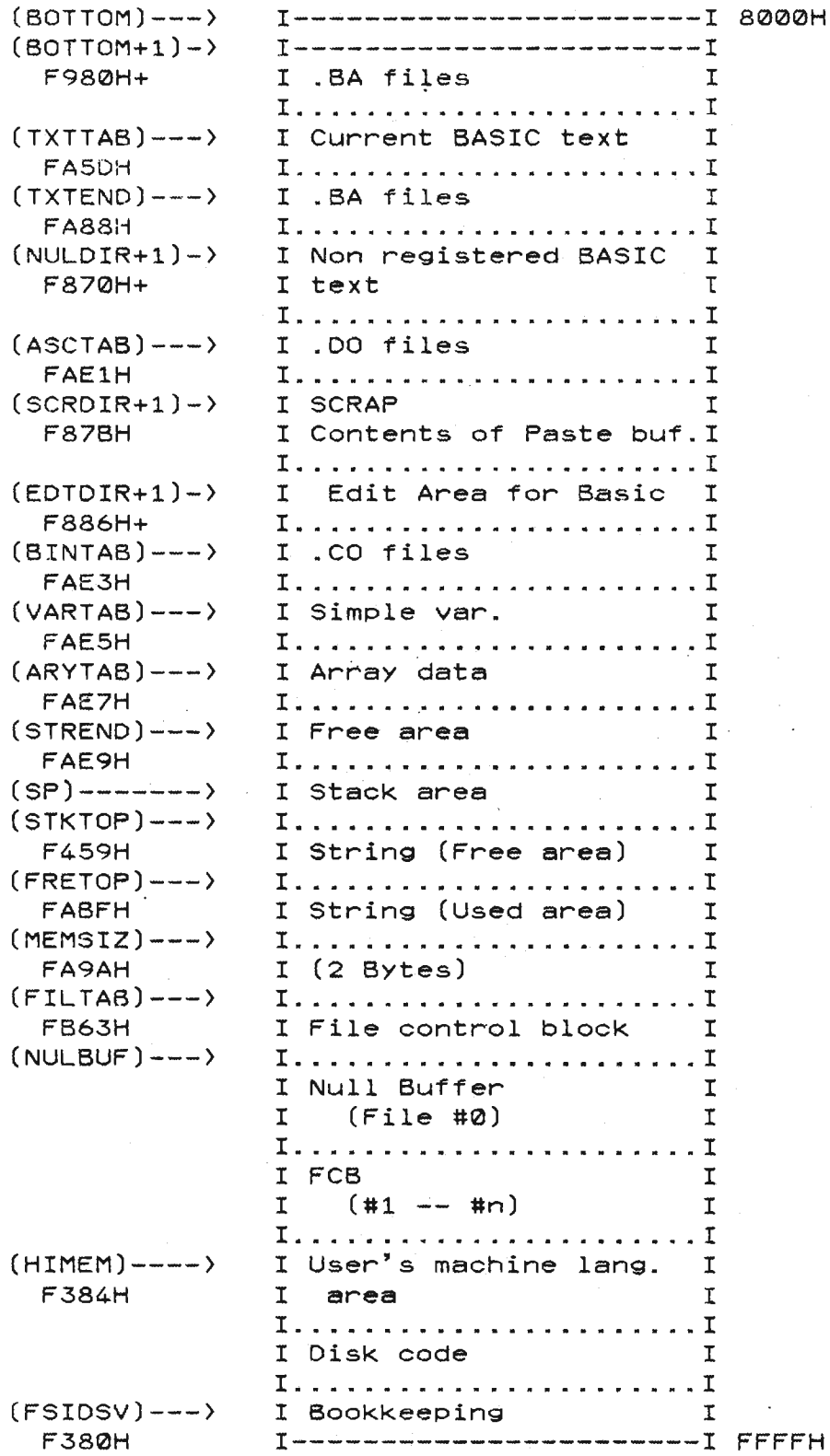
18. TXTEND FA88H (64136D)
End of the current Basic program

19. TXTTAB F45DH (62557D)
Pointer to the beginning of the Basic text

20. USRDIR F891H (63633D)
Directory for the user's files. Points to the first user directory entry.

21. VARTAB FAE5H (64229D)
Pointer to the start of simple variable space

3.11 MEMORY MAP



3.12 Descriptions of the Variables

1. ARYTAB

ADDRESS FAE7H (64231D)
SIZE 2 bytes
PURPOSE Pointer to the beginning of the array table

The Array Table is allocated just above the Variable Table. This points to the beginning address of this Array Table.

2. ASCTAB

ADDRESS FAE1H (64225D)
SIZE 2 bytes
PURPOSE Pointer to the start of the ASCII files

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

3. BINTAB

ADDRESS FAE3H (64227D)
SIZE 2 bytes
PURPOSE Pointer to the start of the Command files

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

4. BOTTOM

ADDRESS F9B0H (63920D)
SIZE 2 bytes
PURPOSE The bottom address of RAM

The lowest available RAM address is saved here. One can easily know how many RAM chips have been installed in a RAM bank by checking this pointer.

5. DIRPNT

ADDRESS F979H (63865D)
SIZE 2 bytes
PURPOSE Pointer to the directory of the current Basic program.

6. DIRTBL

ADDRESS F84FH (63567D)
SIZE 33 bytes
PURPOSE Directory for the programs in the ROM

The names and pointers for the programs in ROM (Basic, Text, and Telcom) are stored in DIRTBL. If these programs are not being used, this area may be used by the user's programs. This area will be kept until a "COLD START" is invoked.

7. EDTDIR

ADDRESS F886H (63622D)
SIZE 11 bytes
PURPOSE Directory for EDIT in Basic

The EDIT command in Basic creates a temporary "DO" file. This slot is used for this file.

8. FILTAB

ADDRESS FB63H (64355D)
SIZE 2 bytes
PURPOSE Points to the address of the file data

This pointer points to the starting address of the file data area. The file data area consists of the FCB address. If the "MAXFILES" command in Basic was not executed after a "COLD START", this table has 4 bytes. The first 2 bytes point to the NULL files buffer (NULBUF points to the same address). The second 2 bytes point to the #1 file's FCB address.

9. FRETOP

ADDRESS FABFH (64191D)
SIZE 2 bytes
PURPOSE The top of the free string space

The highest address (closest to FFFFH) of the free string area is kept in this pointer. The lowest address is kept by STKTOP+1.

10. FSIDSV

ADDRESS F380H (62336D)
SIZE 2 bytes
PURPOSE Check if first power on or not

If FSIDSV is not identical to FRSTID (8A4DH), the initialization routine falls into the "COLD START" routine. If cold start occurs, all of the data files in the PC-8300A are cleared. The "COLD START" routine sets FRSTID to this address after the initialization is done. This ID value may not be changed.

11. HIMEM

ADDRESS F384H (62340D)
SIZE 2 bytes
PURPOSE Highest memory available for Basic

This pointer holds the highest memory address available for Basic. The area between this address and F380H is reserved for machine language files or the user's special working area. No standard program will break the data in this area except the POKE statement in

Basic. The POKE statement can write to anywhere in the RAM, so care should be taken when selecting an address for the POKE statement to store a machine language program or character data into the RAM area. The HIMEM can be changed by the second parameter of the CLEAR statement in Basic. Please refer to the PC-8300A Basic Reference Manual for more information on the CLEAR and POKE statements.

12. MEMSIZ

ADDRESS FA9AH (64154D)
SIZE 2 bytes
PURPOSE The highest location in memory

This pointer points to 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".

13. NULBUF

ADDRESS FB67H (64359D)
SIZE 2 bytes
PURPOSE Points to the address of the file buffer

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

14. NULDIR

ADDRESS F870H (63600D)
SIZE 11 bytes
PURPOSE Directory for non-registered programs

This area is kept for internal use. A non-registered program is a Basic program which has been just typed after selecting BASIC. This area points to the starting address of the Basic program. Please refer to the chapter on BASIC file handling.

15. SCRDIR

ADDRESS F87BH (63611D)
SIZE 11 bytes
PURPOSE Directory for the Scrap Area

The TEXT editor is capable of the following four functions, SELECT, CUT, COPY, and PASTE. This directory is used as a temporary file storage for the Scrap from TEXT. This file is created when some characters are SELECTed and COPYed or CUT (please refer to the PC-8300A User's Guide for more information on these terms). This file is kept even if one exits from TEXT, therefore the contents can be used in other programs

(ie, Basic or Telcom). If one CUTs or COPYs without first SELECTing, the starting address points to Control-Z, showing the Scrap file to be empty.

16. STKTOP

ADDRESS F459H (62553D)
SIZE 2 bytes
PURPOSE The top location to use for the stack

Initially STKTOP is set up by the INIT routine in ROM #), according to the memory size to allow for 256 bytes of string space. This value can be changed by the CLEAR command's first argument. The difference between MEMSIZ and STKTOP means the total string space. The 2 byte space between MEMSIZ and FILTAB are kept for the "VAL" function in Basic. The "VAL" function sets "Ø" at the end of the strings, after evaluating the strings. This two byte area prevents accidental over-write of the FCB area just above the FILTAB.

17. STREND

ADDRESS FAE9H (64233D)
SIZE 2 bytes
PURPOSE End of the storage in use

This pointer keeps the address just above the Array Table. The area between this pointer and the stack pointer can be used as the FREE area.

18. TXTEND

ADDRESS FA88H (64136D)
SIZE 2 bytes
PURPOSE The end of the Current Basic program

19. TXTTAB

ADDRESS F45DH (62557D)
SIZE 2 bytes
PURPOSE Pointer to the beginning of the current Basic program

20. USRDIR

ADDRESS F891H (63633D)
SIZE 231 bytes
PURPOSE Directory for the user's files.

This area is used for the "BA" files, "DO" files and "CO" files which the user makes. Up to 21 files can be registered. The end of the directory area is indicated by FFH (Directory Search Stopper).

17. VARTAB

ADDRESS FAE5H (64229D)
SIZE 2 bytes
PURPOSE Pointer to the simple variable space

This pointer keeps the starting address of the Variable

Table area just above the "CO" files.

CHAPTER 4 PART II of the BOOKKEEPING AREA

VRAM Area For The LCD

ADDRESS	FBC0H (64448D)
SIZE	640 bytes

Part II of the Bookkeeping area is used for the VRAM of the LCD (Liquid Crystal Display). In this area, data is stored as the ANSI character code (refer to Appendix 4 of the PC-8300A Basic Reference Manual). The LCD driver, installed just below the LCD panel, receives this character code and displays it on the LCD. A total of 320 characters (40 X 8) can be shown on the LCD panel at one time. Therefore only the second 320 bytes, from FD00H to FE3FH, are used for the VRAM. The first 320 bytes, FB00H to FCFFH, are used only when the TERM mode is selected in TELCOM. The "PREV" function key in TELCOM's TERM mode, shows the previous screen from TERM (please refer to the PC-8300A Users Guide for more information).

The data in the VRAM appears when the LCD driver is turned on. Please refer to Chapter XX, for information on the control sequence for the LCD management.

CHAPTER 5 . PART III of the BOOKKEEPING AREA

Bookkeeping Area for the BIOS

ADDRESS	FE40H to FFFFH (65088D - 65535D)
SIZE	447 bytes

This area includes the data area for the RS-232C driver, the buffers relevant to the Keyboard driver, and the working area for the LCD driver. Refer to Chapter XX - XX for information on how to use the peripheral drivers and the data in this area.

The variable FILTAB points to the lowest address of the file control data area. FILTAB points to the table of the starting address of the FCB (the FCB Offset) if the file is opened.

Example FILTAB and FCB

FILTAB (FB63H) ---------> F16AH

Dump memory (in hexadecimal)

```
F16A      6E F1 77 F2 .. .. .. ..
```

The first 2 bytes (F16EH) point to the starting address of the FCB of #0 file (NULL buffer). The second 2 bytes (F277H) is the top address of the FCB for file #1. These starting addresses are called FCBOFF (FCB offset address).

The FCB area for NUL and file #1 are allocated by the INITIALIZE routine in ROM #0. The remainder of the Offset for the FCB area is allocated by the Basic language command MAXFILES (refer to the PC-8300A Basic Reference Manual for more information).

The FCB consists of 9 bytes of parameter area and 256 bytes of buffer area except for NULBUF. NULBUF consists of only 256 bytes of buffer area. The purpose and the size of the parameters are listed below. Since the FCB can support a Floppy Disk File, there exists some meaningless parameters for RAM files. These parameters may be used for the users own purposes.

(1) FL.MOD- Null file mode for Open

ADDRESS FCBOFF+0
SIZE 1 byte
PURPOSE

The FL.MOD is the file mode of the FCB. If this byte is not set, this FCB is not used in Basic. If one obeys the Basic rules, you have to set a non-zero value here when you open a file.

1	INPUT only
2	OUTPUT only
8	APPEND only

(2) FL.FCA- First cluster allocated

ADDRESS FCBOFF+1

SIZE 1 byte
PURPOSE
The first cluster is allocated to a file. In the RAM file handling, this parameter has no meaning.

(3) FL.LCA- Last cluster accessed

ADDRESS FCBOFF+2
SIZE 1 byte
PURPOSE
The last cluster is accessed. For the RAM file open, this byte and the next are used for the storage of the Directory address of that RAM file.

(4) FL.LSA- Last sector accessed

ADDRESS FCBOFF+3
SIZE 1 byte
PURPOSE
The last sector accessed. For the RAM file open, this and the previous byte are used for the storage of the Directory address of that RAM file.

(5) FL.DSK- Name of disk drive on which file is opened

ADDRESS FCBOFF+4
SIZE 1 byte
PURPOSE
Disk number of the file or Device ID. The table listed below is the Device ID table in the PC-8300A.

DEVICE NAME	ID NUMBER
LCD	FFH
CRT	FEH
CAS	FDH
COM	FCH
WAND	FBH
LPT	FAH
RAM	F9H

(The CRT and WAND devices are optional I/O)

(6) FL.SLB- Size of last buffer read

ADDRESS FCBOFF+5
SIZE 1 byte
PURPOSE
The size of the last buffer read.

(7) FL.BPS- Current buffer position

ADDRESS FCBOFF+6
SIZE 1 byte
PURPOSE
The position in the buffer for both PRINT and INPUT

with the file #. One of the most important parameters in the FCB.

(8) FL.FLG- Attribute flag for this file

ADDRESS FCBOFF+7
SIZE 1 byte
PURPOSE

This byte and the next byte are used for the offset address of the RAM file which is currently opened. 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 the FCB. In reading or writing to the RAM file ("DO" file), the starting address and this offset show the next byte to be read or written.

(9) FL.OPS- Output position for tabs and commas

ADDRESS FCBOFF+8
SIZE 1 byte
PURPOSE

The high byte of the offset address for the RAM file.

(10) FL.BUF- Start of sector buffer (256 Bytes)

ADDRESS FCBOFF+9
SIZE 256 bytes
PURPOSE

Buffer for the file.

CHAPTER 7 NOTES

When manipulating RAM files without using the Basic's ROM routines, please note the following things, or the files may be broken.

1. Update the pointers (ASCTAB, BINTAB, STREND, etc.) if necessary.
2. Update the address data in the directory, if necessary.
3. Check for out of memory error
4. Set the directory correctly, when create new file.
5. Do not create more the 21 files!

CHAPTER 6
Sample program

Procedure to create new ".DO" file
1. Scan empty directory
2. Check out of memory
3. Make hole where new file should be stored
4. Store ASCII text at the hole
5. Update pointers
6. Update address data in directory
7. Set up directory

```
; Create new ".DO" file without to use BASIC's ROM routine.
START:
; Scan empty directory
          LXI   H,USRDIR        ; Is beginning address of user's
                                  directory
          LXI   B,0011D         ; Is length of directory
LOOP:
          MOV   A,M             ; Get directory flag
          CPI   0FFH            ; Is end of directory
          JZ    FLERR           ; Yes, empty directory does not
                                  exist
          ANI   80H             ; Check master bit (ie: Is the
                                  directory in use)
          JZ    FOUND           ; Found empty directory. Make HL to
                                  point next directory
          DAD   B
          JMP   LOOP            ; Check next directory
FOUND:
          SHLD  TEMP            ; Store address of empty directory
; Check out of memory error
; (ie: Check current SP is greater than the value that sum of
; new (STREND) and  200D.  Where  SP less than that, it is not
; possible that to create new file.)
          LHLD  STREND          ; Compute new (STREND)
          XCHG                  ;
          LHLD  LENGTH          ;
          DAD   D               ;
          LXI   B,0200D         ; add 200 to it
          DAD   B               ;
          XCHG                  ;
          LXI   H,0000H         ;
          DAD   SP              ; Compute SP and it
          MOV   A,H             ;
          SUB   D               ;
          JC    OMERR           ;
          JNZ   OK              ;
          MOV   A,L             ;
          SUB   E               ;
          JC    OMERR           ;

; Make hole at ASCTAB for new file that we want to create. (ie:
; block transfer; ".DO" files, ".CO" files, simple var and ar-
rays.)
```

```
OK:
          LHLD STREND           ; Save source address
          DCX  H                ;
          PUSH H                ;
          XCHG                  ;
          LHLD LENGTH           ;
          DAD  D                ;
          PUSH H                ; Save destination address
          LHLD STREND           ; Compute length of memory area
                                  that will be
                                ; transfered. (Length=[STREND]
                                  -[ASCTAB])
          XCHG                  ;
          LHLD ASCTAB           ;
          MOV  A,E              ;
          SUB  L                ;
          MOV  C,A              ;
          MOV  A,D              ;
          SBB  H                ;
          MOV  B,A              ;
          POP  D                ;
          POP  H                ;
          CALL LDDRSB           ; Block transfer
; Copy the ASCII text into the hole
          LHLD LENGTH           ; Get length of new file
          MOV  B,H              ;
          MOV  C,L              ;
          LHLD ASCTAB           ;Get beginning address where new
                                ; file should be stored
          XCHG                  ;
          LXI H,TEXT            ; Get beginning address where
                                ; ASCII text is stored
          CALL LDIRSB           ; Block transfer the text
; Update pointers
;    (BINTAB,VARTAB,ARYTAB,STREND)
          LHLD LENGTH           ; Get length of new file
          XCHG                  ;
          LHLD BINTAB           ; Update BINTAB
          DAD  D                ;
          SHLD BINTAB           ;
          LHLD VARTAB           ; Update VARTAB
          DAD  D                ;
          SHLD VARTAB           ;
          LHLD ARYTAB           ; Update ARYTAB
          DAD  D                ;
          SHLD ARYTAB           ;
          LHLD STREND           ; Update STREND
          DAD  D                ;
          SHLD STREND           ;
; Update pointers of ASCII file and machine language file
          LHLD LENGTH           ; Get length of new file
          XCHG                  ;
          LXI  H,NULDIR         ; Get beginning address of RAM
                                ; file's directory
          LXI  B,0011D          ;
```

```
UPP01:
            MOV   A,M               ; Get directory flag
            CPI   0FFH              ; Is end of directory
            JZ    ENDFIL            ;
            ANI   11110000B         ;
            CPI   11000000B         ; Is ASCII file in RAM?
            JZ    UPP02             ;
            CPI   10100000B         ; Is machine language file?
            JNZ   UPP03             ;
UPP02:
            INX   H                 ; Yes, ASCII file or machine
                                      language file
            MOV   A,E               ; Update pointer of file
            ADD   M                 ; (ie: add length of new file to
                                      old pointer)
            MOV   M,A               ;
            INX   H                 ;
            MOV   A,D               ;
            ADC   M                 ;
            MOV   M,A               ;
            DCX   H                 ;
            DCX   H                 ;
UPP03:
            DAD   B                 ; Make HL to point next directory
            JMP   UPP01             ; Check next directory
; Set up directory of new file
ENDFIL:
            LHLD  TEMP              ; Get address of empty directory
            MVI   A,11000000B       ; Is directory flag meaning ASCII
                                      file in RAM
            MOV   M,A               ;
            INX   H                 ;
            XCHG                    ; Set address of new file
            LHLD  ASCTAB            ;
            XCHG                    ;
            MOV   M,E               ;
            INX   H                 ;
            MOV   M,D               ;
            INX   H                 ; Set file name
            XCHG                    ;
            LXI   H,FILNAM          ; Is beginning address where file
                                    ; name is stored
            LXI   B,0008D           ; Is length of file name
            CALL  LDIRSB            ; All done
            RET                     ;

FLERR:
              .                     ; Filing limit error
              .                     ; Put your error handling routine
              .
OMERR:
              .                     ; Out of memory error
              .                     ; Put your error handling routine
              .
```

```
LENGTH:
        DW    0006D             ; Is length of ASCII text
FILNAM:
        DB    "SAMPLEDO"        ; Is file name
TEXT:
        DB    "Hello"           ; Contents of new ASCII file
        DB    1AH               ; as terminator of ASCII file
TEMP:
        DS    02D               ; Temporary storage to save address
                                ; of empty directory
```

PC-8300A BASIC INPUT/OUTPUT ROM ENTRY POINTS

Keyboard Driver

LCD Driver

Printer Drivers

RAM Bank Handlers

RAM File Handlers

Physical Cassette Drivers

Physical RS232C Drivers

Time Handlers

Sound Generator

Screen Editing

Error Handling

Miscellaneous Routines

CHAPTER 1 KEYBOARD DRIVER

1.1 CHSNS --- See if key is entered

Description

The CHSNS checks if character is ready in keyboard queue. CHSNS supports function keys and paste key.

