

4.3 Hooks

4.3.1 General

The PINE is provided with some hooks to facilitate the application program to extend the capabilities of the interrupt and BIOS functions. The PINE hooks are:

1. Alarm hook
2. Interrupt hook
3. BIOS hook

This section describes and show the use of the PINE hooks.

4.3.2 Hook Structures

4.3.2.1 Hook jump table

The hook jump table is located between 0FFC0H and 0FFEFH. Its structure is shown below.

FFC0H	ALMHKO	JP EF1FH
FFC3H	ALMHK1	JP EF1FH
FFC6H	ALMHK2	JP EF1FH
FFC9H	ALMHK3	JP EF1FH
FFCCH	ALMHK4	JP EF1FH
FFCFH	RESERVED	JP EF1FH
FFD2H	HK8251	JP EF1FH
FFD5H	ICFHOOK	JP EF1FH
FFD8H	OVFHOOK	JP EF1FH
FFDBH	EXTHOOK	JP EF1FH
FFDEH	TMDT83	JP EF1FH
FFE1H	TMDT85	JP EF1FH
FFE4H	TMDT86	JP EF1FH
FFE7H	BIOSHK	JP EF1FH
FFEAH	RESERVED	JP EF1FH
FFEDH	RESERVED	JP EF1FH

0EF1F contains a RET instruction.

4.3.2.2 Hook types

The PINE hooks are divided into the following types:

- (1) Alarm/wake hooks (5)
 - ALMHK0 (Alarm hook 0)
 - ALMHK1 (Alarm hook 1)
 - ALMHK2 (Alarm hook 2)
 - ALMHK3 (Alarm hook 3)
 - ALMHK4 (Alarm hook 4)
- (2) Interrupt hooks (4)
 - HK8251 (RxDY interrupt hook)
 - ICFHOOK (ICF interrupt hook)
 - OVFHOOK (OVF interrupt hook)
 - EXTHOOK (EXT interrupt hook)
- (3) BIOS hooks (4)
 - TMDT83 (TIMDAT hook)
 - TMDT85 (TIMDAT hook)
 - TMDT86 (TIMDAT hook)
 - BIOSHK (BIOS hook)
- (4) Reserved (3)

HK8251 is not supported by OS version 1.0.

4.3.2.3 Hook structure

The PINE executes its program modules while switching four banks. Depending on which bank the PINE is currently running, the hook processing mode is divided into the following two types:

1. Hooks in which control is transferred to the hook address after bank 0 (RAM) has been selected.
2. Hooks in which control is transferred to the hook address without switching the active bank (system bank).

(1) Hooks which run without switching to bank 0 (RAM)

The active bank is unpredictable when an interrupt occurs. For this reason, interrupt hook processing proceeds as follows:

1. Switch to bank 0 (RAM).
2. Cause the CPU to jump to the specified hook address.
3. Switch to the original bank.

Hooks which run in this mode include OVFHOOK, ICFHOOK, and EXTHOOK.

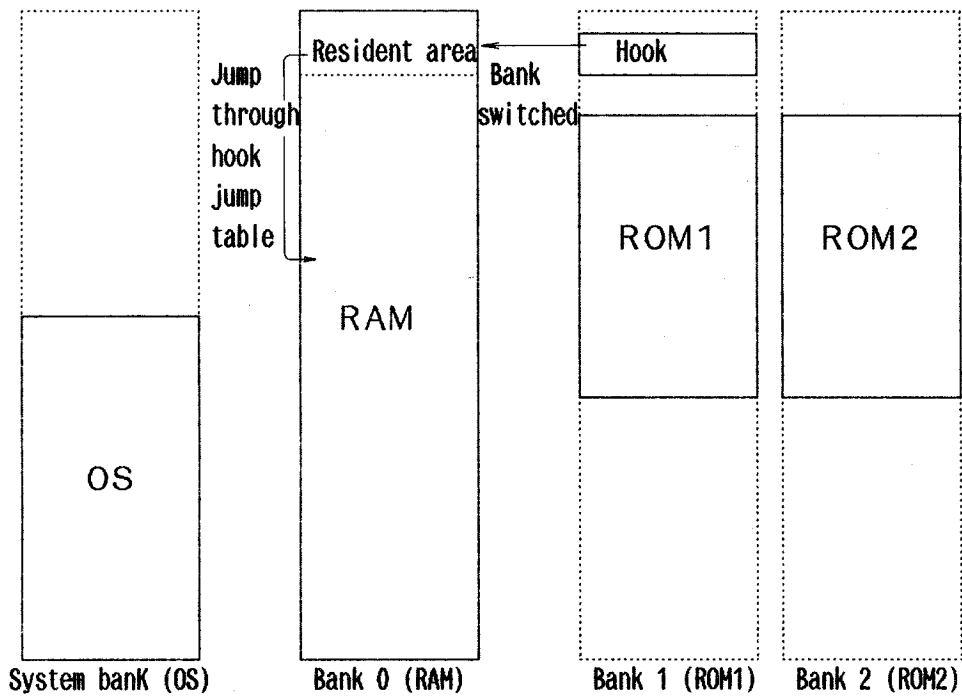


Fig. 4.3.1 Hooks Which Switch Banks

(2) Hooks which run without switching banks.

Hooks which are entered during execution of a module on OS ROM do not switch banks. When rewriting this type of hook, therefore, it is necessary to specify an address not lower than 8000H. Hooks of this type include: ALMHK0, ALMHK1, ALMHK2, ALMHK3, ALMHK4, HK8251, TMDT83, TMDT85, TMDT86, and BIOSHK.

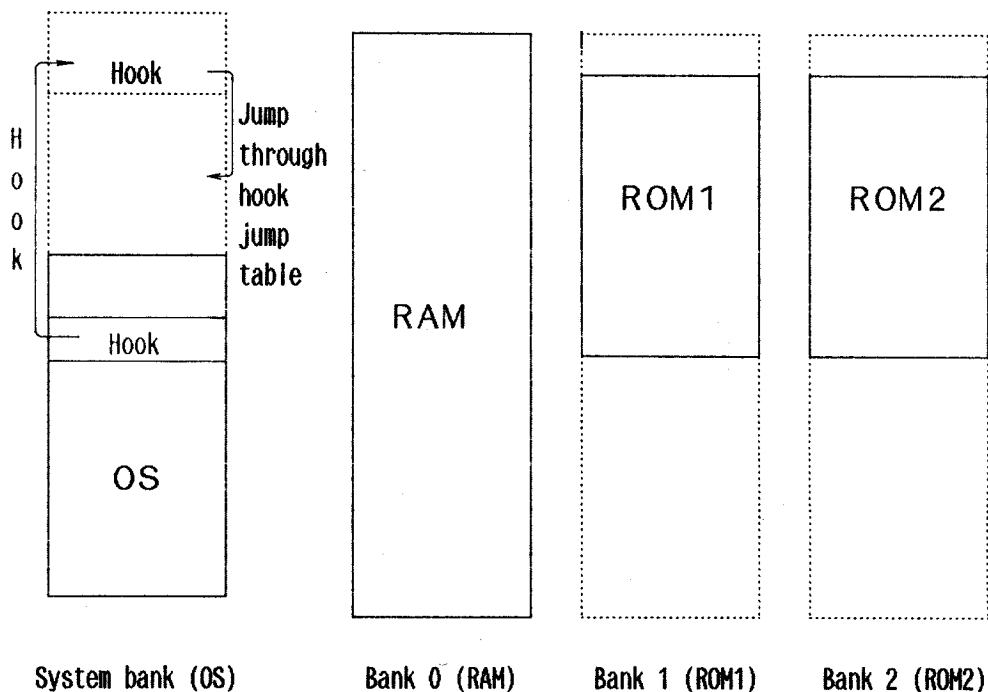


Fig. 4.3.2 Hooks Which Do Not Switch Banks

4.3.2.4 Programming notes

Since hooks are embedded in interrupt and BIOS processing modules, the following programming notes must be taken:

(1) Do not call BIOS/BDOS from a hook.

BIOS and BDOS establish their own stack area at the beginning of execution. Any call to BIOS or BDOS from a hook may destroy the stack area, causing a program hangup. To use a BIOS or BDOS function from a hook, call directly the BIOS or BDOS function on the OS ROM.

(2) Set up a stack area.

A stack overflow condition may occur if the called hook uses the stack established by the calling module. To prevent stack overflows, the hook processing modules must establish their own stack area. The stack area for hooks which do not switch banks must be located in a RAM area not lower than 8000H.