Entry Name	CHSNS
Entry Address	1830H (62050)
Input parameter	None
Output parameter	Zero=1 if character is ready. Zero=0 if no character is ready.
Registers altered	[A] and flags

1.2 CHGET --- Get a character from the keyboard

Description

The CHGET reads a character from the keyboard. Paste and function keys are supported (ie: expanded into character string).

The CHGET also performs time-out checking. If specified time-out interval is gone, control directly goes to auto power down routine, which turns the machine down.

The time out error will occur if a key is not pressed. If this takes place the processor will initiate the auto power down routine.

Entry Name	CHGET
Entry Address	174DH (5965D)
Input parameter	None
Output parameter	[A]=Character typed
Register altered	[A] and flags

1.3 Sample program of CHGET

Following tiny program shows how to use CHGET (and CHPUT which will appear later in this document, and displays a character on LCD). The program is very simple, it reads a character from the keyboard and echoes it on LCD. Typing a ^C terminates the program and return to BASIC. The program is assumed to be executed by the BASIC EXEC statement.

The following sample program demonstrates the use of the routine CHGET. The routine reads a character from the keyboard and echos it to the LCD screen. The routine will perform this function until a Control-C is pressed.

```
;       Read keyboard and echoes all characters on LCD until a ^C is
        given.

ECHO:

        CALL  CHGET             ; Get a character from the keyboard
                                ; Result is passes in [A]
        CPI   03D               ; Is this a ^C?
        RZ                      ; Yes. Return to Basic
        CALL  CHPUT             ; Otherwise display it on LCD.
        JMP   ECHO              ; Loop until ^C is given
```

1.4 BREAKX --- Sense shift+stop keys

Description

The BREAKX is used to sense shift and stop keys. If both are depressed, the BREAKX returns carryset. Note that breakx senses the keyboard directory. CHGET and CHSNS works looking at the keyboard queue rather than seeing the keyboard. So, they do not work if interrupts have been disabled. The BREAKX, in turn, works already. The shift+stop is also queued in the keyboard queue as a ^C character if interrupts are enabled.

The BREAKX routine traps the "Shift" and "Stop" keys. If both are depressed the BREAKX routine returns a condition or carry flag set. If interrupts are enabled the SHIFT+STOP key combination generates a CONTROL-C.

********TEST********

Entry Name	BREAKX
Entry	72DFH (29407D)
Input parameter	None
Output parameter	Carry= 1 If shift+stop have been depressed.
	Carry=0 Shift+Stop not detected.
Registers altered	[A] and flags

2.1 CHPUT --- Display a character on console

Description

The CHPUT displays a character on the system console. The character is placed at the current cursor position. The cursor postion is incremented after the character is displayed. The cursor position may be altered via the "POSIT" routine or via an ESC command.

2.1.1 CHPUT parameters

Entry Name	CHPUT
Entry Address	4363H (17251O)
Input parameter	[A]=Character to be displayed
Output parameter	None
Register altered	A

2.1.2 ESC commands

CHPUT accepts ESC commands and several control characters as well as normal printable characters. Following ESC commands are supported by CHPUT.

The following demonstrates the use of ESC commands via the "CHPUT" routine.

Example:

```
$CHPUT    EQU  04363h
ESC       EQU  27d
          MVI  A,ESC       ;27 decimal,1b hex
          CALL CHPUT

          MVI  A,"j"       ;Clear the screen command
          CALL CHPUT
          END
```

ESC j	Clear the screen
ESC E	Clear the screen
ESC K	Erase to end of line
ESC J	Erase to end of screen
ESC I	Erase entire line
ESC L	Insert a blank line
ESC M	Delete current line
ESC Y (ypos)(xpos)	Locate cursor to ((ypos),(xpos))
ESC A	Move cursor up
ESC B	Move cursor down
ESC C	Move cursor right
ESC D	Move cursor left
ESC H	Move cursor Home
ESC p	Enter reversed video mode
ESC q	Escape from reversed video mode
ESC P	Turn on the cursor
ESC Q	Turn off the cursor
ESC T	Set system line
ESC U	Reset system line
ESC V	Lock screen
ESC W	Unlock screen

The "Set system line" and "Reset system line" commands are used to display function keys and any system messages on the bottom line (ie: "Memory full" error message in TEXT mode). The "Set system line" command allocates the bottom line as a system display line. This line is not used for normal character display.

The "Set System Line" does not display the function key definition. The routine "DSPFNK" Must be used to display the function keys. The "Reset System Line" does not erase the function key line. To erase the function key line use the "ERAFNK" routine.

Note: Once the system line is set, the cursor cannot be located on the system line. To write message on the system line, temporarily reset the system line during putting characters on it.

2.1.3 Control Characters

CHPUT accepts the following control characters.

07D	Beep
08D	Back space
09D	Tab
10D	Line feed
11D	Home
12D	Clear the screen
13D	Carriage return
27D	Invoke ESC command
28D	Move cursor right
29D	Move cursor left
30D	Move cursor up
31D	Move cursor down

2.2 DSPFNK --- Display function keys

Description

The DSPFNK routine is used to display function keys on the system line. If the system line has not been enabled, the DSPFNK automatically enables the system line.

Entry Name	DSPFNK
Entry Address	42E4H (17124D)
Input parameter	None
Output parameter	None
Register altered	All

2.3 Setting function key string

Function Key definitions are stored in the "System Work Keeping Area". 16 bytes are allocated for each function key definition.

Strings must be 15 bytes long. Null fill the string if the definition is less than 15 bytes. The 16th byte of the definition must be null character.

i.e. Function Key 5 set in the following sample program.

```
; Set string "COMMAND" to the function key 5.
FNKSTR    EQU  0F6A5H              ; Function key string area

SETFNK:
          LXI  D,MYSTRG            ; Where original is stored
          LXI  H,FNKSTR+16*5       ;Address of function key 5.
                                   ; each function key consist of
                                     16 bytes
          MVI  B,16                ; Length of  an entry
MOVFNK:
          LDAX D                   ; Copy a character
          MOV  M,A                 ;
          INX  D                   ;
          INX  H                   ;
          ORA  A                   ; All character have been
                                     copied?
          JZ   FILNUL              ; Yes. Fill the rest by NULLs.
          DCR  B                   ; More room in entry?
          JNZ  MOVFNK              ; Yes. Keep copying
          DCX  H                   ; Too long string
                                   ; Force last character to be
NULL:
          MVI  M,00                ;
          RET                      ;

FILNUL:
          DCR  B                   ; Fill the rest of entry by
                                     NULLs
          RZ                       ; All done
          MVI  M,00                ; Put a NULL
          INX  H                   ;
          JMP  FILNUL              ; Keep filling
MYSTRG:
          DB   'COMMAND'           ; String to be set to function
                                     key 5
          DB   00                  ; Terminator
```

2.4 ERAFNK --- Erase function key line

Description

The ERAFNK routine is used to erase the system line and reset the system line. The ERAFNK routine may be used to erase function keys or any message previously displayed on the system line. The ERAFNK routine does nothing if the system line has not been enabled.

Entry Name	ERAFNK
Entry Address	42C3H (17091D)
Input parameter	None
Output parameter	None
Registers altered	All

2.5 Sample program of DSPFNK and ERAFNK

The following sample program performs the following steps. The routines "DSPFNK" and "ERAFNK" will be used.

1) Assign "Hello" to function key 1.
2) Display all 5 function keys.
3) Wait form keyboard input.
4) Erase function key line.
5) Return to Basic

```
; Sample program of DSPFNK and ERAFNK

        LXI   D,FNKSTR          ; Set "Hello" into function
                                  key #1
        LXI   H,HELLO           ; Where "Hello" stored now
        LXI   B,0016            ; Length of the string
        CALL  LDIRSB            ; Copy the string
                                ; LDIRSB is a ROM routine
                                  which
                                ; is similar to Z-80's
                                ; LDIR instruction
        CALL  DSPFNK            ; Then display function key
        CALL  CHGET             ; Wait for any key typed
        CALL  ERAFNK            ; Erase the function key from
                                  LCD
        RET                     ; All done
                                ; Return to BASIC
HELLO:
        DB    'Hello'           ; String to be set  in
                                  function key #1
        DB    0,0,0,0,0         ; Fill by NULL
        DB    0,0,0,0,0         ;
        DB    0                 ;
```

2.6 POSIT --- Locate cursor

Description

This function positions the cursor on the display.

The POSIT routine is functionally equivalent to the "Locate cursor" ESC command. The difference is that the "Locate cursor" command uses coordinates with offset of 32, while POSIT interprets the given coordinates as 1 relative.

Entry Name	POSIT
Entry Address	42BFH (17087D)
Input parameter	[H]=X position [L]=Y position
Output parameter	None
Registers altered	[A],[H],[L] and flags

To locate the cursor to the 5th character on the top line, the following code will be used.

```
; Locate the cursor to the 5th character on the LCD top line

        LXI   H,5*256+1         ; X=5,Y=1
                                ; Since POSIT uses coordinates
                                ; relative to 1. Home is (1,1)
        CALL POSIT              ; Place cursor there
```

2.7 PLOT --- Set a dot on LCD

Description

The PLOT routine sets a dot at the specified dot position on the LCD. Coordinates are relative to 0. The PLOT routine is not affected by the current system line setting. In other words, the PLOT routine can even draw a dot on the system line while the system line is being set.

The PLOT routine does not error check dot coordinates. If a position is given out of the range of video memory, the results are unpredictable.

Entry Name	PLOT
Entry Address	74D0H (29904D)
Input parameter	[D]=X position of the dot
	Valid Range: (0<= X <=239)
	[E]=Y position of the dot
	Valid Range: (0<= Y <=63)
Output parameter	None
Registers altered	All

2.8 UNPLOT --- Reset a dot on LCD

Description

The UNPLOT routine performs opposite function to the PLOT routine described in the previous section. The UNPLOT resets a dot at the specified dot position.

The UNPLOT routine does not error check dot coordinates. If a position is given out of the range of video memory, the results are unpredictable.

Entry Name	UNPLOT
Entry Address	74D1H (29905D)
Input parameter	[D]=X position of the dot
	Valid Range: (0<= X <=239)
	[E]=Y position of the dot
	Valid Range: (0<= Y <=63)
Output parameter	None
Registers altered	All

2.9 Sample program of PLOT and UNPLOT

The following program illustrates how to use PLOT and UNPLOT routines. The program works just as two PC-8800 N88 BASIC statements below are executed.

```
LINE (10,10)-(100,50),7,BF
LINE (20,20)-(90,40),0,BF
```

```
; Sample program of PLOT and UNPLOT routines.

        LXI   D,10*256+10       ; Fill a box of (10,10)
                                  -(100,50)
                                ; Set up starting position
        MVI   H,100             ; Ending X position
FILLOP:
        PUSH  D                 ; Save X position
        CALL  SETLIN            ; Draw a horizontal line from
                                ; ([D],[E]) to ([H],[E])
        POP   D                 ;
        INR   E                 ; Bump Y position
        MOV   A,E               ;
        CPI   51                ; All line where draw?
        JC    FILLOP            ; No, draw next line
        LXI   D,20*256+20       ; Now large box was filled.
                                ; clear small box inside of
                                  the large box.
        MVI   H,90              ; Ending point of X
CLRLOP:
        PUSH  D                 ;
        CALL  CLRLIN            ; Clear a horizontal line of
                                ; ([D],[E]) to ([H],[E])
        POP   D
        INR   E                 ; Bump Y position
        MOV   A,E               ; See if all lines
        CPI   41E               ; are erased
        JCI   CLRLOP            ; No. Begin to PRESET the next
                                  line
        RET                     ; All done
                                ; Return to BASIC.Note that
                                ; this program should be
                                ; invoked by EXEC statement
```

```
; Draw a horizontal line of ([D],[E])-([H],[E])

SETLIN:
            PUSH H                      ; Save ending X in [H]
            PUSH D                      ; Save current position
            CALL PLOT                   ; Set a dot at ([D],[E])
            POP  D                      ;
            POP  H                      ;
            INR  D                      ; Advance current point right
                                          a dot
            MOV  A,H                    ; See if all done
            CMP  D                      ;
            JNC  SETLIN                 ; No. Set next dot
            RET                         ;


; Clear a horizontal line of ([D],[E])-([H],[E])

CLRLIN:
            PUSH H                      ; Save ending X
            PUSH D                      ; Save current X and Y
            CALL UNPLOT                 ; Reset a dot at ([D],[E])
            POP  D                      ;
            POP  H                      ;
            INR  D                      ; Move a dot right
            MOV  A,H                    ; See if reached to ending X
            CMP  D                      ;
            JNC  CLRLIN                 ; No, reset next dot
            RET                         ;
```

2.10 How to interrogate the current cursor position

The CHPUT routine always keeps track of the current cursor position. The current cursor position is stored in the system book keeping area. The cursor position is stored at the addresses shown below. The values stored at these locations are 1 relative.

CSRX	F3E6H (62438D)	Current X position
CSRY	F3E5H (62437D)	Current Y position

CHAPTER 3 PRINTER DRIVERS

Description

The PRINT routine prints a character to system printer. All interrupts except RS-232C are disabled. Shift+Stop will abort the PRINT routine.

3.1 PRINT --- Output a character to the printer

Entry Name	PRINT	
Entry Address	6D92H (28050D)	
Input parameter	[A]=Character code to be printed	
Output parameter	Carry=0	If successful.
	Carry=1	If aborted by shift+stop.
Registers altered	Flags	

3.2 How to check if printer is ready

There is no routine or function which indicate the status of the system printer. The following sample program will list the steps necessary to get the status of the printer.

```
; Check printer status routine

PORTC     EQU  0BBH              ; Printer status port

PRTSNS:
     IN   PORTC                  ; Get printer status
     ANI  00000110B              ; Check BUSY and SELECT bit
     XRI  00000010B              ; Set zero if printer is ready
     RET                         ; On return zero=1 shows printer is
                                   ready.
                                 ; zero=0 denotes printer is
                                   busy.
```

CHAPTER 4

Ram Bank Handlers

The PC-8300 contains 64K bytes of memory. The memory is divided into two banks of memory (32K bytes each). An additional 32k byte ram cartridge may be added to the PC8300.

The ROM provides two primitive routines which access data from these memory banks. Bank selection is made in blocks of 32K bytes. Switching banks without these routines will cause application programs to halt, because the processor will be unable to find the next executable instruction.

Care should be taken when switching memory banks. i.e If the RS-232C is opened while running in memory bank 1 and processor control is transferred to memory bank two without closing the information channel; any characters received will be queued in memory bank 2. Communications protocol set up in bank 2 could be different from the protocal set up in bank 1.

Information regarding the bank control hardware can be found in the hardware reference manual.

Using the ROM supplied bank handlers will eliminate any problems which could arise during memory bank switching. This is possible because the routines will run while the interrupts are disabled.

4.1 GETBNK --- Read a byte from any RAM bank

Description

Read a byte from any memory bank. The memory bank to read does not have to be the current memory bank. The routine temporarily changes the current memory bank reads the byte and then restores memory to the previous bank.

Note: Interrupts should be disabled before calling the GETBNK routine.

Entry Name	GETBNK
Entry Address	7EECH (32492D)
Input parameter	[B]= Bank Number 00:Bank #1 Main Bank 08:Bank #2 0C:Bank #3 [HL]=Address of the byte to be read
Output parameter	[D]=Byte read
Registers altered	[A],[C],[D] and flags

4.2 PUTBNK --- Write a byte into any RAM bank

Description

Write a byte from any memory bank. The memory bank to write does not have to be the current memory bank. The routine temporarily changes the current memory bank write the byte and then restores memory to the previous bank.

Note: Interrupts should be disabled before calling the GETBNK routine.

Entry Name	PUTBNK
Entry Address	7EEBH (32491D)
Input parameter	[B]= Bank Number 00:Bank #1 Main Bank 08:Bank #2 0C:Bank #3 [HL]=Address of the byte to be written
Output parameter	[D]=Byte read
Registers altered	[A],[C],[D] and flags

4.3 Sample program for different bank access

The following example shows how to use the PUTBNK routine.

```
; Change current year to 84
          MVI   D,04D             ; New year is 84
                                  ; Write low digit first
          MVI   B,00D             ; Always write it in bank 1
                                  ; (standard RAM)
          LXI   H,TIMBUF+10D      ; Where year is maintained
                                  ; in the bookkeeping area
          DI                      ; Disable the interrupt
          CALL  PUTBNK            ; Write low digit
                                  ; Then write upper digit of
                                  ; the year
          MVI   D,08D             ;
          CALL  PUTBNK            ; Write it
          EI                      ; All done, allow all further
                                  ; interrupts to come
```

CHAPTER 5 RAM FILE HANDLERS

There are three kinds of RAM files manipulated by PC-8300A, namely text files (xxxxxx.DO), BASIC binary program files (xxxxxx.BA), and machine code files (xxxxxx.CO). Only the text files can be read and written by usual OPEN,READ,WRITE and CLOSE calls (Section 11.1 through 11.5). Deleting and renaming are supported for all types of files.

The RAM file manipulation routines only work on existing files. To open a nonexisting file please refer to chapter 6 "Ram file System" in the "Internal Structure of the PC-8300A" manual.

5.1 Opening a RAM file

The open a RAM file, the NULOPN routine can be used. Before calling this routine, file name should be set up in FILNAM.

The NULOPN routine (and INDSKC,OUTFL1 and CLSFIL routine in subsequent sections as well) supports only text files whose extension is ".DO". BASIC binary program files and machine code file cannot be accessed by these routines.

5.1.1 Setting up file name

The file name should be formatted in FILNAM as below.

FILNAM=FB78H
FILNAM thru FILNAM+5

The name of file may up to six characters long (plus the extension). If the name is less than six characters long the remaining bytes should be filled with spaces. The characters in the name of the file may upper or lowercase. The routine "NULOPN will convert all the characters to uppecase.

FILNAM+6 thru FILNAM+7

The extension is stored here. It is always "DO". ("D" is stored in FILNAM+6, "O" is stored in FILNAM+7).

FILNAM+8

Always a space character.