(3) Registers must be saved by the calling module.

The modules that receive control from a hook save the contents of only the required registers. The hook processing modules, therefore, must save all registers that are to be used during their execution. Failure to take this measure when implementing ICFHOOK or EXTHOOK may cause a program hangup on return from the interrupt processing module.

(4) Maintain the interrupt state.

Some interrupt modules run in the DI state and prevent subsequent interrupts. Accordingly, the hook processing modules must keep track of the interrupt state by themselves. The interrupt state when hooks are entered are as follows:

ALMHK0	DI state
ALMHK1	DI state
ALMHK2	EI state, 7508 disabled
ALMHK3	EI state, 7508 disabled
ALMHK4	EI state
HK8251	DI state
ICFHOOK	DI state
OVFHOOK	DI state
EXTHOOK	DI state
TMDT83	EI state, 7508 disabled
TMDT85	EI state, 7508 disabled
TMDT86	EI state, 7508 disabled
BIOSHK	EI state

(5) Disable interrupts when rewriting a hook.

Disable interrupts when rewriting the hook jump table. Particularly, be sure to disable interrupts when rewriting address data, one byte at a time.

(6)

The hook jump table is initialized during system initialize and reset.

4.3.3 Alarm Hooks

The PINE has five hooks for alarm processing. These hooks are used for automatic alarm/wake data update processing. The locations of the hooks are listed below.

Alarm hook 0: Within the module that is activated when an alarm occurs in the power-off state and immediately before the alarm screen is displayed.

Alarm hook 1: Within the module that is activated when an alarm occurs in the power-off state and immediately after the alarm screen disappears.

Alarm hook 2: Within the module that is activated when an alarm occurs in the power-on state and immediately before the alarm screen is displayed.

Alarm hook 3: Within the module that is activated when an alarm occurs in the power-on state and immediately after the alarm screen disappears.

Alarm hook 4: Within the module that is activated when an alarm occurs and immediately before the module is exited due to a condition such as the depression of the ESC key.

The relationships between the alarm processing modules are shown in Figures. 4.3.3 through 4.3.5.

The alarm/wake update processing module, for example, sets the next alarm or wake time using alarm hook 0 and alarm hook 2 when an alarm or wake condition occurs.

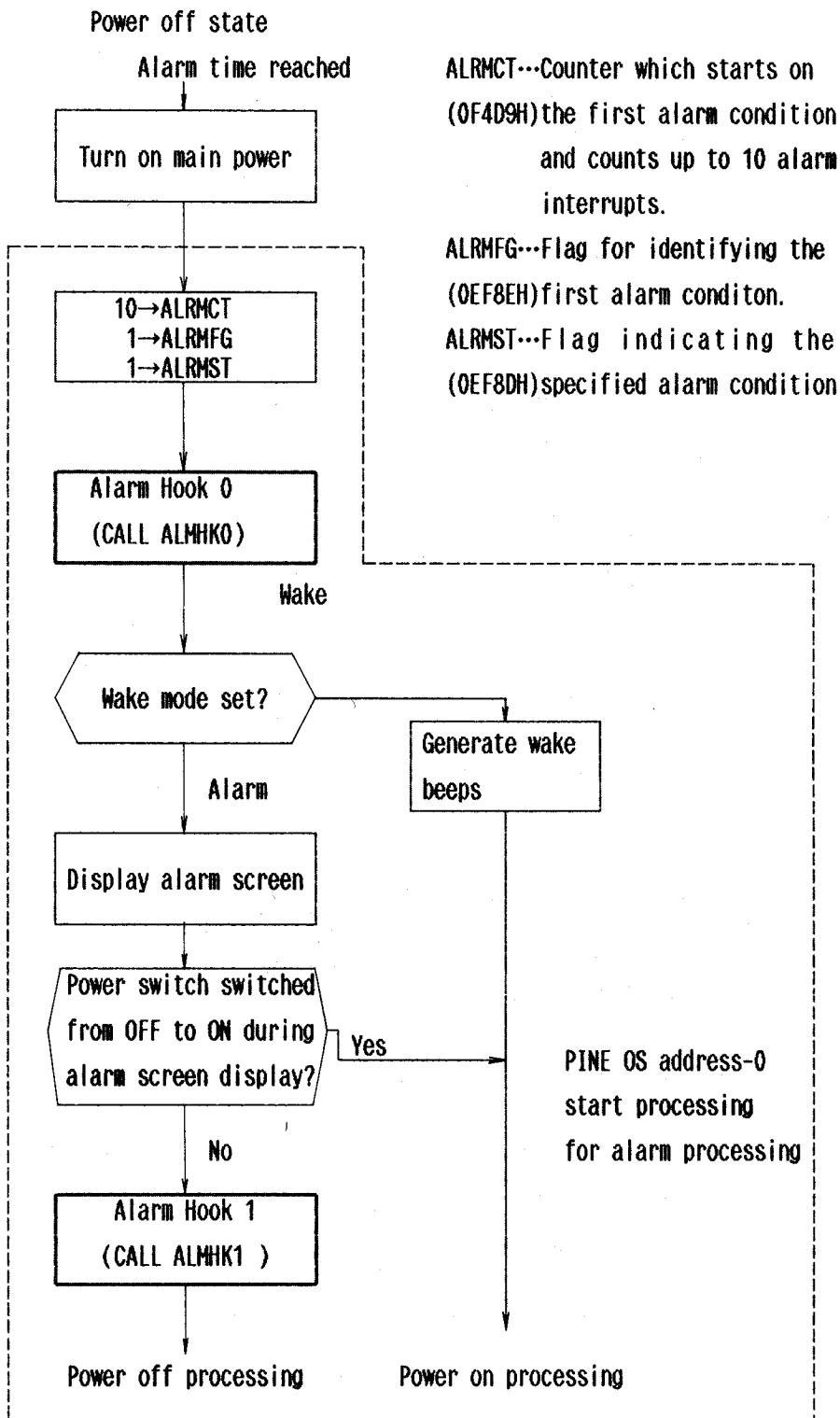
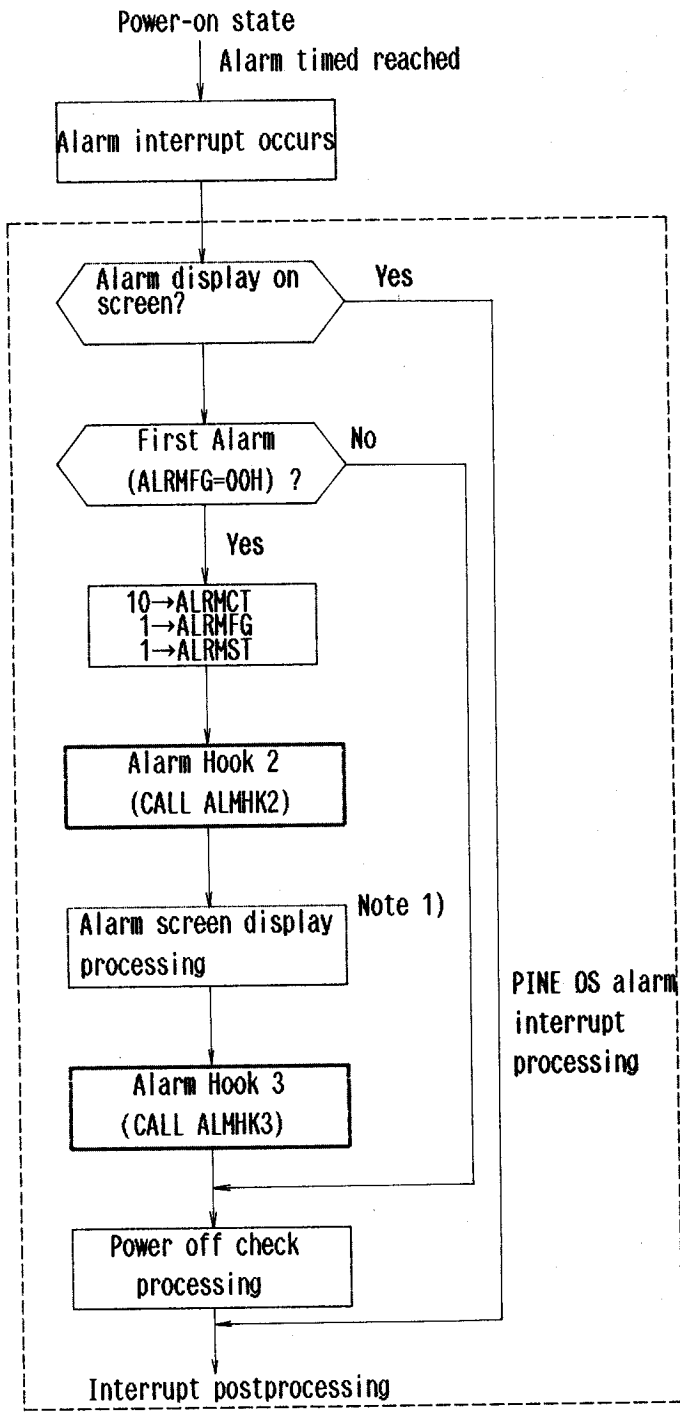