5.1.2 NULOPN --- OPEN FILE

Description

Opens an existing file specified in FILNAM.

If any error was detected during opening a file, control directly goes to BASIC's error routine.

Once the file name is correctly set in FILNAM, now NULOPN can be called.

Entry Name	NULOPN
Entry Address	4EBEH (20158D)
Input parameter	[A]= File number [E]= Open mode 1-Input mode 2-Output mode 8-Append mode [D]=Always F9H (0371D) [HL]=Should point to a NULL character (00)
Output parameter	None
Registers altered	All

5.1.3 FINPRT --- Post processing of OPEN

After the file was successfully opened, you should call "FINPRT" routine. The FINPRT is the post-processor of the NULOPN and sets up some book keeping area correctly.

Entry Name	FINPRT
Entry Address	0F26H (3878D)
Input parameter	None
Output parameter	None
Registers altered	[A] and flags

5.1.4 Sample OPEN code

Shown below is a sample code to open a file, "FILE", for input. File number 1 is used.

```
; Open the file whose name is "FILE" for input using channel #1.
          LXI   D,MYFILE              ; File name is stored now
          LXI   H,FILNAM              ; Where file name is set up
          MVI   B,09D                 ; Length of the name
SETFNM:
          MOV   A,M                   ; Move file name into FILNAM
          STAX  D                     ;
          INX   D                     ;
          INX   H                     ;
          DCR   B                     ;
          JNZ   SETFNM                ;
          MVI   A,01D                 ; Use channel 1
          LXI   D,0F9H*100H+1         ; Open mode is for input
                                      ; [D] is always F9H
          LXI   H,EOL                 ; [HL] should point to a NULL
                                        character
          CALL  NULOPN                ; Open the file
          CALL  FINPRT                ; Set up some variables in
                                      ; system book keeping area
MYFILE:
          DB    "FILE  DO "           ; Name of the file to be
                                        opened
EOL:
          DB    00D                   ; Null character used to call
                                        NULOPN
```

5.2 Reading a character from a RAM file

Description

Read a character from a file.

After a file was opened, reading a character out of the file can be performed by two ROM routines, namely SETFIL and INDSKC.

The SETFIL sets up a file pointer to the file table specified by the file number passed in accumulator. INDSKC reads the character into register A and sets the carry flag if end-of-file is reached.

Entry Name	SETFIL
Entry Address	4E6BH (20075D)
Purpose	Set up file pointer to the file control block
Input parameter	[A]=File number
Output parameter	None
Registers altered	Possibly all
Entry Name	INDSKC
Entry Address	5015H (20501D)
Purpose	Read a character from a file
Input parameter	None
Output parameter	Carry flag set (1) if end of file has been reached: Carry flag reset if successful [A]=Character read
Registers altered	[A] and flags

The following demonstrates the use of SETFIL and INDSKC.

```
; Read a byte from the file whose file number is in [A]

        CALL SETFIL             ; Set up file table pointer of
                                ; file [A]
        CALL INDSKC             ; Try to read a byte
        JC   EOF                ; EOF detected
                                ; Otherwise, [A] contains
                                ; character read
```

5.8 FILOU1 - Writing a character to a RAM file

Description

Write a character to a file.

To write a character to the file that has been opened by the NULOPN routine, SETFIL and FILOU1 are used. FILOU1 writes a character in [A] to the file prepared by the SETFIL call which always precedes the FILOU1 call.

If an error occurs during the write, control is transferred Basic's error routines.

Entry Name	FILOU1
Entry Address	4FEFH (20463D)
Purpose	Write a character to RAM file
Input parameter	[A]=Character to be written
Output parameter	None
Registers altered	None

```
; Write a character in [A] to the file in channel 2.

        PUSH PSW                ; Save a character
        MVI  A,02D              ; Set up channel
        CALL SETFIL             ;
        POP  PSW                ; Restore character
        CALL FILOU1             ; Output it to the file
```

5.4 CLSFIL - Closing a RAM file

Description

Close a Ram File. If an error is detected control is transferred to Basic's error handler.

Entry Name	CLSFIL
Entry Address	4EE4H (20196D)
Purpose	Close a file
Input parameter	[A]=File # of the file to be closed
Output parameter	None
Registers altered	All

5.5 Closing all files

Description

The CLSALL routine closes all files. Not only RAM files but also other devices supported by BASIC's generalized device I/O scheme are closed (this is also true for CLSFIL routine).

Entry Name	CLSALL
Entry Address	4FBFH (20415D)
Purpose	Close all files
Input parameter	None
Output parameter	None
Registers altered	All except [HL]

5.6 Sample file I/O program

The following small program shows how to use file I/O routines. The program copies a text file "SOURCE" to destination file "COPY". If the file named "COPY" already exists, the old one is deleted. This program uses two file channels. Give MAXFILES command in BASIC before trying this program so two or more channels can be used.

```
; Sample file I/O program
; Copy everything in a file "SOURCE" to a file "COPY"
        LXI   H,SOURCE              ; Open "SOURCE" for input as
                                      file #1
                                    ; First set up FILNAM
        LXI   D,FILNAM              ;
        LXI   B,0009D               ; Length of file name
        CALL  LDIRSB                ; Transfer file name into
                                      FILNAM
        MVI   A,01D                 ; Use channel #1
        LXI   D,0F9H*0FFH+01D       ; Open mode is "for input"
        LXI   H,EOL                 ; [HL] should point to null
                                      character
        CALL  NULOPN                ; Do open the file
        CALL  FINPRT                ; Set up something
        LXI   H,COPY                ; Then open destination file
                                    ; for output as #2
        LXI   D,FILNAM              ;
        CALL  LDIRSB                ;
        MVI   A,02D                 ; Use channel #2 for output
                                      file
        LXI   D,0F9H*0FFH+02D       ; Open mode is "for output"
        LXI   H,EOL                 ; [HL] should point to null
                                      character
        CALL  NULOPN                ; Do open output file
        CALL  FINPRT                ; Magic word!
COPLOP:
        MVI   A,01D                 ; Then read a character from
                                    ; source file
        CALL  SETFIL                ; Set up pointer for file #1
        CALL  INDSKC                ; Get a character
        JC    ISEOF                 ; End of file reached
        PUSH  PSW                   ; Save the character
        MVI   A,02D                 ; Then set up for output
        CALL  SETFIL                ;
        POP   PSW                   ; Reget character
        CALL  FILOU1                ; Output it to file #2
        JMP   COPLOP                ; Keep copying until EOF
                                    ; is reported
ISEOF:
        MVI   A,01D                 ; Everything was copied. Close
                                    ; all files
        CALL  CLSFIL                ;
        MVI   A,02D                 ;
        CALL  CLSFIL                ;
        RET                         ; Return to BASIC
                                    ; This program assumes to be
                                    ; called by EXEC statement
```

```
                                        ; from BASIC.
SOURCE:
        DB    "SOURCE"                  ; Source file name
        DB    "DO "                     ;
COPY:
        DB    "COPY   "                 ; Destination file name
        DB    "DO "                     ;
EOL:
        DB    00H                       ; NULOPN requires [HL] to
                                        ; point a null character
```

5.7 Killing a RAM file

The ROM supports three routines, KILASC, KILBAS and KILCOM, each of which deletes a text file, BASIC binary program file and machine code file, respectively. All of them need special setup before being used.

LNKFIL is used to fix up directory structure.

SRCNAM searches for the specified file, and returns some variables referred to by KILASC, KILBAS or KILCOM routines. SRCNAM may also be used to see if a certain file exists, and to establish the type of the file and physical address where the file is stored in RAM. Description of the SRCNAM routine will be found in later section.

5.7.1 LNKFIL --- Fix up directory structure

Description

The LNKFIL routine fixes up all possible incomplete "Links" between files and their directories. Each file is associated with its directory and a link exists between them. During file I/O, there is a chance that the link is not properly maintained. More precisely, there is a chance that the link is intentionally left incorrect assuming it to be fixed up later during further file I/O . Since deleting a file may be made while other file I/O is in progress, and some links may not yet have been fixed up, all the links should be fixed explicitly before the deletion is performed.

Entry Name	LNKFIL
Entry Address	233AH (9018D)
Purpose	Fix up directory pointers
Input parameter	None
Output parameter	None
Registers altered	All

5.7.2 KILASC --- Killing a text file

Description

The KILASC is used to delete a text file.

Entry Name	KILASC
Entry Address	21A8H (8616D)
Purpose	Delete a text (.DO) file
Input parameter	Are given by SRCNAM call
Output parameter	None
Registers altered	All

```
; Delete the text file whose file name is "MYTEXT.DO"

        LXI   D,MYFILE          ; Where file name of the file
                                ; to be deleted is stored now
        LXI   H,FILNAM          ; Move it here. SRCNAM always
                                ; search for a file whose name
                                ; is set in FILNAM
        MVI   B,09D             ; Length of the name
COPFIL:
        LDAX  D                 ; Do copy the filename
        MOV   M,A               ;
        INX   H                 ;
        INX   D                 ;
        DCR   B                 ;
        JNZ   COPFIL            ;
        CALL  LNKFIL            ; Fix up possible missing
                                  links
        CALL  SRCNAM            ; Search for the file to be
                                ; deleted
        JZ    NOTFND            ; The file was not found
                                ; give an error, or do an
                                  everything
                                ; what you like
        MOV   B,A               ; Save file type. File type is
                                ; returned from SRCNAM in [A]
        ANI   02D               ; See if this file is open
        JNZ   FILOPN            ; Yes. Perform your error
                                  processing
        MOV   A,B               ; Reget file type
        ANI   40H               ; see if text file bit is set
        JZ    NOTASC            ; Not text file
        CALL  KILASC            ; This is a text file. Let's
                                  delete it
NOTFND:
        Your error processing routine for the case when file is
        not found
NOTASC:
        Your error processing routine for the case when the
        specified file is not text file
FILOPN:
        Your error processing routine for the case when the
        file is open

MYFILE:
        DB    "MYTEXT"          ; File to be deleted
        DB    "DO "             ; It's extension
```

5.7.8 KILBAS --- Killing a BASIC binary program file

Description

KILBAS is used to delete a BASIC binary file.

Entry Name	KILBAS
Entry Address	2200H (8704D)
Input parameter	Given by SCRNAM call
Output parameter	None
Registers altered	All

```
; Delete the text file whose file name is "TEST.BA"

        LXI   D,MYFILE          ; Where file name of the file
                                ; to be deleted is stored now
        LXI   H,FILNAM          ; Move it here. SRCNAM always
                                ; search for a file whose name
                                ; is set in FILNAM
        MVI   B,09D             ; Length of the name

COPFIL:
        LDAX  D                 ; Do copy the file name
        MOV   M,A               ;
        INX   H                 ;
        INX   D                 ;
        DCR   B                 ;
        JNZ   COPFIL            ;
        CALL  LNKFIL            ; Fix up possible missing
                                  links
        CALL  SRCNAM            ; Search for the file to be
                                  deleted
        JZ    NOTFND            ; The file was not found.
                                ; Give an error or do anything
                                ; what you like
        ANI 60H                 ; See if this is really a
                                ; BASIC binary program file
        JNZ   NOTBAS            ; Not BASIC binary program
                                  file
        CALL KILBAS             ; This is a BASIC binary file.
                                ; Let's delete it

NOTFND:
        Your error processing routine for the case when file is
        not found
NOTBAS:
        Your error processing routine for the case when the
        specified file is not BASIC binary program file
MYFILE:
        DB    "TEST   "         ; FIle to be delete
        DB    "BA "             ; It's extension
```

5.7.4 KILCOM --- Killing a machine code file

Description

The KILCOM is used to delete a machine code file.

Entry Name	KILCOM
Entry Address	21C2H (8642D)
Input parameter	Are given by SRCNAM call
Output parameter	None
Registers altered	All

```
; Delete the machine code file whose file name is "PASCAL.CO"

        LXI   D,MYFILE          ; Where file name of the file
                                ; to be deleted is stored now
        LXI   H,FILNAM          ; Move it here. SRCNAM always
                                ; search for a file whose name
                                ; is set in FILNAM.
        MVI   B,09D             ; Length of the name

COPFIL:
        LDAX  D                 ; Do copy the file name
        MOV   M,A               ;
        INX   H                 ;
        INX   D                 ;
        DCR   B                 ;
        JNZ   COPFIL            ;
        CALL  LNKFIL            ; Fix up possible missing
                                  links
        CALL  SRCNAM            ; Search for the file to be
                                  deleted
        JZ    NOTFND            ; The file was not found.
                                ; Give an error or do anything
                                ; what you like
        ANI   20H               ; See if machine code file bit
                                ; is set
        JZ    NOTCOM            ; Not a machine code file
        CALL  KILCOM            ; This is a machine code file.
                                ; Let's delete it

NOTFND:
        Your error processing routine for the case when file is
        not found.

NOTCOM:
        Your error processing routine for the case when the
        specified file is not a machine code file.

MYFILE:
        DB    "PASCAL"          ; File to be deleted
        DB    "CO "             ; It's extension
```

5.8 Renaming a RAM file

Description

Rename a File.

Note: The two character extension of the filename should remain the same.

If the file to be renamed does not exist or the new file name is already in use control is transferred to Basic's error handler.

Entry Name	NAMEB
Entry Address	228BH (8747D)
Purpose	Rename a file
Input parameter	Old file name is stored in FILNAM. New file name is stored in FILNM2. FILNM2=FB81H Note: FILMN2 uses the same format as FILNAM.
Output parameter	None
Registers altered	All except [HL]

5.9 Search for a RAM file

Description

Search for a filename stored at FILNAM and return its address in RAM.

Note: Directory links should be fixed up before calling the SRCNAM routine.

Entry Name	SRCNAM
Entry Address	229BH (8859D)
Input parameter	File name should be set up in FILNAM
Output parameter	ZeroFlag Set if file does not exist. The following information is valid only when ZeroFlag is reset (0). [A]=File information Bit 7=Always set Bit 6=Set if text file Bit 5=Set if machine code file Bit 4=Set if built in utility (i.e. BASIC,TEXT or TELCOM) Bit 3=Internal use Bit 2=Internal use Bit 1=Set if the file is open Bit 0=Internal use [HL]=Pointer to the directory entry for this file [DE]=Address of the file
Registers altered	All

5.10 Sample program to search for a file

The Following code demonstrates the use of the SRCNAM routine. The routine below searches for a text file. If the file exists the routine will count the number of bytes in the file.

Note: A text file is terminated by a Control-Z (Hex 1A).

```
; Count the number of bytes in the text file whose name is
"SAMPLE.DO". The
; number is returned in [HL]. End of file mark is also included
in the result.
CHKLEN:
          LXI   D,MYFILE          ; Where the name of the file
                                  ; is stored
          LXI   H,FILNAM          ;Copy is here since SRCNAM
                                  ; always gets file
                                  ; name out of here.
          MVI   B,09D             ; Length of the file name
COPFIL:
          LDAX  D                 ;
          MOV   M,A               ;
          INX   D                 ;
          INX   H                 ;
          DCR   B                 ;
          JNZ   COPFIL            ;
          CALL  LNKFIL            ; Fix up all possible incor
                                    rect links
          CALL  SRCNAM            ; Search for the file
          JZ    NOTFND            ; No such a file in RAM.
          ANI   40H               ; File exist. Make sure this
                                  ; is a text file
          JZ    NOTASC            ; Not a text file. Blow him up
          LXI   H,0000D           ; Reset # of bytes in the file
LENLOP:
          LDAX  D                 ; Get a byte out of the file
          INX   D                 ;
          INX   H                 ; Increment # of bytes seen
          CPI   1AH               ; Is this the end of the file?
          JNZ   LENLOP            ; No. Keep counting
          RET                     ; All done. [HL] holds # of
                                  ; bytes in the file
MYFILE:
          DB    "SAMPLE"          ;
          DB    "DO "             ;
NOTFND:
          Place error routine for the case when file not found.
NOTASC:
          Place error routine for the case when the file is not a
          text file.
```

5.11 Changing maximum file number (MAXFILES)

Description

Changes the maximum number of files set up by Basic's "MAXFILE" command.

Updates the file buffer allocation area. This routin also performs a partial file related pointer setup. The remaing file related pointers should be set up by using the CLEARO routine.

Note: DEFILE requires some overhead.

1) Routine CLSALL must be executed to close all files.

2) Routine CLEARO must be executed for initialization.

The DEFILE updates the stack pointer. As long as the DEFILE has been originally designed to be used by BASIC, and BASIC allocates its stack space below the file buffers, BASIC needs stack pointer to be updated whenever the file buffer allocation is changed.

The DEFILE updates the stack pointer. DEFILE was designed to work with BASIC. Basic allocates its stack space below the file buffer area. The stack pointer must be updated whenever the buffer allocation is changed.

Note: Everything push on the stack during an application is lost after the DEFILE routine is executed. Therefore, the return address of basic is also lost. To return to Basic a "JMP" must be invoked.

Entry Name	DEFILE
Entry Address	7F1CH (32540D)
Input parameter	[A]=Number of files
Output parameter	None
Registers altered	All. Stack pointer is also changed

5.11.1 CLEARO - Reset Environment of BASIC

Description

This routine is used in conjuction with the DEFILE routine. CLEARO performs a reset of the pointer setups.

Entry Name	CLEARO
Entry Address	3FFCH (16380D)
Input parameter	None
Output parameter	None
Registers altered	All

```
Change the number of files. New number is passed in [A]
READY   EQU  062DH
        PUSH PSW                ; Save new number of files
        CALL CLSALL             ; Close all open files if any
        POP  PSW                ; Reget # of files
        CALL DEFILE             ; Change the number
        CALL CLEARO             ; Set up some pointers
        JMP  READY              ; Return to BASIC command
                                ; level directly. We cannot
                                  use
                                ; "RET" here even if this
                                ; program was invoked by
                                ; EXEC statement since return
                                ; address in stack has been
                                  lost
```

5.12 How to acquire the current MAXFILES value.

Basic keeps track of the maximum number of files allowed (MAXFILES) in its book keeping area. The MAXFILES always holds the current MAXFILES value.

```
MAXFILE=FB62H                ; Current Maxfiles value
```

6.1 CMTRMT --- Motor control

Description

The CMTRMT (for CMT remote control) is used to turn the cassette motor on or off. The CMTRMT routine will be used just to control cassette motor without subsequent cassette I/O operation. In this sense, CMTRMT is similar to the MOTOR command in BASIC. To turn the motor on for subsequent cassette read/write operation, there is no need to call the CMTRMT routine explicitly since CSRDON or CWRTON routine automatically controls the cassette motor. To turn the motor off after all cassette I/O was finished, use CTOFF.

Entry Name	CMTRMT
Entry Address	6FD9H (28633D)
Input parameter	[E]=0 Turns motor off [E]<>0 Turns motor on
Output parameter	None
Registers altered	[A] and flags

6.2 DATAW --- Write a byte to cassette

Description

The DATAW routine writes a byte stored in [A] to cassette tape. The Baud rate for the write is 600. If SHIFT+STOP is pressed the write is aborted. Before using this routine the cassette motor should be turned on. Generally the CWRTON routine is called to set up for a cassette write operation, including turning the motor on.

Entry Name	DATAW
Entry Address	6FEBH (28651D)
Input parameter	[A]=Character to be written
Output parameter	Carry set if aborted by SHIFT+STOP Carry reset if successful
Registers altered	[A],[B],[C],[D],[E] and flags

3 DATAR --- Read a byte from cassette

Description

The DATAR routine is used to read a byte from the cassette tape drive. (Data must have been written at 600 Baud).

The DATAR routine does not check the stop bit (DATAW writes two stop bits). So, there is no framing or overrun condition. This was done to give the caller enough time to process the byte read before next byte arrives.

Before using the DATAR routine, cassette motor should be turned on.

Entry Name	DATAR
Entry Address	708EH (28814D)
Purpose	Read a byte from cassette tape
Input parameter	None
Output parameter	Carry set if aborted by SHIFT+STOP
	Carry reset if successful
	[A]=Character read from tape
Registers altered	[A],[B],[E],[H],[L] and flags

6.4 CSRDON --- Set up cassette for read

Description

The CSRDON routine is used prior to any read operation. This routine turns on the cassette motor checks for the carrier. If SHIFT+STOP is pressed while this routine is waiting for carrier control is transferred to the I/O error handler after the cassette motor is turned off.

The CSRDON routine disables all further interrupts. The interrupts are usually reenabled when CTOFF is called.

Interrupts should be disalbed to handle serial to parallel conversion and parallel to serial conversion.

Entry Name	CSRDON
Entry Address	18EFH (6383D)
Input parameter	None
Output parameter	None
Registers altered	All

Typical cassette read operation flow is show below.

```
;-Typical cassette read flow

          CALL CSRDON                ; Turn on motor, wait for car-
                                       rier

; To detect correct beginning of the data, special sync characters are usually
; written on the tape.
; Wait for the sync characters to come
; Actual sync character code and its number are implementation dependent



GETSYN:
          MVI  B,10D                 ; Wait for sync character
SYNLOP:
          PUSH B                     ;
          CALL DATAR                 ; Read a byte
          POP  B                     ;
          JC   ABORT                 ; If SHIFT+STOP
          CPI  SYNCHR                ; Is this my sync character?
          JNZ  GETSYN                ; No. Reset # of sync
                                       characters seen
                                     ; And reset from the beginning
          DCR  B                     ; This is my sync char.
                                     ; See if enough sync
                                       characters
                                     ; are seen
          JNZ  SYNLOP                ; No. Keep eating
; Sync characters have been recognized
; Now read data bytes and process them
          CALL DATAR                 ; Get data byte
          JC   ABORT                 ; If SHIFT+STOP
               .
               .
               .
          CALL DATAR                 ; Get next data
          JC   ABORT                 ; If SHIFT+STOP
               .
               .
               .

; All data have been processed
; Finish cassette read operation
          CALL CTOFF                 ; Turn off motor
```

6.5 CWRTON --- Set up cassette for write

Description

Initialize cassette tape drive to perform tape write operations. This routine writes out a null header to the tape. There is no error condition. All further interrupts are disabled. Interrupts are enabled by the CTOFF routine.

Note: Interrupts should be disabled during cassette I/O.

Entry Name	CWRTON1
Entry Address	1901H (6401D)
Input parameter	None
Output parameter	None
Registers altered	All

6 CTOFF ---- Turn off cassette motor
Description

Turn off the cassette motor and enable interrupts.

Entry Name	CTOFF
Entry Address	1915H (6421D)
Input parameter	None
Output parameter	None
Registers altered	[A],[D],[E] and flags

6.7 Sample cassette I/O program

The folowing routine demonstrates the use of CSRDON, DATAR, and CTOFF. The routine will read a file created by N-BASIC and display all the data in that file. N-Basic does not terminate files with an End-Of-File marker. To terminate this routine press Control-C.

```
; Type a cassette data file created by PC-8000 N-BASIC

CASLOP:
        CALL CSRDON             ; Turn on cassette motor and
                                ; wait for carrier This
                                  program
                                ; will be aborted if user
                                  typed
                                ; a SHIFT+STOP during waiting
                                ; for carrier and control
                                  comes
                                ; back direct to BASIC from
                                ; CSRDON. If you want to avoid
                                ; this, use "ERRJMP".
GETSYN:
        MVI  B,06H              ; Wait for 6 sync characters
                                ; (9CH) to come
SYNLOP:
        PUSH B                  ;
        CALL DATAR              ; Read a byte from cassette
        JC   ABORT              ; If aborted
        CPI  9CH                ; Is this a sync characters?
        JNZ GETSYN              ; If not, Reset # of sync
                                ; characters seen and restart
                                ; from very beginning
        DCR  B                  ; This is my sync  characters
                                ; See if 6 sync characters
                                  were seen
        JNZ  SYNLOP             ; No. Continue
        LXI H,BUF               ; Beginning of record was
                                  found.
                                ; Read the record into buffer
                                  called BUF.

REDLOP:
        CALL DATAR              ; Read next characters
        JC   ABORT              ; If aborted by SHIFT+STOP
        MOV  M,A                ; Put it in to buffer
        INX  H                  ;
        CPI  13D                ; End of record?
        JNZ  REDLOP             ; No. keep eating
        CALL CTOFF              ; Everything in this record
                                  was read.
                                ; Stop cassette motor
        LXI  H,BUF              ; Then display this record on
                                  LCD
DSPLOP:
```

```
        MOV   A,M               ; Pick up a character out of
                                  buffer
        CALL  CHPUT             ; Display it on LCD
        INX   H                 ;
        CPI   13H               ; Everything was displayed?
        JNZ   DSPLOP            ; No. Display next
        MVI   A,10D             ; Do line feed
        CALL  CHPUT             ;
        JMP   CASLOP            ; Then try to read next record
                                  again
ABORT:
        CALL  CTOFF             ; Aborted by SHIFT+STOP.Turn
                                  off
                                ; cassette motor and allow
                                  further
                                ; interrupts to come.
        MVI   A,07D             ; Beep to tell abort request
                                  is accepted
        CALL  CHPUT             ;
        RET                     ; And return to BASIC.
```

CHAPTER 7 PHYSICAL RS232C DRIVERS

7.1 INZ232 --- Initialize RS-232C port

Description

The INZ232 routine initializes the USART, sets up the baud rate, parity control, data length, stop bits and RTS and DTR lines. RS232C queue (receive queue. Transmission is not queued) is cleared. In PC-8300A, the USART is used by three devices, namely RS-232C port, floppy disk interface and another serial interfaced storage device. The INZ232 routine selects RS-232C port. If the USART has been used by another device, DU error is reported. Xon/Xoff control and SI/SO control is enabled or disabled by the current default settings (one displayed by the STAT command in Term).

Entry Name: INZ232

Entry Address: 6F58H (28504D)

Input parameter:

[H]=Baud rate specifier
1=75 baud
2=110 baud
3=300 baud
4=600 baud
5=1200 baud
6=2400 baud
7=4800 baud
8=9600 baud
9=19200 baud

[L]=Data length,parity and stop bit specifier

Bit 4	Bit 3	Data Length
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

Bit 2	Bit 1	Parity
0	0	Odd parity
0	1	Even parity
1	x	No parity

Bit 0	Stop bit
0	1 bit
1	2 bits (*1)

(*1) When data length is 5 bits, stop bits is 1.5 bit

[B]=RTS and DTR specifier

Bit 7	RTS
0	1 active
1	0 inactive
Bit 6	**DTR**
0	1 active
1	0 inactive

[C]=Should be FFH

Output parameter	None
Registers altered	None

7.2 SETSER --- Initialize RS-232C port using mode string

Description

The SETSER routine also initializes the RS-232C port. In contrast to the INZ232 routine, the communications protocal is determined by the "mode string". The mode string is ASCII string that has identical format to the string used in the STAT command in TELCOM and OPEN statement in BASIC.

The PC-8300A maintains current default mode string (the mode string specified by most recent STAT command, OPEN statement or SETSER routine itself). The default mode string is called "SERMOD". Following short routine initializes RS-232C port using current mode setting.

Entry Name	SETSER
Entry Address	1C4EH (7246D)
Input parameter	[HL]=Points to the mode string
Output parameter	None

```
; Initialize RS-232C port in current default mode
        LXI  H,SERMOD           ; Where default mode string is
                                  stored
        CALL SETSER             ; Do init using it

        SERMOD=F406H
```

3 SD232C --- Send a character to RS-232C port

Description

The SD232C routine sends a character passed in [A] to the RS-232C port. The RS-232C port must be initialized before using SD232C routine.

The SD232C routine performs Xon/Xoff control if enabled. When Xon/Xoff is enabled, and Xoff (^S) is received, the SD232C waits for Xon (^Q) to come before sending a character.

In addition, the SD232C performs SI/SO control if enabled.

Entry Name	SD232C
Entry Address	6EBEH (28350D)
Input parameter	[A]=Character to be sent
Output parameter	Carry reset if successful Carry set if aborted by SHIFT+STOP
Registers altered	[A] and flags

7.4 RCVX --- Check for character is ready in RS-232C queue

Description

The RCVX routine is used to see if there is a character in the RS-232C queue. Before using this routine, RS232C port must be initialized by the INZ232 routine.

Entry Name	RCVX
Entry Address	6DC2H (28098D)
Input parameter	None
Output parameter	Zero flag set if no character is in queue Zero flag reset if character(s) are in queue. When the Zero flag is reset,[A] shows number of characters in the queue
Registers altered	[A] and flags

5 RV232C --- Get a character from RS-232C queue

The RV232C routine is used to take a character out of the RS-232C receive queue. If no character is available, the RV232C waits for character to come through the port.

The RV232C routine detects a SHIFT+STOP request. If detected, the RV232C is aborts and the Carry Flag is set upon return.

If a character is waiting in the queue the RV232C routine takes it out of the queue. If the recieve is successful the character is returned in the accumulator and the Zero flag is set.

If Xon/Xoff control is enabled queue control handled internally. If the queue becomes full XON/XOFF sends a Control-S and waits for the queue to be emptied. When the queue becomes empty XON/XOFF sends a Cntrol-Q to resume the reception of data. Also, if a Control-S or a Control-Q is recieved they trapped by the ROM routines and not are not sent to the caller.

If SI/SO control is enabled, the RV232C routine appends or removes MSB of the received character before passing it to the caller. SI/SO control characters themselves are not passed to the caller, but they are stripped off internally, unless SI/SO control is disabled.

If the received character in the queue is erroneous, the RV232C returns Zero reset to indicate the existence of an error. In this case, contents of [A] are meaningless.

Entry Name	RV232C
Entry Address	6DD3H (28115D)
Input parameter	None
Output parameter	Carry set if aborted by SHIFT+STOP. Carry reset and Zero flag set if successful [A] holds the character picked up out of the queue. Carry reset and Zero flag reset if there is a receive error [A] contains a garbage character in this case.
Registers altered	[A] and flags.pa

7.6 CLSCOM --- Deactivate RS-232C port

Description

The CLSCOM routine deactivates the RS-232C port. DTR and RTS lines are reset to a false state to minimize power consumption. The multiplexer is released so other devices may use the USART. The CLSCOM routine should be called after RS-232C operation is finished. Otherwise, any attempt to use the floppy disk will result in a DU error.

Entry Name	CLSCOM
Entry Address	6FA8H (28584D)
Input parameter	None
Output parameter	None
Registers altered	[A] and flags

ENABLX --- Enable Xon/Xoff control

Entry Name	ENABLX
Entry Address	6F8DH (28557D)
Input parameter	None
Output parameter	None
Registers altered	[A] and flags

Description

The ENABLX is used to enable Xon/Xoff control for future RS-232C I/O. This routine should be called before INZ232 is called to initialize RS-232C port.

7.8 DSABLX --- Disable Xon/Xoff control

Description

Disable XON/XOFF control. This routine should be prior to the calling of the INZ232 routine.

Entry Name	DSABLX
Entry Address	6F8EH (28558D)
Input parameter	None
Output parameter	None
Registers altered	[A] and flags

9 How to enable/disable SI/SO control

The only way to enable or disable SI/SO is change the memory address which holds the SI/SO status. This should be done prior to calling the INZ232 routine.

```
; Enable SI/SO control
ENABLS:
          MVI   A,"S"                ;
          STA   SERMOD+5             ; Enable SI/SO control
          RET                        ;
; Disable SI/SO control
DSABLS:
          MVI   A,"N"                ;
          STA   SERMOD;5             ; Disable SI/SO control
          RET                        ;
```

Address of SERMOD which tells default RS-232C characteristics.

SERMOD=F406H

CHAPTER 8 TIME HANDLERS

8.1 TIMRD --- Read current time and date

Description

The TIMRD routine is used to read the current time and date (day and month only) from calendar clock chip. The time and date are stored into the buffer pointed by [HL] in the following format.

[HL]	Lower digit of second	(0-9)
[HL]+1	Upper digit of second	(0-5)
[HL]+2	Lower digit of minute	(0-9)
[HL]+3	Upper digit of minute	(0-5)
[HL]+4	Lower digit of hour	(0-9)
[HL]+5	Upper digit of hour	(0-2)
[HL]+6	Lower digit of day	(0-9)
[HL]+7	Upper digit of day	(0-3)
[HL]+8	Day of the week	(0:Sun - 6:Sat)
[HL]+9	Month	(1:Jan - C:Dec)

Note: Interrupts should be disabled prior to the call to this rouine.

Entry Name	TIMRD
Entry Address	7359H (29529D)
Input parameter	[HL]=Pointer to buffer where the current time and data are read into.
Output parameter	Current time and date are stored in the buffer
Registers altered	[A],[B],[C],[D],[H],[L] and flags.

Typical TIMRD usage

```
        LXI   H,TIMBUF          ; Where time and date are read
                                  into.
        DI                      ; Disable interrupt
        CALL  TIMRD             ; Read current time and date
        EI                      ; Allow further interrupt to
                                  come
```

The ROM uses a buffer located at F832h to store the time and date. This buffer may be used when calling the TIMRD routine. TIMBUF + 10 contains the lower digit of the year specified by the Date$ statemenin Basic, TIMBUF+10 contains the upper didgit of the year.

Note: The year bytes are always maintained in bank #1 (main bank). To read the bytes when current bank is not main bank, use GETBNK routine.

TIMBUF=F832H

8.2 TIMSET --- Set time and date

Description

Reset the time and date on the calendar clock chip.

Note: Interrupts should be disabled prior to execution of this routine.

Entry Name	TIMSET
Entry Address	735AH (29530D)
Input parameter	[HL]=Pointer to the buffer where new time and date are set.

[HL]	Lower digit of second	(0-9)
[HL]+1	Upper digit of second	(0-5)
[HL]+2	Lower digit of minute	(0-9)
[HL]+3	Upper digit of minute	(0-5)
[HL]+4	Lower digit of hour	(0-9)
[HL]+5	Upper digit of hour	(0-2)
[HL]+6	Lower digit of day	(0-9)
[HL]+7	Upper digit of day	(0-3)
[HL]+8	Day of the week	(0:Sun - 6:Sat)
[HL]+9	Month	(1:Jan - C:Dec)

Output parameter	None
Registers altered	[A],[B],[C],[D],[H],[L] and flags

CHAPTER 9 SOUND GENERATOR

9.1 MUSIC --- Generate sound

Description

The MUSIC routine is used to generate sound of specified frequency and length. Interrupts are disabled during the sound routine. If SHIFT+STOP key sequence is detected, ROM aborts immediately.

Note: No data is passed back to the calling routine. No error processing is generated if an error occurs.

Entry Name	MUSIC
Entry Address	730DH (29453D)
Input parameter	[DE]=Frequency. [DE] should be less than 0C000H [B]=Length. "0" is interpreted as 256. Frequency and length are interpreted in exactly same way as the SOUND statement in BASIC
Output parameter	None
Registers altered	[A],[B],[C] and flags

CHAPTER 10 SCREEN EDITING

There are two screen editor routines, namely PINLIN and INLIN. In BASIC, PINLIN is used for program input, while data input. (ie: INPUT and LINE INPUT statements) uses the INLIN.

The only one difference in PINLIN and INLIN is the PINLIN always scans lines from the position where cursor was located when INLIN was called. This eliminates the prompt string from being included in the result string.

10.1 PINLIN --- Program input

Entry Name	PINLIN
Entry Address	4798H (18328D)
Input parameter	None
Output parameter	Carry set if PINLIN is aborted by STOP or ^C . Carry reset if successful. (ie: PINLIN is terminated by a carriage return). Entered string is stored in BUF terminated by a null character.
Registers altered	All

Description

The PINLIN is a screen oriented editing routine. PINLIN supports the same cursor and control keys that Basicsupports.

Control is returned to the calling routine when "RETURN", "STOP", or Control-C is pressed. When "Return" is pressed the entered string is passed in BUF. The string is terminated by a null. The carriage return is not stored with the entered string.

The PINLIN always begins to scan a logical line from the beginning. That is if a prompt message was displayed before the PINLIN was called, the string is also considered as a part of the entered string. For such application in that a prompt string is used just like INPUT statement in BASIC, use instead of INLIN. If PINLIN is terminated by a STOP or a ^C, PINLIN returns with carry set. And nothing is stored in BUF (F5A1H).

12.2 INLIN --- Data input

Description

Get a line of data from the screen oriented editor.

The routine scans the line from the cursor position prior to the inlin call as long as the cursor stays on the same line when the routine is terminated.

Entry Name	INLIN
Entry Address	47AAH
Input parameter	Same as PINLIN
Output parameter	Same as PINLIN
Registers altered	Same as PINLIN

CHAPTER 11 ERROR HANDLING

11.1 Taking control on error

Some the of the Rom routines decribed in this manual transfer process control to Basic's error handler. To prevent control from returning to Basic's error handler the ROM provides a method of keeping control of the process.

To exercise this option the memory location "ERRJMP" F3FEh (Bookeeping area) must contain the address where control should be returned to.

When Basic's error handler in invoked the routine checks the value at F3FEh. If the value is 0 (0 is the Default) Basic keeps control. However, if the value is not 0 control is transferred to the routine at that address.

When control returns to the calling routine the error number is stored in register E.

Note: The stack contains garbage when an error occurs. It is the responsibility of the programmer to reset the stack pointer to the correct value.

After control is regained after an error the contents of ERRJMP must be reset to 0.

11.2 Sample program for ERRJMP handling

The following sample program illustrates the use of error trapping technique as described in section 11.1. The routine performs the following.

1) Call routine CSRDON
2) Press SHIFT+STOP while waiting for carrier.
3)

```
; CSRDON call. Use ERRJMP so control comes back to me even when ;
the CSRDON was

; aborted by SHIFT+STOP.

ERRJMP    EQU  0F3FEH              ; Error hook entry
          LXI  H,0000D             ; Get current stack pointer
          DAD  SP                  ;
          SHLD MYSTACK                ; Save it so we can reset
stack
                                   ; in case of error
          LXI H,MYERROR            ; Set up my own error handler
                                   ; address so BASIC does not
                                     take
                                   ; control in case of an
                                   ; error
          SHLD ERRJMP              ;
          CALL CSRDON              ; All setups where over
                                   ; Let's call CSRDON now
          LXI  H,0000D             ; Successfully done.
                                   ; Reset ERRJMP so standard
                                   ; error processing can be
                                   ; activated for further errors
          SHLD ERRJMP              ;
               .
               .
               .
MYERROR:
          MOV  A,E                 ; Error during CSRDON
                                   ; Get error number
          CPI  24D                 ; Make sure this is I/O error
          JNZ  OOPS                ; Should never happen!
                                   ; Handle specially
          LHLD MYSTACK             ; Was I/O error. Reset stack
                                     pointer
          SPHL                     ;
               .
               .
               .
OOPS:
               .                   ; Special handler in case of
               .                   ; internal error here
```

CHAPTER 12 MISCELLANEOUS ROUTINES

Block Transfer Routines

12.1 LDIRSB --- Simulate Z80's LDIR instruction

Description

The LDIRSB simulates Z80's LDIR instruction. Only one difference is the LDIRSB alters [A] and flags.

Entry Name	LDIRSB
Entry Address	6C78H (27768D)
Input parameter	Same as Z80's LDIR [HL]=Source address [DE]=Destination address [BC]=Length
Output parameter	Same as Z80's LDIR instruction
Register altered	All

2 LDDRSB --- Simulate Z80's LDDR instruction

Description

The LDDRSB simulates Z80's LDDR instruction. Similar to the LDIRSB, LDDRSB also alters [A] and flags.