Fig. 4.3.3 Alarm Processing in the Power-off State

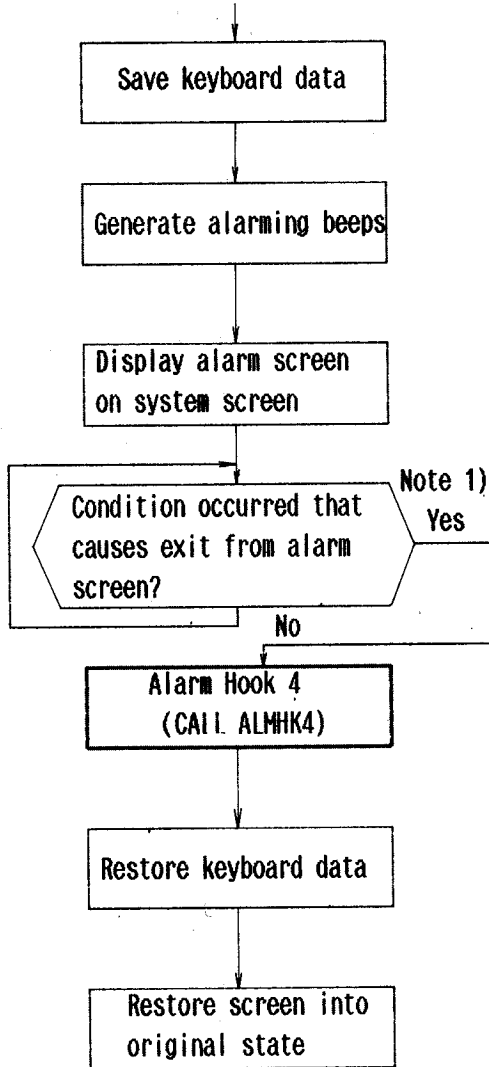


Note 1)
Alarm screen display processing is bypassed if alarm screen is disabled.

PINE OS alarm interrupt processing

Fig. 4.3.4 Alarm Processing in the Power-on State

Alarm screen display processing



Note 1) Conditions that causes the alarm screen processing to be exited include:
ESC is typed.
50 seconds have elapsed.
Power switch is turned off.
Batty is discharged.

Fig. 4.3,5 Alarm Screen Display Processing

4.3.4 Interrupt Hooks

The PINE has four hooks for interrupt processing. The locations of the hooks are listed below.

HK8251: Within the ART interrupt processing module and immediately before data is received.

ICFHOOK: Within the ICF interrupt processing module

OVFHOOK: Within the OVF interrupt processing module and immediately before blink processing.

EXTHOOK: Within the EXT interrupt processing module

HK8251 is not supported by OS Export version 1.0.

4.3.4.1 HK8251

HK8251 is located on OS ROM and provides extended ART interrupt processing. Figure 4.3.6 shows the relationship of HK8251 to interrupt processing.

As seen from the figure, HK8251 just returns control doing nothing when an interrupt occurs unless the RSIOX OPEN function has been executed.

HK8251 is not supported by OS version 1.0.

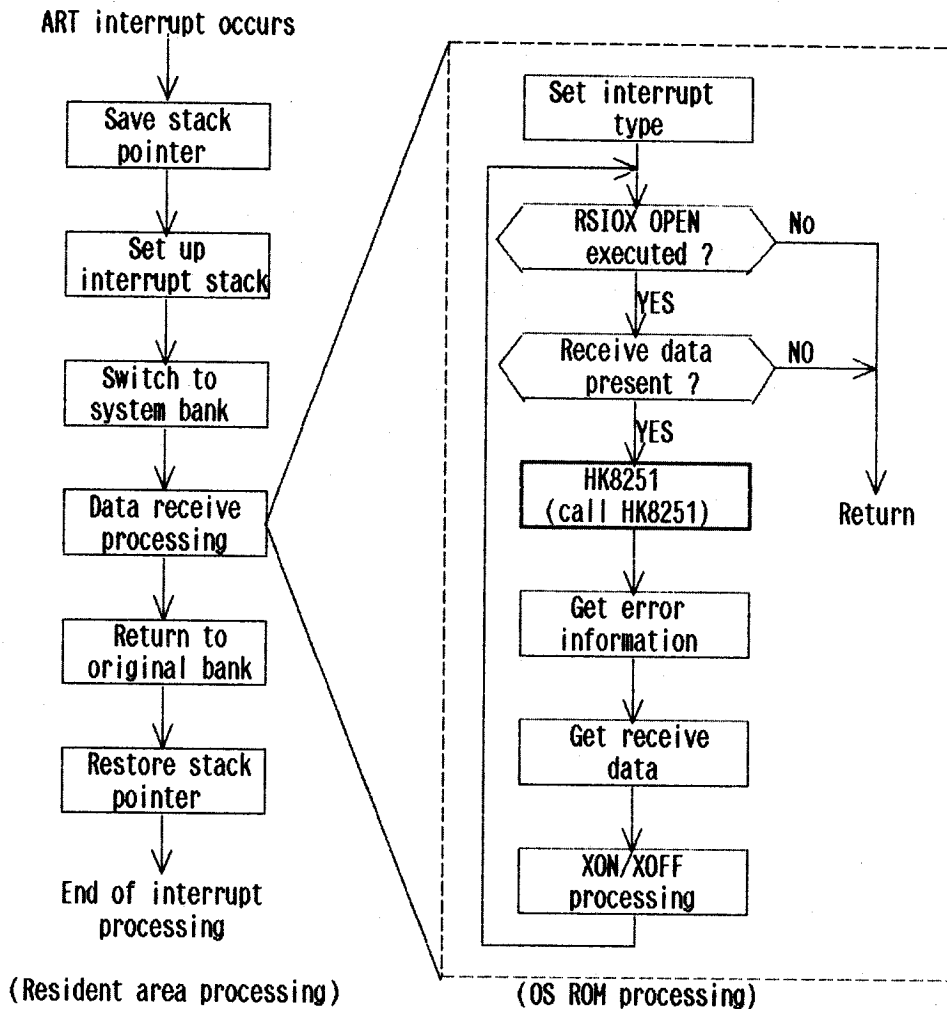


Fig. 4.3.6 ART Interrupt Processing

(2) ICFHOOK, OVFHOOK, and EXTHOOK

During ICF, OVF, and EXT interrupt processing, the active bank is switched to bank 0 (RAM) before the pertinent hook is called. After this, the bank is returned to the original bank.

Figure 4.3.7 shows the relationship of ICFHOOK, OVFHOOK, and EXTHOOK to the interrupt processing modules.

Since the minimum required size of stack area is reserved during ICF, OVF, and EXT processing, it will be necessary to reserve a larger stack area when interrupt processing is to be expanded. Since neither ICF nor EXT interrupt processing modules reset the interrupt signal, it must be reset by the corresponding hook.