Entry Name	LDDRSB
Entry Address	6C83H (27790)
Input parameter	Same as Z80's LDDR instruction [HL]=Source address [DE]=Destination address [BC]=Length
Output parameter	Same as Z80's LDDR instruction
Register altered	All

CHAPTER 1 KEYBOARD DRIVER..... 2
1.1 CHSNS --- See if key is entered.....
1.2 CHGET --- Get a character from the keyboard.....
1.3 Sample program of CHGET.....3
1.4 BREAKX --- Sense shift+stop keys.....5

CHAPTER 2 LCD DRIVERS.....6
2.1 CHPUT --- Display a character on console.....6
2.1.1 CHPUT parameters.....6
2.1.2 ESC commands.....7
2.1.3 Control Characters.....10
2.2 DSPFNK --- Display function keys.....11
2.3 Setting function key string.....12
2.4 ERAFNK --- Erase function key line.....13
2.6 POSIT --- Locate cursor.....15
2.7 PLOT --- Set a dot on LCD.....16
2.8 UNPLOT --- Reset a dot on LCD.....17
2.9 Sample program of PLOT and UNPLOT.....18
2.10 How to interrogate the current cursor position.....20

CHAPTER 3 PRINTER DRIVERS.....21
3.1 PRINT --- Output a character to the printer.....21
3.2 How to check if printer is ready.....22

CHAPTER 4.....23
4.1 GETBNK --- Read a byte from any RAM bank.....24
4.2 PUTBNK --- Write a byte into any RAM bank.....
4.3 Sample program for different bank access.....

CHAPTER 5 RAM FILE HANDLERS.....27
5.1 Opening a RAM file.....28
5.1.1 Setting up file name.....28
5.1.2 NULOPN --- OPEN FILE.....29
5.1.3 FINPRT --- Post processing of OPEN.....30
5.1.4 Sample OPEN code.....31
5.2 Reading a character from a RAM file.....32
5.8 FILOU1 - Writing a character to a RAM file.....33
5.4 CLSFIL - Closing a RAM file.....34
5.5 CLSALL - Closing all files.....35
5.6 Sample file I/O program.....36
5.7 Killing a RAM file.....38
5.7.1 LNKFIL --- Fix up directory structure.....39
5.7.2 KILASC --- Killing a text file.....40
5.7.8 KILBAS --- Killing a BASIC binary program file.....42
5.7.4 KILCOM --- Killing a machine code file.....44
5.8 Renaming a RAM file.....46
5.9 Search for a RAM file.....47
5.10 Sample program to search for a file.....48
5.11 Changing maximum file number (MAXFILES).....49
5.11.1 CLEARO - Reset Environment of BASIC
5.12 How to acquire the current MAXFILES value.....52

CHAPTER 6 PHYSICAL CASSETTE DRIVERS.....53
6.2 DATAW --- Write a byte to cassette.....54
6.3 DATAR --- Read a byte from cassette.....55
6.4 CSRDON --- Set up cassette for read.....56
6.5 CWRTON --- Set up cassette for write.....58
6.6 CTOFF ---- Turn off cassette motor.....59
6.7 Sample cassette I/O program.....60

CHAPTER 7 PHYSICAL RS232C DRIVERS.....62
7.1 INZ232 --- Initialize RS-232C port.....62
7.2 SETSER --- Initialize RS-232C port using mode string.....64
7.3 SD232C --- Send a character to RS-232C port.....65
7.4 RCVX --- Check for character is ready in RS-232C queue.....66
7.5 RV232C --- Get a character from RS-232C queue.....67
7.6 CLSCOM --- Deactivate RS-232C port.....68
7.8 DSABLX --- Disable Xon/Xoff control.....70
7.9 How to enable/disable SI/SO control.....71

CHAPTER 8 TIME HANDLERS.....72
8.1 TIMRD --- Read current time and date.....72
8.2 TIMSET --- Set time and date.....74

CHAPTER 9 SOUND GENERATOR.....75
9.1 MUSIC --- Generate sound.....75

CHAPTER 10 SCREEN EDITING
10.2 INLIN --- Data input.....77

CHAPTER 11 ERROR HANDLING.....78
11.1 Taking control on error.....78
11.2 Sample program for ERRJMP handling.....79

CHAPTER 12 MISCELLANEOUS ROUTINES.....80
Block Transfer Routines.....80
12.1 LDIRSB --- Simulate Z80's LDIR instruction.....80
12.2 LDDRSB --- Simulate Z80's LDDR instruction.....81

PC-8300 HARDWARE

CHAPTER 1 LOGICAL SPECIFICATION

CPU
ROM
RAM
MEMORY STRUCTURE
BANK SWITCHING ARCHITECTURE
LCD INTERFACE
PRINTER INTERFACE
CALENDAR CLOCK INTERFACE
KEYBOARD INTERFACE
SERIAL INTERFACE
CASSETTE (CMT) INTERFACE
BARCODE READER INTERFACE
INTERRUPT FUNCTION
MODEM INTERFACE
SYSTEM SLOT
MEMORY CONTROL CIRCUIT

PC-8300 HARDWARE

1. LOGICAL SPECIFICATION

1.1 CPU

1) CPU to be used
80C85A with operation clock 2.4 MHz

2) Reset action
Power on reset
Manual reset: "Warm Start"- pressing reset switch
"Cold Start"- pressing reset switch while holding both the shift key and the control key down.

1.2 ROM

1) Device
128K bytes of CMOS ROM named ROM #0 split into 4 banks of 32K byte ROM blocks. Bank switching is performed by the I/O port shown below.

OUT A3H

	1	0
	RADR2	RADR1

RADR2	RADR1	Bank Selection of ROM #0
0	0	: Bank #A (00000H-07FFFH)
0	1	: Bank #B (08000H-0FFFFH)
1	0	: Bank #C (10000H-17FFFH)
1	1	: Bank #D (18000H-1FFFFH)

ROM IC	ADDRESS
Bank #A	00000H
Bank #B	08000H
Bank #C	10000H
Bank #D	18000H
	1FFFFH

Device:
UPD23C1000 or equivalent

Bank #A should be selected on power-on

2) User ROM

One chip (32K bytes) ROM named ROM #1 can be installed by the user in the vacant IC socket.

The software can switch between the 128K bytes of standard ROM #0 and the 32K bytes of user ROM #1.

1.3 RAM

1) Standard RAM

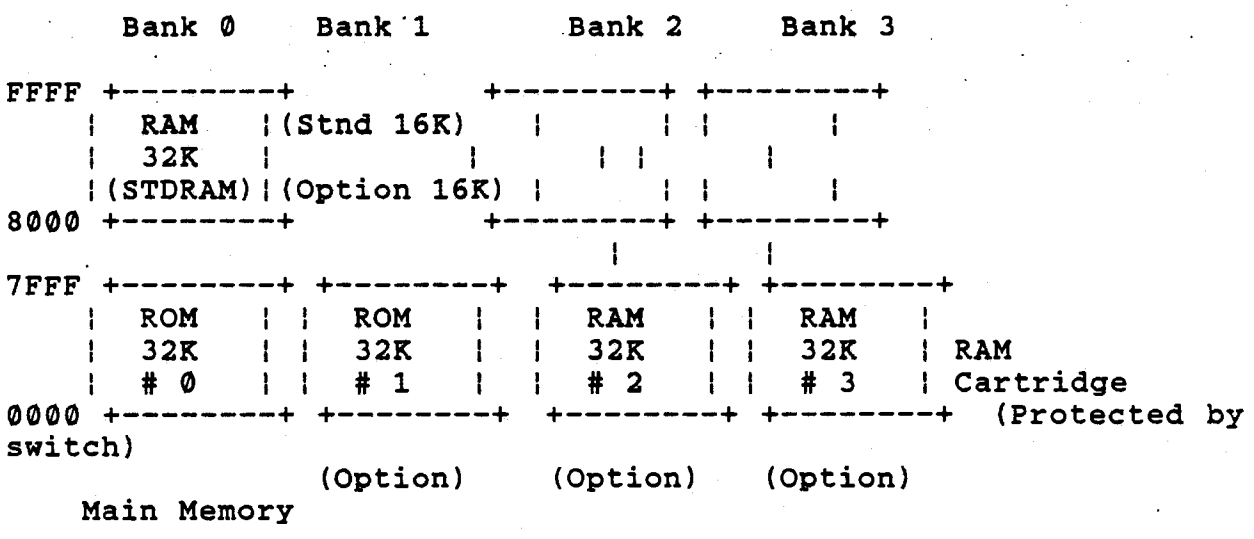
8 CMOS 2K bytes chips [Standard 64K bytes (STD RAM)]

2) Option RAM

There is a 32K bytes RAM Cartridge (Bank #3) available as an external memory upgrade. It can be connected to the system bus of the PC-8300.

RAM #1	uPD4364G- 15LL type x4
RAM #2	uPD4364G- 15LL type x4
RAM #3	utilize PC-8206A

1.4 Memory structure



Main Memory

RAM #2 and RAM #3 can be located both in low or high address, from 0 to 7FFFH or high address from 8000H to FFFFH. This selection can be done by Port access.

Memory can be controlled by the following I/O port

1010	0001	OUT A1H (161D)

3	2	1	0
----	----	----	----	HADR 2	HADR 1	LADR 2	LADR 1

LADR2	LADR1	Address 0000-7FFF selection
0	0	Bank #0 (ROM #0)
0	1	Bank #1 (ROM #1)
1	0	Bank #2 (RAM #2)
1	1	Bank #3 (RAM #3)

HADR2	HADR1	Address 8000-FFFF selection
0	0	Standard RAM
0	1	Not used
1	0	Bank #2 (RAM #2)
1	1	Bank #3 (RAM #3)

In case of using 128K bytes ROM cassette, the following I/O port will be used for switching

1000	0000	OUT 80H

	ROMSEL	A16

A16: Address 16
ROMSEL 128K bytes of ROM cartridge select

0 : Deselect
1 : Select

Low Address for 128K bytes of ROM Cartridge

```
+----+----+
|1000|0100|    OUT 84H
+----+----+
```

7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0

A7 - A0 ROM cartridge low address

High Address for 128K bytes of ROM Cartridge

```
+----+----+
|1000|1000|    OUT 88H
+----+----+
```

7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A9	A8

A15 - A8 ROM cartridge high address

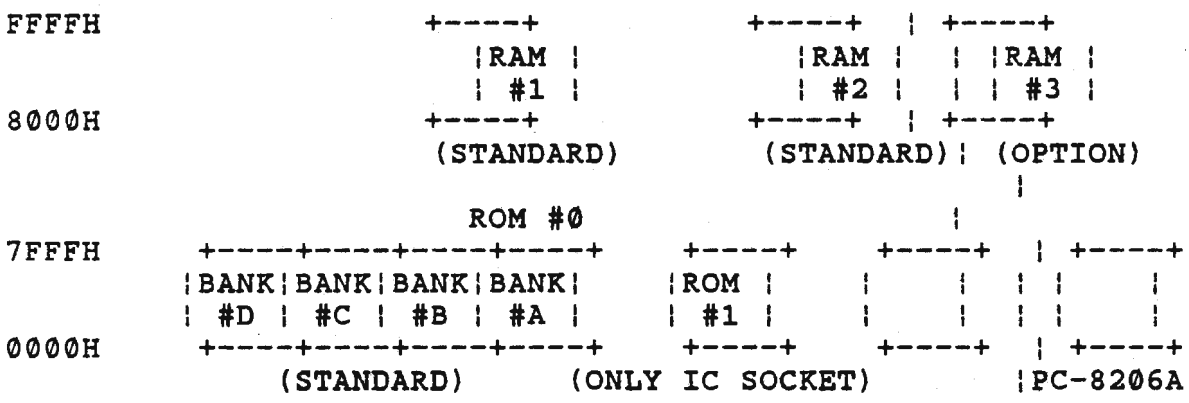
Read Data Out of ROM Cartridge

```
+----+----+
|1000|1100|    OUT 8CH
+----+----+
```

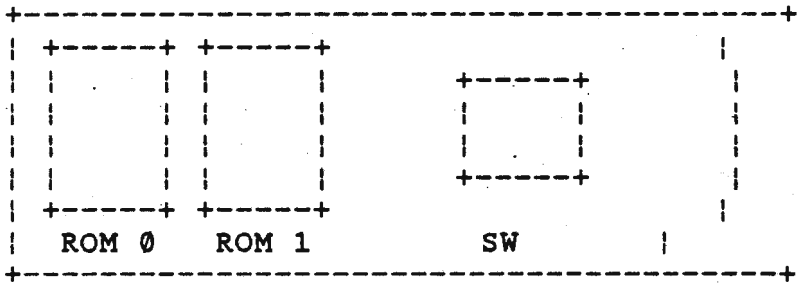
7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0

D7 - D0 ROM data

PC-8300 MEMORY STRUCTURE



PC-8300 Memory Chip Allocation for Expansion



1.5 Bank Switching Architecture

The heart of the PC-8300A is the Intel 80C85, which is an 8-bit processor whose address bus is 16-bit, Thus the 80C85 can access 64K of memory at a time. In the PC-8300A, there is a special memory access function, memory-bank switching, supported. Therefore the 64K barrier in the 8-bit microprocessor can be tricked in the PC-8300A.

The RAM in the PC-8300A 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).

RAM #2 and RAM #3 can be located in two different positions, lower position is from 0000H to 7FFFH and higher position is from 8000H to FFFFH. RAM #3 is a detachable RAM cartridge. The bank-switching is executed every 32K bytes, therefore it is impossible to access half of RAM #1 and half of RAM #2 at the same time. In otherwords, one can not set up the following memory allocation, the lower half of RAM #2, 8000H to BFFFH, and the higher part of RAM #1, C000H to FFFFH as 32K of memory.

RAM #2 and RAM #3 can be protected by a "PROTECT SWITCH". The switch for RAM #2 is located on the rear of the computer. The switch for RAM #3 is located on the side of the cartridge. RAM #1 does not have a protect switch. When the protect switch is ON, the RAM bank can not be used. The PC-8300A uses the highest RAM area, F380H to FFFH, to save the current status of the PC-8300A every time.

All of the RAM chips are backed-up by battery. All of the data and program files in RAM will be kept, even if the power switch is turned off. If one makes a special utility for the 2nd ROM or a special RAM configuration, one has to consider this Power-down sequence. Refer to Chapter XX, to understand the Power-off trap in ROM #0.

1.6 LCD Interface

1) Driver to be used
   HD44102B (10 chips) .... Segment drivers
                            Display RAM = 200 bytes
                            Selected by Port A/B of PPI (81C55)
   HD44013B (2 chips) ..... Common drivers

2) LCD to be used
   LR-202C ..... 240 x 64 dots

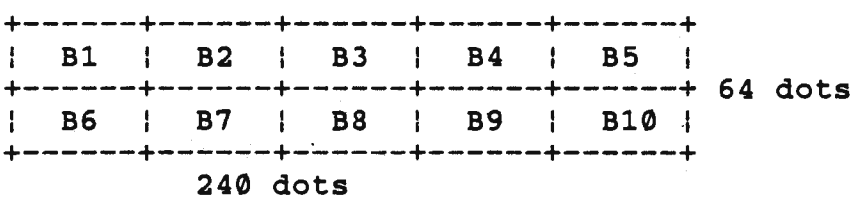
3) Display function
   1. No of display characters
      40 characters per line by 8 lines
      (Display duty = 1/32)

   2. Characters structure
      6 x 8 (Both alpha numeric characters and
             Graphic characters)

   3. LCD I/O address, I/O port
      Command write in to LCD
      Status read out from LCD

   Display's ON/OFF
      LCD can be separated by 10 blocks and can
      switch display (ON/OFF) in each IC block

The LCD is divided into the following IC blocks. Each block has its own Segment Driver with a 200 byte Display RAM. Each IC block can display 50 by 32 dots, however B5 and B10 display only 40 by 32 dots. One may write dots on the remaining area of the Display RAM of B5 and B10 without receiving an error, but the dots will not appear on the screen.



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

1.6.1 I/O PORT RELATED TO LCD

Block Select --- PPI 81C55 PORT A/B

```
msb 7   6   5   4   3   2   1   0 lsb
  ---------------------------------
  |PA7|PA6|PA5|PA4|PA3|PA2|PA1|PA0|     OUT B9H (185D)
  ---------------------------------
  | X | X | X | X | X | X |PB1|PB0|     OUT BAH (186D)
  ---------------------------------
```

PA0 to PA7 are associated to Block1 thru Block8 and PB0, PB1 to Blocks 9,10 respectively.

0 = Not Selected / 1 = Select

Description: Selecting an LCD Block (same as selecting a Segment Driver IC) which one wants to access. One cannot select two blocks at a time.

1.6.2 LCD COMMAND SET

There are 5 commands to control the Segment Driver IC. These commands are executed via PORT FEH (254D), the LCD Command/Status Port.

Display ON/OFF

```
msb 7   6   5   4   3   2   1     0  lsb
  +---+---+---+---+---+---+---+-------+
  | 0 | 0 | 1 | 1 | 1 | 0 | 0 | DISP |    OUT FEH (254D)
  +---+---+---+---+---+---+---+-------+

  DISP: Display ON/OFF
          0 = OFF
          1 = ON
```

Description: DISP decides whether the data in the Display RAM is displayed on the screen. This port does not effect the contents of the Display RAM.

Set Address Counter

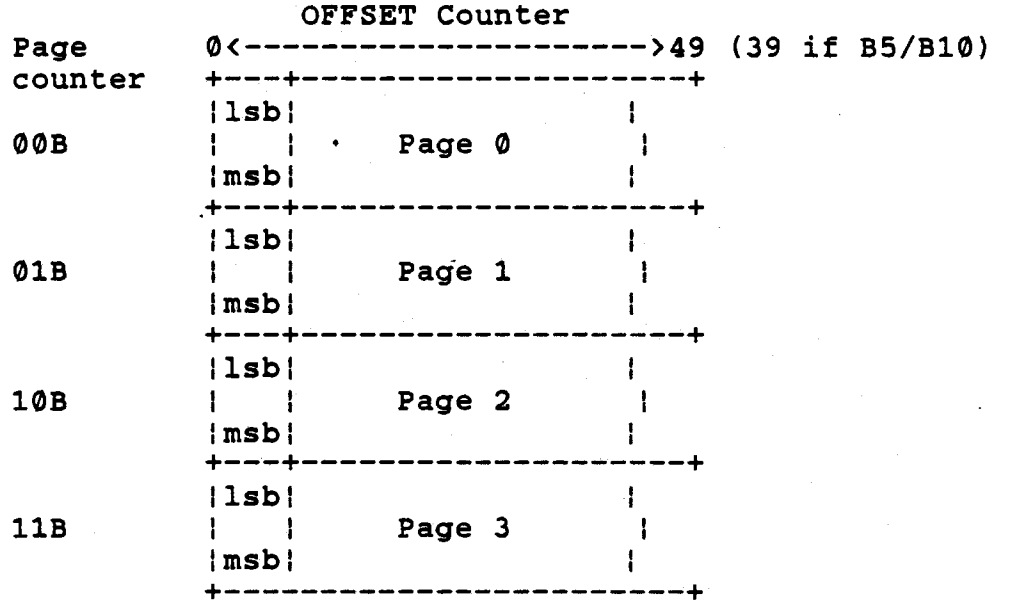
```
msb 7    6    5    4    3    2    1    0 lsb
 +----+----+----+----+----+----+----+----+
 |BIT7|BIT6|BIT5|BIT4|BIT3|BIT2|BIT1|BIT0|   OUT ~~F0H (240D)~~
 +----+----+----+----+----+----+----+----+        FEH (254D)
```

Select PAGE

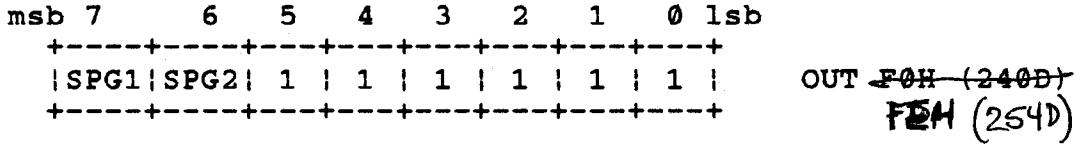
BIT7	BIT6	
0	0	Page0
0	1	Page1
1	0	Page2
1	1	Page3

OFn means "OFfset counter" in each Page, n 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 below. The Segment driver has a PAGE counter and an OFFSET counter. The counters are 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 below. The PAGE counter is not changed by the read/write operation.



Set Starting Page

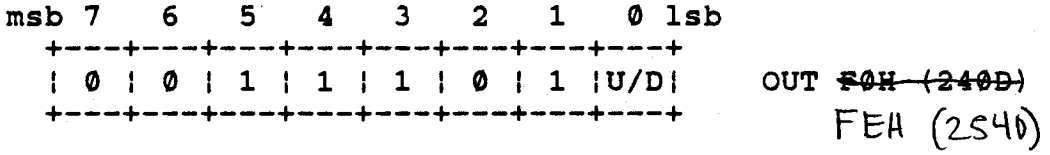


Specify the Starting Page to be displayed on the LCD.
(1/32 Duty)

SPG1	SPG2	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: The LCD block is divided into 4 pages corresponding with the Display RAM. The combination with the Page of the LCD Block and the Display RAM page can be changed. The "Set Starting Page" defines the mapping between the Page in the Display RAM and the Page of the LCD Block.

Select Address Counter Mode



Up/Down (Direction of counter)
0 = Down count
1 = Up count

Description: Set OFFSET Counter Mode.
This address counter loops back to initial by 50 counts and it is automatically incremented or decremented after accessing display data.

Read Status --- Read the status of the Segment Driver

msb 7 6 5 4 3 - 0 lsb

BUSY	UP/DOWN	ON/OFF	RESET	XXXXX

~~IN F0H (240D)~~ FEH (254D)

Bit 4 ----- Status of the RST pin
- 0 Normal
- 1 RST is low level
  Bit7 (BUSY) equals 1 at the same time

Bit 5 ----- Display
- 0 Display OFF
- 1 Display ON

Bit 6 ----- Mode of Address Counter
- 0 Down counter
- 1 Up counter

Bit7
- 0 Others
- 1 Executing in F1H, OUT F0H, or OUT F1H

Write/Read Display Data

D7	D6	D5	D4	D3	D2	D1	D0

IN/OUT ~~F1H (241D)~~ F0H (240D)

Description: Reads the data from the Display RAM that is pointed to by PAGE and OFFSET counter. If one wants to read some portion of the display RAM, use this command after Setting the PAGE counter and the OFFSET counter by the "Set Address Counter" command and the "Set Page Counter" command. Note that one dummy read must be done before using this command in order to get correct data.

1.6.3 SOFTWARE FOR THE LCD

This section describes how to handle the LCD without reading the routines stored in ROM#Ø and how to maintain the bookkeeping area for the LCD in the RAM.

1.6.3.1 How To Initialize the LCD

Initialization Process

1. Set up Address Counter, usually page Ø, offset Ø
2. Set up Offset Counter Mode
3. Set up Starting Page
4. Select Display On/OFF

The following program initializes the LCD's Segment drivers as shown.

```
PAGE COUNTER = Ø
OFFSET COUNTER = Ø
UP COUNTER MODE
START ON PAGE = Ø
DISPLAY ON
```

NOTE:
Whenever the power is turned on, the LCD is initialized by the reset pulse of the hardware. At that time, the Display is turned OFF, the Offset counter is set to count up mode. Other status is not determined.

ROM #Ø always re-initializes the LCD as Display ON, Starting Page = Ø and Offset counter count up mode when a character is displayed.

SAMPLE PROGRAM FOR LCD INITIALIZATION

```
; Initialize Segment Driver
;
; --- Equates ---

PORTA     EQU  B9H
PORTB     EQU  BAH
LCDCOM    EQU  FEH
LCDSTAT   EQU  FEH


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

          CALL LCDBUSY   ; Wait until LCD becomes ready
          XRA  A      ;
          OUT  LCDCOM    ; Reset address counter

          CALL LCDBUSY   ;
          MVI  A,3BH     ; Offset counter UP mode
          OUT  LCDCOM    ;

          CALL LCDBUSY   ;
          MVI  A,3EH     ; Set starting PAGE = 0
          OUT  LCDCOM    ;

          CALL LCDBUSY   ;
          MVI  A,39H     ; Display ON
          OUT  LCDCOM    ;

LCDBUSY:
; Wait until LCD becomes 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,FFH     ;
          OUT  PORTA     ; B9H
          IN   PORTB     ; Get current status
*       ORI  03      ; Select block 9,10
          OUT  PORTB     ;
          RET

          END
```

1.6.3.2 How To Write A Character

Writing a character on the LCD is performed by writing some

Bit patterns into the Display RAM of the Segment Driver. The basic sequence of writing a character on the LCD is as follows:

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

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

SAMPLE PROGRAM TO WRITE A CHARACTER ON THE LCD

```
; This program performs the same function as the following
; BASIC program.
;
; 10 LOCATE 0,0
; 20 PRINT "A"
; 30 END

CSRY      EQU  F3E6H      ; Cursor Y position (1-8)
CSRX      EQU  F3E6H      ; Cursor X position (1-40)
LCTEY     EQU  FEB9       ; Character Y position (0-7)
LCTEX     EQU  FEBA       ; Character X position (0-39)
PORTA     EQU  B9H        ; Segment Driver Select Port   Dec: 185
PORTB     EQU  BAH        ;    Dec: 186
LCDCOM    EQU  FEH        ; LCD command Port  Dec: 254
LCDSTAT   EQU  FEH        ; LCD Status Port   Dec: 254
LCDIO     EQU  ~~F000~~ FFH  ; LCD data I/O Port  Dec: 255

          ORG  F000       ; 61440D

LOCATE:
; Locate 0,0
          LXI  H,0101H    ; To set cursor position
          SHLD CSRY       ;
          LXI  H,0000H
          SHLD LCTEY



PREP:
; -- Select Block 1 to write (1,1)
          DI              ; Inhibit disturbance for Port A/B
                          ; of the 81C55. DI is not necessary
                          ; if the data port of the 81C55 is
                          ; not changed, but you must consider
```

```
                        ; other INT routines
          MVI   A,01H     ; Select Block 1
          OUT   PORTA     ; (185)
          IN    PORTB     ; Get current status (186)
          ANI   11111100B ; Deselect Block 9/10
          OUT   PORTB     ; (186)

          CALL  LCDBUSY   ; Wait until LCD is ready
          MVI   A,0       ; Page 0, Offset 0
          OUT   LCDCOM    ; (254)

          CALL  LCDBUSY   ;
          MVI   A,00110010B    ; Offset counter UP mode
          OUT   LCDCOM    ; (254)

CHROUT:
          LXI   H,FONTA   ; Get start address of Font A
          MVI   C,06H     ; 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 is ready
          MOV   A,M       ; Get font pattern to send
          OUT   LCDIO     ; Write to display RAM of LCD (255)
          INX   H       ; Update Pointer
          DCR   C       ; Update counter
          JNZ   WRITE     ; To send next pattern. Offset counter
                          ; is Auto increment Mode, so we don't
                          ; care about the Offset counter.
          LXI   H,CSRX    ; Update Cursor Pointer
          INR   M       ; No check for the end of line
                          ; in this program
          LXI   H,LCTEY   ;
          INR   M       ;




;---- Set starting Page ----
          MVI   A,FFH     ; Select all block
          OUT   PORTA     ;
          IN    PORTB     ;
          ORI   00000011B ;
```

```
        OUT   PORTB      ; (186)

        CALL LCDBUSY     ; Wait until LCD become Ready
        MVI   A,3FH      ; Starting page 0
        OUT   LCDCOM     ; (254)

        MVI   A,00111001B     ; Insure display ON
        OUT   LCDCOM     ; (254)
        EI
        RET

LCDBUSY:
        IN    LCDSTAT    ; Get LCD status (254)
        RLC              ; Move MSB to CF
        JC    LCDBUSY    ;
        RET

FONTA:   DB    3CH,12H,11H     ; Font data for "A"
        DB    12H,3CH,00H     ;

        END
```

1.6.3.3 How To Set/Reset A Dot On The LCD

The sample program shown below explains how to set/reset a dot on the LCD. it does the 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
```

SAMPLE PROGRAM FOR SET/RESET DOT

```
PORTA     EQU  B9H          ; LCD block select
PORTB     EQU  BAH          ;
LCDCOM    EQU  FEH          ; LCD command
LCDSTAT   EQU  LCDCOM        ; LCD status
LCDIO     EQU  FFH          ; LCD data I/O


PSET:
          DI                ; Disable all interrupts to keep
                              ; current block select
          XRA  A            ; To set SET flag
          STA  SR           ; Set/Reset Flag
          LXI  B,140EH       ; [B]=20 X Count, [C]=14 Y Count
          LXI  H,0A09H       ; [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,FFH          ; To set SR flag
          STA   SR             ; Set Unplot Flag
          LXI   B,ØC06H         ; [B]=12,[C]=Ø6
          LXI   H,ØEØDH         ; ([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  FFH             ; Reverse MASK pattern
          ANA  D               ; [A]= data to write
```

```
DISP:
          MOV  D,A          ;
          CALL WRITE        ;
          DI                ;
          MVI  A,FFH        ; Select all blocks
          OUT  PORTA        ;
          IN   PORTB        ;
          ORI  00000011B      ;
          OUT  PORTB        ;
          CALL LCDBUSY      ; See if LCD is busy
          MVI  A,00111111B    ; Starting Page 0
          OUT  LCDCOM       ;
          CALL LCDBUSY      ;
          MVI  A,00111001     ; 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: OUTOUT [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: NONE
;
; EXIT: [D]=CURRENT DATA IN DISPLAY RAM
;
; REG: A,D AND FLAGS

          CALL LCDBUSY          ; Wait until LCD becomes ready
```

```
          IN   LCDIO              ; Dummy read-must do 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  00000111B          ;
          MOV  L,A                ; Set counter
          MVI  A,~~00B~~
                  80H             ;
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
;
; REG: A,H,L AND FLAGS

          MOV  A,L                ; Get Y position
          RAL                     ; Move Bit4/3 to Bit7/6
          RAL                     ;
          RAL                     ;
          ANI  11000000B          ; Get page
          ORA  H                  ; [A]=Page and Offset
          MOV  L,A                ; Save it.
          CALL LCDBUSY            ; Wait until LCD is ready
          MOV  A,L                ; Retrieve address
          OUT  LCDCOM             ;
          RET
LCDBUSY:
; ENTRY: NONE
;
; FUNC: WAIT UNTIL LCD IS READY
;
; EXIT: NONE
;
; 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,00000010B       ; Select Block-2
          OUT   PORTA            ;
          IN    PORTB            ;
          ANI   11111100B        ;
          OUT   PORTB            ;
          RET                    ;

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

          END
```

HOW TO DEFINE A CHARACTER

This section describes how to define the User Definable characters in the PC-8300A and how to store them in a portion of RAM where ROM #0 can use this new font. BASIC commands will be used to produce some of the operations.

STRUCTURE OF A CHARACTER AND HOW TO DEFINE IT

One character consists of 6 * 8 dots. The vertical 8 dots are handled by a byte. In order to define a character, one must define Sequentially the 6 bytes of data. The data 7CH (124D), 12H (18D), 11H (17D), 12H (18D), 7CH (124D), and 00H (0D) define "A" as follows.

```
; CG pattern for "A"
        DB   7CH,12H,11H,12H,7CH,3CH,00H
```

DATA PATTERN

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

(lsb = 0, msb = 7)

FONT PATTERN

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

HOW TO STORE ONES OWN CG

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

Assume that one has 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 YYZZH.

One can make "FONT.CO" in the following sequence.

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

   CLEAR <length>,<startaddress>

2. Load "FONT.CO" into RAM

   BLOAD "FONT"

3. Register the top address of the CG

   POKE 65216D,<Start Address (High Byte)>
   POKE 65215D,<Start Address (Low Byte)>

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

defined CG.

AVAILABLE SYSTEM WORK AREA

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

HOW TO USE THE CG IN THE SYSTEM ROM

One might want to use the CG of ROM #0 instead of making a new CG by yourself. The Character Generator characters whose code is from 20H to 7EH, are stored in the highest portion of ROM #0, from 78B7H to 7B37H. Each character consists of5 bytes. The sample below 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 a CO file in the RAM files and will be executed with ROM #0.

```
;ENTRY [A]= character code (20H TO 7EH)

EXPAND:
          SUI   A,20H             ;
          MOV   C,A               ;
          ADD   C                 ; *2
          ADD   A                 ; *4
          ADD   C                 ;
          MOV   C,A               ; [C] offset from base of CG
          MVI   B,00H             ;
          LXI   H,CGADR           ;
          DAD   B                 ;
          LXI   B,TEMP            ;
          MVI   D,5H              ; 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
```

VRAM AREA IN THE SYSTEM WORK AREA

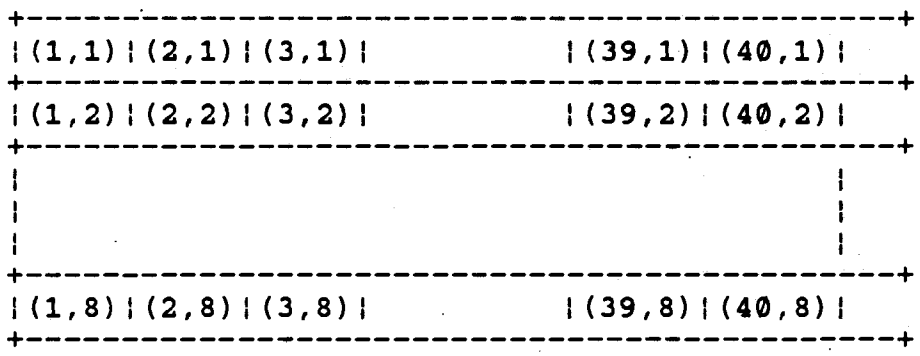
The area from FBC0H to FE3FH in the RAM is reserved for VRAM area of the LCD. It is divided into two portions. Each portion can hold the character codes displayed on the lcd at a time. Each portion has 320 bytes. The attribute data is not saved in this area, only the character code is stored.

```
1. FBC0H - FCFFH        ; Keep previous Page
                          ; in TELCOM
2. FD00H - FE3FH        ; Current displayed
                          ; character is saved
```

The character code of the character displayed at the location (1,1) on the LCD display is stored at FD00H.Therefore the code of the left-lowest character (40,8) is stored at FC3FH. This rule is used in the standard program in ROM #0. For instance, BASIC, TEXT, TEXT, TELCOM use that area like a VRAM in the traditional desk top personal computer. The menu screen also utilizes that area, but you can use the area too. The data in this area does not effect the information on the LCD display, as far as you use your own display routine.

REVERSE THE ATTRIBUTE OF THE SPECIFIED AREA

ROM #0 has the reverse attribute table in the Work area. The attribute data is kept in the area from FA60H (64096D) to FA87H (64135D). Each bit represents each character Box on the LCD. Therefore only 40 bytes can handle the attribute of the whole LCD screen. When the bit is OFF (0), it shows that the character Box is displayed in normal mode. Likewise when the bit is ON (1), the character box is displayed in reverse mode. The relation between the Attribute bit and the Character Box is shown below. The relation of the reverse attribute bit and each character box follows.



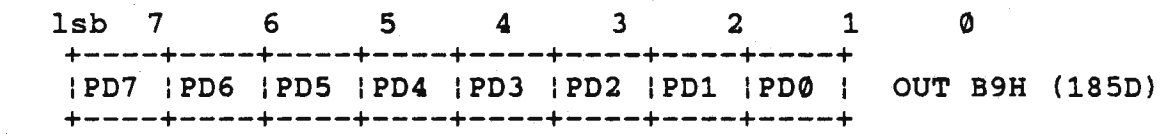
FA60H	Bit0	(01,1)
	Bit1	(02,1)
	Bit2	(03,1)
	Bit3	(04,1)
	Bit4	(05,1)
	Bit5	(06,1)
	Bit6	(07,1)
	Bit7	(08,1)
FA61H	Bit0	(09,1)
	Bit1	(10,1)
	¦	¦
FA87H	Bit0	(33,8)
	Bit1	(34,8)
	Bit2	(35,8)
	Bit3	(36,8)
	Bit4	(37,8)
	Bit5	(38,8)
	Bit6	(39,8)
	Bit7	(40,8)

1.5 Printer interface

Printer interface is the 8 bit parallel interface The Centronics compatible interface on the PC-8300A uses a 26-pin. The following is the I/O port of the printer interface. It sets the data on the port A of 81C55 and inputs the control signal from the printer to port C. Port A takes output. Port C takes input.

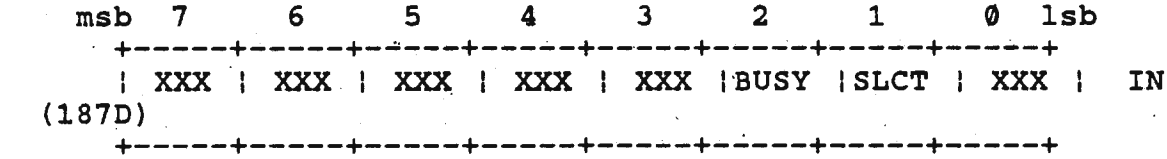
I/O Port For Printer Interface

Port A--- Data transmission (OUTPUT) to printer



Bit 7 -- Bit 0 Printer data port

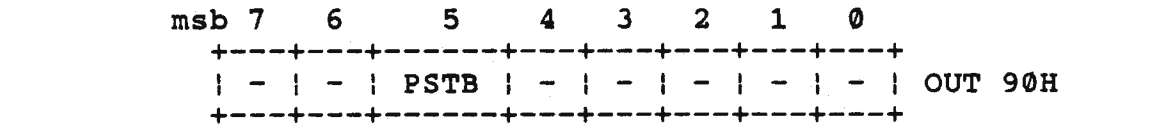
Port C--- BUSY, SLCT Signal Read



BUSY --- 0 Printer READY
1 Printer BUSY

SLCT --- 0 Deselect
1 Select

SCP (System Control Port) --- STROBE Output Port



PSTB ---- 0 Strobe OFF
1 Strobe ON

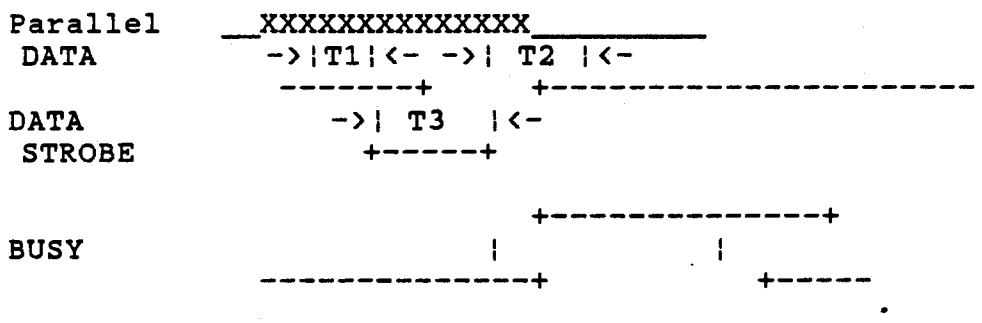
BASIC THEORY OF WRITING DATA TO CENTRONICS

The basic sequence to write data to the Centronics printer is

as follows.

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

TIMING CHART



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

Refer to the printer manual for the actual duration of T1 to T3.

SOFTWARE SPECIFICATION

HOW TO WRITE A BYTE TO THE PRINTER

The program below explains how to send a character to the Parallel port(the same as the BASIC command LPRINT "ABCDEFGHIJ").