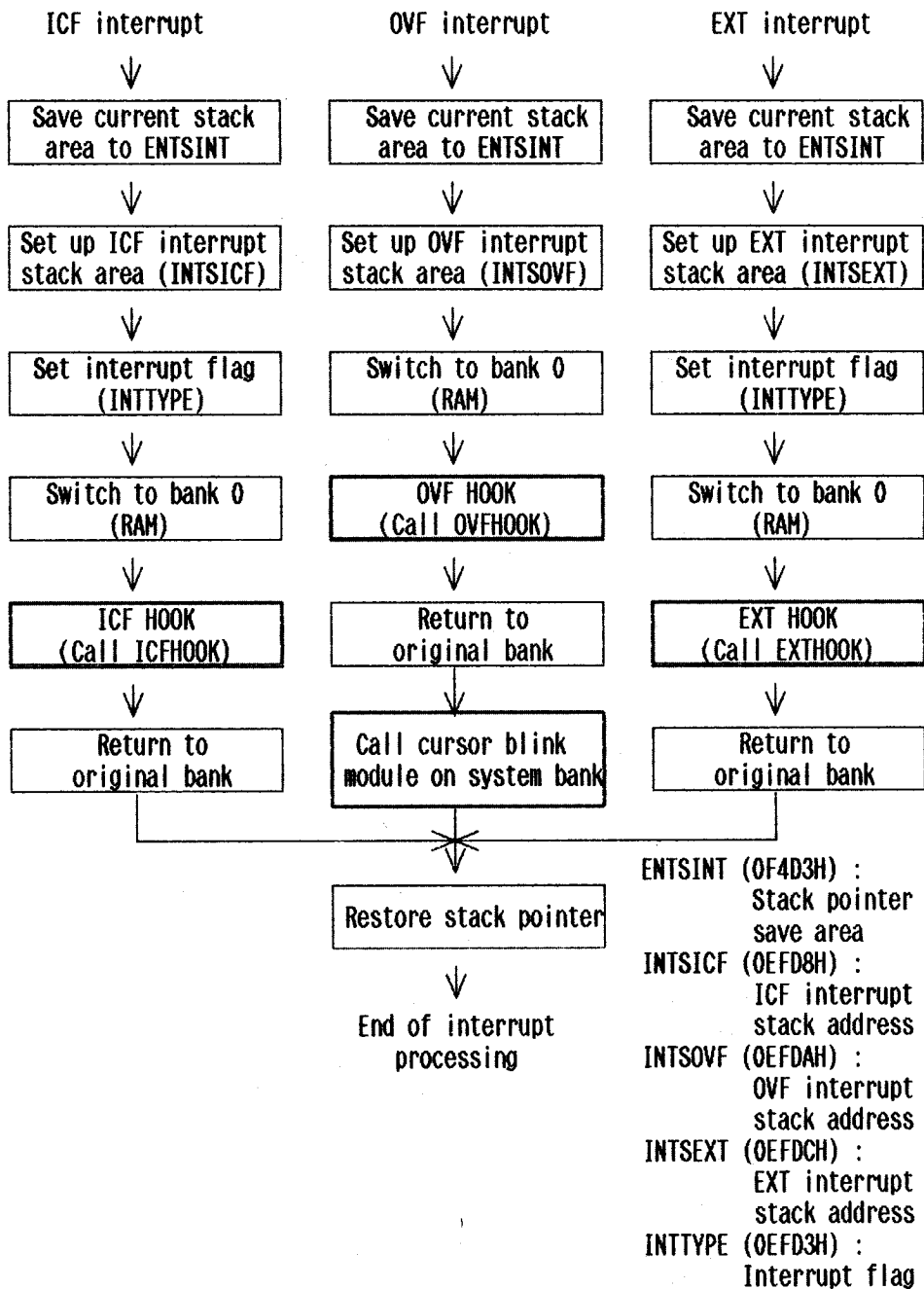


Fig. 4.3.7 ICF/OVF/EXT Interrupt Processing

4.3.5 TIMDAT Hooks

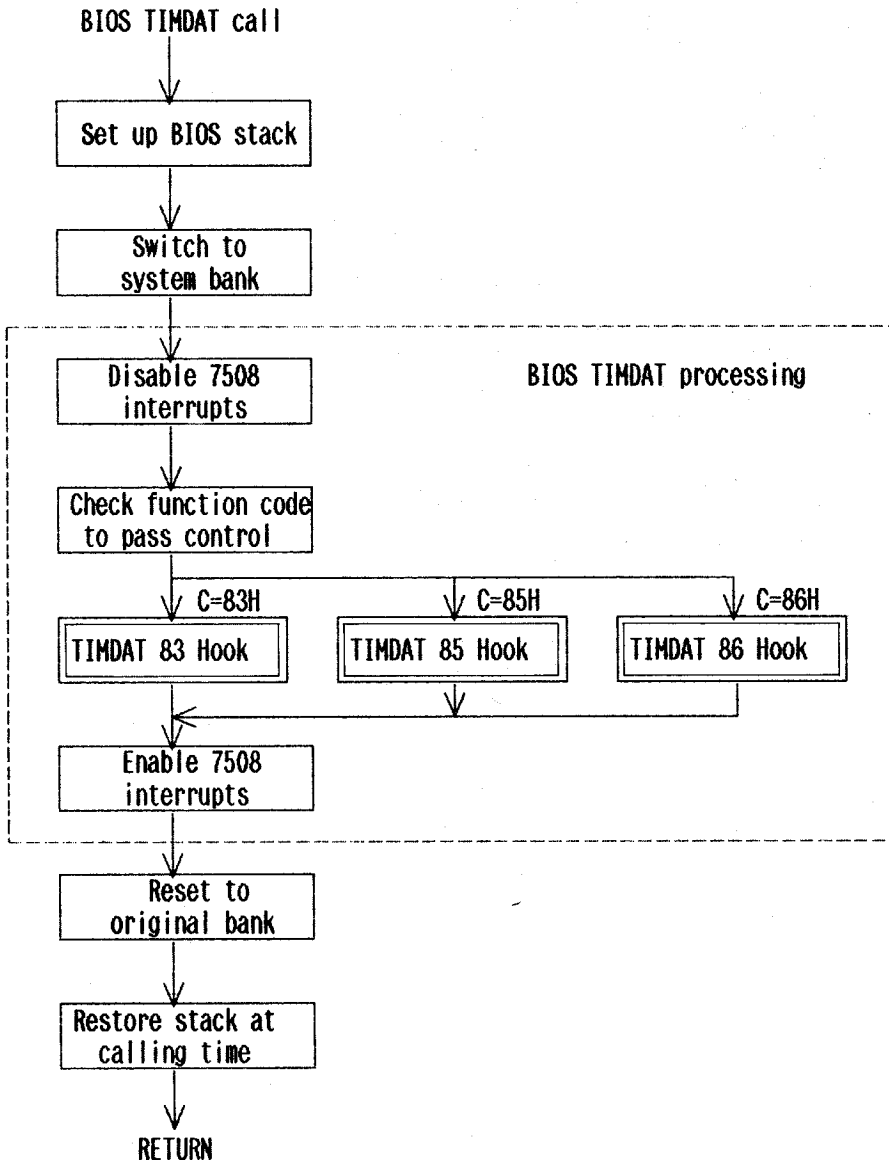
The PINE has the following three hooks for extended BIOS TIMDAT functions.:

TMDT83: Called during BIOS TIMDAT function with C set to 83H.

TMDT85: Called during BIOS TIMDAT function with C set to 85H.

TMDT86: Called during BIOS TIMDAT function with C set to 86H.

These hooks are used to extend the TIMDAT functions. Only the DE registers are preserved when TIMDAT is called.



(TIMDAT processing proper is indicated by broken lines.)

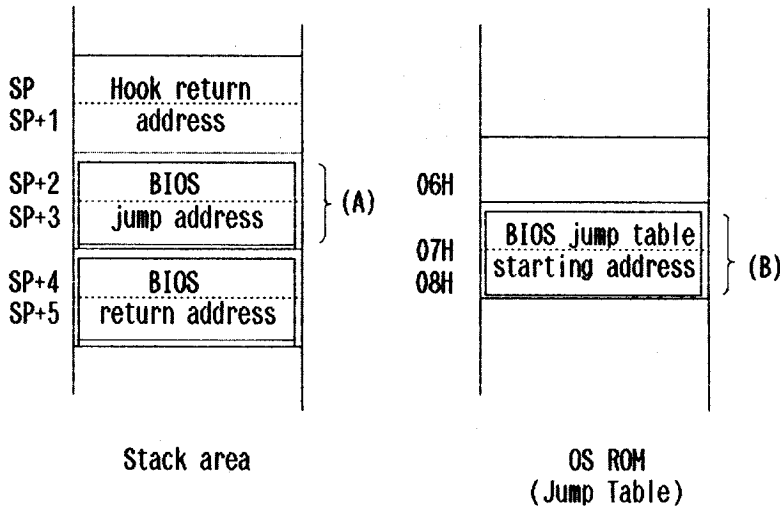
Fig. 4.3.8 TIMDAT Processing

4.3.6 BIOS Hook

BIOSHK is taken when BIOS is called by BDOS or by an application program and provides extended BIOS functions.

BIOSHK processing must proceed as follows:

- (1) Calculate the BIOS function number.



The logical BIOS number can be calculated using the equation:

$$\{(A) - (B)\} / 3$$

where (A) is the BIOS jump address and (B) is the BIOS jump table starting address. For example, the result of the above computation for WBOOT is 01H and that for CONST is 02H.