```
;
 60000D
;--EQUATES--
SCP         EQU     90H             ; System Control Port
PORTA       EQU     B9H             ; Printer Data Port
PORTC       EQU     BBH             ; Printer Status Port
SYSSTAT     EQU     FE44H           ; SCP 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 PortA
        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     00100000B       ; Set STROBE
        OUT     SCP             ;
        MOV     A,B             ;
        OUT     SCP             ;
        MOV     B,03H           ; Set appropriate value for
                                ;  specific printer
WAIT:
        DCR     B               ;
        JNZ     WAIT            ;
        EI                      ;
        INX     H               ; Point to next
        DCR     C               ;
        JNZ     PRINT           ;
        RET                     ;
BUF:     DB      'ABCDEFGHIJ'
        DB      13,10


        END
```

1.6 Calendar clock interface

It contains LSI (uPD1990AC) for clock. The following I/O ports are assigned.

Command data out port

```
+----+----+
I1011I1001I       OUT  B9H (185D)
+----+----+
```

7	6	5	4	3	2	1	0
I	-	I	-	I	CD0	I CCK	I C2 I C1 I C0 I

```
+---+---+---+-----+-----+----+----+----+
```

C2,C1,C0 Command out port

CCK Shift clock
0 OFF
1 ON

CD0 Data out port
Starting value is 05H

Command strobe to clock

```
+----+----+
I1001I0000I       OUT  90H (144D)
+----+----+
```

```
  7   6   5    4     3   2   1   0
+---+---+---+------+---+---+---+---+
I - I - I - I TSTB I - I - I - I - I
+---+---+---+------+---+---+---+---+
```

TSTB Command strobe to clock
0 Strobe OFF
1 Strobe ON

Data from clock

```
+----+----+
I1011I1011I       IN   BBH (187D)
+----+----+
```

```
  7   6   5   4   3   2   1    0
+---+---+---+---+---+---+---+-----+
I - I - I - I - I - I - I - I CDI I
+---+---+---+---+---+---+---+-----+
```

CDI Clock data input data

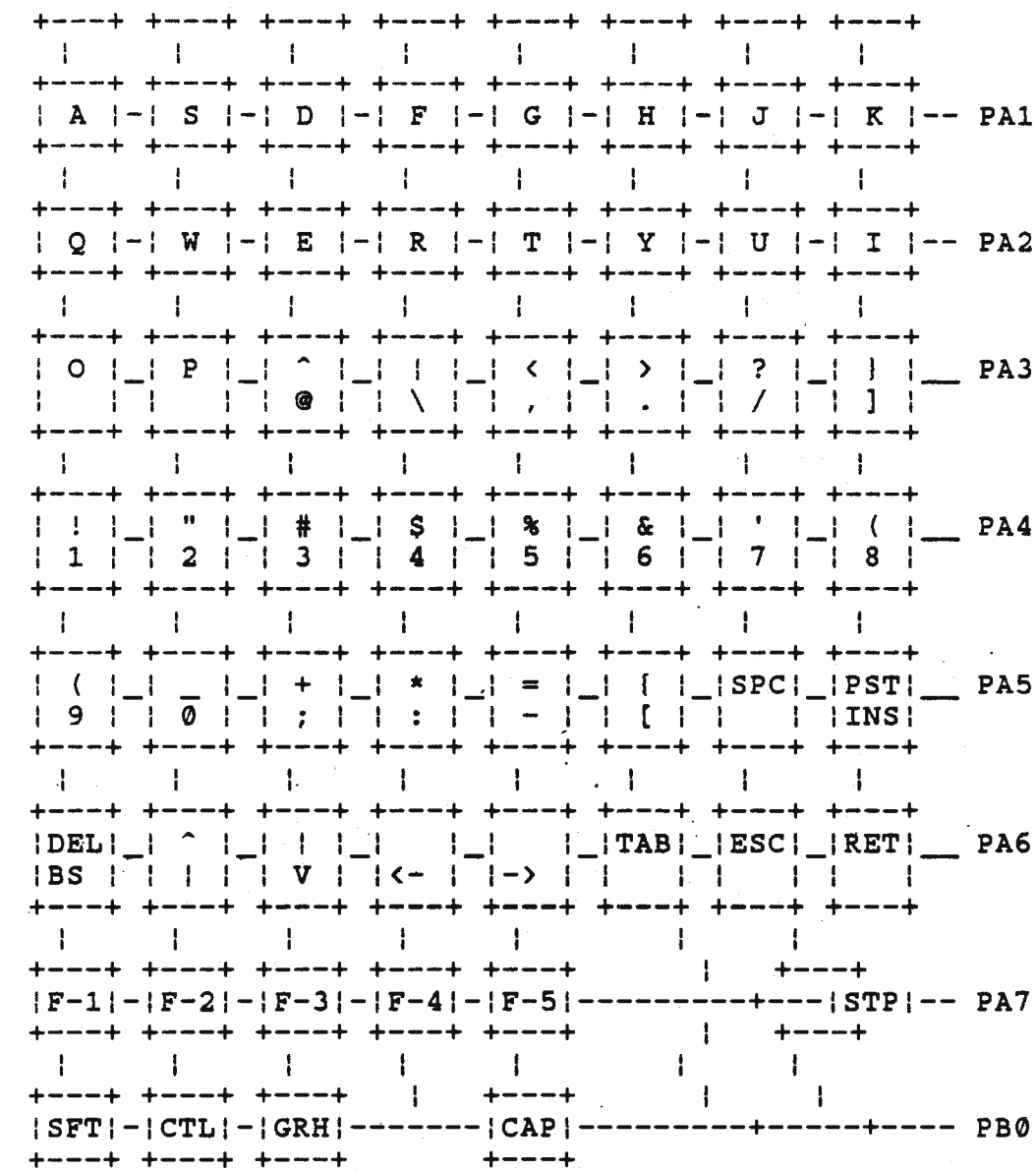
1.7 Keyboard interface

Keyboard is sensed by the soft scan method. It outputs Low signal to port A (of 81C55) and a part of port B (PB0) in order (OUT B9,OUT BA). It sense the pressed key in IN E8H.

PC-8300A KEYBOARD MATRIX

Keyboard Data Port

```
 KD0    KD1    KD2    KD3    KD4    KD5    KD6    KD7
  |      |      |      |      |      |      |      |      Key Strobe Port
+---+  +---+  +---+  +---+  +---+  +---+  +---+  +---+
| Z |-| X |-| C |-| V |-| B |-| N |-| M |-| L |-- PA0
```



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

I/O PORT FOR THE KEYBOARD

KEYBOARD STROBE ----Port A/B of the 81C55

msb 7	6	5	4	3	2	1	0 lsb	
KS7	KS6	KS5	KS4	KS3	KS2	KS1	KS0	OUT B9H (185D)
X	X	X	X	X	X	X	KS8	OUT BAH (186D)

KS8 ... KS0 KEYBOARD Strobe

```
              Ø = Strobe OFF
              1 = Strobe ON
```

KEYIN ---- Read Keyboard Data

```
    msb 7   6   5   4   3   2   1   Ø lsb
      +---+---+---+---+---+---+---+---+
      |KD7|KD6|KD5|KD4|KD3|KD2|KD1|KD0|  OUT EBH (235D)
      +---+---+---+---+---+---+---+---+

      KD7 ... KDØ KEYBOARD Data
              Ø = Depressed
              1 = Not Depressed
```

Read the strobed column of the keyboard. Please refer to the Key Matrix shown earlier to understand the relationship between KDn and the Key on the Keyboard.

KEYBOARD SCANNING

Key scanning must be performed by software. It can be done by the interrupt, RST 7.5, the RST 7.5 Pin of the 8ØC55 is connected to the TP Pin (No. 1Ø) of the calendar clock (uPD199Ø). This interrupt occurs every 4 msec in the standard system.

SOFTWARE FOR KEYBOARD OPERATION

HOW TO READ THE KEYBOARD

Basic keyboard sequence:

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

The following sample program shows how to read the keyboard in detail. The program reads every column and saves the data into the KYBUF (Keyboard Buffer).

```
; Read Current Keyboard Status
;
; Note: Make sure Keyboard strobe is
;       not disturbed while reading the keyboard.
;       Also take care of the other interrupts.
```

```
;
; EQUATES

PORTA     EQU       B9          ; Keyboard strobe Port
PORTB     EQU       BA          ;
KEYIN     EQU       E8          ; Keyboard data port

          ORG       F000H       ;
READKEY:
          LXI       B,KYDATA       ; Get PTR for buffer
          MVI       A,FFH       ; Disable normal key strobe
          OUT       PORTA       ;
          IN      PORTB       ; Get PortB Status
          ANI       FEH         ; SET B0 = OFF
          OUT       PORTB        ; Activate Strobe for Special
key
          IN      KEYIN       ; Read keyboard
          STAX      B           ; Save data
          IN      PORTB       ; Get status of Port B
          ORI       01H         ; SET B0 = ON
          OUT       PORTB       ;Strobe off
          MVI       A,11111110B    ;
```

```
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,FFH           ;
        OUT     PORTA           ; Strobe off
        MOV     A,D             ; Retrieve strobe data
        RLC                     ; Strobe for next column
        JC      NOMAL           ;
        RET                     ; All done return to caller

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


        END
```

1.8 Serial interface

The PC-8300A has three channels of Serial Interface, RS-232C, SIO1 and SIO2. UART (6402) and PPI (81C55) control the Serial Interface. Since they are shared by 3 channels, only one channel is available at one time.

I/O PORT

CHANNEL SELECT -- (System Control Port)

I/O Address and Data Pattern

```
msb  7    6   5 -- 0          lsb
  +----+----+--------------------+
  |SRI2|SRI1| XXXXXXXXXXXXXXXXX  |    OUT 90H (144D)
  +----+----+--------------------+
```

SRI 2/1 Serial Interface Select

SRI2	SRI1	USER
0	0	Not used
0	1	SIO 1
1	0	Floppy disk (SIO 2)
1	1	RS-232C

NOTE: The current status of this port is saved in SYSSTAT (FE44H) by the System ROM.

UART MODE CONTROL

msb 7 - 5 4 3 2 1 0 lsb

XXXXX	CLS2	CLS1	PI	EPE	SBS	OUT D8H (216D)

SBS Stop bit select

0	1 bit
1	2 bits (*)

(*) When data length is 5 bits, stop bits is 1.5 bits.

EPE Even Parity Enable
(meaningless if PI = 1)

0	Odd parity
1	Even parity

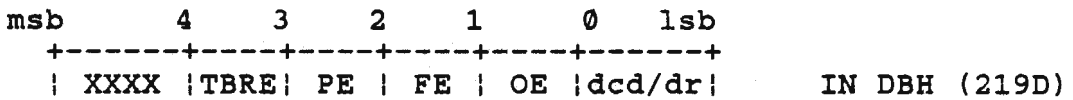
PI Parity inhibit

0	Parity enable
1	Parity disable

CLS2	CLS1	Character Data Length
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

UART STATUS READ

I/O ADDRESS AND DATA PATTERN



dcd/dr DCD/DR (0=on/1=off)

OE Over-run Error (1=Detected)

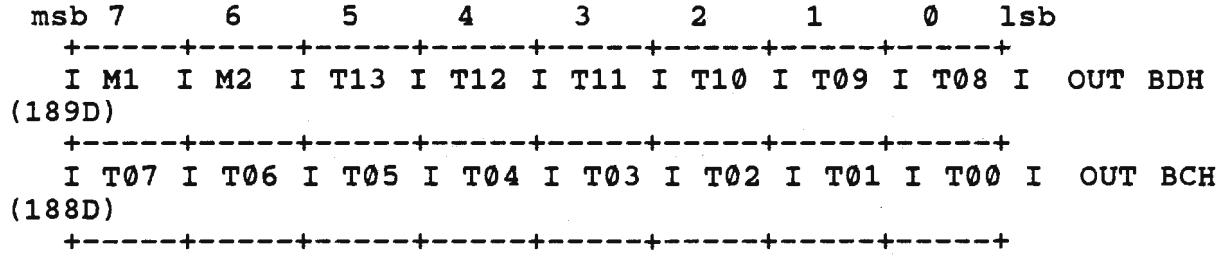
FE Framing Error (1=Detected)

PE Parity Error (1=Detected)

TBRE Transmit Buffer Register Empty
1=ready to receive data to transmit

Set UART Baud Rate (PPI 81C55 Timer Section)

I/O Address and Data Definition



M1/M2 Specify timer output mode

00B = Single Square Wave
01B = Continuous Square Wave
10B = Single Pulse On
11B = Continuous Pulse

Note:
To set a Baud Rate use the values below.

Baud Rate	BCH	BDH
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

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

UART DATA I/O PORT

I/O Port and Data Pattern

msb							lsb	
D7	D6	D5	D4	D3	D2	D1	D0	IN/OUT CBH (203D)

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

SOFTWARE DESCRIPTION

How to Initialize the Serial Port

The basic sequence to initialize the Serial Port is as follows.

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

The following sample program shows the initialization sequence more detailed.

the program explains how to initialize the serial port. This sample program initializes the RS-232C Channel to: 9600 bps, even parity, 7 bit data length, 1 stop bit, and no control for Xon/Xoff or SI/SO. The program updates the work area for ROM #0. One may skip that portion because there will be no problem, even if updating the data is skipped. ROM #0 always initializes the RS-232C Port when entering Terminal mode or "OPEN COM:" of Basic command issued by the Mode string.

```
; Sample Program to Initialize the Serial Port
;
; Data in the system work area must be updated

SERMOD    EQU       F406H       ; 6 bytes for MODE string
;                   F406H       ; Baud rate specifier
;                   F407H       ; Parity mode
;                   F408H       ; Word length
;                   F409H       ; Stop bits
;                   F40AH       ; XON/XOFF control
;                   F40BH       ; SI/SO control
INHDSP
INHIBIT
COMACT    EQU       FE43H       ; current user ID for
                                  ; serial port.
                                  ;    00B = Not used
                                  ;    01B = SIO2
                                  ;    10B = SIO1
                                  ;    11B = RS-232C
SYSSTAT   EQU       FE44H       ; SCP port status
BAUDRT     EQU      FE4AH        ; Baud rate Table entry
address
INHBIT    EQU       FE41H       ; 0 inhibits XON/XOFF control
;
; I/O PORT ADDRESS
;
SCP       EQU       90H         ; System Control Port
PORTB     EQU       BAH         ; RTS/DTR set port
TIMEL     EQU       BCH         ; Timer set Low
TIMEH     EQU       BDH         ; Timer set High
RTSDTR    EQU       3FH         ; RTS/DTR data for RS-232C
                                  ; Use FFH for SIO1/2
INITSERI:
; ENTRY: [C] = USER ID
;      [B] = BAUD RATE SPECIFIER. ASCII NUMBER (1 TO 9)
;            SAME NUMBER "STAT" AS 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 Flag
         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 to
reset
                                      ; the User ID to 00 after all
of
                                      ; tasks are finished,else the
; serial port can not be shared
                                      ; with another user.
          RRC                       ; Move Bit0-1 to Bit 6-7
          RRC                       ;
          MOV       C,A             ; Save it
          LDA       SYSSTAT          ; Get current SCP status
          ANI       00111111B         ; 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 reini-
tializes
                                      ; it with this entry data
                                      ; after finished.
          MOV       A,M             ; Get lower value
          OUT       TIMEL           ;
          INX       H               ;
          MOV       A,M             ; Get higher Value
          OUT       TIMEH           ;
          MVI       A,C3H           ; To start timer
          OUT      B8H               ; Use this value to start
timer
; SET TRANSFER MODE
MODE:
          IN      PORTB         ;
          ANI       RTSDTR          ; If RS232C RTSDTR=3FH to
                                      ; activate RTS/DTR
                                      ; else FFH to unactivate
          OUT       PORTB           ;
```

```
        IN      C8H             ; Dummy read to clear
                                ; receive buffer register
        MVI       A,00001110B   ; 7bit, even parity,1 stopbit
        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/XOFF 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      00H,48H         ;    75 bps
        DB      6BH,45H         ;   150 bps
        DB      00H,42H         ;   300 bps
        DB      00H,41H         ;   600 bps
        DB      80H,40H         ;  1200 bps
        DB      40H,40H         ;  2400 bps
        DB      20H,40H         ;  4800 bps
        DB      10H,40H         ;  9600 bps
        DB      08H,40H         ; 19200 bps
```

SEND DATA TO THE SERIAL PORT

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

```
; SEND DATA TO THE SERIAL PORT
;
; ENTRY: [C] = DATA TO BE SENT
;
WRITE:
        IN      D8H             ; Get UART status
        CPI       00010000B     ; See if transmitter buffer
                                ; register is empty
        JZ      WRITE           ; Wait until TBR becomes empty
        MOV       A,C           ; Get a character to send
        OUT       C8H           ; Send it out the serial port
        RET                     ;
```

28160

READ DATA FROM THE SERIAL PORT

The sample program below explains how to read data from the serial port by using RST 6.5. This program only reads data from the serial port with RST 6.5, no XON/XOFF and no SI/SO control is performed.

```
; ** Read data from the Serial Port using RST 6.5

        ORG     3CH         ; Entry point of RST 6.5
RST65:   DI                  ;
        JMP     READ        ;
        ORG     ????        ;
READ:
        PUSH    H           ; Save all registers
        PUSH    D           ;
        PUSH    B           ;
        PUSH    PSW         ;
        IN      C8H         ; Read the data
        MOV     L,A         ; Save it
        IN      D8H         ; Get error status
        ANI     00001110B       ; 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
```

AVAILABLE SYSTEM AREA

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

The Serial Input Buffer from FE4CH (65100D) to FFC3H (65475D), is reserved by the system ROM, but you can use it for your own routines.

SERMOD saves the RS-232C mode string.

This area has six bytes which indicate the RS-232C string mode, specified by the "STAT" command in TELCOM or OPEN "COM:" command in Basic. The contents are as follows:

SERMOD at F406H (62470D): RS232C String mode buffer

```
F406H     ; Baud rate specifier (1 to 9)
F407H     ; Parity Mode (N/E/O/I)
F408H     ; Word length specifier (5 to 8)
F409H     ; Stop bit (1/2)
F40AH     ; Xon/Xoff control (X/N)
F40BH     ; SI/SO control (S/N)
```

INHIBIT at FE42H (65090D)
This byte is the XON/XOFF Inhibit Flag. 0 inhibit XON/XOFF control, else enabled.

COMMACT at FE43H (65091D)
This byte indicates who is using the serial port. Make sure to reset to 0 after using the serial port, otherwise the serial port is not available for another user.