(2) Standard BIOS function processing

If the result of step (1) is found to be a standard BIOS call, BIOSHK immediately returns control after restoring the registers into the state before BIOSHK was entered.

(3) Extended BIOS function processing

If the result of step (1) is found to be an extended BIOS call, BIOSHK transfers control to the application program for the extended service. When the system BIOS service is to be executed after the extended processing is completed, BIOSHK restores the registers into the state before BIOSHK was entered and returns control to the calling module. When bypassing the system BIOS service after the extended BIOS processing, BIOSHK returns control directly to the point at the BIOS return address that is stored in the stack area shown in paragraph (1). This allows the system to return control to the program that called BIOS after executing PSTBIOS (including bank switching and stack swapping).

See 3.3.3, "BIOS Hook" for use examples of BIOS hooks.

 HOOK SAMPLE PROGRAM

NOTE : This sample program is testing hook,

<> assemble condition <>

.Z80

<> loading address <>

.PHASE 100H

<> constant values <>

BIOS entry

EB03	WBOOT	EQU	0EB03H	; Warm Boot entry
EB09	CONIN	EQU	WBOOT +06H	; Console in entry
EB0C	CONOUT	EQU	WBOOT +09H	; Console out entry
EB69	CALLX	EQU	WBOOT +66H	; Call extra entry
EB87	RESIDENT	EQU	WBOOT +84H	; Resident entry

System area

EF94	TOPRAM	EQU	0EF94H	; Top of User BIOS
EF2D	USERBIOS	EQU	0EF2DH	; Size of User BIOS area
F52E	DISBNK	EQU	0F52EH	; Destination bank for CALLX
F53E	RZIER	EQU	0F53EH	; Interrupt enable register

FFC0	HOOKTBL	EQU	0FFC0H	; Hook table top address
------	---------	-----	--------	--------------------------

k value

00FF	SYSBANK	EQU	0FFH	; System bank
0000	BANK0	EQU	000H	; Bank 0 (RAM)
0001	BANK1	EQU	001H	; Bank 1 (ROM capsel 1)
0002	BANK2	EQU	002H	; Bank 2 (ROM capsel 2)

CBF0	UB_HEAD	EQU	0CBF0H	; Top addr of User BIOS area's header
CBFB	UB_OVWRITE	EQU	UB_HEAD +11	; Over write flag
CBFC	UB_RELEASE	EQU	UB_HEAD +12	; Release address

OS ROM jump table

0006	BIOSJT	EQU	00006H
003C	XUSRSCRN	EQU	0003CH

Resident jump table

FF9C	SELBNK	EQU	0FF9CH
000D	CR	EQU	0DH
000A	LF	EQU	0AH
0012	CLS	EQU	12H

 MAIN PROGRAM

NOTE : Using system hook jump program.

CAUTION : This program uses User BIOS area. Usually, you must check that another already used this area. But this program doesn't check it.

This program doesn't do resetting hook. So, you wish to stop test hook, you must push reset bottom.

0100		MAIN:	JP	START	; Program start
0100	C3 03AD				

***** Output message data

0103		MSGX;	DB	'*** User BIOS size error ***',CR,LF
0103	2A 2A 2A 20			
0107	55 73 65 72			
010B	20 42 49 4F			
010F	53 20 73 69			
0113	7A 65 20 65			
0117	72 72 6F 72			
011B	20 2A 2A 2A			
011F	0D 0A			
0121	20 20 53 65		DB	' Set User BIOS size to 2 k bytes!!'
0125	74 20 55 73			
0129	65 72 20 42			
012D	49 4F 53 20			
0131	73 69 7A 65			
0135	20 74 6F 20			
0139	32 20 6B 20			
013D	62 79 74 65			
0141	73 21 21			
0144	0D 0A		DB	CR,LF
0146	00		DB	00H

0147		MSG0:	DB	CLS,00H
0147	12 00			

```

0149
0149 OD 0A
014B 2A 2A 2A 20
014F 48 6F 6F 6B
0153 20 74 65 73
0157 74 20 70 72
015B 6F 67 72 61
015F 6D 20 2A 2A
0163 2A
0164 OD 0A
0166 20 20 20 31
016A 20 20 20 2D
016E 2D 20 41 4C
0172 4D 48 4B 58
0176 20 20 74 65
017A 73 74 OD 0A
017E 20 20 20 32
0182 2C 33 20 2D
0186 2D 20 52 65
018A 73 65 72 76
018E 65 20 20 20
0192 20 20 OD 0A
0196 20 20 20 33
019A 20 20 20 2D
019E 2D 20 46 4E
01A2 36 32 35 31
01A6 20 20 74 65
01AA 73 74 OD 0A
01AE 20 20 20 34
01B2 20 20 20 2D
01B6 2D 20 49 43
01BA 46 48 4F 4F
01BE 4B 20 74 65
01C2 73 74 OD 0A
01C6 20 20 20 35
01CA 20 20 20 2D
01CE 2D 20 4F 56
01D2 46 48 4F 4F
01D6 4B 20 74 65
01DA 73 74 OD 0A
01DE 20 20 20 36
01E2 20 20 20 2D
01E6 2D 20 45 58
01EA 54 48 4F 4F
01EE 4B 20 74 65
01F2 73 74 OD 0A
01F6 20 20 20 37
01FA 20 20 20 2D
01FE 2D 20 54 49
0202 4D 44 41 54
0206 20 20 74 65
020A 73 74 OD 0A
020E 20 53 65 6C
0212 65 63 74 20
0216 6F 6E 65 20
021A 6F 66 20 31
021E 20 74 6F 20
0222 37 20 2D 2D
0226 20
0227 00

MSG1:
DB CR,LF
DB '*** Hook test program ***'

DB CR,LF
DB ' 1 -- ALMHKX test',CR,LF

DB ' 2.3 -- Reserve ',CR,LF

DB ' 3 -- HK8251 test',CR,LF

DB ' 4 -- ICFHOOK test',CR,LF

DB ' 5 -- OVTHOOK test',CR,LF

DB ' 6 -- EXTHOOK test',CR,LF

DB ' 7 -- TIMDAT test',CR,LF

DB ' Select one of 1 to 7 -- '

DB 00H

MSG3:
DB CR,LF
DB CR,LF
DB '*** Alarm hook test ***'

DB CR,LF
DB ' 1 -- ALMHK1',CR,LF

DB ' 2 -- ALMHK2',CR,LF

DB ' 3 -- ALMHK3',CR,LF

DB ' 4 -- ALMHK4',CR,LF

DB ' 5 -- ALMHK5',CR,LF

DB ' Select one of 1 to 5 -- '

DB 00H

MSG4:
DB CR,LF
DB CR,LF
DB '*** Interrupt hook test ***'

```

```

02C1 70 74 20 68
02C5 6F 6F 68 20
02C9 74 65 73 74
02CD 20 20 2A 2A
02D1 2A
02D2 0D 0A
02D4 20 57 68 65
02D8 6E 20 69 6E
02DC 74 65 72 72
02E0 75 70 74 20
02E4 6F 63 63 75
02E8 72 2C 0D 0A
02EC 20 20 70 72
02F0 69 6E 74 20
02F4 6D 65 73 73
02F8 61 67 65 20
02FC 22 69 6E 74
0300 65 72 72 75
0304 70 74 22 2E
0308 0D 0A
030A 00

030B
030B 0D 0A
030D 0D 0A
030F 2A 2A 2A 20
0313 20 54 49 4D
0317 44 41 54 20
031B 74 65 73 74
031F 20 20 2A 2A
0323 2A
0324 0D 0A
0326 20 57 68 65
032A 6E 20 74 69
032E 6D 64 61 74
0332 20 72 6F 75
0336 74 69 6E 65
033A 20 63 61 6C
033E 6C 2C 0D 0A
0342 20 20 70 72
0346 69 6E 74 20
034A 6D 65 73 75
034E 61 67 65 20
0352 22 54 49 4D
0356 44 41 54 22
035A 2E 0D 0A
035D 20 20 20 31
0361 20 2D 2D 20
0365 54 49 4D 44
0369 41 54 38 33
036D 0D 0A
036F 20 20 20 32
0373 20 2D 2D 20
0377 54 49 4D 44
037B 41 54 38 35
037F 0D 0A
0381 20 20 20 33
0385 20 2D 2D 20
0389 54 49 4D 44
038D 41 54 38 36
0391 0D 0A
0393 20 53 65 6C
0397 65 63 74 20
039B 6F 6E 65 20
039F 6F 66 20 31
03A3 20 74 6F 20
03A7 33 20 2D 2D
03AB 20
03AC 00

03AD
03AD 3A EF2D
03B0 FE 08
03B2 DA 056C

03B5 21 0448
03B8 ED 5B EF94
03BC 01 0124
03BF F3
03C0 ED B0
03C2 FB

03C3 21 0147
03C6 CD 0561

03C9
03C9 31 2000

03CC 0E 01
03CE CD EB87

03D1 21 0149
03D4 CD 0581

03D7
03D7 CD 0575
03DA CA 058D
03DD FE 31
03DF 38 F6
03E1 28 08
03E3 FE 36
03E5 28 2A
03E7 30 E8
03E9 18 18

```

```

DB CR,LF
' When interrupt occur,',CR,LF

DB ' print message "interrupt".',CR,LF

DB 00H

MSG5:
DB CR,LF
DB CR,LF
DB '*** TIMDAT test ***'

DB CR,LF
DB ' When timdat routine call,',CR,LF

DB ' print message "TIMDAT".',CR,LF

DB ' 1 -- TIMDAT83',CR,LF

DB ' 2 -- TIMDAT85',CR,LF

DB ' 3 -- TIMDAT86',CR,LF

DB ' Select one of 1 to 3 -- '

DB 00H

```

```

START:
LD A,(USERBIOS)
CP 08H ; If User BIOS is smaller than 8 k.
JP C,SZERROR ; then error

LD HL,HOOK ; Copy programm data for hook
LD DE,(TOPRAM) ; from hook to topram (userbios)
LD BC,HOOKBTM-HOOK
DI ; Interrupt disable
LDIR

LD HL,MSG0 ; Clear screen
CALL DSPMSG

;
;
;
PTOP1:
LD SP,2000H ; Set stack pointer

LD C,01H ; Set resident flag
CALL RESIDENT

LD HL,MSG1 ; Message (alarm. interrupt. timdat)
CALL DSPMSG

;
;
SELECT2:
CALL CONINF
JP Z,STOPEND ; If stop, then end
CP '1'
JR C,SELECT2 ; If select error, then retry
JR Z,ALARM ; Alarm select
CP '6'
JR Z,TIMDAT ; TIMDAT select
JR NC,SELECT2 ; If select error, then retry
JR INTERRUPT ; else interrupt

***** Alarm hook test *****

```

```

03EB
03EB 21 0228
03EE CD 0581

03F1
03F1 CD 0575
03F4 CA 058D
03F7 FE 31
03F9 38 F6
03FB FE 36
03FD 30 F2

03FF D6 31
0401 18 28

0403
0403 F5
0404 21 02B1
0407 CD 0581
040A F1
040B D6 32
040D C6 05
040F 18 1A

0411
0411 21 030B
0414 CD 0581

0417
0417 CD 0575
041A CA 058D

041D FE 31
041F 38 F6
0421 FE 34
0423 30 F2

0425 D6 31
0427 C6 0A
0429 18 00

042B
042B 4F
042C 87
042D 57
042E 81
042F 4F
0430 06 00
0432 21 FFC0
0435 09
0436 E5

0437 2A EF94
043A 87
043B 4F
043C 09
043D EB

043E E1
043F F3
0440 23
0441 73
0442 23
0443 72
0444 FB

0445 C3 03C9

0448
0448 E5
0449 21 C4A4
044C 18 48

044E E5
044F 21 C4AD
0452 18 42

0454 E5
0455 21 C4B6
0458 18 3C

045A E5
045B 21 C4BF
045E 18 36

0460 E5
0461 21 C4C6
0464 18 30

0466 E5
0467 21 C4D1
046A 18 2A

046C E5
046D 21 C4DA
0470 18 24

0472 E5
0473 21 C4E3

ALARM:
LD HL,MSG3 ; Message (alarm hook 1 -- 5)
CALL DSPMSG

;
SELECT3:
CALL CONINF
LD Z,STOPEND ; If stop, then end
JP '1'
CP '1'
JR C,SELECT3 ; If select error, then retry
CP '6'
JR NC,SELECT3 ; If select error, then retry

;
SUB '1' ; A reg. is 0..3
JR SETHOOK ; Goto hook setting process

;
***** Interrupt hook test *****

;
INTERRUPT:
PUSH AF ; Save selecting number
LD HL,MSG4
CALL DSPMSG ; Interrupt hook select message
POP AF
SUB '2'
ADD A,05H ; A reg. is 5..9
JR SETHOOK ; Goto hook setting process

;
***** TIMDAT hook test *****

;
TIMDAT:
LD HL,MSG5
CALL DSPMSG ; Select TIMDAT message

;
SELECT4:
CALL CONINF
LD Z,STOPEND ; If stop, then end
JP '1'
CP '1'
JR C,SELECT4 ; If select error, then retry
CP '4'
JR NC,SELECT4 ; If select error, then retry

;
SUB '1'
ADD A,10 ; A reg. is 10..12
JR SETHOOK ; Goto hook setting process

;
***** Set hook data for each select *****
A -- Hook logical number (0 -- 12)

;
SETHOOK:
LD C,A ; Get hook data addr
ADD A,A
LD D,A
ADD A,C
LD C,A
LD B,00H
LD HL,HOOKTBL
ADD HL,BC ; HOOKTBL + A*3 --> HL
PUSH HL ; Push target hook table

;
LD HL,(TOPRAM)
ADD A,A
LD C,A
ADD HL,BC ; (TOPRAM) + A*6 --> HL
EX DE,HL

;
POP HL ; Set hook jump addr
DI ; Interrupt disable
INC HL
LD (HL),E
HL HL
INC (HL),D
LD EI

;
JP PTOPI

;
***** User BIOS data *****
This part is copied into user bios area

;
HOOK:
PUSH HL
LD HL,0C400H+PDATA1-HOOK
JR HOOKSTART

;
PUSH HL
LD HL,0C400H+PDATA2-HOOK
JR HOOKSTART

;
PUSH HL
LD HL,0C400H+PDATA3-HOOK
JR HOOKSTART

;
PUSH HL
LD HL,0C400H+PDATA4-HOOK
JR HOOKSTART

;
PUSH HL
LD HL,0C400H+PDATA5-HOOK
JR HOOKSTART

;
PUSH HL
LD HL,0C400H+PDATA6-HOOK
JR HOOKSTART

;
PUSH HL
LD HL,0C400H+PDATA7-HOOK
JR HOOKSTART

;
PUSH HL
LD HL,0C400H+PDATA8-HOOK

```



```

04F0 4D 31 0D 0A
04F4 00
04F5
04F5 41 4C 41 52
04F9 4D 32 0D 0A
04FD 00
04FE
04FE 41 4C 41 52
0502 4D 33 0D 0A
0506 00
0507
0507 41 4C 41 52
050B 4D 34 0D 0A
050F 00
0510
0510 41 4C 41 52
0514 4D 35 0D 0A
0518 00
0519
0519 54 4D 48 4F
051D 4F 4B 0D 0A
0521 00
0522
0522 48 4B 38 32
0526 35 31 0A 0D
052A 00
052B
052B 49 43 46 48
052F 4F 4F 4B 0A
0533 0D 00
0535
0535 4F 56 46 46
0539 4F 4F 4B 0A
053D 0D 00
053F
053F 45 58 54 48
0543 4F 4F 4B 0D
0547 0A 00
0549
0549 54 49 4D 44
054D 41 54 38 33
0551 0D 0A 00
0554
0554 54 49 4D 44
0558 41 54 38 35
055C 0D 0A 00
055F
055F 54 49 4D 44
0563 41 54 38 36
0567 0D 0A 00

056A
056A 0000

056C

056C
056C 21 0103
056F CD 0581
0572 C3 058D

0575
0575 CD EB09
0578 FE 03
057A F5
057B 4F
057C CD EB0C
057F F1
0580 C9

0581
0581 4E
0582 79
0583 B7
0584 C8

0585
0585 CD EB0C
0589 E1
058A 23
058B 18 F4

058D
058D 0E 00
058F CD EB87
0592 C3 EB03