```
00H = No user
01H = SIO2
10H = SIO1
03H = RS-232C
```

CMPNT at FE46H (65094D): Character count in Buffer
This byte has the character count in the serial buffer.

REDADDR at FE46H (65094D)
This byte indicates the last read character displacement.

WTADR at FE47H (65095D)
This byte indicates the last written character displacement.

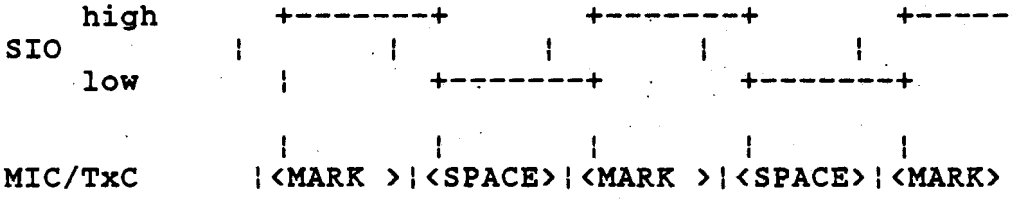
BAUDRT at FE4AH (65098D)
this points to the table of the Baud rate.

1.9 CASSETTE (CMT) INTERFACE

Cassette interface uses the SID (Serial Input Data) pin of 80C85 and the SOD (Serial Output Data) pin. The Motor is controlled by the SCP (System Control Port, 90H). 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, bits per second (the system ROM #0, uses 600 bps to maintain the compatibility with the PC-8001A). The physical interface of the CMT is described in this chapter. Information on how to control the Motor of the CMT, how to write data to the CMT and how to read data from the CMT. For information on File Format please refer to Chapter XX.

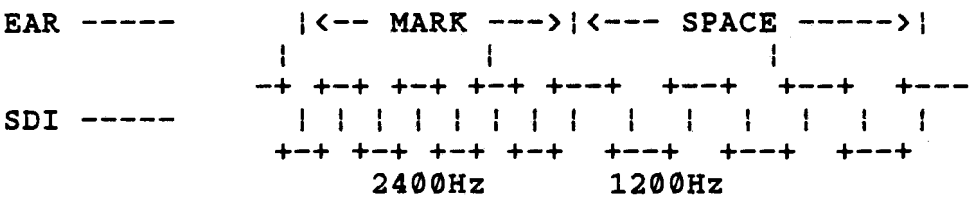
WRITING OPERATION

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



READING OPERATION

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



I/O PORT FOR CMT

SCP ---- System Control Port

```
msb 7   6   5   4     3       2   1   0 lsb
  +---+---+---+---+---------+---+---+---+
  I - I - I - I - I REMOTE I - I - I - I   OUT90H (144D)
  +---+---+---+---+---------+---+---+---+
```

REMOTE CMT Motor Control
- 0 = CMT Motor OFF
- 1 = CMT Motor ON

Description: The current status of this port is saved at SYSSTAT (FE44H), so you have to update this area when you want to change the status of this port.

PPI 81C55 Command Set

```
msb 7   6   5   4   3   2   1   0 lsb
  +---+---+---+---+---+---+---+---+
  ITM2ITM1I 0 I 0 I ? I ? I 1 I 1 I   OUTB8H (184D)
  +---+---+---+---+---+---+---+---+
```

TM2/1 Timer Command for PPI

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

BAUD RATE GENERATION

The Baud Rate is generated by the software timing routine. In the writing operation, the bit data for the SOD Pin is set and it is held during the proper duration by the software wait-routine. On reading, a data bit is read in the proper interval which is controlled by software. The CPU uses a 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 below.

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

WRITING DATA TO THE CMT

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

```
; Write a byte to the CMT, the lowest routine
;
; Assumption:
;    CMT Motor is rotating regularly and CALLED
;    by the Interrupt Disable
;
; INPUT:  [A] = Data to be sent
;
; OUTPUT: None
;
; BAUD Rate = 600 bps
;
WRITE:
        MOV       B,A          ;  4: Save Data
        MIV       A,50H        ;  7: Write start bit
        SIM                    ;  4:
        CALL      HOLD         ; 18: Wait 4043 State
        IN      PORTC        ; 10: Dummy to adjust timing
        MOV       C,08H        ;  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,D0H        ;  7: To send MARK
          JC       BITO          ;10/7: Branch if HIGH
          MVI       A,50H        ;  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,D0H        ;  4: To send stop bit
          RET                    ; 10: It is the responsibility
                                 ;   : of the CALLER routine
                                 ;   : to make 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 the Caller
;
HOLD:
          LXI       H,167        ; 10: For 1 Bit (600 Baud)
HOLD1:
          DCX       H            ;  6:
          MOV       A,L          ;  4:
          ORA       H            ;  4:
          JNZ       HOLD1        ;10/7:
          RET                    ; 10:
```

READING DATA FROM THE CMT

```
; Sample program for reading a Byte from the CMT
;       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 bits
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     80H         ;  7: Isolate SID bit
        MOV     E,A         ;  4: Save it
SYNC1:
        RIM                 ;  4: Get Current status
        ANI     80H         ;  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     80H         ;  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:
```

1.10 BARCODE READER

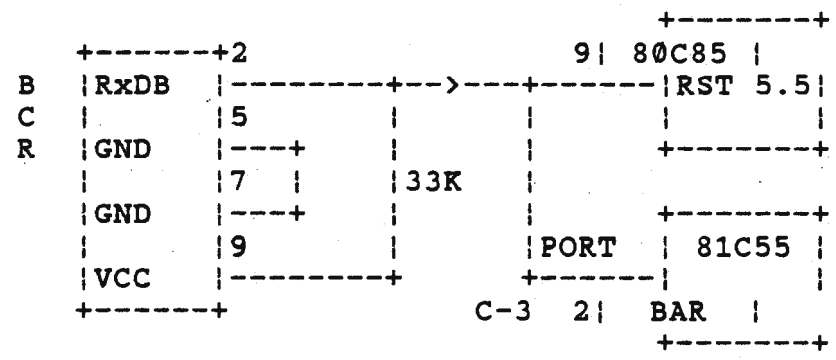
This chapter explains the Electrical specifications and the basic theory of operation of the Barcode Reader. The Barcode Reader programs on the PC-8300A Personal Application Kit Tape, assume that operation is done with the HEDS-3071 (produced by HP Corporation).

ELECTRICAL SPECIFICATION

Refer to the PC-8300A Users Guide for information about the shape and Pin connection of the Barcode interface.

One may connect any Barcode Pen to this interface. But NEC recommends the products of YHP (Yokokawa HP) or Mecano Kogyo. It is better to use a pen that has a Power switch, for saving the electrical power of the PC-8300A.

The data line of the Barcode Reader is connected to Pin-2 of the BCR. This pin is connected to RST 5.5 of the CUP (80C85) nd Port C-3 of the 81C55 as shown below.



While the Barcode Reader is powered on, PIN-2 is kept at a low level, and RST 5.5 is high. The Black Bar is represented by logical Low, SPACE BAR is High respectively.

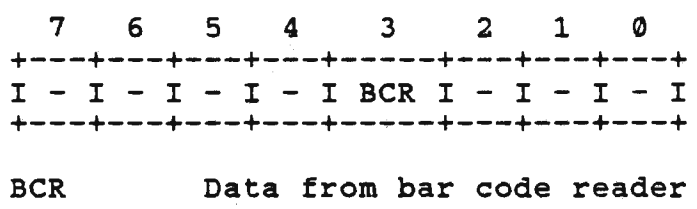
THEORY OF OPERATION

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

1. If power on, RST 5.5 is activated. At the first point of the RST 5.5 routine which is interrupted by RST 5.5 disables all interrupts.
2. Pole the Bar Code DATA port. And calculate the duration of the 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.

Data from bar code reader

```
+----+----+
I1011I1011I       IN   BBH (187D)
+----+----+
```



1.11 Interrupt Function

This machine have a 3 level input with priority and works with input RST 7.5,6.5, and 5.5 of 80C85. Interrupt level and its priority order is the following:

Priority Order	Interrupt channel	Function
High	RST 7.5	Key int
¦	RST 6.5	UART
Low	RST 5.5	BCR

Key int It checks the key input by the 256Hz clock from uPD1990AC
UART Receipt interrupt of UART
BCR Interrupt for bar code reader

These interrupt have their own mask flags and each can be masked independently. Mask flag can be reset by SIM (Set Interrupt Mask) command.

7	6	5	4	3	2	1	0
-	-	-	R7.5	MSE	M7.5	M6.5	M5.5

M5.5 RST 5.5 Mask flag
0 Reset
1 Set

M6.5 RST 6.5 Mask flag
0 Reset
1 Set

M7.5 RST 7.5 Mask flag
0 Reset
1 Set

MSE Mask set enable
0 Mask set disable
1 mask set enable

R7.5 Reset RST 7.5
0 Set
1 Reset (In spite of bit 2 or bit 3)

1.12 I/O port address

Upper I/O address LSB MSB	Function
1000	ROM Cassette
1001	System control port
1010	bank control port
1011	PIO 81C55 port
1100	UART data I/O port
1101	UART data control port
1110	Keyboard
1111	LCD

(1) ROM cassette

128K bytes ROM cassette select and A16

1000	0000	OUT 80H (128D)

7	6	5	4	3	2	1	0
-	-	-	-	-	-	ROM SEL	A16

A16 Address 16

ROM SEL	ROM cassette select
0	128K bytes ROM erase select
1	128K bytes ROM select

128K bytes ROM cassette low address

1000	0100	OUT 84H (132D)

7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0

A7 - A0 ROM cassette low address

128K bytes ROM cassette high address

1000	1000	OUT 88H (136D)

7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A9	A8

A15 - A8 Rom cassette high address

128K bytes ROM data read

1000	1100	IN 8CH (140D)

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0

D7 - D0 ROM data

(2) System control

1001	0000	OUT 90H (144D)

7	6	5	4	3	2	1	0
SELA	SELB	PSTB	TSTB	REMOTE	-	-	-

REMOTE Cassette motor control

0	OFF
1	ON

TSTB Command strobe to clock

0	OFF
1	ON

PSTB Strobe to printer

0	OFF
1	ON

SELA	SELB	Serial I/F select
0	0	Not used
0	1	RAM file
1	0	Floppy disk
1	1	RS-232C

(3) Bank control

```
+----+----+
|1010|0001|       OUT  A1H (161D)
+----+----+
```

```
  7   6   5   4     3       2       1       0
+---+---+---+---+-------+-------+-------+-------+
| - | - | - | - | HADR2 | HADR1 | LADR2 | LADR1 |
+---+---+---+---+-------+-------+-------+-------+
```

LADR2	LADR1	Low Address 0000-7FFF selection
0	0	Bank #0 (ROM #0)
0	1	Bank #1 (ROM #1)
1	0	Bank #2 (RAM #2)
1	1	Bank #3 (RAM #3)

HADR2	HADR1	High Address 8000-FFFF selection
0	0	Standard RAM (RAM #1)
0	1	Not used
1	0	BANK #2 (RAM #2)
1	1	BANK #3 (RAM #3)

(4) Bank status

The current status of the memory, the status of the bank switching, can be examined by the IN instruction. The IN instruction reads 8 data bits from the specified I/O port.

```
      +---+---+---+---+---+---+---+---+
MSB   | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  IN A0H (160D)
      +---+---+---+---+---+---+---+---+
```

Bit 7 --- Serial Interface status #2
Bit 6 --- Serial Interface status #1
Bit 5 --- Not Used
Bit 4 --- Not Used
Bit 3 --- High address (8000H - FFFFH) status #2
Bit 2 --- High address (8000H - FFFFH) status #1
Bit 1 --- Low address (0000H - 7FFFH) status #2
Bit 0 --- Low address (0000H - 7FFFH) 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 XX for more information on the Serial Interface.

(5) PIO 81C55 address

I/O address	Selection
10111000	Inside command/status register
10111001	General I/O port A (PA0-PA7)
10111010	General I/O port B (PB0-PB7)
10111100	General I/O port C (PC0-PC5)

Port A out

1011	1001	OUT B9H (185D)

7	6	5	4	3	2	1	0
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
KS7	KS6	KS5	KS4	KS3	KS2	KS1	KS0
--	--	--	CCK	CD0	C2	C1	C0

PA7 - PA0	LCD chip select
PD7 - PD0	Printer data
KS7 - KS0	Keyboard data
C2 - C0	Clock command Out Port
CD0	Clock data out port
CCK	Calendar shift clock
0	Clock OFF
1	Clock ON

Port B out

1011	1010

OUT BAH (186D)

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

PB1 - PB0 LCD chip select

MC Melody control port
0 ON
1 OFF

DCD/RD RS-232C's DCD/RD select
0 Ring detect
1 Data carrier detect

APO Auto power off out
0 OFF
1 ON

BELL Buzzer out
0 Yes
1 No

DTR RS-232C DTR out, Active low

RTS RTS out, Active low

Port C Input

1011	1011	IN	BBH (187D)

7	6	5	4	3	2	1	0
-	-	DSR	CTS	BCR	BUSY	SLCT	CD

CDI — Clock data in port

SLCT — Printer select
- 0 — Printer deselect (Erase select)
- 1 — Printer select

BUSY — Printer busy
- 0 — Printer ready
- 1 — Printer busy

BCR — Bar code reader data input

CTS — CTS in active low

DSR — RS-232C DSR in active low

(6) UART data I/O port

1100	1000	IN/OUT	C8H (200D)

7	6	5	4	3	2	1	0

UART data

(7) UART control port

Command write

1101	1000	OUT D8H (216D)

7	6	5	4	3	2	1	0
-	-	-	CLS2	CLS1	P	EPE	SBS

SBS — Stop bit select
- 0: Stop bit length 1 bit
- 1: Stop bit length 1.5 bit in case data length is 5 bit other case 2 bit

EPE — Even parity enable
- 0: Odd parity
- 1: Even parity

PI — Parity inhibit
- 0: Parity generation check
- 1: Parity generation check inhibit

CLS2	CLS1	Character length select
0	0	Data length 5 bit
0	1	Data length 6 bit
1	0	Data length 7 bit
1	1	Data length 8 bit

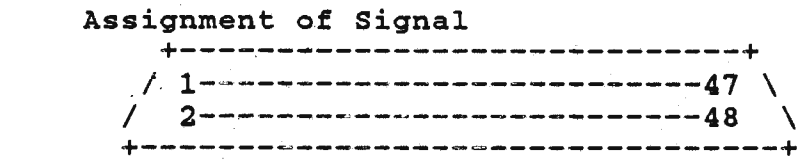
Status read

1101	1000	IN	D8H (216D)

7	6	5	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
0	No error
1	Overrun error generation
FE	Framing error
0	No error
1	Framing error generation
PE	Parity error
0	No error
1	parity error generation
TBRE	Transmitter buffer register empty
0	Empty
1	To transmit new data to TBR
LPS	Low power signal
0	Enough power
1	Falling power voltage

SYSTEM SLOT



Functions of the Pin Signals

1. VDD (OUT) [Pins 1 and 2]
If you don't use the BCD, this Pin can supply the current with 50mA.

2. AD0 - AD7 (IN/OUT) [Pins 3 - 10]
The 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) [Pins 13 - 20]
The most significant 8 bits of the memory address or the I/O address. The output goes off during the Hold mode, it then becomes "H" level, because it is connected to a pull up resister (100k Ohm) inside.

4. /RD (OUT/3-state) [Pin 27]
The read control signal, 3-state during Hold mode.

5. /WR (OUT/3-state) [Pin 28]
The write control signal, 3-state during Hold mode.

6. IO/M (OUT/3-state) [Pin 29]
When this signal is "H" level and "L" level, respectively, the CPU has access to the I/O and the memory. 3-state during Hold mode.

7. ALE (OUT/3-state) [Pin 30]
It is used to strobe the address information (AD0-AD7). 3-state during Hold mode.

8. HOLD (IN) [Pin 31]
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-states and the AD8 - AD15 lines are "H" level.

9. HLDA (OUT) [Pin 32]
It indicates that the CPU has received the HOLD request and that it will relinquish the bus in the next clock cycle.

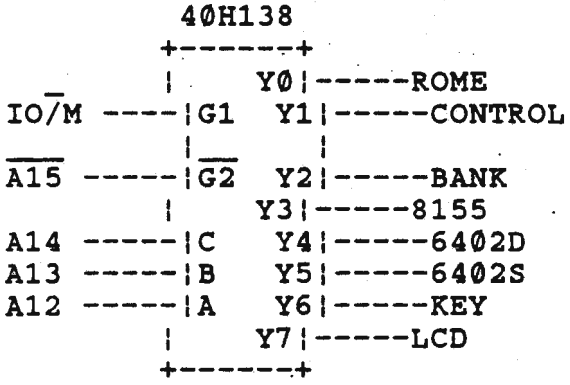
10. INTR (IN) [Pin 33]
    The general purpose interrupt. It is sampled only during the next to the last clock cycle of an instruction and during Hold and Halt states.

11. /INTA (OUT) [Pin 34]
    It is used instead of (and has the same timing as) /RD during the instruction cycle after an INTR is accepted.

12. RESETO (OUT) [Pin 35]
    It indicates the CPU is being reset. Can be used as a system reset.

13. READY (IN) [Pin 36]
    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) [Pin 37]
    The enable signal for external ROM cartridge or general purpose. When the upper 4 bits of the I/O address are 8, it goes "L".



15. E (OUT) [Pin 38]
    It is used as a memory enable signal of the read or write cycle. E is the logical OR (active high) of the /RD and /WR.

16. /BANK 3 (OUT) [Pin 39]
    the memory enable signal of the external RAM cartridge.

17. HADRSD (IN) [Pin 41]
    If it is "H", the memory of the high address (8000H to FFFFH) in the PC is disabled.

18. LADRSD (IN) [Pin 42]
    If it is "H", the memory of the LOW address (0H to 7FFFH) in the PC is disabled.

19. CLK (OUT) [Pin 43]
    2.5MHz clock output. It is the same phase as the CPU clock.

20. POWER (OUT) [Pin 44]
    It is the signal /RESET (connected to the CPU) is reversed.

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,/BANK3	1.1

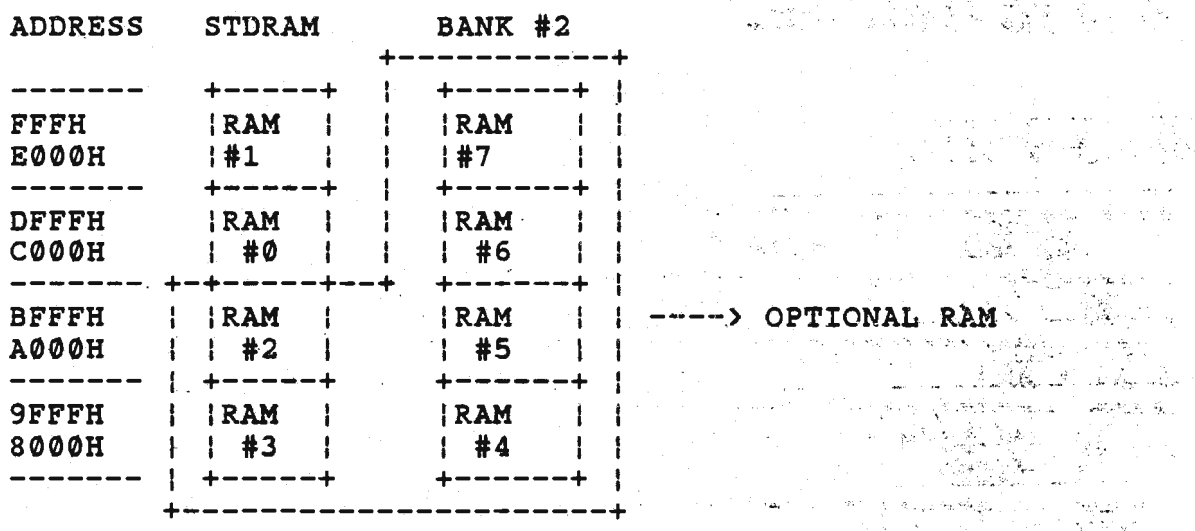
AC CHARACTERISTICS

see page 237 - 239

MEMORY CONTROL CIRCUIT

In this section, RAM #n means the chip number on the main board. The memory of the PC-8300A 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 the user ROM socket (ROM #1) in the PC.

RAM chips (#0 - #7) and ROM (#0 - #1) are 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).



RAM ADDRESSES

The way in which banks are converted is by software control, described in Chapter XX. When the PC is reset, it becomes same mode as 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 64K mode bytes full RAM by optional RAM BANK #2 - #3, one can use a CP/M, etc.