```

```

PDATA2: DB 'ALARM2',CR,LF,0
PDATA3: DB 'ALARM3',CR,LF,0
PDATA4: DB 'ALARM4',CR,LF,0
PDATA5: DB 'ALARMS',CR,LF,0
PDATA6: DB 'TMHOOK',CR,LF,0
PDATA7: DB 'HK8251',0ah,0dh,0
PDATA8: DB 'ICFHOOK',0ah,0dh,0
PDATA9: DB 'OVFHOOK',0ah,0dh,0
PDATA10: DB 'EXTHOOK',CR,LF,0
PDATA11: DB 'TIMDAT83',CR,LF,0
PDATA12: DB 'TIMDAT85',CR,LF,0
PDATA13: DB 'TIMDAT86',CR,LF,0

;
; SAVESP: DW 0000H
;
; HOOKBTM:
;
;
;
; SZERROR:
; LD HL,MSGX ; Error message display
; CALL DSPMSG
; JP STOPEND
;
; ***** Console input routine *****
;
; CONINF:
; CALL CONIN ; Console in
; CP 03H
; PUSH AF ; If stop, then z-flag on
; LD C,A
; CALL CONOUT ; Display inputing char
; POP AF
; RET

;
; ***** Console output routine *****
; IN : HL -- Conout message top addr.
;
; DSPMSG:
; LD C,(HL)
; LD A,C
; OR A ; If data is 0,
; RET Z ; then end of data.
;
; PUSH HL
; CALL CONOUT ; Console output
; POP HL
; INC HL
; JR DSPMSG

;
; ***** Ending process *****
;
; STOPEND
; LD C,00H ; Reset resident flag
; CALL RESIDENT
; JP WBOOT
;
; END

```

4.4 Bank Switching

4.4.1 General

The PINE is provided with four banks of memory which are managed by the PINE CP/M operating system. Normal application programs can access the PINE memory without being aware of these memory banks. There are some advanced application programs which need to control the memory banks directly. For such application programs, this section describes the facilities to control the memory banks and shows how to use those facilities.

4.4.2 Bank Structure

4.4.2.1 Bank types

The PINE memory is divided into the following four banks:

- (1) System bank: Accommodates the OS ROM and the higher portion of the RAM.
- (2) Bank 0: RAM.
- (3) Bank 1: Application program ROM and portion of RAM.
- (4) Bank 2: Application program ROM and portion of RAM.

The four memory banks provide 64K bytes of RAM and 96K bytes (maximum) of ROM.

Banks 1 and 2 are available in three configurations depending on the capacity of the ROM area (8K, 16K, and 32K bytes). PINE OS establishes the memory bank configuration at power-on time by reading the capacity of the ROM area on banks 1 and 2 from their header area. The ROM capacity for bank 1 is 32K bytes (for BASIC interpreter) as standard. This ROM area may be replaced by an application ROM. Figure 4.4.1 shows the PINE memory space.

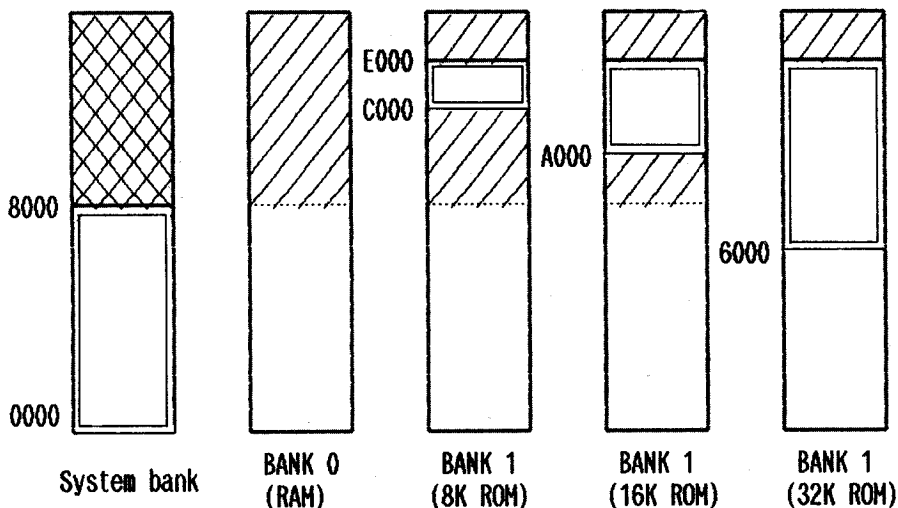
Bank	System	0	1			2		
			8 KB	16 KB	32 KB	8 KB	16 KB	32 KB
FFFF			RAM	RAM	RAM	RAM	RAM	RAM
E000								
	RAM		ROM 1	ROM 1	ROM 1 (BASIC)	ROM 2	ROM 2	ROM 2
C000								
		RAM						
A000								
8000								
6000								
	ROM 0 (OS)							
4000			RAM	RAM	RAM	RAM	RAM	RAM
2000								
0000								

Fig. 4.4.1 PINE Memory Space

4.4.2.2 Common RAM areas

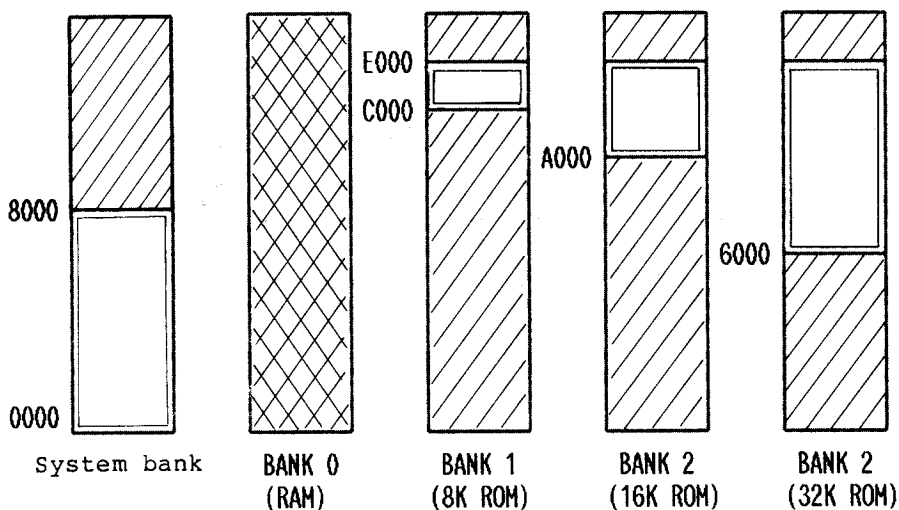
The following areas can be used as common area when banks are switched:

- (1) Switching from system bank to bank 0, 1, or 2



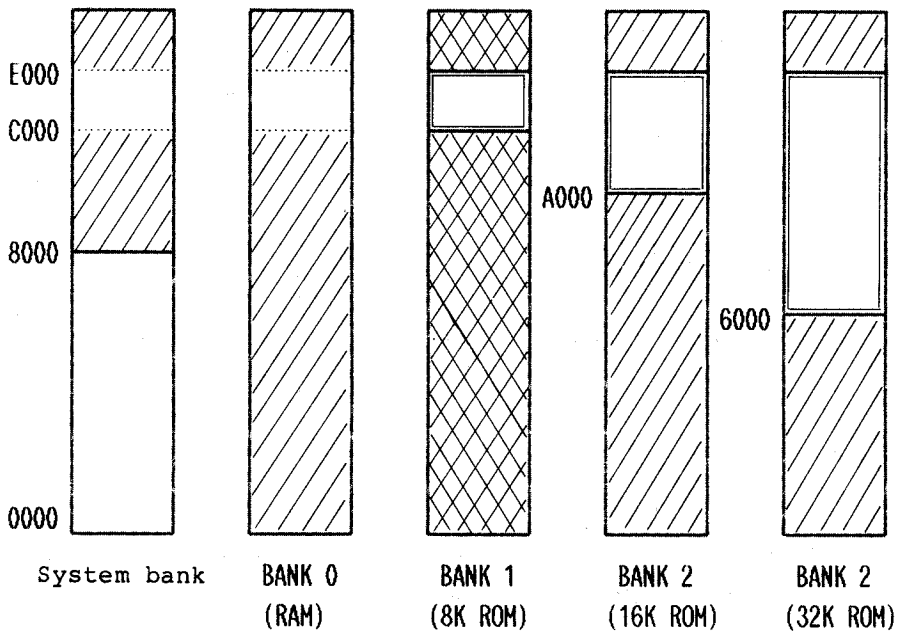
Hatched areas in the above figure are the common RAM areas which remain valid when banks are switched. Areas enclosed in double boxes are made up of ROM. Bank 2 has the same configuration as bank 1.

- (2) Switching from bank 0 to system bank, bank 1 or 2



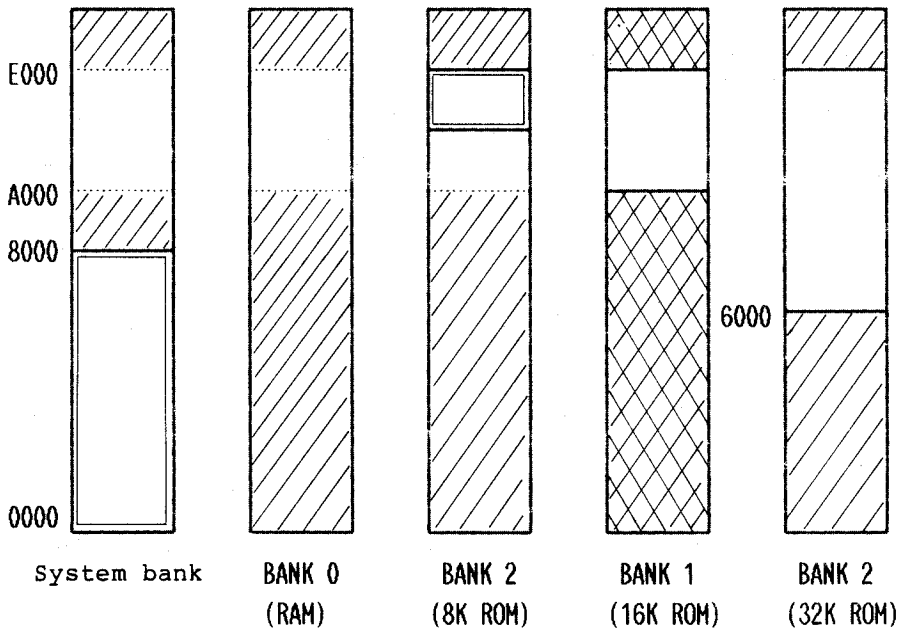
Hatched areas in the above figure are the common RAM areas which remain valid when banks are switched. Bank 2 has the same configuration as bank 1.

(3) Switching from bank 1 to system bank, bank 0 or 2 (8K ROM)



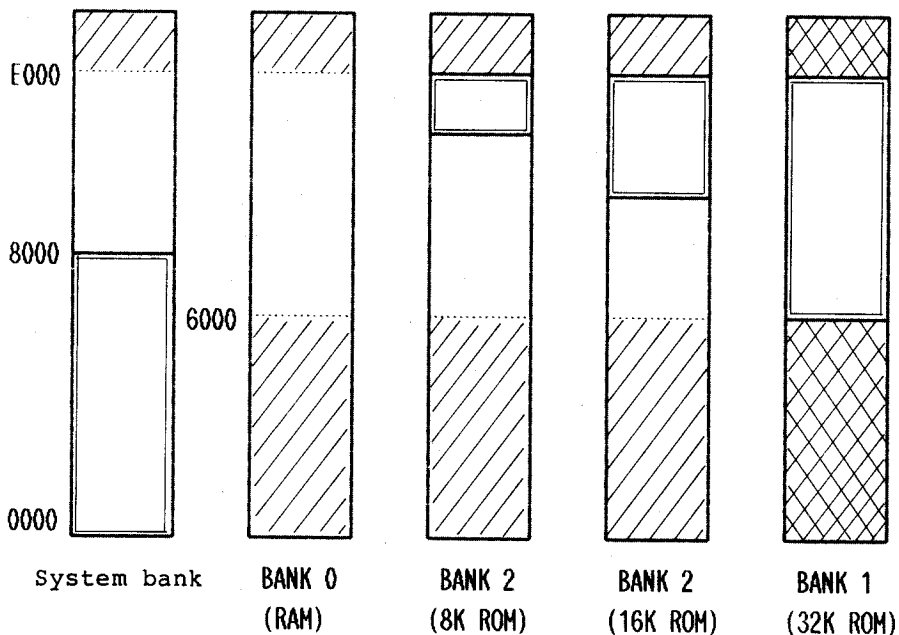
Hatched areas in the above figure are the common RAM areas which remain valid when banks are switched. On bank 2 (8K ROM), the area excluding the ROM area can be used as a common area.

(4) Switching from bank 1 to system bank, bank 0 or 2 (16K ROM)



Hatched areas in the above figure are the common RAM areas which remain valid when banks are switched. On bank 2 (16K ROM), the area excluding the ROM area can be used as a common area.

(5) Switching from bank 1 to system bank, bank 0 or 2 (32K ROM)



Hatched areas in the above figure are the common RAM areas which remain valid when banks are switched. On bank 2 (32K ROM), the area excluding the ROM area can be used as a common area.

4.4.2.3 Programming notes

(1) Banks 1 and 2 have different memory configurations depending on the capacity of the ROM devices in the ROM capsule. PINE OS establishes the memory bank configuration at power-on time by reading the capacity of the ROM area on banks 1 and 2. More specifically, the OS searches for the ROM header and, if one is found, read the ROM size from the header. If no header is found, the OS regards the bank as having no ROM. See the description of ROM capsules in Section 3.8, "Disk Storage" for the ROM header.

(2) The OS switches banks based on the bank information (BNKRG5, BNKRG1, BNKRG2, and BNKRG3) that are examined in paragraph (1).

4.4.3 Utility Routines

PINE OS supplies the following two types of utility routines for bank management:

1. Bank subroutines using BIOS functions
2. Bank subroutines which controls banks directly.

The application program must select one of these subroutine types depending on the purpose of its bank control.

4.4.3.1 Bank switching using BIOS

Bank switching using BIOS is used by ordinary application programs to exchange data between banks. PINE OS provides six BIOS functions for bank switching. Their functions are summarized in the table below.

The application program need not consider the stack area to be used during bank switching because the OS reserves it by itself. See Section 3.4, "BIOS Details" for details of the six BIOS functions.

Name	Address	Description
LOADX	WBOOT + 5AH or 0EB5DH	Reads one byte from the given address on the given bank.
STORX	WBOOT + 5DH or 0EB60H	Writes one byte into the given address on the given bank.
LDIRX	WBOOT + 60H or 0EB63H	Writes the given number of bytes at the given address on the given bank into the bank 0 area at the given address.
JUMPX	WBOOT + 63H or 0EB66H	Causes the CPU to jump to the given address on the given bank.
CALLX	WBOOT + 66H or 0EB69H	Calls the module at the given address on the given bank.
MEMORY	WBOOT + 4EH or 0EB51H	Gets the current bank information.

4.4.3.2 Bank switching using direct bank control

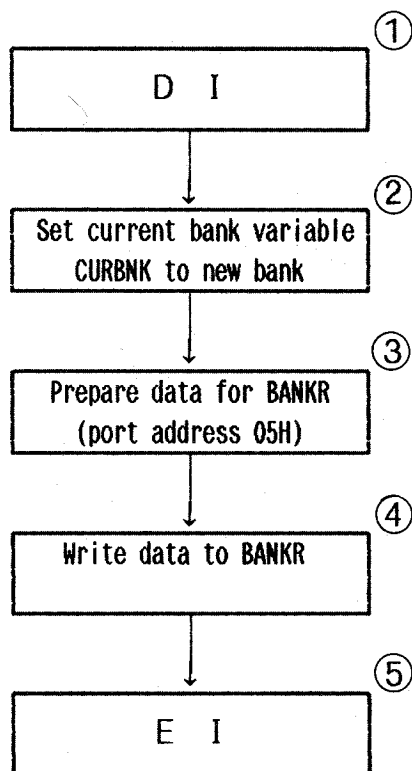
Bank switching using direct bank control is used by special application programs (extending BIOS or interrupt handling functions) which need to exercise direct control over banks. PINE OS provides six bank switching utility routines. Their functions are summarized in the table below.

When using these routines, the application program must reserve a stack area at a location in which it is accessible to the new bank. See Section 4.2, "Jump Tables" for details on the bank switching utility routines.

Name	Address	Description
SELBNK	0FF9CH	Switches to the given bank.
LDAXX	0FF9FH	Reads one byte from the given address on the given bank.
STAXX	0FFA2H	Writes one byte into the given address on the given bank.
LDIRXX	0FFA5H	Writes the given number of bytes at the given address on the given bank into the bank 0 area at the given address.
JUMPXX	0FFA8H	Causes the CPU to jump to the given address on the given bank.
CALLXX	0FFABH	Calls the module at the given address on the given bank.

4.4.3.3 Programming notes

The application program must use the following steps to switch banks using no OS routine:



BANKR (Bank Register) is a write-only register assigned to I/O port address 05H. Its format is shown below.

